VB.NET Programming: Advance

(Integrative-Generative AI Edition)

Contents:

Advanced OOP-1
Mulithreading & Async Programming-69
Database Programming (ADO.NET & Entity Framework)-133
Windows Presentation Foundation (WFF)-194
Working with APIS:
Bibliography-293

Author: Student Price Book Center

คำนำ

ในยุคปัจจุบันที่ซอฟต์แวร์และระบบสารสนเทศมีความซับซ้อนและต้องตอบสนองต่อการเปลี่ยนแปลงอย่างรวดเร็ว ความสามารถในการพัฒนาโปรแกรมอย่างมีประสิทธิภาพและ maintainable จึงเป็นทักษะสำคัญของนักพัฒนาซอฟต์แวร์ VB.NET ยังคงเป็นหนึ่งในภาษาที่ได้รับความนิยมในกลุ่มนักพัฒนาองค์กรและแอปพลิเคชันเดสก์ท็อป ด้วยความสามารถที่ครอบคลุมตั้งแต่การจัดการวัตถุ การทำงานกับฐานข้อมูล ไปจนถึงการสร้างอินเทอร์เฟซที่ทันสมัยและการทำงานแบบ asynchronous หนังสือเล่มนี้มุ่งเน้นการพัฒนา VB.NET ในระดับสูง โดยให้ความสำคัญทั้งกับแนวคิดเชิงทฤษฎีและการประยุกต์ใช้งานจริง

หนังสือเล่มนี้ถูกจัดทำขึ้นสำหรับนักพัฒนาที่มีพื้นฐาน VB.NET แล้วและต้องการยกระดับความสามารถไปสู่การออกแบบโปรแกรมขั้นสูง (Advanced Programming) เนื้อหาครอบคลุมหัวข้อสำคัญ 5 บทหลัก ได้แก่ **Advanced OOP, Multithreading & Async Programming, Database Programming (ADO.NET & Entity Framework), Windows Presentation Foundation (WPF), และ Working with APIs** โดยแต่ละบทจะอธิบายทั้งแนวคิดเชิงลึกและการประยุกต์ใช้ใน VB.NET พร้อมตัวอย่างบูรณาการเพื่อให้นักพัฒนาสามารถทดลองใช้งานจริง

บทที่ 13 **Advanced OOP** จะพาผู้อ่านเจาะลึกแนวคิดเชิงวัตถุขั้นสูง เช่น การใช้ **Abstract Class**, การทำงานร่วมกับ **Multiple Interfaces**, การจัดการ **Delegates และ Events**, และการใช้ **Lambda Expressions** ซึ่งทั้งหมดช่วยให้นักพัฒนาสามารถออกแบบโปรแกรมที่ยืดหยุ่น ลดความซ้ำซ้อน และ maintainable ตัวอย่างบูรณาการในบทนี้ยังช่วยให้เห็นภาพรวมของการประยุกต์ใช้แนวคิดเหล่านี้ในโปรแกรมจริง

บทที่ 14 **Multithreading & Async Programming** จะเน้นการทำงานพร้อมกันและงาน asynchronous ใน VB.NET เริ่มจาก **BackgroundWorker**, **Task Parallel Library (TPL)**, การใช้ **Async/Await**, และการจัดการ **CancellationToken** บทนี้ช่วยให้นักพัฒนาสามารถเพิ่มประสิทธิภาพโปรแกรม ลดเวลาในการรอ และสร้างแอปพลิเคชันที่ตอบสนองต่อผู้ใช้ได้ดี โดยตัวอย่างบูรณาการจะแสดงให้เห็นการประยุกต์ใช้ multithreading และ async programming อย่างเป็นระบบ

บทที่ 15 **Database Programming (ADO.NET & Entity Framework)** มุ่งเน้นการเชื่อมต่อและจัดการฐานข้อมูล SQL Server ผ่าน VB.NET เริ่มตั้งแต่การใช้ **SqlConnection, SqlCommand, SqlDataReader**, การจัดการข้อมูลแบบ disconnected ด้วย **DataSet และ DataTable**, ไปจนถึงการใช้ **Entity Framework และ LINQ to Entities** ซึ่งช่วยให้การจัดการข้อมูลมีประสิทธิภาพ ปลอดภัย และ maintainable ตัวอย่างบูรณาการในบทนี้จะแสดงการออกแบบโค้ดเพื่อทำงานกับฐานข้อมูลอย่างครบวงจร

บทที่ 16 **Windows Presentation Foundation (WPF)** จะช่วยให้นักพัฒนาสามารถสร้าง UI ของแอปพลิเคชัน Windows ที่ทันสมัยและตอบสนองได้ดี โดยเริ่มจาก **XAML Basics**, การเชื่อมโยง

ข้อมูลด้วย **Data Binding**, การออกแบบสถาปัตยกรรม UI แบบ **MVVM (Model-View-ViewModel)**, และการใช้ **Styles และ Resources** เพื่อสร้าง UI ที่สวยงามและ maintainable ตัวอย่างบูรณาการช่วยให้เห็นภาพการออกแบบ UI ที่เชื่อมโยงกับ logic ของโปรแกรมอย่างครบถ้วน

บทที่ 17 **Working with APIs** ครอบคลุมการสื่อสารกับบริการภายนอกผ่าน **REST API** การเรียกใช้งานด้วย **HttpClient**, การจัดการข้อมูล **JSON** ด้วย **Serialization/Deserialization**, และการใช้ไลบรารียอดนิยมอย่าง **Newtonsoft.Json และ System.Text.Json** บทนี้ช่วยให้นักพัฒนาสามารถสร้างโปรแกรมที่เชื่อมต่อกับบริการภายนอกได้อย่างรวดเร็ว ปลอดภัย และ maintainable พร้อมตัวอย่างบูรณาการเพื่อทดลองเรียก API จริง

หนังสือเล่มนี้ไม่เพียงแต่สอน syntax และการใช้งานเครื่องมือ แต่ยังเน้นแนวทางการออกแบบโปรแกรมที่ดี (Best Practices) และการประยุกต์ใช้เทคโนโลยีอย่างครบวงจร ผู้อ่านจะได้เรียนรู้การผสมผสานเทคนิคขั้นสูงทั้ง OOP, multithreading, database, UI, และ API เพื่อสร้างระบบซอฟต์แวร์ที่มีคุณภาพ

ท้ายที่สุด คำแนะนำจากหนังสือเล่มนี้จะช่วยให้นักพัฒนาสามารถยกระดับทักษะ VB.NET ไปสู่ระดับมืออาชีพ สร้างโปรแกรมที่มีประสิทธิภาพ ปลอดภัย และ maintainable พร้อมรับมือความท้าทายในโลกของการพัฒนาแอปพลิเคชันสมัยใหม่ได้อย่างมั่นใจ

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 13 Advanced OOP 1

- Advanced OOP
- Advanced OOP ใน VB.NET
- Abstract Class
- การใช้ Interface หลายตัว (Multiple Interfaces) ใน VB.NET
- Delegates และ Events
- Lambda Expressions
- ตัวอย่างบูรณาการ

บทที่ 14 Multithreading & Async Programming 69

- Multithreading & Async Programming
- เจาะลึก Multithreading & Async Programming ใน VB.NET
- BackgroundWorker ใน VB.NET
- Task Parallel Library (TPL) ใน VB.NET
- Async / Await ใน VB.NET
- CancellationToken ใน VB.NET
- ตัวอย่างบูรณาการ

บทที่ 15 Database Programming (ADO.NET & Entity Framework) 133

- Database Programming (ADO.NET & Entity Framework)
- Database Programming – รายละเอียดเชิงลึก
- การเชื่อมต่อฐานข้อมูล SQL Server ใน VB.NET
- รายละเอียดเชิงลึก ของหัวข้อ SqlConnection, SqlCommand, SqlDataReader
- DataSet และ DataTable – รายละเอียดเชิงลึก
- Entity Framework Basics – รายละเอียดเชิงลึก
- LINQ to Entities – รายละเอียดเชิงลึก
- ตัวอย่างบูรณาการ

บทที่ 16 Windows Presentation Foundation (WPF) 194

- Windows Presentation Foundation (WPF)
- Windows Presentation Foundation (WPF) – รายละเอียดเชิงลึก
- XAML Basics (WPF)
- Data Binding (WPF, VB.NET)
- MVVM Pattern (Model-View-ViewModel) – เบื้องต้น
- Styles และ Resources ใน WPF
- ตัวอย่างบูรณาการ

บทที่ 17 Working with APIs 253

- Working with APIs
- Working with APIs – รายละเอียดเชิงลึก
- การเรียกใช้ REST API ด้วย HttpClient ใน VB.NET
- JSON Serialization / Deserialization ใน VB.NET
- การใช้ Newtonsoft.Json และ System.Text.Json ใน VB.NET
- ตัวอย่างบูรณาการ

บรรณานุกรม 293

บทที่ 13

Advanced OOP

(Advanced OOP)

เนื้อหา

- Advanced OOP
- Advanced OOP ใน VB.NET
- Abstract Class
- การใช้ Interface หลายตัว (Multiple Interfaces) ใน VB.NET
- Delegates และ Events
- Lambda Expressions
- ตัวอย่างบูรณาการ

บทนำ: Advanced OOP

การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) ถือเป็นหนึ่งในแนวทางสำคัญที่ช่วยให้นักพัฒนาสามารถออกแบบระบบซอฟต์แวร์ที่ยืดหยุ่น เข้าใจง่าย และสามารถบำรุงรักษาได้ง่าย ในบทก่อนหน้านี้ เราได้เรียนรู้พื้นฐานของ OOP เช่น คลาส วัตถุ การสืบทอด และการทำงานร่วมกันระหว่างวัตถุ แต่ในโลกของซอฟต์แวร์สมัยใหม่ การเข้าใจและประยุกต์ใช้แนวคิดขั้นสูงของ OOP เป็นสิ่งสำคัญที่จะช่วยให้นักพัฒนาสามารถสร้างระบบที่ซับซ้อนขึ้นได้

บทนี้มุ่งเน้นไปที่แนวคิด **Advanced OOP** ซึ่งประกอบด้วยเทคนิคและโครงสร้างที่เหนือกว่าพื้นฐาน เช่น การสร้างคลาสแบบ Abstract และ Sealed การใช้งาน Interface หลายตัว และการทำงานร่วมกับ Delegates และ Events รวมถึง Lambda Expressions การเข้าใจองค์ประกอบเหล่านี้จะช่วยให้นักพัฒนาสามารถออกแบบระบบที่ยืดหยุ่นมากขึ้น ลดความซ้ำซ้อนของโค้ด และสนับสนุนหลักการของ SOLID ในการพัฒนาโปรแกรม

Abstract Class เป็นแนวคิดที่สำคัญในการออกแบบ OOP ขั้นสูง โดยคลาสประเภทนี้จะไม่สามารถสร้างวัตถุได้โดยตรง แต่ทำหน้าที่เป็นแม่แบบสำหรับคลาสย่อย ซึ่งช่วยให้นักพัฒนาสามารถกำหนดโครงสร้างพื้นฐานและพฤติกรรมที่ต้องมีในคลาสลูกได้อย่างชัดเจน การใช้ Abstract Class อย่างเหมาะสมช่วยให้โค้ดมีความยืดหยุ่นและลดความซ้ำซ้อนในการทำงานของระบบ

ในทางตรงกันข้าม **Sealed Class** เป็นเครื่องมือที่ใช้ป้องกันการสืบทอดเพิ่มเติมจากคลาสที่ถูกประกาศ การเลือกใช้ Sealed Class ช่วยให้นักพัฒนาสามารถควบคุมการสืบทอดและการเปลี่ยนแปลงพฤติกรรมของคลาสได้ ซึ่งมีประโยชน์ในการออกแบบระบบที่ต้องการความมั่นคงและปลอดภัยสูง

การ **ใช้ Interface หลายตัว** เป็นอีกหนึ่งแนวทางสำคัญของ Advanced OOP ที่ช่วยให้วัตถุสามารถสื่อสารและทำงานร่วมกันได้อย่างยืดหยุ่น การรวม Interface หลายตัวในคลาสเดียวทำให้สามารถสร้างพฤติกรรมแบบผสมผสานได้โดยไม่ต้องพึ่งการสืบทอดเชิงซ้อน ซึ่งเหมาะกับสถาปัตยกรรมระบบที่ซับซ้อนและต้องการ modular design

ในส่วนของ **Delegates และ Events** จะช่วยให้นักพัฒนาสามารถจัดการกับเหตุการณ์และ callback functions ได้อย่างยืดหยุ่น Delegates ทำหน้าที่เป็นตัวแทนฟังก์ชัน ขณะที่ Events ช่วยให้คลาสสามารถส่งสัญญาณและแจ้งเตือนผู้สนใจเมื่อเกิดเหตุการณ์บางอย่างขึ้น การใช้แนวคิดเหล่านี้ทำให้การเขียนโปรแกรมเชิง event-driven มีความเป็นระเบียบและลดความซับซ้อนของโค้ด

Lambda Expressions เป็นฟีเจอร์สำคัญที่ทำให้โค้ดสั้นลงและอ่านง่ายขึ้น โดยช่วยให้สามารถสร้างฟังก์ชันแบบ inline ได้อย่างยืดหยุ่น Lambda Expressions ถูกใช้อย่างกว้างขวางในงานที่ต้องใช้ฟังก์ชันเป็นค่าหรือส่งต่อฟังก์ชันเป็นอาร์กิวเมนต์ ทำให้สามารถพัฒนาโค้ดที่ concise และ expressive ได้

ท้ายที่สุด การเข้าใจและผสมผสานองค์ประกอบทั้งหมดในบทนี้จะช่วยให้นักพัฒนามีเครื่องมือและแนวทางในการออกแบบระบบที่มีความซับซ้อนสูง พร้อมทั้งลดข้อผิดพลาดและเพิ่มความสามารถในการบำรุงรักษา ในบทถัดไป เราจะเจาะลึกตัวอย่างการประยุกต์ใช้แนวคิดเหล่านี้ในโปรแกรมจริง เพื่อให้นักพัฒนาสามารถเข้าใจและนำไปใช้งานได้ทันที

Advanced OOP

1. Abstract Class (คลาสเชิงนามธรรม)

- ใน VB.NET ใช้คีย์เวิร์ด **MustInherit**
- ไม่สามารถสร้าง instance ของคลาสนี้ได้โดยตรง
- ใช้สำหรับการสร้าง **แม่แบบ (template)** ของคลาส โดยบังคับให้คลาสลูก (Derived Class) ต้อง implement method ที่เป็น **MustOverride**

☐ ตัวอย่าง

```
MustInherit Class Shape
    Public MustOverride Function GetArea() As Double
End Class

Class Circle
    Inherits Shape
    Private radius As Double
```

```
    Public Sub New(r As Double)
        radius = r
    End Sub

    Public Overrides Function GetArea() As Double
        Return Math.PI * radius * radius
    End Function
End Class

Module Program
    Sub Main()
        Dim c As New Circle(5)
        Console.WriteLine("Circle Area = " & c.GetArea())
    End Sub
End Module
```

☐ ข้อดี: ช่วยกำหนดโครงสร้างที่ชัดเจนสำหรับคลาสลูก

☐ ข้อควรระวัง: ใช้เมื่อ **ต้องการ enforce พฤติกรรมร่วมกัน** ไม่ใช่เพื่อการ reuse ธรรมดา

---

2. Sealed Class (คลาสปิดผนึก)

- ใช้คีย์เวิร์ด **NotInheritable**
- ไม่สามารถสืบทอด (inherit) ได้อีก
- ใช้เมื่อไม่ต้องการให้ใครนำไปขยายเพิ่ม เช่น Utility Class

☐ ตัวอย่าง

```
NotInheritable Class Logger
    Public Shared Sub Log(message As String)
        Console.WriteLine("[LOG] " & message)
    End Sub
End Class

Module Program
    Sub Main()
        Logger.Log("Application started")
    End Sub
```

```
End Module
```

- เหมาะกับ **Utility / Helper Class**
- หากใช้พร่ำเพรื่อ จะทำให้โค้ดยืดหยุ่นน้อยลง

---

3. การใช้ Interface หลายตัว

- VB.NET อนุญาตให้คลาสหนึ่ง **implements ได้หลาย interface**
- ใช้เพื่อแยก **สัญญา (contract)** ของการทำงาน
- คล้าย **Multiple Inheritance (แต่ปลอดภัยกว่า)**

- ตัวอย่าง

```
Interface IPrintable
    Sub Print()
End Interface

Interface ISavable
    Sub Save()
End Interface

Class Report
    Implements IPrintable, ISavable

    Public Sub Print() Implements IPrintable.Print
        Console.WriteLine("Printing report...")
    End Sub

    Public Sub Save() Implements ISavable.Save
        Console.WriteLine("Saving report...")
    End Sub
End Class

Module Program
    Sub Main()
        Dim r As New Report()
        r.Print()
        r.Save()
```

```
    End Sub
End Module
```

- ข้อดี: แยก role ของคลาสได้อย่างชัดเจน
- ถ้ามีหลาย interface ที่ชื่อ method ซ้ำกัน อาจต้องเขียน implement แยกอย่างชัดเจน

---

4. Delegates และ Events

- **Delegate** = ตัวชี้ไปยังเมธอด (Function Pointer) ใน VB.NET
- **Event** = กลไกที่ใช้ delegate แจ้งเตือนเมื่อมีเหตุการณ์เกิดขึ้น
- ใช้เยอะมากใน Windows Forms / WPF

- ตัวอย่าง Delegate + Event

```
Public Delegate Sub NotifyHandler(message As String)

Class Process
    Public Event ProcessCompleted As NotifyHandler

    Public Sub Start()
        Console.WriteLine("Processing...")
        Threading.Thread.Sleep(1000)
        RaiseEvent ProcessCompleted("Process Finished!")
    End Sub
End Class

Module Program
    Sub Main()
        Dim p As New Process()
        AddHandler p.ProcessCompleted, AddressOf ShowMessage
        p.Start()
    End Sub

    Sub ShowMessage(msg As String)
        Console.WriteLine("Event received: " & msg)
    End Sub
End Module
```

☐ ใช้ได้ดีสำหรับ **callback, UI events, async operation**

☐ ต้องระวัง memory leak หากไม่ได้ remove event handler

---

5. Lambda Expressions

- Lambda คือ **ฟังก์ชันแบบย่อ** (Anonymous Function)
- ใช้กับ **LINQ, Delegate, Event handler**
- Syntax:
  - แบบ single-line ⟶ Function(x) x * x
  - แบบ multi-line ⟶ Function(x) As Integer ... End Function

☐ ตัวอย่าง

```
Module Program
    Sub Main()
        ' Lambda สำหรับคำนวณกำลังสอง
        Dim square = Function(x As Integer) x * x
        Console.WriteLine("Square of 5 = " & square(5))

        ' Lambda + LINQ
        Dim numbers = {1, 2, 3, 4, 5}
        Dim evenNumbers = numbers.Where(Function(n) n Mod 2 = 0)
        Console.WriteLine("Even numbers: " & String.Join(", ", evenNumbers))
    End Sub
End Module
```

☐ ทำให้โค้ดสั้นและอ่านง่ายขึ้น

☐ ถ้าใช้ซับซ้อนเกินไปจะอ่านยาก

☐ **สรุป**

- **Abstract Class** ⟶ ใช้เป็นแม่แบบที่ต้องสืบทอด
- **Sealed Class** ⟶ ไม่อนุญาตให้สืบทอด
- **Multiple Interfaces** ⟶ แยก contract หลายบทบาท
- **Delegates & Events** ⟶ ใช้สำหรับ callback และ event-driven programming
- **Lambda Expressions** ⟶ ฟังก์ชันย่อ ใช้กับ LINQ และ Delegate ได้ดี

Advanced OOP ใน VB.NET

1. Abstract Class

แนวคิด

- MustInherit ใช้สร้างคลาส abstract ใน VB.NET
- ไม่สามารถสร้างวัตถุ (instance) จาก Abstract Class โดยตรง
- ใช้เพื่อเป็น **แม่แบบ** (blueprint) ให้กับคลาสลูก (Subclass)
- เมธอดที่บังคับให้ Override จะใช้ MustOverride

ตัวอย่าง

```
Public MustInherit Class Shape
    Public MustOverride Function Area() As Double
    Public MustOverride Function Perimeter() As Double
End Class

Public Class Circle
    Inherits Shape
    Private Radius As Double

    Public Sub New(r As Double)
        Radius = r
    End Sub

    Public Overrides Function Area() As Double
        Return Math.PI * Radius * Radius
    End Function

    Public Overrides Function Perimeter() As Double
        Return 2 * Math.PI * Radius
    End Function
End Class
```

ใช้งานจริงใน **การสร้างกรอบมาตรฐาน (contract)** เช่น ระบบ Geometry, Game Object, Business Entity

2. Sealed Class

□ แนวคิด

- ใช้ NotInheritable เพื่อบอกว่าคลาส **ไม่สามารถสืบทอดได้**
- ป้องกันไม่ให้ถูกนำไปเป็น base class
- ใช้กับ **Utility Class, Helper Class** เพื่อความปลอดภัย

□ ตัวอย่าง

```
Public NotInheritable Class MathHelper
    Public Shared Function Add(a As Integer, b As Integer) As Integer
        Return a + b
    End Function
End Class

' □ Error: Cannot inherit from sealed class
' Public Class MyMath
'     Inherits MathHelper
' End Class
```

□ ใช้งานจริงกับ **คลาสที่ต้องการปิดการสืบทอด** เช่น Logger, Configuration, Cryptography

---

3. การใช้ Interface หลายตัว

□ แนวคิด

- VB.NET รองรับ **Multiple Interface Implementation**
- ใช้ Implements เพื่อให้คลาสบังคับใช้ทุกเมธอดของ Interface ที่เกี่ยวข้อง
- ใช้ได้กับ **Dependency Injection, Service Layer, Polymorphism**

□ ตัวอย่าง

```
Public Interface IReadable
    Sub ReadData()
End Interface

Public Interface IWritable
    Sub WriteData(data As String)
End Interface

Public Class FileManager
    Implements IReadable, IWritable
```

```
    Public Sub ReadData() Implements IReadable.ReadData
        Console.WriteLine("Reading data from file...")
    End Sub

    Public Sub WriteData(data As String) Implements IWritable.WriteData
        Console.WriteLine("Writing data: " & data)
    End Sub
End Class
```

☐ ใช้งานจริงใน **การทำงานหลายหน้าที่** เช่น Class ที่เป็นทั้ง Reader และ Writer, Repository ที่อ่าน/เขียนข้อมูล

---

4. Delegates และ Events

☐ **Delegates**

- ตัวแทนของฟังก์ชัน (Function Pointer ใน VB.NET)
- ใช้แทนเมธอด และสามารถเก็บเป็นตัวแปร
- ใช้กับ **Callback, Event Handling, Multithreading**

```
Public Delegate Function MathOperation(x As Integer, y As Integer) As Integer

Module Program
    Sub Main()
        Dim add As MathOperation = AddressOf Add
        Console.WriteLine(add(5, 10))
    End Sub

    Function Add(a As Integer, b As Integer) As Integer
        Return a + b
    End Function
End Module
```

☐ **Events**

- ใช้ Event ประกาศเหตุการณ์
- ใช้กับ WinForms, WPF, หรือ Custom Event
- Event จะถูก **Raise** และมีการ Handle โดย Handler

```
Public Class Publisher
```

```
    Public Event DataReceived(msg As String)

    Public Sub SendMessage(message As String)
        RaiseEvent DataReceived(message)
    End Sub
End Class

Public Class Subscriber
    Public Sub HandleMessage(msg As String)
        Console.WriteLine("Message received: " & msg)
    End Sub
End Class

Module Program
    Sub Main()
        Dim pub As New Publisher()
        Dim sub1 As New Subscriber()

        AddHandler pub.DataReceived, AddressOf sub1.HandleMessage
        pub.SendMessage("Hello World!")
    End Sub
End Module
```

☐ ใช้งานจริงกับ **Event-Driven Programming** เช่น UI Events (Button.Click), Messaging System, Observer Pattern

---

5. Lambda Expressions

☐ **แนวคิด**

- Lambda = ฟังก์ชันนิรนาม (Anonymous Function)
- ใช้ Function หรือ Sub แบบ inline
- ใช้กับ **LINQ, Delegates, Event Handling, Short Callback**

☐ **ตัวอย่าง**

```
Module Program
    Sub Main()
        ' Lambda แบบง่าย
```

```
        Dim square = Function(x As Integer) x * x
        Console.WriteLine(square(5))    ' 25

        ' Lambda ใช้กับ List
        Dim numbers = New List(Of Integer) From {1, 2, 3, 4, 5}
        Dim evenNumbers = numbers.Where(Function(n) n Mod 2 = 0)

        For Each n In evenNumbers
            Console.WriteLine(n)
        Next
    End Sub
End Module
```

□ ใช้งานจริงกับ **LINQ Queries, Async Callback, Functional Style Programming**

□ **สรุป**

- **Abstract Class** ⟶ ใช้เป็นแม่แบบสำหรับคลาสลูก (Contract + Partial Implementation)
- **Sealed Class** ⟶ ป้องกันการสืบทอด (Final Class)
- **Multiple Interface** ⟶ อนุญาตให้คลาสทำหลายหน้าที่
- **Delegates & Events** ⟶ ใช้กับ Callback และ Event-Driven Programming
- **Lambda Expressions** ⟶ ฟังก์ชันสั้น กระชับ ใช้กับ LINQ และ Delegate

---

Abstract Class

□ **ความหมายของ Abstract Class**

- **Abstract Class** คือคลาสที่ไม่สามารถสร้างอินสแตนซ์โดยตรงได้ (ไม่สามารถ New ได้)
- มีหน้าที่เป็น **แม่แบบ (Blueprint)** ให้กับคลาสอื่น ๆ ที่สืบทอด (Inherit)
- มักใช้เพื่อ **กำหนดโครงร่าง (Contract)** ของคลาส โดยให้คลาสลูกนำไป **Implement (Override)** เอาเอง

□ **คุณสมบัติหลักของ Abstract Class**

1. **ไม่สามารถสร้างวัตถุโดยตรงได้**
2. Dim shape As New Shape() ' □ Error: Cannot create an instance of an abstract class
3. **สามารถมีทั้ง Abstract Methods และ Methods ปกติได้**

- **Abstract Method** ⟶ กำหนดชื่อ เมธอด และพารามิเตอร์ แต่ **ไม่มีการเขียนโค้ดภายใน** (บังคับให้คลาสลูกต้องเขียนเอง)
- **Normal Method** ⟶ เขียนโค้ดใน Abstract Class ได้ปกติ

4. **ใช้คีย์เวิร์ด MustInherit**
5. Public MustInherit Class Shape
6. End Class
7. **Abstract Method ใช้คีย์เวิร์ด MustOverride**
8. Public MustInherit Class Shape
9. Public MustOverride Function Area() As Double
10. End Class
11. **คลาสลูกต้องใช้ Overrides** เพื่อเขียนเมธอดแทน
12. Public Class Circle
13. Inherits Shape
14.
15. Public Overrides Function Area() As Double
16. Return Math.PI * 5 * 5
17. End Function
18. End Class

ตัวอย่างโครงสร้างการใช้งาน

```
' กำหนด Abstract Class
Public MustInherit Class Shape
    Public MustOverride Function Area() As Double
    Public MustOverride Function Perimeter() As Double

    Public Sub Display()
        Console.WriteLine("This is a shape.")
    End Sub
End Class

' คลาสสืบทอด Circle
Public Class Circle
    Inherits Shape
    Private radius As Double
```

```
    Public Sub New(r As Double)
        radius = r
    End Sub

    Public Overrides Function Area() As Double
        Return Math.PI * radius * radius
    End Function

    Public Overrides Function Perimeter() As Double
        Return 2 * Math.PI * radius
    End Function
End Class

' คลาสสืบทอด Rectangle
Public Class Rectangle
    Inherits Shape
    Private width As Double
    Private height As Double

    Public Sub New(w As Double, h As Double)
        width = w
        height = h
    End Sub

    Public Overrides Function Area() As Double
        Return width * height
    End Function

    Public Overrides Function Perimeter() As Double
        Return 2 * (width + height)
    End Function
End Class
```

```
' โปรแกรมหลัก
Module Program
    Sub Main()
        Dim shapes As List(Of Shape) = New List(Of Shape) From {
            New Circle(5),
            New Rectangle(4, 6)
        }

        For Each s In shapes
            s.Display()
            Console.WriteLine("Area: " & s.Area())
            Console.WriteLine("Perimeter: " & s.Perimeter())
            Console.WriteLine("------")
        Next

        Console.ReadLine()
    End Sub
End Module
```

☐ ผลการรัน

```
This is a shape.
Area: 78.5398163397448
Perimeter: 31.4159265358979
------
This is a shape.
Area: 24
Perimeter: 20
------
```

☐ ประเด็นเชิงลึก

1. **Abstract Class vs Interface**
    - Abstract Class ⟶ ใช้ MustInherit, มีทั้ง method + field + property ได้
    - Interface ⟶ ไม่มี field, มีแต่ signature ของ method/property
    - Abstract Class เหมาะเมื่ออยากมี **default implementation** บางส่วนให้ลูกนำไปใช้

2. **Polymorphism**
    - Abstract Class ถูกใช้ร่วมกับ **Polymorphism** ได้ดี
    - เช่นเก็บ Shape หลายชนิดใน List(Of Shape) และเรียก Area() โดยไม่ต้องรู้ว่าเป็น Circle หรือ Rectangle
3. **Best Practices**
    - ใช้ Abstract Class เมื่อต้องการกำหนดโครงร่าง + มี shared code
    - ใช้ Interface เมื่อต้องการบังคับเฉพาะ signature ของ method

---

หัวข้อ **Abstract Class ใน VB.NET**

- **3 โปรแกรมพื้นฐาน (Basic Examples)**
- **3 โปรแกรมแนวประยุกต์ (Applied Examples)**

พร้อม **โครงสร้าง + โค้ดเต็มไฟล์ + คำอธิบาย + ผลการรัน**

□ 1. โปรแกรมพื้นฐาน (Basic Examples)

□ โปรแกรมที่ 1: Abstract Class + Method Override

โครงสร้าง

Project: AbstractDemo1

Module1.vb

โค้ด

```
Module Module1

    ' กำหนด Abstract Class
    MustInherit Class Shape
        Public MustOverride Function Area() As Double
    End Class

    ' สืบทอดและ override method
    Class Circle
        Inherits Shape
        Private radius As Double

        Public Sub New(r As Double)
            radius = r
```

```
        End Sub

        Public Overrides Function Area() As Double
            Return Math.PI * radius * radius
        End Function
    End Class

    Sub Main()
        Dim c As New Circle(5)
        Console.WriteLine("Area of Circle: " & c.Area())
    End Sub

End Module
```

คำอธิบาย

- MustInherit ใช้ประกาศ **Abstract Class**
- MustOverride เป็น method ที่ลูกต้อง implement
- Circle ทำการ Override Area()

ผลการรัน

```
Area of Circle: 78.5398163397448
```

□ โปรแกรมที่ 2: Abstract Class + Multiple Derived

โครงสร้าง

Project: AbstractDemo2

Module1.vb

โค้ด

```
Module Module1

    MustInherit Class Animal
        Public MustOverride Sub Speak()
    End Class

    Class Dog
        Inherits Animal
        Public Overrides Sub Speak()
```

```
            Console.WriteLine("Dog says: Woof Woof")
        End Sub
    End Class

    Class Cat
        Inherits Animal
        Public Overrides Sub Speak()
            Console.WriteLine("Cat says: Meow")
        End Sub
    End Class

    Sub Main()
        Dim animals As Animal() = {New Dog(), New Cat()}
        For Each a In animals
            a.Speak()
        Next
    End Sub

End Module
```

ผลการรัน

```
Dog says: Woof Woof
Cat says: Meow
```

□ โปรแกรมที่ 3: Abstract Class + Property

โครงสร้าง

Project: AbstractDemo3

Module1.vb

โค้ด

```
Module Module1

    MustInherit Class Employee
        Public MustOverride ReadOnly Property Position As String
        Public MustOverride Function GetSalary() As Double
    End Class
```

```
    Class Manager
        Inherits Employee
        Public Overrides ReadOnly Property Position As String
            Get
                Return "Manager"
            End Get
        End Property
        Public Overrides Function GetSalary() As Double
            Return 50000
        End Function
    End Class

    Class Developer
        Inherits Employee
        Public Overrides ReadOnly Property Position As String
            Get
                Return "Developer"
            End Get
        End Property
        Public Overrides Function GetSalary() As Double
            Return 30000
        End Function
    End Class

    Sub Main()
        Dim emp As Employee = New Manager()
        Console.WriteLine(emp.Position & " Salary: " & emp.GetSalary())

        emp = New Developer()
        Console.WriteLine(emp.Position & " Salary: " & emp.GetSalary())
    End Sub

End Module
```

ผลการรัน

Manager Salary: 50000

Developer Salary: 30000

□ 2. โปรแกรมแนวประยุกต์ (Applied Examples)

□ โปรแกรมที่ 4: Abstract Class ในระบบคำนวณพื้นที่ (Geometry App)

โครงสร้าง

Project: ShapeApp

Module1.vb

โค้ด

```
Module Module1

    MustInherit Class Shape
        Public MustOverride Function Area() As Double
        Public MustOverride Function Perimeter() As Double
    End Class

    Class Rectangle
        Inherits Shape
        Private width As Double
        Private height As Double

        Public Sub New(w As Double, h As Double)
            width = w : height = h
        End Sub

        Public Overrides Function Area() As Double
            Return width * height
        End Function

        Public Overrides Function Perimeter() As Double
            Return 2 * (width + height)
        End Function
```

```
    End Class

    Class Triangle
        Inherits Shape
        Private a, b, c As Double

        Public Sub New(x As Double, y As Double, z As Double)
            a = x : b = y : c = z
        End Sub

        Public Overrides Function Area() As Double
            Dim s = (a + b + c) / 2
            Return Math.Sqrt(s * (s - a) * (s - b) * (s - c))
        End Function

        Public Overrides Function Perimeter() As Double
            Return a + b + c
        End Function
    End Class

    Sub Main()
        Dim s1 As Shape = New Rectangle(10, 5)
        Console.WriteLine("Rectangle: Area=" & s1.Area() & ", Perimeter=" & s1.Perimeter())

        Dim s2 As Shape = New Triangle(3, 4, 5)
        Console.WriteLine("Triangle: Area=" & s2.Area() & ", Perimeter=" & s2.Perimeter())
    End Sub

End Module
```

ผลการรัน

```
Rectangle: Area=50, Perimeter=30
Triangle: Area=6, Perimeter=12
```

□ โปรแกรมที่ 5: Abstract Class ใช้ทำ Payment System

โครงสร้าง

Project: PaymentSystem

Module1.vb

โค้ด

```
Module Module1

    MustInherit Class Payment
        Public MustOverride Sub Pay(amount As Double)
    End Class

    Class CreditCardPayment
        Inherits Payment
        Public Overrides Sub Pay(amount As Double)
            Console.WriteLine("Paid " & amount & " using Credit Card")
        End Sub
    End Class

    Class PayPalPayment
        Inherits Payment
        Public Overrides Sub Pay(amount As Double)
            Console.WriteLine("Paid " & amount & " using PayPal")
        End Sub
    End Class

    Sub Main()
        Dim p As Payment

        p = New CreditCardPayment()
        p.Pay(1500)

        p = New PayPalPayment()
        p.Pay(2000)
    End Sub
```

```
End Module
```

ผลการรัน

```
Paid 1500 using Credit Card
Paid 2000 using PayPal
```

□ โปรแกรมที่ 6: Abstract Class ใช้ทำ Vehicle System

โครงสร้าง

Project: VehicleSystem

Module1.vb

โค้ด

```
Module Module1

    MustInherit Class Vehicle
        Public MustOverride Sub Drive()
    End Class

    Class Car
        Inherits Vehicle
        Public Overrides Sub Drive()
            Console.WriteLine("Driving a Car")
        End Sub
    End Class

    Class Motorcycle
        Inherits Vehicle
        Public Overrides Sub Drive()
            Console.WriteLine("Riding a Motorcycle")
        End Sub
    End Class

    Class Truck
        Inherits Vehicle
        Public Overrides Sub Drive()
            Console.WriteLine("Driving a Truck")
```

```
        End Sub
    End Class

    Sub Main()
        Dim vehicles As Vehicle() = {New Car(), New Motorcycle(), New Truck()}
        For Each v In vehicles
            v.Drive()
        Next
    End Sub

End Module
```

ผลการรัน

```
Driving a Car
Riding a Motorcycle
Driving a Truck
```

 ตอนนี้คุณได้ **Abstract Class Example 6 โปรแกรม** ครบถ้วน (Basic 3 + Applied 3) พร้อมโครงสร้าง + โค้ดเต็มไฟล์ + คำอธิบาย + ผลการรัน

คุณอยากให้ผมทำ **แบบบูรณาการ 5 ตัวอย่าง (รวม Abstract Class + Interface + Delegate + Event + Lambda)** ต่อจากนี้เลยไหมครับ?
ดีเลยครับ มาต่อกันที่หัวข้อ **Sealed Class** ใน VB.NET
ผมจะอธิบายก่อน แล้วตามด้วย **ตัวอย่างโปรแกรมเต็มไฟล์ 3 โปรแกรมพื้นฐาน** และ **3 โปรแกรมแนวประยุกต์** (มีโครงสร้าง + อธิบายโค้ด + ผลการรัน) ให้ครับ

 ความหมายของ Sealed Class
ใน VB.NET เราสามารถใช้ **NotInheritable** เพื่อประกาศ **Sealed Class** ได้ ซึ่งหมายความว่า **ไม่สามารถสืบทอด (Inherit)** จากคลาสนั้นได้

- ปกติเราจะใช้ Sealed Class เมื่อ:
    1. ไม่ต้องการให้ใครขยายต่อ (เช่น Security Class)
    2. ต้องการป้องกันการเปลี่ยนพฤติกรรม
    3. เพื่อประสิทธิภาพ (JIT Compiler สามารถ Optimize ได้มากขึ้น)

 คีย์เวิร์ดที่ใช้ใน VB.NET:

```
Public NotInheritable Class MyClass
```

```
    ' ไม่สามารถสืบทอดได้
End Class
```

□ โปรแกรมพื้นฐาน 3 ตัวอย่าง

□ ตัวอย่างที่ 1: Sealed Class พื้นฐาน

ไฟล์: Program1.vb

```
Module Program1
    ' สร้าง Sealed Class ด้วย NotInheritable
    Public NotInheritable Class MathUtils
        Public Shared Function Square(x As Integer) As Integer
            Return x * x
        End Function
    End Class

    Sub Main()
        Console.WriteLine("Square(5) = " & MathUtils.Square(5))

        ' □ ไม่สามารถทำได้: Inherit จาก MathUtils
        ' Public Class NewMathUtils
        '     Inherits MathUtils  ' Error!
        ' End Class

        Console.ReadLine()
    End Sub
End Module
```

อธิบายโค้ด:

- MathUtils ถูกทำให้ **NotInheritable** ⟶ ห้ามสืบทอด
- ฟังก์ชัน Square ใช้เป็น Utility Function
- แสดงผล Square ของ 5

ผลการรัน:

```
Square(5) = 25
```

□ ตัวอย่างที่ 2: Sealed Class + Shared Method

ไฟล์: Program2.vb

```
Module Program2
    ' Sealed Class เก็บค่าคงที่
    Public NotInheritable Class AppConfig
        Public Shared ReadOnly AppName As String = "VB.NET Demo"
        Public Shared ReadOnly Version As String = "1.0"
    End Class

    Sub Main()
        Console.WriteLine("Application: " & AppConfig.AppName)
        Console.WriteLine("Version: " & AppConfig.Version)
        Console.ReadLine()
    End Sub
End Module
```

อธิบายโค้ด:

- ใช้ Sealed Class AppConfig เก็บค่าคงที่ของโปรแกรม
- Shared ReadOnly ทำให้เข้าถึงได้เลยโดยไม่ต้องสร้าง Object

ผลการรัน:

```
Application: VB.NET Demo
Version: 1.0
```

☐ ตัวอย่างที่ 3: Sealed Class + Prevent Override

ไฟล์: Program3.vb

```
Module Program3
    ' Sealed Class
    Public NotInheritable Class Logger
        Public Shared Sub Log(message As String)
            Console.WriteLine("[LOG] " & message)
        End Sub
    End Class

    Sub Main()
        Logger.Log("Application started.")
        Logger.Log("Performing tasks...")
        Logger.Log("Application finished.")
```