โดย ศุภชัย สมพานิช บรรณาธิการ สัจจะ จรัสรุ่งรวีวร

DEV BOOK
Infopress Developer Book

Advance .NET Programming

ฉบับมืออาชีพ

หากคุณอีดอัด เพราะไม่เคยใช้ความสามารถของ .NET Framework VB 2005/VC# 2005 อย่างเต็มที่ หรืออยากรู้ว่า โปรแกรมเมอร์มืออาชีพ (ค่าตัวสูง) ต้องรู้ลึกแค่ไหน หนังสือเล่มนี้มีคำตอบ

- อธิบายครบทุกฟีเจอร์ของ .NET Framework 2.0 ในแบบ OOP พร้อมเจาะลึกด้วยเทคนิคระดับมืออาชีพ
- อธิบายพร้อมกันทั้งภาษา Visual Basic 2005 และ Visual C# 2005
- เหมาะสำหรับโปรแกรมเมอร์ นักเรียนนักศึกษา และผู้สนใจจะไต่ระดับไปสู่โปรแกรมเมอร์มืออาชีพ

FREE CD-ROM

ตัวอย่าง Source Code Patch และ Resource ต่างๆ

พิเศษ! พร้อมใช้งานกับ Windows Vista®

375.-

Advance .NET Programming ฉบับมืออาชีพ

โดย ศุภชัย สมพานิช
บรรณาธิการ สัจจะ จรัสรุ่งรวีวร

DEV BOOK
Infopress Developer Book

Advance .NET Programming ฉบับมืออาชีพ

ผู้แต่ง ศุภชัย สมพานิช thaivb@yahoo.com
บรรณาธิการ สัจจะ จรัสรุ่งรวีวร sajja@infopress2000.com
ออกแบบปก อนัน วาโซะ
ออกแบบและจัดรูปเล่ม วุฒิพันธ์ สมพระเมฆ, จิตราภรณ์ เหมะจันทร
พิสูจน์อักษร ราตรี นาควงค์, มนฤดี ศรีอุทโยภาส, สุนทรี บรรลือศักดิ์, รติมา จันทร์
ประสานงานการผลิต ศุภชัย สิริสุขขจร, ฉัตรชนก แก้วจันทร์
ข้อมูลทางบรรณานุกรม ศุภชัย สมพานิช

Advance .NET Programming ฉบับมืออาชีพ. นนทบุรี : ไอดีซีฯ, 2550
560 หน้า
1. วิชวลเบสิค (โปรแกรมคอมพิวเตอร์) 2. การเขียนโปรแกรม I ชื่อเรื่อง
005.118

บริษัท ไอดีซี อินโฟ ดิสทริบิวเตอร์ เซ็นเตอร์ จำกัด จัดตั้งขึ้นเพื่อเผยแพร่ความรู้เทคโนโลยีสารสนเทศทั้งในระดับพื้นฐานและระดับสูง เรายินดีรับงานเขียนของนักวิชาการและนักเขียนทุกท่าน โดยเฉพาะงานที่เกี่ยวกับสารสนเทศ ท่านผู้สนใจกรุณาติดต่อผ่านทางอีเมลที่ editor@infopress2000.com หรือทางโทรศัพท์หมายเลข 0-2962-1081 (อัตโนมัติ 10 คู่สาย) โทรสาร 0-2962-1084

ISBN 978-974-9749-01-2
พิมพ์ครั้งที่ 1 ตุลาคม 2550

สร้างสรรค์ผลงานโดย DEV BOOK
Infopress Developer Book

จัดพิมพ์และจัดจำหน่ายโดย
IDC Info Distributor Center

บริษัท ไอดีซี อินโฟ ดิสทริบิวเตอร์ เซ็นเตอร์ จำกัด
200 หมู่ 4 ชั้น 5 ห้อง 514 อาคารจัสมินอินเตอร์เนชั่นแนลทาวเวอร์
ถ.แจ้งวัฒนะ อ.ปากเกร็ด จ.นนทบุรี 11120
โทรศัพท์ 0-2962-1081 (อัตโนมัติ 10 คู่สาย) โทรสาร 0-2962-1084

ราคา 375 บาท

บทบรรณาธิการ
EDITOR TALK

DEV BOOK ได้รับคำแนะนำเชิงร้องขอจากผู้อ่านในช่วง 2 ปีที่ผ่านมาว่า อยากมีหนังสือที่เจาะลึกถึงแก่นของ .NET จะด้วยภาษาใดก็ได้ขอให้คุณภาพเทียบได้กับหนังสือฝรั่ง

ความต้องการข้างต้นถูกเก็บนิ่งไว้นานพอสมควร แต่ต้องบอกว่าทั้งตัวผมและผู้เขียนต่างก็พร้อมที่จะนำเสนอเนื้อหาระดับเทพมากำนัลท่านผู้อ่านกันอยู่แล้ว เพียงแต่ความกังวลหลักก็คือ เรื่องของจำนวนผู้อ่านว่าจะมีมากพอให้เราพิมพ์ออกมาได้หรือไม่ แต่เมื่อเสียงเรียกร้องเริ่มมีมากเข้าก็ทำให้ความมั่นใจเพิ่มพูนขึ้น พร้อมกับการเริ่มเตรียมงานเมื่อราวกลางปีที่ผ่านมา

เนื้อหาในหนังสือต้องบอกว่ามือใหม่ไม่เกี่ยว เพราะผู้อ่านหนังสือเล่มนี้จะต้องมีพื้นฐานความรู้ด้านการเขียนโปรแกรมไม่ว่าจะเป็น Visual Basic หรือ C# มาแน่นพอสมควร อีกทั้งมีความปรารถนาจะรีดพลังจาก .NET ออกมาใช้ให้เต็มที่ ไม่ว่าจะเพื่ออาชีพการงาน หรือใช้ประโยชน์ด้านการเล่าเรียน

เนื้อหาในหนังสือเล่มนี้จึงพร้อมที่จะอ่านจากจุดใดๆ ก็ได้ เพียงแต่อาจจะต้องอ่านบทที่ 1 ให้เข้าใจพื้นฐานความรู้บางเรื่องเสียก่อน โดยเนื้อหาทุกหัวข้อและทุกบทมีตัวอย่างประกอบทั้ง 2 ภาษา คือ Visual Basic 2005 และ Visual C# 2005 อธิบายทุกแง่ทุกมุม อธิบายว่าใช้แนวคิดหรือใช้คำศัพท์เทียบเคียงกันอย่างไร ซึ่งจะช่วยให้ผู้อ่านได้รับประโยชน์คุ้มค่าอย่างแท้จริง

แน่นอนว่าไม่มีสิ่งใดที่ไร้ข้อบกพร่อง หากท่านผู้อ่านประสบพบเห็นก็โปรดกรุณาแจ้งมา หรือจะติชมแนะนำเพิ่มเติม ทางผู้เขียนและ DEV BOOK ยินดีรับฟังด้วยความเต็มใจยิ่ง

ด้วยความเคารพ
สัจจะ จรัสรุ่งรวีวร

คำนำ
PREFACE

ถ้าจะกล่าวถึงการเขียนโปรแกรมของภาษาต่างๆ ไม่ว่ากี่ยุค กี่สมัยก็ตาม คุณผู้อ่านมักจะได้ยินคำว่า การเขียนโปรแกรมเชิงวัตถุบ้าง หรือไม่ก็ OOP บ้าง ไม่มากก็น้อย ไม่ว่าคุณจะศึกษาภาษาใดก็ตาม ย่อมหลีกหนีการเขียนโปรแกรมแบบ OOP ไม่พ้นอยู่ดี ไม่มีข้อยกเว้นแม้กระทั่งภาษาในยุค .NET Framework

หนังสือในซีรีส์ OOP ของผู้เขียนไม่ได้มุ่งเน้นการศึกษาภาษา VB หรือ VC# ว่าภาษาใดดีกว่ากัน หรือภาษาใดควรศึกษามากกว่า ฯลฯ แต่เป็นการศึกษาการเขียนโปรแกรมแบบ OOP ในยุคของ .NET ว่า มีหลักการและมีวิธีการอย่างไร และที่สำคัญคือ ดีกว่าการเขียนโปรแกรมแบบเดิมอย่างไร

ไม่ว่าคุณจะใช้ภาษาอะไรก็ตามที่สนับสนุน .NET Framework ท้ายที่สุดแล้ว ภาษาต่างๆ เหล่านี้จะถูกแปลงเป็นภาษา IL (Intermediate Language) ทั้งสิ้น ถ้าจะกล่าวแบบง่ายๆ ก็คือ ภาษาในยุคของ .NET Framework มีเพียงหนึ่งเดียวนั่นคือ ภาษา IL สิ่งที่เราต้องศึกษาคือ เราจะทำอย่างไรจึงจะได้ภาษา IL ที่ดี และมีประสิทธิภาพมากที่สุดต่างหากนี่คือ โจทย์ใหญ่ที่สุดที่ผู้เขียนตั้งไว้กับหนังสือในซีรีส์ของ OOP ส่งผลให้เนื้อหาที่นำเสนอ จึงไม่ได้ขึ้นอยู่กับเวอร์ชันของ .NET Framework หรือ Visual Studio แต่อย่างใด

ท้ายที่สุดนี้ ผู้เขียนขอขอบคุณคุณสัจจะ จรัสรุ่งรวีวร ที่ได้ให้โอกาสผู้เขียนจัดทำหนังสือในซีรีส์ OOP นี้ รวมไปถึงเจ้าหน้าที่ทุกท่านทุกฝ่ายของ DEV BOOK ที่ช่วยกันทำงานอยู่เบื้องหลัง ขอบคุณเพื่อนๆ ที่โทรมาคุยยามดึกๆ ในขณะที่กำลังแต่งต้นฉบับ และที่ลืมไปไม่ได้เลยก็คือ ข้อแนะนำ คำติติง คำทักท้วงจากคุณผู้อ่านที่ติดตามผลงานกันเรื่อยมา รวมถึงต้องขออภัยเป็นอย่างสูง กับการตอบ E-mail ที่ล่าช้าไปบ้างบางจังหวะและบางโอกาส แต่ขอยืนยันว่า E-mail ทุกฉบับมีคุณค่าสำหรับผมเป็นอย่างยิ่งครับ

ศุภชัย สมพานิช

ตุลาคม 2550

Contents

บทที่ 8 การสืบทอดคลาส (Inheritance)

บทที่ 9 Constructor

บทที่ 10 Structure และ Abstract Class

บทที่ 11 Interface

บทที่ 12 Interface Programming

บทที่ 13 อาร์เรย์

บทที่ 14 ตัวเลขใน .NET

บทที่ 15 วันที่และเวลา

บทที่ 16 การจัดการ String

บทที่ 17 โครงสร้างข้อมูลและอัลกอริธึม

บทที่ 18 ดีลีเกต

บทที่ 19 Windows Form

บทที่ 20 การสร้างผู้ช่วยเหลือในระบบธุรกิจ

CHAPTER 1

ก้าวแรกสู่โลกของ OOP

บทนำ

การเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) เรียกสั้นๆ ว่า OOP จัดเป็นการเขียนโปรแกรมที่มีความสำคัญ และมีความยุ่งยากมากที่สุดอีกแขนงหนึ่งในโลกของการเขียนโปรแกรม เป็นอีกโลกหนึ่งที่ผู้พัฒนาโปรแกรมหลายๆ ท่านทั้งชื่นชอบ และหันหลังให้ OOP อย่างสิ้นเชิง

แต่ในยุคปัจจุบันการพัฒนาแอพพลิเคชันโดยอาศัยแนวความคิดแบบ OOP กลายเป็นสิ่งที่หลีกเลี่ยงไม่ได้เสียแล้ว ไม่ยกเว้นแม้กระทั่งภาษาในยุคของ .NET Framework ก็ตาม

สำหรับคุณผู้อ่านที่ยังไม่เคยศึกษาการเขียนโปรแกรมแบบ OOP มาก่อน ผู้เขียนขอแนะนำให้ศึกษาเรียงลำดับตามบท ส่วนคุณผู้อ่านที่เคยใช้ OOP มาบ้างแล้ว สามารถเลือกศึกษาตามเนื้อหาที่สนใจได้เลย

ทำไมต้องใช้ OOP ?

คำถามข้างต้นมักจะเป็นคำถามแรกๆ ที่คุณผู้อ่านหลายๆ ท่านเริ่มศึกษาการเขียนโปรแกรมเชิงวัตถุ ซึ่งเป็นคำถามที่เคยเกิดขึ้นมากับผู้เขียนเช่นกันว่า ทำไมเราต้องเข้าไปโลกของ OOP ด้วย ทั้งๆ ที่การเขียนโปรแกรมแบบเดิม (Structure Programming) ก็ดีอยู่แล้ว เพียงแต่ว่าการเขียนโปรแกรมแบบ OOP ดีกว่าเท่านั้นเอง ผู้เขียนขอยกสิ่งใกล้ตัวที่สุดที่สามารถนำมาอธิบาย OOP นั่นคือ บ้านที่เราอาศัยอยู่

ส่วนประกอบของบ้านเกิดจากหลังคาสำเร็จรูป, ปูนสำเร็จรูป, ก้อนอิฐสำเร็จรูป, สีทาบ้านสำเร็จรูป เป็นต้น โดยมีสถาปนิกเป็นผู้ออกแบบบ้าน มีผู้รับเหมาเป็นผู้นำสิ่งสำเร็จรูปย่อยๆ เหล่านี้ ประกอบเป็นบ้านที่สมบูรณ์แบบ คุณไม่จำเป็นต้องไปสร้างกระเบื้อง หรือไปสร้างก๊อกน้ำ หรือสร้างอะไรก็แล้วแต่ด้วยตัวคุณเอง

กล่าวได้อีกนัยหนึ่งคือ บ้านที่เสร็จสมบูรณ์เกิดจากหน่วยสำเร็จรูปย่อยๆ เหล่านี้ ทำงานร่วมกันนั่นเอง เห็นได้ว่าในโลกของความเป็นจริงแล้ว เราคุ้นเคยกับวัตถุสำเร็จรูปต่างๆ เหล่านี้อยู่แล้ว

ถ้าคุณมองบ้านให้ดี คุณจะพบว่าไม่มีชิ้นส่วนใดเลยของบ้านที่เรียกว่า บ้าน มีแต่ต้นเสา, มีบันได, มีประตู, มีหน้าต่าง เป็นต้น บ้านสำเร็จรูปเกิดจากการประกอบกันของวัสดุสำเร็จรูปต่างๆ เหล่านี้อย่างมีรูปแบบ มีแบบแผน มีกติกา นี่คือวิธีการมองแบบ OOP

แอพพลิเคชันที่คุณต้องการสร้างขึ้นมาไม่ต่างอะไรกับบ้าน การเขียนโปรแกรมแบบ OOP เป็นการนำคุณเข้าไปสู่การสร้างวัสดุสำเร็จรูปต่างๆ เพื่อให้คุณสามารถนำกลับมาใช้ใหม่ได้ครั้งแล้วครั้งเล่า วัสดุสำเร็จรูปที่คุณสร้างขึ้นมาคือ ผู้ช่วยเหลือของคุณ ช่วยให้คุณสร้างแอพพลิเคชันใหญ่ๆ ได้รวดเร็วยิ่งขึ้น

คุณผู้อ่านที่เคยฝันร้ายกับการศึกษาการเขียนโปรแกรมแบบ OOP ผู้เขียนก็เป็นอีกคนหนึ่งที่ไม่ต่างอะไรกับคุณผู้อ่านหลายๆ ท่าน ในเมื่อผู้เขียนมีโอกาสในการนำเสนอการเขียนโปรแกรมแบบ OOP จึงต้องลบล้างปัญหา และอุปสรรคดังกล่าวที่เกิดขึ้นให้ได้เช่นกัน

มีอยู่ 2 คำถามที่เกิดขึ้นในระหว่างการศึกษาการเขียนโปรแกรมแบบ OOP นั่นคือ

1. OOP ทำอย่างไร ?
2. OOP ใช้ตอนไหน ?

ทั้ง 2 คำถามข้างต้นเป็นคำถามที่ถามง่ายมาก แต่ตอบยากที่สุด ผู้เขียนใช้คำถามข้างต้นเป็นโจทย์ใหญ่ที่สุดของการจัดทำหนังสือในซีรีส์ของ OOP โดยยึดแนวทางการอธิบายในเชิงปฏิบัติ ไม่ใช่เชิงนิยาม วิธีคิดของผู้เขียนง่ายๆ ไม่ซับซ้อนนั่นคือ

สมมติว่าการเขียนโปรแกรมแบบ OOP ประกอบไปด้วย เนื้อหา 1-10 การอธิบายในเชิงนิยามหมายความว่า นำเสนอตั้งแต่ข้อ 1...2...3... ไล่ไปเรื่อยๆ จนถึงข้อ 10 ครบถ้วนตามเนื้อหาของ OOP

ส่วนการอธิบายในเชิงปฏิบัติ หมายถึง นำเสนอวิธีการใช้งานแล้วก็บอกว่า นี่คือข้อที่ 2 แล้วก็นำเสนอวิธีใช้งานแล้วก็บอกว่า นี่คือข้อที่ 5 เห็นได้ว่าบทสรุปส่งท้ายที่ได้คือ คุณผู้อ่านรู้ว่า OOP มี 1-10 เหมือนกัน เพียงแต่ว่าวิธีนี้จะทำให้การศึกษา OOP รวดเร็ว และเป็นมิตรมากกว่านั่นเอง

ผู้เขียนขอย้อนกลับไปตรงคำกล่าวที่ว่า การเขียนโปรแกรมแบบ OOP ดีกว่าแบบ Structure Programming อย่างไร สิ่งที่เราสนใจคือ เราใช้อะไรเป็นตัวชี้วัดคำว่าดีกว่า

ถ้าใช้เกณฑ์ความสะดวกสบายมาวัด ผลที่ได้คือ OOP มีความวุ่นวายมากกว่าอย่างแน่นอน เพราะว่า OOP เต็มไปด้วยคำศัพท์, กฎ กติกาต่างๆ มากมาย ทำให้ยากต่อการศึกษานั่นเอง

เกณฑ์ที่นำมาใช้วัดคือ การเขียนโปรแกรมแบบ OOP ช่วยลดความซ้ำซ้อนในการเขียนโปรแกรม ช่วยให้การตรวจสอบโค้ดง่ายยิ่งขึ้น แต่ที่สำคัญที่สุดก็คือ คุณสามารถนำโค้ดที่อยู่ในรูปแบบ OOP กลับมาใช้ใหม่ได้ตามที่คุณต้องการ รวมถึงคนอื่นๆ ก็สามารถนำโค้ดที่อยู่ในรูปแบบ OOP ของคุณไปใช้ได้อีกด้วย ซึ่งจะส่งผลให้ระยะเวลาในการพัฒนาแอพพลิเคชันขนาดใหญ่ลดลง เพราะว่า OOP คือ กติกาสากลที่รับรู้กันโดยทั่วไป

เหตุการณ์หนึ่งที่คุณผู้อ่านต้องเคยเจอนั่นคือ เมื่อคุณเริ่มต้นทำโปรเจ็กต์ใหม่ เรามักจะกลับไป Copy โค้ดจากโปรเจ็กต์เก่าๆ มาใช้ในโปรเจ็กต์ใหม่บ้างไม่มากก็น้อย เห็นได้ว่าโค้ดจากโปรเจ็กต์เก่าๆ ช่วยลดเวลาการเขียนโค้ดให้กับโปรเจ็กต์ใหม่ของคุณ อาจจะต้องมีการแก้ไขโค้ดบ้าง แต่อย่างน้อยก็ไม่ต้องเขียนใหม่ทั้งหมด

คอนโทรลต่างๆ ที่คุณเรียกใช้งานก็ถือว่าเป็นคลาสชนิดหนึ่งเช่นกัน เพียงแต่ว่าเป็นคลาสที่มีส่วนแสดงผล คุณมองเห็น คุณจับต้องได้ จึงไม่รู้สึกว่าการใช้งานคลาสคอนโทรลต่างๆ เป็นเรื่องยุ่งยากแต่อย่างใด เพราะว่าความยุ่งยากเหล่านี้ถูกซ่อนไว้อยู่เบื้องหลัง เห็นได้ว่าการที่คุณเข้ามาสู่ในยุคของ .NET คุณได้เข้ามาโลกของ OOP ครึ่งหนึ่งแล้วนั่นเอง

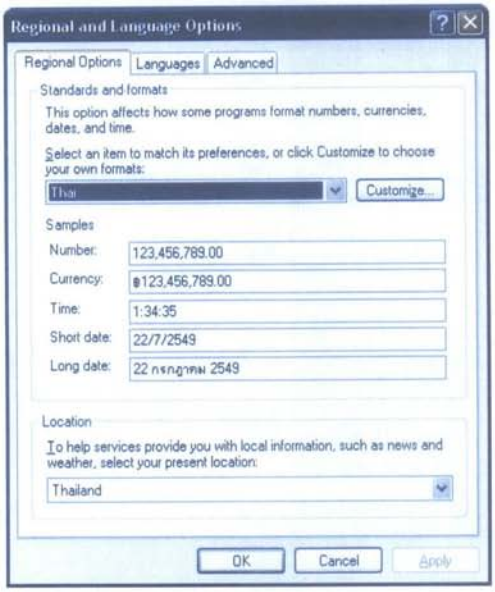
สิ่งที่ผู้เขียนยกมาข้างต้น ล้วนแต่เป็นกฎกติกาของ OOP ทั้งสิ้น เพียงแต่ว่าเราใช้ไปโดยไม่รู้ตัว เนื้อหาที่ผู้เขียนนำเสนอในหนังสือเล่มนี้ จะทำให้คุณมอง OOP เป็นเพื่อน เป็นผู้ช่วยเหลือของคุณ ไม่ใช่ศัตรู หรือเป็นอุปสรรคที่แสนจะน่าเบื่อแต่อย่างใด

โครงสร้างของภาษา VB 2005 และ VC# 2005 ถูกแก้ไขให้เข้ามาหาโลกของ OOP อย่างเต็มตัว โดยที่ OOP มีเพียงหนึ่งเดียว ภาษาคือ สิ่งที่นำ OOP ไปใช้เป็นการใช้งาน OOP ที่ลอยอยู่บน .NET Framework อีกชั้นหนึ่งนั่นเอง

การเตรียมสภาพแวดล้อมก่อนการใช้งาน

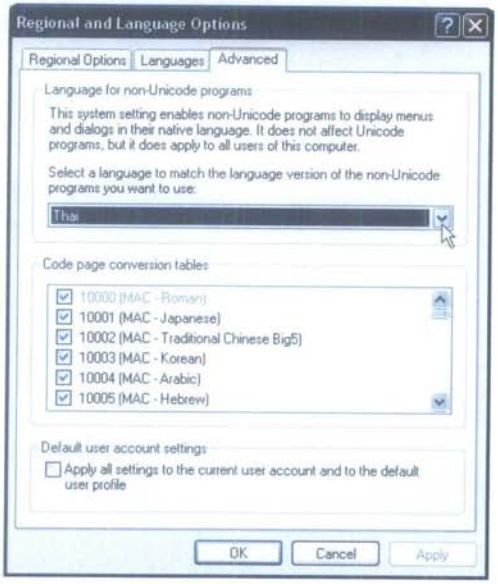
เพื่อให้ผู้เขียนและผู้อ่านเข้าใจตรงกัน ขอให้คุณผู้อ่านเตรียมสภาพแวดล้อม ดังนี้

1. กำหนดให้ Windows XP SP2 Professional และ Windows Vista ใช้ภาษาไทย เป็นภาษาหลักของระบบปฏิบัติการแยกเป็น 2 กรณีกล่าวคือ
   - กรณี Windows XP SP2 ให้คุณคลิกปุ่ม start > Control Panel ดับเบิลคลิกที่ Regional and Language Options ดังรูปที่ 1-1



รูปที่ 1-1 แสดงการกำหนดให้ Windows XP SP2 Professional ใช้งานภาษาไทย

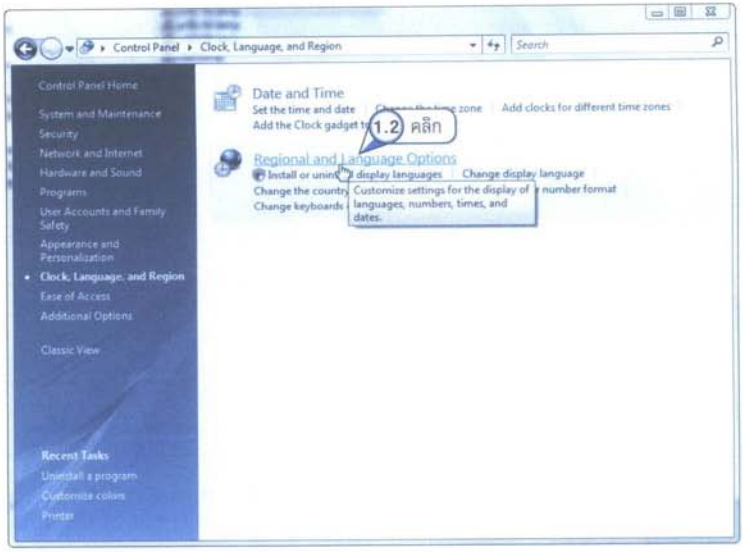
รูปที่ 1-1 (ต่อ) แสดงการกำหนดให้ Windows XP SP2 Professional ใช้งานภาษาไทย



- กรณี Windows Vista ให้คุณคลิกปุ่ม  > Control Panel > Clock, Language, and Region จากนั้นคลิกที่ Regional and Language Options ดังรูปที่ 1-2

รูปที่ 1-2 การกำหนดให้ Windows Vista ใช้งานภาษาไทย

รูปที่ 1-2 (ต่อ)
การกำหนดให้
Windows Vista
ใช้งานภาษาไทย



2. ในไดอะล็อกบ็อกซ์ Regional and Language Options ที่แท็บ Location และแท็บ Formats ให้คุณเลือกภาษาไทย ดังรูปที่ 1-3

รูปที่ 1-3
การกำหนด
ภาษาไทยใน
Windows Vista

3. ให้คุณ download ไฟล์ Visual Studio 2005 SP1 แยกเป็น 2 กรณีคือ
   - **กรณีใช้ Visual Studio 2005 Express Edition** แยกโหลดตามภาษาที่คุณใช้ เช่น Visual Basic 2005 (32 MB) หรือ Visual C# 2005 (24 MB) ได้ที่

http://www.microsoft.com/downloads/details.aspx?familyid=BB4A75AB-E2D4-4C96-B39D-37BAF6B5B1DC&displaylang=en

   - **กรณีใช้ Visual Studio 2005 Professional Edition** หรือ Team Edition ขนาด 431 MB สามารถโหลดได้ที่

http://www.microsoft.com/downloads/details.aspx?FamilyID=7B0B0339-6131-46E6-AB4D-080D4D4A8C4E&displaylang=en

ให้คุณติดตั้งไฟล์ Visual Studio 2005 SP1 ตาม Edition ที่คุณใช้ (ค่อนข้างใช้เวลาในการติดตั้งพอสมควร)

NOTE

ไฟล์ Visual Studio 2005 SP1 เก็บอยู่ในโฟลเดอร์ Patch\Visual Studio 2005 SP1 ของ CD-ROM ที่แถมมากับหนังสือเล่มนี้

4. ในกรณีที่คุณใช้งาน VS 2005 บน Windows Vista ให้คุณ download ไฟล์ VS 2005 SP1 For Vista ขนาด 29 MB ได้จาก http://www.microsoft.com/downloads/details.aspx?FamilyID=90E2942D-3AD1-4873-A2EE-4ACC0AACE5B6&displaylang=en

NOTE

ไฟล์ VS80sp1-KB932232-X86-ENU.exe เก็บอยู่ในโฟลเดอร์ Patch\VS 2005 SP1 For Vista ของ CD-ROM ที่แถมมากับหนังสือเล่มนี้

5. เนื่องจากไมโครซอฟท์แนะนำให้ติดตั้งตัวแก้ไขข้อผิดพลาดในระดับร้ายแรง (Critical) ของ Visual Studio 2005 มีขนาด 6.8 MB สามารถ download ได้ที่

http://www.microsoft.com/downloads/details.aspx?familyid=C2682C53-8E91-4C7D-B782-BE78512DCBFA&displaylang=en

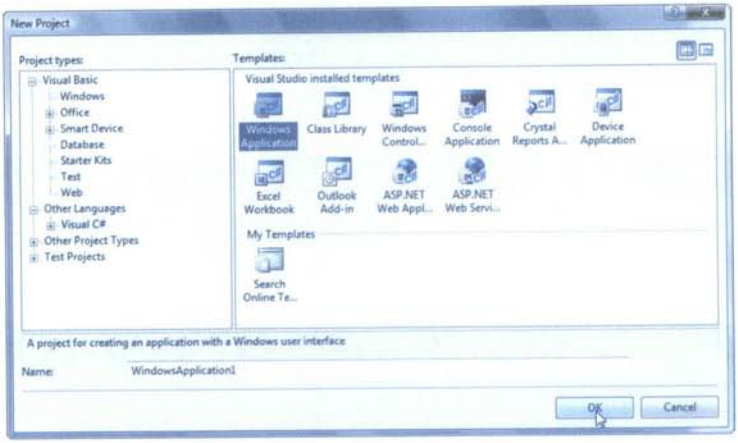
NOTE

ไฟล์ VS80-KB925674-X86.exe เก็บอยู่ในโฟลเดอร์ Patch\Visual Studio 2005 WMISCRIPTUTILS.DLL Security Update ของ CD-ROM ที่แถมมากับหนังสือเล่มนี้

VC# 2005 Concept ใน 30 นาที

สำหรับคุณผู้อ่านที่เคยใช้ภาษา VC# 2005 มาบ้างแล้ว สามารถข้ามหัวข้อนี้ไปศึกษาหัวข้อถัดไปได้เลย โดยจะกล่าวถึงสิ่งที่คุณควรรับทราบในข้างต้นก่อนใช้งานภาษา VC# 2005

รูปที่ 1-4 การเลือกใช้งาน VC# 2005 ของ VS 2005



ภาษา VC# 2005 เป็นภาษาที่ถูกออกแบบมาเพื่อรองรับการทำงานในยุค .NET โดยมีแนวของภาษาเป็นแบบ การเขียนโปรแกรมเชิงวัตถุสมัยใหม่ (Modern Object Oriented Programming) เรียกสั้นๆ ว่า Modern OOP

สำหรับผู้อ่านที่เคยศึกษาภาษา C/C++ มาก่อนย่อมทราบดีว่า การเขียนโปรแกรมในเชิงวัตถุมีความยุ่งยาก และยากต่อการทำความเข้าใจ เพราะต้องอาศัยความรู้หลายๆ อย่างเข้ามาประกอบกัน

แต่สำหรับภาษา VC# 2005 แล้ว การเขียนโปรแกรมเชิงวัตถุไม่ใช่เรื่องยุ่งยากอีกต่อไป โดยที่จุดยืนของภาษา VC# 2005 อยู่ที่อาศัยไวยากรณ์ที่ปรับปรุงมาจาก C/C++ ร่วมกับการเขียนโปรแกรมแบบรองรับเหตุการณ์ของ VB 2005 กลายมาเป็น VC# 2005

แนวความคิดของการเขียนโปรแกรมแบบ Modern OOP เกิดจากการที่ไมโครซอฟท์พัฒนาคลาส (Class) ต้นแบบต่างๆ ขึ้นมา ที่เรียกว่า Base Class Library (BCL) แล้วนำมาจัดหมวดหมู่ให้เป็นระเบียบ เมื่อต้องการเรียกใช้งานคลาสใดก็จะอาศัยระบบเนมสเปซ (Namespaces System) เข้ามาช่วยในการระบุคลาสต้นแบบต่างๆ เพื่อให้ผู้พัฒนาสามารถนำออบเจ็กต์ต่างๆ ที่อยู่ในคลาสนั้นๆ ออกมาใช้งานได้อย่างง่ายดาย

ผู้เขียนมีข้อแนะนำในข้างต้นที่น่าสนใจของภาษา VC# 2005 ที่ขอให้คุณนึกถึงอยู่เสมอ เมื่อมีการเขียนโค้ดในบทต่อๆ ไป ขอให้ยึดถือโดยเคร่งครัด ซึ่งอาจจะไม่สามารถครอบคลุมไวยากรณ์ของภาษา VC# 2005 ได้ทั้งหมด แต่ขอให้ยึดถือไว้ในข้างต้น ดังนี้

- **คำสั่งต่างๆ ของภาษา VC# 2005 จะเป็นตัวอักษรตัวเล็กทั้งหมด** ยกเว้นการเรียกใช้กลุ่มออบเจ็กต์ต่างๆ เท่านั้น ที่มีตัวอักษรใหญ่ปะปนอยู่
- **ตัวอักษรเล็กใหญ่มีความแตกต่างกัน** เช่น strConn กับ strconn ถือว่าเป็นตัวแปรคนละตัวกัน เพราะว่าในภาษา VC# 2005 ยึดถือตัวอักษรเล็ก-ใหญ่อย่างเคร่งครัดทุกกรณี
- **เมื่อจบประโยคจะใช้เครื่องหมาย ; เสมอ** ยกเว้นบรรทัดเปิดบล็อกด้วยเครื่องหมาย { } ที่ไม่ต้องมีเครื่องหมาย ; ต่อท้าย
- **ชนิดของข้อมูลที่เรียกใช้มีความสำคัญเป็นอย่างยิ่ง** กล่าวคือ ตัวเลขที่อยู่ในฐานะข้อความกับตัวเลขที่อยู่ในฐานะตัวเลขมีความแตกต่างกัน รวมถึงชนิดของข้อมูลข้างต้น (Primitive Data Type) ทุกชนิดด้วย เช่น string, int, long เป็นต้น
- **การเรียกใช้งานเนมสเปซต่างๆ จะใช้คำสั่ง using** ซึ่งแตกต่างจาก VB 2005 ที่ใช้คำสั่ง Imports

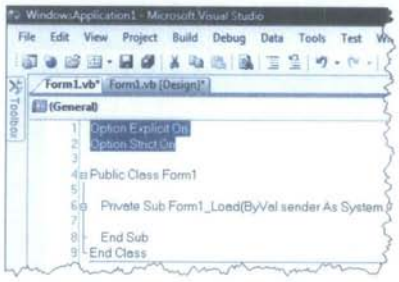
VB 2005

```
Imports System.Data
Imports System.Data.OleDb
```

VC# 2005

```
using System.Data;
using System.Data.OleDb;
```

เพราะความที่ตัวภาษา VC# 2005 เคร่งครัด และเข้มงวดต่อการแปลงข้อมูลจากชนิดหนึ่งไปสู่อีกชนิดหนึ่ง ดังนั้น เพื่อให้กรอบการทำงานมีกติกาเดียวกันทั้ง 2 ภาษา ผู้เขียนจึงกำหนดให้ VB 2005 เคร่งครัดต่อการเปลี่ยนแปลงชนิดของข้อมูลเช่นเดียวกัน ดังรูปที่ 1-5



รูปที่ 1-5
การประกาศคำสั่งเพิ่มเติมใน Editor ของ VB 2005

คำสั่ง Option Explicit On ทำหน้าที่กำหนดว่าตัวแปรทุกตัวที่คุณใช้งาน ต้องมีการประกาศการใช้งานก่อนทุกครั้ง ส่วนคำสั่ง Option Strict On ทำหน้าที่กำหนดว่าการเปลี่ยนแปลงข้อมูลทุกชนิดจะใช้กติกาอย่างเข้มงวดและเคร่งครัด

พื้นฐานอีกอย่างหนึ่งของภาษา VC# 2005 ที่คุณควรทราบก็คือ บล็อกของคำสั่ง ในภาษา VC# 2005 จะใช้เครื่องหมาย { ทำหน้าที่เปิดคำสั่ง และใช้เครื่องหมาย } ปิดบล็อกของคำสั่ง

แต่ในหลายๆ คำสั่งของภาษา VC# 2005 ไม่จำเป็นต้องมีบล็อกคำสั่งก็ได้ ส่งผลให้รูปแบบของคำสั่งมี 2 ลักษณะคือ ทั้งที่มีบล็อกคำสั่ง { } และไม่มีบล็อกคำสั่ง เช่น คำสั่ง if...else...

```
01    if (เงื่อนไขที่ตรวจสอบ)
02    {
03          //กรณีเงื่อนไขเป็นจริง
04    }
05    else
06    {
07          //กรณีเงื่อนไขเป็นเท็จ
08    }
```

จากตัวอย่างข้างต้นเป็นการใช้งานคำสั่ง if...else... ที่มีการใช้บล็อกของคำสั่ง { } โดยที่ถ้าในกรณีเงื่อนไขเป็นจริงก็จะมาทำงานที่โค้ดบรรทัดที่ 2-4 แต่ถ้าในกรณีเงื่อนไขเป็นเท็จโค้ดก็จะมาทำงานที่บรรทัด 6-8 จะเห็นได้ว่าการใช้งานบล็อกของคำสั่ง ทำให้ง่ายต่อการศึกษาโค้ดมากกว่าที่จะไม่ใช้ ซึ่งมีผลเป็นอย่างยิ่งในกรณีที่โค้ดของคุณมีความยาวมาก มีการตรวจสอบเงื่อนไขหลายๆ กรณี

ประโยชน์ของบล็อกคำสั่งก็คือ เป็นการบ่งบอกขอบเขตของคำสั่งนั้นๆ ว่าสิ้นสุด ณ จุดใดของโค้ดของคุณ ไวยากรณ์ที่ปรากฏอยู่ในหนังสือเล่มนี้ทั้งหมด ผู้เขียนจะยึดถือรูปแบบการใช้เครื่องหมาย { } ทุกกรณี เพื่อให้ผู้อ่านสามารถทำความเข้าใจขั้นตอนการทำงาน และทราบถึงขอบเขตของคำสั่งได้อย่างง่ายดาย

ในส่วนของผู้อ่านที่ศึกษาภาษา VC# 2005 จากหนังสือต่างประเทศ อาจจะพบว่ามีการลดรูปของบล็อกคำสั่ง { } เพื่อกระชับต่อการเขียนโค้ด ซึ่งไม่ผิดไวยากรณ์แต่อย่างใด ขึ้นอยู่กับแนวทางของคุณผู้อ่านแต่ละท่าน

ข้อสังเกตอีกอย่างหนึ่งของไวยากรณ์ภาษา VC# 2005 ก็คือ การใช้เครื่องหมาย = และ = = โดยที่

- **เครื่องหมาย =** ใช้สำหรับกำหนดค่าให้กับตัวแปรหรือออบเจ็กต์ เช่น การกำหนดชุดคำสั่ง SQL ให้กับตัวแปร sqlCategory

```
string sqlCategory = "SELECT * FROM Categories";
```

- **เครื่องหมาย = =** ใช้สำหรับเปรียบเทียบหรือตรวจสอบค่า เช่น การตรวจสอบสถานะของการเปิดไดอะล็อกบ็อกซ์ Open

```
if (openFileDialog1.ShowDialog() = = DialogResult.OK)
{
    //โค้ดอื่นๆ...
}
```

โค้ดข้างต้นเป็นการตรวจสอบว่า ถ้าผู้ใช้คลิกปุ่ม OK ในไดอะล็อกบ็อกซ์ Open แล้ว จะทำงานอื่นๆ ไม่ใช่เป็นการกำหนดค่าให้กับ openFileDialog1

NOTE

สรุปได้ว่าถ้าเป็นการกำหนดค่าใช้เครื่องหมาย = แต่ถ้าเป็นการเปรียบเทียบให้ใช้เครื่องหมาย = =

ยังมีเครื่องหมายอีก 2 ตัวเมื่อใช้งานกับออบเจ็กต์ต่างๆ แล้ว สร้างความสับสนเป็นอย่างยิ่งก็คือ เครื่องหมาย ( ) และเครื่องหมาย [ ] โดยที่

- **เครื่องหมาย ( )** หมายถึง การกระทำ (Methods) ของออบเจ็กต์นั้นๆ เช่น

```
OleDbDataAdapter.Fill(DataSet, "Categories");
```

โค้ดข้างต้นหมายความว่า ให้ออบเจ็กต์ OleDbDataAdapter ใส่ข้อมูลลงในออบเจ็กต์ DataSet ตั้งชื่อข้อมูลชุดนี้ว่า Categories กล่าวคือ ออบเจ็กต์ OleDbDataAdapter ใช้เมธอด Fill() กระทำต่อออบเจ็กต์ DataSet

- **เครื่องหมาย [ ]** หมายถึง การอ่านค่าหรือกำหนดค่าให้กับออบเจ็กต์นั้นๆ เช่น

```
ComboBox1.DataSource = DataSet.Tables["Categories"];
```

โค้ดข้างต้นหมายความว่าให้คอนโทรล ComboBox1 อ่านค่าออกมาจากออบเจ็กต์ DataSet จากชุดข้อมูลที่ชื่อว่า Categories

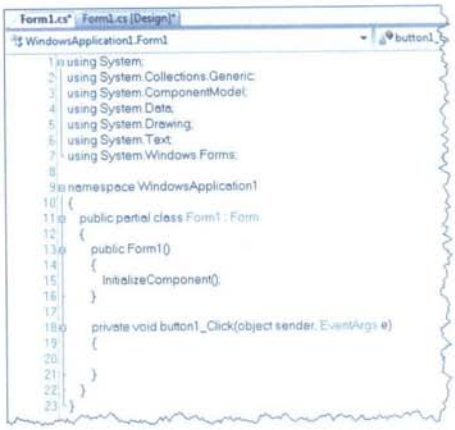
NOTE

สรุปได้ว่าถ้าเป็นการกระทำใช้เครื่องหมาย ( ) แต่ถ้าเป็นการอ่านค่าให้ใช้เครื่องหมาย [ ]

สิ่งสุดท้ายที่คุณควรทราบก็คือ การเพิ่มเหตุการณ์ต่างๆ ให้กับคอนโทรลแต่ละตัว วิธีการของภาษา VC# 2005 แตกต่างจาก VB 2005 อย่างสิ้นเชิง แยกเป็น 3 กรณี

- **กรณีที่** 1 การเพิ่มเหตุการณ์ประจำตัวของคอนโทรลแต่ละตัว ยกตัวอย่างเช่น คอนโทรล button ถ้าคุณต้องการเพิ่มเหตุการณ์ประจำตัวของคอนโทรล button ให้คุณดับเบิลคลิกที่คอนโทรล button แล้ว VC# 2005 IDE ก็จะสร้างเหตุการณ์ประจำตัวคอนโทรลขึ้นมา ดังรูปที่ 1-6

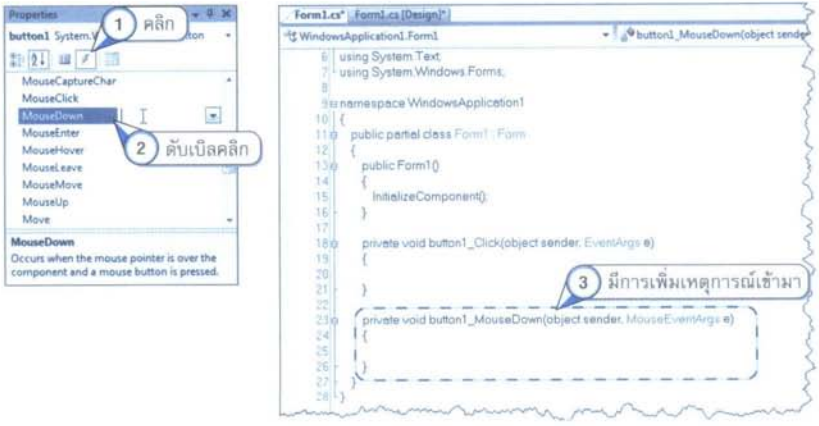
รูปที่ 1-6 กรณีการเพิ่มเหตุการณ์ประจำตัวของคอนโทรล button1 นั่นคือ เหตุการณ์ Click()



จากรูปที่ 1-6 คอนโทรลแต่ละตัวมีเหตุการณ์ประจำตัวไม่เหมือนกัน แตกต่างกันไปตามหน้าที่ของคอนโทรลนั้นๆ

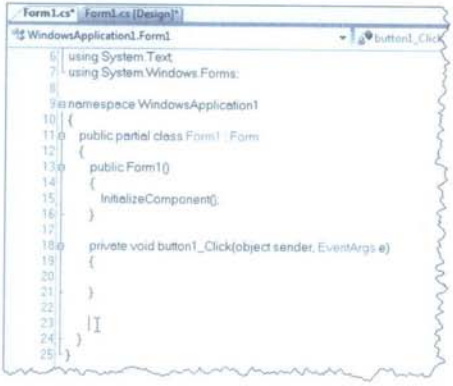
- **กรณีที่** 2 ถ้าคุณต้องการเพิ่มเหตุการณ์อื่นๆ นอกเหนือไปจากเหตุการณ์ประจำตัวคอนโทรล ให้คุณคลิกที่เครื่องหมาย ⚡ ในหน้าต่าง Properties เลือกเหตุการณ์ที่ต้องการใช้งาน แล้วดับเบิลคลิก ดังรูปที่ 1-7

รูปที่ 1-7 กรณีเพิ่มเหตุการณ์ MouseDown() ให้กับคอนโทรล button1



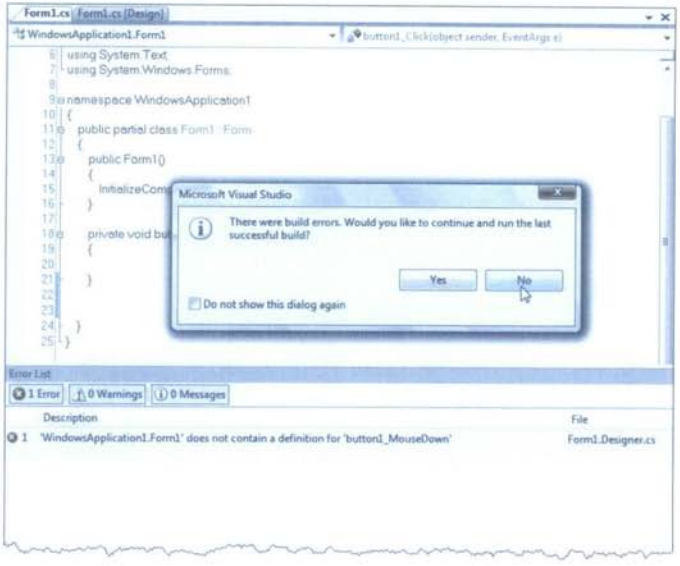
- **กรณีที่** 3 ถ้าคุณต้องการลบเหตุการณ์ใดๆ ออกมีขั้นตอน ดังนี้

รูปที่ 1-8
กรณีลบเหตุการณ์ button1_Mouse Down()



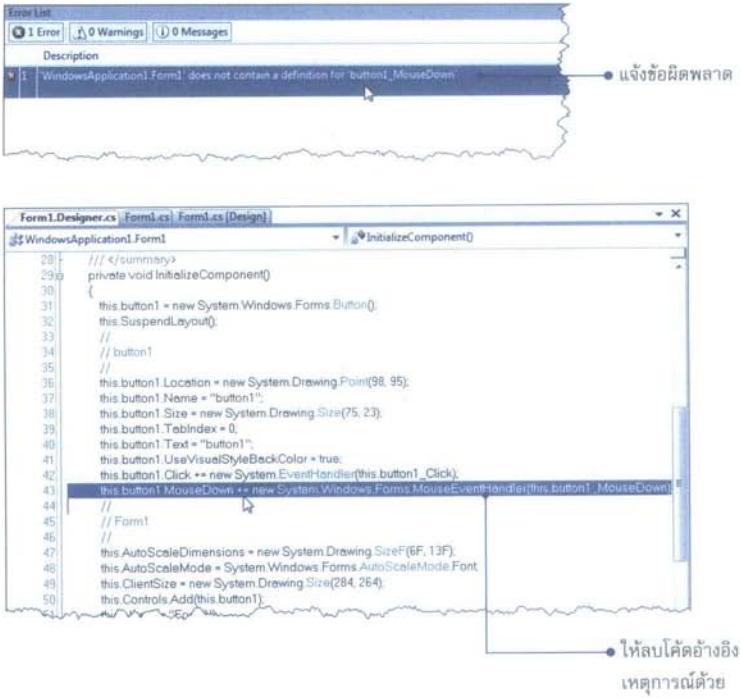
จากรูปที่ 1-8 สมมติว่าลบเหตุการณ์ button1_MouseDown() ออก จากนั้นให้คุณรันโปรเจ็กต์ซึ่งจะเกิดข้อผิดพลาด ดังรูปที่ 1-9

รูปที่ 1-9
ข้อผิดพลาด กรณีหาเหตุการณ์ไม่พบ



จากรูปที่ 1-9 เป็นข้อผิดพลาดที่เกิดจากการค้นหาเหตุการณ์ MouseDown() ของคอนโทรล button1 ไม่พบ อันเนื่องมาจากว่าคุณลบออกไปแล้วนั่นเอง คุณสามารถลบโค้ดบรรทัดที่อ้างอิงเหตุการณ์ MouseDown() ได้โดยการดับเบิลคลิกที่รายการผิดพลาด แล้วลบเหตุการณ์ MouseDown() ออก ดังรูปที่ 1-10

รูปที่ 1-10
การลบโค้ดที่อ้างอิงถึงเหตุการณ์ที่คุณลบไปแล้ว



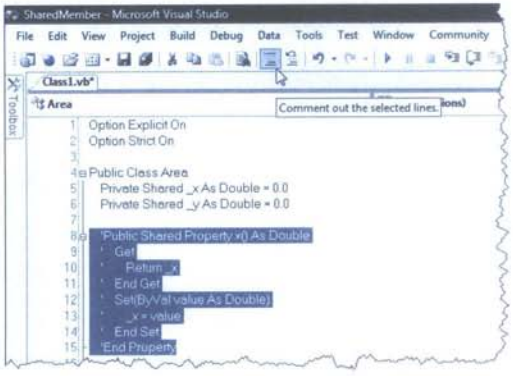
จากรูปที่ 1-10 โค้ดที่อ้างอิงเหตุการณ์และตัวเหตุการณ์ถูกลบออกไปแล้ว ก็จะส่งผลให้ไม่เกิดข้อผิดพลาดในการอ้างอิงเหตุการณ์ button1_MouseDown()

การทำหมายเหตุ (Comment) ใน VB 2005 ใช้เครื่องหมาย ' ส่วน VC# 2005 ทำได้ 2 วิธีคือ

- ใช้เครื่องหมาย // นำหน้าบรรทัดที่ต้องการทำหมายเหตุ
- ใช้เครื่องหมาย /* ... */ ใช้ในกรณีที่คุณต้องการทำหมายเหตุหลายบรรทัด

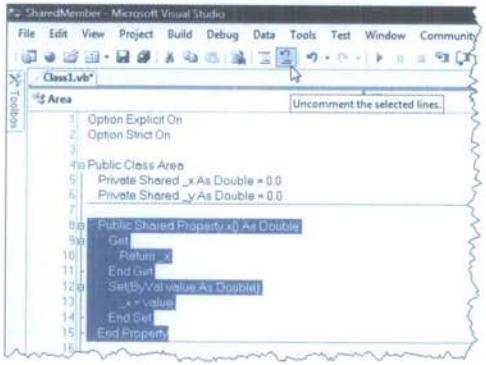
VS 2005 มีเครื่องมือที่ช่วยให้คุณทำหมายเหตุได้สะดวกยิ่งขึ้น วิธีนี้ใช้ได้ทั้ง 2 ภาษา

รูปที่ 1-11
กรณีการทำหมายเหตุหลายบรรทัด



จากรูปที่ 1-11 สมมติว่าคุณต้องการทำหมายเหตุตั้งแต่บรรทัดที่ 8-15 ให้คุณลากเมาส์ครอบคลุมข้อความไว้ก่อน จากนั้นคลิกปุ่ม บนแถบทูลบาร์ของ VS 2005 เพื่อทำหมายเหตุตามจำนวนบรรทัดที่คุณครอบคลุมไว้ ในกรณีที่ต้องการยกเลิกการทำหมายเหตุหลายบรรทัด ให้คุณคลิกปุ่ม ดังรูปที่ 1-12

รูปที่ 1-12
กรณียกเลิกการทำหมายเหตุหลายบรรทัด



การใช้ตัวดำเนินการ AndAlso (&&) และตัวดำเนินการ OrElse (||)

ในภาษา VB 2005 มีการใช้ตัวดำเนินการ AndAlso (&& ในภาษา VC# 2005) และ OrElse (|| ในภาษา VC# 2005) เหตุผลที่ใช้ตัวดำเนินการดังกล่าว ให้คุณดูตารางต่อไปนี้

นิพจน์ที่ 1	นิพจน์ที่ 2	ผลการทำงาน
True	True	True
True	False	False
False	True	False
False	False	False

ตารางข้างต้นเป็นผลการทำงานของตัวดำเนินการ AND เห็นได้ว่าถ้านิพจน์ที่ 1 มีค่าเป็น True เราต้องรอตรวจสอบนิพจน์ที่ 2 ว่าเป็น True หรือ False โดยถ้านิพจน์ที่ 2 เป็น True ผลการทำงานก็คือ เป็น True แต่ถ้านิพจน์ที่ 2 เป็น False ผลการทำงานก็จะเป็น False

ปัญหาก็คือ ถ้านิพจน์ตัวที่ 1 เป็น False ตัวดำเนินการ AND ยังคงกำหนดให้มีการตรวจสอบนิพจน์ที่ 2 ทั้งๆ ที่ไม่ว่านิพจน์ที่ 2 จะเป็น True หรือ False ก็ตาม ผลที่ได้ออกมาจะเหมือนกันคือ False เห็นได้ว่าเป็นการทำงานที่ไม่มีความจำเป็นแต่อย่างใด

ตัวดำเนินการ AndAlso เข้ามาแก้ไขปัญหาตรงส่วนนี้กล่าวคือ ถ้านิพจน์ตัวที่ 1 เป็น False แล้วจะไม่มีการตรวจสอบนิพจน์ที่ 2 อีก ก็จะทำให้ผลการตรวจสอบที่ได้เร็วกว่าตัวดำเนินการ AND นั่นเอง

นิพจน์ที่ 1	นิพจน์ที่ 2	ผลการทำงาน
True	True	True
True	False	True
False	True	True
False	False	False

ตารางข้างต้นเป็นผลการทำงานของตัวดำเนินการ OR เห็นได้ว่าถ้านิพจน์ที่ 1 มีค่าเป็น True แล้ว ไม่ว่านิพจน์ที่ 2 จะมีค่าเป็น True หรือ False ผลการตรวจสอบที่ได้ก็คือ True เสมอ

ข้อแตกต่างระหว่างตัวดำเนินการ OR กับตัวดำเนินการ OrElse ก็คือ ตัวดำเนินการ OR ต้องตรวจสอบทั้ง 2 นิพจน์เสมอ ส่วนตัวดำเนินการ OrElse ถ้านิพจน์ที่ 1 มีค่าเป็น True แล้ว จะไม่มีการตรวจสอบค่าของนิพจน์ที่ 2 แต่อย่างใด ก็จะทำให้ผลการเปรียบเทียบที่ได้ออกมาเร็วกว่าตัวดำเนินการ OR นั่นเอง

การใช้ตัวดำเนินการทั้ง 2 ตัวของภาษา VB 2005 และ VC# 2005 แสดงดังต่อไปนี้

ภาษา VB 2005	ภาษา VC# 2005
AndAlso	&&
OrElse	||

สรุปท้ายบท

เนื้อหาในบทแรกยังไม่ได้กล่าวถึงเนื้อหาการเขียนโปรแกรมแบบ OOP แต่อย่างใด เป็นเพียงข้อควรรู้ในเบื้องต้นก่อนเข้าสู่โลกของ OOP ซึ่งจะเริ่มตั้งแต่บทที่ 2 เป็นต้นไป


Programming in OOP style
Advanced .net

CHAPTER ▸ 2

ตัวแปรและ Type ใน .NET Framework

บทนำ

สิ่งที่เราควรทำความรู้จักเป็นลำดับแรกนั่นคือ ตัวแปร (Variable) และชนิดของข้อมูล (Type) คุณจะได้ศึกษาตัวแปร และระบบ Type ของ .NET Framework ว่ามีโครงสร้างเป็นอย่างไร มีอะไรซ่อนอยู่เบื้องหลังเมื่อคุณประกาศตัวแปรขึ้นมาใช้งาน รวมทั้งเทคนิคการใช้งานตัวแปรที่คุณอาจไม่เคยทราบมาก่อน

การประกาศตัวแปร

ก่อนที่คุณจะใช้งานตัวแปรใดๆ ก็ตาม จะต้องมีการประกาศตัวแปร (Variable Declaration) เพื่อระบุชนิดของข้อมูลให้กับตัวแปรของคุณก่อน ซึ่งก็ขึ้นอยู่กับว่าคุณต้องการให้ตัวแปรนั้นๆ เก็บข้อมูลประเภทใด ดังนี้

VB 2005	VC# 2005
Dim a As Integer Dim b As Double Dim s As String	int a; double b; string c;

ตัวแปร a มี Type เป็น Integer (int), ตัวแปร b มี Type เป็น Double (double) และตัวแปร s มี Type เป็น String (string) ตัวแปรที่คุณสร้างขึ้นมาสามารถกำหนดค่าเริ่มต้นได้เช่นกัน ดังนี้

VB 2005	VC# 2005

Dim a As Integer = 10
Dim b As Double = 5.5
Dim s As String = "ข้อความเริ่มต้น"
	int a = 10;
	double b = 5.5;
	string c = "ข้อความเริ่มต้น";

NOTE

เนื่องจากว่าเนื้อหาของหนังสือเล่มนี้ นำเสนอทั้งภาษา VB 2005 และ VC# 2005 ในเวลาเดียวกัน จึงไม่สะดวกต่อการอ้างอิง Type อย่างสมบูรณ์แบบทั้ง 2 ภาษา ผู้เขียนจะใช้วิธีระบุรูปแบบของภาษา VB 2005 ก่อน แล้วตามด้วยรูปแบบของภาษา VC# โดยใช้วิธีใส่ในวงเล็บกำกับตามหลังไว้

ขอบเขตของตัวแปร (Variable Scope)

หลังจากที่ประกาศตัวแปรที่ต้องการใช้งานแล้ว สิ่งหนึ่งที่ควรรับทราบก็คือ ตัวแปรดังกล่าวมีขอบเขตที่คุณสามารถเรียกใช้งานได้หรือไม่ ซึ่งมีขอบเขตอยู่ 3 ระดับคือ

- **ระดับ** Public หรือเรียกอีกอย่างว่าตัวแปรระดับฟอร์มก็ได้ ถ้าคุณประกาศตัวแปรชนิดนี้ไว้ในฟอร์ม ขอบเขตที่ได้คือ ทุกเหตุการณ์ที่อยู่ในฟอร์มนั้นๆ สามารถเรียกใช้งานได้ แต่ถ้าคุณประกาศตัวแปรชนิดนี้ไว้ในคลาส ขอบเขตที่ได้คือ คุณสามารถเรียกใช้ตัวแปรชนิดนี้ ณ ตำแหน่งใดของโปรเจ็กต์ก็ได้

NOTE

ตัวแปรที่คุณใช้งานในคลาสเราจะเรียกว่า ฟิลด์ (Fields) ไม่ใช่เรียกว่า ตัวแปร ผู้เขียนจะกล่าวถึงเรื่องนี้อีกครั้งเมื่อถึงเนื้อหาของการสร้างคลาส

การประกาศตัวแปรระดับ Public ในคลาส ผู้เขียนแนะนำว่าคุณควรเลือกใช้ตัวแปรลักษณะนี้ให้น้อยที่สุด ควรใช้เท่าที่จำเป็น เนื่องจากว่าในการพัฒนาแอพพลิเคชันขนาดใหญ่ การประการตัวแปรในระดับ Public ยากต่อการควบคุมค่า และยากต่อการตรวจสอบค่าที่ตัวมันเองจัดเก็บอยู่

- **ระดับ** Procedure หรืออาจจะเรียกว่า ระดับ Local ก็ได้ มีขอบเขตขนาดกลางเป็นระดับที่เหมาะสมกับการใช้งานมากที่สุด มีขอบเขตอยู่ในแต่ละเหตุการณ์
- **ระดับ** Block มีขอบเขตขนาดเล็กสุด มักจะใช้เก็บค่าชั่วคราวหรือตัวแปรใช้วนลูปเสียเป็นส่วนใหญ่ เช่น ตัวแปรที่อยู่ในบล็อกของคำสั่งต่างๆ เช่น if...else... เป็นต้น

สำหรับการเลือกใช้ตัวแปรระดับใดก็ตาม ไม่มีข้อกำหนดตายตัวหรือข้อบังคับใดๆ ทั้งสิ้น ทั้งนี้ขึ้นอยู่กับลักษณะของโปรเจ็กต์เสียมากกว่า ขอให้เลือกใช้ตามความเหมาะสม

คำว่า "เหมาะสม" หมายถึง การเรียกใช้งานตัวแปรทั้งหมดที่อยู่ในโปรเจ็กต์ของคุณ สิ่งหนึ่งที่ต้องคำนึงถึงก็คือ หน่วยความจำ (RAM) การเขียนโปรแกรมที่ดีจะต้องมีการใช้งานทรัพยากรระบบให้เหมาะสมกับความสามารถของตัวมันเอง

ขอบเขตของตัวแปรเป็นตัวบ่งบอกว่า เมื่อสิ้นสุดการทำงานของตัวแปรนั้นๆ แล้ว ก็จะคืนทรัพยากรที่จองมาใช้งานกลับสู่ระบบ ถ้าคุณเลือกใช้ขอบเขตและประเภทของตัวแปรได้อย่างเหมาะสมแล้ว โปรเจ็กต์ที่ได้ก็จะมีประสิทธิภาพในด้านการทำงานและการใช้ทรัพยากรระบบ

ตัวแปรระดับ Public และ Procedure

ให้คุณเขียนโค้ดดังนี้

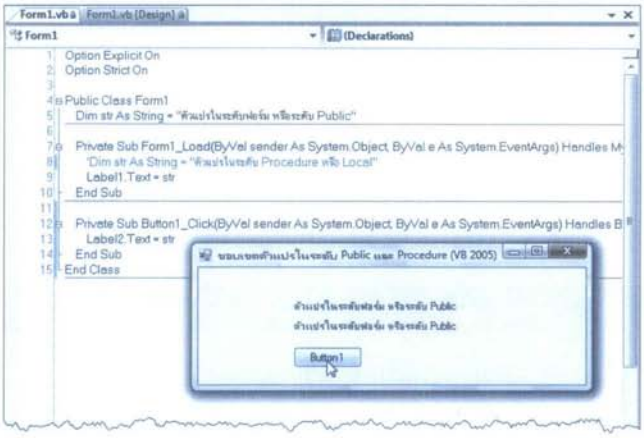
โค้ด VB 2005 ที่ 2-1 ขอบเขตตัวแปรระดับ Public และ Procedure (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Dim str As String = "ตัวแปรในระดับฟอร์ม หรือระดับ Public"

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'Dim str As String = "ตัวแปรในระดับ Procedure หรือ Local"
        Label1.Text = str
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Label2.Text = str
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 2-1 ขอบเขตตัวแปรระดับ Public และ Procedure (Form1.cs)

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    public string str = "ตัวแปรในระดับฟอร์ม หรือระดับ Public";

    private void Form1_Load(object sender, EventArgs e)
    {
        //string str ="ตัวแปรในระดับ Procedure หรือ Local";
        label1.Text = str;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        label2.Text = str;
    }
}
```

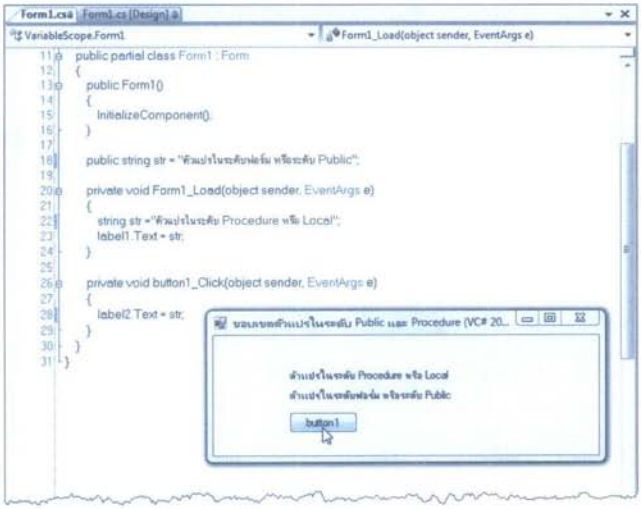
รูปที่ 2-1
ผลการรัน
ตัวอย่างที่ 2-1



จากรูปที่ 2-1 ผู้เขียนประกาศตัวแปรระดับ Public ในฟอร์มที่ชื่อว่า str เห็นได้ว่าคอนโทรล Label1 ถูกกำหนดค่าในเหตุการณ์ Form1_Load() ส่วนคอนโทรล Label2 ถูกกำหนดค่าในเหตุการณ์ Button1_Click() เห็นได้ว่าการประกาศตัวแปรในลักษณะนี้ คุณสามารถใช้งานตัวแปรตัวนี้ได้หลายเหตุการณ์

ให้คุณนำหมายเหตุออกแล้วลองรันตัวอย่างอีกครั้ง ดังรูปที่ 2-2

รูปที่ 2-2
หลังจากเอา
หมายเหตุออก



จากรูปที่ 2-2 ผู้เขียนสร้างตัวแปรอีก 1 ตัว มีขอบเขตระดับ Procedure (หรือระดับ Local) จงใจให้มีชื่อเดียวกับตัวแปรแบบ Public ข้างต้น พบว่าคอนโทรล Label1 แสดงข้อความของตัวแปรในระดับ Procedure สรุปได้ว่า

ถ้ามีตัวแปรชื่อเดียวกันทั้งในระดับ Public กับระดับ Procedure ตัวแปรที่อยู่ในระดับ Procedure จะมีความสำคัญมากกว่าตัวแปรที่อยู่ในระดับ Public

ตัวแปรระดับ Block

ให้คุณเขียนโค้ดดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 2-2 ตัวแปรระดับ Block

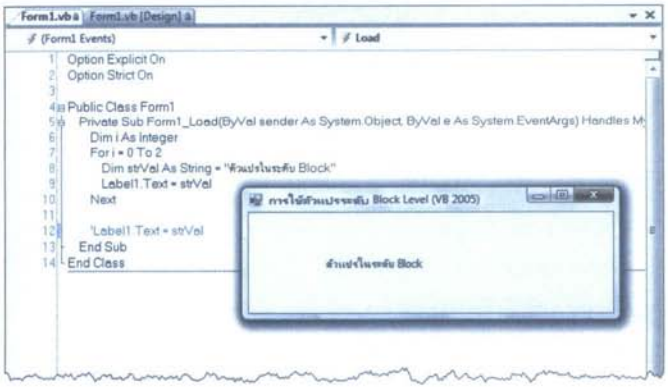
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim i As Integer
        For i = 0 To 2
            Dim strVal As String = "ตัวแปรในระดับ Block"
            Label1.Text = strVal
        Next

        'Label1.Text = strVal
    End Sub
End Class
```

VC# 2005 (Form1.cs)

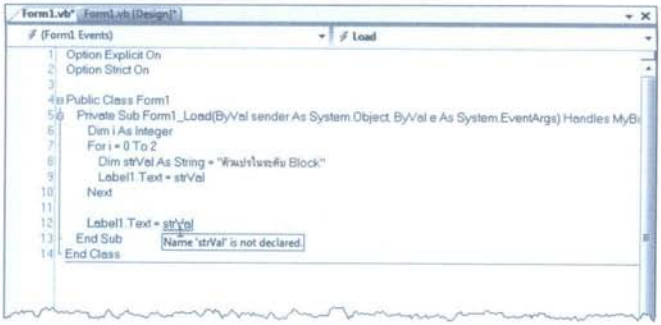
```
private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 0; i < 2; i++)
    {
        string strVal = "ตัวแปรในระดับ Block";
        label1.Text = strVal;
    }

    //label1.Text =strVal;
}
```

รูปที่ 2-3 ผลการรันตัวอย่างที่ 2-2



จากรูปที่ 2-3 ผู้เขียนประกาศตัวแปรที่ชื่อว่า strVal ในบล็อกคำสั่งของลูป For ส่งผลให้ตัวแปรตัวนี้มีขอบเขตการใช้งานในบล็อกของลูป For นี้เท่านั้น ให้คุณเอาหมายเหตุออก จะพบว่าคุณไม่สามารถอ่านค่าของตัวแปร strVal ตัวนี้นอกลูปได้เลย ดังรูปที่ 2-4

รูปที่ 2-4
การใช้งานตัวแปร strVal นอกบล็อกคำสั่ง



จากรูปที่ 2-4 ข้อผิดพลาดที่ปรากฏขึ้นมา หมายถึง ตัวแปร strVal ยังไม่มีการประกาศใช้งาน เพราะว่าเราเรียกใช้งานตัวแปรตัวนี้เกินขอบเขตของมันนั่นเอง

ชนิดของข้อมูลพื้นฐาน (Primitive Data Type)

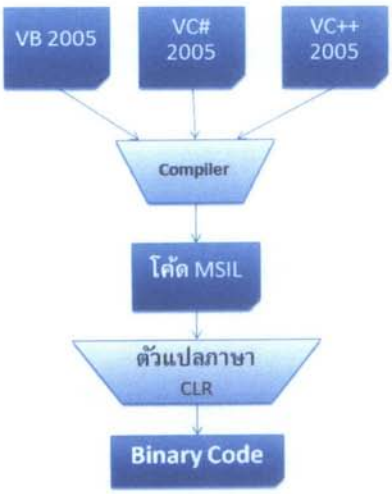
การเลือกใช้ชนิดของตัวแปร (Variables Type) ในโค้ดของคุณ ถือเป็นเรื่องพื้นฐานที่มีความสำคัญเป็นอย่างยิ่ง เพราะว่าใน .NET มองทุกสิ่งทุกอย่างเป็นออบเจ็กต์ ทุกซอกทุกมุมของ .NET Framework มีชนิดของข้อมูล (Type) ประจำตัวมันเองอย่างชัดเจน ขอให้คุณเลือกใช้ชนิดของตัวแปรให้เหมาะสม

โดยพื้นฐานทั้งหมดของภาษาในยุค .NET สิ่งที่คุณควรทำความเข้าใจมีอยู่ 2 อย่างคือ

1. **ทุกสิ่งทุกอย่างที่เรียกใช้งานในการเขียนโค้ดจะถือว่าเป็นออบเจ็กต์ (Object)** หรือเป็นวัตถุทั้งหมด ไม่ว่าจะเป็นตัวแปรที่คุณเรียกใช้, คลาสที่คุณสร้างขึ้นมาเอง, คลาสต่างๆ ของ .NET Framework ที่เรียกว่า Base Class Library (BCL) หรือแม้กระทั่งคอนโทรลที่คุณเรียกใช้งาน เป็นต้น ส่งผลให้ตัวมันเองมีคุณสมบัติและเมธอดประจำตัวติดตัวมาด้วยเสมอ
2. **ชนิดของข้อมูลพื้นฐาน (Primitive Data Type)** เช่น Integer (int), Long (long) เป็นต้น ถ้ามีการเปลี่ยนชนิดของข้อมูลไปจากเดิม เช่น Integer เป็น Double มีความสำคัญเป็นอย่างยิ่ง และมีข้อกำหนดหลายๆ อย่างที่คุณควรทราบในบางกรณี คุณไม่สามารถเปลี่ยนชนิดของข้อมูลตามที่คุณต้องการ ซึ่งผู้เขียนจะอธิบายอย่างละเอียดอีกครั้งในหัวข้อ Type Conversion

ความสำคัญอีกอย่างหนึ่งของภาษาต่างๆ ในยุค .NET ก็คือ ตัวแปลภาษาหรือที่เราเรียกว่า คอมไพเลอร์ (Compiler) จากอดีตที่ผ่านมาเราพบว่าแต่ละภาษาจะมีตัวแปลภาษาเป็นของตัวเอง เช่น Visual C++ ก็จะมีตัวแปลภาษาเป็นของตัวเอง, VB ก็มีตัวแปลภาษาเป็นของตัวเองเช่นกัน

แต่สำหรับภาษาต่างๆ ที่อยู่ใน .NET Framework แล้ว ไมโครซอฟท์ได้ปรับปรุงตัวแปลภาษาเปลี่ยนไปอย่างสิ้นเชิง โดยที่ไม่ว่าคุณจะพัฒนาแอพพลิเคชันด้วยภาษาใดก็ตาม จะอาศัยตัวแปลภาษาที่เรียกว่า CLR (Common Language Runtime) ทำหน้าที่แปลงโค้ดที่คุณเขียนไปสู่ภาษาที่ไมโครซอฟท์เรียกว่า IL (Intermediate Language) ดังรูปที่ 2-5



รูปที่ 2-5 ขั้นตอนการทำงานของ CLR

โค้ดที่คุณเขียนขึ้นมา ไม่ว่าจะมาจากภาษา VB 2005, VC# 2005 หรือภาษาใดๆ ก็ตามที่สนับสนุน .NET Framework จะถูกแปลเป็นภาษากลางที่เรียกว่า ภาษา IL ก่อน จากนั้นจะใช้ CLR เข้ามาทำหน้าที่แปลภาษา IL ให้เป็นภาษาเครื่องอีกทีหนึ่ง ตรงนี้เรียกว่า JIT (Just-In-Time)

จะเห็นได้ว่าด้วยหลักการทำงานของตัวแปลภาษา CLR ดังกล่าวสามารถตีความได้ว่า ในยุค .NET ไมโครซอฟท์ได้พัฒนาให้ทุกๆ ภาษาเข้าสู่จุดศูนย์กลางกล่าวคือ ไม่ว่าคุณจะพัฒนาแอพพลิเคชันด้วยภาษาใดก็ตาม ท้ายที่สุดแล้วก็จะได้โค้ด IL ที่พร้อมจะแปลเป็นภาษาเครื่องโดยอาศัย CLR ภาษาเครื่องที่ได้จาก IL ต้องรันภายใต้ .NET Runtime ส่งผลให้แอพพลิเคชันที่สร้างจาก .NET Framework จึงต้องติดตั้ง .NET Runtime ก่อนเสมอ

โค้ดของ .NET ที่ถูกแปลมาเป็นภาษา IL อยู่ภายใต้การควบคุมของ CLR เราจะเรียกว่า Managed Code หรือเป็นโค้ดที่ CLR ดูแลอยู่ สิทธิประโยชน์ของโค้ดกลุ่มนี้ที่ได้รับก็คือ จะมีผู้รักษาความสะอาดในหน่วยความจำ (RAM) ที่มีชื่อว่า Garbage Collection เรียกสั้นๆ ว่า GC เข้ามาทำหน้าที่กวาดล้างออบเจ็กต์ต่างๆ ที่ไม่ได้ใช้แล้วออกจากหน่วยความจำ เพื่อให้ทรัพยากรระบบถูกใช้ประโยชน์ได้อย่างมีประสิทธิภาพนั่นเอง

TIP

ในกรณีที่มีการเรียกใช้งานออบเจ็กต์ที่ถูกสร้างขึ้นมาก่อนยุคของ .NET เช่น กลุ่มออบเจ็กต์ของสถาปัตยกรรม Component Object Model - COM), กลุ่มออบเจ็กต์ของ Office 2003 และเวอร์ชันก่อนหน้านี้ ฯลฯ กลุ่มออบเจ็กต์เหล่านี้ CLR ไม่ได้เป็นผู้ดูแลหรือจัดการ เราเรียกโค้ดกลุ่มนี้ว่า Unmanaged Code เป็นโค้ดที่อยู่นอกเหนือการดูแลของ CLR ส่งผลให้ต้องมีการเขียนโค้ดเพื่อกำหนดให้คืนทรัพยากรสู่ระบบเอง เพราะโค้ดกลุ่มนี้อยู่นอกเหนือจากการทำงานของ Garbage Collection นั่นเอง ซึ่งผู้ดูแลโค้ดในกลุ่มนี้คือ Marshaling (มา-แชล-ลิ่ง)

เพราะความที่ทุกสิ่งทุกอย่างใน .NET Framework เป็นออบเจ็กต์ แต่ละออบเจ็กต์มี Type ระบุไว้อย่างชัดเจนใน CLR คำถามหนึ่งที่เกิดขึ้นมาก็คือ เมื่อคุณใช้ VB 2005 คุณต้องการใช้ข้อมูลชนิด Integer คุณก็ประกาศตามไวยากรณ์ของภาษา VB 2005 ดังนี้

VB 2005

```
Dim a As Integer
```

ส่วนภาษา VC# 2005 ก็จะประกาศดังนี้

VC# 2005

```
int a;
```

ภาษา VB 2005, VC# 2005 หรือภาษาใดๆ ก็ตามที่สนับสนุน .NET Framework คือ สิ่งที่ลอยอยู่เหนือ CLR เมื่อคุณต้องการระบุ Type ต่างๆ คุณต้องทำตามไวยากรณ์ของภาษานั้นๆ เมื่อมีการแปลโค้ดเป็นภาษา IL แล้ว CLR รู้ได้อย่างไรว่าในแต่ละภาษากำลังใช้ Type อะไรอยู่

จากโค้ดข้างต้นถ้าต้องการใช้ Type ชนิดเลขจำนวนเต็มในภาษา VB 2005 ต้องประกาศเป็น Integer ส่วนในภาษา VC# 2005 กลับประกาศเป็น int สิ่งที่เราสนใจคือ แล้วเลขจำนวนเต็มใน CLR คือ Type อะไร เราเรียก Type มาตรฐานร่วมว่า Common Type System หรือเรียกสั้นๆ ว่า CTS

คำตอบก็คือ ใน CLR มี Type มาตรฐานกลางถูกกำหนดให้ใช้ร่วมกันใน .NET Framework ไว้เพื่อให้ภาษาต่างๆ หยิบนำไปใช้นั่นเอง จะใช้โดยการเขียนไวยากรณ์ของภาษาใดก็แล้วแต่ ที่สำคัญ CLR ต้องรู้ว่า Type ที่ถูกเรียกใช้ตรงกับ Type มาตรฐานอะไรของ CLR ส่งผลให้คลาสหรือแอสเซมบลิ (Assembly) (มีนามสกุล *.dll) ที่ถูกสร้างขึ้นมาในยุคของ .NET Framework จึงมีความสามารถที่จะทำ Cross Language ได้

คำว่า Cross Language หมายถึง คุณสามารถสร้างแอสเซมบลิจากภาษา VB 2005 แต่นำไปใช้ในโปรเจ็กต์ของ VC# 2005 ได้ หรือจะทำกลับกันก็ได้เช่นกัน เห็นได้ว่าเป็นความสามารถที่ช่วยลดความขัดแย้งระหว่างภาษาได้อย่างสิ้นเชิง การตรวจสอบ Type ที่อยู่ใน CLR ให้ดูตัวอย่างที่ 2-3 Type ใน CLR

โค้ด VB 2005 ที่ 2-3 Type ใน CLR (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load
        Dim result As String = ""

        Dim sb As SByte
        result = "ค่า Min : " & SByte.MinValue.ToString() & Environment.NewLine
```

```
        result &= "ค่า Max : " & SByte.MaxValue.ToString() & Environment.NewLine
        result &= "Type : " & sb.GetType().ToString()
        MessageBox.Show(result.ToString(), "SByte")

        Dim sh As Short
        result = "ค่า Min : " & Short.MinValue.ToString() & Environment.NewLine
        result &= "ค่า Max : " & Short.MaxValue.ToString() & Environment.NewLine
        result &= "Type : " & sh.GetType().ToString()
        MessageBox.Show(result.ToString(), "Short")

        Dim i As Integer
        result = "ค่า Min : " & Integer.MinValue.ToString() & Environment.NewLine
        result &= "ค่า Max : " & Integer.MaxValue.ToString() & Environment.NewLine
        result &= "Type : " & i.GetType().ToString()
        MessageBox.Show(result.ToString(), "Integer")

        Dim l As Long
        result = "ค่า Min : " & Long.MinValue.ToString() & Environment.NewLine
        result &= "ค่า Max : " & Long.MaxValue.ToString() & Environment.NewLine
        result &= "Type : " & l.GetType().ToString()
        MessageBox.Show(result.ToString(), "Long")

        Dim b As Byte
        result = "ค่า Min : " & Byte.MinValue.ToString() & Environment.NewLine
        result &= "ค่า Max : " & Byte.MaxValue.ToString() & Environment.NewLine
        result &= "Type : " & b.GetType().ToString()
        MessageBox.Show(result.ToString(), "Byte")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 2-3 Type ใน CLR (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace TypeInCLR
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void Form1_Load(object sender, EventArgs e)
        {
            string result = "";

            SByte sb=0;
            result = "ค่า Min : " + SByte.MinValue.ToString() + Environment.NewLine;
            result += "ค่า Max : " + SByte.MaxValue.ToString() + Environment.NewLine;
            result += "Type : " + sb.GetType().ToString();
            MessageBox.Show(result.ToString(), "SByte");

            short sh = 0;
            result = "ค่า Min : " + short.MinValue.ToString() + Environment.NewLine;
            result += "ค่า Max : " + short.MaxValue.ToString() + Environment.NewLine;
            result += "Type : " + sh.GetType().ToString();
            MessageBox.Show(result.ToString(), "Short");

            int i = 0;
            result = "ค่า Min : " + int.MinValue.ToString() + Environment.NewLine;
            result += "ค่า Max : " + int.MaxValue.ToString() + Environment.NewLine;
            result += "Type : " + i.GetType().ToString();
            MessageBox.Show(result.ToString(), "Integer");

            long l = 0;
            result = "ค่า Min : " + long.MinValue.ToString() + Environment.NewLine;
            result += "ค่า Max : " + long.MaxValue.ToString() + Environment.NewLine;
            result += "Type : " + l.GetType().ToString();
            MessageBox.Show(result.ToString(), "Long");

            byte b = 0;
            result = "ค่า Min : " + byte.MinValue.ToString() + Environment.NewLine;
            result += "ค่า Max : " + byte.MaxValue.ToString() + Environment.NewLine;
            result += "Type : " + b.GetType().ToString();
            MessageBox.Show(result.ToString(), "Byte");
        }
    }
}
```

โค้ดบางส่วนข้างต้นเป็นการอ่านค่าต่ำสุด (MinValue), ค่าสูงสุด (MaxValue) และอ่าน Type ผ่านทางเมธอด GetType() เพื่อแสดง Type ของชนิดข้อมูลพื้นฐานต่างๆ ใน CLR สามารถสรุปเป็นตาราง ดังนี้

VB 2005 Type	VC# 2005 Type	.NET Framework Type (CLR Type)	ขนาดไบต์	ขอบเขตของข้อมูล
SByte	sbyte	System.Sbyte	1	-128 ถึง 127
Short	short	System.Int16	2	-32,768 ถึง 32,767
Integer	int	System.Int32	4	-2,147,483,648 ถึง 2,147,483,647
Long	long	System.Int64	8	-9,223,372,036,854,775,808 ถึง 9,223,372,036,854,775,807
Byte	byte	System.Byte	1	0 ถึง 255
UShort	ushort	System.UInt16	2	0 ถึง 65,535
UInteger	uint	System.UInt32	4	0 ถึง 4,294,967,295
ULong	ulong	System.UInt64	8	0 ถึง 18,446,744,073,709,551,615
Single	float	System.Single	4	-3.402823E+38 ถึง 3.402823E+38
Double	double	System.Double	8	-1.79769313486232E+308 ถึง 1.79769313486232E+308
Decimal	decimal	System.Decimal	12	-79,228,162,514,264,337,593, 543,950,335 ถึง 79,228,162, 514,264,337,593,543,950,335
Char	char	System.Char	2 ไบต์ต่อ 1 ตัวอักษร	เก็บตัวอักษรในรหัส Unicode ทั้งหมด
Boolean	bool	System.Boolean	1	จริง (True) หรือเท็จ (False)

คอลัมน์ที่น่าสนใจคือ .NET Framework Type (CLR Type) นั่นเอง ที่เป็นตัวบ่งบอกว่า CLR รู้จัก Type นั้นๆ ในชื่ออะไร โดยที่ Primitive Data Type เหล่านี้ถูกจัดอยู่ในกลุ่มที่เรียกว่า Value Type โดยมี Primitive Data Type เพียงตัวเดียวที่ถูกจัดอยู่ในกลุ่ม Reference Type นั่นคือ String

VB 2005	VC# 2005
Dim i As Integer	int i;

ตัวอย่างข้างต้นเป็นการประกาศตัวแปร i เป็นข้อมูลชนิด Integer ใช้พื้นที่ 4 ไบต์ตาม Type ที่ประกาศ โดยจะเริ่มใช้ทรัพยากรตามระดับของการประกาศตัวแปรนั้นๆ เช่น

ถ้าคุณประกาศตัวแปร i นี้ในระดับ Block Level ตัวแปร i จะเริ่มใช้ทรัพยากร 4 ไบต์ เมื่อ Block นั้นๆ เริ่มทำงาน เมื่อ Block ดังกล่าวสิ้นสุดการทำงานแล้ว ทรัพยากรของตัวแปร i นี้จะถูกคืนให้ระบบ

VB 2005	VC# 2005

For i As Integer = 0 To 10
Dim j As Long
 	
Next
	for (int i=0;i<10;i++)
	{
	long j;
	
	}

วงจรชีวิตของตัวแปร i และ j เริ่มใช้พื้นที่ 4 ไบต์และ 8 ไบต์ตามลำดับ เมื่อลูปนี้เริ่มทำงานหลังจากที่วนลูปจนครบตามเงื่อนไขแล้ว พื้นที่ทั้ง 12 ไบต์จะถูกคืนกลับเข้าสู่ระบบ

ทำนองเดียวกันในกรณีคุณประกาศตัวแปรในระดับ Procedure พื้นที่ของตัวแปร d ตัวนี้คือ 8 ไบต์ ถูกใช้เมื่อเกิดเหตุการณ์ Form1_Load() และจะคืน 8 ไบต์กลับสู่ระบบ เมื่อสิ้นสุดขอบเขตของมัน นั่นคือ End Sub หรือ } ของเหตุการณ์ดังกล่าวนั่นเอง

VB 2005	VC# 2005

Private Sub Form1_Load(...) Handles MyBase.Load
Dim d As Double
 	
End Sub
	private void Form1_Load(...)
	{
	double d;
	
	}

คำถามที่ตามมาก็คือ แล้ว Primitive Data Type เหล่านี้ถือเป็นออบเจ็กต์หรือไม่ คำตอบคือ ใช่ เพราะว่าทุกสิ่งทุกอย่างที่อยู่ใน .NET Framework เป็นออบเจ็กต์ (มาจาก System.Object) เพียงแต่ว่า Primitive Data Type เหล่านี้เป็น Value Type ไม่ใช่ Reference Type

NOTE

คำว่า Value Type และ Reference Type ผู้เขียนจะกล่าวอย่างละเอียดอีกครั้งในหัวข้อต่อๆ ไป

การรักษาชนิดของข้อมูล (Strong Type)

Type พื้นฐานต่างๆ ที่ผู้เขียนกล่าวในหัวข้อที่ผ่านมา ทำหน้าที่ระบุความชัดเจนเพื่อให้ CLR จัดการได้นั่นเอง เพื่อให้ได้โค้ดที่ทำงานได้อย่างมีประสิทธิภาพ ยังมีกติกาอีกอย่างหนึ่งที่เกี่ยวข้องก็คือ การรักษาชนิดของข้อมูลหรือที่เรียกทับศัพท์ว่า Strong Type หรือ Safe Type

ภาษา VC# 2005 เป็นภาษาที่ให้ความสำคัญกับการรักษาชนิดของข้อมูลเป็นอย่างยิ่ง ส่วนภาษา VB 2005 อาจจะมีบางส่วนที่ละเว้นไว้ เพื่อให้การเขียนโค้ดง่ายกว่าภาษา VC# 2005 นั่นเอง

ตัวแปรหรือออบเจ็กต์ที่คุณนำมาใช้ในโค้ดของคุณ ตั้งแต่เริ่มต้นจนจบการทำงานนั้นถ้าไม่มีความจำเป็นใดๆ ที่ต้องเปลี่ยนชนิดของข้อมูล ให้คุณพยายามเขียนโค้ดเพื่อรักษาชนิดข้อมูลของตัวแปรหรือออบเจ็กต์นั้นไว้ เพื่อป้องกันไม่ให้ประสิทธิภาพส่วนหนึ่งต้องลดต่ำลงไป เพราะต้องเสียเวลาช่วงหนึ่งไปแปลงชนิดข้อมูลก่อน แล้วค่อยมาทำงานหลักนั่นเอง

ในบางกรณีการไม่รักษาชนิดของข้อมูล อาจเกิดผลเสียให้การทำงานผิดไปจากความเป็นจริงอีกด้วย เป็นความผิดพลาดแบบ Logical Error โดยที่เราไม่ตั้งใจให้ดูตัวอย่างที่ 2-4 การรักษาชนิดของข้อมูลเพิ่มเติม

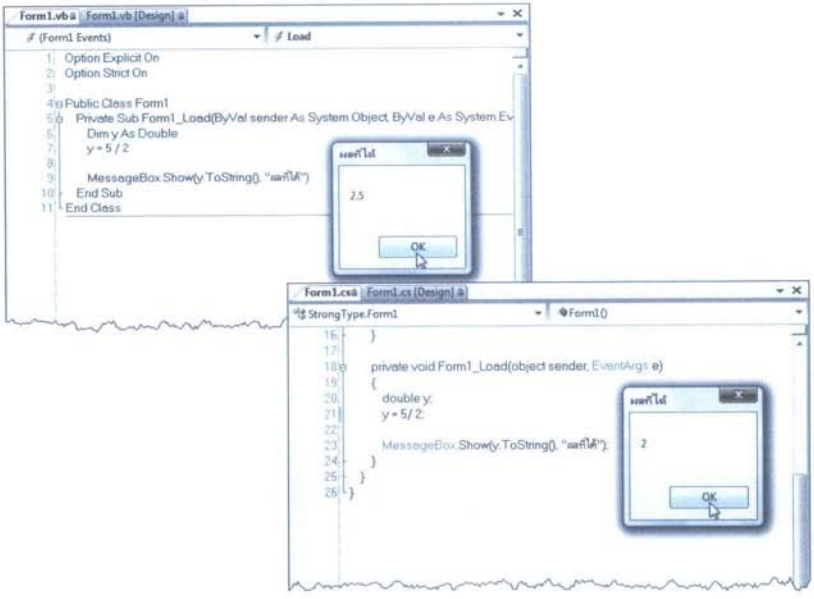
โค้ด VB 2005 และ VC# 2005 ที่ 2-4 การรักษาชนิดของข้อมูล

VB 2005 (Form1.vb)

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As
    System.Object, ByVal e As System.EventArgs)
    Handles MyBase.Load
        Dim y As Double
        y = 5 / 2

        MessageBox.Show(y.ToString(), "ผลที่ได้")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
    {
        double y;
        y = 5/ 2;

        MessageBox.Show(y.ToString(), "ผลที่ได้");
    }
```

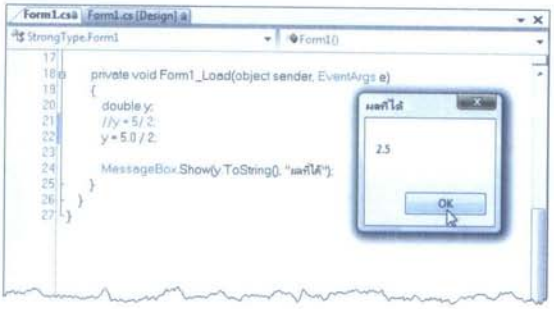


รูปที่ 2-6 ผลการรันตัวอย่างโค้ดที่ 2-4 ของ VB 2005 และ VC# 2005

จากรูปที่ 2-6 เห็นได้ว่าผลการคำนวณของ VC# 2005 ที่ได้ผิดไปจากความเป็นจริง ตรงนี้ไม่ใช่บั๊กของ VC# 2005 แต่อย่างใด สาเหตุก็คือ VC# 2005 เป็นภาษาที่รักษาชนิดของข้อมูลอย่างเคร่งครัด

เลข 5 ที่เป็นตัวตั้ง VC# 2005 มองว่าเป็นเลขจำนวนเต็มเมื่อหารด้วย 2 ผลหารดังกล่าวต้องเป็นเลขจำนวนเต็มด้วย จึงได้ผลหารเป็น 2 นั่นเอง ทางแก้ปัญหาก็คือ เมื่อคุณต้องการผลหารเป็นทศนิยมให้แก้ไขจาก 5 เป็น 5.0 ผลที่ได้ ดังรูปที่ 2-7

รูปที่ 2-7
หลังการแก้ไข



จากรูปที่ 2-7 เห็นได้ว่า VC# 2005 ยังคงรักษาชนิดของข้อมูลอย่างเคร่งครัดเช่นเดิม ตัวตั้งเป็นทศนิยม (5.0) ผลหารที่ได้ต้องเป็นทศนิยม ผลการคำนวณถูกต้องตามความเป็นจริง

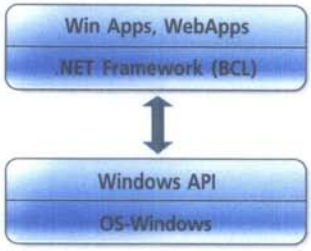
.NET Framework คืออะไร

คุณผู้อ่านที่เคยใช้ VB 6.0 ที่อยู่ในชุดเครื่องมือ Visual Studio 6.0 จะพบว่าในอดีตมีการเขียนโปรแกรมอยู่แขนงหนึ่งที่เรียกว่า Windows Application Programming Interface เรียกสั้นๆ ว่า Windows API เป็นการเรียกใช้ฟังก์ชันต่างๆ ของระบบปฏิบัติการ Windows เพื่อเพิ่มเติมความสามารถให้กับ VB 6.0

ปัญหาใหญ่อย่างหนึ่งของการเรียกใช้ฟังก์ชัน Windows API ก็คือ จะมีกลุ่มฟังก์ชันกลุ่มใหญ่ๆ อยู่กลุ่มหนึ่งที่ VB 6.0 ไม่สามารถเรียกใช้งานได้ สาเหตุก็คือ ชนิดของข้อมูลของฟังก์ชัน API นั้น VB 6.0 ไม่รู้จัก จึงเรียกใช้งานไม่ได้ อันเนื่องมาจากว่าฟังก์ชัน API ถูกเขียนขึ้นมาจากภาษา C และ C++ ส่งผลให้ชนิดของข้อมูลแตกต่างไปจากที่ VB 6.0 รู้จัก ซึ่งเป็นความขัดแย้งที่เกิดขึ้นระหว่างภาษา

ไมโครซอฟท์สร้างคลาสต่างๆ ขึ้นมาเรียกว่า Base Class Library (BCL) ประกอบกันขึ้นมาเป็น .NET Framework เพื่อทำหน้าที่รับผิดชอบด้านต่างๆ เช่น คิวรีข้อมูล, ส่งเมล, อ่านไฟล์, วาดรูป, วาดส่วนแสดงผล (User Interface-UI) เป็นต้น การทำงานแต่ละอย่าง .NET มีคลาสรองรับไว้

รูปที่ 2-8
ตำแหน่งของ .NET Framework



จากรูปที่ 2-8 เห็นได้ว่า BCL บังกลุ่มฟังก์ชัน API ไว้ทั้งหมด เพื่อให้ภาษาในยุค .NET สามารถเรียกใช้คลาส BCL ต่างๆ เหล่านี้ได้สะดวกรวดเร็วยิ่งขึ้น ช่วยลดเวลาการพัฒนาแอพพลิเคชันลงเป็นอย่างมาก ส่งผลให้มีคลาสจำนวนมากอยู่ใน .NET Framework ทำให้แอพพลิเคชันใดๆ ก็ตามที่ถูกพัฒนามาจากภาษาของ .NET จึงต้องมีการติดตั้ง .NET Framework ก่อนเสมอ เพื่อให้สามารถรันแอพพลิเคชันได้นั่นเอง

ผลที่ตามมาก็คือ บทบาทการเรียกใช้ฟังก์ชัน API ใน .NET จึงแทบจะหายไปจากสารระบบ เพราะว่า .NET มีคลาสรองรับไว้พร้อมถูกเรียกใช้เสมอเมื่อรวมกับการระบุ Type ใน CLR อย่างชัดเจน จึงทำให้ภาษาที่สนับสนุน .NET ไม่เกิดปัญหาเข้ากันไม่ได้ ระหว่างชนิดของข้อมูลในแต่ละภาษานั่นเอง

Type ใน .NET Framework

ใน .NET Framework สามารถแบ่ง Type ออกได้ 2 ประเภทคือ

1. **Value Type** ตัวแทนของ Type ประเภทที่เราคุ้นเคยกันดี ได้แก่ Primitive Data Type ต่างๆ ที่ใช้กันอยู่เสมอ เช่น Integer (int), Single (float), Double (double) เป็นต้น
2. **Reference Type** ตัวแทนของ Type ประเภทนี้ ได้แก่ String (string) และคลาสต้นแบบต่างๆ ที่คุณสร้างขึ้นมา

ข้อแตกต่างระหว่าง Value Type กับ Reference Type

ที่สำคัญๆ ได้แก่

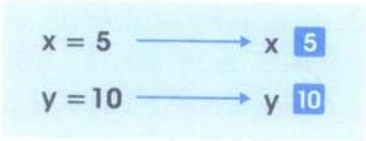
1. **Value Type** ใช้พื้นที่ในหน่วยความจำที่เรียกว่า stack (สแต็ค) และเก็บค่าไว้กับตัวมันเอง เช่น

VB 2005	VC# 2005
Dim x As Integer = 5 Dim y As Integer = 10	int x = 5; int y = 10;

โค้ดข้างต้นเป็นการสร้างตัวแปรขึ้นมา 2 ตัว ชื่อว่า x กับ y มี Type เป็น Integer (int) กล่าวได้อีกนัยหนึ่งคือ ตัวแปร x กับตัวแปร y เป็น ValueType โดยที่ตัวแปร x เก็บค่า 5 ส่วนตัวแปร y เก็บค่า 10 ตัวแปรทั้ง 2 ตัวไม่มีความเกี่ยวข้องใดๆ ต่อกัน

ตัวแปร x และ y ใช้พื้นที่ 4 ไบต์ในหน่วยความจำตาม Type ที่ประกาศคือ Integer (int) วิธีการคืนพื้นที่หน่วยความจำที่ใช้ขึ้นอยู่กับขอบเขตการประกาศใช้งาน ตามที่ผู้เขียนเคยกล่าวไว้ในตัวอย่าง Type ใน CLR ที่ผ่านมา

รูปที่ 2-9
แบบจำลองของ
Value Type



จากรูปที่ 2-9 เห็นได้ว่าทั้งตัวแปร x และ y ต่างเก็บค่าไว้ที่ตัวมันเอง

2. **Reference Type** ใช้พื้นที่ในหน่วยความจำที่เรียกว่า heap (ฮีบ) Type ประเภทนี้ไม่มีการเก็บค่าไว้กับตัวเอง แต่จะเก็บตัวชี้ไปยังออบเจ็กต์ เช่น

VB 2005	VC# 2005
Dim s As String = 'ข้อความต่างๆ'	string s = 'ข้อความต่างๆ';

โค้ดข้างต้นเป็นการประกาศตัวแปรที่ชื่อว่า s มี Type เป็น String (string) กำหนดข้อความเริ่มต้นไว้ด้วย กล่าวได้อีกนัยหนึ่งคือ ตัวแปร s เป็น Reference Type ทำหน้าที่ชี้ไปยังข้อความ "ข้อความต่างๆ"

รูปที่ 2-10
แบบจำลองของ
Reference Type

จากรูปที่ 2-10 ตัวแปร s มี Type เป็น String (string) พร้อมที่จะชี้ออบเจ็กต์ที่มี Type เป็น String (string)

VB 2005	VC# 2005

Public Class Customer
 	
End Class
	public class Customer
	{
	
	}

โค้ดข้างต้นสร้างคลาสที่ชื่อว่า Customer ขึ้นมากล่าวได้อีกนัยหนึ่ง คลาส Customer คือ Reference Type เมื่อคุณต้องการใช้งานคลาส Customer ใน VB 2005 สามารถทำได้ 3 แบบ ส่วน VC# 2005 ทำได้ 2 แบบ

การประกาศแบบที่ 1 และ 2 ของ VB 2005 หรือแบบที่ 1 ของ VC# 2005 เป็นการสร้างตัวแปรที่ชื่อว่า c มี Type เป็น Customer และสร้างออบเจ็กต์ Customer ขึ้นมาด้วยคำสั่ง New (new) จากนั้นสั่งให้ตัวแปร c ชี้ไปยังออบเจ็กต์ Customer ที่เกิดขึ้นมา หรือกล่าวโดยรวมได้ว่าสร้างออบเจ็กต์ Customer ที่ชื่อว่า c ขึ้นมา ดังโค้ดต่อไปนี้

VB 2005	VC# 2005
Dim c As New Customer() หรือ Dim c As Customer = New Customer()	Customer c = new Customer();

ส่วนการประกาศแบบที่ 3 ของ VB 2005 หรือแบบที่ 2 ของ VC# 2005 มี 2 จังหวะคือ

- ประกาศตัวแปร c มี Type เป็น Customer
- สร้างออบเจ็กต์ Customer ขึ้นมาด้วยคำสั่ง New (new) จากนั้นสั่งให้ตัวแปร c ชี้ไปยังออบเจ็กต์ Customer ที่เกิดขึ้นมา

VB 2005	VC# 2005
Dim c As Customer c = New Customer()	Customer c; c = new Customer();

.NET จะใช้ Garbage Collection เรียกสั้นๆ ว่า GC เข้ามาทำหน้าที่ตรวจสอบว่าออบเจ็กต์ต่างๆ ที่เกิดขึ้นมา มีออบเจ็กต์ตัวใดบ้างที่ไม่ถูกเรียกใช้งาน (ไม่ถูกชี้) เห็นได้ว่า GC เข้ามาทำหน้าที่คืนหน่วยความจำของออบเจ็กต์ต่างๆ ที่ไม่ใช้แล้วให้เราโดยอัตโนมัติ

การทำงานของ GC ช่วยให้แอพพลิเคชันที่เกิดจาก .NET ไม่เกิดปัญหาสูญเสียหน่วยความจำไปโดยเปล่าประโยชน์ โดยที่ GC จะไม่เข้าไปยุ่งเกี่ยวกับออบเจ็กต์ที่ยังคงมีการใช้งานอยู่ (ถูกชี้อยู่)

จากที่ผู้เขียนเคยกล่าวไว้ว่า ใน .NET Framework มองทุกสิ่งทุกอย่างเป็นออบเจ็กต์ โดยที่แต่ละออบเจ็กต์มี Type ระบุไว้อย่างชัดเจน การตรวจสอบประเภท Type ใน .NET Framework ให้ดูตัวอย่างที่ 2-5 การตรวจสอบประเภท Type ใน .NET Framework

โค้ด VB 2005 ที่ 2-5 การตรวจสอบประเภท Type ใน .NET Framework (Form1.vb)

```
Option Explicit On
Option Strict On
Imports System

Public Class SimpleClass

End Class

Public Structure SimpleStructure
    Private _test As Integer

End Structure

Public Enum VisualStudio2005
    VB2005
    VCSharp2005
End Enum
```

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
        Handles MyBase.Load
        Dim result As Boolean = False

        Dim TestInteger As Integer = 0
        result = IsValueType(TestInteger)
        MessageBox.Show("TestInteger เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim TestString As String = ""
        result = IsValueType(TestString)
        MessageBox.Show("TestString เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim arrInteger() As Integer = {1, 2, 3}
        result = IsValueType(arrInteger)
        MessageBox.Show("arrInteger เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim arrString() As String = {"ทดสอบ 1", "ทดสอบ 2", "ทดสอบ 3"}
        result = IsValueType(arrString)
        MessageBox.Show("arrString เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim TestEnum As VisualStudio2005
        result = IsValueType(TestEnum)
        MessageBox.Show("TestEnum เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim testObj As Object = Nothing
        result = IsValueType(testObj)
        MessageBox.Show("testObj เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim TestClass As New SimpleClass()
        result = IsValueType(TestClass)
        MessageBox.Show("TestClass เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

        Dim TestStructure As New SimpleStructure()
        result = IsValueType(TestStructure)
        MessageBox.Show("TestStructure เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")
    End Sub

    Public Function IsValueType(ByVal anyType As Object) As Boolean
        Return (TypeOf (anyType) Is ValueType)
    End Function
End Class
```

โค้ด VC# 2005 ที่ 2-5 การตรวจสอบประเภท Type ใน .NET Framework (Form1.cs)

```
public class SimpleClass
{

}

public struct SimpleStructure
{
    private int _test;

}

public enum VisualStudio2005
{
    VB2005,
    VCSharp2005
}

private void Form1_Load(object sender, EventArgs e)
{
    bool result = false;

    int TestInteger = 0;
    result = IsValueType(TestInteger);
    MessageBox.Show("TestInteger เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

    string TestString = "";
    result = IsValueType(TestString);
    MessageBox.Show("TestString เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

    int[] arrInteger = new int[] { 1, 2, 3 };
    result = IsValueType(arrInteger);
    MessageBox.Show("arrInteger เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

    string[] arrString = new string[] { "ทดสอบ 1", "ทดสอบ 2", "ทดสอบ 3" };
    result = IsValueType(arrString);
    MessageBox.Show("arrString เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

    VisualStudio2005 testEnum = VisualStudio2005.VB2005;
    result = IsValueType(testEnum);
    MessageBox.Show("testEnum เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

    object testObj = null;
    result = IsValueType(testObj);
    MessageBox.Show("testObj เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

    SimpleClass TestClass = new SimpleClass();
    result = IsValueType(TestClass);
```

```
        MessageBox.Show("TestClass เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

        SimpleStructure TestStructure = new SimpleStructure();
        result = IsValueType(TestStructure);
        MessageBox.Show("TestStructure เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");
    }

    public bool IsValueType(object anyType)
    {
        return ((anyType) is ValueType);
    }
```

ออบเจ็กต์ต่างๆ ที่ประกอบขึ้นมาเป็น .NET Framework มีมากมาย ผู้เขียนยกตัวอย่างออบเจ็กต์บางส่วนที่น่าสนใจ มาตรวจสอบว่าออบเจ็กต์เหล่านี้เป็น Value Type หรือ Reference Type วิธีการดูโค้ดชุดนี้ก็คือ

- สร้างคลาสที่ชื่อว่า SimpleClass เป็นตัวแทน Class
- สร้างสตรัคเจอร์ที่ชื่อว่า SimpleStructure เป็นตัวแทน Structure (struct)
- สร้าง Enum ที่ชื่อว่า VisualStudio2005 ประกอบด้วยค่าคงที่ 2 ตัวเป็นตัวแทน Enum

VB 2005

```
Public Class SimpleClass

End Class

Public Structure SimpleStructure
    Private _test As Integer

End Structure

Public Enum VisualStudio2005
    VB2005
    VCSharp2005
End Enum
```

VC# 2005

```
public class SimpleClass
{

}

public struct SimpleStructure
{
    private int _test;

}

public enum VisualStudio2005
{
    VB2005,
    VCSharp2005
}
```

การตรวจสอบความเป็น Value Type หรือ Reference Type อยู่ในความรับผิดชอบของเมธอด IsValueType() ต้องการพารามิเตอร์ 1 ตัวคือ anyType หมายถึง ออบเจ็กต์ที่ต้องการตรวจสอบนั่นเอง

กำหนดให้พารามิเตอร์ anyType มี Type เป็น Object เพราะว่าออบเจ็กต์ต่างๆ ที่อยู่ใน .NET Framework ล้วนแต่เกิดมาจาก System.Object นั่นเอง โดยที่

- คืนค่าเป็น True ถ้าพารามิเตอร์ anyType เป็น Value Type
- คืนค่าเป็น False ถ้าพารามิเตอร์ anyType เป็น Reference Type

VB 2005

```
Public Function IsValueType(ByVal anyType As Object)
As Boolean
    Return (TypeOf (anyType) Is ValueType)
End Function
```

VC# 2005

```
public bool IsValueType(object anyType)
{
    return ((anyType) is ValueType);
}
```

รูปที่ 2-11 ผลการตรวจสอบ Integer (int) กับ String (string)

จากรูปที่ 2-11 Integer (int) และ String (string) เป็นตัวแทนของ Primitive Data Type ผลการตรวจสอบที่ได้คือ

- Primitive Data Type เป็น Value Type
- String (string) เป็น Reference Type

VB 2005

```
Dim TestInteger As Integer = 0
result = IsValueType(TestInteger)
MessageBox.Show("TestInteger เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

Dim TestString As String = ""
result = IsValueType(TestString)
MessageBox.Show("TestString เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")
```

VC# 2005

```
int TestInteger = 0;
result = IsValueType(TestInteger);
MessageBox.Show("TestInteger เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

string TestString = "";
result = IsValueType(TestString);
MessageBox.Show("TestString เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");
```

รูปที่ 2-12 ผลการตรวจสอบ Class กับ Structure (struct)

จากรูปที่ 2-12 คลาสต้นแบบที่คุณสร้างขึ้นมาเป็น Reference Type ส่วนสตรัคเจอร์เป็น Value Type

VB 2005

```
Dim TestClass As New SimpleClass()
result = IsValueType(TestClass)
MessageBox.Show("TestClass เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")

Dim TestStructure As New SimpleStructure()
result = IsValueType(TestStructure)
MessageBox.Show("TestStructure เป็น ValueType ใช่หรือไม่? " & result.ToString(), "ผลการตรวจสอบ")
```

VC# 2005

```
SimpleClass TestClass = new SimpleClass();
result = IsValueType(TestClass);
MessageBox.Show("TestClass เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");

SimpleStructure TestStructure = new SimpleStructure();
result = IsValueType(TestStructure);
MessageBox.Show("TestStructure เป็น ValueType ใช่หรือไม่? " + result.ToString(), "ผลการตรวจสอบ");
```

ผลการตรวจสอบของตัวอย่างที่ 2-5 แสดงดังตารางต่อไปนี้

ประเภทออบเจ็กต์	Value Type	Reference Type
Primitive Data Type (ยกเว้น String)	Yes	-
String (string)	-	Yes
อาร์เรย์ (Integer)	-	Yes
อาร์เรย์ (String)	-	Yes
Enum	Yes	-
System.Object	-	Yes
Class	-	Yes
Structure (struct)	Yes	-

การแปลง Type ใน .NET Framework

ในการพัฒนาแอพพลิเคชันโดยอาศัย .NET Framework มีการกระทำอยู่อย่างหนึ่งที่มักจะเกิดขึ้นเสมอ นั่นคือ การแปลงจาก Type หนึ่งไปสู่อีก Type หนึ่ง เราสามารถแยกการแปลง Type ดังกล่าวได้ 2 ลักษณะคือ

1. การแปลงจาก Type หนึ่งไปสู่อีก Type หนึ่ง การแปลงข้อมูลแบบนี้ยังแบ่งได้อีก 2 รูปแบบคือ
   - Implicit Conversion หมายถึง การแปลงข้อมูลโดยนัย หรือโดยอัตโนมัติ
   - Explicit Conversion หมายถึง การบังคับให้เกิดการแปลงข้อมูล
2. การทำ Boxing และ Unboxing กล่าวคือ
   - การแปลงจาก Value Type ไปสู่ Reference Type เรียกว่าการทำ Boxing
   - การแปลงจาก Reference Type ไปสู่ Value Type เรียกว่าการทำ Unboxing

การแปลงข้อมูลโดยอัตโนมัติ (Implicit Conversion)

เกิดจากการที่ข้อมูลแต่ละชนิดมีขอบเขต และความสามารถในการจัดเก็บข้อมูลในแต่ละชนิดไม่เท่ากัน เมื่อมีการแปลงข้อมูลแล้ว ชนิดของข้อมูลดังกล่าวสามารถยอมรับได้ ก็จะสามารถแปลงชนิดของข้อมูลได้โดยอัตโนมัติ ผู้เขียนขอเรียกว่า Implicit Conversion

การแปลงข้อมูลแบบ Implicit ยังแยกออกมาได้อีก 2 กรณีคือ

- **กรณีที่** 1 : **การแปลงชนิดของข้อมูลขนาดเล็กกว่าไปสู่ชนิดของข้อมูลที่ใหญ่กว่า** การแปลงข้อมูลแบบนี้จะอาศัยนัยของชนิดข้อมูลเป็นตัวกำหนด

ชนิดข้อมูลของ VB 2005	ชนิดข้อมูลของ VC# 2005	นัยของข้อมูล
Short	short	ต่ำ ↓ สูง
Integer	int	
Long	long	
Single	float	
Double	double	

ตารางข้างต้นเป็นการแสดงนัยชนิดของข้อมูลที่น่าสนใจ เห็นได้ว่าข้อมูลชนิด Integer (VB 2005) หรือ int (VC# 2005) มีนัยต่ำที่สุด ส่งผลให้การแปลง Integer (int) ไปสู่ Long (long), Single (float) หรือ Double (double) สามารถทำได้โดยที่ข้อมูลดังกล่าวไม่สูญเสียค่า, สูญหาย หรือเปลี่ยนแปลงไป

- **กรณีที่** 2 : **การแปลงชนิดของข้อมูลที่เกิดจากการกำหนดค่าเริ่มต้นให้กับตัวแปร** เป็นการแปลงชนิดของข้อมูลที่เกิดขึ้นในกรณีที่คุณสร้างตัวแปรขึ้นมา แล้วกำหนดค่าเริ่มต้นให้กับตัวแปรดังกล่าว โดยที่คุณกำหนดชนิดข้อมูลเริ่มต้นต่างกับชนิดข้อมูลของตัวแปรนั้นๆ โดยที่ตัวแปรดังกล่าวยอมรับได้ กรณีนี้จะถือว่าเป็นการแปลงข้อมูลแบบ Implicit เช่นกัน การแปลงข้อมูลแบบ Implicit ให้ดูตัวอย่างที่ 2-6 การแปลงข้อมูลแบบ Implicit Conversion

โค้ด VB 2005 และ VC# 2005 ที่ 2-6 การแปลงข้อมูลแบบ Implicit Conversion

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim a As Long
        Dim b As Integer = 50
        a = b

        Dim x As Short = 1
        Dim y As Integer = 2
        Dim z As Integer = x + y

        Dim i As Double = 1.5F
        Dim s As String = i.ToString()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    long a;
    int b = 50;
    a = b;

    short x = 1;
    int y = 2;
    int z = x + y;

    double i = 1.5F;
    string s = i.ToString();
}
```

วิธีการดูตัวอย่างนี้ก็คือ จากกรณีที่ 1 ที่ผู้เขียนบอกว่าเป็นการแปลงชนิดของข้อมูลขนาดเล็กกว่า ไปสู่ชนิดของข้อมูลที่ใหญ่กว่า ผู้เขียนสร้างตัวแปร a เป็นข้อมูลชนิด Long (long) มีนัยสูงกว่าตัวแปร b ที่ประกาศเป็นข้อมูลชนิด Integer (int) กำหนดค่าเริ่มต้นเป็นเลขจำนวนเต็ม 50

จากนั้นผู้เขียนกำหนดให้ค่าของ b เท่ากับค่าของ a เห็นได้ว่า a และ b เป็นตัวแปรที่มีชนิดของข้อมูลต่างกัน โดยที่ b เล็กกว่าคุณสามารถนำค่าของ b ไปกำหนดให้กับ a ได้เลย เพราะว่าชนิดข้อมูลของตัวแปร a ยอมรับค่าของตัวแปร b ได้นั่นเอง ส่งผลให้ตัวแปร a เก็บค่า 50 ได้โดยไม่สูญเสียค่าแต่อย่างใด

VB 2005

```
Dim a As Long
Dim b As Integer = 50
a = b

Dim x As Short = 1
Dim y As Integer = 2
Dim z As Integer = x + y
```

VC# 2005

```
long a;
int b = 50;
a = b;

short x = 1;
int y = 2;
int z = x + y;
```

อีกกรณีหนึ่งผู้เขียนสร้างตัวแปร z เป็นข้อมูลชนิด Integer (int) กำหนดให้รับผลบวกจากตัวแปร x และตัวแปร y พบว่าผลบวกของตัวแปรทั้ง 2 คือ 3 เป็นสิ่งที่ตัวแปร z ยอมรับได้ แม้ว่าตัวแปร x จะแตกต่างจากตัวแปร z ก็ตาม เพราะว่านัยของตัวแปร x ต่ำกว่านัยของตัวแปร z นั่นเอง จึงเกิดการแปลงข้อมูลโดยอัตโนมัตินั่นเอง

ส่วนกรณีที่ 2 ที่ผู้เขียนกล่าวว่าเป็นการแปลงชนิดของข้อมูลที่เกิดจากการกำหนดค่าเริ่มต้นให้กับตัวแปร ให้คุณดูตัวแปร i เป็นข้อมูลชนิด Double (double) กำหนดค่าเริ่มต้นให้กับตัวแปร i เท่ากับ 1.5 แต่ตัวเลข 1.5 ดังกล่าวผู้เขียนกำหนดให้เป็นชนิด float โดยการใช้อักษรควบคุม f (หรือ F)

ส่งผลให้ชนิดของตัวแปร i คือ Double (double) แตกต่างจากค่าเริ่มต้นคือ Float แต่ตัวแปร i สามารถเก็บค่า 1.5 ไว้ได้ เพราะว่านัยของ Double (double) สูงกว่า Float นั่นเอง

VB 2005

```
Dim i As Double = 1.5F
Dim s As String = i.ToString()
```

VC# 2005

```
double i = 1.5F;
string s = i.ToString();
```

การแปลงชนิดของข้อมูลอีกลักษณะก็คือ เราสามารถแปลงให้อยู่ในฐานะข้อมูลชนิด String ได้เช่นกัน โดยการใช้เมธอด ToString() นั่นเอง

การแปลงข้อมูลแบบ Explicit Conversion

เรียกอีกอย่างว่าการ Casting เป็นการแปลงข้อมูลที่เกิดจากคุณเป็นผู้กำหนดให้เกิดการแปลงข้อมูลหรือบังคับให้แปลงข้อมูล อาจจะทำให้ค่าของข้อมูลสูญเสียหรือเปลี่ยนแปลงไป ผู้เขียนขอเรียกสั้นๆ ว่า การแปลงข้อมูลแบบ Explicit Conversion มีอยู่ 3 วิธีคือ

- อาศัยออบเจ็กต์ Convert
- อาศัยออบเจ็กต์ชนิดของข้อมูล
- อาศัยเมธอด Parse() ของออบเจ็กต์ชนิดของข้อมูล

ให้ดูตัวอย่างที่ 2-7 การแปลงข้อมูลแบบ Explicit Conversion

โค้ด VB 2005 และ VC# 2005 ที่ 2-7 การแปลงข้อมูลแบบ Explicit Conversion

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim x As Double = 1.5
        'Dim y As Integer = x

        'Dim y As Integer = Convert.ToInt32(x)
        'Dim y As Integer = CInt(x)
        'MessageBox.Show(y.ToString())

        Dim s1 As String = "100"
        Dim s2 As String = "200"
        Dim z As Integer = 0
        z = Integer.Parse(s1) + Integer.Parse(s2)
        MessageBox.Show(z.ToString())
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    double x = 1.5;
    //int y = x;

    //int y = (int)x;
    //MessageBox.Show(y.ToString());

    //int y = Convert.ToInt32(x);
    //MessageBox.Show(y.ToString());

    string s1 = "100";
    string s2 = "200";
    int z;
    z = int.Parse(s1) + int.Parse(s2);
    MessageBox.Show(z.ToString());
}
```

วิธีการดูโค้ดก็คือ เริ่มต้นประกาศตัวแปร x เป็นข้อมูลชนิด Double (double) ซึ่งเป็นข้อมูลที่มีนัยสูง กำหนดค่าเริ่มต้นคือ 1.5 ต่อมาประกาศตัวแปร y เป็นข้อมูลชนิด Integer (int) เก็บเลขจำนวนเต็มเป็นข้อมูลที่มีนัยต่ำกว่า

VB 2005	VC# 2005
Dim x As Double = 1.5 Dim y As Integer = x	double x = 1.5; int y = x;

จากนั้นให้คุณลองกำหนดให้ x (นัยสูงกว่า) ส่งค่าให้กับตัวแปร y (นัยต่ำกว่า) พบว่าคุณไม่สามารถทำได้ เพราะว่าค่าเริ่มต้นของ x มีความละเอียดในระดับจุดทศนิยม ตัวแปร y ไม่สามารถเก็บได้ส่งผลให้เกิดข้อผิดพลาด ดังรูปที่ 2-13

รูปที่ 2-13
ข้อความผิดพลาดของ VC# 2005

คุณสามารถใช้การแปลงข้อมูลแบบ Explicit หรือ Cast ตัวแปร x เพื่อบังคับให้มีการเปลี่ยนแปลงข้อมูลดังนี้

VB 2005	VC# 2005
Dim y As Integer = Convert.ToInt32(x) Dim y As Integer = CInt(x) MessageBox.Show(y.ToString())	int y = (int)x; MessageBox.Show(y.ToString());

ใน VB 2005 และ VC# 2005 คุณอาจจะใช้ออบเจ็กต์ Convert ร่วมกับเมธอด ToInt32() เพื่อแปลงเป็นข้อมูลชนิด Integer (int) ก็ได้ โดยที่ใน VB 2005 คุณยังสามารถใช้ฟังก์ชัน CInt() เพิ่มเติมได้อีก 1 วิธี

แม้ว่าคุณสามารถแปลงข้อมูลได้ก็จริงแต่ก็ไม่ได้มาฟรี สิ่งที่เสียไปก็คือ ค่าของข้อมูลเพราะว่าตัวแปร y ไม่ได้เก็บค่าเดียวกับตัวแปร x แยกออกเป็น 2 กรณี

- กรณี VB 2005 ค่าของตัวแปร y คือ 2 ผิดไปจากค่าเริ่มต้นของตัวแปร x
- กรณี VC# 2005 ค่าของตัวแปร y คือ 1 ผิดไปจากค่าเริ่มต้นของตัวแปร x เช่นกัน

จะเห็นได้ว่าแม้จะเปลี่ยนชนิดของข้อมูลได้ก็จริง แต่ค่าของข้อมูลที่ได้ผิดไปจากความเป็นจริง ซึ่งจะส่งผลให้เกิดข้อผิดพลาดที่เรียกว่า Logical error เพราะว่าถ้าคุณนำค่านี้ไปใช้คำนวณต่อ ก็อาจจะทำให้ผลการคำนวณผิดพลาดได้เช่นกัน ซึ่งเป็นคนละเรื่องกับโค้ดผิดพลาด

NOTE

ในการพัฒนาแอพพลิเคชัน ถ้าโค้ดผิดพลาดโปรเจ็กต์จะรันไม่ผ่าน VS 2005 จะแจ้งให้คุณทราบว่าเกิดข้อผิดพลาด ณ โค้ดจุดใด แต่ถ้าเป็นกรณี Logical error โปรเจ็กต์ของคุณจะรันได้ตามปกติ แต่ผลการทำงานไม่เป็นไปตามที่คุณต้องการ ข้อผิดพลาดประเภทนี้หายากพอสมควร ถ้าโปรเจ็กต์ของคุณมีขนาดใหญ่มากๆ

ส่วนการแปลงข้อมูลแบบ Explicit อีกแบบหนึ่งคือ การใช้เมธอด Parse() เพื่อทำหน้าที่ตีความข้อมูลชนิด String ก่อนว่า สามารถแปลงข้อมูลที่ส่งเข้ามาได้หรือไม่ ใช้ในกรณีแปลงตัวเลขที่อยู่ในฐานะข้อความ ให้เป็นข้อมูลชนิดตัวเลขที่อยู่ในฐานะตัวเลข Integer (int) (หรือชนิดอื่นๆ แล้วแต่ว่า คุณเรียกเมธอด Parse() ของชนิดข้อมูลตัวใด ตัวอย่างโค้ดนี้เรียกเมธอด Parse() ของชนิดข้อมูล Integer (int))

VB 2005

```
Dim s1 As String = "100"
Dim s2 As String = "200"
Dim z As Integer = 0
z = Integer.Parse(s1) + Integer.Parse(s2)
MessageBox.Show(z.ToString())
```

VC# 2005

```
string s1 = "100";
string s2 = "200";
int z;
z = int.Parse(s1) + int.Parse(s2);
MessageBox.Show(z.ToString());
```

ในกรณีนี้ผลการตีความของเมธอด Parse() สำเร็จ หมายความว่า สามารถแปลงข้อมูลของตัวแปร s1 และตัวแปร s2 เป็นตัวเลขได้ โดยที่อยู่ในฐานะเป็น Integer (int) นั่นเอง

การทำ Boxing และ Unboxing

การแปลงจาก Value Type ไปสู่ Reference Type เรียกว่าการทำ Boxing ส่วนการแปลงจาก Reference Type ไปสู่ Value Type เรียกว่าการทำ Unboxing

การเขียนโค้ดที่ดีไม่ควรให้เกิดการ Boxing หรือ Unboxing มากจนเกินไป เพราะว่าจะส่งผลถึงประสิทธิภาพในการทำงานเป็นอย่างยิ่ง จึงเป็นที่มาของการเขียนโปรแกรมแบบรักษาชนิดของข้อมูล หรือที่เรารู้จักกันดีโดยใช้คำว่า Strong Type (หรือ Safe Type) นั่นเอง

ตั้งแต่ .NET Framework 2.0 (VS 2005) เป็นต้นมา ได้เกิดแนวทางการเขียนโปรแกรมอีกลักษณะหนึ่งที่เรียกว่า Generic ขึ้นมา เพื่อช่วยลดกระบวนการเกิด Boxing หรือ Unboxing ถือเป็นอีก 1 แขนงที่ใหญ่มากๆ ของการเขียนโปรแกรมด้วย .NET Framework ส่วนการทำ Boxing และ Unboxing ให้ดูตัวอย่างที่ 2-8 การทำ Boxing และ Unboxing

โค้ด VB 2005 และ VC# 2005 ที่ 2-8 การทำ Boxing และ Unboxing

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim Val1 As Integer = 5
        Dim Obj As Object
        Obj = Val1

        Dim Val2 As Integer
        Val2 = DirectCast(Obj, Integer)
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    int Val1 = 5;
    object Obj;
    Obj = Val1;

    int Val2;
    Val2 = (int)Obj;
}
```

วิธีการดูโค้ดชุดนี้ก็คือ ประกาศตัวแปรที่ชื่อว่า Val1 มี Type เป็น Integer (int) ตัวแปร Val1 เป็น Value Type กำหนดค่าเริ่มต้นคือ 5 ต่อมาประกาศตัวแปรอีก 1 ตัวชื่อว่า Obj มี Type เป็น Object กล่าวได้อีกนัยหนึ่งว่าตัวแปร Obj เป็น Reference Type

VB 2005

```
Dim Val1 As Integer = 5
Dim Obj As Object
Obj = Val1
```

VC# 2005

```
int Val1 = 5;
object Obj;
Obj = Val1;
```

โค้ดข้างต้นมีการถ่ายค่าจากตัวแปร Val1 (Value Type) ไปสู่ตัวแปร Obj (Reference Type) ตรงจุดนี้จะเกิดกระบวนการ Boxing

ส่วนกระบวนการ Unboxing ก็คือ สร้างตัวแปรอีก 1 ตัวชื่อว่า Val2 มี Type เป็น Integer (int) เห็นได้ว่ามีการถ่ายค่าจากตัวแปร Obj (Reference Type) ไปสู่ตัวแปร Val2 (Value Type) แยกได้ 2 กรณี

VB 2005

```
Dim Val2 As Integer
Val2 = DirectCast(Obj, Integer)
```

VC# 2005

```
int Val2;
Val2 = (int)Obj;
```

- กรณี VB 2005 จะใช้ฟังก์ชัน DirectCast() เพื่อแปลงข้อมูลที่เก็บอยู่ในตัวแปร Obj ไปเป็นข้อมูลชนิด Integer
- กรณี VC# 2005 จะใช้การแปลงข้อมูลแบบ Explicit Conversion

สรุปท้ายบท

จากเนื้อหาที่ผ่านมาเห็นได้ว่า มีรายละเอียดปลีกย่อยมากมายที่ซ่อนอยู่เบื้องหลัง เมื่อคุณรู้จักกับตัวแปรมากขึ้นแล้ว ต่อไปจะเข้าสู่การสร้างคลาสต้นแบบขึ้นมาใช้งานเอง ก็จะใช้ความรู้ที่ได้จากบทนี้เป็นพื้นฐานส่วนหนึ่งด้วยเช่นกัน

CHAPTER ▸ 3

คลาสและเนมสเปซ

บทนำ

เนื้อหาในบทนี้ คุณจะได้ศึกษาวิธีการสร้างคลาสต้นแบบขึ้นมาใช้งานเอง รวมถึงศึกษากฎข้อที่ 1 จาก 3 ข้อหลักของการเขียนโปรแกรมแบบ OOP ที่สำคัญอีกอย่างหนึ่งคือ ทำความรู้จักกับระบบเนมสเปซ (Namespace) ว่าเข้ามาเกี่ยวข้องและมีความสำคัญกับ .NET Framework อย่างไร

ออบเจ็กต์ (Object) และคลาส (Class) คืออะไร

การเขียนโปรแกรมเชิงวัตถุหรือ OOP นั้น เป็นการมองทุกสิ่งทุกอย่างเป็นวัตถุ (Object) โดยที่วัตถุในโลกของ OOP กับโลกของความเป็นจริงต่างกันตรงที่ วัตถุในโลกของ OOP ไม่จำเป็นต้องเป็นสิ่งที่จับต้องก็ได้ เช่น วันที่และเวลาก็ถือว่าเป็นวัตถุได้เช่นกัน

ออบเจ็กต์แต่ละตัวมีหน้าที่ของตัวมันเองอย่างชัดเจน ถ้าจะเปรียบเทียบความสัมพันธ์ระหว่างออบเจ็กต์กับคลาสก็คือ คลาสคือแม่พิมพ์ ส่วนออบเจ็กต์คือ สิ่งที่ได้จากแม่พิมพ์นั่นเอง กล่าวได้อีกนัยหนึ่งก็คือ คลาสต้นแบบที่คุณสร้างขึ้นอยู่ในช่วงออกแบบ (Design Time) ส่วนออบเจ็กต์คือ ช่วงที่นำไปใช้งาน (Run Time)

คลาสต้นแบบที่คุณสร้างขึ้นมาคือ การรวบรวมฟังก์ชันหรือการทำงานที่อยู่ในกลุ่มเดียวกันหรือใกล้เคียงกันมาอยู่รวมกัน เพื่อให้สะดวกต่อการดูแลและง่ายต่อการตรวจสอบหาข้อผิดพลาดของแอพพลิเคชัน เช่น

- เราสร้างคลาสที่ชื่อว่า ClassA ขึ้นมา เรามอบหมายหน้าที่ (สร้างเมธอด) ให้ทำหน้าที่ค้นหาข้อมูลเพียงอย่างเดียว
- เราสร้างคลาสที่ชื่อว่า ClassB ขึ้นมา ทำหน้าที่เขียนข้อมูลอย่างเดียวเท่านั้น

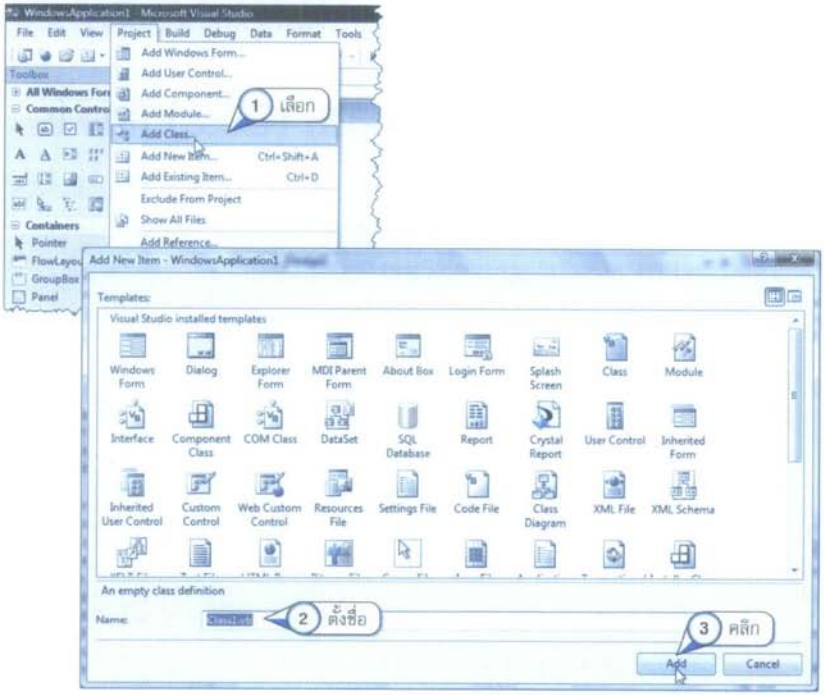
เมื่อคุณนำ ClassA, ClassB และคลาสอื่นๆ อีกมากมาย ไปใช้ในแอพพลิเคชันของคุณ ถ้าการค้นหาข้อมูลผิดพลาด เราจะรู้ได้ทันทีว่าโค้ดที่ผิดพลาดอยู่จุดไหน เพราะหน้าที่การค้นหาข้อมูลเรามอบหมายให้ ClassA ทำหน้าที่ดูแลแต่เพียงผู้เดียว

เมื่อการเขียนข้อมูลผิดพลาด เรารู้ว่าเราต้องไปดูโค้ดของ ClassB เพราะ ClassB ทำอย่างอื่นไม่เป็น ทำได้แต่เขียนข้อมูลอย่างเดียวเท่านั้น และนี่ก็คือการเปรียบเทียบให้เห็นง่ายๆ ของการเขียนโปรแกรมแบบ OOP

ประโยชน์อีกอย่างหนึ่งก็คือ เมื่อคุณสร้างคลาสใหม่ขึ้นมา มันจะทำหน้าที่รับผิดชอบงานอย่างชัดเจน โดยมีกติกาของ OOP เข้ามากำกับไว้ สิ่งที่ตามมาโดยอัตโนมัติคือ คลาสแต่ละคลาสที่คุณสร้างขึ้นมามีระบบมีระเบียบอย่างชัดเจน การนำคลาสต่างๆ ที่คุณมีอยู่กลับมาใช้ใหม่จึงเป็นเรื่องไม่ยากเย็น คุณไม่จำเป็นต้องเขียนโค้ดชุดเดิมซ้ำๆ กัน นี่คือความสามารถที่เข้มแข็งที่สุดของการเขียนโปรแกรมแบบ OOP นั่นคือ การนำกลับมาใช้ใหม่ได้ครั้งแล้วครั้งเล่า (re-use)

ทำความรู้จักกับฟิลด์ (Fields) และคุณสมบัติ (Properties) ในคลาส

ให้คุณเพิ่มคลาสเข้ามาในโปรเจ็กต์ โดยคลิกเมนู Project > Add Class... ตั้งชื่อไฟล์นี้ว่า Class1.vb (VB 2005) หรือ Class1.cs (VC# 2005) ดังรูปที่ 3-1



รูปที่ 3-1 การเพิ่มคลาสว่างเข้ามาในโปรเจ็กต์ของคุณ

จากรูปที่ 3-1 ไฟล์ของคลาสมีนามสกุล *.vb (VB 2005) หรือ *.cs (VC# 2005) เราจะได้คลาสว่างๆ มา 1 คลาส เป็นพื้นที่สำหรับสร้างคลาสต้นแบบของเราขึ้นมานั่นเอง

VB 2005	VC# 2005

Public Class Class1
 	
End Class
	class Class1
	{
	
	}

โค้ดข้างต้นหมายความว่า สร้างคลาสที่ชื่อว่า Class1 ขึ้นมา กำหนดให้เป็นแบบ Public มีคำที่เกี่ยวข้องในข้างต้นนี้อยู่ 2 คำคือ Private และ Public

ระดับการมองเห็นในคลาส

การเขียนโปรแกรมเชิงวัตถุมีกติกาในข้างต้นอยู่อย่างหนึ่งที่เรียกว่า Data Hiding หรือ Information Hiding หมายถึง การปกปิดหรือซ่อนข้อมูลไว้ โดยใช้คำสั่ง Private หรือ Public (หรือคำอื่นๆ ซึ่งจะกล่าวถึงในหัวข้อต่อๆ ไป) ทำหน้าที่ดังกล่าวทั้ง 2 คำสั่งข้างต้น เรียกว่า Access modifier โดยที่

- คำว่า Public (VB 2005) หรือ public (VC# 2005) ทำหน้าที่กำหนดขอบเขตการมองเห็นให้กับส่วนประกอบต่างๆ ของคลาส ให้เห็นได้โดยทั่วไป
- คำว่า Private (VB 2005) หรือ private (VC# 2005) ทำหน้าที่ซ่อนส่วนประกอบต่างๆ ของคลาส

NOTE

คำว่าส่วนประกอบต่างๆ ของคลาส เรียกอีกอย่างว่าเมมเบอร์ (member) ที่ผู้เขียนกล่าวหมายถึง ฟิลด์, คุณสมบัติ หรือเมธอดต่างๆ ที่อยู่ภายในคลาส ซึ่งจะกล่าวถึงในลำดับต่อๆ ไป

คำถามแรกที่เกิดขึ้นก็คือ ทำไมเราต้องกำหนดระดับของการมองเห็นภายในคลาส ผู้เขียนขอยกตัวอย่างของโลกความเป็นจริง เพื่อให้คุณผู้อ่านเห็นภาพ เช่น โทรศัพท์มือถือ เราใช้โทรออก-รับสายเข้า ซึ่งเป็นหน้าที่ประจำตัวของมันเอง เมื่อคุณต้องการโทรออกคุณก็กดปุ่มหมายเลข แล้วก็กดปุ่มโทรออก เป็นขั้นตอนที่เรารับรู้ได้โดยทั่วไป

แต่สิ่งหนึ่งเราไม่ทราบก็คือ หลังจากที่เรากดปุ่มหมายเลขแต่ละตัวลงไป กลไกภายในโทรศัพท์ไปสั่งงานชิป หรือไอซีหมายเลขอะไรทำงานบ้าง เห็นได้ว่ามีงานอยู่ส่วนหนึ่งซึ่งเป็นงานที่ยุ่งยาก ถูกซ่อนอยู่ภายในโทรศัพท์ เพราะว่าผู้ผลิตโทรศัพท์ไม่ต้องการให้ผู้ใช้ทั่วไปรับรู้ว่า ภายในเครื่องโทรศัพท์ประกอบไปด้วยอะไรบ้าง ที่ปกป้องไม่ให้คุณเห็น ผู้ใช้รับรู้แต่เพียงว่าถ้าโทรออกต้องทำอย่างไร, รับสายเข้าทำอย่างไร, ส่ง sms ทำอย่างไร การทำงานดังกล่าวผู้ผลิตโทรศัพท์เป็นผู้กำหนดให้เรา เราต้องทำตามขั้นตอนโทรศัพท์จึงจะทำงานถูกต้อง

ถ้าผู้ผลิตโทรศัพท์เปิดโอกาสให้ผู้ใช้โดยทั่วไปเข้าไปแก้ไข หรือเข้าไปยุ่งกับกลไกภายในโทรศัพท์ได้อย่างสะดวก คุณย่อมทราบดีว่าโอกาสที่โทรศัพท์พังมีสูงมาก เห็นได้ว่าตลอดการใช้งานโทรศัพท์ คุณไม่จำเป็นต้องรู้ว่าภายในโทรศัพท์ประกอบไปด้วยอะไรบ้าง, มีชิปกี่ตัว, มีวงจรการทำงานเป็นอย่างไร คุณเพียงแต่ทำตามกติกาที่โทรศัพท์กำหนดไว้คุณก็โทรคุยได้แล้ว

ส่วนไหนก็ตามที่คุณต้องการปกปิดไว้ในคลาส เราจะระบุขอบเขตด้วย Private ส่วนไหนที่เรายอมให้ผู้ใช้เรียกใช้งานได้เราจะระบุขอบเขต Public เพื่อเปิดโอกาสให้ผู้ใช้งานเรียกใช้ได้

ดังนั้น โทรศัพท์จึงถือว่าการกดเบอร์โทรออก, การวางสาย, หน้าจอแสดงผล ฯลฯ เป็นสิ่งที่ผู้ใช้ควรรู้เรียกใช้ได้นั่นคือ แบบ Public ส่วนแผงวงจรการทำงาน, ชิปต่างๆ เป็นสิ่งที่ผู้ใช้ไม่ควรเห็น ไม่ควรรับรู้นั่นคือ แบบ Private

ในโลกของ OOP คุณผู้อ่านเป็นผู้ผลิตคลาสต่างๆ ขึ้นมาเอง คุณย่อมที่จะมีสิทธิกำหนดได้ว่า ส่วนใดของคลาสต้องปกปิดไว้ ส่วนใดควรเปิดเผย ไล่ไปจนถึงเมื่อเปิดเผยแล้วกระทบต่อการทำงานของคลาสหรือไม่ ขอให้คุณนึกไว้ว่าเมื่อคุณเริ่มสร้างคลาสต้นแบบของคุณก็คือ ปิด (Private) ไว้ก่อน แล้วเปิด (Public) เท่าที่จำเป็น

สิ่งที่เราต้องศึกษาก็คือ OOP มีกฎกติกาอะไรบ้าง ส่งผลให้การศึกษาการเขียนโปรแกรม OOP จึงเต็มไปด้วยคำศัพท์ และเงื่อนไขต่างๆ มากมาย ผู้เขียนขอให้คุณผู้อ่านมองเงื่อนไขดังกล่าวเป็นผู้ช่วยเหลือ ไม่ใช่อุปสรรคของการศึกษาด้าน OOP

เรื่องของฟิลด์ใน OOP

ในการเขียนโปรแกรมแบบปกติ เราใช้ตัวแปรเข้ามาทำหน้าที่เก็บค่าจากการทำงานต่างๆ การสร้างคลาสก็เช่นกัน เราสามารถใช้ตัวแปรในคลาสได้เช่นกัน แต่จะเรียกว่าฟิลด์ (Fields) แทนคำว่าตัวแปร หรืออาจจะใช้คำว่า เมมเบอร์ (Member) ก็ได้ เพราะว่าฟิลด์คือ เมมเบอร์ประเภทหนึ่งที่อยู่ในคลาสเท่านั้น

NOTE

เราสามารถมองได้ว่าตัวแปรที่อยู่ในคลาส เป็นหน่วยจัดเก็บข้อมูลได้เช่นกัน มีลักษณะเช่นเดียวกับฟิลด์ในฐานข้อมูล ดังนั้น ถ้าผู้เขียนใช้คำว่าฟิลด์ จะหมายถึง ตัวแปรในคลาสไม่ใช่ฟิลด์ในฐานข้อมูล ยกเว้นแต่ว่าถ้าเนื้อหาดังกล่าวไปเกี่ยวข้องกับฐานข้อมูลก็จะระบุอย่างชัดเจน

เพื่อให้เกิดความแตกต่างระหว่างตัวแปรต่างๆ ที่คุณใช้ในฟอร์ม กับฟิลด์ต่างๆ ในคลาส ผู้เขียนจะตั้งชื่อฟิลด์โดยใช้เครื่องหมาย _ นำหน้าฟิลด์นั้นๆ

ฟิลด์ที่ใช้ในคลาสถือเป็นส่วนประกอบของคลาสส่วนแรกที่เราต้องทำความรู้จัก ถ้าจะเปรียบเทียบฟิลด์ต่างๆ ที่คุณสร้างขึ้นมาในคลาสกับโทรศัพท์มือถือ คำถามแรกที่ต้องตอบก็คือ ต้องเปิดเผย (Public) หรือปกปิด (Private)

ฟิลด์ต่างๆ ที่คุณประกาศในคลาสถือเป็นหน่วยจัดเก็บข้อมูลภายในคลาส เปรียบเสมือนเป็นชิป เป็นไอซี เป็นเมนบอร์ดที่อยู่ในโทรศัพท์ เป็นสิ่งที่ต้องปกปิดไว้ ดังนั้น เราจึงต้องกำหนดให้ฟิลด์ต่างๆ เป็นแบบ Private

OOP มีกติกากำหนดไว้ว่า ถ้าคุณต้องการอ่านค่า (get) หรือกำหนดค่า (set) ของฟิลด์ใดๆ ก็ตามที่ซ่อนอยู่ภายในคลาส คุณต้องสร้างคุณสมบัติ (Properties) ขึ้นมา ทำหน้าที่เป็นตัวกลางในการอ่านหรือกำหนดค่าฟิลด์นั้นๆ จับกันเป็นคู่ๆ เราเรียกคุณสมบัติกลุ่มนี้ว่า Accessor หรืออาจจะเรียกว่า Properties Accessor ก็ได้

Accessor เป็นช่องทางที่เรา (ในฐานะผู้ผลิตและผู้สร้างคลาสขึ้นมา) กำหนดไว้ว่า ถ้าต้องการอ่านค่าหรือกำหนดค่าฟิลด์ต่างๆ ในคลาสของเรา ต้องทำผ่านคุณสมบัติที่เราสร้างขึ้นมาเท่านั้น เพื่อป้องกันไม่ให้ภายนอกมองเห็นหรือเข้ามาแก้ไขค่าของฟิลด์ต่างๆ ซึ่งอาจจะส่งผลกระทบให้การทำงานของคลาสผิดพลาดได้ ให้ดูตัวอย่างคลาสต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 3-1 ทำความรู้จักกับฟิลด์ (Field) และคุณสมบัติ (Properties) ในคลาส

VB 2005 (Class1.vb)

```
ให้มีการประกาศตัวแปร ก่อนการใช้งาน
Option Explicit On
ให้เคร่งครัดต่อการเปลี่ยนแปลงชนิดของข้อมูล
Option Strict On

Public Class Bank                ชื่อคลาส Bank
'ฟิลด์ _FullName, _AccountNumber
'และฟิลด์ _Balance
    Private _FullName As String = ""
    Private _AccountNumber As String = ""
    Private _Balance As Double = 0.0
'คุณสมบัติ FullName คืนค่าเป็นข้อมูลชนิด String
    Public Property FullName() As String
        Get                      ให้อ่านค่าได้
            Return _FullName
        End Get
'ให้กำหนดค่าได้
        Set(ByVal value As String)
            _FullName = value
        End Set
    End Property

'คุณสมบัติ AccountNumber คืนค่าเป็นข้อมูลชนิด String
    Public Property AccountNumber() As String
        Get                      ให้อ่านค่าได้
            Return _AccountNumber
        End Get
'ให้กำหนดค่าได้
        Set(ByVal value As String)
            _AccountNumber = value
        End Set
    End Property

'คุณสมบัติ Balance คืนค่าเป็นข้อมูลชนิด Double
```

VC# 2005 (Class1.cs)

```
using System;
using System.Collections.Generic;
using System.Text;
//ชื่อเนมสเปซของคลาส
namespace FieldsAndProperties
{
    class Bank                   //ชื่อคลาส Bank
    {
//ฟิลด์ _FullName, _AccountNumber
//และฟิลด์ _Balance
    private string _FullName = "";
    private string _AccountNumber = "";
    private double _Balance = 0.0;
//คุณสมบัติ FullName คืนค่าเป็นข้อมูลชนิด string
    public string FullName
    {
        get                      //ให้อ่านค่าได้
        {
            return _FullName;
        }
        set                      //ให้กำหนดค่าได้
        {
            _FullName = value;
        }
    }

//คุณสมบัติ AccountNumber คืนค่าเป็นข้อมูลชนิด string
    public string AccountNumber
    {
        get                      //ให้อ่านค่าได้
        {
            return _AccountNumber;
        }
        set                      //ให้กำหนดค่าได้
```

```
    Public Property Balance() As Double
        Get                          ให้อ่านค่าได้
            Return _Balance
        End Get
    ให้กำหนดค่าได้
        Set(ByVal value As Double)
            _Balance = value
        End Set
    End Property
End Class
```

```
        {
            _AccountNumber = value;
        }
    }

    //คุณสมบัติ Balance คืนค่าเป็นข้อมูลชนิด double
    public double Balance
    {
        get                          //ให้อ่านค่าได้
        {
            return _Balance;
        }
        set                          //ให้กำหนดค่าได้
        {
            _Balance = value;
        }
    }
}
```

NOTE

เพื่อให้เหมาะสมกับการนำเสนอของภาษา VC# 2005 บล็อกของคุณสมบัติผู้เขียนจะลงโค้ด ดังนี้

VC# 2005

```
public string FullName
{
    get{return _FullName;}
    set{_FullName = value;}
}
```

ผู้เขียนสร้างคลาสต้นแบบขึ้นมาชื่อว่า Bank ประกอบไปด้วย 3 ฟิลด์กำหนดให้เป็นแบบ Private โดยที่

VB 2005

```
Private _FullName As String = ""
Private _AccountNumber As String = ""
Private _Balance As Double = 0.0
```

VC# 2005

```
private string _FullName = "";
private string _AccountNumber = "";
private double _Balance = 0.0;
```

- ฟิลด์ _FullName เป็นข้อมูลชนิด String ทำหน้าที่เก็บชื่อเจ้าของบัญชี
- ฟิลด์ _AccountNumber เป็นข้อมูลชนิด String ทำหน้าที่เก็บเลขที่บัญชี
- ฟิลด์ _Balance เป็นข้อมูลชนิด Double ทำหน้าที่เก็บยอดเงินในบัญชีล่าสุด

จะเห็นได้ว่ามีการกำหนดค่าเริ่มต้นให้กับฟิลด์ทั้ง 3 ไว้ด้วยกล่าวคือ ถ้าเป็นข้อมูลชนิด String จะกำหนดค่าเริ่มต้นเป็นค่าว่าง ส่วนข้อมูลชนิด Double กำหนดค่าเริ่มต้นเป็น 0.0 ตรงนี้เป็นอีกจุดหนึ่งที่เราในฐานะผู้ผลิตคลาสขึ้นมาใช้งานเอง ควรจะทำเพื่อป้องกันไม่ให้เกิดข้อผิดพลาด เมื่อนำคลาสนี้ไปสร้างออบเจ็กต์ โดยที่ผู้ใช้คลาสกำหนดค่าต่างๆ ให้กับฟิลด์ในคลาส ไม่ครบตามข้อกำหนดของคลาส

เหตุที่ผู้เขียนแนะนำดังกล่าว เพราะว่าคลาส Bank ไม่มีการทำงานที่ซับซ้อนแต่อย่างใด คุณผู้อ่านจึงยังไม่เห็นประโยชน์ของการกำหนดค่าเริ่มต้นให้กับฟิลด์ต่างๆ ที่นำมาใช้ในคลาส ทั้งนี้ไม่ได้หมายความว่าถ้าเป็นข้อมูลชนิด String แล้วต้องกำหนดค่าเริ่มต้นเป็นค่าว่าง หรือถ้าเป็น Double แล้วต้องกำหนด 0.0 เป็นต้น ตรงนี้ขึ้นอยู่กับว่าคลาสที่คุณสร้างขึ้นมาเป็นต้นแบบใช้ทำอะไร แต่ละฟิลด์เก็บอะไร แต่ละฟิลด์ควรมีค่าเริ่มต้นอะไร

โดยปกติแล้วเมื่อคุณสร้างตัวแปรขึ้นมาใช้งานในฟอร์ม เรามักจะไม่กำหนดค่าเริ่มต้นไว้ แต่สำหรับฟิลด์ในคลาสแล้วผู้เขียนแนะนำให้กำหนดค่าเริ่มต้นไว้

ต่อมาผู้เขียนสร้างคุณสมบัติของคลาส Bank ขึ้นมาอีก 3 ชุด กำหนดให้ชื่อคุณสมบัติมีชื่อเดียวกับฟิลด์ เพื่อบ่งบอกให้ผู้ใช้คลาสรู้ว่าคุณสมบัตินี้ไปกำหนดค่าหรืออ่านค่าของฟิลด์ใด ก็ให้จับกันเป็นคู่ๆ ไว้ เช่น คุณสมบัติ FullName เมื่อมีการกำหนดค่าก็จะสั่งให้บล็อกของ Set ทำงาน เห็นได้ว่าจะมีอาร์กิวเมนต์ที่ชื่อว่า value ทำหน้าที่เป็นตัวกลาง ส่งค่าต่อไปยังฟิลด์ _FullName

แต่ถ้ามีการอ่านคุณสมบัติ FullName บล็อก Get จะถูกสั่งให้ทำงาน เห็นได้ว่ามีการกำหนดให้คืนค่า (Return) ออกมาจากฟิลด์ _FullName

VB 2005

```
Public Property FullName() As String
    Get
        Return _FullName
    End Get
    Set(ByVal value As String)
        _FullName = value
    End Set
End Property
```

VC# 2005

```
public string FullName
{
    get{return _FullName;}
    set{_FullName = value;}
}
```

อีกประเด็นหนึ่งที่น่าสนใจก็คือ การสร้างคุณสมบัติขึ้นมา 3 ชุด เท่ากับจำนวนฟิลด์ที่ใช้ในคลาส ตรงนี้เพื่อความชัดเจนต้องบอกว่า คุณไม่จำเป็นต้องสร้างคุณสมบัติให้เท่ากับจำนวนฟิลด์ที่ใช้ในคลาส เพราะว่าในบางครั้งฟิลด์บางฟิลด์ทำหน้าที่พักค่าที่เกิดจากการทำงานของคลาส หรือเป็นฟิลด์ที่กำหนดค่าเริ่มต้นตายตัว ซึ่งไม่จำเป็นต้องเปิดโอกาสให้ผู้ใช้คลาสรับรู้หรือมองเห็น

การอ่านค่าหรือกำหนดค่าให้กับออบเจ็กต์ต่างๆ เป็นกิจกรรมที่เราคุ้นเคยกันเป็นอย่างดี เช่น การกำหนดคุณสมบัติ Text ของคอนโทรล TextBox บางครั้งเราต้องการอ่านค่าออกมา แต่บางครั้งเราก็สามารถกำหนดค่าได้เช่นกัน

กล่าวได้ว่าคุณสมบัติ Text ของคอนโทรล TextBox เป็นคุณสมบัติที่กำหนดค่าหรืออ่านค่าได้ เปรียบเทียบได้ว่าคุณสมบัติทั้งหมดของคลาส Bank ที่สร้างขึ้นมาสามารถกำหนดค่าหรืออ่านค่าได้เช่นกัน

เมื่อฟิลด์ต่างๆ ถูกกำหนดค่าแล้ว เราจะนำไปใช้อะไรก็แล้วแต่ว่า คลาสต้นแบบที่เราสร้างขึ้นมามีการทำงานอย่างไร ก็จะเหมือนกับการนำตัวแปรไปใช้นั่นเอง

คลาส Bank ที่สร้างขึ้นมาก็เปรียบเสมือนกับเป็นแม่แบบ และเป็นต้นแบบที่จะนำไปสร้างเป็นออบเจ็กต์ต่อไป คลาสต้นแบบเป็นอย่างไร ออบเจ็กต์ที่ได้ก็เป็นอย่างนั้น

การสร้างออบเจ็กต์จากคลาส

ถ้าเราต้องการใช้งาน เราต้องนำไปสร้างเป็นออบเจ็กต์ (Object) ด้วยคำสั่ง New (VB 2005) หรือคำสั่ง new (VC# 2005) ในข้างต้นนี้ผู้เขียนขอจำกัดความหมายไว้แบบนี้ก่อน เพราะว่ามีการสร้างคลาสอีกแบบที่ไม่ต้องใช้คำสั่ง New (VB 2005) หรือ new (VC# 2005) ได้เช่นกัน

ให้คุณเพิ่มฟอร์มเข้ามาในโปรเจ็กต์ เพื่อทดสอบสร้างออบเจ็กต์จากคลาสต้นแบบ Bank เหตุที่ผู้เขียนใช้ฟอร์มทดสอบ เพราะว่าโดยปกติแล้วในตำราต่างประเทศมักจะใช้โหมด Console เป็นตัวทดสอบคลาส แต่เพื่อให้คุณผู้อ่านคุ้นเคยกับการสร้างคลาสขึ้นมาใช้เอง แล้วใช้ร่วมกับ Windows Application อย่างไร ซึ่งจะกล่าวรายละเอียดในบทต่อๆ ไป จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 3-1 ทำความรู้จักกับฟิลด์ (fields) และคุณสมบัติ (Properties) ในคลาส

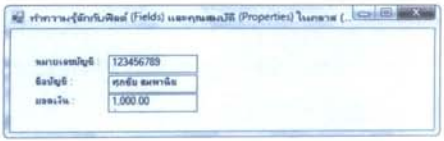
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1

Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
    Dim myBank As New Bank()

    With myBank
        .AccountNumber = "123456789"
        .FullName = "ศุภชัย สมพานิช"
        .Balance = 1000

        lblAccountNumber.Text = .AccountNumber
        lblFullName.Text = .FullName
        lblBalance.Text = .Balance.ToString("#,##0.00")
    End With
End Sub
End Class
```

VC# 2005 (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace FieldsAndProperties
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            Bank myBank = new Bank();

            myBank.AccountNumber = "123456789";
            myBank.FullName = "ศุภชัย สมพานิช";
            myBank.Balance = 1000;

            lblAccountNumber.Text = myBank.AccountNumber;
            lblFullName.Text = myBank.FullName;
            lblBalance.Text = myBank.Balance.ToString("#,##0.00");
        }
    }
}
```

รูปที่ 3-2
ผลการทำงานของคลาส Bank



จากโค้ดข้างต้นมีการทำงานอยู่ 3 ส่วนที่น่าสนใจคือ

1. สร้างออบเจ็กต์ Bank ที่ชื่อว่า myBank ขึ้นมาก่อนด้วยคำสั่ง New (VB 2005) หรือ new (VC# 2005) เราเรียกว่าการทำ Instance ของคลาส Bank()

VB 2005	VC# 2005
Dim myBank As New Bank()	Bank myBank = new Bank();

2. ถ้าต้องการกำหนดค่าให้กับฟิลด์ _AccountNumber, _FullName และฟิลด์ _Balance ให้ทำผ่านทางคุณสมบัติทั้ง 3 ตัวของออบเจ็กต์ myBank

VB 2005	VC# 2005
With myBank .AccountNumber = "123456789" .FullName = "ศุภชัย สมพานิช" .Balance = 1000	myBank.AccountNumber = "123456789"; myBank.FullName = "ศุภชัย สมพานิช"; myBank.Balance = 1000;

3. แต่ถ้าต้องการอ่านค่าของฟิลด์ทั้ง 3 ตัว ให้ทำผ่านทางคุณสมบัติ AccountNumber, FullName และคุณสมบัติ Balance เช่นกัน และแสดงค่าที่อ่านได้ในคอนโทรล Lable แต่ละตัว

VB 2005	VC# 2005

lblAccountNumber.Text = .AccountNumber
lblFullName.Text = .FullName
lblBalance.Text = .Balance.ToString("#,##0.00")
End With
	lblAccountNumber.Text = myBank.AccountNumber;
	lblFullName.Text = myBank.FullName;
	lblBalance.Text = myBank.Balance.ToString("#,##0.00");

4. คุณไม่สามารถกำหนดค่าของฟิลด์ _AccountNumber, _FullName และฟิลด์ _Balance ได้โดยตรง ซึ่งเป็นข้อห้ามไม่ให้ภายนอกเห็นการทำงานภายในของคลาส ดังนั้น เมื่อคุณสร้าง Instance ของคลาส Bank ขึ้นมาชื่อว่า myBank คุณจึงไม่สามารถเขียนโค้ดเพื่อกำหนดค่าให้กับฟิลด์ทั้ง 3 ได้โดยตรง ดังโค้ดต่อไปนี้

VB 2005	VC# 2005

Dim myBank As New Bank()
myBank._FullName = "ศุภชัย สมพานิช"
myBank._AccountNumber = 12345
myBank._Balance = 5000
	Bank myBank = new Bank();
	myBank._FullName = "ศุภชัย สมพานิช";
	myBank._AccountNumber = 12345;
	myBank._Balance = 5000;

จากตัวอย่างคลาสแรกของคุณ ถึงจุดนี้คุณได้ศึกษากติกาหลักของ OOP แล้ว 1 ข้อ (จาก 3 ข้อหลัก) นั่นคือ Encapsulation ให้คุณนึกถึงยาที่เป็นแบบแคปซูล เราไม่รู้ว่าภายในตัวยาแคปซูลประกอบไปด้วยตัวยาย่อยๆ อะไรบ้าง เรารู้แต่เพียงว่าเป็นยาแก้ปวด, ยาบำรุง เป็นต้น

เหตุที่ผู้เขียนนำกติกาของ OOP ข้อนี้มากล่าวเป็นลำดับแรก เพราะว่าการเขียนโปรแกรมแบบ OOP เป็นเรื่องที่ว่าด้วยการสร้างคลาสต้นแบบต่างๆ ขึ้นมาช่วยเหลือ เราต้องรู้ก่อนว่าคลาสที่เราสร้างขึ้นมา เรามีสิทธิกำหนดขอบเขตการมองเห็นของส่วนประกอบต่างๆ ของคลาส เพื่อให้คลาสที่เราสร้างขึ้นมาสามารถทำงานถูกต้องตามหน้าที่ของคลาสนั้นๆ และเราเป็นผู้กำหนดเองว่าส่วนไหนของคลาสให้มองเห็น ส่วนไหนของคลาสที่เราต้องซ่อนไว้

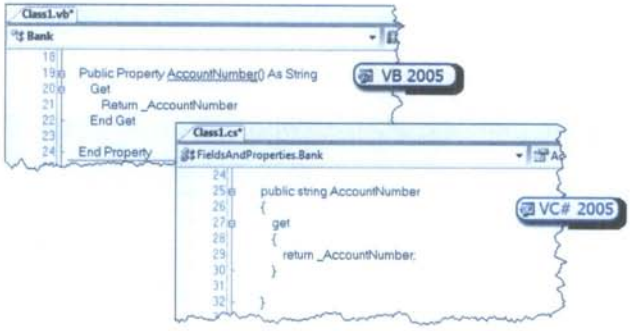
คลาสต้นแบบต่างๆ ที่คุณสร้างขึ้นมามีหน้าที่และมีความรับผิดชอบในตัวของมันเอง รู้แต่เรื่องงานของตัวเองเท่านั้น จึงจะเป็นคลาสต้นแบบที่ดีได้ที่สำคัญก็คือ ต้องมีความสมบูรณ์ในหน้าที่ที่รับผิดชอบอยู่

หน้าที่ของคลาสที่ผู้เขียนกล่าวถึง คุณผู้อ่านซึ่งเป็นผู้สร้างคลาสต้นแบบขึ้นมา เป็นผู้มอบหมายหน้าที่ดังกล่าวให้คลาสต่างๆ เป็นผู้รับผิดชอบนั่นเอง

การกำหนดให้คุณสมบัติอ่าน หรือกำหนดค่าได้เพียงอย่างเดียว

จากตัวอย่างคลาส Bank ที่ผ่านมา พบว่าคุณสมบัติ AccountNumber, FullName และคุณสมบัติ Balance เป็นคุณสมบัติที่สามารถกำหนดค่า หรืออ่านค่าได้ทั้ง 2 แบบ แต่จะมีบางกรณีที่คุณต้องการกำหนดให้บางคุณสมบัติอ่านค่าได้เพียงอย่างเดียว หรือกำหนดค่าได้เพียงอย่างเดียวจะทำอย่างไร

รูปที่ 3-3 การแก้ไขคุณสมบัติ AccountNumber



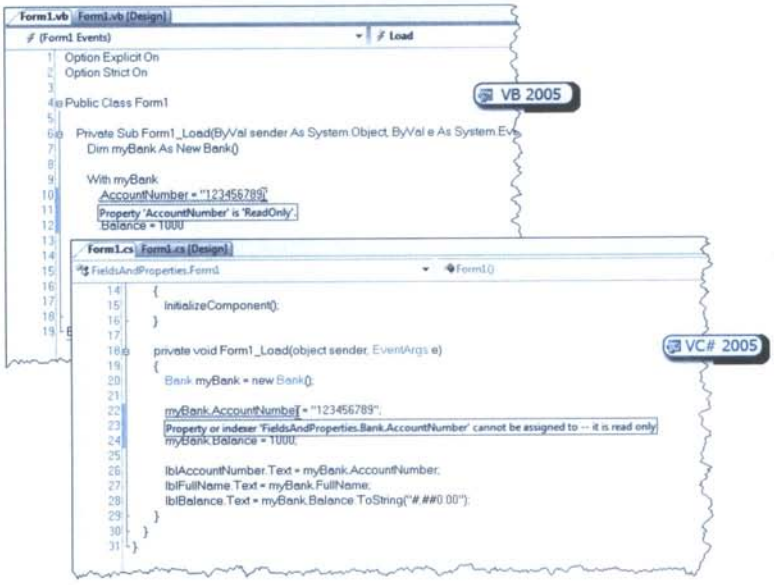
จากรูปที่ 3-3 สมมติว่าคุณต้องการกำหนดให้คุณสมบัติ AccountNumber สามารถอ่านค่าได้เพียงอย่างเดียวเท่านั้น ให้คุณลบโค้ดของบล็อก Set ออก พบว่าใน VB 2005 จะมีการแจ้งเตือนว่าโค้ดผิดพลาด เป็นเพราะว่า เมื่อคุณกำหนดให้คุณสมบัตินี้เหลือแต่บล็อกของ Get แล้ว คุณต้องระบุคำสั่ง ReadOnly (VB 2005) ด้วย

VB 2005	VC# 2005

Public ReadOnly Property AccountNumber() As String
Get
Return _AccountNumber
End Get
End Property
	public string AccountNumber
	{
	get{return _AccountNumber;}
	}

เมื่อคุณรันตัวอย่างอีกครั้งพบว่า จะมีการแจ้งข้อผิดพลาดว่าคุณสมบัติ AccountNumber ของออบเจ็กต์ myBank เป็นคุณสมบัติที่อ่านค่าได้เพียงอย่างเดียวเท่านั้น ดังรูปที่ 3-4

รูปที่ 3-4 ข้อผิดพลาดที่เกิดจากการกำหนดค่าให้กับคุณสมบัติที่อ่านค่าได้เพียงอย่างเดียว



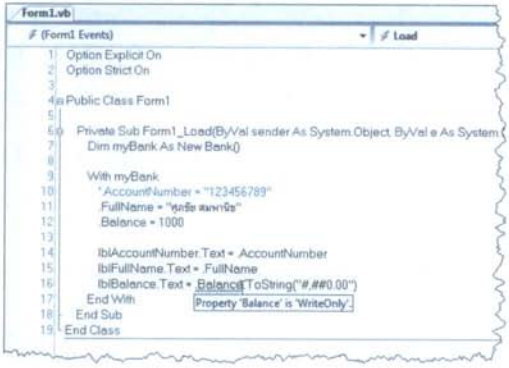
ในทางกลับกันถ้าคุณต้องการให้คุณสมบัติสามารถกำหนดค่าได้เพียงอย่างเดียวเท่านั้น ให้ลบบล็อกของ Get ออก และใส่คำสั่ง WriteOnly ของ VB 2005 เข้าไป เช่น ผู้เขียนกำหนดให้คุณสมบัติ Balance สามารถกำหนดค่าได้เพียงอย่างเดียวเท่านั้น

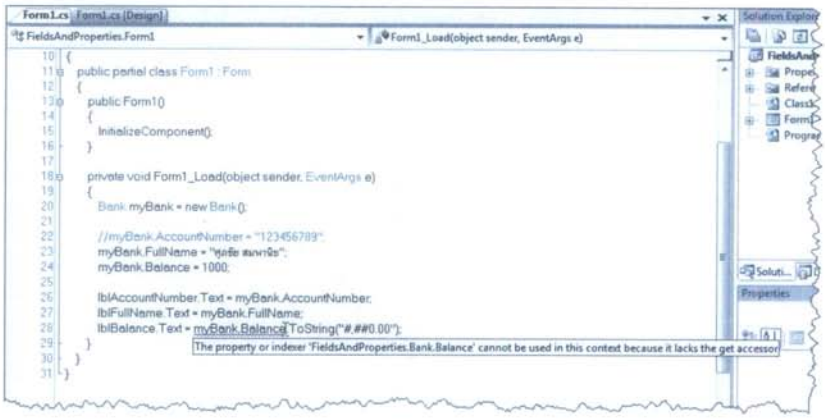
VB 2005	VC# 2005

Public WriteOnly Property Balance() As Double
Set(ByVal value As Double)
_Balance = value
End Set
End Property
	public double Balance
	{
	set
	{
	_Balance = value;
	}
	}

เมื่อคุณรันโปรเจ็กต์พบว่าจะเกิดข้อผิดพลาดขึ้น เมื่อคุณอ่านค่าจากคุณสมบัติที่กำหนดค่าได้เพียงอย่างเดียว ดังรูปที่ 3-5

รูปที่ 3-5 ข้อผิดพลาดที่เกิดจากการอ่านค่าจากคุณสมบัติที่กำหนดค่าได้เพียงอย่างเดียว





การกำหนดให้คุณสมบัติมีสิทธิการอ่าน หรือกำหนดค่าของฟิลด์ที่ซ่อนอยู่ในคลาสไม่มีข้อกำหนดใดๆ ทั้งสิ้น คุณสมบัติต่างๆ ที่สร้างขึ้นมาในคลาสต้นแบบของคุณ เป็นช่องทางที่คุณเป็นผู้กำหนดขึ้นมาเอง เมื่อมีการเปลี่ยนแปลงค่าของคุณสมบัติใดๆ ก็ตามจะส่งผลกระทบไปยังฟิลด์ต่างๆ ที่ซ่อนอยู่ภายในคลาส คุณย่อมควบคุมได้ถูกต้อง มากกว่าที่จะกำหนดให้ฟิลด์ต่างๆ มีขอบเขตเป็นแบบ Public

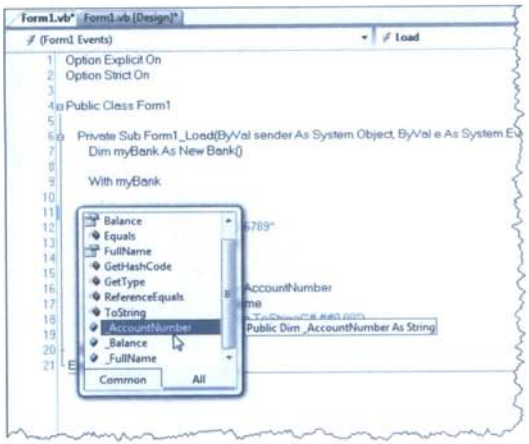
VB 2005

```
Public _FullName As String = ""
Public _AccountNumber As String = ""
Public _Balance As Double = 0.0
```

VC# 2005

```
public string _FullName = "";
public string _AccountNumber = "";
public double _Balance = 0.0;
```

รูปที่ 3-6
กรณีกำหนดให้ฟิลด์ต่างๆ เป็นแบบ Public



จากรูปที่ 3-6 สมมติว่าคุณกำหนดให้ฟิลด์ทั้ง 3 เป็นแบบ Public เมื่อมีการสร้างออบเจ็กต์ myBank ขึ้นมา เห็นได้ว่าทั้ง 3 ฟิลด์ปรากฏขึ้นมาด้วย ส่งผลให้สามารถกำหนดค่าให้กับฟิลด์ทั้ง 3 ได้โดยตรง

เมื่อมีการนำออบเจ็กต์ตัวนี้ไปใช้งาน เราไม่สามารถควบคุมค่าได้ เพราะไม่รู้ว่าผู้ใช้จะกำหนดค่าฟิลด์กลุ่มนี้เมื่อใด โดยเฉพาะอย่างยิ่งถ้ามีฟิลด์ประเภทรอรับค่า หรือพักค่าชั่วคราวไว้ ผลเสียที่เกิดขึ้นก็คือ การกำหนดขอบเขต Public ให้กับฟิลด์ไม่ผิดไวยากรณ์แต่อย่างใด แต่จะผิดในแง่ของ Logical Error กล่าวคือ คลาสอาจจะทำงานผิดพลาดหรือผิดไปจากหน้าเดิมที่เรากำหนดไว้ โดยที่เราจะหาข้อผิดพลาดดังกล่าวไม่พบ หรือหาเจอยากกว่าที่ควรจะเป็นนั่นเอง

เราสามารถจำลองการทำงานของคลาสต้นแบบ Bank กับอินสแตนซ์ myBank ดังรูปที่ 3-7

รูปที่ 3-7
ความสัมพันธ์ระหว่างคลาส Bank กับออบเจ็กต์ myBank

จากรูปที่ 3-7 อินสแตนซ์ myBank, myBank2, myBank3,... ที่เกิดขึ้นมาจากคลาสต้นแบบ Bank คุณสามารถนำไปใช้งานตามหน้าที่ที่มันรับผิดชอบอยู่ โดยที่แต่ละอินสแตนซ์ไม่เกี่ยวข้องกันแต่อย่างใด

VB 2005

```
Option Explicit On
Option Strict On

Public Class Sample
    Private _x As Integer
    Private _y As Integer
    Private _z As Integer

    Public Property x() As Integer
```

VC# 2005

```
public class Sample
{
    private int _x;
    private int _y;
    private int _z;

    public int x
    {
        get { return _x; }
```

```
        Get
            Return _x
        End Get
        Set(ByVal value As Integer)
            _x = value
        End Set
    End Property

    Public Property y() As Integer
        Get
            Return _y
        End Get
        Set(ByVal value As Integer)
            _y = value
        End Set
    End Property

    Public Property z() As Integer
        Get
            Return _z
        End Get
        Set(ByVal value As Integer)
            _z = value
        End Set
    End Property

    Public Function Add() As Integer
        _z = _x + _y
        Return _z
    End Function
End Class
```

```
        set { _x = value; }
    }

    public int y
    {
        get { return _y; }
        set { _y = value; }
    }

    public int z
    {
        get { return _z; }
        set { _z = value; }
    }

    public int Add()
    {
        _z = _x + _y;
        return _z;
    }
}
```

โค้ดข้างต้นผู้เขียนสร้างคลาสที่ชื่อว่า Sample ขึ้นมามี 3 ฟิลด์คือ ฟิลด์ _x, _y และฟิลด์ _z มีขอบเขตแบบ Private กำหนดให้เมธอด Add() ทำหน้าที่บวกฟิลด์ _x กับฟิลด์ _y เก็บผลการบวกไว้ที่ฟิลด์ _z ความผิดปกติของโค้ดชุดนี้อยู่ตรงที่คุณสมบัติ z ที่สร้างขึ้นมา

VB 2005

```
    Public Property z() As Integer
        Get
            Return _z
        End Get
        Set(ByVal value As Integer)
            _z = value
        End Set
    End Property
```

VC# 2005

```
    public int z
    {
        get { return _z; }
        set { _z = value; }
    }
```

ผลบวกที่เก็บอยู่ในฟิลด์ _z ต้องเกิดจากการทำงานของเมธอด Add() เท่านั้น แต่เราสร้างคุณสมบัติ z ขึ้นมา (อ่านค่าหรือกำหนดค่าให้กับฟิลด์ _z) เห็นได้ว่าช่องทางนี้ผู้ใช้สามารถกำหนดค่าให้กับฟิลด์ _z ผ่านทางคุณสมบัติ z ได้ ถือว่าผิดไปจากความตั้งใจของเรา ซึ่งเป็นผู้สร้างคลาส Sample ขึ้นมา

จะเห็นได้ว่าการสร้างคุณสมบัติ z ดังกล่าว ผิดกฎในเรื่องของการปกป้องข้อมูล (Encapsulation) ประเด็นที่นำเสนอก็คือ คุณไม่จำเป็นต้องสร้างคุณสมบัติขึ้นมาให้ครบตามจำนวนฟิลด์ที่ใช้อยู่ภายในคลาสต้นแบบของคุณ คุณสร้างคุณสมบัติต่างๆ ขึ้นมาในคลาสเพื่อให้ผู้ใช้สามารถกำหนดค่าเท่าที่ควรจะมี เท่าที่ควรจะเป็น

แต่ถ้าคุณกำหนดให้คุณสมบัติ z อ่านค่าได้เพียงอย่างเดียว ตรงนี้ถือว่าไม่ผิดปกติแต่อย่างใด

VB 2005

```
Public ReadOnly Property z() As Integer
    Get
        Return _z
    End Get
End Property
```

VC# 2005

```
public int z
{
    get { return _z; }
}
```

การใช้ Snippet

ผู้เขียนขอแนะนำเครื่องมือที่ช่วยให้การสร้างคลาสสะดวกและรวดเร็วยิ่งขึ้น เพราะว่าการสร้างคลาสที่มีคุณสมบัติด้วย (Property) ต้องมีการเขียนโค้ดค่อนข้างยาวพอสมควร เป็นโค้ดที่มีลักษณะคล้ายๆ กัน ใน VS 2005 มีเครื่องมือที่ช่วยในการเขียนโปรแกรมแบบ OOP เท่าที่ทราบมีอยู่ 2 ตัวก็คือ

- **การใช้ Snippet ช่วยเขียนโค้ด** จริงๆ แล้วฟีเจอร์นี้ไม่ได้ถูกออกแบบมาสำหรับเขียนโค้ดแบบ OOP โดยตรง เพียงแต่มีโค้ดสำเร็จรูปทางด้าน OOP ด้วยเช่นกัน หลักการทำงานก็คือ เขียนโค้ดแบบเติมคำในช่องว่าง
- **Class Diagrams (*.cd)** เป็นการจำลองโครงสร้างของคลาสด้วยแผนภาพ ซึ่งทำงานแบบ 2 ทิศทางกล่าวคือ เมื่อมีการแก้ไขที่แผนภาพแล้ว VS 2005 จะสร้างโค้ดขึ้นมาโดยอัตโนมัติตามที่คุณแก้ไข และเมื่อมีการแก้ไขโค้ดที่อยู่ภายในคลาส แผนภาพก็จะเปลี่ยนไปตามโครงสร้างของคลาสที่ถูกแก้ไข

โดยปกติแล้วเมื่อคุณสร้างคลาสขึ้นมา บ่อยครั้งที่จะมีการใช้งานฟิลด์ต่างๆ (ตัวแปรในคลาส) โดยที่ฟิลด์ดังกล่าวถูกกำหนดให้เป็นแบบ Private ตามกติกาของ OOP แล้ว ไม่อนุญาตให้ภายนอกรับรู้หรือแก้ไขฟิลด์ต่างๆ เหล่านี้ได้โดยตรง

ในกรณีที่ต้องการกำหนดให้มีการอ่านค่า (Get) หรือกำหนดค่า (Set) ของฟิลด์ประเภท Private เหล่านี้ ให้คุณสร้างคุณสมบัติจับคู่กับฟิลด์นั้นไว้ เพื่อทำหน้าที่อ่านหรือกำหนดค่าของฟิลด์ดังกล่าว ให้ดูตัวอย่างโค้ดต่อไปนี้

VB 2005

```
Public Class Customer

    Private _FirstName As String
    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property
End Class
```

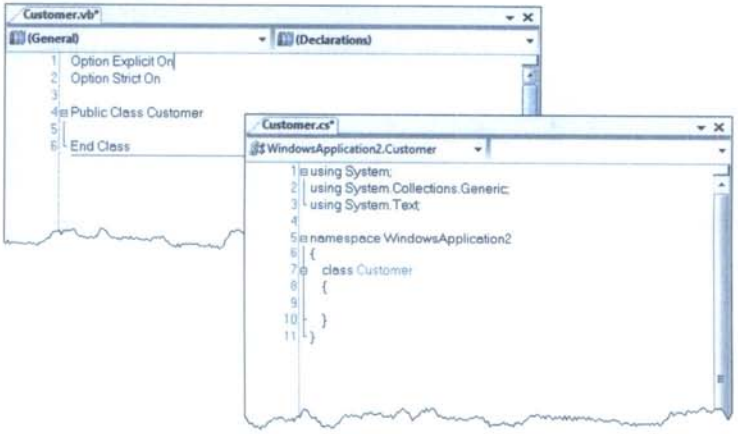
VC# 2005

```
class Customer
{
    private string _FirstName;
    public string FirstName
    {
        get { return _FirstName; }
        set { _FirstName = value; }
    }
}
```

จากโค้ดข้างต้นผู้เขียนสร้างคลาสต้นแบบขึ้นมา 1 คลาส ชื่อว่า Customer กำหนดให้คลาสนี้มีคุณสมบัติที่ชื่อว่า FirstName (เป็นข้อมูลชนิด String) เห็นได้ว่าจะมีฟิลด์ที่ชื่อว่า _FirstName (เป็นข้อมูลชนิด String เช่นกัน) จับคู่ไว้

โดยที่คุณสมบัติ FirstName กำหนดให้สามารถอ่านค่า (Get) และกำหนดค่า (Set) ได้ มีฟิลด์ที่ชื่อว่า Value ทำหน้าที่เป็นตัวกลางคอยรับ-ส่งค่าระหว่างการอ่านค่า หรือกำหนดค่าฟิลด์ต่างๆ ภายในคลาส Customer เห็นได้ว่าคลาสนี้มีคุณสมบัติเพียง 1 อย่าง แต่คุณต้องเขียนโค้ดเยอะพอสมควร จึงเป็นที่มาของการใช้ Snippet เข้ามาช่วยนั่นเอง

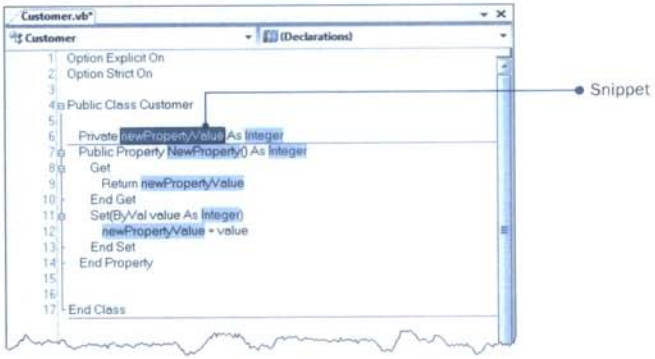
หลักการทำงานของ Snippet ก็คือ การเติมคำในช่องว่าง ให้คุณเพิ่มคลาสที่ชื่อว่า Customer เข้ามาในโปรเจ็กต์ ดังรูปที่ 3-8



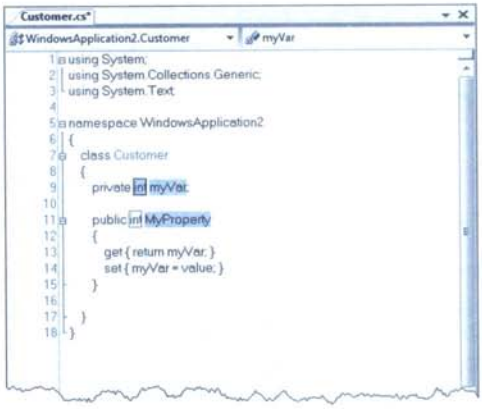
รูปที่ 3-8 โครงสร้างคลาสว่างๆ ที่ชื่อว่า Customer ใน VS 2005

การใช้ Snippet ใน VB 2005 ให้คุณพิมพ์คำว่า property แล้วกดปุ่ม <Tab> ที่คีย์บอร์ด ส่วน VC# 2005 พิมพ์คำว่า prop แล้วกดปุ่ม <Tab> ที่คีย์บอร์ด ดังรูปที่ 3-9 และ 3-10

รูปที่ 3-9
กรณีใช้ Snippet ใน VB 2005

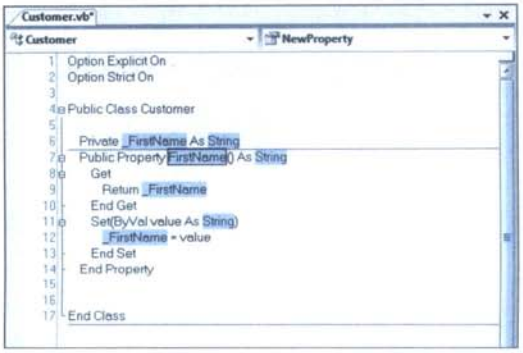


รูปที่ 3-10
กรณีใช้ Snippet ใน VC# 2005

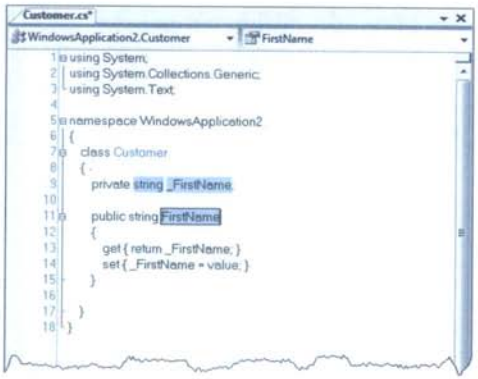


จากรูปพบว่ามีการสร้างโค้ดขึ้นมาโดยอัตโนมัติพร้อมๆ กับโค้ดส่วนหนึ่งที่มีการลงสีพื้นหลังเอาไว้ นั่นคือ โค้ดที่คุณต้องพิมพ์เติมลงไปนั่นเอง การเลื่อนไปยังโค้ดแต่ละช่องให้ใช้ปุ่ม Tab ที่คีย์บอร์ด คุณสามารถพิมพ์โค้ดเติมเข้าไปตามที่ต้องการ เพื่อสร้างคุณสมบัติของคลาส Customer ตามที่ต้องการ ดังรูปที่ 3-11

รูปที่ 3-11
กรณีการเพิ่มคุณสมบัติ FirstName ของคลาส Customer ใน VB 2005



รูปที่ 3-12
กรณีการเพิ่มคุณสมบัติ FirstName ของคลาส Customer ใน VC# 2005



จากรูปที่ 3-12 ในส่วนของภาษา VC# 2005 เป็นรูปแบบของการเขียนโค้ดแบบลดรูป เห็นได้ว่า Snippet ช่วยลดเวลาในการเขียนโค้ดเป็นอย่างยิ่ง เพราะว่าในขณะที่คุณพิมพ์โค้ดเติมเข้าไป บล็อกคำสั่งของ Get กับ Set เปลี่ยนไปตามโค้ดที่คุณพิมพ์โดยอัตโนมัตินั่นเอง

การใช้งาน Class Diagrams

Class Diagrams เป็นผู้ช่วยเหลืออีกตัวหนึ่งของ VS 2005 ที่จะทำให้การสร้างคลาสของคุณสะดวกยิ่งขึ้น เพราะว่าเป็นการแสดงรายละเอียดของคลาสแบบแผนภูมิ มีลักษณะเช่นเดียวกับภาษา UML

NOTE

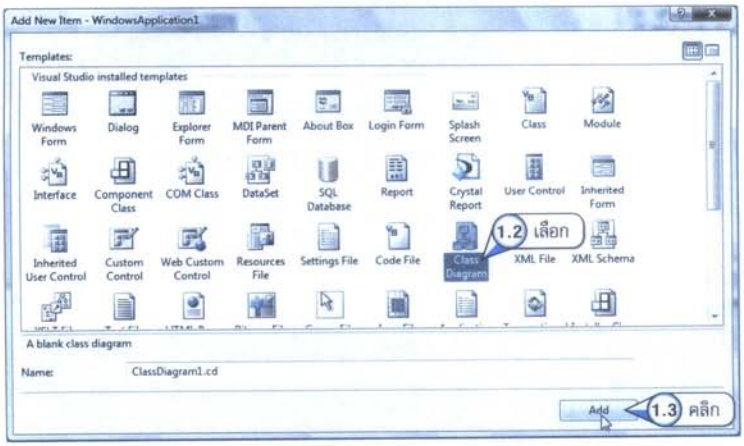
ภาษา UML ย่อมาจาก Unified Modeling Language โดยที่ภาษา UML ทำหน้าที่เป็น "แบบจำลอง" เพื่อสื่อความหมายของหน้าที่คลาสต่างๆ ด้วยแผนภูมิ หรือรูปภาพ ผู้เขียนขอแนะนำให้คุณผู้อ่านศึกษาภาษา UML ได้จากตำราด้านนี้โดยเฉพาะ ซึ่งจะอธิบายรายละเอียดได้ดีกว่าผู้เขียน

วิธีการเพิ่ม Class Diagrams มีขั้นตอนดังนี้

1. คลิกเมนู Project > Add New Item... เลือกไอคอน Class Diagram ดังรูปที่ 3-13

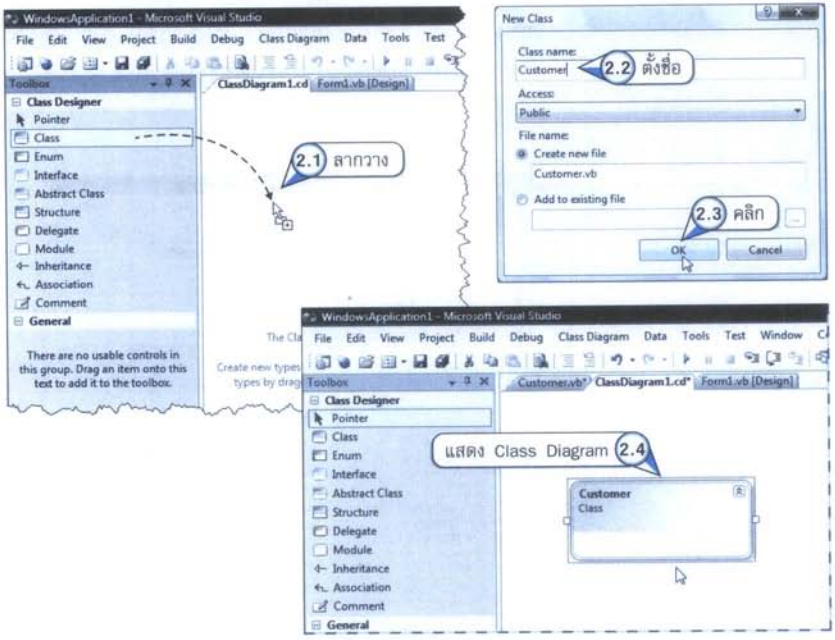
รูปที่ 3-13
การเพิ่ม Class Diagrams เข้ามาในโปรเจ็กต์

รูปที่ 3-13 (ต่อ)
การเพิ่ม Class Diagrams เข้ามาในโปรเจ็กต์



2. สมมติว่าคุณต้องการสร้างคลาส Customer ขึ้นมาที่แถบ Toolbox ของ VS 2005 ให้คุณลากออบเจ็กต์ชนิด Class มาวางในบริเวณพื้นที่ของ ClassDiagram1.cd ดังรูปที่ 3-14

รูปที่ 3-14
ไดอะล็อกบ็อกซ์ New Class



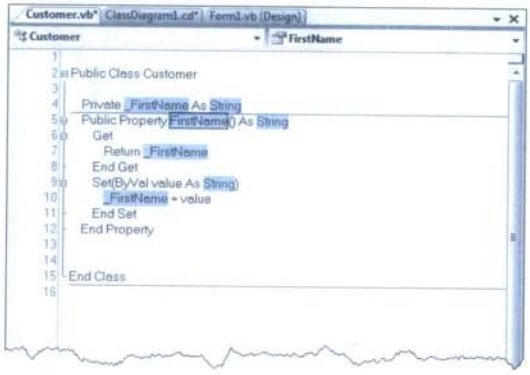
NOTE

ในกรณีที่คุณเลือก Add to existing file หมายความว่า คุณต้องการจำลองคลาสเดิมของคุณที่มีอยู่เป็นแผนภูมิใน ClassDiagram1.cd

จากรูปที่ 3-14 สมมติว่าคลาส Customer เก็บอยู่ในไฟล์ที่ชื่อว่า Customer.vb เป็นแผนภูมิที่แสดงโครงสร้างของคลาส Customer ว่างเปล่า เพราะว่าเป็นการสร้างคลาสใหม่นั่นเอง

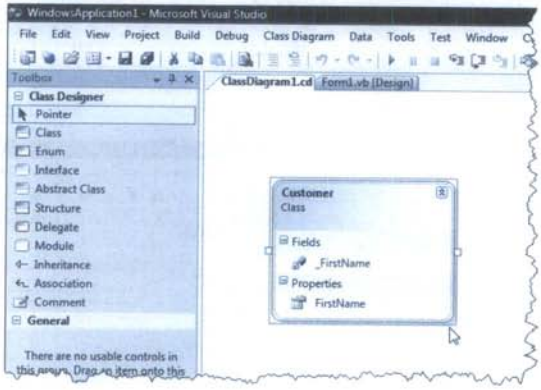
3. ให้ดับเบิลคลิกที่ Customer.vb หรือ Customer.cs ในหน้าต่าง Solution Explorer เพื่อกำหนดคุณสมบัติ FirstName ให้กับคลาส Customer ดังรูปที่ 3-15

รูปที่ 3-15
การเพิ่มคุณสมบัติ FirstName ให้กับคลาส Customer



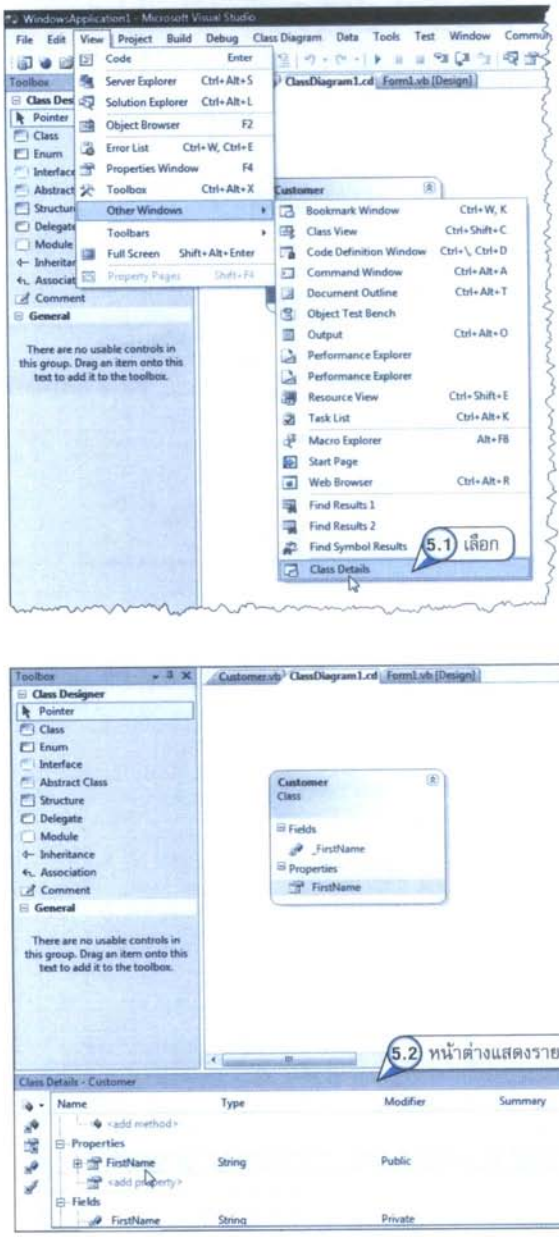
4. ให้ย้อนกลับไปดูแผนภูมิของคลาส Customer ใน ClassDiagram1.cd อีกครั้ง พบว่าฟิลด์ _FirstName และคุณสมบัติ FirstName จะแสดงขึ้นมาตามโครงสร้างของคลาส Customer ปัจจุบัน

รูปที่ 3-16
รายละเอียดปัจจุบันของคลาส Customer



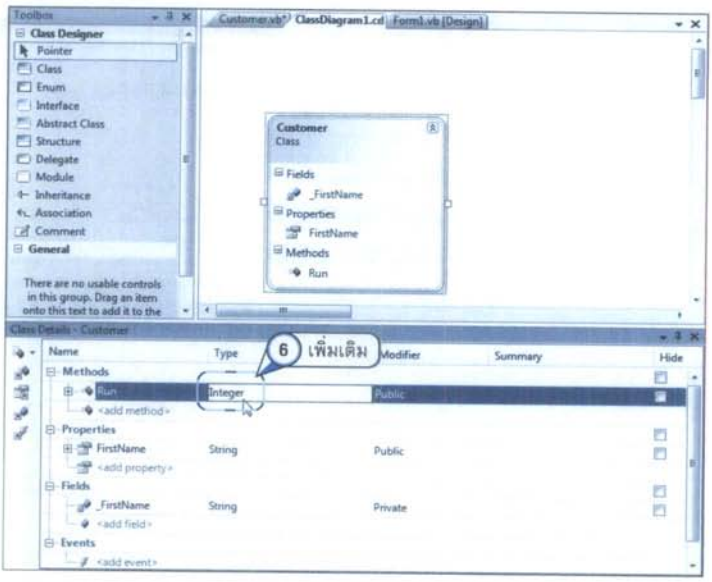
5. ให้คลิกเมนู View > Other Windows > Class Details เพื่อแสดงหน้าต่างรายละเอียดของคลาส ดังรูปที่ 3-17

รูปที่ 3-17 หน้าต่าง Class Details ใน VS 2005



6. ให้แก้ไขคลาส Customer ใหม่ โดยการเพิ่มเมธอด Run() เข้าไปในหน้าต่าง Class Details ดังรูปที่ 3-18

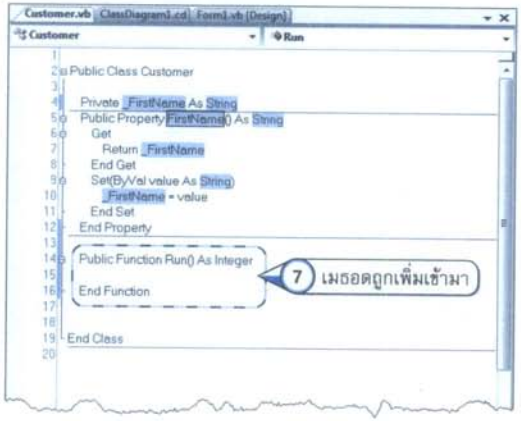
รูปที่ 3-18 การเพิ่มเมธอด Run() ในหน้าต่าง Class Details



จากรูปที่ 3-18 การเปลี่ยนคอลัมน์ถัดไปในหน้าต่าง Class Details ให้กดปุ่ม <Tab> บนคีย์บอร์ด

7. ให้ดับเบิลคลิกคลาส Customer ในหน้าต่าง Solution Explorer อีกครั้ง พบว่าเมธอด Run() ถูกเพิ่มเข้ามาในคลาส Customer ดังรูปที่ 3-19

รูปที่ 3-19 เมธอด Run() ที่ถูกเพิ่มเข้ามาในคลาส Customer

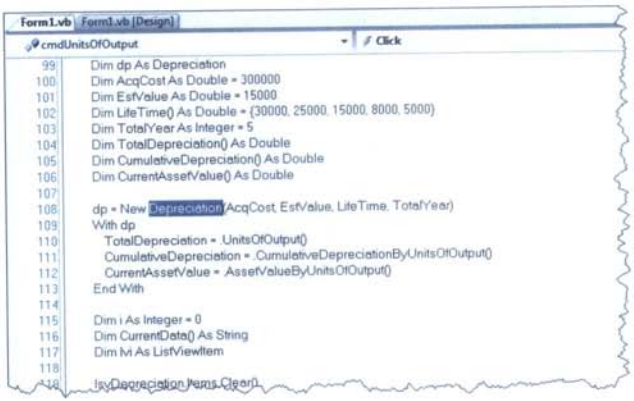


เห็นได้ว่า Class Diagrams มีลักษณะการทำงานแบบ 2 ทิศทางกล่าวคือ เมื่อคุณแก้ไขโค้ดหรือแก้ไขแผนภูมิภาพ ผลของการแก้ไขดังกล่าวจะสอดคล้องกันทั้ง 2 ฝั่ง ช่วยให้การออกแบบหรือวิเคราะห์ระบบ สะดวกยิ่งขึ้นนั่นเอง

การใช้งานเมนู Go To Definition

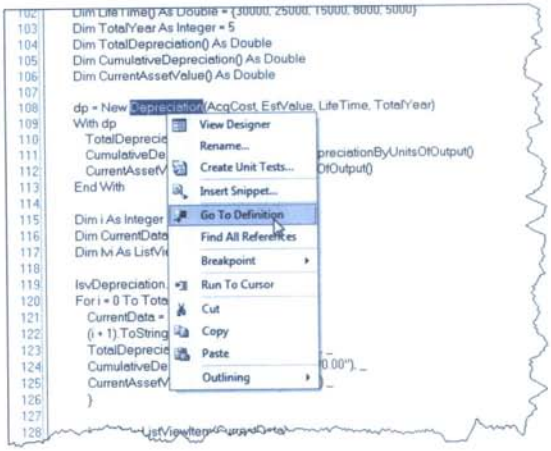
ใน VS 2005 ยังมีความสามารถอีกอย่างหนึ่งที่ช่วยให้โปรแกรมเมอร์ สามารถตรวจสอบโค้ดได้ง่ายขึ้น ซึ่งถือเป็นปัญหาที่หลายๆ ท่านพบเจอ เมื่อต้องการตรวจสอบโค้ดของโปรเจ็กต์ที่มีการสร้างคลาสต่างๆ มากมาย โดยเฉพาะอย่างยิ่งถ้าเป็นคลาสที่เราไม่ได้เขียนเอง เพราะเราไม่ทราบว่าคลาสต่างๆ ที่ถูกเรียกใช้มีรายละเอียดอย่างไร ทำหน้าที่อะไร และเก็บอยู่ในไฟล์ไหน

รูปที่ 3-20
คลาส Depreciation



จากรูปที่ 3-20 เห็นได้ว่ามีการเรียกใช้งานคลาส Depreciation ซึ่งคุณไม่ทราบว่าคลาสนี้ประกอบไปด้วยคุณสมบัติ หรือเมธอดอะไรบ้าง แต่ต้องการตรวจสอบว่าคลาสนี้มีรายละเอียดอย่างไร ให้โฟกัสที่คลาสดังกล่าว แล้วคลิกขวาเลือกคำสั่ง Go To Definition ดังรูปที่ 3-21

รูปที่ 3-21
การเลือกคำสั่ง Go To Definition



รูปที่ 3-21 (ต่อ)
การเลือกคำสั่ง Go To Definition

จากรูปที่ 3-21 เห็นได้ว่า VS 2005 จะโฟกัสไปที่คลาส Depreciation ให้คุณทันที ซึ่งเป็นอีกหนึ่งความสามารถของ VS 2005 ที่ช่วยให้การตรวจสอบโค้ดของคุณสะดวกยิ่งขึ้น

เนมสเปซ (Namespaces) คืออะไร

ในสถาปัตยกรรม .NET Framework ที่ไมโครซอฟท์สร้างขึ้นมา ฟีเจอร์หนึ่งที่ถูกนำมาใช้ก็คือ การใช้ระบบเนมสเปซนั่นเอง เพื่อให้คุณผู้อ่านรู้จักหน้าที่เนมสเปซของ .NET Framework รวดเร็วยิ่งขึ้น ผู้เขียนขอยกตัวอย่างในโลกของความเป็นจริงเข้ามาเปรียบเทียบ กล่าวคือ

รูปที่ 3-22
รูปลิ้นชักเก็บของที่มีป้ายติดอยู่

สมมติว่าบนโต๊ะทำงานของคุณผู้อ่านมีสิ่งของต่างๆ มากมาย วางอยู่เกะกะรกไปหมด ไม่ว่าจะเป็นดินสอ, ปากกา, กระดาษโน้ตชิ้นเล็กชิ้นน้อย, เอกสารทั่วไป, หนังสือ, แผ่น CD, DVD ฯลฯ

เมื่อคุณต้องการใช้สิ่งของเหล่านี้ ไม่ว่าโต๊ะทำงานจะรกขนาดไหนก็ตาม เราจะใช้ความคุ้นเคยของเราหาสิ่งของดังกล่าวเจอ แม้จะใช้เวลาค้นหามากน้อยแล้วแต่จังหวะและโอกาส

คำถามก็คือ จะดีกว่าไหมถ้าเราจะเก็บสิ่งของต่างๆ เหล่านี้ไว้เป็นหมวดหมู่ เป็นกลุ่ม เช่น เก็บไว้ในลิ้นชัก แยกตามหน้าที่การทำงาน แล้วเราก็เขียนกระดาษแปะไว้หน้าลิ้นชักแต่ละอันว่า เก็บของอะไรอยู่ เช่น เครื่องเขียนเก็บไว้ชั้นบนสุด, แผ่น CD ก็แยกเก็บอีกชั้น เป็นต้น

เมื่อคุณต้องการใช้สิ่งของเหล่านี้ คุณก็ต้องมองหาป้ายกระดาษก่อนว่า น่าจะเก็บไว้ในลิ้นชักอันใด โดยดูจากป้ายชื่อที่แปะในแต่ละชั้น แล้วก็เปิดลิ้นชักนั้นๆ ออกมาเพื่อหยิบของที่เราต้องการใช้

เทคโนโลยีที่ออกมาก่อนยุค .NET Framework เราเรียกว่า สถาปัตยกรรม Component Object Model เรียกสั้นๆ ว่า COM ประกอบไปด้วยออบเจ็กต์ต่างๆ มากมาย แยกไปตามหน้าที่ของมัน ปัญหาที่เกิดขึ้นกับ COM ก็คือ ออบเจ็กต์แต่ละตัวจะกระจัดกระจายอยู่กันไม่เป็นกลุ่ม ไม่เป็นระบบ ออบเจ็กต์แต่ละตัวของ COM เปรียบเทียบได้กับสิ่งของที่วางอยู่บนโต๊ะที่ไม่มีการจัดระเบียบ

จากปัญหาข้างต้นส่งผลให้คลาสต่างๆ ที่อยู่ใน .NET Framework จะต้องมีเนมสเปซเป็นของตัวเอง เข้ามาทำหน้าที่ระบุว่าตัวมันเองอยู่ในลิ้นชักอันใด ถ้าต้องการใช้งานคลาสใดก็ตามของ .NET คุณต้องหาให้เจอก่อนว่าคลาสดังกล่าวอยู่ในเนมสเปซอะไร เพราะเนมสเปซคือ ลิ้นชักที่เก็บคลาสต่างๆ ไว้ ถ้าไม่เปิดลิ้นชักออกมา (ไม่ระบุเนมสเปซ) คุณก็ไม่สามารถใช้คลาสดังกล่าวได้นั่นเอง

คลาสจำนวนมากมายที่ไมโครซอฟท์ประกอบกันขึ้นมาเป็น .NET Framework และถูกจัดหมวดหมู่แยกตามหน้าที่ต่างๆ อย่างชัดเจน ทำให้การเรียกใช้งานคลาสต่างๆ ดูไม่วุ่นวายเท่าใดนัก ทั้งๆ ที่มีจำนวนคลาสหลายพันคลาส ถ้าจะกล่าวว่าระบบเนมสเปซคือ ผู้ที่อยู่เบื้องหลังความสำเร็จของ .NET Framework ก็คงจะไม่ผิดเท่าใดนัก

เพราะความที่ VS 2005 ระบุเนมสเปซส่วนหนึ่งให้คุณโดยอัตโนมัติ เป็นกลุ่มเนมสเปซเบื้องต้น ซึ่งก็เพียงพอในระดับหนึ่งที่จะทำให้คุณสามารถเรียกใช้กลุ่มคอนโทรลต่างๆ, Form, ส่วนแสดงผลเบื้องต้น รวมถึง Primitive Data Type ต่างๆ เป็นต้น

VC# 2005

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
```

เนมสเปซกลุ่มนี้เป็นสิ่งที่เราคุ้นเคยกันอย่างดี ถ้าต้องการใช้งานคลาสอื่นๆ ที่อยู่นอกเหนือไปจากเนมสเปซข้างต้น ก็ต้องระบุเนมสเปซของคลาสดังกล่าวก่อนนั่นเอง เนมสเปซในแต่ละระดับคั่นด้วยเครื่องหมาย . โดยที่การระบุเนมสเปซใน VB 2005 จะใช้คำสั่ง Imports ส่วน VC# 2005 ใช้คำสั่ง using

คลาสที่คุณสร้างขึ้นมาเอง คุณสามารถกำหนดให้อยู่ในเนมสเปซได้เช่นกัน ตัวอย่างที่ 3-2 พื้นฐานการใช้งานเนมสเปซ ให้เพิ่มไฟล์ Class1.vb (.cs) เข้ามาในโปรเจ็กต์ จากนั้นเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 3-2 พื้นฐานการใช้งานเนมสเปซ

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Namespace RoomA
    Public Class A1

    End Class

    Public Class A2

    End Class
End Namespace

Namespace RoomA.SubB
    Public Class B1

    End Class
End Namespace
```

VC# 2005 (Class1.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace RoomA
{
    public class A1
    {

    }

    public class A2
    {

    }
}

namespace RoomA.SubB
{
    public class B1
    {

    }
}
```

โค้ดข้างต้นผู้เขียนสร้างคลาสขึ้นมา 3 คลาส โดยที่

- คลาสที่ชื่อว่า A1 และ A2 อยู่ภายใต้เนมสเปซที่ชื่อว่า RoomA
- คลาสที่ชื่อว่า B1 อยู่ภายใต้เนมสเปซที่ชื่อว่า RoomA.SubB

เปรียบเทียบได้ว่าคลาส A1 หรือ A2 ถูกเก็บอยู่ในห้องที่ชื่อว่า RoomA คุณจะใช้คลาสทั้ง 2 ได้ก็ต่อเมื่อคุณเปิดห้อง (ระบุเนมสเปซ) RoomA ก่อนนั่นเอง ส่วนคลาสที่ชื่อว่า B1 ถูกเก็บอยู่ในห้อง RoomA เช่นกัน เพียงแต่ว่าเอาไปซ่อนอยู่ในห้องย่อย SubB อีกชั้นหนึ่ง

โค้ด VB 2005 และ VC# 2005 ที่ 3-2 พื้นฐานการใช้งานเนมสเปซ

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On
Imports RoomA
Imports RoomA.SubB

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim UseA1 As New A1()
        Dim UseA2 As New A2()

        Dim UseB1 As New B1()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using RoomA;
using RoomA.SubB;

namespace IntroNamespace
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            A1 UseA1 = new A1();
            A2 UseA2 = new A2();

            B1 UseB1 = new B1();
        }
    }
}
```

การใช้งานคลาสทั้ง 3 อยู่ใน Form1 เห็นได้ว่าก่อนที่คุณจะใช้คลาส A1, A2 และคลาส B1 ต้องระบุก่อนว่าคลาสดังกล่าวถูกเก็บอยู่ในเนมสเปซอะไรนั่นเอง

VB 2005

```
Imports RoomA
Imports RoomA.SubB
```

VC# 2005

```
using RoomA;
using RoomA.SubB;
```

วิธีการเรียกใช้เนมสเปซ

อีกสิ่งหนึ่งที่น่าสนใจก็คือ เราสามารถเรียกเนมสเปซได้ 3 วิธีคือ

1. อาศัยคำสั่ง Imports (VB 2005) หรือคำสั่ง using (VC# 2005)
2. อาศัยการระบุชื่อเต็ม
3. อาศัยการตั้งชื่อ Alias

การเรียกใช้เนมสเปซแต่ละวิธีมีข้อดี ข้อเสียแตกต่างกันไป ให้ดูตัวอย่างที่ 3-3 วิธีการเรียกใช้งานเนมสเปซวิธีแรก ผู้เขียนกล่าวไปแล้วในหัวข้อที่ผ่านมา ข้อดีของวิธีนี้ก็คือ ในกรณีที่เนมสเปซนั้นประกอบด้วยคลาสต่างๆ เป็นจำนวนมาก การระบุด้วยคำสั่ง Imports (using) เพียง 1 ครั้ง คุณสามารถเรียกใช้คลาสที่อยู่ภายใต้เนมสเปซดังกล่าวได้ทั้งหมด เป็นวิธีที่พบเห็นได้บ่อยครั้งที่สุด

ส่วนวิธีที่ 2 อาศัยการระบุชื่อเต็มข้อดีก็คือ คุณไม่ต้องระบุเนมสเปซไว้ด้านบน แต่ระบุในขณะเรียกใช้คลาสเลย ส่งผลให้เกิดข้อเสียตรงที่ทุกๆ ครั้งที่มีการใช้คลาสอื่นๆ คุณต้องระบุเนมสเปซทุกครั้ง อย่างเช่นกรณีนี้คลาส A1 ต้องระบุเนมสเปซ 1 ครั้ง คลาส A2 ต้องระบุเนมสเปซอีก 1 ครั้ง

โค้ด VB 2005 ที่ 3-3 วิธีการเรียกเนมสเปซ (Form1.vb)

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim UseA1 As New RoomA.A1()
        Dim UseA2 As New RoomA.A2()

        Dim UseB1 As New RoomA.SubB.B1()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 3-3 วิธีการเรียกเนมสเปซ (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        RoomA.A1 UseA1 = new RoomA.A1();
        RoomA.A2 UseA2 = new RoomA.A2();

        RoomA.SubB.B1 UseB1 = new RoomA.SubB.B1();
    }
```

วิธีสุดท้ายอาศัยการตั้งชื่อ Alias คุณสามารถตั้งชื่อเล่นให้กับเนมสเปซได้เช่นกัน ซึ่งเป็นชื่อเรียกย่อๆ เห็นได้ว่าผู้เขียนตั้งชื่อเนมสเปซ RoomA ว่าเป็น MrA ส่วนเนมสเปซ RoomA.SubB ตั้งชื่อเป็น MrB

โค้ด VB 2005 ที่ 3-3 วิธีการเรียกเนมสเปซ (Form1.vb)

```
Option Explicit On
Option Strict On
Imports MrA = RoomA
Imports MrB = RoomA.SubB

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim UseA1 As New MrA.A1()
        Dim UseA2 As New MrA.A2()

        Dim UseB1 As New MrB.B1()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 3-3 วิธีการเรียกเนมสเปซ (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using MrA = RoomA;
using MrB = RoomA.SubB;

namespace IntroNamespace
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            MrA.A1 UseA1 = new MrA.A1();
            MrA.A2 UseA2 = new MrA.A2();

            MrB.B1 UseB1 = new MrB.B1();
        }
    }
}
```

เวลาที่ต้องการใช้คลาส คุณสามารถเรียกผ่านทางชื่อเล่น Alias ได้เลย

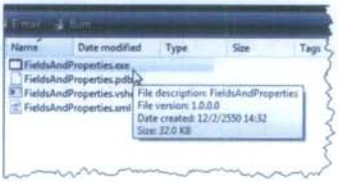
การสร้างและทดสอบไฟล์แอสเซมบลิ (Assembly)

คำว่า แอสเซมบลิ (Assembly) เป็นคำที่ใช้เรียกไฟล์ไลบรารี (Library) ในยุคของ .NET มีนามสกุล *.dll ย่อมาจากคำว่า Dynamic Link Library ไม่ใช่คำเรียกภาษาแอสเซมบลิ (Assembly Language) แต่อย่างใด

การที่คุณสร้างคลาสขึ้นมาใช้งานเอง เราถือว่าคลาสดังกล่าวเป็นผู้ช่วยเหลือ เป็นส่วนประกอบชนิดหนึ่งของโปรเจ็กต์ของคุณ สิ่งที่น่าสนใจคือ ถ้าต้องการนำผู้ช่วยเหลือเหล่านี้ไปใช้งานในโปรเจ็กต์อื่นๆ จะทำอย่างไร

ตัวอย่างการใช้งานคลาส Bank ที่ผ่านมา เป็นการกำหนดให้คลาส Bank ที่เก็บอยู่ในไฟล์ Class1.vb (.cs) ฝังตัวอยู่ในไฟล์ FieldsAndProperties.exe คุณสามารถตรวจสอบได้โดยการเปิดโฟลเดอร์ bin\Debug ของโปรเจ็กต์ FieldsAndProperties ดังรูปที่ 3-23

รูปที่ 3-23 รายการไฟล์ที่ถูกสร้างขึ้นมาของตัวอย่าง FieldsAndProperties



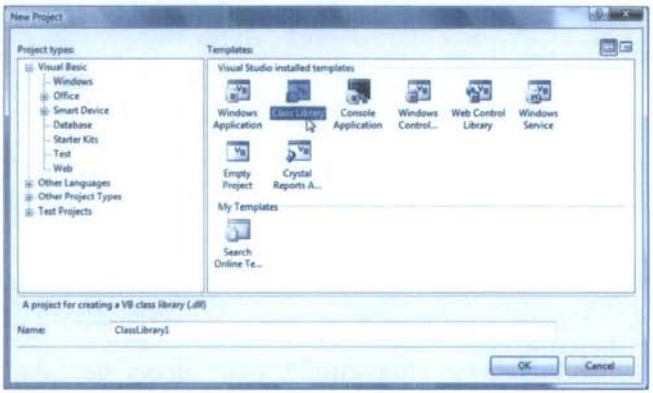
แต่ในทางปฏิบัติแล้ว เมื่อคุณสร้างคลาสต้นแบบขึ้นมาเพื่อรับผิดชอบการทำงานในด้านต่างๆ ย่อมที่จะต้องนำคลาสดังกล่าวมาใช้งานซ้ำใหม่เพื่อช่วยลดภาระในการพัฒนาแอพพลิเคชันใหม่ๆ ของคุณ การใช้งานคลาสตามตัวอย่างข้างต้น จึงไม่เหมาะสมเป็นอย่างยิ่ง คุณควรที่จะแยกคลาสต้นแบบของคุณออกมาจากไฟล์ *.exe ต่างหาก

คลาสต้นแบบที่คุณสร้างขึ้นมาคือ ไฟล์แอสเซมบลิ มีนามสกุล *.dll เป็นไฟล์ที่ไม่สามารถรันได้ด้วยตัวเอง ต้องให้ผู้อื่นเป็นผู้สั่งให้ทำงาน ไฟล์แอสเซมบลิที่ได้เรียกว่า Private Assembly

การสร้างแอสเซมบลิ

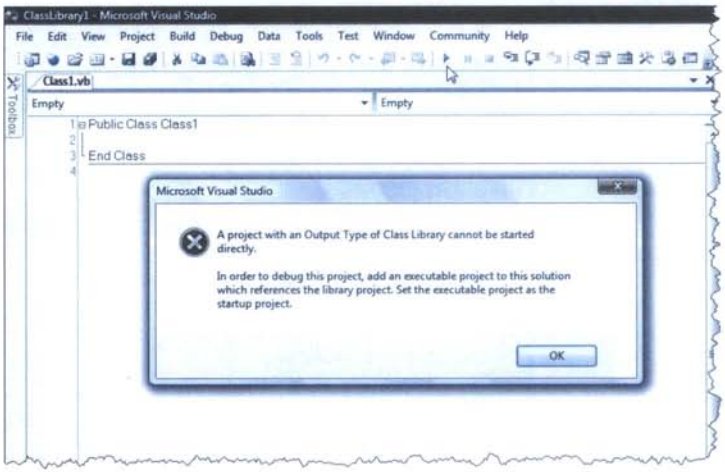
การสร้างแอสเซมบลิให้คุณสร้างในโปรเจ็กต์ชนิด Class Library ดังรูปที่ 3-24

รูปที่ 3-24 การใช้งานโปรเจ็กต์ชนิด Class Library



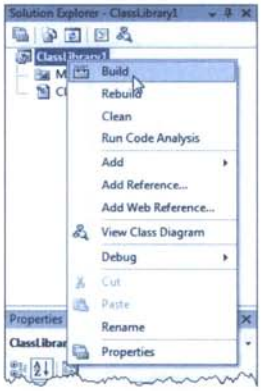
จากรูปที่ 3-24 โปรเจ็กต์ชนิด Class Library จะสร้างคลาสว่างๆ ขึ้นมา 1 คลาส โปรเจ็กต์ชนิดนี้ไม่สามารถรันได้ด้วยตัวเอง

รูปที่ 3-25
ข้อผิดพลาดเมื่อคุณสั่งให้รันโปรเจ็กต์ชนิด Class Library



โปรเจ็กต์ชนิดนี้จะใช้วิธีการ Build เพื่อสร้างไฟล์ *.dll ขึ้นมา โดยการคลิกขวาที่ชื่อโปรเจ็กต์ แล้วเลือกคำสั่ง Build ดังรูปที่ 3-26

รูปที่ 3-26
การ Build โปรเจ็กต์ชนิด Class Library

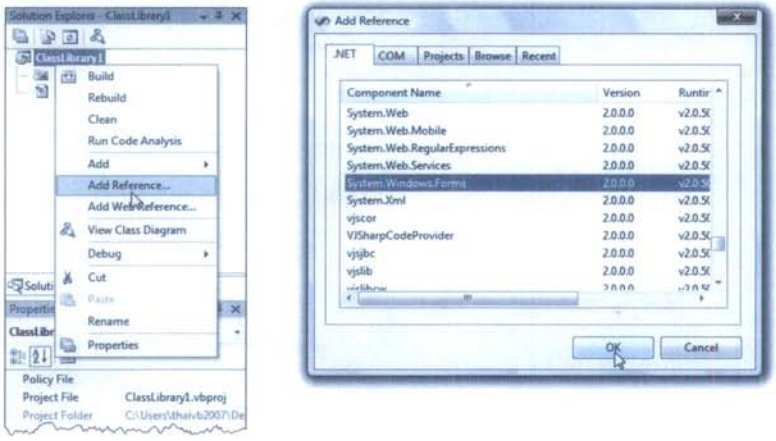


แอสเซมบลิที่ได้ (*.dll) เก็บอยู่ในโฟลเดอร์ bin\Debug ของโปรเจ็กต์ ดังรูปที่ 3-27

รูปที่ 3-27
แสดงไฟล์แอสเซมบลิที่ได้ในโฟลเดอร์ bin\Debug

ในกรณีที่มีการใช้งานคลาสอื่นๆ ของ .NET Framework คุณต้องระบุให้โปรเจ็กต์ Class Library รู้จักเสียก่อน เช่น ถ้าต้องการใช้งาน MessageBox คุณต้อง Reference ไฟล์ System.Windows.Forms เข้ามาก่อน ดังรูปที่ 3-28

รูปที่ 3-28
การ Reference ไฟล์ System.Windows.Forms



ถ้าต้องการตรวจสอบว่าโปรเจ็กต์ Class Library ของคุณ Reference อะไรบ้าง ให้คลิกปุ่ม ดังรูปที่ 3-29

รูปที่ 3-29
รายการไฟล์ที่ถูก Reference ของโปรเจ็กต์ปัจจุบัน

VB 2005

```
Option Explicit On
Option Strict On
Imports System.Windows.Forms

Public Class Sample
    Public Sub ShowData()
        MessageBox.show("ทดสอบการทำงาน")
    End Sub
End Class
```

โดยปกติแล้วเมื่อคุณใช้งานฟอร์มต่างๆ ในโปรเจ็กต์ชนิด Windows Application จะมีการ Reference ไฟล์พื้นฐานที่สำคัญๆ เหล่านี้ให้คุณโดยอัตโนมัติ ซึ่งแตกต่างจากโปรเจ็กต์ชนิด Class Library กล่าวคือ

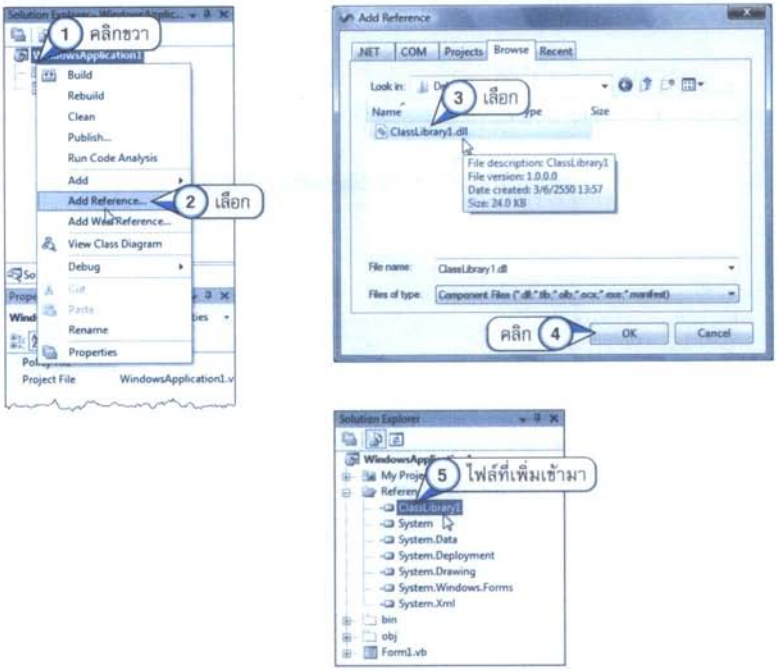
โปรเจ็กต์ชนิด Class Library คือ พื้นที่สำหรับสร้างผู้ช่วยเหลือด้านต่างๆ โดยที่คุณเป็นผู้กำหนดหน้าที่ขึ้นมาเอง คุณจึงต้อง Reference สิ่งที่จำเป็นที่เกี่ยวข้องเอง แต่ไม่จำเป็นต้อง Reference ทุกสิ่งทุกอย่างที่ไม่ได้ใช้งานในคลาสของคุณนั่นเอง

การใช้งานแอสเซมบลิ (*.dll)

ที่น่าสนใจมีอยู่ 2 วิธีคือ

วิธีที่ 1 : สมมติว่าคุณต้องการนำไฟล์แอสเซมบลิดังกล่าวไปใช้ในโปรเจ็กต์ชนิด Windows Application ของคุณ ให้คุณ Reference เข้ามาในโปรเจ็กต์ ดังรูปที่ 3-30

รูปที่ 3-30 กรณี Reference เข้ามาในโปรเจ็กต์ชนิด Windows Application

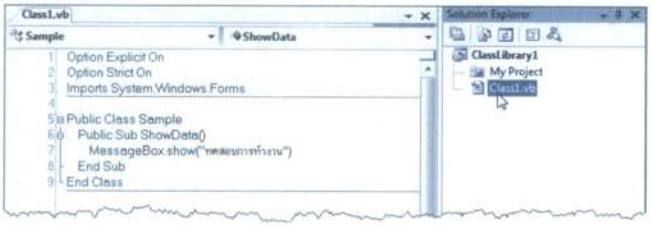


จากรูปที่ 3-30 เมื่อแอสเซมบลิที่ชื่อว่า ClassLibrary1.dll ถูก Reference เข้ามาในโปรเจ็กต์ชนิด Windows Application ของคุณแล้ว คุณสามารถเรียกใช้คลาสต่างๆ ที่อยู่ในแอสเซมบลิดังกล่าวได้ตามที่ต้องการ วิธีนี้เหมาะกับแอสเซมบลิที่สร้างเสร็จสมบูรณ์ พร้อมใช้งานแล้วนั่นเอง

วิธีที่ 2 : สร้างโปรเจ็กต์ขึ้นมาเพื่อสร้างและทดสอบแอสเซมบลิ เป็นวิธีที่ผู้เขียนเลือกมานำเสนอเป็นหลัก เพราะว่าโปรเจ็กต์ชนิด Class Library ไม่สามารถทดสอบผลการทำงานได้ด้วยตัวมันเอง จึงต้องสร้างโปรเจ็กต์ชนิด Windows Application ขึ้นมา เพื่อทดสอบการใช้งานแอสเซมบลิดังกล่าว

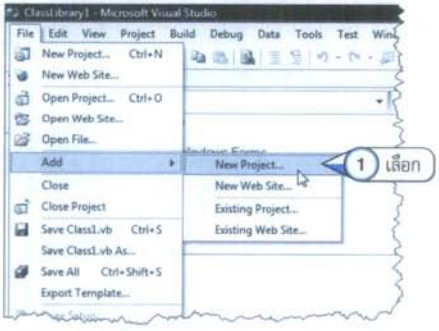
สมมติว่าผู้เขียนสร้างคลาสที่ชื่อว่า Sample ขึ้นมา เก็บอยู่ในโปรเจ็กต์ชนิด Class Library ที่ชื่อว่า ClassLibrary1 ดังรูปที่ 3-31

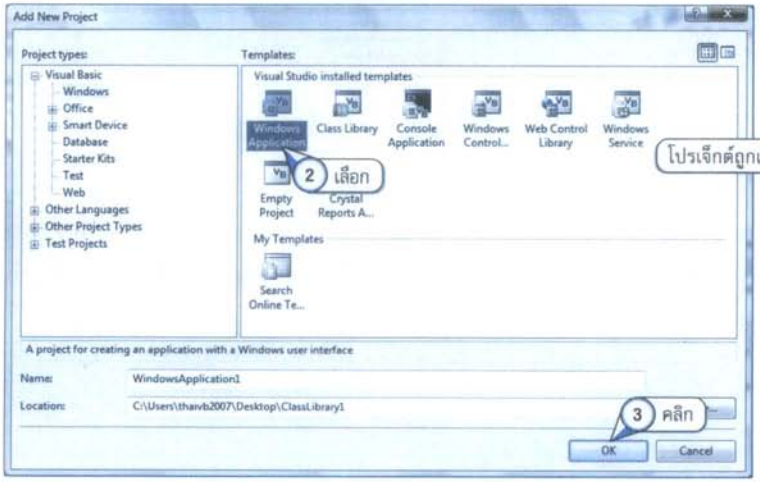
รูปที่ 3-31 คลาสที่ชื่อว่า Sample

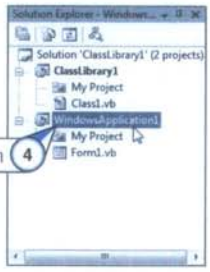


ต่อมาให้คุณเพิ่มโปรเจ็กต์ชนิด Windows Application ขึ้นมาใหม่ เพื่อทดสอบคลาส Sample โดยการคลิกเมนู File > Add > New Project... เลือกโปรเจ็กต์ชนิด Windows Application ดังรูปที่ 3-32

รูปที่ 3-32 การเพิ่มโปรเจ็กต์ชนิด Windows Application เข้ามาใหม่





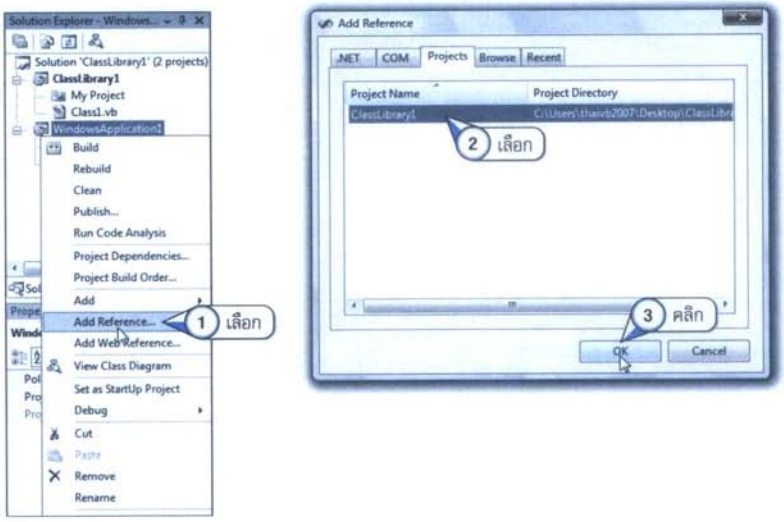


จากรูปที่ 3-32 ถึงจุดนี้คุณมี 2 โปรเจ็กต์ในเวลาเดียวกันคือ

- โปรเจ็กต์ชนิด Class Library ใช้สร้างคลาสต้นแบบของคุณ
- โปรเจ็กต์ชนิด Windows Application ใช้ทดสอบคลาสที่สร้างขึ้นมา

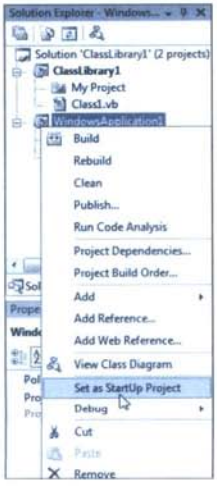
ต่อมาคุณต้องบอกให้โปรเจ็กต์ชนิด Windows Application รู้จักกับแอสเซมบลิที่สร้างในโปรเจ็กต์ชนิด Class Library ให้คลิกขวาเลือกคำสั่ง Add Reference... เลือกโปรเจ็กต์ Class Library ดังรูปที่ 3-33

รูปที่ 3-33
การ Reference ให้กับโปรเจ็กต์ชนิด Windows Application



ถึงจุดนี้โปรเจ็กต์ชนิด Windows Application รู้จักกับแอสเซมบลิที่อยู่ในโปรเจ็กต์ชนิด Class Library แล้ว ต่อมาให้กำหนดลำดับการรันโปรเจ็กต์โดยการคลิกขวาที่ Windows Application เลือกคำสั่ง Set as StartUp Project เพื่อกำหนดให้รันโปรเจ็กต์ชนิด Windows Application ก่อน ดังรูปที่ 3-34

รูปที่ 3-34
การกำหนดลำดับการรันโปรเจ็กต์

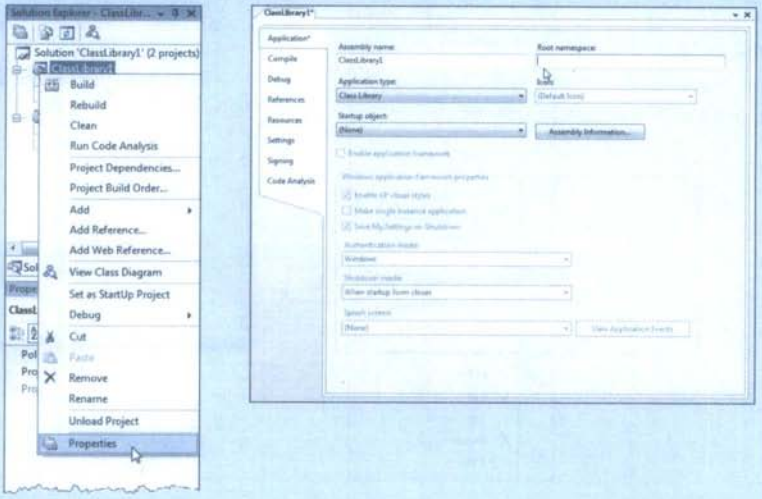


จากรูปที่ 3-34 คุณสามารถทดสอบคลาส Sample ใน Windows Application ของคุณได้แล้ว

NOTE

เฉพาะภาษา VB เท่านั้น ที่มีขั้นตอนเพิ่มขึ้นมาอีก 1 อย่างคือ โปรเจ็กต์ชนิด Class Library จะมีการเพิ่ม Root namespace ขึ้นมาให้คุณโดยอัตโนมัติ ให้คุณลบออกก่อน โดยการคลิกขวาเลือกคำสั่ง Properties ลบข้อความที่ช่อง Root namespace: ดังรูปที่ 3-35

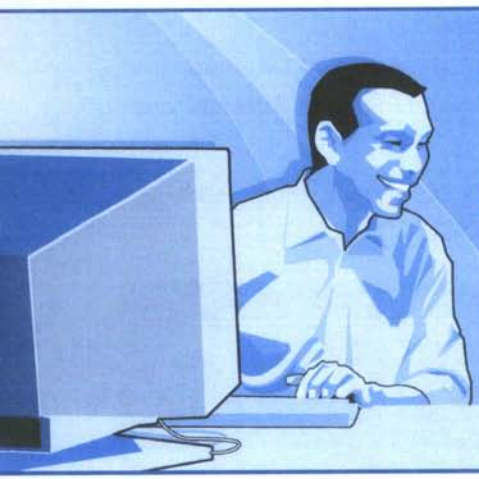
รูปที่ 3-35 การลบ Root namespace ของ VB 2005



วิธีนี้เหมาะสมกับการสร้างและทดสอบการทำงานของแอสเซมบลิ ในเวลาเดียวกันคุณจะได้พบกับตัวอย่างการสร้างแอสเซมบลิลักษณะนี้ ตั้งแต่บทที่ 13 เป็นต้นไป

สรุปท้ายบท

การสร้างคลาสต้นแบบขึ้นมาเพื่อทำหน้าที่เป็นผู้ช่วยเหลือของคุณ ถือเป็นงานหลักของการเขียนโปรแกรมแบบ OOP เนื้อหาของบทนี้จะเป็นจุดเริ่มต้นที่ดี สำหรับคุณผู้อ่านที่กำลังเข้าสู่โลกของ OOP อย่างเต็มตัว



CHAPTER ▸ 4

เมธอด

บทนำ

การทำงานของคลาสต้นแบบของคุณแสดงออกมาทางเมธอด คุณจะได้ศึกษาว่ารายละเอียดของเมธอดที่น่าสนใจมีอะไรบ้าง รวมถึงศึกษาวิธีการทำ Overload และ Override เมธอดของ .NET Framework ว่า ช่วยเหลือคุณในด้านใดบ้าง

ผู้เขียนแยกเนื้อหาของเมธอดออกเป็น 2 ส่วนคือ

- **เมธอด** ซึ่งเป็นเนื้อหาของบทนี้
- **พารามิเตอร์** ซึ่งเป็นเนื้อหาของบทถัดไป

ทำความรู้จักกับเมธอด (Method)

สมาชิกของคลาสที่ผู้เขียนกล่าวไว้ในข้างต้นคือ ฟิลด์ (Fields) และคุณสมบัติ (Properties) ของบทที่แล้ว ทำหน้าที่แสดงลักษณะของออบเจ็กต์เพียงอย่างเดียว เนื้อหาในบทนี้จะกล่าวถึงออบเจ็กต์ในแง่ของพฤติกรรมกันบ้าง สมมติว่าเราสนใจคนธรรมดา เรามองคนธรรมดาเป็นออบเจ็กต์ในโลกของ OOP

ลักษณะหรือคุณสมบัติ	พฤติกรรมหรือเมธอด
ชื่อ-สกุล, ผิวขาว ผิวดำ, อายุ, ชื่อเล่น, ส่วนสูง, น้ำหนัก, อายุ ฯลฯ	เดินได้, วิ่งได้, พูดได้, ร้องไห้ได้, ว่ายน้ำได้, ร้องเพลงได้ ฯลฯ

จากตารางข้างต้นเห็นได้ว่า เมธอดคือ ส่วนที่ออบเจ็กต์แสดงพฤติกรรมออกมานั่นเอง ก่อนที่ผู้เขียนจะกล่าวถึงเนื้อหาของเมธอด จะขอกล่าวถึงคำว่า member function ก่อน

คำว่า member function เป็นคำที่ใช้เรียกสมาชิกของคลาส เราถือว่าคุณสมบัติ (Properties) ที่ผู้เขียนกล่าวไว้ในบทที่แล้วก็คือ member function ชนิดหนึ่ง เป็น member function ที่มีโครงสร้างของบล็อก Set... กับ Get... ทำหน้าที่แสดงลักษณะของออบเจ็กต์ เช่น

- คุณสมบัติ FullName ทำหน้าที่กำหนดค่า (ทำงานที่บล็อก Set) เก็บค่าไว้ที่ฟิลด์ _FullName
- หรืออ่านค่า (ทำงานที่บล็อก Get) ออกมาจากฟิลด์ _FullName

VB 2005

```
Private _FullName As String = ""

Public Property FullName() As String
    Get
        Return _FullName
    End Get
    Set(ByVal value As String)
        _FullName = value
    End Set
End Property
```

VC# 2005

```
private string _FullName = "";

public string FullName
{
    get { return FullName; }
    set { FullName = value; }
}
```

เนื้อหาในบทนี้คือ member function อีกชนิดหนึ่ง เรารู้จัก member function ชนิดนี้ในชื่อเรียกว่า เมธอด (Method) ซึ่งทำหน้าที่แสดงพฤติกรรมของออบเจ็กต์นั่นเอง และมีอยู่ 1 เรื่องที่เราต้องสนใจนั่นคือ คำว่า Signature

คำว่า Signature ประกอบด้วย 3 ส่วนที่อยู่ในเมธอดคือ

- ชื่อของเมธอด (Method/Function Name)
- รายการพารามิเตอร์ (Parameters List) (ถ้ามี)
- ค่าที่ส่งกลับ (Return Type) (ถ้ามี)

VB 2005

```
'เมธอดแรก
Private Function ShowData(ByVal id As Integer) As String
'เมธอดที่ 2
Private Function ShowData(ByVal id As Integer, ByVal name As String) As Integer
```

VC# 2005

```
//เมธอดแรก
private string ShowData(int id)
//เมธอดที่ 2
private int ShowData(int id,string name)
```

โค้ดข้างต้นเป็นการสร้างเมธอดที่ชื่อว่า ShowData() มีขอบเขตแบบ Private ทั้ง 2 เมธอดมี Signature ต่างกันโดยที่

- เมธอดแรก ต้องการพารามิเตอร์ 1 ตัว เป็นข้อมูลชนิด Integer (int)
- เมธอดที่ 2 ต้องการพารามิเตอร์ 2 ตัว เป็นข้อมูลชนิด Integer (int) และ String (string) ตามลำดับ

ส่วนคำว่าพารามิเตอร์ (Parameter) บางครั้งอาจจะเรียกว่า อาร์กิวเมนต์ (argument) จึงหมายถึงรายการตัวแปรที่คุณต้องการนำมาใช้ในเมธอดนั่นเอง

ส่วนค่าที่ส่งกลับมาก็แตกต่างกันกล่าวคือ เมธอดแรกส่งค่ากลับมาเป็นข้อมูลชนิด String (string) ส่วนเมธอดที่ 2 ส่งค่ากลับมาเป็นข้อมูลชนิด Integer (int) เราสามารถแบ่งเมธอดได้ 2 ประเภทคือ

1. เมธอดที่มีการส่งค่ากลับ บางครั้งเรียกว่าฟังก์ชัน (function) ก็ได้
2. เมธอดที่ไม่มีการส่งค่ากลับใน VB 2005 เรียกว่า ซับรูทีน (sub routine) ส่วน VC# 2005 เรียกว่า void function

ให้ดูตัวอย่างที่ 4-1 พื้นฐานการใช้งานเมธอดเพิ่มเติม ให้คุณออกแบบฟอร์ม ดังรูปที่ 4-1

รูปที่ 4-1 ฟอร์มในขณะออกแบบ

ผู้เขียนปรับปรุงคลาส Bank ใหม่ ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 4-1 พื้นฐานการใช้งานเมธอด

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Bank
    Private _FullName As String = ""
    Private _AccountNumber As String = ""
    Private _Balance As Double = 0.0
    Private _InterestRate As Double = 10.0

    Public Property FullName() As String
        Get
            Return _FullName
        End Get
        Set(ByVal value As String)
            _FullName = value
```

VC# 2005 (Class1.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Method
{
    public class Bank
    {
        private string _FullName = "";
        private string _AccountNumber = "";
        private double _Balance = 0.0;
        private double _InterestRate = 10.0;

        public string FullName
        {
```

```
        End Set
    End Property

    Public Property AccountNumber() As String
        Get
            Return _AccountNumber
        End Get
        Set(ByVal value As String)
            _AccountNumber = value
        End Set
    End Property

    Public ReadOnly Property Balance() As Double
        Get
            Return _Balance
        End Get
    End Property

    Public Property InterestRate() As Double
        Get
            Return _InterestRate
        End Get
        Set(ByVal value As Double)
            _InterestRate = value
        End Set
    End Property

    Public Sub CashDeposit(ByVal Amount As Double)
        _Balance = _Balance + Amount
        หรือเขียนอีกแบบ
        '_Balance += Amount
    End Sub

    Public Sub Withdrawal(ByVal Amount As Double)
        If Amount > _Balance Then
            Amount = _Balance
        End If
        _Balance -= Amount
    End Sub

    Public Function CalculateInterest() As Double
        Dim _TotalInterest As Double = 0.0
        _TotalInterest = _Balance * (_InterestRate / 100)
        _Balance += _TotalInterest

        Return _Balance
    End Function
End Class
```

```
        get{return _FullName;}
        set{_FullName = value;}
    }

    public string AccountNumber
    {
        get{return _AccountNumber;}
        set{_AccountNumber = value;}
    }

    public double Balance
    {
        get{return _Balance;}
    }

    public double InterestRate
    {
        get{return _InterestRate;}
        set{_InterestRate = value;}
    }

    public void CashDeposit(double Amount)
    {
        _Balance = _Balance + Amount;
        //หรือเขียนอีกแบบ
        //_Balance += Amount
    }

    public void Withdrawal(double Amount)
    {
        if (Amount > _Balance)
        {
            Amount = _Balance;
        }

        _Balance -= Amount;
    }

    public double CalculateInterest()
    {
        double _TotalInterest = 0.0;
        _TotalInterest = _Balance * (_InterestRate / 100);

        _Balance += _TotalInterest;

        return _Balance;
    }
}
```

สิ่งที่เพิ่มเติมเข้ามาใหม่ของคลาส Bank ก็คือ

คุณสมบัติ/เมธอด	รายละเอียด
คุณสมบัติ Balance	ยอดคงเหลือในบัญชี กำหนดให้อ่านค่าได้เพียงอย่างเดียว
คุณสมบัติ InterestRate	อัตราดอกเบี้ยสามารถอ่านค่าหรือกำหนดค่าได้
ซับรูทีน CashDeposit()	ทำหน้าที่ฝากเงิน
ซับรูทีน Withdrawal()	ทำหน้าที่ถอนเงิน
เมธอด CalculateInterest()	ทำหน้าที่คำนวณดอกเบี้ย

ส่วนการใช้งานคลาส Bank อยู่ใน Form1 มีโค้ดดังนี้

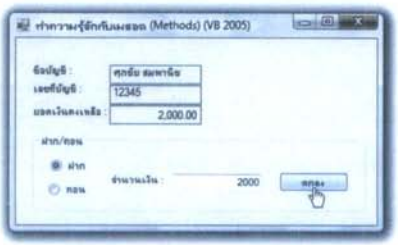
โค้ด VB 2005 ที่ 4-1 พื้นฐานการใช้งานเมธอด (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Dim myBank As New Bank

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        With myBank
            .FullName = "ศุภชัย สมพานิช"
            .AccountNumber = "12345"

            lblFullName.Text = .FullName
            lblAccountNumber.Text = .AccountNumber
            lblBalance.Text = .Balance.ToString("#,##0.00")
        End With
    End Sub

    Private Sub cmdOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdOK.Click
        Try
            If optCashDeposit.Checked = True Then
                myBank.CashDeposit(CDbl(txtAmount.Text))
            ElseIf optWithdrawal.Checked = True Then
                myBank.Withdrawal(CDbl(txtAmount.Text))
            End If

            lblBalance.Text = myBank.Balance.ToString("#,##0.00")
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 4-1 พื้นฐานการใช้งานเมธอด (Form1.cs)

```
    Bank myBank = new Bank();

    private void Form1_Load(object sender, EventArgs e)
    {
        myBank.FullName = "ศุภชัย สมพานิช";
        myBank.AccountNumber = "12345";

        lblFullName.Text = myBank.FullName;
        lblAccountNumber.Text = myBank.AccountNumber;
        lblBalance.Text = myBank.Balance.ToString("#,##0.00");
    }

    private void cmdOK_Click(object sender, EventArgs e)
    {
        try
        {
            if (optCashDeposit.Checked == true)
            {
                myBank.CashDeposit(Convert.ToDouble(txtAmount.Text));
            }
            else if (optWithdrawal.Checked == true)
            {
                myBank.Withdrawal(Convert.ToDouble(txtAmount.Text));
            }

            lblBalance.Text = myBank.Balance.ToString("#,##0.00");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

รูปที่ 4-2
ผลการรัน
ตัวอย่างที่ 4-1



พฤติกรรมของคลาส Bank ที่ทำได้คือ ต้องฝาก (เมธอด CashDeposit()) และถอนเงินได้ (เมธอด Withdrawal()) ที่น่าสนใจอีกอย่างก็คือ ยอดเงินฝาก (คุณสมบัติ Balance) ควรที่จะอ่านค่าได้เพียงอย่างเดียวเท่านั้น เพราะว่ายอดเงินจะเพิ่มขึ้นหรือลดลงต้องเกิดจากการฝากหรือถอนเงินเท่านั้น ไม่ใช่กำหนดค่าได้เอง ดังนั้น คุณสมบัติ Balance จึงกำหนดให้อ่านค่าได้เพียงอย่างเดียว

VB 2005

```
Public ReadOnly Property Balance() As Double
    Get
        Return _Balance
    End Get
End Property
```

VC# 2005

```
public double Balance
{
    get{return _Balance;}
}
```

ทำความรู้จักกับ Overload Method

การสร้างฟังก์ชันขึ้นมาใช้งานในบางครั้ง มีการทำงานเหมือนกันหรือคล้ายกัน แตกต่างกันก็แต่เพียง Type ของพารามิเตอร์ที่ต้องรับเข้ามา สมมติว่าต้องการสร้างฟังก์ชันขึ้นมาเพื่อคำนวณพื้นที่ เราต้องการรักษาชนิดของข้อมูลไว้ด้วย จึงต้องสร้างฟังก์ชันขึ้นมา 2 ฟังก์ชันกล่าวคือ

ฟังก์ชันที่ 1 ถ้าเป็นการหาพื้นที่ของเลขจำนวนเต็ม เราสร้างฟังก์ชันที่ชื่อว่า CalculateAreaInteger() ขึ้นมา

VB 2005

```
Private Function CalculateAreaInteger (ByVal Width As Integer, ByVal Height As Integer) As Integer
    Return Width * Height
End Function
```

VC# 2005

```
private int CalculateAreaInteger (int Width, int Height)
{
    return Width * Height;
}
```

ฟังก์ชันที่ 2 แต่ถ้าเป็นการหาพื้นที่ของเลขทศนิยม ก็จะสร้างอีกฟังก์ชันหนึ่งชื่อว่า CalculateAreaDouble()

VB 2005

```
Private Function CalculateAreaDouble(ByVal Width As Double, ByVal Height As Double) As Double
    Return Width * Height
End Function
```

VC# 2005

```
private double CalculateAreaDouble(double Width, double Height)
{
   return Width * Height;
}
```

จากโค้ดข้างต้นพบว่าการทำงานของทั้ง 2 ฟังก์ชันเหมือนกัน แต่ Type ของพารามิเตอร์ที่เอามาคำนวณแตกต่างกัน มีข้อเสียอย่างน้อย 2 อย่างคือ

- คุณต้องเขียนโค้ด 2 ครั้ง เพื่อทำงานเพียง 1 อย่าง ถ้าเป็นโค้ดที่มีความซับซ้อนมากๆ ถือเป็นปัญหาใหญ่พอสมควร
- คุณต้องสร้างฟังก์ชันต่างๆ ขึ้นมามากเกินกว่าที่ควรจะเป็น ซึ่งก็ก่อให้เกิดปัญหาที่ยากต่อการจดจำ และโค้ดซ้ำซ้อนมากเกินไป เพราะว่าถ้าการทำงานเพียง 1 อย่าง ต้องสร้างหลายฟังก์ชันต่อไปถ้าทำงานหลายๆ อย่าง คุณอาจจะต้องสร้างฟังก์ชันเพิ่มขึ้นมาเป็นทวีคูณก็เป็นได้

ทางแก้ปัญหาของการสร้างฟังก์ชันลักษณะนี้ ใน .NET จึงมีฟีเจอร์หนึ่งที่เรียกว่า การทำ Overload Method เป็นการแก้ปัญหาที่ดีในระดับหนึ่งเท่านั้นกล่าวคือ ใน .NET ยอมให้คุณตั้งชื่อฟังก์ชันหรือเมธอดซ้ำกันได้ แต่ Signature ต้องแตกต่างกัน

ผู้เขียนเคยกล่าวไว้ว่า Signature ของฟังก์ชันหรือเมธอดประกอบไปด้วย ชื่อฟังก์ชัน, รายการพารามิเตอร์ และค่าที่ส่งกลับ ถ้าต้องการให้ Signature แตกต่างกัน Type ของพารามิเตอร์ก็ต้องแตกต่างกันด้วย โค้ดต่อไปนี้พารามิเตอร์ของฟังก์ชันแรกเป็นข้อมูลชนิด Integer (int) ส่วนฟังก์ชันที่ 2 เป็นข้อมูลชนิด Double (double)

VB 2005

```
Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer) As Integer
Private Function CalculateArea(ByVal Width As Double, ByVal Height As Double) As Double
```

VC# 2005

```
private int CalculateArea(int Width, int Height)
private double CalculateArea(double Width, double Height)
```

แต่ถ้าค่าส่งกลับมี Type แตกต่างกันเพียงอย่างเดียว ดังโค้ดต่อไปนี้ถือว่ามี Signature เหมือนกัน ทำ Overload Method ไม่ได้ ฟังก์ชันแรกคืนค่าเป็น Integer (int) ส่วนฟังก์ชันที่ 2 คืนค่าเป็น Double (double) แต่มีจำนวนและชนิดของพารามิเตอร์เหมือนกัน

VB 2005

```
Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer) As Integer
Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer) As Double
```

VC# 2005

```
private int CalculateArea(int Width, int Height)
private double CalculateArea(int Width, int Height)
```

แต่ถ้ามีจำนวนพารามิเตอร์แตกต่างกัน ถือว่า Signature แตกต่างกันทันที แบบนี้ทำ Overload Method ได้ เพราะฟังก์ชันแรกต้องการพารามิเตอร์ 2 ตัว ส่วนฟังก์ชันที่ 2 ต้องการพารามิเตอร์ 3 ตัว

VB 2005

```
Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer) As Integer
Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer, ByVal Thick As Integer) As Double
```

VC# 2005

```
private int CalculateArea(int Width, int Height)
private double CalculateArea(int Width, int Height,int Thick)
```

บทสรุปที่ได้คือ ชนิดข้อมูลของค่าส่งกลับ (Return Type) ไม่ทำให้ Signature แตกต่างกันแต่อย่างใด ให้ดูตัวอย่างที่ 4-2 พื้นฐานการทำ Overload Method เพิ่มเติม

โค้ด VB 2005 ที่ 4-2 พื้นฐานการทำ Overload Method (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As Integer

        result = CalculateArea(20, 40)
        MessageBox.Show(result.ToString(), "พื้นที่ที่คำนวณได้")
    End Sub

    Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer) As Integer
        Return Width * Height
    End Function

    Private Function CalculateArea(ByVal Width As Double, ByVal Height As Double) As Double
        Return Width * Height
    End Function
End Class
```

โค้ด VC# 2005 ที่ 4-2 พื้นฐานการทำ Overload Method (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        int result;

        result = CalculateArea(20, 40);
        MessageBox.Show(result.ToString(), "พื้นที่ที่คำนวณได้");
    }

    private int CalculateArea(int Width, int Height)
    {
        return Width * Height;
    }

    private double CalculateArea(double Width, double Height)
    {
        return Width * Height;
    }
}
```

จากโค้ดข้างต้นผู้เขียนสร้างฟังก์ชันที่ชื่อว่า CalcuateArea() ขึ้นมา 2 ฟังก์ชัน มี Signature แตกต่างกัน ส่งผลให้ทำ Overload Method ได้ ผู้เขียนส่งค่าเลขจำนวนเต็ม 20 กับ 40 เข้ามาเพื่อคำนวณหาพื้นที่

ฟังก์ชัน CalculateArea() ที่ถูกสั่งให้ทำงานคือ ฟังก์ชันที่พารามิเตอร์มี Type ตรงกับค่าที่ส่งเข้ามา ในกรณีนี้คือ ฟังก์ชันที่มีพารามิเตอร์ชนิด Integer (int)

VB 2005

```
Private Function CalculateArea(ByVal Width As Integer, ByVal Height As Integer) As Integer
    Return Width * Height
End Function
```

VC# 2005

```
    private int CalculateArea(int Width, int Height)
    {
        return Width * Height;
    }
```

แต่ถ้าคุณส่งค่าตัวเลขทศนิยมเข้ามาเพียงพารามิเตอร์เดียว ฟังก์ชันที่ถูกเรียกใช้งานคือ ฟังก์ชันที่มี พารามิเตอร์ชนิด Double (double) เพราะว่าพารามิเตอร์ทุกตัวที่มีอยู่ต้องรับค่าที่ถูกส่งเข้ามาให้ได้ เห็นได้ว่า ฟังก์ชันที่ทำ Overload ถูกเรียกใช้ตาม Type ของพารามิเตอร์แต่ละตัวนั่นเอง

VB 2005

```
Private Function CalculateArea(ByVal Width As Double, ByVal Height As Double) As Double
    Return Width * Height
End Function
```

VC# 2005

```
private double CalculateArea(double Width, double Height)
{
    return Width * Height;
}
```

NOTE

เห็นได้ว่าการทำ Overload Method แก้ปัญหาได้เพียงอย่างเดียวนั่นคือ คุณสามารถตั้งชื่อฟังก์ชันหรือเมธอดซ้ำกันได้ โดยที่ฟังก์ชันหรือเมธอดดังกล่าวต้องมี Signature ต่างกัน แต่ยังไม่ได้แก้ปัญหาที่ต้องเขียนโค้ดซ้ำ การแก้ปัญหาในลักษณะนี้ต้องใช้ความสามารถของ .NET ที่เรียกว่าการทำ Generic Method

การทำ Overload Method ในคลาส

การทำ Overload Method สามารถทำกับเมธอดที่อยู่ในคลาสของคุณได้เช่นกัน ให้ดูตัวอย่างที่ 4-3 การทำ Overload Method ในคลาส

โค้ด VB 2005 ที่ 4-3 การทำ Overload Method ในคลาส (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Calculator
    Public Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
        Return (x + y)
    End Function

    Public Function Add(ByVal x As Integer, ByVal y As Integer, ByVal z As Integer) As Integer
        Return ((x + y) + z)
    End Function

    Public Function Add(ByVal x As Double, ByVal y As Double) As Double
        Return (x + y)
    End Function

    Public Function Add(ByVal x As Double, ByVal y As Double, ByVal z As Double) As Double
        Return ((x + y) + z)
    End Function
End Class
```

โค้ด VC# 2005 ที่ 4-3 การทำ Overload Method ในคลาส (Class1.cs)

```
public class Calculator
{
    public int Add(int x, int y)
    {
        return x + y;
    }
    public int Add(int x, int y, int z)
    {
        return x + y + z;
    }
    public double Add(double x, double y)
    {
        return x + y;
    }
    public double Add(double x, double y, double z)
    {
        return x + y + z;
    }
}
```

ผู้เขียนสร้างเครื่องคิดเลขขึ้นมา (คลาส Calculator) บวกเลขได้อย่างเดียว เพียงแต่ว่าบวกได้หลายแบบ เช่น

- ถ้าบวกเลขจำนวนเต็ม 2 ค่าก็ใช้เมธอดนี้

VB 2005

```
Public Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
    Return (x + y)
End Function
```

VC# 2005

```
public int Add(int x, int y)
{
    return x + y;
}
```

- แต่ถ้าต้องการบวกทศนิยม 3 ค่าก็จะมาใช้เมธอดนี้

VB 2005

```
Public Function Add(ByVal x As Double, ByVal y As Double, ByVal z As Double) As Double
    Return ((x + y) + z)
End Function
```

VC# 2005

```
public double Add(double x, double y, double z)
{
    return x + y + z;
}
```

Operator Overload

เราทราบมาแล้วว่าการทำ Overload Method คือ การสร้างเมธอดที่มีชื่อซ้ำกันได้ แต่ Signature ของเมธอดต้องแตกต่างกัน เพื่อให้ตัวแปรภาษาสามารถแยกแยะและสั่งเมธอดให้ทำงานตาม Type หรือตามจำนวนพารามิเตอร์ที่ถูกส่งเข้ามา ยังมีการทำ Overload อีกลักษณะหนึ่ง เรียกว่า Operator Overload ยกตัวอย่าง เช่น ตัวดำเนินการด้านคณิตศาสตร์

โดยปกติแล้วตัวดำเนินการด้านคณิตศาสตร์ เรานำมาใช้กับ Type ที่เป็นตัวเลข ไม่ว่าจะเป็นเลขจำนวนเต็มหรือเลขทศนิยมก็ตาม เราสามารถสร้าง Operator ขึ้นมาอีก 1 ชุด แล้วนำคลาสที่เราสร้างขึ้นมาไปบวก, ลบ, คูณ หรือหารกันได้อีกด้วย ให้ดูตัวอย่างที่ 4-4 การทำ Operator Overload

โค้ด VB 2005 ที่ 4-4 การทำ Operator Overload (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Shape
    Private _Width As Double = 0.0
    Private _Height As Double = 0.0

    Public Property Width() As Double
        Get
            Return _Width
        End Get
        Set(ByVal value As Double)
            _Width = value
        End Set
    End Property

    Public Property Height() As Double
        Get
            Return _Height
        End Get
        Set(ByVal value As Double)
            _Height = value
        End Set
    End Property
```

```
    Public Overloads Shared Operator +(ByVal xShape As Shape, ByVal yShape As Shape) As Shape
        Dim zShape As New Shape()
        zShape.Width = (xShape.Width + yShape.Width)
        zShape.Height = (xShape.Height + yShape.Height)
        Return zShape
    End Operator

    Public Overloads Shared Operator -(ByVal xShape As Shape, ByVal yShape As Shape) As Shape
        Dim zShape As New Shape()
        zShape.Width = (xShape.Width - yShape.Width)
        zShape.Height = (xShape.Height - yShape.Height)
        Return zShape
    End Operator

    Public Overloads Shared Operator *(ByVal xShape As Shape, ByVal yShape As Shape) As Shape
        Dim zShape As New Shape()
        zShape.Width = (xShape.Width * yShape.Width)
        zShape.Height = (xShape.Height * yShape.Height)
        Return zShape
    End Operator

    Public Overloads Shared Operator /(ByVal xShape As Shape, ByVal yShape As Shape) As Shape
        Dim zShape As New Shape()
        zShape.Width = (xShape.Width / yShape.Width)
        zShape.Height = (xShape.Height / yShape.Height)

        Return zShape
    End Operator
End Class
```

โค้ด VC# 2005 ที่ 4-4 การทำ Operator Overload (Class1.cs)

```
public class Shape
{
    private double _Width = 0.0;
    private double _Height = 0.0;

    public double Width
    {
        get{return _Width;}
        set{_Width = value;}
    }

    public double Height
    {
        get{return _Height;}
        set{_Height = value;}
    }
```

```
    public static Shape operator +(Shape xShape, Shape yShape)
    {
        Shape zShape = new Shape();
        zShape.Width = xShape.Width + yShape.Width;
        zShape.Height = xShape.Height + yShape.Height;
        return zShape;
    }

    public static Shape operator -(Shape xShape, Shape yShape)
    {
        Shape zShape = new Shape();
        zShape.Width = xShape.Width - yShape.Width;
        zShape.Height = xShape.Height - yShape.Height;
        return zShape;
    }

    public static Shape operator * (Shape xShape, Shape yShape)
    {
        Shape zShape = new Shape();
        zShape.Width = xShape.Width * yShape.Width;
        zShape.Height = xShape.Height * yShape.Height;
        return zShape;
    }

    public static Shape operator / (Shape xShape, Shape yShape)
    {
        Shape zShape = new Shape();
        try
        {
            zShape.Width = xShape.Width / yShape.Width;
            zShape.Height = xShape.Height / yShape.Height;
        }
        catch (DivideByZeroException ex)
        {

        }
        return zShape;
    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า Shape ประกอบด้วย

คุณสมบัติ/เมธอด	รายละเอียด
คุณสมบัติ Width	ความกว้าง
คุณสมบัติ Height	ความสูง

ผู้เขียนอยากให้คลาส Shape สามารถนำมาบวก, ลบ, คูณ หรือหารกันได้ จึงทำ Operator Overload ตัวดำเนินการด้านคณิตศาสตร์ทั้ง 4 ตัว ยกตัวอย่างเช่น

ตัวดำเนินการบวก (+) สร้างเมธอดแบบ Shared (static) ที่ชื่อว่า Operator + กำหนดให้รับพารามิเตอร์ 2 ตัวที่มีชนิดข้อมูลเป็น Shape เพราะว่าเราต้องการกำหนดให้คลาส Shape บวกกันได้ และเราเป็นผู้กำหนดเองว่าถ้านำคลาส Shape มาบวกกันให้ทำอย่างไร ในกรณีนี้ให้ทำ

- นำความกว้าง (คุณสมบัติ Width) ทั้ง 2 คลาสมาบวกกัน
- นำความสูง (คุณสมบัติ Height) ทั้ง 2 คลาสมาบวกกัน

VB 2005

```
Public Overloads Shared Operator +(ByVal xShape As Shape, ByVal yShape As Shape) As Shape
    Dim zShape As New Shape()
    zShape.Width = (xShape.Width + yShape.Width)
    zShape.Height = (xShape.Height + yShape.Height)
    Return zShape
End Operator
```

VC# 2005

```
public static Shape operator +(Shape xShape, Shape yShape)
{
    Shape zShape = new Shape();
    zShape.Width = xShape.Width + yShape.Width;
    zShape.Height = xShape.Height + yShape.Height;
    return zShape;
}
```

NOTE

การใช้งานส่วนประกอบในคลาสแบบ Shared หรือ static ผู้เขียนจะกล่าวอย่างละเอียดอีกครั้งในบทที่ 6

ส่วนตัวดำเนินการอื่นๆ (ลบ, คูณ และหาร) ให้ทำเช่นกัน เช่น ตัวดำเนินการหารก็จะกำหนดให้

- นำความกว้างของคลาส Shape ตัวที่ 1 (xShape) หารด้วยความกว้างของคลาส Shape ตัวที่ 2 (yShape)
- นำความสูงของคลาส Shape ตัวที่ 1 (xShape) หารด้วยความสูงของคลาส Shape ตัวที่ 2 (yShape)

VB 2005

```
Public Overloads Shared Operator /(ByVal xShape As Shape, ByVal yShape As Shape) As Shape
    Dim zShape As New Shape()
    zShape.Width = (xShape.Width / yShape.Width)
    zShape.Height = (xShape.Height / yShape.Height)

    Return zShape
End Operator
```

VC# 2005

```
public static Shape operator / (Shape xShape, Shape yShape)
{
    Shape zShape = new Shape();
    zShape.Width = xShape.Width / yShape.Width;
    zShape.Height = xShape.Height / yShape.Height;

    return zShape;
}
```

หลังจากที่คุณกำหนดวิธีการบวก, ลบ, คูณ และหารให้กับคลาส Shape แล้ว เขียนโค้ดทดสอบใน Form1 ดังนี้

โค้ด VB 2005 ที่ 4-4 การทำ Operator Overload (Form1.vb)

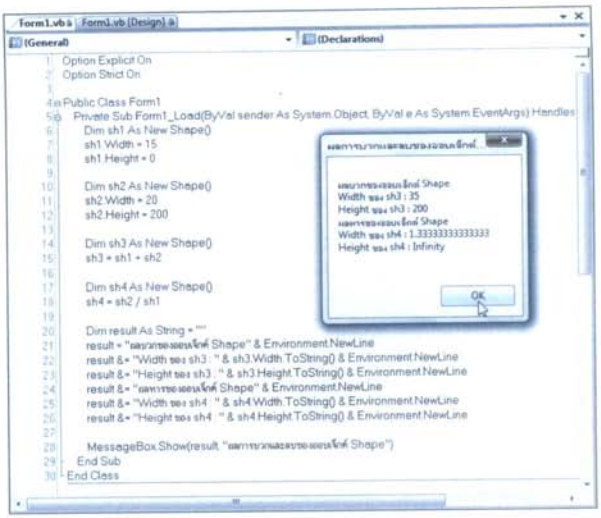
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim sh1 As New Shape()
        sh1.Width = 15
        sh1.Height = 0

        Dim sh2 As New Shape()
        sh2.Width = 20
        sh2.Height = 200

        Dim sh3 As New Shape()
        sh3 = sh1 + sh2

        Dim sh4 As New Shape()
        sh4 = sh2 / sh1

        Dim result As String = ""
        result = "ผลบวกของออบเจ็กต์ Shape" & Environment.NewLine
```

```
        result &= "Width ของ sh3 : " & sh3.Width.ToString() & Environment.NewLine
        result &= "Height ของ sh3 : " & sh3.Height.ToString() & Environment.NewLine
        result &= "ผลหารของออบเจ็กต์ Shape" & Environment.NewLine
        result &= "Width ของ sh4 : " & sh4.Width.ToString() & Environment.NewLine
        result &= "Height ของ sh4 : " & sh4.Height.ToString() & Environment.NewLine


        MessageBox.Show(result, "ผลการบวกและลบของออบเจ็กต์ Shape")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 4-4 การทำ Operator Overload (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Shape sh1 = new Shape();
        sh1.Width = 15;
        sh1.Height = 0;

        Shape sh2 = new Shape();
        sh2.Width = 20;
        sh2.Height = 200;

        Shape sh3 = new Shape();
        sh3 = sh1 + sh2;

        Shape sh4 = new Shape();
        sh4 = sh2 / sh1;

        string result = "";
        result = "ผลบวกของออบเจ็กต์ Shape" + Environment.NewLine;
        result += "Width ของ sh3 : " + sh3.Width.ToString() + Environment.NewLine;
        result += "Height ของ sh3 : " + sh3.Height.ToString() + Environment.NewLine;
        result += "ผลหารของออบเจ็กต์ Shape" + Environment.NewLine;
        result += "Width ของ sh4 : " + sh4.Width.ToString() + Environment.NewLine;
        result += "Height ของ sh4 : " + sh4.Height.ToString() + Environment.NewLine;

        MessageBox.Show(result, "ผลการบวกและลบของออบเจ็กต์ Shape");
    }
```

รูปที่ 4-3
ผลการรัน
ตัวอย่างที่ 4-4



ผู้เขียนสร้างออบเจ็กต์ Shape ขึ้นมา 4 ตัว โดยที่ 2 ตัวแรกชื่อว่า sh1 และ sh2 กำหนดความกว้างและความสูงแตกต่างกัน สังเกตว่าความสูงของออบเจ็กต์ sh1 กำหนดค่าเป็น 0 เพื่อทดสอบกรณีหารนั่นเอง

VB 2005

```
Dim sh1 As New Shape()
sh1.Width = 15
sh1.Height = 0

Dim sh2 As New Shape()
sh2.Width = 20
sh2.Height = 200
```

VC# 2005

```
Shape sh1 = new Shape();
sh1.Width = 15;
sh1.Height = 0;

Shape sh2 = new Shape();
sh2.Width = 20;
sh2.Height = 200;
```

ส่วนออบเจ็กต์ sh3 และ sh4 เกิดจากการบวกและหารกันระหว่างออบเจ็กต์ sh1 กับ sh2

VB 2005

```
Dim sh3 As New Shape()
sh3 = sh1 + sh2

Dim sh4 As New Shape()
sh4 = sh2 / sh1
```

VC# 2005

```
Shape sh3 = new Shape();
sh3 = sh1 + sh2;

Shape sh4 = new Shape();
sh4 = sh2 / sh1;
```

ความกว้างและความสูงของออบเจ็กต์ sh3 เกิดจากความกว้างและความสูงของออบเจ็กต์ sh1 บวกกับออบเจ็กต์ sh2 ส่วนความกว้างและความสูงของออบเจ็กต์ sh4 เกิดจากการหารกันระหว่างออบเจ็กต์ sh2 กับออบเจ็กต์ sh1 พบว่าในกรณีที่ส่วนเป็นศูนย์แล้วไม่มีการดักจับ Error ค่าที่ได้คือ Infinity

บทสรุปที่ได้ก็คือ ตัวดำเนินการคณิตศาสตร์ใช้กับ Type ชนิดตัวเลข แต่เมื่อคุณทำ Overload Operator คุณสามารถนำมาใช้กับคลาสของคุณได้เช่นกัน

พื้นฐานการทำ Override Method

การทำ Override Method เป็นการแก้ไขการทำงานของเมธอดเดิมที่มีอยู่ให้แตกต่างไปจากเดิม ถ้าจะแปลความหมายของคำว่า Override ก็จะได้ประมาณเขียน (การทำงาน) ทับของเดิมที่มีอยู่ แต่เนื่องจากว่าเนื้อหาการทำ Override Method เกี่ยวข้องกับการสืบทอดคลาส (Inheritance) มากพอสมควร ในข้างต้นนี้ผู้เขียนขอกล่าวถึงวิธีการทำ Override Method ของ .NET ที่น่าสนใจก่อน เพื่อให้คุณผู้อ่านเห็นประโยชน์ของการทำ Override

NOTE

การทำ Override Method ผู้เขียนจะกล่าวอย่างละเอียดอีกครั้ง เมื่อถึงเนื้อหาของการสืบทอดคลาสในบทที่ 8 เพราะว่าเมธอดที่คุณต้องการทำ Override ต้องมีการยินยอมหรืออนุญาตให้ทำก่อน

การทำ Override เมธอด ToString()

เมธอด ToString() เป็นของคลาส Object ทำหน้าที่แปลงชนิดข้อมูลปัจจุบันเป็นข้อมูลชนิด String เป็นเมธอดของ .NET Framework ที่ใกล้ชิดกับเรา และถูกนำมาใช้บ่อยครั้งมากที่สุด โดยเฉพาะอย่างยิ่งการแสดงผลของคอนโทรลต่างๆ โดยส่วนใหญ่แล้วต้องการข้อมูลชนิด String นั่นเอง เช่น คอนโทรล TextBox, Label ผ่านทางคุณสมบัติ Text เป็นต้น

หน้าที่ปกติของเมธอด ToString() คือ แปลงชนิดข้อมูลปัจจุบันเป็นข้อมูลชนิด String แต่คุณสามารถแก้ไข (Override) ให้เมธอด ToString() ทำงานแตกต่างไปจากเดิมได้เช่นกัน โดยที่เมธอด ToString() ของคลาส Object ยินยอมให้คุณทำ Override ได้ ให้ดูตัวอย่างที่ 4-5 การทำ Override เมธอด ToString()

โค้ด VB 2005 และ VC# 2005 ที่ 4-5 การทำ Override เมธอด ToString()

VB 2005 (Customer.vb)

```
Public Class Customer
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
```

VC# 2005 (Customer.cs)

```
public class Customer
{
    private string _FirstName="";
    private string _LastName = "";

    public string FirstName
    {
        get { return _FirstName; }
        set { _FirstName = value; }
    }

    public string LastName
    {
        get { return _LastName; }
        set { _LastName = value; }
```

```
        Return _LastName
    End Get
    Set(ByVal value As String)
        _LastName = value
    End Set
End Property

Public Overrides Function ToString() As String
    Return _FirstName + " " + _LastName
End Function
End Class
```

```
    }

    public override string ToString()
    {
        return _FirstName + " " + _LastName;
    }
}
```

ผู้เขียนสร้างคลาส Customer ขึ้นมาประกอบด้วย 2 คุณสมบัติคือ FirstName และ LastName ที่เพิ่มเติมขึ้นมาคือ มีการใช้คำสั่ง Overrides (override) เมธอด ToString() ด้วย

VB 2005

```
Public Overrides Function ToString() As String
    Return _FirstName + " " + _LastName
End Function
```

VC# 2005

```
public override string ToString()
{
    return _FirstName + " " + _LastName;
}
```

ผู้เขียนกำหนดไว้ว่าเมื่อใดก็ตามที่มีการใช้เมธอด ToString() เฉพาะของคลาส Customer ให้คืนค่ากลับมาเป็นชื่อ (ฟิลด์ _FirstName) และนามสกุล (ฟิลด์ _LastName) แล้วเว้นวรรคช่องว่างระหว่างชื่อและนามสกุลด้วย

เห็นได้ว่าเมธอด ToString() ของคลาส Customer ทำงานแตกต่างไปจากเมธอด ToString() เดิมของคลาส Object เพราะเราแก้ไขโดยการทำ Override เมธอด ToString() นั่นเอง การใช้งานคลาส Customer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 4-5 การทำ Override เมธอด ToString()

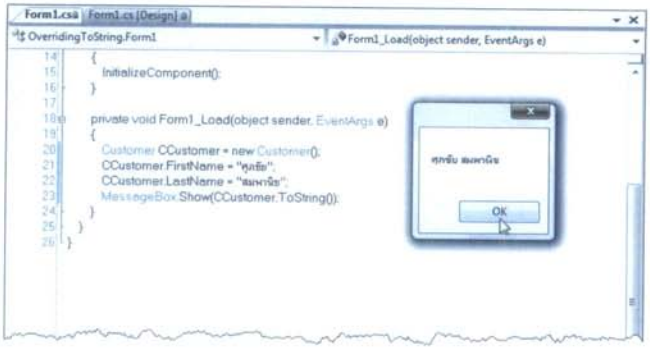
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim CCustomer As New Customer()
        CCustomer.FirstName = "ศุภชัย"
        CCustomer.LastName = "สมพานิช"
        MessageBox.Show(CCustomer.ToString())
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
    {
        Customer CCustomer = new Customer();
        CCustomer.FirstName = "ศุภชัย";
        CCustomer.LastName = "สมพานิช";
        MessageBox.Show(CCustomer.ToString());
    }
```

รูปที่ 4-4 ผลการรันตัวอย่างที่ 4-5



จากรูปที่ 4-4 ผู้เขียนสร้างออบเจ็กต์ Customer ที่ชื่อว่า CCustomer ขึ้นมา แล้วกำหนดชื่อและนามสกุลตามที่ต้องการ เมื่อมีการใช้เมธอด ToString() ของออบเจ็กต์ Customer ผลที่ได้คือ ชื่อ-สกุลตามที่กำหนดไว้นั่นเอง เห็นได้ว่าเราสามารถกำหนดการทำงานของเมธอด ToString() ใหม่ตามที่เราต้องการ โดยการทำ Override Method

การทำ Overload เมธอดที่ถูก Override

เนื้อหาที่ผ่านมาสรุปได้ว่าการทำ Overload หมายถึง การสร้างเมธอดที่มีชื่อซ้ำกัน แต่ Signature ต่างกัน ส่วนการทำ Override หมายถึง การแก้ไขการทำงานของเมธอดเดิมที่มีอยู่ให้แตกต่างไปจากเดิม

เราสามารถทำ Overload กับเมธอดที่ถูกแก้ไข (Override) ได้เช่นกัน เพราะว่าเมธอดที่คุณ Override มาก็คือ เมธอดหนึ่งเท่านั้น ให้ดูตัวอย่างที่ 4-6 การทำ Overload เมธอดที่ถูก Override

โค้ด VB 2005 และ VC# 2005 ที่ 4-6 การทำ Overload เมธอดที่ถูก Override

VB 2005 (Customer.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property
```

VC# 2005 (Customer.cs)

```
public class Customer
{
    private string _FirstName = "";
    private string _LastName = "";

    public string FirstName
    {
        get { return _FirstName; }
        set { _FirstName = value; }
    }

    public string LastName
    {
        get { return _LastName; }
        set { _LastName = value; }
```

```
    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return _FirstName + " " + _LastName
    End Function

    Public Overloads Function ToString(ByVal Initial As String)
As String
        Return Initial & _FirstName & " " & _LastName
    End Function
End Class
```

```
    }

    public override string ToString()
    {
        return _FirstName + " " + _LastName;
    }

    public string ToString(string Initial)
    {
        return Initial + _FirstName + " " + _LastName;
    }
}
```

ผู้เขียนแก้ไขคลาส Customer ใหม่ เห็นได้ว่าเป็นการทำ Override กับเมธอด ToString() เช่นเดิม เพียงแต่ว่าผู้เขียนสร้างเมธอด ToString() ขึ้นมาอีก 1 เมธอด ตามกติกาของการทำ Overload ระบุไว้ว่า Signature ต้องแตกต่างกัน ดังนั้น เมธอด ToString() ชุดที่ 2 จึงกำหนดให้รับพารามิเตอร์ 1 ตัว เป็นข้อมูลชนิด String (string) เพื่อให้ผู้ใช้ส่งคำนำหน้าชื่อ (พารามิเตอร์ Initial) เข้ามานั่นเอง

VB 2005

```
    Public Overrides Function ToString() As String
        Return _FirstName + " " + _LastName
    End Function

    Public Overloads Function ToString(ByVal Initial As String) As String
        Return Initial & _FirstName & " " & _LastName
    End Function
```

VC# 2005

```
    public override string ToString()
    {
        return _FirstName + " " + _LastName;
    }

    public string ToString(string Initial)
    {
        return Initial + _FirstName + " " + _LastName;
    }
```

ส่งผลให้เมื่อใดก็ตามที่มีการเรียกใช้เมธอด ToString() เฉพาะของคลาส Customer จะมีอยู่ 2 แบบคือ

1. ถ้าไม่ส่งพารามิเตอร์ใดๆ เข้ามา ก็จะคืนค่าเป็นชื่อ-สกุล
2. แต่ถ้าส่งพารามิเตอร์เข้ามา เช่น คำนำหน้าชื่อ ค่าที่ส่งกลับคือ คำนำหน้าชื่อ (พารามิเตอร์ Initial) + ชื่อ (ฟิลด์ _FirstName) + นามสกุล (ฟิลด์ _LastName)

การใช้งานคลาส Customer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 4-6 การทำ Overload เมธอดที่ถูก Override

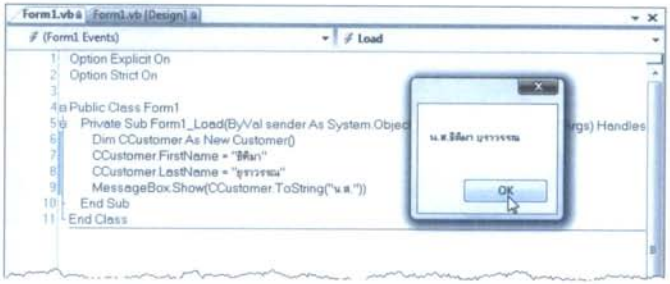
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On


Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim CCustomer As New Customer()
        CCustomer.FirstName = "ธิติมา"
        CCustomer.LastName = "ยุราวรรณ"
        MessageBox.Show(CCustomer.ToString("น.ส."))
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Customer CCustomer = new Customer();
    CCustomer.FirstName = "ธิติมา";
    CCustomer.LastName = "ยุราวรรณ";
    MessageBox.Show(CCustomer.ToString("น.ส."));
}
```

รูปที่ 4-5 ผลการรันตัวอย่างที่ 4-6



จากรูปที่ 4-5 ผู้เขียนทดสอบส่งคำนำหน้าชื่อให้กับเมธอด ToString() ของออบเจ็กต์ Customer ที่ชื่อว่า CCustomer เห็นได้ว่าค่าที่คืนกลับมาจากเมธอด ToString() คำนำหน้าชื่อ + ชื่อ + นามสกุลตามที่เรากำหนดไว้นั่นเอง

การทำ Override เมธอด Equals()

เมธอด Equals() เป็นเมธอดของคลาส Object ทำหน้าที่เปรียบเทียบของ 2 อย่าง ถ้าจะแปลเมธอด Equals() เป็นคำพูดก็จะได้ประมาณ "เท่ากันใช่หรือไม่?" การเปรียบเทียบว่าเท่ากันหรือไม่ ต้องดูว่าเราเปรียบเทียบอะไร เพราะว่า Type ของ .NET สามารถแยกได้ 2 ประเภทคือ Value Type และ Reference Type

การเปรียบเทียบ Value Type (ชนิดข้อมูลตัวเลขต่างๆ Integer (int), Double (double)) Type ประเภทนี้ดูไม่ยาก เช่น เรารู้ว่า 5 เท่ากับ 5 ส่วน 6 น้อยกว่า 7 เป็นต้น

ประเด็นที่เราต้องสนใจก็คือ คำว่าเท่ากันหรือไม่เท่ากันของ Reference Type (คลาสที่คุณสร้างขึ้นมาเอง, String) คืออะไร ให้ดูตัวอย่างที่ 4-7 พื้นฐานการใช้งานเมธอด Equals()

โค้ด VB 2005 และ VC# 2005 ที่ 4-7 พื้นฐานการใช้งานเมธอด Equals()

VB 2005 (Customer.vb)

```
Option Explicit On
Option Strict On

Public Class Customer

End Class
```

VC# 2005 (Customer.cs)

```
public class Customer
{

}
```

ผู้เขียนสร้างคลาสขึ้นมา 1 คลาสชื่อว่า Customer เป็นตัวแทนของ Reference Type เราต้องการหาความหมายของคำว่าเท่ากันหรือไม่เท่ากันของ Reference Type ดังนั้น คลาส Customer จึงไม่มีโค้ดใดๆ ส่วนการทดสอบอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 4-7 พื้นฐานการใช้งานเมธอด Equals()

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim C1 As New Customer()
        Dim C2 As New Customer()
        If C1.Equals(C2) = True Then
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
เหมือนกัน")
        Else
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
ไม่เหมือนกัน")
        End If
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Customer C1 = new Customer();
    Customer C2 = new Customer();
    if (C1.Equals(C2) == true)
    {
        MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือนกัน");
    }
    else
    {
        MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : ไม่เหมือน
กัน");
    }

    C1 = C2;
    if (C1.Equals(C2) == true)
    {
```

```
        C1 = C2
        If C1.Equals(C2) = True Then
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือน
กัน")
        End If

        Dim C3 As Customer = Nothing
        Dim C4 As Customer = Nothing
        If C3 Is Nothing Then
            MessageBox.Show("C3 เก็บค่าเริ่มต้น Nothing")
        End If

        If C4 Is Nothing Then
            MessageBox.Show("C4 เก็บค่าเริ่มต้น Nothing")
        End If

        If Object.Equals(C3, C4) Then
            MessageBox.Show("C3 กับ C4 เหมือนกัน")
        End If
    End Sub
End Class
```

```
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือนกัน");
        }

        Customer C3 = null;
        Customer C4 = null;
        if (C3 == null)
        {
            MessageBox.Show("C3 เก็บค่าเริ่มต้น null");
        }

        if (C4 == null)
        {
            MessageBox.Show("C4 เก็บค่าเริ่มต้น null");
        }

        if (object.Equals(C3, C4))
        {
            MessageBox.Show("C3 กับ C4 เหมือนกัน");
        }
    }
```

วิธีการดูโค้ดชุดนี้ก็คือ ให้ดูทีละส่วน ส่วนแรกผู้เขียนสร้างอินสแตนซ์ของคลาส Customer ขึ้นมา 2 ชุด ด้วยคำสั่ง New (new) สิ่งที่ได้คือ ออบเจ็กต์ Customer ที่ชื่อว่า C1 และ C2 ตามลำดับ เพราะความที่ Reference Type ไม่ได้เก็บค่าไว้กับตัวมันเอง แต่เป็นการชี้หรืออ้างอิงตำแหน่งของออบเจ็กต์ Customer ไว้ โดยที่

- ออบเจ็กต์ C1 ชี้ไปยังอินสแตนซ์ของคลาส Customer
- ส่วนออบเจ็กต์ C2 ก็ชี้ไปยังอินสแตนซ์ของคลาส Customer เช่นกัน แต่เป็นคนละอินสแตนซ์

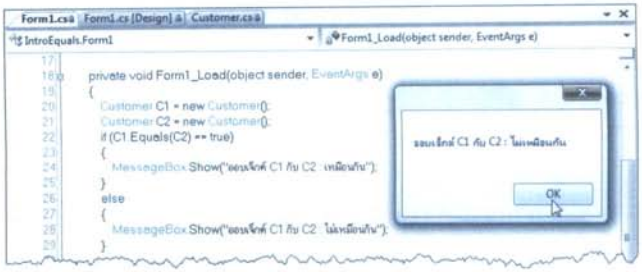
VB 2005

```
Dim C1 As New Customer()
Dim C2 As New Customer()
If C1.Equals(C2) = True Then
    MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือนกัน")
Else
    MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : ไม่เหมือนกัน")
End If
```

VC# 2005

```
Customer C1 = new Customer();
Customer C2 = new Customer();
if (C1.Equals(C2) == true)
{
    MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือนกัน");
}
else
{
    MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : ไม่เหมือนกัน");
}
```

รูปที่ 4-6
ผลการทดสอบ
ส่วนแรก



ผลที่ได้คือ ออบเจ็กต์ C1 และออบเจ็กต์ C2 เป็นข้อมูลชนิดเดียวกัน (มี Type เดียวกัน) คือ Customer (พร้อมที่จะชี้หรืออ้างอิงออบเจ็กต์ Customer) แต่ออบเจ็กต์ C1 และออบเจ็กต์ C2 ชี้คนละอินสแตนซ์ ส่งผลให้ออบเจ็กต์ C1 ไม่เท่ากับ C2 ด้วยเหตุผลนี้นั่นเอง

คำถามที่ตามมาก็คือ เราจะทำให้ออบเจ็กต์ C1 เท่ากับออบเจ็กต์ C2 ได้อย่างไร วิธีการก็คือ เราต้องกำหนดให้ออบเจ็กต์ทั้ง 2 ตัวนี้ชี้ที่อินสแตนซ์เดียวกัน ดังโค้ดต่อไปนี้

VB 2005

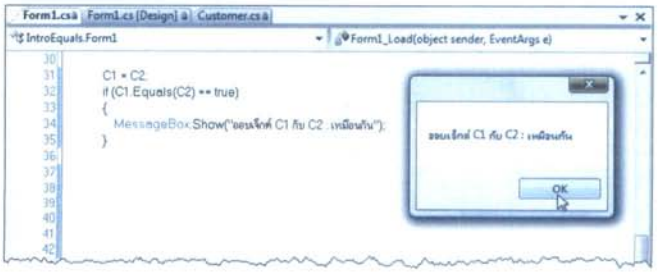
```
C1 = C2
If C1.Equals(C2) = True Then
   MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือนกัน")
End If
```

VC# 2005

```
C1 = C2;
if (C1.Equals(C2) == true)
{
   MessageBox.Show("ออบเจ็กต์ C1 กับ C2 : เหมือนกัน");
}
```

เห็นได้ว่าการกำหนดค่าให้กับ Reference Type โดยใช้เครื่องหมาย = ไม่ได้หมายความว่า ให้ถ่ายค่าจากออบเจ็กต์ C2 ไปสู่ C1 แต่เป็นการกำหนดให้ออบเจ็กต์ C2 ชี้หรืออ้างอิงตำแหน่งเดียวกับออบเจ็กต์ C1 ที่มันชี้อยู่ ส่งผลให้ออบเจ็กต์ C1 และออบเจ็กต์ C2 ชี้ที่อินสแตนซ์เดียวกัน ผลที่ได้ดังรูปที่ 4-7

รูปที่ 4-7
ผลการทดสอบ
ส่วนที่ 2



บทสรุปที่ได้คือ Reference Type จะเหมือนกันหรือเท่ากันก็ต่อเมื่อ กำหนดให้ชี้ที่อินสแตนซ์เดียวกันนั่นเอง

NOTE

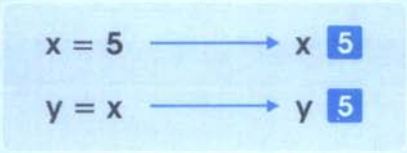
ส่วนการกำหนดค่าให้กับ Value Type ด้วยเครื่องหมาย = แตกต่างจาก Reference Type อย่างไร ทดสอบได้โดยการสร้างตัวแปร x และตัวแปร y ขึ้นมา มี Type เป็น Integer (int) ดังโค้ดต่อไปนี้

VB 2005	VC# 2005

Dim x As Integer
Dim y As Integer
x = 5
y = x
	int x;
	int y;
	x = 5;
	y = x;

กำหนดค่าให้ตัวแปร x เท่ากับ 5 เมื่อมีการกำหนดให้ y=x การทำงานที่เกิดขึ้นก็คือ ค่าของตัวแปร x จะถ่ายค่าไปยังตัวแปร y ส่งผลให้ตัวแปร y เก็บค่าของ 5 ไว้เช่นกัน กล่าวได้อีกนัยหนึ่งก็คือ ทั้งตัวแปร x และตัวแปร y ต่างก็เก็บค่า 5 ไว้ที่ตัวมันเอง ดังรูปที่ 4-8

รูปที่ 4-8
การเก็บค่าของ Value Type



อีกกรณีหนึ่งที่เราสามารถทำให้ Reference Type เท่ากันได้ก็คือ ถ้าคุณประกาศตัวแปรออบเจ็กต์ C3 และ C4 ขึ้นมาเป็นข้อมูลชนิด Customer ในกรณีนี้เราประกาศไว้เฉยๆ ว่า C3 และ C4 เป็นข้อมูลชนิด Customer (มี Type เป็น Customer) แต่ยังไม่ได้ชี้ที่อินสแตนซ์ใดๆ เพราะว่ายังไม่มีการสร้างอินสแตนซ์ด้วยคำสั่ง New (new) ส่งผลให้ค่าเริ่มต้นของ C3 และ C4 คือ ค่า Nothing ใน VB 2005 ส่วน VC# 2005 คือ null

VB 2005	VC# 2005

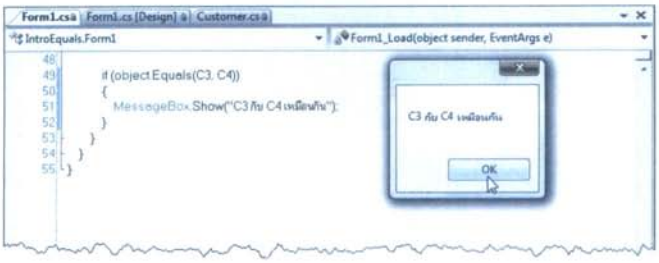
Dim C3 As Customer = Nothing
Dim C4 As Customer = Nothing
If C3 Is Nothing Then
MessageBox.Show("C3 เก็บค่าเริ่มต้น Nothing")
End If
 	
If C4 Is Nothing Then
MessageBox.Show("C4 เก็บค่าเริ่มต้น Nothing")
End If
	Customer C3 = null;
	Customer C4 = null;
	if (C3 == null)
	{
	MessageBox.Show("C3 เก็บค่าเริ่มต้น null");
	}
	
	if (C4 == null)
	{
	MessageBox.Show("C4 เก็บค่าเริ่มต้น null");
	}

เมื่อตัวแปรออบเจ็กต์ทั้ง 2 ตัว ยังไม่มีการชี้ไปที่อินสแตนซ์ใดๆ ผลที่ได้คือ ตัวแปรออบเจ็กต์ทั้ง 2 ตัวจะเท่ากัน ดังรูปที่ 4-9

VB 2005	VC# 2005

If Object.Equals(C3, C4) Then
MessageBox.Show("C3 กับ C4 เหมือนกัน")
End If
	if (object.Equals(C3, C4))
	{
	MessageBox.Show("C3 กับ C4 เหมือนกัน");
	}

รูปที่ 4-9
ผลการทดลอบส่วนที่ 3



บทสรุปที่ได้ก็คือ Reference Type จะเท่ากันอีกกรณีหนึ่งคือ ไม่มีการชี้ไปที่อินสแตนซ์ใดๆ นั่นเอง โดยที่มันจะแสดงเป็นค่า Nothing ใน VB 2005 หรือ null ใน VC# 2005

การตรวจสอบการเท่ากันหรือไม่เท่ากันของ Reference Type ใน .NET ยังมีอีก 1 วิธีคือ อาศัยเมธอด ReferenceEquals() ของคลาส System.Object ให้ดูตัวอย่างที่ 4-8 การตรวจสอบ Reference Type โดยอาศัยเมธอด ReferenceEquals()

โค้ด VB 2005 และ VC# 2005 ที่ 4-8 การตรวจสอบ Reference Type โดยอาศัยเมธอด ReferenceEquals()

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim C1 As New Customer()
        Dim C2 As New Customer()

        If Object.ReferenceEquals(C1, C2) = True Then
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
เหมือนกัน")
        Else
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
ไม่เหมือนกัน")
        End If

        C1 = C2
        If Object.ReferenceEquals(C1, C2) = True Then
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
เหมือนกัน")
        End If

        Dim C3 As Customer = Nothing
        Dim C4 As Customer = Nothing
        If C3 Is Nothing Then
            MessageBox.Show("C3 เก็บค่าเริ่มต้น Nothing")
        End If

        If C4 Is Nothing Then
            MessageBox.Show("C4 เก็บค่าเริ่มต้น Nothing")
        End If

        If Object.ReferenceEquals(C3, C4) Then
            MessageBox.Show("C3 กับ C4 เหมือนกัน")
        End If
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Customer C1 = new Customer();
        Customer C2 = new Customer();
        if (object.ReferenceEquals(C1,C2)==true)
        {
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
เหมือนกัน");
        }
        else
        {
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
ไม่เหมือนกัน");
        }

        C1 = C2;
        if (object.ReferenceEquals(C1, C2) == true)
        {
            MessageBox.Show("ออบเจ็กต์ C1 กับ C2 :
เหมือนกัน");
        }

        Customer C3 = null;
        Customer C4 = null;
        if (C3 == null)
        {
            MessageBox.Show("C3 เก็บค่าเริ่มต้น null");
        }

        if (C4 == null)
        {
            MessageBox.Show("C4 เก็บค่าเริ่มต้น null");
        }

        if (object.ReferenceEquals(C3, C4) == true)
        {
            MessageBox.Show("C3 กับ C4 เหมือนกัน");
        }
    }
```

ตัวอย่างโค้ดและผลการทำงานของตัวอย่างที่ 4-8 คล้ายกับตัวอย่างที่ 4-7 แตกต่างกันก็แต่เพียงเปลี่ยนวิธีการตรวจสอบอินสแตนซ์แต่ละชุดด้วยเมธอด ReferenceEquals() แทน

การ Override เมธอด Equals()

จากบทสรุปคำว่าเท่ากันหรือไม่เท่ากันของ Reference Type ที่ได้คือ ถ้าเป็นการชี้ไปที่อินสแตนซ์เดียวกัน หรือไม่มีการชี้ไปยังออบเจ็กต์ใดๆ (Nothing ใน VB 2005 หรือ null ใน VC# 2005) ผลที่ได้คือ Reference Type ดังกล่าวมีค่าเท่ากัน

คุณสามารถแก้ไขกฎเกณฑ์ของเมธอด Equals() ได้เช่นกัน โดยการทำ Override เพราะว่าเมธอด Equals() อนุญาตให้คุณทำ Override ได้เช่นกัน ให้ดูตัวอย่างที่ 4-9 การ Override เมธอด Equals()

โค้ด VB 2005 ที่ 4-9 การ Override เมธอด Equals() (Customer.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property

    Public Overrides Function Equals(ByVal other As Object) As Boolean
        If other IsNot Nothing Then
            If TypeOf other Is Customer Then
                If DirectCast(other, Customer)._FirstName = Me._FirstName _
                    AndAlso DirectCast(other, Customer)._LastName = Me._LastName Then
                    Return True
                End If
            End If
        End If
        Return False
    End Function
End Class
```

โค้ด VC# 2005 ที่ 4-9 การ Override เมธอด Equals() (Customer.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace OverridingEqual
{
    public class Customer
    {
        private string _FirstName = "";
        private string _LastName = "";

        public string FirstName
        {
            get{return _FirstName;}
            set{_FirstName = value;}
        }

        public string LastName
        {
            get{return _LastName;}
            set{_LastName = value;}
        }

        public override bool Equals(object other)
        {
            if (other != null)
            {
                if (other is Customer)
                {
                    if (((Customer)other)._FirstName == this._FirstName && ((Customer)other)._LastName == this._LastName)
                    {
                        return true;
                    }
                }
            }
            return false;
        }

        public override int GetHashCode()
        {
            return base.GetHashCode();
        }
    }
}
```

ผู้เขียนสร้างคลาส Customer ขึ้นมาประกอบด้วย 2 คุณสมบัติคือ FirstName และ LastName ที่น่าสนใจก็คือ เมธอด Equals() ที่ต้องการทำ Override นั่นเอง

วิธีการคือ เราต้องตรวจสอบระดับแรกก่อนเลยว่า ถ้าพารามิเตอร์ other (พารามิเตอร์ other มี Type เป็น Object ตาม Signature เดิมของเมธอด Equals()) เป็นออบเจ็กต์ Customer แล้วมีการทำงาน 2 ขั้นตอนซ้อนกันอยู่กล่าวคือ

1. ให้แปลง other จากชนิดข้อมูล Object เป็นออบเจ็กต์ Customer ก่อน ใน VB 2005 ใช้ฟังก์ชัน DirectCast() ส่วน VC# 2005 แปลง Type โดยการ Cast เมื่อถูกแปลง Type จาก Object เป็น Customer แล้ว ก็จะถึงขั้นตอนที่ว่าจะเปรียบเทียบกันอย่างไร ตามเงื่อนไขที่เราต้องการ
2. ในกรณีนี้กำหนดเงื่อนไขของคำว่าเท่ากันของออบเจ็กต์ Customer ไว้ 2 อย่างคือ
   - ชื่อ (ฟิลด์ _FirstName) ของออบเจ็กต์ Customer ที่ถูกส่งเข้ามา (ผ่านทางพารามิเตอร์ other) เท่ากับชื่อ (ฟิลด์ _FirstName) ของคลาสปัจจุบัน
   - นามสกุล (ฟิลด์ _LastName) ของออบเจ็กต์ Customer ที่ถูกส่งเข้ามา (ผ่านทางพารามิเตอร์ other เช่นกัน) เท่ากับนามสกุล (ฟิลด์ _LastName) ของคลาสปัจจุบัน

VB 2005

```
Public Overrides Function Equals(ByVal other As Object) As Boolean
    If other IsNot Nothing Then
        If TypeOf other Is Customer Then
            If DirectCast(other, Customer)._FirstName = Me._FirstName _
                AndAlso DirectCast(other, Customer)._LastName = Me._LastName Then
                Return True
            End If
        End If
    End If
    Return False
End Function
```

VC# 2005

```
public override bool Equals(object other)
{
    if (other != null)
    {
        if (other is Customer)
        {
            if (((Customer)other)._FirstName == this._FirstName && ((Customer)other)._LastName == this._LastName)
            {
                return true;
            }
        }
    }
    return false;
}

public override int GetHashCode()
{
    return base.GetHashCode();
}
```

NOTE

เนื่องจากการ Override เมธอด Equals() ภาษา VC# 2005 บังคับให้ต้อง Override เมธอด GetHashCode() ด้วย ในกรณีนี้เราไม่ต้องแก้ไขการทำงานใดๆ ของเมธอดนี้ จึงสั่งให้เมธอด GetHashCode() ทำงานเหมือนเดิม โดยใช้คำสั่ง base.GetHashCode()

เรากำหนดเงื่อนไขคำว่าเท่ากันของออบเจ็กต์ Customer ไว้ 2 อย่างคือ ต้องมีชื่อและนามสกุลเหมือนกัน ส่วนการทดสอบอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 4-9 การ Override เมธอด Equals()

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim C1 As New Customer()
        C1.FirstName = "ศุภชัย"
        C1.LastName = "สมพานิช"

        Dim C2 As New Customer()
        C2.FirstName = "ชลธิชา"
        C2.LastName = "ศรีดา"

        Dim C3 As New Customer()
        C3.FirstName = "ศุภชัย"
        C3.LastName = "สมพานิช"

        If C1.Equals(C2) = True Then
            MessageBox.Show("C1 กับ C2 : เหมือนกัน")
        Else
            MessageBox.Show("C1 กับ C2 : ต่างกัน")
        End If

        If C1.Equals(C3) = True Then
            MessageBox.Show("C1 กับ C3 : เหมือนกัน")
        Else
            MessageBox.Show("C1 กับ C3 : ต่างกัน")
        End If
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Customer C1 = new Customer();
    C1.FirstName = "ศุภชัย";
    C1.LastName = "สมพานิช";

    Customer C2 = new Customer();
    C2.FirstName = "ชลธิชา";
    C2.LastName = "ศรีดา";

    Customer C3 = new Customer();
    C3.FirstName = "ศุภชัย";
    C3.LastName = "สมพานิช";

    if (C1.Equals(C2) == true)
    {
        MessageBox.Show("C1 กับ C2 : เหมือนกัน");
    }
    else
    {
        MessageBox.Show("C1 กับ C2 : ต่างกัน");
    }

    if (C1.Equals(C3) == true)
    {
        MessageBox.Show("C1 กับ C3 : เหมือนกัน");
    }
    else
    {
        MessageBox.Show("C1 กับ C3 : ต่างกัน");
    }
}
```

วิธีการเปรียบเทียบก็คือ ผู้เขียนสร้างออบเจ็กต์ Customer ขึ้นมา 3 ตัว โดยตั้งชื่อว่า C1, C2 และ C3 ตามลำดับ สังเกตได้ว่าออบเจ็กต์ Customer ที่ชื่อว่า C1 กับ C3 มีชื่อ-นามสกุลเหมือนกัน

VB 2005

```
Dim C1 As New Customer()
C1.FirstName = "ศุภชัย"
C1.LastName = "สมพานิช"

Dim C2 As New Customer()
C2.FirstName = "ชลธิชา"
C2.LastName = "ศรีดา"

Dim C3 As New Customer()
C3.FirstName = "ศุภชัย"
C3.LastName = "สมพานิช"
```

VC# 2005

```
Customer C1 = new Customer();
C1.FirstName = "ศุภชัย";
C1.LastName = "สมพานิช";

Customer C2 = new Customer();
C2.FirstName = "ชลธิชา";
C2.LastName = "ศรีดา";

Customer C3 = new Customer();
C3.FirstName = "ศุภชัย";
C3.LastName = "สมพานิช";
```

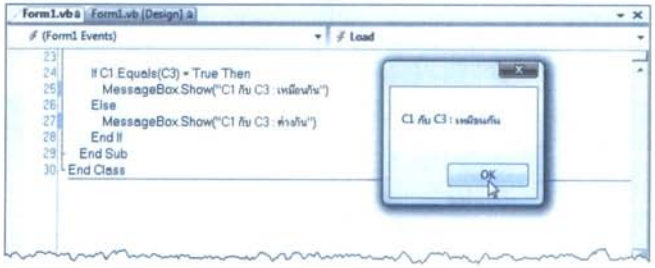
ออบเจ็กต์ C1 กับ C2 ต่างกันแน่นอนเพราะชื่อ-สกุลไม่เหมือนกัน ส่วน C1 กับ C3 เหมือนกันตรงตามเงื่อนไขที่เรากำหนดไว้นั่นเอง ดังรูปที่ 4-10

VB 2005

```
If C1.Equals(C3) = True Then
   MessageBox.Show("C1 กับ C3 : เหมือนกัน")
Else
   MessageBox.Show("C1 กับ C3 : ต่างกัน")
End If
```

VC# 2005

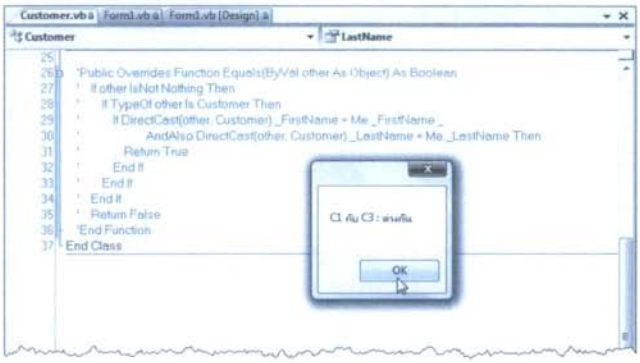
```
if (C1.Equals(C3) == true)
{
   MessageBox.Show("C1 กับ C3 : เหมือนกัน");
}
else
{
   MessageBox.Show("C1 กับ C3 : ต่างกัน");
}
```



รูปที่ 4-10 ผลการเปรียบเทียบระหว่าง C1 กับ C3

ให้คุณลองทำหมายเหตุโค้ดของเมธอด Equals() ที่อยู่ในคลาส Customer เพื่อยกเลิกการทำ Override ชั่วคราว แล้วทดสอบอีกรอบ ดังรูปที่ 4-11

รูปที่ 4-11 ผลการเปรียบเทียบระหว่าง C1 กับ C3 หลังจากยกเลิกการ Override เมธอด Equals()



จากรูปที่ 4-11 พบว่าออบเจ็กต์ C1 แตกต่างจาก C3 เป็นเพราะว่าเมื่อคุณใช้เมธอด Equals() เดิมที่ไม่มีการ Override ออบเจ็กต์ C1 ซึ่งคนละอินสแตนซ์กับออบเจ็กต์ C3 ผลที่ได้คือ ไม่เท่ากันหรือไม่เหมือนกันนั่นเอง

การ Override เหตุการณ์ (Events)

เหตุการณ์ (Events) ถือเป็น member function ประเภทหนึ่งเช่นกัน ซึ่งถูกกระตุ้นให้ทำงานเมื่อเกิดสภาวะใดสภาวะหนึ่ง คุณผู้อ่านที่พัฒนาแอพพลิเคชันประเภท Windows Application คงคุ้นเคยกับ member function ชนิดนี้เป็นอย่างดี เช่น เมื่อปุ่ม Button1 ถูกคลิก (Click()) แล้ว ให้ทำอะไรที่จะกล่าวในส่วนนี้ก็คือ คุณสามารถทำ Override กับเหตุการณ์ได้เช่นกัน

โดยปกติแล้วเหตุการณ์หนึ่งๆ ถูกกระตุ้นให้ทำงาน เมื่อมีการเชื่อมโยงกันระหว่างออบเจ็กต์กับเหตุการณ์นั้นๆ ผ่านทาง event handler แต่เนื้อหาของหัวข้อนี้ จะนำเสนอเฉพาะประเด็นการทำ Override เหตุการณ์เท่านั้น ให้ดูตัวอย่างที่ 4-10 การทำ Override เหตุการณ์

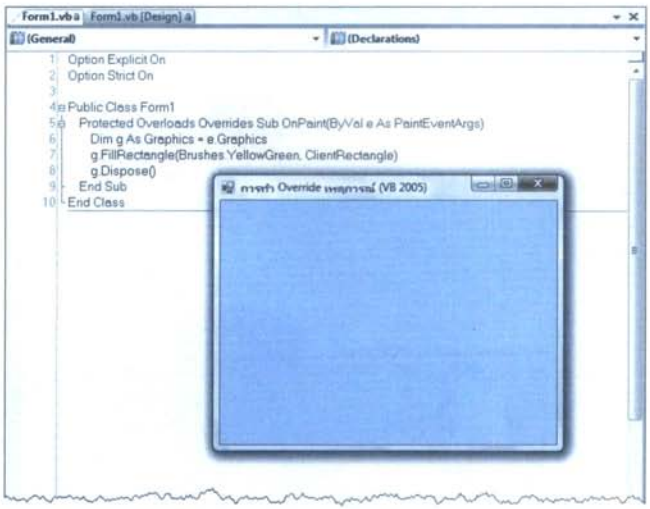
โค้ด VB 2005 ที่ 4-10 การ Override เหตุการณ์ Paint() ของฟอร์ม

```
Option Explicit On
Option Strict On

Public Class Form1
    Protected Overloads Overrides Sub OnPaint(ByVal e As PaintEventArgs)
        Dim g As Graphics = e.Graphics
        g.FillRectangle(Brushes.YellowGreen, ClientRectangle)
        g.Dispose()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 4-10 การ Override เหตุการณ์ Paint() ของฟอร์ม

```
protected override void OnPaint(PaintEventArgs e)
{
    Graphics g = e.Graphics;

    g.FillRectangle(Brushes.YellowGreen, ClientRectangle);
    g.Dispose();
}
```

รูปที่ 4-12
ผลการรัน
ตัวอย่างที่ 4-10

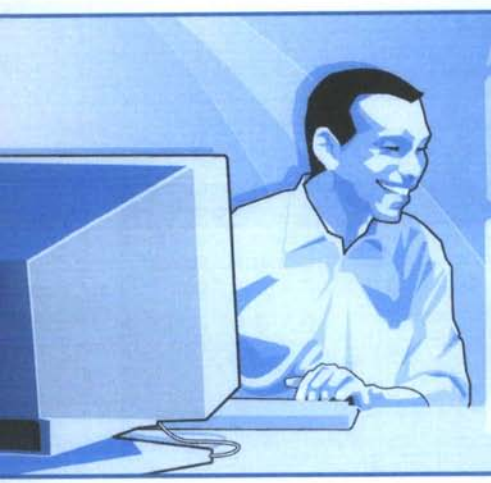


จากรูปที่ 4-12 เราเข้าไปแก้ไข (Override) เหตุการณ์ Paint() ของฟอร์ม กำหนดไว้ว่าให้ลงสีพื้นหลังของฟอร์มเป็นสีเขียว ซึ่งแตกต่างไปจากเหตุการณ์ Form1_Paint() เดิมที่ไม่มีการลงสีพื้นหลังแต่อย่างใด

สรุปท้ายบท

จากเนื้อหาที่ผ่านมาพบว่า เมธอดมีรายละเอียดค่อนข้างมากพอสมควร ล้วนแต่เป็นรากฐานที่สำคัญก่อนเข้าไปสู่ส่วนที่ 2 คือ ทำความรู้จักกับพารามิเตอร์ ซึ่งเป็นส่วนประกอบส่วนหนึ่งของเมธอดนั่นเอง


Programming in OOP style
Advanced .net



CHAPTER 5

พารามิเตอร์

บทนำ

องค์ประกอบส่วนหนึ่งของเมธอดที่มีความสำคัญเป็นอย่างยิ่งนั่นคือ พารามิเตอร์ (Parameter) หรืออาจจะเรียกอีกอย่างหนึ่งว่า อาร์กิวเมนต์ (Argument) ก็ได้ คุณจะได้ศึกษารายละเอียดปลีกย่อยที่จะทำให้คุณรู้จักกับพารามิเตอร์มากยิ่งขึ้น

ลักษณะของพารามิเตอร์ในเมธอด

พารามิเตอร์ต่างๆ ที่คุณนำมาใช้ในเมธอดบางครั้งจะเรียกว่า อาร์กิวเมนต์ (Argument) ซึ่งเป็นเนื้อหาอีกส่วนที่น่าสนใจเป็นอย่างยิ่ง เพราะโดยปกติแล้ว เราใช้พารามิเตอร์กันเสมอเมื่อมีการสร้างฟังก์ชันหรือเมธอดขึ้นมาใช้งาน ดังนั้น พารามิเตอร์อยู่ใกล้ชิดกับเราพอสมควร ต่อไปเราจะดูว่ามีอะไรน่าสนใจเกี่ยวกับพารามิเตอร์บ้าง

พารามิเตอร์ถ้าแบ่งตามพฤติกรรมของตัวมันเอง สามารถแบ่งได้ 3 รูปแบบคือ

- พารามิเตอร์ชนิดรับเข้า (Input) เป็นรูปแบบที่เราพบเห็นมากที่สุด
- พารามิเตอร์ชนิดส่งออก (Output)
- พารามิเตอร์ชนิดรับเข้าและส่งออกในเวลาเดียวกัน

พารามิเตอร์ชนิดรับเข้า

ให้คุณดูตัวอย่างโค้ดต่อไปนี้

VB 2005

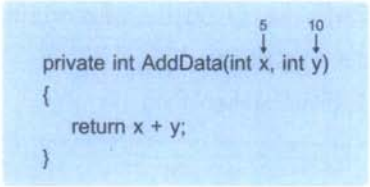
```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim result As Integer
    result = AddData(5, 10)
End Sub

Private Function AddData(ByVal x As Integer, ByVal y As Integer) As Integer
    Return x + y
End Function
```

VC# 2005

```
private void Form1_Load(object sender, EventArgs e)
{
    int result;
    result = AddData(5, 10);
}

private int AddData(int x, int y)
{
    return x + y;
}
```

โค้ดข้างต้นผู้เขียนสร้างฟังก์ชัน AddData() ขึ้นมาทำหน้าที่บวกตัวเลข 2 ตัว โดยกำหนดไว้ว่าต้องส่งค่าให้กับพารามิเตอร์ x และ y ซึ่งมี Type เป็น Integer (int) ผลบวกที่ได้คืนค่ากลับเป็นข้อมูลชนิด Integer (int)

รูปที่ 5-1
พารามิเตอร์แบบรับเข้า



ในเหตุการณ์ Form1_Load() เป็นจุดเรียกใช้ฟังก์ชัน AddData() เห็นได้ว่าต้องส่งค่าเข้ามา 2 ค่า เพื่อให้ตรงตาม Signature ของฟังก์ชัน AddData() พารามิเตอร์ x และ y มีลักษณะเป็นพารามิเตอร์ชนิดรับเข้าอย่างเดียว เพราะว่าทำหน้าที่รับค่าเข้ามาในฟังก์ชันนี้แล้วจะทำอะไรต่อ ก็แล้วแต่หน้าที่ของฟังก์ชันนั้นๆ

พารามิเตอร์ชนิดส่งออก

ให้ดูตัวอย่างที่ 5-1 การใช้งานคีย์เวิร์ด out

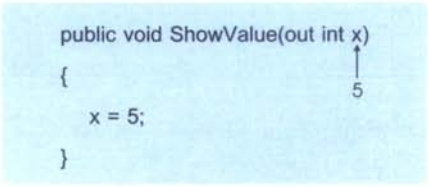
โค้ด VB 2005 ที่ 5-1 การใช้งานคีย์เวิร์ด out (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim test As Integer = 0
        ShowValue(test)

        MessageBox.Show(test.ToString(), "ค่าปัจจุบัน")
    End Sub

    Private Sub ShowValue(<Runtime.InteropServices.Out()> ByRef x As Integer)
        x = 5
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 5-1 การใช้งานคีย์เวิร์ด out (Form1.cs)

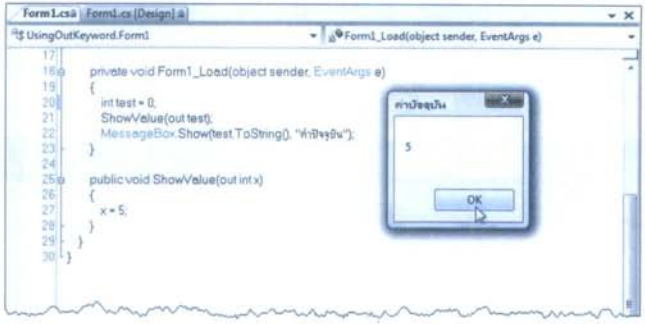
```
private void Form1_Load(object sender, EventArgs e)
{
    int test=0;
    ShowValue(out test);
    MessageBox.Show(test.ToString(), "ค่าปัจจุบัน");
}

public void ShowValue(out int x)
{
    x = 5;
}
```

รูปที่ 5-2
พารามิเตอร์แบบส่งออก



โค้ดข้างต้นผู้เขียนสร้างซับรูทีน (VB 2005) หรือ void function (VC# 2005) ที่ชื่อว่า ShowValue() ขึ้นมา โดยปกติแล้วซับรูทีนไม่มีการคืนค่าใดๆ แต่การใช้พารามิเตอร์แบบส่งออกเป็นคนละแบบกับการคืนค่า (Return Value)

รูปที่ 5-3
ค่าของตัวแปร test



การใช้พารามิเตอร์แบบส่งออก จะส่งตำแหน่งหน่วยความจำเข้ามาไม่ใช่ส่งค่าเข้ามา ในกรณีนี้จะส่งตำแหน่งของตัวแปร test เข้ามา ส่วนการทำงานในซับรูทีน ShowValue() กำหนดให้เปลี่ยนค่า x เป็น 5 หมายถึงให้ใส่ค่า 5 ให้กับพารามิเตอร์ x ณ หน่วยความจำตำแหน่งเดิม

ส่งผลให้ที่จุดเรียกใช้ซับรูทีนนี้ (ใน Form1_Load()) ค่าของตัวแปร test จึงเปลี่ยนจากค่าเริ่มต้น 0 เป็น 5 เห็นได้ว่าเราใช้พารามิเตอร์ x ในฐานะส่งค่าออกมา และค่าที่ได้ไม่ใช่ค่าที่ส่งกลับคืนมาจากซับรูทีน ShowValue() แต่อย่างใด

ผู้เขียนเปลี่ยน Type ของพารามิเตอร์ x ใหม่เป็น Double (double) กำหนดค่าเป็น 5.5 (ความละเอียดระดับทศนิยม) เมื่อคุณส่งค่าตัวแปร test เข้ามาซับรูทีน ShowValue() ไม่สามารถเขียนค่า 5.5 ลงในตำแหน่งหน่วยความจำของตัวแปร test ได้ เพราะว่าตัวแปร test มี Type เป็น Integer (int) หรือเลขจำนวนเต็มนั่นเอง ส่งผลให้โค้ดชุดนี้รันไม่ผ่าน และที่น่าสนใจมีอยู่ 1 กรณีคือ

VB 2005

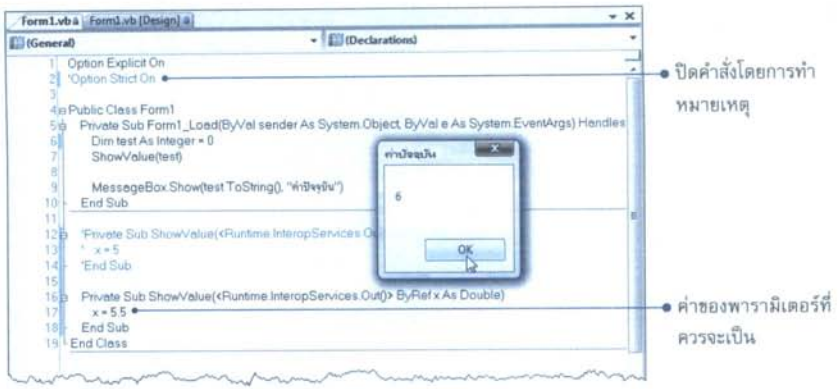
```
Private Sub ShowValue(<Runtime.InteropServices.Out()> ByRef x As Double)
    x = 5.5
End Sub
```

VC# 2005

```
public void ShowValue(out double x)
{
    x = 5.5;
}
```

ใน VC# 2005 ให้ความสำคัญกับ Type อย่างเคร่งครัด เพราะไม่ต้องการให้เกิดการสูญเสียค่าที่เก็บไว้นั่นเอง โค้ดชุดนี้จึงรันไม่ผ่านใน VC# 2005

ส่วน VB 2005 ถ้าคุณปิดเงื่อนไข Option Strict Off หมายถึง ลดความเข้มงวดในการตรวจสอบ Type ลง โค้ดชุดนี้จะรันผ่าน แต่ค่าที่ได้ผิดไปจากความเป็นจริงนั่นคือ ตัวแปร test จะมีค่าเป็น 6 ถ้านำไปใช้คำนวณต่อเกิด Logical Error แน่นอน



รูปที่ 5-4 ค่าที่อ่านได้

เห็นได้ว่าการเปิด Option Strict On ใน VB 2005 ช่วยป้องกัน Logical Error ได้อีกทางหนึ่ง การตรวจสอบและรักษา Type อย่างเคร่งครัดเป็นเรื่องที่ดีอีกเรื่องหนึ่งที่ต้องให้ความสนใจ เพื่อให้ได้โค้ดที่มีประสิทธิภาพมากที่สุด

พารามิเตอร์ชนิดรับเข้าและส่งออกในเวลาเดียวกัน

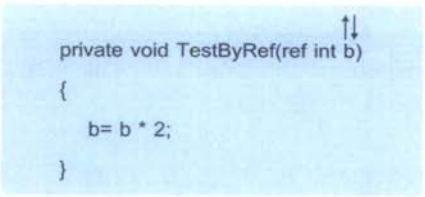
เรารู้จักพารามิเตอร์ลักษณะนี้ในชื่อที่เรียกว่า การส่งพารามิเตอร์แบบ Reference ให้ดูตัวอย่างที่ 5-2 พารามิเตอร์ชนิดรับเข้าและส่งออกในเวลาเดียวกัน

โค้ด VB 2005 ที่ 5-2 พารามิเตอร์ชนิดรับเข้าและส่งออกในเวลาเดียวกัน (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim OriginalNumber As Integer = 5

        TestByVal(OriginalNumber)
        MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByVal")

        TestByRef(OriginalNumber)
        MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByRef")
    End Sub

    Private Sub TestByVal(ByVal a As Integer)
        a = a * 2
    End Sub

    Private Sub TestByRef(ByRef b As Integer)
        b = b * 2
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 5-2 พารามิเตอร์ชนิดรับเข้าและส่งออกในเวลาเดียวกัน (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    int OriginalNumber = 5;

    TestByVal(OriginalNumber);
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByVal");

    TestByRef(ref OriginalNumber);
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByRef");
}

private void TestByVal(int a)
{
    a = a * 2;
}

private void TestByRef(ref int b)
{
    b = b * 2;
}
```



รูปที่ 5-5 พารามิเตอร์แบบนำเข้าและส่งออกในเวลาเดียวกัน

วิธีการดูโค้ดชุดนี้ก็คือ ผู้เขียนสร้างซับรูทีนขึ้นมา 2 ชุด ตั้งชื่อว่า TestByVal() และ TestByRef() ทั้ง 2 ซับรูทีนทำงานเหมือนกันคือ ให้คูณค่าที่รับเข้ามาด้วย 2 โดยที่

- ซับรูทีน TestByVal() เป็นการส่งค่าตามปกติ หรือเรียกอีกอย่างว่าเป็นพารามิเตอร์ชนิดรับเข้า
- ซับรูทีน TestByRef() กำหนดให้ส่งพารามิเตอร์แบบ Reference

VB 2005

```
Private Sub TestByVal(ByVal a As Integer)
    a = a * 2
End Sub

Private Sub TestByRef(ByRef b As Integer)
    b = b * 2
End Sub
```

VC# 2005

```
private void TestByVal(int a)
{
    a = a * 2;
}

private void TestByRef(ref int b)
{
    b = b * 2;
}
```

ที่จุดเรียกใช้ซับรูทีนอยู่ในเหตุการณ์ Form1_Load() ผู้เขียนสร้างตัวแปรขึ้นมา 1 ตัวชื่อว่า OriginalNumber กำหนดค่าเริ่มต้นเท่ากับ 5 จากนั้นส่งตัวแปรตัวนี้ ให้กับซับรูทีน TestByVal() กับ TestByRef() ตามลำดับ

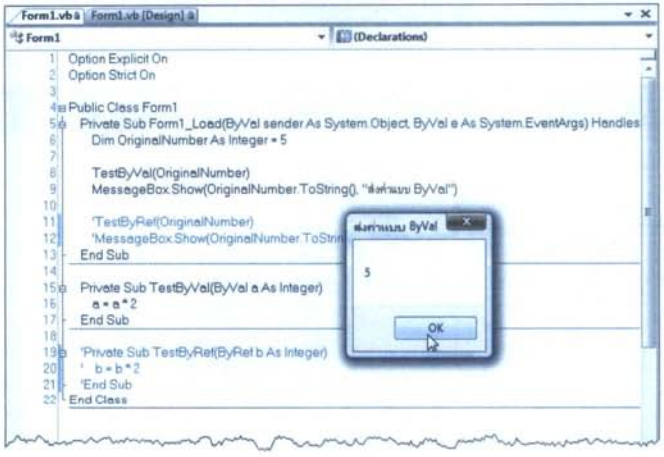
VB 2005

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim OriginalNumber As Integer = 5

    TestByVal(OriginalNumber)
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByVal")

    TestByRef(OriginalNumber)
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByRef")
End Sub
```

VC# 2005

```
private void Form1_Load(object sender, EventArgs e)
{
    int OriginalNumber = 5;

    TestByVal(OriginalNumber);
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByVal");

    TestByRef(ref OriginalNumber);
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByRef");
}
```

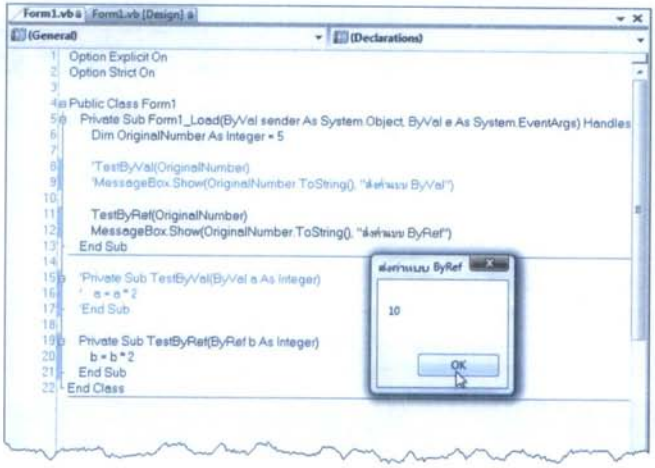
หลังจากซับรูทีน TestByVal() ทำงานผลที่ได้ ดังรูปที่ 5-6



รูปที่ 5-6 ผลการทำงานของซับรูทีน TestByVal()

จากรูปที่ 5-6 ค่าของตัวแปร OriginalNumber ไม่เปลี่ยนแปลงแต่อย่างใด การส่งค่าแบบ Value เป็นเพียงการนำค่าที่ส่งเข้ามาไปใช้งาน ซึ่งไม่เกี่ยวข้องกันเพราะเป็นตัวแปรคนละตัว

รูปที่ 5-7
หลังจากซับรูทีน TestByRef() ทำงาน



แต่ถ้าเป็นกรณีการส่งพารามิเตอร์แบบ Reference ตัวแปร OriginalNumber กลับถูกคูณค่าเป็น 10 ตามการทำงานของซับรูทีน TestByRef()

เหตุผลก็คือ การส่งพารามิเตอร์แบบ Reference เป็นการส่งตำแหน่งหน่วยความจำของตัวแปร OriginalNumber เข้ามา เมื่อถูกเปลี่ยนค่าโดยการคูณ 2 ส่งผลให้เกิดการเขียนค่า 10 ลงไปยังตำแหน่งหน่วยความจำเดิมนั่นเอง จึงทำให้ตัวแปร OriginalNumber เปลี่ยนค่าจาก 5 เป็น 10

เห็นได้ว่าพารามิเตอร์ชนิด Reference ทำหน้าที่ทั้งรับเข้ามา และส่งออกไปในเวลาเดียวกัน คำถามที่ตามมาก็คือ แล้วการส่งออกแบบ out กับ ref แตกต่างกันอย่างไร ตัวอย่างของ VC# 2005 มีคำตอบให้คุณ

ผู้เขียนแก้ไขโค้ดของการส่งพารามิเตอร์ชนิด out ใหม่กล่าวคือ ตัวแปร test ไม่มีการกำหนดค่าเริ่มต้น มี Type เป็น int เช่นเดิม ผลการส่งพารามิเตอร์แบบ out ค่าที่ได้คือ 5 ตามการทำงานซับรูทีน ShowValue() ถือว่าปกติ

VC# 2005

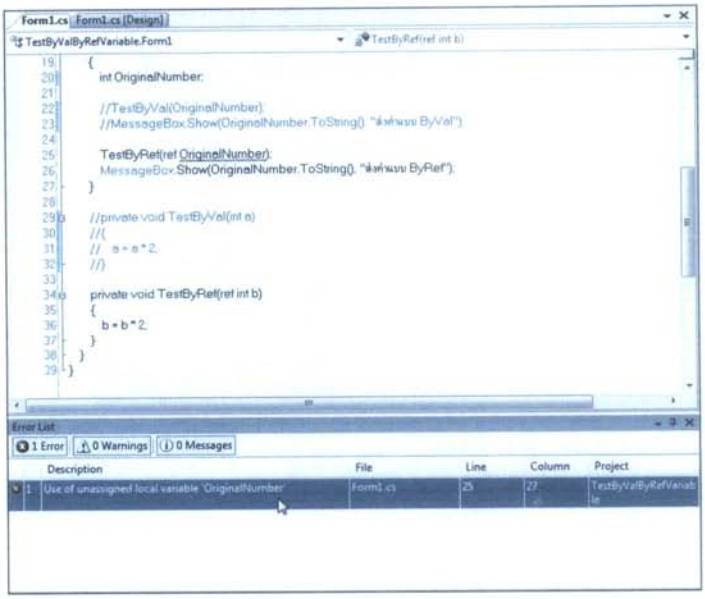
```
private void Form1_Load(object sender, EventArgs e)
{
    int test;
    ShowValue(out test);
    MessageBox.Show(test.ToString(), "ค่าปัจจุบัน");
}

public void ShowValue(out int x)
{
    x = 5;
}
```

ส่วนการส่งพารามิเตอร์แบบ Reference ผู้เขียนแก้โค้ดใหม่เช่นกัน โดยที่ไม่กำหนดค่าเริ่มต้นให้กับตัวแปร OriginalNumber ผลที่ได้คือ โค้ดชุดนี้กลับรันไม่ผ่าน ดังรูปที่ 5-8

VC# 2005

```
private void Form1_Load(object sender, EventArgs e)
{
    int OriginalNumber;
    TestByRef(ref OriginalNumber);
    MessageBox.Show(OriginalNumber.ToString(), "ส่งค่าแบบ ByRef");
}

private void TestByRef(ref int b)
{
    b = b * 2;
}
```

รูปที่ 5-8 กรณีการส่งพารามิเตอร์แบบ Reference



เหตุผลที่การส่งพารามิเตอร์แบบ Reference ดังกล่าวรันไม่ผ่าน เพราะว่าการส่งพารามิเตอร์ชนิด out คือ การส่งออกเพียงอย่างเดียวเท่านั้น แต่การส่งพารามิเตอร์แบบ Reference คือ ต้องรับเข้ามาและส่งออกไป

คุณไม่ได้กำหนดค่าเริ่มต้นให้กับตัวแปร OriginalNumber จึงผิดเงื่อนไขตั้งแต่รับเข้ามาแล้ว เพราะต้องเกิดเหตุก่อน (รับค่าเข้ามาก่อน) ผลจึงตามมา (ส่งออกได้)

ผู้เขียนเคยกล่าวไว้ว่าการสร้างฟิลด์ในคลาส ควรกำหนดค่าเริ่มต้นไว้ ตัวอย่างนี้พิสูจน์ให้เห็นอีกอย่างหนึ่งว่า ตัวแปรที่คุณนำมาใช้ก็ควรกำหนดค่าเริ่มต้นไว้ด้วยเช่นกัน

การกำหนดลักษณะเพิ่มเติมของพารามิเตอร์

เนื้อหาที่ผ่านมาเราทราบแล้วว่าพารามิเตอร์มี 3 รูปแบบ คุณสามารถกำหนดลักษณะเพิ่มเติมเพื่อการส่งค่าให้กับพารามิเตอร์แต่ละตัว ซึ่งมีวิธีการรับค่าหลากหลายยิ่งขึ้นในข้างต้นนี้ที่น่าสนใจมีอยู่ 3 แบบคือ

1. การใช้งานพารามิเตอร์แบบ Optional
2. การส่งค่าพารามิเตอร์แบบระบุชื่อ
3. การส่งพารามิเตอร์แบบ Params

การใช้งานพารามิเตอร์แบบ Optional

พารามิเตอร์ที่ถูกกำหนดให้เป็น Optional หมายความว่า พารามิเตอร์ตัวนี้ที่จุดเรียกใช้จะส่งค่าเข้ามาหรือไม่ส่งก็ได้ พารามิเตอร์ที่ถูกกำหนดให้เป็นแบบ Optional ต้องกำหนดค่าเริ่มต้นไว้ด้วย

สำหรับการใช้พารามิเตอร์แบบนี้ มีความแตกต่างกันระหว่าง VB 2005 และ VC# 2005 ให้ดูตัวอย่างที่ 5-3 การใช้งานพารามิเตอร์แบบ Optional เพิ่มเติม แยกอธิบายทีละภาษา

โค้ด VB 2005 ที่ 5-3 การใช้งานพารามิเตอร์แบบ Optional (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As Double = CalculateVAT(1000)
        'Dim result As Double = CalculateVAT(1000, 7.0)
        MessageBox.Show(result.ToString(), "ผลการคำนวณ")
    End Sub

    Private Function CalculateVAT(ByVal ProductPrice As Double, Optional ByVal VATRate As Double = 0.0) As Double
        Dim CurrentVAT As Double = 0.0
        CurrentVAT = ProductPrice * (VATRate / 100)

        Return ProductPrice + CurrentVAT
    End Function
End Class
```

ในกรณี VB 2005 โค้ดข้างต้นผู้เขียนสร้างฟังก์ชัน CalculateVAT() ขึ้นมา ทำหน้าที่คำนวณราคาสินค้ารวมภาษี ต้องการพารามิเตอร์ 2 ตัว โดยที่

- พารามิเตอร์ ProductPrice มี Type เป็น Double หมายถึง ราคาสินค้า
- พารามิเตอร์ VATRate มี Type เป็น Double เช่นกัน หมายถึง อัตราภาษี แต่กำหนดให้เป็นแบบ Optional เพราะว่าสินค้าบางตัวอาจจะไม่มีการคิดภาษีก็ได้ โดยกำหนดอัตราภาษีเริ่มต้นคือ 0.0%

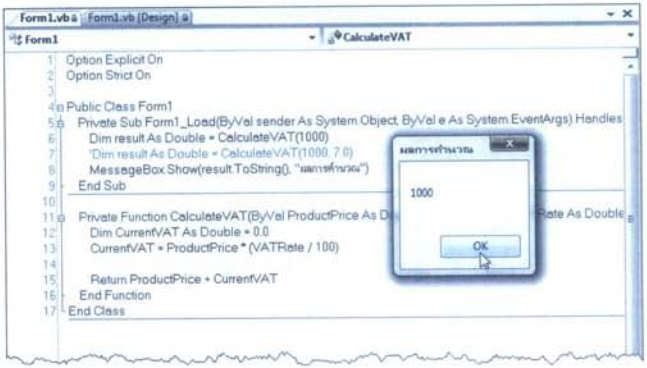
ที่จุดเรียกใช้ฟังก์ชัน CalculateVAT() อยู่ในเหตุการณ์ Form1_Load() คุณสามารถส่งค่าให้กับฟังก์ชันนี้ได้ 2 แบบคือ

1. ส่งราคาสินค้าเข้ามาเพียง 1 ตัวให้กับพารามิเตอร์ ProductPrice
2. ส่งราคาสินค้าให้กับพารามิเตอร์ ProductPrice และส่งอัตราภาษีให้กับพารามิเตอร์ VATRate ด้วย

VB 2005

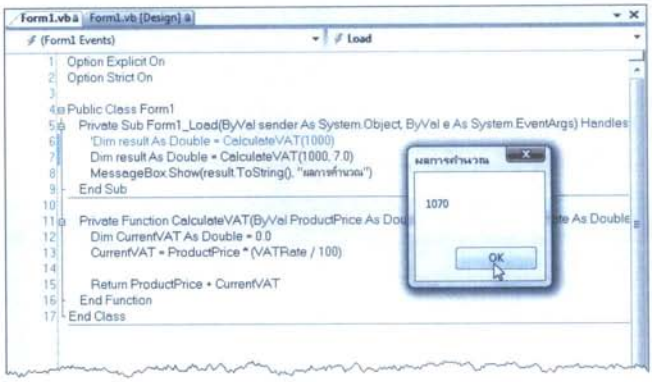
```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim result As Double = CalculateVAT(1000)
    'Dim result As Double = CalculateVAT(1000, 7.0)
    MessageBox.Show(result.ToString(), "ผลการคำนวณ")
End Sub
```

รูปที่ 5-9 กรณีส่งพารามิเตอร์ 1 ตัว



จากรูปที่ 5-9 เป็นการส่งพารามิเตอร์ 1 ตัว ผลที่ได้คือ ไม่มีอัตราภาษีเพราะว่าพารามิเตอร์ VATRate กำหนดค่าเริ่มต้นเป็น 0.0 นั่นเอง ส่วนรูป 5-10 เป็นการส่งพารามิเตอร์ 2 ตัว ก็จะคิดอัตราภาษีตามค่าที่ส่งเข้าไป

รูปที่ 5-10 กรณีส่งพารามิเตอร์ 2 ตัว



ส่วนภาษา VC# 2005 ไม่มีการส่งค่าพารามิเตอร์แบบ Optional โดยตรง แต่จะอาศัยการทำ Overload ฟังก์ชันเข้ามาช่วย ดังโค้ดต่อไปนี้

โค้ด VC# 2005 ที่ 5-3 การใช้งานพารามิเตอร์แบบ Optional (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    double result = CalculateVAT(1000);
    //double result = CalculateVAT(1000, 7.0);
    MessageBox.Show(result.ToString(), "ผลการคำนวณ");
}

private double CalculateVAT(double ProductPrice)
{
    return ProductPrice;
}

private double CalculateVAT(double ProductPrice, double VATRate)
{
    double CurrentVAT;
    CurrentVAT = ProductPrice * (VATRate / 100);

    return ProductPrice + CurrentVAT;
}
```

จากโค้ดข้างต้นเห็นได้ว่าผู้เขียนสร้างฟังก์ชัน CalculateVAT() ขึ้นมา 2 ฟังก์ชัน มี Signature แตกต่างกัน เพราะว่ามีจำนวนพารามิเตอร์ไม่เท่ากันนั่นเอง แยกออกเป็น 2 กรณี

- **กรณีที่ส่งพารามิเตอร์เข้ามา 1 ตัว** ตีความว่าไม่มีการคิดภาษี ก็จะคืนค่าราคาสินค้าออกไป ฟังก์ชันที่ถูกเลือกทำงานคือ

VC# 2005

```
private double CalculateVAT(double ProductPrice)
{
    return ProductPrice;
}
```

- **กรณีที่ส่งพารามิเตอร์เข้ามา 2 ตัว** ตีความได้ว่ามีการคิดภาษีด้วย ซึ่งก็จะคำนวณก่อนแล้วคืนค่าที่คำนวณได้ออกไป ฟังก์ชันที่ถูกเลือกให้ทำงานคือ

VC# 2005

```
private double CalculateVAT(double ProductPrice, double VATRate)
{
    double CurrentVAT;
    CurrentVAT = ProductPrice * (VATRate / 100);

    return ProductPrice + CurrentVAT;
}
```

การส่งค่าพารามิเตอร์แบบระบุชื่อ

ตามปกติแล้วค่าที่คุณส่งให้กับพารามิเตอร์แต่ละตัวต้องเรียงลำดับให้ถูกต้อง แต่ใน VB 2005 มีการส่งพารามิเตอร์แบบหนึ่งที่คุณสามารถระบุชื่อของพารามิเตอร์กับค่าที่ส่งเข้าไป โดยที่คุณไม่ต้องสนใจกับลำดับของพารามิเตอร์แต่อย่างใด (ส่วน VC# 2005 เท่าที่ผู้เขียนทราบไม่มีความสามารถแบบนี้) ให้ดูตัวอย่างที่ 5-4 การส่งค่าพารามิเตอร์แบบระบุชื่อ

โค้ด VB 2005 ที่ 5-4 การส่งค่าพารามิเตอร์แบบระบุชื่อ (Form1.vb)

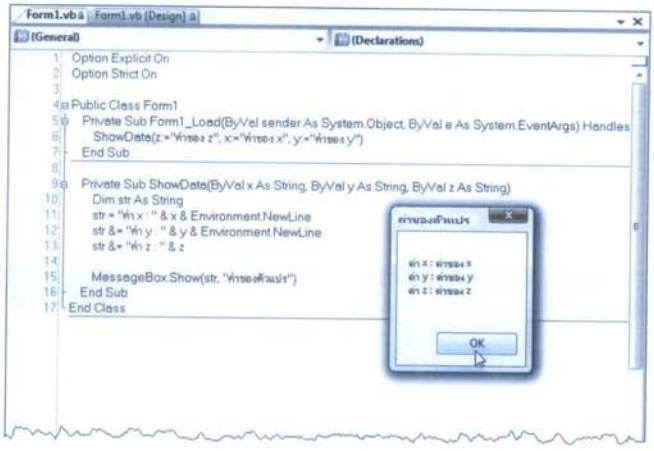
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ShowData(z:="ค่าของ z", x:="ค่าของ x", y:="ค่าของ y")
    End Sub


    Private Sub ShowData(ByVal x As String, ByVal y As String, ByVal z As String)
        Dim str As String
        str = "ค่า x : " & x & Environment.NewLine
        str &= "ค่า y : " & y & Environment.NewLine
        str &= "ค่า z : " & z

        MessageBox.Show(str, "ค่าของตัวแปร")
    End Sub
End Class
```

เห็นได้ว่าผู้เขียนสร้างซับรูทีนที่ชื่อว่า ShowData() ต้องการพารามิเตอร์ 3 ตัวคือ x, y และ z ตามลำดับ ซับรูทีนนี้ทำหน้าที่แสดงค่าของพารามิเตอร์แต่ละตัว ที่จุดเรียกซับรูทีนนี้อยู่ในเหตุการณ์ Form1_Load() ให้สังเกตวิธีการกำหนดค่าให้กับพารามิเตอร์แต่ละตัว ผู้เขียนจงใจกำหนดค่าพารามิเตอร์แต่ละตัวสลับลำดับกันแต่มีการระบุชื่อพารามิเตอร์ไว้

VB 2005

```
ShowData(z:="ค่าของ z", x:="ค่าของ x", y:="ค่าของ y")
```

รูปที่ 5-11 ผลการรันตัวอย่างที่ 5-4



จากรูปที่ 5-11 แม้ว่าจะกำหนดค่าให้กับพารามิเตอร์แต่ละตัวสลับลำดับกัน แต่เรากำหนดชื่อและค่าของพารามิเตอร์กำกับไว้แต่ละตัว ส่งผลให้การส่งค่าให้กับพารามิเตอร์ทั้ง 3 ตัว ถูกต้องตามที่เราต้องการ ส่วน VC# 2005 ยังคงต้องส่งค่าให้กับพารามิเตอร์แต่ละตัวตามลำดับที่ปรากฏขึ้นมา

โค้ด VC# 2005 ที่ 5-4 การส่งค่าพารามิเตอร์แบบระบุชื่อ (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    ShowData("ค่าของ x", "ค่าของ y", "ค่าของ z");
}

private void ShowData(string x, string y, string z)
{
    string str;
    str = "ค่า x : " + x + Environment.NewLine;
    str += "ค่า y : " + y + Environment.NewLine;
    str += "ค่า z : " + z;

    MessageBox.Show(str, "ค่าของตัวแปร");
}
```

การส่งพารามิเตอร์แบบ Params

ในบางครั้งพารามิเตอร์ที่เราต้องการนำมาใช้มีจำนวนไม่แน่นอน ให้คุณใช้คีย์เวิร์ด Params กำกับพารามิเตอร์ดังกล่าวไว้ ให้ดูตัวอย่างที่ 5-5 การส่งพารามิเตอร์แบบ Params

โค้ด VB 2005 ที่ 5-5 การส่งพารามิเตอร์แบบ Params (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim a As Integer = 1
        Dim b As Integer = 2
        Dim c As Integer = 3
        Dim x As Integer = 10
        Dim y As Integer = 20
        Dim z As Integer = 30

        Dim result As Integer = CalculateTotal(a, b, c)
        'Dim result As Integer = CalculateTotal(a, b, c, x, y, z)
        MessageBox.Show(result.ToString(), "ผลรวม")
    End Sub

    Public Function CalculateTotal(ByVal ParamArray arrInt As Integer()) As Integer
        Dim Total As Integer = 0
        Dim i As Integer = 0
        Do While (i < arrInt.Length)
            Total = (Total + arrInt(i))
            i += 1
        Loop
        Return Total
    End Function
End Class
```

โค้ด VC# 2005 ที่ 5-5 การส่งพารามิเตอร์แบบ Params (Form1.cs)

```
        private void Form1_Load(object sender, EventArgs e)
        {
            int a = 1;
            int b = 2;
            int c = 3;
            int x = 10;
            int y = 20;
            int z = 30;

            int result = CalculateTotal(a,b,c);
```

```
        //int result = CalculateTotal(a, b, c, x, y, z);
        MessageBox.Show(result.ToString(), "ผลรวม");
    }

    public int CalculateTotal(params int[] arrInt)
    {
        int Total = 0;
        for (int i=0; i < arrInt.Length; i++)
            Total += arrInt[i];
        return Total;
    }
```

วิธีดูโค้ดชุดนี้คือ ผู้เขียนสร้างฟังก์ชันชื่อว่า CalculateTotal() ทำหน้าที่หาผลรวมทั้งหมดของพารามิเตอร์ที่ชื่อว่า arrInt มี Type เป็น Integer (int) พารามิเตอร์ต้องเป็นอาร์เรย์ แล้วใช้คีย์เวิร์ด ParamArray (params) กำกับไว้

VB 2005

```
    Public Function CalculateTotal(ByVal ParamArray arrInt
As Integer()) As Integer
        Dim Total As Integer = 0
        Dim i As Integer = 0
        Do While (i < arrInt.Length)
            Total = (Total + arrInt(i))
            i += 1
        Loop
        Return Total
    End Function
```

VC# 2005

```
    public int CalculateTotal(params int[] arrInt)
    {
        int Total = 0;
        for (int i=0; i < arrInt.Length; i++)
            Total += arrInt[i];
        return Total;
    }
```

ที่น่าสนใจอยู่ที่จุดเรียกใช้ฟังก์ชันนี้ ในเหตุการณ์ Form1_Load() ผู้เขียนสร้างตัวแปรขึ้นมา 6 ตัว มี Type เป็น Integer (int) ทั้งหมด เวลาที่คุณเรียกใช้ฟังก์ชัน CalculateTotal() คุณจะส่งค่าเข้าไปกี่ตัวก็ได้ ดังรูปที่ 5-12 และ 5-13

VB 2005

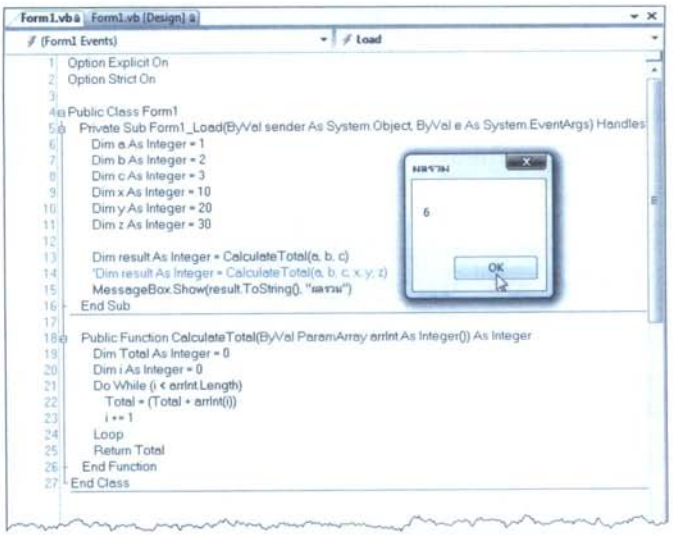
```
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim a As Integer = 1
        Dim b As Integer = 2
        Dim c As Integer = 3
        Dim x As Integer = 10
        Dim y As Integer = 20
        Dim z As Integer = 30

        Dim result As Integer = CalculateTotal(a, b, c)
        'Dim result As Integer = CalculateTotal(a, b, c, x, y, z)
        MessageBox.Show(result.ToString(), "ผลรวม")
    End Sub
```

VC# 2005

```
    private void Form1_Load(object sender, EventArgs e)
    {
        int a = 1;
        int b = 2;
        int c = 3;
        int x = 10;
        int y = 20;
        int z = 30;

        int result = CalculateTotal(a,b,c);
        //int result = CalculateTotal(a, b, c, x, y, z);
        MessageBox.Show(result.ToString(), "ผลรวม");
    }
```

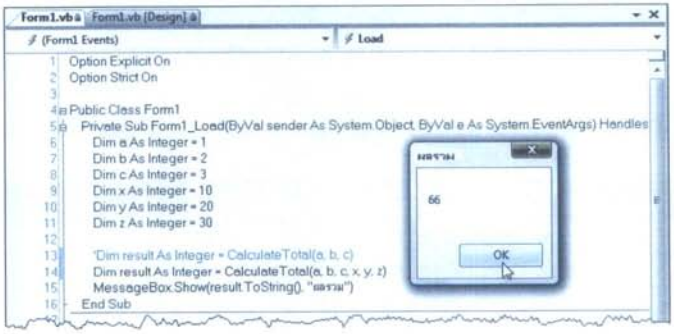
รูปที่ 5-12

ผลรวม
กรณีส่งเข้าไป
3 ตัว



รูปที่ 5-13

ผลรวม
กรณีส่งเข้าไป
6 ตัว



Type อื่นๆ ที่น่าสนใจของพารามิเตอร์

Primitive Data Type ต่างๆ (Integer, int, Long, double ฯลฯ), คลาสต่างๆ ที่อยู่ใน .NET Framework คุณสามารถนำมาใช้เป็น Type ให้กับพารามิเตอร์ของคุณได้เช่นกัน ในข้างต้นนี้นำเสนอ 4 ชนิดคือ

1. การใช้งานพารามิเตอร์แบบอาร์เรย์
2. การใช้งาน Enum ในฐานะ Type ของพารามิเตอร์
3. การใช้งานคลาสที่สร้างขึ้นมาเองในฐานะ Type ของพารามิเตอร์
4. การใช้งานคลาสที่สร้างขึ้นมาเองในฐานะ Type ของพารามิเตอร์แบบ Reference

การใช้งานพารามิเตอร์แบบอาร์เรย์

อาร์เรย์คือ ชุดของตัวแปรที่มี Type และชื่อตัวแปรเหมือนกัน ซึ่งระบุตัวแปรแต่ละตัวด้วยดัชนี (index) ให้ดูตัวอย่างที่ 5-6 การใช้งานพารามิเตอร์แบบอาร์เรย์

NOTE

รายละเอียดและการทำงานกับอาร์เรย์ ผู้เขียนจะกล่าวถึงอีกครั้งในบทที่ 13

โค้ด VB 2005 ที่ 5-6 การใช้งานพารามิเตอร์แบบอาร์เรย์ (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim arrTest() As Double = {1.0, 2.5, 3, 4.5, 5}
        Dim result As Double = CalculateArray(arrTest)

        MessageBox.Show(result.ToString(), "ผลการบวกอาร์เรย์")
    End Sub

    Private Function CalculateArray(ByVal arrNumber() As Double) As Double
        Dim i As Integer = 0
        Dim Total As Double = 0.0
        For i = 0 To arrNumber.Length - 1
            Total += arrNumber(i)
        Next

        Return Total
    End Function
End Class
```

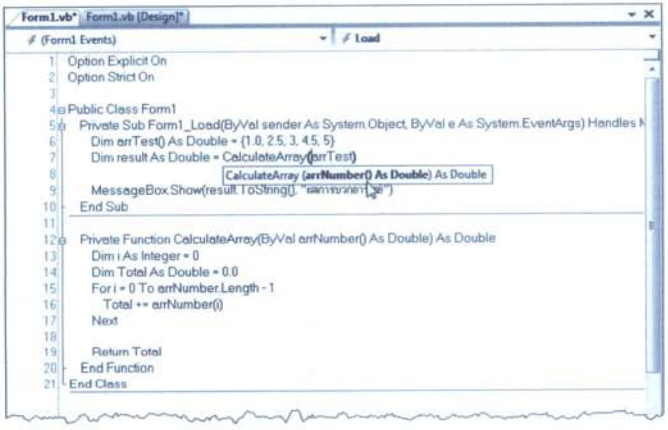
โค้ด VC# 2005 ที่ 5-6 การใช้งานพารามิเตอร์แบบอาร์เรย์ (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    double[] arrTest = new double[] { 1.0, 2.5, 3, 4.5, 5 };
    double result = CalculateArray(arrTest);

    MessageBox.Show(result.ToString(), "ผลการบวกอาร์เรย์");
}

private double CalculateArray(double[] arrNumber)
{
    double Total = 0.0;
    for (int i = 0; i <= arrNumber.Length - 1; i++)
    {
        Total += arrNumber[i];
    }

    return Total;
}
```

โค้ดข้างต้นผู้เขียนสร้างตัวแปรอาร์เรย์ที่ชื่อว่า arrTest กำหนดให้มีสมาชิก 5 ตัว มี Type เป็น Double (double)

VB 2005	VC# 2005
`Dim arrTest() As Double = {1.0, 2.5, 3, 4.5, 5}`	`double[] arrTest = new double[] { 1.0, 2.5, 3, 4.5, 5 };`

ส่วนที่ฟังก์ชัน CalculateArray() ทำหน้าที่หาผลรวมของอาร์เรย์ชนิด Double (double) กำหนดให้พารามิเตอร์ arrNumber เป็นพารามิเตอร์แบบอาร์เรย์ ใน VB 2005 ใช้เครื่องหมาย () กำกับไว้หลังชื่อตัวแปร ส่วน VC# 2005 ใช้เครื่องหมาย [] กำกับไว้หลัง Type ในกรณีนี้คือ double คุณต้องส่งค่าเป็นตัวแปรอาร์เรย์เข้าไป แล้วส่งตัวแปรธรรมดาก็ไม่ได้ ในกรณีนี้คือ ตัวแปรอาร์เรย์ arrTest ดังรูปที่ 5-14



รูปที่ 5-14 พารามิเตอร์แบบอาร์เรย์

สรุปท้ายบท

ถ้าจะเปรียบเทียบว่าเมธอดเป็นพี่ใหญ่ พารามิเตอร์ก็ไม่ต่างอะไรกับน้องฝาแฝดที่ต้องทำงานและอยู่ร่วมกันเป็นหนึ่งเดียว เห็นได้ว่ามีฟีเจอร์หลายอย่างที่ช่วยอำนวยความสะดวกให้การสร้างเมธอดมีความยืดหยุ่นขึ้นพอสมควร

CHAPTER 6

คลาสและเมมเบอร์แบบ Shared (static)

บทนำ

เนื้อหาของบทนี้จะกล่าวถึงเมมเบอร์อีกประเภทที่เรียกว่า เมมเบอร์แบบ Shared (static) คุณจะได้ศึกษาถึงข้อแตกต่างระหว่างเมมเบอร์แบบ Dynamic กับเมมเบอร์แบบ Shared (static) และกติกาการอยู่ร่วมกับระหว่างเมมเบอร์ทั้ง 2 ประเภท

ผู้เขียนขอแนะนำให้คุณผู้อ่านจับประเด็นของข้อแตกต่างเป็นสำคัญ เพราะว่าจะทำให้คุณผู้อ่านรู้จักและเข้าใจวิธีการใช้งานคลาสที่คุณสร้างขึ้นมาเอง และคลาสอื่นๆ ที่มากับ .NET Framework ได้อย่างชัดเจนยิ่งขึ้น

การสร้างคลาสแบบ Shared (VB 2005) หรือแบบ static (VC# 2005)

คลาสแบบ Shared (VB 2005) หรือแบบ static (VC# 2005) เป็นคลาสที่แตกต่างไปจากคลาสที่ผู้เขียนกล่าวมาตั้งแต่ต้น กล่าวได้อีกนัยหนึ่งคือ คลาสที่คุณได้ศึกษามาทั้งหมดเรียกว่า Dynamic Class ข้อแตกต่างระหว่าง Dynamic Class กับ Shared Class (Static Class) อยู่ตรงที่วิธีการเรียกใช้งาน และมีสิ่งที่เกี่ยวข้องอยู่ 2 อย่างคือ

- คำว่าเมธอดของอินสแตนซ์กับเมธอดของคลาส
- การใช้งานคำสั่ง Me (VB 2005) หรือ this (VC# 2005)

เราสร้างคลาสต้นแบบขึ้นมา เมื่อต้องการสร้างออบเจ็กต์จากคลาสดังกล่าว เราสั่งให้ออบเจ็กต์เกิดขึ้นมาด้วยคำสั่ง New (new) เรียกออบเจ็กต์ที่เกิดขึ้นมาว่าเป็นอินสแตนซ์ (Instance) ของคลาสนั้นๆ ส่งผลให้เราสามารถเรียกใช้สมาชิกแบบ Dynamic ของคลาสดังกล่าวผ่านทางอินสแตนซ์ที่สร้างขึ้นมา

VB 2005

```
Public Class A
    Public Sub ShowData()

    End Sub
End Class
```

VC# 2005

```
public class A
{
    public void ShowData()
    {

    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า A ประกอบด้วย 1 เมธอด เป็นเมธอดแบบ Dynamic Method ชื่อว่าเมธอด ShowData() ถ้าเราต้องการใช้งานเมธอด ShowData() ของคลาส A เราต้องสร้างออบเจ็กต์จากคลาส A ขึ้นมาก่อน โดยใช้คำสั่ง New (new) ดังโค้ดต่อไปนี้

VB 2005

```
Dim myA As New A()
myA.ShowData()
```

VC# 2005

```
A myA = new A();
myA.ShowData();
```

โค้ดข้างต้นสร้างออบเจ็กต์ A ที่ชื่อว่า myA ขึ้นมาก่อน ก็จะทำให้คุณใช้เมธอด ShowData() ได้ เราเรียกเมธอด ShowData() ว่าเป็นเมธอดของอินสแตนซ์ (ของคลาส A)

แต่ถ้าเป็นสมาชิกแบบ Shared (static) คุณไม่ต้องสร้างอินสแตนซ์ขึ้นมา คุณสามารถใช้สมาชิกดังกล่าวจากคลาสได้โดยตรง ให้ดูตัวอย่างที่ 6-1 การใช้งาน Shared (static) Member ในคลาสเพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 6-1 การใช้งาน Shared (static) Member ในคลาส

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Area
    Private Shared _x As Double = 0.0
    Private Shared _y As Double = 0.0

    Public Shared Property x() As Double
        Get
            Return _x
        End Get
        Set(ByVal value As Double)
            _x = value
```

VC# 2005 (Class1.cs)

```
public class Area
{
    private static double _x = 0.0;
    private static double _y = 0.0;

    public static double x
    {
        get{return _x;}
        set{_x = value;}
    }

    public static double y
    {
```

```
        End Set
    End Property

    Public Shared Property y() As Double
        Get
            Return _y
        End Get
        Set(ByVal value As Double)
            _y = value
        End Set
    End Property

    Public Shared Function Calculate(ByVal x As Double,
ByVal y As Double) As Double
        _x = x
        _y = y

        Return _x * _y
    End Function
End Class
```

```
        get{return _y;}
        set{_y = value;}
    }

    public static double Calculate(double x, double y)
    {
        _x = x;
        _y = y;

        return _x * _y;
    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า Area ประกอบด้วยคุณสมบัติและเมธอด ดังนี้

คุณสมบัติ/เมธอด	รายละเอียด
คุณสมบัติ x	อ่านค่าหรือกำหนดค่าของฟิลด์ _x
คุณสมบัติ y	อ่านค่าหรือกำหนดค่าของฟิลด์ _y
เมธอด Calculate()	คำนวณหาพื้นที่ โดยใช้สูตร _x*_y

เห็นได้ว่า member function (คุณสมบัติ x, y และเมธอด Calculate()) ของคลาส Area กำหนดให้เป็นแบบ Shared (static) ทั้งหมด ส่วนการใช้งานคลาส Area อยู่ใน Form1

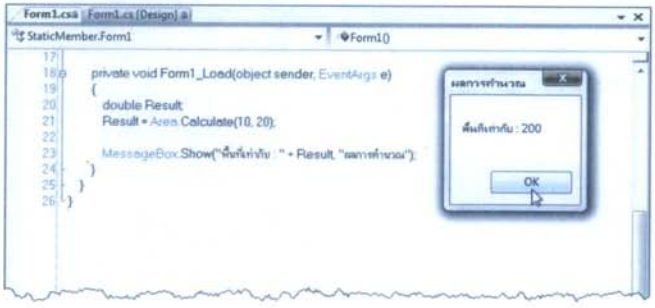
โค้ด VB 2005 ที่ 6-1 การใช้งาน Shared (static) Member ในคลาส (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim Result As Double
        Result = Area.Calculate(10, 20)

        MessageBox.Show("พื้นที่เท่ากับ : " & Result, "ผลการคำนวณ")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 6-1 การใช้งาน Shared (static) Member ในคลาส (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    double Result;
    Result = Area.Calculate(10, 20);

    MessageBox.Show("พื้นที่เท่ากับ : " + Result, "ผลการคำนวณ");
}
```

รูปที่ 6-1 ผลการทำงานของเมธอด Calculate() ของคลาส Area



จากรูปที่ 6-1 เห็นได้ว่าคุณสามารถเรียกใช้เมธอด Calculate() จากคลาส Area ได้เลย เพราะว่าเป็นเมธอดของคลาส Area ไม่ใช่เมธอดของอินสแตนซ์ของคลาส Area (ไม่ต้องสร้างอินสแตนซ์ของคลาส Area ด้วยคำสั่ง New (new))

อีกสิ่งหนึ่งที่เกี่ยวข้องก็คือ การใช้คำสั่ง Me (VB 2005) หรือ this (VC# 2005) ให้คุณดูโค้ดต่อไปนี้

VB 2005

```
Public Shared Function Calculate(ByVal x As Double, ByVal y As Double) As Double
    Me._x = x
    Me._y = y

    Return _x * _y
End Function
```

VC# 2005

```
public static double Calculate(double x, double y)
{
    this._x = x;
    this._y = y;

    return _x * _y;
}
```

โดยปกติแล้วถ้าเป็นสมาชิกแบบ Dynamic คุณสามารถใช้คำสั่ง Me (VB 2005) หรือคำสั่ง this (VC# 2005) ทำหน้าที่แทนชื่อคลาสปัจจุบันกำกับไว้หน้าสมาชิกได้เลย

แต่ถ้าเป็นสมาชิกแบบ Shared (static) คุณไม่สามารถใช้คำสั่ง Me หรือ this ดังโค้ดข้างต้นได้ เพราะว่าคำสั่ง Me (VB 2005) หรือ this (VC# 2005) ใช้เรียกแทนอินสแตนซ์เท่านั้น ให้แก้ไขโค้ดใหม่ ดังนี้

VB 2005

```
Public Shared Function Calculate(ByVal x As Double, ByVal y As Double) As Double
    Area._x = x
    Area._y = y

    Return _x * _y
End Function
```

VC# 2005

```
public static double Calculate(double x, double y)
{
    Area._x = x;
    Area._y = y;

    return _x * _y;
}
```

ในกรณีนี้เมธอด Calculate() เป็นสมาชิกของคลาส Area แบบ Shared (static) ถ้าต้องการระบุชื่อคลาสต้องใช้ชื่อเต็มของคลาสนั่นคือ Area นั่นเอง

ผลสรุปที่ได้ก็คือ ถ้าเป็นสมาชิกแบบ Dynamic คุณสามารถใช้คำสั่ง Me (VB 2005) หรือ this (VC# 2005) แทนชื่อคลาส แต่ถ้าเป็นสมาชิกแบบ Shared (static) ต้องใช้ชื่อเต็มของคลาสดังกล่าว

NOTE

เราคุ้นเคยการใช้คำสั่ง Me (this) มากที่สุดก็คือ ใช้ใน Form ที่อยู่ในโปรเจ็กต์ชนิด Windows Application นั่นเอง Form1, Form2,... Form n ถือเป็นคลาสประเภท Dynamic เช่นกัน สืบทอดมาจากคลาสต้นแบบ Form อยู่ในเนมสเปซ System.Windows.Forms

ถ้าคุณใช้คำสั่ง Me (this) ในคลาส Form1 ก็จะทำให้คำสั่ง Me (this) หมายถึงชื่อคลาส Form1 นั่นเอง แต่ถ้าคุณใช้คำสั่ง Me (this) ในคลาสประเภท Dynamic อื่นๆ คำสั่ง Me (this) ดังกล่าวก็จะหมายถึงชื่อคลาสปัจจุบันที่คำสั่ง Me (this) อาศัยอยู่

ทำความรู้จักกับฟิลด์แบบ Const

จากตัวอย่างที่ผ่านมาเราทราบแล้วว่า ถ้าเราใช้คำสั่ง Shared (static) กำกับไว้ที่ส่วนประกอบใดก็ตามของคลาส ก็จะทำให้ส่วนประกอบดังกล่าวเป็นของคลาสไม่ใช่ของอินสแตนซ์ ส่งผลให้การเรียกใช้ส่วนประกอบของคลาสมี 2 แบบคือ

1. แบบ Dynamic เรียกใช้โดยการสร้างอินสแตนซ์ขึ้นมาด้วยคำสั่ง New (new)
2. แบบ Shared (static) เรียกใช้จากคลาสโดยตรง

แต่มีสมาชิกของคลาสอยู่ 1 ชนิดที่ไม่ต้องใช้คำสั่ง Shared (static) กำกับไว้ แต่เป็นสมาชิกแบบ Shared (static) นั่นคือ ฟิลด์ชนิด Const หรือฟิลด์ที่มีค่าคงที่ (Constant) ให้ดูตัวอย่างที่ 6-2 ทำความรู้จักกับฟิลด์แบบ Const

โค้ด VB 2005 และ VC# 2005 ที่ 6-2 ทำความรู้จักกับฟิลด์แบบ Const

VB 2005 (Product.vb)

```
Option Explicit On
Option Strict On

Public Class Product
    Private _ProductName As String
    Private _Price As Double
    Public Const _VatRate As Integer = 10

    Public Property ProductName() As String
        Get
            Return _ProductName
        End Get
        Set(ByVal value As String)
            _ProductName = value
        End Set
    End Property

    Public Property Price() As Double
        Get
            Return _Price
        End Get
        Set(ByVal value As Double)
            _Price = value
        End Set
    End Property
End Class
```

VC# 2005 (Product.cs)

```
public class Product
{
    private string _ProductName;
    private double _Price;
    public const int _VatRate = 10;

    public string ProductName
    {
        get{return _ProductName;}
        set{_ProductName = value;}
    }

    public double Price
    {
        get{return _Price;}
        set{_Price = value;}
    }
}
```

คลาส Product ประกอบด้วย

คุณสมบัติ/เมธอด	รายละเอียด
คุณสมบัติ ProductName	อ่านค่าหรือกำหนดชื่อสินค้าเก็บไว้ที่ฟิลด์ _ProductName
คุณสมบัติ Price	อ่านค่าหรือกำหนดราคาสินค้าเก็บไว้ที่ฟิลด์ _Price

จะเห็นได้ว่าส่วนประกอบของคลาส Product เป็นแบบ Dynamic ทั้งหมด เพราะไม่มีการใช้คำสั่ง Shared (static) กำกับไว้แต่อย่างใด ให้ฟิลด์ _VatRate ทำหน้าที่เก็บอัตราภาษี กำหนดค่าเริ่มต้นเป็น 10% เป็นฟิลด์แบบค่าคงที่ และกำหนดขอบเขตเป็นแบบ Public(public) เพราะต้องการทดสอบฟิลด์ชนิดนี้ จึงอนุญาตให้ภายนอกมองเห็นได้ ส่วนการใช้งานคลาส Prouduct อยู่ใน Form1

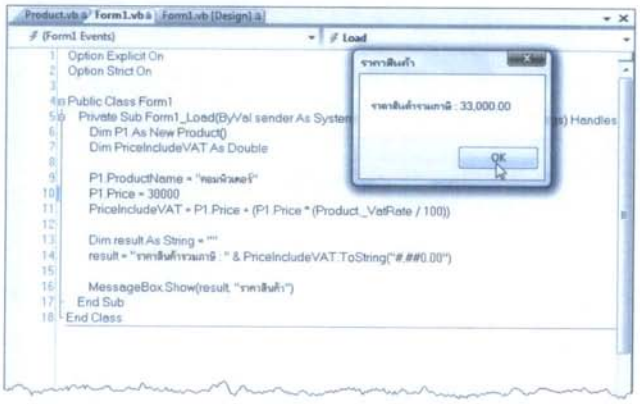
โค้ด VB 2005 ที่ 6-2 ทำความรู้จักกับฟิลด์แบบ Const (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim P1 As New Product()
        Dim PriceIncludeVAT As Double

        P1.ProductName = "คอมพิวเตอร์"
        P1.Price = 30000
        PriceIncludeVAT = P1.Price + (P1.Price * (Product._VatRate / 100))

        Dim result As String = ""
        result = "ราคาสินค้ารวมภาษี : " & PriceIncludeVAT.ToString("#,##0.00")

        MessageBox.Show(result, "ราคาสินค้า")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 6-2 ทำความรู้จักกับฟิลด์แบบ Const (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Product P1 = new Product();
        double PriceIncludeVAT;

        P1.ProductName = "คอมพิวเตอร์";
        P1.Price = 30000;
        PriceIncludeVAT = P1.Price + (P1.Price * ((double)Product._VatRate / 100));

        string result = "";
        result = "ราคาสินค้ารวมภาษี : " + PriceIncludeVAT.ToString("#,##0.00");

        MessageBox.Show(result, "ราคาสินค้า");
    }
```

รูปที่ 6-2
ผลการทำงาน
ตัวอย่างที่ 6-2



วิธีการดูโค้ดชุดนี้ก็คือ ผู้เขียนสร้างออบเจ็กต์ Product ที่ชื่อว่า P1 ขึ้นมา (P1 คืออินสแตนซ์ของคลาส Product)

VB 2005	VC# 2005
Dim P1 As New Product()	Product P1 = new Product();

คุณสมบัติ ProductName และคุณสมบัติ Price เป็นแบบ Dynamic เรียกผ่านอินสแตนซ์ P1 ตามปกติ

VB 2005	VC# 2005
P1.ProductName = "คอมพิวเตอร์" P1.Price = 30000	P1.ProductName = "คอมพิวเตอร์"; P1.Price = 30000;

แต่เมื่อต้องการคำนวณหาราคาสินค้ารวมภาษี ซึ่งต้องใช้อัตราภาษีที่เก็บอยู่ในฟิลด์ _VatRate ด้วย เห็นได้ว่าคุณไม่สามารถเรียกใช้ฟิลด์ _VatRate ด้วยโค้ดดังต่อไปนี้

VB 2005	VC# 2005
P1._VatRate	P1._VatRate;

การเรียกใช้งานฟิลด์ _VatRate คุณไม่สามารถเรียกใช้ผ่านทางอินสแตนซ์ P1 ได้ คุณต้องเรียกใช้ผ่านทางคลาส Product เท่านั้น เห็นได้ว่าฟิลด์ _VatRate ไม่มีคำสั่ง Shared (static) กำกับไว้แต่อย่างใด แต่ถูกจัดให้เป็นสมาชิกแบบ Shared (static)

VB 2005
PriceIncludeVAT = P1.Price + (P1.Price * (Product._VatRate / 100))

VC# 2005

```
PriceIncludeVAT = P1.Price + (P1.Price * ((double)Product._VatRate / 100));
```

NOTE

โค้ดของ VC# 2005 ตัวแปร PriceIncludeVAT มีความละเอียดระดับทศนิยม double แต่อัตราภาษีที่เก็บอยู่ในฟิลด์ _VatRate มี Type เป็น int จึงต้อง Casting ให้ผลการคำนวณมี Type เป็น double ก่อนนั่นเอง

ตัวอย่างคลาส Product นี้กำหนดให้ฟิลด์ _VatRate มีขอบเขตแบบ Public เพื่อต้องการทดสอบลักษณะของฟิลด์แบบ Const เท่านั้น แต่ในกรณีที่คุณกำหนดให้ฟิลด์ _VatRate มีขอบเขตแบบ Private ส่งผลให้ภายนอกมองไม่เห็นฟิลด์ _VatRate

คุณจึงต้องสร้างคุณสมบัติ VatRate ขึ้นมา และต้องกำหนดให้อ่านค่าได้เพียงอย่างเดียวเท่านั้น เพราะว่าฟิลด์ชนิด Const เป็นฟิลด์ที่มีค่าคงที่ ไม่สามารถกำหนดค่าใหม่ได้นั่นเอง ดังโค้ดต่อไปนี้

VB 2005

```
Public ReadOnly Property VatRate() As Integer
    Get
        Return _VatRate
    End Get
End Property
```

VC# 2005

```
public int VatRate
{
    get { return _VatRate; }
}
```

ในกรณีที่คุณสร้างคุณสมบัติ VatRate ขึ้นมา ถือว่าคุณสมบัติ VatRate เป็นส่วนประกอบแบบ Dynamic ก็จะอ่านค่าอัตราภาษีผ่านทางอินสแตนซ์ P1 ได้ตามปกติ

การใช้ฟิลด์แบบ Public ในฐานะเป็นตัวแปรแบบ Public

เทคนิคอย่างหนึ่งของการใช้ฟิลด์ชนิด Shared (static) ก็คือ ใช้ในฐานะเป็นตัวแปรที่สามารถเรียกใช้จุดใดก็ได้ของโปรเจ็กต์ ผู้เขียนขอแนะนำให้คุณใช้ตัวแปรลักษณะนี้ให้น้อยที่สุด ใช้เท่าที่จำเป็นเท่านั้น

เหตุผลก็คือ ตัวแปรที่อ่านค่าหรือกำหนดค่าได้อย่างอิสระแบบนี้ สะดวกต่อการใช้งานก็จริง แต่คุณไม่ได้มาฟรีสิ่งที่เสียไปก็คือ เมื่อสามารถกำหนดค่าได้อย่างอิสระจากจุดใดก็ได้ในโปรเจ็กต์ ก็กลายเป็นผลเสียที่คุณควบคุมค่าที่มันเก็บอยู่ไม่ได้นั่นเอง ให้ดูตัวอย่างที่ 6-3 การใช้ฟิลด์แบบ Public ในฐานะเป็นตัวแปรแบบ Public

โค้ด VB 2005 และ VC# 2005 ที่ 6-3 การใช้ฟิลด์แบบ Public ในฐานะเป็นตัวแปรแบบ Public

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class CommonData
    Public Shared _ID As String = "0001"
    Public Shared _Name As String = "ศุภชัย"
End Class
```

VC# 2005 (Class1.cs)

```
public class CommonData
{
    public static string _ID = "0001";
    public static string _Name = "ศุภชัย";
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า CommonData ขึ้นมา ประกอบด้วยฟิลด์ _ID และ _Name ตามลำดับ กำหนดค่าเริ่มต้นไว้ด้วย ซึ่งเป็นส่วนประกอบแบบ Shared (static) มีขอบเขตแบบ Public ส่งผลให้ภายนอกมองเห็นทั้ง 2 ฟิลด์ การใช้งานคลาส CommonData อยู่ใน Form1 และ Form2

โค้ด VB 2005 และ VC# 2005 ที่ 6-3 การใช้ฟิลด์แบบ Public ในฐานะเป็นตัวแปรแบบ Public

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles MyBase.
Load
        lblID.Text = CommonData._ID
        lblFullName.Text = CommonData._Name
    End Sub

    Private Sub cmdForm2Show_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
cmdForm2Show.Click
        Dim frm As New Form2()
        frm.Show()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        lblID.Text = CommonData._ID;
        lblFullName.Text = CommonData._Name;
    }

    private void cmdForm2Show_Click(object sender,
EventArgs e)
    {
        Form2 frm = new Form2();
        frm.Show();
    }
```

โค้ด VB 2005 และ VC# 2005 ที่ 6-3 การใช้ฟิลด์แบบ Public ในฐานะเป็นตัวแปรแบบ Public

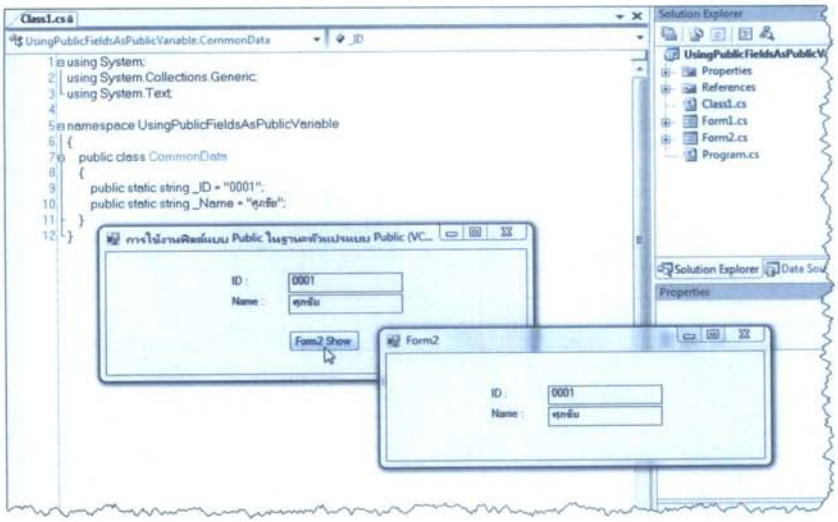
VB 2005 (Form2.vb)

```
Option Explicit On
Option Strict On

Public Class Form2
    Private Sub Form2_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        lblID.Text = CommonData._ID
        lblFullName.Text = CommonData._Name
    End Sub
End Class
```

VC# 2005 (Form2.cs)

```
    private void Form2_Load(object sender, EventArgs e)
    {
        lblID.Text = CommonData._ID;
        lblFullName.Text = CommonData._Name;
    }
```

รูปที่ 6-3 ผลการรันตัวอย่างที่ 6-3



เห็นได้ว่าคุณสามารถใช้ฟิลด์ _ID และฟิลด์ _Name ใน Form1, Form2 หรือฟอร์มใดๆ ก็ได้ที่อยู่ในโปรเจ็กต์ของคุณ กล่าวได้อีกนัยหนึ่งก็คือ ฟิลด์ทั้ง 2 ทำหน้าที่เป็นตัวแปรที่มีขอบเขตการเรียกใช้งานทั้งโปรเจ็กต์นั่นเอง

การใช้งาน Shared (static) เมมเบอร์ร่วมกับ Dynamic เมมเบอร์

จากเนื้อหาที่กล่าวมา สมาชิกที่อยู่ในคลาสมีอยู่ 2 ประเภทคือ

- สมาชิกแบบ Shared (static) เรียกใช้จากคลาส
- สมาชิกแบบ Dynamic เรียกใช้จากอินสแตนซ์

คำถามที่ตามมาก็คือ ถ้าใน 1 คลาสประกอบไปด้วยสมาชิกทั้งแบบ Shared (static) และแบบ Dynamic การอยู่ร่วมกันระหว่างสมาชิกทั้ง 2 แบบเป็นอย่างไร เมื่อใดควรใช้สมาชิกแบบ Shared (static) เมื่อใดควรใช้สมาชิกแบบ Dynamic ให้ดูตัวอย่างที่ 6-4 การใช้งาน Shared (static) เมมเบอร์ร่วมกับ Dynamic เมมเบอร์

โค้ด VB 2005 และ VC# 2005 ที่ 6-4 การใช้งาน Shared (static) เมมเบอร์ร่วมกับ Dynamic เมมเบอร์

VB 2005 (Product.vb)

```
Option Explicit On
Option Strict On

Public Class Product
    Private _ProductName As String = ""
    Private _Price As Double = 0.0
    Private Shared _VATRate As Double = 10.0

    Public Property ProductName() As String
```

VC# 2005 (Product.cs)

```
public class Product
{
    private string _ProductName = "";
    private double _Price = 0.0;
    private static double _VATRate = 10.0;

    public string ProductName
    {
        get{return _ProductName;}
```

```
        Get
            Return _ProductName
        End Get
        Set(ByVal value As String)
            _ProductName = value
        End Set
    End Property

    Public Property Price() As Double
        Get
            Return _Price
        End Get
        Set(ByVal value As Double)
            _Price = value
        End Set
    End Property

    Public Property VATRate() As Double
        Get
            Return _VATRate
        End Get
        Set(ByVal value As Double)
            _VATRate = value
        End Set
    End Property

    Public Sub New(ByVal ProductName As String, ByVal
Price As Double)
        Me._ProductName = ProductName
        Me._Price = Price
    End Sub

    Public Function CalculatePriceWithVat() As Double
        Dim result As Double = 0.0

        result = _Price + (_Price * (_VATRate / 100))
        Return result
    End Function
End Class
```

```
        set{_ProductName = value;}
    }

    public double Price
    {
        get{return _Price;}
        set{_Price = value;}
    }

    public double VATRate
    {
        get{return _VATRate;}
        set{_VATRate = value;}
    }

    public Product(string ProductName, double Price)
    {
        this._ProductName = ProductName;
        this._Price = Price;
    }

    public double CalculatePriceWithVat()
    {
        double result = 0.0;

        result = _Price + (_Price * (_VATRate / 100));
        return result;
    }
}
```

ผู้เขียนแก้ไขคลาส Product ใหม่ ประกอบด้วยคุณสมบัติและเมธอด ดังนี้

คุณสมบัติ/เมธอด	รายละเอียด
คุณสมบัติ ProductName	ชื่อสินค้า
คุณสมบัติ Price	ราคาสินค้า
คุณสมบัติ VATRate	อัตราภาษีกำหนดค่าเริ่มต้น 10%
เมธอด CalculatePriceWithVat()	คำนวณราคาสินค้ารวมภาษี

เห็นได้ว่าสมาชิกในคลาส Product เป็นแบบ Dynamic เกือบทั้งหมด มีเพียงฟิลด์ _VATRate เท่านั้นที่เป็นแบบ Shared (static) กำหนดอัตราภาษีเริ่มต้น 10% มี Type เป็น Double (double)

VB 2005

```
Private Shared _VATRate As Double = 10.0
```

VC# 2005

```
private static double _VATRate = 10.0;
```

คลาส Product มีการใช้คอนสตรัคเตอร์แบบมีพารามิเตอร์ด้วย ดังโค้ดต่อไปนี้

VB 2005

```
    Public Sub New(ByVal ProductName As String, ByVal
Price As Double)
        Me._ProductName = ProductName
        Me._Price = Price
    End Sub
```

VC# 2005

```
    public Product(string ProductName, double Price)
    {
        this._ProductName = ProductName;
        this._Price = Price;
    }
```

NOTE

รายละเอียดของการใช้คอนสตรัคเตอร์ (Constructor) ผู้เขียนจะกล่าวถึงอีกครั้งในบทที่ 9

คอนสตรัคเตอร์เป็นเมธอดชนิดหนึ่งที่ทำงานโดยอัตโนมัติ เมื่อออบเจ็กต์เกิดขึ้นมาด้วยคำสั่ง New (new) ในกรณีนี้คลาส Product กำหนดให้คอนสตรัคเตอร์มีพารามิเตอร์ 2 ตัว ดังนั้น เมื่อคุณสั่งให้สร้างออบเจ็กต์ Product ขึ้นมา คุณจึงต้องส่งพารามิเตอร์เข้ามา 2 ตัวด้วยคอนสตรัคเตอร์ใน VB 2005 ชื่อว่า Public Sub New() ส่วน VC# 2005 จะมีชื่อเดียวกับคลาส ส่งผลให้คลาส Product มีคอนสตรัคเตอร์ชื่อว่า public Product() ส่วนการใช้งานคลาส Product อยู่ใน Form1

โค้ด VB 2005 ที่ 6-4 การใช้งาน Shared (static) เมมเบอร์ร่วมกับ Dynamic เมมเบอร์ (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim PC1 As New Product("คอมพิวเตอร์เครื่องที่ 1", 30000)
        Dim result1 As Double
        result1 = PC1.CalculatePriceWithVat()
        MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " & result1, "ภาษี 10%")

        Dim PC2 As New Product("คอมพิวเตอร์เครื่องที่ 2", 40000)
        Dim result2 As Double
```

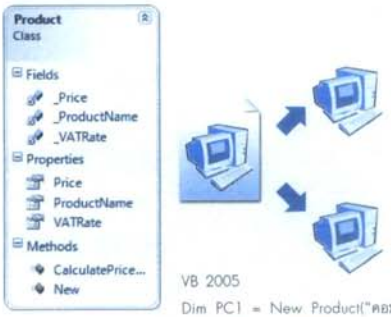
```
        result2 = PC2.CalculatePriceWithVat()
        MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " & result2, "ภาษี 10%")

        PC1.VATRate = 5
        result1 = PC1.CalculatePriceWithVat()
        MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " & result1, "ภาษี 5%")

        result2 = PC2.CalculatePriceWithVat()
        MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " & result2, "ภาษี 5%")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 6-4 การใช้งาน Shared (static) เมมเบอร์ร่วมกับ Dynamic เมมเบอร์ (Form1.cs)

```
        private void Form1_Load(object sender, EventArgs e)
        {
            Product PC1 = new Product("คอมพิวเตอร์เครื่องที่ 1", 30000);
            double result1;
            result1 = PC1.CalculatePriceWithVat();
            MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " + result1, "ภาษี 10%");

            Product PC2 = new Product("คอมพิวเตอร์เครื่องที่ 2", 40000);
            double result2;
            result2 = PC2.CalculatePriceWithVat();
            MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " + result2, "ภาษี 10%");

            PC1.VATRate = 5;
            result1 = PC1.CalculatePriceWithVat();
            MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " + result1, "ภาษี 5%");

            result2 = PC2.CalculatePriceWithVat();
            MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " + result2, "ภาษี 5%");
        }
```

วิธีการดูโค้ดชุดนี้ก็คือ ผู้เขียนสร้างออบเจ็กต์ Product ขึ้นมา 2 อินสแตนซ์ ชื่อว่า PC1 และ PC2 ตามลำดับ กล่าวได้อีกนัยหนึ่งคือ ออบเจ็กต์ PC1 และออบเจ็กต์ PC2 เป็นอินสแตนซ์ของคลาสต้นแบบ Product โดยการคำนวณราคารวมภาษีอยู่ในความรับผิดชอบของเมธอด CalculatePriceWithVAT() ดังรูปที่ 6-4

รูปที่ 6-4
การเปรียบเทียบโค้ดตัวอย่างที่ 6-4



VB 2005
```
Dim PC2 As New Product("คอมพิวเตอร์เครื่องที่ 2", 40000)
Dim result2 As Double
result2 = PC2.CalculatePriceWithVat()
```

VC# 2005
```
Product PC2 = new Product("คอมพิวเตอร์เครื่องที่ 2", 40000)
Double result2;
result2 = PC.CalculatePriceWithVat();
```

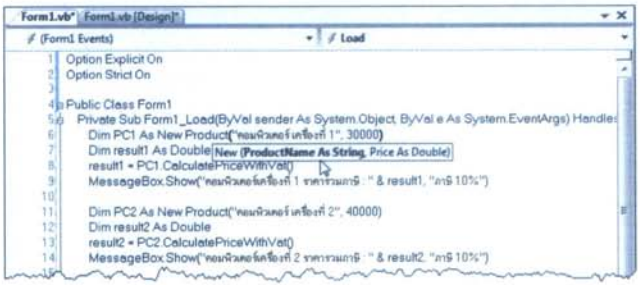
VB 2005
```
Dim PC1 = New Product("คอมพิวเตอร์เครื่องที่ 1", 30000)
Dim result1 As Double
result1 = PC1.CalculatePriceWithVat()
```

VC# 2005
```
Product PC1 = new Product("คอมพิวเตอร์เครื่องที่ 1", 30000)
Double result1;
result1 = PC1.CalculatePriceWithVat();
```

เนื่องจากคลาส Product กำหนดให้คอนสตรัคเตอร์ต้องส่งพารามิเตอร์เข้ามา 2 ตัวด้วยคือ

- พารามิเตอร์ตัวที่ 1 คือ ชื่อสินค้า
- พารามิเตอร์ตัวที่ 2 คือ ราคาสินค้า

รูปที่ 6-5
คอนสตรัคเตอร์ของคลาส Product



VB 2005

```
Dim PC1 As New Product("คอมพิวเตอร์เครื่องที่ 1", 30000)
Dim result1 As Double
result1 = PC1.CalculatePriceWithVat()
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " & result1, "ภาษี 10%")

Dim PC2 As New Product("คอมพิวเตอร์เครื่องที่ 2", 40000)
Dim result2 As Double
result2 = PC2.CalculatePriceWithVat()
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " & result2, "ภาษี 10%")
```

VC# 2005

```
Product PC1 = new Product("คอมพิวเตอร์เครื่องที่ 1", 30000);
double result1;
result1 = PC1.CalculatePriceWithVat();
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " + result1, "ภาษี 10%");

Product PC2 = new Product("คอมพิวเตอร์เครื่องที่ 2", 40000);
double result2;
result2 = PC2.CalculatePriceWithVat();
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " + result2, "ภาษี 10%");
```

เมธอด CalculatePriceWithVAT() เป็นเมธอดแบบ Dynamic ต้องเรียกใช้ผ่านอินสแตนซ์ เราต้องมองว่าการคำนวณราคารวมภาษีเป็นหน้าที่ของคลาส หรือเป็นหน้าที่ของอินสแตนซ์

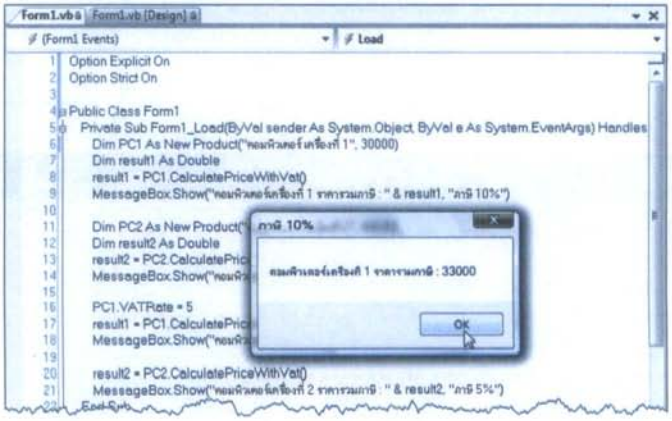
ออบเจ็กต์ PC1 มีราคาสินค้าเป็นของตัวเอง (30,000 บาท) ส่วนออบเจ็กต์ PC2 ก็มีราคาสินค้าเป็นของตัวเองเช่นกัน (40,000 บาท) ส่งผลให้การคำนวณราคารวมภาษีเป็นหน้าที่ของแต่ละอินสแตนซ์

แต่อัตราภาษี (ฟิลด์ _VATRate) ต้องใช้อัตราเดียวกันทุกอินสแตนซ์ ซึ่งถือเป็นหน้าที่ของคลาสจึงกำหนดให้ฟิลด์ _VATRate เป็นสมาชิกแบบ Shared (static) เพราะว่าสมาชิกแบบ Shared (static) เป็นของคลาสมีเพียง 1 ชุดเท่านั้น เมื่อมีการสร้างอินสแตนซ์จากคลาส Product ขึ้นมา สิ่งที่ทุกๆ อินสแตนซ์ไปด้วยคือ

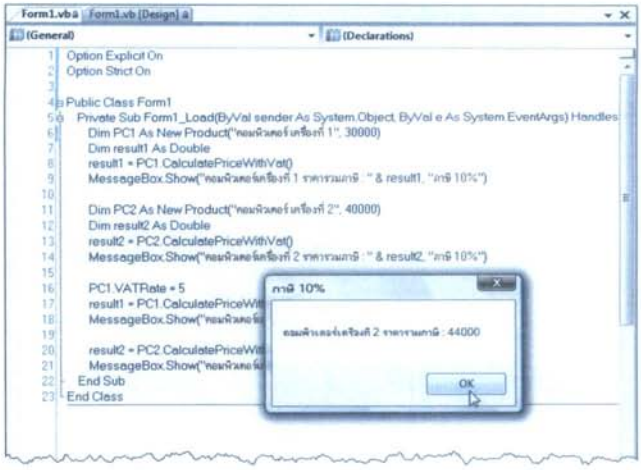
- สมาชิกที่เป็นแบบ Dynamic โดยในแต่ละอินสแตนซ์ไม่เกี่ยวข้องกันแต่อย่างใด
- สมาชิกที่เป็นแบบ Shared (static) ในแต่ละอินสแตนซ์ใช้ร่วมกัน เพราะว่าสมาชิกแบบนี้มีเพียง 1 ชุดเท่านั้นอยู่ที่คลาสต้นแบบ

อินสแตนซ์ PC1 มีชื่อและราคาสินค้าเป็นของตัวเอง ส่วนอินสแตนซ์ PC2 ก็มีชื่อและราคาสินค้าเป็นของตัวเองเช่นกัน แต่ทั้ง 2 อินสแตนซ์ต้องใช้อัตราภาษีร่วมกัน เพราะว่าฟิลด์ _VATRate เป็นแบบ Shared (static) นั่นเอง

รูปที่ 6-6
ผลการรัน
ตัวอย่างที่ 6-4



รูปที่ 6-6 (ต่อ)
ผลการรัน
ตัวอย่างที่ 6-4



จากรูปที่ 6-6 ผู้เขียนกำหนดอัตราภาษีเริ่มต้น 10% ส่งผลให้ราคารวมภาษีของออบเจ็กต์ PC1 คือ 33,000 บาท ส่วนของออบเจ็กต์ PC2 คือ 44,000 บาท

ต่อมาผู้เขียนเปลี่ยนอัตราภาษีใหม่จากอัตราเดิม 10% เป็นอัตราใหม่ 5% คุณจะทำผ่านทางอินสแตนซ์ PC1 หรือ PC2 ก็ได้ ในกรณีนี้กำหนดค่าใหม่ผ่านทางคุณสมบัติ VATRate ของอินสแตนซ์ PC1 พบว่าราคารวมภาษีของทั้งอินสแตนซ์ PC1 และอินสแตนซ์ PC2 เปลี่ยนมาใช้ 5% ทั้งคู่ ดังรูปที่ 6-7

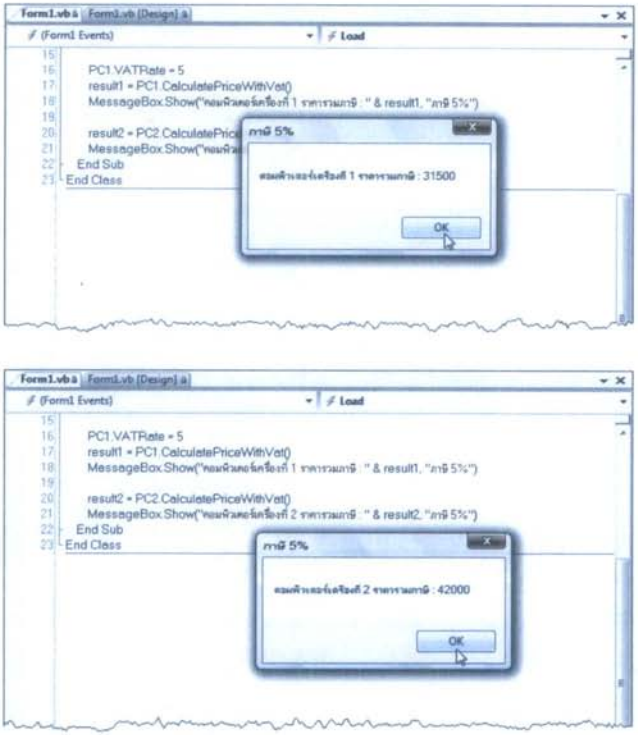
VB 2005

```
PC1.VATRate = 5
result1 = PC1.CalculatePriceWithVat()
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " & result1, "ภาษี 5%")

result2 = PC2.CalculatePriceWithVat()
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " & result2, "ภาษี 5%")
```

VC# 2005

```
PC1.VATRate = 5;
result1 = PC1.CalculatePriceWithVat();
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 1 ราคารวมภาษี : " + result1, "ภาษี 5%");

result2 = PC2.CalculatePriceWithVat();
MessageBox.Show("คอมพิวเตอร์เครื่องที่ 2 ราคารวมภาษี : " + result2, "ภาษี 5%");
```

รูปที่ 6-7
ราคาสินค้ารวมภาษีหลังจากเปลี่ยนอัตราภาษีใหม่



เหตุผลที่ราคาสินค้าเปลี่ยนมาใช้อัตราภาษีใหม่ทั้ง 2 อินสแตนซ์ เป็นเพราะว่าฟิลด์ _VATRate เป็นสมาชิกแบบ Shared (static) นั่นเอง เมื่อมีการเปลี่ยนแปลงค่าของฟิลด์ _VATRate จึงส่งผลกระทบทุกอินสแตนซ์นั่นเอง

กล่าวได้อีกนัยหนึ่งก็คือ สมาชิกแบบ Shared (static) ทำหน้าที่เปรียบเสมือนกับเป็นตัวแปรแบบ Public ที่สามารถเรียกใช้ร่วมกันได้ทุกอินสแตนซ์นั่นเอง

สิทธิการเรียกใช้งานระหว่างสมาชิกแบบ Shared (static) กับ Dynamic

นอกจากการใช้งานสมาชิกแบบ Shared (static) กับสมาชิกแบบ Dynamic จะแตกต่างกันแล้ว สิทธิของสมาชิกทั้ง 2 แบบ ก็ยังไม่เท่าเทียมกันอีกด้วย ให้ดูตัวอย่างที่ 6-5 สิทธิการเรียกใช้งานระหว่างสมาชิกแบบ Shared (static) กับ Dynamic

โค้ด VB 2005 และ VC# 2005 ที่ 6-5 สิทธิการเรียกใช้งานระหว่างสมาชิกแบบ Shared (static) กับ Dynamic

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Sample
    Private Shared _A As Integer = 0
    'Private _A As Integer = 0

    Public Shared Property A() As Integer
        Get
            Return _A
        End Get
        Set(ByVal value As Integer)
            _A = value
        End Set
    End Property

    'Public Property A() As Integer
    '    Get
    '        Return _A
    '    End Get
    '    Set(ByVal value As Integer)
    '        _A = value
    '    End Set
    'End Property
End Class
```

VC# 2005 (Class1.cs)

```
public class Sample
{
    private static int _A = 0;
    //private int _A=0;

    public static int A
    {
        get { return _A; }
        set { _A = value; }
    }


    //public int A
    //{
    // get { return _A; }
    // set { _A = value; }
    //}
}
```

พบว่าในโค้ดทำหมายเหตุไว้บางส่วน วิธีการดูโค้ดชุดนี้ให้คุณดูเป็นคู่ๆ โดยที่

- คู่ที่ 1 ผู้เขียนสร้างฟิลด์ที่ชื่อว่า _A เป็นตัวแทนสมาชิกแบบ Shared (static) ส่วนคุณสมบัติ A เป็นตัวแทนสมาชิกแบบ Shared (static) เช่นกัน โค้ดชุดนี้รันผ่านตามปกติ เพราะว่าสมาชิกแบบ Shared (static) สามารถเรียกใช้กันเองได้

VB 2005

```
    Private Shared _A As Integer = 0

    Public Shared Property A() As Integer
        Get
            Return _A
        End Get
        Set(ByVal value As Integer)
            _A = value
        End Set
    End Property
```

VC# 2005

```
    private static int _A = 0;

    public static int A
    {
        get { return _A; }
        set { _A = value; }
    }
```

- คู่ที่ 2 ผู้เขียนเปลี่ยนฟิลด์ _A เป็นแบบ Dynamic ส่วนคุณสมบัติ A ยังคงเป็นสมาชิกแบบ Shared (static) เช่นเดิม โค้ดชุดนี้รันไม่ผ่านเพราะว่าสมาชิกแบบ Shared (static) ไม่สามารถเรียกใช้สมาชิกแบบ Dynamic ได้

VB 2005

```
Private _A As Integer = 0

Public Shared Property A() As Integer
   Get
      Return _A
   End Get
   Set(ByVal value As Integer)
      _A = value
   End Set
End Property
```

VC# 2005

```
private int _A = 0;

public static int A
{
   get { return _A; }
   set { _A = value; }
}
```

- คู่ที่ 3 ฟิลด์ A เป็นแบบ Shared (static) ส่วนคุณสมบัติ A เป็นแบบ Dynamic โค้ดชุดนี้รันผ่าน เพราะว่าสมาชิกแบบ Dynamic เรียกใช้สมาชิกแบบ Shared (static) ได้

VB 2005

```
Private Shared _A As Integer = 0

Public Property A() As Integer
   Get
      Return _A
   End Get
   Set(ByVal value As Integer)
      _A = value
   End Set
End Property
```

VC# 2005

```
private static int _A = 0;

public int A
{
   get { return _A; }
   set { _A = value; }
}
```

- คู่ที่ 4 ฟิลด์ A และคุณสมบัติ A เป็นแบบ Dynamic โค้ดชุดนี้รันผ่าน เพราะว่าสมาชิก Dynamic เรียกใช้กันเองได้

VB 2005

```
Private _A As Integer = 0

Public Property A() As Integer
   Get
      Return _A
   End Get
   Set(ByVal value As Integer)
      _A = value
   End Set
End Property
```

VC# 2005

```
private int _A = 0;

public int A
{
   get { return _A; }
   set { _A = value; }
}
```

จากการทดสอบทั้ง 4 รูปแบบ สรุปเป็นตารางดังนี้

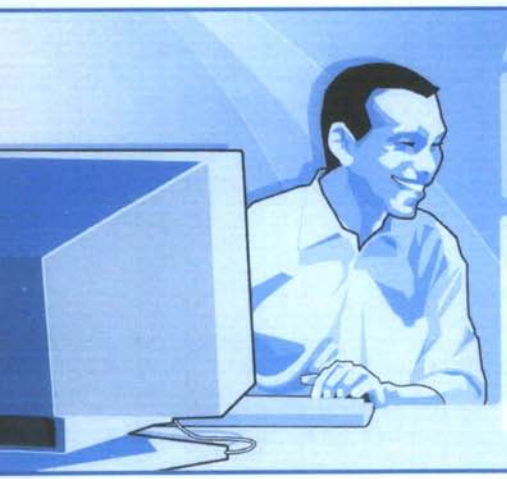
	Shared (static)	Dynamic
Shared (static)	ได้	ไม่ได้
Dynamic	ได้	ได้

จากตารางข้างต้นพบว่าสมาชิกแบบ Dynamic สามารถเรียกใช้สมาชิกได้ทั้ง 2 แบบ ส่วนสมาชิกแบบ Shared (static) สามารถเรียกใช้ได้แต่พวกเดียวกันเท่านั้น

สรุปท้ายบท

คำถามหนึ่งที่ผู้เขียนมักจะพบเจอเสมอนั่นคือ ทำไมบางครั้งการสร้างออบเจ็กต์ต้องใช้คำสั่ง New (new) แต่บางครั้งไม่จำเป็นต้องใช้ อาจจะเป็นคำถามที่เคยค้างคาใจคุณผู้อ่านหลายๆ ท่านมาก่อน เนื้อหาของบทนี้คือ คำตอบส่วนหนึ่งที่ชัดเจนพอสมควร


Programming in OOP style
Advanced .net



CHAPTER ▸ 7

Default Properties (Indexer)

บทนำ

สำหรับเนื้อหาในบทนี้ คุณจะได้ศึกษาเมมเบอร์อีกประเภทหนึ่งที่สามารถทำให้คลาสต้นแบบของคุณ มีลักษณะเหมือนกับอาร์เรย์

พื้นฐานการใช้งาน Default Properties (Indexer)

Default Properties (Indexer) คือ member function อีกชนิดหนึ่งที่สามารถเป็นสมาชิกในคลาสของคุณ ได้มีโครงสร้างคล้ายๆ กับคุณสมบัติ (Properties) กล่าวคือ มีบล็อกของ Set/Get เช่นเดียวกัน แตกต่างกันก็แต่เพียงว่า Default Properties (Indexer) ทำให้คลาสมีลักษณะคล้ายอาร์เรย์ และมีลำดับอ้างอิง ให้ดูตัวอย่างที่ 7-1 พื้นฐานการใช้งาน Default Properties (Indexer)

โค้ด VB 2005 และ VC# 2005 ที่ 7-1 พื้นฐานการใช้งาน Default Properties (Indexer)

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On
Imports System.Text

Public Class IntegerBox
    Private _arrInteger() As Integer = {100, 200, 300, 400,
500}

    Default Public Property Item(ByVal index As Integer)
As Integer
        Get
            Return _arrInteger(index)
        End Get
        Set(ByVal value As Integer)
            _arrInteger(index) = value
        End Set
    End Property
    Public Overrides Function ToString() As String
        Dim sb As New StringBuilder()
        sb.Remove(0, sb.Length)

        Dim _i As Integer
        For Each _i In _arrInteger
            sb.Append(_i & " ")
        Next
        Return sb.ToString()
    End Function
End Class
```

VC# 2005 (Class1.cs)

```
public class IntegerBox
{
    private int[] _arrInteger = new int[] { 100, 200, 300,
400, 500 };

    public int this[int index]
    {
        get{return _arrInteger[index];}
        set{_arrInteger[index] = value;}
    }

    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.Remove(0, sb.Length);
        foreach (int _i in _arrInteger)
        {
            sb.Append(_i+ " ");
        }
        return sb.ToString();
    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า IntegerBox ทำหน้าที่เป็นกล่องใส่เลขจำนวนเต็ม สร้างฟิลด์แบบอาร์เรย์ที่ชื่อว่า _arrInteger() ขึ้นมา กำหนดให้มีสมาชิก 5 ตัวคือ 100, 200, 300, 400 และ 500 แยกออกเป็น 2 กรณีคือ

- **กรณี** VB 2005 : สร้าง Default Properties ที่ชื่อว่า Item ขึ้นมา มีพารามิเตอร์ 1 ตัวชื่อว่า index มี Type เป็น Integer ทำหน้าที่บอกลำดับของคลาส

VB 2005

```
Default Public Property Item(ByVal index As Integer) As Integer
    Get
        Return _arrInteger(index)
    End Get
    Set(ByVal value As Integer)
        _arrInteger(index) = value
    End Set
End Property
```

- **กรณี** VC# 2005 : สร้างคุณสมบัติที่ชื่อว่า this ขึ้นมา มีพารามิเตอร์ 1 ตัวชื่อว่า index เช่นกัน

VC# 2005

```
public int this[int index]
{
    get{return _arrInteger[index];}
    set{_arrInteger[index] = value;}
}
```

ก็จะกำหนดให้ Default Properties (indexer) ไปอ่านค่า (Get) หรือกำหนดค่า (Set) ฟิลด์แบบอาร์เรย์ที่ชื่อว่า _arrInteger() นั่นเอง

ต่อมา Override เมธอด ToString() ของคลาส IntegerBox ใหม่ ทำหน้าที่แสดงค่าของสมาชิกแต่ละตัวที่อยู่ในฟิลด์ _arrInteger ส่วนการใช้งานคลาส IntegerBox อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 7-1 พื้นฐานการใช้งาน Default Properties (Indexer)

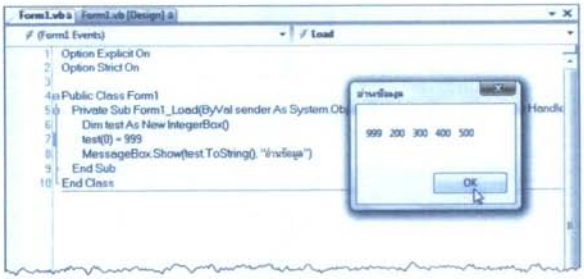
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim test As New IntegerBox()
        test(0) = 999
        MessageBox.Show(test.ToString(), "อ่านข้อมูล")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    IntegerBox test = new IntegerBox();
    test[0] = 999;
    MessageBox.Show(test.ToString(), "อ่านข้อมูล");
}
```

รูปที่ 7-1 ผลการรันตัวอย่างที่ 7-1



โค้ดข้างต้นสร้างตัวแปรออบเจ็กต์ IntegerBox ที่ชื่อว่า test ขึ้นมา การอ้างอิง Default Properties (Indexer) จะใช้วิธีการระบุลำดับไม่ใช่ระบุชื่อ

สมมติว่าต้องการเปลี่ยนค่าของสมาชิกที่อยู่ในฟิลด์ _arrInteger ตัวแรก (ลำดับอ้างอิง 0) จากค่าเดิม 100 เป็น 999 ก็จะระบุออบเจ็กต์ test ลำดับที่ 0 แสดงค่าของสมาชิกแต่ละตัวผ่านทางเมธอด ToString() ที่ได้ Override ไว้ก่อนหน้านี้นั่นเอง

การใช้งาน Default Properties (Indexer) ในรูปแบบเก็บรายการออบเจ็กต์

คุณสามารถใช้ Default Properties (Indexer) ทำหน้าที่เก็บรายการออบเจ็กต์ได้อีกด้วย ให้ดูตัวอย่างที่ 7-2 การใช้งาน Default Properties (Indexer) แบบเก็บออบเจ็กต์ไว้ในออบเจ็กต์

โค้ด VB 2005 และ VC# 2005 ที่ 7-2 การใช้งาน Default Properties (Indexer) แบบเก็บออบเจ็กต์ไว้ในออบเจ็กต์

VB 2005 (Programmer.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return _FirstName + " " + _LastName
    End Function
End Class
```

VC# 2005 (Programmer.cs)

```
public class Programmer
{
    private string _FirstName="";
    private string _LastName="";

    public string FirstName
    {
        get { return _FirstName; }
        set { _FirstName = value; }
    }

    public string LastName
    {
        get { return _LastName; }
        set { _LastName = value; }
    }

    public override string ToString()
    {
        return _FirstName + " " + _LastName;
    }
}
```

ผู้เขียนสร้างคลาส Programmer ขึ้นมา ประกอบด้วยคุณสมบัติ FirstName และคุณสมบัติ LastName มีการ Override เมธอด ToString() กำหนดให้คืนค่าเป็นชื่อ (ฟิลด์ _FirstName) และนามสกุล (ฟิลด์ _LastName)

โค้ด VB 2005 ที่ 7-2 การใช้งาน Default Properties (Indexer) แบบเก็บออบเจ็กต์ไว้ในออบเจ็กต์ (Company.vb)

```
Option Explicit On
Option Strict On
Imports System.Text

Public Class Company
    Private ProgrammerList As Programmer() = New Programmer(2) {}

    Default Public Property Item(ByVal index As Integer) As Programmer
        Get
            Return ProgrammerList(index)
        End Get
        Set(ByVal value As Programmer)
            If index < 0 OrElse index > 2 Then
                Throw New IndexOutOfRangeException("Index Out of Range !!!")
            Else
                ProgrammerList(index) = value
            End If
        End Set
    End Property

    Public Overrides Function ToString() As String
        Dim sb As New StringBuilder()
        sb.Append("Programmer List : " + Environment.NewLine)

        For Each p As Programmer In ProgrammerList
            sb.Append(Environment.NewLine + p.ToString())
        Next
        Return sb.ToString()
    End Function
End Class
```

โค้ด VC# 2005 ที่ 7-2 การใช้งาน Default Properties (Indexer) แบบเก็บออบเจ็กต์ไว้ในออบเจ็กต์ (Company.cs)

```
public class Company
{
    private Programmer[] ProgrammerList = new Programmer[3];

    public Programmer this[int index]
    {
        get
        {
            return ProgrammerList[index];
        }
        set
        {
            if (index < 0 || index > 2)
            {
```

```
                throw new IndexOutOfRangeException("Index Out of Range !!!");
            }
            else
            {
                ProgrammerList[index] = value;
            }
        }
    }

    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.Append("Programmer List : " + Environment.NewLine);

        foreach (Programmer p in ProgrammerList)
        {
            sb.Append(Environment.NewLine + p.ToString());
        }
        return sb.ToString();
    }
}
```

ต่อมาสร้างคลาส Company ขึ้นมา บริษัทนี้ต้องการโปรแกรมเมอร์ (คลาส Programmer) 3 คน ที่น่าสนใจอยู่ตรงบล็อกของ Set (กำหนดค่า) ก็ต้องมีการตรวจสอบด้วยว่า ถ้ามีการระบุลำดับติดลบหรือระบุลำดับมากกว่าจำนวนที่บริษัทนี้ต้องการ ให้แสดงข้อผิดพลาดออกมา ส่วนการ Override เมธอด ToString() ก็เพื่อแสดงรายชื่อโปรแกรมเมอร์ที่อยู่ในบริษัทนั่นเอง การใช้งานอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 7-2 การใช้งาน Default Properties (Indexer) แบบเก็บออบเจ็กต์ไว้ในออบเจ็กต์

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p1 As New Programmer()
        p1.FirstName = "นายศุภชัย"
        p1.LastName = "สมพานิช"

        Dim p2 As New Programmer()
        p2.FirstName = "น.ส ชลธิชา"
        p2.LastName = "ศรีดา"

        Dim p3 As New Programmer()
        p3.FirstName = "น.ส อิติมา"
```

VC# 2005 (Form1.cs)

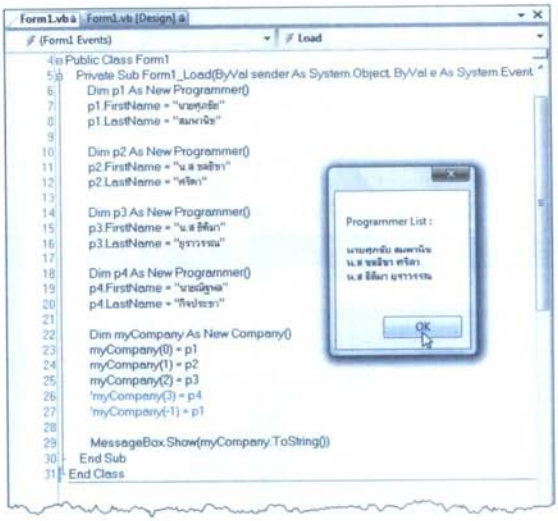
```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer p1 = new Programmer();
    p1.FirstName = "นายศุภชัย";
    p1.LastName = "สมพานิช";

    Programmer p2 = new Programmer();
    p2.FirstName = "น.ส ชลธิชา";
    p2.LastName = "ศรีดา";

    Programmer p3 = new Programmer();
    p3.FirstName = "น.ส อิติมา";
    p3.LastName = "ยุรางรรณ";

    Programmer p4 = new Programmer();
    p4.FirstName = "นายณัฐพล";
```

```
        p3.LastName = "ยุวาวรรณ"

        Dim p4 As New Programmer()
        p4.FirstName = "นายณัฐพล"
        p4.LastName = "กิจประชา"

        Dim myCompany As New Company()
        myCompany(0) = p1
        myCompany(1) = p2
        myCompany(2) = p3

        MessageBox.Show(myCompany.ToString())
    End Sub
End Class
```

```
        p4.LastName = "กิจประชา";
        Company myCompany = new Company();
        myCompany[0] = p1;
        myCompany[1] = p2;
        myCompany[2] = p3;

        MessageBox.Show(myCompany.ToString());
    }
```

ผู้เขียนสร้างโปรแกรมเมอร์ (คลาส Programmer) ขึ้นมา 4 คน สร้างบริษัท Company ที่ชื่อว่า myCompany ขึ้นมา การเพิ่มโปรแกรมเมอร์ 3 คนแรกให้กับบริษัท myCompany ไม่มีปัญหาแต่อย่างใด เพราะว่าอยู่ในขอบเขตตามที่บริษัทนี้ต้องการ ผลการทำงานแสดงดังรูปที่ 7-2

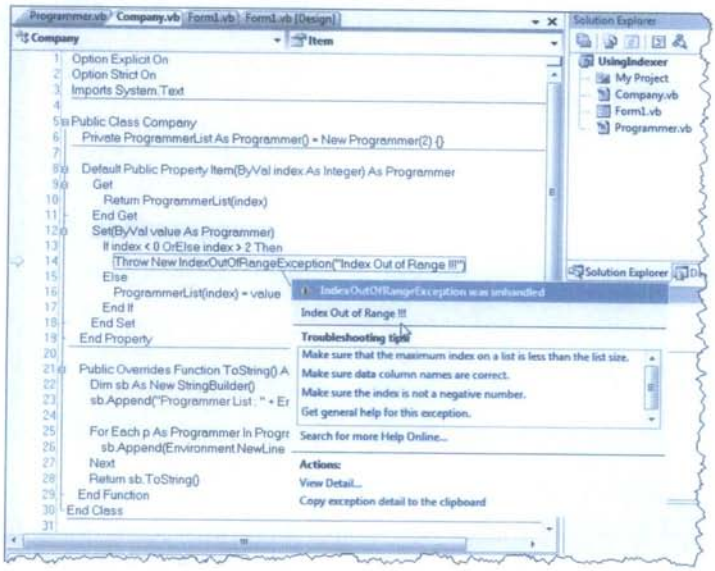
รูปที่ 7-2
ผลการรัน
ตัวอย่างที่ 7-2



แต่ถ้ามีการระบุลำดับเกินจากความต้องการของบริษัท myCompany หรือระบุลำดับติดลบ ก็จะเกิดข้อผิดพลาดตามที่ดักจับไว้ในบล็อก Set ของคลาส Company ดังรูปที่ 7-3

VB 2005	VC# 2005
myCompany(3) = p4 myCompany(-1) = p1	myCompany[3] = p4; myCompany[-1] = p1;

รูปที่ 7-3
กรณีระบุลำดับผิดพลาด



การทำ Overload Default Properties (Overload Indexer)

คุณสามารถทำ Overload กับ Default Properties (Indexer) ได้อีกด้วย ให้ดูตัวอย่างที่ 7-3 การทำ Overload Default Properties (Overload Indexer)

โค้ด VB 2005 และ VC# 2005 ที่ 7-3 การทำ Overload Default Properties (Overload Indexer)

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On
Imports System.Text

Public Class IntegerBox
    Private _arrInteger1() As Integer = {100, 200, 300,
400, 500}
    Private _arrInteger2(,) As Integer = {{1, 3, 5, 7}, {2, 4,
6, 8}}

    Default Public Property Item(ByVal index As Integer)
As Integer
        Get
            Return _arrInteger1(index)
        End Get
        Set(ByVal value As Integer)
            _arrInteger1(index) = value
        End Set
```

VC# 2005 (Class1.cs)

```
    public class IntegerBox
    {
        private int[] _arrInteger1 = new int[] {100, 200, 300,
400, 500};
        private int[,] _arrInteger2 = new int[,] {{1,3,5,7},
{2,4,6,8}};

        public int this[int index]
        {
            get{return _arrInteger1[index];}
            set{_arrInteger1[index] = value;}
        }

        public int this[int index1,int index2]
        {
            get{return _arrInteger2[index1,index2];}
            set{_arrInteger2[index1,index2] = value;}
        }
```

```
    End Property

    Default Public Property Item(ByVal index1 As Integer,
ByVal index2 As Integer) As Integer
        Get
            Return _arrInteger2(index1, index2)
        End Get
        Set(ByVal value As Integer)
            _arrInteger2(index1, index2) = value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Dim sb As New StringBuilder()
        sb.Remove(0, sb.Length)

        Dim _i As Integer
        For Each _i In _arrInteger1
            sb.Append(_i & " ")
        Next
        Return sb.ToString()
    End Function

    Public Function ShowData() As String
        Dim sb As New StringBuilder()
        sb.Remove(0, sb.Length)

        Dim _i As Integer
        For Each _i In _arrInteger2
            sb.Append(_i & " ")
        Next
        Return sb.ToString()
    End Function
End Class
```

```
    public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.Remove(0, sb.Length);
        foreach (int _i in _arrInteger1)
        {
            sb.Append(_i+ " ");
        }
        return sb.ToString();
    }

    public string ShowData()
    {
        StringBuilder sb = new StringBuilder();
        sb.Remove(0, sb.Length);
        foreach (int _i in _arrInteger2)
        {
            sb.Append(_i + " ");
        }
        return sb.ToString();
    }
}
```

ผู้เขียนสร้างกล่องใส่ตัวเลขจำนวนเต็มเช่นเดิม (คลาส IntegerBox) เพียงแต่ว่าเพิ่มฟิลด์แบบอาร์เรย์ 2 มิติที่ชื่อว่า _arrInteger2 ขึ้นมา กำหนดค่าให้กับสมาชิกแต่ละตัวมีค่าต่ำกว่า 10

เห็นได้ว่ามีการสร้าง Default Properties (Indexer) จำนวน 2 ชุด มี Signature แตกต่างกันกล่าวคือ รับพารามิเตอร์ไม่เท่ากัน เพราะว่าฟิลด์ _arrInteger1 เป็นอาร์เรย์แบบ 1 มิติ ส่วนฟิลด์ _arrInteger2 เป็นอาร์เรย์แบบ 2 มิติ

ถ้าต้องการอ่านค่าของสมาชิกที่อยู่ในฟิลด์ _arrInteger1 ให้อ่านจากเมธอด ToString() ซึ่งถูกทำ Override อยู่ แต่ถ้าต้องการอ่านค่าของสมาชิกที่อยู่ในฟิลด์ _arrInteger2 ให้อ่านจากเมธอด ShowData() ส่วนการใช้งานอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 7-3 การทำ Overload Default Properties (Overload Indexer)

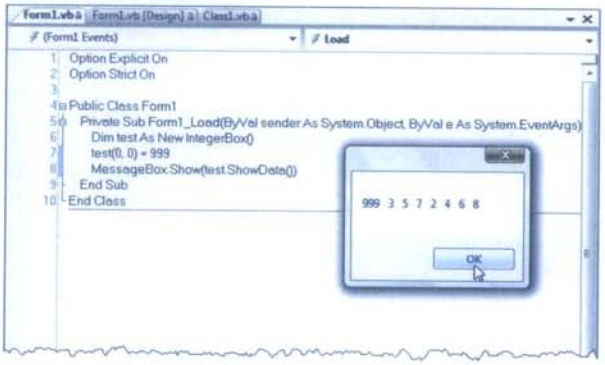
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim test As New IntegerBox()
        test(0, 0) = 999
        MessageBox.Show(test.ShowData())
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    IntegerBox test = new IntegerBox();
    test[0, 0] = 100;
    MessageBox.Show(test.ShowData());
}
```

รูปที่ 7-4 ผลการรันตัวอย่างที่ 7-3



จากรูปที่ 7-4 ถ้าคุณต้องการกำหนดค่าให้กับฟิลด์ _arrInteger2 ซึ่งเป็นอาร์เรย์แบบ 2 มิติ คุณต้องระบุลำดับของออบเจ็กต์ IntegerBox ที่ชื่อว่า test แบบ 2 มิติเช่นกัน ในกรณีนี้ผู้เขียนต้องการเปลี่ยนค่าของสมาชิกลำดับแรกนั่นเอง

สรุปท้ายบท

แม้ว่าเนื้อหาของการใช้ Default Properties (Indexer) ผู้เขียนอาจจะนำเสนอไม่มากนัก แต่อย่างน้อยที่สุดก็เลือกนำเสนอเฉพาะประเด็นที่สำคัญ ที่จะทำให้คุณผู้อ่านรู้จักกับ Default Properties (Indexer) ได้รวดเร็วยิ่งขึ้นนั่นเอง

CHAPTER 8

การสืบทอดคลาส (Inheritance)

บทนำ

การสืบทอดคลาสถือเป็นความสามารถที่สำคัญอย่างยิ่งของการเขียนโปรแกรมแบบ OOP คุณจะได้ศึกษาในบทนี้ รวมถึงกฎและกติกาหลักข้อที่ 2 และข้อ 3 ของ OOP

อีกประเด็นหนึ่งที่น่าสนใจคือ การทำ Partial Class ซึ่งเป็นคุณสมบัติใหม่ที่มากับ .NET Framework 2.0 ว่าช่วยอำนวยความสะดวกให้คุณสร้างคลาสต้นแบบได้อย่างไร

การสืบทอดคลาสคืออะไร ?

ข้อกำหนดข้อที่ 2 (จากทั้งหมด 3 ข้อ) ของการเขียนโปรแกรมแบบ OOP ก็คือ การสืบทอดคลาส (Inheritance) หรือการถ่ายทอดความสามารถของคลาสแม่ไปสู่คลาสลูก โดยที่คลาสลูกมีความสามารถของคลาสแม่ และยังสามารถเพิ่มเติมหรือแก้ไขให้คลาสลูกเก่งกว่าคลาสแม่ได้อีกด้วย ความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูกแบบนี้ เราจะเรียกว่า is-a Relationship

Inheritance เป็นข้อกำหนดที่ระบุว่า คลาสต้นแบบที่ถูกสร้างขึ้นมา ซึ่งเรียกว่าคลาสแม่ (Base Class หรือ Parent Class) สามารถที่จะทำสำเนาตัวมันเองได้ คลาสที่ได้คือ คลาสลูก (Derived Class หรือ Child Class) โดยการทำสำเนาดังกล่าวจะเรียกว่า การสืบทอดคลาส ส่งผลให้คลาสลูกมีความสามารถเหมือนกับคลาสแม่ โดยที่คุณสามารถเพิ่มหรือแก้ไขความสามารถให้คลาสลูกเก่งกว่าคลาสแม่ได้อีกด้วย เช่น คุณมีกระดาษอยู่ใบหนึ่ง ในกระดาษดังกล่าวมีสูตรคณิตศาสตร์อยู่ เมื่อคุณนำกระดาษใบนี้ไปถ่ายเอกสาร กระดาษที่ถ่ายเอกสารออกมาก็มีสูตรคณิตศาสตร์อยู่ด้วยเช่นกัน และคุณยังสามารถเพิ่มสูตรอื่นๆ เข้าไปที่กระดาษสำเนาได้อีกด้วย หรือแม้แต่จะแก้ไขสูตรคณิตศาสตร์ของคลาสลูกให้แตกต่างไปจากคลาสแม่ได้อีกด้วย

คำนิยามดังกล่าวเป็นเพียงการกล่าวโดยภาพรวม ภายใต้กติกาของการสืบทอดคลาส ยังประกอบไปด้วยกติกาย่อยต่างๆ อีกพอสมควร ผู้เขียนจะนำเสนออย่างชัดเจนให้เห็นว่า กติกาย่อยต่างๆ ช่วยเราสร้างคลาสอย่างไรบ้าง

ทำไม OOP ต้องมีความสามารถในการสืบทอดคลาส

สมมติว่าการเขียน OOP ไม่มีความสามารถของการสืบทอดคลาส เมื่อคุณต้องการสร้างคลาสขึ้นมา 2 คลาส คุณต้องเขียนโค้ด ดังนี้

VB 2005

```
Public Class Person
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property
End Class
```

VC# 2005

```
Public Class Manager
Private _FirstName As String = ""
Private _LastName As String = ""

Public Property FirstName() As String
    Get
        Return _FirstName
    End Get
    Set(ByVal value As String)
        _FirstName = value
    End Set
End Property

Public Property LastName() As String
    Get
        Return _LastName
    End Get
    Set(ByVal value As String)
        _LastName = value
    End Set
End Property

Public Function CalcuateSalary() As Double

End Function
End Class
```

จากโค้ดข้างต้นเห็นได้ว่ามีโค้ดส่วนหนึ่งซ้ำกันอยู่ เพราะว่าคนธรรมดา (คลาส Person) ต้องมีชื่อ-สกุล ส่วนผู้จัดการ (คลาส Manager) ก็ต้องมีชื่อ-สกุลเช่นกัน ที่เพิ่มขึ้นมาก็คือ ต้องมีการคำนวณเงินเดือน โดยการเพิ่มเมธอด CalculateSalary() เข้ามา

เห็นได้ว่าคุณต้องเสียเวลาส่วนหนึ่งไปกับการเขียนโค้ดเหมือนเดิม เพื่อให้คลาสใหม่มีความสมบูรณ์ในตัวของมันเอง จุดเข้มแข็งที่สุดของการเขียนโปรแกรมแบบ OOP ก็คือ การนำโค้ดกลับมาใช้ใหม่ได้ครั้งแล้วครั้งเล่า หรือเรียกทับศัพท์กันว่า re-use

จริงๆ แล้วในการเขียนโปรแกรมแบบเดิม (Structure Programming) คุณผู้อ่านก็ใช้แนวความคิดของการ re-use เช่นกัน โดยที่เราไม่รู้ตัว เหตุการณ์หนึ่งที่คุณผู้อ่านพบเจอเสมอก็คือ เวลาที่เราทำโปรเจ็กต์ใหม่เรามักจะกลับไป Copy โค้ดของโปรเจ็กต์เดิมบางส่วน (จะมากหรือน้อยก็แล้วแต่) ถ้าโชคดีหน่อยก็อาจจะไม่ต้องแก้ไขโค้ดมาก แต่ก็ไม่ต้องเขียนโค้ดใหม่ทั้งหมดเสียทีเดียว

หรือแม้กระทั่งการสร้างฟังก์ชัน (ซับรูทีน) ขึ้นมาใช้งานเอง ก็ถือเป็นการทำ re-use เช่นกัน โดยเกิดจากแนวความคิดที่ว่าถ้าโค้ดส่วนใดก็ตาม มีโอกาสทำงานมากกว่า 1 ครั้ง ให้เขียนโค้ดแยกออกเป็นฟังก์ชันต่างหากไว้ เช่น

VB 2005	VC# 2005

Private Function AddData(ByVal x As Double, ByVal y As Double) As Double
 	
End Function
	private double AddData(double x, double y)
	{
	
	}

โค้ดข้างต้นเป็นฟังก์ชันที่ทำหน้าที่บวกตัวเลข ถ้าคุณคิดว่าในโปรเจ็กต์ของคุณมีโอกาสบวกตัวเลขมากกว่า 1 ครั้ง ก็ต้องสร้างฟังก์ชันขึ้นมาทำหน้าที่นี้แทนที่จะเขียนโค้ด (ส่วนของการบวกเลข) ซ้ำตามจำนวนครั้งที่ต้องการ

เมื่อคุณเริ่มทำโปรเจ็กต์ใหม่ ถ้ามีขั้นตอนการบวกตัวเลข คุณสามารถ re-use ฟังก์ชันนี้ไปใช้ในโปรเจ็กต์ใหม่ของคุณได้ เห็นได้ว่า re-use ในระดับฟังก์ชันจะมีระเบียบมากกว่าการ Copy โค้ดของกรณีแรก เพราะว่าถ้าฟังก์ชันที่คุณสร้างขึ้นมายอดเยี่ยมดีพอ คุณสามารถใช้ฟังก์ชันดังกล่าวซ้ำได้ครั้งแล้วครั้งเล่า โดยที่ไม่ต้องแก้ไขโค้ดในฟังก์ชันเลยก็ได้

เหตุการณ์ข้างต้นเป็นการ re-use แบบฉบับของเราเอง ไม่มีกฎ กติกาใดมาบังคับ คิดจะ Copy โค้ดส่วนใดก็แล้วแต่เรา เราแก้ไขโค้ดตามที่เราต้องการ เพื่อให้สามารถใช้กับโปรเจ็กต์ใหม่ได้เป็นพอ เห็นได้ว่าแท้ที่จริงแล้วเรารู้จักการทำ re-use เพียงแต่ว่า re-use ของ OOP มีกฎกติกามาบังคับไว้นั่นเอง

กฎกติกาของ OOP ที่มีต่อการทำ re-use มีประโยชน์ตรงที่ เมื่อคุณสร้างคลาสต้นแบบขึ้นมาแล้ว เมื่อมีใครก็ตามนำคลาสดังกล่าวไปใช้หรือไปแก้ไข เขาสามารถนำไปใช้ได้อย่างถูกต้อง เพราะว่ากฎกติกาของ OOP เป็นมาตรฐานสากลรับรู้กันโดยทั่วไป OOP มีเพียง 1 เดียวจึงเป็นการนำโค้ดกลับมาใช้ใหม่ได้ครั้งแล้วครั้งเล่าที่เป็นมาตรฐานสากลนั่นเอง

พื้นฐานการสืบทอดคลาส

ตัวอย่างที่ 8- 1 พื้นฐานการสืบทอดคลาสให้คุณเขียนโค้ด ดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-1 พื้นฐานการสืบทอดคลาส

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Public Sub Speak()
        MessageBox.Show("Person พูด")
    End Sub

    Public Sub Walk()
        MessageBox.Show("Person เดิน")
    End Sub
End Class

Public Class Programmer
    Inherits Person

    Public Sub Programming()
        MessageBox.Show("Programmer เขียนโปรแกรมได้")
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Person
{
    public void Speak()
    {
        MessageBox.Show("Person พูด");
    }

    public void Walk()
    {
        MessageBox.Show("Person เดิน");
    }
}

public class Programmer : Person
{
    public void Programming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมได้");
    }
}
```

โค้ดข้างต้นผู้เขียนสร้างคลาสต้นแบบที่ชื่อว่า Person ขึ้นมามี 2 เมธอดคือ เมธอด Speak() และเมธอด Walk() เป็นเมธอดแบบ Public กล่าวคือ คนธรรมดา (คลาส Person) ต้องพูดได้ (เมธอด Speak()) และเดินได้ (เมธอด Walk())

จากนั้นสร้างอีก 1 คลาสชื่อว่า Programmer โดยการสืบทอดคลาสจากคลาส Person โดยที่ใน VB 2005 ใช้คำสั่ง Inherits ส่วน VC# 2005 จะระบุชื่อคลาสที่ต้องการสืบทอด โดยใช้เครื่องหมาย : อาจจะเรียกการสืบทอดคลาสแบบนี้ว่าเป็นแบบ Single Inheritance ก็ได้

โดยที่คลาส Programmer ผู้เขียนเพิ่มเมธอดแบบ Public อีก 1 เมธอด ชื่อว่า Programming() ซึ่งหมายถึง คลาส Programmer ต้องเขียนโปรแกรมได้นั่นเอง

เราเรียกคลาส Person ว่า เป็น Parent Class หรือคลาสแม่ ส่วนคลาส Programmer เป็น Child คลาสหรือคลาสลูก ในกรณีนี้เราเพิ่มความสามารถให้คลาสลูกมีเมธอดใหม่เพิ่มเข้ามา ส่งผลให้คลาสลูกเก่งกว่าคลาสแม่

ในตอนต้นผู้เขียนกล่าวไว้ว่า การสืบทอดคลาสจะเรียกความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูกว่าเป็นความสัมพันธ์แบบ is-a เพราะว่า

> โปรแกรมเมอร์ (คลาส Programmer) ก็คือ (is-a) คนธรรมดา (คลาส Person) เช่นกัน

ดังนั้น โปรแกรมเมอร์ต้องพูดได้ (เมธอด Speak()), เดินได้ (เมธอด Walk) ที่เพิ่มมาใหม่คือ ต้องเขียนโปรแกรมได้ (เมธอด Programming()) จากนั้นให้เขียนโค้ดทดสอบคลาสข้างต้นดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-1 พื้นฐานการสืบทอดคลาส

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p1 As New Person()
        p1.Speak()
        p1.Walk()

        Dim p2 As New Programmer()
        p2.Speak()
        p2.Walk()
        p2.Programming()
    End Sub
End Class
```

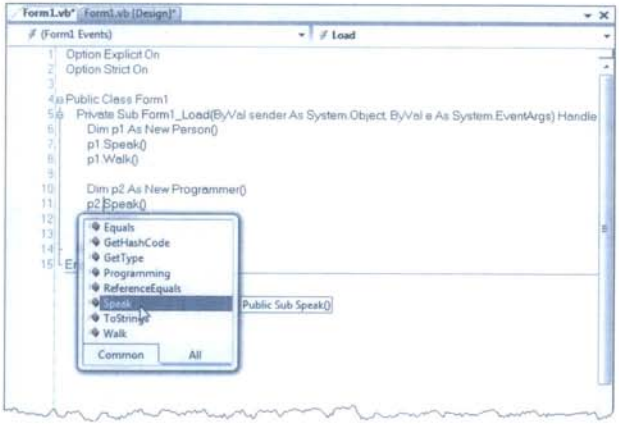
VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Person p1 = new Person();
    p1.Speak();
    p1.Walk();

    Programmer p2 = new Programmer();
    p2.Speak();
    p2.Walk();
    p2.Programming();
}
```

เริ่มต้นสร้างออบเจ็กต์ Person ที่ชื่อว่า p1 ขึ้นมา เห็นได้ว่าจะสร้างออบเจ็กต์ Person ขึ้นมาต้องสั่งให้มันเกิดด้วยคำสั่ง New (new) เพราะว่าเมธอดในคลาส Person เป็นแบบ Dynamic นั่นเอง เราจะเรียกออบเจ็กต์ p1 ว่าเป็น Instance ของคลาส Person โดยที่ออบเจ็กต์ p1 ทำได้ 2 อย่างคือ พูด (เมธอด Speak()) และเดิน (เมธอด Walk())

ต่อมาสร้างออบเจ็กต์ Programmer ที่ชื่อว่า p2 ขึ้นมา p2 คือ อินสแตนซ์ของคลาส Programmer พบว่า p2 สามารถเรียกใช้เมธอด Speak() และเมธอด Walk() ของคลาส Person (คลาสแม่) ได้ด้วย ดังรูปที่ 8-1

รูปที่ 8-1
เมธอดของคลาสลูก Programmer



เหตุที่คลาสลูก Programmer เรียกใช้เมธอดทั้ง 2 ของคลาสแม่ (คลาส Person) ได้ เป็นเพราะว่าคลาสแม่กำหนดให้เมธอดทั้ง 2 มีขอบเขตเป็นแบบ Public นั่นเอง ผู้เขียนขอสรุปเป็นกติกาย่อยข้างต้นคือ

> ส่วนประกอบของคลาสแม่ที่มีขอบเขตแบบ Public เมื่อมีการสืบทอดคลาส จะถูกถ่ายทอดไปยังคลาสลูกด้วย

Implicit Inheritance

การสร้างคลาสใน .NET มีการสืบทอดคลาสอีกลักษณะหนึ่ง เรียกว่า Implicit Inheritance หมายถึง เมื่อมีการสร้างคลาสขึ้นมา คลาสดังกล่าวจะสืบทอดมาจาก System.Object ซึ่งเป็นคลาสแม่ของคลาสต่างๆ ใน .NET

VB 2005	VC# 2005

Public Class Customer
 	
End Class
	public class Customer
	{
	
	}

โค้ดข้างต้นเป็นการสร้างคลาส Customer ขึ้นมา ถือว่าเกิดการสืบทอดคลาสแบบ Implicit Inheritance แล้ว โดยที่คลาส Customer สืบทอดมาจาก System.Object นั่นเอง

ข้อแตกต่างอีกอย่างหนึ่งระหว่าง Class กับ Structure (struct) ก็คือ เรื่องของการทำ Explicit Inheritance หรือสืบทอดคลาสโดยที่เราเป็นผู้กำหนดเอง ให้ดูตัวอย่างที่ 8-2 ข้อแตกต่างของ Implicit & Explicit Inheritance ระหว่างคลาสกับสตรัคเจอร์ ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-2 ข้อแตกต่างของ Implicit & Explicit Inheritance ระหว่างคลาสกับสตรัคเจอร์

VB 2005

```
Option Explicit On
Option Strict On

Public Class Class1

End Class

Public Class Class2
    Inherits Object

End Class

Public Structure Struct1
    Public _x As Integer

End Structure
-----------------------------
Public Structure Struct2
    Inherits System.ValueType
    Public _x As Integer
End Structure
```

VC# 2005

```
public class Class1
    {

    }

    public class Class2:System.Object
    {

    }

    public struct struct1
    {

    }
-----------------------------
    public struct struct2 : System.ValueType
    {

    }
```

ให้คุณมองเป็น 2 กรณีกล่าวคือ

กรณีที่ 1 : ผู้เขียนสร้างคลาสขึ้นมา 2 ชุด โดยที่

- คลาสที่ชื่อว่า Class1 เกิดการสืบทอดคลาสแบบ Implicit Inheritance
- คลาสที่ชื่อว่า Class2 คุณสามารถกำหนดให้สืบทอดคลาสจาก System.Object เป็นการทำ Explicit Inheritance

กรณีที่ 2 : ผู้เขียนสร้าง Structure ขึ้นมา 2 ชุด โดยที่

- สตรัคเจอร์ที่ชื่อว่า Struct1 เกิดการสืบทอดคลาสแบบ Implicit Inheritance จาก System.ValueType เพราะ Structure เป็น Value Type ชนิดหนึ่งเช่นกัน
- สตรัคเจอร์ที่ชื่อว่า Struct2 คุณไม่สามารถทำ Explicit Inheritance จาก System.ValueType ได้ โค้ดส่วนนี้จะเกิดข้อผิดพลาด

ทำความรู้จักกับขอบเขตแบบ Protected

ผู้เขียนจะขอยกอีก 1 ตัวอย่าง เพื่อให้เห็นภาพของพื้นฐานการสืบทอดคลาสชัดเจนยิ่งขึ้นให้ดูตัวอย่างที่ 8-3 ทำความรู้จักกับขอบเขตแบบ Protected ให้คุณเขียนโค้ด ดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-3 ทำความรู้จักกับขอบเขตแบบ Protected

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Rectangle
    Protected _Width As Double = 0.0
    Protected _Height As Double = 0.0

    Public Property Width() As Double
        Get
            Return _Width
        End Get
        Set(ByVal value As Double)
            _Width = value
        End Set
    End Property

    Public Property Height() As Double
        Get
            Return _Height
        End Get
        Set(ByVal value As Double)
            _Height = value
        End Set
    End Property

    Public Function Calculate() As Double
        Return _Width * _Height
    End Function
End Class

Public Class Cube
    Inherits Rectangle

    Private _Thick As Double = 0.0

    Public Property Thick() As Double
        Get
            Return _Thick
        End Get
```

VC# 2005 (Class1.cs)

```
public class Rectangle
{
    protected double _Width = 0.0;
    protected double _Height = 0.0;

    public double Width
    {
        get{return _Width;}
        set{_Width = value;}
    }

    public double Height
    {
        get{return _Height;}
        set{_Height = value;}
    }

    public double Calculate()
    {
        return _Width * _Height;
    }
}

public class Cube : Rectangle
{
    private double _Thick = 0.0;

    public double Thick
    {
        get{return _Thick;}
        set{_Thick = value;}
    }

    public double CalculateCube()
    {
        return _Width * _Height * _Thick;
    }
}
```

```
        Set(ByVal value As Double)
            _Thick = value
        End Set
    End Property

    Public Function CalculateCube() As Double
        Return _width * _height * _Thick
    End Function
End Class
```

ตัวอย่างนี้ผู้เขียนสร้างคลาสต้นแบบชื่อว่า Rectangle ทำหน้าที่คำนวณหาพื้นที่ประกอบด้วย 2 คุณสมบัติกับอีก 1 เมธอด และกำหนดขอบเขตให้เป็นแบบ Public

คุณสมบัติ/เมธอด	รายละเอียด
คุณสมบัติ Height	อ่านค่าหรือกำหนดความยาว
คุณสมบัติ Width	อ่านค่าหรือกำหนดความกว้าง
เมธอด Calculate()	ทำหน้าที่คำนวณพื้นที่ หาได้จากสูตร ความยาว*ความกว้าง

แต่ว่าคุณสมบัติ Height และคุณสมบัติ Width ต้องทำงานร่วมกับฟิลด์ _Height และฟิลด์ _Width เสมอ เมื่อมีการสร้างอินสแตนซ์ของคลาส Rectangle ขึ้นมาด้วยคำสั่ง New (new) ผู้ใช้สามารถกำหนดความยาวหรือความสูงผ่านทางคุณสมบัติทั้ง 2 ส่งผลให้ฟิลด์ _Height เก็บความยาว ส่วนฟิลด์ _Width เก็บความกว้างไว้

จากกฎของการทำ Encapsulation ที่กำหนดไว้ว่า ฟิลด์ต่างๆ ที่อยู่ในคลาสถือเป็นข้อมูลภายใน ภายนอกมองไม่เห็น จึงต้องกำหนดขอบเขตเป็นแบบ Private แต่ฟิลด์ _Height และฟิลด์ _Width ของคลาส Rectangle ผู้เขียนกำหนดขอบเขตเป็นแบบ Protected

ขอบเขตแบบ Protected ยังคงมีความเข้มงวดเหมือนกับขอบเขต Private กล่าวคือ ภายนอกไม่สามารถกำหนดค่าโดยตรงให้กับฟิลด์ทั้ง 2 ได้เช่นเดิม ข้อแตกต่างก็คือ เมื่อมีการสืบทอดคลาส ฟิลด์แบบ Private ของคลาสแม่ไม่สามารถถ่ายทอดไปสู่คลาสลูกได้ แต่ฟิลด์แบบ Protected สามารถถ่ายทอดไปสู่คลาสลูกได้

ถ้าเราจะกล่าวว่าขอบเขตแบบ Private เข้มงวดมากที่สุด ส่วนขอบเขตแบบ Public ไม่มีความเข้มงวด ดังนั้น ขอบเขตแบบ Protected จึงอยู่ระหว่าง Private กับ Public นั่นเอง สรุปได้ว่า

> ส่วนประกอบของคลาสแม่ที่เป็น Public และ Protected สามารถถ่ายทอดสู่คลาสลูกได้

ผู้เขียนสร้างคลาส Cube ขึ้นมาทำหน้าที่คำนวณปริมาตร โดยการสืบทอดจากคลาส Rectangle การคำนวณปริมาตรเกิดจากความยาว*ความกว้าง*ความหนา ดังนั้น ที่คลาสลูกผู้เขียนเพิ่มคุณสมบัติ Thick เพื่อกำหนดความหนาเก็บไว้ที่ฟิลด์ _Thick และเพิ่มเมธอดใหม่ชื่อว่า CalculateCube() ทำหน้าที่คำนวณปริมาตรที่เกิดจากความยาว (ฟิลด์ _Height ที่ได้จากคลาสแม่)*ความกว้าง (ฟิลด์ _Width ที่ได้จากคลาสแม่เช่นกัน)*ความหนา (ฟิลด์ _Thick ที่เพิ่มเข้ามาใหม่ในคลาสลูก)

VB 2005

```
Public Function CalculateCube() As Double
    Return _width * _height * _Thick
End Function
```

VC# 2005

```
public double CalculateCube()
{
    return _Width * _Height * _Thick;
}
```

ส่วนการใช้งานคลาสทั้ง 2 อยู่ใน Form1 แสดงโค้ดดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-3 ทำความรู้จักกับขอบเขตแบบ Protected

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim r As New Rectangle()
        Dim result1 As Double = 0.0
        r.Width = 10
        r.Height = 20
        result1 = r.Calculate()
        MessageBox.Show("พื้นที่ : " & result1.ToString(),
"ผลการคำนวณพื้นที่")

        Dim c As New Cube()
        Dim result2 As Double = 0.0
        c.Width = 5
        c.Height = 10
        c.Thick = 20
        result2 = c.CalculateCube()
        MessageBox.Show("ปริมาตร : " & result2.ToString(),
"ผลการคำนวณปริมาตร")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Rectangle r = new Rectangle();
    double result1 = 0.0;
    r.Width = 10;
    r.Height = 20;
    result1 = r.Calculate();
    MessageBox.Show("พื้นที่ : " + result1.ToString(),
"ผลการคำนวณพื้นที่");

    Cube c = new Cube();
    double result2 = 0.0;
    c.Width = 5;
    c.Height = 10;
    c.Thick = 20;
    result2 = c.CalculateCube();
    MessageBox.Show("ปริมาตร : " + result2.ToString(),
"ผลการคำนวณปริมาตร");
}
```

ที่คลาสแม่ผู้เขียนสร้างออบเจ็กต์ Rectangle ที่ชื่อว่า r ขึ้นมา กำหนดความกว้างเป็น 10 และส่วนความยาวเท่ากับ 20 คำนวณพื้นที่ด้วยเมธอด Calculate() พื้นที่ที่ได้คือ 200

VB 2005

```
Dim r As New Rectangle()
Dim result1 As Double = 0.0
r.Width = 10
r.Height = 20
result1 = r.Calculate()
MessageBox.Show("พื้นที่ : " & result1.ToString(), "ผลการคำนวณพื้นที่")
```

VC# 2005

```
Rectangle r = new Rectangle();
double result1 = 0.0;
r.Width = 10;
r.Height = 20;
result1 = r.Calculate();
MessageBox.Show("พื้นที่ : " + result1.ToString(), "ผลการคำนวณพื้นที่");
```

ส่วนที่คลาสลูกสร้างออบเจ็กต์ Cube ที่ชื่อว่า c กำหนดความกว้าง 5 ความยาว 10 ความหนา 20 คำนวณปริมาตรด้วยเมธอด CalculateCube() ปริมาตรที่ได้คือ 1000

VB 2005

```
Dim c As New Cube()
Dim result2 As Double = 0.0
c.Width = 5
c.Height = 10
c.Thick = 20
result2 = c.CalculateCube()
MessageBox.Show("ปริมาตร : " & result2.ToString(), "ผลการคำนวณปริมาตร")
```

VC# 2005

```
Cube c = new Cube();
double result2 = 0.0;
c.Width = 5;
c.Height = 10;
c.Thick = 20;
result2 = c.CalculateCube();
MessageBox.Show("ปริมาตร : " + result2.ToString(), "ผลการคำนวณปริมาตร");
```

จากผลการทำงานทั้ง 2 คลาส เห็นได้ว่าแม้จะมีการถ่ายทอดฟิลด์ของคลาสแม่ไปสู่คลาสลูก แต่ในการเก็บค่าต่างๆ ไม่มีความเกี่ยวข้องกันแต่อย่างใด เพราะเป็นการทำงานของแต่ละ Instance แตกต่างจากสมาชิกแบบ Shared (static) ที่ถูกใช้งานร่วมกันทุก Instance นั่นเอง

ที่คลาสแม่ให้คุณลองกำหนดค่าของฟิลด์ _Width และฟิลด์ _Height โดยตรง พบว่าคุณไม่สามารถทำได้ เพราะว่าขอบเขตแบบ Protected ไม่ยอมให้ภายนอกมองเห็นเช่นเดียวกับขอบเขตแบบ Private นั่นเอง ดังนั้น จึงต้องกำหนดค่าผ่านทางคุณสมบัติ Width และ Height ซึ่งเป็นช่องทางที่เรากำหนดไว้เท่านั้น

VB 2005

```
Dim r As New Rectangle()
r._Width = 10
r._Height = 20
```

VC# 2005

```
Rectangle r = new Rectangle();
r._Width = 10;
r._Height = 20;
```

ส่วนที่คลาสลูก Cube คุณสามารถคำนวณหาพื้นที่โดยการเรียกใช้เมธอด Calculate() ได้อีกด้วย ซึ่งได้มาจากคลาสแม่นั่นเอง ส่งผลให้คลาสลูกเก่งกว่าคลาสแม่ เพราะว่าคำนวณได้ทั้งปริมาตรและพื้นที่นั่นเอง

การทำ Partial Class

การทำ Partial Class เป็นอีกหนึ่งวิธีการที่คุณสามารถเพิ่มเติมให้คลาสของคุณเก่งขึ้น โดยการแบ่งโค้ดออกเป็นส่วนๆ แยกเก็บในหลายๆ ไฟล์ และใช้คีย์เวิร์ด Partial กำกับไว้ที่คลาสที่คุณต้องการทำ Partial Class

สมมติว่าคุณมีโค้ดอยู่ 1 ชุด เก็บไว้ในไฟล์ DAL.vb (DAL.cs) อาจจะเป็นโค้ดที่มีขนาดใหญ่มากๆ หรือได้มาจากโปรแกรมสร้างโค้ดอัตโนมัติ (Generate Code) ถ้าคุณต้องการเพิ่มเติมให้คลาสที่เก็บอยู่ในไฟล์ DAL.vb (DAL.cs) เก่งขึ้น แต่คุณไม่ต้องการเข้าไปแก้ไขในไฟล์ DAL โดยตรง เพราะว่าในไฟล์ DAL ที่ได้มามีรายละเอียดโค้ดมากมาย หลายพัน หลายหมื่นบรรทัด เราไม่ต้องการเข้าไปยุ่ง คุณสามารถสร้างไฟล์ใหม่แล้วเขียนโค้ดชุดใหม่เข้าไปในไฟล์ใหม่ ด้วยคุณสมบัติของการทำ Partial Class ทั้งโค้ดชุดเก่าที่อยู่ในไฟล์ DAL เดิม และโค้ดชุดใหม่ที่อยู่ในไฟล์ใหม่จะถือเป็นคลาสเดียวกัน ให้ดูตัวอย่างที่ 8-4 การทำ Partial Class

โค้ด VB 2005 และ VC# 2005 ที่ 8-4 การทำ Partial Class (ส่วนที่ 1)

VB 2005 (Customer.Part1.vb)

```
Option Explicit On
Option Strict On

Partial Public Class Customer
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Property FirstName() As String
        Get
        Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property
End Class
```

VC# 2005 (Customer.Part1.cs)

```
public partial class Customer
{
    private string _FirstName = "";
    private string _LastName = "";

    public string FirstName
    {
        get{return _FirstName;}
        set{_FirstName = value;}
    }

    public string LastName
    {
        get{return _LastName;}
        set{_LastName = value;}
    }
}
```

ผู้เขียนสร้างคลาส Customer ขึ้นมา กำหนดให้เป็นแบบ Partial Class โดยใช้คีย์เวิร์ด Partial เก็บไว้ในไฟล์ Customer.Part1.vb (Customer.Part1.cs) มี 2 คุณสมบัติคือ FirstName และ LastName ต่อมาผู้เขียนเห็นว่าคลาส Customer เดิม มีความสามารถไม่เพียงพอต่อการใช้งาน จึงสร้างใหม่ขึ้นมาอีก 1 ส่วน เก็บอยู่ในไฟล์ Customer.Part2.vb (Customer.Part2.cs)

โค้ด VB 2005 และ VC# 2005 ที่ 8-4 การทำ Partial Class (ส่วนที่ 2)

VB 2005 (Customer.Part2.vb)

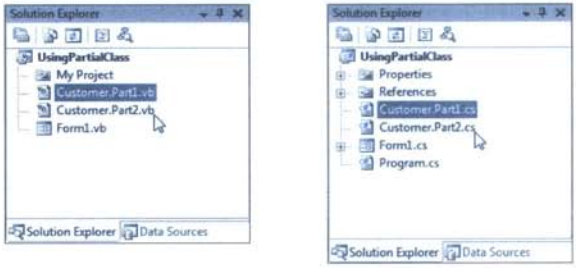
```
Option Explicit On
Option Strict On

Partial Public Class Customer
    Private _Address As String = ""

    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
        End Set
    End Property
End Class
```

VC# 2005 (Customer.Part2.cs)

```
public partial class Customer
{
    private string _Address = "";

    public string Address
    {
        get{return _Address;}
        set{_Address = value;}
    }
}
```

ส่วนที่ 2 ผู้เขียนคิดว่าคลาส Customer ต้องมีที่อยู่ได้ด้วย จึงสร้างคุณสมบัติ Address ขึ้นมา ตอนนี้คลาส Customer ถูกแยกออกเป็น 2 ส่วนเก็บไว้ใน 2ไฟล์ ดังรูปที่ 8-2

รูปที่ 8-2 Partial Class ของคลาส Customer ใน Solution Explorer



ส่วนการใช้งานคลาส Customer อยู่ใน Form1 ดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-4 การทำ Partial Class

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim C1 As New Customer
        C1.FirstName = "ศุภชัย"
        C1.LastName = "สมพานิช"
        C1.Address = "กรุงเทพ"
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Customer C1 = new Customer();
    C1.FirstName = "ศุภชัย";
    C1.LastName = "สมพานิช";
    C1.Address = "กรุงเทพ";
}
```

จากโค้ดข้างต้นแม้ว่าคลาส Customer จะถูกเก็บแยกต่างไฟล์กัน แต่มีการทำ Partial Class ไว้ ส่งผลให้ภายนอกยังคงมองเห็นคลาส Customer เป็นหนึ่งเดียว เพราะว่าสามารถเรียกใช้คุณสมบัติ FirstName, LastName ที่เก็บอยู่ในไฟล์ส่วนที่ 1 และคุณสมบัติ Address ที่เก็บอยู่ในไฟล์ส่วนที่ 2 ได้ตามปกติ

การทำ Partial Class ของคลาส Customer ผู้เขียนระบุคำว่า Partial ไว้ทั้ง 2 ส่วน แต่ .NET ยอมให้มี 1 ส่วนที่ไม่จำเป็นต้องระบุ Partial กำกับไว้ก็ได้ ดังนั้น คุณผู้อ่านไม่ต้องใส่คำว่า Partial ไว้ทั้ง 2 ส่วนตามผู้เขียนก็ได้ เช่น คลาสของคุณอาจจะเก็บแยกไว้ 5 ไฟล์ โดยที่ใน 5 ไฟล์นี้คุณจะระบุ Partial ครบทั้ง 5 ไฟล์หรือ 4 ไฟล์ก็ได้ไม่ผิดแต่อย่างใด

เหตุผลที่ .NET ยอมให้มี 1 ส่วนไม่ต้องระบุ Partial กำกับไว้ เป็นเพราะว่าในกรณีที่คุณต้องการทำ Partial Class กับคลาสที่สร้างจาก .NET เวอร์ชันก่อนหน้านี้ (ซึ่งไม่มีความสามารถของการทำ Partial Class) สามารถทำได้โดยที่ไม่ต้องไประบุ Partial ครบทุกส่วนนั่นเอง

วิธีการเลือกใช้งานระหว่าง Partial Class กับการทำ Inheritance

จากที่ผู้เขียนกล่าวไว้ว่าการทำ Partial Class คือ การแบ่งโค้ดออกเป็นส่วนๆ แยกเก็บในหลายๆ ไฟล์ เพื่อเพิ่มเติมความสามารถต่างๆ ให้กับคลาสของคุณ ส่วนการทำ Inheritance หมายถึง การสืบทอดคลาสเพื่อเพิ่มความสามารถของคลาสลูกให้เก่งกว่าคลาสแม่ หรือต้องการแก้ไขความสามารถของคลาสลูกให้แตกต่างไปจากคลาสแม่

เห็นได้ว่าทั้ง 2 กรณีไม่ว่าจะเป็นการทำ Partial Class หรือการทำ Inheritance ล้วนแต่เป็นการเพิ่มเติมความสามารถให้กับคลาสของคุณทั้งสิ้น คำถามก็คือ เมื่อใดควรใช้ Partial Class เมื่อใดควรใช้ Inheritance ให้คุณดูตัวอย่างต่อไปนี้

สมมติว่าคุณมีอยู่ 1 คลาส ชื่อว่า SuperClass เป็นคลาสที่สมบูรณ์ และยอดเยี่ยมที่สุดเท่าที่โลกนี้เคยมีมา คุณต้องการเพิ่มเติมเมธอดใหม่ที่ชื่อว่า ABC() เข้าไปในคลาส เพื่อให้คลาสนี้เก่งขึ้น คุณมี 2 ทางเลือกคือ

1. ทำ Partial Class เพราะความที่คลาส SuperClass เขียนโค้ดไว้อย่างสุดยอด คุณไม่ต้องการเข้าไปยุ่งกับโค้ดของคลาสนี้ จึงใช้วิธีการทำ Partial Class ดีกว่า โดยการแยกโค้ดของเมธอด ABC() ไว้อีกไฟล์ ส่งผลให้คลาส SuperClass มีเมธอด ABC() เพิ่มขึ้นมา
2. สร้างคลาสลูกใหม่ขึ้นมาอีกคลาส โดย Inheritance มาจากคลาส SuperClass ก่อน แล้วค่อยเพิ่มเมธอด ABC() ให้กับคลาสลูก แล้วก็นำคลาสลูกไปใช้งาน เห็นได้ว่าวิธีนี้ก็ดีไปอีกแบบ เพราะคุณก็ไม่ได้ยุ่งกับโค้ดที่อยู่ในคลาส SuperClass เช่นกัน

วิธีการก็คือ ขอให้คุณดูว่าเมธอด ABC() ที่คุณต้องการเพิ่มเข้ามาเป็นเมธอดหลักของคลาส เป็นการทำงานที่คลาสดังกล่าวต้องรู้ ต้องทำได้ ต้องมี ถ้าคำตอบที่ได้คือ ใช่ ให้คุณเลือกวิธีทำ Partial Class แต่ถ้าเมธอด ABC() เป็นเพียงการทำงานเฉพาะเจาะจงบางกรณี ใช้เฉพาะอย่าง หรือใช้กับบางเงื่อนไข ขอให้คุณใช้วิธี Inheritance ทั้งนี้ขอให้คุณผู้อ่านดูความจำเป็นอื่นๆ ประกอบการตัดสินใจด้วย

การทำ Override เมธอดของคลาสแม่

จากตัวอย่างที่ผ่านมา เห็นได้ว่าประโยชน์ของการสืบทอดคลาสที่เห็นอย่างชัดเจนก็คือ เราสามารถเพิ่มเติมเมธอดใหม่ให้คลาสลูกเก่งกว่าคลาสแม่ ทำให้เราไม่ต้องเขียนโค้ดใหม่ทั้งหมดเพียงเพื่อทำงานที่คลาสแม่ไม่มี โดยการสืบทอดคลาสของ OOP ยังรวมไปถึงการแก้ไขเมธอดที่คลาสลูกได้มาจากคลาสแม่ ให้ทำงานแตกต่างไปจากเดิมได้อีกด้วย

เพื่อให้การศึกษาความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูกง่ายยิ่งขึ้น รูปแบบที่ผู้เขียนนำมาใช้ก็คือ คลาสแม่ตั้งชื่อว่า Parent ส่วนคลาสลูกตั้งชื่อว่า Child ให้ดูตัวอย่างที่ 8-5 การทำ Override เมธอดของคลาสแม่

โค้ด VB 2005 และ VC# 2005 ที่ 8-5 การทำ Override เมธอดของคลาสแม่

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On
Imports System.Windows.Forms

Public Class Parent
    Public Sub Method1()
        MessageBox.Show("Method1 ของคลาสแม่ทำงาน")
    End Sub

    Public Overridable Sub Method2()
        MessageBox.Show("Method2 ของคลาสแม่ทำงาน")
    End Sub
End Class

Public Class Child
```

VC# 2005 (Class1.cs)

```
public class Parent
{
    public void Method1()
    {
        MessageBox.Show("Method1 ของคลาสแม่ทำงาน");
    }

    public virtual void Method2()
    {
        MessageBox.Show("Method2 ของคลาสแม่ทำงาน");
    }
}

public class Child : Parent
{
```

```
    Inherits Parent

    Public Overloads Sub Method1()
        MessageBox.Show("Method1 ของคลาสลูกทำงาน")
    End Sub

    Public Overrides Sub Method2()
        MessageBox.Show("Method2 ของคลาสลูกทำงาน")
    End Sub
End Class
```

```
    public new void Method1()
    {
        MessageBox.Show("Method1 ของคลาสลูกทำงาน");
    }

    public override void Method2()
    {
        MessageBox.Show("Method2 ของคลาสลูกทำงาน");
    }
}
```

เนื่องจากว่ามีการใช้งาน MessageBox ในคลาส จึงต้องเพิ่มเนมสเปซต่อไปนี้ก่อน

VB 2005

```
mports System.Windows.Forms
```

VC# 2005

```
using System.Windows.Forms;
```

วิธีการดูโค้ดชุดนี้ก็คือ ให้คุณดูว่าเมธอดที่ถูกสั่งให้ทำงานคือ เมธอดของคลาสแม่หรือคลาสลูก โดยสังเกตจาก MessageBox ที่ปรากฏขึ้นมาว่าเป็นของเมธอดใด จะทำให้คุณศึกษากติกาของการสืบทอดคลาสหรือเรื่องอื่นๆ ได้เร็วยิ่งขึ้น

ที่คลาสแม่ Parent ประกอบด้วย 2 เมธอด ชื่อว่า Method1() และ Method2() ตามลำดับ แยกได้ 2 กรณีคือ

- **กรณี VB 2005** : เมธอดที่ชื่อว่า Method1() เป็นเมธอดปกติ ส่วนเมธอดที่ชื่อว่า Method2() กำหนดให้เป็นเมธอดแบบ Overridable ซึ่งหมายถึง อนุญาตให้แก้ไขได้
- **กรณี VC# 2005** : เมธอด Method1() เป็นเมธอดปกติเช่นกัน ส่วนเมธอด Method2() กำหนดให้เป็นเมธอดแบบ virtual

VB 2005

```
Public Class Parent
    Public Sub Method1()
        MessageBox.Show("Method1 ของคลาสแม่ทำงาน")
    End Sub

    Public Overridable Sub Method2()
        MessageBox.Show("Method2 ของคลาสแม่ทำงาน")
    End Sub
End Class
```

VC# 2005

```
public class Parent
{
    public void Method1()
    {
        MessageBox.Show("Method1 ของคลาสแม่ทำงาน");
    }

    public virtual void Method2()
    {
        MessageBox.Show("Method2 ของคลาสแม่ทำงาน");
    }
}
```

ที่คลาสลูก Child กำหนดให้สืบทอดคลาสมาจากคลาสแม่ Parent ถ้าเป็นการสืบทอดตามปกติ คลาสลูก Child จะได้เมธอด Method1() และเมธอด Method2() มาด้วย ส่งผลให้เมธอดทั้ง 2 ที่อยู่ในคลาสลูกทำงานเหมือนกับคลาสแม่

VB 2005

```
Public Class Child
    Inherits Parent

End Class
```

VC# 2005

```
public class Child : Parent
{

}
```

คำถามที่ตามมาก็คือ ถ้าเราต้องการกำหนดให้เมธอดทั้ง 2 ที่คลาสลูกได้มาจากคลาสแม่ ทำงานแตกต่างไปจากคลาสแม่ต้องทำอย่างไร วิธีการก็คือ ให้คุณดูเมธอดของคลาสแม่ว่าเป็นเมธอดลักษณะใด แยกออกเป็น 2 กรณี

- **กรณี VB 2005** : พบว่าที่เมธอด Method1() เป็นเมธอดปกติที่คลาสลูก ให้คุณทำ Overloads เมธอดนี้ ในส่วนของเมธอด Method2() พบว่าเป็นเมธอดที่อนุญาตให้แก้ไขได้ (Overridable) ที่คลาสลูก ให้คุณทำ Overrides
- **กรณี VC# 2005** : เมธอด Method1() เป็นเมธอดปกติที่คลาสลูก ให้คุณทำ new เมธอดนี้ ส่วนเมธอด Method2() เป็นเมธอดแบบ virtual ที่คลาสลูก ให้คุณทำ override เมธอดนี้

VB 2005

```
Public Class Child
    Inherits Parent

    Public Overloads Sub Method1()
        MessageBox.Show("Method1 ของคลาสลูกทำงาน")
    End Sub

    Public Overrides Sub Method2()
        MessageBox.Show("Method2 ของคลาสลูกทำงาน")
    End Sub
End Class
```

VC# 2005

```
public class Child : Parent
{
    public new void Method1()
    {
        MessageBox.Show("Method1 ของคลาสลูกทำงาน");
    }

    public override void Method2()
    {
        MessageBox.Show("Method2 ของคลาสลูกทำงาน");
    }
}
```

NOTE

VC# 2005 คำสั่ง new ที่คุณนำมาใช้ในเมธอดเป็นการระบุว่าเมธอดนี้เป็นเมธอดที่ต้องการแก้ไขใหม่

ถึงจุดนี้เมธอดทั้ง 2 ระหว่างคลาสแม่กับคลาสลูกทำงานแตกต่างกันแล้ว การทดสอบคลาส Parent กับคลาส Child อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-5 การทำ Override เมธอดของคลาสแม่

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Parent()
        p.Method1()
        p.Method2()

        Dim c As New Child()
        c.Method1()
        c.Method2()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Parent p = new Parent();
    p.Method1();
    p.Method2();

    Child c = new Child();
    c.Method1();
    c.Method2();
}
```

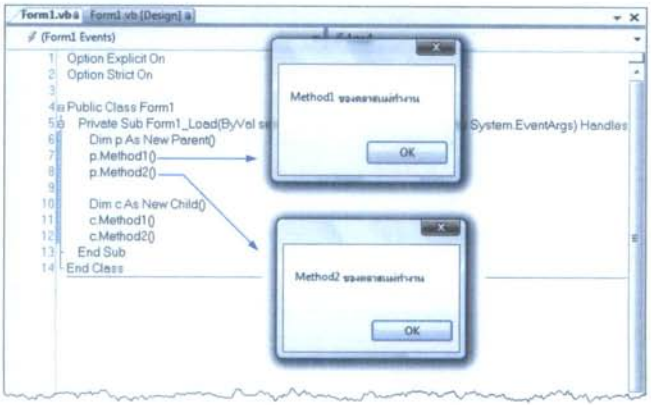
ผู้เขียนสร้างออบเจ็กต์ Parent ที่ชื่อว่า p ขึ้นมา สั่งให้เมธอดทั้ง 2 ทำงาน ดังรูปที่ 8-3

VB 2005

```
Dim p As New Parent()
p.Method1()
p.Method2()
```

VC# 2005

```
Parent p = new Parent();
p.Method1();
p.Method2();
```



รูปที่ 8-3 เมธอดของคลาสแม่ Parent ทำงาน

ข้อความของออบเจ็กต์ p ที่ปรากฏขึ้นมา เป็นข้อความที่เรากำหนดไว้ในคลาสแม่ ตีความได้ว่าเมธอดที่ถูกสั่งให้ทำงานคือ เมธอด Method1() และเมธอด Method2() ที่อยู่ในคลาสแม่

ต่อมาผู้เขียนสร้างออบเจ็กต์ Child ที่ชื่อว่า c ขึ้นมา แล้วสั่งให้เมธอดทั้ง 2 ทำงานเช่นกัน ดังรูปที่ 8-4

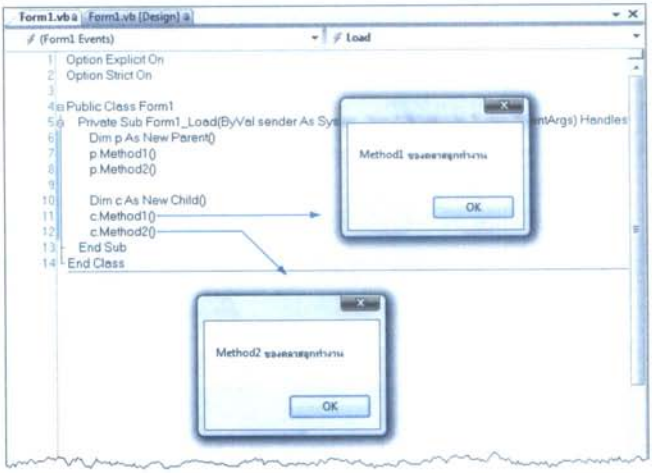
VB 2005

```
Dim c As New Child()
c.Method1()
c.Method2()
```

VC# 2005

```
Child c = new Child();
c.Method1();
c.Method2();
```

รูปที่ 8-4 เมธอดของคลาสลูก Child ทำงาน



จากรูปที่ 8-4 พบว่าข้อความที่ปรากฏขึ้นมาเป็นข้อความที่อยู่ในเมธอดของคลาสลูก ตีความได้ว่าเมธอดที่ถูกสั่งให้ทำงานคือ เมธอดทั้ง 2 ที่อยู่ในคลาสลูก

NOTE

โค้ด VC# 2005 ที่คลาสลูกคุณไม่สามารถทำ override กับเมธอด Method1() (เป็นเมธอดปกติ) ดังโค้ดต่อไปนี้

VC# 2005

```
public override void Method1()
    {
MessageBox.Show("Method1 ของคลาสลูกทำงาน");
    }
```

ข้อความที่แตกต่างกันระหว่างคลาสแม่กับคลาสลูกข้างต้น สื่อให้เห็นว่าถ้าเปลี่ยนจากข้อความเป็นการทำงานของโค้ดต่างๆ มากมายที่อยู่ในแต่ละเมธอด ผลที่ได้คือ คุณสามารถแก้ไขให้เมธอดของคลาสลูก (ที่ได้มาจากคลาสแม่) ทำงานแตกต่างไปจากคลาสแม่นั่นเองนี่คือ การสืบทอดคลาสที่มีลักษณะแก้ไขการทำงานของคลาสแม่

ผู้เขียนนำคลาส Rectangle และคลาส Cube มาแก้ไขใหม่ โดยที่คลาสลูก Cube เป็นคลาสที่สืบทอดมาจากคลาสแม่ Rectangle ถ้าเราคิดว่าการคำนวณหาพื้นที่ (เมธอด Calculate() ของคลาส Rectangle) ไม่ควรไปอยู่ในคลาส Cube เพราะว่าหน้าที่หลักของคลาส Cube ต้องคำนวณหาปริมาตร ไม่ใช่หาพื้นที่

ส่งผลให้ถ้าต้องการคำนวณหาพื้นที่ ให้ใช้เมธอด Calculate() ของคลาสแม่ Rectangle แต่ถ้าต้องการคำนวณหาปริมาตร ให้ใช้เมธอด Calculate() ของคลาสลูก Cube ตัวอย่างโค้ดหลังจากแก้ไข ให้ดูตัวอย่างที่ 8-6 การสืบทอดคลาสโดยการแก้ไขเมธอดของคลาสแม่

โค้ด VB 2005 และ VC# 2005 ที่ 8-6 การสืบทอดคลาสโดยการแก้ไขเมธอดของคลาสแม่

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Rectangle
    Protected _Width As Double = 0.0
    Protected _Height As Double = 0.0

    Public Property Width() As Double
        Get
            Return _Width
        End Get
        Set(ByVal value As Double)
            _Width = value
        End Set
    End Property

    Public Property Height() As Double
        Get
            Return _Height
        End Get
        Set(ByVal value As Double)
            _Height = value
        End Set
    End Property

    Public Function Calculate() As Double
        Return _Width * _Height
    End Function
End Class
```

VC# 2005 (Class1.cs)

```
public class Rectangle
{
    protected double _Width = 0.0;
    protected double _Height = 0.0;

    public double Width
    {
        get{return _Width;}
        set{_Width = value;}
    }

    public double Height
    {
        get{return _Height;}
        set{_Height = value;}
    }

    public double Calculate()
    {
        return _Width * _Height;
    }
}

public class Cube : Rectangle
{
    private double _Thick = 0.0;

    public double Thick
    {
```

```
Public Class Cube
    Inherits Rectangle

    Private _Thick As Double = 0.0

    Public Property Thick() As Double
        Get
            Return _Thick
        End Get
        Set(ByVal value As Double)
            _Thick = value
        End Set
    End Property

    Public Overloads Function Calculate() As Double
        Return _Width * _Height * _Thick
    End Function
End Class
```

```
        get{return _Thick;}
        set{_Thick = value;}
    }

    public new double Calculate()
    {
        return _Width * _Height * _Thick;
    }
}
```

โค้ด VB 2005 และ VC# 2005 ที่ 8-6 การสืบทอดคลาสโดยการแก้ไขเมธอดของคลาสแม่

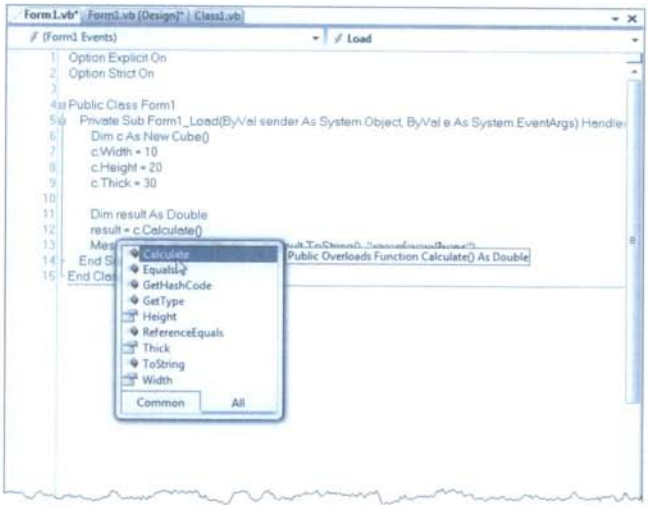
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim c As New Cube()
        c.Width = 10
        c.Height = 20
        c.Thick = 30

        Dim result As Double
        result = c.Calculate()
        MessageBox.Show("ปริมาตร : " & result.ToString(),
"ผลการคำนวณปริมาตร")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Cube c = new Cube();
        c.Width = 10;
        c.Height = 20;
        c.Thick = 30;

        double result;
        result = c.Calculate();
        MessageBox.Show("ปริมาตร : " + result.ToString(),
"ผลการคำนวณปริมาตร");
    }
```

รูปที่ 8-5
เมธอด Calculate() ของคลาสลูก Cube



เมธอด Calculate() ที่คุณเห็นในคลาสลูก Cube() เป็นเมธอดที่ได้มาจากคลาสแม่ Rectangle เพียงแต่ว่าเมธอด Calculate() เดิมของคลาสแม่ Rectangle ถูกบังไว้ ไม่ให้คลาสลูกเห็นนั่นเอง

ทำความรู้จักกับ Shadows

ใน VB 2005 นอกจากการบังเมธอดของคลาสแม่แล้ว ยังมีอีกวิธีหนึ่งนั่นคือ การทำ Shadows (ส่วนใน VC# 2005 ตรงกับคำสั่ง new ของหัวข้อที่แล้ว) ให้ดูตัวอย่างที่ 8-7 ทำความรู้จักกับ Shadows

โค้ด VB 2005 และ VC# 2005 ที่ 8-7 ทำความรู้จักกับ Shadows (new)

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Parent
    Protected _x As Integer = 5
    Protected _y As Integer = 10

    Public Sub Calculate()
        Dim _result As Integer = _x + _y
        MessageBox.Show(_result.ToString())
    End Sub
End Class

Public Class Child
    Inherits Parent
    Private _z As Integer = 15
```

VC# 2005 (Class1.cs)

```
public class Parent
{
    protected int _x = 5;
    protected int _y = 10;

    public void Calculate()
    {
        int _result = _x + _y;
        MessageBox.Show(_result.ToString());
    }
}

public class Child : Parent
{
    private int _z = 15;

    public new void Calculate()
```

```
    Public Shadows Sub Calculate()
        Dim _result As Integer = _x + _y + _z
        MessageBox.Show(_result.ToString())
    End Sub
End Class
```

```
    {
        int _result = _x + _y + _z;
        MessageBox.Show(_result.ToString());
    }
}
```

ผู้เขียนสร้างคลาสแม่ Parent ขึ้นมาประกอบด้วย 1 เมธอดแบบปกติ ชื่อว่า Calculate() ขอบเขตแบบ Public ทำหน้าที่หาผลรวมทั้งหมดของฟิลด์ _x และฟิลด์ _y กำหนดให้ทั้ง 2 ฟิลด์มีขอบเขตแบบ Protected กำหนดค่าเริ่มต้น 5 และ 10 ตามลำดับ

ส่วนที่คลาสลูก Child สืบทอดคลาสมาจากคลาสแม่ Parent เพิ่มฟิลด์ที่ชื่อว่า _z ขึ้นมา กำหนดค่าเริ่มต้น 15 ได้เมธอด Calculate() มาจากคลาสแม่ด้วย

แต่ว่าเมธอด Calculate() ที่ได้มาไม่สามารถหาผลรวมฟิลด์ทั้งหมดที่อยู่ในคลาสลูกได้ เพราะว่าคลาสลูกมีฟิลด์ _z เพิ่มขึ้นมาใหม่ จึงต้องแก้ไขเมธอด Calculate() โดยการทำ Shadows การใช้งานอยู่ใน Form1

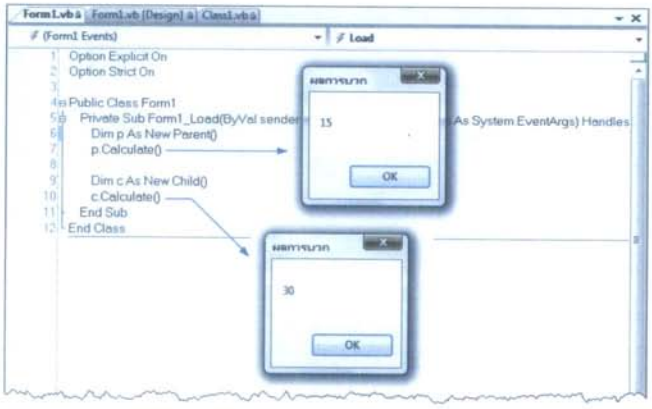
โค้ด VB 2005 ที่ 8-7 ทำความรู้จักกับ Shadows (new) (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Parent()
        p.Calculate()

        Dim c As New Child()
        c.Calculate()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 8-7 ทำความรู้จักกับ Shadows (new) (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Parent p = new Parent();
        p.Calculate();

        Child c = new Child();
        c.Calculate();
    }
```

รูปที่ 8-6
ผลการรัน
ตัวอย่างที่ 8-7



จากรูปที่ 8-6 เห็นได้ว่าเมธอด Calculate() ของคลาสแม่และคลาสลูกทำงานแตกต่างกัน กล่าวคือ เมธอด Calculate() ของคลาสแม่ Parent หาผลรวมของฟิลด์ _x และฟิลด์ _y ส่วนเมธอด Calculate() ของคลาสลูก Child หาผลรวมของฟิลด์ _x, _y และฟิลด์ _z

การใช้งานฟิลด์ _x และฟิลด์ _y มีขอบเขตแบบ Protected หมายถึง ภายนอกมองไม่เห็น แต่ถ่ายทอดไปยังคลาสลูกได้โดยการสืบทอดคลาส จึงทำให้การหาผลรวมของคลาสลูก Child เกิดจากฟิลด์ _x, _y และฟิลด์ _z นั่นเอง ส่วนคำว่าภายนอกมองไม่เห็น หมายถึง คุณไม่สามารถกำหนดค่าให้ฟิลด์ _x และฟิลด์ _y จากภายนอกได้ ดังโค้ดต่อไปนี้

VB 2005	VC# 2005

Dim p As New Parent()
p._x = 100
p._y = 200
	Parent p = new Parent();
	p._x = 100;
	p._y = 200;

ทำความรู้จักกับ MyBase (VB 2005) และ base (VC# 2005)

จากตัวอย่างการสืบทอดคลาสที่ผ่านมา พบว่า OOP เปิดโอกาสให้คลาสลูกทำงานแตกต่างไปจากคลาสแม่ได้ หรือที่เรียกว่าการทำ Data Member Hiding

คำถามที่ตามมาคือ ถ้าคลาสลูกทำงานแตกต่างจากคลาสแม่ มีหนทางใดบ้างที่สามารถสั่งให้เกิดการทำงานของคลาสแม่จากคลาสลูก คำตอบคือ การใช้งานคีย์เวิร์ด MyBase (VB 2005) หรือ base (VC# 2005) นั่นเอง ให้ดูตัวอย่างที่ 8-8 ทำความรู้จักกับ MyBase (VB 2005) และ base (VC# 2005) เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 8-8 ทำความรู้จักกับ MyBase (VB 2005) และ base (VC# 2005)

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Parent
    Protected _x As Integer = 5
    Protected _y As Integer = 10


    Public Sub Method1()
        MessageBox.Show("เมธอดแม่ทำงาน")
    End Sub
End Class

Public Class Child
    Inherits Parent

    Private Shadows _x As Integer = 50

    Public Sub Calculate()
        Dim _result As Integer = _x + _y
        MessageBox.Show(_result.ToString())
    End Sub

    Public Shadows Sub Method1()
        MyBase.Method1()
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Parent
{
    protected int _x = 5;
    protected int _y = 10;

    public void Method1()
    {
        MessageBox.Show("เมธอดแม่ทำงาน");
    }
}

public class Child : Parent
{
    private new int _x = 50;

    public void Calculate()
    {
        int _result = _x + _y;
        MessageBox.Show(_result.ToString());
    }

    public new void Method1()
    {
        base.Method1();
    }
}
```

วิธีการดูโค้ดชุดนี้คือ ที่คลาสแม่ Parent ประกอบด้วยฟิลด์ _x และฟิลด์ _y มีขอบเขตแบบ Protected กำหนดค่าเริ่มต้น 5 และ 10 ตามลำดับ และมีเมธอดที่ชื่อว่า Method1() มีขอบเขตแบบ Public

VB 2005

```
Public Class Parent
    Protected _x As Integer = 5
    Protected _y As Integer = 10

    Public Sub Method1()
        MessageBox.Show("เมธอดแม่ทำงาน")
    End Sub
End Class
```

VC# 2005

```
public class Parent
{
    protected int _x = 5;
    protected int _y = 10;

    public void Method1()
    {
        MessageBox.Show("เมธอดแม่ทำงาน");
    }
}
```

ส่วนที่คลาสลูก Child กำหนดให้สืบทอดมาจากคลาสแม่ Parent สิ่งที่ได้มาจากคลาสแม่คือ ฟิลด์ _x (มีค่า 5) และฟิลด์ _y (มีค่า 10) และเมธอดที่ชื่อว่า Method1()

สมมติว่าค่าของฟิลด์ _x ที่ได้จากคลาสแม่มีค่าน้อยเกินไป ทำให้นำมาใช้ในคลาสลูกไม่ได้ จึงเปลี่ยนค่าฟิลด์ _x จาก 5 เป็น 50 โดยการทำ Shadows (VB 2005) หรือคำสั่ง new (VC# 2005)

VB 2005

```
Public Class Child
    Inherits Parent

    Private Shadows _x As Integer = 50

    Public Sub Calculate()
        Dim _result As Integer = _x + _y
        MessageBox.Show(_result.ToString())
    End Sub

    Public Shadows Sub Method1()
        MyBase.Method1()
    End Sub
End Class
```

VC# 2005

```
public class Child : Parent
{
    private new int _x = 50;

    public void Calculate()
    {
        int _result = _x + _y;
        MessageBox.Show(_result.ToString());
    }

    public new void Method1()
    {
        base.Method1();
    }
}
```

NOTE

ถ้ามองโค้ดชุดนี้อีกนัยหนึ่งคือ การที่คุณสร้างฟิลด์ _x ขึ้นมาใหม่ในคลาสลูกอีกฟิลด์หนึ่ง ซ้ำกับฟิลด์ _x ที่ได้มาจากคลาสแม่ เพราะว่าฟิลด์ _x มีขอบเขตแบบ Protected นั่นเอง ซึ่งก่อให้เกิดความสับสนแน่นอน เพราะที่คลาสลูกจะมีฟิลด์ที่ชื่อว่า _x จำนวน 2 ตัว ส่งผลให้โค้ดต่อไปนี้รันไม่ผ่าน

VB 2005

```
Public Class Child
    Inherits Parent

    Private _x As Integer = 50

End Class
```

VC# 2005

```
public class Child : Parent
{
    private int _x = 50;

}
```

ที่คลาสลูก Child เพิ่มเมธอด Calculate() ใหม่ขึ้นมา ทำหน้าที่หาผลรวมของฟิลด์ _x และฟิลด์ _y

VB 2005

```
Public Sub Calculate()
    Dim _result As Integer = _x + _y
    MessageBox.Show(_result.ToString())
End Sub
```

VC# 2005

```
public void Calculate()
{
    int _result = _x + _y;
    MessageBox.Show(_result.ToString());
}
```

อาจจะใช้คำสั่ง Me (VB 2005) หรือคำสั่ง this (VC# 2005) กำกับฟิลด์ _x ไว้ก็ได้ ดังโค้ดต่อไปนี้

VB 2005

```
Public Sub Calculate()
    Dim _result As Integer = Me._x + _y
    MessageBox.Show(_result.ToString())
End Sub
```

VC# 2005

```
public void Calculate()
{
    int _result = this._x + _y;
    MessageBox.Show(_result.ToString());
}
```

ส่วนเมธอด Method1() ที่ได้มาจากคลาสแม่ กำหนดให้แก้ไขเมธอดนี้ใหม่ด้วยคำสั่ง Shadows (VB 2005) หรือคำสั่ง new (VC# 2005) สมมติว่ามีความจำเป็นต้องสั่งให้เมธอด Method1() ของคลาสแม่ทำงานด้วย

ด้วยความจำเป็นข้างต้นให้ใช้คำสั่ง MyBase (VB 2005) หรือคำสั่ง base เพื่อสั่งให้การทำงานของคลาสแม่เกิดขึ้นในคลาสลูกด้วย ในกรณีนี้คือเมธอด Method1() ของคลาสแม่

VB 2005

```
Public Shadows Sub Method1()
    MyBase.Method1()
End Sub
```

VC# 2005

```
public new void Method1()
{
    base.Method1();
}
```

การใช้งานคลาส Child อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-8 ทำความรู้จักกับ MyBase (VB 2005) และ base (VC# 2005)

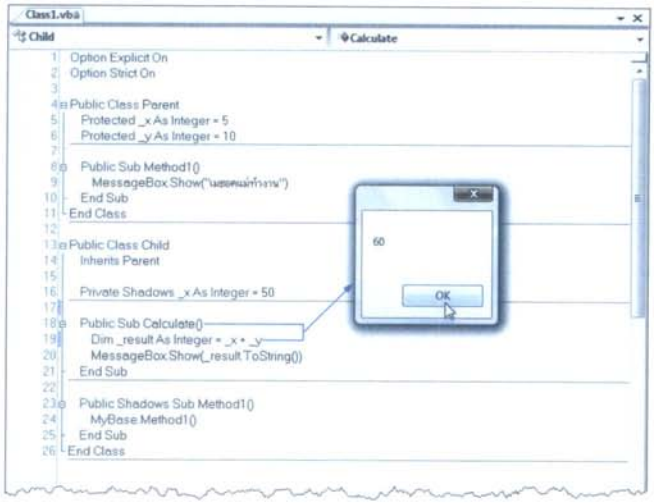
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim c As New Child()
        c.Calculate()
        c.Method1()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

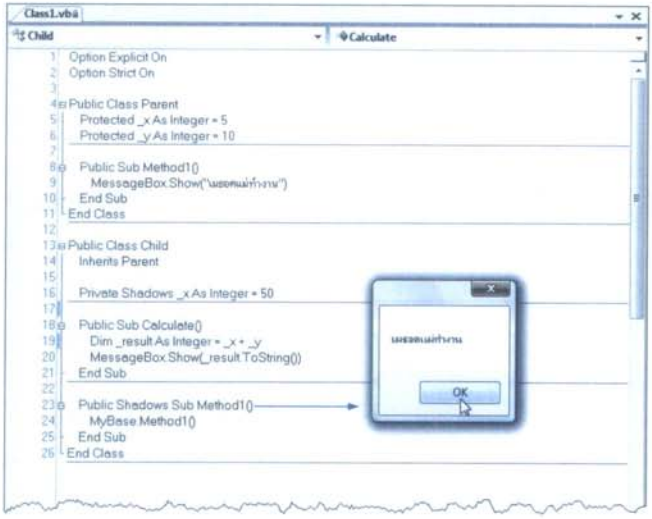
```
private void Form1_Load(object sender, EventArgs e)
{
    Child c = new Child();
    c.Calculate();
    c.Method1();
}
```

รูปที่ 8-7
ผลการทำงานของเมธอด Calculate() ของคลาสลูก Child



จากรูปที่ 8-7 แม้ว่าค่าเริ่มต้นของฟิลด์ _x (ที่อยู่ในคลาสแม่) คือ 5 แต่ในคลาสลูกฟิลด์ _x ถูกแก้ไขค่าใหม่เป็น 50 ผลที่ได้คือ 50+10 นั่นเอง

รูปที่ 8-8
ผลการทำงานของเมธอด Method1() ของคลาสลูก Child



ส่วนเมธอด Method1() ที่อยู่ในคลาสลูก ผลจากการใช้คำสั่ง MyBase (base) จึงทำให้เมธอด Method1() ของคลาสแม่ทำงานในคลาสลูกด้วย คำถามที่ตามมาก็คือ ถ้าเราต้องการใช้ค่าของฟิลด์ _x ที่อยู่ในคลาสแม่ทำอย่างไร

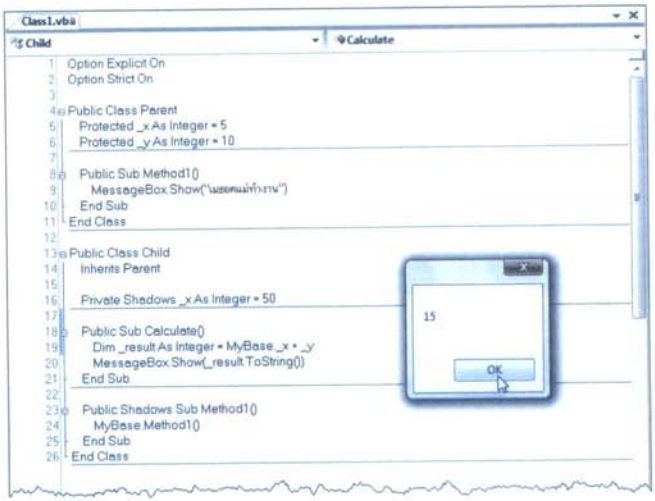
VB 2005

```
Public Class Child
    Inherits Parent

    Private Shadows _x As Integer = 50

    Public Sub Calculate()
        Dim _result As Integer = MyBase._x + _y
        MessageBox.Show(_result.ToString())
    End Sub
End Class
```

VC# 2005

```
public class Child : Parent
{
    private new int _x = 50;

    public void Calculate()
    {
        int _result = base._x + _y;
        MessageBox.Show(_result.ToString());
    }
}
```

รูปที่ 8-9 กรณีใช้ค่าฟิลด์ _x ที่อยู่ในคลาสแม่ Parent



จากโค้ดข้างต้นแม้ว่าค่าของฟิลด์ _x ที่อยู่ในคลาสลูก Child ถูกแก้ไขเป็น 50 แต่เราต้องการใช้ค่าฟิลด์ _x ที่อยู่ในคลาสแม่ จึงระบุคำสั่ง MyBase._x (base._x) ผลรวมที่ได้เกิดจาก 5+10 นั่นเองและเพื่อไม่ให้เกิดความสับสน ผู้เขียนขอสรุปตารางแสดงการเปรียบเทียบคำสั่งระหว่าง VB 2005 กับ VC# 2005 เฉพาะที่เกี่ยวข้องกับการสืบทอดคลาส ดังนี้

VB 2005	VC# 2005
Overloads	-
Shadows	new
Overrides	override
Overridable	virtual

ทำความรู้จักกับ Polymorphism

กฎหลักข้อสุดท้ายของ OOP คือ Polymorphism (โพลี่-มอ-ฟี-ซึ่ม) สำหรับกฎข้อนี้เป็นเรื่องของการมองออบเจ็กต์ในฐานะต่างๆ ออบเจ็กต์แต่ละตัวมีหลายมุมมอง เช่น ถ้าเรามองสุนัขในโลกของ OOP แล้วถามว่าคืออะไร

คำตอบที่ได้คือ สัตว์เลี้ยงประจำบ้าน, สัตว์สี่ขา, สัตว์บก, สัตว์ออกลูกเป็นตัว เป็นต้น เห็นได้ว่าเรามองสุนัขได้หลายมุมมอง ถ้าสุนัขตัวนี้เห่าต่อหน้านักวิทยาศาสตร์ (แสดงพฤติกรรมออกมา หรือเมธอดเห่าเริ่มทำงาน) มีความหมายแตกต่างจากการเห่าต่อหน้าคนทั่วไป เราอาจจะมองว่าสุนัขเห่าน่ารักดี แต่นักวิทยาศาสตร์อาจจะมองว่าการเห่าดังกล่าวกำลังแสดงหรือสื่ออะไรอยู่

ในโลกของ OOP ก็เช่นกัน คลาสต้นแบบที่คุณสร้างขึ้นมาถูกนำไปสร้างเป็นออบเจ็กต์ต่างๆ ตามหน้าที่หลักของมัน เช่น สร้างคลาส Customer ขึ้นมาเพื่อเก็บข้อมูลลูกค้า ออบเจ็กต์ Customer ที่ได้ถูกมองในฐานะเป็นอินสแตนซ์ของคลาส Customer

ออบเจ็กต์ Customer ที่ได้มาอาจจะถูกมองในแง่ของคนธรรมดา Person ก็ได้ เมื่อมอง Customer ในฐานะ Person แล้วจะเป็นอย่างไร นี่คือสิ่งที่เราต้องศึกษา ให้ดูตัวอย่างที่ 8-9 ทำความรู้จักกับ Polymorphism

โค้ด VB 2005 และ VC# 2005 ที่ 8-9 ทำความรู้จักกับ Polymorphism

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Public Sub Speak()
    MessageBox.Show("Person พูด")
    End Sub

    Public Sub Walk()
        MessageBox.Show("Person เดิน")
    End Sub
End Class

Public Class Programmer
    Inherits Person


    Public Overloads Sub Speak()
        MessageBox.Show("Programmer พูด")
    End Sub

    Public Overloads Sub Walk()
        MessageBox.Show("Programmer เดิน")
    End Sub
```

VC# 2005 (Class1.cs)

```
public class Person
{
    public void Speak()
    {
        MessageBox.Show("Person พูด");
    }

    public void Walk()
    {
        MessageBox.Show("Person เดิน");
    }
}

public class Programmer : Person
{
    public new void Speak()
    {
        MessageBox.Show("Programmer พูด");
    }

    public new void Walk()
    {
        MessageBox.Show("Programmer เดิน");
```

```
    Public Sub Programming()
        MessageBox.Show("Programmer เขียนโปรแกรมได้")
    End Sub
End Class
```

```
    }

    public void Programming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมได้");
    }
}
```

สร้างคลาสแม่ที่ชื่อว่า Person ประกอบด้วย 2 เมธอดแบบปกติ ชื่อว่าเมธอด Speak() และ Walk() สร้างคลาสลูกชื่อว่า Programmer สืบทอดคลาสมาจากคลาส Person กำหนดให้แก้ไขเมธอด Speak() และเมธอด Walk() ที่ได้จากคลาสแม่ใหม่ และเพิ่มเมธอดใหม่ชื่อว่า Programming() การใช้งานคลาสทั้ง 2 อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-9 ทำความรู้จักกับ Polymorphism

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p1 As Programmer = New Programmer()
        p1.Speak()
        p1.Walk()
        p1.Programming()

        Dim p2 As Person = New Programmer()
        p2.Speak()
        p2.Walk()
        p2.Programming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer p1 = new Programmer();
    p1.Speak();
    p1.Walk();
    p1.Programming();

    Person p2 = new Programmer();
    p2.Speak();
    p2.Walk();
    //p2.Programming();
}
```

วิธีการดูโค้ดชุดนี้คือ ผู้เขียนสร้างออบเจ็กต์ Programmer ที่ชื่อว่า p1 ส่งผลให้เราใช้เมธอดของออบเจ็กต์ Programmer ได้ตามปกติคือ เมธอด Speak(), Walk() และเมธอด Programming() ดังรูปที่ 8-10

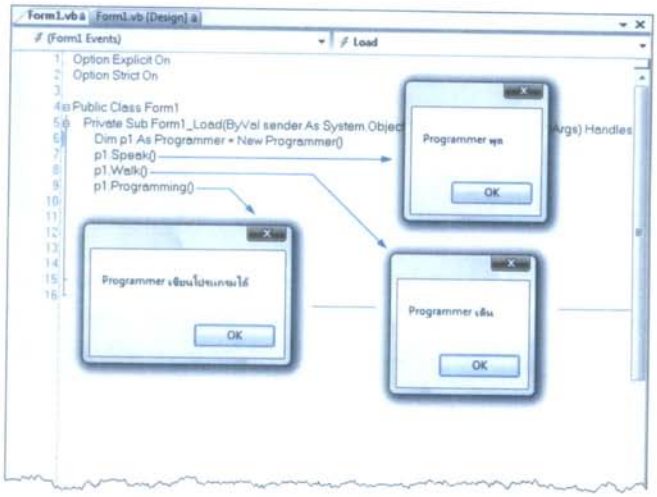
VB 2005

```
Dim p1 As Programmer = New Programmer()
p1.Speak()
p1.Walk()
p1.Programming()
```

VC# 2005

```
Programmer p1 = new Programmer();
p1.Speak();
p1.Walk();
p1.Programming();
```

รูปที่ 8-10
กรณีสร้างออบเจ็กต์ Programmer ตามปกติ



เราจะมองโค้ดให้ละเอียดยิ่งขึ้น พบว่าเราสร้างตัวแปรชื่อว่า p1 ให้มี Type เป็น Programmer และสั่งให้ชี้ไปยังออบเจ็กต์ Programmer ที่เกิดขึ้นมาด้วยคำสั่ง New (new) ส่งผลให้ออบเจ็กต์ Programmer ถูกมองในฐานะเป็น Programmer Type เป็นสิ่งที่เราคุ้นเคยกันเป็นอย่างดี นั่นคือ Type และออบเจ็กต์ที่เกิดเป็นชนิดเดียวกัน

ต่อมาสร้างตัวแปรที่ชื่อว่า p2 มี Type เป็น Person (คลาสแม่) แต่ชี้ไปยังออบเจ็กต์ Programmer (คลาสลูก) ดังรูปที่ 8-11

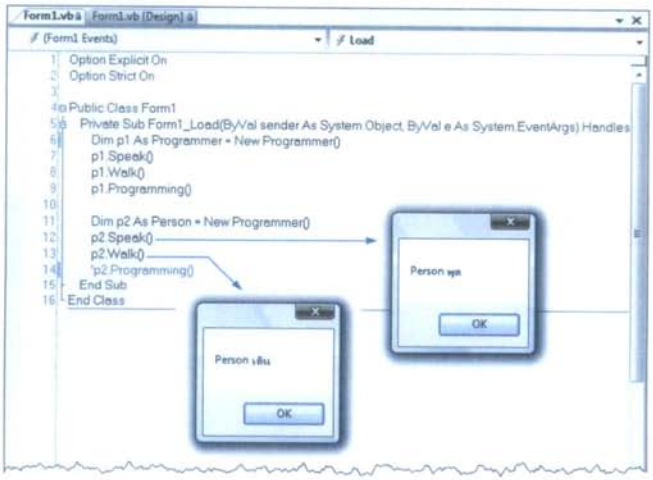
VB 2005

```
Dim p2 As Person = New Programmer()
p2.Speak()
p2.Walk()
```

VC# 2005

```
Person p2 = new Programmer();
p2.Speak();
p2.Walk();
```

รูปที่ 8-11
กรณีสร้างออบเจ็กต์ Programmer แต่มี Type เป็น Person



จากโค้ดข้างต้นคุณต้องมองออบเจ็กต์ Programmer ที่ชื่อว่า p2 ในฐานะ Person ส่งผลให้ความเป็น Programmer ต้องหมดไปอย่างสิ้นเชิง คุณต้องมอง p2 ในฐานะ Person เท่านั้น คุณจึงใช้ได้แต่เมธอด Speak() และเมธอด Walk() ของ Person

เหตุผลที่คุณไม่สามารถใช้เมธอด Programming() ภายใต้ออบเจ็กต์ p2 เพราะว่าคุณต้องมองออบเจ็กต์ p2 ในฐานะ Programmer ก่อน จึงจะใช้เมธอดนี้ได้ (ดังตัวอย่างโค้ดของออบเจ็กต์ p1 นั่นเอง) นี่คือพื้นฐานการมองออบเจ็กต์แบบหลายมุมมอง

เมธอดแบบ Overridable (virtual) กับ Polymorphism

การมองออบเจ็กต์หลายมุมมอง หลายฐานะระหว่างคลาสแม่กับคลาสลูก ก่อให้เกิดขอบเขตการใช้งานเมธอดที่อยู่ในคลาส ฐานะที่มองเป็นตัวกำหนดขอบเขตออบเจ็กต์ที่เกิดขึ้นมา ดังเช่นตัวอย่างที่ผ่านมา ความเป็น Programmer หายไปอย่างสิ้นเชิง เมื่อมองในฐานะของ Person แต่กฎเกณฑ์นี้มีข้อยกเว้นเช่นกัน ให้ดูตัวอย่างที่ 8-10 เมธอดแบบ Overridable (virtual) กับ Polymorphism เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 8-10 เมธอดแบบ Overridable (virtual) กับ Polymorphism

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Public Overridable Sub Speak()
        MessageBox.Show("Person พูดได้")
    End Sub
End Class

Public Class American
    Inherits Person

    Public Shadows Sub Speak()
        MessageBox.Show("American พูด")
    End Sub
End Class

Public Class Korean
    Inherits Person

    Public Overrides Sub Speak()
        MessageBox.Show("Korean พูด")
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Person
{
    public virtual void Speak()
    {
        MessageBox.Show("Person พูดได้");
    }
}

public class American : Person
{
    public new void Speak()
    {
        MessageBox.Show("American พูด");
    }
}

public class Korean : Person
{
    public override void Speak()
    {
        MessageBox.Show("Korean พูด");
    }
}
```

วิธีการดูโค้ดชุดนี้คือ ผู้เขียนสร้างคลาสแม่ขึ้นมาเป็นคลาสคนธรรมดาชื่อว่า Person มี 1 เมธอดชื่อว่า Speak() หมายถึง คนพูดได้เป็นเมธอดแบบ Overridable (virtual) ซึ่งหมายถึง เป็นเมธอดที่ยอมให้แก้ไขได้

VB 2005

```
Public Class Person
    Public Overridable Sub Speak()
        MessageBox.Show("Person พูดได้")
    End Sub
End Class
```

VC# 2005

```
public class Person
{
    public virtual void Speak()
    {
        MessageBox.Show("Person พูดได้");
    }
}
```

จากนั้นสร้างคลาสลูกเป็นฝรั่งชื่อว่า American แก้ไขเมธอด Speak() ที่ได้จากคลาสแม่ใหม่ โดยการทำ Shadows (new) เพื่อให้เมธอด Speak() ของคลาส American ทำงานแตกต่างจากเมธอด Speak() ของคลาสแม่ Person

VB 2005

```
Public Class American
    Inherits Person

    Public Shadows Sub Speak()
        MessageBox.Show("American พูด")
    End Sub
End Class
```

VC# 2005

```
public class American : Person
{
    public new void Speak()
    {
        MessageBox.Show("American พูด");
    }
}
```

ต่อมาสร้างคลาสลูกอีก 1 คลาสเป็นคนเกาหลี แก้ไขเมธอด Speak() เช่นกัน แต่แก้ไขโดยการทำ Overrides (override) เพื่อให้เมธอด Speak() ของคลาส Korean ทำงานแตกต่างจากเมธอด Speak() ของคลาสแม่ Person เช่นกัน

VB 2005

```
Public Class Korean
    Inherits Person

    Public Overrides Sub Speak()
        MessageBox.Show("Korean พูด")
    End Sub
End Class
```

VC# 2005

```
public class Korean : Person
{
    public override void Speak()
    {
        MessageBox.Show("Korean พูด");
    }
}
```

การใช้งานคลาสทั้ง 3 อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 8-10 เมธอดแบบ Overridable (virtual) กับ Polymorphism

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim usa As Person = New American()
        usa.Speak()

        Dim kr As Person = New Korean()
        kr.Speak()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Person usa = new American();
    usa.Speak();

    Person kr = new Korean();
    kr.Speak();
}
```

โค้ดช่วงแรกมีฝรั่ง (American) ชื่อว่า usa แต่มองฝรั่งในฐานะคนธรรมดา (Person) ผลที่ได้คือ ฝรั่งพูดได้ (หมายถึง เมธอด Speak() ของ Person ทำงาน) ไม่ใช่พูดภาษาฝรั่งแต่อย่างใด (หมายถึง เมธอด Speak() ของ American ไม่ถูกสั่งให้ทำงาน) เห็นได้ว่าเป็นขอบเขตการมองออบเจ็กต์หลายมุมมองตามปกติ

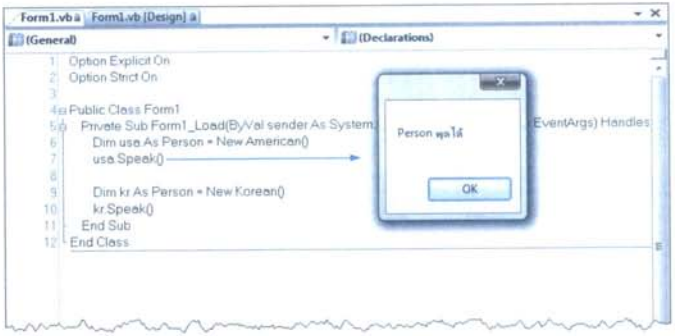
VB 2005

```
Dim usa As Person = New American()
usa.Speak()
```

VC# 2005

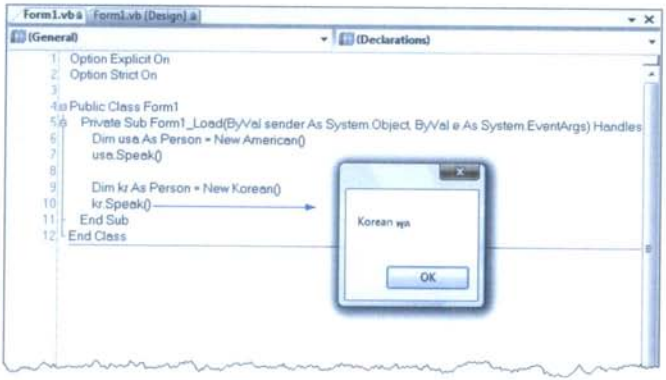
```
Person usa = new American();
usa.Speak();
```

รูปที่ 8-12 การทำงานของเมธอด Speak() ของกรณีแรก



โค้ดช่วงที่ 2 มีคนเกาหลี (Korean) ชื่อว่า kr มองในฐานะคนธรรมดา (Person) เช่นกัน แต่คนเกาหลีกลับพูดภาษาเกาหลีได้ (เมธอด Speak() ของ Korean ทำงาน) ทั้งๆ ที่มองคนเกาหลีในฐานะคนธรรมดา ดังรูปที่ 8-13

VB 2005	VC# 2005
Dim kr As Person = New Korean() kr.Speak()	Person kr = new Korean(); kr.Speak();



รูปที่ 8-13 การทำงานของเมธอด Speak() ของกรณีที่ 2

นี่คือข้อแตกต่างของการแก้ไขเมธอดแบบ Shadows (new) กับ Overrides (override) จากเมธอดแบบ Overridable (virtual) ของคลาสแม่ในมุมของ Polymorphism

เพราะว่าคลาส American แก้ไขเมธอด Speak() โดยการทำ Shadows (new) ส่วนคลาส Korean แก้ไขแบบ Overrides (override) ผลที่ได้แตกต่างกัน

การสืบทอดคลาสข้ามภาษา

จากที่ผู้เขียนกล่าวไว้ในตอนต้นของเนื้อหา Type ใน .NET แล้วว่า ทุกสิ่งทุกอย่างที่อยู่ใน .NET เป็นออบเจ็กต์ และมี Type เป็นของตัวเองอย่างชัดเจน โดยที่ใน CLR มี Type มาตรฐานกลางเพื่อให้ภาษาต่างๆ ที่สนับสนุน .NET นำไปอ้างอิง Type ได้อย่างถูกต้องตรงกัน แม้ว่าไวยากรณ์ของแต่ละภาษาจะแตกต่างกันก็ตาม

ด้วยเหตุนี้เองการสร้างคลาสต้นแบบขึ้นมาใช้งานในยุคของ .NET จึงมีความสมบูรณ์ในแง่ของความเข้ากันได้ระหว่างภาษาเป็นอย่างมาก ภาษาที่สนับสนุน .NET สามารถทำความรู้จักกัน เปรียบเสมือนกับเป็นเนื้อเดียวกัน

วิธีการหนึ่งที่ผู้เขียนทดสอบแบบง่ายๆ ก็คือ สร้างคลาสแม่จากภาษาหนึ่ง แล้วใช้อีกภาษาหนึ่งสืบทอดคลาสจากคลาสแม่ดังกล่าวนำมาสร้างคลาสลูก ให้ดูตัวอย่างที่ 8-11 การสืบทอดคลาสข้ามภาษา

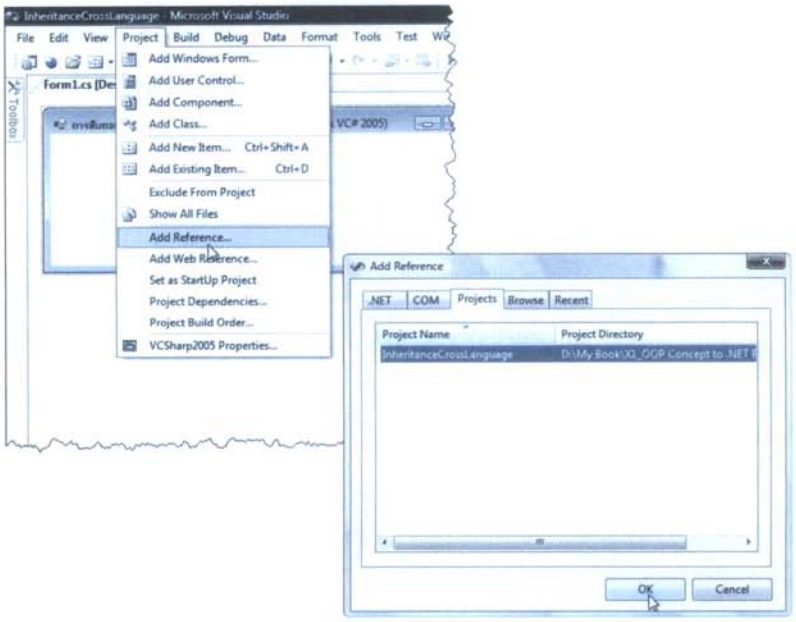
โค้ด VB 2005 ที่ 8-11 การสืบทอดคลาสข้ามภาษา (VBClass.vb)

```
Option Explicit On
Option Strict On

Public Class VBParent
    Public Sub VB2005Method(ByVal str As String)
        MessageBox.Show("เมธอดจากคลาสแม่ของ VB 2005 " & str)
    End Sub
End Class
```

ผู้เขียนสร้างคลาสแม่ที่ชื่อว่า VBParent มี 1 เมธอดชื่อว่า VB2005Method() ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น String เป็นโปรเจ็กต์ของ VB 2005

ส่วนที่คลาสลูกเป็นโปรเจ็กต์ของ VC# 2005 ให้คลิกเมนู Project > Add Reference... คลิกแท็บ Projects เพื่ออ้างอิงคลาส VBParent ของโปรเจ็กต์ VB 2005 ดังรูปที่ 8-14

รูปที่ 8-14 การอ้างอิงโปรเจ็กต์ VB 2005



ตั้งชื่อคลาสลูกว่า CSChild สืบทอดคลาสจากคลาสแม่ VBParent ดังโค้ดต่อไปนี้

โค้ด VC# 2005 ที่ 8-11 การสืบทอดคลาสข้ามภาษา (CSClass.cs)

```
public class CSChild:VBParent
{

}
```

ส่วนการใช้งานอยู่ใน Form1.cs ดังนี้

โค้ด VC# 2005 ที่ 8-11 การสืบทอดคลาสข้ามภาษา (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    CSChild cs = new CSChild();
    cs.VB2005Method("พารามิเตอร์จาก VC# 2005");
}
```

รูปที่ 8-15 ผลการรันตัวอย่างที่ 8-11

จากรูปที่ 8-15 พบว่าที่คลาสลูก CSChild ของโปรเจ็กต์ VC# 2005 ซึ่งสืบทอดมาจากคลาสแม่ VBParent สามารถเรียกใช้งานเมธอด VB2005Method() ได้ตามปกติ เหมือนกับว่าเกิดจากภาษาเดียวกัน ทำให้ผู้พัฒนาแอพพลิเคชันด้วยภาษาที่สนับสนุน .NET ไม่ต้องถามคำถามที่ว่า คลาสของคุณสร้างจากภาษาอะไร

การห้ามสืบทอดคลาส

ในกรณีที่คุณไม่ต้องการให้คลาสต้นแบบของคุณถูกสืบทอดคลาส คุณสามารถใช้คำสั่ง NotInheritable (VB 2005) หรือ sealed (VC# 2005) กำกับไว้ที่คลาสต้นแบบของคุณได้เช่นกัน ดังตัวอย่างที่ 8-12 การห้ามสืบทอดคลาส

โค้ด VB 2005 และ VC# 2005 ที่ 8-12 การห้ามสืบทอดคลาส

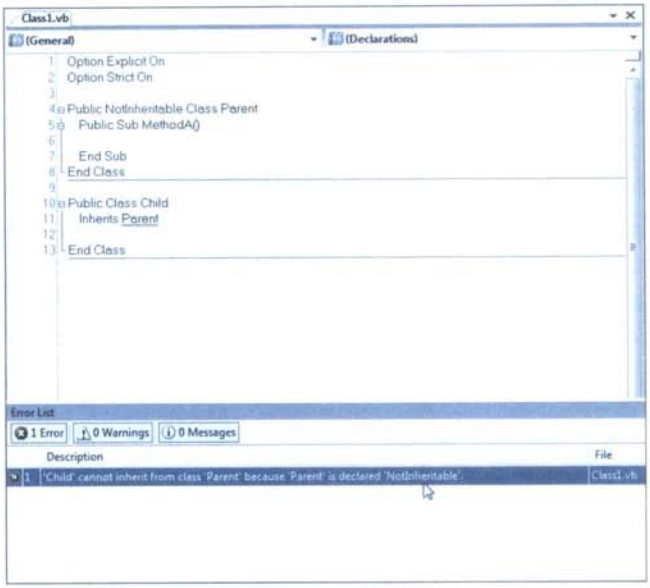
VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public NotInheritable Class Parent
    Public Sub MethodA()

    End Sub
End Class
-----------------------------
Public Class Child
    Inherits Parent

End Class
```

VC# 2005 (Class1.cs)

```
public sealed class Parent
{
    public void MethodA()
    {

    }
}
-----------------------------
public class Child : Parent
{

}
```

รูปที่ 8-16 ข้อผิดพลาดของตัวอย่างที่ 8-12

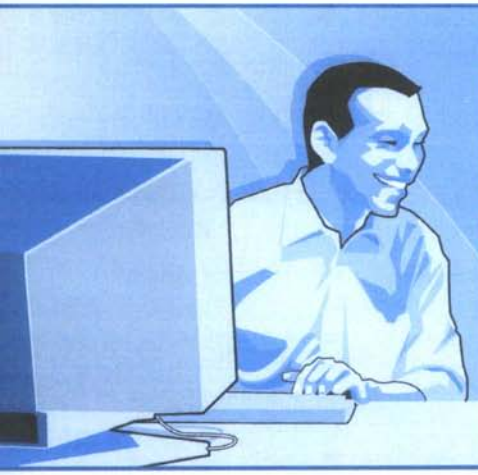


จากรูปที่ 8-16 เป็นข้อผิดพลาดที่เกิดขึ้นเมื่อคุณต้องการสืบทอดคลาส จากคลาสต้นแบบที่มีการใช้คำสั่ง NotInheritable (VB 2005) หรือ sealed (VC# 2005) ไว้

สรุปท้ายบท

รายละเอียดของการสืบทอดคลาสและ Polymorphism ค่อนข้างเยอะพอสมควร แต่กติกาดังกล่าวก็ช่วยให้คุณรู้จักการอยู่ร่วมกันระหว่างคลาสแม่กับคลาสลูก ต่อเนื่องไปจนถึงการนำไปใช้งานนั่นเอง


Programming in OOP style
Advanced .net



Advance Visual Studio 2005

CHAPTER ▸ 9

Constructor

บทนำ

หลังจากที่คุณได้ศึกษาวิธีการสร้างและใช้งานออบเจ็กต์มาบ้างแล้ว คุณเคยสงสัยหรือไม่ว่าแล้วจุดกำเนิดของออบเจ็กต์คืออะไร มีอะไรน่าสนใจบ้าง ซึ่งเป็นเนื้อหาของบทนี้นั่นเอง

ทำความรู้จักกับ Constructor และการระบุสมาชิกด้วยคำสั่ง Me (VB 2005) หรือ this (VC# 2005)

Constructor (คอนสตรัคเตอร์) เป็นเมธอดพิเศษที่ทำงานทันทีเมื่อมีการสร้างอินสแตนซ์ (Instance) ของคลาสขึ้นมา การสร้างอินสแตนซ์ใน VB 2005 ใช้คำสั่ง New ส่วน VC# 2005 ใช้คำสั่ง new ซึ่งเป็นคำสั่งที่เราคุณเคยกันเป็นอย่างดี เวลาที่เราต้องการใช้งานออบเจ็กต์บางตัวของ .NET เช่น ออบเจ็กต์ DataSet เป็นต้น

VB 2005	VC# 2005

Public Sub New()
 	
End Sub
	public Customer()
	{
	
	}

ใน VB 2005 การสร้างคอนสตรัคเตอร์จะใช้ซับรูทีนแบบ Public ที่ชื่อว่า New() ส่วน VC# 2005 จะใช้เมธอดที่มีชื่อเดียวกับคลาส เช่น ถ้าเราสร้างคลาสที่ชื่อว่า Customer ส่งผลให้คอนสตรัคเตอร์ของคลาส Customer จะชื่อว่า Customer() มีขอบเขตเป็นแบบ Public เช่นกัน

คอนสตรัคเตอร์เป็นเมธอดพิเศษที่ทำงานทันทีโดยอัตโนมัติ เมื่อมีการสร้างอินสแตนซ์ของคลาสขึ้นมา คุณสามารถใช้ประโยชน์ตรงจุดนี้ได้หลายอย่าง เช่น นำมาใช้กำหนดค่าเริ่มต้นให้กับฟิลด์ต่างๆ ภายในคลาสก็ได้ เป็นต้น

ในคลาส 1 คลาส คุณสามารถกำหนดให้มีคอนสตรัคเตอร์ได้มากกว่า 1 ชุด โดยที่เราสามารถแบ่งคอนสตรัคเตอร์ได้ 2 ลักษณะคือ

- คอนสตรัคเตอร์แบบปกติกล่าวคือ ไม่มีการรับค่าพารามิเตอร์ใดๆ เข้ามา ถือเป็นคอนสตรัคเตอร์แบบปกติเรียกว่า Default Constructor
- คอนสตรัคเตอร์แบบรับค่าพารามิเตอร์กล่าวคือ เป็นการอำนวยความสะดวกให้กับผู้ใช้ เวลาที่นำคลาส ของคุณไปใช้งาน สามารถส่งค่าพารามิเตอร์ต่างๆ เข้ามาในขณะที่สร้างอินสแตนซ์ได้โดยตรง

VB 2005

```
Public Sub New()

End Sub

Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
    Me._FirstName = FirstName
    Me._LastName = LastName
End Sub
```

VC# 2005

```
public Customer()
{

}

public Customer(string FirstName, string LastName)
{
this._FirstName = FirstName;
this._LastName = LastName;
}
```

ถ้าคุณสร้างคลาสขึ้นมาแล้วกำหนดให้มีคอนสตรัคเตอร์มากกว่า 1 ชุด เราเรียกการกระทำดังกล่าวว่าเป็นการทำ Constructor Overloading จากโค้ดข้างต้นเห็นได้ว่า เป็นตัวอย่างคอนสตรัคเตอร์ของคลาส Customer มีคอนสตรัคเตอร์ 2 ชุดคือ แบบปกติ และแบบที่มีการรับพารามิเตอร์

ถึงจุดนี้จะเกิดคำถามหนึ่งขึ้นมาก็คือ เมื่อใดเราควรใช้คอนสตรัคเตอร์แบบปกติ เมื่อใดควรใช้แบบรับค่าพารามิเตอร์ ซึ่งมีจุดที่ใช้ในการตัดสินใจ ดังนี้

การเลือกใช้คอนสตรัคเตอร์แบบปกติ หรือแบบรับค่าพารามิเตอร์ให้ดูคลาสที่คุณสร้างขึ้นมาเป็นหลัก กล่าวคือ ถ้าคลาสที่คุณสร้างขึ้นมาต้องการพารามิเตอร์ต่างๆ เพื่อที่จะนำไปใช้คำนวณ, หาค่า หรือประมวลผล เป็นต้น โดยที่ต้องส่งค่าพารามิเตอร์เข้ามาให้ครบตามจำนวนที่คุณต้องการ ขาดตัวใดตัวหนึ่งไม่ได้ คุณควรใช้คอนสตรัคเตอร์แบบมีการรับค่าพารามิเตอร์เข้ามา

เหตุผลก็คือ เมื่อมีการสร้างอินสแตนซ์ของคลาสขึ้นมาด้วยคำสั่ง New (VB 2005) หรือคำสั่ง new (VC# 2005) ผู้ใช้ต้องส่งพารามิเตอร์ต่างๆ เข้ามาให้ครบตามจำนวนเท่านั้น จึงจะสามารถสร้างอินสแตนซ์ขึ้นมาได้

ถ้าผู้ใช้ส่งค่าพารามิเตอร์เข้ามาไม่ครบตามจำนวน ก็จะไม่สามารถสร้างอินสแตนซ์ของคลาสนั้นๆ ขึ้นมาได้เลย ตีความได้ว่า เราบังคับให้ผู้ใช้ต้องส่งค่าพารามิเตอร์เข้ามาให้ครบตามจำนวนที่เราต้องการ เพื่อที่จะนำค่าดังกล่าวไปใช้งานต่อภายในคลาสนั่นเอง ให้ดูตัวอย่างที่ 9.1 ทำความรู้จักกับ Constrauctor เพิ่มเติม

NOTE

คลาสต่างๆ ของ .NET Framework คุณจะพบว่าบางครั้งคุณสามารถสร้างอินสแตนซ์ของคลาสขึ้นมาได้หลายวิธี เช่น ใส่พารามิเตอร์หรือไม่ใส่ก็ได้ หรือบางครั้งใส่พารามิเตอร์ได้หลายรูปแบบ หมายความว่า ไมโครซอฟท์กำหนดให้คลาสดังกล่าวเกิดขึ้นมาได้หลายวิธีนั่นเอง

ผู้เขียนสร้างคลาส Customer ขึ้นมามีโค้ดดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 9-1 ทำความรู้จักกับ Constructor

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
    Private _FirstName As String
    Private _LastName As String

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property

    Public Sub New()

    End Sub

    Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
        Me._FirstName = FirstName
        Me._LastName = LastName
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Constructor
{
    public class Customer
    {
        private string _FirstName;
        private string _LastName;

        public string FirstName
        {
            get{return _FirstName;}
            set{_FirstName = value;}
        }

        public string LastName
        {
            get{return _LastName;}
            set{_LastName = value;}
        }

        public Customer()
        {

        }

        public Customer(string FirstName, string LastName)
        {
            this._FirstName = FirstName;
            this._LastName = LastName;
        }
    }
}
```

คลาส Customer ประกอบด้วย 2 คุณสมบัติ 2 คอนสตรัคเตอร์ที่น่าสนใจอยู่ตรงคอนสตรัคเตอร์ตัวที่ 2 ผู้เขียนกำหนดให้ส่งพารามิเตอร์เข้ามา 2 ตัว เพื่อกำหนดค่าให้กับฟิลด์ _FirstName และ _LastName ส่วนที่ Form1 ให้คุณเขียนโค้ดดังต่อไปนี้

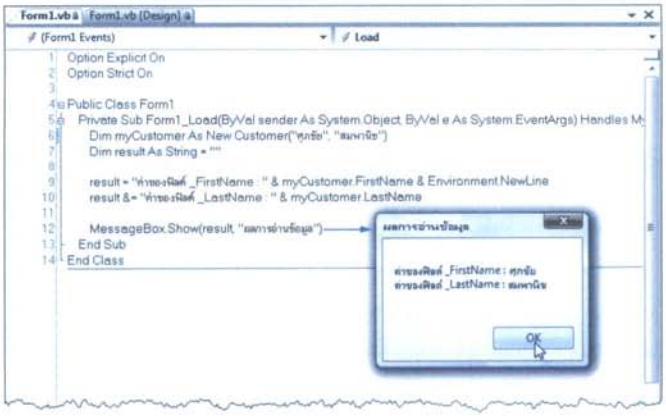
โค้ด VB 2005 ที่ 9-1 ทำความรู้จักกับ Constructor (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim myCustomer As New Customer("ศุภชัย", "สมพานิช")
        Dim result As String = ""

        result = "ค่าของฟิลด์ _FirstName : " & myCustomer.FirstName & Environment.NewLine
        result &= "ค่าของฟิลด์ _LastName : " & myCustomer.LastName

        MessageBox.Show(result, "ผลการอ่านข้อมูล")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 9-1 ทำความรู้จักกับ Constructor (Form1.cs)

```
namespace Constructor
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            Customer myCustomer = new Customer("ศุภชัย", "สมพานิช");
            string result = "";

            result = "ค่าของฟิลด์ _FirstName : " + myCustomer.FirstName + Environment.NewLine;
            result += "ค่าของฟิลด์ _LastName : " + myCustomer.LastName;

            MessageBox.Show(result, "ผลการอ่านข้อมูล");
        }
    }
}
```

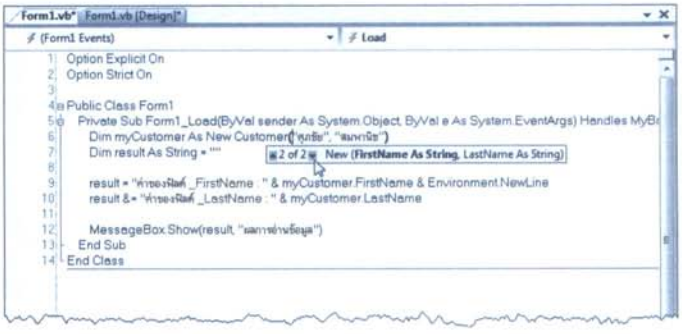
รูปที่ 9-1
ผลการทำงาน

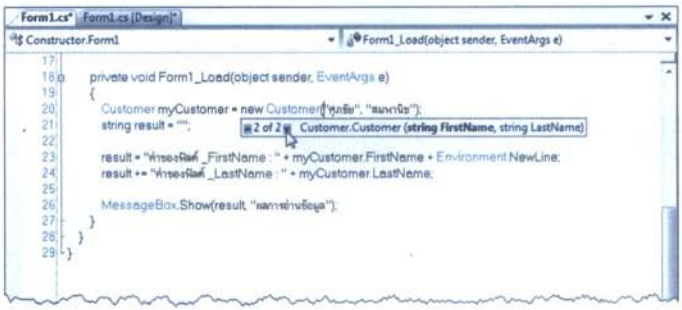


จากโค้ดใน Form1 ข้างต้น ผู้เขียนสร้างออบเจ็กต์ Customer ที่ชื่อว่า myCustomer ขึ้นมา โดยส่งชื่อและนามสกุลของผู้เขียนเข้าไปในขณะที่สร้างออบเจ็กต์ตัวนี้ขึ้นมา และสั่งให้แสดงค่าที่เก็บอยู่ในฟิลด์ _FirstName และฟิลด์ _LastName ผ่านทางคุณสมบัติ FirstName และ LastName ตามลำดับ

จากคลาส Customer ต้นแบบที่สร้างขึ้นมา เรากำหนดให้ผู้ใช้สามารถสร้างออบเจ็กต์ขึ้นมาได้ 2 วิธี ตามจำนวนคอนสตรัคเตอร์ กล่าวคือ

รูปที่ 9-2
จำนวนคอนสตรัคเตอร์ของคลาส Customer





- ถ้าผู้ใช้ไม่ได้ส่งพารามิเตอร์ใดๆ เข้ามา คอนสตรัคเตอร์ที่ถูกสั่งให้ทำงานคือ Default Constructor ตัวอย่างคลาสนี้ไม่มีการเขียนโค้ดใดๆ ส่งผลให้ฟิลด์ _FirstName และฟิลด์ _LastName มีค่าเริ่มต้นตามที่ระบุไว้ในคลาส ในกรณีนี้คือ ค่าว่าง

VB 2005

```
Public Sub New()

End Sub
```

VC# 2005

```
public Customer()
{

}
```

- แต่ถ้าผู้ใช้ส่งค่าพารามิเตอร์เข้ามา 2 ตัว โดยที่ต้องมีชนิดของข้อมูลตรงตามที่กำหนดไว้ ในกรณีนี้คือ ข้อมูลชนิด String ทั้ง 2 ตัว คอนสตรัคเตอร์ที่ถูกสั่งให้ทำงานในกรณีนี้คือ

VB 2005

```
Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
    Me._FirstName = FirstName
    Me._LastName = LastName
End Sub
```

VC# 2005

```
public Customer(string FirstName, string LastName)
{
    this._FirstName = FirstName;
    this._LastName = LastName;
}
```

จะเห็นได้ว่าเมื่อมีการส่งค่าเข้ามาแล้ว ก็จะมีการส่งต่อไปยังฟิลด์ _FirstName และฟิลด์ _LastName ตามลำดับ คุณจะนำฟิลด์ทั้ง 2 นี้ไปทำอะไรก็แล้วแต่หน้าที่คลาสของคุณ คำถามที่ตามมาก็คือ ทำไมผู้เขียนระบุคำสั่ง Me (VB 2005) หรือ this (VC# 2005) กำกับหน้าฟิลด์ทั้ง 2 ไว้ด้วย

ผู้อ่านสังเกตหรือไม่ว่า พารามิเตอร์ของคอนสตรัคเตอร์ตัวนี้มีชื่อคล้ายๆ กับฟิลด์ทั้ง 2 ต่างกันก็แต่เพียงไม่มีเครื่องหมาย _ นำหน้า

การตั้งชื่อพารามิเตอร์ในคอนสตรัคเตอร์ หรือในฟังก์ชันใดๆ ก็ตาม ก็เพื่อบ่งบอกให้ผู้ใช้รู้ว่าพารามิเตอร์ตัวนั้นๆ คืออะไร, ไปกำหนดค่าอะไร, ทำหน้าที่อะไรอยู่ ฯลฯ ในกรณีนี้เพื่อกำหนดชื่อ (FirstName) และนามสกุล (LastName) ของฟิลด์ _FirstName และฟิลด์ _LastName ในคลาสตามลำดับ

เนื่องจากผู้เขียนกำหนดให้การตั้งชื่อฟิลด์ในคลาสต้องเริ่มต้นด้วยเครื่องหมาย _ เสมอ จึงทำให้คุณผู้อ่านไม่เห็นความสับสนที่เกิดขึ้น ขอให้คุณดูตัวอย่างโค้ดต่อไปนี้

VB 2005

```
Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
    FirstName = FirstName
    LastName = LastName
End Sub
```

VC# 2005

```
public Customer(string FirstName, string LastName)
{
    FirstName = FirstName;
    LastName = LastName;
}
```

สมมติว่าเป็นโค้ดที่ไม่ได้ใช้ตั้งชื่อฟิลด์ในคลาสด้วยเครื่องหมาย _ นำหน้า ความสับสนที่เกิดขึ้นก็คือ FirstName ตัวแรกกับ FirstName ตัวที่ 2 คืออะไร ตัวไหนคือพารามิเตอร์ที่ส่งเข้ามา ตัวไหนคือฟิลด์ที่อยู่ในคลาส เพราะว่ากฎการตั้งชื่อฟิลด์ด้วยเครื่องหมาย _ ไม่มีการบังคับไว้ โปรแกรมเมอร์แต่ละคนก็มีสไตล์ไม่เหมือนกัน จะตั้งชื่อฟิลด์อย่างไรก็ได้ ดังนั้น ถ้าคุณพบเห็นโค้ดในลักษณะนี้ เห็นได้ว่า จะสร้างความสับสนให้คุณพอสมควร

NOTE

มีกติกาอยู่อย่างหนึ่งที่เกี่ยวข้องกับการระบุสมาชิกของคลาส (ฟิลด์เป็นสมาชิกของคลาสประเภทหนึ่งเช่นกัน) นั่นคือ คุณสามารถใช้คำสั่ง Me (VB 2005) หรือคำสั่ง this (VC# 2005) กำกับไว้ด้านหน้าสมาชิกของคลาสได้ โดยที่สมาชิกดังกล่าวต้องเป็นสมาชิกแบบ Dynamic เท่านั้น

ในกรณีนี้คือ ฟิลด์ FirstName และฟิลด์ LastName มีชื่อซ้ำกับพารามิเตอร์ FirstName และ LastName ทางแก้ปัญหานี้ก็คือ

VB 2005

```
Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
    Me.FirstName = FirstName
    Me.LastName = LastName
End Sub
```

VC# 2005

```
public Customer(string FirstName, string LastName)
{
    this.FirstName = FirstName;
    this.LastName = LastName;
}
```

จะเห็นได้ว่า FirstName ที่มีคำสั่ง Me หรือ this กำกับไว้ด้านหน้าคือ ฟิลด์ในคลาส ส่วน FirstName และ LastName ที่เหลือคือ พารามิเตอร์ที่ถูกส่งเข้ามาทางคอนสตรัคเตอร์นั่นเองเห็นได้ว่า เราสามารถแยกระหว่างพารามิเตอร์และสมาชิกที่มีชื่อซ้ำกันได้ โดยการระบุ Me (VB 2005) หรือ this (VC# 2005) กำกับสมาชิกไว้นั่นเอง

การใช้งาน Shared (static) และ Dynamic Constructor

เราทราบมาแล้วว่าคอนสตรัคเตอร์เป็นเมธอดพิเศษที่ทำงานโดยอัตโนมัติ เมื่อมีการสร้างอินสแตนซ์ขึ้นมา ผู้เขียนเน้นว่าเป็นเรื่องของอินสแตนซ์ เราก็ทราบมาอีกว่าสมาชิกแบบ Shared (static) เป็นเรื่องของคลาส เน้นว่าเป็นเรื่องของคลาส แล้วคอนสตรัคเตอร์แบบ Shared (static) คืออะไร

คอนสตรัคเตอร์ก็มี 2 แบบเช่นเดียวกับสมาชิกอื่นๆ ที่อยู่ในคลาส กล่าวคือ

- คอนสตรัคเตอร์แบบ Dynamic ได้แก่ Default Constructor และ Constructor ที่มีการรับพารามิเตอร์
- คอนสตรัคเตอร์แบบ Shared (static)

คำถามที่เกิดขึ้นก็คือ เราจะใช้คอนสตรัคเตอร์แบบ Shared (static) ตอนไหน ในเมื่อการใช้งานสมาชิกแบบ Shared (static) ต้องทำผ่านคลาส ส่วนคอนสตรัคเตอร์แบบ Dynamic ต้องทำผ่านอินสแตนซ์

สมาชิกที่อยู่ในคลาสมีทั้งแบบ Shared (static) และแบบ Dynamic เรารู้มาว่าสมาชิกแบบ Shared (static) สามารถอ้างอิงสมาชิกประเภทเดียวกันเท่านั้น ไปยุ่งกับสมาชิกแบบ Dynamic ไม่ได้เด็ดขาด ส่งผลให้คอนสตรัคเตอร์แบบ Shared (static) จึงมีขอบเขตเพียงแค่ใช้กำหนดค่าเริ่มต้นต่างๆ ให้กับสมาชิกแบบ Shared (static) ที่อยู่ในคลาสเท่านั้น

คำถามที่ตามมาก็คือ คอนสตรัคเตอร์แบบ Shared (static) สามารถรับพารามิเตอร์ได้หรือไม่ คำตอบคือ ไม่ได้ เนื่องจากว่าไม่รู้จะส่งค่าให้กับพารามิเตอร์ตอนไหน เพราะสมาชิกแบบ Shared (static) เป็นเรื่องของคลาส ไม่เกี่ยวข้องกับอินสแตนซ์แต่อย่างใด ให้ดูตัวอย่างที่ 9-2 การใช้งาน Shared (static) และ Dynamic Constructor เพิ่มเติม

```
Option Explicit On
```

โค้ด VB 2005 ที่ 9-2 การใช้งาน Shared (static) และ Dynamic Constructor (Class1.vb)

```
Option Strict On

Public Class Customer
    Private Shared _FullName As String

    Public Sub New()
        _FullName = "ข้อความจาก Default Constructor"
    End Sub

    Shared Sub New()
        _FullName = "ข้อความจาก Shared (static) Constructor"
    End Sub

    Public Shared ReadOnly Property SharedFullName() As String
        Get
            Return _FullName
        End Get
    End Property

    Public ReadOnly Property DynamicFullName() As String
        Get
            Return _FullName
        End Get
    End Property
End Class
public class Customer
```

โค้ด VC# 2005 ที่ 9-2 การใช้งาน Shared (static) และ Dynamic Constructor (Class1.cs)

```
{
    private static string _FullName;

    public Customer()
    {
        _FullName = "ข้อความจาก Default Constructor";
    }

    static Customer()
    {
        _FullName = "ข้อความจาก Shared (static) Constructor";
    }

    public static string staticFullName
    {
        get{return _FullName;}
    }

    public string DynamicFullName
    {
        get{return _FullName;}
    }
}
```

ผู้เขียนสร้างคลาสชื่อว่า Customer มีฟิลด์ที่ชื่อว่า _FullName เป็นแบบ Shared (static) มีคอนสตรัคเตอร์ทั้ง 2 แบบ กำหนดค่าให้กับฟิลด์ _FullName แตกต่างกัน โดยที่

- คอนสตรัคเตอร์แบบ Dynamic กำหนดข้อความจาก Default Constructor
- ส่วนคอนสตรัคเตอร์แบบ Shared (static) กำหนดข้อความจาก Shared (static) Constructor

มีคุณสมบัติ staticFullName เป็นสมาชิกแบบ Shared (static) ส่วนคุณสมบัติ DynamicFullName เป็นสมาชิกแบบ Dynamic ทำหน้าที่อ่านค่าออกมาจากฟิลด์ _FullName เหมือนกัน การใช้งานคลาส Customer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 9-2 การใช้งาน Shared (static) และ Dynamic Constructor

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result1 As String
        result1 = Customer.SharedFullName
        MessageBox.Show(result1)

        Dim c As New Customer()
        Dim result2 As String
        result2 = c.DynamicFullName
        MessageBox.Show(result2)
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    string result1 = Customer.staticFullName;
    MessageBox.Show(result1);

    Customer c = new Customer();
    string result2 = c.DynamicFullName;
    MessageBox.Show(result2);
}
```

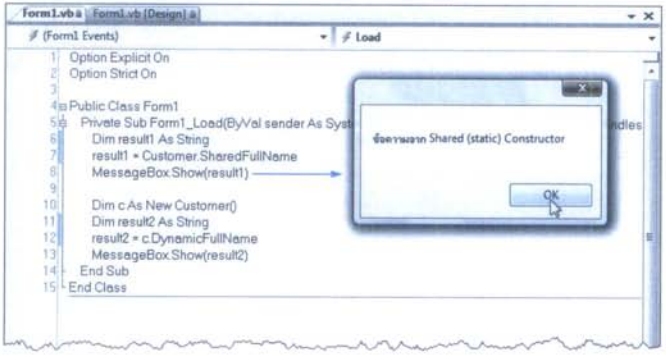
โค้ดช่วงแรกกำหนดให้ตัวแปร result1 อ่านค่าออกมาจากฟิลด์ _FullName ผ่านทางคุณสมบัติ staticFullName ผลที่ได้ แสดงดังรูปที่ 9-3

VB 2005

```
Dim result1 As String
result1 = Customer.SharedFullName
MessageBox.Show(result1)
```

VC# 2005

```
string result1 = Customer.staticFullName;
MessageBox.Show(result1);
```



รูปที่ 9-3 ผลการทำงานของ Shared (static) Constructor

จากรูปที่ 9-3 ข้อความที่ปรากฏขึ้นมาถูกกำหนดมาจากการทำงานของ Shared (static) Constructor ส่วนโค้ดช่วงที่ 2 สร้างอินสแตนซ์ของคลาส Customer ขึ้นมา ชื่อว่า c ลองอ่านค่าจากฟิลด์ _FullName เช่นกัน ผ่านทางคุณสมบัติ DynamicFullName

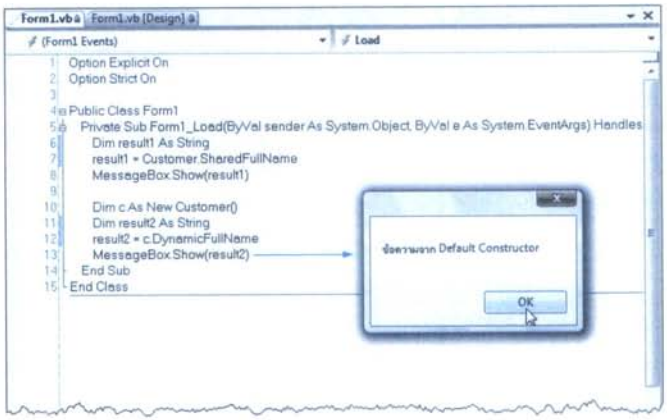
VB 2005

```
Dim c As New Customer()
Dim result2 As String
result2 = c.DynamicFullName
MessageBox.Show(result2)
```

VC# 2005

```
Customer c = new Customer();
string result2 = c.DynamicFullName;
MessageBox.Show(result2);
```

รูปที่ 9-4 ผลการทำงานของ Dynamic Constructor



ผลการทำงานที่ได้สื่อให้เห็นว่า การทำงานของ Shared (static) Constructor เกิดขึ้นก่อน Dynamic Constructor เพราะว่าเมื่อคุณอ่านค่าของฟิลด์ _FullName ผ่านทางคุณสมบัติ DynamicFullName ของออบเจ็กต์ c ข้อความที่ได้เกิดจากการกำหนดใน Dynamic Constructor ไปทับข้อความที่มาจาก Shared (static) Constructor ซึ่งทำงานก่อนหน้านี้นั่นเอง

การ Copy Constructor

เทคนิคอย่างหนึ่งที่คุณสามารถนำคอนสตรัคเตอร์มาใช้งานได้ก็คือ การ Copy ข้อมูลจากออบเจ็กต์หนึ่งไปสู่อีกออบเจ็กต์หนึ่ง แต่การ Copy ดังกล่าวไม่ได้หมายความว่าออบเจ็กต์ทั้ง 2 ตัวเท่ากัน ให้ดูตัวอย่างที่ 9-3 การ Copy Constructor เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 9-3 การ Copy Constructor

VB 2005 (Customer.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
        Private _FirstName As String = ""
        Private _LastName As String = ""

        Public Property FirstName() As String
            Get
                Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property

    Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
        Me._FirstName = FirstName
        Me._LastName = LastName
    End Sub

    Public Sub New(ByVal CCustomer As Customer)
        Me._FirstName = CCustomer.FirstName
        Me._LastName = CCustomer.LastName
    End Sub
End Class
```

VC# 2005 (Customer.cs)

```
public class Customer
{
    private string _FirstName;
    private string _LastName;

    public string FirstName
    {
        get{return _FirstName;}
        set{_FirstName = value;}
    }

    public string LastName
    {
        get{return _LastName;}
        set{_LastName = value;}
    }

    public Customer(string FirstName, string LastName)
    {
        this._FirstName = FirstName;
        this._LastName = LastName;
    }

    public Customer(Customer CCustomer)
    {
        this._FirstName = CCustomer.FirstName;
        this._LastName = CCustomer.LastName;
    }
}
```

ผู้เขียนสร้างคลาส Customer ขึ้นมาประกอบด้วยคุณสมบัติ FirstName และคุณสมบัติ LastName ที่น่าสนใจอยู่ตรงที่คอนสตรัคเตอร์ตัวที่ 2 ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น Customer เราจะใช้คอนสตรัคเตอร์ตัวนี้ Copy ข้อมูล

เนื่องจากคลาส Customer มีเพียง 2 คุณสมบัติ ก็จะกำหนดให้เกิดการถ่ายค่ากันระหว่างฟิลด์ _FirstName กับคุณสมบัติ FirstName และฟิลด์ _LastName กับคุณสมบัติ LastName ของออบเจ็กต์ Customer ที่ถูกส่งเข้ามา

VB 2005

```
Public Sub New(ByVal CCustomer As Customer)
    Me._FirstName = CCustomer.FirstName
    Me._LastName = CCustomer.LastName
End Sub
```

VC# 2005

```
public Customer(Customer CCustomer)
{
    this._FirstName = CCustomer.FirstName;
    this._LastName = CCustomer.LastName;
}
```

การใช้งานคลาส Customer อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 9-3 การ Copy Constructor (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim C1 As New Customer("ศุภชัย", "สมพานิช")
        Dim C2 As New Customer(C1)

        Dim result As String = ""
        result = "ชื่อ-สกุลของ C2 : " & C2.FirstName & " " & C2.LastName
        MessageBox.Show(result, "รายละเอียดของ C2")

        If C1.Equals(C2) Then
            MessageBox.Show("เท่ากัน")
        End If

        If Object.Equals(C1, C2) Then
            MessageBox.Show("เท่ากัน")
        End If
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 9-3 การ Copy Constructor (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Customer C1 = new Customer("ศุภชัย", "สมพานิช");
    Customer C2 = new Customer(C1);

    string result = "";
    result = "ชื่อ-สกุลของ C2 : " + C2.FirstName + " " + C2.LastName;
    MessageBox.Show(result, "รายละเอียดของ C2");

    if (C1.Equals(C2))
    {
        MessageBox.Show("เท่ากัน");
    }

    if (Object.Equals(C1,C2))
    {
        MessageBox.Show("เท่ากัน");
    }
}
```

วิธีการดูโค้ดชุดนี้ก็คือ สร้างออบเจ็กต์ Customer ชื่อว่า C1 ส่งชื่อ-สกุลผู้เขียนเข้าไป ส่งผลให้ออบเจ็กต์ C1 คือ อินสแตนซ์ชุดที่ 1 ของคลาส Customer ต่อมาสร้างออบเจ็กต์ Customer ขึ้นมาอีก 1 อินสแตนซ์ ชื่อว่า C2 เห็นได้ว่าออบเจ็กต์ C2 เกิดขึ้นมาโดยการส่งอินสแตนซ์ C1 เข้าไป

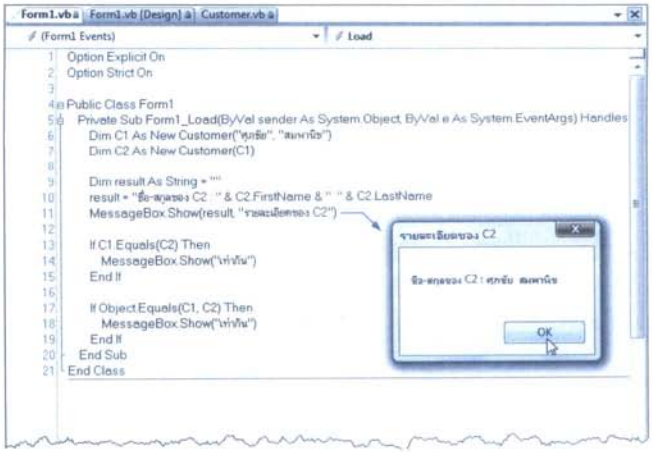
VB 2005

```
Dim C1 As New Customer("ศุภชัย", "สมพานิช")
Dim C2 As New Customer(C1)

Dim result As String = ""
result = "ชื่อ-สกุลของ C2 : " & C2.FirstName & " " & C2.LastName
MessageBox.Show(result, "รายละเอียดของ C2")
```

VB 2005

```
Customer C1 = new Customer("ศุภชัย", "สมพานิช");
Customer C2 = new Customer(C1);

string result = "";
result = "ชื่อ-สกุลของ C2 : " + C2.FirstName + " " + C2.LastName;
MessageBox.Show(result, "รายละเอียดของ C2");
```

รูปที่ 9-5
ผลการรัน
ตัวอย่างที่ 9-3



จากรูปที่ 9-5 เห็นได้ว่าอินสแตนซ์ C2 มีข้อมูลเหมือนกับอินสแตนซ์ C1 นั่นคือ เก็บชื่อ-สกุลของผู้เขียน เนื่องจากว่าออบเจ็กต์เป็น Reference Type ไม่ได้เก็บค่าไว้กับตัวเอง แต่เก็บตำแหน่งหน่วยความจำไว้ คำว่าเหมือนกันจึงแตกต่างจากคำว่าเท่ากัน

ทดสอบโดยการใช้เมธอด Equals() ของคลาส Object เพื่อตรวจสอบว่า ออบเจ็กต์ C1 กับออบเจ็กต์ C2 เท่ากันหรือไม่ ผลที่ได้คือ ไม่เท่ากัน

VB 2005

```
If C1.Equals(C2) Then
    MessageBox.Show("เท่ากัน")
End If

If Object.Equals(C1, C2) Then
    MessageBox.Show("เท่ากัน")
End If
```

VC# 2005

```
if (C1.Equals(C2))
{
    MessageBox.Show("เท่ากัน");
}

if (Object.Equals(C1,C2))
{
    MessageBox.Show("เท่ากัน");
}
```

ความหมายข้างต้นคือ ออบเจ็กต์ C1 และออบเจ็กต์ C2 เป็นคนละอินสแตนซ์ เพียงแต่ว่ามีข้อมูลเหมือนกัน เป็นอีก 1 ตัวอย่างที่แสดงให้เห็นถึงความแตกต่างระหว่าง Reference Type กับ Value Type คำว่าเหมือนกันของ Value Type หมายถึง เท่ากันด้วย เช่น

VB 2005

```
Dim x As Integer = 5
Dim y As Integer = 5
```

VC# 2005

```
int x = 5;
int y = 5;
```

ตัวแปร x มี Type เป็น Integer (int) มีค่าเหมือนกันกับตัวแปร y (ตัวแปร x กับตัวแปร y มีค่าเท่ากันด้วย)

การสืบทอดคลาสกับ Default Constructor

การสืบทอดคลาสระหว่างคลาสแม่กับคลาสลูก เราทราบมาแล้วว่าสิ่งที่คลาสลูกได้จากคลาสแม่ก็คือสมาชิกที่มีขอบเขตแบบ Protected และขอบเขตแบบ Public ของคลาสแม่ ยังมีอีกส่วนหนึ่งของคลาสแม่ที่คลาสลูกได้ไปด้วย นั่นคือ

- การทำงานที่อยู่ใน Default Constructor
- การทำงานที่อยู่ใน Shared (static) Constructor

ให้ดูตัวอย่างที่ 9-4 การสืบทอดคลาสกับ Default Constructor เพิ่มเติม

NOTE

สิ่งที่คลาสลูกได้จากคลาสแม่ในส่วนของ Shared (static) Constructor จะอธิบายในหัวข้อถัดไป

โค้ด VB 2005 ที่ 9-4 การสืบทอดคลาสกับ Default Constructor

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Parent
    Private _x As Integer = 0
    Private _y As Integer = 0

    Public ReadOnly Property x() As Integer
        Get
            Return _x
        End Get
    End Property

    Public ReadOnly Property y() As Integer
        Get
            Return _y
        End Get
    End Property

    Public Sub New()
        MessageBox.Show("Default Constructor
ของคลาสแม่ทำงาน")
        _x = 20
        _y = 30
    End Sub
```

VC# 2005 (Class1.cs)

```
public class Parent
{
    private int _x = 0;
    private int _y = 0;

    public int x
    {
        get{return _x;}
    }

    public int y
    {
        get{return _y;}
    }

    public Parent()
    {
        MessageBox.Show("Default Constructor ของคลาสแม่
ทำงาน");
        _x = 20;
        _y = 30;
    }
}

public class Child : Parent
```

```
End Class

Public Class Child
    Inherits Parent

    Public Sub New()

    End Sub
End Class
```

```
{
    public Child()
    {

    }
}
```

ผู้เขียนสร้างคลาสแม่ชื่อว่า Parent ประกอบด้วย 2 คุณสมบัติคือ คุณสมบัติ x กับ y มีขอบเขตแบบ Public ทำหน้าที่อ่านค่าเพียงอย่างเดียวออกมาจากฟิลด์ x และฟิลด์ y ตามลำดับ

ในคลาสแม่ Parent มี Default Constructor ทำหน้าที่กำหนดค่าเริ่มต้นให้กับฟิลด์ x และฟิลด์ y เพื่อให้ชัดเจนยิ่งขึ้นจึงกำหนดให้แสดง MessageBox ขึ้นมา เมื่อ Default Constructor ของคลาสแม่ Parent ทำงาน ส่วนที่คลาสลูก Child สืบทอดคลาสมาจากคลาสแม่ Parent มีแต่ Default Constructor ของคลาสลูก การทดสอบอยู่ใน Form1 ดังโค้ดต่อไปนี้

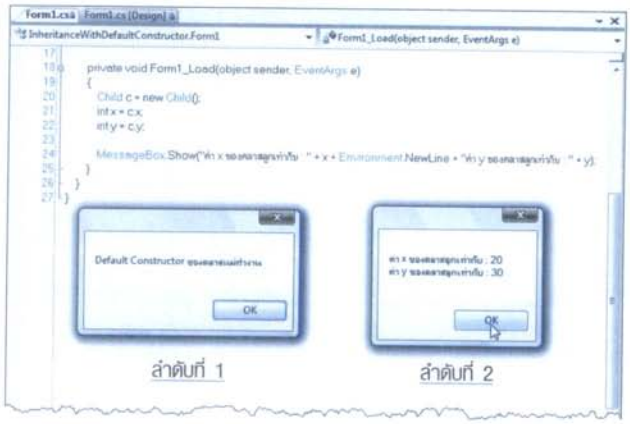
โค้ด VB 2005 ที่ 9-4 การสืบทอดคลาสกับ Default Constructor (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim c As New Child()
        Dim x As Integer = c.x
        Dim y As Integer = c.y

        MessageBox.Show("ค่า x ของคลาสลูกเท่ากับ : " & x & Environment.NewLine & "ค่า y ของคลาสลูกเท่ากับ : " & y)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 9-4 การสืบทอดคลาสกับ Default Constructor (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Child c = new Child();
        int x = c.x;
        int y = c.y;

        MessageBox.Show("ค่า x ของคลาสลูกเท่ากับ : " + x + Environment.NewLine + "ค่า y ของคลาสลูกเท่ากับ : " + y);
    }
```

รูปที่ 9-6
ผลการทำงาน
ตัวอย่างที่ 9-4



จากโค้ดทดสอบข้างต้น ผู้เขียนสร้างอินสแตนซ์ของคลาสลูก Child ขึ้นมาชื่อว่า c กำหนดให้อ่านค่าของฟิลด์ x และฟิลด์ y ผ่านทางคุณสมบัติ x และ y

พบว่า MessageBox ที่อยู่ใน Default Constructor ของคลาสแม่ทำงาน ในขณะที่ออบเจ็กต์ c ของคลาสลูก Child เกิดขึ้น รวมถึงค่าของฟิลด์ x และฟิลด์ y ที่อ่านค่าได้คือค่าเดียวกับที่ถูกกำหนดใน Default Constructor ของคลาสแม่ ตีความได้ว่าคลาสลูกได้การทำงานที่อยู่ใน Default Constructor ของคลาสแม่ไปด้วยนั่นเอง

ลำดับการทำงานของคอนสตรัคเตอร์ระหว่างคลาสแม่กับคลาสลูก

ลำดับการทำงานของคอนสตรัคเตอร์ระหว่างคลาสแม่กับคลาสลูก เป็นอีกเรื่องหนึ่งที่มีความสำคัญเป็นอย่างยิ่งเพราะว่า เรามักจะใช้คอนสตรัคเตอร์สำหรับกำหนดค่าเริ่มต้น หรือรับค่าเข้ามาเพื่อนำไปใช้งานต่อภายในคลาส

ส่งผลให้ลำดับการทำงานก่อน-หลังของคอนสตรัคเตอร์จึงมีความสำคัญเป็นอย่างยิ่ง ดังตัวอย่างที่ 9-5 ลำดับการทำงานของคอนสตรัคเตอร์ระหว่างคลาสแม่กับคลาสลูก

โค้ด VB 2005 ที่ 9-5 ลำดับการทำงานของคอนสตรัคเตอร์ระหว่างคลาสแม่กับคลาสลูก (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Parent
    Shared Sub New()
        MessageBox.Show("Shared Constructor ของคลาสแม่ทำงาน")
    End Sub

    Public Sub New()
        MessageBox.Show("Default Constructor ของคลาสแม่ทำงาน")
    End Sub
```

```
    Public Sub New(ByVal str As String)
        MessageBox.Show("Constructor แบบมีพารามิเตอร์ของคลาสแม่ทำงาน")
    End Sub

    Public Sub New(ByVal str1 As String, ByVal str2 As String)
        MessageBox.Show("Overload Constructor ของคลาสแม่ทำงาน")
    End Sub
End Class

Public Class Child
    Inherits Parent

    Shared Sub New()
        MessageBox.Show("Shared Constructor ของคลาสลูกทำงาน")
    End Sub

    Public Sub New()
        MessageBox.Show("Default Constructor ของคลาสลูกทำงาน")
    End Sub

    Public Sub New(ByVal str As String)
        MessageBox.Show("Constructor แบบมีพารามิเตอร์ของคลาสลูกทำงาน")
    End Sub

    Public Sub New(ByVal str1 As String, ByVal str2 As String)
        MessageBox.Show("Overload Constructor ของคลาสลูกทำงาน")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 9-5 ลำดับการทำงานของคอนสตรัคเตอร์ระหว่างคลาสแม่กับคลาสลูก (Class1.cs)

```
public class Parent
{
    static Parent()
    {
        MessageBox.Show("static Constructor ของคลาสแม่ทำงาน");
    }

    public Parent()
    {
        MessageBox.Show("Default Constructor ของคลาสแม่ทำงาน");
    }

    public Parent(string str)
    {
        MessageBox.Show("Constructor แบบมีพารามิเตอร์ของคลาสแม่ทำงาน");
    }

    public Parent(string str1, string str2)
    {
```

```
        MessageBox.Show("Overload Constructor ของคลาสแม่ทำงาน");
    }
}

public class Child : Parent
{
    static Child()
    {
        MessageBox.Show("static Constructor ของคลาสลูกทำงาน");
    }

    public Child()
    {
        MessageBox.Show("Default Constructor ของคลาสลูกทำงาน");
    }

    public Child(string str)
    {
        MessageBox.Show("Constructor แบบมีพารามิเตอร์ของคลาสลูกทำงาน");
    }

    public Child(string str1,string str2)
    {
    MessageBox.Show("Overload Constructor ของคลาสลูกทำงาน");
    }
}
```

วิธีการทดสอบก็คือ ในคลาสแม่ Parent และคลาสลูก Child มีคอนสตรัคเตอร์ครบทั้ง 4 ประเภทคือ Shared (static) Constructor, Default Constructor คอนสตรัคเตอร์แบบมีพารามิเตอร์ และคอนสตรัคเตอร์ที่ถูก Overload (หมายถึง คอนสตรัคเตอร์ที่มีพารามิเตอร์แตกต่างกัน)

ในคอนสตรัคเตอร์แต่ละตัวจะมี MessageBox ระบุข้อความไว้ว่า MessageBox ดังกล่าวมาจากไหน การทดสอบอยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 9-5 ลำดับการทำงานของคอนสตรัคเตอร์ระหว่างคลาสแม่กับคลาสลูก

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    Dim c As New Child()
    'Dim c As New Child("ทดสอบ")
    'Dim c As New Child("ทดสอบ1", "ทดสอบ2")
End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Child c = new Child();
    //Child c = new Child("ทดสอบ");
    //Child c = new Child("ทดสอบ1", "ทดสอบ2");
}
```

การทดสอบนี้สนใจที่คลาสลูกเป็นหลัก โดยการสร้างออบเจ็กต์ Child ที่ชื่อว่า c ขึ้นมา แยกการทดสอบออกเป็น 3 กรณีคือ

1. กรณี Default Constructor ลำดับการทำงานที่ได้คือ
   - 1.1 Shared (static) Constructor ของคลาสลูก
   - 1.2 Shared (static) Constructor ของคลาสแม่
   - 1.3 Default Constructor ของคลาสแม่
   - 1.4 Default Constructor ของคลาสลูก

VB 2005	VC# 2005
Dim c As New Child()	Child c = new Child();

2. กรณีคอนสตรัคเตอร์แบบมีพารามิเตอร์ ลำดับการทำงานที่ได้คือ
   - 2.1 Shared (static) Constructor ของคลาสลูก
   - 2.2 Shared (static) Constructor ของคลาสแม่
   - 2.3 Default Constructor ของคลาสแม่
   - 2.4 คอนสตรัคเตอร์แบบมีพารามิเตอร์ของคลาสลูก

VB 2005	VC# 2005
Dim c As New Child("ทดสอบ")	Child c = new Child("ทดสอบ");

3. กรณี Overload Constructor ลำดับการทำงานที่ได้คือ
   - 3.1 Shared (static) Constructor ของคลาสลูก
   - 3.2 Shared (static) Constructor ของคลาสแม่
   - 3.3 Default Constructor ของคลาสแม่
   - 3.4 Overload Constructor ของคลาสลูก

VB 2005	VC# 2005
Dim c As New Child("ทดสอบ1", "ทดสอบ2")	Child c = new Child("ทดสอบ1", "ทดสอบ2");

การทำ Constructor Chaining

การทำ Constructor Chaining หมายถึง การสั่งให้คอนสตรัคเตอร์ทำงานติดต่อกัน เราทราบมาแล้วว่า คลาสลูกได้รับ Shared (static) Constructor และ Default Constructor ของคลาสแม่มาด้วย ประเด็นที่น่าสนใจคือถ้าเราต้องการให้คอนสตรัคเตอร์แบบมีพารามิเตอร์ของคลาสแม่มาทำงานในคลาสลูกด้วย จะทำอย่างไร

ในบางกรณีการกำหนดค่าเริ่มต้นต่างๆ ภายในคลาสแม่ไม่จำเป็นต้องทำใน Default Constructor แต่อาจจะต้องรับค่าตอนสร้างอินสแตนซ์ก็ได้ ให้ดูตัวอย่างที่ 9-6 การทำ Constructor Chaining

โค้ด VB 2005 และ VC# 2005 ที่ 9-6 การทำ Constructor Chaining

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Parrent
    Public Sub New()
        MessageBox.Show("Default Constructor ของคลาสแม่
ทำงาน")
    End Sub

    Public Sub New(ByVal x As Integer)
        MessageBox.Show("Constructor แบบมีพารามิเตอร์ของ
คลาสแม่ทำงาน")
    End Sub
End Class


Public Class Child
    Inherits Parrent

    Public Sub New()
        MyBase.New(5)
        MessageBox.Show("Default Constructor ของคลาสลูก
ทำงาน")
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Parrent
{
    public Parrent()
    {
        MessageBox.Show("Default Constructor
ของคลาสแม่ทำงาน");
    }

    public Parrent(int x)
    {
        MessageBox.Show("Constructor แบบมีพารามิเตอร์ของ
คลาสแม่ทำงาน");
    }
}

public class Child : Parrent
{
    public Child(): base(5)
    {
        MessageBox.Show("Default Constructor ของคลาสลูก
ทำงาน");
    }
}
```

ที่คลาสแม่ Parrent มีคอนสตรัคเตอร์ 2 ชนิดคือ Default Constructor และคอนสตรัคเตอร์แบบมีพารามิเตอร์ 1 ตัว มี Type เป็นเลขจำนวนเต็ม Integer (int) ที่น่าสนใจอยู่ที่คลาสลูก เห็นได้ว่าที่ Default Constructor ของคลาสลูกมีการสั่งให้คอนสตรัคเตอร์แบบมีพารามิเตอร์ของคลาสแม่ทำงาน ผ่านทางคำสั่ง MyBase (base) โดยการส่งค่า 5 (เลขจำนวนเต็ม) เข้าไปนั่นเอง

โค้ด VB 2005 และ VC# 2005 ที่ 9-6 การทำ Constructor Chaining

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim c As New Child()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Child c = new Child();
}
```

ลำดับการทำงานที่เกิดขึ้นก็คือ

1. คอนสตรัคเตอร์แบบมีพารามิเตอร์ของคลาสแม่
2. Default Constructor ของคลาสลูก

จากลำดับการทำงานข้างต้นเห็นได้ว่า เราสามารถสั่งให้คอนสตรัคเตอร์แบบมีพารามิเตอร์ของคลาสแม่ ทำงานจากคลาสลูกได้เช่นกัน โดยอาศัยคำสั่ง MyBase (base) นั่นเอง ถ้าจะแปลโค้ดข้างต้นเป็นคำพูดก็จะได้ประมาณ ตอนคลาสแม่เกิดต้องทำอะไรก็ให้ทำในคลาสลูกด้วย

ข้อแตกต่างระหว่างฟิลด์ชนิด Constant กับฟิลด์ชนิด ReadOnly

หน้าที่ของคอนสตรัคเตอร์อีกอย่างหนึ่งก็คือ การกำหนดค่าให้กับฟิลด์ชนิด ReadOnly คำถามหนึ่งที่เกิดขึ้นมาก็คือ ฟิลด์ชนิดค่าคงที่ (Constant) กับฟิลด์ชนิด ReadOnly แตกต่างกันอย่างไร

ฟิลด์ชนิดค่าคงที่เราไม่สามารถกำหนดค่าใหม่ได้ เพราะมันเป็นค่าคงที่ ส่วนฟิลด์ชนิด ReadOnly เราก็ไม่สามารถกำหนดค่าใหม่ได้เช่นกัน เพราะว่าฟิลด์ชนิดนี้อ่านค่าได้เพียงอย่างเดียว ความแตกต่างระหว่างฟิลด์ทั้ง 2 ประเภท เกี่ยวข้องกับการทำงานของคอนสตรัคเตอร์ ให้ดูตัวอย่างที่ 9-7 ข้อแตกต่างระหว่างฟิลด์ชนิด Constant กับฟิลด์ชนิด ReadOnly

โค้ด VB 2005 และ VC# 2005 ที่ 9-7 ข้อแตกต่างระหว่างฟิลด์ชนิด Constant กับฟิลด์ชนิด ReadOnly

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Sample
    Public Const x As String = "ข้อความเริ่มต้น"

End Class
```

VC# 2005 (VC# 2005)

```
public class Sample
{
    public const string x = "ข้อความเริ่มต้น";

}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า Sample มีสมาชิก 1 ตัว เป็นฟิลด์ชนิดค่าคงที่ชื่อว่า x มีขอบเขตแบบ Public เพื่อทดสอบฟิลด์ชนิด Const กำหนดข้อความเริ่มต้นไว้

โค้ด VB 2005 และ VC# 2005 ที่ 9-7 ข้อแตกต่างระหว่างฟิลด์ชนิด Constant กับฟิลด์ชนิด ReadOnly

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As String = Sample.x
        MessageBox.Show(result, "ค่า x ปัจจุบัน")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    string result = Sample.x;
    MessageBox.Show(result, "ค่า x ปัจจุบัน");
}
```

รูปที่ 9-7
ผลการอ่านค่าฟิลด์ x

จากรูปที่ 9-7 ทดสอบอ่านค่าฟิลด์ x ผ่านทางคลาส Sample ผลที่ได้คือ ฟิลด์ x เก็บข้อความเริ่มต้นไว้ถือว่าปกติ จากนั้นในคลาส Sample ให้คุณเพิ่ม Default Constructor ขึ้นมาใหม่ ทำหน้าที่กำหนดข้อความให้กับฟิลด์ x ใหม่ ดังโค้ดต่อไปนี้

VB 2005 (Class1.vb)

```
Public Sub New()
    x = "กำหนดข้อความใหม่จาก Default Constructor"
End Sub
```

VC# 2005 (Class1.cs)

```
public Sample()
{
    x = "กำหนดข้อความใหม่จาก Default Constructor";
}
```

เห็นได้ว่าคลาส Sample เกิดข้อผิดพลาดจากโค้ดข้างต้น เพราะว่าคุณไม่สามารถกำหนดค่าใหม่ให้กับฟิลด์ x ซึ่งเป็นฟิลด์ชนิด Const ส่งผลให้เกิดข้อผิดพลาดขึ้นมา ผู้เขียนแก้ไขคลาส Sample ใหม่ ดังโค้ดต่อไปนี้

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Sample
    Public ReadOnly x As String = "ข้อความเริ่มต้น"

    Public Sub New()
        x = "กำหนดข้อความใหม่จาก Default Constructor"
    End Sub

    Public Sub New(ByVal str As String)
        x = str
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Sample
{
    public readonly string x ="ข้อความเริ่มต้น";

    public Sample()
    {
        x = "กำหนดข้อความใหม่จาก Default Constructor";
    }

    public Sample(string str)
    {
        x = str;
    }
}
```

ส่วนที่แก้ไขใหม่ก็คือ เปลี่ยนฟิลด์ x จากชนิด Const เป็นชนิด ReadOnly และเพิ่มคอนสตรัคเตอร์ขึ้นมา 2 ตัวคือ Default Constructor และคอนสตรัคเตอร์แบบมีพารามิเตอร์ โดยที่ในแต่ละคอนสตรัคเตอร์ทำหน้าที่กำหนดข้อความให้กับฟิลด์ x ใหม่ ข้อแตกต่างที่เกิดขึ้นคือ

โค้ดข้างต้นไม่เกิดข้อผิดพลาดแต่อย่างใด ฟิลด์ชนิด ReadOnly ยอมให้คุณแก้ไขค่าใหม่ผ่านทางคอนสตรัคเตอร์ ส่งผลให้ในแต่ละอินสแตนซ์ก็จะมีค่าของฟิลด์ x เป็นของตัวเอง คุณสามารถใช้ช่องทางนี้กำหนดให้ฟิลด์ชนิด ReadOnly มีค่าแตกต่างกันไปตามความต้องการของแต่ละอินสแตนซ์นั่นเอง กล่าวได้อีกนัยหนึ่ง ฟิลด์ชนิด ReadOnly เป็นเรื่องของอินสแตนซ์ ส่วนฟิลด์ชนิด Const เป็นสมาชิกแบบ Shared (static) เป็นเรื่องของคลาส

การทดสอบคลาส Sample ใหม่อยู่ใน Form1 ดังโค้ดต่อไปนี้

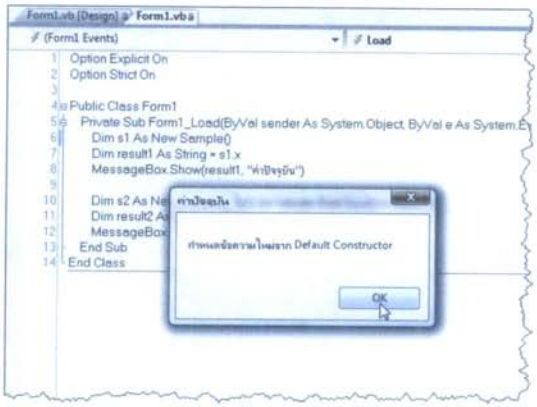
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim s1 As New Sample()
        Dim result1 As String = s1.x
        MessageBox.Show(result1, "ค่าปัจจุบัน")

        Dim s2 As New Sample("ข้อความจากคอนสตรัคเตอร์
แบบมีพารามิเตอร์")
        Dim result2 As String = s2.x
        MessageBox.Show(result2, "ค่าปัจจุบัน")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Sample s1 = new Sample();
        string result1 = s1.x;
        MessageBox.Show(result1, "ค่าปัจจุบัน");

        Sample s2 = new Sample("ข้อความจากคอนสตรัคเตอร์
แบบมีพารามิเตอร์");
        string result2 = s2.x;
        MessageBox.Show(result2, "ค่าปัจจุบัน");
    }
```

รูปที่ 9-8
ข้อความของฟิลด์ x หลังจากสร้างอินสแตนซ์ s



จากรูปที่ 9-8 เห็นได้ว่าข้อความของฟิลด์ x ของอินสแตนซ์ s1 เปลี่ยนไปจากข้อความเริ่มต้นเดิม ข้อความที่ปรากฏขึ้นมาถูกกำหนดค่าใหม่จากการทำงานของ Default Constructor ซึ่งทำงานตอนสร้างอินสแตนซ์ s1 ขึ้นมานั่นเอง

VB 2005

```
Dim s1 As New Sample()
Dim result1 As String = s1.x
MessageBox.Show(result1, "ค่าปัจจุบัน")
```

VC# 2005

```
Sample s1 = new Sample();
string result1 = s1.x;
MessageBox.Show(result1, "ค่าปัจจุบัน");
```

ต่อมาสร้างออบเจ็กต์ Sample ที่ชื่อว่า s2 ขึ้นมาอีก 1 อินสแตนซ์ กำหนดข้อความให้กับฟิลด์ x ใหม่ผ่านทางคอนสตรัคเตอร์แบบมีพารามิเตอร์ ผลการทำงานแสดงดังรูปที่ 9-9

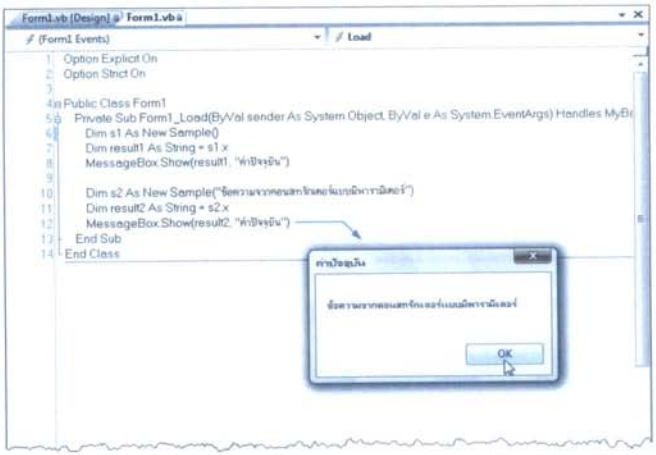
VB 2005

```
Dim s2 As New Sample("ข้อความจากคอนสตรัคเตอร์แบบมีพารามิเตอร์")
Dim result2 As String = s2.x
MessageBox.Show(result2, "ค่าปัจจุบัน")
```

VC# 2005

```
Sample s2 = new Sample("ข้อความจากคอนสตรัคเตอร์แบบมีพารามิเตอร์");
string result2 = s2.x;
MessageBox.Show(result2, "ค่าปัจจุบัน");
```

รูปที่ 9-9
ข้อความของฟิลด์ x หลังจากสร้างอินสแตนซ์ s2



เห็นได้ว่าค่าของฟิลด์ x ในอินสแตนซ์ s1 กับ s2 แตกต่างกัน นี่คือข้อแตกต่างระหว่างฟิลด์ชนิด Const กับฟิลด์ชนิด ReadOnly

ถ้าคุณเขียนโค้ดเพื่อแก้ไขค่าของฟิลด์ x ผ่านทางอินสแตนซ์ ก็จะเกิดข้อผิดพลาดขึ้น เพราะว่าฟิลด์ชนิด ReadOnly อ่านค่าได้เพียงอย่างเดียว แก้ไขผ่านทางอินสแตนซ์ไม่ได้

VB 2005	VC# 2005
s2.x = 'กำหนดค่าใหม่อีกครั้ง' ⊗	s2.x = 'กำหนดค่าใหม่อีกครั้ง'; ⊗

สรุปท้ายบท

Constructor เป็นจุดเกิด เป็นประตูด่านแรกที่ควบคุมออบเจ็กต์ที่เกิดขึ้นมา เห็นได้ว่า Constructor มีความสำคัญไม่ยิ่งหย่อนไปกว่า member function ประเภทอื่นๆ เลย ออบเจ็กต์ที่เกิดขึ้นมาจะทำงานถูกต้องหรือไม่ ส่วนหนึ่งก็ต้องดูจุดที่มันเกิดขึ้นมาด้วยเช่นกัน


Programming in OOP style
Advanced .net

CHAPTER ▸ 10

Structure และ Abstract Class

บทนำ

เนื้อหาในบทนี้คุณจะได้ศึกษา Structure และ Abstract Class สิ่งที่น่าสนใจคือ อะไรคือความแตกต่างระหว่างคลาสปกติ, Structure และ Abstract Class

พื้นฐานการใช้งาน Structure

Structure มีลักษณะและการใช้งานคล้ายกับคลาส ถ้าคลาสคือพี่ใหญ่ สตรัคเจอร์ก็ไม่ต่างอะไรกับน้องเล็ก ข้อแตกต่างที่สำคัญที่สุดระหว่างพี่น้องคู่นี้คือ คลาสเป็น Reference Type ส่วนสตรัคเจอร์เป็น Value Type คำถามแรกที่เกิดขึ้นมาก็คือ เมื่อใดควรใช้คลาส เมื่อใดควรใช้สตรัคเจอร์

การใช้งานคลาสที่คุณสร้างขึ้นมา คุณต้องสร้างอินสแตนซ์ของคลาสขึ้นมาก่อน เพื่อให้ได้ออบเจ็กต์ไปทำงาน งานที่ออบเจ็กต์สามารถทำได้จะเป็นไปตามคลาสต้นแบบที่เราเป็นผู้กำหนดหน้าที่ไว้

แต่ถ้าคุณมีงานเล็กๆ น้อยๆ เช่น ที่พักเก็บข้อความ, เก็บโครงสร้างข้อมูลเฉพาะบางอย่างที่ไม่ซับซ้อนมากนัก งานลักษณะนี้คุณไม่จำเป็นต้องใช้งานคลาส อาจจะยกงานแบบนี้ให้สตรัคเจอร์รับไปทำแทนก็ได้ เพราะว่าถ้างานเล็กๆ น้อยๆ คุณยังคงเรียกใช้คลาส ก็จะทำให้แอพพลิเคชันของคุณเต็มไปด้วยอินสแตนซ์มากเกินความจำเป็นนั่นเอง ให้ดูตัวอย่างที่ 10-1 พื้นฐานการใช้งาน Structure

โค้ด VB 2005 และ VC# 2005 ที่ 10-1 พื้นฐานการใช้งาน Structure

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Structure SimpleStructure
        Public x As Integer
        Public y As Integer
    End Structure

    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ss As SimpleStructure
        ss.x = 5
        ss.y = 10
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
public struct SimpleStructure
{
    public int x;
    public int y;
}

private void Form1_Load(object sender, EventArgs e)
{
    SimpleStructure ss;
    ss.x = 5;
    ss.y = 10;
}
```

โค้ดข้างต้นสร้างสตรัคเจอร์ที่ชื่อว่า SimpleStructure ขึ้นมา ประกอบด้วย 2 ฟิลด์คือ ฟิลด์ x และฟิลด์ y มี Type เป็น Integer (int) มีขอบเขตแบบ Public กำหนดให้สตรัคเจอร์ SimpleStructure ทำหน้าที่เก็บข้อมูลชนิด Integer (int) ได้ 2 ค่า

คุณไม่สามารถกำหนดค่าเริ่มต้นให้กับฟิลด์ทั้ง 2 ดังโค้ดต่อไปนี้

VB 2005

```
Public x As Integer = 5
Public y As Integer = 10
```

VC# 2005

```
public int x = 5;
public int y = 10;
```

เพราะความที่สตรัคเจอร์เป็น Value Type จึงไม่จำเป็นต้องใช้คำสั่ง New (new) แต่อย่างใด สมมติว่าการทำงานของคุณต้องการเก็บเลขจำนวนเต็ม 2 จำนวนไว้ใช้งานในภายหลัง เช่น เก็บค่า 5 ไว้กับฟิลด์ x เก็บค่า 10 ไว้กับฟิลด์ y เป็นต้น

VB 2005

```
Dim ss As SimpleStructure
ss.x = 5
ss.y = 10
```

VC# 2005

```
SimpleStructure ss;
ss.x = 5;
ss.y = 10;
```

การใช้งาน Structure กับ member functions

สตรัคเจอร์สามารถมี member functions ได้เช่นเดียวกับคลาส ไม่ว่าจะเป็นคุณสมบัติหรือเมธอด เป็นต้น ให้ดูตัวอย่างที่ 10-2 การใช้งาน Structure กับ member functions เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 10-2 การใช้งาน Structure กับ member functions

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Structure Calculator
    Private _x As Double
    Private _y As Double

    Public Property x() As Double
        Get
            Return _x
        End Get
        Set(ByVal value As Double)
            _x = value
        End Set
    End Property

    Public Property y() As Double
        Get
            Return _y
        End Get
        Set(ByVal value As Double)
            _y = value
        End Set
    End Property

    Public Function Add() As Double
        Return _x + _y
    End Function

    Public Function Subtract() As Double
        Return _x - _y
    End Function

    Public Function Multiply() As Double
        Return _x * _y
    End Function

    Public Function Divide() As Double
        Return _x / _y
    End Function
End Structure
```

VC# 2005 (Class1.cs)

```
public struct Calculator
{
    private double _x;
    private double _y;

    public double x
    {
        get{return _x;}
        set{_x = value;}
    }

    public double y
    {
        get{return _y;}
        set{_y = value;}
    }

    public double Add()
    {
        return _x + _y;
    }

    public double Subtract()
    {
        return _x - _y;
    }

    public double Multiply()
    {
        return _x * _y;
    }

    public double Divide()
    {
        return _x / _y;
    }
}
```

ผู้เขียนสร้างสตรัคเจอร์ที่ชื่อว่า Calculator ทำหน้าที่เป็นเครื่องคิดเลขง่ายๆ ประกอบด้วยคุณสมบัติ x และคุณสมบัติ y มี Type เป็น Double (double) ทำหน้าที่อ่านค่าหรือกำหนดตัวเลขเข้ามา และมีเมธอดบวก (Add()), ลบ (Subtract()), คูณ (Multiply()) และหาร (Divide())

โค้ด VB 2005 และ VC# 2005 ที่ 10-2 การใช้งาน Structure กับ member functions

VB 2005 (Form1.vb)

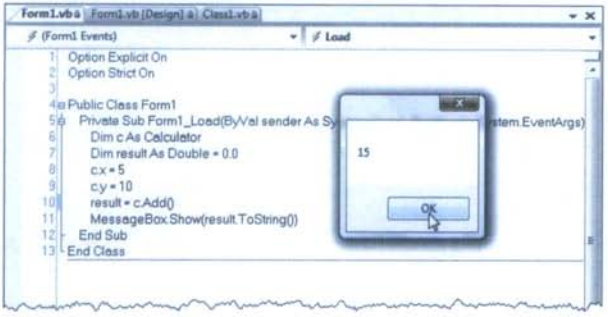
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim c As Calculator
        Dim result As Double = 0.0
        c.x = 5
        c.y = 10
        result = c.Add()
        MessageBox.Show(result.ToString())
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Calculator c;
    double result = 0.0;
    c.x = 5;
    c.y = 10;
    result = c.Add();
    MessageBox.Show(result.ToString());
}
```

สร้างเครื่องคิดเลข Calculator ที่ชื่อว่า c ขึ้นมา ให้บวกเลขทั้ง 2 เข้าด้วยกันผลการทำงานแสดงดังรูปที่ 10-1

รูปที่ 10-1 ผลการทำงาน



ข้อแตกต่างระหว่าง Structure (struct) กับ Class

สิ่งที่น่าสนใจที่สุดก็คือ ข้อแตกต่างระหว่างสตรัคเจอร์กับคลาสนั่นเอง เพราะว่าถ้าคุณไม่ทราบข้อแตกต่างแล้วจะมีจุดตัดสินใจในการเลือกใช้ได้อย่างไร ให้ดูตัวอย่างที่ 10-3 ข้อแตกต่างระหว่าง Structure (struct) กับ Class

โค้ด VB 2005 และ VC# 2005 ที่ 10-3 ข้อแตกต่างระหว่าง Structure (struct) กับ Class

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
    Private _FullName As String = ""
    Public Property FullName() As String
        Get
            Return _FullName
        End Get
        Set(ByVal value As String)
            _FullName = value
        End Set
    End Property
End Class

Public Structure Student
    Private _FullName As String
    Public Property FullName() As String
        Get
            Return _FullName
        End Get
        Set(ByVal value As String)
            _FullName = value
        End Set
    End Property
End Structure
```

VC# 2005 (Class1.cs)

```
public class Customer
{
    private string _FullName = "";
    public string FullName
    {
        get{return _FullName;}
        set{_FullName = value;}
    }
}

public struct Student
{
    private string _FullName;
    public string FullName
    {
        get{return _FullName;}
        set{_FullName = value;}
    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า Customer และสตรัคเจอร์ที่ชื่อว่า Student ขึ้นมา กำหนดให้มีคุณสมบัติ FullName เพียงคุณสมบัติเดียว เพื่อเปรียบเทียบข้อแตกต่างระหว่างกัน

โค้ด VB 2005 และ VC# 2005 ที่ 10-3 ข้อแตกต่างระหว่าง Structure (struct) กับ Class

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim C1 As New Customer()
        C1.FullName = "ศุภชัย สมพานิช"
        Dim C2 As Customer = C1
        C1.FullName = "อดิเทพ ตรังคานนท์"

        Dim r1 As String = ""
        r1 = "C1 : " & C1.FullName & Environment.NewLine
        r1 &= "C2 : " & C2.FullName
        MessageBox.Show(r1)

        Dim S1 As New Student()
        S1.FullName = "ศุภชัย สมพานิช"
        Dim S2 As Student = S1

        S1.FullName = "อดิเทพ ตรังคานนท์"
        Dim r2 As String = ""
        r2 = "S1 : " & S1.FullName & Environment.NewLine
        r2 &= "S2 : " & S2.FullName
        MessageBox.Show(r2)
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Customer C1 = new Customer();
    C1.FullName = "ศุภชัย สมพานิช";
    Customer C2 = C1;
    C1.FullName = "อดิเทพ ตรังคานนท์";

    string r1 = "";
    r1 = "C1 : " + C1.FullName + Environment.NewLine;
    r1 += "C2 : " + C2.FullName;
    MessageBox.Show(r1);

    Student S1 = new Student();
    S1.FullName = "ศุภชัย สมพานิช";
    Student S2 = S1;
    S1.FullName = "อดิเทพ ตรังคานนท์";

    string r2 = "";
    r2 = "S1 : " + S1.FullName + Environment.NewLine;
    r2 += "S2 : " + S2.FullName;
    MessageBox.Show(r2);
}
```

วิธีการดูโค้ดชุดนี้ให้ดูเป็นช่วงๆ กล่าวคือ สร้างออบเจ็กต์ Customer ที่ชื่อว่า C1 ขึ้นมา กำหนดชื่อสกุลผู้เขียนให้กับออบเจ็กต์ C1

VB 2005

```
Dim C1 As New Customer()
C1.FullName = "ศุภชัย สมพานิช"
```

VC# 2005

```
Customer C1 = new Customer();
C1.FullName = "ศุภชัย สมพานิช";
```

ต่อมาสร้างตัวแปรที่ชื่อว่า C2 ขึ้นมา มี Type เป็น Customer ยังไม่มีการสร้างออบเจ็กต์ Customer แต่อย่างใด เพราะว่าไม่มีการใช้คำสั่ง New (new) การกำหนดให้ตัวแปร C2 เท่ากับออบเจ็กต์ C1 หมายความว่า กำหนดให้ตัวแปร C2 ชี้ที่อินสแตนซ์เดียวกับตัวแปร C1 นั่นเอง

ผู้เขียนลองเปลี่ยนชื่อ-สกุลของอินสแตนซ์ C1 เป็นชื่อบุคคลอื่น ผลการทำงานแสดงดังรูปที่ 10-2

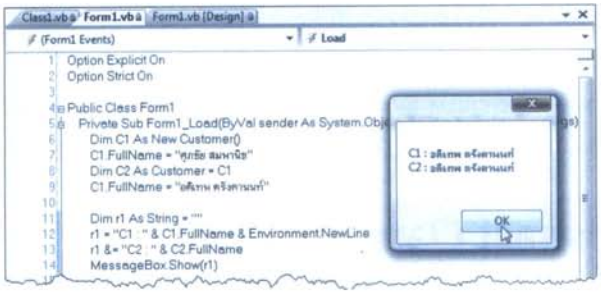
VB 2005

```
Dim C2 As Customer = C1
C1.FullName = "อดิเทพ ตรังคานนท์"

Dim r1 As String = ""
r1 = "C1 : " & C1.FullName & Environment.NewLine
r1 &= "C2 : " & C2.FullName
MessageBox.Show(r1)
```

VC# 2005

```
Customer C2 = C1;
C1.FullName = "อดิเทพ ตรังคานนท์";

string r1 = "";
r1 = "C1 : " + C1.FullName + Environment.NewLine;
r1 += "C2 : " + C2.FullName;
MessageBox.Show(r1);
```

รูปที่ 10-2 ผลการทำงานของช่วงแรก



จากรูปที่ 10-2 เหตุผลที่ตัวแปร C1 และตัวแปร C2 มีค่าเหมือนกัน เป็นเพราะว่าชี้ที่อินสแตนซ์เดียวกันนั่นเอง

ส่วนโค้ดในช่วงที่ 2 ผู้เขียนจงใจใช้สตรัคเจอร์ Student โดยการใช้คำสั่ง New (new) ทำเหมือนกับการใช้งานคลาส Customer ทุกประการ ผลการทำงานของสตรัคเจอร์ Student แสดงดังรูปที่ 10-3

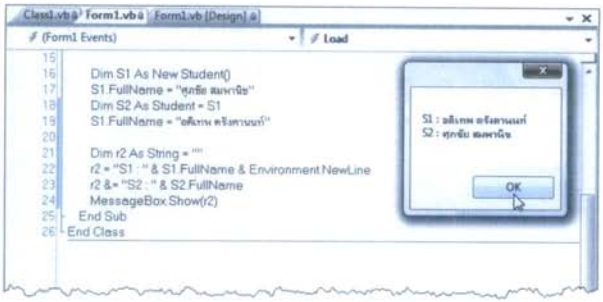
VB 2005

```
Dim S1 As New Student()
S1.FullName = "ศุภชัย สมพานิช"
Dim S2 As Student = S1
S1.FullName = "อดิเทพ ตรังคานนท์"

Dim r2 As String = ""
r2 = "S1 : " & S1.FullName & Environment.NewLine
r2 &= "S2 : " & S2.FullName
MessageBox.Show(r2)
```

VC# 2005

```
Student S1 = new Student();
S1.FullName = "ศุภชัย สมพานิช";
Student S2 = S1;
S1.FullName = "อดิเทพ ตรังคานนท์";

string r2 = "";
r2 = "S1 : " + S1.FullName + Environment.NewLine;
r2 += "S2 : " + S2.FullName;
MessageBox.Show(r2);
```

รูปที่ 10-3 ผลการทำงานของช่วงหลัง



จากรูปที่ 10-3 เห็นได้ว่าตัวแปร S1 และตัวแปร S2 เก็บค่าแตกต่างกัน เพราะว่าสตรัคเจอร์เป็น Value Type เก็บค่าไว้กับตัวเอง ไม่ใช่การชี้อินสแตนซ์แบบ Reference Type ของคลาส

การกำหนดให้ตัวแปร S2 เท่ากับ S1 จึงแตกต่างจากตัวแปร C2 เท่ากับ C1 ตรงที่ว่า เป็นการกำหนดให้ตัวแปร S2 เก็บค่าเหมือนกับตัวแปร S1 ต่างคนต่างมีค่าเป็นของตัวเอง ไม่ใช่ชี้ไปที่ตำแหน่งเดียวกัน ดังเช่นตัวแปร C1 และตัวแปร C2

การใช้งาน Abstract Class

Abstract Class เป็นคลาสอีกลักษณะหนึ่งแตกต่างไปจากคลาสปกติตรงที่ ในกรณีที่คุณสร้างคลาสขึ้นมา แล้วมีบางเมธอดที่คุณไม่สามารถเขียนโค้ดทำงานได้ทันที ณ ขณะออกแบบ เมธอดที่คุณไม่สามารถเขียนโค้ดได้ คุณต้องระบุไปว่าเป็นเมธอดแบบ Abstract ซึ่งหมายถึง เป็นเมธอดว่างเปล่า เวลาใช้งานต้องไปเขียนโค้ดเอง คำว่าเวลาใช้งานต้องไปเขียนโค้ดเอง เรียกอีกอย่างว่าการทำ Implements (อิม-พลี-เมนต์)

เมื่อคลาสที่คุณออกแบบมีเมธอดอย่างน้อย 1 เมธอด เป็นแบบ Abstract ส่งผลให้คลาสที่คุณสร้างขึ้นมาต้องเป็น Abstract Class ไปด้วย

เมื่อคลาสปกติกลายเป็น Abstract Class ก็จะส่งผลให้คุณไม่สามารถสร้างออบเจ็กต์จากคลาสดังกล่าวได้โดยตรง คุณต้องสร้างคลาสปกติขึ้นมาเพื่อสืบทอด Abstract Class ก่อน ให้ดูตัวอย่างที่ 10-4 การใช้งาน Abstract Class เพิ่มเติม

โค้ด VB 2005 ที่ 10-4 การใช้งาน Abstract Class (Class1.vb)

```
Option Explicit On
Option Strict On

Public MustInherit Class absCalculator
    Public MustOverride Function Add(ByVal x As Double, ByVal y As Double) As Double
    Public MustOverride Function Subtract(ByVal x As Double, ByVal y As Double) As Double

    Public Overridable Function Multiply(ByVal x As Double, ByVal y As Double) As Double
        Return (x * y)
    End Function
```

```
    Public Function Divide(ByVal x As Double, ByVal y As Double) As Double
        Return x / y
    End Function
End Class
```

โค้ด VC# 2005 ที่ 10-4 การใช้งาน Abstract Class (Class1.cs)

```
public abstract class absCalculator
{
    public abstract double Add(double x, double y);
    public abstract double Subtract(double x, double y);

    public virtual double Multiply(double x, double y)
    {
        return x * y;
    }

    public double Divide(double x, double y)
    {
        return x / y;
    }
}
```

สมมติว่าผู้เขียนต้องการสร้างเครื่องคิดเลขขึ้นมาประกอบด้วย 4 เมธอดคือ บวก (Add()), ลบ (Subtract()), คูณ (Multiply()) และหาร (Divide())

ในระหว่างการเขียนโค้ด สมมติว่าผู้เขียนไม่ทราบวิธีการบวกและลบว่าทำอย่างไร เขียนโค้ดได้แต่การคูณและหาร จึงต้องกำหนดให้เมธอด Add() และเมธอด Subtract() เป็นเมธอดแบบ Abstract ไว้ก่อน โดยที่ใน VB 2005 ใช้คำสั่ง MustOverride ส่วน VC# 2005 ใช้คำสั่ง abstract กำกับเมธอดทั้ง 2 ไว้ กล่าวได้อีกนัยหนึ่งคือ เมธอด Add() และเมธอด Subtract() เป็นเมธอดว่างเปล่าไม่มีโค้ดทำงานระบุไว้

เมื่อเครื่องคิดเลขของผู้เขียนมีเมธอดแบบ Abstract ถึง 2 เมธอด จึงต้องกำหนดให้เครื่องคิดเลขเป็นคลาสแบบ Abstract Class ไปโดยปริยาย จึงต้องใช้คำสั่ง MustOverride ใน VB 2005 หรือคำสั่ง abstract ใน VC# 2005 กำกับไว้ที่ส่วนของการสร้างคลาส absCalculate เช่นกัน

คุณไม่สามารถสร้างอินสแตนซ์จาก Abstract Class โดยตรง ดังโค้ดต่อไปนี้

VB 2005	VC# 2005
`Dim c As New absCalculator()` ✗	`absCalculator c = new absCalculator();` ✗

เหตุผลก็คือ Abstract Class มีเมธอดว่างเปล่าเป็นสมาชิกอยู่ในคลาส ถ้านำไปสร้างออบเจ็กต์ได้แล้ว เมธอดนั้นๆ จะทำงานได้อย่างไร จึงเป็นเหตุเป็นผลในตัวของมันเองอยู่แล้ว เมื่อมีเมธอดว่างเปล่าอยู่ เวลานำไปใช้งานจึงต้องเขียนทับ (Override) เมธอดเดิมก่อนนั่นเอง

เมื่อคุณต้องการใช้งาน Abstract Class ที่ชื่อว่า absCalculator จึงต้องสร้างคลาสปกติขึ้นมาอีก 1 คลาส ชื่อว่า Calculator สืบทอดจากคลาส absCalculator ดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 10-4 การใช้งาน Abstract Class (Class1.vb)

```
Public Class Calculator
    Inherits absCalculator

    Public Overrides Function Add(ByVal x As Double, ByVal y As Double) As Double
        Return (x + y)
    End Function

    Public Overrides Function Subtract(ByVal x As Double, ByVal y As Double) As Double
        Return (x - y)
    End Function

    Public Overrides Function Multiply(ByVal x As Double, ByVal y As Double) As Double
        MessageBox.Show("คุณตัวเลขเสร็จแล้ว")
        Return (x * y)
    End Function
End Class
```

โค้ด VC# 2005 ที่ 10-4 การใช้งาน Abstract Class (Class1.cs)

```
public class Calculator:absCalculator
{
    public override double Add(double x, double y)
    {
        return x + y;
    }

    public override double Subtract(double x, double y)
    {
        return x - y;
    }

    public override double Multiply(double x, double y)
    {
        MessageBox.Show("คุณตัวเลขเสร็จแล้ว");
        return x * y;
    }
}
```

โค้ดข้างต้นเป็นการสร้างคลาสที่ชื่อว่า Calculator ขึ้นมา สืบทอดจาก Abstract Class ที่ชื่อว่า absCalculator เห็นได้ว่าคุณต้องเขียนทับ (Override) เมธอด Add() และเมธอด Subtract() เดิมเป็นเมธอดว่างเปล่า

ถ้าคุณคิดว่าการคูณ (เมธอด Multiply()) ของเดิมยังไม่ดีพอ สามารถเขียนทับเมธอดเดิมเพื่อแก้ไขการทำงานใหม่ได้อีกด้วย เช่น ให้แสดง MessageBox เพิ่มเติม เป็นต้น

คำถามหนึ่งที่ต้องเกิดขึ้นมาก็คือ ทำไมเราต้องใช้ Abstract Class ให้ดูซับซ้อน วุ่นวายด้วย เขียนโค้ดทำงานให้เสร็จภายในคลาสเลยไม่ดีกว่าหรือ ผู้เขียนขอให้คุณมอง Abstract Class ในมุมกลับกันจากที่ผู้เขียนกล่าวไว้ข้างต้นทั้งหมด

เมธอดว่างเปล่าแปลได้อีกนัยหนึ่งว่า เป็นเมธอดที่ยังไม่เสร็จ เพราะว่ายังไม่มีโค้ดทำงานใดๆ แต่ก็แปลความได้อีกอย่างหนึ่งว่า เป็นเมธอดที่เปิดโอกาสให้เขียนโค้ดทำงานเองก็ได้

ผลจากการที่คุณเปิดโอกาสให้เขียนโค้ดเอง จึงทำให้คลาสลูกที่ได้มีลักษณะดังนี้

- แตกต่างกันบางเมธอด เพราะว่าต้องเขียนโค้ดเองแล้วแต่ความต้องการ
- เหมือนกันบางเมธอด เพราะเมธอดดังกล่าวมีโค้ดที่ทำงานเสร็จสมบูรณ์อยู่ใน Abstract Class แล้ว

นี่คือประโยชน์ของ Abstract Class ไม่ใช่ความยุ่งยากแต่อย่างใด

โค้ด VB 2005 และ VC# 2005 ที่ 10-4 การใช้งาน Abstract Class (Class1.vb)

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On


Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As Double = 0.0
        Dim c As New Calculator()
        result = c.Add(5, 10)
        MessageBox.Show(result.ToString())

        result = c.Subtract(10, 3)
        MessageBox.Show(result.ToString())

        result = c.Multiply(2, 3)
        MessageBox.Show(result.ToString())

        result = c.Divide(4, 2)
        MessageBox.Show(result.ToString())
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    double result = 0.0;
    Calculator c = new Calculator();
    result = c.Add(5, 10);
    MessageBox.Show(result.ToString());

    result = c.Subtract(10, 3);
    MessageBox.Show(result.ToString());

    result = c.Multiply(2, 3);
    MessageBox.Show(result.ToString());

    result = c.Divide(4, 2);
    MessageBox.Show(result.ToString());
}
```

ถึงจุดนี้ผู้เขียนได้เครื่องคิดเลขที่สามารถบวก, ลบ, คูณและหารตามที่ต้องการแล้ว

การใช้งาน Abstract Class ในฐาน: Dynamic Type

แม้ว่าคุณจะไม่สามารถสร้างออบเจ็กต์จาก Abstract Class ได้โดยตรง แต่คุณสามารถใช้ Abstract Class ในฐานะ Type ประเภทหนึ่งได้เช่นกัน และนี่เป็นอีก 1 บทบาทที่น่าสนใจ ให้ดูตัวอย่างที่ 10-5 การใช้งาน Abstract Class ในฐานะ Dynamic Type

โค้ด VB 2005 และ VC# 2005 ที่ 10-5 การใช้งาน Abstract Class ในฐานะ Dynamic Type

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public MustInherit Class absProgrammer
    Public MustOverride Sub Programming()
End Class

Public Class Programmer
    Inherits absProgrammer

    Public Overrides Sub Programming()
        MessageBox.Show("Programmer เขียนโปรแกรมได้")
    End Sub
End Class

Public Class SA
    Inherits absProgrammer

    Public Overrides Sub Programming()
        MessageBox.Show("SA เขียนโปรแกรมได้")
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public abstract class absProgrammer
{
    public abstract void Programming();
}

public class Programmer : absProgrammer
{
    public override void Programming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมได้");
    }
}

public class SA : absProgrammer
{
    public override void Programming()
    {
        MessageBox.Show("SA เขียนโปรแกรมได้");
    }
}
```

ผู้เขียนสร้าง Abstract Class ที่ชื่อว่า absProgrammer มีเพียง 1 เมธอดคือ เมธอด Programming() เป็นเมธอดว่างเปล่า บอกไว้ว่าถ้าต้องการเป็นโปรแกรมเมอร์ควรจะเขียนโปรแกรมได้

ต่อมาสร้างคลาส Programmer และคลาส SA ขึ้นมา สืบทอดมาจากคลาส absProgrammer แก้ไขเมธอด Programming() ใหม่ เพื่อบอกให้รู้ว่าทั้ง Programmer และ SA ต่างก็เขียนโปรแกรมได้เช่นกัน ส่วนการใช้งานอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 10-5 การใช้งาน Abstract Class ในฐานะ Dynamic Type

VB 2005 (Form1.vb)

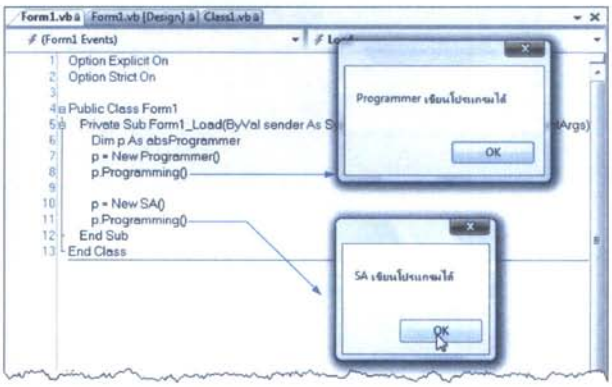
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As absProgrammer
        p = New Programmer()
        p.Programming()

        p = New SA()
        p.Programming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    absProgrammer p;
    p = new Programmer();
    p.Programming();

    p = new SA();
    p.Programming();
}
```

ประเด็นที่ต้องการนำเสนอคือ ตัวแปร p มี Type เป็น Abstract Class สามารถรองรับได้ทั้งอินสแตนซ์ของคลาส Programmer และคลาส SA ดังรูปที่ 10-4

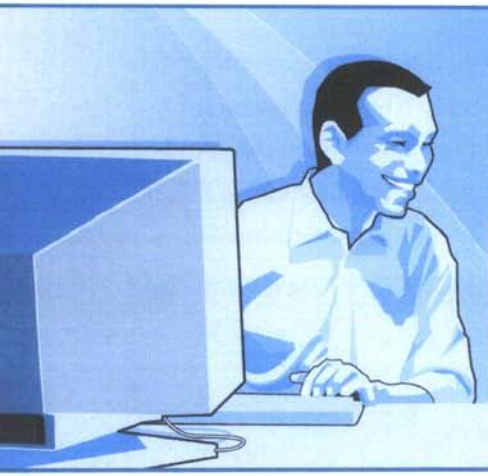
รูปที่ 10-4 ผลการทำงานตัวอย่างที่ 10-5



สรุปท้ายบท

สำหรับการใช้งาน Abstract Class ผู้เขียนจะกล่าวอย่างละเอียดครั้ง เมื่อถึงเนื้อหาของการใช้งานอินเตอร์เฟส จะทำให้คุณเห็นประโยชน์ของ Abstract Class มากยิ่งขึ้น


Programming in OOP style
Advanced .net



CHAPTER ▸ 11

Interface

บทนำ

อินเตอร์เฟส (Interface) เป็นเรื่องที่ผู้เขียนชื่นชอบเป็นส่วนตัวมากที่สุด เพราะทำให้การเขียนโปรแกรมแบบ OOP มีความยืดหยุ่นเป็นอย่างยิ่ง แต่มีรายละเอียดมากพอสมควร จึงได้แบ่งเนื้อหาออกเป็น 2 ส่วนคือ

1. Interface กล่าวถึงพื้นฐานการใช้งานอินเตอร์เฟส ซึ่งเป็นเนื้อหาของบทนี้
2. Interface Programming เป็นเนื้อหาที่กล่าวถึงการนำอินเตอร์เฟสไปใช้ในโลกของ OOP

ทำความรู้จักกับ Interface

อินเตอร์เฟส (Interface) ถือเป็นอีก 1 เรื่องที่มีความสำคัญเป็นอย่างยิ่ง เมื่อคุณเข้ามาสู่โลกของ OOP ตามความเห็นของผู้เขียน อินเตอร์เฟสเป็นทั้งเรื่องที่ทำความเข้าใจได้ง่ายที่สุดและยากที่สุดในเวลาเดียวกัน เพราะว่าจุดเริ่มต้นที่จะทำความรู้จักกับอินเตอร์เฟสยากที่สุด แต่เมื่อคุณรู้จักกับอินเตอร์เฟสแล้วก็จะกลายเป็นเรื่องง่ายที่สุดเช่นกัน โดยที่ผู้เขียนขอให้คำจำกัดความของอินเตอร์เฟสหมายถึง ส่วนขยายหรือส่วนเพิ่มเติมของคลาส

แม้ว่าอินเตอร์เฟสคือ Type ชนิดหนึ่งใน .NET Framework แต่คุณไม่สามารถนำอินเตอร์เฟสมาสร้างเป็นออบเจ็กต์โดยตรงได้ และคำถามแรกที่เกิดขึ้นมาก็คือ อินเตอร์เฟสคืออะไร ใช้อย่างไร ให้ดูตัวอย่างที่ 11-1 ทำความรู้จักกับ Interface เพื่อให้คุณทำความรู้จักกับอินเตอร์เฟสให้เร็วที่สุด ให้ดูตามขั้นตอนดังต่อไปนี้

1. การสร้างอินเตอร์เฟส ผู้เขียนสร้างอินเตอร์เฟสที่ชื่อว่า IMoveable ขึ้นมา ประกอบด้วย 2 ซับรูทีน คือ ซับรูทีน Walk() และซับรูทีน Run() ไม่มีการเขียนโค้ดใดๆ อยู่ภายในอินเตอร์เฟส ดังนี้

VB 2005 (Class1.vb)

```
Public Interface IMoveable
    Sub Walk()
    Sub Run()
End Interface
```

VC# 2005 (Class1.cs)

```
public interface IMoveable
{
    void Walk();
    void Run();
}
```

ชื่ออินเตอร์เฟสจะนำหน้าด้วยตัวอักษร I ลงท้ายด้วยคำว่า -able สื่อความหมายว่า ให้มีความสามารถอะไรบ้าง ในกรณีนี้อินเตอร์เฟส IMoveable จึงหมายถึง ถ้ามีความสามารถในการเคลื่อนไหว ต้องมีซับรูทีน Walk() และ Run() ด้วย เพราะความที่อินเตอร์เฟสไม่มีโค้ดอยู่ภายใน คุณจึงไม่สามารถสร้างออบเจ็กต์จากอินเตอร์เฟสโดยตรง ดังโค้ดต่อไปนี้

VB 2005 (Form1.vb)

```
Dim m As New IMoveable()
```

VC# 2005 (Form1.cs)

```
IMoveable m = new IMoveable();
```

การนำอินเตอร์เฟสไปใช้คุณต้องเขียนโค้ดเอง เราเรียกการกระทำนี้ว่าการ Implements Interface (อิม-พลี-เมนต์ อินเตอร์เฟส) ดังนั้น เมื่อคุณได้ยินคำว่า อิมพลีเมนต์ จึงหมายถึง การเขียนโค้ดเพิ่มเติมนั่นเอง

ตอนนี้คุณมีความสามารถในการเคลื่อนไหว (IMoveable) แล้ว ให้ระบุไว้ว่าสิ่งที่เคลื่อนไหวได้ต้องประกอบด้วยการเดิน (ซับรูทีน Walk()) และการวิ่ง (ซับรูทีน Run()) สมมติว่าคุณสร้างคลาสคนธรรมดา (Person) และคลาสแมว (Cat) ขึ้นมา พบว่าทั้งคนธรรมดาและแมวต้องมีความสามารถในการเคลื่อนไหวได้ คุณจึงต้องอิมพลีเมนต์อินเตอร์เฟส IMoveable ให้กับคลาส Person และคลาส Cat ดังโค้ดต่อไปนี้

VB 2005 (Class1.vb)

```
Public Class Person
    Implements IMoveable

    Public Sub Walk() Implements IMoveable.Walk

    End Sub

    Public Sub Run() Implements IMoveable.Run

    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Person : IMoveable
{
    public void Walk()
    {

    }

    public void Run()
    {

    }
}
```

```
Public Class Cat
    Implements IMoveable

    Public Sub Walk() Implements IMoveable.Walk


    End Sub

    Public Sub Run() Implements IMoveable.Run


    End Sub
End Class
```

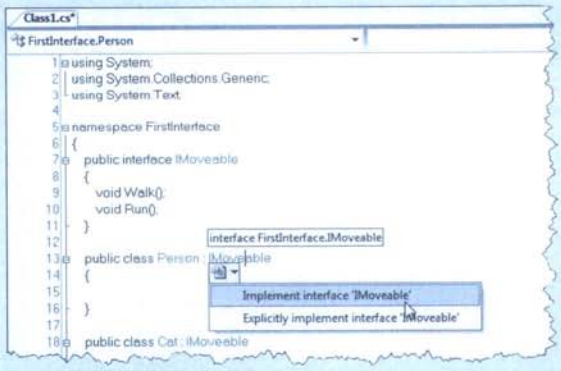
```
    }

public class Cat : IMoveable
{
    public void Walk()
    {

    }

    public void Run()
    {

    }
}
```

หลังจากที่คุณพิมพ์คำสั่ง Implements IMoveable ใน VB 2005 แล้วกดปุ่ม <Enter> พบว่า VS 2005 จะกำหนดให้คุณต้องเขียนโค้ดให้กับซับรูทีน Walk() และซับรูทีน Run() ตามข้อกำหนดที่อยู่ในอินเตอร์เฟส IMoveable

NOTE

ส่วน VC# 2005 ให้คุณเอาเมาส์ไปโฟกัสที่อินเตอร์เฟส IMoveable จะมีตัวช่วยเหลือในข้างต้นนี้ให้คุณเลือกคำสั่ง Implements Interface ไปก่อน ดังรูปที่ 11-1

รูปที่ 11-1
คำสั่งช่วยเหลือของ VC# 2005



ความแตกต่างระหว่างคนกับแมวก็คือ คนเดินและวิ่ง 2 ขา ส่วนแมวเดินและวิ่ง 4 ขา กล่าวได้อีกนัยหนึ่งคือ ซับรูทีน Walk() และ Run() ของคลาส Person ทำงานแตกต่างจากซับรูทีน Walk() และ Run() ของคลาส Cat โค้ดทั้งหมดของตัวอย่างที่ 11-1 แสดงดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 11-1 ทำความรู้จักกับ Interface

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Interface IMoveable
    Sub Walk()
    Sub Run()
End Interface

Public Class Person
    Implements IMoveable

    Public Sub Walk() Implements IMoveable.Walk
        MessageBox.Show("คนเดิน 2 ขา")
    End Sub

    Public Sub Run() Implements IMoveable.Run
        MessageBox.Show("คนวิ่ง 2 ขา")
    End Sub
End Class

Public Class Cat
    Implements IMoveable

    Public Sub Walk() Implements IMoveable.Walk
        MessageBox.Show("แมวเดิน 4 ขา")
    End Sub

    Public Sub Run() Implements IMoveable.Run
        MessageBox.Show("แมววิ่ง 4 ขา")
    End Sub
End Class
```

VC# 2005 (Class1.cs)

```
public class Person : IMoveable
{
    public void Walk()
    {
        MessageBox.Show("คนเดิน 2 ขา");
    }

    public void Run()
    {
        MessageBox.Show("คนวิ่ง 2 ขา");
    }
}

public class Cat : IMoveable
{
    public void Walk()
    {
        MessageBox.Show("แมวเดิน 4 ขา");
    }

    public void Run()
    {
        MessageBox.Show("แมววิ่ง 4 ขา");
    }
}
```

ส่วนการใช้งานคลาส Person และคลาส Cat อยู่ใน Form1 ดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 11-1 ทำความรู้จักกับ Interface

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        p.Walk()
        p.Run()

        Dim c As New Cat()
        c.Walk()
        c.Run()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
Person p = new Person();
p.Walk();
p.Run();

Cat c = new Cat();
c.Walk();
c.Run();
```

รูปที่ 11-2 ผลการทำงานของเมธอด Walk() ของออบเจ็กต์ p และ c

โค้ดข้างต้นเป็นการสร้างออบเจ็กต์ Person ที่ชื่อว่า p ขึ้นมา สั่งให้เมธอด Walk() และ Run() ทำงาน ต่อมาสร้างออบเจ็กต์ Cat ที่ชื่อว่า c ขึ้นมา สั่งให้เมธอด Walk() และ Run() ทำงานเช่นกัน พบว่าทั้งออบเจ็กต์ p และ c ทำงานแตกต่างกัน ดูได้จาก Message Box ที่ปรากฏขึ้นมานั่นเอง

รูปที่ 11-3 ผลการทำงานของเมธอด Run() ของออบเจ็กต์ p และ c

จากรูปที่ 11-2 และ 11-3 เห็นได้ว่าออบเจ็กต์ p และ c ต่างก็มีเมธอด Walk() และ Run() เป็นของตัวเอง โดยที่ทั้ง 2 เมธอดทำงานแตกต่างกัน คุณเป็นผู้กำหนดการทำงานของทั้ง 2 เมธอดเอง คำถามที่ตามมาก็คือ ทำไมเราต้องใช้อินเตอร์เฟสด้วย ทั้งๆ ที่เราสามารถสร้างเมธอด Walk() และเมธอด Run() ในคลาส Person กับคลาส Cat ให้ทำงานแตกต่างกันได้โดยตรง ซึ่งไม่จำเป็นต้องใช้อินเตอร์เฟสแต่อย่างใด

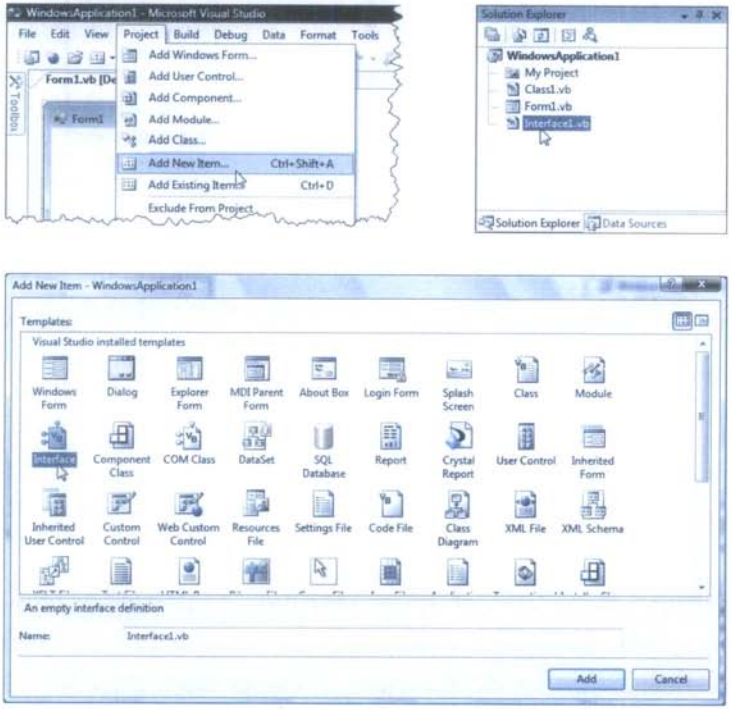
ประโยชน์ที่คุณได้จากอินเตอร์เฟสจะแสดงในเนื้อหาต่อๆ ไป ผู้เขียนขอติดค้างคำตอบนี้กับคุณผู้อ่านไว้ก่อน เพราะว่าเจตนาของตัวอย่างนี้ต้องการให้คุณรู้จักและวิธีใช้อินเตอร์เฟสในเบื้องต้นก่อนเท่านั้น

การ Implements Interface เรียกอีกอย่างหนึ่งว่า การทำ Interface Inheritance ซึ่งแตกต่างจากการสืบทอดคลาสแบบปกติ (Inheritance) ตรงที่ถ้าเป็นการ Inheritance แบบปกติ หมายถึง การ Copy ความสามารถของคลาสแม่ไปสู่คลาสลูก ส่งผลให้ที่คลาสลูก โดยคุณไม่ต้องเขียนโค้ดทำงานเอง เพราะสืบทอดคลาสมาแล้วใช้ได้เลย ให้ทำงานเหมือนเดิมหรือแก้ไขการทำงานใหม่ ก็แล้วแต่ความต้องการของคุณ

ส่วนการทำ Interface Inheritance คุณต้องเขียนโค้ดการทำงานให้กับคลาสลูกเอง ถือว่าอินเตอร์เฟสทำหน้าที่เป็นคลาสแม่ ส่วนคลาสที่ Implements Interface มาทำหน้าที่เป็นคลาสลูก จากตัวอย่างที่ผ่านมา อินเตอร์เฟส IMoveable ทำหน้าที่เป็นคลาสแม่ ส่วนคลาส Person กับคลาส Cat ทำหน้าที่เป็นคลาสลูก เพราะว่าทั้ง 2 คลาส Implements อินเตอร์เฟส IMoveable มานั่นเอง

คุณเขียนการทำงานเองในคลาสลูกที่ Implements Interface มา ซึ่งมีความหมายในเชิง OOP ก็คือ คุณเขียนทับ (Override) เมธอด Walk() และ Run() ที่อยู่ในอินเตอร์เฟส IMoveable ซึ่งไม่มีโค้ดทำงานใดๆ มาเป็นเมธอด Walk() และ Run() ที่มีการทำงาน (แสดง Message Box) ที่อยู่ในคลาส Person และคลาส Cat นั่นเอง

อีกประเด็นหนึ่งที่น่าสนใจก็คือ ผู้เขียนสร้างอินเตอร์เฟส IMoveable, คลาส Person และคลาส Cat เก็บอยู่ในไฟล์ Class1.vb (Class1.cs) เดียวกัน คุณสามารถแยกอินเตอร์เฟสที่คุณสร้างขึ้นมาออกเป็นไฟล์ต่างหากได้เช่นกัน ดังรูปที่ 11-4



รูปที่ 11-4 การเพิ่มไฟล์เก็บอินเตอร์เฟสโดยเฉพาะ

จากรูปที่ 11-4 การแยกอินเตอร์เฟสออกมาเป็นไฟล์ต่างหาก ช่วยให้โปรเจ็กต์ของคุณมีระเบียบมากยิ่งขึ้น

พื้นฐานการ Implements Interface

จากที่ผู้เขียนกล่าวไว้ว่า การ Implements Interface คือ การทำ Interface Inheritance สิ่งที่น่าสนใจก็คือ เราจะได้ประโยชน์อะไรบ้างเมื่อนำอินเตอร์เฟสมาใช้ในโลกของ OOP รวมถึงการทำ Interface Inheritance คืออะไร ให้ดูตัวอย่างที่ 11-2 พื้นฐานการ Implements Interface

โค้ด VB 2005 ที่ 11-2 พื้นฐานการ Implements Interface (Class1.vb)

```
Option Explicit On
Option Strict On

Public Interface IVBProgramable
    Sub VBProgramming()
    Sub VBDatabaseProgramming()
End Interface

Public Class Programmer
    Implements IVBProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("โปรแกรมเมอร์เขียนภาษา VB")
    End Sub

    Public Sub VBDatabaseProgramming() Implements IVBProgramable.VBDatabaseProgramming
        MessageBox.Show("โปรแกรมเมอร์เขียนภาษา VB ด้านฐานข้อมูล")
    End Sub
End Class

Public Class Person
    Implements IVBProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("คนธรรมดาเขียนภาษา VB")
    End Sub

    Public Sub VBDatabaseProgramming() Implements IVBProgramable.VBDatabaseProgramming
        MessageBox.Show("คนธรรมดาเขียนภาษา VB ด้านฐานข้อมูล")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 11-2 พื้นฐานการ Implements Interface (Class1.cs)

```
public interface IVBProgramable
{
    void VBProgramming();
    void VBDatabaseProgramming();
}

public class Programmer : IVBProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("โปรแกรมเมอร์เขียนภาษา VB");
    }

    public void VBDatabaseProgramming()
    {
        MessageBox.Show("โปรแกรมเมอร์เขียนภาษา VB ด้านฐานข้อมูล");
    }
}

public class Person : IVBProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("คนธรรมดาเขียนภาษา VB");
    }

    public void VBDatabaseProgramming()
    {
        MessageBox.Show("คนธรรมดาเขียนภาษา VB ด้านฐานข้อมูล");
    }
}
```

โค้ด VB 2005 และ VC# 2005 ที่ 11-2 พื้นฐานการ Implements Interface

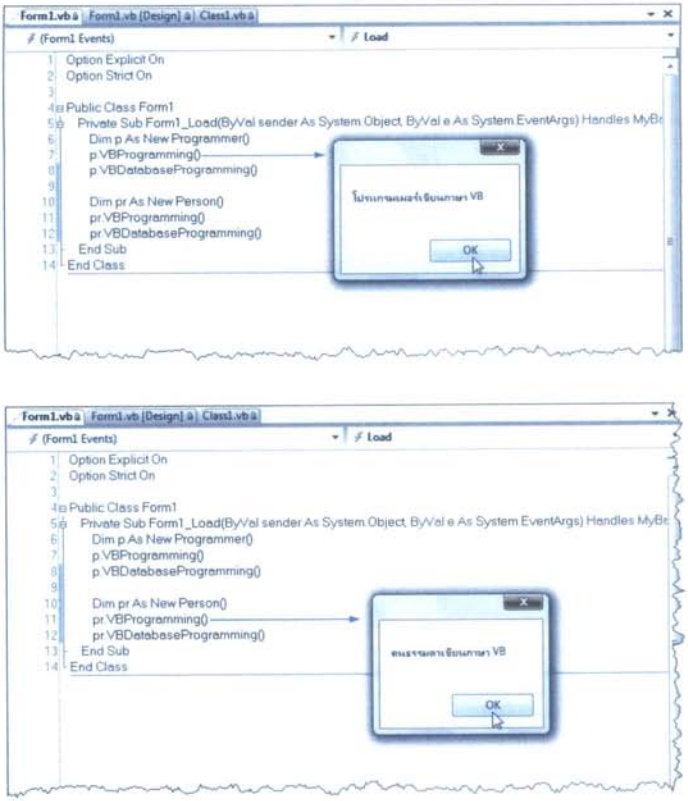
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Programmer()
        p.VBProgramming()
        p.VBDatabaseProgramming()

        Dim pr As New Person()
        pr.VBProgramming()
        pr.VBDatabaseProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer p = new Programmer();
    p.VBProgramming();
    p.VBDatabaseProgramming();

    Person pr = new Person();
    pr.VBProgramming();
    pr.VBDatabaseProgramming();
}
```

รูปที่ 11-5 ผลการทำงานของเมธอด VBProgramming() ของคลาส Programmer และคลาส Person



วิธีการดูโค้ดชุดนี้ก็คือ ผู้เขียนสร้างอินเตอร์เฟสที่ชื่อว่า IVBProgramable เงื่อนไขของอินเตอร์เฟสนี้ก็คือ บังคับไว้ว่าถ้ามีความสามารถในการเขียนโปรแกรม (IVBProgramable) ต้องมี

- เขียนภาษา VB ได้ (ต้องมีซับรูทีน VBProgramming())
- เขียนโปรแกรมด้านฐานข้อมูลด้วย VB ได้ (ต้องมีซับรูทีน VBDatabaseProgramming())

VB 2005

```
Public Interface IVBProgramable
    Sub VBProgramming()
    Sub VBDatabaseProgramming()
End Interface
```

VC# 2005

```
public interface IVBProgramable
{
    void VBProgramming();
    void VBDatabaseProgramming();
}
```

ต่อมาผู้เขียนสร้างคลาส Programmer และคลาส Person ขึ้นมา กำหนดให้ทั้ง 2 คลาส Implements อินเตอร์เฟส IVBProgramable มาด้วย ส่งผลให้ทั้งคนธรรมดา (คลาส Person) และโปรแกรมเมอร์ (คลาส Programmer) ต่างก็เขียนภาษา VB (ซับรูทีน VBProgramming()) และเขียนโปรแกรมด้านฐานข้อมูล (ซับรูทีน VBDatabaseProgramming()) ได้

อินเตอร์เฟส IVBProgramable ทำหน้าที่เป็นคลาสแม่ ส่วนคลาส Programmer และคลาส Person ทำหน้าที่เป็นคลาสลูก ซึ่งเป็นการทำ Interface Inheritance ความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูกที่เกิดขึ้นเรียกว่า ความสัมพันธ์แบบ Has-a Relationship ไม่เหมือนกับการทำ Inheritance แบบปกติที่เรียกว่า Is-a Relationship ข้อแตกต่างระหว่างความสัมพันธ์ทั้ง 2 อยู่ที่วิธีการมองกล่าวคือ

ถ้าคุณสร้างคลาส Person เป็นคลาสแม่ และสร้างคลาสลูก Programmer สืบทอด (Inheritance) มาจากคลาสแม่ เราถือว่าโปรแกรมเมอร์ (คลาส Programmer) คือ (Is-a) คนธรรมดา (คลาส Person)

แต่ถ้าคุณสร้างอินเตอร์เฟส IVBProgramable ทำหน้าที่เป็นคลาสแม่ และสร้างคลาสลูก Programmer และคลาส Person ขึ้นมา กำหนดให้ทั้ง 2 คลาส Implements อินเตอร์เฟส IVBProgramable (Interface Inheritance) เราถือว่า

- โปรแกรมเมอร์ (คลาส Programmer) มีความสามารถ (Has-a) ในการเขียนโปรแกรม (ตามเงื่อนไขที่ระบุในอินเตอร์เฟส IVBProgramable)
- คนธรรมดา (คลาส Person) มีความสามารถ (Has-a) ในการเขียนโปรแกรมเช่นกัน (อินเตอร์เฟส IVBProgramable)

จะเห็นได้ว่าความสัมพันธ์ระหว่าง 2 คลาส เกิดจากการมีความสามารถเช่นเดียวกันกล่าวคือ มีอินเตอร์เฟส IVBProgramable นั่นเอง เราจะใช้ประโยชน์ตรงจุดนี้ ไปช่วยให้การเขียนโปรแกรมในโลกของ OOP ยืดหยุ่นมากยิ่งขึ้น

การระบุคุณสมบัติในอินเตอร์เฟส

การใช้งานอินเตอร์เฟสที่ผ่านมาเป็นเพียงการใช้ member function ประเภทเมธอดเพียงอย่างเดียวเท่านั้น คุณสามารถใช้ member function ประเภทคุณสมบัติระบุเป็นข้อบังคับในอินเตอร์เฟสได้อีกด้วย ให้ดูตัวอย่างที่ 11-3 การระบุคุณสมบัติในอินเตอร์เฟสเพิ่มเติม โดยที่ตัวอย่างนี้ผู้เขียนแยกอินเตอร์เฟสออกเป็นไฟล์ต่างหากด้วย

โค้ด VB 2005 และ VC# 2005 ที่ 11-3 การระบุคุณสมบัติในอินเตอร์เฟส

VB 2005 (IVBProgramable.vb)

```
Option Strict On

Public Interface IVBProgramable
    Property FullName() As String
    Property Salary() As Integer

    Sub VBProgramming()
    Sub VBDatabaseProgramming()
End Interface

public interface ICSProgramable
```

VC# 2005 (ICSProgramable.cs)

```
{
    string FullName
    {
        get;
        set;
    }

    int Salary
    {
        get;
        set;
    }

    void CSProgramming();
    void CSDatabaseProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟสที่ชื่อว่า IVBProgramable (ICSProgramable) ขึ้นมา มีข้อบังคับไว้ว่า

- ต้องมีคุณสมบัติ FullName, คุณสมบัติ Salary กำหนดให้สามารถอ่านค่า (Get) และกำหนดค่า (Set) คุณสมบัติทั้ง 2 นี้ได้ด้วย
- ต้องมีเมธอด VBProgramming() (CSProgramming()) และเมธอด VBDatabaseProgramming() (CSDatabaseProgramming()) ที่น่าสนใจอยู่ตรงที่การระบุคุณสมบัติในอินเตอร์เฟส ซึ่งเห็นได้ว่ามีแต่โครงของบล็อก Get กับ Set ไม่มีการเขียนโค้ดแต่อย่างใด

โค้ด VB 2005 ที่ 11-3 การระบุคุณสมบัติในอินเตอร์เฟส (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Implements IVBProgramable

    Private _FullName As String = ""
    Private _Salary As Integer = 0

    Public Property FullName() As String Implements IVBProgramable.FullName
        Get
            Return _FullName
        End Get
        Set(ByVal value As String)
            _FullName = value
        End Set
    End Property

    Public Property Salary() As Integer Implements IVBProgramable.Salary
        Get
            Return _Salary
        End Get
        Set(ByVal value As Integer)
            _Salary = value
        End Set
    End Property

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา VB")
    End Sub

    Public Sub VBDatabaseProgramming() Implements IVBProgramable.VBDatabaseProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา VB ด้านฐานข้อมูล")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 11-3 การระบุคุณสมบัติในอินเตอร์เฟส (Class1.cs)

```
public class Programmer : ICSProgramable
{
    private string _FullName = "";
    private int _Salary = 0;

    public string FullName
    {
        get{return _FullName;}
        set{_FullName = value;}
    }

    public int Salary
    {
        get{return _Salary;}
        set{_Salary = value;}
    }

    public void CSProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา VC#");
    }

    public void CSDatabaseProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา VC# ด้านฐานข้อมูล");
    }
}
```

ผู้เขียนกำหนดให้คลาส Programmer ทำการ Implements อินเตอร์เฟส IVBProgramable (ICSProgramable) ส่งผลให้คุณต้องเขียนโค้ดให้กับคุณสมบัติ FullName, คุณสมบัติ Salary, เมธอด VBProgramming() (CSProgramming()) และเมธอด VBDatabaseProgramming() (CSDatabaseProgramming()) ทั้งหมดตามข้อบังคับที่อยู่ในอินเตอร์เฟสที่ Implements มานั่นเอง

โค้ด VB 2005 และ VC# 2005 ที่ 11-3 การระบุคุณสมบัติในอินเตอร์เฟส

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Programmer()
        p.FullName = "ศุภชัย สมพานิช"
        p.Salary = 10000
        p.VBProgramming()
        p.VBDatabaseProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer p = new Programmer();
    p.FullName = "ศุภชัย สมพานิช";
    p.Salary = 10000;
    p.CSProgramming();
    p.CSDatabaseProgramming();
}
```

ส่วนการใช้งานคลาส Programmer อยู่ใน Form1 เมื่อคุณ Implements อินเตอร์เฟส IVBProgramable (ICSProgramable) มา ก็จะทำให้คลาส Programmer สามารถใช้คุณสมบัติและเมธอดที่อยู่ในอินเตอร์เฟสดังกล่าวได้ตามโค้ดที่คุณเขียนไว้

การ Implements อินเตอร์เฟสของ .NET Framework

นอกจากคุณสามารถ Implements อินเตอร์เฟสที่คุณสร้างขึ้นมาเองได้แล้ว คุณยังสามารถ Implements อินเตอร์เฟสที่มากับ .NET Framework ได้อีกด้วย เพื่อให้คลาสของคุณมีความสามารถเพิ่มขึ้น ในขั้นต้นนี้ผู้เขียนขอนำเสนออินเตอร์เฟสที่น่าสนใจ 3 ตัว ได้แก่

1. อินเตอร์เฟส IComparable ทำให้คลาสของคุณสามารถเปรียบเทียบกันได้ เมื่อคลาสของคุณเปรียบเทียบได้ก็จะไปสู่การเรียงลำดับนั่นเอง
2. อินเตอร์เฟส IComparer เป็นการสร้างตัวเปรียบเทียบคลาสขึ้นมา
3. อินเตอร์เฟส ICloneable ทำให้คลาสของคุณสามารถเรียกใช้เมธอด Clone() เพื่อสำเนาคลาสได้

การ Implements อินเตอร์เฟส IComparable

ในกรณีที่คุณต้องการให้คลาสที่คุณสร้างขึ้นมาสามารถเปรียบเทียบกันได้ คุณต้อง Implements อินเตอร์เฟส IComparable ของ .NET Framework ข้อบังคับของอินเตอร์เฟส IComparable มีเพียง 1 อย่างเท่านั้นก็คือ คุณต้อง Override เมธอด CompareTo() เพื่อบอกให้เมธอดนี้รู้ว่าเราจะเอาอะไรมาเปรียบเทียบกัน ให้ดูตัวอย่างที่ 11-4 การ Implements อินเตอร์เฟส IComparable เพิ่มเติม

โค้ด VB 2005 ที่ 11-4 การ Implements อินเตอร์เฟส IComparable (Customer.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
    Implements IComparable

    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Sub New(ByVal FirstName As String, ByVal LastName As String)
        Me._FirstName = FirstName
        Me._LastName = LastName
    End Sub

    Public Overrides Function ToString() As String
        Return _FirstName + " " + _LastName
    End Function
```

```
    Public Function CompareTo(ByVal MyObject As Object) As Integer Implements IComparable.CompareTo
        Dim NextCustomer As Customer
        NextCustomer = DirectCast(MyObject, Customer)


        Return String.Compare(Me.ToString(), NextCustomer.ToString())
    End Function
End Class
```

โค้ด VC# 2005 ที่ 11-4 การ Implements อินเตอร์เฟส IComparable (Customer.cs)

```
public class Customer : IComparable
{
    private string _FirstName = "";
    private string _LastName = "";

    public Customer(string FirstName, string LastName)
    {
        this._FirstName = FirstName;
        this._LastName = LastName;
    }

    public override string ToString()
    {
        return _FirstName + " " + _LastName;
    }

    public int CompareTo(object MyObject)
    {
        Customer NextCustomer;
        NextCustomer = ((Customer)(MyObject));

        return string.Compare(this.ToString(), NextCustomer.ToString());
    }
}
```

โค้ดข้างต้นผู้เขียนสร้างคลาสที่ชื่อว่า Customer ขึ้นมา กำหนดให้ Implements อินเตอร์เฟส IComparable ของ .NET Framework ด้วย ต่อมาสร้างฟิลด์ที่ชื่อว่า _FirstName และฟิลด์ _LastName ทำหน้าที่เก็บชื่อ-สกุลตามลำดับ มีขอบเขตเป็นแบบ Private

ผู้เขียนเลือกใช้คอนสตรัคเตอร์แบบมีการรับพารามิเตอร์ เพื่อบังคับให้การสร้างออบเจ็กต์ Customer ต้องส่งชื่อ-สกุลเข้ามาด้วย เพื่อนำไปใช้เปรียบเทียบนั่นเอง ให้เก็บชื่อ-สกุลที่ส่งเข้ามาไว้ที่ฟิลด์ _FirstName และฟิลด์ _LastName ตามลำดับ

VB 2005

```
Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
    Me._FirstName = FirstName
    Me._LastName = LastName
End Sub
```

VC# 2005

```
public Customer(string FirstName, string LastName)
{
    this._FirstName = FirstName;
    this._LastName = LastName;
}
```

ต่อมาผู้เขียน Override เมธอด ToString() ใหม่ โดยการกำหนดให้เมธอดนี้คืนค่าเป็นชื่อ-นามสกุล ตามลำดับ (หรือนามสกุล-ชื่อก็ได้ แล้วแต่ความต้องการ) ส่งผลให้เมื่อมีการใช้เมธอด ToString() ของออบเจ็กต์ Customer ผลที่ได้คือ ชื่อ-สกุลลูกค้าของออบเจ็กต์ Customer ปัจจุบัน

VB 2005

```
Public Overrides Function ToString() As String
    Return _FirstName + " " + _LastName
End Function
```

VC# 2005

```
public override string ToString()
{
    return _FirstName + " " + _LastName;
}
```

หัวใจสำคัญอยู่ที่การ Override เมธอด CompareTo() ที่มากับอินเตอร์เฟส IComparable นั่นเอง เมธอดนี้ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น Object ซึ่งหมายถึง รับออบเจ็กต์ใดๆ ก็ได้ที่เป็นของ .NET Framework

วิธีคิดก็คือ เราจะนำออบเจ็กต์ Customer ที่ถูกส่งเข้ามาทางพารามิเตอร์ MyObject ของเมธอด CompareTo() ให้คุณแปลงชนิดของข้อมูลจาก Object Type มาเป็น Customer Type ใน VB 2005 ใช้ฟังก์ชัน DirectCast() ส่วน VC# 2005 ให้ทำ Explicit Conversion (หรือการทำ Cast)

จากนั้นนำออบเจ็กต์ Customer ที่ชื่อว่า NextCustomer (ที่ผ่านการแปลง Type มาแล้ว) ไปเปรียบเทียบกับออบเจ็กต์ Customer ปัจจุบัน (Me.ToString()) โดยการใช้เมธอด Compare() ของ String เข้ามาช่วย

VB 2005

```
Public Function CompareTo(ByVal MyObject As Object) As Integer Implements IComparable.CompareTo
    Dim NextCustomer As Customer
    NextCustomer = DirectCast(MyObject, Customer)

    Return String.Compare(Me.ToString(), NextCustomer.ToString())
End Function
```

VC# 2005

```
public int CompareTo(object MyObject)
{
    Customer NextCustomer;
    NextCustomer = ((Customer)(MyObject));

    return string.Compare(this.ToString(), NextCustomer.ToString());
}
```

เหตุผลที่ใช้เมธอด Compare() ของ String เข้ามาเปรียบเทียบคลาส Customer ก็คือ เราเป็นผู้กำหนดให้ใช้ชื่อ-สกุลเป็นเงื่อนไขในการเปรียบเทียบ โดยอาศัยผลจากการทำ Override เมธอด ToString() เข้ามาช่วยอีกทีหนึ่งนั่นเอง

ถึงจุดนี้คลาส Customer สามารถเปรียบเทียบกันได้แล้ว โดยใช้ชื่อ-สกุลเป็นเงื่อนไขในการเปรียบเทียบ ส่วนการใช้งานคลาส Customer อยู่ใน Form1

โค้ด VB 2005 ที่ 11-4 การ Implements อินเตอร์เฟส IComparable (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim CCustomer(4) As Customer
        CCustomer(0) = New Customer("A", "AAA")
        CCustomer(1) = New Customer("C", "CCC")
        CCustomer(2) = New Customer("Z", "ZZZ")
        CCustomer(3) = New Customer("X", "XXX")
        CCustomer(4) = New Customer("B", "BBB")

        Array.Sort(CCustomer)

        Dim i As Integer
        Dim result As String = ""
        For i = 0 To CCustomer.Length - 1
            result &= "ลำดับที่ : " & i + 1 & " " & CCustomer(i).ToString() & Environment.NewLine
        Next
        MessageBox.Show(result, "ผลการเรียงลำดับ")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 11-4 การ Implements อินเตอร์เฟส IComparable (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Customer[] CCustomer = new Customer[5];
        CCustomer[0] = new Customer("A", "AAA");
        CCustomer[1] = new Customer("C", "CCC");
        CCustomer[2] = new Customer("Z", "ZZZ");
        CCustomer[3] = new Customer("X", "XXX");
        CCustomer[4] = new Customer("B", "BBB");

        Array.Sort(CCustomer);

        string result = "";
        for (int i = 0; i <= CCustomer.Length-1; i++)
        {
```

```
            result += "ลำดับที่ : " + (i + 1) + " " + CCustomer[i].ToString() + Environment.NewLine;
        }

        MessageBox.Show(result, "ผลการเรียงลำดับ");
    }
```

วิธีการดูโค้ดชุดนี้ก็คือ ผู้เขียนสร้างอาร์เรย์ของออบเจ็กต์ Customer ที่ชื่อว่า CCustomer ขึ้นมา 5 ตัว ผู้เขียนจงใจส่งชื่อ-สกุลลูกค้าเป็นตัวอักษรภาษาอังกฤษ เพื่อให้ดูผลการทำงานสะดวกยิ่งขึ้น และไม่มีการเรียงลำดับชื่อ-สกุลแต่อย่างใด

VB 2005

```
Dim CCustomer(4) As Customer
CCustomer(0) = New Customer("A", "AAA")
CCustomer(1) = New Customer("C", "CCC")
CCustomer(2) = New Customer("Z", "ZZZ")
CCustomer(3) = New Customer("X", "XXX")
CCustomer(4) = New Customer("B", "BBB")
```

VC# 2005

```
Customer[] CCustomer = new Customer[5];
CCustomer[0] = new Customer("A", "AAA");
CCustomer[1] = new Customer("C", "CCC");
CCustomer[2] = new Customer("Z", "ZZZ");
CCustomer[3] = new Customer("X", "XXX");
CCustomer[4] = new Customer("B", "BBB");
```

เพราะความที่คลาส Customer สามารถเปรียบเทียบได้จึงนำไปสู่การเรียงลำดับนั่นเอง การเรียงลำดับอาร์เรย์เราจะใช้เมธอด Sort() ของคลาส Array เข้ามาช่วย ผลการทำงานที่ได้แสดงดังรูปที่ 11-6

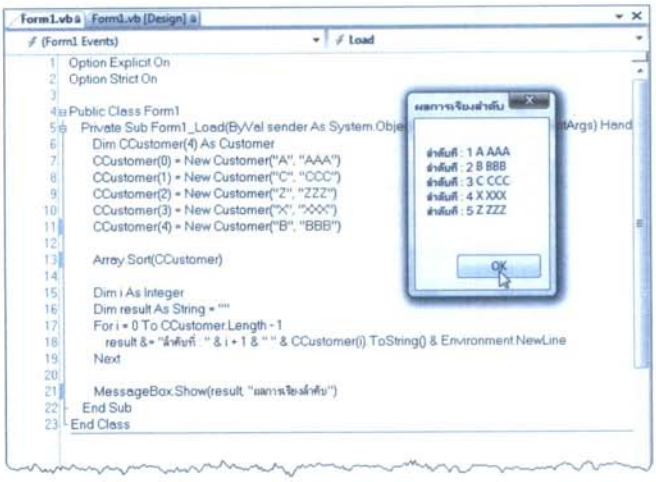
VB 2005

```
Array.Sort(CCustomer)
```

VC# 2005

```
Array.Sort(CCustomer);
```

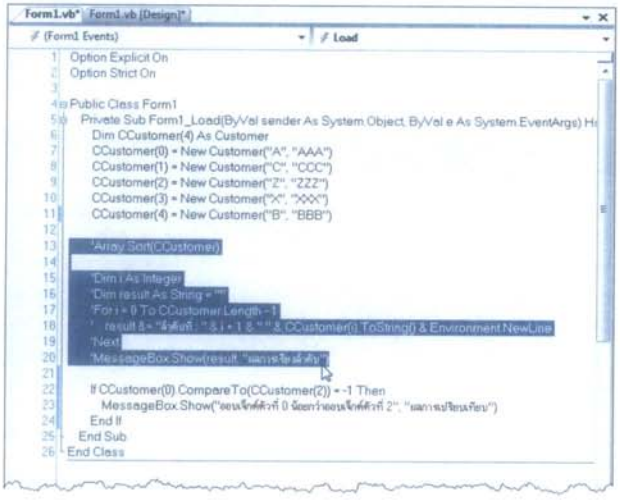
รูปที่ 11-6 ผลการทำงานตัวอย่างที่ 11-4

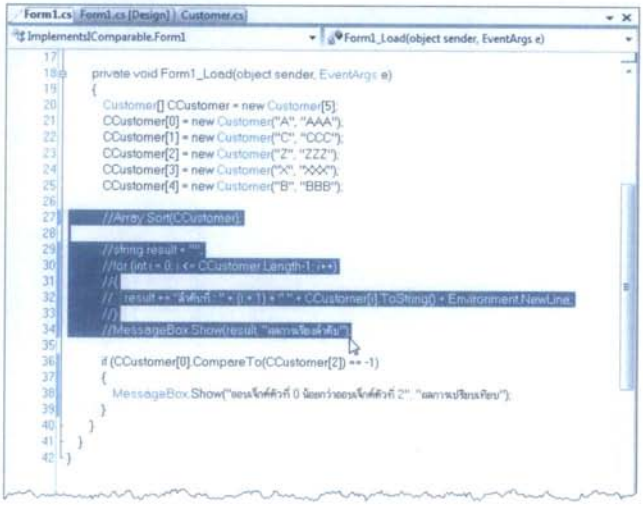


จากรูปที่ 11-6 อาร์เรย์ของออบเจ็กต์ Customer ที่ชื่อว่า CCustomer เรียงลำดับตามชื่อ-สกุลตรงตามเงื่อนไขที่เราเป็นผู้กำหนดไว้ในเมธอด CompareTo() ของคลาส Customer นั่นเอง

ที่น่าสนใจอีกอย่างก็คือ วิธีการทำงานของเมธอด CompareTo() ให้คุณทำหมายเหตุโค้ดบรรทัดที่ 13-20 ของ VB 2005 หรือบรรทัดที่ 27-34 ของ VC# 2005 เพื่อยกเลิกการเรียงลำดับออบเจ็กต์ CCustomer

รูปที่ 11-7 การทำหมายเหตุในโค้ดของ VB 2005 และ VC# 2005





จากนั้นให้คุณเพิ่มโค้ดดังต่อไปนี้

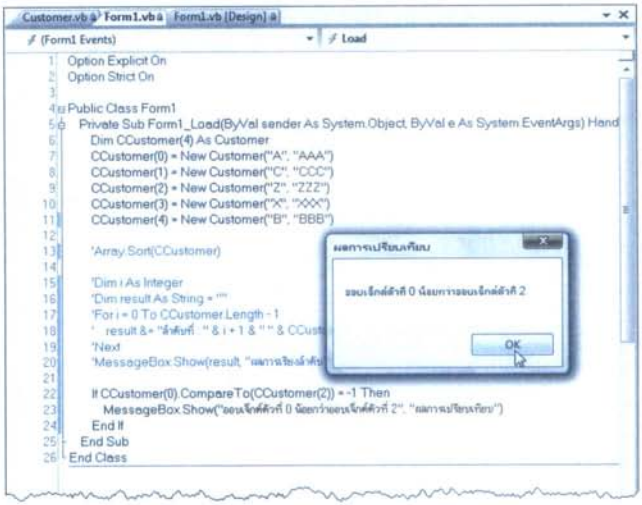
VB 2005

```
If CCustomer(0).CompareTo(CCustomer(2)) = -1 Then
    MessageBox.Show("ออบเจ็กต์ตัวที่ 0 น้อยกว่าออบเจ็กต์ตัวที่ 2", "ผลการเปรียบเทียบ")
End If
```

VC# 2005

```
if (CCustomer[0].CompareTo(CCustomer[2]) == -1)
{
    MessageBox.Show("ออบเจ็กต์ตัวที่ 0 น้อยกว่าออบเจ็กต์ตัวที่ 2", "ผลการเปรียบเทียบ");
}
```

รูปที่ 11-8 ผลการเปรียบเทียบออบเจ็กต์ Customer ตัวที่ 1 กับตัวที่ 3



โค้ดข้างต้นเป็นการนำออบเจ็กต์ Customer ตัวแรก (ลำดับอ้างอิง 0) ที่ชื่อว่า A นามสกุล AAA กับออบเจ็กต์ Customer ตัวที่ 3 (ลำดับอ้างอิง 2) ชื่อว่า Z นามสกุล ZZZ มาเปรียบเทียบกันว่าน้อยกว่าใช่หรือไม่ (-1) คำตอบที่ได้คือ ใช่ เพราะว่าตัวอักษร A มาก่อนตัวอักษร Z

ถ้าคุณต้องการเปรียบเทียบเท่ากันให้เปลี่ยนเงื่อนไขจาก -1 เป็น 0 แต่ถ้าคุณต้องการเปรียบเทียบมากกว่าให้เปลี่ยนจาก -1 เป็น 1

การ Implements อินเตอร์เฟส IComparer

ยังมีการเปรียบเทียบออบเจ็กต์อีกลักษณะหนึ่ง โดยการ Implements อินเตอร์เฟส IComparer วิธีนี้แตกต่างจากวิธีที่ผ่านมาตรงที่ เราจะสร้าง "คนกลาง" เข้ามาทำหน้าที่เปรียบเทียบออบเจ็กต์ตามเงื่อนไขที่เราระบุ ดังตัวอย่างที่ 11-5

โค้ด VB 2005 และ VC# 2005 ที่ 11-5 การ Implements อินเตอร์เฟส IComparer

VB 2005 (Customer.vb)

```
Option Explicit On
Option Strict On

Public Class Customer
    Private _FirstName As String = ""
    Private _LastName As String = ""

    Public Sub New(ByVal FirstName As String, ByVal
LastName As String)
        _FirstName = FirstName
        _LastName = LastName
    End Sub

    Public Property FirstName() As String
        Get
            Return _FirstName
        End Get
        Set(ByVal value As String)
            _FirstName = value
        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal value As String)
            _LastName = value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return _FirstName + " " + _LastName
    End Function
End Class
```

VC# 2005 (Customer.cs)

```
public class Customer
{
    private string _FirstName = "";
    private string _LastName = "";

    public Customer(string FirstName, string LastName)
    {
        _FirstName = FirstName;
        _LastName = LastName;
    }

    public string FirstName
    {
        get{return _FirstName;}
        set{_FirstName = value;}
    }

    public string LastName
    {
        get{return _LastName;}
        set{_LastName = value;}
    }

    public override string ToString()
    {
        return _FirstName + " " + _LastName;
    }
}
```

ผู้เขียนสร้างคลาส Customer ขึ้นมา ประกอบด้วยคุณสมบัติ FirstName และคุณสมบัติ LastName เก็บชื่อ-สกุลที่มีการทำ Override เมธอด ToString() เช่นเดิม แต่ไม่มีการ Implements อินเตอร์เฟสใดๆ ทั้งสิ้น

ต่อมาสร้างอีกคลาสหนึ่งชื่อว่า CustomerComparer ทำหน้าที่เป็นคนกลางคอยเปรียบเทียบออบเจ็กต์ Customer เท่านั้น คุณต้องกำหนดให้คลาส CustomerComparer ต้องไป Implements อินเตอร์เฟส IComparer มาด้วย ซึ่งมีเงื่อนไขเพียง 1 อย่างคือ คุณต้อง Override เมธอด Compare()

โค้ด VB 2005 ที่ 11-5 การ Implements อินเตอร์เฟส IComparer (CustomerComparer.vb)

```
Option Explicit On
Option Strict On

Public Class CustomerComparer
    Implements IComparer

    Public Function Compare(ByVal Customer1 As Object, ByVal Customer2 As Object) As Integer Implements IComparer.Compare
        Dim x As Customer = DirectCast(Customer1, Customer)
        Dim y As Customer = DirectCast(Customer2, Customer)


        Return String.Compare(x.ToString(), y.ToString())
    End Function
End Class
```

โค้ด VC# 2005 ที่ 11-5 การ Implements อินเตอร์เฟส IComparer (CustomerComparer.cs)

```
public class CustomerComparer : IComparer
{
    public int Compare(object Customer1, object Customer2)
    {
        Customer x = (Customer)Customer1;
        Customer y = (Customer)Customer2;

        return string.Compare(x.ToString(), y.ToString());
    }
}
```

เมธอด Compare() ต้องการพารามิเตอร์ 2 ตัวมี Type เป็น Object วิธีการก็คือ

- ในกรณี VB 2005 ให้ใช้ฟังก์ชัน DirectCast() แปลงข้อมูลจาก Object Type มาเป็น Customer Type ให้กับพารามิเตอร์ทั้ง 2 ตัว
- ส่วน VC# 2005 ใช้วิธีการแปลงข้อมูลแบบ Explicit Conversion (Cast) จาก Object Type มาเป็น Customer Type เช่นกัน ผลการแปลงพารามิเตอร์ทั้ง 2 ก็จะกำหนดให้ตัวแปร x และตัวแปร y ที่สร้างขึ้นมาใหม่ชี้ไว้

ท้ายที่สุดใช้เมธอด Compare() ของ String เปรียบเทียบออบเจ็กต์ Customer ทั้ง 2 ตัว โดยอาศัยชื่อ-สกุลเป็นเงื่อนไขในการเปรียบเทียบ ผ่านทางเมธอด ToString() ของออบเจ็กต์ Customer ที่เรา Override ไว้นั่นเอง ส่วนการใช้งานคลาส Customer และ CustomerComparer อยู่ใน Form1 ดังนี้

โค้ด VB 2005 ที่ 11-5 การ Implements อินเตอร์เฟส IComparer (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim cc As New CustomerComparer()
        Dim arrCustomer(1) As Customer
        arrCustomer(0) = New Customer("ศุภชัย", "สมพานิช")
        arrCustomer(1) = New Customer("ณัฐพล", "กิจประชา")
        Array.Sort(arrCustomer, cc)

        Dim i As Integer = 0
        Dim CustomerAfterSort As String = ""
        For i = 0 To arrCustomer.Length - 1
            CustomerAfterSort &= arrCustomer(i).ToString() & Environment.NewLine
        Next
        MessageBox.Show(CustomerAfterSort, "ผลการเรียงลำดับ")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 11-5 การ Implements อินเตอร์เฟส IComparer (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        CustomerComparer cc = new CustomerComparer();
        Customer[] arrCustomer = new Customer[2];
        arrCustomer[0] = new Customer("ศุภชัย", "สมพานิช");
        arrCustomer[1] = new Customer("ณัฐพล", "กิจประชา");
        Array.Sort(arrCustomer, cc);

        string CustomerAfterSort = "";
        for (int i = 0; i <= arrCustomer.Length - 1; i++)
        {
            CustomerAfterSort += arrCustomer[i].ToString() + Environment.NewLine;
        }
        MessageBox.Show(CustomerAfterSort, "ผลการเรียงลำดับ");
    }
```

วิธีการดูโค้ดนี้ก็คือ สร้างออบเจ็กต์ CustomerComparer ที่ชื่อว่า cc ทำหน้าที่เป็นผู้เปรียบเทียบออบเจ็กต์ Customer และสร้างอาร์เรย์ออบเจ็กต์ Customer ที่ชื่อว่า arrCustomer โดยมีสมาชิก 2 ตัว ผู้เขียนจงใจให้สมาชิกตัวแรกเป็นชื่อผู้เขียน (ตัวอักษร ศ) ส่วนสมาชิกที่ 2 เป็นอักษร ณ ซึ่งเป็นตัวอักษรที่มาก่อน ศ เพื่อแสดงให้เห็นว่าสมาชิกในอาร์เรย์ arrCustomer ไม่มีการเรียงลำดับใดๆ

VB 2005

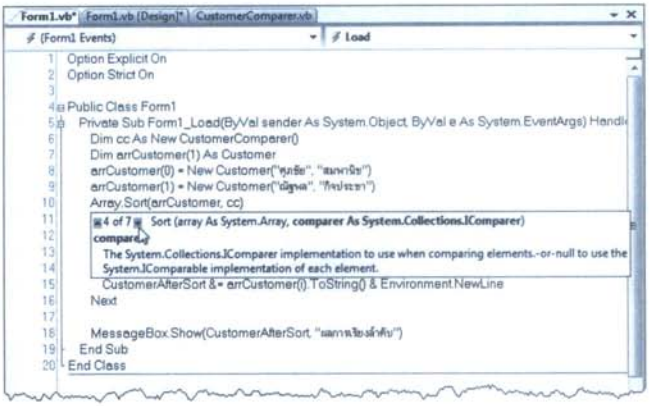
```
Dim cc As New CustomerComparer()
Dim arrCustomer(1) As Customer
arrCustomer(0) = New Customer("ศุภชัย", "สมพานิช")
arrCustomer(1) = New Customer("ณัฐพล", "กิจประชา")
Array.Sort(arrCustomer, cc)
```

VC# 2005

```
CustomerComparer cc = new CustomerComparer();
Customer[] arrCustomer = new Customer[2];
arrCustomer[0] = new Customer("ศุภชัย", "สมพานิช");
arrCustomer[1] = new Customer("ณัฐพล", "กิจประชา");
Array.Sort(arrCustomer, cc);
```

การเรียงลำดับอาร์เรย์เราจะใช้เมธอด Sort() ของคลาส Array เช่นเดิม เพียงแต่ว่าคุณไม่สามารถเรียงลำดับอาร์เรย์ arrCustomer โดยตรง เพราะว่าคลาส Customer ของตัวอย่างนี้เปรียบเทียบไม่ได้ ต้องอาศัยคนกลางเข้ามาช่วยนั่นคือ ออบเจ็กต์ CustomerComparer ที่ชื่อว่า cc เข้ามาช่วยเปรียบเทียบนั่นเอง

รูปที่ 11-9 เมธอด Sort() แบบที่ 4 ของคลาส Array



จากรูปที่ 11-9 เมธอด Sort() แบบที่ 4 ของคลาส Array ระบุไว้ว่าต้องการพารามิเตอร์ 2 ตัวคือ

- อาร์เรย์ที่ต้องการเรียงลำดับ (array)
- ตัวเปรียบเทียบ (comparer) ที่ต้อง Implements อินเตอร์เฟส IComparer

ท้ายที่สุดก็จะสั่งให้อ่านค่าสมาชิกที่อยู่ในตัวแปรอาร์เรย์ arrCustomer ออกมา เพื่อดูผลการเปรียบเทียบและเรียงลำดับ ดังรูปที่ 11-10

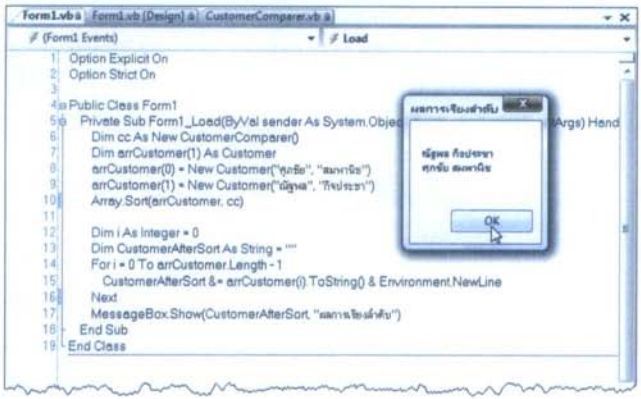
VB 2005

```
Dim CustomerAfterSort As String = ""
For i = 0 To arrCustomer.Length - 1
    CustomerAfterSort &= arrCustomer(i).ToString() & Environment.NewLine
Next
MessageBox.Show(CustomerAfterSort, "ผลการเรียงลำดับ")
```

VC# 2005

```
string CustomerAfterSort = "";
for (int i = 0; i <= arrCustomer.Length - 1; i++)
{
    CustomerAfterSort += arrCustomer[i].ToString() + Environment.NewLine;
}
MessageBox.Show(CustomerAfterSort, "ผลการเรียงลำดับ");
```

รูปที่ 11-10 ผลการเรียงลำดับของอาร์เรย์ arrCustomer



จากรูปที่ 11-10 ผลการเรียงลำดับตัวอักษร ณ ต้องมาก่อนตัวอักษร ศ

การ Implements อินเตอร์เฟส ICloneable

อินเตอร์เฟส ICloneable ช่วยให้คลาสของคุณมีความสามารถสำเนาข้อมูลผ่านทางเมธอด Clone() โดยที่คุณเป็นผู้กำหนดเองว่าต้องการทำสำเนาอะไร ให้ดูตัวอย่างที่ 11-6 การ Implements อินเตอร์เฟส ICloneable เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 11-6 การ Implements อินเตอร์เฟส ICloneable

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class MasterClass
    Implements ICloneable

    Private _str As String = ""
    Public Property str() As String
        Get
            Return _str
        End Get
        Set(ByVal value As String)
            _str = value
        End Set
    End Property

    Public Sub New(ByVal str As String)
        _str = str
    End Sub

    Public Function Clone() As Object Implements
ICloneable.Clone
        Dim _CloneString As String = ""
        _CloneString = DirectCast(_str.Clone, String)

        Return New MasterClass(_CloneString)
    End Function
End Class
```

VC# 2005 (Class1.cs)

```
public class MasterClass : ICloneable
{
    private string _str = "";
    public string str
    {
        get{return _str;}
        set{_str = value;}
    }

    public MasterClass(string str)
    {
        _str = str;
    }

    public object Clone()
    {
        string _CloneString = "";
        _CloneString = (string)_str.Clone();

        return new MasterClass(_CloneString);
    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า MasterClass ขึ้นมา มีหน้าที่แสดงข้อความจากฟิลด์ _str ผ่านทางคุณสมบัติ str เท่านั้น โดยเลือกใช้คอนสตรัคเตอร์แบบมีพารามิเตอร์ เพื่อบังคับให้ต้องส่งค่าให้กับฟิลด์ _str ในขณะที่สร้างออบเจ็กต์ด้วยคำสั่ง New (new) นั่นเอง

กำหนดให้ Implements อินเตอร์เฟส ICloneable ซึ่งมีข้อบังคับไว้ว่า คุณต้อง Override เมธอด Clone() สิ่งที่น่าสนใจอยู่ตรงที่เมธอด Clone() คืนค่าเป็น Object Type

วิธีคิดก็คือ เพราะความที่คลาส MasterClass ไม่มีการทำงานอะไรมากนัก ดังนั้น วิธีการสำเนาข้อมูลก็เพียงแต่กำหนดให้สำเนาข้อมูลที่เก็บอยู่ในฟิลด์ _str ให้กับอินสแตนซ์ใหม่ที่เกิดขึ้นมากล่าวคือ

VB 2005

```
    Public Function Clone() As Object Implements
ICloneable.Clone
        Dim _CloneString As String = ""
        _CloneString = DirectCast(_str.Clone, String)

        Return New MasterClass(_CloneString)
    End Function
```

VC# 2005

```
    public object Clone()
    {
        string _CloneString = "";
        _CloneString = (string)_str.Clone();

        return new MasterClass(_CloneString);
    }
```

เราจะแปลง Type ของค่าส่งกลับจากเมธอด Clone() (_str.Clone) ซึ่งมี Type เป็น Object ไปเป็น String Type ก่อน และเก็บไว้ที่ตัวแปร _CloneString

ท้ายที่สุดให้คืนค่าเป็นอินสแตนซ์ใหม่ของคลาส MasterClass โดยใช้คอนสตรัคเตอร์แบบมีพารามิเตอร์ที่เรากำหนดไว้ก่อนหน้านี้ ค่าที่ส่งเข้าไปคือ ค่าที่เก็บไว้ที่ตัวแปร _CloneString นั่นเอง ส่วนวิธีใช้งานคลาส MasterClass อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 11-6 การ Implements อินเตอร์เฟส ICloneable

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim A As New MasterClass("ข้อความเริ่มต้น A")
        MessageBox.Show(A.str, "ข้อความจาก A")

        Dim B As MasterClass
        B = DirectCast(A.Clone(), MasterClass)
        MessageBox.Show(B.str, "ข้อความจาก B")

        B.str = "ข้อความใหม่ B"
        MessageBox.Show(B.str, "ข้อความใหม่ B")
        MessageBox.Show(A.str, "ข้อความใหม่ A")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    MasterClass A = new MasterClass("ข้อความเริ่มต้น A");
    MessageBox.Show(A.str, "ข้อความจาก A");

    MasterClass B;
    B = (MasterClass)A.Clone();
    MessageBox.Show(B.str, "ข้อความจาก B");

    B.str = "ข้อความใหม่ B";
    MessageBox.Show(B.str, "ข้อความใหม่ B");
    MessageBox.Show(A.str, "ข้อความใหม่ A");
}
```

วิธีการดูโค้ดชุดนี้ให้คุณดูทีละส่วนกล่าวคือ เริ่มต้นสร้างออบเจ็กต์ MasterClass ที่ชื่อว่า A ขึ้นมา จากนั้นผู้เขียนส่งข้อความเริ่มต้น A เข้าไป เพื่อบ่งบอกให้รู้ว่าข้อความนี้เกิดจากออบเจ็กต์ A กล่าวได้อีกนัยหนึ่งคือ ข้อความนี้ถูกเก็บไว้ที่ฟิลด์ _str ที่อยู่ภายในคลาส MasterClass คุณสามารถอ่านค่าของฟิลด์ _str ผ่านทางคุณสมบัติ str นั่นเอง

VB 2005

```
Dim A As New MasterClass("ข้อความเริ่มต้น A")
MessageBox.Show(A.str, "ข้อความจาก A")
```

VC# 2005

```
MasterClass A = new MasterClass("ข้อความเริ่มต้น A");
MessageBox.Show(A.str, "ข้อความจาก A");
```

ต่อมาผู้เขียนสร้างตัวแปร B ขึ้นมา มี Type เป็น MasterClass ไม่มีการสร้างออบเจ็กต์ MasterClass แต่อย่างใด เพราะเราต้องการให้ตัวแปร B สำเนาข้อมูลจากออบเจ็กต์ A ผ่านทางเมธอด Clone()

เพราะความที่เมธอด Clone() คืนค่าเป็น Object Type จึงต้องแปลงข้อมูลเป็น MasterClass Type โดยที่เรากำหนดการทำงานภายในเมธอด Clone() ไว้ว่าค่าที่คืนกลับมาคือ อินสแตนซ์ใหม่ของคลาส MasterClass

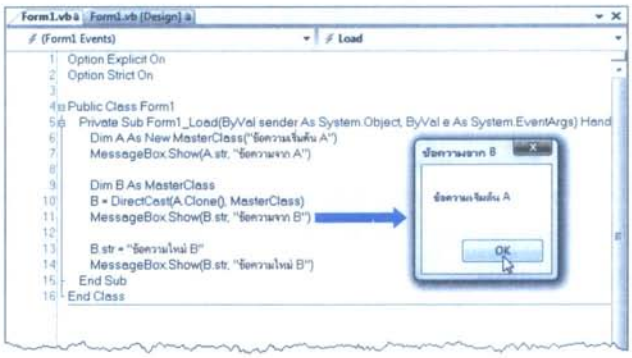
ส่งผลให้โค้ดที่คุณระบุไว้ว่าหมายถึงกำหนดให้ตัวแปร B ชี้ไปยังอินสแตนซ์ใหม่ที่เกิดจากการทำงานของเมธอด Clone() นั่นเอง ผลที่ได้แสดงดังรูปที่ 11-11

VB 2005

```
Dim B As MasterClass
B = DirectCast(A.Clone(), MasterClass)
MessageBox.Show(B.str, "ข้อความจาก B")
```

VC# 2005

```
MasterClass B;
B = (MasterClass)A.Clone();
MessageBox.Show(B.str, "ข้อความจาก B");
```



รูปที่ 11-11 ค่าที่เก็บอยู่ในออบเจ็กต์ MasterClass ที่ชื่อว่า B

จากรูปที่ 11-11 เห็นได้ว่าฟิลด์ _str ที่คุณอ่านค่าจากคุณสมบัติ str ของออบเจ็กต์ B เป็นข้อความที่ได้จากออบเจ็กต์ A นั่นเอง นี่คือ ผลการทำงานของเมธอด Clone() ที่เราเป็นผู้กำหนดเอง

สมมติว่าคุณกำหนดข้อความใหม่ให้กับออบเจ็กต์ B ผลที่ได้คือ อินสแตนซ์ A และอินสแตนซ์ B ของคลาส MasterClass เก็บข้อความแตกต่างกัน เพราะแต่ละอินสแตนซ์ไม่มีความเกี่ยวข้องกันแต่อย่างใด ดังรูปที่ 11-12

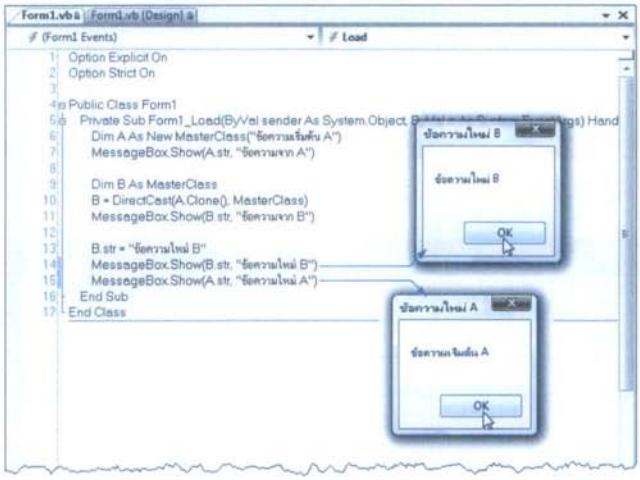
VB 2005

```
B.str = "ข้อความใหม่ B"
MessageBox.Show(B.str, "ข้อความใหม่ B")
MessageBox.Show(A.str, "ข้อความใหม่ A")
```

VC# 2005

```
B.str = "ข้อความใหม่ B";
MessageBox.Show(B.str, "ข้อความใหม่ B");
MessageBox.Show(A.str, "ข้อความใหม่ A");
```

รูปที่ 11-12
ผลการทำงานหลังจากแก้ไขข้อความใหม่ของออบเจ็กต์ B



สรุปท้ายบท

เนื้อหาของบทนี้ยังคงเป็นพื้นฐานการใช้งานอินเตอร์เฟส ไม่ว่าจะเป็นการ Implements อินเตอร์เฟสที่สร้างขึ้นมาเอง หรืออินเตอร์เฟสที่มากับ .NET Framework ที่ผู้เขียนยกตัวอย่างมา 3 ชนิด ในบทต่อไปจะเข้าไปสู่โลกของอินเตอร์เฟสอย่างเต็มตัว

CHAPTER ▸ 12

Interface Programming

บทนำ

เนื้อหาของบทนี้จะกล่าวถึงรูปแบบการใช้งานอินเตอร์เฟส รวมถึงกติกา ข้อบังคับ และข้อควรรับทราบของอินเตอร์เฟส ซึ่งมีรายละเอียดค่อนข้างมาก ขอให้คุณผู้อ่านให้น้ำหนักกับจุดเด่นของแต่ละตัวอย่างเป็นสำคัญ

Interface Programming

ผู้เขียนขอสรุปภาพรวมของข้อแตกต่างระหว่างการสืบทอดคลาสแบบปกติ (Inheritance) กับ Interface Inheritance ที่น่าสนใจ ก่อนที่จะเข้าสู่หาเนื้อหาของ Interface Programming

การสืบทอดคลาสแบบปกติ (Inheritance) คือ การ Copy ความสามารถของคลาสแม่มาสู่คลาสลูก สิ่งที่คลาสลูกได้จากคลาสแม่มาก็คือ ส่วนประกอบที่มีขอบเขต Protected, Public, Default Constructor และ Shared (static) Constructor โดยที่คลาสลูกสามารถเพิ่มความสามารถใหม่ หรือแก้ไขความสามารถเดิมที่ได้จากคลาสแม่ เรียกว่า Single Inheritance ความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูกที่เกิดขึ้น เรียกว่า ความสัมพันธ์แบบ Is-a Relationship

ส่วน Interface Inheritance หมายถึง การที่คลาสใดๆ ก็ตามที่ Implements อินเตอร์เฟสมาแล้ว ต้องเขียนการทำงานดังกล่าวเอง ส่งผลให้คลาสลูกแต่ละคลาสอาจจะทำงานแตกต่างกันก็ได้ แม้ว่าจะมีชื่อเมธอดเหมือนกัน โดยมองว่าอินเตอร์เฟสอยู่ในฐานะคลาสแม่ ส่วนคลาสที่ Implements อินเตอร์เฟสดังกล่าวไปอยู่ในฐานะคลาสลูก เราเรียกความสัมพันธ์แบบนี้ว่า Has-a Relationship

รูปแบบการ Implements อินเตอร์เฟส

การ Implements อินเตอร์เฟสมี 2 รูปแบบคือ

- หลายคลาส Implements อินเตอร์เฟสเดียวกัน
- 1 คลาส Implements หลายอินเตอร์เฟส

หลายคลาส Implements อินเตอร์เฟสเดียวกัน

ถ้ากล่าวในแง่ของการออกแบบแล้ว หมายถึง แต่ละคลาสมีชื่อเมธอด (หรืออื่นๆ) เหมือนกัน แต่ทำงานแตกต่างกัน ให้ดูตัวอย่างที่ 12-1 หลายคลาส Implements อินเตอร์เฟสเดียวกันเพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 12-1 หลายคลาส Implements อินเตอร์เฟสเดียวกัน

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IProgramable
    Sub DatabaseProgramming()
    Sub OOPProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IProgramable
{
    void DatabaseProgramming();
    void OOPProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟสที่ชื่อว่า IProgramable หมายถึง ความสามารถในการเขียนโปรแกรม มีข้อบังคับไว้ว่า

- ต้องเขียนโปรแกรมด้านฐานข้อมูลได้ (มีเมธอด DatabaseProgramming())
- ต้องเขียนโปรแกรมแบบ OOP ได้อีกด้วย (มีเมธอด OOPProgramming())

โค้ด VB 2005 ที่ 12-1 หลายคลาส Implements อินเตอร์เฟสเดียวกัน (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Implements IProgramable

    Public Sub DatabaseProgramming() Implements IProgramable.DatabaseProgramming
        MessageBox.Show("เขียนโปรแกรมด้านฐานข้อมูลด้วย VB 2005")
    End Sub

    Public Sub OOPProgramming() Implements IProgramable.OOPProgramming
        MessageBox.Show("เขียน OOP ด้วย VB 2005")
    End Sub
End Class
```

```
Public Class Programmer
    Implements IProgramable

    Public Sub DatabaseProgramming() Implements IProgramable.DatabaseProgramming
        MessageBox.Show("เขียนโปรแกรมด้านฐานข้อมูลด้วย VC# 2005")
    End Sub

    Public Sub OOPProgramming() Implements IProgramable.OOPProgramming
        MessageBox.Show("เขียน OOP ด้วย VC# 2005")
    End Sub

    Public Sub OOPProgramming(ByVal LINQ As String)
        MessageBox.Show("เขียน OOP ด้วย VC# 2005 " & LINQ)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-1 หลายคลาส Implements อินเตอร์เฟสเดียวกัน (Class1.cs)

```
public class Person : IProgramable
{
    public void DatabaseProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมด้านฐานข้อมูลด้วย VB 2005");
    }

    public void OOPProgramming()
    {
        MessageBox.Show("เขียน OOP ด้วย VB 2005");
    }
}

public class Programmer : IProgramable
{
    public void DatabaseProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมด้านฐานข้อมูลด้วย VC# 2005");
    }

    public void OOPProgramming()
    {
        MessageBox.Show("เขียน OOP ด้วย VC# 2005");
    }

    public void OOPProgramming(string LINQ)
    {
        MessageBox.Show("เขียน OOP ด้วย VC# 2005 " + LINQ);
    }
}
```

ต่อมาผู้เขียนสร้างคลาสขึ้นมา 2 คลาสคือ คนธรรมดา (คลาส Person) และโปรแกรมเมอร์ (คลาส Programmer) กำหนดให้ทั้ง 2 คลาส Implements อินเตอร์เฟสเดียวกัน นั่นคือ IProgramable ที่น่าสนใจมีอยู่ 2 ประการคือ

1. กำหนดให้คลาส Person มีความสามารถเขียนโปรแกรมได้เช่นเดียวกับคลาส Programmer แต่ทำงานแตกต่างกัน โดยการกำหนดให้แสดง MessageBox แตกต่างกันนั่นเอง สมมติว่าคลาส Person ให้เขียนภาษา VB 2005 ส่วน Programmer ให้เขียนภาษา VC# 2005
2. เพื่อให้คลาส Programmer เก่งกว่าคลาส Person จึงได้เพิ่มเมธอด OOPProgramming() ขึ้นมาอีก 1 เมธอด โดยการทำ Overload เมธอด OOPProgramming() ที่ได้มาจากอินเตอร์เฟส IProgramable

VB 2005

```
Public Sub OOPProgramming() Implements IProgramable.OOPProgramming
    MessageBox.Show("เขียน OOP ด้วย VC# 2005")
End Sub

Public Sub OOPProgramming(ByVal LINQ As String)
    MessageBox.Show("เขียน OOP ด้วย VC# 2005 " & LINQ)
End Sub
```

VC# 2005

```
public void OOPProgramming()
{
    MessageBox.Show("เขียน OOP ด้วย VC# 2005");
}

public void OOPProgramming(string LINQ)
{
    MessageBox.Show("เขียน OOP ด้วย VC# 2005 " + LINQ);
}
```

การทำ Overload เมธอดมีเงื่อนไขคือ คุณต้องกำหนดให้เมธอด OOPProgramming() มี Signature แตกต่างไปจากเดิม สมมติว่าเมธอดใหม่ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น String (string) บอกไว้ว่า คลาส Programmer เขียน OOP ด้วย VC# 2005 และภาษา LINQ ได้อีกด้วย ส่วนการใช้งานคลาส Person และ Programmer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-1 หลายคลาส Implements อินเตอร์เฟสเดียวกัน

VB 2005 (Form1.vb)

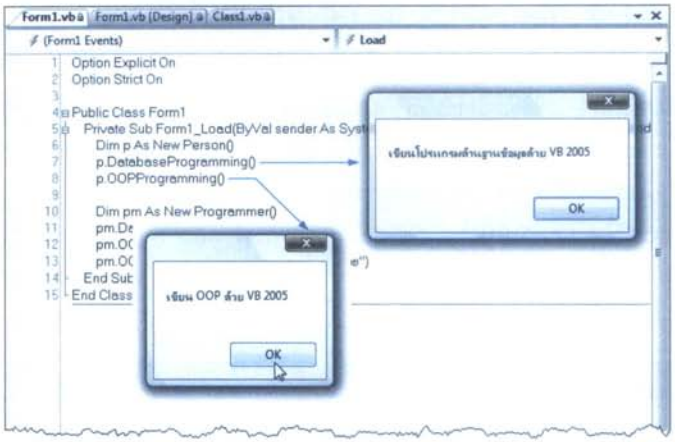
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        p.DatabaseProgramming()
        p.OOPProgramming()

        Dim pm As New Programmer()
        pm.DatabaseProgramming()
        pm.OOPProgramming()
        pm.OOPProgramming("และภาษา LINQ ได้อีกด้วย")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Person p = new Person();
    p.DatabaseProgramming();
    p.OOPProgramming();

    Programmer pm = new Programmer();
    pm.DatabaseProgramming();
    pm.OOPProgramming();
    pm.OOPProgramming("และภาษา LINQ ได้อีกด้วย");
}
```

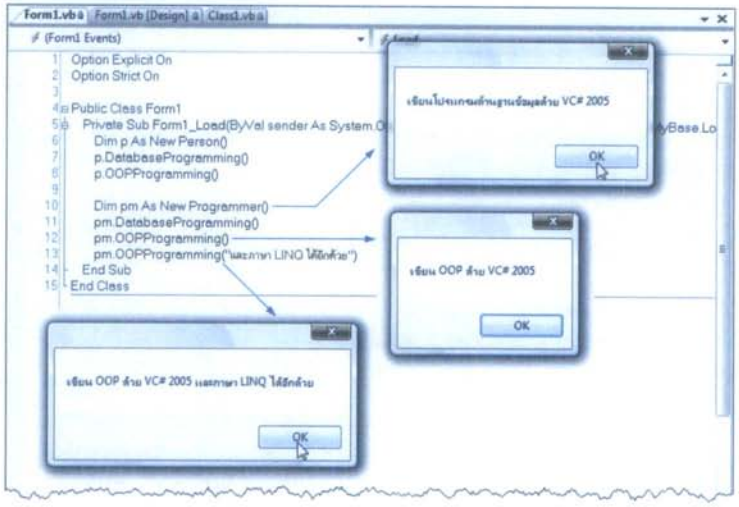
โค้ดข้างต้นมีคนธรรมดาชื่อว่า p เขียนโปรแกรมด้านฐานข้อมูล และเขียนโปรแกรมแบบ OOP ด้วยภาษา VB 2005 ดังรูปที่ 12-1

รูปที่ 12-1 ผลการทำงานของออบเจ็กต์ Person ที่ชื่อว่า p



ส่วนโปรแกรมเมอร์ที่ชื่อว่า pm สามารถเขียนโปรแกรมด้านฐานข้อมูล, เขียนโปรแกรมแบบ OOP และใช้ภาษา LINQ ด้วย VC# 2005 ได้อีกด้วย เห็นได้ว่าการทำงานของออบเจ็กต์ pm แตกต่างไปจากออบเจ็กต์ p แม้ว่าจะมีชื่อเมธอดเหมือนกันก็ตาม ดังรูปที่ 12-2

รูปที่ 12-2 ผลการทำงานของออบเจ็กต์ Programmer ที่ชื่อว่า pm



1 คลาส Implements หลายอินเตอร์เฟส

ในบางกรณีคลาสที่คุณสร้างขึ้นมา อาจจะต้องมีการ Implements อินเตอร์เฟสมากกว่า 1 อินเตอร์เฟสในเวลาเดียวกัน เรียกว่า การทำ Implement Multiple Interfaces คุณจะได้ศึกษาวิธีการแก้ปัญหา และเห็นประโยชน์ที่เกิดจากการใช้งานอินเตอร์เฟสชัดเจนยิ่งขึ้น ก่อนที่จะเข้าเนื้อหาหลักส่วนนี้ ผู้เขียนขอกล่าวถึงที่มาของการทำ Implement Multiple Interfaces เสียก่อน

คุณสมบัติหลักอย่างหนึ่งของ OOP ก็คือ การสืบทอดคลาส (Inheritance) ผู้เขียนขอจำกัดวงให้แคบลง ให้เหลือเพียงแค่ Concept การเขียน OOP ด้วยภาษา VB 2005 และ VC# 2005 คุณสามารถกำหนดให้คลาสลูกสืบทอดคลาสจากคลาสแม่ได้เพียง 1 คลาสในเวลาเดียวกัน ดังโค้ดต่อไปนี้

VB 2005	VC# 2005

Public Class Parent
 	
End Class
 	
Public Class Child
Inherits Parent
 	
End Class
	public class Parent
	{
	
	}
	
	public class Child : Parent
	{
	
	}

เราเรียกการสืบทอดคลาสข้างต้นว่า Single Inheritance ข้อจำกัดอย่างหนึ่งของการใช้ OOP ด้วยภาษา VB 2005 และ VC# 2005 ก็คือ คุณไม่สามารถกำหนดให้คลาสลูกสืบทอดคลาสจากคลาสแม่หลายๆ คลาสในเวลาเดียวกัน ดังโค้ดต่อไปนี้

VB 2005

```
Public Class Parent1

End Class

Public Class Parent2

End Class

Public Class Child
    Inherits Parent1, Parent2

End Class
```

VC# 2005

```
public class Parent1
{
}

public class Parent2
{
}

public class Child : Parent1,Parent2
{
}
```

เหตุผลสำคัญอย่างหนึ่งที่ทำให้ภาษา VB 2005 และ VC# 2005 ไม่สามารถทำ Multiple Inheritance ก็คือ สมมติว่าในคลาสแม่ Parent1 และคลาส Parent2 มีเมธอดที่ชื่อเหมือนกัน ปัญหาที่เกิดขึ้นก็คือ แล้วเมธอดใดของคลาสแม่จะถูกถ่ายทอดไปสู่คลาสลูก ถ้าทั้ง 2 เมธอดมีขอบเขตแบบ Public หรือ Protected

ผู้สร้างภาษา C# ไม่ต้องการให้เกิดความยุ่งยากดังกล่าว จึงได้ตัดความสามารถในการทำ Multiple Inheritance ออกไป แล้วใช้วิธีการทำ Implement Multiple Interfaces เข้ามาทดแทน

การทำ Implement Multiple Interfaces เป็นการกำหนดให้คลาส 1 คลาส เรียกใช้อินเตอร์เฟสหลายอินเตอร์เฟสในเวลาเดียวกัน เราจะมาดูว่าปัญหาที่เกิดขึ้นมีวิธีการแก้ไขอย่างไร ให้ดูตัวอย่างที่ 12-2 พื้นฐานการทำ Implement Multiple Interfaces ก่อน

โค้ด VB 2005 และ VC# 2005 ที่ 12-2 พื้นฐานการทำ Implement Multiple Interfaces

VB 2005 (IProgramable.vb)

```
Public Interface IVBProgramable
    Sub VBProgramming()
    Sub VBDatabaseProgramming()
End Interface

Public Interface ICSProgramable
    Sub CSProgramming()
    Sub CSOOPProgramming()
    Sub CSDatabaseProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IVBProgramable
{
    void VBProgramming();
    void VBDatabaseProgramming();
}

public interface ICSProgramable
{
    void CSProgramming();
    void CSOOPProgramming();
    void CSDatabaseProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟสขึ้นมา 2 ชุดคือ

1. ความสามารถในการเขียนภาษา VB (IVBProgramable) มีข้อบังคับไว้ว่า
   - ต้องเขียนโปรแกรมเบื้องต้นด้วยภาษา VB ได้ (ต้องมีเมธอด VBProgramming())
   - ต้องเขียนโปรแกรมด้านฐานข้อมูลด้วยภาษา VB ได้ (ต้องมีเมธอด VBDatabase Programming())
2. ความสามารถในการเขียนภาษา VC# (ICSProgramable) มีข้อบังคับไว้ว่า
   - ต้องเขียนโปรแกรมเบื้องต้นด้วยภาษา VC# ได้ (ต้องมีเมธอด CSProgramming())
   - ต้องเขียนโปรแกรมแบบ OOP ด้วยภาษา C# ได้ (ต้องมีเมธอด CSOOPProgramming())
   - ต้องเขียนโปรแกรมด้านฐานข้อมูลด้วยภาษา C# ได้ (ต้องมีเมธอด CSDatabaseProgramming())

โค้ด VB 2005 ที่ 12-2 พื้นฐานการทำ Implement Multiple Interfaces (Programmer.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Implements IVBProgramable, ICSProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา VB")
    End Sub

    Public Sub VBDatabaseProgramming() Implements IVBProgramable.VBDatabaseProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา VB ด้านฐานข้อมูล")
    End Sub

    Public Sub CSProgramming() Implements ICSProgramable.CSProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา C#")
    End Sub

    Public Sub CSOOPProgramming() Implements ICSProgramable.CSOOPProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา C# เชิงวัตถุ")
    End Sub

    Public Sub CSDatabaseProgramming() Implements ICSProgramable.CSDatabaseProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา C# ด้านฐานข้อมูล")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-2 พื้นฐานการทำ Implement Multiple Interfaces (Programmer.cs)

```
public class Programmer : IVBProgramable, ICSProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา VB");
    }

    public void VBDatabaseProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา VB ด้านฐานข้อมูล");
    }

    public void CSProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา C#");
    }

    public void CSOOPProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา C# เชิงวัตถุ");
    }

    public void CSDatabaseProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา C# ด้านฐานข้อมูล");
    }
}
```

ต่อมาสร้างคลาส Programmer ขึ้นมา เราต้องการให้ Programmer เขียนโปรแกรมได้ทั้ง VB และ C# จึง Implements อินเตอร์เฟส IVBProgramable และอินเตอร์เฟส ICSProgramable เข้ามา คุณต้องเขียนโค้ดให้ทั้ง 5 เมธอดเองเช่นเดิม ส่วนการใช้งานคลาส Programmer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-2 พื้นฐานการทำ Implement Multiple Interfaces

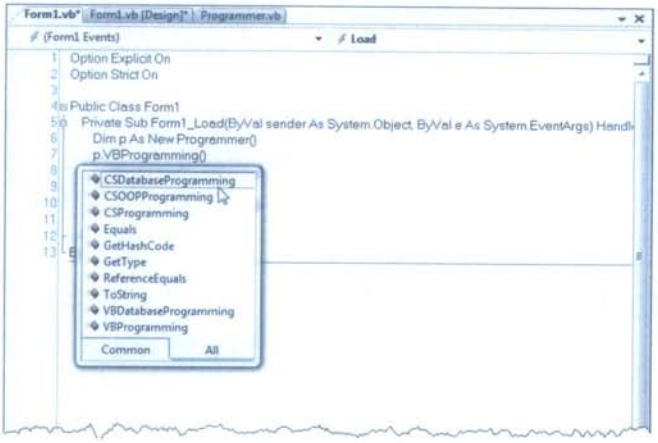
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Programmer()
        p.VBProgramming()
        p.VBDatabaseProgramming()
        p.CSProgramming()
        p.CSDatabaseProgramming()
        p.CSOOPProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer p = new Programmer();
    p.VBProgramming();
    p.VBDatabaseProgramming();
    p.CSProgramming();
    p.CSDatabaseProgramming();
    p.CSOOPProgramming();
}
```

โค้ดข้างต้นสร้างโปรแกรมเมอร์ที่ชื่อว่า p ขึ้นมา ผลที่ได้คือ โปรแกรมเมอร์คนนี้สามารถเขียนโปรแกรมได้ทั้ง VB และ C# ครบทั้ง 5 เมธอด ตามเงื่อนไขที่บังคับไว้ในอินเตอร์เฟสที่ Implements เข้ามานั่นเอง ดังรูปที่ 12-3



รูปที่ 12-3 เมธอดทั้ง 5 ของออบเจ็กต์ p ของคลาส Programmer

เห็นได้ว่าคลาส Programmer ถูกเพิ่มความสามารถใหม่ขึ้นมา โดยการ Implements อินเตอร์เฟสต่างๆ เข้ามานี่คือ ที่มาของคำนิยามที่ผู้เขียนบอกไว้ในขั้นต้นว่า อินเตอร์เฟสคือ ความสามารถหรือคุณสมบัติเพิ่มเติมของคลาส

การ Implement Multiple Interfaces กรณีชื่อเมธอดซ้ำ

ปัญหาหลักอย่างหนึ่งที่คุณต้องพบเจออย่างแน่นอน เมื่อคุณเรียกใช้อินเตอร์เฟสมากกว่า 1 ชุดในเวลาเดียวกันนั่นคือ ข้อบังคับที่อยู่ในอินเตอร์เฟสที่คุณ Implements มามีชื่อซ้ำกัน ทางแก้ปัญหากรณีนี้เท่าที่ผู้เขียนทราบมีอยู่ 3 วิธี จุดตัดสินใจในการเลือกใช้วิธีใดก็ตาม ขึ้นอยู่กับวิธีการออกแบบและเงื่อนไขการใช้งานเป็นหลัก

- วิธี Implements 1 เมธอด
- วิธี Explicit Implements Interface แบบทำงาน 1 เมธอด
- วิธี Explicit Implements Interface แบบทำงานทุกเมธอด

วิธี Implements 1 เมธอด

การเลือกใช้วิธีนี้หมายความว่า คุณมองว่าข้อบังคับที่มีชื่อเหมือนกันคือ สิ่งเดียวกัน ส่งผลให้เป็นการ Implements เงื่อนไขทั้งหมดที่อยู่ในอินเตอร์เฟส แต่กำหนดให้ทำงานเหมือนกันนั่นเอง ให้คุณดูตัวอย่างที่ 12-3 วิธี Implements 1 เมธอด

โค้ด VB 2005 และ VC# 2005 ที่ 12-3 วิธี Implements 1 เมธอด

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IVBProgramable
    Sub BasicProgramming()
End Interface

Public Interface ICSProgramable
    Sub BasicProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IVBProgramable
{
    void BasicProgramming();
}

public interface ICSProgramable
{
    void BasicProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟสที่ชื่อว่า IVBProgramable และอินเตอร์เฟส ICSProgramable ขึ้นมา ผู้เขียนจงใจให้ข้อบังคับทั้ง 2 อินเตอร์เฟสมีชื่อเหมือนกันนั่นคือ มีเมธอด BasicProgramming()

โค้ด VB 2005 ที่ 12-3 วิธี Implements 1 เมธอด (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Implements IVBProgramable, ICSProgramable

    Public Sub BasicProgramming() Implements IVBProgramable.BasicProgramming
        MessageBox.Show("พื้นฐานเขียนโปรแกรม")
    End Sub

    Public Sub BasicProgramming1() Implements ICSProgramable.BasicProgrammning
        Me.BasicProgramming()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-3 วิธี Implements 1 เมธอด (Class1.cs)

```
public class Programmer :IVBProgramable,ICSProgramable
{
    public void BasicProgramming()
    {
        MessageBox.Show("พื้นฐานเขียนโปรแกรม");
    }
}
```

ต่อมาสร้างคลาส Programmer ขึ้นมา กำหนดให้ Implements อินเตอร์เฟส IVBProgramable และอินเตอร์เฟส ICSProgramable เนื่องจากว่าทั้ง 2 อินเตอร์เฟสมีชื่อเมธอดเหมือนกัน ซึ่งแยกออกเป็น 2 กรณี กล่าวคือ

- **กรณี** VB 2005 : ถ้าเรามองว่าชื่อเมธอดที่ซ้ำกันคือ ข้อบังคับเดียวกัน ก็จะกำหนดให้เมธอดใดเมธอดหนึ่งจากอินเตอร์เฟสใดก็ได้เป็นหลัก ส่วนเมธอดที่เหลือให้เรียกเมธอดหลักทำงานซ้ำ ในกรณีนี้กำหนดให้เมธอด BasicProgramming() ของอินเตอร์เฟส IVBProgramable เป็นหลัก

VB 2005

```
Public Sub BasicProgramming() Implements IVBProgramable.BasicProgramming
    MessageBox.Show("พื้นฐานเขียนโปรแกรม")
End Sub

Public Sub BasicProgramming1() Implements ICSProgramable.BasicProgramming
    Me.BasicProgramming()
End Sub
```

VB 2005 บังคับให้คุณต้อง Implements เมธอด BasicProgramming() ครบทั้ง 2 เมธอด เพียงแต่ว่าต้องตั้งชื่อเมธอดไม่เหมือนกันนั่นคือ เมธอด BasicProgramming() กับเมธอด BasicProgramming1()

เมื่อเรามองว่าทั้ง 2 เมธอดทำงานเหมือนกัน ส่งผลให้การทำงานของเมธอด BasicProgramming1() เป็นการสั่งให้เมธอด BasicProgramming() ทำงานทางอ้อมนั่นเอง

- **กรณี** VC# 2005 : ให้ Implements เพียง 1 เมธอดตามปกติ ดังโค้ดต่อไปนี้

VC# 2005

```
public void BasicProgramming()
{
    MessageBox.Show("พื้นฐานเขียนโปรแกรม");
}
```

ส่วนการใช้งานคลาส Programmer อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 12-3 วิธี Implements 1 เมธอด

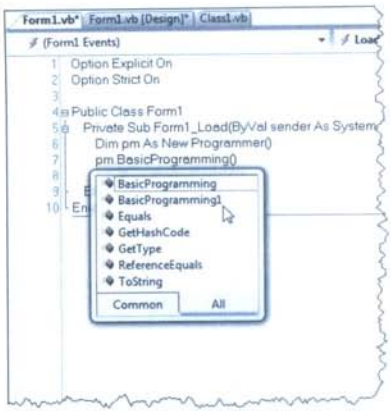
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim prn As New Programmer()
        prn.BasicProgramming()
        'prn.BasicProgramming1()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer prn = new Programmer();
    prn.BasicProgramming();
}
```

รูปที่ 12-4
เมธอด Basic Programming() ของ VB 2005



จากรูปที่ 12-4 ในกรณี VB 2005 แม้ว่าจะมีเมธอด BasicProgramming() และเมธอด Basic Programming1() ปรากฏขึ้นมา 2 เมธอด แต่ก็ทำงานเหมือนกัน ส่วน VC# 2005 ปรากฏขึ้นมา 1 เมธอดตามที่ Implements ในคลาส Programmer

รูปที่ 12-5
เมธอด Basic Programming() ของ VC# 2005

วิธี Explicit Implements Interface แบบทำงาน 1 เมธอด

การ Implements อินเตอร์เฟสที่ผู้เขียนกล่าวมาก่อนหน้านี้ทั้งหมด เรียกโดยรวมว่า Implicit Implements Interface กล่าวคือ เป็นการ Implements ตามกฎเกณฑ์ปกติของการใช้อินเตอร์เฟส

ยังมีการ Implements อินเตอร์เฟสอีกลักษณะหนึ่งที่เรียกว่า Explicit Implements Interface กล่าวคือ เราเป็นผู้บังคับให้เกิดการ Implements อินเตอร์เฟสนั่นเอง หลักการของวิธีนี้อยู่ที่วิธีการมองออบเจ็กต์ที่สนใจอยู่ เช่น

VB 2005	VC# 2005
Dim c As Customer = New Customer()	Customer c = new Customer();

โค้ดข้างต้นผู้เขียนขอกล่าว 2 แบบ ดังนี้

1. สร้างออบเจ็กต์ Customer ที่ชื่อว่า c ขึ้นมา
2. ให้ตัวแปร c (มี Type เป็น Customer) ชี้ไปยัง (=) ออบเจ็กต์ Customer ที่เกิดขึ้นมาด้วยคำสั่ง New (new)

เห็นได้ว่า Type ของตัวแปร c กับออบเจ็กต์ที่เกิดเป็นชนิดเดียวกัน เรามองออบเจ็กต์ Customer ในฐานะ Customer Type แต่คุณต้องไม่ลืมว่าอินเตอร์เฟสก็คือ Type ชนิดหนึ่งใน .NET เช่นกัน ถ้าเราจะมองออบเจ็กต์ในฐานะอินเตอร์เฟสบ้าง ผลที่ได้จะเป็นอย่างไร ให้คุณดูตัวอย่างที่ 12-4 วิธี Explicit Implements Interface แบบทำงาน 1 เมธอดเพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 12-4 วิธี Explicit Implements Interface แบบทำงาน 1 เมธอด

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IVBProgramable
    Sub BasicProgramming()
End Interface

Public Interface ICSProgramable
    Sub BasicProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IVBProgramable
{
    void BasicProgramming();
}

public interface ICSProgramable
{
    void BasicProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟสขึ้นมา 2 ชุด กล่าวคือ

- อินเตอร์เฟสที่ชื่อว่า IVBProgramable หมายถึง ความสามารถในการเขียนโปรแกรมภาษา VB มีข้อบังคับคือ ต้องเขียนโปรแกรมขั้นพื้นฐานได้ (ต้องมีเมธอด BasicProgramming())
- อินเตอร์เฟสที่ชื่อว่า ICSProgramable หมายถึง ความสามารถในการเขียนโปรแกรมภาษา VC# มีข้อบังคับคือ ต้องเขียนโปรแกรมขั้นพื้นฐานได้ (ต้องมีเมธอด BasicProgramming()) เช่นกัน

ผู้เขียนสร้างคลาส Programmer ขึ้นมาเช่นเดิม แต่ใช้วิธีการทำ Explicit Implements Interface กับอินเตอร์เฟสทั้ง 2 โดยที่ใน VB 2005 และ VC# 2005 มีวิธีการแตกต่างกัน กล่าวคือ

- **กรณี VB 2005** : ให้คุณ Implements อินเตอร์เฟสทั้ง 2 ตามปกติ ผู้เขียนจะตั้งชื่อเมธอดใหม่ให้เหมาะสม ดังโค้ดต่อไปนี้

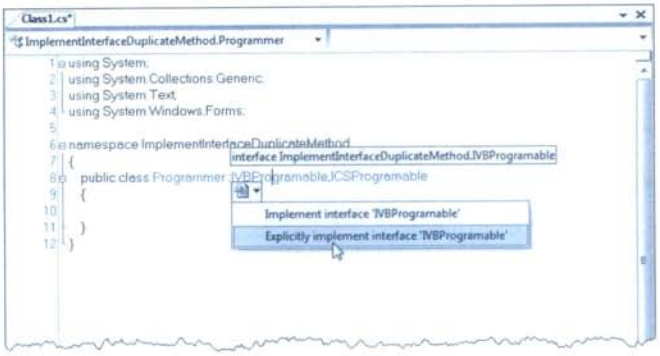
โค้ด VB 2005 ที่ 12-4 วิธี Explicit Implements Interface แบบทำงาน 1 เมธอด (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Implements IVBProgramable, ICSProgramable

    Public Sub VBBasicProgramming() Implements IVBProgramable.BasicProgramming
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VB")
    End Sub

    Public Sub CSBasicProgramming() Implements ICSProgramable.BasicProgramming
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VC#")
    End Sub
End Class
```

เมธอดที่ชื่อว่า VBBasicProgramming() มาจากข้อบังคับ BasicProgramming ที่อยู่ในอินเตอร์เฟส IVBProgramable มองว่าถ้ามีความสามารถในการเขียนโปรแกรมด้วยภาษา VB ก็ต้องเขียนด้วยภาษา VB เท่านั้น

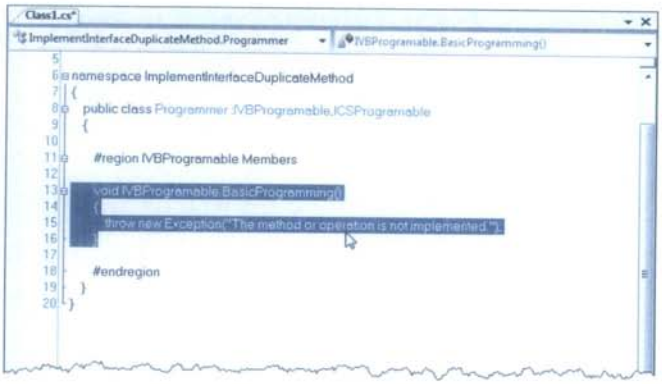
ส่วนเมธอดที่ชื่อว่า CSBasicProgramming() มาจากข้อบังคับ BasicProgramming ที่อยู่ในอินเตอร์เฟส ICSProgramable มองว่าถ้ามีความสามารถในการเขียนโปรแกรมด้วยภาษา VC# ก็ต้องเขียนด้วยภาษา VC# เท่านั้น เห็นได้ว่าทั้ง 2 เมธอดทำงานแตกต่างกันอย่างสิ้นเชิง เพราะว่าข้อความใน MessageBox จะแตกต่างกันนั่นเอง

- **กรณี VC# 2005** : ให้คุณนำเมาส์ไปโฟกัสที่อินเตอร์เฟส IVBProgramable และอินเตอร์เฟส ICSProgramable เลือกคำสั่ง Explicitly Implement Interface ดังรูปที่ 12-6 และ 12-7

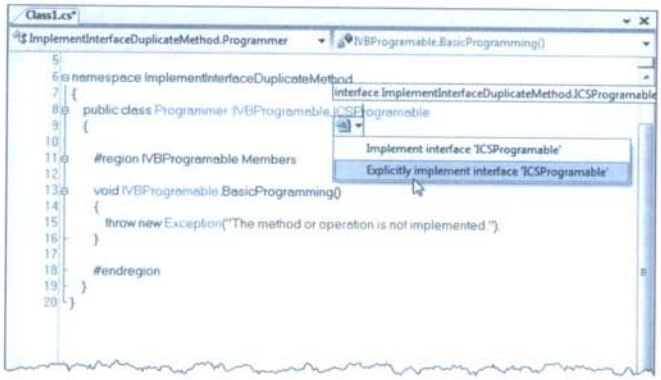


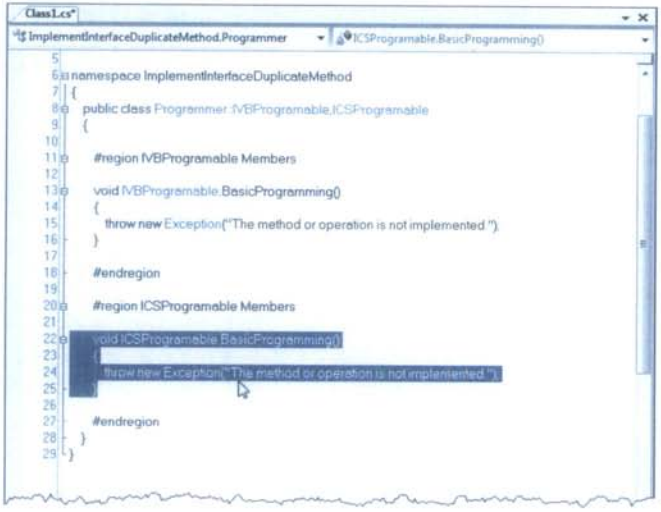
รูปที่ 12-6 การทำ Explicit Implements Interface ที่ชื่อว่า IVBProgramable

รูปที่ 12-6 (ต่อ)
การทำ Explicit Implements Interface ที่ชื่อว่า IVBProgramable



รูปที่ 12-7
การทำ Explicit Implements Interface ที่ชื่อว่า ICSProgramable





จากรูปที่ 12-6 และ 12-7 เห็นได้ว่า VS 2005 จะสร้างเมธอดว่างให้คุณ 2 เมธอด ตามข้อบังคับที่อยู่ในอินเตอร์เฟส ให้คุณแก้ไขโค้ดของเมธอดทั้ง 2 ใหม่ ดังนี้

โค้ด VC# 2005 ที่ 12-4 วิธี Explicit Implements Interface แบบทำงาน 1 เมธอด (Class1.cs)

```
public class Programmer :IVBProgramable,ICSProgramable
{
    void IVBProgramable.BasicProgramming()
    {
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VB");
    }

    void ICSProgramable.BasicProgramming()
    {
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VC#");
    }
}
```

ถ้าเรียกใช้เมธอด BasicProgramming() ของอินเตอร์เฟส IVBProgramable หมายถึง ถ้ามีความสามารถในการเขียนโปรแกรมเบื้องต้นด้วยภาษา VB ก็ต้องใช้ภาษา VB เท่านั้น แต่ถ้าเป็นการเรียกใช้เมธอด BasicProgramming() ของอินเตอร์เฟส ICSProgramable ก็ต้องใช้ภาษา VC# เท่านั้นเช่นกัน ส่วนการใช้งานคลาส Programmer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-4 วิธี Explicit Implements Interface แบบทำงาน 1 เมธอด

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim vb As IVBProgramable
        vb = New Programmer()
        vb.BasicProgramming()

        'Dim cs As ICSProgramable
        'cs = New Programmer()
        'cs.BasicProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    IVBProgramable vb;
    vb = new Programmer();
    vb.BasicProgramming();

    //ICSProgramable cs;
    //cs = new Programmer();
    //cs.BasicProgramming();
}
```

จากที่ผู้เขียนกล่าวไว้ในตอนต้นว่า หลักการของวิธีนี้อยู่ที่การมองออบเจ็กต์ในฐานะของอินเตอร์เฟส กล่าวคือ ให้คุณมองออบเจ็กต์ Programmer ที่ชื่อว่า p ในฐานะโปรแกรมเมอร์ภาษา VB ผลที่ได้แสดงดังรูปที่ 12-8

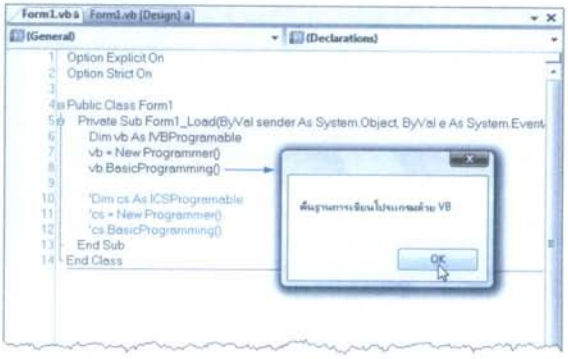
VB 2005

```
Dim vb As IVBProgramable
vb = New Programmer()
vb.BasicProgramming()
```

VC# 2005

```
IVBProgramable vb;
vb = new Programmer();
vb.BasicProgramming();
```

รูปที่ 12-8 เมื่อมองออบเจ็กต์ Programmer ในฐานะอินเตอร์เฟส IVBProgramable



จากรูปที่ 12-8 ข้อความที่ปรากฏขึ้นมาแสดงให้เห็นว่ามาจากข้อบังคับ BasicProgramming() ของอินเตอร์เฟส IVBProgramable นั่นเอง ในทางกลับกันถ้ามองออบเจ็กต์ Programmer ในฐานะโปรแกรมเมอร์ภาษา VC# บ้าง ผลที่ได้แสดงดังรูปที่ 12-9

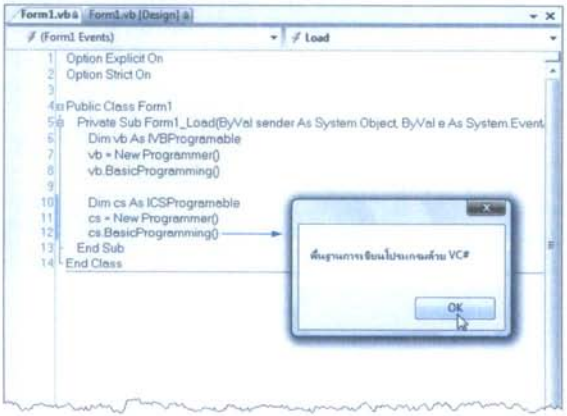
VB 2005

```
Dim cs As ICSProgramable
cs = New Programmer()
cs.BasicProgramming()
```

VC# 2005

```
ICSProgramable cs;
cs = new Programmer();
cs.BasicProgramming();
```

รูปที่ 12-9 เมื่อมองออบเจ็กต์ Programmer ในฐานะอินเตอร์เฟส ICSProgramable



จากรูปที่ 12-9 ข้อความที่ได้เกิดจากการทำงานของเมธอด BasicProgramming() ของอินเตอร์เฟส ICSProgramable นั่นเอง

วิธี Explicit Implements Interface แบบทำงานทุกเมธอด

วิธีนี้ต่อยอดมาจากวิธีที่แล้วกล่าวคือ เมธอด BasicProgramming() ที่ถูกสั่งให้ทำงานขึ้นอยู่กับว่าเราจะมองออบเจ็กต์ Programmer ในฐานะอินเตอร์เฟส IVBProgramable หรืออินเตอร์เฟส ICSProgramable

คำถามที่ตามมาก็คือ ถ้าเรามีความจำเป็นหลีกเลี่ยงไม่ได้ที่ต้องสั่งให้เมธอด BasicProgramming() ที่มาจากทั้ง 2 อินเตอร์เฟสทำงานจะทำอย่างไร ให้คุณดูตัวอย่างที่ 12-5 วิธี Explicit Implements Interface แบบทำงานทุกเมธอด

โค้ด VB 2005 และ VC# 2005 ที่ 12-5 วิธี Explicit Implements Interface แบบทำงานทุกเมธอด

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IVBProgramable
    Sub BasicProgramming()
End Interface

Public Interface ICSProgramable
    Sub BasicProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IVBProgramable
{
    void BasicProgramming();
}

public interface ICSProgramable
{
    void BasicProgramming();
}
```

โค้ด VB 2005 ที่ 12-5 วิธี Explicit Implements Interface แบบทำงานทุกเมธอด (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Implements IVBProgramable, ICSProgramable

    Public Sub VBBasicProgramming() Implements IVBProgramable.BasicProgramming
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VB")
    End Sub

    Public Sub CSBasicProgramming() Implements ICSProgramable.BasicProgramming
        Dim _vb As IVBProgramable = Me
        _vb.BasicProgramming()
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VC#")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-5 วิธี Explicit Implements Interface แบบทำงานทุกเมธอด (Class1.cs)

```
public class Programmer :IVBProgramable,ICSProgramable
{
    void IVBProgramable.BasicProgramming()
    {
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VB");
    }

    void ICSProgramable.BasicProgramming()
    {
        IVBProgramable _vb = this;
        _vb.BasicProgramming();
        MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VC#");
    }
}
```

ข้อแตกต่างอยู่ในคลาส Programmer กล่าวคือ ผู้เขียนมองว่าโปรแกรมเมอร์ที่เขียนภาษา VC# ได้ ที่น่าจะเขียน VB ได้ด้วยเช่นกัน จึงกำหนดให้เมธอด BasicProgramming() ของอินเตอร์เฟส ICSProgramable สั่งให้เมธอด BasicProgramming() ทำงานด้วยเช่นกัน วิธีคิดคือ

VB 2005

```
Public Sub CSBasicProgramming() Implements ICSProgramable.BasicProgramming
    Dim _vb As IVBProgramable = Me
    _vb.BasicProgramming()
    MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VC#")
End Sub
```

VC# 2005

```
void ICSProgramable.BasicProgramming()
{
    IVBProgramable _vb = this;
    _vb.BasicProgramming();
    MessageBox.Show("พื้นฐานการเขียนโปรแกรมด้วย VC#");
}
```

เราจะสร้างตัวแปรขึ้นมา 1 ตัวชื่อว่า _vb มี Type เป็น IVBProgramable ทำหน้าที่สั่งให้เมธอด BasicProgramming() ของอินเตอร์เฟส IVBProgramable ทำงาน ที่น่าสนใจคือ วิธีการระบุออบเจ็กต์ Programmer ให้กับ _vb ผู้เขียนใช้คำสั่ง Me (this) กล่าวคือ

คำสั่ง Me (this) คุณใช้ในที่ใด หมายถึง คลาสนั้นๆ เช่น

- คุณใช้ในคลาส Form1 คำสั่ง Me (this) จึงหมายถึง คลาส Form1 ปัจจุบัน
- คุณใช้ในคลาส Customer คำสั่ง Me (this) จึงหมายถึง คลาส Customer ปัจจุบัน

ในกรณีนี้ใช้ในคลาส Programmer ส่งผลให้คำสั่ง Me (this) จึงหมายถึง คลาส Programmer นั่นเอง ส่วนวิธีการใช้งานคลาส Programmer อยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-5 วิธี Explicit Implements Interface แบบทำงานทุกเมธอด

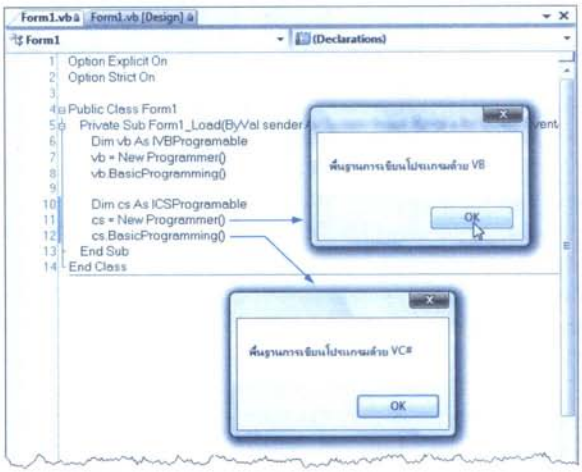
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim vb As IVBProgramable
        vb = New Programmer()
        vb.BasicProgramming()

        Dim cs As ICSProgramable
        cs = New Programmer()
        cs.BasicProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    IVBProgramable vb;
    vb = new Programmer();
    vb.BasicProgramming();

    ICSProgramable cs;
    cs = new Programmer();
    cs.BasicProgramming();
}
```

รูปที่ 12-10 ผลการรันตัวอย่างที่ 12-5



จากรูปที่ 12-10 โปรแกรมเมอร์คนนี้เก่งกว่าคนที่แล้ว เพราะว่าถ้าคุณมองโปรแกรมเมอร์คนนี้ในฐานะ ICSProgramable เขาสามารถเขียนได้ทั้ง 2 ภาษาคือ VB และ VC#

VB 2005

```
Dim cs As ICSProgramable
cs = New Programmer()
cs.BasicProgramming()
```

VC# 2005

```
ICSProgramable cs;
cs = new Programmer();
cs.BasicProgramming();
```

การใช้งาน Is-a Inheritance ในฐานะพารามิเตอร์

คำถามหนึ่งที่เกิดขึ้นมาเมื่อมีการใช้งานอินเตอร์เฟสนั่นคือ อินเตอร์เฟสช่วยเหลือเราอย่างไร, ให้ประโยชน์อะไรบ้าง และอินเตอร์เฟสใช้ตอนไหน ผู้เขียนค้างคำตอบคุณผู้อ่านไว้ว่า ทำไมเราต้องใช้อินเตอร์เฟส หัวข้อนี้คุณจะได้ศึกษาประโยชน์ของการอินเตอร์เฟสชัดเจนยิ่งขึ้น

โดยปกติแล้วการสร้างเมธอดขึ้นมาใช้งานในคลาสของคุณ ก็จะประกอบไปด้วยรายการพารามิเตอร์ต่างๆ ที่ผู้ใช้งานต้องส่งค่าเข้ามา เพื่อนำไปใช้ในการทำงานของเมธอดนั้นๆ โดยที่พารามิเตอร์แต่ละตัวก็จะมี Type ระบุไว้อย่างชัดเจน ส่งผลให้ผู้ใช้ต้องส่งค่าให้ตรงกับ Type ของพารามิเตอร์แต่ละตัวอย่างเคร่งครัด

คลาสต้นแบบที่คุณสร้างขึ้นมาก็สามารถนำมาใช้เป็น Type ให้กับพารามิเตอร์ได้อีกด้วย ประเด็นก็คือถ้าคุณนำคลาสที่มีการสืบทอดคลาสมาใช้เป็น Type ให้กับพารามิเตอร์ในเมธอดของคุณ จะมีผลอย่างไรบ้าง โดยที่การสืบทอดคลาสใน .NET มีอยู่ 2 แบบคือ

1. การสืบทอดคลาสแบบปกติ ความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูก เรียกว่า Is-a Relationship
2. การ Implements Interface ความสัมพันธ์ระหว่างคลาสแม่กับคลาสลูก เรียกว่า Has-a Relationship

คำถามที่ตามมาก็คือ ถ้าเรานำคลาสที่เกิดจาก Is-a Relationship หรือ Has-a Relationship ไปทำหน้าที่เป็นพารามิเตอร์จะมีผลอย่างไรบ้าง ตัวอย่างที่ 12-6 การใช้งาน Is-a Inheritance ในฐานะพารามิเตอร์

โค้ด VB 2005 และ VC# 2005 ที่ 12-6 การใช้งาน Is-a Inheritance ในฐานะพารามิเตอร์

VB 2005 (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Private _FullName As String = ""

    Public Property FullName() As String
        Get
            Return _FullName
        End Get
        Set(ByVal value As String)
            _FullName = value
        End Set
    End Property

    Public Sub BasicProgramming()
        MessageBox.Show("พื้นฐานการเขียนโปรแกรม", "Person")
    End Sub
End Class

Public Class Programmer
    Inherits Person
```

VC# 2005 (Class1.cs)

```
public class Person
{
    private string _FullName = "";
    public string FullName
    {
        get{return _FullName;}
        set{_FullName = value;}
    }

    public void BasicProgramming()
    {
        MessageBox.Show("พื้นฐานการเขียนโปรแกรม",
"Person");
    }
}

public class Programmer : Person
{
    public void VBProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา VB",
"Programmer");
```

```
    Public Sub VBProgramming()
        MessageBox.Show("เขียนโปรแกรมภาษา VB",
"Programmer")
    End Sub

    Public Sub CSProgramming()
        MessageBox.Show("เขียนโปรแกรมภาษา C#",
"Programmer")
    End Sub
End Class
```

```
    }

    public void CSProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา C#",
"Programmer");
    }
}
```

ผู้เขียนสร้างคลาสคนธรรมดา Person ขึ้นมา ก็ต้องมีชื่อ-สกุล (คุณสมบัติ FullName) คนธรรมดาคนนี้สามารถเขียนโปรแกรมขั้นพื้นฐานได้เพียงอย่างเดียวเท่านั้น (มีเมธอด BasicProgrmming())

ส่วนอีกคลาสคือ โปรแกรมเมอร์ Programmer กำหนดให้สืบทอดคลาสมาจากคลาส Person ส่งผลให้คลาส Person ทำหน้าที่เป็นคลาสแม่ ส่วนคลาส Programmer ทำหน้าที่เป็นคลาสลูก

เพราะความที่ Programmer ควรจะเก่งกว่าคนธรรมดา Person จึงกำหนดให้ Programmer สามารถเขียนโปรแกรมด้วยภาษา VB (มีเมธอด VBProgramming()) และภาษา VC# (มีเมธอด CSProgramming()) ได้อีกด้วย

โค้ด VB 2005 และ VC# 2005 ที่ 12-6 การใช้งาน is-a Inheritance ในฐานะพารามิเตอร์

VB 2005 (Company.vb)

```
Option Explicit On
Option Strict On

Public Class Company
    Public Sub AdmitPerson(ByVal p As Person)
        p.BasicProgramming()
    End Sub

    Public Sub AdmitProgrammer(ByVal p As Programmer)
        p.BasicProgramming()
        p.VBProgramming()
        p.CSProgramming()
    End Sub
End Class
```

VC# 2005 (Company.cs)

```
public class Company
{
    public void AdmitPerson(Person p)
    {
        p.BasicProgramming();
    }

    public void AdmitProgrammer(Programmer p)
    {
        p.BasicProgramming();
        p.VBProgramming();
        p.CSProgramming();
    }
}
```

ต่อมาผู้เขียนตั้งบริษัทขึ้นมาชื่อว่า Company เวลาที่รับสมัครคนเข้าทำงานสามารถประกาศรับสมัครได้ 3 แบบคือ

1. ประกาศรับสมัครคนธรรมดาทั่วไป อยู่ในความรับผิดชอบของเมธอด AdmitPerson() ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น Person เพราะเราต้องการคนธรรมดาเข้ามาทำงาน

VB 2005

```
Public Sub AdmitPerson(ByVal p As Person)
    p.BasicProgramming()
End Sub
```

VC# 2005

```
public void AdmitPerson(Person p)
{
    p.BasicProgramming();
}
```

เห็นได้ว่าเมื่อรับสมัครคนธรรมดาเข้ามา ก็จะทำได้แต่เพียงเขียนโปรแกรมพื้นฐานเท่านั้น กล่าวได้อีกนัยหนึ่งคือ เรียกใช้เมธอด BasicProgramming() ได้เพียง 1 เมธอด

2. ประกาศรับสมัครโปรแกรมเมอร์ อยู่ในความรับผิดชอบของเมธอด AdmitProgrammer() ต้องการพารามิเตอร์ 1 ตัวเช่นกัน แต่มี Type เป็น Programmer

VB 2005

```
Public Sub AdmitProgrammer(ByVal p As Programmer)
    p.BasicProgramming()
    p.VBProgramming()
    p.CSProgramming()
End Sub
```

VC# 2005

```
public void AdmitProgrammer(Programmer p)
{
    p.BasicProgramming();
    p.VBProgramming();
    p.CSProgramming();
}
```

เห็นได้ว่าโปรแกรมเมอร์ทำได้หลายอย่างไม่ว่าจะเป็น

- เขียนโปรแกรมพื้นฐานหรือสามารถใช้เมธอด BasicProgramming() ได้
- เขียนภาษา VB หรือสามารถใช้เมธอด VBProgramming() ได้
- และเขียนภาษา VC# ได้อีกด้วย หรือใช้เมธอด CSProgramming() ได้

โค้ด VB 2005 และ VC# 2005 ที่ 12-6 การใช้งาน Is-a Inheritance ในฐานะพารามิเตอร์

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        p.FullName = "ธิติมา ยุราวรรณ"

        Dim pm As New Programmer()
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Person p = new Person();
    p.FullName = "ธิติมา ยุราวรรณ";

    Programmer pm = new Programmer();
    pm.FullName = "ศุภชัย สมพานิช";

    Company myCompany = new Company();
    myCompany.AdmitPerson(p);
```

```
        pm.FullName = "ศุภชัย สมพานิช"

        Dim myCompany As New Company()
        myCompany.AdmitPerson(p)
        myCompany.AdmitPerson(pm)

        'myCompany.AdmitProgrammer(p)
        myCompany.AdmitProgrammer(pm)
    End Sub
End Class
```

```
        myCompany.AdmitPerson(pm);

        //myCompany.AdmitProgrammer(p);
        myCompany.AdmitProgrammer(pm);
    }
```

NOTE

ส่วนการรับสมัครแบบที่ 3 ผู้เขียนจะกล่าวในหัวข้อถัดไป

วิธีการดูโค้ดชุดนี้ให้คุณดูทีละส่วนกล่าวคือ สมมติว่ามีคนเข้ามาสมัครงานอยู่ 2 คนคือ

- คนธรรมดาที่ชื่อว่า ธิติมา ยุราวรรณ (ออบเจ็กต์ Person ที่ชื่อว่า p)
- โปรแกรมเมอร์ที่ชื่อว่า ศุภชัย สมพานิช (ออบเจ็กต์ Programmer ที่ชื่อว่า pm)

VB 2005

```
Dim p As New Person()
p.FullName = "ธิติมา ยุราวรรณ"

Dim pm As New Programmer()
pm.FullName = "ศุภชัย สมพานิช"
```

VC# 2005

```
Person p = new Person();
p.FullName = "ธิติมา ยุราวรรณ";

Programmer pm = new Programmer();
pm.FullName = "ศุภชัย สมพานิช";
```

ต่อมามีบริษัทที่ชื่อว่า myCompany ประกาศรับสมัครคนธรรมดา เห็นได้ว่าทั้งคนธรรมดา p และโปรแกรมเมอร์ pm สามารถสมัครเข้าทำงานได้ ทั้งๆ ที่พารามิเตอร์ของเมธอด AdmitPerson() มี Type เป็น Person

VB 2005

```
Dim myCompany As New Company()
myCompany.AdmitPerson(p)
myCompany.AdmitPerson(pm)
```

VC# 2005

```
Company myCompany = new Company();
myCompany.AdmitPerson(p);
myCompany.AdmitPerson(pm);
```

เหตุผลก็คือ คลาส Person ทำหน้าที่เป็นคลาสแม่ ส่วนคลาส Programmer ทำหน้าที่เป็นคลาสลูก การนำคลาสแม่มาเป็น Type ของพารามิเตอร์ผลที่ตามมาคือ คุณสามารถส่งคลาสลูกเข้ามาได้

แม้ว่าโปรแกรมเมอร์ pm (อยู่ในฐานะเป็นคลาสลูก) จะเก่งกว่าคนธรรมดา p แต่เมื่อถูกส่งเข้ามาในฐานะเป็น Person แล้ว ก็จะทำให้ความเป็นโปรแกรมเมอร์ต้องหมดไป ทำให้โปรแกรมเมอร์ pm คนนี้สามารถเขียนโปรแกรมเบื้องต้นได้เพียงอย่างเดียวเท่านั้น กล่าวได้อีกนัยหนึ่งคือ เรียกใช้เมธอด BasicProgramming() ได้เพียงเมธอดเดียวนั่นเอง

แต่ถ้าบริษัท myCompany ประกาศรับสมัครโปรแกรมเมอร์ ผลที่ได้คือ โปรแกรมเมอร์ pm คนเดียวเท่านั้นที่สามารถสมัครเข้ามาได้ เพราะว่าพารามิเตอร์ของเมธอด AdmitProgrammer() มี Type เป็น Programmer กล่าวได้อีกนัยหนึ่งคือ เมื่อคุณนำคลาสลูกมาเป็น Type ของพารามิเตอร์ คุณไม่สามารถส่งคลาสแม่ Person เข้ามาได้นั่นเอง

VB 2005	VC# 2005
myCompany.AdmitProgrammer(pm)	myCompany.AdmitProgrammer(pm);

ถ้าคุณส่งคนธรรมดา p (อยู่ในฐานะเป็นคลาสแม่) ให้กับเมธอด AdmitProgrammer() ซึ่งมีพารามิเตอร์เป็น Programmer Type (คลาสลูก) ก็จะเกิดข้อผิดพลาดขึ้น

VB 2005	VC# 2005
myCompany.AdmitProgrammer(p) ✗	myCompany.AdmitProgrammer(p); ✗

ผู้เขียนขอย้อนกลับไปกรณีรับสมัครคนธรรมดาอีกครั้ง (เมธอด AdmitPerson() ของคลาส Company) แม้ว่าโปรแกรมเมอร์สามารถสมัครเข้ามาในบริษัทได้ก็จริง แต่ก็ถูกจำกัดให้มองโปรแกรมเมอร์ในฐานะคนธรรมดาเท่านั้น

คำถามที่ตามมาก็คือ ในเมื่อเรามีโปรแกรมเมอร์อยู่ในบริษัทก็ควรที่จะให้โปรแกรมเมอร์ทำงานอย่างเต็มความสามารถ (ใช้ได้ทุกเมธอดของคลาส Programmer) ไม่ควรถูกจำกัดความสามารถ จะทำอย่างไร

วิธีคิดก็คือ ทั้งคนธรรมดา Person และโปรแกรมเมอร์ Programmer มีความสัมพันธ์ระหว่างกันในฐานะ Is-a Relationship คุณสามารถแปลงข้อมูลด้วยวิธี Cast ได้ดังโค้ดต่อไปนี้

VB 2005	VC# 2005

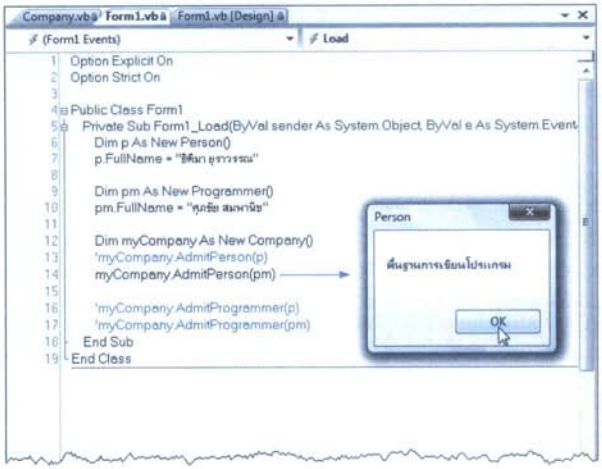
VB 2005 VC# 2005
Option Explicit On
Option Strict On
 	
Public Class Company
Public Sub AdmitPerson(ByVal p As Person)
If TypeOf p Is Programmer Then
Dim _p As Programmer = DirectCast(p, Programmer)
_p.BasicProgramming()
_p.VBProgramming()
_p.CSProgramming()
	public class Company
	{
	public void AdmitPerson(Person p)
	{
	if (p is Programmer)
	{
	Programmer _p = (Programmer)p;
	_p.BasicProgramming();
	_p.VBProgramming();
	_p.CSProgramming();
	}

```
        Else
            p.BasicProgramming()
        End If
    End Sub

    Public Sub AdmitProgrammer(ByVal p As Program-
mer)
        p.BasicProgramming()
        p.VBProgramming()
        p.CSProgramming()
    End Sub
End Class
```

```
        else
        {
            p.BasicProgramming();
        }
    }

    public void AdmitProgrammer(Programmer p)
    {
        p.BasicProgramming();
        p.VBProgramming();
        p.CSProgramming();
    }
}
```

เห็นได้ว่าที่เมธอด AdmitPerson() มีการตรวจสอบพารามิเตอร์ p ก่อนเลยว่า ถ้าเป็น Programmer ให้ Cast พารามิเตอร์ p (มี Type เป็น Person) ไปเป็น Programmer Type โดยเก็บไว้ที่ตัวแปร _p ส่งผลให้ถ้ามีการส่ง Programmer Type เข้ามาที่เมธอด AdmitPerson() โปรแกรมเมอร์ก็จะแสดงความสามารถได้ครบทุกเมธอดตามที่ควรจะเป็น

ที่น่าสนใจอีกประเด็นก็คือ แม้ว่าคุณจะ Cast พารามิเตอร์ p จาก Person Type มาเป็น Programmer Type ได้ก็จริง แต่เมธอด BasicProgramming() ของออบเจ็กต์ _p ไม่ใช่เมธอดของคลาส Programmer แต่เป็นเมธอดของคลาส Person ดังรูปที่ 12-11



รูปที่ 12-11 ผลการทำงานเมธอด Basic Programming() ของออบเจ็กต์ _p

การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์

ผู้เขียนขอกล่าวถึงการรับสมัครงานของตัวอย่างที่ผ่านมาว่ามีข้อเสีย และข้อจำกัดอย่างไร เช่น ถ้าคนธรรมดากับโปรแกรมเมอร์ไม่ได้เกี่ยวข้องกัน (ไม่มีการสืบทอดคลาสระหว่างกัน) จะทำอย่างไร ถ้าใช้วิธีของตัวอย่างที่ผ่านมา บริษัทต้องประกาศรับสมัครโดยการสร้างเมธอดที่มีพารามิเตอร์เป็นทั้งคนธรรมดา และโปรแกรมเมอร์ และถ้ามี SA, Admin, Manager ฯลฯ เพิ่มเข้ามาอีก เห็นได้ว่าบริษัทต้องมีเมธอดที่คอยรับอาชีพต่างๆ เหล่านี้มากมาย ระบบของคุณก็จะบวมจนเกินไป เพราะว่าคลาสเป็น Reference Type ก็จะเต็มไปด้วยอินสแตนซ์ต่างๆ เหล่านี้มากเกินความจำเป็น

ปัญหาที่ยกมาข้างต้นตรงกับโลกของความเป็นจริงที่ว่า คนที่มาสมัครงานมีมากมายหลายอาชีพ จึงมีโอกาสสูงที่แต่ละคน จะไม่มีความเกี่ยวข้องกันแต่อย่างใด ดังโค้ดต่อไปนี้

VB 2005

```
Public Class Company
    Public Sub AdmitPerson(ByVal p As Person)

    End Sub

    Public Sub AdmitProgrammer(ByVal p As Programmer)

    End Sub

    Public Sub AdmitSA(ByVal p As SA)

    End Sub

    Public Sub AdmitManager(ByVal p As Manager)

    End Sub
End Class
```

VC# 2005

```
public class Company
{
    public void AdmitPerson(Person p)
    {
    }

    public void AdmitProgrammer(Programmer p)
    {
    }

    public void AdmitSA(SA p)
    {
    }

    public void AdmitManager(Manager p)
    {
    }
}
```

เห็นได้ว่าคลาส Company ของคุณต้องมีเมธอดรับสมัครงานตามจำนวน Type ของพารามิเตอร์ที่รับเข้ามา ถ้ามี 100 อาชีพ ก็ต้องประกาศรับสมัครกัน 100 เมธอด ถ้าเป็นคนธรรมดามาสมัครงาน ก็เรียกเมธอด AdmitPerson() ทำงาน แต่ถ้าเป็น SA เข้ามาสมัครงานก็เรียกเมธอด AdmitSA() ทำงาน

ผู้เขียนจะแก้ปัญหาดังกล่าว โดยการใช้อินเตอร์เฟสเข้ามาช่วยเป็นการแก้ปัญหาวิธีหนึ่งเท่านั้น กล่าวคือ ถ้าเราประกาศรับสมัครว่าต้องการคนที่เขียนโปรแกรมได้ เห็นได้ว่าเราไม่สนใจอาชีพ และอาชีพใดก็ตามที่เขียนโปรแกรมได้เรารับได้หมด เราสนใจแต่เพียงว่าถ้ามีความสามารถในการเขียนโปรแกรมได้ก็เพียงพอแล้ว ให้ดูตัวอย่างที่ 12-7 การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์ เป็นการประกาศรับสมัครงานแบบที่ 3

โค้ด VB 2005 และ VC# 2005 ที่ 12-7 การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IProgramable
    Sub VBProgramming()
    Sub CSProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IProgramable
{
    void VBProgramming();
    void CSProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟส IProgramable ขึ้นมา กำหนดข้อบังคับไว้ว่าถ้ามีความสามารถในการเขียนโปรแกรม ต้องเขียนภาษา VB (มีเมธอด VBProgramming()) และภาษา VC# (มีเมธอด CSProgramming()) ได้

โค้ด VB 2005 ที่ 12-7 การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์ (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Implements IProgramable

    Public Sub VBProgramming() Implements IProgramable.VBProgramming
        MessageBox.Show("Person เขียน VB ได้")
    End Sub

    Public Sub CSProgramming() Implements IProgramable.CSProgramming
        MessageBox.Show("Person เขียน VC# ได้")
    End Sub
End Class

Public Class Programmer
    Implements IProgramable

    Public Sub VBProgramming() Implements IProgramable.VBProgramming
        MessageBox.Show("Progammer เขียน VB ได้")
    End Sub

    Public Sub CSProgramming() Implements IProgramable.CSProgramming
        MessageBox.Show("Programmer เขียน VC# ได้")
    End Sub

    Public Sub OOPProgramming()
        MessageBox.Show("Programmer เขียน OOP ได้")
    End Sub
End Class

Public Class SA

End Class
```

โค้ด VC# 2005 ที่ 12-7 การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์ (Class1.cs)

```
public class Person : IProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Person เขียน VB ได้");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Person เขียน VC# ได้");
    }
}

public class Programmer : IProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Progammer เขียน VB ได้");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Programmer เขียน VC# ได้");
    }

    public void OOPProgramming()
    {
        MessageBox.Show("Programmer เขียน OOP ได้");
    }
}

public class SA
{

}
```

ผู้เขียนยกตัวอย่างขึ้นมา 3 อาชีพ กำหนดให้ทั้ง 3 อาชีพไม่มีความเกี่ยวข้องกันแต่อย่างใด (ไม่มีการสืบทอดคลาสระหว่างกัน) โดยที่

- คนธรรมดา (คลาส Person) เขียนโปรแกรมได้ เพราะว่า Implements อินเตอร์เฟส IProgramable
- โปรแกรมเมอร์ (คลาส Programmer) เขียนโปรแกรมได้เช่นกัน เพราะว่า Implements อินเตอร์เฟส IProgramable นั่นเอง ผู้เขียนคิดว่าโปรแกรมเมอร์ควรจะเก่งกว่าคนธรรมดา จึงกำหนดให้โปรแกรมเมอร์เขียนโปรแกรมแบบ OOP ได้อีกด้วย (มีเมธอด OOPProgramming() เพิ่มขึ้นมา)
- ผู้ดูแลระบบ (คลาส SA) เขียนโปรแกรมไม่ได้

โค้ด VB 2005 และ VC# 2005 ที่ 12-7 การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์

VB 2005 (Company.vb)

```
Option Strict On

Public Class Company
    Public Sub AdmitProgramable(ByVal p As IProgramable)
        If TypeOf p Is Programmer Then
            Dim _p As Programmer = DirectCast(p,
Programmer)
            _p.VBProgramming()
            _p.CSProgramming()
            _p.OOPProgramming()
        Else
            p.VBProgramming()
            p.CSProgramming()
        End If
    End Sub
End Class
```

VC# 2005 (Company.cs)

```
public class Company
{
    public void AdmitProgramable(IProgramable p)
    {
        if (p is Programmer)
        {
            Programmer _p = (Programmer)p;
            _p.VBProgramming();
            _p.CSProgramming();
            _p.OOPProgramming();
        }
        else
        {
            p.VBProgramming();
            p.CSProgramming();
        }
    }
}
```

ต่อมามีบริษัท Company ประกาศรับสมัครคนที่เขียนโปรแกรมได้ โดยการสร้างเมธอดที่ชื่อว่า AdmitProgramable() ต้องการพารามิเตอร์ 1 ตัว ชื่อว่า p มี Type เป็นอินเตอร์เฟส IProgramable

เพราะความที่คลาส Programmer เก่งกว่า จึงต้องมีการตรวจสอบก่อนว่าถ้าพารามิเตอร์ p มี Type เป็น Programmer แล้ว ให้ Cast พารามิเตอร์ p มี Type เป็น Programmer ก่อน เก็บไว้ที่ตัวแปร _p ก็จะทำให้ Programmer สามารถแสดงฝีมือได้ครบทุกเมธอดที่กำหนดไว้ในคลาส Programmer นั่นเอง ส่วนการใช้งานอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-7 การใช้งาน Has-a Inheritance ในฐานะพารามิเตอร์

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        Dim pm As New Programmer()
        Dim s As New SA()

        Dim myCompany As New Company()
        myCompany.AdmitProgramable(p)
        myCompany.AdmitProgramable(pm)
        'myCompany.AdmitProgramable(s)
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Person p = new Person();
    Programmer pm = new Programmer();
    SA s = new SA();

    Company myCompany = new Company();
    myCompany.AdmitProgramable(p);
    myCompany.AdmitProgramable(pm);
    //myCompany.AdmitProgramable(s);
}
```

มีคนเข้ามาสมัครงานอยู่ 3 คนคือ คนธรรมดาที่ชื่อว่า p, โปรแกรมเมอร์ที่ชื่อว่า pm และผู้ดูแลระบบที่ชื่อว่า s

VB 2005	VC# 2005
Dim p As New Person() Dim pm As New Programmer() Dim s As New SA()	Person p = new Person(); Programmer pm = new Programmer(); SA s = new SA();

ต่อมามีบริษัทที่ชื่อว่า myCompany ประกาศรับสมัครคนที่เขียนโปรแกรมได้ (สั่งให้เมธอด AdmitProgramable()) ทำงาน พบว่าคนธรรมดา p และโปรแกรมเมอร์ pm สามารถสมัครงานได้ เพราะว่าทั้ง 2 อาชีพเขียนโปรแกรมได้นั่นเอง (Implements อินเตอร์เฟส IProgramable)

VB 2005	VC# 2005

Dim myCompany As New Company()
myCompany.AdmitProgramable(p)
myCompany.AdmitProgramable(pm)
	Company myCompany = new Company();
	myCompany.AdmitProgramable(p);
	myCompany.AdmitProgramable(pm);

ส่วนผู้ดูแลระบบ s สมัครไม่ได้ เพราะว่าเขียนโปรแกรมไม่ได้นั่นเอง ดังโค้ดต่อไปนี้

VB 2005	VC# 2005
myCompany.AdmitProgramable(s) ⊗	myCompany.AdmitProgramable(s); ⊗

เห็นได้ว่าการแก้ปัญหาโดยการนำอินเตอร์เฟสเข้ามาทำหน้าที่เป็น Type ให้กับพารามิเตอร์ค่อนข้างยืดหยุ่นพอสมควร โดยที่คุณไม่ต้องสนใจว่าคนธรรมดากับโปรแกรมเมอร์จะเกี่ยวข้องกันหรือไม่ ให้สนใจแต่เพียงว่า Type ที่รับเข้ามามีการ Implements อินเตอร์เฟสที่สนใจอยู่หรือไม่เท่านั้นเอง

จากตัวอย่างข้างต้นพบว่า ผู้ดูแลระบบ SA สมัครงานไม่ได้ เพราะว่าไม่มีความสามารถในการเขียนโปรแกรม กล่าวอีกนัยหนึ่งคือ คลาส SA ไม่ได้ Implements อินเตอร์เฟส IProgramable ซึ่งเป็น Type ของพารามิเตอร์ p ของเมธอด AdmitProgramable() นั่นเอง

เพื่อให้ดูวุ่นวายและโหดร้ายยิ่งขึ้น ผู้เขียนแก้ไขคลาส SA เป็นคลาสลูกสืบทอดมาจากคลาสแม่ Programmer ดังโค้ดต่อไปนี้

VB 2005 (Class1.vb)	VC# 2005 (Class1.cs)

Public Class SA
  Inherits Programmer
 	
  Public Sub Admin()
  MessageBox.Show("SA เป็น Admin ได้ด้วย")
  End Sub
End Class
	public class SA:Programmer
	{
	  public void Admin()
	  {
	  MessageBox.Show("SA เป็น Admin ได้ด้วย");
	  }
	}

ผู้ดูแล SA คนนี้เก่งกว่าคนเดิม เพราะว่าสามารถเขียนโปรแกรมได้ด้วย ไม่ว่าจะเป็น VB, VC#, OOP รวมถึงเป็นผู้ดูแลระบบได้อีกด้วย กล่าวคือ เมธอด VBProgramming(), CSProgramming() และเมธอด OOPProgramming() ได้มาจากคลาสแม่ Programmer เพราะว่าเมธอดเหล่านี้มีขอบเขตแบบ Public นั่นเอง ส่วนเมธอด Admin() เป็นความสามารถของคลาส SA

ในกรณีที่คุณเจอการสืบทอดคลาสซ้อนกับการ Implements อินเตอร์เฟสแบบนี้ เวลาที่คุณต้องการให้คลาส SA แสดงความสามารถ หรือกล่าวได้อีกนัยหนึ่งคือ ต้องการสั่งให้เมธอดต่างๆ ของคลาส SA ทำงาน ให้คุณดูลำดับของการตรวจสอบเป็นสำคัญ สรุปประเด็นที่น่าสนใจมี 3 ส่วนคือ

1. คลาส SA ทำหน้าที่เป็นคลาสลูกสืบทอดมาจากคลาสแม่ Programmer
2. การประกาศรับสมัครงาน (เมธอด AdmitProgramable() ของคลาส Company()) ต้องการคนที่เขียนโปรแกรมได้ (พารามิเตอร์ p มี Type เป็น IProgramable())
3. แต่ความสามารถในการเขียนโปรแกรมของคลาสลูก SA ได้มาจากคลาสแม่ Programmer ไม่ใช่ความสามารถจากคลาส SA โดยตรง

ถ้าคุณพบเงื่อนไขในลักษณะนี้ วิธีการตรวจสอบ Type ของพารามิเตอร์ p ของเมธอด Admit Programable() ต้องเปลี่ยนไปนั่นคือ คุณต้องตรวจสอบ Type ของคลาสลูก SA ก่อนคลาสแม่ Programmer เสมอ เพื่อให้คลาสลูก SA มีสิทธิหรือมีโอกาสได้ทำงาน ส่งผลให้เมธอด AdmitProgramable() ของคลาส Company ต้องแก้ไขโค้ดใหม่ ดังนี้

VB 2005 (Company.vb)

```
Option Explicit On
Option Strict On

Public Class Company
    Public Sub AdmitProgramable(ByVal p As IProgramable)
        If TypeOf p Is SA Then
            Dim _s As SA = DirectCast(p, SA)
            _s.VBProgramming()
            _s.CSProgramming()
            _s.OOPProgramming()
            _s.Admin()
        ElseIf TypeOf p Is Programmer Then
            Dim _p As Programmer = DirectCast(p,
Programmer)
            _p.VBProgramming()
            _p.CSProgramming()
            _p.OOPProgramming()
        Else
            p.VBProgramming()
            p.CSProgramming()
        End If
    End Sub
End Class
```

VC# 2005 (Company.cs)

```
public class Company
{
    public void AdmitProgramable(IProgramable p)
    {
        if (p is SA)
        {
            SA _s = (SA)p;
            _s.VBProgramming();
            _s.CSProgramming();
            _s.OOPProgramming();
            _s.Admin();
        }
        else if (p is Programmer)
        {
            Programmer _p = (Programmer)p;
            _p.VBProgramming();
            _p.CSProgramming();
            _p.OOPProgramming();
        }
        else
        {
            p.VBProgramming();
            p.CSProgramming();
        }
    }
}
```

โค้ดข้างต้นเราจะตรวจสอบพารามิเตอร์ p ก่อนเลยว่า มี Type เป็น SA (คลาสลูก) หรือไม่ ถ้าบริษัท myCompany ประกาศรับสมัครงานใหม่ แล้ว SA คนนี้ไปสมัครงาน ดังโค้ดต่อไปนี้

VB 2005 (Form1.vb)

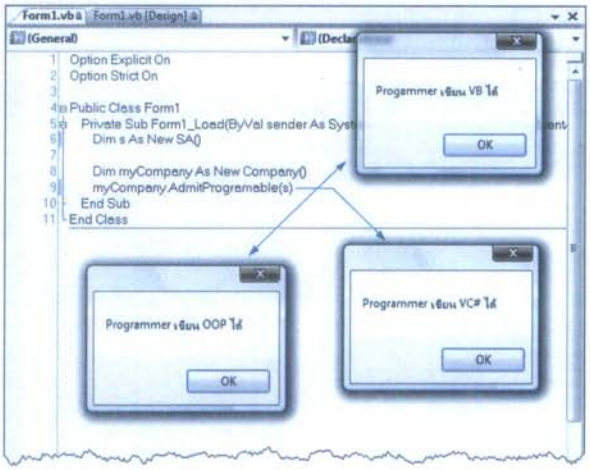
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim s As New SA()

        Dim myCompany As New Company()
        myCompany.AdmitProgramable(s)
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    SA s = new SA();

    Company myCompany = new Company();
    myCompany.AdmitProgramable(s);
}
```

เห็นได้ว่าคราวนี้ SA ที่ชื่อว่า s สมัครงานได้แล้ว (โค้ดรันผ่าน) แสดงฝีมือได้อย่างเต็มที่ ดังรูปที่ 12-12

รูปที่ 12-12 เมธอดทั้งหมดของออบเจ็กต์ SA ที่ชื่อว่า s



จากรูปที่ 12-12 แม้ว่า SA คนนี้จะเขียนโปรแกรมได้ก็จริง แต่เป็นความสามารถที่ได้มาจากคลาสแม่ Programmer ถ้าบริษัท myCompany ต้องการเพียงเท่านี้ ถือว่าได้คนเข้าทำงานตามความต้องการเสร็จสิ้นภารกิจ

แต่ถ้าบริษัท myCompany คิดว่าเป็น SA ทั้งที น่าจะเขียนโปรแกรมได้เก่งกว่า Programmer จะได้ทำงานอย่างมีประสิทธิภาพขึ้นไปอีก ให้คุณแก้ไขความสามารถของ SA ใหม่ แยกเป็น 2 กรณีคือ

- **กรณี VB 2005** : ให้คุณทำ Overload เมธอด VBProgramming(), CSProgramming() และเมธอด OOPProgramming() ที่ได้จากคลาสแม่ Programmer() เพื่อแก้ไขให้ SA เขียนโปรแกรมเก่งกว่า

VB 2005 (Class1.vb)

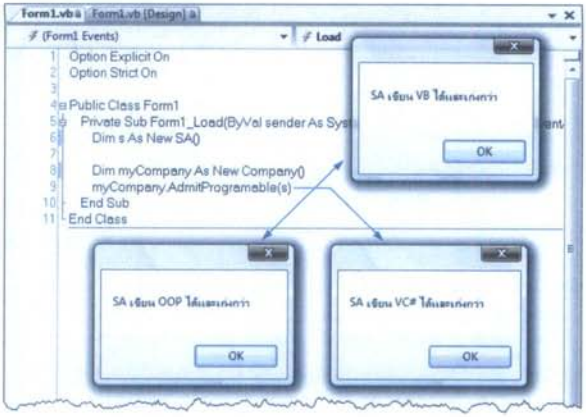
```
Public Class SA
    Inherits Programmer

    Public Overloads Sub VBProgramming()
        MessageBox.Show("SA เขียน VB ได้และเก่งกว่า")
    End Sub

    Public Overloads Sub CSProgramming()
        MessageBox.Show("SA เขียน VC# ได้และเก่งกว่า")
    End Sub

    Public Overloads Sub OOPProgramming()
        MessageBox.Show("SA เขียน OOP ได้และเก่งกว่า")
    End Sub

    Public Sub Admin()
        MessageBox.Show("SA เป็น Admin ได้ด้วย")
    End Sub
End Class
```

- **กรณี** VC# 2005 : ให้คุณใช้คำสั่ง new ทั้ง 3 เมธอดใหม่ ดังโค้ดต่อไปนี้

VC# 2005 (Class1.cs)

```
public class SA:Programmer
{
    public new void VBProgramming()
    {
        MessageBox.Show("SA เขียน VB ได้และเก่งกว่า");
    }

    public new void CSProgramming()
    {
        MessageBox.Show("SA เขียน VC# ได้และเก่งกว่า");
    }

    public new void OOPProgramming()
    {
        MessageBox.Show("SA เขียน OOP ได้และเก่งกว่า");
    }

    public void Admin()
    {
        MessageBox.Show("SA เป็น Admin ได้ด้วย");
    }
}
```

รูปที่ 12-13
ผลการทำงานของ SA คนใหม่



จากรูปที่ 12-13 เห็นได้ว่า SA คนใหม่ทำงานได้อย่างดีเยี่ยม ดูแลระบบและเขียนโปรแกรมเองก็ได้

การใช้งานอินเตอร์เฟสในฐานะ Type

นอกจากที่เราจะใช้อินเตอร์เฟสในฐานะความสามารถ หรือคุณลักษณะเพิ่มเติมของคลาสแล้ว เรายังใช้อินเตอร์เฟสในฐานะ Type ประเภทหนึ่งได้อีกด้วย ดังตัวอย่างที่ 12-8 การใช้งานอินเตอร์เฟสในฐานะ Type

โค้ด VB 2005 และ VC# 2005 ที่ 12-8 การใช้งานอินเตอร์เฟสในฐานะ Type

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IVBProgramable
    Sub VBProgramming()
    Sub VBDatabaseProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface ICSProgramable
{
    void CSProgramming();
    void CSDatabaseProgramming();
}
```

โค้ด VB 2005 ที่ 12-8 การใช้งานอินเตอร์เฟสในฐานะ Type (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Implements IVBProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา VB")
    End Sub
```

```
    Public Sub VBDatabaseProgramming() Implements IVBProgramable.VBDatabaseProgramming
        MessageBox.Show("เขียนโปรแกรมภาษา VB ด้านฐานข้อมูล")
    End Sub
End Class

Public Class SA
    Public Sub ManageSystem()
        MessageBox.Show("ดูแลระบบ")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-8 การใช้งานอินเตอร์เฟสในฐานะ Type (Class1.cs)

```
public class Programmer : ICSProgramable
{
    public void CSProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา CS");
    }

    public void CSDatabaseProgramming()
    {
        MessageBox.Show("เขียนโปรแกรมภาษา CS ด้านฐานข้อมูล");
    }
}

public class SA
{
    public void ManageSystem()
    {
        MessageBox.Show("ดูแลระบบ");
    }
}
```

ผู้เขียนสร้างคลาส Programmer และคลาส SA ขึ้นมา กำหนดให้คลาส Programmer เพียงคลาสเดียวที่ Implements อินเตอร์เฟส IVBProgramable (ICSProgramable) ส่วนการใช้งานอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-8 การใช้งานอินเตอร์เฟสในฐานะ Type

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On


Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Programmer()
        Dim s As New SA()

        If (TypeOf p Is IVBProgramable) Then
            MessageBox.Show("ออบเจ็กต์ p มีอินเตอร์เฟส
IVBProgramable")
        End If

        If (TypeOf s Is IVBProgramable) Then
            MessageBox.Show("ออบเจ็กต์ s มีอินเตอร์เฟส
IVBProgramable")
        End If

        Dim pi As IVBProgramable
        pi = New Programmer()
        pi.VBProgramming()
        pi.VBDatabaseProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        Programmer p = new Programmer();
        SA s = new SA();

        if (p is ICSProgramable)
        {
            MessageBox.Show("ออบเจ็กต์ p มีอินเตอร์เฟส
ICSProgramable");
        }

        if (s is ICSProgramable)
        {
            MessageBox.Show("ออบเจ็กต์ s มีอินเตอร์เฟส
ICSProgramable");
        }

        ICSProgramable pi;
        pi = new Programmer();
        pi.CSProgramming();
        pi.CSDatabaseProgramming();
    }
```

วิธีการตรวจสอบคลาสที่เราสนใจว่ามีการ Implements อินเตอร์เฟสใดๆ อยู่หรือไม่ ให้คุณใช้ตัวดำเนินการ Is (is) ผลการทำงาน ดังรูปที่ 12-14

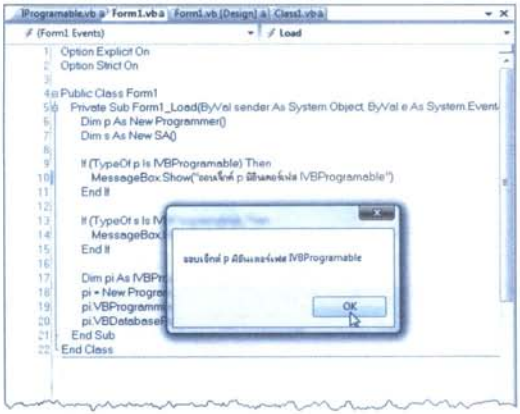
VB 2005

```
        Dim p As New Programmer()
        Dim s As New SA()

        If (TypeOf p Is IVBProgramable) Then
            MessageBox.Show("ออบเจ็กต์ p มีอินเตอร์เฟส
IVBProgramable")
        End If

        If (TypeOf s Is IVBProgramable) Then
            MessageBox.Show("ออบเจ็กต์ s มีอินเตอร์เฟส
IVBProgramable")
        End If
```

VC# 2005

```
        if (p is ICSProgramable)
        {
            MessageBox.Show("ออบเจ็กต์ p มีอินเตอร์เฟส
ICSProgramable");
        }

        if (s is ICSProgramable)
        {
            MessageBox.Show("ออบเจ็กต์ s มีอินเตอร์เฟส
ICSProgramable");
        }
```

รูปที่ 12-14
ผลการตรวจสอบอินเตอร์เฟส IVBProgramable (ICSProgramable)



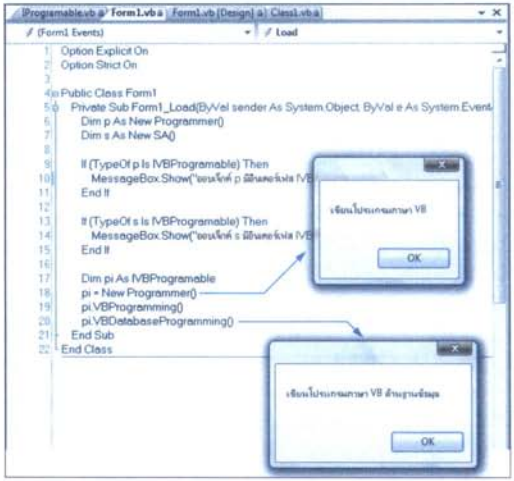
จากรูปที่ 12-14 ออบเจ็กต์ Programmer ที่ชื่อว่า p มีการ Implements อินเตอร์เฟส IVBProgramable (ICSProgramable)

ในกรณีที่คุณนำอินเตอร์เฟสมาใช้เป็น Type ให้กับตัวแปร คุณสามารถสร้างอินสแตนซ์ของคลาสที่ Implements อินเตอร์เฟสดังกล่าวได้อีกด้วย ดังโค้ดต่อไปนี้

VB 2005	VC# 2005

Dim pi As IVBProgramable
pi = New Programmer()
pi.VBProgramming()
pi.VBDatabaseProgramming()
	ICSProgramable pi;
	pi = new Programmer();
	pi.CSProgramming();
	pi.CSDatabaseProgramming();

รูปที่ 12-15
ผลการทำงานของออบเจ็กต์ pi



ออบเจ็กต์ Programmer ที่ชื่อว่า pi ที่ได้มาก็จะใช้ความสามารถของอินสแตนซ์ Programmer ในฐานะคนเขียนโปรแกรม IVBProgramable (ICSProgramable) ได้นั่นเอง

การทำ Explicit Cast Interface

อินเตอร์เฟสคือ Type ประเภทหนึ่ง จึงไม่แปลกที่เราสามารถแปลงออบเจ็กต์ให้อยู่ในฐานะอินเตอร์เฟส และมีเงื่อนไขเพียงอย่างเดียวนั่นคือ ออบเจ็กต์ที่นำมาแปลงต้อง Implements อินเตอร์เฟสดังกล่าวด้วย ให้ดูตัวอย่างที่ 12-9 การทำ Explicit Cast Interface

โค้ด VB 2005 และ VC# 2005 ที่ 12-9 การทำ Explicit Cast Interface

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IProgramable
    Sub DatabaseProgramming()
    Sub OOPProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IProgramable
{
    void DatabaseProgramming();
    void OOPProgramming();
}
```

ผู้เขียนสร้างอินเตอร์เฟสที่ชื่อว่า IProgramable ระบุข้อบังคับไว้ว่าถ้ามีความสามารถในการเขียนโปรแกรม ต้องเขียนโปรแกรมด้านฐานข้อมูล (มีเมธอด DatabaseProgramming()) และเขียนโปรแกรม OOP ได้ (มีเมธอด OOPProgramming())

โค้ด VB 2005 ที่ 12-9 การทำ Explicit Cast Interface (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Implements IProgramable

    Public Sub DatabaseProgramming() Implements IProgramable.DatabaseProgramming
        MessageBox.Show("Person เขียนโปรแกรมด้านฐานข้อมูลด้วย VB")
    End Sub

    Public Sub OOPProgramming() Implements IProgramable.OOPProgramming
        MessageBox.Show("Person เขียน OOP ด้วย VB")
    End Sub
End Class

Public Class Programmer
    Implements IProgramable

    Public Sub DatabaseProgramming() Implements IProgramable.DatabaseProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมด้านฐานข้อมูลด้วย VC#")
    End Sub
```

```
    Public Sub OOPProgramming() Implements IProgramable.OOPProgramming
        MessageBox.Show("Programmer เขียน OOP ด้วย VC#")
    End Sub

    Public Sub DocumentWriter()
        MessageBox.Show("Programmer ทำเอกสารประกอบโปรเจ็กต์")
    End Sub
End Class

Public Class Manager

End Class
```

โค้ด VC# 2005 ที่ 12-9 การทำ Explicit Cast Interface (Class1.cs)

```
public class Person : IProgramable
{
    public void DatabaseProgramming()
    {
        MessageBox.Show("Person เขียนโปรแกรมด้านฐานข้อมูลด้วย VB");
    }

    public void OOPProgramming()
    {
        MessageBox.Show("Person เขียน OOP ด้วย VB");
    }
}

public class Programmer : IProgramable
{
    public void DatabaseProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมด้านฐานข้อมูลด้วย VC#");
    }

    public void OOPProgramming()
    {
        MessageBox.Show("Programmer เขียน OOP ด้วย VC#");
    }

    public void DocumentWriter()
    {
        MessageBox.Show("Programmer ทำเอกสารประกอบโปรเจ็กต์");
    }
}

public class Manager
{

}
```

ผู้เขียนสร้างคลาสขึ้นมา 3 คลาส ได้แก่

- คนธรรมดา (คลาส Person) มีความสามารถในการเขียนโปรแกรม (Implements อินเตอร์เฟส IProgramable)
- โปรแกรมเมอร์ (คลาส Programmer) มีความสามารถในการเขียนโปรแกรมเช่นกัน (Implements อินเตอร์เฟส IProgramable) โดยที่เพิ่มเติมความสามารถให้โปรแกรมเมอร์ทำเอกสารประกอบโปรเจ็กต์ได้ (มีเมธอด DocumentWriter())
- ผู้จัดการ (คลาส Manager) ไม่สามารถเขียนโปรแกรมได้

โค้ด VB 2005 และ VC# 2005 ที่ 12-9 การทำ Explicit Cast Interface

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ip As IProgramable

        Dim p As New Person()
        ip = DirectCast(p, IProgramable)
        ip.OOPProgramming()
        ip.DatabaseProgramming()

        Dim pm As New Programmer()
        p.OOPProgramming()
        p.DatabaseProgramming()
        pm.DocumentWriter()

        ip = DirectCast(pm, IProgramable)
        ip.OOPProgramming()
        ip.DatabaseProgramming()
        'ip.DocumentWriter()

        'Dim m As New Manager()
        'ip = DirectCast(m, IProgramable)
        'ip.OOPProgramming()
        'ip.DatabaseProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    IProgramable ip;

    Person p = new Person();
    ip = (IProgramable)p;
    ip.OOPProgramming();
    ip.DatabaseProgramming();

    Programmer pm = new Programmer();
    pm.OOPProgramming();
    pm.DatabaseProgramming();
    pm.DocumentWriter();

    ip = (IProgramable)pm;
    ip.OOPProgramming();
    ip.DatabaseProgramming();
    //ip.DocumentWriter();

    //Manager m = new Manager();
    //ip = (IProgramable)m;
    //ip.OOPProgramming();
    //ip.DatabaseProgramming();
}
```

วิธีการดูโค้ดชุดนี้ ให้ดูทีละส่วน กล่าวคือ สร้างตัวแปรที่ชื่อว่า ip มี Type เป็นอินเตอร์เฟส IProgramable

VB 2005	VC# 2005
`Dim ip As IProgramable`	`IProgramable ip;`

ต่อมาสร้างออบเจ็กต์ Person ชื่อว่า p ขึ้นมา จากนั้นให้ Cast ออบเจ็กต์ p (มี Type เป็น Person) ให้อยู่ในฐานะอินเตอร์เฟส IProgramable เพราะความที่คลาส Person (Implements อินเตอร์เฟส IProgramable) ไม่มีการเพิ่มเมธอดใหม่ใดๆ เข้าไป

ส่งผลให้การมองคนธรรมดา (p) ในฐานะความสามารถในการเขียนโปรแกรม (ip) ไม่มีผลกระทบแต่อย่างใด เพราะว่าคุณสามารถใช้เมธอด OOPProgramming() และเมธอด DatabaseProgramming() ครบทุกเมธอดที่อยู่ในคลาส Person ได้ตามปกติ

VB 2005

```
Dim p As New Person()
ip = DirectCast(p, IProgramable)
ip.OOPProgramming()
ip.DatabaseProgramming()
```

VC# 2005

```
Person p = new Person();
ip = (IProgramable)p;
ip.OOPProgramming();
ip.DatabaseProgramming();
```

ต่อมาลองสร้างออบเจ็กต์ Programmer ที่ชื่อว่า pm ขึ้นมา พบว่าเราสามารถใช้เมธอด OOP Programming(), DatabaseProgramming() และเมธอด DocumentWriter() ครบถ้วนทุกเมธอดที่อยู่ในคลาส Programmer ตามปกติ เป็นการมองโปรแกรมเมอร์ (New Programmer()) ในฐานะโปรแกรมเมอร์ (คลาส Programmer) ดังรูปที่ 12-6

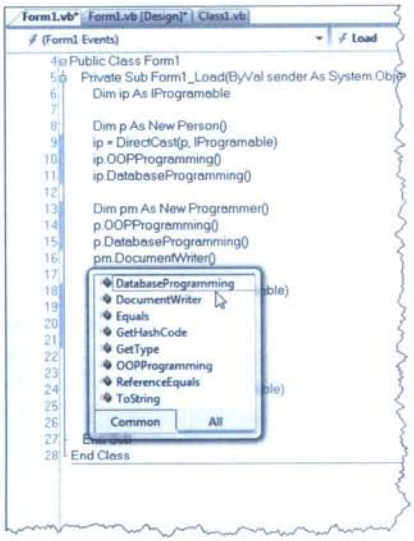
VB 2005

```
Dim pm As New Programmer()
p.OOPProgramming()
p.DatabaseProgramming()
pm.DocumentWriter()
```

VC# 2005

```
Programmer pm = new Programmer();
pm.OOPProgramming();
pm.DatabaseProgramming();
pm.DocumentWriter();
```

รูปที่ 12-16 เมธอดของออบเจ็กต์ Programmer ในฐานะ Programmer Type



แต่ถ้าคุณมองโปรแกรมเมอร์ pm ในฐานะของความสามารถในการเขียนโปรแกรม (อินเตอร์เฟส IProgramable) ก็จะทำให้ความเป็นโปรแกรมเมอร์หมดไปเหลือแต่เพียงความสามารถในการเขียนโปรแกรมที่ระบุไว้ในอินเตอร์เฟส IProgramable เท่านั้น ดังรูปที่ 12-17

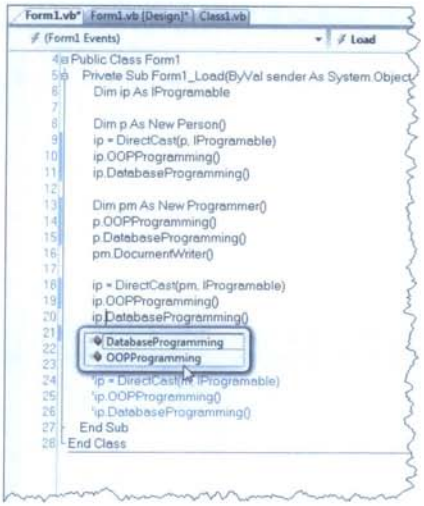
VB 2005

```
ip = DirectCast(pm, IProgramable)
ip.OOPProgramming()
ip.DatabaseProgramming()
```

VC# 2005

```
ip = (IProgramable)pm;
ip.OOPProgramming();
ip.DatabaseProgramming();
```

รูปที่ 12-17 เมธอดของออบเจ็กต์ Programmer ในฐานะอินเตอร์เฟส IProgramable



ถ้าคุณสั่งให้ ip เรียกใช้เมธอด DocumentWriter() ซึ่งเป็นของคลาส Programmer ก็จะเกิดข้อผิดพลาดขึ้น เพราะว่าความสามารถในการเขียนโปรแกรมไม่ได้ระบุไว้ว่าต้องทำเอกสารประกอบโปรเจ็กต์ได้ กล่าวคือ อินเตอร์เฟส IProgramable ไม่มีเมธอด DocumentWriter() นั่นเอง

VB 2005

```
ip.DocumentWriter()
```

VC# 2005

```
ip.DocumentWriter();
```

ส่วนผู้จัดการ Manager ที่ชื่อว่า m คุณไม่สามารถมองผู้จัดการในฐานะคนเขียนโปรแกรมได้อย่างแน่นอน เพราะว่าคลาส Manager ไม่ได้ Implements อินเตอร์เฟส IProgramable ไว้นั่นเอง

VB 2005

```
Dim m As New Manager()
ip = DirectCast(m, IProgramable)
ip.OOPProgramming()
ip.DatabaseProgramming()
```

VC# 2005

```
Manager m = new Manager();
ip = (IProgramable)m;
ip.OOPProgramming();
ip.DatabaseProgramming();
```

การใช้งาน Interface ในฐาน: Return Type

การใช้งานอินเตอร์เฟสในฐานะค่าที่ส่งกลับมาจากเมธอด (Return Type) เป็นอีกเทคนิคหนึ่งที่ช่วยให้การเขียนโปรแกรมมีความยืดหยุ่นเพิ่มมากขึ้น คุณสามารถประยุกต์ใช้อินเตอร์เฟสในฐานะ Return Type มาทำหน้าที่เป็นโรงงานผลิตออบเจ็กต์ โดยมีข้อแม้ว่าออบเจ็กต์ที่ผลิตออกมาต้อง Implements อินเตอร์เฟสที่ทำหน้าที่ Return Type นั่นเอง ให้ดูตัวอย่างที่ 12-10 การใช้งาน Interface ในฐานะ Return Type

โค้ด VB 2005 และ VC# 2005 ที่ 12-10 การใช้งาน Interface ในฐานะ Return Type

VB 2005 (Programable.vb)

```
Option Explicit On
Option Strict On

Public Interface IProgramable
    Sub VBProgramming()
    Sub CSProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IProgramable
{
    void VBProgramming();
    void CSProgramming();
}
```

เริ่มต้นสร้างอินเตอร์เฟสที่ชื่อว่า IProgramable ขึ้นมา หมายถึง ความสามารถในการเขียนโปรแกรม

โค้ด VB 2005 ที่ 12-10 การใช้งาน Interface ในฐานะ Return Type (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Implements IProgramable

    Public Sub VBProgramming() Implements IProgramable.VBProgramming
        MessageBox.Show("Person เขียน VB ได้")
    End Sub

    Public Sub CSProgramming() Implements IProgramable.CSProgramming
        MessageBox.Show("Person เขียน VC# ได้")
    End Sub
End Class

Public Class Programmer
    Implements IProgramable

    Public Sub VBProgramming() Implements IProgramable.VBProgramming
        MessageBox.Show("Programmer เขียน VB ได้")
    End Sub

    Public Sub CSProgramming() Implements IProgramable.CSProgramming
        MessageBox.Show("Programmer เขียน VC# ได้")
    End Sub

    Public Sub OOPProgramming()
        MessageBox.Show("Programmer เขียน OOP ได้")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-10 การใช้งาน Interface ในฐานะ Return Type (Class1.cs)

```
public class Person : IProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Person เขียน VB ได้");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Person เขียน VC# ได้");
    }
}

public class Programmer : IProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Programmer เขียน VB ได้");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Programmer เขียน VC# ได้");
    }

    public void OOPProgramming()
    {
        MessageBox.Show("Programmer เขียน OOP ได้");
    }
}
```

ต่อมาสร้างคลาส Programmer และคลาส Person ขึ้นมา กำหนดให้ทั้ง 2 คลาส Implements อินเตอร์เฟส IProgramable

โค้ด VB 2005 และ VC# 2005 ที่ 12-10 การใช้งาน Interface ในฐานะ Return Type

VB 2005 (ProgramableFactory.vb)

```
Option Explicit On
Option Strict On

Public Class ProgramableFactory
    Public Function Generator(ByVal str As String) As
IProgramable
        If str.ToLower() = "p" Then
            Return New Person()
        End If

        Return New Programmer()
    End Function
End Class
```

VC# 2005 (ProgramableFactory.cs)

```
public IProgramable Generator(string str)
{
    if (str.ToLower() == "p")
    {
        return new Person();
    }

    return new Programmer();
}
```

ต่อมาสร้างคลาสที่ชื่อว่า ProgramableFactory ขึ้นมา ทำหน้าที่เป็นโรงงานผลิตสิ่งที่เขียนโปรแกรมได้ โดยการสร้างเมธอดที่ชื่อว่า Generator() ขึ้นมา ต้องการพารามิเตอร์ 1 ตัวชื่อว่า str มี Type เป็น String (string) คืนค่ากลับเป็นอินเตอร์เฟส IProgramable

พารามิเตอร์ str ทำหน้าที่เป็นคนกลางคอยรับเงื่อนไขในการผลิตสิ่งที่เขียนโปรแกรมได้ ในกรณีนี้มี 2 คลาสคือ คลาส Person และคลาส Programmer กล่าวคือ

- ถ้าเป็นตัวอักษร p ให้คืนค่าเป็นอินสแตนซ์ของออบเจ็กต์ Person
- แต่ถ้าเป็นตัวอักษรอื่นๆ ให้คืนค่าเป็นอินสแตนซ์ของออบเจ็กต์ Programmer

โค้ด VB 2005 และ VC# 2005 ที่ 12-10 การใช้งาน Interface ในฐานะ Return Type

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim fp As New ProgramableFactory()
        Dim IP1 As IProgramable
        IP1 = fp.Generator("p")
        IP1.VBProgramming()
        IP1.CSProgramming()

        Dim IP2 As IProgramable
        IP2 = fp.Generator("abc")
        IP2.VBProgramming()
        IP2.CSProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    ProgramableFactory fp = new ProgramableFactory();
    IProgramable IP1;
    IP1 = fp.Generator("p");
    IP1.VBProgramming();
    IP1.CSProgramming();

    IProgramable IP2;
    IP2 = fp.Generator("abc");
    IP2.VBProgramming();
    IP2.CSProgramming();
}
```

เริ่มต้นสร้างโรงงานผลิตที่ชื่อว่า fp ขึ้นมาก่อน

VB 2005

```
Dim fp As New ProgramableFactory()
```

VC# 2005

```
ProgramableFactory fp = new ProgramableFactory();
```

ถ้าคุณต้องการผลิตออบเจ็กต์ Person ให้ส่งเงื่อนไขการผลิตเป็นตัวอักษร p ให้กับเมธอด Generator() ของออบเจ็กต์ fp

VB 2005

```
Dim IP1 As IProgramable
IP1 = fp.Generator("p")
IP1.VBProgramming()
IP1.CSProgramming()
```

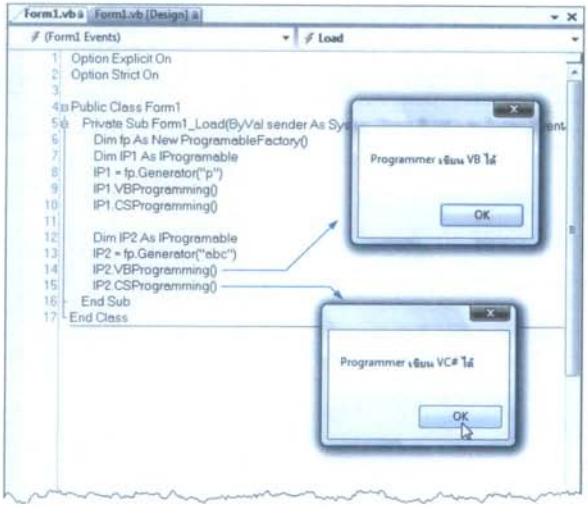
VC# 2005

```
IProgramable IP1;
IP1 = fp.Generator("p");
IP1.VBProgramming();
IP1.CSProgramming();
```

แต่ถ้าต้องการผลิตออบเจ็กต์ Programmer ให้ส่งตัวอักษรอะไรก็ได้เข้าไป ดังรูปที่ 12-18

VB 2005	VC# 2005
Dim IP2 As IProgramable IP2 = fp.Generator("abc") IP2.VBProgramming() IP2.CSProgramming()	IProgramable IP2; IP2 = fp.Generator("abc"); IP2.VBProgramming(); IP2.CSProgramming();

รูปที่ 12-18
ออบเจ็กต์ Programmer ในฐานะอินเตอร์เฟส IP2 ที่ได้



การใช้งาน Abstract Class ร่วมกับ Interface

ในการออกแบบสร้างคลาสขึ้นมาใช้ในโปรเจ็กต์ของคุณ จะมีความต้องการหรือเงื่อนไขบางอย่าง ที่บังคับให้คุณต้องสร้างคลาสที่ทำหน้าที่เหมือนกันและแตกต่างกันบางส่วน เพื่อรองรับการทำงานที่แตกต่างกัน หรือรองรับเงื่อนไขในการใช้งานไม่เหมือนกัน เราจะออกแบบอย่างไร วิธีการหนึ่งที่นำมาใช้แก้ปัญหาดังกล่าว คือ การใช้งาน Abstract Class ร่วมกับ Interface เข้ามาช่วย

ให้มองว่าคุณสมบัติหรือความสามารถที่เหมือนกัน ให้เก็บไว้ที่ Abstract Class ส่วนคุณสมบัติหรือความสามารถเฉพาะเจาะจง ให้เพิ่มเติมใหม่เองในภายหลัง โดยการ Implements อินเตอร์เฟสแทน ให้คุณดูตัวอย่างที่ 12-11 การใช้งาน Abstract Class ร่วมกับ Interface เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 12-11 การใช้งาน Abstract Class ร่วมกับ Interface

VB 2005 (absProgrammer.vb)

```
Option Explicit On
Option Strict On

Public MustInherit Class absProgrammer
    Protected _FullName As String = ""

    Public Property FullName() As String
        Get
            Return _FullName
        End Get
        Set(ByVal value As String)
            _FullName = value
        End Set
    End Property

    Public Sub BasicProgramming()
        MessageBox.Show("พื้นฐานการเขียนโปรแกรม")
    End Sub
End Class
```

VC# 2005 (absProgrammer.cs)

```
public abstract class absProgrammer
{
    protected string _FullName = "";
    public string FullName
    {
        get { return _FullName; }
        set { _FullName = value; }
    }

    public void BasicProgramming()
    {
        MessageBox.Show("พื้นฐานการเขียนโปรแกรม");
    }
}
```

ผู้เขียนคิดว่าถ้าจะกล่าวถึงอาชีพการเขียนโปรแกรม (Abstract Class ที่ชื่อว่า absProgrammer) คิดว่าโปรแกรมเมอร์น่าจะเป็นต้นแบบได้ดีที่สุด อย่างน้อยก็ต้องมีชื่อ-สกุล (คุณสมบัติ FullName) และมีความสามารถในการเขียนโปรแกรมขั้นพื้นฐาน (มีเมธอด BasicProgramming())

โค้ด VB 2005 และ VC# 2005 ที่ 12-11 การใช้งาน Abstract Class ร่วมกับ Interface

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On


Public Interface IVBProgramable
    Sub VBProgramming()
End Interface

Public Interface ICSProgramable
    Sub CSProgramming()
End Interface

Public Interface IJAVAProgramable
    Sub JAVAProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IVBProgramable
{
    void VBProgramming();
}

public interface ICSProgramable
{
    void CSProgramming();
}

public interface IJAVAProgramable
{
    void JAVAProgramming();
}
```

ต่อมาสร้างความสามารถในการเขียน VB (IVBProgramable), VC# (ICSProgramable) และภาษา JAVA (JAVAProgramable) ขึ้นมา เราจะมองว่านี่คือ ความสามารถส่วนบุคคล ซึ่งแต่ละคนมีความถนัดไม่เท่ากันและไม่เหมือนกันด้วย

โค้ด VB 2005 ที่ 12-11 การใช้งาน Abstract Class ร่วมกับ Interface (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Programmer
    Inherits absProgrammer
    Implements IVBProgramable, ICSProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("Programmer เขียน VB")
    End Sub

    Public Sub CSProgramming() Implements ICSProgramable.CSProgramming
        MessageBox.Show("Programmer เขียน VC#")
    End Sub
End Class

Public Class SA
    Inherits absProgrammer
    Implements IVBProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("SA เขียน VB")
    End Sub

    Public Sub ManageSystem()
        MessageBox.Show("SA ดูแลระบบได้")
    End Sub
End Class

Public Class Hacker
    Inherits absProgrammer
    Implements IVBProgramable, ICSProgramable, IJAVAProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("Hacker เขียน VB")
    End Sub

    Public Sub CSProgramming() Implements ICSProgramable.CSProgramming
        MessageBox.Show("Hacker เขียน VC#")
    End Sub

    Public Sub JAVAProgramming() Implements IJAVAProgramable.JAVAProgramming
```

```
        MessageBox.Show("Hacker เขียน JAVA")
    End Sub

    Public Sub DDos()
        MessageBox.Show("Hacker โจมตีระบบแบบ DDos ได้")
    End Sub

    Public Sub BruceForce()
        MessageBox.Show("Hacker เจาะรหัสผ่านแบบ BruceForce ได้")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-11 การใช้งาน Abstract Class ร่วมกับ Interface (Class1.cs)

```
public class Programmer : absProgrammer ,IVBProgramable ,ICSProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Programmer เขียน VB");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Programmer เขียน VC#");
    }
}

public class SA : absProgrammer, IVBProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("SA เขียน VB");
    }

    public void ManageSystem()
    {
        MessageBox.Show("SA ดูแลระบบได้");
    }
}

public class Hacker : absProgrammer, IVBProgramable, ICSProgramable, IJAVAProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Hacker เขียน VB");
    }
```

```
    public void CSProgramming()
    {
        MessageBox.Show("Hacker เขียน VC#");
    }

    public void JAVAProgramming()
    {
        MessageBox.Show("Hacker เขียน JAVA");
    }

    public void DDos()
    {
        MessageBox.Show("Hacker โจมตีระบบแบบ DDos ได้");
    }

    public void BruceForce()
    {
        MessageBox.Show("Hacker เจาะรหัสผ่านแบบ BruceForce ได้");
    }
}
```

ผู้เขียนยกตัวอย่างขึ้นมา 3 อาชีพ ได้แก่ Programmer, SA และ Hacker ทั้ง 3 คลาสมีความเหมือนและแตกต่างกัน แต่สิ่งที่ต้องมีเหมือนกันคือ ต้องมีชื่อ-นามสกุล และต้องเขียนโปรแกรมขั้นพื้นฐานได้ แต่มีความสามารถเฉพาะตัวแตกต่างกัน

สิ่งที่เหมือนกันให้สืบทอดคลาสจาก Abstract Class ส่วนความแตกต่างของแต่ละอาชีพ ให้ Implements อินเตอร์เฟสตามที่ต้องการ ส่งผลให้คลาส Programmer ที่ต้องมีคือ ชื่อ-สกุล (คุณสมบัติ FullName), พื้นฐานการเขียนโปรแกรม (เมธอด BasicProgramming())ได้มาจากการสืบทอดคลาส absProgrammer

สมมติว่า Programmer ควรจะเขียน VB และ VC# ได้ จึงกำหนดให้ Implements อินเตอร์เฟส IVBProgramable และอินเตอร์เฟส ICSProgramable

VB 2005

```
Public Class Programmer
    Inherits absProgrammer
    Implements IVBProgramable, ICSProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("Programmer เขียน VB")
    End Sub

    Public Sub CSProgramming() Implements ICSProgramable.CSProgramming
        MessageBox.Show("Programmer เขียน VC#")
    End Sub
End Class
```

VC# 2005

```
public class Programmer : absProgrammer ,IVBProgramable ,ICSProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Programmer เขียน VB");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Programmer เขียน VC#");
    }
}
```

คลาส SA ที่ต้องมีคือ ชื่อ-สกุล (คุณสมบัติ FullName), พื้นฐานการเขียนโปรแกรม (เมธอด BasicProgramming()) ได้มาจากการสืบทอดคลาส absProgrammer สมมติว่า SA อาจจะเขียน VB ได้ จึง Implements อินเตอร์เฟส IVBProgramable เข้ามาด้วย

คลาส SA ต้องมีความสามารถในการดูแลระบบได้ ซึ่งเป็นความสามารถโดยเฉพาะของอาชีพนี้ จึงสร้างเมธอด ManageSystem() ขึ้นมาด้วย

VB 2005

```
Public Class SA
    Inherits absProgrammer
    Implements IVBProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("SA เขียน VB")
    End Sub

    Public Sub ManageSystem()
        MessageBox.Show("SA ดูแลระบบได้")
    End Sub
End Class
```

VC# 2005

```
public class SA : absProgrammer, IVBProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("SA เขียน VB");
    }

    public void ManageSystem()
    {
        MessageBox.Show("SA ดูแลระบบได้");
    }
}
```

ส่วน Hacker ทำเป็นหมดทุกอย่าง โดยความสามารถเฉพาะตัวของคลาส Hacker คือ โจมตีระบบได้ (เมธอด DDos()) และหารหัสผ่านได้ (เมธอด BruceForce())

VB 2005

```
Public Class Hacker
    Inherits absProgrammer
    Implements IVBProgramable, ICSProgramable, IJAVAProgramable

    Public Sub VBProgramming() Implements IVBProgramable.VBProgramming
        MessageBox.Show("Hacker เขียน VB")
    End Sub

    Public Sub CSProgramming() Implements ICSProgramable.CSProgramming
        MessageBox.Show("Hacker เขียน VC#")
    End Sub

    Public Sub JAVAProgramming() Implements IJAVAProgramable.JAVAProgramming
        MessageBox.Show("Hacker เขียน JAVA")
    End Sub

    Public Sub DDos()
        MessageBox.Show("Hacker โจมตีระบบแบบ DDos ได้")
    End Sub

    Public Sub BruceForce()
        MessageBox.Show("Hacker เจาะรหัสผ่านแบบ BruceForce ได้")
    End Sub
End Class
```

VC# 2005

```
public class Hacker : absProgrammer, IVBProgramable, ICSProgramable, IJAVAProgramable
{
    public void VBProgramming()
    {
        MessageBox.Show("Hacker เขียน VB");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Hacker เขียน VC#");
    }

    public void JAVAProgramming()
    {
        MessageBox.Show("Hacker เขียน JAVA");
    }
```

```
    public void DDos()
    {
        MessageBox.Show("Hacker โจมตีระบบแบบ DDos ได้");
    }

    public void BruceForce()
    {
        MessageBox.Show("Hacker เจาะรหัสผ่านแบบ BruceForce ได้");
    }
}
```

ส่วนการใช้งานคลาสทั้ง 3 อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 12-11 การใช้งาน Abstract Class ร่วมกับ Interface

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Programmer()
        p.FullName = "ศุภชัย สมพานิช"
        p.BasicProgramming()
        p.VBProgramming()
        p.CSProgramming()

        Dim s As New SA()
        s.FullName = "ชลธิชา ศรีดา"
        s.BasicProgramming()
        s.VBProgramming()
        s.ManageSystem()

        Dim h As New Hacker()
        h.FullName = "ธิติมา ยุราวรรณ"
        h.BasicProgramming()
        h.VBProgramming()
        h.CSProgramming()
        h.JAVAProgramming()
        h.DDos()
        h.BruceForce()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Programmer p = new Programmer();
    p.FullName = "ศุภชัย สมพานิช";
    p.BasicProgramming();
    p.VBProgramming();
    p.CSProgramming();

    SA s = new SA();
    s.FullName = "ชลธิชา ศรีดา";
    s.BasicProgramming();
    s.VBProgramming();
    s.ManageSystem();

    Hacker h = new Hacker();
    h.FullName = "ธิติมา ยุราวรรณ";
    h.BasicProgramming();
    h.VBProgramming();
    h.CSProgramming();
    h.JAVAProgramming();
    h.DDos();
    h.BruceForce();
}
```

สิ่งที่คุณได้จากการออกแบบลักษณะนี้ ให้มองออกเป็น 3 ส่วนคือ

1. คุณสมบัติ FullName และเมธอด BasicProgramming() ของทั้ง 3 คลาสทำงานเหมือนกัน เพราะได้มาจาก Abstract Class
2. ส่วนเมธอด VBProgramming(), CSProgramming() และเมธอด JAVAProgramming() ของแต่ละคลาสได้มาจากการ Implements อินเตอร์เฟส แม้ว่าจะมีชื่อเมธอดเหมือนกันแต่ทำงานต่างกัน ดูได้จาก MessageBox ของแต่ละคลาสที่แสดงข้อความไม่เหมือนกันนั่นเอง
3. เมธอดเพิ่มเติมของแต่ละคลาสเป็นไปตามหน้าที่ที่มันรับผิดชอบอยู่

Interface สืบทอด Interface

คุณสามารถกำหนดให้อินเตอร์เฟสสามารถสืบทอดระหว่างกันได้อีกด้วย ให้ดูตัวอย่างที่ 12-12 Interface สืบทอด Interface

โค้ด VB 2005 และ VC# 2005 ที่ 12-12 Interface สืบทอด Interface

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IProgramable
    Sub BasicProgramming()
End Interface

Public Interface IDotNETProgramable
    Inherits IProgramable

    Sub VBProgramming()
    Sub CSProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IProgramable
{
    void BasicProgramming();
}

public interface IDotNETProgramable : IProgramable
{
    void VBProgramming();
    void CSProgramming();
}
```

ผู้เขียนสร้างความสามารถในการเขียนโปรแกรมทั่วไปขึ้นมา ตั้งชื่อว่า IProgramable() มีข้อบังคับคือต้องเขียนโปรแกรมขั้นพื้นฐานได้ (มีเมธอด BasicProgramming())

ส่วนความสามารถในการเขียนโปรแกรมด้วย .NET ตั้งชื่อว่า IDotNETProgramable ผู้เขียนคิดว่าถ้าจะเขียนโปรแกรมด้วย .NET ควรที่จะเขียนโปรแกรมขั้นพื้นฐานได้บ้าง จึงกำหนดให้อินเตอร์เฟส IDotNET Programable สืบทอดอินเตอร์เฟส IProgramable กำหนดข้อบังคับของการเขียนโปรแกรมด้วย .NET ไว้ว่าต้องใช้ภาษา VB (มีเมธอด VBProgramming()) และ VC# ได้ (มีเมธอด CSProgramming())

โค้ด VB 2005 ที่ 12-12 Interface สืบทอด Interface (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Implements IProgramable

    Public Sub BasicProgramming() Implements IProgramable.BasicProgramming
        MessageBox.Show("Person เขียนโปรแกรมเบื้องต้นได้")
    End Sub
End Class

Public Class Programmer
    Implements IDotNETProgramable

    Public Sub BasicProgramming() Implements IProgramable.BasicProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้นได้")
    End Sub

    Public Sub VBPrograming() Implements IDotNETProgramable.VBProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมภาษา VB ได้")
    End Sub

    Public Sub CSProgramming() Implements IDotNETProgramable.CSProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมภาษา VC# ได้")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-12 Interface สืบทอด Interface (Class1.cs)

```
public class Person : IProgramable
{
    public void BasicProgramming()
    {
        MessageBox.Show("Person เขียนโปรแกรมเบื้องต้นได้");
    }
}

public class Programmer : IDotNETProgramable
{
    public void BasicProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้นได้");
    }

    public void VBProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมภาษา VB ได้");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมภาษา VC# ได้");
    }
}
```

สมมติว่าคนธรรมดา Person เขียนโปรแกรมทั่วไปได้เท่านั้น จึงกำหนดให้คลาส Person ทำการ Implements อินเตอร์เฟส IProgramable เพียงอย่างเดียว ส่วนโปรแกรมเมอร์ Programmer น่าจะเก่งกว่าคนธรรมดา จึงกำหนดให้ Implements อินเตอร์เฟส IDotNETProgramable

เพราะความที่อินเตอร์เฟส IDotNETProgramable สืบทอดอินเตอร์เฟส IProgramable มาด้วย ส่งผลให้คลาส Programmer ต้องเขียนโปรแกรมพื้นฐาน (เมธอด BasicProgramming()) ได้ด้วย

โค้ด VB 2005 และ VC# 2005 ที่ 12-12 Interface สืบทอด Interface

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        p.BasicProgramming()

        Dim pm As New Programmer()
        pm.BasicProgramming()
        pm.VBPrograming()
        pm.CSProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
    {
        Person p = new Person();
        p.BasicProgramming();

        Programmer pm = new Programmer();
        pm.BasicProgramming();
        pm.VBProgramming();
        pm.CSProgramming();
    }
```

คนธรรมดา Person ที่ชื่อว่า p เขียนโปรแกรมพื้นฐานได้เพียงอย่างเดียวเท่านั้น ตามข้อบังคับที่อยู่ในอินเตอร์เฟส IProgramable

VB 2005

```
Dim p As New Person()
p.BasicProgramming()
```

VC# 2005

```
Person p = new Person();
p.BasicProgramming();
```

ใน Programmer ที่ชื่อว่า pm สามารถเขียนโปรแกรมได้ทั้งแบบพื้นฐาน และแบบ .NET กล่าวได้อีกนัยหนึ่งคือ คลาส Programmer มีความสามารถในการเขียนโปรแกรมทั่วไป (อินเตอร์เฟส IProgramable) และมีความสามารถในการเขียนโปรแกรมด้วย .NET (อินเตอร์เฟส IDotNETProgramable) ดังรูปที่ 12-19

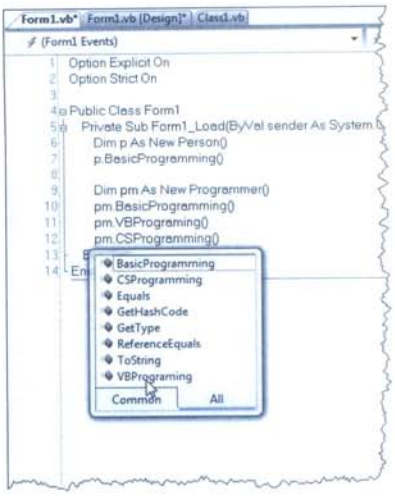
VB 2005

```
Dim pm As New Programmer()
pm.BasicProgramming()
pm.VBPrograming()
pm.CSProgramming()
```

VC# 2005

```
Programmer pm = new Programmer();
pm.BasicProgramming();
pm.VBProgramming();
pm.CSProgramming();
```

รูปที่ 12-19 เมธอดของออบเจ็กต์ Programmer ที่ชื่อว่า pm



การทำ Data Hiding ใน Interface สืบทอด Interface

ในการทำ Interface สืบทอด Interface ยังมีอีก 1 ประเด็นที่น่าสนใจนั่นคือ ถ้าข้อบังคับที่อยู่ในอินเตอร์เฟสที่สืบทอดกันอยู่ เกิดมีชื่อซ้ำกันจะทำอย่างไร คำตอบที่ได้คงไม่ใช่เพียงแค่การเปลี่ยนชื่อใหม่อย่างแน่นอน ให้ดูตัวอย่างที่ 12-13 การทำ Data Hiding ใน Interface สืบทอด Interface

โค้ด VB 2005 และ VC# 2005 ที่ 12-13 การทำ Data Hiding ใน Interface สืบทอด Interface

VB 2005 (IProgramable.vb)

```
Option Explicit On
Option Strict On

Public Interface IProgramable
    Sub BasicProgramming()
End Interface

Public Interface IDotNETProgramable
    Inherits IProgramable

    Overloads Sub BasicProgramming()
    Sub VBPrograming()
    Sub CSProgramming()
End Interface
```

VC# 2005 (IProgramable.cs)

```
public interface IProgramable
{
    void BasicProgramming();
}

public interface IDotNETProgramable : IProgramable
{
    new void BasicProgramming();
    void VBProgramming();
    void CSProgramming();
}
```

จากอินเตอร์เฟสข้างต้น ผู้เขียนจงใจกำหนดให้มีเมธอด BasicProgramming() อยู่ในอินเตอร์เฟส IDotNETProgramable ด้วย ส่งผลให้พื้นฐานการเขียนโปรแกรมทั่วไปกับพื้นฐานการเขียนโปรแกรมด้วย .NET อาจจะเหมือนกันหรือแตกต่างกันก็ได้ ดังนั้น เมื่อมีข้อบังคับซ้ำกันในอินเตอร์เฟสที่สืบทอดกันอยู่ใน VB 2005 ให้ทำ Overloads ไว้ ส่วน VC# ใช้คำสั่ง new

โค้ด VB 2005 ที่ 12-13 การทำ Data Hiding ใน Interface สืบทอด Interface (Class1.vb)

```
Option Explicit On
Option Strict On

Public Class Person
    Implements IProgramable

    Public Sub BasicProgramming() Implements IProgramable.BasicProgramming
        MessageBox.Show("Person เขียนโปรแกรมเบื้องต้นได้")
    End Sub
End Class

Public Class Programmer
    Implements IDotNETProgramable

    Public Sub BasicProgramming1() Implements IProgramable.BasicProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้น")
    End Sub

    Public Sub BasicProgramming2() Implements IDotNETProgramable.BasicProgramming
        Me.BasicProgramming1()
    End Sub

    Public Sub VBProgramming() Implements IDotNETProgramable.VBPrograming
        MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VB")
    End Sub

    Public Sub CSProgramming() Implements IDotNETProgramable.CSProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VC#")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 12-13 การทำ Data Hiding ใน Interface สืบทอด Interface (Class1.cs)

```
public class Person : IProgramable
{
    public void BasicProgramming()
    {
        MessageBox.Show("Person เขียนโปรแกรมเบื้องต้นได้");
    }
}
```

```
public class Programmer : IDotNETProgramable
{
    public void BasicProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้น");
    }

    public void VBProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VB");
    }

    public void CSProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VC#");
    }
}
```

ถ้าคุณมองว่าข้อบังคับที่มีชื่อเหมือนกันคือ สิ่งเดียวกันเหมือนกันก็จะใช้วิธี Implements อินเตอร์เฟส ตามปกติ ในกรณีนี้กำหนดให้คลาส Person ทำงาน Implements อินเตอร์เฟส IProgramable ส่วนคลาส Programmer ก็จะ Implements อินเตอร์เฟส IDotNETProgramable ส่วนการใช้งานอยู่ใน Form1

โค้ด VB 2005 และ VC# 2005 ที่ 12-13 การทำ Data Hiding ใน Interface สืบทอด Interface

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        p.BasicProgramming()

        Dim pm As New Programmer()
        pm.BasicProgramming1()
        pm.BasicProgramming2()
        pm.VBProgramming()
        pm.CSProgramming()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    Person p = new Person();
    p.BasicProgramming();

    Programmer pm = new Programmer();
    pm.BasicProgramming();
    pm.VBProgramming();
    pm.CSProgramming();
}
```

สำหรับโค้ดของ VB 2005 แม้ว่าจะมีเมธอด BasicProgramming() อยู่ 2 เมธอด แต่ทำงานเหมือนกัน

ในกรณีที่คุณมองว่าข้อบังคับการเขียนโปรแกรมพื้นฐานทั้ง 2 เมธอดทำงานไม่เหมือนกัน ก็ต้องแก้ไขโค้ดของคลาส Programmer ใหม่ แยกออกเป็น 2 กรณีคือ

- **กรณี VB 2005** : กำหนดให้เมธอด BasicProgramming1() และเมธอด BasicProgramming2() ทำงานแตกต่างกัน (แสดงข้อความใน MessageBox ไม่เหมือนกัน)

VB 2005 (Class1.vb)

```
Public Class Programmer
    Implements IDotNETProgramable

    Public Sub BasicProgramming1() Implements IProgramable.BasicProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้น")
    End Sub

    Public Sub BasicProgramming2() Implements IDotNETProgramable.BasicProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้นใน .NET")
    End Sub

    Public Sub VBProgramming() Implements IDotNETProgramable.VBPrograming
        MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VB")
    End Sub

    Public Sub CSProgramming() Implements IDotNETProgramable.CSProgramming
        MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VC#")
    End Sub
End Class
```

- **กรณี VC# 2005** : จะใช้วิธี Explicit Implements Interface เพื่อกำหนดให้เมธอด Basic Programming() ทำงานแตกต่างกัน ขึ้นอยู่กับว่าจะมองคลาส Programmer ในฐานะใด

VC# 2005 (Class1.cs)

```
public class Programmer : IDotNETProgramable
{
    void IProgramable.BasicProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้น");
    }

    void IDotNETProgramable.BasicProgramming()
    {
        MessageBox.Show("Programmer เขียนโปรแกรมเบื้องต้นใน .NET");
    }

    void IDotNETProgramable.VBProgramming()
```

```
        {
            MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VB");
        }

        void IDotNETProgramable.CSProgramming()
        {
            MessageBox.Show("Programmer เขียนโปรแกรมด้วยภาษา VC#");
        }
    }
```

ส่วนการใช้งานคลาส Programmer ใหม่อยู่ใน Form1 แยกออกเป็น 2 กรณีเช่นกันคือ

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim p As New Person()
        p.BasicProgramming()

        Dim pm As New Programmer()
        pm.BasicProgramming1()
        pm.BasicProgramming2()
        pm.VBProgramming()
        pm.CSProgramming()
    End Sub
End Class
```

โค้ดข้างต้นเมธอด BasicProgramming1() กับเมธอด BasicProgramming2() ของออบเจ็กต์ Programmer ที่ชื่อว่า pm ทำงานไม่เหมือนกัน

- **กรณี** VC# 2005 : ขึ้นอยู่กับการมองคลาส Programmer ว่าจะมองในฐานะใดกล่าวคือ ถ้ามองในฐานะความสามารถในการเขียนโปรแกรมทั่วไป (อินเตอร์เฟส IProgramable) เมธอด Basic Programming() ที่ได้คือ เมธอดที่เป็นข้อบังคับของอินเตอร์เฟส IProgramable

VC# 2005 (Form1.cs)

```
        IProgramable Ipm1;
        Ipm1 = new Programmer();
        Ipm1.BasicProgramming();
```

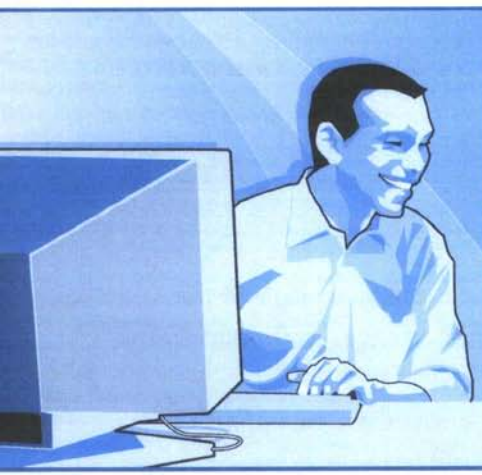
แต่ถ้าคุณมองคลาส Programmer ในฐานะความสามารถในการเขียนโปรแกรมด้วย .NET (อินเตอร์เฟส IDotNETProgramable) ก็จะได้เมธอด BasicProgramming() ที่เป็นข้อบังคับของอินเตอร์เฟส IDotNETProgramable นั่นเอง

VC# 2005

```
IDotNETProgramable ipm2;
ipm2 = new Programmer();
ipm2.BasicProgramming();
ipm2.VBProgramming();
ipm2.CSProgramming();
```

สรุปท้ายบท

เห็นได้ว่าประโยชน์ของอินเตอร์เฟสมีมากมาย ผู้เขียนขอให้คุณผู้อ่านนำแนวความคิดที่ผู้เขียนเลือกมานำเสนอไปประยุกต์ใช้ต่อในแอพพลิเคชันของคุณ



Advance Visual Studio 2005

CHAPTER ▸ 13

อาร์เรย์

บทนำ

อาร์เรย์ (Array) เป็นการเก็บข้อมูลแบบชุด ซึ่งข้อมูลแต่ละตัวมีชื่อเหมือนกัน อ้างอิงสมาชิกแต่ละตัวโดยใช้ลำดับที่เรียกว่า ดัชนี (Index) คือ เนื้อหาที่ผู้เขียนจะนำเสนอในบทนี้

การใช้งานตัวแปรอาร์เรย์

อาร์เรย์ (Array) เป็นคำที่ใช้เรียกตัวแปร หรือออบเจ็กต์ที่คุณสร้างขึ้นมาแล้วมีลักษณะอยู่กันเป็นกลุ่มมีชื่อเดียวกัน เป็นข้อมูลชนิดเดียวกัน สมาชิกที่อยู่ในกลุ่มแต่ละตัวจะถูกอ้างอิงด้วยลำดับของสมาชิก (Index)

โดยถ้าเป็นตัวแปรเมื่อใช้งานในลักษณะอาร์เรย์ก็จะเรียกว่า ตัวแปรอาร์เรย์ แต่ถ้าเป็นออบเจ็กต์จะเรียกว่า อาร์เรย์ออบเจ็กต์ โดยที่ผู้เขียนจะกล่าวถึงตัวแปรอาร์เรย์เพียงอย่างเดียว

เนื่องจากว่าสมาชิกแต่ละตัวที่อยู่ในอาร์เรย์มีชื่อเหมือนกัน ดังนั้น การเรียกใช้งานตัวแปรอาร์เรย์แต่ละตัวจึงอาศัยลำดับของตัวมันเองเป็นตัวระบุว่า เราต้องการใช้งานตัวแปรตัวใด ลำดับดังกล่าวเรียกว่า ดัชนี (Index) ซึ่งเป็นตัวบ่งบอกให้ตัวแปรแต่ละตัวแตกต่างกัน โดยที่สมาชิกลำดับแรกจะมีลำดับที่ 0 เสมอ

การประกาศใช้ตัวแปรอาร์เรย์ใน VB 2005 ใช้เครื่องหมาย () กำกับไว้หลังชื่อตัวแปร ส่วน VC# 2005 ใช้เครื่องหมาย [] กำกับไว้หลังชนิดของข้อมูล เช่น

VB 2005	VC# 2005
Dim salary(10) As Integer	int[] salary = new int[10];

โค้ดข้างต้นเป็นการสร้างตัวแปรที่ชื่อว่า salary ขึ้นมา 10 ตัว เป็นข้อมูลชนิด Integer โดยที่ตัวแปรแต่ละตัวจะมีลำดับอ้างอิงตั้งแต่ 0 ถึง 9 เราเรียกตัวแปรอาร์เรย์ดังกล่าวว่าเป็น Static array หรือตัวแปรอาร์เรย์ที่มีการระบุจำนวนสมาชิกแน่นอน

ในกรณีที่คุณต้องการระบุค่าเริ่มต้นให้กับสมาชิกแต่ละตัวที่อยู่ในอาร์เรย์ สามารถทำได้ดังนี้

VB 2005	VC# 2005
Dim salary() As Double = {5, 4, 3, 2, 1}	int[] salary = {5,4,3,2,1};

จากโค้ดข้างต้นเป็นการสร้างตัวแปรที่ชื่อว่า salary มีสมาชิก 5 ตัว โดยที่

VB 2005	VC# 2005
salary (0) หมายถึง สมาชิกตัวที่ 1 (ลำดับที่ 0) มีค่าเท่ากับ 5 salary (1) หมายถึง สมาชิกตัวที่ 2 (ลำดับที่ 1) มีค่าเท่ากับ 4 salary (2) หมายถึง สมาชิกตัวที่ 3 (ลำดับที่ 2) มีค่าเท่ากับ 3 salary (3) หมายถึง สมาชิกตัวที่ 4 (ลำดับที่ 3) มีค่าเท่ากับ 2 salary (4) หมายถึง สมาชิกตัวที่ 5 (ลำดับที่ 4) มีค่าเท่ากับ 1	salary [0] หมายถึง สมาชิกตัวที่ 1 (ลำดับที่ 0) มีค่าเท่ากับ 5 salary [1] หมายถึง สมาชิกตัวที่ 2 (ลำดับที่ 1) มีค่าเท่ากับ 4 salary [2] หมายถึง สมาชิกตัวที่ 3 (ลำดับที่ 2) มีค่าเท่ากับ 3 salary [3] หมายถึง สมาชิกตัวที่ 4 (ลำดับที่ 3) มีค่าเท่ากับ 2 salary [4] หมายถึง สมาชิกตัวที่ 5 (ลำดับที่ 4) มีค่าเท่ากับ 1

จะเห็นได้ว่าในกรณีนี้ไม่มีการระบุจำนวนสมาชิกแต่อย่างใด แต่ใช้วิธีการระบุค่าแทน คุณก็จะได้จำนวนสมาชิกของตัวแปรอาร์เรย์เท่ากับค่าที่คุณกำหนดโดยอัตโนมัติ โดยใช้เครื่องหมาย , เป็นตัวคั่นระหว่างสมาชิกแต่ละตัว การอ่านค่าสมาชิกแต่ละตัวที่อยู่ในอาร์เรย์ให้ดูตัวอย่างต่อไปนี้

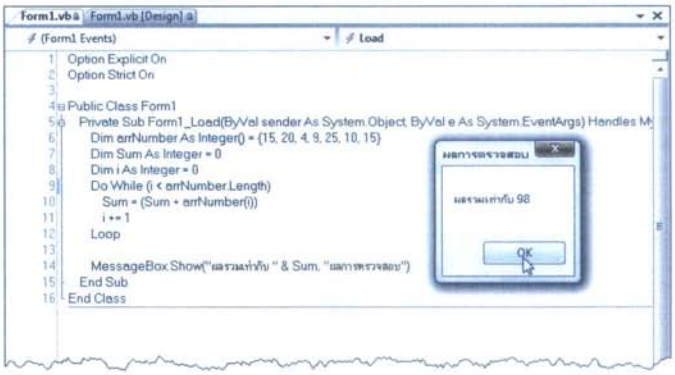
โค้ด VB 2005 และ VC# 2005 ที่ 13-1 พื้นฐานการใช้งานตัวแปรอาร์เรย์

VB 2005 (Form1.vb)	VC# 2005 (Form1.cs)

Public Class Form1
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
Dim arrNumber As Integer() = {15, 20, 4, 9, 25, 10, 15}
Dim Sum As Integer = 0
Dim i As Integer = 0
Do While (i < arrNumber.Length)
Sum = (Sum + arrNumber(i))
i += 1
Loop
	private void Form1_Load(object sender, EventArgs e)
	{
	int[] arrNumber = { 15, 20, 4, 9, 25, 10, 15 };
	int Sum = 0;
	for (int i = 0; i < arrNumber.Length; i++)
	{
	Sum += arrNumber[i];
	}
	MessageBox.Show("ผลรวมเท่ากับ " + Sum, "ผลการตรวจสอบ");
	}

```
        MessageBox.Show("ผลรวมเท่ากับ " & Sum,
"ผลการตรวจสอบ")
    End Sub
End Class
```

รูปที่ 13-1 ผลการรันตัวอย่างที่ 13-1



จากรูปที่ 13-1 จำนวนตัวแปรอาร์เรย์ทั้งหมดอ่านได้จากคุณสมบัติ Length เราจะสั่งให้วนลูปตั้งแต่สมาชิกตัวแรก (ลำดับอ้างอิง 0) จนถึงตัวแปรอาร์เรย์ตัวสุดท้าย เพื่อหาค่ารวมทั้งหมด

การเรียงลำดับตัวแปรอาร์เรย์ด้วยเมธอด Sort() ของคลาส Array

ใน .NET Framework ไมโครซอฟท์สร้างคลาส Array ขึ้นมาเพื่อใช้กับตัวแปรอาร์เรย์ ในกรณีที่ต้องการเรียงลำดับตัวแปรอาร์เรย์ของคุณ ให้เรียกใช้งานเมธอด Sort() ของคลาส Array ดังตัวอย่างโค้ดต่อไปนี้

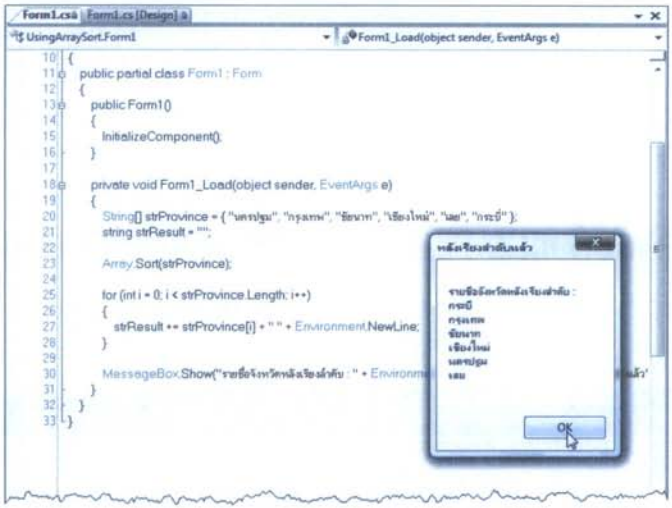
โค้ด VB 2005 ที่ 13-2 การเรียงลำดับตัวแปรอาร์เรย์ด้วยเมธอด Sort() ของคลาส Array (Form1.vb)

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim strProvince As String() = {"นครปฐม", "กรุงเทพ", "ชัยนาท", "เชียงใหม่", "เลย", "กระบี่"}
        Dim strResult As String = ""
        Array.Sort(strProvince)

        Dim i As Integer = 0
        Do While i < strProvince.Length
            strResult &= strProvince(i) & " " & Environment.NewLine
            i += 1
        Loop

        MessageBox.Show("รายชื่อจังหวัดหลังเรียงลำดับ : " & Environment.NewLine & strResult, "หลังเรียงลำดับแล้ว")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 13-2 การเรียงลำดับตัวแปรอาร์เรย์ด้วยเมธอด Sort() ของคลาส Array (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    String[] strProvince = { "นครปฐม", "กรุงเทพ", "ชัยนาท", "เชียงใหม่", "เลย", "กระบี่" };
    string strResult = "";

    Array.Sort(strProvince);

    for (int i = 0; i < strProvince.Length; i++)
    {
        strResult += strProvince[i] + " " + Environment.NewLine;
    }

    MessageBox.Show("รายชื่อจังหวัดหลังเรียงลำดับ : " + Environment.NewLine + strResult, "หลังเรียงลำดับแล้ว");
}
```

รูปที่ 13-2 การเรียงลำดับด้วยเมธอด Sort() ของคลาส Array



จากรูปที่ 13-2 ผู้เขียนสร้างอาร์เรย์ของ String ขึ้นมา 1 ชุด เป็นรายชื่อจังหวัดแบบไม่เรียงลำดับ จากนั้นสั่งให้เรียงลำดับด้วยเมธอด Sort() และสั่งให้วนลูปเพื่อแสดงสมาชิกแต่ละตัวของตัวแปรอาร์เรย์ strProvince หลังจากเรียงลำดับแล้ว

การใช้ลูป For Each... (foreach...) กับตัวแปรอาร์เรย์

สิ่งหนึ่งที่คุณได้มากับการใช้งานตัวแปรอาร์เรย์ก็คือ การวนลูปโดยอาศัยคำสั่ง For Each... (foreach...) เราจะใช้ลูปดังกล่าวเพื่ออ่านค่าสมาชิกแต่ละตัวที่อยู่ในอาร์เรย์ที่สนใจ ตัวอย่างที่ 13-3 การใช้ลูป For Each... (foreach...) กับตัวแปรอาร์เรย์ ให้คุณออกแบบฟอร์ม ดังรูปที่ 13-3

รูปที่ 13-3 ฟอร์มในขณะออกแบบ

โค้ด VB 2005 และ VC# 2005 ที่ 13-3 การใช้ลูป For Each... (foreach...) กับตัวแปรอาร์เรย์

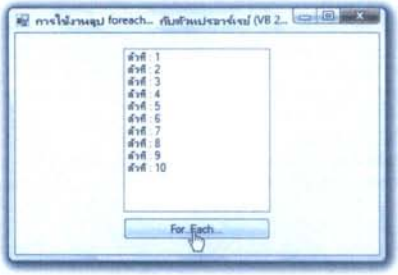
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub cmdForEach_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdForEach.Click
        Dim i() As Integer = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

        For Each j As Integer In i
            listBox1.Items.Add("ตัวที่ : " & j)
        Next
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void cmdForEach_Click(object sender, EventArgs e)
{
    int[] i = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

    foreach (int j in i)
    {
        listBox1.Items.Add("ตัวที่ : " + j);
    }
}
```

รูปที่ 13-4 ผลการรันตัวอย่างที่ 13-3



โค้ดข้างต้นผู้เขียนสร้างตัวแปรอาร์เรย์ที่ชื่อว่า i มี Type เป็น Integer (int) ประกอบด้วยสมาชิก 10 ตัวมีค่าตั้งแต่ 1 ถึง 10 จากนั้นสร้างตัวแปรอีก 1 ตัวชื่อว่า j มี Type เป็น Integer เช่นกัน ทำหน้าที่เก็บค่าของสมาชิกแต่ละตัวที่อยู่ในตัวแปรอาร์เรย์ i

ต่อมาสั่งให้วนลูปเพื่ออ่านค่าสมาชิกตั้งแต่ตัวแรกจนถึงตัวสุดท้ายด้วยลูป For Each... ในแต่ละรอบของการวนลูป จะนำค่าของสมาชิกแต่ละตัวที่อยู่ในตัวแปรอาร์เรย์ i ที่อ่านได้ไปเพิ่มในคอนโทรล ListBox1

การทำงานกับตัวแปรอาร์เรย์ที่น่าสนใจ

เพราะความที่ตัวแปรอาร์เรย์คือ ตัวแปรที่มีข้อมูลชนิดเดียวกัน มีชื่อเดียวกัน แตกต่างกันก็แต่เพียงลำดับของตัวแปรแต่ละตัวนั่นเอง เราจะลองดูการทำงานขั้นต้นที่น่าสนใจเมื่อมีการใช้งานตัวแปรอาร์เรย์ ให้ดูตัวอย่างที่ 13-4 การทำงานกับตัวแปรอาร์เรย์ที่น่าสนใจ

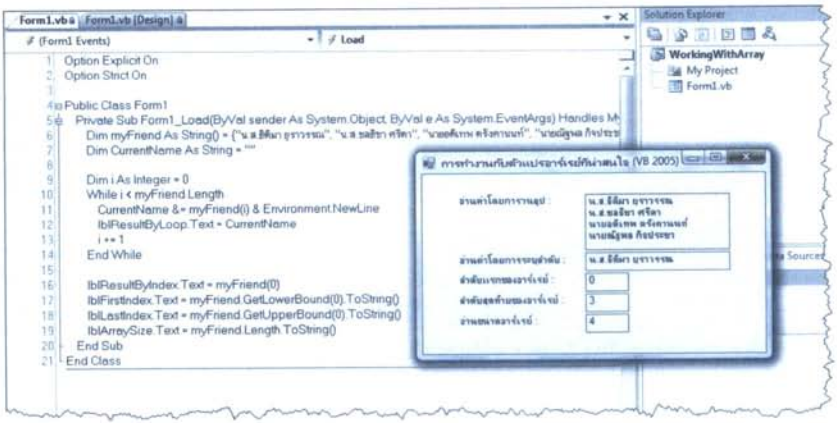
โค้ด VB 2005 ที่ 13-4 การทำงานกับตัวแปรอาร์เรย์ที่น่าสนใจ (Form1.vb)

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim myFriend As String() = {"น.ส.ธิติมา ยุราวรรณ", "น.ส.ชลธิชา ศรีดา", "นายอดิเทพ ตรังคานนท์", "นายณัฐพล กิจประชา"}
        Dim CurrentName As String = ""

        Dim i As Integer = 0
        While i < myFriend.Length
            CurrentName &= myFriend(i) & Environment.NewLine
            lblResultByLoop.Text = CurrentName
            i += 1
        End While

        lblResultByIndex.Text = myFriend(0)
        lblFirstIndex.Text = myFriend.GetLowerBound(0).ToString()
        lblLastIndex.Text = myFriend.GetUpperBound(0).ToString()
        lblArraySize.Text = myFriend.Length.ToString()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 13-4 การทำงานกับตัวแปรอาร์เรย์ที่น่าสนใจ (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        string[] myFriend ={ "น.ส.ธิติมา ยุราวรรณ", "น.ส.ชลธิชา ศรีดา", "นายอดิเทพ ตรังคานนท์", "นายณัฐพล กิจประชา" };
        string CurrentName = "";

        for (int i = 0; i < myFriend.Length; i++)
        {
            CurrentName += myFriend[i] + Environment.NewLine;
            lblResultByLoop.Text = CurrentName;
        }

        lblResultByIndex.Text = myFriend[0];
        lblFirstIndex.Text = myFriend.GetLowerBound(0).ToString();
        lblLastIndex.Text = myFriend.GetUpperBound(0).ToString();
        lblArraySize.Text = myFriend.Length.ToString();
    }
```

รูปที่ 13-5 ผลการรันตัวอย่างที่ 13-4



ผู้เขียนสร้างตัวแปรอาร์เรย์ชนิดข้อมูล String ขึ้นมา 1 ชุด โดยที่

- คอนโทรล lblResultByLoop แสดงการอ่านค่าออกมาจากตัวแปรอาร์เรย์แต่ละตัว
- คอนโทรล lblResultByIndex แสดงการอ่านค่าของตัวแปรอาร์เรย์ออกมา โดยการระบุลำดับสมาชิก (index)
- คอนโทรล lblFirstIndex แสดงการอ่านลำดับสมาชิกของตัวแปรอาร์เรย์ตัวแรก โดยอาศัยเมธอด GetLowerBount()
- คอนโทรล lblLastIndex แสดงการอ่านลำดับสมาชิกของตัวแปรอาร์เรย์ตัวสุดท้าย โดยอาศัยเมธอด GetUpperBound()
- คอนโทรล lblArraySize แสดงการอ่านจำนวนสมาชิกที่อยู่ในอาร์เรย์นี้ทั้งหมด

การสร้างคลาสที่ทำงานเกี่ยวข้องกับอาร์เรย์

คลาสที่ทำงานเกี่ยวข้องกับอาร์เรย์ชื่อว่า ArrayTools อยู่ในเนมสเปซ

GAF

คลาส ArrayTools ประกอบด้วย 5 เมธอดแบบ Shared (static) ดังนี้

เมธอด	หน้าที่
เมธอด PrintData()	พิมพ์สมาชิกทั้งหมดที่อยู่ในอาร์เรย์
เมธอด SearchMember()	ค้นหาสมาชิกที่อยู่ในอาร์เรย์ คืนค่าเป็นตำแหน่งของสมาชิกที่พบ แต่ถ้าไม่พบจะคืนค่า -1
เมธอด SummaryArray()	หาผลรวมทั้งหมดที่อยู่ในอาร์เรย์
เมธอด Max()	หาค่าสูงสุดที่อยู่ในอาร์เรย์
เมธอด Min()	หาค่าต่ำสุดที่อยู่ในอาร์เรย์

ข้อจำกัดของคลาส ArrayTools คือ สามารถใช้ได้กับข้อมูลชนิด Double (double) เท่านั้น ดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 13-5 การสร้างคลาสที่ทำงานเกี่ยวข้องกับอาร์เรย์ (GAF.ArrayTools.vb)

```
Option Explicit On
Option Strict On
Imports System.Text

Namespace GAF
    Public Class ArrayTools
        Public Shared Function PrintData(ByVal ArrToPrint() As Double) As String
            Dim _i As Integer = 0
            Dim _sw As New StringBuilder()
            _sw.Remove(0, _sw.Length)

            For _i = 0 To ArrToPrint.Length - 1
                _sw.Append(ArrToPrint(_i) & " ")
            Next

            Dim _Data As String = ""
            _Data = _sw.ToString()
            Return _Data.Trim()
        End Function

        Public Shared Function SearchMember(ByVal ArrToSearch() As Double, ByVal ItemToSearch As Double) As Integer
            Dim _Success As Integer = 0
            For _Success = 0 To ArrToSearch.Length - 1
                If ArrToSearch(_Success) = ItemToSearch Then
                    Return _Success
                        Exit Function
                    End If
            Next
            Return -1
        End Function

        Public Shared Function SummaryArray(ByVal ArrToSummary() As Double) As Double
            Dim _i As Integer = 0
            Dim _Total As Double = 0.0
            For _i = 0 To ArrToSummary.Length - 1
                _Total += ArrToSummary(_i)
            Next
            Return _Total
        End Function

        Public Shared Function Max(ByVal ArrToSearch() As Double) As Double
            Dim _i As Integer = 0
            Dim _Max As Double = 0.0
            For _i = 0 To ArrToSearch.Length - 1
```

```
            If ArrToSearch(_i) > _Max Then
                _Max = ArrToSearch(_i)
            End If
        Next
        Return _Max
    End Function

    Public Shared Function Min(ByVal ArrToSearch() As Double) As Double
        Dim _i As Integer = 0
        Dim _Min As Double = 0.0
        For _i = 0 To ArrToSearch.Length - 1
            If ArrToSearch(_i) < _Min Then
                _Min = ArrToSearch(_i)
            End If
        Next
        Return _Min
    End Function
End Class
End Namespace
```

โค้ด VC# 2005 ที่ 13-5 การสร้างคลาสที่ทำงานเกี่ยวข้องกับอาร์เรย์ (GAF.ArrayTools.cs)

```
namespace GAF
{
    public class ArrayTools
    {
        public static string PrintData(double[] ArrToPrint)
        {
            int _i = 0;
            StringBuilder _sw = new StringBuilder();
            _sw.Remove(0, _sw.Length);

            for (_i = 0; _i <= ArrToPrint.Length-1; _i++)
            {
                _sw.Append(ArrToPrint[_i] + " ");
            }

            string _Data = "";
            _Data = _sw.ToString();
            return _Data.Trim();
        }

        public static int SearchMember(double[] ArrToSearch, double ItemToSearch)
        {
            int _Success = 0;
            for (_Success = 0; _Success <= ArrToSearch.Length-1; _Success++)
            {
                if (ArrToSearch[_Success] == ItemToSearch)
                {
                    return _Success;
                }
```

```
            }
            return -1;
        }

        public static double SummaryArray(double[] ArrToSummary)
        {
            int _i = 0;
            double _Total = 0.0;
            for (_i = 0; _i <= ArrToSummary.Length-1; _i++)
            {
                _Total += ArrToSummary[_i];
            }
            return _Total;
        }

        public static double Max(double[] ArrToSearch)
        {
            int _i = 0;
            double _Max = 0.0;
            for (_i = 0; _i <= ArrToSearch.Length-1; _i++)
            {
                if (ArrToSearch[_i] > _Max)
                {
                    _Max = ArrToSearch[_i];
                }
            }
            return _Max;
        }

        public static double Min(double[] ArrToSearch)
        {
            int _i = 0;
            double _Min = 0.0;
            for (_i = 0; _i <= ArrToSearch.Length-1; _i++)
            {
                if (ArrToSearch[_i] < _Min)
                {
                    _Min = ArrToSearch[_i];
                }
            }
            return _Min;
        }
    }
}
```

หลักการทำงานทั้ง 5 เมธอดคล้ายๆ กันกล่าวคือ สั่งให้วนลูปตั้งแต่สมาชิกตัวที่ 1 (ลำดับอ้างอิง 0) ไปจนถึงสมาชิกลำดับสุดท้าย (คุณสมบัติ Length-1) ในแต่ละรอบของการวนลูปจะทำอะไรก็แล้วแต่จุดประสงค์ของเมธอดนั้นๆ ส่วนการใช้งานคลาส ArrayTools อยู่ใน Form1 ดังโค้ดต่อไปนี้

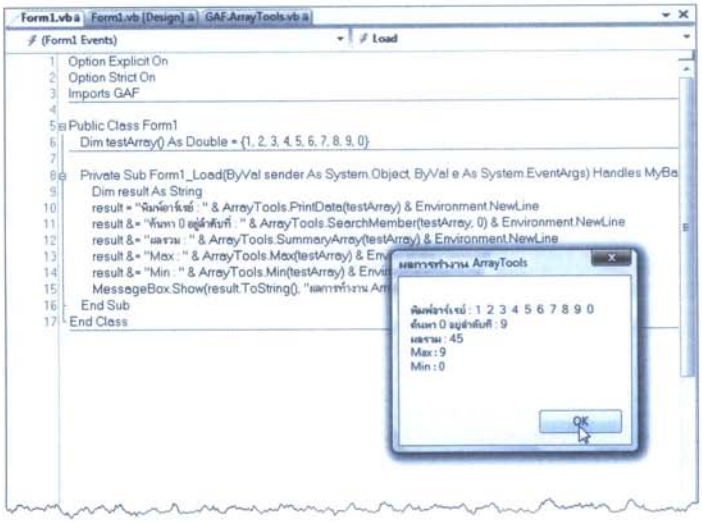
โค้ด VB 2005 ที่ 13-5 การสร้างคลาสที่ทำงานเกี่ยวข้องกับอาร์เรย์ (Form1.vb)

```
Dim testArray() As Double = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0}

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim result As String
    result = "พิมพ์อาร์เรย์ : " & ArrayTools.PrintData(testArray) & Environment.NewLine
    result &= "ค้นหา 0 อยู่ลำดับที่ : " & ArrayTools.SearchMember(testArray, 0) & Environment.NewLine
    result &= "ผลรวม : " & ArrayTools.SummaryArray(testArray) & Environment.NewLine
    result &= "Max : " & ArrayTools.Max(testArray) & Environment.NewLine
    result &= "Min : " & ArrayTools.Min(testArray) & Environment.NewLine
    MessageBox.Show(result.ToString(), "ผลการทำงาน ArrayTools")
End Sub
```

โค้ด VC# 2005 ที่ 13-5 การสร้างคลาสที่ทำงานเกี่ยวข้องกับอาร์เรย์ (Form1.cs)

```
double[] testArray = new double[] {1,2,3,4,5,6,7,8,9,0};

private void Form1_Load(object sender, EventArgs e)
{
    string result;
    result = "พิมพ์อาร์เรย์ : " + ArrayTools.PrintData(testArray) + Environment.NewLine;
    result += "ค้นหา 0 อยู่ลำดับที่ : " + ArrayTools.SearchMember(testArray, 0) + Environment.NewLine;
    result += "ผลรวม : " + ArrayTools.SummaryArray(testArray) + Environment.NewLine;
    result += "Max : " + ArrayTools.Max(testArray) + Environment.NewLine;
    result += "Min : " + ArrayTools.Min(testArray) + Environment.NewLine;
    MessageBox.Show(result.ToString(), "ผลการทำงาน ArrayTools");
}
```

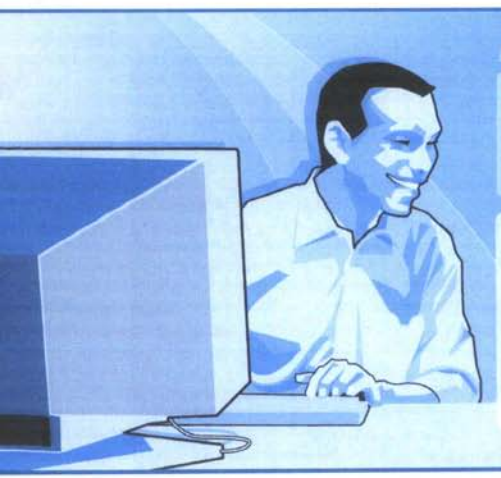
รูปที่ 13-6 ผลการรันตัวอย่างที่ 13-5



จากรูปที่ 13-6 การค้นหาเลข 0 ในอาร์เรย์ testArray ของเมธอด SearchMember() ผลที่ได้คือ 9 หมายความว่าเลข 0 เป็นสมาชิกตัวที่ 10 (ลำดับอ้างอิง 9) ของอาร์เรย์ testArray นั่นเอง

สรุปท้ายบท

ประเด็นที่ผู้เขียนเลือกมานำเสนอ อาจจะไม่ได้ครอบคลุมเนื้อหาของอาร์เรย์ทั้งหมด ขอให้คุณผู้อ่านศึกษาเพิ่มเติมเพราะจะมีรายละเอียดอื่นๆ อีกพอสมควรที่ยังไม่ได้กล่าวถึง



Advance Visual Studio 2005

CHAPTER ▸ 14

ตัวเลขใน .NET

บทนำ

เนื้อหาในบทนี้เป็นการกล่าวถึงโลกของตัวเลขใน .NET Framework ว่ามีอะไรน่าสนใจบ้าง

การทดสอบเครื่องหมายคูณและหาร

มีประเด็นที่น่าสนใจอยู่ 1 อย่าง สำหรับการคูณและหารตัวเลขใน .NET Framework นั่นคือ การหารตัวเลขช้ากว่าการคูณ ให้คุณดูผลการทดสอบของตัวอย่างที่ 14-1 เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 14-1 การทดสอบเครื่องหมายคูณและหาร

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Dim i As Integer
    Dim j As Double
    Dim sw As New Stopwatch()

    Private Sub cmdMultiply_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdMultiply.Click
        sw.Reset()
        sw.Start()

        For i = 1 To 99999999
            j = i * 0.5
        Next
```

VC# 2005 (Form1.cs)

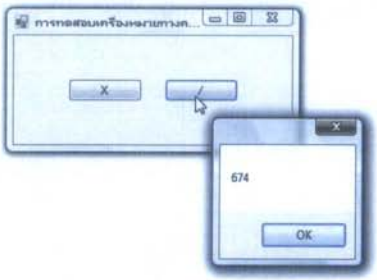
```
int i;
double j;
Stopwatch sw = new Stopwatch();

private void cmdMultiply_Click(object sender, EventArgs e)
{
    sw.Reset();
    sw.Start();

    for (i = 1; i <= 99999999; i++)
    {
        j = i * 0.5;
    }

    sw.Stop();
    MessageBox.Show( sw.ElapsedMilliseconds.ToString());
}
```

```
        sw.Stop()
        MessageBox.Show( sw.ElapsedMilliseconds.ToString())
    End Sub

    Private Sub cmdDivide_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cmdDivide.Click
        sw.Reset()
        sw.Start()

        For i = 1 To 99999999
            j = i / 2
        Next

        sw.Stop()
        MessageBox.Show( sw.ElapsedMilliseconds.ToString())
    End Sub
End Class
```

```
private void cmdDivide_Click(object sender, EventArgs e)
{
    sw.Reset();
    sw.Start();

    for (i = 1; i <= 99999999; i++)
    {
        j = i / 2;
    }

    sw.Stop();
    MessageBox.Show( sw.ElapsedMilliseconds.ToString());
}
```

รูปที่ 14-1
เวลาที่ใช้ในการคูณ

รูปที่ 14-2
เวลาที่ใช้ในการหาร



ผู้เขียนสั่งให้วนลูปประมาณ 100 ล้านรอบ เพื่อหาค่าครึ่งหนึ่งของตัวแปรวนลูป i แยกเป็น 2 กรณี

- ถ้าเป็นการคูณจะใช้วิธีคูณด้วย 0.5
- แต่ถ้าเป็นการหารจะใช้วิธีหารด้วย 2

ผลทดสอบที่ได้คือ การคูณใช้เวลาประมาณ 600 ต้นๆ ส่วนการหารจะใช้เวลาประมาณ 600 ปลายๆ เลยไปถึง 700 กว่า จะเห็นได้ว่าการหารช้ากว่าการคูณเป็นอีก 1 ช่องทางที่ช่วยให้โค้ดของคุณทำงานเร็วขึ้น

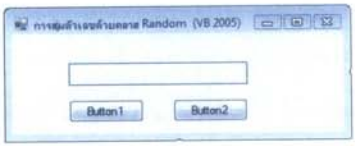
การใช้งานออบเจ็กต์ Random

การสุ่มตัวเลขด้วยออบเจ็กต์ Random สามารถทำได้ 2 วิธีคือ

- ระบุค่าสูงสุด ค่าที่สุ่มได้จะไม่เกินค่าสูงสุด โดยที่ไม่รวมค่าสูงสุด
- ระบุค่าต่ำสุดและสูงสุด ค่าที่ได้จะอยู่ระหว่างค่าต่ำสุดและค่าสูงสุด โดยที่รวมค่าต่ำสุดแต่ไม่รวมค่าสูงสุด

ตัวอย่างที่ 14-2 การสุ่มค่าโดยอาศัยออบเจ็กต์ Random ให้คุณออกแบบฟอร์ม ดังรูปที่ 14-3

รูปที่ 14-3 ฟอร์มในขณะออกแบบ



จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 14-2 การสุ่มค่าโดยอาศัยออบเจ็กต์ Random

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim rd As New Random()
        lblResult.Text = rd.Next(5).ToString()
        'lblResult.Text = rd.NextDouble().ToString()
    End Sub

    Private Sub Button2_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles Button2.Click
        Dim rd As New Random()
        lblResult.Text = rd.Next(1, 3).ToString()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void Button1_Click(object sender, EventArgs e)
{
    Random rd = new Random();
    lblResult.Text = rd.Next(5).ToString();
    //lblResult.Text = rd.NextDouble().ToString();
}

private void Button2_Click(object sender, EventArgs e)
{
    Random rd = new Random();
    lblResult.Text = rd.Next(1, 3).ToString();
}
```

การสุ่มตัวเลขแบบที่ 1 (เหตุการณ์ Button1_Click()) เป็นการสุ่มเลขจำนวนเต็มโดยการกำหนดค่าสูงสุดไว้ ส่งผลให้การสุ่มค่าที่ได้จะไม่เกินค่าสูงสุด โดยที่จะไม่รวมค่าสูงสุด กรณีนี้กำหนดค่าสูงสุดไว้ที่ 5 ค่าที่สุ่มได้คือ 0-4 นั่นเอง

VB 2005

```
Private Sub Button1_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles Button1.Click
    Dim rd As New Random()
    lblResult.Text = rd.Next(5).ToString()
    'lblResult.Text = rd.NextDouble().ToString()
End Sub
```

VC# 2005

```
private void Button1_Click(object sender, EventArgs e)
{
    Random rd = new Random();
    lblResult.Text = rd.Next(5).ToString();
    //lblResult.Text = rd.NextDouble().ToString();
}
```

การสุ่มตัวเลขแบบที่ 2 (เหตุการณ์ Button2_Click()) เป็นการระบุค่าต่ำสุดและค่าสูงสุด ส่งผลให้ตัวเลขที่สุ่มขึ้นมาจะมีค่าไม่เกินค่าสูงสุด แต่จะรวมค่าต่ำสุด ในกรณีนี้กำหนดค่าต่ำสุดคือ 1 ส่วนค่าสูงสุดคือ 3 ค่าที่สุ่มออกมาได้คือ 1 และ 2

VB 2005

```
Private Sub Button2_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles Button2.Click
    Dim rd As New Random()
    lblResult.Text = rd.Next(1, 3).ToString()
End Sub
```

VC# 2005

```
private void Button2_Click(object sender, EventArgs e)
{
    Random rd = new Random();
    lblResult.Text = rd.Next(1, 3).ToString();
}
```

NOTE

ในกรณีที่คุณต้องการสุ่มค่าทศนิยม Double ให้คุณเรียกใช้เมธอด NextDouble() ดังโค้ดต่อไปนี้

VB 2005

```
lblResult.Text = rd.NextDouble().ToString()
```

VC# 2005

```
lblResult.Text = rd.NextDouble().ToString();
```

การจัดรูปแบบตัวเลข

การจัดรูปแบบตัวเลขเป็นอีกกรณีหนึ่งที่มีความสำคัญเช่นกัน โดยมีอักษรควบคุมที่ทำหน้าที่จัดรูปแบบตัวเลขที่น่าสนใจ มีดังต่อไปนี้

รูปแบบ	คำอธิบาย
C	จัดรูปแบบตัวเลขทางการเงิน เช่น 12345.99 คืนค่า ฿12,345.99
E	จัดรูปแบบตัวเลขยกกำลัง เช่น 12345.99 คืนค่า 1.234599E+004
F	จัดตัวเลขทศนิยม เช่น 12345 คืนค่า 12345.00
G	จัดรูปแบบตัวเลขยกกำลัง เช่น 12345 คืนค่า 12345
N	จัดรูปแบบตัวเลขให้มีทศนิยม 2 ตำแหน่ง เช่น 12345.6 คืนค่า 12,345.60
P	จัดรูปแบบตัวเลขเปอร์เซ็นต์ เช่น 1 คืนค่า 100.00%
C3	จัดรูปแบบตัวเลขทางการเงิน มีทศนิยม 3 ตำแหน่ง เช่น 100 คืนค่า ฿100.000
E3	จัดรูปแบบตัวเลขยกกำลังแบบมีเลขทศนิยม 3 ตำแหน่งปัดเศษขึ้น เช่น 12345 คืนค่า 1.235E+004
F3	จัดรูปแบบตัวเลขทศนิยม 3 ตำแหน่ง เช่น 12345 คืนค่า 12345.000
G4	จัดรูปแบบตัวเลขยกกำลัง มีทศนิยม 4 ตำแหน่ง เช่น 12345 คืนค่า 1.235E+04
N4	จัดรูปแบบตัวเลขทศนิยม 4 ตำแหน่ง เช่น 12345.99 คืนค่า 12,345.9900
P4	จัดรูปแบบตัวเลขเปอร์เซ็นต์ มีทศนิยม 4 ตำแหน่ง เช่น 1 คืนค่า 100.0000%
0.00	จัดรูปแบบตัวเลขทศนิยม 2 ตำแหน่ง เช่น 1 คืนค่า 1.00
00.00	จัดรูปแบบตัวเลขทศนิยม 2 ตำแหน่งมี 0 เติมด้านหน้า เช่น 1 คืนค่า 01.00
#0.00	จัดรูปแบบทศนิยม 2 ตำแหน่ง ไม่มี 0 เติมด้านหน้า เช่น 99 คืนค่า 99.00
#,##0	เติมเครื่องหมาย , ทุกๆ 3 ตำแหน่ง เช่น 1234567 คืนค่า 1,234,567
#,##0.00	เติมเครื่องหมาย , ทุกๆ 3 ตำแหน่ง และมีทศนิยม 2 ตำแหน่ง เช่น 1234567 คืนค่า 1,234,567.00
#%	จัดรูปแบบตัวเลขเปอร์เซ็นต์ เช่น 1 คืนค่า 100%
#.##0%	จัดรูปแบบตัวเลขเปอร์เซ็นต์ มีทศนิยม 3 ตำแหน่ง เช่น 1 คืนค่า 100.000%
#,##0.00%	จัดรูปแบบตัวเลขเปอร์เซ็นต์ มีทศนิยม 2 ตำแหน่ง เช่น 1 คืนค่า 100.00%
฿#,##0.00	เติมอักษรการเงินและเครื่องหมาย , ทุกๆ 3 ตำแหน่ง มีทศนิยม 2 ตำแหน่ง เช่น 9999 คืนค่า ฿9,999.00

ตัวอย่างโปรเจ็กต์ที่ 14-3 การจัดรูปแบบตัวเลข ให้คุณออกแบบฟอร์ม ดังรูปที่ 14-4

รูปที่ 14-4
แสดงฟอร์ม
ในขณะออกแบบ

จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 ที่ 14-3 การจัดรูปแบบตัวเลข (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        cboStyle.Items.Add("C")
        cboStyle.Items.Add("E")
        cboStyle.Items.Add("F")
        cboStyle.Items.Add("G")
        cboStyle.Items.Add("N")
        cboStyle.Items.Add("P")
        cboStyle.Items.Add("C3")
        cboStyle.Items.Add("E3")
        cboStyle.Items.Add("F3")
        cboStyle.Items.Add("G4")
        cboStyle.Items.Add("N4")
        cboStyle.Items.Add("P4")
        cboStyle.Items.Add("0.00")
        cboStyle.Items.Add("00.00")
        cboStyle.Items.Add("#0.00")
        cboStyle.Items.Add("#,##0")
        cboStyle.Items.Add("#,##0.00")
        cboStyle.Items.Add("#%")
        cboStyle.Items.Add("#.##0%")
        cboStyle.Items.Add("#,##0.00%")
        cboStyle.Items.Add("฿#,##0.00")
        cboStyle.SelectedIndex = 0

        txtData.MaxLength = 10
    End Sub

    Private Sub cmdOK_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdOK.Click
        Dim TestNumber As Double = 0.0
        Dim StyleNumber As String = ""

        If (txtData.Text.Trim() <> "") Then
```

```
        Try
            TestNumber = Double.Parse(txtData.Text)
            StyleNumber = cboStyle.Text
            lblDisplay.Text = TestNumber.ToString(StyleNumber)
        Catch
            MessageBox.Show("กรุณาป้อนข้อมูลที่เป็นตัวเลข")
        End Try
    End If
  End Sub
End Class
```

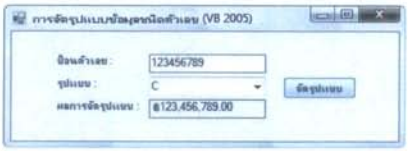
โค้ด VC# 2005 ที่ 14-3 การจัดรูปแบบตัวเลข (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        cboStyle.Items.Add("C");
        cboStyle.Items.Add("E");
        cboStyle.Items.Add("F");
        cboStyle.Items.Add("G");
        cboStyle.Items.Add("N");
        cboStyle.Items.Add("P");
        cboStyle.Items.Add("C3");
        cboStyle.Items.Add("E3");
        cboStyle.Items.Add("F3");
        cboStyle.Items.Add("G4");
        cboStyle.Items.Add("N4");
        cboStyle.Items.Add("P4");
        cboStyle.Items.Add("0.00");
        cboStyle.Items.Add("00.00");
        cboStyle.Items.Add("#0.00");
        cboStyle.Items.Add("#,##0");
        cboStyle.Items.Add("#,##0.00");
        cboStyle.Items.Add("#%");
        cboStyle.Items.Add("#,##0%");
        cboStyle.Items.Add("#,##0.00%");
        cboStyle.Items.Add("฿#,##0.00");
        cboStyle.SelectedIndex = 0;

        txtData.MaxLength = 10;
    }

    private void cmdOK_Click(object sender, EventArgs e)
    {
        double TestNumber = 0.0;
        string StyleNumber = "";

        if (txtData.Text.Trim()!="")
        {
```

```
        try
        {
            TestNumber = double.Parse(txtData.Text);
            StyleNumber = cboStyle.Text;
            lblDisplay.Text = TestNumber.ToString(StyleNumber);
        }
        catch
        {
            MessageBox.Show("กรุณาป้อนข้อมูลที่เป็นตัวเลข");
        }
    }
}
```

ให้คุณทดสอบป้อนตัวเลขจำนวนจริง และเลือกรูปแบบที่ต้องการจัดแล้วคลิกปุ่ม จัดรูปแบบ ผลการทำงานแสดงดังรูปที่ 14-5

รูปที่ 14-5 ผลการจัดรูปแบบตัวเลข



ในเหตุการณ์ Form_Load() จะเพิ่มรูปแบบไว้ในคอนโทรล cboStyle ก่อน ก็จะกำหนดให้รายการแรกปรากฏขึ้นมา โดยการกำหนดที่คุณสมบัติ SelectedIndex = 0 และจำกัดจำนวนตัวเลขที่จะป้อนไม่เกิน 10 ตัวอักษร

VB 2005

```
cboStyle.Items.Add("C")
cboStyle.Items.Add("E")
cboStyle.Items.Add("F")
cboStyle.Items.Add("G")
cboStyle.Items.Add("N")
cboStyle.Items.Add("P")
cboStyle.Items.Add("C3")
cboStyle.Items.Add("E3")
cboStyle.Items.Add("F3")
cboStyle.Items.Add("G4")
cboStyle.Items.Add("N4")
cboStyle.Items.Add("P4")
cboStyle.Items.Add("0.00")
cboStyle.Items.Add("00.00")
cboStyle.Items.Add("#0.00")
cboStyle.Items.Add("#,##0")
cboStyle.Items.Add("#,##0.00")
cboStyle.Items.Add("#%")
```

VC# 2005

```
cboStyle.Items.Add("C");
cboStyle.Items.Add("E");
cboStyle.Items.Add("F");
cboStyle.Items.Add("G");
cboStyle.Items.Add("N");
cboStyle.Items.Add("P");
cboStyle.Items.Add("C3");
cboStyle.Items.Add("E3");
cboStyle.Items.Add("F3");
cboStyle.Items.Add("G4");
cboStyle.Items.Add("N4");
cboStyle.Items.Add("P4");
cboStyle.Items.Add("0.00");
cboStyle.Items.Add("00.00");
cboStyle.Items.Add("#0.00");
cboStyle.Items.Add("#,##0");
cboStyle.Items.Add("#,##0.00");
cboStyle.Items.Add("#%");
```

```
cboStyle.Items.Add("#,##0%")
cboStyle.Items.Add("#,##0.00%")
cboStyle.Items.Add("฿#,##0.00")
cboStyle.SelectedIndex = 0

txtData.MaxLength = 10
```

```
cboStyle.Items.Add("#,##0%");
cboStyle.Items.Add("#,##0.00%");
cboStyle.Items.Add("฿#,##0.00");
cboStyle.SelectedIndex = 0;

txtData.MaxLength = 10;
```

ในเหตุการณ์ cmdOK_Click() เริ่มต้นจะมีการตรวจสอบก่อนว่า ผู้ใช้ต้องป้อนตัวอักษรเข้ามา ก็จะมีการเรียกใช้งาน Double Type ร่วมกับเมธอด Parse() ทำหน้าที่แปลงตัวเลขที่ผู้ใช้ป้อนเข้ามาเป็นข้อมูลชนิด Double เก็บไว้ที่ตัวแปร strTmp ก็จะมีการดัก Error ไว้ด้วยบล็อก Try...Catch... ป้องกันในกรณีที่ป้อนข้อความเข้ามา

VB 2005

```
If (txtData.Text.Trim() <> "") Then
    Try
        TestNumber = Double.Parse(txtData.Text)
        StyleNumber = cboStyle.Text
        lblDisplay.Text = TestNumber.ToString(StyleNumber)
    Catch
        MessageBox.Show("กรุณาป้อนข้อมูลที่เป็นตัวเลข")
    End Try
End If
```

VC# 2005

```
if (txtData.Text.Trim()!="")
{
    try
    {
        TestNumber = double.Parse(txtData.Text);
        StyleNumber = cboStyle.Text;
        lblDisplay.Text = TestNumber.ToString(StyleNumber);
    }
    catch
    {
        MessageBox.Show("กรุณาป้อนข้อมูลที่เป็นตัวเลข");
    }
}
```

ท้ายที่สุดให้อ่านรูปแบบที่ผู้ใช้เลือกในคอนโทรล cboStyle เก็บไว้ในตัวแปร StyleNumber และจัดรูปแบบโดยอาศัยเมธอด ToString()

การสร้างคลาสที่ทำงานด้านคณิตศาสตร์

ผู้ช่วยเหลือด้านคณิตศาสตร์ชื่อว่า MathTools อยู่ภายใต้เนมสเปซ

GAF

คลาส MathTools ประกอบด้วย 11 เมธอด เป็นเมธอดแบบ Shared (static) ทั้งหมด ดังนี้

เมธอด	หน้าที่
เมธอด Factorial()	หาค่า Factorial
เมธอด SqrtN()	หาค่ารากที่ n
เมธอด AlwayRoundUp()	ปัดเศษขึ้นเสมอ
เมธอด AlwayRoundUpStang()	ปัดเศษขึ้นเสมอ ให้ตรงกับหน่วยเหรียญสตางค์
เมธอด AlwayRoundDownStang()	ปัดเศษลงเสมอ ให้ตรงกับหน่วยเหรียญสตางค์
เมธอด AlwayRoundDown()	ปัดเศษลงเสมอ
เมธอด ToKByteUnit()	แปลงเป็นหน่วยกิโลไบต์ (KB)
เมธอด ToMByteUnit()	แปลงเป็นหน่วยเมกะไบต์ (MB)
เมธอด ToGByteUnit()	แปลงเป็นหน่วยกิกะไบต์ (GB)
เมธอด ToTByteUnit()	แปลงเป็นหน่วยเทร่าไบต์ (TB)
เมธอด CurrencyOnly()	กำหนดให้คอนโทรล TextBox รับได้แต่ตัวเลขด้านการเงิน

การหาค่ารากที่ n

โดยปกติแล้วเมธอด Sqrt() ของคลาส Math ที่มากับ .NET Framework หาค่าได้แต่รากที่ 2 เราสามารถใช้กฎเกณฑ์ทางด้านคณิตศาสตร์มาช่วยหาค่ารากอื่นๆ กล่าวคือ

- รากที่ 2 ของ 5 หมายถึง 5 ยกกำลัง $^1/_2$
- รากที่ 3 ของ 5 หมายถึง 5 ยกกำลัง $^1/_3$

โค้ด VB 2005 และ VC# 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการหาค่ารากที่ n

VB 2005 (GAF.MathTools.vb)

```
Public Shared Function SqrtN(ByVal d As Double,
ByVal n As Double) As Double
    Dim result As Double = 0.0
    result = Math.Pow(d, 1 / n)

    Return result
End Function
```

VC# 2005 (GAF.MathTools.cs)

```
public static double SqrtN(double d, double n)
{
    double result = 0.0;
    result = Math.Pow(d, 1 / n);

    return result;
}
```

เมธอด SqrtN() ทำหน้าที่หาค่ารากที่ n ต้องการพารามิเตอร์ 2 ตัวคือ d หมายถึง ตัวเลขที่ต้องการหาค่าราก ส่วนพารามิเตอร์ n หมายถึง รากก็จะยกกำลังโดยอาศัยเมธอด Pow() ของคลาส Math ที่มากับ .NET Framework เข้ามาช่วย

การปัดเศษขึ้นหรือลงเสมอ

การปัดเศษขึ้นเสมออาศัยเมธอด Ceiling() ส่วนการปัดเศษลงเสมออาศัยเมธอด Floor() ของคลาส Math ที่มากับ .NET Framework

โค้ด VB 2005 และ VC# 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการปัดเศษขึ้นหรือลงเสมอ

VB 2005 (GAF.MathTools.vb)

```
    Public Shared Function AlwayRoundUp(ByVal a As
Double) As Double
        Return Math.Ceiling(a)
End Function

Public Shared Function AlwayRoundDown(ByVal
a As Double) As Double
        Return Math.Floor(a)
End Function
```

VC# 2005 (GAF.MathTools.cs)

```
    public static double AlwayRoundUp(double a)
    {
        return Math.Ceiling(a);
    }

    public static double AlwayRoundDown(double a)
    {
        return Math.Floor(a);
    }
```

การปัดเศษขึ้นหรือลงเสมอให้ตรงกับเหรียญสตางค์

ในบางกรณีการปัดเศษขึ้นหรือลงให้เป็นตัวเลขจำนวนเต็มเพียงอย่างเดียว อาจจะเป็นสเกลที่กว้างเกินไป เช่น การขายสินค้าตามน้ำหนักที่ลูกค้าซื้อ ดังเช่นที่คุณเห็นในห้างสรรพสินค้าทั่วไป จะมีการคิดราคาสินค้าในหน่วยสตางค์ และเป็นหน่วยสตางค์ที่ลูกค้าสามารถจ่ายได้ด้วย แยกเป็น 2 กรณี

- ถ้าเป็นการปัดเศษขึ้นเสมอตามเหรียญสตางค์ อยู่ในความรับผิดชอบของเมธอด AlwayRoundUpStang()
- แต่ถ้าเป็นการปัดเศษลงเสมอตามเหรียญสตางค์ อยู่ในความรับผิดชอบของเมธอด AlwayRoundDownStang()

โค้ด VB 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการปัดเศษขึ้นหรือลงเสมอตามเหรียญสตางค์ (GAF.MathTools.vb)

```
    Public Shared Function AlwayRoundUpStang(ByVal a As Double) As Double
        Dim _a As String = ""
        _a = a.ToString("#,##0.00")

        Dim _Floating() As String = _a.Split(CChar("."))

        If CDbl(_Floating(1)) = 0 Then
            Return a
        End If

        If CDbl(_Floating(1)) >= 1 AndAlso CDbl(_Floating(1)) <= 24 Then
            Return CDbl(_Floating(0)) + 0.25
        ElseIf CDbl(_Floating(1)) >= 26 AndAlso CDbl(_Floating(1)) <= 49 Then
            Return CDbl(_Floating(0)) + 0.5
        ElseIf CDbl(_Floating(1)) >= 51 AndAlso CDbl(_Floating(1)) <= 74 Then
            Return CDbl(_Floating(0)) + 0.75
        ElseIf CDbl(_Floating(1)) >= 76 AndAlso CDbl(_Floating(1)) <= 99 Then
            Return CDbl(_Floating(0)) + 1
        Else
            Return a
        End If
    End Function

    Public Shared Function AlwayRoundDownStang(ByVal a As Double) As Double
        Dim _a As String = ""
        _a = a.ToString("#,##0.00")

        Dim _Floating() As String = _a.Split(CChar("."))

        If CDbl(_Floating(1)) = 0 Then
            Return a
        End If

        If CDbl(_Floating(1)) >= 1 AndAlso CDbl(_Floating(1)) <= 24 Then
            Return Math.Floor(a)
        ElseIf CDbl(_Floating(1)) >= 26 AndAlso CDbl(_Floating(1)) <= 49 Then
            Return CDbl(_Floating(0)) + 0.25
        ElseIf CDbl(_Floating(1)) >= 51 AndAlso CDbl(_Floating(1)) <= 74 Then
            Return CDbl(_Floating(0)) + 0.5
        ElseIf CDbl(_Floating(1)) >= 76 AndAlso CDbl(_Floating(1)) <= 99 Then
            Return CDbl(_Floating(0)) + 0.75
        Else
            Return a
        End If
    End Function
```

โค้ด VC# 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการปัดเศษขึ้นหรือลงเสมอตามเหรียญสตางค์ (GAF.MathTools.cs)

```
public static double AlwayRoundUpStang(double a)
{
    string _a = "";
    _a = a.ToString("#,##0.00");

    string[] _Floating = _a.Split(Convert.ToChar("."));

    if (Convert.ToDouble(_Floating[1]) == 0)
    {
        return a;
    }

    if (Convert.ToDouble(_Floating[1]) >= 1 && Convert.ToDouble(_Floating[1]) <= 24)
    {
        return Convert.ToDouble(_Floating[0]) + 0.25;
    }
    else if (Convert.ToDouble(_Floating[1]) >= 26 && Convert.ToDouble(_Floating[1]) <= 49)
    {
        return Convert.ToDouble(_Floating[0]) + 0.5;
    }
    else if (Convert.ToDouble(_Floating[1]) >= 51 && Convert.ToDouble(_Floating[1]) <= 74)
    {
        return Convert.ToDouble(_Floating[0]) + 0.75;
    }
    else if (Convert.ToDouble(_Floating[1]) >= 76 && Convert.ToDouble(_Floating[1]) <= 99)
    {
        return Convert.ToDouble(_Floating[0]) + 1;
    }
    else
    {
        return a;
    }
}

public static double AlwayRoundDownStang(double a)
{
    string _a = "";
    _a = a.ToString("#,##0.00");

    string[] _Floating = _a.Split(Convert.ToChar("."));

    if (Convert.ToDouble(_Floating[1]) == 0)
    {
        return a;
    }
```

```
        if (Convert.ToDouble(_Floating[1]) >= 1 && Convert.ToDouble(_Floating[1]) <= 24)
        {
            return Math.Floor(a);
        }
        else if (Convert.ToDouble(_Floating[1]) >= 26 && Convert.ToDouble(_Floating[1]) <= 49)
        {
            return Convert.ToDouble(_Floating[0]) + 0.25;
        }
        else if (Convert.ToDouble(_Floating[1]) >= 51 && Convert.ToDouble(_Floating[1]) <= 74)
        {
            return Convert.ToDouble(_Floating[0]) + 0.5;
        }
        else if (Convert.ToDouble(_Floating[1]) >= 76 && Convert.ToDouble(_Floating[1]) <= 99)
        {
            return Convert.ToDouble(_Floating[0]) + 0.75;
        }
        else
        {
            return a;
        }
    }
```

หลักการทำงานคือ แบ่งตัวเลขออกเป็น 2 ส่วน โดยใช้จุดเป็นตัวแบ่งแยกตรวจสอบเฉพาะตัวเลขหลังจุดทศนิยมกล่าวคือ

ค่าของตัวเลขหลังทศนิยม	ปัดเศษขึ้น	ปัดเศษลง
0.01-0.24	0.25	0.00
0.26-0.49	0.50	0.25
0.51-0.74	0.75	0.50
0.76-0.99	1.00	0.75

การแปลงตัวเลขให้อยู่ในหน่วย KB, MB, GB และ TB

ใช้ค่าทวีคูณของ 2 เป็นตัวหารกล่าวคือ

KB (กิโลไบต์)	MB (เมกะไบต์)	GB (กิกะไบต์)	TB (เทร่าไบต์)
1024	1048576	1073741824	1099511627776

โค้ด VB 2005 และ VC# 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการแปลงตัวเลขให้อยู่ในหน่วย KB, MB, GB และ TB

VB 2005 (GAF.MathTools.vb)

```
    Public Shared Function ToKByteUnit(ByVal a As
Double) As Double
        Return a / 1024
    End Function

    Public Shared Function ToMByteUnit(ByVal a As
Double) As Double
        Return a / 1048576
    End Function

    Public Shared Function ToGByteUnit(ByVal a As
Double) As Double
        Return a / 1073741824
    End Function

    Public Shared Function ToTByteUnit(ByVal a As
Double) As Double
        Return a / 1099511627776
    End Function
```

VC# 2005 (GAF.MathTools.cs)

```
    public static double ToKByteUnit(double a)
    {
        return a / 1024;
    }

    public static double ToMByteUnit(double a)
    {
        return a / 1048576;
    }

    public static double ToGByteUnit(double a)
    {
        return a / 1073741824;
    }

    public static double ToTByteUnit(double a)
    {
        return a / 1099511627776;
    }
```

การกำหนดให้คอนโทรล TextBox รับได้แต่ตัวเลขด้านการเงิน

ในบางกรณีเราอาจจะต้องการกำหนดให้คอนโทรล TextBox รับแต่เฉพาะตัวเลขด้านการเงิน เช่น ราคาสินค้า, จำนวนสินค้าที่เหลืออยู่, อัตราเงินเดือน เป็นต้น เห็นได้ว่าข้อมูลเหล่านี้ต้องเป็นตัวเลขด้านการเงินเสมอ และอยู่ในความรับผิดชอบของเมธอด CurrencyOnly() เข้ามาทำหน้าที่เป็นยามรับผิดชอบตรวจสอบว่าข้อความที่อยู่ในคอนโทรล TextBox เป้าหมายมีตัวอักษรที่เราต้องการหรือไม่ โดยมีกติกาดังนี้

1. ต้องรับได้แต่ตัวเลขเท่านั้น
2. รับเครื่องหมาย . ได้ แต่มีข้อแม้ว่าต้องมี . เพียง 1 ครั้งเท่านั้น
3. ควรที่จะกดปุ่ม <Delete> หรือปุ่ม <BackSpace> บนคีย์บอร์ด เพื่อลบตัวเลขที่ไม่ต้องการได้
4. ส่วนปุ่มหรือตัวอักษรที่เหลืออยู่บนคีย์บอร์ด ห้ามป้อนโดยเด็ดขาด

หลักการทำงานก็คือ เราจะอาศัยค่าของแฮนเดิล (Handle) ของเหตุการณ์ TextBox_KeyPress() เข้ามาช่วย โดยที่เหตุการณ์ KeyPress มีกติกาดังนี้

ถ้าคุณสมบัติ Handled ของอาร์กิวเมนต์ e ของเหตุการณ์ KeyPress() มีค่าเป็น True หมายความว่าเราต้องการยกเลิกเหตุการณ์ KeyPress()

แต่ถ้าคุณสมบัติ Handled ของอาร์กิวเมนต์ e ของเหตุการณ์ KeyPress() มีค่าเป็น False หมายความว่า เราต้องการให้เกิดเหตุการณ์ KeyPress() ตามปกติ

โค้ด VB 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการกำหนดให้คอนโทรล TextBox รับได้แต่ตัวเลขด้านการเงิน (GAF.MathTools.vb)

```
    Public Shared Function CurrencyOnly(ByVal TargetTextBox As TextBox, ByVal CurrentChar As Char) As Boolean
        If IsNumeric(CurrentChar) = True Then                    'ถ้าเป็นตัวเลข
            Return False                                         'คืนค่า False
        End If

'ถ้าเป็นการป้อนเครื่องหมาย . ครั้งที่ 2
        If CBool(((Convert.ToString(CurrentChar) = "." AndAlso CBool(InStr(TargetTextBox.Text, "."))))) Then
            Return True                                          'คืนค่า True
        End If

'ถ้าเป็นเครื่องหมาย . หรือไม่ก็เป็นปุ่ม Back
        If Convert.ToString(CurrentChar) = "." OrElse CurrentChar = vbBack Then
            Return False                                         'คืนค่า False
        End If
        Return True                                              'ตัวอักษรที่เหลือคืนค่า True
    End Function
End Class
```

โค้ด VC# 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์เฉพาะการกำหนดให้คอนโทรล TextBox รับได้แต่ตัวเลขด้านการเงิน (GAF.MathTools.cs)

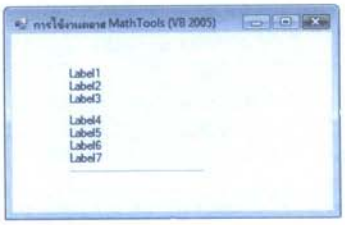
```
    public static bool CurrencyOnly(TextBox TargetTextBox, char CurrentChar)
    {
        if ((int)CurrentChar >= 48 && (int)CurrentChar <= 57)    //ถ้าเป็นตัวเลข
        {
            return false;                                        //คืนค่า False
        }

//ถ้าเป็นการป้อนเครื่องหมาย . ครั้งแรก
        if (Convert.ToString(CurrentChar) == "." && TargetTextBox.Text.IndexOf(".") == -1)
        {
            return false;                                        //คืนค่า false
        }

        if (CurrentChar == Convert.ToChar(Keys.Back))            //ถ้ากดปุ่ม Back แล้ว
        {
            return false;                                        //คืนค่า false
        }
        return true;                                             //ตัวอักษรที่เหลือคืนค่า true
    }
```

เมธอด CurrencyOnly() ทำหน้าที่ตรวจสอบข้อความที่ป้อนเข้ามา คืนค่าเป็นข้อมูลชนิด Boolean ต้องการพารามิเตอร์ 2 ตัวคือ

- TargetTextBox หมายถึง คอนโทรล TextBox เป้าหมายที่ต้องการกำหนดให้รับได้แต่ค่าตัวเลขด้านการเงิน
- CurrentChar หมายถึง ตัวอักษรปัจจุบันที่กำลังตรวจสอบ

การใช้งานคลาส MathTools อยู่ใน Form1 ดังโค้ดต่อไปนี้

รูปที่ 14-6 ฟอร์มในขณะออกแบบ



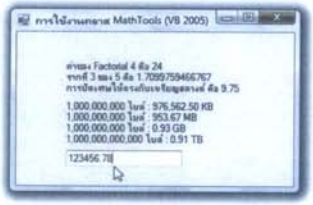
โค้ด VB 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์ (Form1.vb)

```
Option Explicit On
Option Strict On
Imports GAF

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As Decimal = 0


        result = MathTools.Factorial(20)
        Label1.Text = "ค่าของ Factorial 4 คือ " & MathTools.Factorial(4).ToString("#,##0")
        Label2.Text = "รากที่ 3 ของ 5 คือ " & MathTools.SqrtN(5, 3).ToString()
        Label3.Text = "การปัดเศษให้ตรงกับเหรียญสตางค์คือ " & MathTools.AlwayRoundDownStang(9.99).ToString("#,##0.00")

        Label4.Text = "1,000,000,000 ไบต์ : " & MathTools.ToKByteUnit(1000000000).ToString("#,##0.00") & " KB"
        Label5.Text = "1,000,000,000 ไบต์ : " & MathTools.ToMByteUnit(1000000000).ToString("#,##0.00") & " MB"
        Label6.Text = "1,000,000,000 ไบต์ : " & MathTools.ToGByteUnit(1000000000).ToString("#,##0.00") & " GB"
        Label7.Text = "1,000,000,000,000 ไบต์ : " & MathTools.ToTByteUnit(1000000000000).ToString("#,##0.00") & " TB"
    End Sub

    Private Sub TextBox1_KeyPress(ByVal sender As System.Object, ByVal e As System.Windows.Forms.KeyPressEventArgs)
Handles TextBox1.KeyPress
        e.Handled = MathTools.CurrencyOnly(TextBox1, e.KeyChar)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 14-4 การสร้างคลาสที่ทำงานด้านคณิตศาสตร์ (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    decimal result = 0;
    result = MathTools.Factorial(20);
    label1.Text = "ค่าของ Factorial 4 คือ " + MathTools.Factorial(4).ToString("#,##0");
    label2.Text = "รากที่ 3 ของ 5 คือ " + MathTools.SqrtN(5, 3).ToString();
    label3.Text = "การปัดเศษให้ตรงกับเหรียญสตางค์คือ " + MathTools.AlwayRoundDownStang(9.99).ToString("#,##0.00");

    label4.Text = "1,000,000,000 ไบต์ : " + MathTools.ToKByteUnit(1000000000).ToString("#,##0.00") + " KB";
    label5.Text = "1,000,000,000 ไบต์ : " + MathTools.ToMByteUnit(1000000000).ToString("#,##0.00") + " MB";
    label6.Text = "1,000,000,000 ไบต์ : " + MathTools.ToGByteUnit(1000000000).ToString("#,##0.00") + " GB";
    label7.Text = "1,000,000,000,000 ไบต์ : " + MathTools.ToTByteUnit(1000000000000).ToString("#,##0.00") + " TB";
}

private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    e.Handled = MathTools.CurrencyOnly(textBox1, e.KeyChar);
}
```

รูปที่ 14-7
ผลการทำงาน
ตัวอย่างที่ 14-4



NOTE

ในกรณีที่คุณต้องการกำหนดให้คอนโทรล TextBox รับได้แต่เลขจำนวนเต็ม สามารถเขียนโค้ดดังนี้

VB 2005

```
Private Sub TextBox1_KeyPress(ByVal sender As
System.Object, ByVal e As System.Windows.Forms.
KeyPressEventArgs) Handles TextBox1.KeyPress
    If e.KeyChar < "0" Or e.KeyChar > "9" Then
        e.Handled = True
    End If
End Sub
```

VC# 2005

```
private void textBox1_KeyPress(object sender,
KeyPressEventArgs e)
{
    if (e.KeyChar < '0' || e.KeyChar > '9')
    {
        e.Handled = true;
    }
}
```

สรุปท้ายบท

เนื้อหาที่ผู้เขียนเลือกมานำเสนอ เป็นมุมมองของตัวเลขที่เกี่ยวข้องกับการพัฒนาแอพพลิเคชันด้านธุรกิจเป็นหลัก น่าจะมีประโยชน์ในการใช้งานอีกประเด็นหนึ่ง

CHAPTER 15

วันที่และเวลา

บทนำ

ถ้าจะกล่าวถึงชนิดของข้อมูลที่ใกล้ชิดกับเรามากที่สุด คงหนีไม่พ้นข้อมูลชนิด DateTime ซึ่งมีประเด็นปลีกย่อยหลายอย่างที่น่าสนใจอยู่พอสมควร ซึ่งในอดีตการจัดการเกี่ยวกับวันที่และเวลาเป็นเรื่องที่สับสนวุ่นวายและไม่มีมาตรฐาน แต่สำหรับ .NET Framework การจัดการวันที่และเวลาเป็นเรื่องที่เข้าใจได้ไม่ยากและประยุกต์ใช้งานได้อย่างเร็ว

พื้นฐานการใช้งานออบเจ็กต์ DateTime

วันที่และเวลาถือเป็นออบเจ็กต์เช่นกัน และเป็นออบเจ็กต์ที่จับต้องไม่ได้ เป็น Primitive Data Type พื้นฐานอีกตัวหนึ่งที่มีการใช้งานสม่ำเสมอ โดยการใช้คำสั่ง DateTime ในขั้นต้นมีอยู่ 2 คุณสมบัติที่น่าสนใจคือ

คุณสมบัติ	หน้าที่
Now	กำหนดให้ออบเจ็กต์ DateTime แสดงวันที่และเวลาปัจจุบัน
Today	กำหนดให้ออบเจ็กต์ DateTime แสดงเฉพาะวันที่ปัจจุบันเท่านั้น

รูปแบบการแสดงวันที่จาก DateTime ให้คุณดูตัวอย่างที่ 15-1 พื้นฐานการใช้งานออบเจ็กต์ DateTime

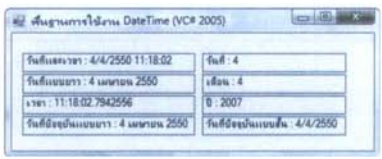
โค้ด VB 2005 ที่ 15-1 พื้นฐานการใช้งานออบเจ็กต์ DateTime (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        lblNow.Text = "วันที่และเวลา : " & DateTime.Now.ToString()
        lblDate.Text = "วันที่แบบยาว : " & DateTime.Now.Date.ToLongDateString()
        lblTimeOfDay.Text = "เวลา : " & DateTime.Now.TimeOfDay.ToString()
        lblDay.Text = "วันที่ : " & DateTime.Now.Day.ToString()
        lblMonth.Text = "เดือน : " & DateTime.Now.Month.ToString()
        lblYear.Text = "ปี : " & DateTime.Now.Year.ToString()
        lblTodayLong.Text = "วันที่ปัจจุบันแบบยาว : " & DateTime.Today.ToLongDateString()
        lblTodayShort.Text = "วันที่ปัจจุบันแบบสั้น : " & DateTime.Today.ToShortDateString()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 15-1 พื้นฐานการใช้งานออบเจ็กต์ DateTime (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        lblNow.Text = "วันที่และเวลา : " + DateTime.Now.ToString();
        lblDate.Text = "วันที่แบบยาว : " + DateTime.Now.Date.ToLongDateString();
        lblTimeOfDay.Text = "เวลา : " + DateTime.Now.TimeOfDay.ToString();
        lblDay.Text = "วันที่ : " + DateTime.Now.Day.ToString();
        lblMonth.Text = "เดือน : " + DateTime.Now.Month.ToString();
        lblYear.Text = "ปี : " + DateTime.Now.Year.ToString();
        lblTodayLong.Text = "วันที่ปัจจุบันแบบยาว : " + DateTime.Today.ToLongDateString();
        lblTodayShort.Text = "วันที่ปัจจุบันแบบสั้น : " + DateTime.Today.ToShortDateString();
    }
```

รูปที่ 15-1 ผลการรันตัวอย่างที่ 15-1



วันที่และเวลาที่ได้จาก DateTime ข้างต้นเป็นรูปแบบที่พบเห็นได้โดยทั่วไป เช่น บางครั้งต้องการวันที่เต็มรูปแบบ บางครั้งต้องการแต่เดือนปัจจุบัน เป็นต้น เห็นได้ว่าการเลือกใช้ค่อนข้างสะดวกพอสมควร

การจัดรูปแบบวันที่และเวลา

ใน .NET มีการอำนวยความสะดวกให้การจัดรูปแบบวันที่และเวลาผ่านทางตัวอักษรควบคุม คุณจะได้ศึกษาว่าเราสามารถจัดรูปแบบของวันที่และเวลาในลักษณะใดบ้าง

ตารางต่อไปนี้เป็นตัวอักษรควบคุม ทำหน้าที่จัดรูปแบบข้อมูลชนิดวันที่และเวลาที่น่าสนใจ ซึ่งคุณสามารถปรับแต่งให้มีรูปแบบมากกว่านี้ได้เช่นกัน ที่สำคัญก็คือ การจัดรูปแบบวันที่และเวลาจะขึ้นอยู่กับการระบุภาษาท้องถิ่นด้วย

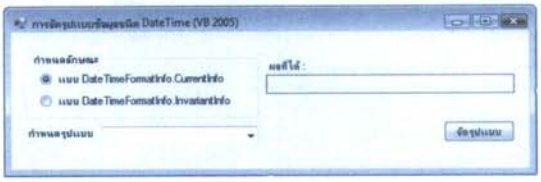
รูปแบบ	คำอธิบาย
d	คืนค่าเป็นวันที่แบบสั้น ● ถ้าเป็นภาษาอังกฤษอยู่ในรูปแบบ เดือน/วัน/ปี ค.ศ. ● ถ้าเป็นภาษาไทยจะอยู่ในรูปแบบ วัน/เดือน/ปี พ.ศ.
dd	คืนค่าวันที่แบบสั้น เช่น 15 หมายถึง วันที่ 15 ทั้งภาษาไทยและอังกฤษ
ddd	คืนค่าอักษรย่อของวันที่ เช่น ● ส. หมายถึง วันเสาร์กรณีภาษาไทย ● ส่วนภาษาอังกฤษคืนค่า Sat
dddd	คืนค่าเป็นวันที่แบบยาว เช่น ● เสาร์กรณีภาษาไทย ● Saturday กรณีภาษาอังกฤษ
M	คืนค่าเป็นวันที่และเดือนแบบยาว เช่น ● 14 มีนาคม กรณีภาษาไทย ● March 14 กรณีภาษาอังกฤษ
MM	คืนค่าเป็นเดือน (ตัวเลขแบบสั้น) เช่น เดือนกุมภาพันธ์จะคืนค่า 02
MMM	คืนค่าเป็นเดือน (อักษรย่อ) เช่น พฤศจิกายน ● จะคืนค่า พ.ย. กรณีภาษาไทย ● Nov กรณีภาษาอังกฤษ
MMMM	คืนค่าเป็นเดือนแบบยาว เช่น มกราคม ● จะคืนค่าเป็นมกราคม กรณีภาษาไทย ● January กรณีภาษาอังกฤษ
yy	คืนค่าเป็นปีแบบสั้น เช่น 2546 ● จะคืนค่าเป็น 46 กรณีภาษาไทย ● 03 กรณีภาษาอังกฤษ

รูปแบบ	คำอธิบาย
yyyy	คืนค่าเป็นปีแบบยาว เช่น 2546 • จะคืนค่าเป็น 2546 กรณีภาษาไทย • 2003 กรณีภาษาอังกฤษ
d MMMM yyyy	คืนค่าเป็นวันเดือนปีแบบยาว เช่น 14 กุมภาพันธ์ 2546 • จะคืนค่าเป็น 14 กุมภาพันธ์ 2546 กรณีภาษาไทย • 14 February 2003 กรณีภาษาอังกฤษ
d MM yyyy	คืนค่าเป็นวันเดือนปีแบบสั้น เช่น 14 กุมภาพันธ์ 2546 • จะคืนค่าเป็น 14 02 2546 กรณีภาษาไทย • 14 02 2003 กรณีภาษาอังกฤษ
d MM yy	จะคืนค่าเป็นวันเดือนปีแบบสั้นทั้งหมด เช่น 2 มีนาคม 2546 • คืนค่า 02 03 46 กรณีภาษาไทย • และคืนค่า 02 03 03 กรณีภาษาอังกฤษ
dd-MM-yy	คืนค่าเป็นวันเดือนปีแบบสั้นทั้งหมดมีขีด เช่น 4 มีนาคม 2546 • คืนค่า 04-03-46 กรณีภาษาไทย • 04-03-03 กรณีภาษาอังกฤษ
dddd, MMMM dd	คืนค่าเป็นเดือนวันปีแบบยาว เช่น 15 กุมภาพันธ์ 2546 • คืนค่า เสาร์, กุมภาพันธ์ 15 กรณีภาษาไทย • Saturday, February 15 กรณีภาษาอังกฤษ
u	คืนค่าเป็นปี ค.ศ. เดือนวันแบบสั้น และเวลารูปแบบชั่วโมง : นาที : วินาที Z ทั้งภาษาไทยและภาษาอังกฤษ
U	• คืนค่าเป็นวันที่ เดือนแบบยาว ปี ค.ศ. และเวลากรณีภาษาไทย • ชื่อวัน, วันที่ เดือนแบบยาว ปี ค.ศ และเวลากรณีภาษาอังกฤษ
t	• คืนค่าเวลา 6 : 30 กรณีภาษาไทย • 06 : 30 กรณีภาษาอังกฤษ
T	• คืนค่าเวลา 6 : 20 : 54 กรณีภาษาไทย • 06 : 20 : 54 กรณีภาษาอังกฤษ
m	คืนค่าเป็นวันที่และเดือนแบบยาว เช่น 15 กุมภาพันธ์ 2546 • คืนค่า 15 กุมภาพันธ์ กรณีภาษาไทย • February 15 กรณีภาษาอังกฤษ
r	ทั้ง 2 ภาษาคืนค่าเป็นวันและเวลามาตรฐาน เช่น 15 กุมภาพันธ์ 2546 คืนค่าเป็น Sat, 15 Feb 2003 01 : 22 : 49 GMT

รูปแบบ	คำอธิบาย
s	ทั้ง 2 ภาษาคืนค่าเป็น ปี เดือน วัน และเวลา เช่น 15 กุมภาพันธ์ 2546 คืนค่าเป็น 2003-02-15T01 : 22 : 51
y	คืนค่าเป็นเดือนและปีแบบยาว เช่น 15 กุมภาพันธ์ 2546 คืนค่าเป็น • กุมภาพันธ์ 2546 กรณีภาษาไทย • February 2003 กรณีภาษาอังกฤษ
g	• คืนค่าเป็น วัน/เดือน/ปี พ.ศ. แบบยาวและเวลา กรณีภาษาไทย เช่น 15 กุมภาพันธ์ 2546 1 : 45 คืนค่าเป็น 15/2/2546 1 : 45 • คืนค่าเป็น เดือน/วัน/ปีแบบยาวและเวลา กรณีภาษาอังกฤษ เช่น 15 กุมภาพันธ์ 2546 1 : 45 คืนค่าเป็น 02/15/2003 01 : 45
G	ให้ผลเช่นเดียวกับตัวอักษร g
f	• คืนค่าเป็นวันที่ เดือนแบบยาว ปี พ.ศ. แบบยาวและเวลา กรณีภาษาไทย เช่น 15 กุมภาพันธ์ 2546 1 : 45 • คืนค่าเป็น ชื่อวัน เดือนแบบยาว วันที่ ปี ค.ศ. แบบยาว กรณีภาษาอังกฤษ เช่น 15 กุมภาพันธ์ 2546 คืนค่าเป็น Saturday,15 February 2003 01 : 45
F	ให้ผลเช่นเดียวกับตัวอักษร f

ตัวอย่างที่ 15-2 การจัดรูปแบบวันที่และเวลา ให้คุณออกแบบฟอร์มดังรูปที่ 15-2

รูปที่ 15-2 ฟอร์มในขณะออกแบบ



จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 15-2 การจัดรูปแบบวันที่และเวลา

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On
Imports System.Globalization

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        cboStyle.Items.Add("d")
        cboStyle.Items.Add("dd")
        cboStyle.Items.Add("ddd")
        cboStyle.Items.Add("dddd")
        cboStyle.Items.Add("M")
        cboStyle.Items.Add("MM")
        cboStyle.Items.Add("MMM")
        cboStyle.Items.Add("MMMM")
        cboStyle.Items.Add("yy")
        cboStyle.Items.Add("yyyy")
        cboStyle.Items.Add("d MMMM yyyy")
        cboStyle.Items.Add("d MM yyyy")
        cboStyle.Items.Add("d MM yy")
        cboStyle.Items.Add("dd-MM-yy")
        cboStyle.Items.Add("dddd, MMMM dd ")
        cboStyle.Items.Add("u")
        cboStyle.Items.Add("U")
        cboStyle.Items.Add("t")
        cboStyle.Items.Add("T")
        cboStyle.Items.Add("m")
        cboStyle.Items.Add("r")
        cboStyle.Items.Add("s")
        cboStyle.Items.Add("y")
        cboStyle.Items.Add("g")
        cboStyle.Items.Add("G")
        cboStyle.Items.Add("f")
        cboStyle.Items.Add("F")
        cboStyle.SelectedIndex = 0
    End Sub

    Private Sub cmdFormat_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdFormat.Click
        Dim dt As DateTime = DateTime.Now
        Dim dtfInfo As DateTimeFormatInfo

        Dim DateStyle As String = ""
        If optInvariant.Checked Then
```

VC# 2005 (Form1.cs)

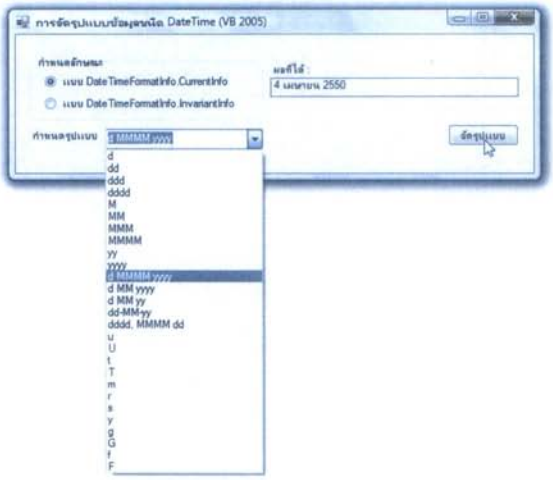
```
private void Form1_Load(object sender, EventArgs e)
{
    cboStyle.Items.Add("d");
    cboStyle.Items.Add("dd");
    cboStyle.Items.Add("ddd");
    cboStyle.Items.Add("dddd");
    cboStyle.Items.Add("M");
    cboStyle.Items.Add("MM");
    cboStyle.Items.Add("MMM");
    cboStyle.Items.Add("MMMM");
    cboStyle.Items.Add("yy");
    cboStyle.Items.Add("yyyy");
    cboStyle.Items.Add("d MMMM yyyy");
    cboStyle.Items.Add("d MM yyyy");
    cboStyle.Items.Add("d MM yy");
    cboStyle.Items.Add("dd-MM-yy");
    cboStyle.Items.Add("dddd, MMMM dd ");
    cboStyle.Items.Add("u");
    cboStyle.Items.Add("U");
    cboStyle.Items.Add("t");
    cboStyle.Items.Add("T");
    cboStyle.Items.Add("m");
    cboStyle.Items.Add("r");
    cboStyle.Items.Add("s");
    cboStyle.Items.Add("y");
    cboStyle.Items.Add("g");
    cboStyle.Items.Add("G");
    cboStyle.Items.Add("f");
    cboStyle.Items.Add("F");
    cboStyle.SelectedIndex = 0;
}

private void cmdFormat_Click(object sender, EventArgs e)
{
    DateTime dt = DateTime.Now;
    DateTimeFormatInfo dtfInfo;

    string DateStyle = "";
    if (optInvariant.Checked)
    {
        dtfInfo = DateTimeFormatInfo.InvariantInfo;
    }
    else
    {
```

```
            dtfInfo = DateTimeFormatInfo.InvariantInfo
        Else
            dtfInfo = DateTimeFormatInfo.CurrentInfo
        End If

        DateStyle = cboStyle.Text
        lblDisplay.Text = dt.ToString(DateStyle, dtfInfo)
    End Sub
End Class
```

```
            dtfInfo = DateTimeFormatInfo.CurrentInfo;
        }

        DateStyle = cboStyle.Text;
        lblDisplay.Text = dt.ToString(DateStyle, dtfInfo);
    }
```

ตัวอย่างนี้จะอาศัยออบเจ็กต์ DateTimeFormatInfo จัดรูปแบบวันที่และเวลาให้กับออบเจ็กต์ DateTime ผลการทำงานแสดงดังรูปที่ 15-3 และ 15-4

รูปที่ 15-3 กรณีจัดรูปแบบวันที่ของภาษาท้องถิ่น ในกรณีนี้คือ ภาษาไทย



รูปที่ 15-4 กรณีจัดรูปแบบวันที่ของภาษาสากล

ในเหตุการณ์ Form_Load() จะเพิ่มรายการรูปแบบการจัดวันที่และเวลาเข้าไปในคอนโทรล cboStyle ก่อน

VB 2005

```
cboStyle.Items.Add("d")
cboStyle.Items.Add("dd")
cboStyle.Items.Add("ddd")
cboStyle.Items.Add("dddd")
cboStyle.Items.Add("M")
cboStyle.Items.Add("MM")
cboStyle.Items.Add("MMM")
cboStyle.Items.Add("MMMM")
cboStyle.Items.Add("yy")
cboStyle.Items.Add("yyyy")
cboStyle.Items.Add("d MMMM yyyy")
cboStyle.Items.Add("d MM yyyy")
cboStyle.Items.Add("d MM yy")
cboStyle.Items.Add("dd-MM-yy")
cboStyle.Items.Add("dddd, MMMM dd ")
cboStyle.Items.Add("u")
cboStyle.Items.Add("U")
cboStyle.Items.Add("t")
cboStyle.Items.Add("T")
cboStyle.Items.Add("m")
cboStyle.Items.Add("r")
cboStyle.Items.Add("s")
cboStyle.Items.Add("y")
cboStyle.Items.Add("g")
cboStyle.Items.Add("G")
cboStyle.Items.Add("f")
cboStyle.Items.Add("F")
cboStyle.SelectedIndex = 0
```

VC# 2005

```
cboStyle.Items.Add("d");
cboStyle.Items.Add("dd");
cboStyle.Items.Add("ddd");
cboStyle.Items.Add("dddd");
cboStyle.Items.Add("M");
cboStyle.Items.Add("MM");
cboStyle.Items.Add("MMM");
cboStyle.Items.Add("MMMM");
cboStyle.Items.Add("yy");
cboStyle.Items.Add("yyyy");
cboStyle.Items.Add("d MMMM yyyy");
cboStyle.Items.Add("d MM yyyy");
cboStyle.Items.Add("d MM yy");
cboStyle.Items.Add("dd-MM-yy");
cboStyle.Items.Add("dddd, MMMM dd ");
cboStyle.Items.Add("u");
cboStyle.Items.Add("U");
cboStyle.Items.Add("t");
cboStyle.Items.Add("T");
cboStyle.Items.Add("m");
cboStyle.Items.Add("r");
cboStyle.Items.Add("s");
cboStyle.Items.Add("y");
cboStyle.Items.Add("g");
cboStyle.Items.Add("G");
cboStyle.Items.Add("f");
cboStyle.Items.Add("F");
cboStyle.SelectedIndex = 0;
```

ส่วนในเหตุการณ์ cmdFormat_Click() เริ่มต้นประกาศตัวแปรออบเจ็กต์ DateTime ที่ชื่อว่า dt อ่านวันที่และเวลาปัจจุบันจากคุณสมบัติ Now และประกาศตัวแปรออบเจ็กต์ DateTimeFormatInfo ที่ชื่อว่า dtfInfo ทำหน้าที่จัดรูปแบบวันที่และเวลา

VB 2005

```
Dim dt As DateTime = DateTime.Now
Dim dtfInfo As DateTimeFormatInfo
```

VC# 2005

```
DateTime dt = DateTime.Now;
DateTimeFormatInfo dtfInfo;
```

ต่อมาจะตรวจสอบว่า ถ้าผู้ใช้คลิกเลือก Invariant ก็จะมาทำงานที่บล็อกคำสั่งนี้ ซึ่งเป็นการกำหนดวิธีการจัดรูปแบบวันที่และเวลาว่าจะเป็นแบบใด โดยที่ในบล็อกคำสั่งนี้จะกำหนดเป็นแบบสากล (ภาษาอังกฤษ) โดยการอ่านจากคุณสมบัติ InvariantInfo ของออบเจ็กต์ DateTimeFormatInfo เก็บไว้ที่ตัวแปร dtfInfo

VB 2005

```
Dim DateStyle As String = ""
If optInvariant.Checked Then
    dtfInfo = DateTimeFormatInfo.InvariantInfo
```

VC# 2005

```
string DateStyle = "";
if (optInvariant.Checked)
{
    dtfInfo = DateTimeFormatInfo.InvariantInfo;
}
```

แต่ถ้าผู้ใช้คลิกเลือก Current ก็จะมาทำงานที่บล็อกคำสั่งนี้ โดยการกำหนดให้เป็นรูปแบบภาษาท้องถิ่น ซึ่งก็คือภาษาไทยนั่นเอง โดยการอ่านจากคุณสมบัติ CurrentInfo ของออบเจ็กต์ DateTimeFormatInfo เก็บไว้ที่ตัวแปร dtfInfo

VB 2005

```
Else
    dtfInfo = DateTimeFormatInfo.CurrentInfo
End If
```

VC# 2005

```
else
{
    dtfInfo = DateTimeFormatInfo.CurrentInfo;
}
```

ท้ายที่สุดอ่านรูปแบบที่ผู้ใช้เลือกในคอนโทรล cboStyle เก็บไว้ที่ตัวแปร DateStyle จัดรูปแบบวันที่และเวลาด้วยเมธอด ToString() โดยอาศัยรูปแบบที่อยู่ในตัวแปร DateStyle ส่วนวิธีการจัดรูปแบบจะอยู่ในตัวแปร dtfInfo

VB 2005

```
    DateStyle = cboStyle.Text
    lblDisplay.Text = dt.ToString(DateStyle, dtfInfo)
End Sub
```

VC# 2005

```
    DateStyle = cboStyle.Text;
    lblDisplay.Text = dt.ToString(DateStyle, dtfInfo);
}
```

การจัดรูปแบบ DateTime ของ .NET Framework

การจัดรูปแบบของ DateTime ของ .NET มีมากมายถึง 120 รูปแบบ แสดงดังตัวอย่างที่ 15-3

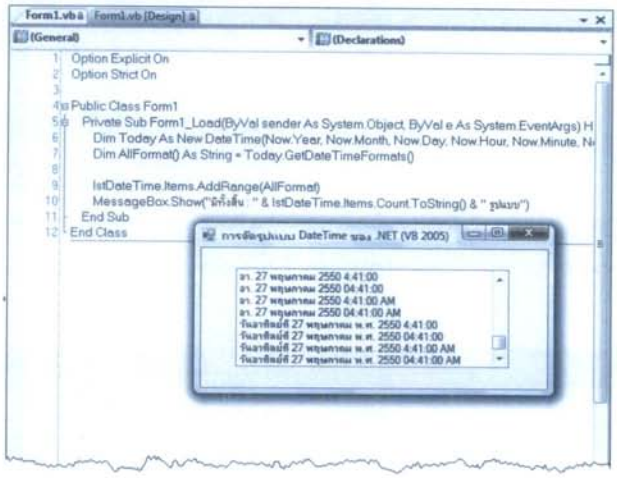
โค้ด VB 2005 ที่ 15-3 การจัดรูปแบบ DateTime ของ .NET Framework (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim Today As New DateTime(Now.Year, Now.Month, Now.Day, Now.Hour, Now.Minute, Now.Second)
        Dim AllFormat() As String = Today.GetDateTimeFormats()

        lstDateTime.Items.AddRange(AllFormat)
        MessageBox.Show("มีทั้งสิ้น : " & lstDateTime.Items.Count.ToString() & " รูปแบบ")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 15-3 การจัดรูปแบบ DateTime ของ .NET Framework (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    DateTime Today = new DateTime(DateTime.Now.Year, DateTime.Now.Month, DateTime.Now.Day, DateTime.Now.Hour,
DateTime.Now.Minute, DateTime.Now.Second);
    string[] AllFormat = Today.GetDateTimeFormats();

    lstDateTime.Items.AddRange(AllFormat);
    MessageBox.Show("มีทั้งสิ้น : " + lstDateTime.Items.Count.ToString() + " รูปแบบ");
}
```

รูปที่ 15-5
ผลการรัน
ตัวอย่างที่ 15-3



การใช้งานวันที่ในรูปแบบภาษาไทย

ความสามารถอีกอย่างหนึ่งของระบบอ้างอิงวันที่และเวลาในสถาปัตยกรรม .NET ก็คือ การสนับสนุนภาษาท้องถิ่นต่างๆ ถือเป็นคุณสมบัติที่ยอดเยี่ยมอีกส่วนหนึ่งของ .NET โดยมีเนมสเปซที่รับผิดชอบเกี่ยวกับระบบวันที่และเวลาของภาษาท้องถิ่นคือ

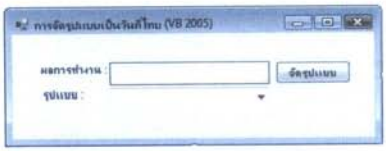
System.Globalization

ออบเจ็กต์ที่ทำหน้าที่เกี่ยวกับวันที่และเวลาของภาษาท้องถิ่นคือ

- ออบเจ็กต์ CultureInfo ทำหน้าที่ระบุภาษาท้องถิ่น
- ออบเจ็กต์ DateTimeFormatInfo ทำหน้าที่อ่านรูปแบบวันที่และเวลาที่ภาษานั้นๆ สนับสนุนอยู่

ตัวอย่างที่ 15-4 การใช้งานวันที่ในรูปแบบภาษาไทย ให้คุณออกแบบฟอร์ม ดังรูปที่ 15-6

รูปที่ 15-6 ฟอร์มในขณะออกแบบ



ให้คุณเพิ่มเนมสเปซต่อไปนี้ก่อนเขียนโค้ด

VB 2005	VC# 2005
Imports System.Globalization	using System.Globalization;

จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 ที่ 15-4 การใช้งานวันที่ในรูปแบบภาษาไทย (Form1.vb)

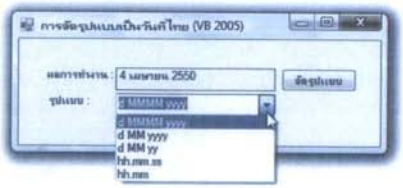
```
Option Explicit On
Option Strict On
Imports System.Globalization

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        cboStyle.Items.Add("d MMMM yyyy")
        cboStyle.Items.Add("d MM yyyy")
        cboStyle.Items.Add("d MM yy")
        cboStyle.Items.Add("hh.mm.ss")
        cboStyle.Items.Add("hh.mm")
        cboStyle.SelectedIndex = 0
    End Sub

    Private Sub cmdFormat_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdFormat.Click
        Dim DateStyle As String = ""
        Dim dt As DateTime = DateTime.Now
        Dim Thai_CInfo As New CultureInfo("th-TH")
        Dim Thai_dtfInfo As DateTimeFormatInfo = Thai_CInfo.DateTimeFormat

        DateStyle = cboStyle.Text
        lblDisplay.Text = dt.ToString(DateStyle, Thai_dtfInfo)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 15-4 การใช้งานวันที่ในรูปแบบภาษาไทย (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    cboStyle.Items.Add("d MMMM yyyy");
    cboStyle.Items.Add("d MM yyyy");
    cboStyle.Items.Add("d MM yy");
    cboStyle.Items.Add("hh.mm.ss");
    cboStyle.Items.Add("hh.mm");
    cboStyle.SelectedIndex = 0;
}

private void cmdFormat_Click(object sender, EventArgs e)
{
    string DateStyle = "";
    DateTime dt = DateTime.Now;
    CultureInfo Thai_CInfo = new CultureInfo("th-TH");
    DateTimeFormatInfo Thai_dtfInfo = Thai_CInfo.DateTimeFormat;
    DateStyle = cboStyle.Text;
    lblDisplay.Text = dt.ToString(DateStyle, Thai_dtfInfo);
}
```

ให้คุณเลือกรูปแบบวันที่หรือเวลาที่อยู่ในคอนโทรล comboBox แล้วคลิกปุ่ม จัดรูปแบบ ผลการจัดรูปแบบ แสดงดังรูปที่ 15-7

รูปที่ 15-7
ผลการรัน
ตัวอย่างที่ 15-4



ในเหตุการณ์ Form_Load() จะเพิ่มรูปแบบของวันที่และเวลาเข้าไปในคอนโทรล ComboBox ที่ชื่อว่า cboStyle เพื่อให้ผู้ใช้เลือกและกำหนดให้รายการแรกปรากฏขึ้นมา โดยการกำหนดคุณสมบัติ SelectedIndex = 0

VB 2005

```
cboStyle.Items.Add("d MMMM yyyy")
cboStyle.Items.Add("d MM yyyy")
cboStyle.Items.Add("d MM yy")
cboStyle.Items.Add("hh.mm.ss")
cboStyle.Items.Add("hh.mm")
cboStyle.SelectedIndex = 0
```

VC# 2005

```
cboStyle.Items.Add("d MMMM yyyy");
cboStyle.Items.Add("d MM yyyy");
cboStyle.Items.Add("d MM yy");
cboStyle.Items.Add("hh.mm.ss");
cboStyle.Items.Add("hh.mm");
cboStyle.SelectedIndex = 0;
```

ในเหตุการณ์ cmdFormat_Click() เริ่มต้นประกาศตัวแปรออบเจ็กต์ DateTime ที่ชื่อว่า dt อ่านวันที่และเวลาปัจจุบันออกมาจากคุณสมบัติ Now ก่อน จากนั้นประกาศตัวแปรออบเจ็กต์ CultureInfo ที่ชื่อว่า Thai_CInfo โดยระบุภาษาท้องถิ่นคือ ภาษาไทย (th-TH)

สร้างตัวแปรออบเจ็กต์ DateTimeFormatInfo ที่ชื่อว่า Thai_dtfInfo ก็จะมีการอ่านรูปแบบของวันที่และเวลาที่ภาษาท้องถิ่นนั้นๆ สนับสนุน ผ่านทางคุณสมบัติ DateTimeFormat ของออบเจ็กต์ Thai_CInfo

VB 2005

```
Dim DateStyle As String = ""
Dim dt As DateTime = DateTime.Now
Dim Thai_CInfo As New CultureInfo("th-TH")
Dim Thai_dtfInfo As DateTimeFormatInfo = Thai_CInfo.DateTimeFormat

DateStyle = cboStyle.Text
lblDisplay.Text = dt.ToString(DateStyle, Thai_dtfInfo)
```

VC# 2005

```
string DateStyle = "";
DateTime dt = DateTime.Now;
CultureInfo Thai_CInfo = new CultureInfo("th-TH");
DateTimeFormatInfo Thai_dtfInfo = Thai_CInfo.DateTimeFormat;
DateStyle = cboStyle.Text;
lblDisplay.Text = dt.ToString(DateStyle, Thai_dtfInfo);
```

ท้ายที่สุดอ่านรูปแบบวันที่ หรือเวลาที่ผู้ใช้เลือกในคอนโทรล cboStyle เก็บไว้ที่ตัวแปร DateStyle แสดงรูปแบบที่ได้ออกมาที่คอนโทรล label โดยใช้เมธอด ToString() โดยอาศัยรูปแบบที่ผู้ใช้เลือก ซึ่งเก็บอยู่ในตัวแปร DateStyle และรูปแบบของภาษาท้องถิ่นที่เก็บอยู่ในตัวแปรออบเจ็กต์ Thai_dtfInfo

การตรวจสอบวันที่ปัจจุบัน

ในการพัฒนาแอพพลิชัน มักจะใช้วันที่มาเป็นส่วนหนึ่งของเงื่อนไขในระบบเสมอ เราจะพบว่าบางกรณีมีการโฆษณาจูงใจให้ลูกค้ามาซื้อสินค้าในวันจันทร์ จะได้ราคาถูกกว่าวันอื่นๆ เป็นการกระตุ้นยอดขายของวันที่ที่มีคนมาใช้บริการน้อยกว่าวันอื่นๆ เห็นได้ว่าเป็นกลยุทธ์ที่เราพบเห็นได้โดยทั่วไป คำถามก็คือ ถ้าเราเป็นผู้พัฒนาระบบดังกล่าวเราจะทำอย่างไร ให้คุณดูตัวอย่างต่อไปนี้

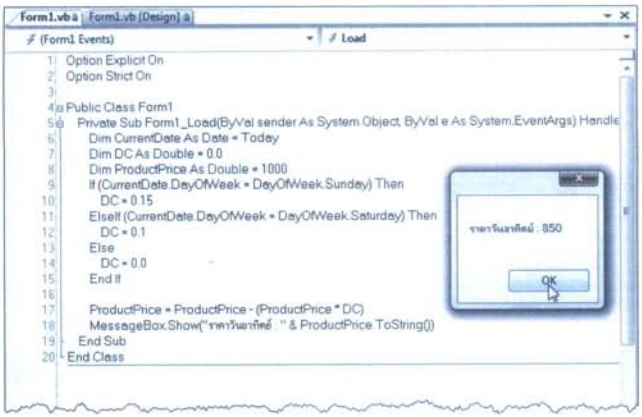
โค้ด VB 2005 ที่ 15-5 การตรวจสอบวันที่ปัจจุบัน (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim CurrentDate As Date = Today
        Dim DC As Double = 0.0
        Dim ProductPrice As Double = 1000
        If (CurrentDate.DayOfWeek = DayOfWeek.Sunday) Then
            DC = 0.15
        ElseIf (CurrentDate.DayOfWeek = DayOfWeek.Saturday) Then
            DC = 0.1
        Else
            DC = 0.0
        End If

        ProductPrice = ProductPrice - (ProductPrice * DC)
        MessageBox.Show("ราคาวันอาทิตย์ : " & ProductPrice.ToString())
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 15-5 การตรวจสอบวันที่ปัจจุบัน (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        DateTime CurrentDate = DateTime.Today;
        double DC = 0.0;
        double ProductPrice = 1000;
        if (CurrentDate.DayOfWeek == DayOfWeek.Sunday)
        {
            DC = 0.15;
        }
        else if (CurrentDate.DayOfWeek == DayOfWeek.Saturday)
        {
            DC = 0.1;
        }
            else
        {
            DC = 0.0;
        }

        ProductPrice = ProductPrice - (ProductPrice * DC);
         MessageBox.Show("ราคาวันอาทิตย์ : " + ProductPrice.ToString());
    }
```

รูปที่ 15-8
ผลการรัน
ตัวอย่างที่ 15-5



จากรูปที่ 15-8 สมมติว่าวันปัจจุบันคือ วันอาทิตย์ ตั้งเงื่อนไขไว้ว่า

- ถ้าเป็นวันอาทิตย์ มีส่วนลด 15% (ตัวแปร DC = 0.15)
- แต่ถ้าเป็นวันเสาร์ มีส่วนลด 10% (ตัวแปร DC = 0.10)
- แต่ถ้าเป็นวันอื่นๆ ไม่มีส่วนลด (ตัวแปร DC = 0.0)

ราคาสินค้า 1000 บาท เมื่อซื้อในวันอาทิตย์ก็จะได้ส่วนลด 15% เหลือ 850 บาท

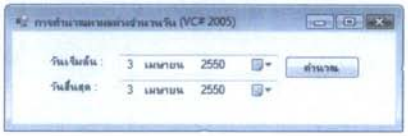
การหาผลต่างในหน่วยวัน

คอนโทรล DateTimePicker ทำหน้าที่แสดงวันที่แบบปฏิทิน มีลักษณะคล้ายๆ กับคอนโทรล comboBox โดยมีระยะเวลาตั้งแต่วันที่ 1 มกราคม พ.ศ. 2296 ถึงวันที่ 31 ธันวาคม พ.ศ. 10541 คุณสมบัติที่น่าสนใจ มีดังนี้

คุณสมบัติ	หน้าที่
Value	แสดงค่าวันที่ปัจจุบันที่ถูกเลือกอยู่ ถ้าไม่มีการเลือกวันที่เกิดขึ้น จะคืนค่าเป็นวันที่ปัจจุบันของระบบ
Format	กำหนดรูปแบบวันที่ที่จะแสดงออกมา โดยที่ • Long แสดงวันที่แบบยาว • Short แสดงวันที่แบบสั้น • Time แสดงเฉพาะเวลา • Custom ปรับแต่งรูปแบบเอง
CustomFormat	กำหนดรูปแบบวันที่หรือเวลาเอง โดยที่คุณสมบัติ Format ต้องเท่ากับ Custom

การหาผลต่างจำนวนวันเป็นการทำงานอีกอย่างที่เราพบเห็นได้ในแอพพลิเคชันทั่วๆ ไป เช่น นำไปคิดค่าบริการ, ดอกเบี้ย, ค่าปรับตามจำนวนวัน เป็นต้น ให้ดูตัวอย่างที่ 15-6

รูปที่ 15-9 ฟอร์มในขณะออกแบบ



จากนั้นให้คุณเขียนโค้ดดังนี้

โค้ด VB 2005 ที่ 15-6 การหาผลต่างในหน่วยวัน (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        dtpStart.Format = DateTimePickerFormat.Short
        dtpEnd.Format = DateTimePickerFormat.Long

        Dim Today As DateTime = DateTime.Now
        dtpStart.Value = Today
        dtpEnd.Value = Today.AddMonths(1)
    End Sub

    Private Sub cmdCalculate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCalculate.Click
        Dim NumberDay As Integer = 0
        NumberDay = dtpEnd.Value.Subtract(dtpStart.Value).Days
        MessageBox.Show("ผลต่างของวันที่คุณเลือกเท่ากับ " & NumberDay.ToString() & " วัน", "คำนวณผลต่างในหน่วยวัน")

        'NumberDay = CInt(DateDiff(DateInterval.Day, dtpStart.Value, dtpEnd.Value))
        'MessageBox.Show("ผลต่างของวันที่คุณเลือกเท่ากับ " & NumberDay.ToString() & " วัน", "คำนวณผลต่างในหน่วยวัน")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 15-6 การหาผลต่างในหน่วยวัน (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        dtpStart.Format = DateTimePickerFormat.Short;
        dtpEnd.Format = DateTimePickerFormat.Long;

        DateTime Today = DateTime.Now;

        dtpStart.Value = Today;
        dtpEnd.Value = Today.AddMonths(1);
    }
```

```
private void cmdCalculate_Click(object sender, EventArgs e)
{
    int NumberDay=0;
    NumberDay = dtpEnd.Value.Subtract(dtpStart.Value).Days;
    MessageBox.Show("ผลต่างของวันที่คุณเลือกเท่ากับ " + NumberDay.ToString() + " วัน", "คำนวณผลต่างในหน่วยวัน");
}
```

รูปที่ 15-10
ผลการรัน
โปรเจ็กต์ที่ 15-6

จากรูปที่ 15-10 ผู้เขียนจงใจกำหนดให้คอนโทรล DateTimePicker ตัวบนแสดงวันที่แบบสั้น ส่วนคอนโทรล DateTimePicker ตัวล่างแสดงวันที่แบบยาว ซึ่งเกิดจากการกำหนดด้วยวิธีการเขียนโค้ด ให้คลิกปุ่ม คำนวณ เพื่อคำนวณผลต่างระหว่างวันที่ที่แสดงอยู่ในคอนโทรล DateTimePicker ทั้ง 2 ตัว

ในเหตุการณ์ Form1_Load() สร้างตัวแปรออบเจ็กต์ DateTime ที่ชื่อว่า Today อ่านค่าวันที่และเวลาปัจจุบันผ่านทางคุณสมบัติ Now

VB 2005	VC# 2005
`Dim Today As DateTime = DateTime.Now`	`DateTime Today = DateTime.Now;`

ต่อมาก็จะกำหนดให้คอนโทรล DateTimePicker ตัวบน (dtpStart) แสดงวันที่ปัจจุบันผ่านทางตัวแปรออบเจ็กต์ Today ส่วนคอนโทรล DateTimePicker ตัวล่าง (dtpEnd) กำหนดให้แสดงวันที่นับจากวันที่ปัจจุบันไปอีก 1 เดือน โดยการใช้เมธอด AddMonths()

```
dtpStart.Value  = Today;
dtpEnd.Value =Today.AddMonths(1);
```

NOTE

คุณสามารถเพิ่มในหน่วยอื่นๆ ได้เช่นกัน โดยการเปลี่ยนจากเมธอด AddMonths() เป็นเมธอดอื่นๆ เช่น เพิ่มในหน่วยของวัน, ชั่วโมง ฯลฯ ได้ตามที่คุณต้องการ

ในกรณีที่คุณต้องการลดจำนวนเดือน สามารถทำได้โดยการส่งค่าลบเข้าไปได้เช่นกัน เช่น แสดงวันที่ย้อนหลังกลับไป 1 เดือน นับจากวันที่ปัจจุบัน เป็นต้น

VB 2005	VC# 2005
`dtpEnd.Value = Today.AddMonths(-1)`	`dtpEnd.Value = Today.AddMonths(-1);`

ในส่วนของการคำนวณผลต่างระหว่างวันที่ อยู่ในเหตุการณ์ cmdCal_Click() เริ่มต้นจะประกาศตัวแปร Integer (int) ที่ชื่อว่า NumberDay ทำหน้าที่เก็บผลต่างที่คำนวณได้ ก็จะอาศัยเมธอด Subtract() ทำหน้าที่คำนวณค่าที่ส่งเข้าไปก็คือ วันที่เริ่มต้นเป็นตัวลบจากวันที่ล่าสุดนั่นเอง

VB 2005

```
Dim NumberDay As Integer = 0
NumberDay = dtpEnd.Value.Subtract(dtpStart.Value).Days
```

VC# 2005

```
nt NumberDay=0;
NumberDay = dtpEnd.Value.Subtract(dtpStart.Value).Days;
```

การทำงานของเมธอด Subtract() สามารถเปรียบเทียบได้ดังนี้

```
NumberDay=dtpEnd-dtpStart
```

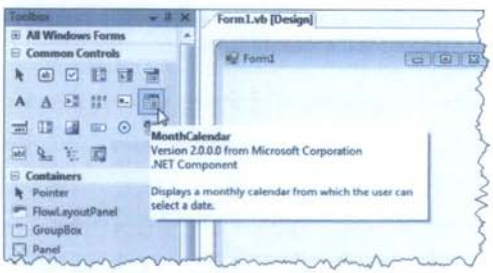
ในกรณีของ VB 2005 คุณสามารถใช้ฟังก์ชัน DateDiff() เพื่อคำนวณหาจำนวนวันได้เช่นกัน

VB 2005

```
NumberDay = CInt(DateDiff(DateInterval.Day, dtpStart.Value, dtpEnd.Value))
MessageBox.Show("ผลต่างของวันที่คุณเลือกเท่ากับ " & NumberDay.ToString() & " วัน", "คำนวณผลต่างในหน่วยวัน")
```

การใช้งานคอนโทรล MonthCalendar

คอนโทรล MonthCalendar มีหน้าที่เช่นเดียวกับคอนโทรล DateTimePicker กล่าวคือ ใช้สำหรับแสดงวันที่เช่นเดียวกัน ข้อแตกต่างก็คือ จะมีลักษณะเป็นแบบปฏิทิน ดังรูปที่ 15-11



รูปที่ 15-11 คอนโทรล MonthCalendar ในแถบเครื่องมือของ VS 2005

คุณสมบัติที่น่าสนใจมีดังนี้

คุณสมบัติ	หน้าที่
ShowToday	กำหนดให้คอนโทรล MonthCalendar แสดงวันที่ปัจจุบันบริเวณด้านล่างคอนโทรล
MaxSelectionCount	กำหนดจำนวนวันสูงสุดที่ผู้ใช้สามารถคลิกเลือกได้ มีค่าไม่เกิน 42 วันต่อการเลือก 1 ครั้ง
SelectionRange	ระยะวันที่ผู้ใช้คลิกเลือกในคอนโทรล MonthCalendar

ตัวอย่างที่ 15-7 การใช้งานคอนโทรล MonthCalendar ให้คุณออกแบบฟอร์ม ดังรูปที่ 15-12

รูปที่ 15-12 ฟอร์มในขณะออกแบบ

จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 ที่ 15-7 การใช้งานคอนโทรล MonthCalendar (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim Today As DateTime = DateTime.Now
        mclTest.ShowToday = True
        mclTest.TodayDate = Today
        mclTest.MaxSelectionCount = 42
    End Sub

    Private Sub cmdCalculateDate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdCalculateDate.Click
        Dim NumberDay As Integer
        Dim StartDate As DateTime = mclTest.SelectionRange.Start
        Dim EndDate As DateTime = mclTest.SelectionRange.End
        NumberDay = EndDate.Subtract(StartDate).Days + 1

        MessageBox.Show("จำนวนวันที่ที่คุณเลือกเท่ากับ " & NumberDay.ToString() & " วัน", "คำนวณวันที่เลือก")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 15-7 การใช้งานคอนโทรล MonthCalendar (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    DateTime Today = DateTime.Now;
    mclTest.ShowToday = true;
    mclTest.TodayDate = Today;
    mclTest.MaxSelectionCount = 42;
}

private void cmdCalculateDate_Click(object sender, EventArgs e)
{
    int NumberDay;
    DateTime StartDate = mclTest.SelectionRange.Start;
    DateTime EndDate = mclTest.SelectionRange.End;

    NumberDay = (EndDate.Subtract(StartDate).Days + 1);
    MessageBox.Show("จำนวนวันที่ที่คุณเลือกเท่ากับ " + NumberDay.ToString() + " วัน", "คำนวณวันที่เลือก");
}
```

ตัวอย่างนี้แสดงให้เห็นวิธีการคำนวณวันที่ที่ถูกผู้ใช้เลือกในคอนโทรล MonthCalendar ดังรูปที่ 15-13 ให้คุณเลือกวันที่โดยการลากวันที่ครอบคลุมหลายๆ วันที่ แล้วคลิกปุ่ม คำนวณวันที่ถูกเลือก เพื่อคำนวณจำนวนวันที่ถูกคลิกเลือก

รูปที่ 15-13
ผลการทำงาน
ตัวอย่างที่ 15-7

ในเหตุการณ์ Form_Load() สร้างตัวแปร DateTime ที่ชื่อว่า Today โดยการอ่านวันที่และเวลาออกมาจากคุณสมบัติ Now จากนั้นกำหนดให้คอนโทรล MonthCalendar แสดงวันที่ปัจจุบัน, กำหนดวันที่ปัจจุบัน และกำหนดจำนวนวันที่ถูกเลือกสูงสุดไม่เกิน 42 วัน

VB 2005

```
Dim Today As DateTime = DateTime.Now
mclTest.ShowToday = True
mclTest.TodayDate = Today
mclTest.MaxSelectionCount = 42
```

VC# 2005

```
DateTime Today = DateTime.Now;
mclTest.ShowToday = true;
mclTest.TodayDate = Today;
mclTest.MaxSelectionCount = 42;
```

ส่วนในเหตุการณ์ cmdCalculateDate_Click() จะอ่านวันที่เริ่มต้นที่ถูกผู้ใช้คลิกเลือกจากคุณสมบัติ SelectionRange ร่วมกับคุณสมบัติ Start เก็บไว้ในตัวแปรออบเจ็กต์ที่ชื่อว่า StartDate ส่วนวันที่สิ้นสุดที่ถูกเลือกอ่านได้จากคุณสมบัติ End ของคอนโทรล MonthCalendar เก็บไว้ที่ตัวแปร EndDate คำนวณจำนวนวันที่ถูกเลือก โดยใช้เมธอด Subtract() และแสดงค่าที่คำนวณได้

VB 2005

```
Dim NumberDay As Integer = 0
Dim StartDate As DateTime = mclTest.SelectionRange.Start
Dim EndDate As DateTime = mclTest.SelectionRange.End

NumberDay = EndDate.Subtract(StartDate).Days + 1
MessageBox.Show("จำนวนวันที่ที่คุณเลือกเท่ากับ " & NumberDay.ToString() & " วัน", "คำนวณวันที่เลือก")
```

VC# 2005

```
int NumberDay = 0;
DateTime StartDate = mclTest.SelectionRange.Start;
DateTime EndDate = mclTest.SelectionRange.End;

NumberDay = (EndDate.Subtract(StartDate).Days + 1);
MessageBox.Show("จำนวนวันที่ที่คุณเลือกเท่ากับ " + NumberDay.ToString() + " วัน", "คำนวณวันที่เลือก");
```

การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime

เราจะสร้างคลาสเพิ่มเติมโดยให้มีความสามารถครอบคลุมเรื่องต่อไปนี้

- การหาจำนวนวันของเดือนและจำนวนวันของปีปัจจุบัน
- การอ่านชื่อวันที่เป็นภาษาอังกฤษและภาษาไทย
- การคิดเวลาใช้งานแบบโทรศัพท์มือถือ
- การหาวันทำงานถัดไป

การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime

สำหรับผู้ช่วยเหลือของบทนี้ชื่อว่า DateTimeTools รับผิดชอบงานด้านวันที่และเวลาอยู่ในเนมสเปซ

GAF

คลาส DateTimeTools ประกอบด้วย 6 เมธอดหลัก ทั้งหมดเป็นเมธอดแบบ Shared (static) ดังนี้

เมธอด	หน้าที่
เมธอด GetDayInMonth()	หาจำนวนวันของเดือนปัจจุบัน
เมธอด GetDayInYear()	หาจำนวนวันของปีปัจจุบัน
เมธอด GetDayName()	แสดงชื่อวันเป็นภาษาอังกฤษ
เมธอด GetThaiDayName()	แสดงชื่อวันเป็นภาษาไทย
เมธอด TelephoneTime()	คำนวณเวลาที่ใช้งานแบบโทรศัพท์มือถือ
เมธอด NextWorkingDay()	หาวันทำงานที่ใกล้เคียงที่สุด

เฉพาะภาษา VC# 2005 เพิ่มอีก 1 เมธอดคือ

เมธอด	หน้าที่
เมธอด DateDiff()	ทำหน้าที่หาผลต่างในหน่วยวัน มีหน้าที่เหมือนกับฟังก์ชัน DateDiff() ของ VB 2005

ตัวอย่างที่ 15-8 การใช้งาน DateTimeTools ให้คุณออกแบบฟอร์ม ดังรูปที่ 15-14

รูปที่ 15-14 ฟอร์มในขณะออกแบบ

การจัดรูปแบบให้กับคอนโทรล DateTimePicker ทั้ง 2 ตัว จะทำผ่านทางคุณสมบัติ Format ว่าต้องการแสดงวันที่ในรูปแบบสั้นหรือแบบยาว

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime (Form1.vb)

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim Today As DateTime
    Today = DateTime.Now
    dtpStart.Value = Today
    dtpStart.Format = DateTimePickerFormat.Short

    dtpEnd.Value = Today.AddMonths(1)
    dtpEnd.Format = DateTimePickerFormat.Long
End Sub
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    DateTime Today = DateTime.Now;
    dtpStart.Value = Today;
    dtpStart.Format = DateTimePickerFormat.Short;

    dtpEnd.Value = Today.AddMonths(1);
    dtpEnd.Format = DateTimePickerFormat.Long;
}
```

การหาจำนวนวันของเดือนและจำนวนวันของปีปัจจุบัน

รายละเอียดคลาส DateTimeTools เฉพาะเมธอด GetDayInMonth() และเมธอด GetDayInYear() แสดงดังนี้

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาจำนวนวันของเดือนและจำนวนวันของปีปัจจุบัน (GAF.DateTimeTools.vb)

```
Public Shared Function GetDayInMonth(ByVal ThaiDate As DateTime) As Integer
    Dim _dt As DateTime = ThaiDate
    Dim dtfInfo As DateTimeFormatInfo = DateTimeFormatInfo.CurrentInfo
    Dim _Year As Integer = Convert.ToInt32(_dt.ToString("yyyy", dtfInfo))
    Dim _Month As Integer = Convert.ToInt32(_dt.ToString("MM", dtfInfo))

    Dim _ThaiCalendar As New ThaiBuddhistCalendar
    Return _ThaiCalendar.GetDaysInMonth(_Year, _Month)
End Function

Public Shared Function GetDayInYear(ByVal ThaiDate As DateTime) As Integer
    Dim _dt As DateTime = ThaiDate
    Dim dtfInfo As DateTimeFormatInfo = DateTimeFormatInfo.CurrentInfo
    Dim _Year As Integer = Convert.ToInt32(_dt.ToString("yyyy", dtfInfo))

    Dim _ThaiCalendar As New ThaiBuddhistCalendar
    Return _ThaiCalendar.GetDaysInYear(_Year)
End Function
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาจำนวนวันของเดือนและจำนวนวันของปีปัจจุบัน (GAF.DateTimeTools.cs)

```
public static int GetDayInMonth(DateTime ThaiDate)
{
    DateTime _dt = ThaiDate;
    DateTimeFormatInfo dtfInfo = DateTimeFormatInfo.CurrentInfo;
    int _Year = Convert.ToInt32(_dt.ToString("yyyy", dtfInfo));
    int _Month = Convert.ToInt32(_dt.ToString("MM", dtfInfo));

    ThaiBuddhistCalendar _ThaiCalendar = new ThaiBuddhistCalendar();
    int _result=0;
    _result = _ThaiCalendar.GetDaysInMonth(_Year, _Month);
    return _result;
}

public static int GetDayInYear(DateTime ThaiDate)
{
    DateTime _dt = ThaiDate;
    DateTimeFormatInfo dtfInfo = DateTimeFormatInfo.CurrentInfo;
    int _Year = Convert.ToInt32(_dt.ToString("yyyy", dtfInfo));

    ThaiBuddhistCalendar _ThaiCalendar = new ThaiBuddhistCalendar();
    int _result = 0;
    _result = _ThaiCalendar.GetDaysInYear(_Year);
    return _result;
}
```

เมธอด GetDayInMonth() และเมธอด GetDayInYear() ต้องการพารามิเตอร์ 1 ตัวคือ ThaiDate มี Type เป็น DateTime รับวันที่ปัจจุบันรูปแบบภาษาไทย เพื่อหาจำนวนวันของเดือนปัจจุบัน และจำนวนวันของปีปัจจุบัน

ประเด็นที่น่าสนใจคือ ใน .NET Framework มีคลาสรับผิดชอบโดยเฉพาะกับวันที่รูปแบบภาษาไทยด้วยชื่อว่าคลาส ThaiBuddhistCalendar แยกเป็น 2 กรณีคือ

- เมธอด GetDayInMonth() ถ้าต้องการทราบจำนวนวันของเดือนปัจจุบันต้องส่งปี พ.ศ. (ฟิลด์ _Year) และเดือนปัจจุบัน (ฟิลด์ _Month) ให้กับเมธอด GetDaysInMonth() ของออบเจ็กต์ ThaiBuddhistCalendar ที่ชื่อว่า _ThaiCalendar

VB 2005

```
Dim _ThaiCalendar As New ThaiBuddhistCalendar
Return _ThaiCalendar.GetDaysInMonth(_Year, _Month)
```

VC# 2005

```
ThaiBuddhistCalendar _ThaiCalendar = new ThaiBuddhistCalendar();
int _result=0;
_result = _ThaiCalendar.GetDaysInMonth(_Year, _Month);
return _result;
```

- เมธอด GetDayInYear() แต่ถ้าต้องการหาจำนวนวันของปีปัจจุบันให้ส่งแค่ปี พ.ศ. เพียงอย่างเดียว

VB 2005

```
Dim _ThaiCalendar As New ThaiBuddhistCalendar
Return _ThaiCalendar.GetDaysInYear(_Year)
```

VC# 2005

```
ThaiBuddhistCalendar _ThaiCalendar = new ThaiBuddhistCalendar();
int _result = 0;
_result = _ThaiCalendar.GetDaysInYear(_Year);
return _result;
```

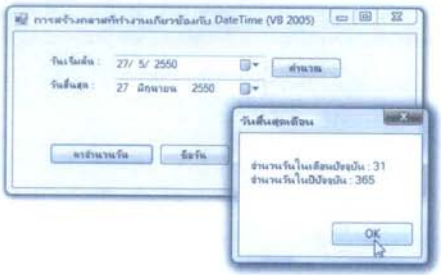
ส่วนการใช้งานเมธอด GetDayInMonth() และเมธอด GetDayInYear() อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาจำนวนวันของเดือนและจำนวนวันของปีปัจจุบัน (Form1.vb)

```
    Private Sub cmdCheckDayOfMonth_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdCheckDayOfMonth.Click
        Dim result As String = ""
        result = "จำนวนวันในเดือนปัจจุบัน : " & DateTimeTools.GetDayInMonth(dtpStart.Value) & Environment.NewLine
        result &= "จำนวนวันในปีปัจจุบัน : " & DateTimeTools.GetDayInYear(dtpStart.Value)
        MessageBox.Show(result.ToString(), "วันสิ้นสุดเดือน")
    End Sub
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาจำนวนวันของเดือนและจำนวนวันของปีปัจจุบัน (Form1.cs)

```
    private void cmdCheckDayOfMonth_Click(object sender, EventArgs e)
    {
        string result;
        result = "จำนวนวันในเดือนปัจจุบัน : " + DateTimeTools.GetDayInMonth(dtpStart.Value) + Environment.NewLine;
        result += "จำนวนวันในปีปัจจุบัน : " + DateTimeTools.GetDayInYear(dtpStart.Value);
        MessageBox.Show(result.ToString(), "วันสิ้นสุดเดือน");
    }
```



รูปที่ 15-15 ผลการหาจำนวนวันของเดือนและปีปัจจุบัน

จากรูปที่ 15-15 ผู้เขียนตรวจสอบวันที่ที่แสดงอยู่ในคอนโทรล DateTimePicker ที่ชื่อว่า dtpStart วันที่ตรวจสอบคือ 27 พฤษภาคม 2550 ผลการตรวจสอบที่ได้คือ เดือนพฤษภาคมมี 31 วัน ปี 2550 มี 365 วัน

การอ่านชื่อวันที่เป็นภาษาอังกฤษและภาษาไทย

รายละเอียดของคลาส DateTimeTools เฉพาะเมธอด GetDayName() และเมธอด GetThaiDayName() แสดงดังนี้

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการอ่านชื่อวันที่เป็นภาษาอังกฤษและภาษาไทย (GAF.DateTimeTools.vb)

```
Public Shared Function GetDayName(ByVal InputDate As DateTime) As String
    Return InputDate.DayOfWeek.ToString
End Function

Public Shared Function GetThaiDayName(ByVal InputDate As DateTime) As String
    Dim _result As String = ""
    _result = DateTimeTools.GetDayName(InputDate)
    Dim _ThaiName As String = ""

    Select Case _result
        Case "Monday"
            _ThaiName = "วันจันทร์"
        Case "Tuesday"
            _ThaiName = "วันอังคาร"
        Case "Wednesday"
            _ThaiName = "วันพุธ"
        Case "Thursday"
            _ThaiName = "วันพฤหัสบดี"
        Case "Friday"
            _ThaiName = "วันศุกร์"
        Case "Saturday"
            _ThaiName = "วันเสาร์"
        Case "Sunday"
            _ThaiName = "วันอาทิตย์"
    End Select

    Return _ThaiName
End Function
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการอ่านชื่อวันที่เป็นภาษาอังกฤษและภาษาไทย (GAF.DateTimeTools.cs)

```
public static string GetDayName(DateTime InputDate)
{
    return InputDate.DayOfWeek.ToString();
}

public static string GetThaiDayName(DateTime InputDate)
{
    string _result = "";
    _result =DateTimeTools.GetDayName(InputDate);
    string _ThaiName = "";

    switch (_result)
    {
        case "Monday" :
            _ThaiName = "วันจันทร์";
            break;
        case "Tuesday" :
            _ThaiName = "วันอังคาร";
            break;
        case "Wednesday":
            _ThaiName = "วันพุธ";
            break;
        case "Thursday":
            _ThaiName = "วันพฤหัสบดี";
            break;
        case "Friday":
            _ThaiName = "วันศุกร์";
            break;
        case "Saturday":
            _ThaiName = "วันเสาร์";
            break;
        case "Sunday":
            _ThaiName = "วันอาทิตย์";
            break;
    }
    return _ThaiName;
}
```

เมธอด GetDayName() ทำหน้าที่แสดงชื่อวันที่เป็นภาษาอังกฤษ ส่วนเมธอด GetThaiDayName() ทำหน้าที่แสดงชื่อวันที่เป็นภาษาไทย ต้องการพารามิเตอร์ 1 ตัวคือ InputDate มี Type เป็น DateTime หมายถึง วันที่ที่ต้องการอ่านชื่อภาษาอังกฤษหรือภาษาไทย ส่วนการใช้งานเมธอดทั้ง 2 อยู่ใน Form1

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการอ่านชื่อวันที่เป็นภาษาอังกฤษและภาษาไทย (Form1.vb)

```
    Private Sub cmdGetDayName_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdGetDayName.Click
        Dim result As String = ""
        result = "ชื่อวันสากล : " & DateTimeTools.GetDayName(dtpStart.Value) & Environment.NewLine
        result &= "ชื่อวันไทย : " & DateTimeTools.GetThaiDayName(dtpStart.Value)
        MessageBox.Show(result, "ชื่อวันที่ปัจจุบัน")
    End Sub
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการอ่านชื่อวันที่เป็นภาษาอังกฤษและภาษาไทย (Form1.cs)

```
    private void cmdGetDayName_Click(object sender, EventArgs e)
    {
        string result;
        result = "ชื่อวันสากล : " + DateTimeTools.GetDayName(dtpStart.Value) + Environment.NewLine;
        result += "ชื่อวันไทย : " + DateTimeTools.GetThaiDayName(dtpStart.Value);
        MessageBox.Show(result, "ชื่อวันที่ปัจจุบัน");
    }
```

รูปที่ 15-16 ผลการอ่านชื่อวันที่ภาษาอังกฤษและภาษาไทย

การคิดเวลาใช้งานแบบโทรศัพท์มือถือ

ในปัจจุบันเราคุ้นเคยกับโปรโมชันของมือถือที่ออกมาอย่างมากมาย อาจจะกล่าวได้ว่าแทบจะเป็นโปรโมชันรายวันเลยก็ได้ ที่น่าสนใจคือ ถ้าเราต้องการคิดเวลาแบบโปรโมชันของมือถือบ้างจะทำอย่างไร เช่น นาทีแรก 3 บาท นาทีต่อไปนาทีละ 25 สตางค์ อยู่ในความรับผิดชอบของเมธอด MobileTime()

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการคิดเวลาใช้งานแบบโทรศัพท์มือถือ (GAF.DateTimeTools.vb)

```
    Public Shared Function MobileTime(ByVal UseSecond As Integer, ByVal CostForFirstMinute As Double, ByVal CostForEveryMinute
As Double) As Double
        Dim _TotalCost As Double = 0.0

        If (UseSecond < 60) Then
            _TotalCost = CostForFirstMinute
        Else
            _TotalCost = CostForFirstMinute + (((UseSecond - 60) * (1.0 / 60)) * CostForEveryMinute)
        End If
        Return _TotalCost
    End Function
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการคิดเวลาใช้งานแบบโทรศัพท์มือถือ (GAF.DateTimeTools.cs)

```
    public static double MobileTime(int UseSecond,double CostForFirstMinute,double CostForEveryMinute)
    {
        double _TotalCost = 0.0;
        if (UseSecond < 60)
        {
            _TotalCost = CostForFirstMinute;
        }
        else
        {
            _TotalCost = CostForFirstMinute + (((UseSecond - 60) * (1.0 / 60)) * CostForEveryMinute);
        }
        return _TotalCost;
    }
```

เมธอด MobileTime() ต้องการพารามิเตอร์ 3 ตัวคือ

- พารามิเตอร์ UseSecond หมายถึง เวลาที่ใช้มีหน่วยเป็นวินาที
- พารามิเตอร์ CostForFirstMinute หมายถึง อัตราค่าโทรสำหรับนาทีแรก
- พารามิเตอร์ CostForEveryMinute หมายถึง อัตราค่าโทรสำหรับนาทีที่ 2 เป็นต้นไป

วิธีคิดคือ ถ้าโทรน้อยกว่า 1 นาทีก็จะคิดเต็ม 1 นาทีแรก แต่ถ้าโทรตั้งแต่ 2 นาทีขึ้นไป จะคิดค่าโทรตามวินาทีจริง การใช้งานอยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการคิดเวลาใช้งานแบบโทรศัพท์มือถือ (Form1.vb)

```
    Private Sub cmdTelTime_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdTelTime.Click
        Dim result As Double
        result = DateTimeTools.MobileTime(60, 5, 0.25)
        MessageBox.Show(result.ToString(), "ค่าโทรศัพท์")
    End Sub
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการคิดเวลาใช้งานแบบโทรศัพท์มือถือ (Form1.cs)

```
private void cmdTelTime_Click(object sender, EventArgs e)
{
    double result;
    result = DateTimeTools.MobileTime(60, 5, 0.25);
    MessageBox.Show(result.ToString(), "ค่าโทรศัพท์");
}
```

รูปที่ 15-17
ผลการคิดค่าโทรศัพท์ตามจริง

โค้ดข้างต้นหมายความว่าถ้าโทร 1 นาที (60) อัตราค่าโทรนาทีแรก 5 บาท (5) ส่วนนาทีต่อไป 25 สตางค์ (0.25) สมมติว่าเปลี่ยนอัตราค่าโทรใหม่เป็นถ้าโทร 1 นาทีครึ่ง (90) อัตราค่าโทรนาทีแรก 3 บาท (3) นาทีต่อไป 25 สตางค์ (0.25) ผลที่ได้แสดงดังรูปที่ 15-18

VB 2005

```
result = DateTimeTools.MobileTime(90, 3, 0.25)
MessageBox.Show(result.ToString(), "ค่าโทรศัพท์")
```

VC# 2005

```
result = DateTimeTools.MobileTime(90, 3, 0.25);
MessageBox.Show(result.ToString(), "ค่าโทรศัพท์");
```

รูปที่ 15-18
กรณีเปลี่ยนอัตราค่าโทรใหม่

ค่าโทรที่คิดได้คือ นาทีแรก 3 บาท ส่วนนาทีที่ 2 โทรเพียง 30 วินาที จึงคิดครึ่งหนึ่งของ 0.25 สตางค์ คือ 0.125 สตางค์ อาจจะมีการตรวจสอบเพิ่มเติมเพื่อปัดเศษสตางค์ให้ตรงกับอัตราที่ลูกค้าสามารถจ่ายได้จริง

การหาวันทำงานถัดไป

ในบางครั้งถ้ามีการคำนวณวันที่แล้วเกิดกรณีว่าวันที่ดังกล่าวไปตรงกับวันหยุด อาจจะต้องมีการเลื่อนวันที่ไปยังวันทำงานที่ใกล้เคียงที่สุด สมมติว่าอีก 3 วันข้างหน้าเป็นวันรับสินค้าที่สั่งซื้อ แต่วันดังกล่าวตรงกับวันอาทิตย์ จึงต้องเลื่อนวันรับสินค้าไปเป็นวันจันทร์ เพื่อให้ตรงกับความเป็นจริงในทางปฏิบัติ ซึ่งอยู่ในความรับผิดชอบของเมธอด NextWorkingDay()

โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาวันทำงานถัดไป (GAF.DateTimeTools.vb)

```
Public Shared Function NextWorkingDay(ByVal CurrnetDate As DateTime) As DateTime
Dim ReturnDate As DateTime = CurrentDate
    If (CurrentDate.DayOfWeek = DayOfWeek.Sunday) Then
        Return CurrentDate.AddDays(1)
    End If
    If (CurrentDate.DayOfWeek = DayOfWeek.Saturday) Then
        ReturnDate = CurrentDate.AddDays(2)
    End If
    Return ReturnDate
End Function
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาวันทำงานถัดไป (GAF.DateTimeTools.cs)

```
public static DateTime NextWorkingDay(DateTime CurrentDate)
{
    DateTime ReturnDate= CurrentDate;
    if (CurrentDate.DayOfWeek== DayOfWeek.Sunday)
    {
        ReturnDate = CurrentDate.AddDays(1);
    }
    else if (CurrentDate.DayOfWeek == DayOfWeek.Saturday)
    {
        ReturnDate = CurrentDate.AddDays(2);
    }
        return ReturnDate;
}
```

เมธอด NextWorkingDay() ต้องการพารามิเตอร์ 1 ตัวคือ CurrentDate มี Type เป็น DateTime หมายถึง วันที่ที่ต้องการตรวจสอบ วิธีการก็คือ

- ถ้าวันที่ส่งเข้ามาตรงกับวันอาทิตย์ให้บวกเพิ่มอีก 1 วัน
- แต่ถ้าวันที่ส่งเข้ามาตรงกับวันเสาร์ให้บวกเพิ่มอีก 2 วัน

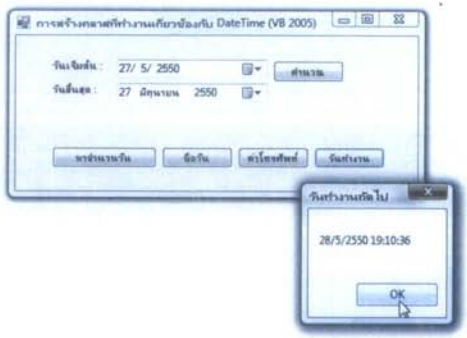
โค้ด VB 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาวันทำงานถัดไป (Form1.vb)

```
    Private Sub cmdWorkingDay_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdWorkingDay.Click
        Dim result As DateTime
        result = DateTimeTools.NextWorkingDay(dtpStart.Value)
        MessageBox.Show(result.ToString(), "วันทำงานถัดไป")
    End Sub
```

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาวันทำงานถัดไป (Form1.cs)

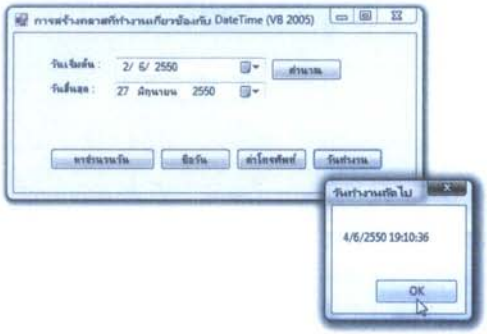
```
    private void cmdWorkingDay_Click(object sender, EventArgs e)
    {
        DateTime result;
        result = DateTimeTools.NextWorkingDay(dtpStart.Value);
        MessageBox.Show(result.ToString(), "วันทำงานถัดไป");
    }
```

รูปที่ 15-19
กรณีตรวจสอบวันอาทิตย์



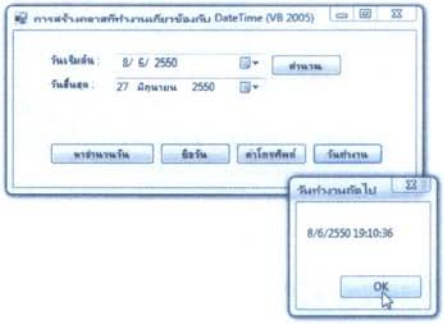
จากรูปที่ 15-19 กรณีตรวจสอบวันอาทิตย์ที่ 27 พฤษภาคม พ.ศ. 2550 วันทำงานที่ใกล้เคียงที่สุด คือ วันจันทร์ที่ 28 พฤษภาคม 2550

รูปที่ 15-20
กรณีตรวจสอบวันเสาร์



จากรูปที่ 15-20 กรณีตรวจสอบวันเสาร์ที่ 2 มิถุนายน พ.ศ. 2550 วันทำงานที่ใกล้เคียงที่สุดคือวันจันทร์ที่ 4 มิถุนายน พ.ศ. 2550

รูปที่ 15-21
กรณีตรวจสอบวันทำงานปกติ



จากรูปที่ 15-21 กรณีวันที่ตรวจสอบเป็นวันทำงานปกติ ก็จะคืนค่าเป็นวันที่รับเข้ามานั่นเอง

ในส่วนของเมธอด DateDiff() มีเฉพาะใน VC# 2005 ส่วน VB 2005 มีฟังก์ชัน DateDiff() อยู่แล้ว เมธอด DateDiff() ต้องการพารามิเตอร์ 2 ตัวคือ StartDate และ EndDate มี Type เป็น DateTime หมายถึงวันเริ่มต้นและวันสิ้นสุดที่ต้องการหาผลต่างในหน่วยวัน

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาวันทำงานถัดไป (GAF.DateTimeTools.cs)

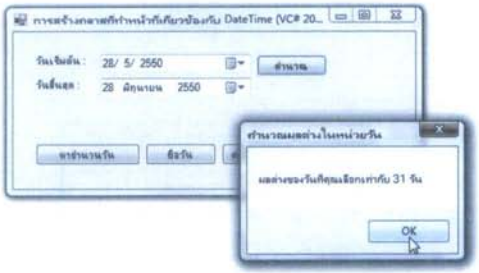
```
public static int DateDiff(DateTime StartDate, DateTime EndDate)
{
    TimeSpan _ts = EndDate.Subtract(StartDate);
    return _ts.Days;
}
```

การหาผลต่างระหว่างวันผลที่ได้เป็นออบเจ็กต์ TimeSpan เก็บไว้ในฟิลด์ _ts ก็จะสั่งให้คืนค่าในหน่วยวัน (_ts.Days) ส่วนการใช้งานอยู่ใน Form1

โค้ด VC# 2005 ที่ 15-8 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ DateTime เฉพาะการหาวันทำงานถัดไป (Form1.cs)

```
private void cmdCalculate_Click(object sender, EventArgs e)
{
    int NumberDay = 0;
    NumberDay = DateTimeTools.DateDiff(dtpStart.Value , dtpEnd.Value);
    MessageBox.Show("ผลต่างของวันที่คุณเลือกเท่ากับ " + NumberDay.ToString() + " วัน", "คำนวณผลต่างในหน่วยวัน");
}
```

รูปที่ 15-22
การหาผลต่าง
จำนวนวัน



สรุปท้ายบท

การใช้งานข้อมูลชนิด DateTime ที่ผู้เขียนเลือกนำมาเสนอ เป็นการทำงานที่คุณสามารถพบเห็นในการพัฒนาแอพพลิเคชันด้านธุรกิจต่างๆ

Advance Visual Studio 2005

CHAPTER ▸ 16

การจัดการ String

บทนำ

String (string) และข้อมูลชนิด Char (char) เป็นชนิดข้อมูลอีกประเภทหนึ่งที่ใกล้ชิดเราเป็นอย่างยิ่ง โดยผู้เขียนจะเลือกนำเสนอเฉพาะการนำไปใช้งานเท่านั้น

การต่อสตริงโดยอาศัยออบเจ็กต์ StringBuilder

โดยปกติแล้วการต่อสตริงใช้ในกรณีที่เราต้องการเก็บข้อความที่มีความยาว เช่น การสร้างชุดคำสั่ง SQL เป็นต้น ในภาษา VB 2005 ใช้เครื่องหมาย & ส่วนภาษา VC# 2005 ใช้เครื่องหมาย + เป็นวิธีการที่ใช้กันอยู่โดยทั่วไป

ไมโครซอฟท์แนะนำว่าการต่อสตริงขอให้ใช้ออบเจ็กต์ StringBuilder เข้ามาทำหน้าที่แทนการใช้เครื่องหมาย & ในภาษา VB 2005 หรือเครื่องหมาย + ในภาษา VC# 2005 ผู้เขียนนำข้อแนะนำดังกล่าวมาทดสอบเป็นตัวอย่างที่ 16-1 การต่อสตริงโดยอาศัยออบเจ็กต์ StringBuilder ประกอบด้วยเทคนิคดังนี้

- การใช้ออบเจ็กต์ StringBuilder ทำหน้าที่ต่อสตริง
- การใช้ออบเจ็กต์ StopWatch ทำหน้าที่เป็นนาฬิกาจับเวลา

ให้คุณออกแบบฟอร์ม ดังรูปที่ 16-1

รูปที่ 16-1 ฟอร์มในขณะออกแบบ

จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 ที่ 16-1 การต่อสตริงโดยอาศัยออบเจ็กต์ StringBuilder (Form1.vb)

```
Option Explicit On                                  ให้มีการประกาศตัวแปร ก่อนการใช้งาน
Option Strict On                                    ให้เคร่งครัดต่อการเปลี่ยนแปลงชนิดของข้อมูล
Imports System.Text                                 'เนมสเปซของออบเจ็กต์ StringBuilder

Public Class Form1

'เหตุการณ์คลิกที่ปุ่ม ใช้เครื่องหมาย &
    Private Sub cmdNormal_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdNormal.Click
        Dim sw As New Stopwatch()                   'ตัวแปรออบเจ็กต์ Stopwatch ที่ชื่อว่า sw
        sw.Start()                                  'เริ่มต้นจับเวลา

        Dim i As Integer                            'ตัวแปรวนลูป
        Dim strTest As String = ""                  'ตัวแปรเก็บข้อความทดสอบ
        For i = 1 To 10000                          'วนลูป 1 หมื่นรอบ
            strTest &= i.ToString()                 'ต่อข้อความเก็บไว้ในตัวแปร strTest
        Next

        sw.Stop()                                   'หยุดจับเวลา

'แสดงเวลาทำงานที่ได้
        lblResult.Text = sw.ElapsedMilliseconds & " ms."
    End Sub

'เหตุการณ์คลิกที่ปุ่ม ใช้ StringBuilder
        Private Sub cmdStringBuilder_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdStringBuilder.Click
        Dim sw As New Stopwatch()                   'ตัวแปรออบเจ็กต์ Stopwatch ที่ชื่อว่า sw
        sw.Start()                                  'เริ่มต้นจับเวลา

        Dim i As Integer                            'ตัวแปรวนลูป i
        Dim sb As New StringBuilder()               'ตัวแปรออบเจ็กต์ StringBuilder ที่ชื่อว่า sb
        For i = 1 To 10000                          'วนลูป 1 หมื่นรอบ
            sb.Append(i.ToString)                   'ต่อข้อความเก็บไว้ในออบเจ็กต์ sb
        Next

        sw.Stop()                                   'หยุดจับเวลา

'แสดงเวลาทำงานที่ได้
        lblResult.Text = sw.ElapsedMilliseconds & " ms."
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 16-1 การต่อสตริงโดยอาศัยออบเจ็กต์ StringBuilder (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;                                  //เนมสเปซของออบเจ็กต์ StringBuilder
using System.Windows.Forms;
using System.Diagnostics;                           //เนมสเปซของออบเจ็กต์ Stopwatch

namespace UsingStringBuilder
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

//เหตุการณ์คลิกที่ปุ่ม ใช้เครื่องหมาย +
        private void cmdNormal_Click(object sender, EventArgs e)
        {
            Stopwatch sw = new Stopwatch();         //ตัวแปรออบเจ็กต์ Stopwatch ที่ชื่อว่า sw
            sw.Start();                             //เริ่มจับเวลา

            string strTest = "";                    //ตัวแปรเก็บข้อความทดสอบ
            for (int i = 1; i <= 10000; i++)        //วนลูป 1 หมื่นรอบ
            {
                strTest += i.ToString();            //ต่อข้อความ เก็บไว้ในตัวแปร strTest
            }

            sw.Stop();                              //หยุดจับเวลา
//แสดงเวลาทำงานที่ได้
            lblResult.Text = sw.ElapsedMilliseconds + " ms.";
        }

//เหตุการณ์คลิกที่ปุ่ม ใช้ StringBuilder
        private void cmdStringBuilder_Click(object sender, EventArgs e)
        {
            Stopwatch sw = new Stopwatch();         //ตัวแปรออบเจ็กต์ Stopwatch ที่ชื่อว่า sw
            sw.Start();                             //เริ่มต้นจับเวลา

            StringBuilder sb = new StringBuilder(); //ตัวแปรออบเจ็กต์ StringBuilder ที่ชื่อว่า sb
            for (int i = 1; i <= 10000; i++)        //วนลูป 1 หมื่นรอบ
            {
                sb.Append(i.ToString());            //ต่อข้อความเก็บไว้ในออบเจ็กต์ sb
            }

            sw.Stop();                              //หยุดจับเวลา
//แสดงเวลาทำงานที่ได้
            lblResult.Text = sw.ElapsedMilliseconds + " ms.";
        }
    }
}
```

รูปที่ 16-2 ผลการรันตัวอย่างที่ 16-1

จากรูปที่ 16-2 พบว่าความเร็วของการต่อสตริงโดยอาศัยออบเจ็กต์ StringBuilder เร็วกว่าการใช้เครื่องหมาย & ในภาษา VB 2005 หรือเครื่องหมาย + ในภาษา VC# 2005 ดังนั้น การต่อข้อความต่างๆ เราควรใช้วิธีการต่อสตริงแบบอาศัยออบเจ็กต์ StringBuilder จะมีความเร็วในการทำงานมากกว่า

การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String

ผู้ช่วยเหลือด้าน String ชื่อว่าคลาส StringTools เก็บอยู่ในเนมสเปซ

GAF

สำหรับคลาส StringTools ระหว่าง VB 2005 กับ VC# 2005 แตกต่างกัน เพราะว่าภาษา VB 2005 มีฟังก์ชันสำเร็จรูปที่เกี่ยวข้องกับ String อยู่พอสมควร ดังนั้น ผู้เขียนจึงกำหนดให้คลาส StringTools ใช้คุณสมบัติของการทำ Partial Class เข้ามาช่วยกล่าวคือ

ในไฟล์ GAF.StringTools.Part1.cs ของ VC# 2005 จะเก็บเมธอดที่มีหน้าที่เหมือนกับฟังก์ชันของ VB 2005 ส่วนไฟล์ GAF.StringTools.Part2.vb และ GAF.StringTools.Part2.cs เก็บเมธอดที่สร้างขึ้นมาใหม่ทำหน้าที่จัดการเกี่ยวกับ String Type สำหรับเมธอดที่สร้างขึ้นมาเพื่อทำหน้าที่เหมือนกับฟังก์ชันของ VB 2005 ประกอบด้วย 5 เมธอด เป็นเมธอดแบบ static ทั้งหมดคือ

เมธอด	หน้าที่
เมธอด Left()	ตัดข้อความนับจากซ้ายมือ
เมธอด Right()	ตัดข้อความนับจากขวามือ
เมธอด Mid()	ตัดข้อความจากตำแหน่งที่ต้องการมี 2 แบบคือ ● กำหนดตำแหน่งที่ต้องการตัด ● กำหนดตำแหน่งที่ต้องการตัด และกำหนดจำนวนตัวอักษรที่ต้องการตัด
เมธอด ReverseString()	สลับตำแหน่งข้อความจากด้านหน้าไปด้านหลัง
เมธอด InStr()	ค้นหาข้อความกล่าวคือ ● คืนค่า true เมื่อพบข้อความที่ค้นหา ● คืนค่า false เมื่อไม่พบข้อความที่ค้นหา

การตัดข้อความจากทางซ้าย, ขวา และกำหนดตำแหน่งที่ตัด

ให้เขียนโค้ดดังต่อไปนี้

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการตัดข้อความจากทางซ้าย, ขวา และกำหนดตำแหน่งที่ตัด (GAF.StringTools.Part1.cs)

```
public static string Left(string str, int Length)
{
    if (Length > 0 && str.Length >= Length)
    {
        return str.Substring(0, Length);
    }
    else
    {
        return str;
    }
}

public static string Right(string str, int Length)
{
    if (Length > 0 && str.Length >= Length)
    {
        return str.Substring(str.Length - Length, Length);
    }
    else
    {
        return str;
    }
}
```

เมธอด Left() และเมธอด Right() ต้องการพารามิเตอร์ 2 ตัวคือ

- พารามิเตอร์ str มี Type เป็น string หมายถึง ข้อความที่ต้องการตัด
- พารามิเตอร์ Length มี Type เป็น int หมายถึง จำนวนตัวอักษรที่ต้องการตัด

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการตัดข้อความจากทางซ้าย, ขวา และกำหนดตำแหน่งที่ตัด (GAF.StringTools.Part1.cs)

```
public static string Mid(string str, int strStart)
{
    return str.Substring(strStart);
}

public static string Mid(string str, int strStart, int strLength)
{
    return str.Substring(strStart, strLength);
}
```

ส่วนเมธอด Mid() มี 2 แบบ ต้องการพารามิเตอร์ 2 หรือ 3 ตัว เป็นการทำ Overload Method เพราะว่าจำนวนพารามิเตอร์แตกต่างกันนั่นเอง (มี Signature ต่างกัน) กล่าวคือ

- พารามิเตอร์ str มี Type เป็น string หมายถึง ข้อความที่ต้องการตัด
- พารามิเตอร์ strStart มี Type เป็น int หมายถึง ตำแหน่งเริ่มต้นที่ต้องการตัด
- พารามิเตอร์ strLength มี Type เป็น int หมายถึง จำนวนตัวอักษรที่ต้องการตัด

ทั้ง 4 เมธอดอาศัยการเลือกข้อความที่ต้องการผ่านทางเมธอด SubString() ของ String Type เป็นหลัก จะแตกต่างกันก็แต่เพียงวิธีการเลือกเท่านั้น

การสลับตำแหน่งข้อความจากด้านหน้าไปด้านหลัง

ให้เขียนโค้ดดังต่อไปนี้

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการสลับตำแหน่งข้อความจากด้านหน้าไปด้านหลัง (GAF.StringTools.Part1.cs)

```
public static string ReverseString(string str)
{
    char[] _c = str.ToCharArray();
    Array.Reverse(_c);

    StringBuilder _sb = new StringBuilder();
    _sb.Remove(0, _sb.Length);
    for (int _i = 0; _i < _c.Length; _i++)
    {
        _sb.Append(_c[_i]);
    }
    return _sb.ToString();
}
```

หลักการสลับตำแหน่งจากหน้าไปหลังคือ เราจะแปลงข้อความ string ให้เป็นอาร์เรย์ของ char (อาร์เรย์ของตัวอักษร) โดยอาศัยเมธอด ToCharArray() ก่อน

จากนั้นใช้เมธอด Reverse() ของคลาส Array เข้ามาทำหน้าที่สลับตำแหน่งของอาร์เรย์ char ท้ายที่สุดประกอบตัวอักษร (char) เป็นข้อความ (string) โดยอาศัยออบเจ็กต์ StringBuilder ที่ชื่อว่า _sb

การค้นหาข้อความ

ให้เขียนโค้ดดังต่อไปนี้

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการค้นหาข้อความ (GAF.StringTools.Part1.cs)

```
public static bool InStr(string str, string strToSearch,bool IsCaseSensitive)
{
    if (IsCaseSensitive == false)
    {
        if (str.Contains(strToSearch.ToUpper()))
        {
            return true;
        }

        if (str.Contains(strToSearch.ToLower()))
        {
            return true;
        }
    }

    return str.Contains(strToSearch);
}
```

เมธอด InStr() ต้องการพารามิเตอร์ 3 ตัวคือ

- พารามิเตอร์ str มี Type เป็น string หมายถึง ข้อความที่จะค้นหา
- พารามิเตอร์ strToSearch มี Type เป็น string หมายถึง ข้อความที่ต้องการหา
- พารามิเตอร์ IsCaseSensitive มี Type เป็น bool หมายถึง ต้องการคิดตัวอักษรเล็ก-ใหญ่แตกต่างกันหรือไม่

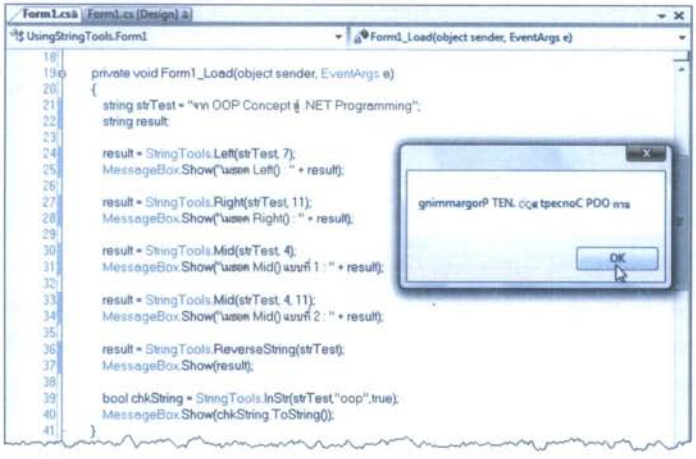
หลักการคือ อาศัยเมธอด Contains() ของ String Type ทำหน้าที่ตรวจสอบว่ามีข้อความที่สนใจอยู่หรือไม่ ปัญหาอยู่ที่การค้นหาข้อความที่เป็นภาษาอังกฤษกล่าวคือ

ถ้าไม่คิดตัวอักษรเล็ก-ใหญ่ (พารามิเตอร์ IsCaseSensitive = false) คุณต้องตรวจสอบทั้งตัวอักษรใหญ่ (เรียกใช้เมธอด ToUpper()) และตัวอักษรเล็ก (เรียกใช้เมธอด ToLower()) ด้วย แต่ถ้าคิดตัวอักษรเล็ก-ใหญ่แตกต่างกัน ก็จะคืนค่าที่ได้จากเมธอด Contains() โดยตรง ส่วนการใช้งานอยู่ใน Form1

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการค้นหาข้อความ (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    string strTest = "จาก OOP Concept สู่ .NET Programming";
    string result;

    result = StringTools.Left(strTest, 7);
    MessageBox.Show("เมธอด Left() : " + result);

    result = StringTools.Right(strTest, 11);
    MessageBox.Show("เมธอด Right() : " + result);

    result = StringTools.Mid(strTest, 4);
    MessageBox.Show("เมธอด Mid() แบบที่ 1 : " + result);

    result = StringTools.Mid(strTest, 4, 11);
    MessageBox.Show("เมธอด Mid() แบบที่ 2 : " + result);

    result = StringTools.ReverseString(strTest);
    MessageBox.Show(result);

    bool chkString = StringTools.InStr(strTest,"oop",true);
    MessageBox.Show(chkString.ToString());
}
```

รูปที่ 16-3 ผลการลลับตำแหน่งของเมธอด ReverseString()



ผู้เขียนสร้างข้อความตัวอย่างขึ้นมาเพื่อทดสอบใช้งานเมธอดทั้ง 5 ของคลาส StringTools

การสร้างตัวอักษรแบบสุ่ม

ใน .NET Framework ไม่มีคลาสที่ทำหน้าที่สุ่มตัวอักษรโดยตรง มีแต่สุ่มตัวเลข โดยหลักการทำงานก็จะอาศัยองค์ประกอบ 2 ส่วนคือ

1. กำหนดขอบเขตตัวอักษรที่คุณต้องการสุ่ม ให้อยู่รวมกันเป็นข้อความ String (string)
2. ให้แปลง String (string) จากข้อที่ 1 เป็นอาร์เรย์ของตัวอักษร หรืออาร์เรย์ของ Char (char) แล้วใช้การสุ่มตัวเลข เพื่อหาลำดับสมาชิกที่อยู่ในอาร์เรย์ Char (char) นั่นเอง

จากวิธีคิดข้างต้นสามารถแปลงเป็นโค้ดดังนี้

โค้ด VB 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการสร้างตัวอักษรแบบสุ่ม (GAF.StringTools.Part2.vb)

```
    Public Shared Function RandomString(ByVal Length As Integer, ByVal IsIncludeThaiCharacter As Boolean, ByVal
IsIncludeSpecialCharacter As Boolean) As String
        Dim _sb As New StringBuilder()
        _sb.Remove(0, _sb.Length)

        Dim _Eng As String = ""
        _Eng = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

        Dim _Thai As String = ""
        _Thai = "๐๑๒๓๔๕๖๗๘๙กขคฆงจฉชฌญฎฏฐฑฒณดตถทธนบปผฝพฟภมยรฤลฦวศษสหฬอฮ"

        Dim _Special As String = ""
        _Special = "!@#$%^&*?.|"

        _sb.Append(_Eng)
        If IsIncludeThaiCharacter Then
            _sb.Append(_Thai)
        End If

        If IsIncludeSpecialCharacter Then
            _sb.Append(_Special)
        End If

        Dim _TotalString As String = _sb.ToString()
        Dim _c As Char() = _TotalString.ToCharArray()
        Dim _TotalLength As Integer = _TotalString.Length

        Dim _rd As New Random()
        _sb.Remove(0, _sb.Length)

        Dim _i As Integer = 0
        For _i = 0 To Length - 1
            _sb.Append(_c(_rd.Next(0, _TotalLength)))
        Next

        Dim _result As String = _sb.ToString()
        Return _result
    End Function
```

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการสร้างตัวอักษรแบบสุ่ม (GAF.StringTools.Part2.cs)

```
public static string RandomString(int Length, bool IsIncludeThaiCharacter, bool IsIncludeSpecialCharacter)
{
    StringBuilder _sb = new StringBuilder();
    _sb.Remove(0, _sb.Length);

    string _Eng = "";
    _Eng = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

    string _Thai = "";
    _Thai = "๐๑๒๓๔๕๖๗๘๙กขคฆงจฉชฌญฎฏฐฑฒณดตถทธนบปผฝพฟภมยรฤลฦวศษสหฬอฮ";

    string _Special = "";
    _Special = "!@#$%^&*?.|";

    _sb.Append(_Eng);
    if (IsIncludeThaiCharacter)
    {
        _sb.Append(_Thai);
    }

    if (IsIncludeSpecialCharacter)
    {
        _sb.Append(_Special);
    }

    string _TotalString = _sb.ToString();
    char[] _c = _TotalString.ToCharArray();
    int _TotalLength = _TotalString.Length;

    Random _rd = new Random();
    _sb.Remove(0, _sb.Length);

    for (int _i = 0; _i < Length; _i++)
    {
        _sb.Append(_c[_rd.Next(0, _TotalLength)]);
    }

    string _result = _sb.ToString();
    return _result;
}
```

เมธอด RandomString() ต้องการพารามิเตอร์ 3 ตัวคือ

- พารามิเตอร์ Length มี Type เป็น Integer (int) หมายถึง จำนวนตัวอักษรที่ต้องการสร้างขึ้นมา
- พารามิเตอร์ IsIncludeThaiCharacter มี Type เป็น Boolean (bool) หมายถึง ตัวอักษรที่สุ่มขึ้นมา รวมตัวอักษรภาษาไทยด้วยหรือไม่
- พารามิเตอร์ IsIncludeSpecialCharacter มี Type เป็น Boolean (bool) หมายถึง ตัวอักษรที่สุ่มขึ้นมา รวมตัวอักษรพิเศษหรือไม่

ผู้เขียนแบ่งตัวอักษรออกเป็น 3 กลุ่มคือ

1. ตัวอักษรภาษาอังกฤษเป็นตัวอักษรหลัก ประกอบด้วย 0-9, A-Z และ a-z เก็บไว้ในฟิลด์ _Eng
2. ตัวอักษรภาษาไทยเป็นทางเลือก ประกอบด้วย ๐-๙ และ ก-ฮ เก็บไว้ในฟิลด์ _Thai
3. ตัวอักษรพิเศษต่างๆ ที่อยู่บนคีย์บอร์ดเป็นทางเลือกเช่นกัน หมายถึง กลุ่มตัวอักษรที่อยู่นอกเหนือจาก 2 กลุ่มข้างต้นเก็บไว้ในฟิลด์ _Special

สำหรับตัวอักษรของกลุ่มที่ 3 คุณจะกำหนดตัวอักษรอะไรก็แล้วแต่ความต้องการของคุณ

VB 2005

```
Dim _Eng As String = ""
_Eng = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

Dim _Thai As String = ""
_Thai = "๐๑๒๓๔๕๖๗๘๙กขคฆงจฉชฌญฏฐฑฒณฌณดตถทธนบปผฝพฟภมยรฤลฦวศษสหฬอฮ"

Dim _Special As String = ""
Special = "!@#$%^&*?.|"
```

VC# 2005

```
string _Eng = "";
_Eng = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

string _Thai = "";
_Thai = "๐๑๒๓๔๕๖๗๘๙กขคฆงจฉชฌญฏฐฑฒณฌณดตถทธนบปผฝพฟภมยรฤลฦวศษสหฬอฮ";

string _Special = "";
Special = "!@#$%^&*?.|";
```

ข้อควรระวังก็คือ ตัวอักษรที่มีลักษณะคล้ายๆ กัน เช่น ตัวอักษรโอเล็ก (o) กับเลขศูนย์ของภาษาไทย (๐) เห็นได้ว่าผู้เขียนตัดเลขศูนย์ภาษาไทยออก แล้วใช้เลขศูนย์อาราบิกแทน (0) เพื่อป้องกันไม่ให้ตัวอักษรที่สุ่มขึ้นมาสร้างความสับสนนั่นเอง

ตัวอักษรที่ต้องการสุ่มต้องมีตัวอักษรภาษาอังกฤษ (ฟิลด์ _Eng) แน่นอน ส่วนตัวอักษรภาษาไทย (ฟิลด์ _Thai) หรือตัวอักษรพิเศษ (ฟิลด์ _Special) ก็แล้วแต่ค่าของพารามิเตอร์ IsIncludeThaiCharacter และพารามิเตอร์ IsIncludeSpecialCharacter ว่ามีค่าเป็น True (true) หรือไม่

VB 2005

```
_sb.Append(_Eng)
If IsIncludeThaiCharacter Then
    _sb.Append(_Thai)
End If

If IsIncludeSpecialCharacter Then
    _sb.Append(_SpecialCharacter)
End If
```

VC# 2005

```
_sb.Append(_Eng);
if (IsIncludeThaiCharacter)
{
    _sb.Append(_Thai);
}

if (IsIncludeSpecialCharacter)
{
    _sb.Append(_Special);
}
```

ขอบเขตตัวอักษรทั้งหมดที่สามารถสุ่มได้เก็บไว้ที่ฟิลด์ _TotalString ให้แปลงเป็นอาร์เรย์ของ Char (char) เก็บไว้ในฟิลด์อาร์เรย์ที่ชื่อว่า _c และอ่านจำนวนตัวอักษรทั้งหมดที่สามารถสุ่มได้เก็บไว้ที่ฟิลด์ _TotalLength

VB 2005

```
Dim _TotalString As String = _sb.ToString()
Dim _c As Char() = _TotalString.ToCharArray()
Dim _TotalLength As Integer = _TotalString.Length
```

VC# 2005

```
string _TotalString = _sb.ToString();
char[] _c = _TotalString.ToCharArray();
int _TotalLength = _TotalString.Length;
```

ต่อมาสร้างออบเจ็กต์ Random ที่ชื่อว่า _rd ทำหน้าที่สุ่มตัวเลขที่มีขอบเขตไม่เกินค่าที่เก็บอยู่ในฟิลด์ _TotalLength สั่งให้วนลูปตามจำนวนรอบของพารามิเตอร์ Length ในแต่ละรอบของการวนลูป ก็จะได้ตัวอักษรที่อยู่ในอาร์เรย์ _c แบบสุ่มนั่นเอง สะสมตัวอักษรที่ได้ไว้ในฟิลด์ _result และคืนค่าฟิลด์นี้ออกไป

VB 2005

```
Dim _rd As New Random()
_sb.Remove(0, _sb.Length)

Dim _i As Integer = 0
For _i = 0 To Length - 1
    _sb.Append(_c(_rd.Next(0, _TotalLength)))
Next

Dim _result As String = _sb.ToString()
Return _result
```

VC# 2005

```
Random _rd = new Random();
_sb.Remove(0, _sb.Length);

for (int _i = 0; _i < Length; _i++)
{
    _sb.Append(_c[_rd.Next(0, _TotalLength)]);
}

string _result = _sb.ToString();
return _result;
```

การสุ่มค่า GUID

ยังมีการสุ่มอีกแบบหนึ่งที่เรียกว่า Globally Unique Identifier (GUID) หรือค่าที่มีโอกาสซ้ำกันน้อยมาก อยู่ในความรับผิดชอบของเมธอด GetGUID()

โค้ด VB 2005 และ VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการสร้างตัวอักษรแบบสุ่ม

VB 2005 (GAF.StringTools.Part2.vb)

```
Public Shared Function GetGUID() As Guid
    Dim _g As Guid = Guid.NewGuid()
    Return _g
End Function
```

VC# 2005 (GAF.StringTools.Part2.cs)

```
public static Guid GetGUID()
{
    Guid _g = Guid.NewGuid();
    return _g;
}
```

ส่วนการใช้งานอยู่ใน Form1 แสดงดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการสร้างตัวอักษรแบบสุ่ม (Form1.vb)

```
    Private Sub cmdStringRandom_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdStringRandom.Click
        Dim password As String = StringTools.RandomString(8, False, True)
        MessageBox.Show(password, "สตริงที่สุ่มได้")
    End Sub

    Private Sub cmdGuid_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdGuid.Click
        Dim result As String = StringTools.GetGUID.ToString
        MessageBox.Show(result, "Guid ที่สุ่มได้")
    End Sub
```

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการสร้างตัวอักษรแบบสุ่ม (Form1.cs)

```
    private void cmdStringRandom_Click(object sender, EventArgs e)
    {
        string password = StringTools.RandomString(8, false , true);
        MessageBox.Show(password, "สตริงที่สุ่มได้");
    }

    private void cmdGuid_Click(object sender, EventArgs e)
    {
        string result = StringTools.GetGUID().ToString();
        MessageBox.Show(result, "Guid ที่สุ่มได้");
    }
```

รูปที่ 16-4
ผลการทำงานของเมธอด RandomString()

จากรูปที่ 16-4 เกิดจากต้องการตัวอักษร 8 ตัว (8) ไม่รวมตัวอักษรภาษาไทย (False) แต่รวมตัวอักษรพิเศษ (True)

VB 2005

```
Dim password As String = StringTools.RandomString(8, False, True)
```

VC# 2005

```
string password = StringTools.RandomString(8, false , true);
```

รูปที่ 16-5
ผลการทำงานของเมธอด GetGUID()

การตรวจสอบประเภทตัวอักษร

ตัวอักษรที่คุณเห็นอยู่บนคีย์บอร์ด ถ้าจะนับตามความรู้สึกของเราเองสามารถแบ่งออกเป็น 3 กลุ่มใหญ่ๆ คือ

1. ตัวอักษร (ภาษาไทยหรือภาษาอังกฤษ)
2. ตัวเลข
3. ตัวอักษรอื่นๆ

แต่ความเป็นจริงแล้ว ใน .NET แบ่งตัวอักษรออกเป็นหลายประเภทที่น่าสนใจมีดังนี้

ประเภท	ความหมาย
Control	ตัวอักษรควบคุมต่างๆ
Digit	ตัวเลข
Letter	ตัวอักษร
Punctuation	เครื่องหมายวรรคตอน
Separator	เครื่องหมายแบ่งแยก
Symbol	สัญลักษณ์ต่างๆ
Whitespace	ช่องว่าง

เราสามารถตรวจสอบประเภทของตัวอักษรผ่านทางข้อมูลชนิด Char (char) อยู่ในความรับผิดชอบของเมธอดที่ขึ้นต้นด้วย Is... ของ Char

โค้ด VB 2005 และ VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการตรวจสอบประเภทตัวอักษร

VB 2005 (GAF.StringTools.Part2.vb)

```
    Public Shared Function GetCharType(ByVal chr As
Char) As String
        Dim _CharType As String = "Unknown"

        If Char.IsControl(chr) Then
            _CharType = "Control"
        ElseIf Char.IsDigit(chr) Then
            _CharType = "Digit"
        ElseIf Char.IsLetter(chr) Then
            _CharType = "Letter"
        ElseIf Char.IsPunctuation(chr) Then
            _CharType = "Punctuation"
        ElseIf Char.IsSeparator(chr) Then
            _CharType = "Separator"
        ElseIf Char.IsSymbol(chr) Then
            _CharType = "Symbol"
        ElseIf Char.IsWhiteSpace(chr) Then
            _CharType = "Whitespace"
        Else
            _CharType = "Unknown"
        End If

        Return _CharType
    End Function
```

VC# 2005 (GAF.StringTools.Part2.cs)

```
    public static string GetCharType (char chr)
    {
        string _CharType="Unknown";
        if (Char.IsControl(chr))
        {
            _CharType= "Control";
        }
        else if (Char.IsDigit(chr))
        {
            _CharType = "Digit";
        }
        else if (Char.IsLetter(chr))
        {
            _CharType = "Letter";
        }
        else if (Char.IsPunctuation(chr))
        {
            _CharType = "Punctuation";
        }
        else if (Char.IsSeparator(chr))
        {
            _CharType = "Separator";
        }
        else if (Char.IsSymbol(chr))
        {
            _CharType = "Symbol";
        }
```

```
        else if (Char.IsWhiteSpace(chr))
        {
            _CharType = "Whitespace";
        }
        else
        {
            _CharType = "Unknown";
        }
        return _CharType;
    }
```

ส่วนการใช้งานเมธอด GetCharType() อยู่ใน Form1 ดังโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการตรวจสอบประเภทตัวอักษร (Form1.vb)

```
Private Sub cmdCharType_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdCharType.Click
    Dim result As String = StringTools.GetCharType(Convert.ToChar(txtChar.Text))
    MessageBox.Show(result.ToString, "ประเภทตัวอักษร")
End Sub
```

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการตรวจสอบประเภทตัวอักษร (Form1.cs)

```
private void cmdCharType_Click(object sender, EventArgs e)
{
    string result = StringTools.GetCharType(Convert.ToChar(txtChar.Text));
    MessageBox.Show(result.ToString(), "ประเภทตัวอักษร");
}
```

โดยปกติแล้วเราจะใช้การกดแป้นพิมพ์โดยตรง เพื่อพิมพ์ตัวอักษรตามที่ต้องการ แต่คุณสามารถพิมพ์ตัวอักษรที่อยู่นอกเหนือจากที่แสดงบนคีย์บอร์ดได้อีกด้วย โดยการกดปุ่ม <Alt> ค้างไว้ แล้วกดรหัสตัวเลข เช่น Alt กับ 23 ตัวอักษรที่ได้ แสดงดังรูปที่ 16-6

รูปที่ 16-6 การตรวจสอบประเภทตัวอักษร

ให้คุณลองเปลี่ยนตัวเลขอื่นๆ บ้าง เพื่อตรวจสอบประเภทตัวอักษรที่ได้

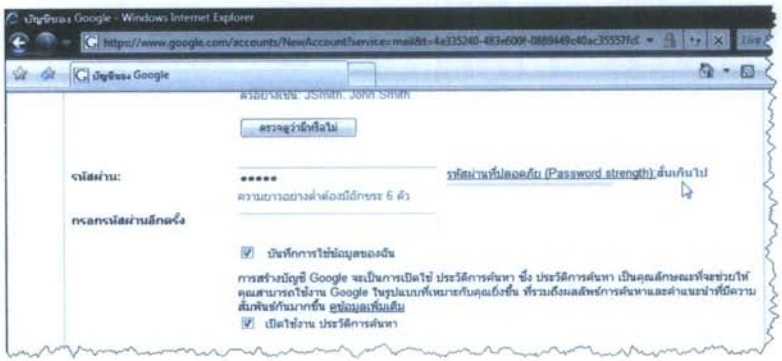
การประเมินรหัสผ่าน (Password)

การพัฒนาแอพพลิเคชันใดๆ ก็ตาม มักจะมีขั้นตอนที่กำหนดให้ผู้ใช้แอพพลิเคชันกำหนดชื่อผู้ใช้ (Username) และรหัสผ่าน (Password) เพื่อ Login เข้าสู่ระบบ จุดสำคัญที่สุดของการทำงานนี้ก็คือ รหัสผ่านของผู้ใช้แต่ละคนนั่นเอง

รหัสผ่านที่ผู้ใช้งานแต่ละคนเป็นผู้กำหนดขึ้นมา เป็นอีก 1 ช่องทางที่อาจจะก่อให้เกิดช่องโหว่ในระบบขึ้นมาได้เช่นกัน การตั้งรหัสผ่านที่ง่ายจนเกินไปจึงเป็นสิ่งที่ควรหลีกเลี่ยง

คำว่ารหัสผ่านที่ง่ายจนเกินไป หมายถึง ง่ายต่อการคาดเดา ส่งผลให้เกิดการปลอมแปลงสิทธิของผู้ใช้เข้ามาในระบบ ซึ่งถือเป็นผลเสียอย่างร้ายแรง เช่น รหัสผ่านที่เป็นคำศัพท์ที่อยู่ในดิกชันนารี, รหัสผ่านที่มีจำนวนตัวอักษรน้อย หรือมีตัวอักษรซ้ำกันเป็นจำนวนมากๆ, รหัสผ่านที่เกิดจากสิ่งรอบตัวหรือสิ่งคุ้นเคย เป็นต้น

คุณสามารถสร้างกฎเกณฑ์กำหนดเงื่อนไขขึ้นมา เพื่อตรวจสอบว่ารหัสผ่านที่ผู้ใช้ระบุดีพอที่จะใช้งานได้หรือไม่ เพราะว่าถ้าคิดในทางกลับกัน การตั้งรหัสผ่านที่ยากมากๆ ก่อให้เกิดปัญหาในการจดจำเช่นกัน แต่ก็คงไม่ใช่เรื่องที่ต้องให้น้ำหนักเท่าใดนัก



รูปที่ 16-7 การตรวจสอบรหัสผ่านของเว็บ www.gmail.com

การกำหนดรหัสผ่านของเว็บ gmail มีการตรวจสอบรหัสผ่านด้วยว่าเหมาะสมในการใช้งานหรือไม่ จึงเป็นที่มาของเมธอด CheckStrongPassword() ผู้เขียนสร้างกฎเกณฑ์การให้คะแนนรหัสผ่านขึ้นมา ซึ่งเกิดจากฐานความคิดที่ว่า

1. คะแนนรหัสผ่านเกิดจากคะแนนของตัวอักษรแต่ละตัวคูณกับตัวคูณ โดยที่คะแนนเริ่มต้นกับตัวคูณเริ่มต้น เกิดจากการใช้ตัวอักษรแต่ละตัว
2. ยิ่งมีจำนวนตัวอักษรมากยิ่งได้คะแนนมาก โดยเฉพาะอย่างยิ่งถ้ามีตัวอักษรจากหลายๆ กลุ่ม จะมีคะแนนโบนัสเพิ่มเติม
3. ถ้ามีการใช้ตัวอักษรหรือตัวเลขเพียงอย่างเดียว ถือว่าเป็นรหัสผ่านที่ไม่ดีจะมีการลดคะแนน และ/หรือลดตัวคูณด้วย
4. ถ้ามีจำนวนตัวอักษรน้อยกว่า 4 ตัวอักษร ถือว่าเป็นรหัสผ่านที่ไม่ดีเช่นกัน ก็จะมีการลดคะแนนเช่นกัน

คุณผู้อ่านไม่จำเป็นต้องใช้หลักการเหมือนกับผู้เขียน โดยจากหลักการ 4 ข้อข้างต้นสามารถแปลงเป็นโค้ดดังนี้

โค้ด VB 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการประเมินรหัสผ่าน (Password) (GAF.StringTools.Part2.vb)

```
Public Shared Function CheckStrongPassword(ByVal password As String) As Double
    Dim _i As Integer = 0
    Dim _point As Double = 0
    Dim _multiply As Double = 2
    Dim _PwdCount As Double = password.Length
    Dim _pwd As Char() = password.ToCharArray()

    _point = _PwdCount / 5

    If _PwdCount <= 4 Then
        _multiply -= 0.5
    End If

    If _PwdCount >= 12 Then
        _point += 0.4
        _multiply += 0.7
    ElseIf _PwdCount >= 8 Then
        _point += 0.7
        _multiply += 0.4
    ElseIf _PwdCount > 4 Then
        _point += 0.5
    End If

    Dim _IsDigit As Boolean = False
    Dim _IsLetter As Boolean = False
    Dim _IsPunctuation As Boolean = False
    Dim _IsSeparator As Boolean = False
    Dim _IsSymbol As Boolean = False

    For _i = 0 To _pwd.Length - 1
        If StringTools.GetCharType(_pwd(_i)) = "Digit" Then
            _IsDigit = True
        End If

        If StringTools.GetCharType(_pwd(_i)) = "Letter" Then
            _IsLetter = True
        End If

        If StringTools.GetCharType(_pwd(_i)) = "Punctuation" Then
            _IsPunctuation = True
            _point += 0.1
            _multiply += 0.1
        End If

        If StringTools.GetCharType(_pwd(_i)) = "Separator" Then
```

```
            _IsSeparator = True
            _point += 0.1
            _multiply += 0.1
        End If

        If StringTools.GetCharType(_pwd(_i)) = "Symbol" Then
            _IsSymbol = True
            _point += 0.1
            _multiply += 0.1
        End If
    Next

    If (_IsDigit = True) AndAlso (_IsLetter = False) AndAlso (_IsPunctuation = False) AndAlso (_IsSeparator = False)
AndAlso (_IsSymbol = False) Then
        _point -= 0.8
        _multiply -= 0.8
    End If

    If (_IsDigit = False) AndAlso (_IsLetter = True) AndAlso (_IsPunctuation = False) AndAlso (_IsSeparator = False)
AndAlso (_IsSymbol = False) Then
        _point -= 0.8
        _multiply -= 0.8
    End If

    If (_IsDigit = True) AndAlso (_IsLetter = True) Then
        _point += 0.1
        _multiply += 0.1
    End If

    If (_IsPunctuation = True) AndAlso (_IsSeparator = True) AndAlso (_IsSymbol = True) Then
        _point += 1
        _multiply += 1
    End If

    If (_IsDigit = True) AndAlso (_IsLetter = True) AndAlso (_IsPunctuation = True) AndAlso (_IsSeparator = True)
AndAlso (_IsSymbol = True) Then
        _point += 1.3
        _multiply += 1.3
    End If

    Dim _final As Double = 0
    _final = _point * _multiply

    If _final > 10 Then
        _final = 10
    ElseIf _final < 0 Then
        _final = 0
    End If

    Return _final
End Function
```

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการประเมินรหัสผ่าน (Password) (GAF.StringTools.Part2.cs)

```
public static double CheckStrongPassword(string password)
{
    int _i = 0;
    double _point = 0.0;
    double _multiply = 2.0;
    double _PwdCount = password.Length;
    char[] _pwd = password.ToCharArray();

    _point = _PwdCount / 5;

    if (_PwdCount <= 4)
    {
        _multiply -= 0.5;
    }

    if (_PwdCount >= 12)
    {
        _point += 0.4;
        _multiply += 0.7;
    }
    else if (_PwdCount >= 8)
    {
        _point += 0.7;
        _multiply += 0.4;
    }
    else if (_PwdCount > 4)
    {
        _point += 0.5;
    }

    bool _IsDigit = false;
    bool _IsLetter = false;
    bool _IsPunctuation = false;
    bool _IsSeparator = false;
    bool _IsSymbol = false;

    for (_i = 0; _i < _pwd.Length; _i++)
    {
        if (StringTools.GetCharType(_pwd[_i])=="Digit")
        {
            _IsDigit = true;
        }

        if (StringTools.GetCharType(_pwd[_i]) == "Letter")
        {
            _IsLetter = true;
```

```
            }

            if (StringTools.GetCharType(_pwd[_i]) == "Punctuation")
            {
                _IsPunctuation = true;
                _point += 0.1;
                _multiply += 0.1;
            }

            if (StringTools.GetCharType(_pwd[_i]) == "Separator")
            {
                _IsSeparator = true;
                _point += 0.1;
                _multiply += 0.1;
            }

            if (StringTools.GetCharType(_pwd[_i]) == "Symbol")
            {
                _IsSymbol = true;
                _point += 0.1;
                _multiply += 0.1;
            }
        }

        if ((_IsDigit==true) && (_IsLetter==false) && (_IsPunctuation==false) && (_IsSeparator==false) && (_IsSymbol==false))
        {
            _point -= 0.8;
            _multiply -= 0.8;
        }

        if ((_IsDigit == false) && (_IsLetter == true) && (_IsPunctuation == false) && (_IsSeparator == false) &&
(_IsSymbol == false))
        {
            _point -= 0.8;
            _multiply -= 0.8;
        }

        if ((_IsDigit == true) && (_IsLetter == true))
        {
            _point += 0.1;
            _multiply += 0.1;
        }

        if ((_IsPunctuation == true) && (_IsSeparator == true) && (_IsSymbol == true))
        {
            _point += 1.0;
            _multiply += 1.0;
        }
```

```
        if ((_IsDigit == true) && (_IsLetter == true) && (_IsPunctuation == true) && (_IsSeparator == true) && (_IsSymbol
== true))
        {
            _point += 1.3;
            _multiply += 1.3;
        }

        double _final=0.0;
        _final=_point * _multiply;
        if (_final>10)
        {
            _final=10;
        }
        else if (_final < 0)
        {
            _final = 0;
        }
        return _final;
    }
```

เมธอด CheckStrongPassword() ต้องการพารามิเตอร์ 1 ตัวคือ Password มี Type เป็น String (string) หมายถึง รหัสผ่านที่ต้องการประเมินและคืนค่าเป็นคะแนนที่ได้มี Type เป็น Double (double) โดยที่

- ฟิลด์ _point หมายถึง คะแนนของตัวอักษรแต่ละตัว
- ฟิลด์ multiply หมายถึง ตัวคูณ กำหนดค่าเริ่มต้นเป็นคูณ 2
- ฟิลด์ _PwdCount หมายถึง จำนวนตัวอักษรของรหัสผ่าน
- ฟิลด์ _pwd หมายถึง อาร์เรย์ของตัวอักษร มาจากรหัสผ่านที่ต้องการประเมินนั่นเอง

VB 2005

```
Public Shared Function CheckStrongPassword(ByVal password As String) As Double
    Dim _i As Integer = 0
    Dim _point As Double = 0
    Dim _multiply As Double = 2
    Dim _PwdCount As Double = password.Length
    Dim _pwd As Char() = password.ToCharArray()
```

VC# 2005

```
public static double CheckStrongPassword(string password)
{
    int _i = 0;
    double _point = 0.0;
    double _multiply = 2.0;
    double _PwdCount = password.Length;
    char[] _pwd = password.ToCharArray();
```

คะแนนเริ่มต้น (ฟิลด์ _point) เกิดจากจำนวนตัวอักษรของรหัสผ่าน (ฟิลด์ _PwdCount) หารด้วย 5 กล่าวคือ

VB 2005

```
_point = _PwdCount / 5
```

VC# 2005

```
_point = _PwdCount / 5;
```

- ถ้ารหัสผ่านมีน้อยกว่าหรือเท่ากับ 4 ตัวอักษร ให้ลดตัวคูณ (ฟิลด์ _multiply) ลง 0.5
- ถ้ามากกว่าหรือเท่ากับ 12 ตัวอักษร ให้เพิ่มคะแนน (ฟิลด์ _point) 0.4 และเพิ่มตัวคูณ (ฟิลด์ _multiply) 0.7
- ถ้ามากกว่าหรือเท่ากับ 8 ตัวอักษร ให้เพิ่มคะแนน (ฟิลด์ _point) 0.7 และเพิ่มตัวคูณ (ฟิลด์ _multiply) 0.4
- ถ้ามากกว่า 4 ตัวอักษรให้เพิ่มคะแนน (ฟิลด์ _point) 0.5

VB 2005

```
If _PwdCount <= 4 Then
   _multiply -= 0.5
End If

If _PwdCount >= 12 Then
   _point += 0.4
   _multiply += 0.7
ElseIf _PwdCount >= 8 Then
   _point += 0.7
   _multiply += 0.4
ElseIf _PwdCount > 4 Then
   _point += 0.5
End If
```

VC# 2005

```
if (_PwdCount <= 4)
{
    _multiply -= 0.5;
}

if (_PwdCount >= 12)
{
    _point += 0.4;
    _multiply += 0.7;
}
else if (_PwdCount >= 8)
{
    _point += 0.7;
    _multiply += 0.4;
}
else if (_PwdCount > 4)
{
    _point += 0.5;
}
```

ต่อมาสั่งให้วนลูปเพื่อตรวจสอบประเภทของตัวอักษรแต่ละตัว โดยอาศัยเมธอด GetCharType() เข้ามาช่วย ก็จะใช้ฟิลด์ที่ขึ้นต้นด้วย _Is... เพื่อบอกว่ารหัสผ่านปัจจุบันประกอบด้วยตัวอักษรประเภทใดบ้าง โดยที่

ถ้ามีการใช้เครื่องหมายวรรคตอน (Punctuation), เครื่องหมายแบ่งแยก (Separator) หรือสัญลักษณ์ (Symbol) ให้เพิ่มคะแนน (ฟิลด์ _point) และเพิ่มตัวคูณ (ฟิลด์ _multiply) ตัวละ 0.1

VB 2005

```
For _i = 0 To _pwd.Length - 1
   If StringTools.GetCharType(_pwd(_i)) = "Digit" Then
      _IsDigit = True
   End If

   If StringTools.GetCharType(_pwd(_i)) = "Letter" Then
      _IsLetter = True
   End If

   If StringTools.GetCharType(_pwd(_i)) = "Punctuation" Then
      _IsPunctuation = True
      _point += 0.1
      _multiply += 0.1
   End If

   If StringTools.GetCharType(_pwd(_i)) = "Separator" Then
      _IsSeparator = True
      _point += 0.1
      _multiply += 0.1
   End If

   If StringTools.GetCharType(_pwd(_i)) = "Symbol" Then
      _IsSymbol = True
      _point += 0.1
      _multiply += 0.1
   End If
Next
```

VC# 2005

```
for (_i = 0; _i < _pwd.Length; _i++)
{
   if (StringTools.GetCharType(_pwd[_i])=="Digit")
   {
      _IsDigit = true;
   }

   if (StringTools.GetCharType(_pwd[_i]) == "Letter")
   {
      _IsLetter = true;
   }

   if (StringTools.GetCharType(_pwd[_i]) == "Punctuation")
   {
      _IsPunctuation = true;
      _point += 0.1;
      _multiply += 0.1;
```

```
            }

            if (StringTools.GetCharType(_pwd[_i]) == "Separator")
            {
                _IsSeparator = true;
                _point += 0.1;
                _multiply += 0.1;
            }

            if (StringTools.GetCharType(_pwd[_i]) == "Symbol")
            {
                _IsSymbol = true;
                _point += 0.1;
                _multiply += 0.1;
            }
        }
```

ต่อมาตรวจสอบว่าถ้ารหัสผ่านมีตัวเลขเพียงอย่างเดียว (ฟิลด์ _IsDigit = True) หรือมีตัวอักษรเพียงอย่างเดียว (ฟิลด์ _IsLetter = True) ให้ลดค่าคะแนน (ฟิลด์ _point) และลดค่าตัวคูณ (ฟิลด์ _multiply) ลง 0.8

VB 2005

```
        If (_IsDigit = True) AndAlso (_IsLetter = False) AndAlso (_IsPunctuation = False) AndAlso (_IsSeparator = False)
AndAlso (_IsSymbol = False) Then
            _point -= 0.8
            _multiply -= 0.8
        End If

        If (_IsDigit = False) AndAlso (_IsLetter = True) AndAlso (_IsPunctuation = False) AndAlso (_IsSeparator = False)
AndAlso (_IsSymbol = False) Then
            _point -= 0.8
            _multiply -= 0.8
        End If
```

VC# 2005

```
            if ((_IsDigit==true) && (_IsLetter==false) && (_IsPunctuation==false) && (_IsSeparator==false) && (_IsSymbol==false))
            {
                _point -= 0.8;
                _multiply -= 0.8;
            }

            if ((_IsDigit == false) && (_IsLetter == true) && (_IsPunctuation == false) && (_IsSeparator == false) &&
(_IsSymbol == false))
            {
                _point -= 0.8;
                _multiply -= 0.8;
            }
```

แต่ถ้ามีทั้งตัวเลข (ฟิลด์ _IsDigit = True) และตัวอักษร (ฟิลด์ _IsLetter = True) ให้เพิ่มคะแนน (ฟิลด์ _point) และตัวคูณ (ฟิลด์ _multiply) อีก 0.1

VB 2005

```
If (_IsDigit = True) AndAlso (_IsLetter = True) Then
    _point += 0.1
    _multiply += 0.1
End If
```

VC# 2005

```
if ((_IsDigit == true) && (_IsLetter == true))
{
    _point += 0.1;
    _multiply += 0.1;
}
```

แต่ถ้ามีการใช้เครื่องหมายวรรคตอน (ฟิลด์ IsPunctuation = True) และเครื่องหมายแบ่งแยก (ฟิลด์ _IsSeparator) และสัญลักษณ์ (ฟิลด์ _IsSymbol = True) ให้เพิ่มคะแนน (ฟิลด์ _point) และตัวคูณ (ฟิลด์ _multiply) อีก 1

VB 2005

```
If (_IsPunctuation = True) AndAlso (_IsSeparator = True) AndAlso (_IsSymbol = True) Then
    _point += 1
    _multiply += 1
End If
```

VC# 2005

```
if ((_IsPunctuation == true) && (_IsSeparator == true) && (_IsSymbol == true))
{
    _point += 1.0;
    _multiply += 1.0;
}
```

แต่ถ้ามีการใช้ตัวอักษรครบทุกกลุ่ม ให้เพิ่มคะแนน (ฟิลด์ _point) และตัวคูณ (ฟิลด์ _multiply) อีก 1.3

VB 2005

```
If (_IsDigit = True) AndAlso (_IsLetter = True) AndAlso (_IsPunctuation = True) AndAlso (_IsSeparator = True)
AndAlso (_IsSymbol = True) Then
    _point += 1.3
    _multiply += 1.3
End If
```

VC# 2005

```
        if ((_IsDigit == true) && (_IsLetter == true) && (_IsPunctuation == true) && (_IsSeparator == true) && (_IsSymbol
== true))
        {
            _point += 1.3;
            _multiply += 1.3;
        }
```

ท้ายที่สุดคะแนนของรหัสผ่านที่ได้ (ฟิลด์ _final) เกิดจากคะแนน (ฟิลด์ _point) คูณกับตัวคูณ (ฟิลด์ _multiply) ผลการทดสอบ แสดงดังรูปที่ 16-8 และ 16-9

VB 2005

```
        Dim _final As Double = 0
        _final = _point * _multiply

        If _final > 10 Then
            _final = 10
        ElseIf _final < 0 Then
            _final = 0
        End If

        Return _final
```

VC# 2005

```
        double _final=0.0;
        _final=_point * _multiply;
        if (_final>10)
        {
            _final=10;
        }
        else if (_final < 0)
        {
            _final = 0;
        }
        return _final;
```

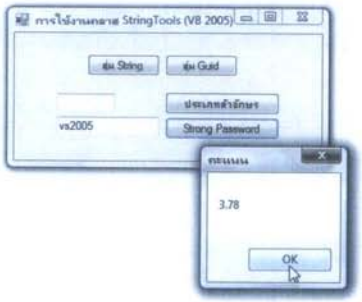
โค้ด VB 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการประเมินรหัสผ่าน (Password) (Form1.vb)

```
    Private Sub txtCheckStrongPwd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles txtCheckStrongPwd.Click
        Dim result As Double
        result = StringTools.CheckStrongPassword(txtPwd.Text)
        MessageBox.Show(result.ToString, "คะแนน")
    End Sub
```

โค้ด VC# 2005 ที่ 16-2 การสร้างคลาสที่ทำงานเกี่ยวข้องกับ String เฉพาะการประเมินรหัสผ่าน (Password) (Form1.cs)

```
        private void txtCheckStrongPwd_Click(object sender, EventArgs e)
        {
            double result;
            result = StringTools.CheckStrongPassword(txtPwd.Text);
            MessageBox.Show(result.ToString(), "คะแนน");
        }
```

รูปที่ 16-8
ผลการประเมิน
รหัสผ่าน vs2005



รูปที่ 16-9
ผลการประเมิน
รหัสผ่าน
vs2007+&%

จากรูปที่ 16-9 รหัสผ่าน vs2007+&% ได้คะแนนค่อนข้างสูง เพราะว่าประกอบไปด้วยตัวอักษรหลายกลุ่มที่ยากต่อการคาดเดา ส่วนแนวทางการนำไปใช้งาน เช่น ถ้าได้คะแนนน้อยกว่า 5 ถือว่าเป็นรหัสผ่านที่ใช้ไม่ได้ เป็นต้น

สรุปท้ายบท

เนื้อหาของบทนี้จะแตกต่างไปจากบทอื่นๆ ตรงที่ว่า เป็นการนำเสนอการใช้งานเป็นหลักไม่ใช่เนื้อหา ผู้เขียนคิดว่าแนวทางนี้น่าจะเกิดประโยชน์สูงสุดต่อคุณผู้อ่านมากกว่า

CHAPTER 17

โครงสร้างข้อมูลและอัลกอริธึม

บทนำ

คุณผู้อ่านหลายๆ ท่านที่จบการศึกษาสาขาคอมพิวเตอร์มา จะมีอยู่ 1 วิชาที่ทุกๆ ท่านคุ้นเคยกันดีนั่นคือ วิชาโครงสร้างข้อมูลและอัลกอริธึม เนื้อหาในบทนี้ผู้เขียนนำเสนอเนื้อหาของโครงสร้างข้อมูลและอัลกอริธึมที่มีความคลาสสิกเป็นอย่างยิ่ง ไม่ว่าคุณจะเริ่มต้นศึกษาการเขียนโปรแกรมด้วยภาษาใดก็ตาม

การใช้งาน ArrayList

เป็นรูปแบบการเก็บข้อมูลเป็นชุด มีลักษณะเหมือนกับอาร์เรย์ที่สามารถใช้ลูปชนิด For...Each... (foreach...) เพื่ออ่านค่าของสมาชิกที่อยู่ใน ArrayList ได้อีกด้วย ให้ดูตัวอย่างที่ 17-1 การใช้งาน ArrayList เพิ่มเติม

โค้ด VB 2005 ที่ 17-1 การใช้งาน ArrayList (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim al As New ArrayList()
        ShowCapacityAndCount(al)

        Dim i As Integer
        For i = 1 To 5
            al.Add("รายการที่ " & i)
        Next

        ShowCapacityAndCount(al)
```

```
        ShowElement(al)
    End Sub

    Private Sub ShowCapacityAndCount(ByVal CurrentArrayList As ArrayList)
        Dim result As String
        result = "Capacity : " & CurrentArrayList.Capacity & Environment.NewLine
        result &= "Count : " & CurrentArrayList.Count

        MessageBox.Show(result.ToString())
    End Sub

    Private Sub ShowElement(ByVal CurrentArrayList As ArrayList)
        Dim TotalElement As String = ""
        For Each CurrentElement As String In CurrentArrayList
            TotalElement &= CurrentElement & Environment.NewLine
        Next

        MessageBox.Show(TotalElement)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 17-1 การใช้งาน ArrayList (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        ArrayList al = new ArrayList();
        ShowCapacityAndCount(al);

        int i;
        for (i = 1; i <= 5; i++)
        {
            al.Add("รายการที่ " + i);
        }

        ShowCapacityAndCount(al);
        ShowElement(al);
    }

    private void ShowCapacityAndCount(ArrayList CurrentArrayList)
    {
        string result;
        result = "Capacity : " + CurrentArrayList.Capacity + Environment.NewLine;
        result += "Count : " + CurrentArrayList.Count;

        MessageBox.Show(result.ToString());
    }

    private void ShowElement(ArrayList CurrentArrayList)
    {
        string TotalElement = "";
        foreach (string CurrentElement in CurrentArrayList)
        {
            TotalElement += CurrentElement + Environment.NewLine;
        }

        MessageBox.Show(TotalElement);
    }
```

ในเหตุการณ์ Form1_Load() ผู้เขียนสร้าง ArrayList ที่ชื่อว่า al ขึ้นมา สิ่งที่น่าสนใจมีอยู่ 2 อย่างคือ

1. คุณสมบัติ Capacity หมายถึง ความจุที่คลาส ArrayList เผื่อไว้สำหรับเพิ่มสมาชิกใหม่ เพิ่มขึ้นเป็นทวีคูณของ 8 เช่น 8, 16, 24...
2. คุณสมบัติ Count หมายถึง จำนวนสมาชิกจริงที่อยู่ใน ArrayList

กำหนดให้เพิ่มสมาชิก 5 รายการให้กับออบเจ็กต์ al เห็นได้ว่าเป็นจำนวนที่ยังอยู่ในพื้นที่ความจุ 8 ที่แรก โดยการอ่านความจุ (คุณสมบัติ Capacity) และจำนวนสมาชิกจริง (คุณสมบัติ Count) อยู่ในความรับผิดชอบของซับรูทีน ShowCapacityAndCount()

แต่ถ้า ArrayList ไม่มีสมาชิกอยู่เลย ความจุและจำนวนสมาชิกจริงที่ได้คือ 0

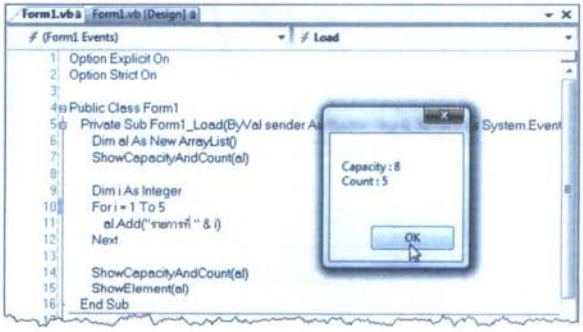
VB 2005

```
Private Sub ShowCapacityAndCount(ByVal CurrentArrayList As ArrayList)
    Dim result As String
    result = "Capacity : " & CurrentArrayList.Capacity & Environment.NewLine
    result &= "Count : " & CurrentArrayList.Count

    MessageBox.Show(result.ToString())
End Sub
```

VC# 2005

```
private void ShowCapacityAndCount(ArrayList CurrentArrayList)
{
    string result;
    result = "Capacity : " + CurrentArrayList.Capacity + Environment.NewLine;
    result += "Count : " + CurrentArrayList.Count;

    MessageBox.Show(result.ToString());
}
```

ส่วนการแสดงค่าของสมาชิกแต่ละรายการอยู่ในความรับผิดชอบของซับรูทีน ShowElement()



รูปที่ 17-1 กรณี ArrayList มีสมาชิก 5 รายการ

ผู้เขียนแก้ไขโค้ดใหม่กำหนดให้เพิ่มรายการเป็น 9 รายการ เป็นจำนวนที่เกินจากความจุเดิมที่มีอยู่ ผลการทำงาน แสดงดังรูปที่ 17-2

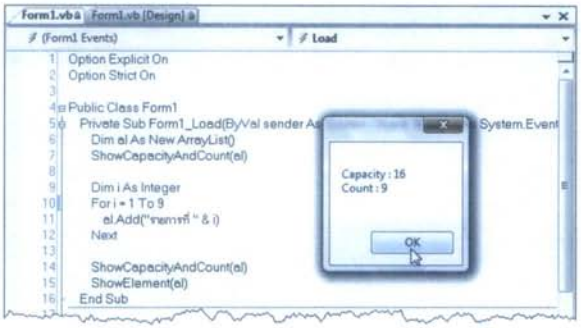
VB 2005

```
For i = 1 To 9
    al.Add("รายการที่ " & i)
Next
```

VC# 2005

```
for (i = 1; i <= 9; i++)
{
    al.Add("รายการที่ " + i);
}
```

รูปที่ 17-2
กรณี ArrayList มีสมาชิก 9 รายการ



จากรูปที่ 17-2 เห็นได้ว่าความจุ (คุณสมบัติ Capacity) ของ al ถูกขยายเป็น 16 เป็นค่าทวีคูณของ 8 นั่นเอง

การใช้งาน Stack

โครงสร้างข้อมูลแบบ Stack เป็นการจัดเก็บข้อมูลเรียงลำดับก่อน-หลังแบบ Last In First Out (LIFO) หมายถึง เข้าทีหลังแต่ออกก่อน ให้คุณนึกถึงการวางลูกปิงปองลงในกระบอก คุณต้องหยิบลูกปิงปองลูกบนสุดก่อนเสมอ ดังรูปที่ 17-3

รูปที่ 17-3
การเก็บลูกปิงปองในกระบอก

การวางข้อมูลเข้าไปใน Stack เรียกว่า Push ส่วนการหยิบข้อมูลออกเรียกว่า Pop ให้ดูตัวอย่างที่ 17-2 การใช้งาน Stack เพิ่มเติม

รูปที่ 17-4 ฟอร์มในขณะออกแบบ

โค้ด VB 2005 และ VC# 2005 ที่ 17-2 การใช้งาน Stack

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Dim myStack As New Stack()

    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        myStack.Push("1")
        myStack.Push("2")
        myStack.Push("3")
        myStack.Push("4")
    End Sub

    Private Sub cmdPush_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdPush.Click
        If txtData.Text.Trim() <> "" Then
            myStack.Push(txtData.Text)
            ShowStack(myStack)

            txtData.Text = ""
            txtData.Focus()
        End If
    End Sub

    Private Sub cmdPop_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdPop.Click
        If myStack.Count > 0 Then
            myStack.Pop()
            ShowStack(myStack)
        End If
    End Sub

    Private Sub cmdShow_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdShow.Click
        ShowStack(myStack)
    End Sub
```

VC# 2005 (Form1.cs)

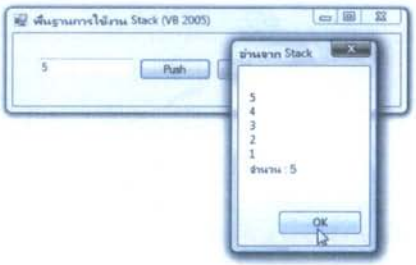
```
Stack myStack = new Stack();

private void Form1_Load(object sender, EventArgs e)
{
    myStack.Push("1");
    myStack.Push("2");
    myStack.Push("3");
    myStack.Push("4");
}

private void cmdPush_Click(object sender, EventArgs e)
{
    if (txtData.Text.Trim() != "")
    {
        myStack.Push(txtData.Text);
        ShowStack(myStack);

        txtData.Text = "";
        txtData.Focus();
    }
}

private void cmdPop_Click(object sender, EventArgs e)
{
    if (myStack.Count > 0)
    {
        myStack.Pop();
        ShowStack(myStack);
    }
}

private void cmdShow_Click(object sender, EventArgs e)
{
    ShowStack(myStack);
}

private void ShowStack(Stack st)
```

```
    Private Sub ShowStack(ByVal st As Stack)
        Dim AllItem As String = ""
        If st.Count > 0 Then
            For Each strItem As String In st
                AllItem &= strItem & Environment.NewLine
            Next
            AllItem = AllItem & "จำนวน : " & st.Count
        End If

        MessageBox.Show(AllItem, "อ่านจาก Stack")
    End Sub
End Class
```

```
{
    string AllItem = "";
    if (st.Count > 0)
    {
        foreach (string strItem in st)
        {
            AllItem += strItem + Environment.NewLine;
        }
        AllItem = AllItem + "จำนวน : " + st.Count;
    }

    MessageBox.Show(AllItem, "อ่านจาก Stack");
}
```

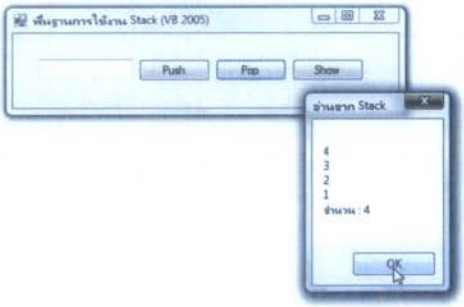
ผู้เขียนสร้าง Stack ที่ชื่อว่า myStack ขึ้นมา กำหนดให้ Push ข้อมูล 1, 2, 3 และ 4 ตามลำดับ ให้คุณลอง Push ข้อมูลใหม่เข้าไปยัง myStack ดังรูปที่ 17-5

รูปที่ 17-5
การ Push ข้อมูลลงใน Stack



จากรูปที่ 17-5 เป็นการ Push เลข 5 เข้าไปใน myStack พบว่าเลข 5 อยู่ลำดับบนสุด ให้คุณลอง Pop ข้อมูลออก ดังรูปที่ 17-6

รูปที่ 17-6
การ Pop ข้อมูลออกจาก Stack



ส่วนการ Pop ข้อมูลออกจาก Stack เลข 5 ซึ่งอยู่ลำดับบนสุดจะถูกนำออกไปก่อนนั่นเอง

การใช้งาน HashTable

โครงสร้างข้อมูลแบบ HashTable ประกอบด้วย 2 ส่วนคือ คีย์ (keys) และค่าที่เก็บอยู่ (Value) จะใช้คีย์เป็นตัวนำทางไปสู่ข้อมูลที่เก็บอยู่ ส่งผลให้การค้นหาข้อมูลที่เก็บอยู่ใน HashTable มีความรวดเร็ว ให้ดูตัวอย่างที่ 17-3 การใช้งาน HashTable เพิ่มเติม

โค้ด VB 2005 ที่ 17-3 การใช้งาน HashTable (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ht As New Hashtable
        ht.Add("VB2005", "Visual Basic 2005")
        ht.Add("VC#2005", "Visual C# 2005")
        ht.Add("VS2005", "Visual Studio 2005")

        Dim result As String = ""
        For Each Ckeys As String In ht.Keys
            result &= "คีย์ : " & Ckeys & Environment.NewLine
        Next

        For Each CValue As String In ht.Values
            result &= "ค่า : " & CValue & Environment.NewLine
        Next

        result &= "จำนวน : " & ht.Count & " รายการ" & Environment.NewLine

        Dim myLanguage As String = DirectCast(ht("VB2005"), String)
        result &= "การอ่านค่าแบบระบุคีย์แบบที่ 1 : " & myLanguage & Environment.NewLine
        result &= "การอ่านค่าแบบระบุคีย์แบบที่ 2 : " & ht.Item("VC#2005").ToString()

        MessageBox.Show(result, "ผลการอ่าน HashTable")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 17-3 การใช้งาน HashTable (Form1.cs)

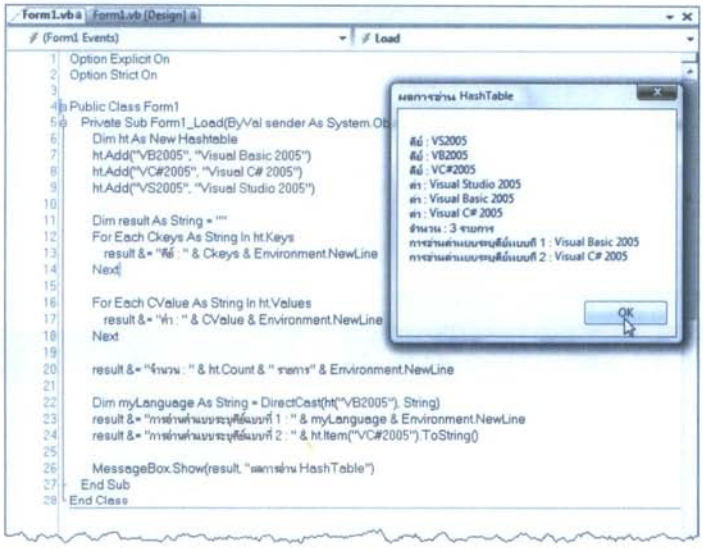
```
    private void Form1_Load(object sender, EventArgs e)
    {
        Hashtable ht = new Hashtable();
        ht.Add("VB2005", "Visual Basic 2005");
        ht.Add("VC#2005", "Visual C# 2005");
        ht.Add("VS2005", "Visual Studio 2005");

        string result = "";
        foreach (string Ckeys in ht.Keys)
        {
```

```
        result += "คีย์ : " + Ckeys + Environment.NewLine;
    }

    foreach (string CValue in ht.Values)
    {
        result += "ค่า : " + CValue + Environment.NewLine;
    }

    result += "จำนวน : " + ht.Count + " รายการ" + Environment.NewLine;

    string myLanguage = (string)ht["VB2005"];
    result += "การอ่านค่าแบบระบุคีย์แบบที่ 1 : " + myLanguage +Environment.NewLine;
    result += "การอ่านค่าแบบระบุคีย์แบบที่ 2 : " + ht["VC#2005"].ToString();

    MessageBox.Show(result, "ผลการอ่าน HashTable");
}
```

ผู้เขียนสร้าง HashTable ที่ชื่อว่า ht ขึ้นมา สั่งให้เพิ่มรายการเข้าไป 3 รายการ แต่ละรายการประกอบด้วยคีย์ และค่าที่เก็บอยู่

การอ่านค่าที่เก็บอยู่ใน HashTable คุณสามารถใช้ลูป For...Each... (foreach...) ได้เช่นเดียวกับตัวแปรอาร์เรย์ โดยการระบุคีย์ที่ต้องการค้นหา ส่วนจำนวนรายการที่เก็บอยู่ใน HashTable อ่านได้จากคุณสมบัติ Count

รูปที่ 17-7 ผลการค้นหาข้อมูลที่มีคีย์ VB2005 และ VC#2005



การใช้งานคิว (Queue)

สำหรับโครงสร้างข้อมูลแบบคิวมีลักษณะแบบ First In First Out (FIFO) หมายถึง เข้าก่อน-ออกก่อน ให้ดูตัวอย่างที่ 17-4 การใช้งานคิวเพิ่มเติม

โค้ด VB 2005 ที่ 17-4 การใช้งานคิว (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim s() As String = {"VB 2005", "VC# 2005", "ASP.NET 2.0"}
        Dim myQ As New Queue()
        Dim str As String = ""

        For Each str In s
            myQ.Enqueue(str)
        Next
        ShowData(myQ)

        For i As Integer = 0 To myQ.Count - 1
            MessageBox.Show(myQ.Dequeue.ToString())
        Next
        ShowData(myQ)
    End Sub

    Private Sub ShowData(ByVal q As Queue)
        Dim result As String = ""
        Dim str As String = ""
        For Each str In q
            result &= str & " "
        Next
        MessageBox.Show(result, "คิว")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 17-4 การใช้งานคิว (Form1.cs)

```
        private void Form1_Load(object sender, EventArgs e)
        {
            string[] s = new string[] { "VB 2005", "VC# 2005", "ASP.NET 2.0" };
            Queue myQ = new Queue();
            string str = "";

            foreach (string tempLoopVar_str in s)
            {
                str = tempLoopVar_str;
                myQ.Enqueue(str);
            }
            ShowData(myQ);
```

```
        for (int i = 0; i <= myQ.Count - 1; i++)
        {
            MessageBox.Show(myQ.Dequeue().ToString());
        }
        ShowData(myQ);
    }

    private void ShowData(Queue q)
    {
        string result = "";
        string str = "";
        foreach (string tempLoopVar_str in q)
        {
            str = tempLoopVar_str;
            result += str + "  ";
        }
        MessageBox.Show(result, "คิว");
    }
```

ผู้เขียนสร้างอาร์เรย์ของ String ขึ้นมา 1 ชุด ประกอบด้วยข้อความ VB 2005, VC# 2005 และ ASP.NET 2.0 ตามลำดับ เก็บไว้ในตัวแปรอาร์เรย์ s กำหนดให้ใส่ข้อความเข้าไปใน Queue ที่ชื่อว่า myQ ตามลำดับ ด้วยเมธอด Enqueue() การแสดงรายการที่อยู่ในคิว myQ อยู่ในความรับผิดชอบของเมธอด ShowData() ดังรูปที่ 17-8

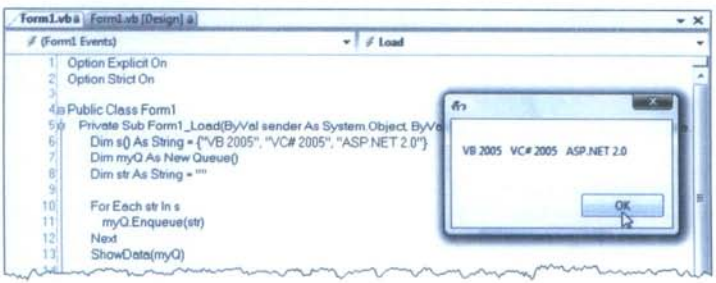
VB 2005

```
Dim s() As String = {"VB 2005", "VC# 2005",
"ASP.NET 2.0"}
    Dim myQ As New Queue()
    Dim str As String = ""

    For Each str In s
        myQ.Enqueue(str)
    Next
    ShowData(myQ)
```

VC# 2005

```
string[] s = new string[] { "VB 2005", "VC# 2005",
"ASP.NET 2.0" };
    Queue myQ = new Queue();
    string str = "";

    foreach (string tempLoopVar_str in s)
    {
        str = tempLoopVar_str;
        myQ.Enqueue(str);
    }
    ShowData(myQ);
```

ข้อความที่อยู่ในคิว myQ อยู่ตามลำดับที่ถูกส่งเข้ามา ดังรูปที่ 17-8



รูปที่ 17-8 ข้อความที่อยู่ในคิว myQ

ส่วนการนำออกจากคิวให้ใช้เมธอด Dequeue() ดังรูปที่ 17-9

VB 2005

```
For i As Integer = 0 To myQ.Count - 1
    MessageBox.Show(myQ.Dequeue.ToString())
Next
ShowData(myQ)
```

VC# 2005

```
for (int i = 0; i <= myQ.Count - 1; i++)
{
    MessageBox.Show(myQ.Dequeue().ToString());
}
ShowData(myQ);
```

รูปที่ 17-9 การนำออกจากคิวตามลำดับที่ถูกส่งเข้ามา

ลำดับแรก

ลำดับที่ 2

ลำดับที่ 3

การใช้งานฟังก์ชันแบบ Recursive

การใช้งานฟังก์ชันแบบ Recursive หมายถึง การเขียนโค้ดให้ฟังก์ชันเรียกตัวมันเองขึ้นมาทำงาน หลักการของวิธีนี้อยู่ตรงที่คุณต้องกำหนดเงื่อนไขสิ้นสุดการเรียกใช้นั่นเอง ไม่เช่นนั้นการเรียกดังกล่าวก็จะไม่มีวันสิ้นสุด ให้ดูตัวอย่างที่ 17-5 การใช้งานฟังก์ชันแบบ Recursive เพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 17-5 การใช้งานฟังก์ชันแบบ Recursive

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As Decimal

        result = Factorial(4)
        MessageBox.Show(result.ToString("#,##0"),
"ผลการคำนวณค่า Factorial")
    End Sub

    Private Function Factorial(ByVal a As Short) As Decimal
        If a <= 1 Then
            Return 1
        Else
            Return a * Factorial(CShort(a - 1))
        End If
    End Function
End Class
```

VC# 2005 (Form1.cs)

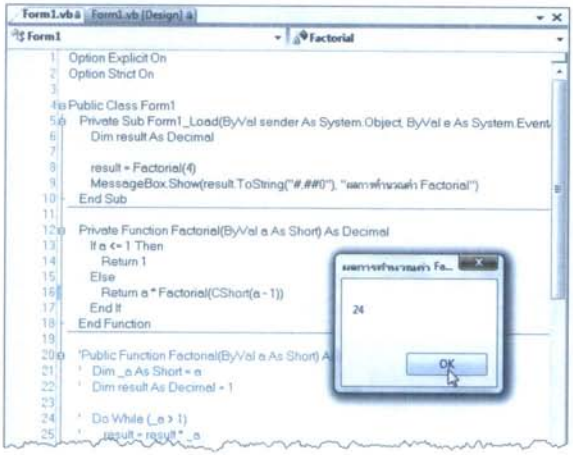
```
    private void Form1_Load(object sender, EventArgs e)
    {
        decimal result;

        result = Factorial(4);
        MessageBox.Show(result.ToString("#,##0"),
"ผลการคำนวณค่า Factorial");
    }

    private decimal Factorial(short a)
    {
        if (a <= 1)
        {
            return 1;
        }
        else
        {
            return a * Factorial(Convert.ToInt16(a - 1));
        }
    }
}
```

การหาค่า Factorial เป็นการหาผลคูณของตัวเลขที่ลดลงเรื่อยๆ จนเหลือ 1 เช่น 4! หมายถึง 4*3*2*1 มีค่าเท่ากับ 24 เราจะใช้ 1 เป็นเงื่อนไขของการหยุดเรียกตัวเองนั่นเอง

จะเห็นได้ว่าในฟังก์ชัน Factorial() เราจะตรวจสอบก่อนเลยว่า ถ้าพารามิเตอร์ a มีค่าน้อยกว่าหรือเท่ากับ 1 ให้คืนค่าเป็น 1 ซึ่งเป็นค่าสุดท้ายของการคูณใน Factorial

แต่ถ้าพารามิเตอร์ a ยังไม่ใช่ค่า 1 ให้เรียกฟังก์ชัน Factorial() ทำงานซ้ำ แต่ให้ลดค่าของพารามิเตอร์ a ลงอีก 1 และเปลี่ยนเป็น Short Type (Int16) ผลการทำงาน แสดงดังรูปที่ 17-10

รูปที่ 17-10
ผลการหาค่า 4!



ถ้าคุณต้องการหาค่า Factorial แต่ไม่ต้องการใช้วิธี Recursive คุณต้องเขียนโค้ดวนลูปหาผลคูณของพารามิเตอร์ a เอง ดังโค้ดต่อไปนี้

VB 2005

```
Public Function Factorial(ByVal a As Short) As Decimal
    Dim _a As Short = a
    Dim result As Decimal = 1

    Do While (_a > 1)
        result = result * _a
        _a = CShort(_a - 1)
    Loop

    Return result
End Function
```

VC# 2005

```
public decimal Factorial(short a)
{
    short _a = a;
    decimal result = 1;

    while (_a > 1)
    {
        result = result * _a;
        _a--;
    }

    return result;

}
```

การสลับค่า

การสลับค่าเรียกอีกอย่างว่า Swap Data ให้ดูตัวอย่างที่ 17-6 การสลับค่าดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 17-6 การสลับค่า

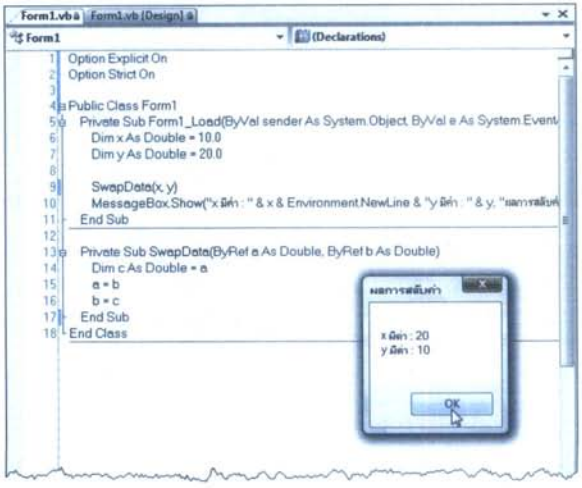
VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim x As Double = 10.0
        Dim y As Double = 20.0

        SwapData(x, y)
        MessageBox.Show("x มีค่า : " & x & Environment.
NewLine & "y มีค่า : " & y, "ผลการสลับค่า")
    End Sub

    Private Sub SwapData(ByRef a As Double, ByRef b
As Double)
        Dim c As Double = a
        a = b
        b = c
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        double x = 10.0;
        double y = 20.0;

        SwapData(ref x,ref  y);
        MessageBox.Show("x มีค่า : " + x + Environment.
NewLine + "y มีค่า : " + y, "ผลการสลับค่า");
    }

    private void SwapData(ref double a, ref double b)
    {
        double c = a;
        a = b;
        b = c;
    }
```

รูปที่ 17-11
ผลการสลับค่าของตัวแปร x กับตัวแปร y



จากรูปที่ 17-11 กำหนดค่าเริ่มต้นของตัวแปร x = 10 ตัวแปร y = 20 การสลับค่าเป็นหน้าที่ของฟังก์ชัน SwapData() ต้องการพารามิเตอร์ 2 ตัว กำหนดให้พารามิเตอร์ a กับ b รับค่าแบบ Reference เพราะว่าต้องการให้มีผลกับตัวแปร x และ y ที่ส่งเข้ามานั่นเอง

หลักการก็คือ สร้างตัวแปรขึ้นมา 1 ตัวชื่อว่า c ทำหน้าที่พักค่าไว้ก่อน จากนั้นใช้วิธีถ่ายค่ากันระหว่างพารามิเตอร์ a, พารามิเตอร์ b และตัวแปร c นั่นเอง สมมติเงื่อนไขเพิ่มเติมว่าห้ามสร้างตัวแปรตัวที่ 3 จะสลับค่าอย่างไร ให้คุณดูโค้ดต่อไปนี้

VB 2005

```
Private Sub SwapData(ByRef a As Double, ByRef b As
Double)
    a = a + b
    b = a - b
    a = a - b
End Sub
```

VC# 2005

```
private void SwapData(ref double a, ref double b)
{
    a = a + b;
    b = a - b;
    a = a - b;
}
```

วิธีคิดก็คือ ให้ใช้วิธีบวกลบค่าฝากไว้ที่พารามิเตอร์ตัวใดตัวหนึ่งก่อนก็ได้

การหาค่ามากกว่าหรือน้อยกว่า

เป็นการทำงานพื้นฐานอีกอย่างหนึ่งที่พบเจอได้เสมอนั่นคือ การเปรียบเทียบค่า 2 ค่า เพื่อหาค่าที่มากกว่าหรือน้อยกว่า ให้ดูตัวอย่างที่ 17-7 การหาค่ามากกว่าหรือน้อยกว่าเพิ่มเติม

โค้ด VB 2005 และ VC# 2005 ที่ 17-7 การหาค่ามากกว่าหรือน้อยกว่า

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim result As Double = 0.0

        result = Max(10, 20)
        MessageBox.Show("ค่ามากกว่า : " & result.ToString(),
"ผลการตรวจสอบค่ามากกว่า")

        result = Min(30, 40)
        MessageBox.Show("ค่าน้อยกว่า : " & result.ToString(),
"ผลการตรวจสอบค่าน้อยกว่า")
    End Sub

    Private Function Max(ByVal a As Double, ByVal b As
Double) As Double
        If a >= b Then
```

VC# 2005 (Form1.cs)

```
    private void Form1_Load(object sender, EventArgs e)
    {
        double result = 0.0;
        result = Max(10, 20);
        MessageBox.Show("ค่ามากกว่า : " + result.ToString(),
"ผลการตรวจสอบค่ามากกว่า");

        result = Min(30, 40);
        MessageBox.Show("ค่าน้อยกว่า : " + result.ToString(),
"ผลการตรวจสอบค่าน้อยกว่า");
    }

    private double Max(double a, double b)
    {
        if (a >= b)
        {
            return a;
        }
        else
        {
```

```
        Return a
      Else
        Return b
      End If
    End Function

    Private Function Min(ByVal a As Double, ByVal b As
Double) As Double
      If a <= b Then
        Return a
      Else
        Return b
      End If
    End Function
End Class
```

```
        return b;
      }
    }

    private double Min(double a, double b)
    {
      if (a <= b)
      {
        return a;
      }
      else
      {
        return b;
      }
    }
```

การค้นหาข้อมูลแบบ Sequential

การค้นหาข้อมูลแบบ Sequential (Sequential Search) หมายถึง การค้นหาตามลำดับ เราจะใช้การค้นหาแบบนี้เพื่อตรวจสอบว่า สมาชิกที่เราสนใจเก็บอยู่ในอาร์เรย์หรือไม่ ให้ดูตัวอย่างที่ 17-8 การค้นหาข้อมูลแบบ Sequential เพิ่มเติม

โค้ด VB 2005 ที่ 17-8 การค้นหาข้อมูลแบบ Sequential (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim testNumber() As Integer = {0, 5, 4, 9, 3, 7, 6, 1, 2, 8}
        Dim result As Integer = SequentialSearch(testNumber, 2)
        If result = -1 Then
            MessageBox.Show("ไม่พบหมายเลขที่คุณระบุ", "ผลการค้นหา")
        Else
            MessageBox.Show("พบหมายเลขที่คุณระบุ อยู่ในตำแหน่งที่ : " & result, "ผลการค้นหา")
        End If
    End Sub

    Private Function SequentialSearch(ByVal ArrToSearch() As Integer, ByVal ItemToSearch As Integer) As Integer
        Dim Success As Integer = -1
        For Success = 0 To ArrToSearch.Length - 1
            If ArrToSearch(Success) = ItemToSearch Then
                Return Success
                Exit Function
            End If
        Next

        Return -1
    End Function
End Class
```

โค้ด VC# 2005 ที่ 17-8 การค้นหาข้อมูลแบบ Sequential (Form1.cs)

```
private void Form1_Load(object sender, EventArgs e)
{
    int[] testNumber = new int[] { 0, 5, 4, 9, 3, 7, 6, 1, 2, 8 };
    int result = SequentialSearch(testNumber, 2);
    if (result == -1)
    {
        MessageBox.Show("ไม่พบหมายเลขที่คุณระบุ", "ผลการค้นหา");
    }
    else
    {
        MessageBox.Show("พบหมายเลขที่คุณระบุ อยู่ในตำแหน่งที่ : " + result, "ผลการค้นหา");
    }
}

private int SequentialSearch(int[] ArrToSearch, int ItemToSearch)
{
    int Success = -1;
    for (Success = 0; Success <= ArrToSearch.Length - 1; Success++)
    {
        if (ArrToSearch[Success] == ItemToSearch)
        {
            return Success;
        }
    }

    return -1;
}
```

วิธีการดูโค้ดชุดนี้คือ ผู้เขียนสร้างฟังก์ชันที่ชื่อว่า SequentialSearch() ขึ้นมา โดยต้องการพารามิเตอร์ 2 ตัวคือ

- พารามิเตอร์ ArrToSearch มี Type เป็นอาร์เรย์ของ Integer (int) หมายถึง ตัวแปรอาร์เรย์ที่ต้องการค้นหา มีสมาชิกเป็นเลขจำนวนเต็ม Integer (int) เท่านั้น
- พารามิเตอร์ ItemToSearch มี Type เป็น Integer (int) เช่นกัน หมายถึง ค่าที่ต้องการค้นหา

ถ้าฟังก์ชัน SequentialSearch() ค้นพบจะคืนค่าเป็นตำแหน่งที่เจอ แต่ถ้าไม่พบจะคืนค่าเป็น -1 วิธีการก็คือ สั่งให้วนลูปตั้งแต่สมาชิกตัวแรก (ลำดับอ้างอิง 0) ไปจนถึงสมาชิกลำดับสุดท้าย (คุณสมบัติ Length-1) ที่อยู่ในตัวแปรอาร์เรย์ ArrToSearch ในแต่ละรอบของการวนลูป จะนำค่าของสมาชิกปัจจุบัน (ค่าของตัวแปร Success) ไปเปรียบเทียบกับพารามิเตอร์ ItemToSearch ว่าถ้าสมาชิกปัจจุบันมีค่าเท่ากับพารามิเตอร์ ItemToSearch แล้ว ตีความได้ว่าสมาชิกปัจจุบันคือ สิ่งที่ต้องการค้นหา ก็จะคืนค่าของตัวแปร Success ออกไป เพราะว่าตัวแปร Success เก็บลำดับของสมาชิกปัจจุบันไว้นั่นเอง แต่ถ้าไม่พบก็จะคืนค่าเป็น -1

ในเหตุการณ์ Form_Load() ก็จะสร้างอาร์เรย์ของ Integer (int) ที่ชื่อว่า testNumber กำหนดสมาชิกไว้ 10 ตัวคือ เลข 0 ถึง 9 สลับตำแหน่งกัน

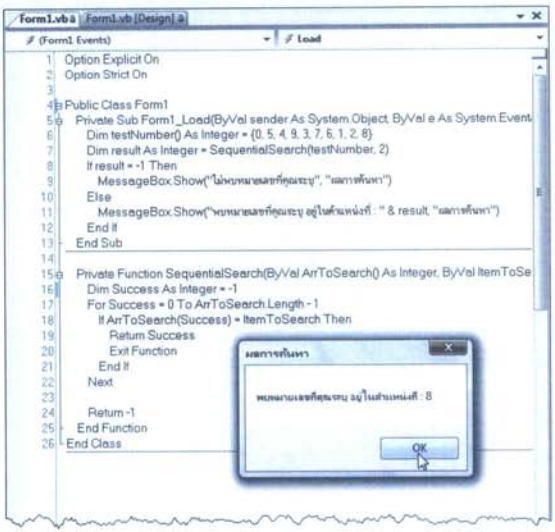
VB 2005

```
Dim testNumber() As Integer = {0, 5, 4, 9, 3, 7, 6, 1, 2, 8}
Dim result As Integer = SequentialSearch(testNumber, 2)
If result = -1 Then
    MessageBox.Show("ไม่พบหมายเลขที่คุณระบุ", "ผลการค้นหา")
Else
    MessageBox.Show("พบหมายเลขที่คุณระบุ อยู่ในตำแหน่งที่ : " & result, "ผลการค้นหา")
End If
```

VC# 2005

```
int[] testNumber = new int[] { 0, 5, 4, 9, 3, 7, 6, 1, 2, 8 };
int result = SequentialSearch(testNumber, 2);
if (result == -1)
{
    MessageBox.Show("ไม่พบหมายเลขที่คุณระบุ", "ผลการค้นหา");
}
else
{
    MessageBox.Show("พบหมายเลขที่คุณระบุ อยู่ในตำแหน่งที่ : " + result, "ผลการค้นหา");
}
```

จากนั้นลองค้นหาเลข 2 ว่าอยู่ในตัวแปรอาร์เรย์ testNumber หรือไม่ ตรวจสอบได้จากค่าที่ส่งกลับมาจากฟังก์ชัน SequentialSearch() นั่นเอง ผลการทำงาน แสดงดังรูปที่ 17-12



รูปที่ 17-12 ผลการค้นหาเลข 2 ในตัวแปรอาร์เรย์ testNumber

จากรูปที่ 17-12 เลข 2 คือสมาชิกตัวที่ 9 (ลำดับอ้างอิง 8) ของตัวแปรอาร์เรย์ testNumber

การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูล

อัลกอริธึมที่เราคุ้นเคยในระหว่างการเรียนสาขาคอมพิวเตอร์นั่นคือ การเรียงลำดับข้อมูล เป็นอัลกอริธึมที่คลาสสิกไม่แพ้การเขียนโปรแกรมแสดงคำว่า Hello World ตั้งชื่อคลาสนี้ว่า SortTools เก็บอยู่ในเนมสเปซ

GAF.Algorithm

ผู้เขียนนำเสนอการเรียงลำดับ 2 แบบคือ

1. Bubble Sort
2. Insertion Sort

การเรียงลำดับแบบ Bubble Sort

หลักการของการเรียงลำดับแบบ Bubble Sort คือ การเปรียบเทียบข้อมูลปัจจุบันกับข้อมูลตัวถัดไป ซึ่งเกิดการกระทำอย่างใดอย่างหนึ่งคือ

1. ถ้าข้อมูลปัจจุบันน้อยกว่า (หรือมากกว่าแล้วแต่กรณี) ข้อมูลตัวถัดไปให้สลับค่า (swap data) กัน
2. แต่ถ้าเป็นอย่างอื่นให้เฉยไว้

การเปรียบเทียบอาจจะเริ่มจากข้อมูลแรกสุดกับข้อมูลถัดไป หรือข้อมูลสุดท้ายกับข้อมูลรองสุดท้ายก็ได้ ดังแสดงโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะ Bubble Sort (GAF.Algorithm.SortTools.BubbleSort.vb)

```
Option Explicit On
Option Strict On

Namespace GAF.Algorithm
    Partial Public Class SortTools
เมธอด BubbleSort() ทำหน้าที่เรียงลำดับข้อมูลที่อยู่ในอาร์เรย์ชนิด Double
ต้องการพารามิเตอร์ 1 ตัวคือ ArrToSort หมายถึง อาร์เรย์ที่ต้องการเรียงลำดับข้อมูล
        Public Shared Sub BubbleSort(ByRef ArrToSort() As Double)
            Dim _i As Integer = 0
            Dim _j As Integer = 0
            Do While (_j < (ArrToSort.Length - 1))              ให้วนลูปตั้งแต่ตัวแรกจนถึงตัวสุดท้าย
                _i = 0
                Do While (_i < (ArrToSort.Length - 1))          ให้วนลูปตั้งแต่ตัวแรกจนถึงตัวสุดท้าย
                    If (ArrToSort(_i) > ArrToSort((_i + 1))) Then   ถ้าข้อมูลปัจจุบันมากกว่าตัวถัดไปแล้ว
                        Helper.SwapData(ArrToSort, _i, (_i + 1))    เรียกเมธอด SwapData() ทำงาน เพื่อสลับข้อมูล
                    End If
                    _i += 1
                Loop
                _j += 1
            Loop
            _i = 0
        End Sub
    End Class
End Namespace
```

โค้ด VC# 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะ Bubble Sort (GAF.Algorithm.SortTools.BubbleSort.cs)

```
namespace GAF.Algorithm
{
    public partial class SortTools
    {
//เมธอด BubbleSort() ทำหน้าที่เรียงลำดับข้อมูลที่อยู่ในอาร์เรย์ชนิด double
//ต้องการพารามิเตอร์ 1 ตัวคือ ArrToSort หมายถึง อาร์เรย์ที่ต้องการเรียงลำดับข้อมูล
        public static void BubbleSort(ref double[] ArrToSort)
        {
            int _i = 0;
            int _j = 0;
            while (_j < (ArrToSort.Length - 1))                    //ให้วนลูปตั้งแต่ตัวแรกจนถึงตัวสุดท้าย
            {
                _i = 0;
                while (_i < (ArrToSort.Length - 1))                //ให้วนลูปตั้งแต่ตัวแรกจนถึงตัวสุดท้าย
                {
                    if (ArrToSort[_i] > ArrToSort[(_i + 1)])       //ถ้าข้อมูลปัจจุบันมากกว่าตัวถัดไปแล้ว
                    {
                        Helper.SwapData(ref ArrToSort, _i, _i + 1);  //เรียกเมธอด SwapData() ทำงาน เพื่อสลับข้อมูล
                    }
                    _i++;
                }
                _j++;
            }
            _j = 0;
        }
    }
}
```

เมธอด BubbleSort() ข้างต้น เป็นการเปรียบเทียบข้อมูลตัวสุดท้ายกับรองสุดท้าย โดยวนลูปมาจนถึงข้อมูลตัวแรก ในแต่ละรอบของการวนลูปก็จะตรวจสอบว่า ถ้าข้อมูลปัจจุบันมากกว่าข้อมูลถัดไปให้สลับค่ากัน

แต่ถ้าคุณต้องการเปรียบเทียบข้อมูลตัวแรกกับตัวถัดไป ให้วนลูปไปจนถึงตัวสุดท้าย ดังแสดงโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะ Bubble Sort แบบที่ 2 (GAF.Algorithm.SortTools.BubbleSort.vb)

```
Option Explicit On
Option Strict On

Namespace GAF.Algorithm
    Partial Public Class SortTools
        Public Shared Sub BubbleSort(ByRef ArrToSort() As Double)
            Dim _i As Integer = (ArrToSort.Length - 1)
```

```
        Dim _j As Integer = 1

        Do While (_i >= 0)
            _j = 1
            Do While (_j <= _i)
                If (ArrToSort((_j - 1)) > ArrToSort(_j)) Then
                    Helper.SwapData(ArrToSort(_j), ArrToSort(_j - 1))
                End If
                _j += 1
            Loop
            _i -= 1
        Loop
    End Sub
End Class
End Namespace
```

โค้ด VC# 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะ Bubble Sort แบบที่ 2 (GAF.Algorithm.SortTools.BubbleSort.cs)

```
namespace GAF.Algorithm
{
    public partial class SortTools
    {
        public static void BubbleSort(ref double[] ArrToSort)
        {
            int _i = ArrToSort.Length - 1;
            int _j = 1;

            while (_i >= 0)
            {
                _j = 1;
                while (_j <= _i)
                {
                    if (ArrToSort[(_j - 1)] > ArrToSort[_j])
                    {
                        Helper.SwapData(ref ArrToSort[_j], ref ArrToSort[_j - 1]);
                    }
                    _j++;
                }
                _i--;
            }
        }
    }
}
```

การสลับค่ากันอยู่ในความรับผิดชอบของเมธอด SwapData() เก็บอยู่ในคลาส Helper ดังแสดงโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะคลาส Helper (GAF.Algorithm.Helper.vb)

```
Option Explicit On
Option Strict On
Imports System.Text

Public Class Helper
'เมธอด SwapData() ทำหน้าที่สลับข้อมูลแบบรับพารามิเตอร์ 2 ตัว
    Public Shared Sub SwapData(ByRef a As Double, ByRef b As Double)
        Dim c As Double = a
        a = b
        b = c
    End Sub

'เมธอด SwapData() ทำหน้าที่สลับข้อมูลแบบรับพารามิเตอร์ 3 ตัว
    Public Shared Sub SwapData(ByRef ArrToSwap() As Double, ByVal a As Integer, ByVal b As Integer)
        Dim c As Double = ArrToSwap(a)
        ArrToSwap(a) = ArrToSwap(b)
        ArrToSwap(b) = c
    End Sub

'เมธอด PrintData() ทำหน้าที่พิมพ์ข้อมูลที่อยู่ในอาร์เรย์ชนิด Double
    Public Shared Function PrintData(ByVal ArrToPrint() As Double) As String
        Dim _i As Integer = 0
        Dim _sw As New StringBuilder()
        _sw.Remove(0, _sw.Length)

        For _i = 0 To ArrToPrint.Length - 1
            _sw.Append(ArrToPrint(_i) & " ")
        Next

        Dim _Data As String = ""
        _Data = _sw.ToString()
        Return _Data.TrimEnd()
    End Function
End Class
```

โค้ด VC# 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะคลาส Helper (GAF.Algorithm.Helper.cs)

```
public class Helper
{
//เมธอด SwapData() ทำหน้าที่สลับข้อมูลแบบรับพารามิเตอร์ 2 ตัว
    public static void SwapData(ref double a, ref double b)
    {
        double c = a;
        a = b;
        b = c;
    }

//เมธอด SwapData() ทำหน้าที่สลับข้อมูลแบบรับพารามิเตอร์ 3 ตัว
    public static void SwapData(ref double[] ArrToSwap, int a, int b)
    {
        double c = ArrToSwap[a];
        ArrToSwap[a] = ArrToSwap[b];
        ArrToSwap[b] = c;
    }

//เมธอด PrintData() ทำหน้าที่พิมพ์ข้อมูลที่อยู่ในอาร์เรย์ชนิด double
    public static string PrintData(double[] ArrToPrint)
    {
        int _i = 0;
        StringBuilder _sw = new StringBuilder();
        _sw.Remove(0, _sw.Length);

        for (_i = 0; _i <= ArrToPrint.Length - 1; _i++)
        {
            _sw.Append(ArrToPrint[_i] + " ");
        }

        string _Data = "";
        _Data = _sw.ToString();
        return _Data.TrimEnd(null);
    }
}
```

คลาส Helper ประกอบด้วย 3 เมธอดกล่าวคือ

- เมธอด SwapData() แบบรับพารามิเตอร์ 2 ตัว ทำหน้าที่สลับข้อมูล
- เมธอด SwapData() แบบรับพารามิเตอร์ 3 ตัว ทำหน้าที่สลับข้อมูล
- เมธอด PrintData() ทำหน้าที่พิมพ์ข้อมูลที่เก็บอยู่ในอาร์เรย์ชนิด Double

การเรียงลำดับแบบ Insertion Sort

หลักการของการเรียงลำดับแบบ Insertion Sort คือ หยิบข้อมูลจากทางขวามาเปรียบเทียบ เพื่อหาที่แทรกข้อมูล วนลูปจนครบข้อมูลทุกตัว ดังแสดงโค้ดต่อไปนี้

โค้ด VB 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะ Insertion Sort (GAF.Algorithm.SortTools.InsertionSort.vb)

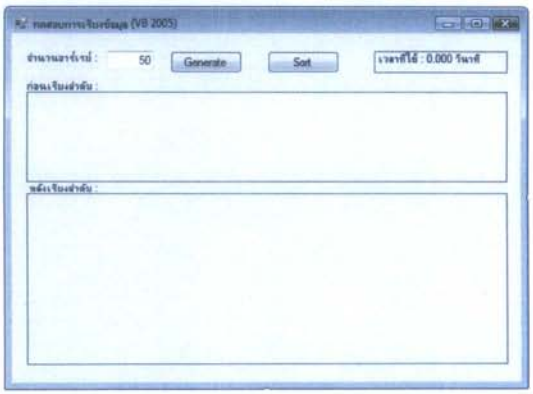
```
Option Explicit On
Option Strict On

Namespace GAF.Algorithm
    Partial Public Class SortTools
'เมธอด InsertionSort() ทำหน้าที่เรียงลำดับข้อมูลที่อยู่ในอาร์เรย์ชนิด Double
'ต้องการพารามิเตอร์ 1 ตัวคือ ArrToSort() หมายถึง อาร์เรย์ที่ต้องการเรียงลำดับ
        Public Shared Sub InsertionSort(ByVal ArrToSort() As Double)
            Dim _i As Integer = 0
            Dim _j As Integer = 0
            Dim _Max As Double = 0.0

            _i = 1
            Do While (_i < ArrToSort.Length)          ให้วนลูปตั้งแต่ตัวแรกจนถึงตัวสุดท้าย
                _Max = ArrToSort(_i)                  ให้ข้อมูลตัวปัจจุบันคือ ค่าสูงสุด
                _j = (_i - 1)
                Do While (_j >= 0)                    ให้วนลูปตั้งแต่ตัวสุดท้ายจนถึงตัวแรก
                    If (ArrToSort(_j) <= _Max) Then   ถ้าข้อมูลปัจจุบันน้อยกว่าหรือเท่ากับค่าที่เก็บอยู่ในฟิลด์ _Max แล้ว
                        Exit Do                       ให้ออกจากลูป
                    End If
                    ArrToSort((_j + 1)) = ArrToSort(_j)   ให้ถ่ายค่าจากข้อมูลปัจจุบันไปยังข้อมูลถัดไป
                        _j -= 1
                Loop
                ArrToSort((_j + 1)) = _Max            ให้ถ่ายค่าสูงสุดไปยังข้อมูลตัวถัดไป
                _j += 1
            Loop
        End Sub
    End Class
End Namespace
```

โค้ด VC# 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูลเฉพาะ Insertion Sort (GAF.Algorithm.SortTools.InsertionSort.cs)

```
namespace GAF.Algorithm
{
    public partial class SortTools
    {
//เมธอด InsertionSort() ทำหน้าที่เรียงลำดับข้อมูลที่อยู่ในอาร์เรย์ชนิด double
//ต้องการพารามิเตอร์ 1 ตัวคือ ArrToSort() หมายถึง อาร์เรย์ที่ต้องการเรียงลำดับ
        public static void InsertionSort(ref double[] ArrToSort)
        {
            int _i = 0;
            int _j = 0;
            double _Max = 0.0;

            _i = 1;
            while (_i < ArrToSort.Length)                    //ให้วนลูปตั้งแต่ตัวแรกจนถึงตัวสุดท้าย
            {
                _Max = ArrToSort[_i];                        //ให้ข้อมูลตัวปัจจุบันคือ ค่าสูงสุด
                _j = _i - 1;
                while (_j >= 0)                              //ให้วนลูปตั้งแต่ตัวสุดท้ายจนถึงตัวแรก
                {
                if (ArrToSort[_j] <= _Max)                   //ถ้าข้อมูลปัจจุบันน้อยกว่าหรือเท่ากับค่าที่เก็บอยู่ในฟิลด์ _Max แล้ว
                    {
                        break;                               //ให้ออกจากลูป
                    }
                    ArrToSort[(_j + 1)] = ArrToSort[_j];     //ให้ถ่ายค่าจากข้อมูลปัจจุบันไปยังข้อมูลถัดไป
                    _j--;
                }
                ArrToSort[(_j + 1)] = _Max;                  //ให้ถ่ายค่าสูงสุดไปยังข้อมูลตัวถัดไป
                _i++;
            }
        }
    }
```

ส่วนการใช้งานคลาส SortTools อยู่ใน Form1 ให้คุณออกแบบ ดังรูปที่ 17-13

รูปที่ 17-13 ฟอร์มในขณะออกแบบ



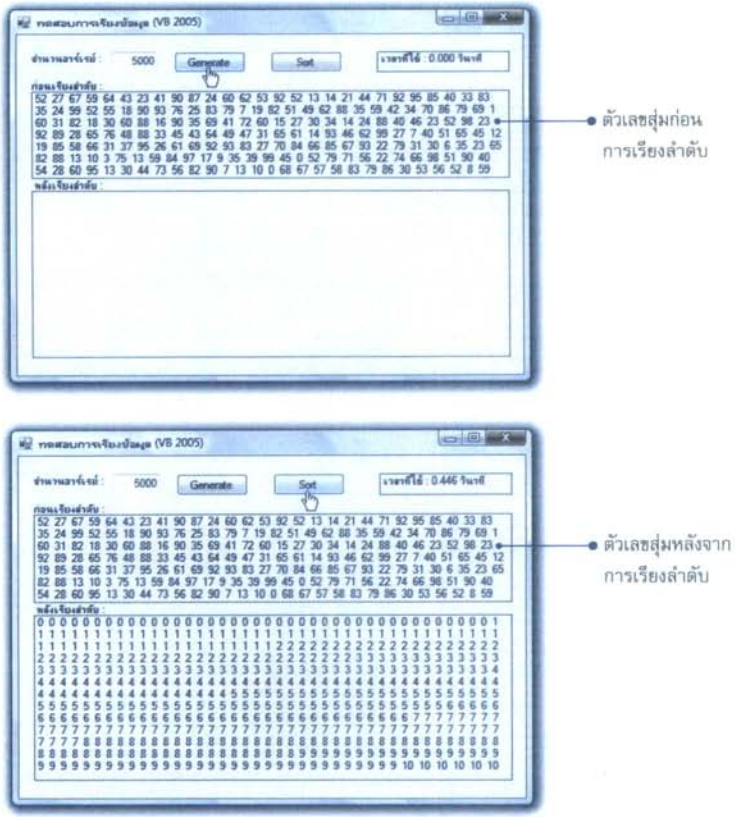
โค้ด VB 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูล (Form1.vb)

```
Option Explicit On
Option Strict On
Imports System.Text
Imports GAF.Algorithm

Public Class Form1
    Dim rdNumber() As Double

    Private Sub cmdGenerate_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdGenerate.Click
        Dim i As Integer
        Dim tmp As String = ""
        Dim rd As New Random()
        Dim sb As New StringBuilder()
        sb.Remove(0, sb.Length)

        ReDim rdNumber(CInt(txtTotal.Text) - 1)
        For i = 0 To CInt(txtTotal.Text) - 1
            tmp = rd.Next(100).ToString()

            rdNumber(i) = CDbl(tmp)
            sb.Append(tmp & " ")
        Next

        tmp = sb.ToString()
        lblBefore.Text = tmp
    End Sub

    Private Sub cmdSort_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSort.Click
        Dim sw As New Stopwatch
        sw.Reset()
        sw.Start()

        SortTools.BubbleSort(rdNumber)

        sw.Stop()
        Dim SortTime As Double = CDbl(sw.ElapsedMilliseconds) / 1000
        lblTime.Text = "เวลาที่ใช้ : " & SortTime.ToString() & " วินาที"

        lblAfter.Text = ""
        lblAfter.Text = Helper.PrintData(rdNumber)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 17-9 การสร้างคลาสที่ทำหน้าที่เรียงลำดับข้อมูล (Form1.cs)

```
    double[] rdNumber;

    private void cmdGenerate_Click(object sender, EventArgs e)
    {
        string tmp = "";
        Random rd = new Random();
        StringBuilder sb = new StringBuilder();
        sb.Remove(0, sb.Length);

        rdNumber = new double[Convert.ToInt32(txtTotal.Text)];
        for (int i = 0; i <= Convert.ToInt32(txtTotal.Text) - 1; i++)
        {
            tmp = rd.Next(100).ToString();

            rdNumber[i] = Convert.ToDouble(tmp);
            sb.Append(tmp + " ");
        }

        tmp = sb.ToString();
        lblBefore.Text = tmp;
    }

    private void cmdSort_Click(object sender, EventArgs e)
    {
        Stopwatch sw = new Stopwatch();
        sw.Reset();
        sw.Start();

        SortTools.BubbleSort(ref rdNumber);

        sw.Stop();
        double SortTime = Convert.ToDouble(sw.ElapsedMilliseconds) / 1000;
        lblTime.Text = "เวลาที่ใช้ : " + SortTime.ToString() + " วินาที";

        lblAfter.Text = "";
        lblAfter.Text = Helper.PrintData(rdNumber);
    }
}
```

รูปที่ 17-14
ผลการรัน
ตัวอย่างที่ 17-9



จากรูปที่ 17-14 ให้คลิกปุ่ม Generate เพื่อสุ่มตัวเลขขึ้นมาตามจำนวนที่ระบุไว้ เห็นได้ว่าผู้เขียนสุ่มตัวเลขขึ้นมา 5000 ตัว แล้วคลิกปุ่ม Sort เพื่อเรียงลำดับตัวเลขที่สุ่มได้ ผู้เขียนกำหนดค่าเริ่มต้นให้เรียงลำดับด้วยวิธี Bubble Sort ซึ่งอยู่ในความรับผิดชอบของเมธอด BubbleSort() ของคลาส SortTools

การสุ่มตัวเลขอยู่ในเหตุการณ์ cmdGenerate_Click() หลักการก็คือ สั่งให้วนลูปตามจำนวนรอบที่ระบุในคอนโทรล txtTotal ในแต่ละรอบของการวนลูปก็จะใช้ออบเจ็กต์ Random ที่ชื่อว่า rn สุ่มตัวเลขที่มีค่าไม่เกิน 100 (เมธอด Next(100))

VB 2005

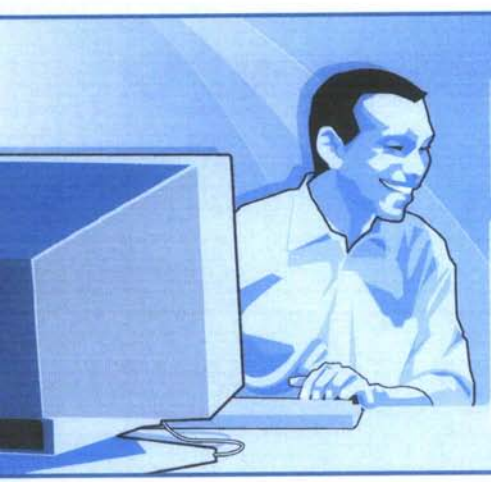
```
ReDim rdNumber(CInt(txtTotal.Text) - 1)
For i = 0 To CInt(txtTotal.Text) - 1
    tmp = rd.Next(100).ToString()

    rdNumber(i) = CDbl(tmp)
    sb.Append(tmp & " ")
Next
```

VC# 2005

```
rdNumber = new double[Convert.ToInt32(txtTotal.Text)];
for (int i = 0; i <= Convert.ToInt32(txtTotal.Text) - 1; i++)
{
    tmp = rd.Next(100).ToString();

    rdNumber[i] = Convert.ToDouble(tmp);
    sb.Append(tmp + " ");
}
```

สรุปท้ายบท

การศึกษาโครงสร้างข้อมูลและอัลกอริธึม ช่วยให้คุณผู้อ่านสามารถเรียบเรียงลำดับความคิดได้เป็นอย่างดี ผู้เขียนขอแนะนำให้คุณผู้อ่านศึกษาเพิ่มเติมจากตำราที่กล่าวถึงเนื้อหาด้านนี้โดยเฉพาะ ซึ่งจะมีเนื้อหาและคำอธิบายละเอียดกว่าที่ผู้เขียนนำเสนอ



CHAPTER 18

ดีลีเกต

บทนำ

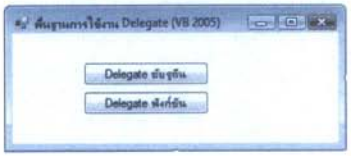
ดีลีเกต (Delegate) อาจจะเป็นคำที่ไม่คุ้นเคยเท่าใดนัก อาจจะเป็นฟีเจอร์ของ .NET Framework ที่หลายๆ ท่านยังไม่รู้จักและไม่เคยใช้งาน คุณจะได้ศึกษาในบทนี้ ซึ่งขอบเขตของเนื้อหาที่นำเสนอเป็นเพียงพื้นฐานการใช้งานดีลีเกตเท่านั้น

พื้นฐานการใช้งานดีลีเกต (Delegate)

ในการพัฒนาแอพพลิเคชันที่ต้องการความยืดหยุ่นมากๆ จะมีการใช้ดีลีเกตเข้ามาช่วยโหลดซับรูทีนหรือฟังก์ชันแทนการเรียกใช้โดยตรง การทำความรู้จักกับดีลีเกตที่เร็วที่สุด ผู้เขียนคิดว่าขอให้ดูตัวอย่างที่ 18-1 พื้นฐานการใช้งานดีลีเกตดีกว่า

ให้คุณออกแบบฟอร์ม ดังรูปที่ 18-1

รูปที่ 18-1 ฟอร์มในขณะออกแบบ



โค้ด VB 2005 ที่ 18-1 พื้นฐานการใช้งานดีลีเกต (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Public Delegate Sub dlgSub()
    Public Delegate Function dlgFunction(ByVal str As String) As String

    Private Sub cmdTestSub_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdTestSub.Click
        Dim testSub As dlgSub
        testSub = AddressOf ShowMessage
        testSub()
    End Sub

    Private Sub cmdTestFunction_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdTestFunction.Click
        Dim testFunction As dlgFunction
        testFunction = AddressOf ShowMessageWithName

        Dim result As String = testFunction("ศุภชัย สมพานิช")
        MessageBox.Show(result, "ฟังก์ชันทำงาน")
    End Sub

    Public Sub ShowMessage()
        MessageBox.Show("สวัสดีครับ", "ซับรูทีนทำงาน")
    End Sub

    Public Function ShowMessageWithName(ByVal strName As String) As String
        Dim tmp As String
        tmp = "สวัสดีครับ คุณ" & strName
        Return tmp
    End Function
End Class
```

โค้ด VC# 2005 ที่ 18-1 พื้นฐานการใช้งานดีลีเกต (Form1.cs)

```
    public delegate void dlgSub();
    public delegate string dlgFunction(string str);

    private void cmdTestSub_Click(object sender, EventArgs e)
    {
        dlgSub testSub= new dlgSub(this.ShowMessage);
        testSub();
    }

    private void cmdTestFunction_Click(object sender, EventArgs e)
    {
        dlgFunction testFunction= new dlgFunction(this.ShowMessageWithName);

        string result = testFunction("ศุภชัย สมพานิช");
```

```
        MessageBox.Show(result, "ฟังก์ชันทำงาน");
    }

    public void ShowMessage()
    {
        MessageBox.Show("สวัสดีครับ", "ซับรูทีนทำงาน");
    }

    public string ShowMessageWithName(string strName)
    {
        string tmp;
        tmp = "สวัสดีครับ คุณ" + strName;

        return tmp;
    }
```

ผู้เขียนสร้างซับรูทีนที่ชื่อว่า ShowMessage() และสร้างฟังก์ชันที่ชื่อว่า ShowMessageWithName() ขึ้นมา มี Signature คือ ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น String (string) และคืนค่าเป็น String (string)

VB 2005

```
Public Sub ShowMessage()
    MessageBox.Show("สวัสดีครับ", "ซับรูทีนทำงาน")
End Sub

Public Function ShowMessageWithName(ByVal strName As String) As String
    Dim tmp As String
    tmp = "สวัสดีครับ คุณ" & strName
    Return tmp
End Function
```

VC# 2005

```
    public void ShowMessage()
    {
        MessageBox.Show("สวัสดีครับ", "ซับรูทีนทำงาน");
    }

    public string ShowMessageWithName(string strName)
    {
        string tmp;
        tmp = "สวัสดีครับ คุณ" + strName;

        return tmp;
    }
```

ถ้าคุณต้องการเรียกใช้ซับรูทีน ShowMessage() หรือฟังก์ชัน ShowMessageWithName() โดยปกติแล้วก็จะระบุชื่อซับรูทีนหรือชื่อฟังก์ชัน ณ จุดที่ต้องการเรียกใช้งาน แต่ก็มีบางกรณีเช่นกันที่เราไม่ทราบว่าซับรูทีนหรือฟังก์ชันใดจะถูกสั่งให้ทำงาน ทราบแต่เพียง Signature เท่านั้น ความต้องการแบบนี้ต้องอาศัยดีลีเกตเข้ามาช่วยเหลือ

ดีลีเกตทำหน้าที่โหลดซับรูทีนหรือฟังก์ชันขึ้นมาเพื่อสั่งให้ทำงาน ส่งผลให้ซับรูทีนหรือฟังก์ชันจะอยู่ในฐานะพารามิเตอร์ คุณเคยใช้ตัวแปรในฐานะพารามิเตอร์มาแล้ว แต่ดีลีเกตจะทำให้ซับรูทีนหรือฟังก์ชันอยู่ในฐานะพารามิเตอร์บ้าง

ต่อมาผู้เขียนสร้างดีลีเกตขึ้นมา 2 ตัวชื่อว่า dlgSub() ทำหน้าที่โหลดซับรูทีน และดีลีเกตที่ชื่อว่า dlgFunction ทำหน้าที่โหลดฟังก์ชัน ดีลีเกตที่สามารถโหลดซับรูทีนหรือฟังก์ชันเป้าหมายที่เราสนใจ ต้องดูที่ Signature ของดีลีเกตเป็นสำคัญกล่าวคือ

VB 2005

```
Public Delegate Sub dlgSub()
Public Delegate Function dlgFunction(ByVal str As String) As String
```

VC# 2005

```
public delegate void dlgSub();
public delegate string dlgFunction(string str);
```

- ดีลีเกตที่ชื่อว่า dlgSub มี Signature คือ ต้องไม่มีพารามิเตอร์
- ส่วนดีลีเกตที่ชื่อว่า dlgFunction มี Signature คือ ต้องมีพารามิเตอร์ 1 ตัวมี Type เป็น String (string) และคืนค่าเป็น String (string)

เห็นได้ว่าดีลีเกต dlgSub สามารถโหลดซับรูทีน ShowMessage() ได้ ส่วนดีลีเกต dlgFunction สามารถโหลดฟังก์ชัน ShowMessageWithName() ได้ เพราะว่ามี Signature เหมือนกัน

สมมติว่าที่เหตุการณ์ cmdTestSub_Click() คือ จุดเรียกใช้ซับรูทีนจึงสร้างตัวแปรที่ชื่อว่า testSub ขึ้นมามี Type เป็นดีลีเกต dlgSub ก็จะกำหนดให้ดีลีเกต testSub ชี้ไปยังซับรูทีน ShowMessage() ท้ายที่สุดสั่งให้ดีลีเกต testSub ทำงาน ดังรูปที่ 18-2

VB 2005

```
Private Sub cmdTestSub_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdTestSub.Click
    Dim testSub As dlgSub
    testSub = AddressOf ShowMessage
    testSub()
End Sub
```

VC# 2005

```
private void cmdTestSub_Click(object sender, EventArgs e)
{
    dlgSub testSub= new dlgSub(this.ShowMessage);
    testSub();
}
```

รูปที่ 18-2 ชับรูทีน ShowMessage() ทำงานผ่านทางดีลีเกต dlgSub

ส่วนการสั่งให้ฟังก์ชัน ShowMessageWithName() ทำงาน อยู่ในเหตุการณ์ cmdTestFunction_Click() ก็จะทำงานผ่านทางดีลีเกต dlgFunction ที่ชื่อว่า testFunction ดังรูปที่ 18-3

VB 2005

```
Private Sub cmdTestFunction_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
cmdTestFunction.Click
    Dim testFunction As dlgFunction
    testFunction = AddressOf ShowMessageWithName

    Dim result As String = testFunction("ศุภชัย สมพานิช")
    MessageBox.Show(result, "ฟังก์ชันทำงาน")
End Sub
```

VC# 2005

```
private void cmdTestFunction_Click(object sender, EventArgs e)
{
    dlgFunction testFunction= new dlgFunction(this.ShowMessageWithName);

    string result = testFunction("ศุภชัย สมพานิช");
    MessageBox.Show(result, "ฟังก์ชันทำงาน");
}
```

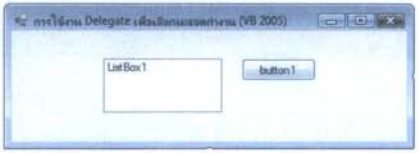
รูปที่ 18-3
ขับรูทีน ShowMessage WithName() ทำงานผ่านทาง ดีลีเกต dlgFunction

เห็นได้ว่าเมื่อดีลีเกต testFunction มี Signature ต้องการพารามิเตอร์ 1 ตัว มี Type เป็น String (string) คุณก็ต้องส่งค่าให้กับดีลีเกต testFunction ตาม Signature ดังกล่าวด้วย รวมถึงต้องสร้างตัวแปรที่มี Type เป็น String (string) มารอรับค่าส่งกลับจากดีลีเกต testFunction อีกด้วย

การใช้ดีลีเกตทำหน้าที่เลือกเมธอดทำงาน

ดีลีเกตสามารถสั่งให้เมธอดต่างๆ ที่อยู่ในคลาสของคุณทำงานได้เช่นกัน ให้คุณออกแบบฟอร์มดังรูปที่ 18-4

รูปที่ 18-4
ฟอร์มในขณะออกแบบ



โค้ด VB 2005 และ VC# 2005 ที่ 18-2 การใช้ดีลีเกตทำหน้าที่เลือกเมธอดทำงาน

VB 2005 (Sample.vb)

```
Option Explicit On
Option Strict On

Public Class Sample
    Public Shared Sub MethodA(ByVal arg As Double)
        MessageBox.Show("เมธอด A ทำงาน " & arg)
    End Sub

    Public Shared Sub MethodB(ByVal arg As Double)
        MessageBox.Show("เมธอด B ทำงาน " & arg)
    End Sub
End Class
```

VC# 2005 (Sample.cs)

```
public class Sample
{
    public static void MethodA(double arg)
    {
        MessageBox.Show("เมธอด A ทำงาน " + arg);
    }

    public static void MethodB(double arg)
    {
        MessageBox.Show("เมธอด B ทำงาน "+ arg);
    }
}
```

ผู้เขียนสร้างคลาสที่ชื่อว่า Sample ขึ้นมาประกอบด้วย 2 เมธอดที่ชื่อว่า MethodA() และเมธอด MethodB() ทั้ง 2 เมธอดต้องการพารามิเตอร์ 1 ตัวมี Type เป็น Double (double) เป็นเมธอดแบบ Shared (static)

โค้ด VB 2005 ที่ 18-2 การใช้ดีลีเกตทำหน้าที่เลือกเมธอดทำงาน (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Public Delegate Sub dlgSelect(ByVal para As Double)
    Dim Selecter As dlgSelect

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ListBox1.Items.Add("A")
        ListBox1.Items.Add("B")
    End Sub

    Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles button1.Click
        Select Case ListBox1.Text
            Case "A"
                Selecter = New dlgSelect(AddressOf Sample.MethodA)
            Case "B"
                Selecter = New dlgSelect(AddressOf Sample.MethodB)
            Case Else
                Selecter = New dlgSelect(AddressOf Sample.MethodA)
        End Select

        Selecter(555)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 18-2 การใช้ดีลีเกตทำหน้าที่เลือกเมธอดทำงาน (Form1.cs)

```
    public delegate void dlgSelect(double para);
    dlgSelect Selecter;

    private void Form1_Load(object sender, EventArgs e)
    {
        listBox1.Items.Add("A");
        listBox1.Items.Add("B");
    }

    private void button1_Click(object sender, EventArgs e)
    {
        string SelectedList = listBox1.Text;
        switch (SelectedList)
        {
```

```
            case "A":
                {
                    Selecter = new dlgSelect(Sample.MethodA);
                    break;
                }
            case "B":
                {
                    Selecter = new dlgSelect(Sample.MethodB);
                    break;
                }
            default:
                Selecter = new dlgSelect(Sample.MethodA);
                break;
        }
        Selecter(555);
    }
```

ถ้าคุณเลือกรายการ A เมธอดที่ทำงานคือ เมธอด MethodA() แต่ถ้าคุณเลือกรายการ B เมธอดที่ถูกสั่งให้ทำงานคือ เมธอด MethodB() ดังรูปที่ 18-5 และ 18-6

รูปที่ 18-5
กรณีเลือกรายการ A

รูปที่ 18-6
กรณีเลือกรายการ B

ผู้เขียนสร้างดีลีเกตที่ชื่อว่า dlgSelect มี Signature คือ ต้องการพารามิเตอร์ 1 ตัวมี Type เป็น Double (double) และสร้างตัวแปรที่ชื่อว่า Selecter มี Type เป็นดีลีเกต dlgSelect

VB 2005

```
Public Delegate Sub dlgSelect(ByVal para As Double)
Dim Selecter As dlgSelect
```

VC# 2005

```
public delegate void dlgSelect(double para);
dlgSelect Selecter;
```

ส่วนการทำงานอยู่ในเหตุการณ์ button1_Click() ก็จะตรวจสอบว่ารายการใดในคอนโทรล ListBox1 ถูกเลือก ก็จะสั่งให้เมธอด MethodA() หรือเมธอด MethodB() ทำงานผ่านทางดีลีเกตที่ชื่อว่า Selecter โดยที่ถ้าไม่มีการเลือกรายการใดๆ เลย จะกำหนดให้เมธอด MethodA() ทำงาน

VB 2005

```
Private Sub button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles button1.Click
    Select Case ListBox1.Text
        Case "A"
            Selecter = New dlgSelect(AddressOf Sample.MethodA)
        Case "B"
            Selecter = New dlgSelect(AddressOf Sample.MethodB)
        Case Else
            Selecter = New dlgSelect(AddressOf Sample.MethodA)
    End Select

    Selecter(555)
End Sub
```

VC# 2005

```
private void button1_Click(object sender, EventArgs e)
{
    string SelectedList = listBox1.Text;
    switch (SelectedList)
    {
        case "A":
            {
                Selecter = new dlgSelect(Sample.MethodA);
                break;
            }
        case "B":
            {
                Selecter = new dlgSelect(Sample.MethodB);
                break;
            }
        default:
            Selecter = new dlgSelect(Sample.MethodA);
            break;
    }

    Selecter(555);
}
```

การใช้งานดีลีเกตอ้างอิงมากกว่า 1 ฟังก์ชัน

เพราะความที่ดีลีเกตสนใจ Signature จึงไม่แปลกอะไรที่จะสามารถอ้างอิงได้มากกว่า 1 ฟังก์ชัน ให้ดูตัวอย่างที่ 18-3 การใช้งานดีลีเกตอ้างอิงมากกว่า 1 ฟังก์ชัน

โค้ด VB 2005 ที่ 18-3 การใช้งานดีลีเกตอ้างอิงมากกว่า 1 ฟังก์ชัน (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Public Delegate Sub dlgShow()

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim testDlg As dlgShow
        testDlg = AddressOf ShowCustomer
        testDlg()

        testDlg = AddressOf ShowProduct
        testDlg()
    End Sub

    Private Sub ShowCustomer()
        MessageBox.Show("ShowCustomer ทำงาน", "Multiple Methods")
    End Sub

    Private Sub ShowProduct()
        MessageBox.Show("ShowProduct ทำงาน", "Multiple Methods")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 18-3 การใช้งานดีลีเกตอ้างอิงมากกว่า 1 ฟังก์ชัน (Form1.cs)

```
    public delegate void dlgShow();

    private void Form1_Load(object sender, EventArgs e)
    {
        dlgShow testDlg = new dlgShow(this.ShowCustomer);
        testDlg();

        testDlg = new dlgShow(this.ShowProduct);
        testDlg();
    }

    private void ShowCustomer()
    {
        MessageBox.Show("ShowCustomer ทำงาน", "Multiple Methods");
```

```
    }

    private void ShowProduct()
    {
        MessageBox.Show("ShowProduct ทำงาน", "Multiple Methods");
    }
```

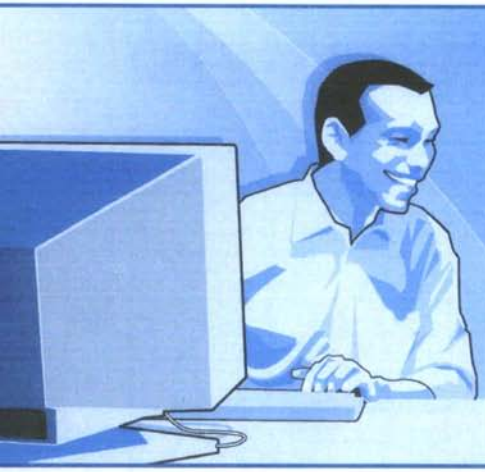
โค้ดข้างต้นสร้างดีลีเกตที่ชื่อว่า dlgShow ขึ้นมา มี 2 ซับรูทีนที่เป็นเป้าหมายคือ ซับรูทีน ShowCustomer() และซับรูทีน ShowProduct() เห็นได้ว่ามี Signature เหมือนกับดีลีเกต dlgShow

ต่อมาสร้างตัวแปรที่ชื่อว่า testDlg มี Type เป็นดีลีเกต dlgShow สามารถโหลดซับรูทีน ShowCustomer() และซับรูทีน ShowProduct() ได้ทั้งหมด

สรุปท้ายบท

ความคาดหวังอย่างหนึ่งของการนำเสนอเรื่องดีลีเกตของผู้เขียนก็คือ อยากให้คุณผู้อ่านไปศึกษาเพิ่มเติมต่อจากที่ผู้เขียนนำเสนอในบทนี้ เพราะว่ายังมีรายละเอียดอื่นๆ อีกมากมายที่ยังไม่ได้กล่าวถึง


Programming in OOP style
Advanced .net



Advance Visual Studio 2005

CHAPTER 19

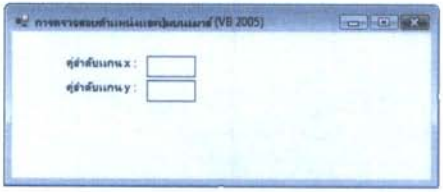
Windows Form

บทนำ

ถ้าจะกล่าวถึงคลาสที่คุ้นเคยและใกล้ชิดเรามากที่สุด คงหนีไม่พ้นฟอร์มนั่นเอง โดยเฉพาะอย่างยิ่งคุณผู้อ่านที่พัฒนาโปรเจ็กต์แบบ Windows Application เพียงอย่างเดียว ซึ่งเป็นเนื้อหาของบทนี้นั่นเอง

การตรวจสอบตำแหน่งและปุ่มบนเมาส์

ในส่วนของการใช้งานเมาส์เคอร์เซอร์ที่น่าสนใจก็คือ การตรวจสอบตำแหน่งของเมาส์เคอร์เซอร์ และการตรวจสอบการคลิกปุ่มของเมาส์นั่นเอง ให้ดูตัวอย่างที่ 19-1 การตรวจสอบตำแหน่งและปุ่มบนเมาส์ ให้คุณออกแบบฟอร์ม ดังรูปที่ 19-1



รูปที่ 19-1 ฟอร์มในขณะออกแบบ

โค้ด VB 2005 ที่ 19-1 การตรวจสอบตำแหน่งและปุ่มบนเมาส์ (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_MouseMove(ByVal sender As System.Object, ByVal e As System.Windows.Forms.MouseEventArgs)
Handles MyBase.MouseMove
        lblXAxis.Text = e.X.ToString()
        lblYAxis.Text = e.Y.ToString()
    End Sub

    Private Sub Form1_MouseUp(ByVal sender As System.Object, ByVal e As System.Windows.Forms.MouseEventArgs)
Handles MyBase.MouseUp
        If e.Button = MouseButtons.Left Then
            MessageBox.Show("คุณคลิกปุ่มซ้าย !!!", "ผลการตรวจสอบ")
        ElseIf e.Button = Windows.Forms.MouseButtons.Right Then
            MessageBox.Show("คุณคลิกปุ่มขวา !!!", "ผลการตรวจสอบ")
        ElseIf e.Button = Windows.Forms.MouseButtons.Middle Then
            MessageBox.Show("คุณคลิกปุ่มกลาง !!!", "ผลการตรวจสอบ")
        End If
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 19-1 การตรวจสอบตำแหน่งและปุ่มบนเมาส์ (Form1.cs)

```
    private void Form1_MouseMove(object sender, MouseEventArgs e)
    {
        lblXAxis.Text = e.X.ToString();
        lblYAxis.Text = e.Y.ToString();
    }

    private void Form1_MouseUp(object sender, MouseEventArgs e)
    {
        if (e.Button == MouseButtons.Left)
        {
            MessageBox.Show("คุณคลิกปุ่มซ้าย !!!", "ผลการตรวจสอบ");
        }
        else if (e.Button == MouseButtons.Right)
        {
            MessageBox.Show("คุณคลิกปุ่มขวา !!!", "ผลการตรวจสอบ");
        }
        else if (e.Button == MouseButtons.Middle)
        {
            MessageBox.Show("คุณคลิกปุ่มกลาง !!!", "ผลการตรวจสอบ");
        }
    }
```

รูปที่ 19-2
กรณีคลิกเมาส์
ปุ่มซ้ายบนฟอร์ม

การคลิกเมาส์ปุ่มกลาง หมายถึง การกดตัวเลื่อน (Wheel Scrolling) ลง ดังรูปที่ 19-3

รูปที่ 19-3
กรณีคลิกเมาส์
ปุ่มกลางบนฟอร์ม

ส่วนการตรวจสอบตำแหน่งปัจจุบันของเมาส์อยู่ในรูปแบบพิกัดคู่ลำดับ (x,y) โดยการอ่านค่าจากคุณสมบัติ X และคุณสมบัติ Y ของพารามิเตอร์ e ของเหตุการณ์ Form1_MouseMove()

การใช้งานคุณสมบัติ KeyPreview ของฟอร์ม

คุณสมบัติที่น่าสนใจอย่างหนึ่งของฟอร์มก็คือ คุณสมบัติ KeyPreview ซึ่งทำหน้าที่ดักจับการกดปุ่มที่คีย์บอร์ดของผู้ใช้ ในขณะที่ฟอร์มอยู่ในสถานะได้รับโฟกัส (Focus) หรือได้รับความสนใจ (Activate)

ลักษณะการนำไปใช้งานก็คือ คุณอาจจะเพิ่มความสะดวกให้ผู้ใช้แอพพลิเคชันของคุณ โดยการกำหนดปุ่มลัด เพื่อให้ทำงานอะไรบางอย่าง โดยที่ไม่ต้องใช้การคลิกเมนูหรือคลิกปุ่ม เช่น เมื่อกดปุ่ม <F1> แล้วให้แสดงหน้าจอช่วยเหลือ เป็นต้น ให้ดูตัวอย่างที่ 19-2 การใช้งานคุณสมบัติ KeyPreview ของฟอร์ม

โค้ด VB 2005 ที่ 19-2 การใช้งานคุณสมบัติ KeyPreview ของฟอร์ม (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub Form1_KeyPress(ByVal sender As Object, ByVal e As System.Windows.Forms.KeyPressEventArgs) Handles
Me.KeyPress
        MessageBox.Show("คุณกดปุ่ม : " & e.KeyChar, "ผลการตรวจสอบ")
    End Sub

    Private Sub Form1_KeyDown(ByVal sender As System.Object, ByVal e As System.Windows.Forms.KeyEventArgs) Handles
MyBase.KeyDown
```

```
        MessageBox.Show("มีรหัสเท่ากับ : " & e.KeyValue, "ผลการตรวจสอบ")
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 19-2 การใช้งานคุณสมบัติ KeyPreview ของฟอร์ม (Form1.cs)

```
private void Form1_KeyPress(object sender, KeyPressEventArgs e)
{
    MessageBox.Show("คุณกดปุ่ม : " + e.KeyChar, "ผลการตรวจสอบ");
}

private void Form1_KeyDown(object sender, KeyEventArgs e)
{
    MessageBox.Show("มีรหัสเท่ากับ : " + e.KeyValue, "ผลการตรวจสอบ");
}
```

ให้คุณกำหนดคุณสมบัติ KeyPreview = True ของ Form1 ผลการทำงาน ดังรูปที่ 19-4

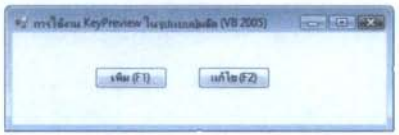
รูปที่ 19-4 กรณีกดปุ่ม <a> บนคีย์บอร์ด

ตัวอักษรของปุ่มอ่านได้จากคุณสมบัติ KeyChar ของพารามิเตอร์ e ของเหตุการณ์ Form1_KeyPress() ส่วนรหัสตัวอักษรอ่านได้จากคุณสมบัติ KeyValue ของพารามิเตอร์ e ของเหตุการณ์ Form1_KeyDown()

การใช้ KeyPreview ในรูปแบบปุ่มลัด

ส่วนการใช้คุณสมบัติ KeyPreview ในรูปแบบปุ่มลัด ให้ดูตัวอย่างที่ 19-3 การใช้ KeyPreview ในรูปแบบปุ่มลัด ให้คุณออกแบบฟอร์ม ดังรูปที่ 19-5

รูปที่ 19-5 ฟอร์มในขณะออกแบบ



โค้ด VB 2005 และ VC# 2005 ที่ 19-3 การใช้ KeyPreview ในรูปแบบปุ่มลัด

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub cmdAdd_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdAdd.Click
        AddData()
    End Sub

    Private Sub cmdEdit_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdEdit.Click
        EditData()
    End Sub

    Private Sub Form1_KeyDown(ByVal sender As System.
Object, ByVal e As System.Windows.Forms.KeyEventArgs)
Handles MyBase.KeyDown
        If e.KeyValue = 112 Then
            AddData()
        End If
        If e.KeyValue = 113 Then
            EditData()
        End If
    End Sub

    Private Sub AddData()
        MessageBox.Show("เพิ่มข้อมูล")
    End Sub

    Private Sub EditData()
        MessageBox.Show("แก้ไขข้อมูล")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void cmdAdd_Click(object sender, EventArgs e)
    {
        AddData();
    }

    private void cmdEdit_Click(object sender, EventArgs e)
    {
        EditData();
    }

    private void Form1_KeyDown(object sender, Key
EventArgs e)
    {
        if (e.KeyValue == 112)
        {
            AddData();
        }
        if (e.KeyValue == 113)
        {
            EditData();
        }
    }

    private void AddData()
    {
        MessageBox.Show("เพิ่มข้อมูล");
    }

    private void EditData()
    {
        MessageBox.Show("แก้ไขข้อมูล");
    }
```

รูปที่ 19-6
กรณีกดปุ่ม <F1> เพื่อเพิ่มข้อมูล

จากรูปที่ 19-6 ผู้เขียนจำลองการทำงานที่ว่า ถ้าเป็นการเพิ่มข้อมูลอยู่ในความรับผิดชอบของซับรูทีน AddData() แต่ถ้าเป็นการแก้ไขข้อมูลก็จะเรียกซับรูทีน EditData() ทำงาน

คุณสามารถสั่งให้ซับรูทีนทั้ง 2 ทำงาน โดยการกดปุ่ม <F1> (มีรหัส 112) หรือปุ่ม <F2> (มีรหัส 113) ตามลำดับบนคีย์บอร์ด หรือคลิกปุ่ม เพิ่ม (F1) หรือปุ่ม แก้ไข (F2) ก็ได้เช่นกัน

การส่งข้อมูลระหว่างฟอร์ม

ในการพัฒนา Windows Application ย่อมที่จะมีการใช้งานฟอร์มหลายๆ ฟอร์ม เพื่อทำหน้าที่เป็นส่วนแสดงผลในโปรเจ็กต์ของคุณ การส่งข้อมูลระหว่างฟอร์มเป็นเทคนิคอีกอย่างหนึ่งที่ช่วยให้คุณไม่ต้องใช้ตัวแปรแบบ Public ที่มีขอบเขตการเรียกใช้งานทั้งโปรเจ็กต์มากจนเกินไป เพราะว่าตัวแปรชนิดนี้ควบคุมค่าที่ตัวมันเองเก็บอยู่ค่อนข้างยากนั่นเอง

การส่งข้อมูลระหว่างฟอร์มที่นำเสนอ จึงใช้ฐานความคิดที่ว่าเป็นการส่งข้อมูลจากฟอร์มหนึ่งไปสู่อีกฟอร์มหนึ่งโดยตรง เท่าที่ผู้เขียนทำได้มีอยู่ 5 วิธีคือ

1. โดยการสร้าง Properties
2. โดยอาศัย Delegate
3. โดยอาศัยตัวแปร Form Type
4. โดยอาศัย Constructor
5. โดยอาศัย Constructor ร่วมกับการส่งค่าพารามิเตอร์แบบ Reference

ส่งข้อมูลระหว่างฟอร์มโดยการสร้าง Properties

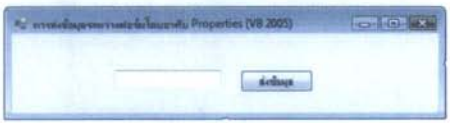
ผู้เขียนขอนำ Form ที่เราใช้งานบ่อยครั้งและคุ้นเคยกันมากที่สุดเข้ามาสู่โลกของ OOP คุณเพิ่มฟอร์มใหม่เข้ามาในโปรเจ็กต์ของคุณ สิ่งที่ได้มาก็คือ คลาสลูกที่ชื่อว่า Form1, Form2,... Form n สืบทอดมาจากคลาสต้นแบบที่ชื่อว่า Form ของ .NET Framework คุณสามารถเปลี่ยนชื่อฟอร์มได้จากคุณสมบัติ Name กล่าวได้อีกนัยหนึ่งก็คือ คุณกำลังเปลี่ยนชื่อคลาสลูกจาก Form1 ไปเป็นชื่อคลาสใหม่ตามที่ต้องการ

คุณสร้างซับรูทีนหรือฟังก์ชันต่างๆ ขึ้นมาใช้งานในฟอร์มของคุณ หากมองในแง่ของ OOP แล้ว คุณกำลังเพิ่มเมธอดใหม่ให้กับคลาสลูก Form1 ที่คุณใช้งานอยู่นั่นเอง

เราคุ้นเคยกับการใช้งาน member function เพียงชนิดเดียวในฟอร์มนั่นคือ สร้างเมธอดใหม่ให้กับคลาสลูก Form1 เราไม่คุ้นเคยที่จะสร้างคุณสมบัติ (Properties) ใหม่ให้กับฟอร์มเท่าใดนัก

ในเมื่อเรามองว่าคลาสลูก Form1 เป็นเพียงคลาสๆ หนึ่ง ที่สามารถเพิ่มคุณสมบัติใหม่ขึ้นมาได้ จึงไม่แปลกอะไรที่เราจะอ่านค่า หรือกำหนดค่าของข้อมูลออกมาจากคลาสฟอร์มลูกที่เราสนใจ ให้ดูตัวอย่างที่ 19-4 การส่งข้อมูลระหว่างฟอร์มโดยการสร้าง Properties

รูปที่ 19-7
Form1 ในขณะออกแบบ



รูปที่ 19-8
Form2 ในขณะออกแบบ

เราต้องการส่งค่าที่ถูกป้อนในคอนโทรล TextBox ที่ชื่อว่า txtSource ที่อยู่ใน Form1 ไปแสดงในคอนโทรล Label ที่ชื่อว่า lblDestination ที่อยู่ใน Form2 มองอีกนัยหนึ่งคลาสลูก Form2 ไม่มีคุณสมบัติที่ทำหน้าที่กำหนดค่าให้กับคอนโทรล lblDestination นั่นเอง ที่ Form2 ให้เขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 19-4 การส่งข้อมูลระหว่างฟอร์มโดยการสร้าง Properties

VB 2005 (Form2.vb)

```
Option Explicit On
Option Strict On

Public Class Form2
    Friend WriteOnly Property SetlblDestination() As String
        Set(ByVal value As String)
            lblDestination.Text = value
        End Set
    End Property
End Class
```

VC# 2005 (Form2.cs)

```
    internal string SetlblDestination
    {
        set{lblDestination.Text = value;}
    }
```

โค้ดข้างต้นสร้างคุณสมบัติใหม่ขึ้นมาชื่อว่า SetlblDestination ให้กับคลาสลูก Form2 ทำหน้าที่กำหนดค่าให้กับคอนโทรล lblDestination มีขอบเขตแบบ Friend (internal) หมายถึง ถ้าอยู่ใน Assembly เดียวกันสามารถเรียกใช้งานได้ ในกรณีนี้คือ อยู่ในโปรเจ็กต์เดียวกันนั่นเอง ส่วนที่ Form1 ซึ่งทำหน้าที่ส่งข้อมูล เขียนโค้ดดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 19-4 การส่งข้อมูลระหว่างฟอร์มโดยการสร้าง Properties

VB 2005 (Form1.vb)

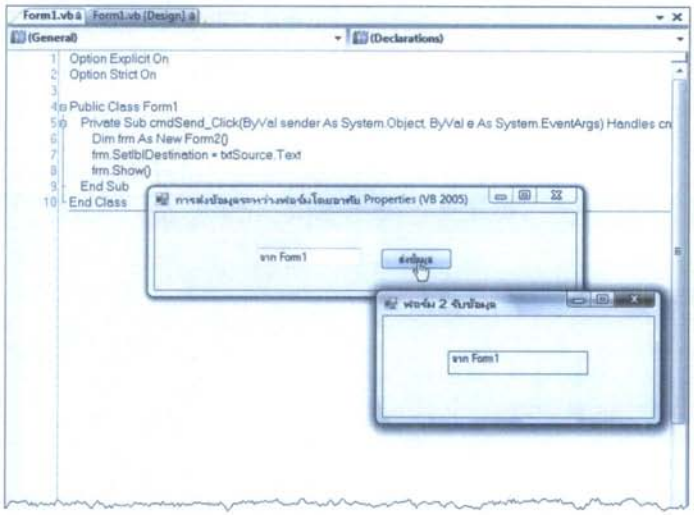
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub cmdSend_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdSend.Click
        Dim frm As New Form2()
        frm.SetlblDestination = txtSource.Text
        frm.Show()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void cmdSend_Click(object sender, EventArgs e)
{
    Form2 frm = new Form2();
    frm.SetlblDestination = txtSource.Text;
    frm.Show();
}
```

จากโค้ดข้างต้นเราสร้างออบเจ็กต์ Form2 ที่ชื่อว่า frm เห็นได้ว่ามีคุณสมบัติ SetlblDestination ตามที่เราสร้างไว้ในคลาสลูก Form2 ก็จะอ่านค่าจากคอนโทรล txtSource ที่อยู่ในคลาสลูก Form1 นั่นเอง ผลการทำงาน ดังรูปที่ 19-9

รูปที่ 19-9
ผลการรัน
ตัวอย่างที่ 19-4



ผู้เขียนขอกล่าวในเชิงของ OOP ที่น่าสนใจอีกประเด็นหนึ่งคือ การใช้งานคอนโทรล TextBox1 ใน Form1 เป็นการใช้งานคลาส TextBox1 ภายในคลาส Form1 จึงไม่แปลกอะไรที่คุณจะกำหนดให้คลาส Form1 มีคุณสมบัติใหม่เพิ่มขึ้นมา ทำหน้าที่กำหนดค่าหรืออ่านค่าจากคลาส TextBox1 ที่อยู่ภายใน

การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Delegate

หลักการทำงานคือ การสั่งเมธอดทำงานข้ามฟอร์ม โดยอาศัย Delegate (delegate) นั่นเอง ให้ดูตัวอย่างที่ 19-5 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Delegate (delegate) ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 19-5 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Delegate (delegate)

VB 2005 (Form2.vb)

```
Option Explicit On
Option Strict On

Public Class Form2
    Public Sub ReceivedData(ByVal txt As TextBox)
        lblDestination.Text = txt.Text
    End Sub
End Class
```

VC# 2005 (Form2.cs)

```
public void ReceivedData(TextBox txt)
{
    lblDestination.Text = txt.Text;
}
```

ที่คลาสลูก Form2 สร้างเมธอด (ซับรูทีน) ที่ชื่อว่า ReceivedData() ขึ้นมา ทำหน้าที่กำหนดค่าให้กับคอนโทรล lblDestination เห็นได้ว่าพารามิเตอร์มี Type เป็นคอนโทรล TextBox เพื่อให้สามารถรับค่ามาจากคอนโทรล TextBox ที่อยู่ในคลาสลูก Form1 นั่นเอง

ถ้าจะมองโค้ดชุดนี้อีกนัยหนึ่งก็คือ เราสร้าง member function ชนิดเมธอด (Method) ขึ้นมาใหม่ในคลาสลูก Form2 นั่นเอง แตกต่างจากวิธีที่แล้วซึ่งใช้ member function ชนิดคุณสมบัติ (Properties) ส่วนที่คลาสลูก Form1 เขียนโค้ดดังนี้

โค้ด VB 2005 ที่ 19-5 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Delegate (delegate) (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Public Delegate Sub dlgSendData(ByVal txt As TextBox)

    Private Sub cmdSend_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles cmdSend.Click
        Dim frm As New Form2()
        Dim dlg As dlgSendData
        dlg = AddressOf frm.ReceivedData
        dlg(Me.txtSource)
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 19-5 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Delegate (delegate) (Form1.cs)

```
    public delegate void dlgSendData(TextBox txt);

    private void cmdSend_Click(object sender, EventArgs e)
    {
        Form2 frm = new Form2();
        dlgSendData dlg = new dlgSendData(frm.ReceivedData);
        dlg.Invoke(this.txtSource);
        frm.Show();
    }
```

วิธีการดูโค้ดชุดนี้ก็คือ เมธอด ReceivedData() ที่อยู่ในคลาสลูก Form2 มี Signature ต้องการพารามิเตอร์ 1 ตัว มี Type เป็นคอนโทรล TextBox และไม่มีการส่งค่ากลับ

VB 2005	VC# 2005
Public Sub ReceivedData(ByVal txt As TextBox)	public void ReceivedData(TextBox txt)

ข้ามมาที่คลาสลูก Form1 สร้าง Delegate (delegate) ตาม Signature ของเมธอด ReceivedData() ข้างต้นชื่อว่า dlgSendData

VB 2005

```
Public Delegate Sub dlgSendData(ByVal txt As TextBox)
```

VC# 2005

```
public delegate void dlgSendData(TextBox txt);
```

ต่อมาสั่งให้เมธอด ReceivedData() ที่อยู่ในคลาสลูก Form2 ทำงานผ่านทาง Delegate ที่ชื่อว่า dlg นั่นเอง เห็นได้ว่าเราจะส่งค่าของคอนโทรล txtSource ที่อยู่ในคลาสลูก Form1 เข้าไป ดังรูปที่ 19-10

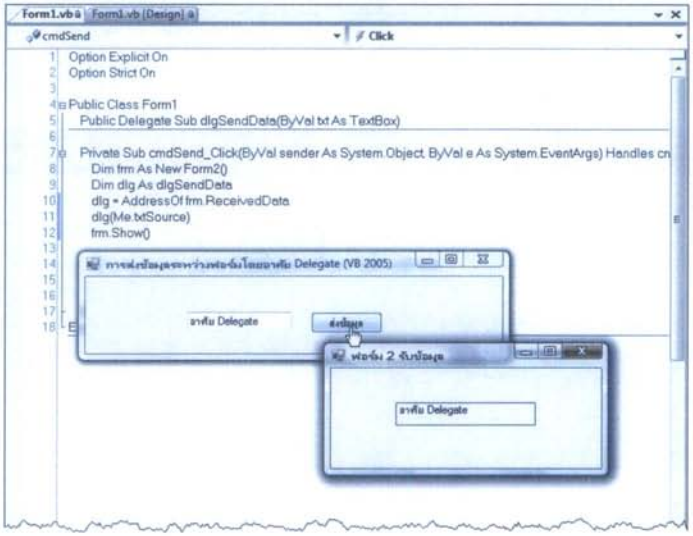
VB 2005

```
Dim frm As New Form2()
Dim dlg As dlgSendData
dlg = AddressOf frm.ReceivedData
dlg(Me.txtSource)
frm.Show()
```

VC# 2005

```
Form2 frm = new Form2();
dlgSendData dlg = new dlgSendData(frm.ReceivedData);
dlg.Invoke(this.txtSource);
frm.Show();
```

รูปที่ 19-10
ผลการรัน
ตัวอย่างที่ 19-5



สำหรับ VB 2005 สามารถเขียนโค้ดให้สั้นลงได้ดังนี้

VB 2005

```
Dim dlg As New dlgSendData(AddressOf frm.ReceivedData)
dlg.Invoke(Me.txtSource)
frm.Show()
```

การส่งข้อมูลระหว่างฟอร์มโดยอาศัยตัวแปร Form Type

หลักการของวิธีนี้ก็คือ การมองฟอร์มเป็น Type ชนิดหนึ่งนั่นเอง ให้ดูตัวอย่างที่ 19-6 การส่งข้อมูลระหว่างฟอร์มโดยอาศัยตัวแปร Form Type

โค้ด VB 2005 ที่ 19-6 การส่งข้อมูลระหว่างฟอร์มโดยอาศัยตัวแปร Form Type (Form2.vb)

```
Option Explicit On
Option Strict On

Public Class Form2
    Public frm1 As Form1

    Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        lblDestination.Text = frm1.txtSource.Text
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 19-6 การส่งข้อมูลระหว่างฟอร์มโดยอาศัยตัวแปร Form Type (Form2.cs)

```
    public Form1 frm1;

    private void Form2_Load(object sender, EventArgs e)
    {
        lblDestination.Text = frm1.txtSource.Text;
    }
```

ที่คลาสลูก Form2 สร้างตัวแปร 1 ตัวชื่อว่า frm1 มี Type เป็น Form1 มีขอบเขตแบบ Public เพื่อให้คลาสลูก Form1 สามารถเรียกใช้งานตัวแปร frm1 ตัวนี้ได้ เมื่อตัวแปร frm1 มี Type เป็น Form1 ส่งผลให้คุณสามารถเรียกใช้งานคอนโทรล txtSource ได้นั่นเอง

ส่วนที่คลาสลูก Form1 ให้คุณเขียนโค้ดดังนี้

โค้ด VB 2005 และ VC# 2005 ที่ 19-6 การส่งข้อมูลระหว่างฟอร์มโดยอาศัยตัวแปร Form Type

VB 2005 (Form1.vb)

```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub cmdSend_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdSend.Click
        Dim frm As New Form2()
        frm.frm1 = Me
        frm.Show()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void cmdSend_Click(object sender,
EventArgs e)
    {
        Form2 frm = new Form2();
        frm.frm1 = this;
        frm.Show();
    }
```

จุดสำคัญของวิธีนี้อยู่ตรงที่คุณต้องกำหนดให้ตัวแปร frm1 ที่อยู่ในคลาสลูก Form2 คือ Form1 Type เห็นได้ว่าเราใช้คำสั่ง Me (this) ในคลาส Form1 ส่งผลให้คำสั่ง Me (this) จึงหมายถึงคลาส Form1 นั่นเอง

VB 2005	VC# 2005
frm.frm1 = Me	frm.frm1 = this;

การส่งข้อมูลระหว่างฟอร์มด้วย Constructor

หลักการของวิธีนี้ก็คือ โดยปกติแล้วคอนสตรัคเตอร์ของคลาสฟอร์ม ไม่ต้องการพารามิเตอร์ใดๆ เมื่อฟอร์มก็เป็นคลาสๆ หนึ่ง จึงไม่แปลกอะไรที่จะมีคอนสตรัคเตอร์มากกว่า 1 ชุด ให้ดูตัวอย่างที่ 19-7 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor

โค้ด VB 2005 และ VC# 2005 ที่ 19-7 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor

VB 2005 (Form2.vb)

```
Option Explicit On
Option Strict On

Public Class Form2
    Public Sub New(ByVal str As String)
        InitializeComponent()

        lblDestination.Text = str
    End Sub
End Class
```

VC# 2005 (Form2.cs)

```
public partial class Form2 : Form
{
    public Form2(string str)
    {
        InitializeComponent();

        lblDestination.Text = str;
    }
}
```

ที่คลาสลูก Form2 ผู้เขียนสร้างคอนสตรัคเตอร์แบบมีพารามิเตอร์ขึ้นมาใหม่ ต้องการพารามิเตอร์ 1 ตัวชื่อว่า str มี Type เป็น String (string) ก็จะนำค่าของพารามิเตอร์ str ไปกำหนดค่าให้กับคอนโทรล lblDestination นั่นเอง แต่เนื่องจากว่าฟอร์มทำหน้าที่เป็นตัวบรรจุคอนโทรลต่างๆ ที่คุณใช้งานอยู่บนฟอร์ม จึงต้องมีการสั่งให้ซับรูทีน InitializeComponent() ทำงานด้วย

โค้ด VB 2005 และ VC# 2005 ที่ 19-7 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor

VB 2005 (Form1.vb)

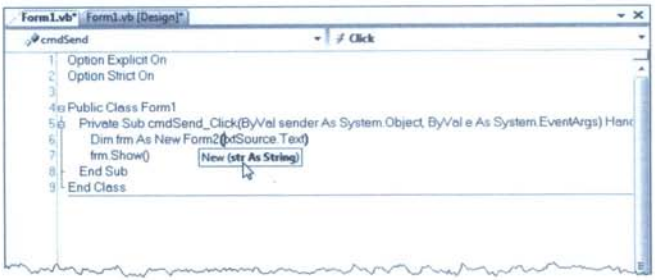
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub cmdSend_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdSend.Click
        Dim frm As New Form2(txtSource.Text)
        frm.Show()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
private void cmdSend_Click(object sender, EventArgs e)
{
    Form2 frm = new Form2(txtSource.Text);
    frm.Show();
}
```

จากโค้ดข้างต้นพบว่าในขณะที่สั่งให้ออบเจ็กต์ Form2 ที่ชื่อว่า frm เกิดขึ้นมา จะมีคอนสตรัคเตอร์เพิ่มขึ้นมาอีก 1 ชุด ตามที่เรากำหนดไว้ให้กับคลาสลูก Form2 ดังรูปที่ 19-11

รูปที่ 19-11 คอนสตรัคเตอร์ตัวที่ 2 ของคลาสลูก Form2



ให้คุณส่งค่าที่ผู้ใช้ป้อนในคอนโทรล txtSource ที่อยู่ใน Form1 ให้กับพารามิเตอร์ str นั่นเอง

การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor ร่วมกับการส่งค่าพารามิเตอร์แบบ Reference

หลักการของวิธีนี้เหมือนกับวิธีที่ผ่านมา แต่สิ่งที่เพิ่มเติมขึ้นมาก็คือ พารามิเตอร์ที่อยู่ในคอนสตรัคเตอร์จะส่งค่าแบบ Reference ส่งผลให้เมื่อค่าของพารามิเตอร์มีการเปลี่ยนแปลง จะทำให้ค่าของต้นทาง (Form1) และปลายทาง (Form2) เปลี่ยนแปลงพร้อมๆ กัน ให้ดูตัวอย่างที่ 19-8 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor ร่วมกับการส่งค่าพารามิเตอร์แบบ Reference ให้คุณออกแบบฟอร์ม ดังรูปที่ 19-12 และ 19-13

รูปที่ 19-12 Form1 ในขณะออกแบบ

รูปที่ 19-13 Form2 ในขณะออกแบบ

โค้ด VB 2005 ที่ 19-8 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor ร่วมกับการส่งค่าพารามิเตอร์แบบ Reference (Form2.vb)

```
Option Explicit On
Option Strict On

Public Class Form2
    Private _Textbox As New TextBox

    Public Sub New(ByRef refTextBox As TextBox)
        InitializeComponent()

        _Textbox = refTextBox
    End Sub

    Private Sub txtSend_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
txtSend.TextChanged
        _Textbox.Text = txtSend.Text
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 19-8 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor ร่วมกับการส่งค่าพารามิเตอร์แบบ Reference (Form2.cs)

```
        private TextBox _Textbox = new TextBox();

        public Form2(ref TextBox refTextBox)
        {
            InitializeComponent();

            _Textbox = refTextBox;
        }

        private void txtSend_TextChanged(object sender, EventArgs e)
        {
            _Textbox.Text = txtSend.Text;
        }
```

วิธีคิดก็คือ เราจะสร้างคอนโทรล TextBox ขึ้นมา 1 ตัวชื่อว่า _Textbox มีขอบเขตเป็นแบบ Private ทำหน้าที่เป็นตัวกลาง ให้คุณมองว่าคอนโทรล _Textbox ตัวนี้คือ คอนโทรล TextBox ที่อยู่ใน Form1

VB 2005

```
Private _Textbox As New TextBox

Public Sub New(ByRef refTextBox As TextBox)
    InitializeComponent()

    _Textbox = refTextBox
End Sub
```

VC# 2005

```
private TextBox _Textbox = new TextBox();

public Form2(ref TextBox refTextBox)
{
    InitializeComponent();

    _Textbox = refTextBox;
}
```

ส่วนในคอนสตรัคเตอร์กำหนดให้มีพารามิเตอร์ 1 ตัวชื่อว่า refTextBox มี Type เป็นคอนโทรล TextBox กำหนดให้ส่งค่าแบบ Reference ก็จะส่งค่าต่อมาให้กับคอนโทรล _Textbox

การส่งข้อมูลด้วยวิธีนี้แตกต่างจากวิธีที่ผ่านมาตรงที่ เราจะใช้คอนโทรล TextBox ที่อยู่ใน Form2 ทำหน้าที่ส่งข้อมูล และใช้คอนโทรล TextBox ที่อยู่ใน Form1 เป็นตัวรับข้อมูล

ในเหตุการณ์ txtSend_TextChanged() ก็จะกำหนดให้ส่งค่าที่ถูกป้อนในคอนโทรล txtSend ไปยังคอนโทรล _Textbox กล่าวได้อีกนัยหนึ่งก็คือ ส่งต่อไปยังคอนโทรล TextBox ที่อยู่ใน Form1 นั่นเอง

VB 2005

```
Private Sub txtSend_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
txtSend.TextChanged
    _Textbox.Text = txtSend.Text
End Sub
```

VC# 2005

```
private void txtSend_TextChanged(object sender, EventArgs e)
{
    _Textbox.Text = txtSend.Text;
}
```

ส่วนที่ Form1 ให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 19-8 การส่งข้อมูลระหว่างฟอร์มโดยอาศัย Constructor ร่วมกับการส่งค่าพารามิเตอร์แบบ Reference

VB 2005 (Form1.vb)

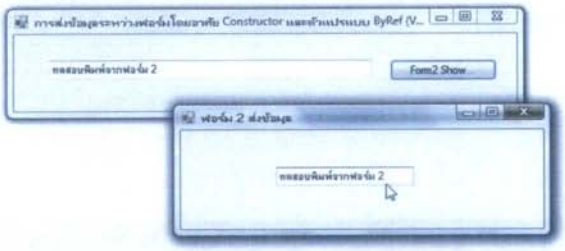
```
Option Explicit On
Option Strict On

Public Class Form1
    Private Sub cmdForm2Show_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
cmdForm2Show.Click
        Dim frm As New Form2(txtReceived)
        frm.Show()
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
    private void cmdForm2Show_Click(object sender,
EventArgs e)
    {
        Form2 frm = new Form2(ref txtReceived);
        frm.Show();
    }
```

จะเห็นได้ว่าเมื่อสั่งให้ออบเจ็กต์ Form2 ที่ชื่อว่า frm เกิดขึ้นมา ต้องส่งคอนโทรล txtReceived ที่อยู่ใน Form1 เข้าไปด้วย ส่งผลให้คอนโทรล txtReceived กับคอนโทรล _Textbox ที่อยู่ใน Form2 เป็นคอนโทรลตัวเดียวกันผ่านทางพารามิเตอร์ refTextBox ซึ่งกำหนดให้ส่งค่าแบบ Reference นั่นเอง ดังรูปที่ 19-14

รูปที่ 19-14 ผลการทำงานของตัวอย่างที่ 19-8



จากรูปที่ 19-14 เมื่อคุณพิมพ์ข้อความในคอนโทรล txtSend ที่อยู่ใน Form2 พบว่าคอนโทรล txtReceived ที่อยู่ใน Form1 มีข้อความที่คุณพิมพ์ปรากฏขึ้นมาด้วยแบบ Real Time

การทำ Multiple Handle Event

ปัญหาอีกอย่างหนึ่งของการใช้ฟอร์มที่ผู้เขียนเคยพบก็คือ ในบางกรณีเราต้องการทำงานอะไรบางอย่าง ให้มีผลกับทุกฟอร์มที่มีอยู่ในโปรเจ็กต์ของเรา วิธีที่จะนำเสนอต่อไปนี้เป็นอีก 1 วิธีเรียกว่า การทำ Multiple Handle Event

ผู้เขียนขอยกตัวอย่างเรื่องของการลงสีพื้นหลังของฟอร์มแบบ Gradient (การลงสีพื้นหลังแบบไล่โทนสี) เห็นได้ว่าคุณต้องเขียนโค้ดต่อไปนี้ ในเหตุการณ์ Form_Paint() ของทุกๆ ฟอร์มที่อยู่ในโปรเจ็กต์ของคุณ

VB 2005

```
Option Explicit On
Option Strict On
Imports System.Drawing.Drawing2D

Public Class Form1
    Private Sub Form1_Paint(ByVal sender As System.Object, ByVal e As System.Windows.Forms.PaintEventArgs) Handles
MyBase.Paint
        Dim lgb As LinearGradientBrush = New LinearGradientBrush(ClientRectangle, Color.Yellow, Color.YellowGreen,
LinearGradientMode.Vertical)
        e.Graphics.FillRectangle(lgb, ClientRectangle)
        lgb.Dispose()
    End Sub
End Class
```

VC# 2005

```
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            LinearGradientBrush lgb = new LinearGradientBrush(ClientRectangle, Color.Yellow, Color.YellowGreen, LinearGradientMode.
Vertical);
            e.Graphics.FillRectangle(lgb, ClientRectangle);
            lgb.Dispose();
        }
```

โค้ดข้างต้นกำหนดให้ลงสีพื้นหลังแบบ Gradient ไล่โทนสีจากเหลืองไปโทนสีเขียวในแนวตั้ง (Vertical) ปัญหาที่เกิดขึ้นก็คือ คุณต้องเขียนโค้ดชุดนี้ซ้ำกันทุกฟอร์มเท่าที่มีอยู่ในโปรเจ็กต์ของคุณ ถ้าต้องการเปลี่ยนโทนสี คุณต้องแก้ไขโค้ดชุดนี้ที่อยู่ในแต่ละฟอร์ม ซึ่งเป็นการทำงานซ้ำซ้อนโดยไม่จำเป็น

เราจะใช้ความสามารถในการทำ Multiple Handle Event เข้ามาช่วยแก้ไขปัญหานี้ ให้ดูตัวอย่างที่ 19-9 การทำ Multiple Handle Event ให้คุณเพิ่มฟอร์มเข้ามาในโปรเจ็กต์หลายๆ ฟอร์ม ในกรณีนี้มี 3 ฟอร์ม ดังรูปที่ 19-15

รูปที่ 19-15
ฟอร์ม 3 ฟอร์มใน Solution Explorer

โค้ด VB 2005 ที่ 19-9 การทำ Multiple Handle Event (Form1.vb)

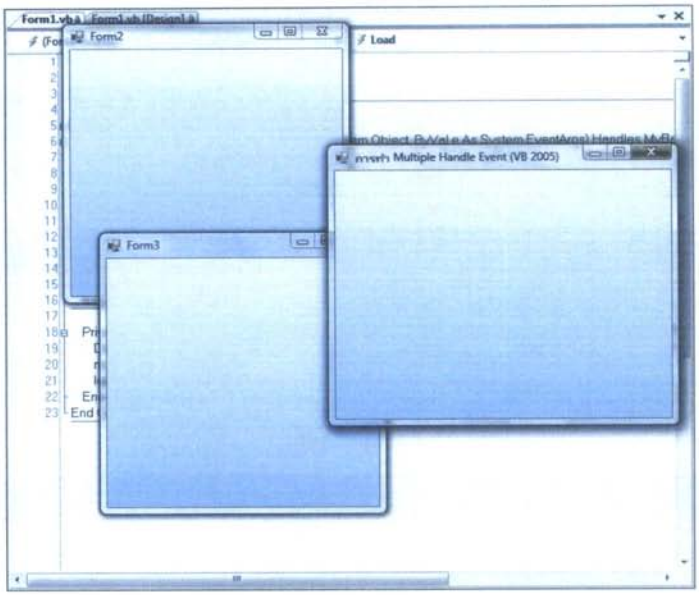
```
Option Explicit On
Option Strict On
Imports System.Drawing.Drawing2D

Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        AddHandler MyBase.Paint, New PaintEventHandler(AddressOf Me.myPainter)

        Dim f2 As New Form2()
        AddHandler f2.Paint, New PaintEventHandler(AddressOf Me.myPainter)
        f2.Show()

        Dim f3 As New Form3()
        AddHandler f3.Paint, New PaintEventHandler(AddressOf Me.myPainter)
        f3.Show()
    End Sub

    Private Sub myPainter(ByVal mySender As System.Object, ByVal myE As System.Windows.Forms.PaintEventArgs)
        Dim lgb As LinearGradientBrush = New LinearGradientBrush(ClientRectangle, Color.Yellow, Color.YellowGreen,
LinearGradientMode.Vertical)
        myE.Graphics.FillRectangle(lgb, ClientRectangle)
        lgb.Dispose()
    End Sub
End Class
```

โค้ด VC# 2005 ที่ 19-9 การทำ Multiple Handle Event (Form1.cs)

```
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Paint += new PaintEventHandler(this.myPainter);

            Form2 f2 = new Form2();
            f2.Paint += new PaintEventHandler(myPainter);
            f2.Show();

            Form3 f3 = new Form3();
            f3.Paint += new PaintEventHandler(myPainter);
            f3.Show();
        }

        public void myPainter(object mySender, PaintEventArgs myE)
        {
            LinearGradientBrush lgb = new LinearGradientBrush(ClientRectangle, Color.Yellow, Color.YellowGreen,
LinearGradientMode.Vertical);
            myE.Graphics.FillRectangle(lgb,ClientRectangle);
            lgb.Dispose();
        }
```

จะเห็นได้ว่าสีพื้นหลังทั้ง 3 ฟอร์มถูกลงสีเป็นแบบ Gradient โดยการเขียนโค้ดเพียง 1 ชุด ถ้าคุณต้องการแก้ไขสีใหม่ก็ทำเพียง 1 ครั้งเท่านั้น ดังรูปที่ 19-16

รูปที่ 19-16 ผลการรันตัวอย่างที่ 19-9



วิธีการดูโค้ดชุดนี้ก็คือ เราจะสร้างเหตุการณ์กลางขึ้นมา 1 ชุด กำหนดให้มี Signature เหมือนกับเหตุการณ์ Form_Paint() ทำหน้าที่เป็นช่างทาสีชื่อว่า myPainter เก็บไว้ในฟอร์มไหนก็ได้ ในกรณีนี้เก็บไว้ที่ Form1

VB 2005

```
    Private Sub myPainter(ByVal mySender As System.Object, ByVal myE As System.Windows.Forms.PaintEventArgs)
        Dim lgb As LinearGradientBrush = New LinearGradientBrush(ClientRectangle, Color.Yellow, Color.YellowGreen,
LinearGradientMode.Vertical)
        myE.Graphics.FillRectangle(lgb, ClientRectangle)
        lgb.Dispose()
    End Sub
```

VC# 2005

```
        public void myPainter(object mySender, PaintEventArgs myE)
        {
            LinearGradientBrush lgb = new LinearGradientBrush(ClientRectangle, Color.Yellow, Color.YellowGreen, LinearGradientMode.
Vertical);
            myE.Graphics.FillRectangle(lgb,ClientRectangle);
            lgb.Dispose();
        }
```

ส่วนวิธีการสั่งให้ช่างทาสีคนนี้ทำงาน แล้วแต่ว่าจะทำอย่างไร ในกรณีนี้อาศัยเหตุการณ์ Form1_Load() เป็นผู้สั่งให้ทำงานนั่นเอง แยกออกเป็น 2 กรณี

- **กรณีที่** 1 เนื่องจากในคลาส Form1 เราไม่ได้สร้างเหตุการณ์ Form1_Paint() ขึ้นมา จึงต้องเพิ่มเหตุการณ์ Paint ให้กับคลาส Form1 โดยการเขียนโค้ดเอง สั่งให้ช่างทาสีที่ชื่อว่า myPainter ทำงาน

VB 2005

```
AddHandler MyBase.Paint, New PaintEventHandler(AddressOf Me.myPainter)
```

VC# 2005

```
this.Paint += new PaintEventHandler(this.myPainter);
```

- **กรณีที่** 2 ส่วนฟอร์มอื่นๆ (Form2 และ Form3) ก็จะเพิ่มเหตุการณ์ Paint() ให้กับคลาส Form2 และคลาส Form3 โดยการเขียนโค้ดเช่นกัน

VB 2005

```
Dim f2 As New Form2()
AddHandler f2.Paint, New PaintEventHandler(AddressOf Me.myPainter)
f2.Show()

Dim f3 As New Form3()
AddHandler f3.Paint, New PaintEventHandler(AddressOf Me.myPainter)
f3.Show()
```

VC# 2005

```
Form2 f2 = new Form2();
f2.Paint += new PaintEventHandler(myPainter);
f2.Show();

Form3 f3 = new Form3();
f3.Paint += new PaintEventHandler(myPainter);
f3.Show();
```

การสร้างคลาสที่ทำงานเกี่ยวกับฟอร์ม

สำหรับคลาสที่ทำงานเกี่ยวข้องกับฟอร์มชื่อว่า FormTools อยู่ภายใต้เนมสเปซ

GAF

คลาส FormTools ประกอบด้วย 3 เมธอดคือ

- เมธอด FadeIn() ทำหน้าที่เพิ่มความเข้มของฟอร์ม
- เมธอด FadeOut() ทำหน้าที่ลดความเข้มของฟอร์ม
- เมธอด ShowFormFullScreen() ทำหน้าที่แสดงฟอร์มแบบเต็มหน้าจอแสดงผล (Full Screen)

การทำฟอร์มแบบ Fade-In และ Fade-Out

ฟอร์มแบบ Fade-In หมายถึง การกำหนด Effect ให้ฟอร์มค่อยๆ ปรากฏขึ้นมา โดยเพิ่มความเข้มของฟอร์มมากขึ้นเรื่อยๆ จนฟอร์มปรากฏขึ้นมา ส่วนการทำ Effect ฟอร์มแบบ Fade-Out หมายถึง เมื่อมีการปิดฟอร์มแล้ว ความเข้มของฟอร์มค่อยๆ จางลงจนกระทั่งฟอร์มหายไป การปรับความเข้ม-จางของฟอร์มสามารถกำหนดได้ที่คุณสมบัติ Opacity โดยที่ความเข้มในระดับปกติอยู่ที่ 100%

- ฟอร์มที่มีค่าความเข้ม-จาง (คุณสมบัติ Opacity) เท่ากับ 0 (0%) คือ ฟอร์มที่มองไม่เห็น อยู่ในความรับผิดชอบของเมธอด FadeOut()
- ฟอร์มที่มีค่าความเข้ม-จางเท่ากับ 1 (100%) คือ ฟอร์มที่มีความเข้มตามปกติ อยู่ในความรับผิดชอบของเมธอด FadeIn()

การแสดงฟอร์มแบบเต็มหน้าจอ (Full Screen)

โดยปกติแล้วเราจะคุ้นเคยกับการแสดงฟอร์มเพียง 2 สถานะคือ Minimized และ Maximized แต่ถ้าคุณต้องการกำหนดให้ฟอร์มแสดงแบบเต็มหน้าจอ (Full Screen) คุณไม่สามารถกำหนดได้โดยตรง จึงเป็นที่มาของเมธอด ShowFormFullScreen()

โค้ด VB 2005 ที่ 19-10 การสร้างคลาสที่ทำงานเกี่ยวกับฟอร์ม (GAF.FormTools.vb)

```
Option Explicit On
Option Strict On
Imports System.Windows
Imports System.Windows.Forms

Namespace GAF
    Public Class FormTools
        Private _i As Double = 0.0                      ตัวนับวนลูป
        Private _MinOpacity As Integer = 0              'ความจางของฟอร์มน้อยที่สุด
        Private _MaxOpacity As Integer = 1              'ความเข้มของฟอร์มมากที่สุด (ปกติ)
```

```
    Private _BorderStyle As FormBorderStyle          ฟิลด์เก็บรูปแบบขอบของฟอร์ม
    Private _WindowState As FormWindowState          ฟิลด์เก็บสถานะของฟอร์ม
    Private _IsFullScreen As Boolean = False         ฟิลด์บอกสถานะว่า แสดงเต็มจออยู่หรือไม่

เมธอด FadeIn() ทำหน้าที่เพิ่มความเข้มของฟอร์ม ต้องการพารามิเตอร์ 2 ตัวคือ
พารามิเตอร์ TargetForm หมายถึง ฟอร์มเป้าหมาย
พารามิเตอร์ FadeStep หมายถึง ระดับความเข้ม-จางที่เพิ่มขึ้นเรื่อยๆ
    Public Sub FadeIn(ByVal TargetForm As Form, ByVal FadeStep As Double)
      If TargetForm.Visible = False Then          ถ้าฟอร์มอยู่ในสถานะมองไม่เห็นแล้ว
        TargetForm.Opacity = 0                    ให้ความเข้มเป็น 0
        TargetForm.Visible = True                 ให้มองเห็นฟอร์ม
      End If
ให้วนลูปเพื่อเพิ่มความเข้มของฟอร์ม ตามค่าของพารามิเตอร์ FadeStep
      For _i = 0 To _MaxOpacity Step FadeStep
        TargetForm.Opacity = _i
        TargetForm.Refresh()
      Next
    End Sub

เมธอด FadeOut() ทำหน้าที่เพิ่มความเข้มของฟอร์ม ต้องการพารามิเตอร์ 2 ตัวคือ
พารามิเตอร์ TargetForm หมายถึง ฟอร์มเป้าหมาย
พารามิเตอร์ FadeStep หมายถึง ระดับความเข้ม-จางที่ลดลงเรื่อยๆ
    Public Sub FadeOut(ByVal TargetForm As Form, ByVal FadeStep As Double)
      If TargetForm.Visible = False Then          ถ้าฟอร์มอยู่ในสถานะมองไม่เห็น
        Return                                    ให้ออกจากเมธอด
      End If
ให้วนลูปเพื่อลดความเข้มของฟอร์ม ตามค่าของพารามิเตอร์ FadeStep
      For _i = 1 To _MinOpacity Step -FadeStep
        TargetForm.Opacity = _i
        TargetForm.Refresh()
      Next
    End Sub

เมธอด ShowFormFullScreen() ทำหน้าที่แสดงฟอร์มแบบเต็มหน้าจอ ต้องการพารามิเตอร์ 2 ตัว คือ
พารามิเตอร์ TargetForm หมายถึง ฟอร์มเป้าหมาย
พารามิเตอร์ OnOff หมายถึง คำสั่งเปิด-ปิด การแสดงฟอร์มเต็มหน้าจอ
โดยใช้คำว่า on ทำหน้าที่กำหนดให้แสดงฟอร์มแบบเต็มหน้าจอ
    Public Sub ShowFormFullScreen(ByVal TargetForm As Form, ByVal OnOff As String)
      If (OnOff.ToLower() = "on") Then            ถ้าพารามิเตอร์ OnOff เป็นคำว่า on แล้ว
        If (_IsFullScreen = False) Then           ถ้าฟอร์มไม่ได้อยู่ในสถานะเต็มหน้าจอแล้ว
เก็บรูปแบบขอบ และสถานะฟอร์มเดิมไว้ก่อน
          _BorderStyle = TargetForm.FormBorderStyle
          _WindowState = TargetForm.WindowState
กำหนดให้ฟอร์มเป้าหมายไม่ต้องแสดงขอบ
          TargetForm.FormBorderStyle = FormBorderStyle.None
กำหนดให้ฟอร์มเป้าหมายอยู่ในสถานะ Maximized
          TargetForm.WindowState = FormWindowState.Maximized
          TargetForm.ShowInTaskbar = False        ไม่ต้องแสดงฟอร์มใน Taskbar
          _IsFullScreen = True                    บอกสถานะฟอร์มไว้ว่า อยู่ในสถานะเต็มหน้าจอ
```

```
            End If
        Else                                        'แต่ถ้าพารามิเตอร์ OnOff เป็นค่าอื่นๆ
            If (_IsFullScreen = True) Then          'ถ้าอยู่ในสถานะเต็มหน้าจอแล้ว
ให้แสดงขอบและสถานะเดิมของฟอร์ม
                TargetForm.FormBorderStyle = _BorderStyle
                TargetForm.WindowState = _WindowState
                TargetForm.ShowInTaskbar = True     'ให้แสดงฟอร์มใน Taskbar
                _IsFullScreen = False               'บอกสถานะฟอร์มไว้ว่า ไม่อยู่ในสถานะเต็มหน้าจอ
            End If
        End If
    End Sub
End Class
End Namespace
```

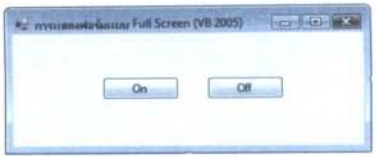
โค้ด VC# 2005 ที่ 19-10 การสร้างคลาสที่ทำงานเกี่ยวกับฟอร์ม (GAF.FormTools.cs)

```
namespace GAF
{
    public partial class FormTools
    {
        private double _i = 0.0;                    //ตัวนับวนลูป
        private int _MinOpacity = 0;                //ความจางของฟอร์มน้อยที่สุด
        private int _MaxOpacity = 1;                //ความเข้มของฟอร์มมากที่สุด (ปกติ)

        private FormBorderStyle _BorderStyle;       //ฟิลด์เก็บรูปแบบขอบของฟอร์ม
        private FormWindowState _WindowState;       //ฟิลด์เก็บสถานะของฟอร์ม
        private bool _IsFullScreen = false;         //ฟิลด์บอกสถานะว่า แสดงเต็มจออยู่หรือไม่

//เมธอด FadeIn() ทำหน้าที่เพิ่มความเข้มของฟอร์ม ต้องการพารามิเตอร์ 2 ตัวคือ
//พารามิเตอร์ TargetForm หมายถึง ฟอร์มเป้าหมาย
//พารามิเตอร์ FadeStep หมายถึง ระดับความเข้ม-จางที่เพิ่มขึ้นเรื่อยๆ
        public void FadeIn(Form TargetForm, double FadeStep)
        {
            if (TargetForm.Visible == false)        //ถ้าฟอร์มอยู่ในสถานะมองไม่เห็นแล้ว
            {
                TargetForm.Opacity = 0;             //ให้ความเข้มเป็น 0
                TargetForm.Visible = true;          //ให้มองเห็นฟอร์ม
            }
//ให้วนลูปเพื่อเพิ่มความเข้มของฟอร์ม ตามค่าของพารามิเตอร์ FadeStep
            for (_i = 0; _i <= _MaxOpacity; _i += FadeStep)
            {
                TargetForm.Opacity = _i;
                TargetForm.Refresh();
            }
        }

//เมธอด FadeOut() ทำหน้าที่เพิ่มความเข้มของฟอร์ม ต้องการพารามิเตอร์ 2 ตัวคือ
//พารามิเตอร์ TargetForm หมายถึง ฟอร์มเป้าหมาย
```

```
//พารามิเตอร์ FadeStep หมายถึง ระดับความเข้ม-จางที่ลดลงเรื่อยๆ
        public void FadeOut(Form TargetForm, double FadeStep)
        {
            if (TargetForm.Visible == false)                    //ถ้าฟอร์มอยู่ในสถานะมองไม่เห็น
            {
                return;                                         //ให้ออกจากเมธอด
            }
//ให้วนลูปเพื่อลดความเข้มของฟอร์ม ตามค่าของพารามิเตอร์ FadeStep
            for (_i = 1; _i >= _MinOpacity; _i += -FadeStep)
            {
                TargetForm.Opacity = _i;
                TargetForm.Refresh();
            }
        }

//เมธอด ShowFormFullScreen() ทำหน้าที่แสดงฟอร์มแบบเต็มหน้าจอ ต้องการพารามิเตอร์ 2 ตัวคือ
//พารามิเตอร์ TargetForm หมายถึง ฟอร์มเป้าหมาย
//พารามิเตอร์ OnOff หมายถึง คำสั่งเปิด-ปิด การแสดงฟอร์มเต็มหน้าจอ
//โดยใช้คำว่า on ทำหน้าที่กำหนดให้แสดงฟอร์มแบบเต็มหน้าจอ
        public void ShowFormFullScreen(Form TargetForm, string OnOff)
        {
            if (OnOff.ToLower() == "on")                        //ถ้าพารามิเตอร์ OnOff เป็นคำว่า on แล้ว
            {
                if (_IsFullScreen == false)                     //ถ้าฟอร์มไม่ได้อยู่ในสถานะเต็มหน้าจอแล้ว
                {
                    _BorderStyle = TargetForm.FormBorderStyle;  //เก็บรูปแบบขอบ และสถานะฟอร์มเดิมไว้ก่อน
                    _WindowState = TargetForm.WindowState;
//กำหนดให้ฟอร์มเป้าหมายไม่ต้องแสดงขอบ
                    TargetForm.FormBorderStyle = FormBorderStyle.None;
//กำหนดให้ฟอร์มเป้าหมายอยู่ในสถานะ Maximized
                    TargetForm.WindowState = FormWindowState.Maximized;
                    TargetForm.ShowInTaskbar = false;           //ไม่ต้องแสดงฟอร์มใน Taskbar
                    _IsFullScreen = true;                       //บอกสถานะฟอร์มไว้ว่า อยู่ในสถานะเต็มหน้าจอ
                }
            }
            else                                                //แต่ถ้าพารามิเตอร์ OnOff เป็นคำอื่นๆ
            {
                if (_IsFullScreen == true)                      //ถ้าอยู่ในสถานะเต็มหน้าจอแล้ว
                {
                    TargetForm.FormBorderStyle = _BorderStyle;  //ให้แสดงขอบและสถานะเดิมของฟอร์ม
                    TargetForm.WindowState = _WindowState;
                    TargetForm.ShowInTaskbar = true;            //ให้แสดงฟอร์มใน Taskbar
                    _IsFullScreen = false;                      //บอกสถานะฟอร์มไว้ว่า ไม่อยู่ในสถานะเต็มหน้าจอ
                }
            }
        }
    }
}
```

การใช้งานเมธอด ShowFormFullScreen() อยู่ใน Form1 ดังรูปที่ 19-17

รูปที่ 19-17
ฟอร์มในขณะออกแบบ



โค้ด VB 2005 และ VC# 2005 ที่ 19-10 การสร้างคลาสที่ทำงานเกี่ยวกับฟอร์ม

VB 2005 (Form1.vb)

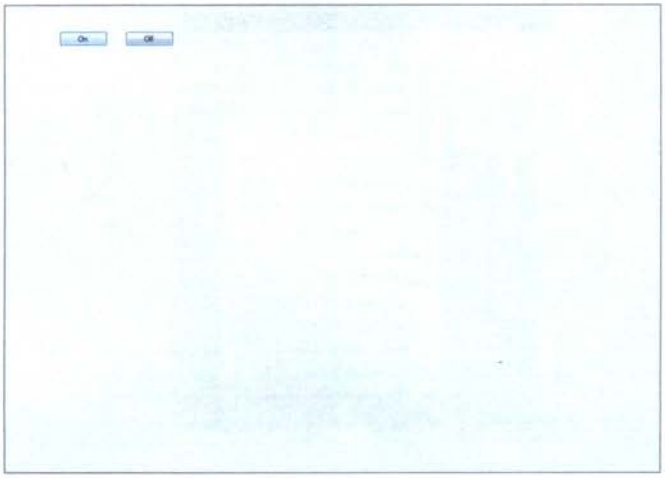
```
Option Explicit On
Option Strict On
Imports GAF

Public Class Form1
    Dim ft As New FormTools()

    Private Sub cmdOn_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cmdOn.Click
        ft.ShowFormFullScreen(Me, "on")
    End Sub

    Private Sub cmdOff_Click(ByVal sender As System.
Object, ByVal e As System.EventArgs) Handles cmdOff.Click
        ft.ShowFormFullScreen(Me, "off")
    End Sub
End Class
```

VC# 2005 (Form1.cs)

```
FormTools ft = new FormTools();

private void cmdOn_Click(object sender, EventArgs e)
{
    ft.ShowFormFullScreen(this, "on");
}

private void cmdOff_Click(object sender, EventArgs e)
{
    ft.ShowFormFullScreen(this, "off");
}
```

ปุ่ม Button ทั้ง 2 ทำหน้าที่เปิด-ปิดสถานะเต็มจอ ดังรูปที่ 19-18

รูปที่ 19-18
Form2 อยู่ในสถานะเต็มหน้าจอ



ส่วนการใช้งานเมธอด FadeIn() และเมธอด FadeOut() อยู่ใน Form2 ดังโค้ดต่อไปนี้

โค้ด VB 2005 และ VC# 2005 ที่ 19-10 การสร้างคลาสที่ทำงานเกี่ยวกับฟอร์ม

VB 2005 (Form2.vb)

```
Option Explicit On
Option Strict On
Imports GAF

Public Class Form2
    Private Sub Form2_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ft As New FormTools()
        ft.FadeIn(Me, 0.01)
    End Sub

    Private Sub Form2_FormClosing(ByVal sender As System.
Object, ByVal e As System.Windows.Forms.FormClosing
EventArgs) Handles MyBase.FormClosing
        Dim ft As New FormTools()
        ft.FadeOut(Me, 0.01)
    End Sub
End Class
```

VC# 2005 (Form2.cs)

```
    private void Form2_Load(object sender, EventArgs e)
    {
        FormTools ft = new FormTools();
        ft.FadeIn(this, 0.01);
    }

    private void Form2_FormClosing(object sender, Form
ClosingEventArgs e)
    {
        FormTools ft = new FormTools();
        ft.FadeOut(this, 0.01);
    }
```

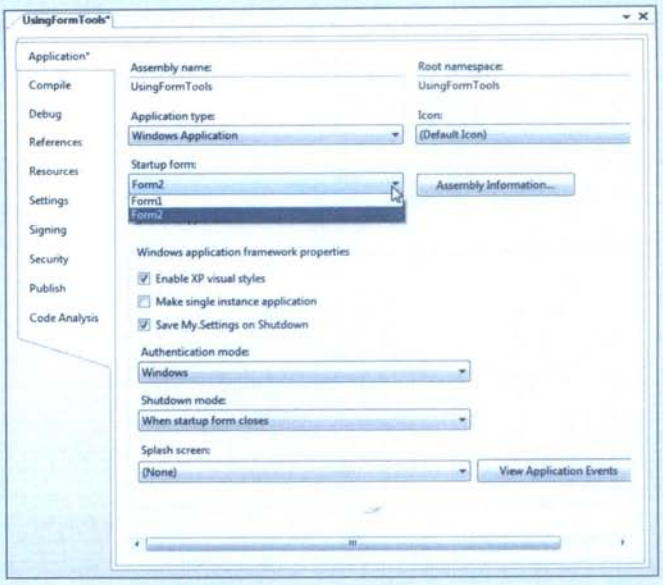
NOTE

การกำหนดฟอร์มเริ่มต้นของ VB 2005 และ VC# 2005 มีวิธีกำหนดที่แตกต่างกัน กล่าวคือ

1. ในกรณี VB 2005 คุณสามารถคลิกเมนู Project > UsingFormTools Properties... ที่รายการ Startup form: ให้เลือกฟอร์มที่ต้องการ ดังรูปที่ 19-19

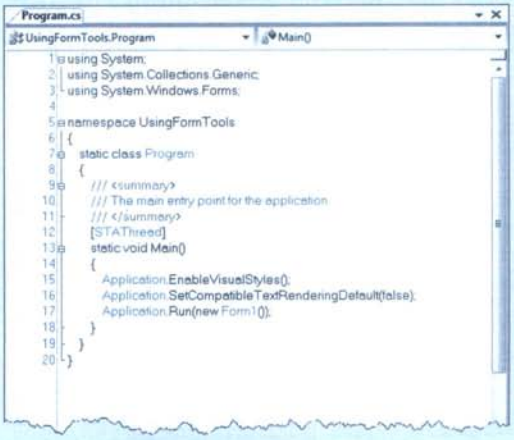
รูปที่ 19-19 การกำหนดฟอร์มเริ่มต้นของ VB 2005

รูปที่ 19-19 (ต่อ) การกำหนดฟอร์มเริ่มต้นของ VB 2005



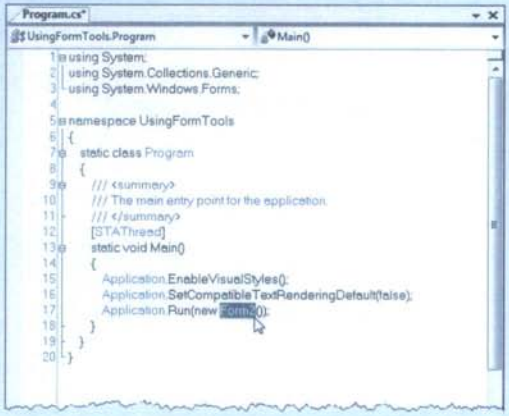
2. ในกรณี VC# 2005 ให้คุณเปิดคลาส Program.cs ในหน้าต่าง Solution Explorer ดังรูปที่ 19-20

รูปที่ 19-20 คลาส Program.cs ในหน้าต่าง Solution Explorer



จากรูปที่ 19-20 คลาส Program ของ VC# 2005 มีเมธอด Main() ส่งผลให้กลายเป็นจุดเริ่มต้นรันโปรเจ็กต์

3. ให้คุณกำหนดชื่อฟอร์มที่ต้องการให้เป็นฟอร์มเริ่มต้น ในกรณีนี้คือ ฟอร์ม Form2 ให้กับเมธอด Run() ของออบเจ็กต์ Application ดังรูปที่ 19-21



รูปที่ 19-21 การกำหนดฟอร์มเริ่มต้นของ VC# 2005

สรุปท้ายบท

เทคนิคต่างๆ ที่ผู้เขียนนำเสนอในบทนี้ เป็นเพียงส่วนหนึ่งที่ช่วยเสริมให้การใช้งาน Windows Form ของคุณมีประสิทธิภาพเพิ่มมากยิ่งขึ้น

CHAPTER 20

การสร้างผู้ช่วยเหลือในระบบธุรกิจ

บทนำ

เนื้อหาบทนี้เป็นบทสุดท้าย ผู้เขียนขอกล่าวถึงการสร้างผู้ช่วยเหลือในระบบธุรกิจ 2 ส่วนคือ การแปลงตัวเลขเป็นคำอ่านภาษาไทย และการคิดค่าเสื่อมราคาของระบบบัญชี (Accounting System)

เราจะลองสร้างคลาสที่ใช้กับแอพพลิเคชันด้านธุรกิจกันบ้าง ตั้งชื่อคลาสนี้ว่า MoneyTools ทำหน้าที่ 2 อย่างคือ

1. การเปลี่ยนตัวเลขอาราบิก (0123...89) เป็นตัวเลขไทย (๐๑๒๓...๘๙)
2. การอ่านตัวเลขด้านการเงินเป็นคำอ่านภาษาไทย

การทำงานทั้ง 2 กรณีข้างต้นสามารถใช้กับธุรกิจได้โดยทั่วไป จึงกำหนดให้อยู่ในเนมสเปซ ดังนั้น เมื่อคุณต้องการใช้งานคลาส MoneyTools คุณจึงต้องระบุเนมสเปซนี้ก่อนเสมอ

```
GAF.Biz
```

หลักการทำงานของคลาส MoneyTools

การเปลี่ยนตัวเลขอาราบิกเป็นตัวเลขไทย หลักการทำงานไม่ซับซ้อนแต่อย่างใด เราจะใช้วิธีการแทนที่ตัวเลขแต่ละตัวไปตามลำดับนั่นเอง ซึ่งมีเงื่อนไขอยู่ 2 อย่างคือ

1. ถ้าเป็นเลข 0 หรือเครื่องหมาย . ไม่ต้องเปลี่ยน
2. ถ้าเป็นเครื่องหมาย , ให้เอาออกก่อน

ส่วนวิธีการอ่านตัวเลขด้านการเงินของภาษาไทย ค่อนข้างมีเงื่อนไขในการอ่านยุ่งยากเล็กน้อย โดยเฉพาะอย่างยิ่งตัวเลข 1 และ 2 ส่วนตัวเลข 3 ถึง 9 ไม่มีปัญหาการอ่านแต่อย่างใด ดังนี้

ตัวเลข	คำอ่าน	ข้อสังเกต
1	หนึ่ง	กรณีเลข 1 เห็นได้ว่าเมื่อมีเลขตัวเดียว เราอ่านว่า หนึ่ง
11	สิบเอ็ด	แต่ถ้ามีเลข 2 ตัว เลข 1 ที่อยู่ในหลักหน่วยจะอ่านว่า เอ็ด ส่วนที่อยู่ในหลักสิบจะอ่านว่า สิบ
1000000	หนึ่งล้าน	ในกรณีเลข 1 อยู่ในหลักที่ 7 (หลักล้าน) ก็จะอ่านว่า หนึ่ง
11000000	สิบเอ็ดล้าน	แต่ถ้ามี 8 หลัก เลข 1 ในหลักที่ 7 กลับอ่านว่า เอ็ด ส่วนหลักที่ 8 อ่านว่า สิบ
101000000	หนึ่งร้อยหนึ่งล้าน	และถ้ามี 9 หลักขึ้นไป โดยที่หลักที่ 8 เป็นเลข 0 เลข 1 ในหลักที่ 7 กลับอ่านว่า หนึ่ง
111000000	หนึ่งร้อยสิบเอ็ดล้าน	มี 9 หลักเช่นกัน แต่หลักที่ 8 ไม่ใช่เลข 0 ตัวเลข 1 ในหลักที่ 7 กลับอ่านว่า เอ็ด
121	หนึ่งร้อยยี่สิบเอ็ด	เลข 2 เมื่ออยู่ในหลักที่ 2 จะอ่านว่า ยี่
102	หนึ่งร้อยสอง	แต่ถ้าเลข 2 อยู่ในหลักแรก จะอ่านว่า สอง
21000000	ยี่สิบเอ็ดล้าน	แต่ถ้าเลข 2 อยู่ในหลักที่ 8 จะอ่านว่า ยี่

จากตารางข้างต้นเห็นได้ว่า ตำแหน่งของตัวเลขเป็นตัวกำหนดคำอ่าน โดยที่มีเลข 0 เป็นเงื่อนไขเสริมที่มีผลต่อคำอ่านเลข 1 และ 2 ด้วยเช่นกัน โดยมีข้อจำกัดคือ

สามารถอ่านค่าตัวเลขแบบทศนิยม 2 ตำแหน่ง มีค่าอยู่ระหว่าง 0.00 ถึง 9,999,999,999,999.99 (น้อยกว่าสิบล้านล้าน)

ในกรณีที่มีการป้อนทศนิยม 3 ตำแหน่งขึ้นไป จะมีการปัดทศนิยมตั้งแต่ตำแหน่งที่ 3 ขึ้นไป ตามหลักคณิตศาสตร์

ให้ดูตัวอย่างที่ 20-1 การเปลี่ยนตัวเลขอาราบิกเป็นตัวเลขไทยและการอ่านตัวเลขด้านการเงินเป็นคำอ่านภาษาไทย

ผู้เขียนจะลงแต่โค้ดของ VB 2005 อย่างเดียวเท่านั้น โค้ดของคลาส MoneyTools แสดงดังนี้

โค้ด VB 2005 ที่ 20-1 การเปลี่ยนตัวเลขอาราบิกเป็นตัวเลขไทย และการอ่านตัวเลขด้านการเงินเป็นคำอ่านภาษาไทย (GAF.Biz.MoneyTools.vb)

```
Option Explicit On                          'ให้มีการประกาศตัวแปรก่อนการใช้งาน
Option Strict On                            'ให้เคร่งครัดต่อการเปลี่ยนแปลงชนิดของข้อมูล
Imports System.Text                         'เนมสเปซของคลาส StringBuilder

Namespace GAF.Biz                           'ชื่อเนมสเปซ
    Public Class MoneyTools                 'ชื่อคลาส

'เมธอด ToThaiNumber() แบบ Public ทำหน้าที่เปลี่ยนตัวเลขอาราบิก เป็นตัวเลขไทย
'ต้องการพารามิเตอร์ 1 ตัวคือ ตัวเลขทศนิยม 2 ตำแหน่ง
        Public Function ToThaiNumber(ByVal InputNumber As Double) As String
            If InputNumber.ToString() = "" Then     'ถ้าพารามิเตอร์ InputNumber เป็นค่าว่าง
                InputNumber = 0.0                   'ให้กำหนดค่าเริ่มต้นเป็น 0.0
            End If

            Dim _NewInputNumber As String = ""      'ฟิลด์เก็บตัวเลขที่ป้อนเข้ามา
'ให้เอาเครื่องหมาย , ออก และกำหนดให้มีทศนิยม 2 ตำแหน่ง
            _NewInputNumber = InputNumber.ToString("###0.00")

            Dim _i As Integer = 0               'ฟิลด์วนลูป
            Dim _CNumber As String = ""         'ฟิลด์เก็บตัวเลขปัจจุบัน
            Dim _FinalNumber As String = ""     'ฟิลด์เก็บตัวเลขไทย

'ให้วนลูปตั้งแต่ตัวเลขตัวแรกจนถึงตัวสุดท้าย
            For _i = 0 To _NewInputNumber.Length - 1
'ให้อ่านตัวเลขปัจจุบันออกมา 1 ตัว เก็บไว้ที่ฟิลด์ _CNumber
                _CNumber = _NewInputNumber.Substring(_i, 1)
                Select Case _Cnumber            'ตรวจสอบตัวเลขปัจจุบัน
                    Case "0"                    'กรณีเลข 0
                        _CNumber = "๐"
                    Case "1"                    'กรณีเลข 1
                        _CNumber = "๑"
                    Case "2"                    'กรณีเลข 2
                        _CNumber = "๒"
                    Case "3"                    'กรณีเลข 3
                        _CNumber = "๓"
                    Case "4"                    'กรณีเลข 4
                        _CNumber = "๔"
                    Case "5"                    'กรณีเลข 5
                        _CNumber = "๕"
                    Case "6"                    'กรณีเลข 6
                        _CNumber = "๖"
                    Case "7"                    'กรณีเลข 7
                        _CNumber = "๗"
```

```
            Case "8"                                'กรณีเลข 8
                _CNumber = "๘"
            Case "9"                                'กรณีเลข 9
                _CNumber = "๙"
            Case "."                                'กรณีเครื่องหมาย
                _CNumber = "."
        End Select
'สะสมตัวเลขที่เปลี่ยนเป็นตัวเลขไทยแล้ว ไว้ในฟิลด์ _FinalNumber
        _FinalNumber &= _CNumber
    Next

    ToThaiNumber = _FinalNumber
    Return ToThaiNumber                             'คืนค่าเป็นตัวเลขไทย
End Function


'เมธอด NumberToThaiWord() แบบ Public ทำหน้าที่แปลงตัวเลขด้านการเงิน เป็นคำอ่านภาษาไทย
'ต้องการพารามิเตอร์ 1 ตัวคือ ตัวเลขทศนิยม 2 ตำแหน่ง
Public Function NumberToThaiWord(ByVal InputNumber As Double) As String
    If InputNumber.ToString() = "" Then             'ถ้าตัวเลขที่ป้อนเข้ามาเป็นค่าว่าง
        InputNumber = 0.0                           'กำหนดค่าเริ่มต้นเป็น 0.0
    End If

    If InputNumber = 0 Then                         'ถ้ามีค่าเป็น 0
        NumberToThaiWord = "ศูนย์บาทถ้วน"
        Return NumberToThaiWord
    End If

    Dim _NewInputNumber As String = ""              'ฟิลด์เก็บตัวเลขที่ป้อนเข้ามา
'เปลี่ยนตัวเลขที่ป้อนเข้ามาเป็นข้อมูลชนิด String กำหนดให้เอาเครื่องหมาย , ออก
'และกำหนดให้มีทศนิยม 2 ตำแหน่ง ถ้ามีมากกว่า 2 ตำแหน่ง จะปัดเศษตามหลักคณิตศาสตร์

    _NewInputNumber = InputNumber.ToString("###0.00")

'ถ้าตัวเลขที่ป้อนเข้ามามีค่ามากกว่าหรือเท่ากับ 10 ล้านล้าน
    If CDbl(_NewInputNumber) >= 10000000000000 Then
        NumberToThaiWord = ""                       'คืนค่าว่าง
        Return NumberToThaiWord
    End If

    Dim _tmpNumber(2) As String                     'ฟิลด์อาร์เรย์เก็บตัวเลขด้านหน้าและหลังจุดทศนิยม
    Dim _FirstNumber As String = ""                 'ฟิลด์เก็บตัวเลขด้านหน้าจุดทศนิยม
    Dim _LastNumber As String = ""                  'ฟิลด์เก็บตัวเลขหลังจุดทศนิยม

'แยกตัวเลขที่ป้อนเข้ามาด้วยตัวอักษรจุด
    _tmpNumber = _NewInputNumber.Split(CChar("."))
    _FirstNumber = tmpNumber(0)                     'ตัวเลขหน้าจุดทศนิยม อ่านได้จากฟิลด์อาร์เรย์ตัวแรก
    _LastNumber = tmpNumber(1)                      'ตัวเลขหลังจุดทศนิยม อ่านได้จากฟิลด์อาร์เรย์ตัวที่ 2
```

```
        Dim _nLength As Integer = 0                 'ฟิลด์เก็บจำนวนตัวเลขว่า มีกี่ตัว
'อ่านจำนวนตัวเลขหน้าจุดทศนิยมว่ามีกี่ตัว เก็บไว้ในฟิลด์ _nLength
        _nLength = CInt(_FirstNumber.Length)

        Dim _i As Integer = 0                       'ฟิลด์วนลูป
        Dim _CNumber As Integer = 0                 'ฟิลด์เก็บตัวเลขปัจจุบัน
        Dim _CNumberBak As Integer = 0              'ฟิลด์เก็บตัวเลขสำรองไว้
        Dim _strNumber As String = ""               'ฟิลด์เก็บคำอ่านตัวเลข
        Dim _strPosition As String = ""             'ฟิลด์เก็บคำอ่านของหน่วย
        Dim _CountPos As Integer = 0                'ฟิลด์นับตำแหน่งตัวเลขปัจจุบัน

    Dim _FinalWord As String = ""                   'ฟิลด์เก็บข้อความที่อ่านได้
        Dim _sw As New StringBuilder()              'ออบเจ็กต์ StringBuilder ที่ชื่อว่า _sw
        _sw.Remove(0, _sw.Length)                   'ล้างค่าก่อน


        For _i = _nLength To 1 Step -1              'วนลูปตั้งแต่หลักแรกจนถึงหลักสุดท้าย
            _CNumberBak = _Cnumber                  'สำรองตัวเลขปัจจุบันเก็บไว้ก่อน
'อ่านตัวเลขปัจจุบันครั้งละ 1 ตัว เก็บไว้ที่ฟิลด์ _CNumber
            _CNumber = CInt(_FirstNumber.Substring(_CountPos, 1))
'ถ้าเป็นเลข 0 และอยู่ในหลักที่ 7 (หลักล้าน)
            If _CNumber = 0 AndAlso _i = 7 Then
                _strPosition = "ล้าน"               'ให้อ่านว่าล้าน
            ElseIf _CNumber = 0 Then                'แต่ถ้าเป็นเลข 0
                _strPosition = ""                   'ไม่ต้องอ่าน
            Else                                    'แต่ถ้าเป็นเลขอื่นๆ
                _strPosition = PositionToText(_i)   'เรียกฟังก์ชัน PositionToText() ทำงาน
            End If
'ถ้าเป็นเลข 2 และอยู่ในหลักที่ 2 (หลักสิบ)
            If _CNumber = 2 AndAlso _strPosition = "สิบ" Then
                _strNumber = "ยี่"                  'ให้อ่านว่า ยี่
'ถ้าเป็นเลข 1 และอยู่ในหลักสิบ
            ElseIf _CNumber = 1 AndAlso _strPosition = "สิบ" Then
                _strNumber = ""                     'ไม่ต้องอ่าน
'แต่ถ้าเป็นเลข 1 และอยู่ในหลักที่ 7 (หลักล้าน) และมีจำนวนตัวเลขมากกว่า 8 ตัวขึ้นไป
            ElseIf _CNumber = 1 AndAlso _strPosition = "ล้าน" AndAlso _nLength >= 8 Then
                If _CNumberBak = 0 Then             'ถ้าตัวเลขก่อนหน้าเป็นเลข 0
                    _strNumber = "หนึ่ง"            'ให้อ่านว่า หนึ่ง
                Else                                'แต่ถ้าตัวเลขก่อนหน้า เป็นเลขอื่นๆ
                    _strNumber = "เอ็ด"             'ให้อ่านว่า เอ็ด
                End If
'แต่ถ้าเป็นเลข 1 และคำอ่านตำแหน่งปัจจุบันเป็นค่าว่าง และมีจำนวนตัวเลขมากกว่า 1 ตัวขึ้นไป
            ElseIf _CNumber = 1 AndAlso _strPosition = "" AndAlso _nLength > 1 Then
                _strNumber = "เอ็ด"                 'ให้อ่านว่า เอ็ด
            Else                                    'แต่ถ้าเป็นเลขอื่นๆ ให้เรียกเมธอด NumberToText() ทำงาน
                _strNumber = NumberToText(_CNumber)
            End If
```

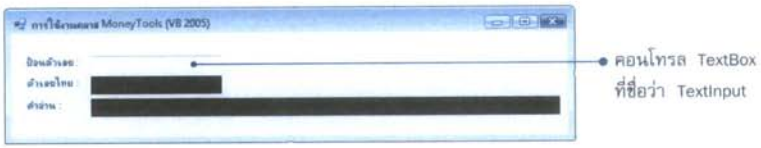
```
            _CountPos = _CountPos + 1           'เพิ่มค่าตัวนับอีก 1
'สะสมคำอ่านที่ได้ไว้ในออบเจ็กต์ _sw รวมกับเมธอด Append
            _sw.Append(_strNumber & _strPosition)
        Next
        _FinalWord = _sw.ToString()             'ส่งคำอ่านได้ไว้ในตัวแปร _FinalWord

        _CountPos = 0                           'ล้างค่าฟิลด์ใหม่
        _CNumberBak = 0
'อ่านจำนวนตัวเลขหลังจุดทศนิยมว่ามีกี่ตัว เก็บไว้ในฟิลด์ nLength
        _nLength = CInt(_LastNumber.Length)

        Dim _Stang As String = ""               'ฟิลด์เก็บหน่วยสตางค์
        Dim _FinalStang As String = ""          'ฟิลด์เก็บหน่วยสตางค์ที่อ่านได้
        _sw.Remove(0, _sw.Length)               'ล้างค่าก่อน

        If CDbl(_LastNumber) > 0.0 Then         'ถ้าเลขหลังทศนิยมมีค่ามากกว่า 0 แล้ว
          For _i = _nLength To 1 Step -1        'วนลูปตั้งแต่หลักแรก จนถึงหลักสุดท้าย
            _CNumberBak = _Cnumber              'เก็บค่าตัวเลขก่อนหน้าไว้ก่อน
'อ่านตัวเลขปัจจุบันครั้งละ 1 ตัว เก็บไว้ที่ฟิลด์ _CNumber
            _CNumber = CInt(_LastNumber.Substring(_CountPos, 1))
'ถ้าเป็นเลข 1 และอยู่ในหลักที่ 2 (หลักสิบ)
            If _CNumber = 1 AndAlso _i = 2 Then
              _strPosition = "สิบ"              'ให้อ่านว่า สิบ
            ElseIf _CNumber = 0 Then            'แต่ถ้าเป็นเลข 0
              _strPosition = ""                 'ไม่ต้องอ่าน
            Else                                'แต่ถ้าเป็นค่าอื่นๆ
'เรียกฟังก์ชัน PositionToText() ทำงาน
              _strPosition = PositionToText(_i)
            End If
'ถ้าเป็นเลข 2 และอยู่ในหลักที่ 2 (หลักสิบ)
            If _CNumber = 2 AndAlso _strPosition = "สิบ" Then
              _Stang = "ยี่"                    'ให้อ่านว่า ยี่
'ถ้าเป็นเลข 1 และอยู่ในหลักที่ 2 (หลักสิบ)
            ElseIf _CNumber = 1 AndAlso _i = 2 Then
              _Stang = ""                       'ไม่ต้องอ่าน
'แต่ถ้าเป็นเลข 1 และคำอ่านปัจจุบันเป็นค่าว่าง และมีจำนวนตัวเลขมากกว่า 1 ตัวขึ้นไป
            ElseIf _CNumber = 1 AndAlso _strPosition = "" AndAlso _nLength > 1 Then
              If _CNumberBak = 0 Then           'ถ้าตัวเลขก่อนหน้าเป็นเลข 0
                  _Stang = "หนึ่ง"              'ให้อ่านว่า หนึ่ง
              Else                              'แต่ถ้าเป็นตัวเลขอื่นๆ
                  _Stang = "เอ็ด"               'ให้อ่านว่า เอ็ด
              End If
            Else                                'แต่ถ้าเป็นตัวเลขอื่นๆ
'เรียกเมธอด NumberText() ทำงาน
              _Stang = NumberToText(_CNumber)
            End If

            _CountPos = _CountPos + 1           'เพิ่มค่าตัวนับอีก 1
```

```
'สะสมคำอ่านสตางค์ไว้ที่ออบเจ็กต์ _sw
            _sw.Append(_Stang & _strPosition)
        Next

        _sw.Append("สตางค์")
        _FinalStang = _sw.ToString()          'ส่งคำอ่านสตางค์ไว้ที่ตัวแปร _FinalStang
      Else                                    'แต่ถ้าตัวเลขหลังจุดทศนิยมมีค่าเป็น 0
        _FinalStang = ""                      'ไม่มีคำอ่านของหน่วยสตางค์
      End If

      Dim _SubUnit As String = ""
      If _FinalStang = "" Then                'ถ้าไม่มีหน่วยสตางค์
        _SubUnit = "บาทถ้วน"                  'ให้ลงท้ายด้วยคำว่า บาทถ้วน
      Else                                    'แต่ถ้ามีหน่วยสตางค์
'ถ้าตัวเลขหน้าจุดทศนิยมไม่เท่ากับ 0
        If CDbl(_FirstNumber) <> 0.0 Then
          _SubUnit = "บาท"                    'ให้ลงท้ายด้วยคำว่า บาท
        Else                                  'แต่ถ้ามีค่าเท่ากับ 0
          _SubUnit = ""                       'ไม่ต้องอ่าน
        End If
      End If
'รวมคำอ่านตัวเลขหน้าและหลังจุดทศนิยม
      NumberToThaiWord = _FinalWord & _SubUnit & _FinalStang
    End Function

'เมธอด NumberToText() แบบ Private ทำหน้าที่แปลงตัวเลขเป็นคำอ่าน
'ต้องการพารามิเตอร์ 1 ตัวคือ ตัวเลขที่ต้องการอ่าน
    Private Function NumberToText(ByVal CurrentNum As Integer) As String
      Dim _nText As String = ""

      Select Case CurrentNum
        Case 0                                'เลข 0
          _nText = ""
        Case 1                                'เลข 1
          _nText = "หนึ่ง"
        Case 2                                'เลข 2
          _nText = "สอง"
        Case 3                                'เลข 3
          _nText = "สาม"
        Case 4                                'เลข 4
          _nText = "สี่"
        Case 5                                'เลข 5
          _nText = "ห้า"
        Case 6                                'เลข 6
          _nText = "หก"
        Case 7                                'เลข 7
          _nText = "เจ็ด"
        Case 8                                'เลข 8
          _nText = "แปด"
```

```
            Case 9                          'เลข 9
                _nText = "เก้า"
            End Select

            NumberToText = _nText           'คืนค่าเป็นคำอ่าน
        End Function

'เมธอด PositionToText() แบบ Private ทำหน้าที่แปลงตำแหน่งตัวเลขปัจจุบัน เป็นคำอ่าน
'ต้องการพารามิเตอร์ 1 ตัวคือ ตำแหน่งตัวเลขปัจจุบัน
        Private Function PositionToText(ByVal CurrentPos As Integer) As String
            Dim _nPos As String = ""

            Select Case CurrentPos
                Case 0
                    _nPos = ""
                Case 1                      'หลักหน่วย
                    _nPos = ""
                Case 2                      'หลักสิบ
                    _nPos = "สิบ"
                Case 3                      'หลักร้อย
                    _nPos = "ร้อย"
                Case 4                      'หลักพัน
                    _nPos = "พัน"
                Case 5                      'หลักหมื่น
                    _nPos = "หมื่น"
                Case 6                      'หลักแสน
                    _nPos = "แสน"
                Case 7                      'หลักล้าน
                    _nPos = "ล้าน"
                Case 8                      'หลักสิบล้าน
                    _nPos = "สิบ"
                Case 9                      'หลักร้อยล้าน
                    _nPos = "ร้อย"
                Case 10                     'หลักพันล้าน
                    _nPos = "พัน"
                Case 11                     'หลักหมื่นล้าน
                    _nPos = "หมื่น"
                Case 12                     'หลักแสนล้าน
                    _nPos = "แสน"
                Case 13                     'หลักล้านล้าน
                    _nPos = "ล้าน"
            End Select

            PositionToText = _nPos          'คืนค่าเป็นคำอ่าน
        End Function
    End Class
End Namespace
```

ให้คุณเพิ่มโปรเจ็กต์ใหม่เข้ามา เพื่อทดสอบการทำงานของคลาส MoneyTools และออกแบบฟอร์ม ดังรูปที่ 20-1

รูปที่ 20-1 ฟอร์มในขณะออกแบบ

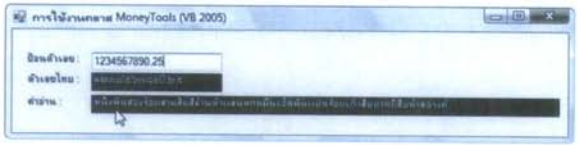


จากนั้นให้คุณเขียนโค้ดดังต่อไปนี้

โค้ด VB 2005 ที่ 20-1 การเปลี่ยนตัวเลขอาราบิกเป็นตัวเลขไทย และการอ่านตัวเลขด้านการเงินเป็นคำอ่านภาษาไทย (Form1.vb)

```
Option Explicit On                          ให้มีการประกาศตัวแปรก่อนการใช้งาน
Option Strict On                            ให้เคร่งครัดต่อการเปลี่ยนแปลงชนิดของข้อมูล
Imports GAF.Biz                             เนมสเปซของคลาส MoneyTools

Public Class Form1

'เหตุการณ์เปลี่ยนข้อความในคอนโทรล txtInput
    Private Sub txtInput_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles txtInput.TextChanged
        Dim mne As New MoneyTools           'ตัวแปรออบเจ็กต์ MoneyTools ที่ชื่อว่า mne

        Try                                 ให้ทำ
'เรียกใช้เมธอด ToThaiNumber() และเมธอด NumberToThaiWord()
            lblThaiNumber.Text = mne.ToThaiNumber(CDbl(txtInput.Text))
            lblThaiWord.Text = mne.NumberToThaiWord(CDbl(txtInput.Text))
        Catch ex As Exception               'กรณีเกิดข้อผิดพลาด
            lblThaiNumber.Text = ''         ให้ล้างข้อความในคอนโทรล Label ทั้ง 2 ตัว
            lblThaiWord.Text = ''
        End Try
    End Sub
End Class
```

รูปที่ 20-2
ผลการทำงาน
ตัวอย่างที่ 20-1



จากรูปที่ 20-2 เมื่อคุณพิมพ์ตัวเลขลงในคอนโทรล TextBox จะมีการเปลี่ยนให้เป็นตัวเลขไทย และแสดงคำอ่านตัวเลขดังกล่าวออกมา

อธิบายการทำงาน

สำหรับตัวอย่างที่ 20-1 มีคำอธิบายเพื่อเพิ่มความเข้าใจดังนี้

1. การเปลี่ยนตัวเลขอาราบิกเป็นตัวเลขไทย อยู่ในความรับผิดชอบของเมธอด ToThaiNumber() เป็นเมธอดแบบ Public คืนค่าเป็นข้อมูลชนิด String ก็จะอาศัยหลักการแทนที่ตัวเลขทีละตัวด้วยเมธอด Substring() ของคลาส String นั่นเอง โดยที่ถ้าเป็นเครื่องหมาย . หรือตัวเลข 0 ให้คืนค่าเดิม

VB 2005

```
Public Function ToThaiNumber(ByVal InputNumber As Double) As String
    If InputNumber.ToString() = "" Then
        InputNumber = 0.0
    End If

    Dim _NewInputNumber As String = ""
    _NewInputNumber = InputNumber.ToString("###0.00")

    Dim _i As Integer = 0
    Dim _CNumber As String = ""
    Dim _FinalNumber As String = ""

    For _i = 0 To _NewInputNumber.Length - 1
        _CNumber = _NewInputNumber.Substring(_i, 1)
        Select Case _CNumber
            Case "0"
                _CNumber = "๐"
            Case "1"
                _CNumber = "๑"
            Case "2"
                _CNumber = "๒"
            Case "3"
                _CNumber = "๓"
            Case "4"
                _CNumber = "๔"
            Case "5"
                _CNumber = "๕"
```

```
            Case "6"
                _CNumber = "๖"
            Case "7"
                _CNumber = "๗"
            Case "8"
                _CNumber = "๘"
            Case "9"
                _CNumber = "๙"
            Case ""
                _CNumber = ""
        End Select

        _FinalNumber &= _CNumber
    Next

    ToThaiNumber = _FinalNumber
    Return ToThaiNumber
End Function
```

2. ส่วนการอ่านตัวเลขเป็นคำอ่านภาษาไทย อยู่ในความรับผิดชอบของเมธอด NumberToThaiWord() เป็นเมธอดแบบ Public คืนค่าเป็นข้อมูลชนิด String เห็นได้ว่ามีการเรียกใช้เมธอดแบบ Private อีก 2 เมธอดคือ
   - เมธอด NumberToText() ทำหน้าที่แปลงตัวเลข (012...89) เป็นคำอ่าน (ศูนย์หนึ่งสอง... แปดเก้า)
   - เมธอด PositionToText() ทำหน้าที่แปลงตำแหน่งของตัวเลขเป็นคำอ่าน เช่น สิบ, ยี่, ร้อย, พัน เป็นต้น

เหตุผลที่กำหนดให้ทั้ง 2 เมธอดข้างต้นเป็นแบบ Private เพราะว่าคำอ่านที่ได้จากทั้ง 2 เมธอดดังกล่าวยังไม่สมบูรณ์ ถือเป็นการทำงานภายในคลาส ภายนอกไม่ควรรับรู้นั่นเอง และนี่เป็นอีก 1 ตัวอย่างที่แสดงให้เห็นว่าเมธอดแบบ Private ถูกเรียกใช้งานภายในคลาส (ภายในตัวมันเอง)

การคิดค่าเสื่อมราคาในระบบบัญชี

ในการทำระบบบัญชี (Accounting System) มีงานอยู่อย่างหนึ่งที่เราสามารถแยกออกมาทำเป็นคลาสได้นั่นคือ การคิดค่าเสื่อมราคาของสินทรัพย์ (Depreciation)

สินทรัพย์ (Asset) ที่คุณนำมาใช้ในกิจการย่อมที่จะมีมูลค่าลดลงไปตามเวลา มูลค่าของสินทรัพย์ที่ลดลงไปดังกล่าว ในทางบัญชีเรียกว่าค่าเสื่อมราคา โดยที่เราจะนำค่าเสื่อมราคาดังกล่าวไปลดมูลค่าของสินทรัพย์ต่างๆ ที่อยู่ในกิจการของคุณ การคิดค่าเสื่อมราคาเท่าที่ผู้เขียนทราบมีอยู่ 6 วิธี ดังนี้

1. แบบเส้นตรง (Straight-Line)
2. แบบคิดตามหน่วยที่ผลิต (Units Of Output)
3. แบบ 2 เท่าของเส้นตรง โดยคิดจากยอดที่ลดลง (Double Declining Balance)
4. แบบคิดอัตราลดลง (Decreasing Charge)
5. แบบคำนวณสินทรัพย์คงเหลือ (Inventory)
6. แบบผลบวกของลำดับปีที่ใช้งาน (Sum of the Years Degits)

NOTE

สำหรับรายละเอียดและข้อดีข้อเสียของการคำนวณค่าเสื่อมราคาทั้ง 6 วิธี ผู้เขียนแนะนำให้คุณผู้อ่านศึกษาได้จากตำราทางด้านบัญชีโดยเฉพาะ ซึ่งอธิบายได้ครบถ้วนดีกว่าผู้เขียน

เพื่อให้เหมาะสมกับการนำเสนอ ผู้เขียนขอถือว่ารอบบัญชีคือ วันที่ 1 มกราคม 25xx ถึง 31 ธันวาคม 25xx เพื่อให้คำว่า ปีใหม่ และรอบบัญชีใหม่คือวันเดียวกัน นั่นคือ วันที่ 1 มกราคม 25xx

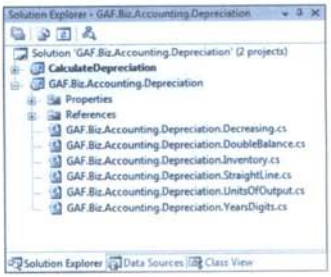
โดยที่ผู้เขียนขอนำเสนอแต่วิธีการคำนวณเท่านั้น โดยตั้งชื่อคลาสนี้ว่า Depreciation กำหนดให้อยู่ในเนมสเปซ ดังนี้

```
GAF.Biz.Accounting
```

เนื่องจากการคิดค่าเสื่อมราคาใช้ในระบบบัญชี จึงกำหนดให้อยู่ในระดับชั้นย่อยที่ต่อจาก Biz เพื่อแยกคลาสตามธุรกิจแต่ละประเภทต่อไป โดยที่ผู้เขียนจะลงเฉพาะโค้ดของภาษา VC# 2005 เท่านั้น

เพื่อให้เหมาะสมกับการนำเสนอ ผู้เขียนจะอธิบายวิธีการหรือสูตรการคิดค่าเสื่อมราคาของวิธีนั้นๆ ก่อนแล้วจึงค่อยโยงเข้าสู่คลาส Depreciation โดยที่ผู้เขียนใช้ความสามารถของการทำ Partial Class เข้ามาช่วยกำหนดให้ทั้ง 6 วิธี แยกกันอยู่ 6 ไฟล์ กำหนดให้วิธีคำนวณแบบเส้นตรงเป็นจุดเริ่มต้น ดังนี้

รูปที่ 20-3
Partial Class ของคลาส Depreciation ทั้ง 6 ไฟล์ใน Solution Explorer



การคิดค่าเสื่อมราคาแบบเส้นตรง (Straight-Line)

สินทรัพย์ที่นำมาคำนวณแบบเส้นตรง จะถือว่ากิจการได้รับผลประโยชน์จากการใช้สินทรัพย์เท่ากันทุกปี (หรือทุกเดือน) ส่งผลให้ค่าเสื่อมราคาเท่ากันทุกปี (หรือทุกเดือน) นั่นเอง สูตรการคำนวณคือ

$$\text{ค่าเสื่อมราคา} = \frac{(C-S)}{Y}$$

โดยที่ C = ต้นทุนสินทรัพย์, S = ราคาซากเมื่อขายสินทรัพย์, Y = อายุการใช้งานสินทรัพย์ สามารถแยกคำนวณได้ 2 ลักษณะคือ

- แบบเต็มรอบบัญชี หมายถึง กิจการเริ่มใช้สินทรัพย์ในวันเริ่มต้นของรอบบัญชี
- แบบระหว่างรอบบัญชี หมายถึง กิจการเริ่มใช้สินทรัพย์ระหว่างรอบบัญชี เช่น

บริษัท ABC จำกัด สั่งซื้อเครื่องจักรในราคา 300,000 บาท นำมาใช้งานในวันที่ 1 มกราคม อายุการใช้งาน 4 ปี ราคาซากที่คาดว่าจะขายเครื่องจักรนี้คือ 75,000 บาท คำนวณหาค่าเสื่อมราคาของเครื่องจักรแบบเส้นตรงได้ ดังนี้

$$\text{ค่าเสื่อมราคาต่อปี} = \frac{(300,000-75,000)}{4}$$

ผู้เขียนเจาะจงให้วันที่เริ่มใช้เครื่องจักร เป็นวันเดียวกับวันที่เริ่มรอบบัญชีใหม่ (1 มกราคม 25xx) ผลที่ได้คือ ตลอดอายุการใช้งานทั้ง 4 ปี จะคิดค่าเสื่อมราคาปีละ 56,250 บาท

สมมติว่าเปลี่ยนวันที่เริ่มใช้เครื่องจักรเป็นวันที่ 1 ตุลาคม 25xx พบว่าในรอบบัญชีของปีแรก เครื่องจักรถูกใช้ไปเพียง 3 เดือนเท่านั้น ส่งผลให้อายุของเครื่องจักรจะกินระยะรอบบัญชีไปถึงปีที่ 5 โดยที่ในรอบบัญชีปีสุดท้าย จะเหลือเศษของอายุการใช้งานอยู่ 9 เดือน ดังรูปที่ 20-4

รูปที่ 20-4
กรณีเริ่มใช้งานเครื่องจักรระหว่างรอบบัญชี

รายละเอียดของคลาส Depreciation เฉพาะการคำนวณแบบเส้นตรงแสดงดังนี้

โค้ด VC# 2005 ที่ 20-2 การคำนวณค่าเสื่อมราคาแบบเส้นตรง (GAF.Biz.Accounting.Depreciation.StraightLine.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GAF.Biz.Accounting                    //เนมสเปซของคลาส Depreciation
{
    public partial class Depreciation           //กำหนดให้เป็นแบบ Partial Class
    {
        private int _i = 0;                     //ฟิลด์ _i ทำหน้าที่วนลูป
        private double _CostOfAsset = 0.0;      //ต้นทุนสินค้า
        private double _SalvageValue = 0.0;     //ราคาซาก
        private int _EstimatedUseLife = 1;      //อายุการใช้งานสินทรัพย์
//จำนวนเดือนที่ใช้สินทรัพย์ในปีแรกเท่านั้น กำหนดค่าเริ่มต้น 12 หมายถึง ปีแรกใช้ครบทั้งปี
        private int _MonthInFirstYear = 12;

        public double CostOfAsset               //คุณสมบัติ CostOfAsset คืนค่าเป็นข้อมูลชนิด double
        {
            get{return _CostOfAsset;}           //ให้อ่านค่าได้
            set{_CostOfAsset = value;}          //ให้กำหนดค่าได้
        }

        public double SalvageValue              //คุณสมบัติ SalvageValue คืนค่าเป็นข้อมูลชนิด SalvageValue
        {
            get{return _SalvageValue;}          //ให้อ่านค่าได้
            set{_SalvageValue = value;}         //ให้กำหนดค่าได้
        }

        public int EstimatedUseLife             //คุณสมบัติ EstimatedUseLife คืนค่าเป็นข้อมูลชนิด int
        {
            get{return _EstimatedUseLife;}      //ให้อ่านค่าได้
            set{_EstimatedUseLife = value;}     //ให้คืนค่าได้
        }

//default คอนสตรัคเตอร์ ถือเป็นคอนสตรัคเตอร์ตัวที่ 1
        public Depreciation()
        {

        }

//คอนสตรัคเตอร์ตัวที่ 2 ต้องการพารามิเตอร์ 3 ตัวคือ
//พารามิเตอร์ CostOfAsset หมายถึง ต้นทุนสินค้า
//พารามิเตอร์ SalvageValue หมายถึง ราคาซาก
//พารามิเตอร์ EstimatedUseLife หมายถึง อายุการใช้งาน
```

```
        public Depreciation(double CostOfAsset, double SalvageValue, int EstimatedUseLife)
        {
            if (SalvageValue > CostOfAsset)
            {
                throw (new Exception("กรุณาระบุราคาซาก (SalvageValue) น้อยกว่าราคาสินทรัพย์ (CostOfAsset)"));
            }

            if (EstimatedUseLife == 0)
            {
                throw (new Exception("กรุณาระบุอายุการใช้งาน (EstimatedUseLife) ที่มีค่ามากกว่า 0"));
            }

            _CostOfAsset = CostOfAsset;                    //กำหนดค่าให้กับฟิลด์ต่างๆ
            _SalvageValue = SalvageValue;
            _EstimatedUseLife = EstimatedUseLife;
        }
//เมธอด StraightLine() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบเส้นตรง คืนค่าเป็นข้อมูลชนิด double
        public double StraightLine()
        {
            double _DepreciationPerYear = 0.0;             //ฟิลด์เก็บค่าเสื่อมราคาแต่ละปีที่คำนวณได้
//คำนวณค่าเสื่อมราคาตามสูตร
            _DepreciationPerYear =(_CostOfAsset - _SalvageValue) / _EstimatedUseLife;

            return _DepreciationPerYear;                   //คืนค่าเป็นผลการคำนวณค่าเสื่อมราคา
        }
//เมธอด StraighLine() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบเส้นตรงเช่นกัน
//แต่ใช้กับสินทรัพย์ที่คิดระหว่างรอบบัญชี ต้องการพารามิเตอร์ 1 ตัวคือ
//พารามิเตอร์ MonthInFirstYear หมายถึง จำนวนเดือนที่ใช้สินทรัพย์ในปีแรกเท่านั้น
//คืนค่าเป็นอาร์เรย์ของ double
        public double[] StraightLine(int MonthInFirstYear)
        {
            double _NormalDepreciation = 0.0;              //ฟิลด์เก็บค่าเสื่อมราคาที่ได้ในแต่ละปี
//สั่งให้เมธอด StraightLine() ทำงาน ผลการทำงานเก็บไว้ที่ฟิลด์ _NormalDepreciation
            _NormalDepreciation = this.StraightLine();
//อ่านจำนวนเดือนที่ใช้สินทรัพย์ในปีแรก เก็บไว้ที่ฟิลด์ _MonthInFirstYear
            _MonthInFirstYear = MonthInFirstYear;
//สร้างอาร์เรย์ของ double ที่ชื่อว่า _TotalDepreciation
//ให้มีจำนวนสมาชิกมากกว่าอายุการใช้งานสินทรัพย์อยู่ 1
            double[] _TotalDepreciation = new double[_EstimatedUseLife + 1];
            Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife+1); //ล้างค่าฟิลด์อาร์เรย์

            double _DepreciationOfFirstYear = 0.0;         //ฟิลด์เก็บค่าเสื่อมราคาปีแรก
            double _DepreciationOfLastYear = 0.0;          //ฟิลด์เก็บค่าเสื่อมราคาปีสุดท้าย
//คำนวณค่าเสื่อมราคาของปีแรก และปีสุดท้าย
            _DepreciationOfFirstYear = _NormalDepreciation * (Convert.ToDouble(_MonthInFirstYear) / 12);
            _DepreciationOfLastYear = _NormalDepreciation - _DepreciationOfFirstYear;
//กำหนดค่าเสื่อมราคาปีแรกให้กับฟิลด์อาร์เรย์ _TotalDepreciation ตัวแรก (ลำดับอ้างอิง 0)
            _TotalDepreciation[0] = _DepreciationOfFirstYear;
```

```
//วนลูปตั้งแต่สมาชิกอาร์เรย์ที่ 2 (ลำดับอ้างอิง 1) จนถึงสมาชิกตัวรองสุดท้าย
        for (_i = 1; _i <= _TotalDepreciation.Length - 2; _i++)
        {  //เก็บค่าเสื่อมราคาของปีที่ 2 จนถึงปีรองสุดท้าย
           _TotalDepreciation[_i] = _NormalDepreciation;
        }
//กำหนดค่าเสื่อมราคาปีสุดท้าย ให้กับฟิลด์อาร์เรย์ _TotalDepreciation ตัวสุดท้าย
        _TotalDepreciation[_EstimatedUseLife] = _DepreciationOfLastYear;

        return _TotalDepreciation;                                  //คืนค่าเป็นค่าเสื่อมราคาแต่ละปี
    }
//เมธอด CumulativeDepreciationByStraightLine() แบบ Public กรณีเต็มรอบปีบัญชี
//ทำหน้าที่คำนวณค่าเสื่อมราคาสะสม คืนค่าเป็นอาร์เรย์ double
    public double[] CumulativeDepreciationByStraightLine()
    {
        double _NormalDepreciation = 0.0;                           //ฟิลด์เก็บค่าเสื่อมราคาแต่ละปี
//สั่งให้เมธอด StraightLine() ทำงาน และเก็บผลการคำนวณที่ได้
        _NormalDepreciation = this.StraightLine();
//ฟิลด์อาร์เรย์ double ที่ชื่อว่า _TotalDepreciation ทำหน้าที่เก็บค่าเสื่อมราคาสะสมแต่ละปี
        double[] _TotalDepreciation = new double[_EstimatedUseLife];
        Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife);      //ล้างฟิลด์อาร์เรย์

        double _CDepreciation = 0.0;                                //ฟิลด์เก็บค่าเสื่อมราคาปีปัจจุบัน
//วนลูปตั้งแต่ปีแรก จนถึงปีสุดท้าย
        for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
        {
           _CDepreciation += _NormalDepreciation;                   //คำนวณค่าเสื่อมราคาสะสมปีปัจจุบัน
           _TotalDepreciation[_i] += _CDepreciation;                //เก็บค่าเสื่อมราคาสะสมปีปัจจุบันไว้
        }

        return _TotalDepreciation;                                  //คืนค่า
    }
//เมธอด CumulativeDepreciationByStraightLine() แบบ Public กรณีระหว่างปีบัญชี
//ทำหน้าที่คำนวณค่าเสื่อมราคาสะสม คืนค่าเป็นอาร์เรย์ double
//ต้องการพารามิเตอร์ 1 ตัวคือ MonthInFirstYear หมายถึง จำนวนเดือนที่ใช้สินทรัพย์ในปีแรกเท่านั้น
    public double[] CumulativeDepreciationByStraightLine(int MonthInFirstYear)
    {  //ฟิลด์อาร์เรย์ _NormalDepreciation ทำหน้าที่เก็บค่าเสื่อมราคาสะสม กรณีระหว่างรอบบัญชี
        double[] _NormalDepreciation = new double[_EstimatedUseLife + 1];
        Array.Clear(_NormalDepreciation, 0, _EstimatedUseLife+1);   //ล้างฟิลด์อาร์เรย์
//อ่านจำนวนเดือนของปีแรก เก็บไว้ที่ฟิลด์ _MonthInFirstYear
        _MonthInFirstYear = MonthInFirstYear;
//สั่งให้เมธอด StraightLine() แบบระหว่างรอบบัญชีทำงาน โดยการส่งค่าฟิลด์ _MonthInFirstYear เข้าไป
        _NormalDepreciation = this.StraightLine(_MonthInFirstYear);
//ฟิลด์อาร์เรย์ _TotalDepreciation ทำหน้าที่เก็บค่าเสื่อมราคาสะสมแต่ละปี
        double[] _TotalDepreciation = new double[_EstimatedUseLife + 1];
        Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife+1);    //ล้างค่าฟิลด์อาร์เรย์

        double _CDepreciation = 0.0;                                //ฟิลด์เก็บค่าเสื่อมราคาปีปัจจุบัน
//วนลูปตั้งแต่รอบบัญชีปีแรก จนถึงปีสุดท้าย
        for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
```

```
            {
                _CDepreciation += _NormalDepreciation[_i];                    //สะสมค่าเสื่อมราคาปีปัจจุบัน
                _TotalDepreciation[_i] += _CDepreciation;                     //เก็บค่าเสื่อมราคาสะสม
            }

            return _TotalDepreciation;                                        //คืนค่า
        }
    //เมธอด AssetValueByStraightLne() แบบ Public ทำหน้าที่หามูลค่าของสินทรัพย์ กรณีเต็มรอบปีบัญชี
    //คืนค่าเป็นอาร์เรย์ของ double
        public double[] AssetValueByStraightLine()
        {
    //สั่งให้เมธอด CumulativeDepreciationByStraightLine() ทำงาน
            double[] _CumulativeDepreciation = this.CumulativeDepreciationByStraightLine();
    //ฟิลด์อาร์เรย์ _TotalAssetValue ทำหน้าที่เก็บมูลค่าของสินทรัพย์แต่ละปี
            double[] _TotalAssetValue = new double[_EstimatedUseLife];
            Array.Clear(_TotalAssetValue, 0, _EstimatedUseLife);              //ล้างค่าฟิลด์อาร์เรย์
    //วนลูปตั้งแต่รอบบัญชีปีแรก จนถึงรอบบัญชีปีสุดท้าย
            for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
            {  //มูลค่าสินทรัพย์ปีปัจจุบัน = ต้นทุนสินค้า - ค่าเสื่อมราคาสะสมของรอบบัญชีปีปัจจุบัน
            }

            return _TotalAssetValue;                                          //คืนค่า
        }
    //เมธอด AssetValueByStraightLine() แบบ Public ทำหน้าที่หามูลค่าของสินทรัพย์ กรณีระหว่างรอบปีบัญชี
    //ต้องการพารามิเตอร์ 1 ตัวคือ MonthInFirstYear หมายถึง จำนวนเดือนที่ใช้สินทรัพย์ในปีแรก
    //คืนค่าเป็นอาร์เรย์ของ double
        public double[] AssetValueByStraightLine(int MonthInFirstYear)
        {  //กำหนดค่าให้กับฟิลด์ _MonthInFirstYear
            _MonthInFirstYear = MonthInFirstYear;
    //สั่งให้เมธอด CumulativeDepreciationByStraightLine() กรณีไม่ครบรอบปีบัญชีทำงาน
    //โดยการส่งค่าของฟิลด์ _MonthInFirstYear เข้าไป
            double[] _CumulativeDepreciation = this.CumulativeDepreciationByStraightLine(_MonthInFirstYear);
    //ฟิลด์อาร์เรย์ _TotalAssetValue ทำหน้าที่เก็บมูลค่าสินทรัพย์ของแต่ละปี
            double[] _TotalAssetValue = new double[_EstimatedUseLife + 1];
            Array.Clear(_TotalAssetValue, 0, _EstimatedUseLife+1);            //ล้างฟิลด์อาร์เรย์
    //วนลูปตั้งแต่รอบบัญชีปีแรก จนถึงรอบบัญชีปีสุดท้าย
            for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
            {  //มูลค่าสินทรัพย์ปีปัจจุบัน = ต้นทุนสินค้า - ค่าเสื่อมราคาสะสมของรอบบัญชีปีปัจจุบัน
                _TotalAssetValue[_i] = _CostOfAsset - _CumulativeDepreciation[_i];
            }

            return _TotalAssetValue;                                          //คืนค่า
        }
    }
}
```

การทำงานของคลาส Depreciation เฉพาะการคำนวณแบบเส้นตรงประกอบด้วย 3 คุณสมบัติ 6 เมธอด ก็จะกำหนดให้คุณสมบัติมีชื่อเดียวกันกับพารามิเตอร์ของคอนสตรัคเตอร์ของคลาส Depreciation เพราะว่าเมื่อนำไปใช้งานแล้วมีการเปลี่ยนแปลงค่าของต้นทุนสินทรัพย์ (CostOfAsset), ราคาซาก (SalvageValue) หรืออายุการใช้งานสินทรัพย์ (EstimatedUseLife) ผู้ใช้สามารถกำหนดค่าใหม่ ตามชื่อของคุณสมบัติได้โดยตรง

คุณสามารถคำนวณค่าเสื่อมราคาแบบเส้นตรงได้ 3 อย่างคือ

1. ค่าเสื่อมราคาแต่ละปีแบบเส้นตรง อยู่ในความรับผิดชอบของเมธอด StraightLine() มีอยู่ 2 แบบ คือ
   - **กรณีที่** 1 ถ้าเป็นการคิดค่าเสื่อมราคาแบบเส้นตรง กรณีเต็มรอบปีบัญชี ค่าเสื่อมราคาที่ได้ในแต่ละปีเท่ากัน ดังนั้น จึงคืนค่าเป็นข้อมูลชนิด double

VC# 2005

```
public double StraightLine()
```

   - **กรณีที่** 2 ถ้าเป็นการคิดค่าเสื่อมราคากับสินทรัพย์ที่อยู่ระหว่างรอบปีบัญชี ส่งผลให้ค่าเสื่อมราคาที่ได้ในปีแรก มีค่าตามจำนวนเดือนที่ใช้ไล่ไปจนครบอายุของสินทรัพยทำให้ค่าเสื่อมราคาของปีแรกและปีสุดท้ายแตกต่างจากรอบปีบัญชีอื่นๆ จึงต้องส่งค่ากลับมาเป็นอาร์เรย์ของ double

VC# 2005

```
public double[] StraightLine(int MonthInFirstYear)
```

แม้ว่าจะมีชื่อเมธอด StraightLine() เหมือนกัน แต่ Signature ต่างกัน เป็นการทำ Overload Method กล่าวคือ กรณีที่ 2 ต้องการพารามิเตอร์ชนิด int ชื่อว่า MonthInFirstYear และคืนค่าเป็นอาร์เรย์ของ double ต่างจากกรณีที่ 1 ที่ไม่ต้องการพารามิเตอร์ และคืนค่าเป็นข้อมูลชนิด double

2. ค่าเสื่อมราคาสะสมแบบเส้นตรงเป็นอีก 1 คำถามที่เราควรทราบว่า เมื่อใช้สินทรัพย์ไประยะเวลาหนึ่ง สินทรัพย์ดังกล่าวมีค่าเสื่อมราคาสะสมปัจจุบันเท่าใดแล้ว อยู่ในความรับผิดชอบของเมธอด CumulativeDepreciationByStraightLine() มี 2 กรณีเช่นกัน กล่าวคือ
   - **กรณีที่** 1 เป็นการคำนวณค่าเสื่อมราคาสะสมแบบเส้นตรง กรณีคิดค่าเสื่อมราคาเต็มรอบปีบัญชีคืนค่าเป็นอาร์เรย์ของ double

VC# 2005

```
public double[] CumulativeDepreciationByStraightLine()
```

- **กรณีที่** 2 เป็นการคำนวณค่าเสื่อมราคาสะสมเช่นกันแต่เป็นกรณีคิดค่าเสื่อมราคาระหว่างรอบปีบัญชีคืนค่าเป็นอาร์เรย์ของ double เช่นกัน

VC# 2005

```
public double[] CumulativeDepreciationByStraightLine(int MonthInFirstYear)
```

3. มูลค่าสินทรัพย์หลังจากหักค่าเสื่อมราคาแบบเส้นตรง (หรือราคาตามบัญชี) เป็นคำถามต่อมาโดยอัตโนมัติว่ามูลค่าของสินทรัพย์ปัจจุบันเหลือเท่าใด อยู่ในความรับผิดชอบของเมธอด AssetValue ByStraightLine() มีอยู่ 2 กรณีเช่นกัน โดยที่
   - **กรณีที่** 1 ราคาตามบัญชีหลังจากหักค่าเสื่อมราคาแบบเส้นตรง กรณีคิดเต็มรอบปีบัญชี

VC# 2005

```
public double[] AssetValueByStraightLine()
```

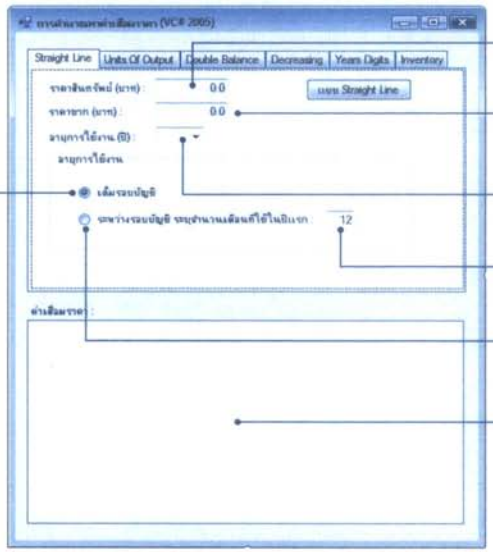
   - **กรณีที่** 2 ราคาตามบัญชีหลังจากหักค่าเสื่อมราคาแบบเส้นตรง กรณีคิดระหว่างรอบปีบัญชี

VC# 2005

```
public double[] AssetValueByStraightLine(int MonthInFirstYear)
```

ให้คุณออกแบบฟอร์ม ดังรูปที่ 20-5

รูปที่ 20-5 ฟอร์มในขณะออกแบบ



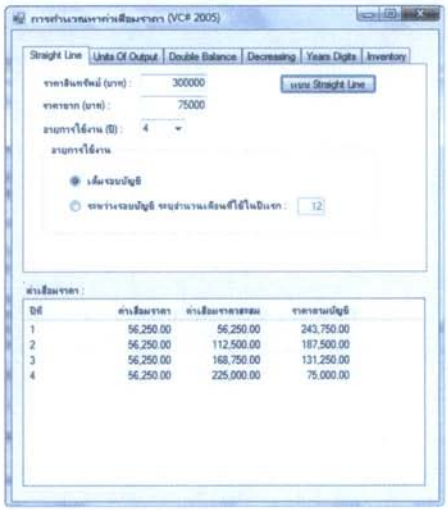
จากนั้นให้คุณเขียนโค้ดเฉพาะการคิดค่าเสื่อมราคาแบบเส้นตรง ดังนี้

โค้ด VC# 2005 ที่ 20-2 การคำนวณค่าเสื่อมราคาแบบเส้นตรง (Form1.cs)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using GAF.Biz.Accounting;                    //เนมสเปซของคลาส Depreciation

namespace CalculateDepreciation
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        int i;                               //ตัวแปรวนลูป
        string[] CurrentData;                //ตัวแปรอาร์เรย์ เก็บข้อมูลที่จะเพิ่มในคอนโทรล ListView
        ListViewItem lvi;                    //ตัวแปรออบเจ็กต์ ListViewItem
        Depreciation dp;                     //ตัวแปรออบเจ็กต์ Depreciation
//เหตุการณ์ฟอร์มโหลด
        private void Form1_Load(object sender, EventArgs e)
        {  //เพิ่มคอลัมน์เข้าไปในคอนโทรล ListView 4 คอลัมน์
            lsvDepreciation.Columns.Add("ปีที่", 70, HorizontalAlignment.Left);
            lsvDepreciation.Columns.Add("ค่าเสื่อมราคา", 100, HorizontalAlignment.Right);
            lsvDepreciation.Columns.Add("ค่าเสื่อมราคาสะสม", 100, HorizontalAlignment.Right);
            lsvDepreciation.Columns.Add("ราคาตามบัญชี", 100, HorizontalAlignment.Right);

            lsvDepreciation.View = View.Details;     //แสดงรายการแบบ Details
            lsvDepreciation.GridLines = true;        //แสดงเส้นตาราง

            for (i = 1; i <= 20; i++)                //เพิ่มหมายเลข 1-20 ลงในคอนโทรล ComboBox
            {
                cboYear.Items.Add(i);
            }
            cboYear.SelectedIndex = 0;               //ให้รายการแรกปรากฏขึ้นมา
        }
//เหตุการณ์คลิกที่ปุ่ม cmdStraight
        private void cmdStraight_Click(object sender, EventArgs e)
        {
            double TotalDepreciation1;               //ตัวแปรเก็บค่าเสื่อมราคา กรณีที่ 1
            double[] TotalDepreciation2;             //ตัวแปรเก็บค่าเสื่อมราคา กรณีที่ 2
```

```
            double[] CumulativeDepreciation1;          //ตัวแปรเก็บค่าเสื่อมราคาสะสม กรณีที่ 1
            double[] CumulativeDepreciation2;          //ตัวแปรเก็บค่าเสื่อมราคาสะสม กรณีที่ 2

            double[] CAssetValue1;                     //ตัวแปรเก็บมูลค่าสินทรัพย์ กรณีที่ 1
            double[] CAssetValue2;                     //ตัวแปรเก็บมูลค่าสินทรัพย์ กรณีที่ 2

            try                                        //ให้ทำ
            {
               lsvDepreciation.Items.Clear();          //ล้างรายการก่อน
               if (optFullYear.Checked == true)        //ถ้าตัวเลือกเต็มรอบบัญชีถูกเลือก
               {
//สร้างออบเจ็กต์ Depreciation ที่ชื่อว่า dp ส่งค่าพารามิเตอร์เข้าไป 3 ตัวคือ
//ต้นทุนสินทรัพย์, ราคาซาก, และอายุการใช้งาน
                  dp = new Depreciation(Convert.ToDouble(txtCostOfAsset.Text), Convert.ToDouble(txtSalvageValue.Text),
Convert.ToInt32(cboYear.Text));

//คำนวณค่าเสื่อมราคา โดยการเรียกเมธอด StraightLine() ทำงาน
                  TotalDepreciation1 = dp.StraightLine();
//คำนวณค่าเสื่อมราคาสะสม โดยการเรียกเมธอด CumulativeDepreciationByStraightLine() ทำงาน
                  CumulativeDepreciation1 = dp.CumulativeDepreciationByStraightLine();
//คำนวณราคาตามบัญชี โดยการเรียกเมธอด CumulativeDepreciationByStraightLine() ทำงาน
                  CAssetValue1 = dp.AssetValueByStraightLine();
//วนลูปตามอายุการใช้งานของสินทรัพย์
                  for (i = 0; i <= Convert.ToInt32(cboYear.Text) - 1; i++)
                  {  //กำหนดค่าต่างๆ ลงในตัวแปรอาร์เรย์ CurrentData
                     CurrentData = new string[] { (i + 1).ToString(), TotalDepreciation1.ToString("#,##0.00"),
CumulativeDepreciation1[i].ToString("#,##0.00"), CAssetValue1[i].ToString("#,##0.00") };

                     lvi = new ListViewItem(CurrentData);          //เพิ่มรายการลงในออบเจ็กต์ ListViewItem
                     lsvDepreciation.Items.Add(lvi);               //เพิ่มรายการลงในคอนโทรล ListView
                  }
               }
               else  //แต่ถ้าตัวเลือกระหว่างรอบบัญชี ระบุจำนวนเดือนที่ใช้ในปีแรก : ถูกเลือก
               {
//สร้างออบเจ็กต์ Depreciation ที่ชื่อว่า dp ส่งค่าพารามิเตอร์เข้าไป 3 ตัวคือ
//ต้นทุนสินทรัพย์, ราคาซาก, และอายุการใช้งาน
                  dp = new Depreciation(Convert.ToDouble(txtCostOfAsset.Text), Convert.ToDouble(txtSalvageValue.Text),
Convert.ToInt32(cboYear.Text));

//คำนวณค่าเสื่อมราคา โดยการเรียกเมธอด StraightLine() ทำงาน โดยส่งจำนวนเดือนของปีแรกเข้าไปด้วย
                  TotalDepreciation2 = dp.StraightLine(Convert.ToInt32(txtMonthInFirstYear.Text));
//คำนวณค่าเสื่อมราคาสะสม โดยส่งจำนวนเดือนของปีแรกเข้าไปด้วย
                  CumulativeDepreciation2 = dp.CumulativeDepreciationByStraightLine(Convert.ToInt32(txtMonthInFirstYear.Text));
//คำนวณราคาตามบัญชี โดยส่งจำนวนเดือนของปีแรกเข้าไปด้วย
                  CAssetValue2 = dp.AssetValueByStraightLine(Convert.ToInt32(txtMonthInFirstYear.Text));

//วนลูปตามอายุการใช้งานของสินทรัพย์
                  for (i = 0; i <= TotalDepreciation2.Length - 1; i++)
                  {  //กำหนดค่าต่างๆ ลงในตัวแปรอาร์เรย์ CurrentData
```

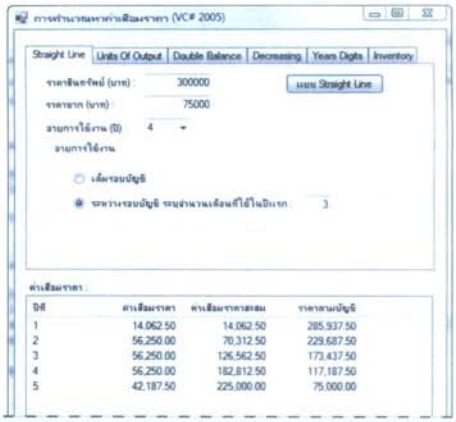
```
                CurrentData = new string[] { (i + 1).ToString(), TotalDepreciation2[i].ToString("#,##0.00"),
CumulativeDepreciation2[i].ToString("#,##0.00"), CAssetValue2[i].ToString("#,##0.00") };

                lvi = new ListViewItem(CurrentData);            //เพิ่มรายการลงในออบเจ็กต์ ListViewItem
                lsvDepreciation.Items.Add(lvi);                 //เพิ่มรายการลงในคอนโทรล ListView
            }
        }
    }
    catch (Exception ex)                                        //กรณีเกิดข้อผิดพลาด
    {
        MessageBox.Show(ex.Message);                            //แสดงข้อความผิดพลาด
    }
}

//เหตุการณ์คลิกที่ตัวเลือกเต็มรอบบัญชี
    private void optFullYear_CheckedChanged(object sender, EventArgs e)
    {
        txtMonthInFirstYear.Enabled = false;
        txtMonthInFirstYear.Text = "12";
    }

//เหตุการณ์คลิกที่ตัวเลือกระหว่างรอบบัญชี ระบุจำนวนเดือนที่ใช้ในปีแรก :
    private void optMYear_CheckedChanged(object sender, EventArgs e)
    {
        txtMonthInFirstYear.Enabled = true;
        txtMonthInFirstYear.Focus();
    }
```



รูปที่ 20-6 ผลการคำนวณค่าเสื่อมราคาแบบเส้นตรง กรณีครบรอบปีบัญชี

จากรูปที่ 20-6 การคิดค่าเสื่อมราคาแบบเส้นตรง กรณีครบรอบปีบัญชีจะได้ค่าเสื่อมราคาเท่ากันทุกปี จนครบอายุการใช้งานของสินทรัพย์

รูปที่ 20-7
ผลการคำนวณค่าเสื่อมราคาแบบเส้นตรง กรณีระหว่างรอบปีบัญชี



จากรูปที่ 20-7 ถึงแม้ว่าอายุของสินทรัพย์ 4 ปี แต่ค่าเสื่อมราคาจะคำนวณยาวมาถึงรอบปีบัญชีที่ 5 อีก 9 เดือน เพื่อให้ครบอายุการใช้งานของสินทรัพย์นั่นเอง

NOTE

การใช้งานคลาส Depreciation อีกวิธีหนึ่งก็คือ การใช้งานผ่านทาง Default Constructor เป็นการสร้างออบเจ็กต์ Depreciation ที่ชื่อว่า dp ขึ้นมาก่อน ด้วยคำสั่ง new (New ใน VB 2005) เห็นได้ว่าไม่มีการส่งค่าพารามิเตอร์ใดๆ เข้าไป

Default Constructor ของ VC# 2005

```
public Depreciation()
{

}
```

Default Constructor ของ VB 2005

```
Public Sub New()

End Sub
```

จากนั้นจึงค่อยกำหนดค่าต่างๆ ให้กับคุณสมบัติ CostOfAsset, SalvageValue และคุณสมบัติ EstimatedUseLife ครบตามสูตรของการคำนวณค่าเสื่อมราคาแบบเส้นตรง คุณสามารถทำแบบนี้กับการคำนวณค่าเสื่อมราคาที่เหลืออยู่อีก 5 วิธีได้เช่นกัน

VC# 2005

```
Depreciation dp = new Depreciation();
dp.CostOfAsset=Convert.ToDouble(txtCostOfAsset.Text);
dp.SalvageValue=Convert.ToDouble(txtSalvageValue.Text);
dp.EstimatedUseLife = Convert.ToInt32(cboYear.Text);

TotalDepreciation1 = dp.StraightLine();
CumulativeDepreciation1 = dp.CumulativeDepreciationByStraightLine();
CAssetValue1 = dp.AssetValueByStraightLine();
```

การคิดค่าเสื่อมราคาแบบคิดตามหน่วยที่ผลิต (Units Of Output)

หลักการของวิธีคิดค่าเสื่อมราคาแบบนี้ก็คือ เมื่อใช้สินทรัพย์เพื่อผลิตสินค้ามากเท่าใด ก็จะส่งผลให้สินทรัพย์ดังกล่าวเสื่อมลงตามการใช้งานเท่านั้น สูตรการคิดมี 2 ขั้นตอนคือ

$$\text{ค่าเสื่อมราคาต่อหน่วยผลิต} = \frac{(C-S)}{EP}$$

โดยที่ C = ต้นทุนสินทรัพย์, S = ราคาซากเมื่อขายสินทรัพย์, EP = จำนวนสินค้าที่สามารถผลิตได้ตลอดอายุสินทรัพย์

เมื่อได้ค่าเสื่อมราคาต่อการผลิตสินค้า 1 หน่วยแล้ว ก็จะนำไปคิดค่าเสื่อมราคาแยกตามรายปีจนครบอายุของสินทรัพย์ เช่น ซื้อเครื่องจักรผลิตสินค้าราคา 220,000 บาท อายุการใช้งาน 5 ปี ราคาซากที่คาดว่าจะขายได้คือ 25,000 บาท ในช่วง 5 ปีของการใช้เครื่องจักร คาดว่าสามารถผลิตสินค้าได้ดังนี้

ปีที่	จำนวนสินค้าที่ผลิตได้ (หน่วย)
1	35,000
2	25,000
3	16,000
4	10,000
5	7,000

ให้คิดค่าเสื่อมราคาต่อหน่วยก่อนโดยการแทนค่าในสูตร ดังนี้

$$\frac{(220,000-25,000)}{(35,000+25,000+16,000+10,000+7,000)}$$

ค่าเสื่อมราคาที่ได้คือ 2.09 บาท หมายความว่า เมื่อใช้เครื่องจักรนี้ผลิตสินค้า 1 หน่วย ทำให้เครื่องจักรเสื่อมค่าลงไป 2.09 บาทนั่นเอง รายละเอียดของคลาส Depreciation เฉพาะการคำนวณแบบคิดตามหน่วยที่ผลิต ดังนี้

โค้ด VC# 2005 ที่ 20-3 การคำนวณค่าเสื่อมราคาแบบคิดตามหน่วยที่ผลิต (GAF.Biz.Accounting.Depreciation.UnitsOfOutput.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GAF.Biz.Accounting                    //เนมสเปซของคลาส Depreciation
{
    public partial class Depreciation           //กำหนดให้เป็นแบบ Partial Class
    {
        private double[] _TotalLifeTime;        //ฟิลด์เก็บจำนวนสินค้าที่ผลิตได้
        private int _TotalYear = 1;             //อายุการใช้งาน
        private double _DepreciationPerUnit = 0.0;  //ค่าเสื่อมราคาต่อหน่วย
```

```
        public double[] TotalLifeTime                    //คุณสมบัติ TotalLifeTime คืนค่าเป็นอาร์เรย์ของ double
        {
            get{return _TotalLifeTime;}                  //ให้อ่านค่าได้
            set{_TotalLifeTime = value;}                 //ให้กำหนดค่าได้
        }

        public int TotalYear                             //คุณสมบัติ TotalYear คืนค่าเป็นข้อมูลชนิด int
        {
            get{return _TotalYear;}                      //ให้อ่านค่าได้
            set{_TotalYear = value;}                     //ให้กำหนดค่าได้
        }

//คอนสตรัคเตอร์ตัวที่ 3 ต้องการพารามิเตอร์ 4 ตัวคือ
//พารามิเตอร์ CostOfAsset หมายถึง ต้นทุนสินทรัพย์
//พารามิเตอร์ SalvageValue หมายถึง ราคาซาก
//พารามิเตอร์ TotalLifeTime หมายถึง จำนวนสินค้าที่สามารถผลิตได้ตลอดอายุสินทรัพย์
//พารามิเตอร์ TotalYear หมายถึง อายุสินทรัพย์

public Depreciation(double CostOfAsset, double SalvageValue, double[] TotalLifeTime, int TotalYear)
        {
            if (TotalYear == 0)
            {
                throw (new Exception("กรุณาระบุอายุการใช้งาน (TotalYear) ที่มีค่ามากกว่า 0"));
            }

            _CostOfAsset = CostOfAsset;                  //กำหนดต้นทุนสินทรัพย์ให้กับฟิลด์ _CostOfAsset
            _SalvageValue = SalvageValue;                //กำหนดราคาซากให้กับฟิลด์ _SalvageValue
            _TotalLifeTime = TotalLifeTime;              //กำหนดจำนวนสินค้าที่ผลิตได้ให้กับฟิลด์ _TotalLifeTime
            _TotalYear = TotalYear;                      //อายุสินทรัพย์

            double _SummeryLifeTime = 0.0;               //ฟิลด์เก็บจำนวนสินค้าทั้งหมดที่ผลิตได้
            for (_i = 0; _i <= _TotalLifeTime.Length - 1; _i++)
            {  //คำนวณจำนวนสินค้าที่สามารถผลิตได้ทั้งหมด ตลอดอายุสินทรัพย์
                _SummeryLifeTime += _TotalLifeTime[_i];
            }
//คำนวณค่าเสื่อมราคาต่อหน่วยตามสูตร เก็บไว้ในฟิลด์ _DepreciationPerUnit
            _DepreciationPerUnit = (_CostOfAsset - _SalvageValue) / _SummeryLifeTime;
        }

//เมธอด UnitsOfOutput() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบคิดตามหน่วยผลิต คืนค่าเป็นอาร์เรย์ของ double
        public double[] UnitsOfOutput()
        {
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
            double[] _AllDepreciation = new double[_TotalYear ];
            Array.Clear(_AllDepreciation, 0, _AllDepreciation.Length); //ล้างฟิลด์อาร์เรย์

            for (_i = 0; _i <= _TotalYear - 1; _i++)                   //วนลูปจนหมดอายุสินทรัพย์
            {  //คำนวณค่าเสื่อมราคาแต่ละปีตามสูตร เก็บไว้ในฟิลด์อาร์เรย์ _AllDepreciation
```

```
            _AllDepreciation[_i] += _TotalLifeTime[_i] * _DepreciationPerUnit;
        }

        return _AllDepreciation;                                    //คืนค่า
    }

//เมธอด CumulativeDepreciationByUnitsOfOutput() แบบ Public
//ทำหน้าที่คำนวณค่าเสื่อมราคาสะสมแบบคิดตามหน่วยผลิต คืนค่าเป็นอาร์เรย์ของ double
    public double[] CumulativeDepreciationByUnitsOfOutput()
    {
        double[] _NormalDepreciation;                               //ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
//สั่งให้เมธอด UnitsOfOutput() ทำงาน เพื่อคำนวณค่าเสื่อมราคาแต่ละปีก่อน
        _NormalDepreciation = this.UnitsOfOutput();
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาสะสม
        double[] _TotalDepreciation = new double[_TotalYear];
        Array.Clear(_TotalDepreciation, 0, _TotalYear);             //ล้างค่าฟิลด์อาร์เรย์

        double _CDepreciation = 0.0;                                //ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาสะสมปีปัจจุบัน
        for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
        {
            _CDepreciation += _NormalDepreciation[_i];              //คำนวณค่าเสื่อมราคาสะสม
            _TotalDepreciation[_i] += _CDepreciation;               //เก็บไว้ในฟิลด์ _TotalDepreciation
        }

        return _TotalDepreciation;                                  //คืนค่า
    }

//เมธอด AssetValueByUnitsOfOutput() ทำหน้าที่คำนวณมูลค่าสินทรัพย์หลังหักค่าเสื่อมราคาแบบคิดตามหน่วยผลิต
//คืนค่าเป็นอาร์เรย์ของ double
    public double[] AssetValueByUnitsOfOutput()
    {
//สั่งให้เมธอด CumulativeDepreciationByUnitsOfOutput() ทำงาน เพื่อคำนวณค่าเสื่อมราคาสะสมก่อน
//เก็บไว้ที่ฟิลด์อาร์เรย์ _CumulativeDepreciation
        double[] _CumulativeDepreciation = this.CumulativeDepreciationByUnitsOfOutput();
//ฟิลด์อาร์เรย์เก็บมูลค่าสินทรัพย์หลังหักค่าเสื่อมราคาแบบคิดตามหน่วยผลิตในแต่ละปี
        double[] _TotalAssetValue = new double[_TotalYear];
        Array.Clear(_TotalAssetValue, 0, _TotalYear);               //ล้างค่าฟิลด์อาร์เรย์

        for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
        {  //มูลค่าสินทรัพย์ปีปัจจุบัน = ต้นทุนสินทรัพย์ - ค่าเสื่อมราคาสะสมปีปัจจุบัน
            _TotalAssetValue[_i] = _CostOfAsset - _CumulativeDepreciation[_i];
        }

        return _TotalAssetValue;                                    //คืนค่า
    }
  }
}
```

ในส่วนของการคำนวณค่าเสื่อมราคาแบบคิดตามหน่วยผลิต เนื่องจากมีสูตรการคิดไม่เหมือนกับแบบเส้นตรง ดังนั้น จึงต้องสร้างคอนสตรัคเตอร์ขึ้นมาอีก 1 ชุด เพื่อรับค่าต่างๆ ให้ครบถ้วนตามสูตร

VC# 2005

```
public Depreciation(double CostOfAsset, double SalvageValue, double[] TotalLifeTime, int TotalYear)
```

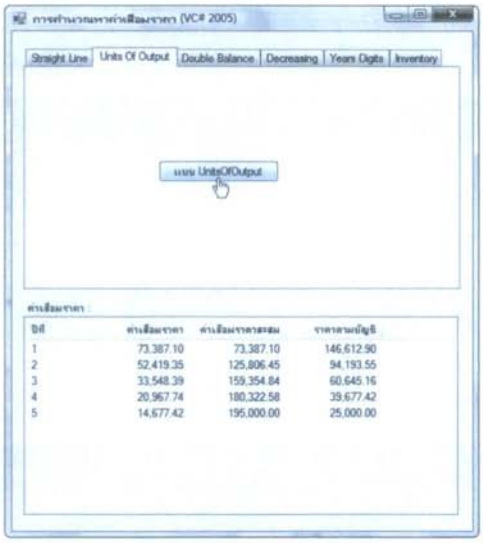
ที่น่าสนใจอยู่ตรงพารามิเตอร์ _TotalLifeTime เป็นข้อมูลชนิดอาร์เรย์ของ double ซึ่งทำหน้าที่เก็บจำนวนสินค้าที่ผลิตได้ตลอดอายุสินทรัพย์ ผู้เขียนคิดว่าจำนวนสินค้าที่ผลิตได้ย่อมที่จะมีหลายปี ดังนั้น จึงกำหนดให้เป็นตัวแปรอาร์เรย์ เพื่อให้สมาชิกแต่ละตัวในอาร์เรย์ดังกล่าวรับจำนวนสินค้าที่ผลิตได้ในแต่ละปีนั่นเอง

สำหรับวิธีการเรียกใช้การคำนวณค่าเสื่อมราคาแบบคิดตามหน่วยผลิต มีโค้ดดังนี้

โค้ด VC# 2005 ที่ 20-3 การคำนวณค่าเสื่อมราคาแบบคิดตามหน่วยผลิต (Form1.cs)

```
//เหตุการณ์คลิกที่ปุ่ม cmdUnitsOfOutput
        private void cmdUnitsOfOutput_Click(object sender, EventArgs e)
        {
            double Cost = 220000;                       //ต้นทุนสินทรัพย์
            double EstValue = 25000;                    //ราคาซาก
//ตัวแปรอาร์เรย์เก็บจำนวนสินค้าที่ผลิตได้ ตลอดอายุสินทรัพย์ (5 ปี)
            double[] LifeTime = new double[] { 35000, 25000, 16000, 10000, 7000 };
            int TotalYear = 5;                          //อายุสินทรัพย์
            double[] TotalDepreciation;                 //ตัวแปรอาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
            double[] CumulativeDepreciation;            //ตัวแปรอาร์เรย์เก็บค่าเสื่อมราคาสะสม
            double[] CurrentAssetValue;                 //ตัวแปรอาร์เรย์เก็บมูลค่าสินทรัพย์แต่ละปี
//สร้างออบเจ็กต์ Depreciation ที่ชื่อว่า dp
            dp = new Depreciation(Cost, EstValue, LifeTime, TotalYear);
//คำนวณค่าเสื่อมราคาแบบคิดตามหน่วยผลิต โดยการเรียกเมธอด UnitsOfOutput() ทำงาน
            TotalDepreciation = dp.UnitsOfOutput();
//คำนวณค่าเสื่อมราคาสะสม โดยการเรียกเมธอด CumulativeDepreciationByUnitsOfOutput() ทำงาน
            CumulativeDepreciation = dp.CumulativeDepreciationByUnitsOfOutput();
//คำนวณมูลค่าสินทรัพย์ โดยการเรียกเมธอด AssetValueByUnitsOfOutput() ทำงาน
            CurrentAssetValue = dp.AssetValueByUnitsOfOutput();

            lsvDepreciation.Items.Clear();
            for (i = 0; i <= TotalYear - 1; i++)
            {
                CurrentData = new string[] { (i + 1).ToString(), TotalDepreciation[i].ToString("#,##0.00"), CumulativeDepreciation[i].
ToString("#,##0.00"), CurrentAssetValue[i].ToString("#,##0.00") };

                lvi = new ListViewItem(CurrentData);
                lsvDepreciation.Items.Add(lvi);          //เพิ่มรายการในคอนโทรล ListView
            }
        }
```

รูปที่ 20-8
ผลการคำนวณค่าเสื่อมราคาแบบคิดตามหน่วยผลิต



การคำนวณค่าเสื่อมราคาแบบ 2 เท่าของเส้นตรง โดยคิดจากยอดที่ลดลง (Double Declining Balance)

หลักการของวิธีคิดค่าเสื่อมราคาแบบนี้ก็คือ มูลค่าสินทรัพย์ที่จะนำมาคำนวณค่าเสื่อมราคาควรเป็นมูลค่าที่หลังจากหักค่าเสื่อมตามระยะเวลาที่ใช้สินทรัพย์นั้นไปแล้ว มี 2 ขั้นตอนคือ

$$\text{อัตราค่าเสื่อมราคา} = \frac{(100\%)}{Y} \times 2$$

โดยที่ Y หมายถึง อายุการใช้งานสินทรัพย์

เมื่อได้อัตราค่าเสื่อมราคามาแล้ว ก็จะนำไปคิดค่าเสื่อมราคากับต้นทุนสินทรัพย์ โดยที่ในปีถัดไปต้นทุนสินค้าจะลดลงตามค่าของค่าเสื่อมราคาในปีก่อนหน้าด้วย เช่น ซื้อเครื่องจักรราคา 350,000 บาท อายุการใช้งาน 4 ปี อัตราค่าเสื่อมราคาที่ได้คือ

$$\frac{(100\%)}{4} \times 2$$

อัตราค่าเสื่อมราคาที่ได้คือ 50% หมายความว่า เมื่อใช้เครื่องจักรสิ้นสุดใน 1 ปีแรกมูลค่าของเครื่องจักรลดลงไป 50% ส่งผลให้ในต้นปีที่ 2 ต้นทุนเครื่องจักรเหลือ 175,000 บาท (ลดลง 50% ของ 350,000) เมื่อใช้เครื่องจักรสิ้นสุดในปีที่ 2 มูลค่าเครื่องจักรเหลือ 87,500 บาท ลดลงอีก 50% (50% ของ 175,000) ไล่ไปเรื่อยๆ จนครบอายุการใช้งานของเครื่องจักร รายละเอียดของคลาส Depreciation เฉพาะการคำนวณแบบ 2 เท่าของเส้นตรง โดยคิดจากยอดที่ลดลง ดังนี้

โค้ด VC# 2005 ที่ 20-4 การคำนวณค่าเสื่อมราคาแบบ 2 เท่าของเส้นตรง โดยคิดจากยอดที่ลดลง (GAF.Biz.Accounting.Depreciation.DoubleBalance.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GAF.Biz.Accounting                                    //เนมสเปซของคลาส Depreciation
{
    public partial class Depreciation                           //กำหนดให้เป็นแบบ Partial Class
    {
        private double _DBRate = 0.0;                           //ฟิลด์เก็บอัตราค่าเสื่อมราคา
//คอนสตรัคเตอร์ตัวที่ 4 ต้องการพารามิเตอร์ 2 ตัวคือ
//พารามิเตอร์ CostOfAsset หมายถึง ต้นทุนสินทรัพย์
//พารามิเตอร์ EstimatedUseLife หมายถึง อายุการใช้งานสินทรัพย์
        public Depreciation(double CostOfAsset, int EstimatedUseLife)
        {
            if (EstimatedUseLife == 0)
            {
                throw (new Exception("กรุณาระบุอายุการใช้งาน (UseLifeTime) ที่มีค่ามากกว่า 0"));
            }

            _CostOfAsset = CostOfAsset;                         //กำหนดต้นทุนสินค้าให้กับฟิลด์ _CostOfAsset
            _EstimatedUseLife = EstimatedUseLife;               //กำหนดอายุใช้งานสินทรัพย์ให้กับฟิลด์ _EstimatedUseLife
            _DBRate = (100.0 / _EstimatedUseLife) * 2;          //คำนวณอัตราค่าเสื่อมราคาเก็บไว้ที่ฟิลด์ _DBRate
        }

//เมธอด DoubleBalance() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบ 2 เท่าลดลง
//คืนค่าเป็นอาร์เรย์ของ double
        public double[] DoubleBalance()
        {   //ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
            double[] _TotalDepreciation = new double[_EstimatedUseLife];
            Array.Clear(_TotalDepreciation, 0, _TotalDepreciation.Length);

            double _CAssetValue;                                //ฟิลด์เก็บต้นทุนสินทรัพย์ปัจจุบัน
            _CAssetValue = _CostOfAsset;                        //อ่านค่าออกมาจากฟิลด์ _CostOfAsset
            for (_i = 0; _i <= _EstimatedUseLife - 1; _i++)
            {   //คำนวณค่าเสื่อมราคาของปีปัจจุบันตามสูตร
                _TotalDepreciation[_i] = _CAssetValue * (_DBRate / 100.0);
//ลดมูลค่าสินทรัพย์ลง ตามค่าเสื่อมราคาของปีปัจจุบัน
                _CAssetValue -= _TotalDepreciation[_i];
            }

            return _TotalDepreciation;                          //คืนค่า
        }

//เมธอด CumulativeDepreciationByDoubleBalance() ทำหน้าที่คำนวณค่าเสื่อมราคาสะสมแบบ 2 เท่าอัตราลดลง
//คืนค่าเป็นอาร์เรย์ของ double
```

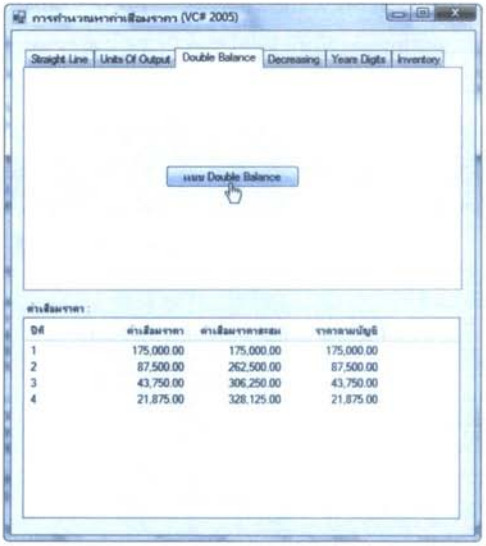
```
        public double[] CumulativeDepreciationByDoubleBalance()
        {
            double[] _NormalDepreciation;                          //ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
//สั่งให้เมธอด DoubleBalance() ทำงาน เก็บค่าเสื่อมราคาที่คำนวณได้ไว้ในฟิลด์ _NormalDepreciation
            _NormalDepreciation = this.DoubleBalance();
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาสะสม
            double[] _TotalDepreciation = new double[_EstimatedUseLife];
            Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife);     //ล้างค่าฟิลด์อาร์เรย์

            double _CDepreciation = 0.0;                           //ฟิลด์เก็บค่าเสื่อมราคาสะสมของปีปัจจุบัน
            for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
            {
              _CDepreciation += _NormalDepreciation[_i];
              _TotalDepreciation[_i] += _CDepreciation;            //เก็บค่าเสื่อมราคาสะสม
            }

            return _TotalDepreciation;                             //คืนค่า
        }

//เมธอด AssetValueByDoubleBalance() แบบ Public ทำหน้าที่คำนวณมูลค่าสินทรัพย์ปัจจุบัน
//คืนค่าเป็นอาร์เรย์ของ double
        public double[] AssetValueByDoubleBalance()
        {
//สั่งให้เมธอด CumulativeDepreciationByDoubleBalance() ทำงาน
//เพื่อเก็บค่าเสื่อมราคาสะสมไว้ในฟิลด์อาร์เรย์ _CumulativeDepreciation ก่อน
            double[] _CumulativeDepreciation = this.CumulativeDepreciationByDoubleBalance();
//ฟิลด์อาร์เรย์เก็บมูลค่าสินทรัพย์ของปีปัจจุบัน
            double[] _TotalAssetValue = new double[_EstimatedUseLife];
            Array.Clear(_TotalAssetValue, 0, _EstimatedUseLife);     //ล้างค่าฟิลด์อาร์เรย์

            for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
            {  //มูลค่าสินทรัพย์ปีปัจจุบัน = ต้นทุนสินทรัพย์ - ค่าเสื่อมราคาสะสมปีปัจจุบัน
              _TotalAssetValue[_i] = _CostOfAsset - _CumulativeDepreciation[_i];
            }

            return _TotalAssetValue;                               //คืนค่า
        }
    }
}
```

ส่วนที่น่าสนใจของการคิดค่าเสื่อมราคาแบบ 2 เท่าของเส้นตรงในอัตราลดลงอยู่ที่คอนสตรัคเตอร์ ผู้เขียนกำหนดให้ฟิลด์ _DBRate ทำหน้าที่เก็บอัตราค่าเสื่อมราคาเป็นข้อมูลชนิด double หมายถึงต้องการเก็บความละเอียดในระดับทศนิยม

จากสูตรของวิธีนี้คุณต้องใช้ค่า 100% มาเป็นตัวตั้ง แต่ภาษา VC# เป็นภาษาที่รักษาชนิดของข้อมูล ถ้าคุณใช้ตัวเลข 100 ภาษา VC# จะถือว่าชนิดของข้อมูลเริ่มต้นคือ เลขจำนวนเต็ม ดังนั้น ผู้เขียนจึงใช้ตัวเลข 100.0 แทนตัวเลข 100

VC# 2005

```
public partial class Depreciation
{
    private double _DBRate = 0.0;

    public Depreciation(double CostOfAsset, int EstimatedUseLife)
    {
        if (EstimatedUseLife == 0)
        {
            throw (new Exception("กรุณาระบุอายุการใช้งาน (UseLifeTime) ที่มีค่ามากกว่า 0"))
        }

        _CostOfAsset = CostOfAsset;
        _EstimatedUseLife = EstimatedUseLife;
        _DBRate = (100.0 / _EstimatedUseLife) * 2;
    }
```



รูปที่ 20-9 ผลการคิดค่าเสื่อมราคาแบบ 2 เท่าของเส้นตรง

การคิดค่าเสื่อมราคาแบบคิดอัตราลดลง (Decreasing Charge)

หลักการของวิธีนี้คิดแบบง่ายๆ ก็คือ สินทรัพย์ที่คุณนำมาใช้ใหม่ๆ ย่อมที่จะมีประสิทธิภาพมากที่สุด ดังนั้น ค่าเสื่อมราคาย่อมมีมากที่สุดเช่นกัน โดยที่ค่าเสื่อมราคาดังกล่าวลดลงเรื่อยๆ ตามอายุการใช้งานของสินทรัพย์นั่นเอง สูตรในการคำนวณคือ

$$\text{อัตราค่าเสื่อมราคา} = 1-\sqrt[n]{\frac{S}{C}} \times 100$$

โดยที่ n หมายถึง อายุสินทรัพย์, S หมายถึง ราคาซาก, C หมายถึง ต้นทุนสินทรัพย์ เช่น

ซื้อเครื่องจักรราคา 90,000 บาท อายุใช้งาน 4 ปี ราคาซาก 12,000 บาท แทนค่าในสูตรเพื่อหาอัตราค่าเสื่อมราคาก่อน ดังนี้

$$\left(1-\sqrt[4]{\frac{12,000}{90,000}}\right) \times 100$$

ค่าที่ได้คือ 39% หมายความว่าอัตราค่าเสื่อมราคาที่คิดในแต่ละปีเท่ากับ 39% โดยที่ต้นทุนสินทรัพย์ในปีแรกจะมีค่าสูงที่สุด เพราะว่าเป็นราคาที่ซื้อมานั่นเองแต่เมื่อเริ่มคิดค่าเสื่อมราคาของปีที่ 2 ต้นทุนสินทรัพย์จะถูกหักค่าเสื่อมราคาของปีแรกออกก่อนส่งผลให้ค่าเสื่อมราคาจะอยู่ในอัตราลดลงเรื่อยๆ ไปจนถึงปีสุดท้ายของอายุสินทรัพย์ รายละเอียดของคลาส Depreciation เฉพาะการคำนวณแบบคิดอัตราลดลง ดังนี้

โค้ด VC# 2005 ที่ 20-5 การคำนวณค่าเสื่อมราคาแบบคิดอัตราลดลง (GAF.Biz.Accounting.Depreciation.Decreasing.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GAF.Biz.Accounting                    //เนมสเปซของคลาส Depreciaion
{
    public partial class Depreciation           //กำหนดให้เป็น Partial Class
    {
//เมธอด Decreasing() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบคิดอัตราลดลง
//คืนค่าเป็นอาร์เรย์ของ double
        public double[] Decreasing()
        {
            double _DecreasingRate = 0.0;       //ฟิลด์เก็บอัตราค่าเสื่อมราคา
//คำนวณอัตราค่าเสื่อมราคาตามสูตรก่อน เก็บไว้ที่ฟิลด์ _DecreasingRate
            _DecreasingRate = 100.0 * (1.0 - Math.Pow((_SalvageValue / _CostOfAsset), (1.0 / _EstimatedUseLife)));
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
            double[] _AllDepreciation = new double[_EstimatedUseLife];
            Array.Clear(_AllDepreciation, 0, _AllDepreciation.Length); //ล้างค่าฟิลด์อาร์เรย์

            double _CCost = 0.0;                //ฟิลด์เก็บต้นทุนสินค้าปัจจุบัน
            _CCost = _CostOfAsset;              //อ่านต้นทุนสินค้าจากฟิลด์ _CostOfAsset
            for (_i = 0; _i <= _EstimatedUseLife - 1; _i++)
```

```
        {  //คำนวณค่าเสื่อมราคาปีปัจจุบันตามสูตร
           _AllDepreciation[_i] += _CCost * (Math.Floor(_DecreasingRate) / 100.0);
           _CCost -= _AllDepreciation[_i];                    //ลดต้นทุนสินค้าตามค่าเสื่อมราคาปีปัจจุบัน
        }

        return _AllDepreciation;                              //คืนค่า
    }

//เมธอด CumulativeDepreciationByDecreasing() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาสะสม
//คืนค่าเป็นอาร์เรย์ของ double
    public double[] CumulativeDepreciationByDecreasing()
    {
        double[] _NormalDepreciation;                         //ฟิลด์เก็บค่าเสื่อมราคาแต่ละปี
//สั่งให้เมธอด Decreasing() ทำงานเพื่อคำนวณค่าเสื่อมราคาแบบคิดอัตราลดลงก่อน
        _NormalDepreciation = this.Decreasing();
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาสะสมแต่ละปี
        double[] _TotalDepreciation = new double[_EstimatedUseLife];
        Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife);    //ล้างฟิลด์อาร์เรย์

        double _CDepreciation = 0.0;                          //ฟิลด์เก็บค่าเสื่อมราคาปีปัจจุบัน
        for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
        {  //อ่านค่าเสื่อมราคาปีปัจจุบัน
           _CDepreciation += _NormalDepreciation[_i];
           _TotalDepreciation[_i] += _CDepreciation;          //คำนวณค่าเสื่อมราคาสะสม
        }

        return _TotalDepreciation;                            //คืนค่า
    }

//เมธอด AssetValueByDecreasing() แบบ Public ทำหน้าที่คำนวณมูลค่าสินทรัพย์ปัจจุบัน
//คืนค่าเป็นอาร์เรย์ของ double
    public double[] AssetValueByDecreasing()
    {
//สั่งให้เมธอด CumulativeDepreciationByDecreasing() ทำงาน เพื่อคำนวณค่าเสื่อมราคาสะสมก่อน
        double[] _CumulativeDepreciation = this.CumulativeDepreciationByDecreasing();
//ฟิลด์อาร์เรย์เก็บมูลค่าสินทรัพย์แต่ละปี
        double[] _TotalAssetValue = new double[_EstimatedUseLife];
        Array.Clear(_TotalAssetValue, 0, _EstimatedUseLife);  //ล้างค่าฟิลด์อาร์เรย์

        for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
        {  //มูลค่าสินทรัพย์ปัจจุบัน = ต้นทุนสินค้า - ค่าเสื่อมราคาสะสมปีปัจจุบัน
           _TotalAssetValue[_i] = _CostOfAsset - _CumulativeDepreciation[_i];
        }

        return _TotalAssetValue;                              //คืนค่า
    }
  }
}
```

สิ่งที่น่าสนใจของการคำนวณค่าเสื่อมราคาแบบคิดอัตราลดลงมีอยู่ 2 อย่างคือ

1. จากสูตรคำนวณอัตราค่าเสื่อมราคาแบบคิดอัตราลดลง (1- $\sqrt{(n\&S/C)}$)x 100 เห็นได้ว่าเราสามารถใช้ฟิลด์ที่เกิดจากคอนสตรัคเตอร์ของวิธีคำนวณแบบเส้นตรงได้เลย (ไฟล์ GAF.Biz.Accounting.Depreciation.StraightLine.cs) เพราะว่าการทำ Partial Class ถือว่าเป็นคลาสเดียวกันนั่นเอง โดยที่

VC# 2005

```
private double _CostOfAsset = 0.0;
private double _SalvageValue = 0.0;
private int _EstimatedUseLife = 1;
```

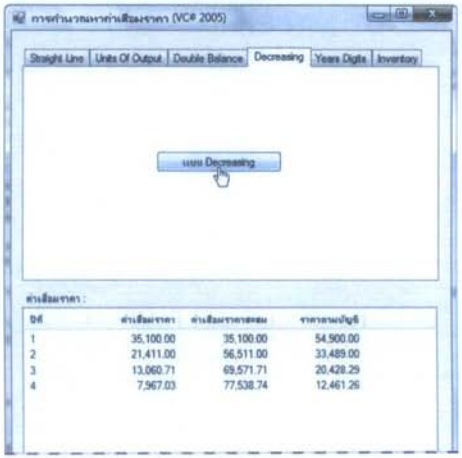
- ฟิลด์ _CostOfAsset แทนค่าต้นทุนสินทรัพย์ (C)
- ฟิลด์ _SalvageValue แทนราคาซาก (S)
- ฟิลด์ _EstimatedUseLife แทนอายุใช้งานสินทรัพย์ (n)

2. เนื่องจากการหาค่าราก ($\sqrt{x}$) อยู่ในความรับผิดชอบของเมธอด Sqrt() ของคลาส Math สามารถหาได้แต่เพียงค่ารากที่ 2 เท่านั้น เราจะใช้ความสามารถเทียบเท่าของหลักคณิตศาสตร์มาช่วยหาค่ารากที่ n ทำหน้าที่แทนเมธอด Sqrt() วิธีคิดคือ
   - การหารากที่ 2 ของ 5 หมายถึง 5 ยกกำลัง $^1/_2$
   - การหารากที่ 3 ของ 7 หมายถึง 7 ยกกำลัง $^1/_3$

จากวิธีคิดดังกล่าว จึงเป็นที่มาของการใช้เมธอด Pow() ของคลาส Math ซึ่งทำหน้าที่หาค่ายกกำลังนั่นเอง

VC# 2005

```
_DecreasingRate = 100.0 * (1.0 - Math.Pow((_SalvageValue / _CostOfAsset), (1.0 / _EstimatedUseLife)));
```



รูปที่ 20-10 ผลการคิดค่าเสื่อมราคาแบบคิดอัตราลดลง

การคำนวณค่าเสื่อมราคาแบบคำนวณสินทรัพย์คงเหลือ (Inventory)

เป็นวิธีที่ใช้กับสินทรัพย์หน่วยเล็กๆ มีราคาไม่มาก คิดรวมกันเป็นกลุ่ม เช่น อุปกรณ์เครื่องมือต่างๆ ในออฟฟิศ วิธีการคิดก็คือ ต้นปีมีมูลค่าเท่าไรบวกกับระหว่างปีซื้อเพิ่มเท่าไร ลบกับที่เหลืออยู่ปลายปี ค่าที่ได้คือค่าเสื่อมราคานั่นเอง เช่น

ตรวจนับอุปกรณ์ขนาดเล็กต่างๆ ที่อยู่ในออฟฟิต เมื่อวันที่ 1 มกราคม 25xx มีมูลค่า 35,000 บาท ระหว่างปีซื้อมาอีก 15,000 บาท เมื่อตรวจนับวันที่ 31 ธันวาคม 25xx มูลค่าอุปกรณ์เหลือ 9,000 บาท ค่าเสื่อมราคาที่ได้คือ 41,000 บาท เกิดจาก

```
(35,000+15,000)-9,000
```

รายละเอียดของคลาส Depreciation เฉพาะการคำนวณแบบคำนวณสินทรัพย์คงเหลือ ดังนี้

โค้ด VC# 2005 ที่ 20-2 การคำนวณค่าเสื่อมราคาแบบคำนวณสินทรัพย์คงเหลือ (GAF.Biz.Accounting.Depreciation.Inventory.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GAF.Biz.Accounting                    //เนมสเปซของคลาส Depreciation
{
    public partial class Depreciation           //กำหนดให้เป็น Partial Class
    {
        private double[] _TotalAsset;           //ฟิลด์เก็บมูลค่าสินทรัพย์ทั้งหมด
        private double _AssetValueInventory = 0.0;  //ฟิลด์เก็บมูลค่าสินทรัพย์คงเหลือ

        public double[] TotalAsset              //คุณสมบัติ TotalAsset คืนค่าเป็นอาร์เรย์ของ double
        {
            get{return _TotalAsset;}            //ให้อ่านค่าได้
            set{_TotalAsset = value;}           //กำหนดค่าได้
        }

        public double AssetValueInventory       //คุณสมบัติ AssetValueInventory คืนค่าเป็น double
        {
            get{return _AssetValueInventory;}   //ให้อ่านค่าได้
            set{_AssetValueInventory = value;}  //กำหนดค่าได้
        }

    //คอนสตรัคเตอร์ตัวที่ 5 ต้องการพารามิเตอร์ 2 ตัวคือ
    //พารามิเตอร์ TotalAsset หมายถึง มูลค่าสินทรัพย์ทั้งหมด
    //พารามิเตอร์ AssetValueInventory หมายถึง มูลค่าสินทรัพย์ที่เหลืออยู่
        public Depreciation(double[] TotalAsset, double AssetValueInventory)
        {
            _TotalAsset = TotalAsset;           //เก็บสินทรัพย์ทั้งหมดไว้ที่ฟิลด์ _TotalAsset
```

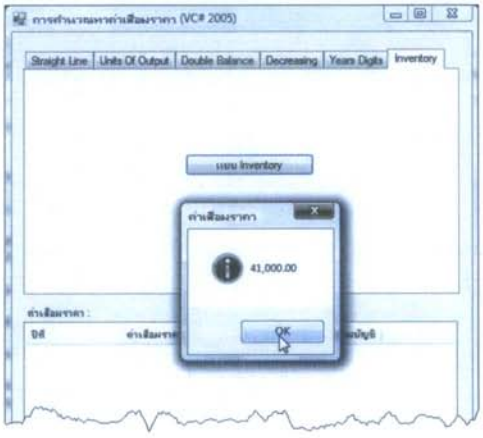
```
//เก็บมูลค่าสินทรัพย์ที่เหลือไว้ที่ฟิลด์ _AssetValueInventory
            _AssetValueInventory = AssetValueInventory;
        }

//เมธอด Inventory() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบคำนวณสินทรัพย์คงเหลือ
//คืนค่าเป็นข้อมูลชนิด double
        public double Inventory()
        {
            double _TotalValue = 0.0;                    //ฟิลด์เก็บมูลค่าสินทรัพย์ทั้งหมด
            for (_i = 0; _i <= _TotalAsset.Length - 1; _i++)
            {
                _TotalValue += _TotalAsset[_i];          //หามูลค่าสินทรัพย์ทั้งหมดก่อน
            }

//มูลค่าสินทรัพย์ทั้งหมด - มูลค่าสินทรัพย์คงเหลือ ผลที่ได้คือ ค่าเสื่อมราคา
            return _TotalValue - _AssetValueInventory;
        }
    }
}
```

เนื่องจากมูลค่าสินทรัพย์ขนาดเล็กๆ ที่อยู่ระหว่างรอบปีบัญชี สามารถซื้อเพิ่มได้เรื่อยๆ ตีความได้ว่ามีหลายค่า ผู้เขียนจึงกำหนดให้พารามิเตอร์ TotalAsset รับค่าเป็นแบบอาร์เรย์ double คุณอาจจะแก้ไขพารามิเตอร์ตัวนี้ให้เป็นแบบรับมูลค่าสินทรัพย์รวมทั้งหมดก็ได้

VC# 2005

```
        public Depreciation(double[] TotalAsset, double AssetValueInventory)
        {
            _TotalAsset = TotalAsset;
            _AssetValueInventory = AssetValueInventory;
        }
```

รูปที่ 20-11 ผลการคิดค่าเสื่อมราคาแบบคำนวณสินทรัพย์คงเหลือ



การคำนวณค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน

เป็นวิธีที่ซับซ้อนมากที่สุด วิธีคิดก็คือ เราจะถือว่าสินทรัพย์ที่นำมาใช้ในปีแรกๆ มีประสิทธิภาพมากที่สุด ส่งผลให้ได้รับประโยชน์มากที่สุดเช่นกัน ในทางกลับกันก็จะมีค่าเสื่อมราคามากตามไปด้วย โดยใช้อัตราส่วนที่เรียกว่า Sum of the digits เข้ามาคำนวณด้วยค่าของ Sum of the digits หาได้จากสูตร

$$\text{Sum of the digits} = \frac{nx(n+1)}{2}$$

โดยที่ n หมายถึง อายุสินทรัพย์

เมื่อได้ค่า Sum of the digits มาแล้วจะหาอัตราส่วน โดยถือว่าผลประโยชน์ในปีแรกจะมีค่าสูงสุดตามอายุการใช้งาน เช่น ซื้อเครื่องจักรมาในวันที่ 1 มกราคม 25xx ในราคา 350,000 บาท ราคาซาก 65,000 บาท อายุการใช้งานประมาณ 4 ปี ค่า Sum of the digits แทนตามสูตรได้ ดังนี้

$$\frac{4x(4+1)}{2}$$

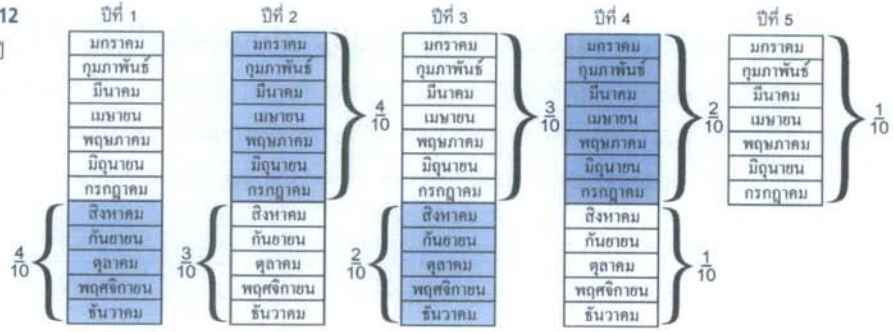
ค่าที่ได้คือ 10.0 จะนำค่านี้ไปคิดอัตราส่วนของค่าเสื่อมราคา ดังตารางต่อไปนี้

ปีที่	อัตราส่วน
1	$\frac{4}{10}$
2	$\frac{3}{10}$
3	$\frac{2}{10}$
4	$\frac{1}{10}$

จากตารางข้างต้นเห็นได้ว่า อายุการใช้งานสินทรัพย์ลดลงปีละ 1 จนเหลือค่าเท่ากับ 1 ท้ายที่สุดนำอัตราส่วนดังกล่าวไปคิดค่าเสื่อมราคา ดังตารางต่อไปนี้

ปีที่	คำนวณค่าเสื่อมราคา	ค่าเสื่อมราคา
1	$\frac{4}{10}$x(350,000-65,000)	114,000
2	$\frac{3}{10}$x(350,000-65,000)	85,500
3	$\frac{2}{10}$x(350,000-65,000)	57,000
4	$\frac{1}{10}$x(350,000-65,000)	28,500

ค่าเสื่อมราคาดังกล่าวเป็นการคิดเต็มรอบปีบัญชี เพราะว่าซื้อเครื่องจักรมาเมื่อวันที่ 1 มกราคม 25xx แต่ในกรณีที่คิดค่าเสื่อมราคาระหว่างรอบปีบัญชี เราจะต้องแยกอัตราส่วนที่จะนำมาคิดด้วย ดังรูป



รูปที่ 20-12 อัตราส่วนแต่ละปีตลอดอายุการใช้งาน

สมมติว่าคุณซื้อเครื่องจักรมาในวันที่ 1 สิงหาคม 25xx เห็นได้ว่าในปีแรกใช้เพียง 5 เดือน อายุการใช้งานของเครื่องจักรนี้ 4 ปี จึงคิดไปถึงรอบปีบัญชีที่ 5 ด้วย โดยที่

- 12 เดือนแรก (1 ปี) ของการใช้งานเครื่องจักร มีอัตราส่วน 4/10 ส่งผลให้ในรอบปีบัญชีแรก คิดเพียง 5 เดือน โดยใช้อัตราส่วน 4/10
- ในรอบปีบัญชีที่ 2 พบว่ามี 2 อัตราส่วน โดยใช้อัตราส่วน 4/10 จำนวน 7 เดือน บวกกับอัตราส่วน 3/10 อีก 5 เดือน
- ในรอบปีบัญชีที่ 3 มี 2 อัตราส่วนเช่นกัน โดยใช้อัตราส่วน 3/10 จำนวน 7 เดือน บวกกับอัตราส่วน 2/10 อีก 5 เดือน
- ในรอบปีบัญชีที่ 4 มี 2 อัตราส่วนเช่นกัน โดยใช้อัตราส่วน 2/10 จำนวน 7 เดือน บวกกับอัตราส่วน 1/10 อีก 5 เดือน
- ในรอบปีบัญชีที่ 5 ใช้อัตราส่วน 1/10 จำนวน 7 เดือน ถือว่าหมดอายุของเครื่องจักรตัวนี้

ปีที่	คำนวณค่าเสื่อมราคา	ค่าเสื่อมราคา
1	$\frac{4}{10}$ x(350,000-65,000)x $\frac{5}{12}$	47,500
2	($\frac{4}{10}$ x(350,000-65,000)x $\frac{7}{12}$)+($\frac{3}{10}$ x(350,000-65,000)x $\frac{5}{12}$)	102,125
3	($\frac{3}{10}$ x(350,000-65,000)x $\frac{7}{12}$)+($\frac{2}{10}$ x(350,000-65,000)x $\frac{5}{12}$)	73,625
4	($\frac{2}{10}$ x(350,000-65,000)x $\frac{7}{12}$)+($\frac{1}{10}$ x(350,000-65,000)x $\frac{5}{12}$)	45,125
5	$\frac{1}{10}$ x(350,000-65,000)x $\frac{7}{12}$	16,625

ดังนั้น การคิดค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน จึงต้องมี 2 แบบเช่นเดียวกับวิธีการคิดแบบเส้นตรงนั่นเอง ทั้งนี้ขึ้นอยู่กับว่าสินทรัพย์นั้นๆ นำมาคิดในวันเริ่มต้นรอบปีบัญชีใหม่หรือนำมาใช้ระหว่างรอบปีบัญชี รายละเอียดของคลาส Depreciation เฉพาะการคำนวณแบบผลบวกของลำดับปีที่ใช้งาน ดังนี้

โค้ด VC# 2005 ที่ 20-7 การคำนวณค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน (GAF.Biz.Accounting.Depreciation.YearsDigits.cs)

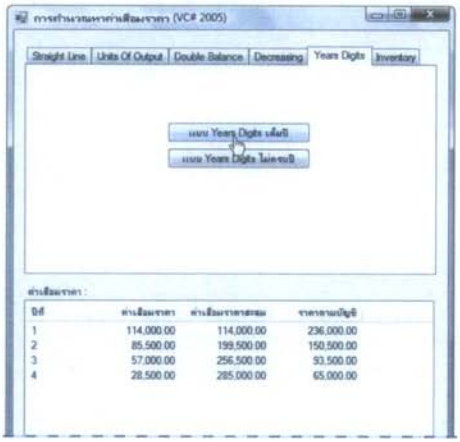
```
using System;
using System.Collections.Generic;
using System.Text;

namespace GAF.Biz.Accounting                                    //เนมสเปซของคลาส Depreciaion
{
    public partial class Depreciation                           //กำหนดให้เป็น Partial Class
    {
        private double _AssetAfter1st = 0.0;                    //ฟิลด์เก็บต้นทุนสินทรัพย์หลังจากปีที่ 1

//เมธอด YearsDigits() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน
//ใช้ในกรณีคิดเต็มรอบปีบัญชี คืนค่าเป็นอาร์เรย์ของ double
        public double[] YearsDigits()
        {
            double _SumYearsDigitsRate = 0.0;                   //ฟิลด์เก็บค่าของ Sum of the digits
//คำนวณค่า Sum of the digits ตามสูตร
            _SumYearsDigitsRate = Convert.ToDouble(_EstimatedUseLife) * ((Convert.ToDouble(_EstimatedUseLife) + 1) / 2);

//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
            double[] _TotalDepreciation = new double[_EstimatedUseLife];
            Array.Clear(_TotalDepreciation, 0, _TotalDepreciation.Length);      //ล้างค่าฟิลด์อาร์เรย์
//ต้นทุนสินทรัพย์ – ราคาซากเก็บไว้ในฟิลด์ _AssetAfter1st
            _AssetAfter1st = _CostOfAsset - _SalvageValue;
//อายุการใช้งานสินทรัพย์เก็บไว้ในฟิลด์ _CUseLife
            int _CUseLife = _EstimatedUseLife;
            for (_i = 0; _i <= _EstimatedUseLife - 1; _i++)
            {
//คำนวณค่าเสื่อมราคาตามสูตร เก็บไว้ในฟิลด์ _TotalDepreciation
                _TotalDepreciation[_i] = _AssetAfter1st * (_CUseLife / _SumYearsDigitsRate);
                _CUseLife--;                                    //ลดค่าอายุสินทรัพย์ลง 1
            }

            return _TotalDepreciation;                          //คืนค่า
        }

//เมธอด YearsDigits() แบบ Public ทำหน้าที่คำนวณค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน
//ใช้ในกรณีคิดระหว่างรอบปีบัญชี คืนค่าเป็นอาร์เรย์ของ double
//ต้องการพารามิเตอร์ 1 ตัวคือ MonthInFirstYear หมายถึง จำนวนเดือนที่ใช้สินทรัพย์ในปีแรก
        public double[] YearsDigits(int MonthInFirstYear)
        {
            double _SumYearsDigitsRate = 0.0;                   //ฟิลด์เก็บค่าของ Sum of the digits
//คำนวณค่าของ Sum of the digits ตามสูตร
            _SumYearsDigitsRate = Convert.ToDouble(_EstimatedUseLife) * ((Convert.ToDouble(_EstimatedUseLife) + 1) / 2);
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
            double[] _TotalDepreciation = new double[_EstimatedUseLife + 2];
```

```
            Array.Clear(_TotalDepreciation, 0, _TotalDepreciation.Length);          //ล้างค่าฟิลด์อาร์เรย์
//ต้นทุนสินทรัพย์ – ราคาซากเก็บไว้ในฟิลด์ _AssetAfter1st
            _AssetAfter1st = _CostOfAsset - _SalvageValue;
//อายุการใช้งานสินทรัพย์เก็บไว้ในฟิลด์ _CUseLife
            double _CUseLife = _EstimatedUseLife;
//คำนวณค่าเสื่อมราคาปีแรก ตามจำนวนเดือนที่ใช้สินทรัพย์
            _TotalDepreciation[0] = _AssetAfter1st * (_CUseLife / _SumYearsDigitsRate) * (Convert.ToDouble(MonthInFirstYear)
/ 12);

            double _FirstSection = 0.0;                                             //ฟิลด์เก็บค่าเสื่อมราคาส่วนแรก
            double _LastSection = 0.0;                                              //ฟิลด์เก็บค่าเสื่อมราคาส่วนหลัง
//คำนวณค่าเสื่อมราคาตั้งแต่ปีที่ 2 ถึงปีรองสุดท้าย
            for (_i = 1; _i <= _EstimatedUseLife; _i++)
            {
//คำนวณค่าเสื่อมราคาส่วนแรก จากอัตราส่วนแรกและจำนวนเดือนที่ใช้ส่วนแรก
              _FirstSection = (_AssetAfter1st * (_CUseLife / _SumYearsDigitsRate)) * (12.0 - MonthInFirstYear) / 12;

//คำนวณค่าเสื่อมราคาส่วนหลัง จากอัตราส่วนหลังและจำนวนเดือนที่เหลือจากส่วนแรก
              _LastSection = (_AssetAfter1st * ((_CUseLife - 1) / _SumYearsDigitsRate)) * (Convert.ToDouble(MonthInFirstYear)
/ 12);

//ค่าเสื่อมราคาตั้งแต่ปีที่ 2 จนถึงปีรองสุดท้าย = ค่าเสื่อมราคาส่วนแรก + ค่าเสื่อมราคาส่วนหลัง
              _TotalDepreciation[_i] = _FirstSection + _LastSection;
              _CUseLife--;                                                          //ลดค่าอายุสินทรัพย์ลง 1
            }

//คำนวณค่าเสื่อมราคาปีสุดท้าย
            _TotalDepreciation[_EstimatedUseLife + 1] = _AssetAfter1st * (_CUseLife / _SumYearsDigitsRate);

            return _TotalDepreciation;                                              //คืนค่า
        }

//เมธอด CumulativeDepreciationByYearsDigits() แบบ Public คืนค่าเป็นอาร์เรย์ของ double
//ทำหน้าที่คำนวณค่าเสื่อมราคาสะสมกรณีคิดเต็มรอบปีบัญชี
        public double[] CumulativeDepreciationByYearsDigits()
        {
            double[] _NormalDepreciation;                                           //ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาแต่ละปี
//สั่งให้เมธอด YearsDigits() เพื่อคำนวณค่าเสื่อมราคาแต่ละปีก่อน
            _NormalDepreciation = this.YearsDigits();
//ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคาสะสม
            double[] _TotalDepreciation = new double[_EstimatedUseLife];
            Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife);                  //ล้างค่าฟิลด์อาร์เรย์

            double _CDepreciation = 0.0;                                            //ฟิลด์เก็บค่าเสื่อมราคาปีปัจจุบัน
            for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
            {  //คำนวณค่าเสื่อมราคาสะสม
              _CDepreciation += _NormalDepreciation[_i];
              _TotalDepreciation[_i] += _CDepreciation;
            }
```

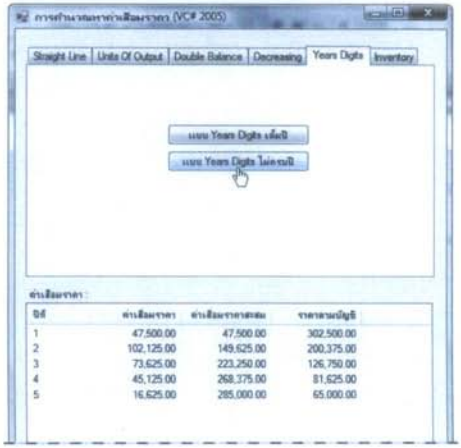
```
        return _TotalDepreciation;                                  //คืนค่า
    }

//เมธอด CumulativeDepreciationByYearsDigits() แบบ Public คืนค่าเป็นอาร์เรย์ของ double
//ทำหน้าที่คำนวณค่าเสื่อมราคาสะสมกรณีคิดระหว่างรอบปีบัญชี
//ต้องการพารามิเตอร์ 1 ตัวคือ MonthInFirstYear หมายถึง จำนวนเดือนที่ใช้สินทรัพย์ในปีแรก
    public double[] CumulativeDepreciationByYearsDigits(int MonthInFirstYear)
    {
        double[] _NormalDepreciation;                               //ฟิลด์อาร์เรย์เก็บค่าเสื่อมราคา
//สั่งให้เมธอด YearsDigits() ทำงาน โดยการส่งค่า MonthInFirstYear เข้าไป เพื่อคำนวณค่าเสื่อมราคาก่อน
        _NormalDepreciation = this.YearsDigits(MonthInFirstYear);
//ฟิลด์เก็บค่าเสื่อมราคาสะสม
        double[] _TotalDepreciation = new double[_EstimatedUseLife + 2];
        Array.Clear(_TotalDepreciation, 0, _EstimatedUseLife);      //ล้างค่าฟิลด์อาร์เรย์

        double _CDepreciation = 0.0;                                //ฟิลด์เก็บค่าเสื่อมราคาปีปัจจุบัน
        for (_i = 0; _i <= _TotalDepreciation.Length - 1; _i++)
        { //คำนวณค่าเสื่อมราคาสะสม
          _CDepreciation += _NormalDepreciation[_i];
          _TotalDepreciation[_i] += _CDepreciation;
        }

        return _TotalDepreciation;                                  //คืนค่า
    }

//เมธอด AssetValueByYearsDigits() แบบ Public คืนค่าเป็นอาร์เรย์ของ double
//ใช้ในกรณีเต็มรอบปีบัญชี ทำหน้าที่หามูลค่าสินทรัพย์
    public double[] AssetValueByYearsDigits()
    { //สั่งให้เมธอด CumulativeDepreciationByYearsDigits() ทำงาน เพื่อหาค่าเสื่อมราคาสะสมก่อน
        double[] _CumulativeDepreciation = this.CumulativeDepreciationByYearsDigits();
//ฟิลด์อาร์เรย์เก็บมูลค่าสินทรัพย์แต่ละปี
        double[] _TotalAssetValue = new double[_EstimatedUseLife];
        Array.Clear(_TotalAssetValue, 0, _EstimatedUseLife);        //ล้างค่าฟิลด์อาร์เรย์

        for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
        { //มูลค่าสินทรัพย์ปีปัจจุบัน = ต้นทุนสินทรัพย์ - ค่าเสื่อมราคาสะสมปีปัจจุบัน
          _TotalAssetValue[_i] = _CostOfAsset - _CumulativeDepreciation[_i];
        }

        return _TotalAssetValue;                                    //คืนค่า
    }

//เมธอด AssetValueByYearsDigits() แบบ Public คืนค่าเป็นอาร์เรย์ของ double
//ใช้ในกรณีเต็มรอบปีบัญชี ทำหน้าที่หามูลค่าสินทรัพย์ ใช้ในกรณีคิดระหว่างรอบปีบัญชี
//ต้องการพารามิเตอร์ 1 ตัวคือ MonthInFirstYear หมายถึง จำนวนเดือนที่ใช้สินทรัพย์ในปีแรก
    public double[] AssetValueByYearsDigits(int MonthInFirstYear)
    {
//สั่งให้เมธอด CumulativeDepreciationByYearsDigits() ทำงาน เพื่อหาค่าเสื่อมราคาสะสมก่อน
```

```
        //โดยการส่งจำนวนเดือนที่ใช้สินทรัพย์ในปีแรกเข้าไปด้วย
        double[] _CumulativeDepreciation = this.CumulativeDepreciationByYearsDigits(MonthInFirstYear);
    //ฟิลด์อาร์เรย์เก็บมูลค่าสินทรัพย์แต่ละปี
        double[] _TotalAssetValue = new double[_EstimatedUseLife + 2];
        Array.Clear(_TotalAssetValue, 0, _EstimatedUseLife);                    //ล้างค่าฟิลด์อาร์เรย์

        for (_i = 0; _i <= _CumulativeDepreciation.Length - 1; _i++)
        {  //มูลค่าสินทรัพย์ปีปัจจุบัน = ต้นทุนสินทรัพย์ - ค่าเสื่อมราคาสะสมปีปัจจุบัน
           _TotalAssetValue[_i] = _CostOfAsset - _CumulativeDepreciation[_i];
        }

        return _TotalAssetValue;                                                //คืนค่า
    }
  }
}
```

รูปที่ 20-13
ผลการคำนวณค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน กรณีคิดเต็มรอบปีบัญชี



รูปที่ 20-14
ผลการคำนวณค่าเสื่อมราคาแบบผลบวกของลำดับปีที่ใช้งาน กรณีคิดระหว่างรอบปีบัญชี



ข้อควรระวังเมื่อนำคลาส Depreciation ไปใช้งานจริงมีอยู่ 2 ประการคือ

1. เมื่อมีการคำนวณค่าเสื่อมราคาไม่ว่าวิธีใดก็ตาม ผลรวมทั้งหมดของค่าเสื่อมราคา ตลอดอายุการใช้งานของสินทรัพย์นั้นๆ อาจจะขาดหรือเกินจากมูลค่าเดิมของสินทรัพย์ เช่น

ต้นทุนสินทรัพย์ราคา 10,000 บาท แต่เมื่อคิดค่าเสื่อมราคาบวกกับราคาซากแล้ว อาจจะมีมูลค่า 9,999.66 บาท หรือ 10,000.03 บาท เห็นได้ว่าผิดไปจากความเป็นจริง ซึ่งจะส่งผลกระทบต่อการจัดทำงบทางบัญชี ปัญหานี้เกิดจากการหารไม่ลงตัวของทศนิยม 2 ตำแหน่ง (เป็นจำนวนทศนิยมที่ใช้บันทึกในระบบบัญชี) ของต้นทุนสินค้า

วิธีการแก้ไขปัญหานี้ก็คือ ก่อนการบันทึกค่าเสื่อมราคา คุณอาจจะต้องมีการตรวจสอบก่อนว่า ถ้าค่าเสื่อมราคาบวกกับราคาซากแล้วไม่เท่ากับต้นทุนสินค้า อาจจะเปิดโอกาสให้แก้ไขค่าเสื่อมราคาในปีสุดท้ายได้ เพื่อให้ค่าเท่ากับต้นทุนสินค้านั่นเอง

2. สินทรัพย์ที่ผู้เขียนยกมาใช้กับคลาส Depreciation ตามตัวอย่างข้างต้น เป็นการคิดค่าเสื่อมราคาของสินทรัพย์ใหม่ หรือกล่าวได้อีกนัยหนึ่งก็คือ เป็นสินทรัพย์ที่เริ่มคำนวณ และเริ่มใช้ตั้งแต่ปีแรก แต่ในกรณีที่มีการนำไปใช้กับสินทรัพย์ที่มีการคิดค่าเสื่อมราคาอยู่แล้วจะทำอย่างไร

วิธีการก็คือ เนื่องจากผู้เขียนกำหนดให้การคำนวณค่าเสื่อมราคาวิธีต่างๆ คืนค่าเป็นอาร์เรย์ของ double อยู่แล้ว ให้คุณเลือกใช้ค่าเสื่อมราคาตามปีของสินทรัพย์นั้นๆ ไปใช้ เพราะว่าการคืนค่าเป็นอาร์เรย์หมายถึง มีค่าเสื่อมราคาตั้งแต่ปีแรกจนถึงปีสุดท้ายของอายุการใช้งานสินทรัพย์นั้นๆ คุณจะนำค่าเสื่อมราคาในปีใดไปใช้ก็เลือกตามลำดับสมาชิกของอาร์เรย์นั่นเอง เช่น ถ้าต้องการค่าเสื่อมราคาตั้งแต่ปีที่ 2 คุณก็สั่งให้อ่านค่าตั้งแต่สมาชิกที่ 2 (ลำดับอ้างอิง 1) นั่นเอง

สรุปท้ายเล่ม

เนื้อหาโดยรวมทั้งหมดของหนังสือเล่มนี้ อาจจะยังไม่ใช่ภาคปฏิบัติสักทีเดียว ผู้เขียนคาดหวังว่าหลักการ, เนื้อหา และวิธีการที่อยู่ในหนังสือเล่มนี้ จะเป็นรากฐานที่ดีก่อนเข้าสู่โลกของ OOP อย่างเต็มตัว สวัสดีครับ... ^_^


Programming in OOP style
Advanced .net

Index

Advance .NET Programming ฉบับมืออาชีพ

เจาะลึกและตีแผ่ทุกความสามารถ ของ .NET Framework 2.0 โดยประสบการณ์ตรงของโปรแกรมเมอร์มืออาชีพที่คลุกวงในมานานเกือบ 10 ปี อธิบายควบคู่กันไปทั้ง 2 ภาษา (VB 2005 กับ VC# 2005) เพื่อให้คุณสามารถเทียบเคียงความสามารถแต่ละภาษาได้อย่างสะดวก บูรณาการความรู้ในอดีตของคุณอย่างกลมกล่อม ด้วยสไตล์การอธิบายที่เรียบง่ายในแบบ OOP ชนิดไม่กลัวง่วง

เนื้อหาภายในเล่ม ประกอบไปด้วย

- ก้าวแรกกับการเขียนโปรแกรมแบบ OOP
- ตัวแปร และ Type ใน .NET Framework
- รู้จัก Class และ Namespace
- เจาะลึกเรื่อง Method
- เรื่องของ Parameter
- รู้จัก Shared Class (Static Class)
- เรื่องของ Default Properties (Indexer)
- เจาะลึกเรื่อง Inheritance
- เรื่องของ Constructor
- เจาะ Structure และรู้จักกับ Abstract Class
- เรื่องของ Interface
- เจาะลึกการเขียนโปรแกรมกับ Interface
- ใช้งาน Array ให้ลึกกว่าเคย
- เรื่องของตัวเลขใน .NET
- จัดการวันเดือนปีและเวลา
- จัดการ String
- Data Structure และ Algorithm
- รู้จักและใช้งาน Delegate
- รวมเทคนิคเกี่ยวกับ Form
- การสร้างผู้ช่วยเหลือในระบบธุร

ผู้แต่ง ศุภชัย สมพานิช

เป็นนักเขียนระดับ Best Seller ที่มีผลงานขายดี ที่ได้รับความนิยมมาแล้วมากมาย อาทิ

- คู่มือ ASP .NET 2.0 ฉบับสมบูรณ์
- พัฒนาระบบงานฐานข้อมูลด้วย VB 6
- สร้างระบบงานฐานข้อมูลด้วย VB .NET
- สร้างระบบฐานข้อมูลด้วย VB 2005 & VC# 2005 ฉบับมืออาชีพ

ปัจจุบันเป็นนักเขียนอิสระ และให้คำปรึกษา

DEV BOOK
Enterprise Developer Book

โดย ศุภชัย สมพานิช
บรรณาธิการ สัจจะ จรัสรุ่งรวีวร

ISBN 978-974-9749-01-2
ราคา 375 บาท
9 789749 749012