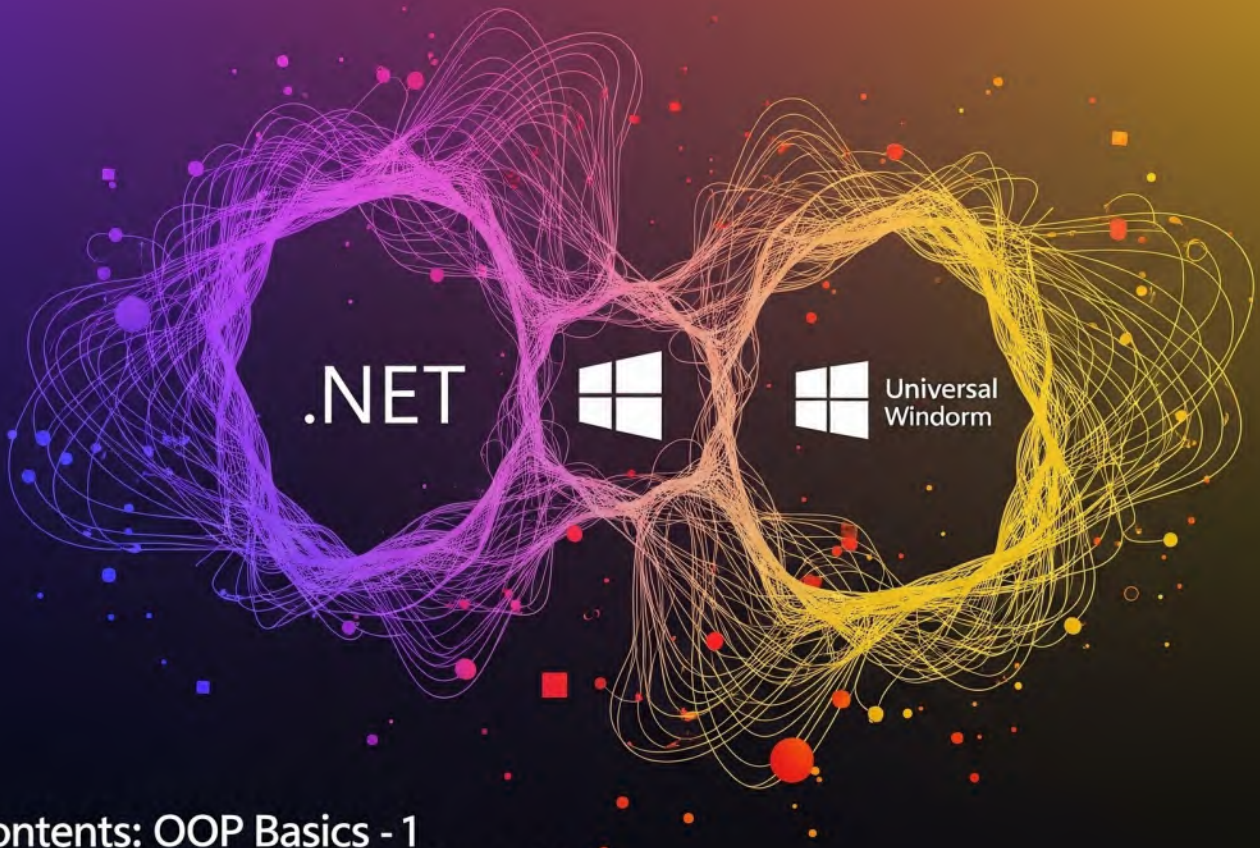
VB.NET Programming: Intermediate (Integrative-Generative AI Edition)



Author: Student Price Book Center

คำนำ

ในยุคที่เทคโนโลยีซอฟต์แวร์และการพัฒนาแอปพลิเคชันมีความซับซ้อนมากขึ้น ความสามารถในการเขียนโปรแกรมอย่างมีโครงสร้างและมีประสิทธิภาพถือเป็นสิ่งจำเป็นสำหรับนักพัฒนาซอฟต์แวร์ VB.NET เป็นหนึ่งในภาษาที่ให้ทั้งความยืดหยุ่นและความง่ายในการใช้งาน ทำให้นักพัฒนาสามารถสร้างโปรแกรมที่มีคุณภาพสูงได้อย่างรวดเร็ว หนังสือเล่มนี้ถูกออกแบบมาสำหรับผู้ที่ผ่านขั้นพื้นฐานของ VB.NET มาแล้วและต้องการก้าวเข้าสู่ระดับ **Intermediate** ด้วยการเจาะลึกแนวคิดสำคัญและการประยุกต์ใช้งานจริง

หนังสือเล่มนี้เริ่มต้นด้วย **บทที่ 7: Object-Oriented Programming (OOP) Basics** ซึ่งเน้นการเข้าใจแนวคิดเชิงวัตถุใน VB.NET อย่างละเอียด ทั้ง Class, Object, Properties, Methods, Fields และการสร้าง Constructor รวมถึงวิธีการสร้าง Instance ของ Object ผ่านตัวอย่างบูรณาการที่ช่วยให้นักพัฒนานำแนวคิดไปใช้งานได้ทันที การเรียนรู้ OOP อย่างลึกซึ้งเป็นรากฐานสำคัญที่จะช่วยให้เข้าใจการออกแบบโปรแกรมที่ยืดหยุ่นและง่ายต่อการบำรุงรักษา

ต่อด้วย **บทที่ 8: Inheritance & Polymorphism** หนังสือจะเจาะลึกการสืบทอด (Inheritance) การเขียนทับเมธอด (Overriding Methods) การใช้งานเมธอดแบบ Overloading และการใช้ Interfaces ใน VB.NET ผ่านตัวอย่างโปรแกรมแบบบูรณาการ ความเข้าใจในหัวข้อเหล่านี้จะช่วยให้นักพัฒนาสามารถสร้างโค้ดที่ยืดหยุ่นและปรับเปลี่ยนพฤติกรรมของ Object ได้ตามต้องการ ลดการทำซ้ำโค้ด และออกแบบระบบซอฟต์แวร์ที่มีความซับซ้อนได้อย่างเป็นระบบ

บทที่ 9: Collections & Generics จะเน้นการจัดการข้อมูลจำนวนมากอย่างมีประสิทธิภาพ ครอบคลุม Arrays, List(Of T), Dictionary(Of TKey, TValue), Queue และ Stack รวมถึงการใช้ LINQ พื้นฐานเพื่อกรองและเลือกข้อมูลแบบ From ... Where ... Select หนังสือเล่มนี้ให้ตัวอย่างเชิงลึกใน VB.NET เพื่อให้นักพัฒนาสามารถจัดการข้อมูลและใช้โครงสร้างข้อมูลอย่างเต็มศักยภาพ

การจัดการข้อผิดพลาดเป็นหัวใจสำคัญของการพัฒนาโปรแกรมที่เสถียร ใน **บทที่ 10: Exception Handling** หนังสือจะอธิบายการใช้ Try, Catch, Finally, การสร้าง Custom Exception, การ Throw Exception และการใช้ Using … End Using ผ่านตัวอย่างบูรณาการ เพื่อให้นักพัฒนาสามารถสร้างโปรแกรมที่ปลอดภัยต่อข้อผิดพลาดและสามารถฟื้นตัวจากเหตุการณ์ไม่คาดคิดได้อย่างมีประสิทธิภาพ

บทที่ 11: Working with Files จะสอนการอ่านและเขียน Text File ด้วย StreamReader และ StreamWriter การทำงานกับ Binary Files รวมถึงการใช้ My.Computer.FileSystem เพื่อจัดการไฟล์และโฟลเดอร์อย่างปลอดภัย หนังสือมีตัวอย่างโปรแกรมบูรณาการที่ช่วยให้นักพัฒนาสามารถจัดการข้อมูลและเก็บไฟล์ได้อย่างมีประสิทธิภาพ

สุดท้าย **บทที่ 12: Windows Forms Basics** จะพาผู้อ่านเข้าสู่การพัฒนาแอปพลิเคชันแบบกราฟิก (GUI) บน Windows โดยครอบคลุมการสร้าง Form การใช้ Controls เช่น Button, Label, TextBox การจัดการ Event เช่น Handles Button1.Click และการแสดงข้อความแจ้งเตือนด้วย

MessageBox ผ่านตัวอย่างเชิงปฏิบัติ หนังสือเล่มนี้จึงช่วยให้นักพัฒนาสามารถสร้างแอปพลิเคชันที่มีอินเทอร์เฟซใช้งานง่ายและตอบสนองต่อผู้ใช้ได้อย่างครบถ้วน

หนังสือเล่มนี้ถูกออกแบบให้ผู้อ่านสามารถเรียนรู้ได้อย่างเป็นขั้นตอน ตั้งแต่การเข้าใจแนวคิด OOP, การจัดการ Collections, การควบคุมข้อผิดพลาด, การจัดการไฟล์ ไปจนถึงการสร้าง GUI แบบพื้นฐาน ทุกบทมีตัวอย่างโค้ดและการประยุกต์ใช้งานจริงเพื่อเสริมความเข้าใจและเพิ่มทักษะในการพัฒนาโปรแกรม VB.NET ระดับ Intermediate

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

บทที่ 7

Object-Oriented Programming (OOP) Basics

(Object-Oriented Programming (OOP) Basics)

เนื้อหา

บทนำ: Object-Oriented Programming (OOP) Basics

Object-Oriented Programming หรือ OOP เป็นแนวคิดการเขียนโปรแกรมที่เน้นการใช้ “วัตถุ” (Objects) เป็นหน่วยหลักในการจัดการข้อมูลและพฤติกรรมของโปรแกรม ซึ่งช่วยให้การพัฒนาโปรแกรมมีความเป็นระเบียบ อ่านง่าย และสามารถนำกลับมาใช้ซ้ำได้ การเข้าใจพื้นฐานของ OOP จึงเป็นสิ่งจำเป็นสำหรับนักพัฒนาที่ต้องการสร้างซอฟต์แวร์ที่มีความซับซ้อนแต่ยังคงรักษาความชัดเจนในการออกแบบ

หัวใจสำคัญของ OOP คือ **Class และ Object** โดย Class เป็นเหมือนแม่แบบหรือแบบแปลนที่กำหนดคุณสมบัติและพฤติกรรมของวัตถุ ส่วน Object คือสิ่งที่ถูกสร้างจาก Class ซึ่งมีสถานะเฉพาะตัวและสามารถทำงานตามคุณสมบัติที่กำหนดไว้ใน Class การแยกความแตกต่างระหว่าง Class และ Object ช่วยให้นักพัฒนาสามารถจัดการและควบคุมโครงสร้างโปรแกรมได้อย่างมีระบบ

ในแต่ละ Object จะประกอบด้วย **Properties, Methods, และ Fields** ซึ่ง Properties เป็นคุณลักษณะของ Object เช่น ชื่อ อายุ หรือสถานะ Methods คือฟังก์ชันหรือพฤติกรรมที่ Object สามารถทำได้ เช่น การคำนวณหรือการเปลี่ยนค่า Fields เป็นตัวแปรภายใน Class ที่ใช้เก็บข้อมูลสำคัญของ Object การแยกองค์ประกอบเหล่านี้อย่างชัดเจนทำให้โปรแกรมมีความยืดหยุ่นและง่ายต่อการบำรุงรักษา

การสร้าง Object จำเป็นต้องเข้าใจการใช้ **Constructors** ซึ่งเป็นเมธอดพิเศษที่ถูกเรียกเมื่อ Object ถูกสร้างขึ้น Constructors ช่วยในการกำหนดค่าเริ่มต้นให้กับ Properties ของ Object ทำให้

สามารถเตรียม Object ให้พร้อมใช้งานได้ทันที การออกแบบ Constructor ที่ดีช่วยลดความซับซ้อนในการสร้าง Object หลายตัวและป้องกันข้อผิดพลาดจากการกำหนดค่าเริ่มต้นที่ไม่เหมาะสม

หนึ่งในแนวคิดสำคัญคือการ **สร้าง Instance** ของ Class ซึ่งหมายถึงการสร้าง Object ใหม่จากแม่แบบ Class การสร้าง Instance ช่วยให้นักพัฒนาสามารถทำงานกับข้อมูลและฟังก์ชันของ Object ได้อย่างอิสระ การเข้าใจวิธีสร้างและจัดการ Instance อย่างถูกต้องเป็นพื้นฐานที่สำคัญต่อการพัฒนาโปรแกรมเชิงวัตถุ

บทนี้จะเน้นการสอน OOP ในเชิงปฏิบัติ โดยใช้ตัวอย่างโค้ดที่เข้าใจง่าย พร้อมคำอธิบายทีละขั้นตอน เพื่อให้นักอ่านสามารถนำแนวคิดไปประยุกต์ใช้ในโปรแกรมจริงได้อย่างรวดเร็ว การเรียนรู้จากตัวอย่างจริงจะช่วยให้เข้าใจทั้งโครงสร้างและการทำงานของ Class, Object, Properties, Methods, Fields, Constructor และ Instance

สุดท้าย การเข้าใจพื้นฐาน OOP ไม่เพียงแต่ช่วยให้เขียนโปรแกรมได้ดีขึ้น แต่ยังเป็นพื้นฐานสำหรับการเรียนรู้แนวคิดขั้นสูง เช่น Inheritance, Polymorphism และ Encapsulation ซึ่งจะช่วยให้นักพัฒนาสามารถออกแบบโปรแกรมที่ซับซ้อนและมีคุณภาพสูงได้ในอนาคต

Object-Oriented Programming (OOP) Basics

- Class และ Object
- Properties, Methods, Fields
- Constructors
- การสร้าง Instance

ต่อไปนี้คือคำอธิบายเชิงลึกสำหรับ **บทที่ 7: Object-Oriented Programming (OOP) Basics ใน VB.NET** พร้อมตัวอย่างโค้ดแบบครบถ้วนและคำอธิบายแนวปฏิบัติที่ถูกต้อง

บทที่ 7: OOP Basics (VB.NET)

1) Class และ Object

- **Class** คือ “แบบพิมพ์เขียว” (Blueprint) กำหนดโครงสร้างข้อมูล (Fields/Properties) และพฤติกรรม (Methods)
- **Object (Instance)** คือสิ่งที่ถูกสร้างขึ้นจริงจาก Class แต่ละออบเจ็กต์มีสถานะ (state) ของตัวเอง
- ใน VB.NET คลาสเป็น **Reference Type**: ตัวแปรที่อ้างถึงออบเจ็กต์จะเก็บ “อ้างอิง” ไม่ใช่สำเนาข้อมูลจริง

```
' File: Person.vb
Option Strict On
Option Infer On
Option Explicit On
```

```
Public Class Person
    ' ===== 2) Fields, Properties, Methods จะอธิบายในหัวข้อถัดไป =====
End Class
```

คำสำคัญ: Class ... End Class, การตั้งชื่อ PascalCase, แยกไฟล์ตามคลาสเพื่อความเป็นระเบียบ

---

2) Fields, Properties, Methods

Fields

- ตัวแปรที่อยู่ภายในคลาส ใช้เก็บสถานะจริง ๆ
- ควรเป็น Private เพื่อ **encapsulation** และป้องกันการแก้ค่าตรง ๆ
- ใช้ ReadOnly เมื่อค่าควรถูกกำหนดครั้งเดียว (เช่น รหัสถาวร), ใช้ Const เมื่อเป็นค่าคงที่เวลา Compile

```
Public Class Person
    Private _firstName As String          ' backing field
    Private _lastName As String
    Public ReadOnly Id As Guid = Guid.NewGuid()  ' field แบบ ReadOnly
    Private Const MaxNameLength As Integer = 100 ' ค่าคงที่ของคลาส
End Class
```

Properties

- อินเตอร์เฟซสาธารณะสำหรับอ่าน/เขียนสถานะ โดยสามารถแทรกตรรกะตรวจสอบข้อมูล (validation) ได้
- **Auto-Implemented Property**: สั้น กระชับ เมื่อไม่ต้องการตรรกะพิเศษ
- ควบคุมการเข้าถึงได้ เช่น Public Property Name As String หรือ Public Property Name As String Private Set

```
Public Class Person
    Private _firstName As String
    Private _lastName As String

    ' Auto-implemented (ไม่มีตรรกะพิเศษ)
    Public Property Age As Integer

    ' Property มีตรรกะ validate ผ่าน backing field
    Public Property FirstName As String
        Get
```

```
            Return _firstName
        End Get
        Set(value As String)
            If value Is Nothing Then Throw New ArgumentNullException(NameOf(value))
            If value.Length = 0 OrElse value.Length > 100 Then
                Throw New ArgumentException("FirstName length must be 1..100.")
            End If
            _firstName = value
        End Set
    End Property

    Public Property LastName As String
        Get
            Return _lastName
        End Get
        Set(value As String)
            If String.IsNullOrWhiteSpace(value) Then
                Throw New ArgumentException("LastName is required.")
            End If
            _lastName = value.Trim()
        End Set
    End Property

    ' Computed/Derived Property (อ่านอย่างเดียว คำนวณจาก state ภายใน)
    Public ReadOnly Property FullName As String
        Get
            Return $"{FirstName} {LastName}"
        End Get
    End Property
End Class
```

แนวปฏิบัติ: ให้ Property เป็น public interface หลัก แทนการเปิด Field ตรง ๆ; ใช้ Private Set เพื่อความเป็น “กึ่ง immutable”

Methods

- พฤติกรรม/ฟังก์ชันของออบเจ็กต์

- Sub (ไม่คืนค่า) vs Function (คืนค่า)
- สนับสนุน **Overloading**, Optional parameters, ParamArray, ByVal/ByRef
- ใช้ Shared สำหรับเมธอดระดับคลาส (static)

```
Public Class Person
    ' ตัวอย่างเมธอดเปลี่ยนสถานะภายในอย่างปลอดภัย
    Public Sub Rename(newFirst As String, newLast As String)
        Me.FirstName = newFirst
        Me.LastName = newLast
    End Sub

    ' เมธอดคำนวณ/คืนค่า (Function)
    Public Function IsAdult() As Boolean
        Return Age >= 18
    End Function

    ' Overloading
    Public Function Greet() As String
        Return $"Hello, I'm {FullName}."
    End Function

    Public Function Greet(toName As String) As String
        Return $"Hello {toName}, I'm {FullName}."
    End Function

    ' Shared method (ระดับคลาส)
    Public Shared Function CreateDefaultAdult(first As String, last As String) As Person
        Return New Person(first, last, 18)
    End Function
End Class
```

เคล็ดลับ:

- ใช้ ByRef เมื่อเมธอดต้องแก้ค่าพารามิเตอร์ภายนอก
- ใช้ Optional และ := (Named arguments) เพื่อให้อ่านง่าย
- แยกเมธอด “ปรับสถานะ” ออกจากเมธอด “คำนวณค่า”

3) Constructors

- คอนสตรัคเตอร์คือ Sub New(...) สำหรับกำหนดสถานะเริ่มต้นเมื่อสร้างออบเจ็กต์
- สนับสนุน **Overloading**, **Constructor Chaining** ผ่าน Me.New(...) และการเรียกคอนสตรัคเตอร์ฐานผ่าน MyBase.New(...)
- มี **Shared Constructor** (Shared Sub New()) รันหนึ่งครั้งต่อคลาส เหมาะสำหรับการเตรียมทรัพยากรระดับคลาส

```
Public Class Person
    ' ===== Fields/Properties ตามตัวอย่างก่อนหน้า =====

    ' คอนสตรัคเตอร์หลัก
    Public Sub New(first As String, last As String, age As Integer)
        Me.FirstName = first   ' ผ่าน Property -> ได้ validation
        Me.LastName = last
        Me.Age = age
    End Sub

    ' คอนสตรัคเตอร์ overloading ที่ใช้ค่าเริ่มต้นบางส่วน
    Public Sub New(first As String, last As String)
        Me.New(first, last, 0) ' constructor chaining
    End Sub

    ' คอนสตรัคเตอร์ว่าง (สำหรับ serializer/ORM/Designer)
    Public Sub New()
        ' อาจกำหนดค่าเริ่มต้นขั้นต่ำ
        Me.FirstName = "Unknown"
        Me.LastName = "Unknown"
        Me.Age = 0
    End Sub

    ' Shared constructor (รันครั้งเดียวต่อ AppDomain)
    Shared Sub New()
        ' เช่น โหลด configuration ระดับคลาส หรือเตรียม cache
    End Sub
End Class
```

ลำดับการ initialize โดยย่อ: ค่าเริ่มต้นของฟิลด์ ⟶ ตัวกำหนดค่าเริ่มต้นของฟิลด์ (ถ้ามี) ⟶ ตัวเรียกฐาน (MyBase.New) ⟶ เนื้อใน Sub New ปัจจุบัน

---

4) การสร้าง Instance (Object Creation)

- ใช้คำสั่ง New สร้างออบเจ็กต์จากคลาส
- รองรับ **Object Initializer** เพื่อเซ็ตพร็อพเพอร์ตีแบบย่อ
- ใช้ **Type Inference** (Dim x = New Person(...)) ได้เมื่อเปิด Option Infer On

```
' ตัวอย่างการสร้างและใช้งาน
Module Program
    Sub Main()
        Dim p1 As New Person("Ada", "Lovelace", 28)
        Console.WriteLine(p1.FullName)          ' Ada Lovelace
        Console.WriteLine(p1.IsAdult())         ' True

        ' Object initializer (อ่านง่ายเวลาเซ็ตหลายพร็อพเพอร์ตี)
        Dim p2 As New Person With {
            .FirstName = "Alan",
            .LastName = "Turing",
            .Age = 23
        }

        ' ใช้เมธอดระดับคลาส (Shared) เพื่อสร้างค่าเริ่มต้นมาตรฐาน
        Dim p3 = Person.CreateDefaultAdult("Grace", "Hopper")

        ' แก้สถานะผ่านเมธอด (encapsulation)
        p2.Rename("Alan M.", "Turing")

        Console.WriteLine(p2.Greet("Developer"))
        ' Output: Hello Developer, I'm Alan M. Turing.
    End Sub
End Module
```

หมายเหตุด้านอายุการใช้งานออบเจ็กต์: .NET ใช้ **Garbage Collector** จัดการหน่วยความจำโดยอัตโนมัติ; หากออบเจ็กต์ถือทรัพยากรไม่จัดการ (เช่นไฟล์/เครือข่าย) ให้รองรับ IDisposable และใช้งานผ่าน Using ... End Using

ตัวอย่างคลาสเชิงปฏิบัติ: BankAccount

สาธิต Fields/Properties/Methods/Constructors/Validation/Shared Members อย่างครบถ้วน

```
' File: BankAccount.vb
Option Strict On
Option Infer On
Option Explicit On

Public Class BankAccount
    ' ===== Fields =====
    Private Shared _accountsCreated As Integer = 0   ' ตัวนับระดับคลาส
    Private ReadOnly _number As String               ' หมายเลขบัญชี (immutable)
    Private _balance As Decimal

    ' ===== Properties =====
    Public ReadOnly Property Number As String
        Get
            Return _number
        End Get
    End Property

    Public ReadOnly Property Balance As Decimal
        Get
            Return _balance
        End Get
    End Property

    ' ตัวอย่าง Property ที่มีการตรวจสอบค่าใน set (ผ่านเมธอด Deposit/Withdraw แทนการ set ตรง)
    Public Property OwnerName As String

    ' Shared (static) property
    Public Shared ReadOnly Property AccountsCreated As Integer
        Get
            Return _accountsCreated
```

```
        End Get
    End Property

    ' ===== Constructors =====
    Public Sub New(number As String, ownerName As String, Optional opening As Decimal =
0D)
        If String.IsNullOrWhiteSpace(number) Then Throw New ArgumentException("number
required")
        If String.IsNullOrWhiteSpace(ownerName) Then Throw New
ArgumentException("ownerName required")
        If opening < 0D Then Throw New ArgumentOutOfRangeException(NameOf(opening),
"opening must be >= 0")

        _number = number
        OwnerName = ownerName
        _balance = opening

        Threading.Interlocked.Increment(_accountsCreated)
    End Sub

    ' ===== Methods =====
    Public Sub Deposit(amount As Decimal, Optional note As String = Nothing)
        If amount <= 0D Then Throw New ArgumentOutOfRangeException(NameOf(amount),
"amount must be > 0")
        _balance += amount
        ' บันทึก log/โน้ตได้ตามต้องการ
    End Sub

    Public Sub Withdraw(amount As Decimal)
        If amount <= 0D Then Throw New ArgumentOutOfRangeException(NameOf(amount))
        If amount > _balance Then Throw New InvalidOperationException("Insufficient funds")
        _balance -= amount
    End Sub
```

```
    Public Overrides Function ToString() As String
        Return $"#{Number} ({OwnerName}) Balance={Balance:n2}"
    End Function
End Class
```

การใช้งาน:

```
Module Program
    Sub Main()
        Dim acc = New BankAccount("001-0001", "Chouvalit", opening:=1000D)
        acc.Deposit(250D)
        acc.Withdraw(100D)

        Console.WriteLine(acc) ' => #001-0001 (Chouvalit) Balance=1,150.00
        Console.WriteLine($"Accounts created: {BankAccount.AccountsCreated}")
    End Sub
End Module
```

คำสำคัญ/ตัวปรับแต่งการเข้าถึง (Access Modifiers)

- Public เข้าถึงได้ทุกที่
- Friend เข้าถึงได้ภายใน Assembly เดียวกัน
- Protected เข้าถึงได้ในคลาสและคลาสลูก
- Private ใช้ได้เฉพาะภายในคลาส
- Protected Friend (หรือ Private Protected ใน VB รุ่นใหม่) ผสมข้อจำกัดสำหรับสถานการณ์เฉพาะ

แนวทาง: ค่าเริ่มต้นควร “ปิดที่สุดเท่าที่ทำได้” แล้วค่อยเปิดเมื่อจำเป็น เพื่อความปลอดภัยและการบำรุงรักษา

เคล็ดลับเชิงสถาปัตยกรรม (แม้ยังอยู่ในระดับพื้นฐาน)

- ใช้ **Properties** แทนการเปิด **Fields** ตรง ๆ เพื่อรักษา invariant
- แยก **เมธอดที่เปลี่ยนสถานะ** ออกจาก **เมธอดคำนวณค่า**
- ใช้ Shared เฉพาะเมื่อเป็น “ข้อมูล/พฤติกรรมระดับคลาส” จริง ๆ
- เมื่อคอนสตรัคเตอร์เริ่มซับซ้อน พิจารณา **Factory Method** หรือ **Builder**
- สำหรับออบเจ็กต์ที่ควรเปลี่ยนค่าไม่ได้ ให้ใช้ **ReadOnly fields + Private Set** หรือออกแบบให้ **Immutable**

สรุปสั้น ๆ

- **Class** กำหนดโครงสร้างและพฤติกรรม; **Object** คืออินสแตนซ์จริง
- **Fields** เก็บสถานะจริง ⟶ ซ่อนด้วย Private และโชว์ผ่าน **Properties**
- **Properties** เป็นจุดตรวจสอบ/ควบคุมข้อมูลเข้าออก + ทำให้ API ของคลาสมั่นคง
- **Methods** นิยามพฤติกรรม, รองรับ Overload/Optional/ByRef/Shared
- **Constructors** กำหนดสถานะเริ่มต้น; ใช้ Me.New(...) ทำ chaining, Shared Sub New สำหรับเตรียมระดับคลาส
- **การสร้าง Instance** ใช้ New, รองรับ Object Initializer, จัดการอายุการใช้งานด้วย GC/Using เมื่อมีทรัพยากรไม่จัดการ

ถ้าต้องการ ผมสามารถ:

- ทดสอบ/รันตัวอย่างแบบ Console ครบไฟล์ (โครงสร้างโครงการ + .vb แต่ละไฟล์)
- แปลงตัวอย่างเดียวกันเป็น Windows Forms/WPF (มีฟอร์มผูก Properties)
- เพิ่มหัวข้อสืบทอดคลาส (Inheritance), Polymorphism (Overridable/Overrides), Interface (Implements) ต่อไปครับ

ดีมากครับ □

รายละเอียดเชิงลึก (Deep Dive) ของ บทที่ 7: Object-Oriented Programming (OOP) Basics ใน VB.NET

ผมจะลงไปในระดับ **Concept ⟶ Technical Implementation ⟶ Best Practices ⟶ Pitfalls ⟶ Advanced Notes** เพื่อให้คุณเข้าใจทั้ง **ทฤษฎีและการใช้งานจริง**

บทที่ 7: Object-Oriented Programming (OOP) Basics (เชิงลึก)

1) Class และ Object

แนวคิดเชิงลึก

- **Class** = แบบพิมพ์เขียว (Blueprint) ⟶ กำหนดว่า Object จะ “มีอะไร” (Fields/Properties) และ “ทำอะไรได้” (Methods)
- **Object** = อินสแตนซ์ของคลาส ⟶ มี **state** (ค่าที่เก็บอยู่) และ **behavior** (พฤติกรรมที่กระทำได้)

VB.NET เป็น **ภาษา OOP แบบ Reference Type**:

- Object อยู่ใน **Heap** (หน่วยความจำสำหรับ reference type)
- ตัวแปร (เช่น Dim p As Person) จะเก็บเพียง **reference** (address) ที่ชี้ไปยัง Object ใน Heap

- เมื่อ Object ไม่มี reference ที่ชี้ถึงอีก ⟶ GC (Garbage Collector) จะคืนหน่วยความจำอัตโนมัติ

☐ ตัวอย่าง:

```
Dim p1 As New Person("Ada", "Lovelace", 28)
Dim p2 As Person = p1     ' p2 ชี้ไปยัง object เดียวกับ p1
p2.Age = 30
Console.WriteLine(p1.Age)   ' Output: 30 (เพราะชี้ object เดียวกัน)
```

หมายเหตุ: ถ้าต้องการ “clone” object ควรใช้การทำ **Deep Copy/Prototype Pattern** ไม่ใช่การ assign reference

---

2) Fields, Properties, Methods

Fields (รายละเอียดเชิงลึก)

- เก็บข้อมูล **จริง** ของ object
- มักใช้ **Private** หรือ **Protected** เพื่อบังคับให้เข้าผ่าน **Properties**
- **Static (Shared) Field** ⟶ ค่าถูกแชร์ข้ามทุกอินสแตนซ์

☐ ตัวอย่าง Pitfall:

```
Public Class Counter
    Public Shared Count As Integer = 0
    Public Sub New()
        Count += 1
    End Sub
End Class

Dim c1 As New Counter()
Dim c2 As New Counter()
Console.WriteLine(Counter.Count)  ' Output: 2 (เพราะเป็น Shared)
```

Properties (รายละเอียดเชิงลึก)

- **Encapsulation**: ป้องกันไม่ให้เปลี่ยนค่าภายในตรง ๆ
- สามารถ:
    - **Validate input** (ตรวจสอบก่อนกำหนดค่า)
    - **Compute value** (คำนวณทุกครั้งที่เรียกใช้)
    - **Private Set** ⟶ อนุญาตให้อ่านได้จากภายนอก แต่เขียนได้เฉพาะภายในคลาส

☐ ตัวอย่าง Property แบบ Advanced:

```
Public Property Salary As Decimal
    Get
        Return _salary
    End Get
    Private Set(value As Decimal)
        If value < 0 Then
            Throw New ArgumentException("Salary cannot be negative.")
        End If
        _salary = value
    End Set
End Property
```

แนวทางที่ถูกต้อง (Best Practice):

- ไม่เปิด field ตรง ๆ (Public Field) เพราะไม่สามารถควบคุม invariant
- ใช้ Property แทนเสมอ

Methods (รายละเอียดเชิงลึก)

- **Instance Method** ⟶ ทำงานบน object ที่สร้างแล้ว
- **Shared Method** ⟶ ทำงานในระดับ class (เหมือน static ใน C#)
- **Overloading** ⟶ ใช้ชื่อ method เดียว แต่ parameter ต่างกัน
- **ByVal vs ByRef**:
    - ByVal (ค่า default) ⟶ ส่งสำเนาของค่า (ถ้าเป็น value type เช่น Integer จะ copy ค่า , ถ้าเป็น reference type จะ copy reference)
    - ByRef ⟶ ส่งตัวแปร reference โดยตรง ⟶ เมธอดสามารถแก้ค่าจริงได้

 ตัวอย่าง:

```
Public Sub Increment(ByRef number As Integer)
    number += 1
End Sub

Dim x As Integer = 5
Increment(x)
Console.WriteLine(x)  ' Output: 6
```

---

3) Constructors (รายละเอียดเชิงลึก)

คุณสมบัติ

- Sub New(...) ⟶ เรียกอัตโนมัติเมื่อ New object
- สนับสนุน:
  - **Overloading** (หลาย constructor ใน class เดียว)
  - **Constructor Chaining** (Me.New(...)) ⟶ ลด duplication
  - **Calling Base Constructor** (MyBase.New(...)) ⟶ ใช้กรณี inheritance

☐ ตัวอย่าง Constructor Chaining:

```
Public Class Student
    Public Property Name As String
    Public Property Age As Integer

    Public Sub New()
        Me.New("Unknown", 0)   ' เรียก constructor หลัก
    End Sub

    Public Sub New(name As String, age As Integer)
        Me.Name = name
        Me.Age = age
    End Sub
End Class
```

Shared Constructor (รายละเอียดเชิงลึก)

- Shared Sub New() ⟶ เรียกเพียงครั้งเดียวเมื่อ class ถูกโหลดครั้งแรกใน AppDomain
- ใช้สำหรับ **ค่า default ระดับ class**, **Cache**, **Configuration**

☐ ตัวอย่าง:

```
Public Class Config
    Public Shared ReadOnly Setting As String
    Shared Sub New()
        Setting = "Loaded once"
    End Sub
End Class
```

---

4) การสร้าง Instance

แบบปกติ

```
Dim p As New Person("Alan", "Turing", 23)
```

Object Initializer

```
Dim p As New Person With {
    .FirstName = "Alan",
    .LastName = "Turing",
    .Age = 23
}
```

Factory Method (Best Practice)

แทนที่จะสร้าง instance โดยตรง ⟶ ใช้เมธอด static เพื่อควบคุม invariant

```
Public Shared Function CreateAdult(name As String) As Person
    Return New Person(name, "Unknown", 18)
End Function
```

Best Practices (VB.NET OOP)

1. **Encapsulation** ⟶ ใช้ Private fields + Public Properties
2. **Constructor Validation** ⟶ ตรวจสอบค่า input ตั้งแต่ต้น
3. **Immutability (ถ้าเป็นไปได้)** ⟶ ใช้ ReadOnly fields และ Private Set
4. **Separation of Concerns** ⟶ แยก logic ธุรกิจออกจาก data model
5. **Use Factory/Builder** ⟶ ถ้า constructor มี parameter เยอะ
6. **Dispose Pattern** ⟶ ถ้า class ใช้ resource ภายนอก (File, DB Connection)

Pitfalls ที่พบบ่อย

- ใช้ Public Field ⟶ ทำให้ข้อมูลภายในเปลี่ยนได้โดยตรง (ผิดหลัก encapsulation)
- ใช้ constructor หลายตัวโดยไม่ chain ⟶ โค้ดซ้ำซ้อน
- ลืมใช้ Shared Sub New() ⟶ โหลด config/reusable data ซ้ำหลายครั้ง
- สับสน ByVal กับ ByRef เมื่อใช้ reference type (เช่น Object, List)
- สร้าง object แต่ไม่ implement IDisposable เมื่อจัดการทรัพยากร (เช่น DB Connection ⟶ memory leak)

Advanced Notes (เชื่อมโยงไปบทถัดไป)

- **Inheritance (การสืบทอด)**: ใช้ Inherits เพื่อ reuse code
- **Polymorphism**: ใช้ Overridable และ Overrides
- **Interfaces**: ใช้ Implements เพื่อกำหนดสัญญา (contract)
- **Abstract Class**: ใช้ MustInherit สำหรับ class ที่ไม่ควรถูกสร้างตรง ๆ

☐ สรุปเชิงลึก

ใน VB.NET **OOP Basics**:

- **Class** = แบบพิมพ์เขียว, **Object** = อินสแตนซ์จริง
- ใช้ **Fields** ⟶ **Properties** ⟶ **Methods** อย่างมีชั้นเชิง
- ควบคุม **การสร้าง object** ด้วย **Constructors / Factory Methods**
- ใช้ Encapsulation, Validation, Shared Members, ByVal vs ByRef อย่างเหมาะสม
- วางรากฐานเพื่อเข้าสู่หัวข้อ **Inheritance, Polymorphism, Interfaces** ในบทถัดไป

Class และ Object

☐ **Class และ Object ใน VB.NET**

1) แนวคิดพื้นฐาน

- **Class** = "แบบพิมพ์เขียว" (Blueprint) หรือ "แม่พิมพ์" สำหรับการสร้างออบเจ็กต์
  - กำหนดว่าออบเจ็กต์นั้น "มีข้อมูลอะไร" (**Fields/Properties**)
  - กำหนดว่าออบเจ็กต์นั้น "ทำอะไรได้" (**Methods**)
- **Object (Instance)** = สิ่งที่ถูกสร้างขึ้นจริงจาก Class
  - แต่ละออบเจ็กต์มีสถานะ (state) เป็นของตัวเอง
  - แต่ใช้โครงสร้างร่วมกันจาก Class

☐ พูดง่าย ๆ ⟶ **Class = แผนที่ / Object = บ้านจริง ๆ ที่สร้างขึ้นจากแผนที่**

2) การประกาศ Class ใน VB.NET

โครงสร้างพื้นฐาน:

```
Public Class Person
    ' === Fields ===
    Private _name As String

    ' === Property ===
    Public Property Name As String
        Get
            Return _name
        End Get
        Set(value As String)
```

```
            _name = value
        End Set
    End Property

    ' === Method ===
    Public Sub SayHello()
        Console.WriteLine($"Hello, my name is {Name}.")
    End Sub
End Class
```

- Public Class ... End Class ⟶ นิยามคลาส
- Private / Public ⟶ ควบคุมการเข้าถึงข้อมูล
- **Fields** ⟶ ตัวแปรเก็บค่าภายใน (state)
- **Properties** ⟶ ตัวกลางสำหรับอ่าน/เขียนข้อมูล (encapsulation)
- **Methods** ⟶ พฤติกรรม/การทำงาน

---

3) การสร้าง Object (Instance)

เมื่อมี Class แล้ว สามารถสร้าง Object ได้ด้วย New

```
Module Program
    Sub Main()
        ' สร้าง Object จาก Class
        Dim p1 As New Person()
        p1.Name = "Ada Lovelace"
        p1.SayHello()
        ' Output: Hello, my name is Ada Lovelace.

        ' สร้างอีก Object
        Dim p2 As New Person()
        p2.Name = "Alan Turing"
        p2.SayHello()
        ' Output: Hello, my name is Alan Turing.
    End Sub
End Module
```

□ จุดสำคัญ ⟶ p1 และ p2 คือ **Object คนละตัว** ถึงแม้จะมาจาก Class เดียวกัน แต่มี state ของตัวเอง

4) ความสัมพันธ์ระหว่าง Class และ Object

- Class = กำหนดว่า Object มีอะไรบ้าง
- Object = อินสแตนซ์จริง ๆ ที่ใช้ในโปรแกรม
- Object แต่ละตัวมี state เป็นของตนเอง
- แต่ใช้ code เดียวกันจาก Class

□ ภาพจำลอง (เชิงแนวคิด):

Class Person
├── Field: Name
└── Method: SayHello()

Object A (จาก Person)
└── Name = "Ada Lovelace"

Object B (จาก Person)
└── Name = "Alan Turing"

5) เทคนิคเชิงลึก

1. **Reference Type**
    - Class ใน VB.NET เป็น reference type ⟶ ตัวแปร object เก็บเพียง "reference" (ตำแหน่งหน่วยความจำ)
    - การ assign object จะ copy reference ไม่ใช่ copy ข้อมูล
2. Dim p1 As New Person()
3. p1.Name = "Grace Hopper"
4.
5. Dim p2 As Person = p1
6. p2.Name = "Changed Name"
7.
8. Console.WriteLine(p1.Name) ' Output: Changed Name (เพราะ p1, p2 ชี้ object เดียวกัน)
9. **Object Lifetime**
    - เมื่อไม่มี reference ใดชี้ object ⟶ GC (Garbage Collector) จะเก็บ object นั้นไปโดยอัตโนมัติ
    - ไม่ต้อง delete ด้วยตัวเอง (เหมือนภาษา C/C++)
10. **Anonymous Object (VB.NET 9.0+)**

- VB.NET ยังสามารถสร้าง **object แบบไม่มีคลาสที่ชัดเจน** ได้ (Anonymous Type)

```
11. Dim anon = New With {Key .Id = 1, .Name = "VB.NET"}
12. Console.WriteLine(anon.Name)  ' VB.NET
```

---

6) Best Practices

☐ ใช้ **Private Fields + Public Properties** ⟶ เพื่อบังคับการเข้าถึง

☐ ตั้งชื่อ Class ⟶ PascalCase (เช่น BankAccount, Person)

☐ ใช้ Option Strict On และ Option Explicit On เพื่อป้องกันการ cast ผิดประเภท

☐ ใช้ Constructor (Sub New) สำหรับการตั้งค่าเริ่มต้นของ object

---

7) Pitfalls (ข้อผิดพลาดที่พบบ่อย)

☐ ใช้ Public Field ตรง ๆ (ละเมิด Encapsulation)

☐ สับสนระหว่าง p1 = p2 (assign reference ⟶ object ชี้ตัวเดียวกัน) กับ “copy object” (ต้อง implement clone เอง)

☐ ลืมใช้ New ⟶ ทำให้ตัวแปร object เป็น Nothing แล้วเรียกใช้ ⟶ เกิด NullReferenceException

```
Dim p As Person
p.SayHello() ' ☐ Error: Object reference not set to an instance of an object.
```

☐ **สรุป**

- **Class** = แบบพิมพ์เขียว ⟶ กำหนดโครงสร้างและพฤติกรรม
- **Object** = สิ่งที่สร้างจาก Class ⟶ มี state ของตัวเอง
- Class อยู่ที่ระดับ design-time / code, Object เกิดขึ้นจริงตอน runtime
- VB.NET ใช้ **reference type** ⟶ ต้องเข้าใจเรื่อง reference และ object lifetime

---

ตัวอย่างโปรแกรม VB.NET เรื่อง Class และ Object แบบ **เต็มไฟล์ + โครงสร้าง + คำอธิบายโค้ด + ผลการรัน**

แบ่งเป็น 2 ส่วน:

- **โปรแกรมพื้นฐาน 3 โปรแกรม** ⟶ เน้นโครงสร้าง Class + Object (Basic OOP)
- **โปรแกรมแนวประยุกต์ 3 โปรแกรม** ⟶ ใช้ Class + Object ในบริบทจริง

☐ **ส่วนที่ 1: โปรแกรมพื้นฐาน (Basic)**

☐ **โปรแกรมที่ 1: Person Class**

โครงสร้างไฟล์

```
VBNetOOP_Basic1/
  ├── Person.vb
  └── Program.vb
```

Person.vb

```
Option Strict On
Option Explicit On

Public Class Person
    Public Property Name As String
    Public Property Age As Integer

    Public Sub SayHello()
        Console.WriteLine($"Hello, my name is {Name}, I'm {Age} years old.")
    End Sub
End Class
```

Program.vb

```
Module Program
    Sub Main()
        Dim p1 As New Person()
        p1.Name = "Ada Lovelace"
        p1.Age = 28
        p1.SayHello()

        Dim p2 As New Person()
        p2.Name = "Alan Turing"
        p2.Age = 23
        p2.SayHello()
    End Sub
End Module
```

ผลการรัน

```
Hello, my name is Ada Lovelace, I'm 28 years old.
Hello, my name is Alan Turing, I'm 23 years old.
```

□ โปรแกรมที่ 2: Rectangle Class

โครงสร้างไฟล์

```
VBNetOOP_Basic2/
├── Rectangle.vb
└── Program.vb
```

Rectangle.vb

```
Option Strict On
Option Explicit On

Public Class Rectangle
    Public Property Width As Double
    Public Property Height As Double

    Public Function Area() As Double
        Return Width * Height
    End Function

    Public Function Perimeter() As Double
        Return 2 * (Width + Height)
    End Function
End Class
```

Program.vb

```
Module Program
    Sub Main()
        Dim rect As New Rectangle()
        rect.Width = 5
        rect.Height = 3

        Console.WriteLine($"Width={rect.Width}, Height={rect.Height}")
        Console.WriteLine($"Area={rect.Area()}, Perimeter={rect.Perimeter()}")
    End Sub
End Module
```

ผลการรัน

```
Width=5, Height=3
Area=15, Perimeter=16
```

□ โปรแกรมที่ 3: Student Class with Constructor

โครงสร้างไฟล์

```
VBNetOOP_Basic3/
├── Student.vb
└── Program.vb
```

Student.vb

```
Option Strict On
Option Explicit On

Public Class Student
    Public Property Id As Integer
    Public Property Name As String
    Public Property Score As Double

    ' Constructor
    Public Sub New(id As Integer, name As String, score As Double)
        Me.Id = id
        Me.Name = name
        Me.Score = score
    End Sub

    Public Function IsPassed() As Boolean
        Return Score >= 50
    End Function
End Class
```

Program.vb

```
Module Program
    Sub Main()
        Dim s1 As New Student(1, "Alice", 75)
        Dim s2 As New Student(2, "Bob", 45)

        Console.WriteLine($"{s1.Name} Passed? {s1.IsPassed()}")
        Console.WriteLine($"{s2.Name} Passed? {s2.IsPassed()}")
```

```
    End Sub
End Module
```

ผลการรัน

```
Alice Passed? True
Bob Passed? False
```

☐ ส่วนที่ 2: โปรแกรมแนวประยุกต์ (Applied)

☐ โปรแกรมที่ 4: Bank Account System

โครงสร้างไฟล์

```
VBNetOOP_App1/
├── BankAccount.vb
└── Program.vb
```

BankAccount.vb

```
Option Strict On
Option Explicit On

Public Class BankAccount
    Public Property AccountNumber As String
    Public Property Owner As String
    Private _balance As Decimal

    Public Sub New(number As String, owner As String, Optional balance As Decimal = 0)
        AccountNumber = number
        Me.Owner = owner
        _balance = balance
    End Sub

    Public Sub Deposit(amount As Decimal)
        If amount <= 0 Then Throw New ArgumentException("Invalid amount")
        _balance += amount
    End Sub

    Public Sub Withdraw(amount As Decimal)
        If amount > _balance Then Throw New InvalidOperationException("Insufficient funds")
```

```
        _balance -= amount
    End Sub

    Public ReadOnly Property Balance As Decimal
        Get
            Return _balance
        End Get
    End Property

    Public Overrides Function ToString() As String
        Return $"{AccountNumber} - {Owner}, Balance: {_balance:C}"
    End Function
End Class
```

Program.vb

```
Module Program
    Sub Main()
        Dim acc As New BankAccount("001-1001", "Chouvalit", 1000D)
        acc.Deposit(500D)
        acc.Withdraw(200D)

        Console.WriteLine(acc.ToString())
    End Sub
End Module
```

ผลการรัน

```
001-1001 - Chouvalit, Balance: ฿1,300.00
```

□ โปรแกรมที่ 5: Library Book Management

โครงสร้างไฟล์

```
VBNetOOP_App2/
├── Book.vb
└── Program.vb
```

Book.vb

```
Option Strict On
Option Explicit On
```

```
Public Class Book
    Public Property Title As String
    Public Property Author As String
    Public Property IsBorrowed As Boolean

    Public Sub New(title As String, author As String)
        Me.Title = title
        Me.Author = author
        Me.IsBorrowed = False
    End Sub

    Public Sub Borrow()
        If IsBorrowed Then
            Console.WriteLine($"'{Title}' is already borrowed.")
        Else
            IsBorrowed = True
            Console.WriteLine($"You borrowed '{Title}'.")
        End If
    End Sub

    Public Sub ReturnBook()
        If Not IsBorrowed Then
            Console.WriteLine($"'{Title}' was not borrowed.")
        Else
            IsBorrowed = False
            Console.WriteLine($"You returned '{Title}'.")
        End If
    End Sub
End Class
```

Program.vb

```
Module Program
    Sub Main()
        Dim b1 As New Book("Clean Code", "Robert C. Martin")
```

```
        b1.Borrow()
        b1.Borrow()
        b1.ReturnBook()
    End Sub
End Module
```

ผลการรัน

```
You borrowed 'Clean Code'.
'Clean Code' is already borrowed.
You returned 'Clean Code'.
```

□ โปรแกรมที่ 6: E-Commerce Shopping Cart

โครงสร้างไฟล์

```
VBNetOOP_App3/
  ├── Product.vb
  ├── Cart.vb
  └── Program.vb
```

Product.vb

```
Option Strict On
Option Explicit On

Public Class Product
    Public Property Name As String
    Public Property Price As Decimal

    Public Sub New(name As String, price As Decimal)
        Me.Name = name
        Me.Price = price
    End Sub
End Class
```

Cart.vb

```
Option Strict On
Option Explicit On
Imports System.Collections.Generic
```

```
Public Class Cart
    Private items As New List(Of Product)

    Public Sub AddItem(p As Product)
        items.Add(p)
    End Sub

    Public Function Total() As Decimal
        Dim sum As Decimal = 0
        For Each p In items
            sum += p.Price
        Next
        Return sum
    End Function

    Public Sub ShowItems()
        For Each p In items
            Console.WriteLine($"{p.Name} - {p.Price:C}")
        Next
    End Sub
End Class
```

Program.vb

```
Module Program
    Sub Main()
        Dim cart As New Cart()
        cart.AddItem(New Product("Laptop", 25000D))
        cart.AddItem(New Product("Mouse", 500D))
        cart.AddItem(New Product("Keyboard", 1200D))

        Console.WriteLine("Items in cart:")
        cart.ShowItems()
        Console.WriteLine($"Total = {cart.Total():C}")
    End Sub
End Module
```

ผลการรัน

Items in cart:

Laptop - ฿25,000.00

Mouse - ฿500.00

Keyboard - ฿1,200.00

Total = ฿26,700.00

☐ **สรุป**

- **3 โปรแกรมพื้นฐาน** ⟶ สอนการใช้ Class + Object เบื้องต้น (Person, Rectangle, Student)
- **3 โปรแกรมประยุกต์** ⟶ ใช้ OOP ในงานจริง (BankAccount, Library Book, Shopping Cart)
- ครอบคลุม Field, Property, Method, Constructor, Instance, Encapsulation

---

Properties, Methods, Fields

☐ **อธิบายเชิงลึก**

1. Fields

- คือ **ตัวแปรภายในคลาส** (เก็บสถานะของ Object)
- มักประกาศเป็น Private เพื่อซ่อนรายละเอียด (Encapsulation)
- ตัวอย่าง:
- Private name As String
- Private age As Integer

2. Properties

- คือ **ตัวกลางสำหรับเข้าถึง Fields**
- มี Get (ดึงค่า) และ Set (กำหนดค่า)
- ใช้เพื่อควบคุมข้อมูล เช่น ตรวจสอบเงื่อนไขก่อนกำหนดค่า
- ตัวอย่าง:
- Public Property Age As Integer
- Get
- Return age
- End Get
- Set(value As Integer)
- If value >= 0 Then

- age = value
- End If
- End Set
- End Property

3. Methods

- คือ **ฟังก์ชันหรือพฤติกรรมของ Object**
- ใช้ประมวลผล / ทำงานกับข้อมูลใน Fields/Properties
- ตัวอย่าง:
- Public Sub ShowInfo()
- Console.WriteLine($"Name: {name}, Age: {age}")
- End Sub

☐ ตัวอย่างโปรแกรมพื้นฐาน (3 โปรแกรม)

☐ โปรแกรมที่ 1: Fields + Properties เบื้องต้น

โครงสร้างไฟล์

```
OOP_Basics1/
└── Module1.vb
```

Module1.vb

```
Module Module1
    Public Class Person
        ' Field
        Private name As String

        ' Property
        Public Property NameProperty As String
            Get
                Return name
            End Get
            Set(value As String)
                name = value
            End Set
        End Property
    End Class
```

```
    Sub Main()
        Dim p As New Person()
        p.NameProperty = "Alice"
        Console.WriteLine("Name: " & p.NameProperty)
    End Sub
End Module
```

ผลการรัน

```
Name: Alice
```

□ โปรแกรมที่ 2: Fields + Properties + Methods

โครงสร้างไฟล์

```
OOP_Basics2/
└── Module1.vb
```

Module1.vb

```
Module Module1
    Public Class Rectangle
        ' Fields
        Private width As Double
        Private height As Double

        ' Properties
        Public Property Width As Double
            Get
                Return width
            End Get
            Set(value As Double)
                If value > 0 Then width = value
            End Set
        End Property

        Public Property Height As Double
            Get
                Return height
```

```
            End Get
            Set(value As Double)
                If value > 0 Then height = value
            End Set
        End Property

        ' Method
        Public Function Area() As Double
            Return width * height
        End Function
    End Class

    Sub Main()
        Dim rect As New Rectangle()
        rect.Width = 5
        rect.Height = 10
        Console.WriteLine("Area: " & rect.Area())
    End Sub
End Module
```

ผลการรัน

```
Area: 50
```

□ โปรแกรมที่ 3: Fields Private + Encapsulation ผ่าน Property

โครงสร้างไฟล์

```
OOP_Basics3/
└── Module1.vb
```

Module1.vb

```
Module Module1
    Public Class BankAccount
        Private balance As Decimal

        ' Property มีการตรวจสอบ
        Public Property Balance As Decimal
            Get
```

```
            Return balance
        End Get
        Set(value As Decimal)
            If value >= 0 Then
                balance = value
            Else
                Console.WriteLine("Balance cannot be negative!")
            End If
        End Set
    End Property

    ' Method ฝากเงิน
    Public Sub Deposit(amount As Decimal)
        Balance += amount
    End Sub

    ' Method ถอนเงิน
    Public Sub Withdraw(amount As Decimal)
        If Balance >= amount Then
            Balance -= amount
        Else
            Console.WriteLine("Insufficient funds!")
        End If
    End Sub
End Class

Sub Main()
    Dim acc As New BankAccount()
    acc.Deposit(1000)
    acc.Withdraw(300)
    Console.WriteLine("Balance: " & acc.Balance)

    acc.Withdraw(1000)
End Sub
```

```
End Module
```

ผลการรัน

```
Balance: 700
Insufficient funds!
```

☐ ตัวอย่างโปรแกรมแนวประยุกต์ (3 โปรแกรม)

☐ โปรแกรมที่ 4: Student Class + GPA Calculator

```
OOP_Student/
└── Module1.vb
```

```
Module Module1
    Public Class Student
        Private name As String
        Private scores As New List(Of Integer)

        Public Property NameProperty As String
            Get
                Return name
            End Get
            Set(value As String)
                name = value
            End Set
        End Property

        ' Method เพิ่มคะแนน
        Public Sub AddScore(score As Integer)
            scores.Add(score)
        End Sub

        ' Method คำนวณ GPA
        Public Function GetAverage() As Double
            If scores.Count = 0 Then Return 0
            Return scores.Average()
        End Function
```