Unit III-Object-Oriented Programming in VB.Net:

Introduction

A programming paradigm based on the concept of "objects", which can contain data in the form of fields (attributes) and code in the form of procedures (methods). OOP in VB.NET allows developers to create modular, reusable, and scalable code by organizing it into classes and objects.

Sub Procedures and Functions:

- Sub Procedures: Sub procedures, also known as methods, are blocks of code in VB.NET that perform a specific task. They can be invoked (called) from other parts of the program and can accept parameters.

- Functions: Functions are similar to sub procedures but return a value after performing a specific task. They are defined using the `Function` keyword and have a return type specified in the declaration.

Code

```
Public Class Calculator
    Public Sub Add(num1 As Integer, num2 As Integer)
        Console.WriteLine("Sum: " & (num1 + num2))
    End Sub

    Public Function Multiply(num1 As Integer, num2 As Integer) As Integer
        Return num1 * num2
    End Function
End Class

Module Program
    Sub Main()
        Dim calc As New Calculator()
        calc.Add(5, 3)
        Dim product As Integer = calc.Multiply(4, 2)
        Console.WriteLine("Product: " & product)
    End Sub
End Module
```

Classes and Objects in VB.Net:

- Classes: In VB.NET, a class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of the class will have. Classes encapsulate data and provide methods to operate on that data.

- Objects: Objects are instances of classes. They represent real-world entities and are created using the `New` keyword followed by the class name. Each object has its own set of properties and can invoke methods defined in its class.

Code

```
' Define a class named Car
Public Class Car
    ' Define properties
    Public Property Model As String
    Public Property Year As Integer

    ' Constructor
    Public Sub New(model As String, year As Integer)
        Me.Model = model
        Me.Year = year
    End Sub
End Class

' Create an object of the Car class
Module Program
    Sub Main()
        ' Instantiate a Car object using Constructor
        Dim myCar As New Car("Toyota", 2020)
        ' Access properties
        Console.WriteLine("Model: " & myCar.Model & ", Year: " & myCar.Year)
    End Sub
End Module
```

Constructors and Destructors in VB.Net:

- Constructors: Constructors are special methods in VB.NET classes that are called automatically when an object of the class is created. They initialize the object's state, setting values to its properties or performing other initialization tasks.

- Destructors: In VB.NET, destructors are called finalizers. They are special methods used for releasing resources held by an object before it is destroyed by the garbage collector. Finalizers are invoked automatically when the object is being garbage-collected.

code

```
' Define a class with Constructor and Destructor
Public Class MyClass
    ' Constructor
    Public Sub New()
        Console.WriteLine("Constructor called.")
    End Sub

    ' Destructor
    Protected Overrides Sub Finalize()
        Console.WriteLine("Destructor called.")
        MyBase.Finalize()
    End Sub
End Class

' Create an object of the MyClass class
Module Program
    Sub Main()
        ' Instantiate a MyClass object
        Dim obj As New MyClass()
        ' Some operations...
        obj = Nothing ' Manually releasing the object
        GC.Collect() ' Trigger garbage collection
    End Sub
End Module
```

Inheritance in VB.Net, Simple Inheritance using Override Keyword:

- Inheritance: Inheritance is a fundamental concept in OOP that allows a class (subclass or derived class) to inherit properties and methods from another class (superclass or base class). In VB.NET, classes can inherit from other classes using the `Inherits` keyword.

- Override Keyword: The `Override` keyword in VB.NET is used to provide a new implementation for a method or property that is inherited from a base class. It allows subclasses to customize the behavior of inherited members.

```
' Define a base class Animal
Public Class Animal
    Public Overridable Sub Speak()
        Console.WriteLine("Animal speaks")
    End Sub
End Class

' Define a derived class Dog
Public Class Dog
    Inherits Animal
    Public Overrides Sub Speak()
        Console.WriteLine("Dog barks")
    End Sub
End Class

' Create objects and invoke methods
Module Program
    Sub Main()
        Dim animal As New Animal()
        Dim dog As New Dog()

        animal.Speak() ' Output: Animal speaks
        dog.Speak() ' Output: Dog barks
    End Sub
End Module
```

Overloading, Overriding, and Shadowing:

- Overloading: Overloading is the ability to define multiple methods or constructors in a class with the same name but different parameter lists. VB.NET allows method overloading, which provides flexibility and improves code readability.

- Overriding: Overriding is the process of providing a new implementation for a method or property that is inherited from a base class. It allows subclasses to modify or extend the behavior of inherited members using the `Overrides` keyword.

- Shadowing: Shadowing is a mechanism in VB.NET that allows a derived class to define a member with the same name as a member in the base class. This hides the base class member within the derived class, effectively shadowing it.

```
' Define a class with Overloading, Overriding, and Shadowing
Public Class MathOperations
    ' Overloaded method
    Public Sub Add(num1 As Integer, num2 As Integer)
        Console.WriteLine("Sum: " & (num1 + num2))
    End Sub

    Public Sub Add(num1 As Double, num2 As Double)
        Console.WriteLine("Sum: " & (num1 + num2))
    End Sub

    ' Method to be overridden
    Public Overridable Sub Display()
        Console.WriteLine("Displaying from base class.")
    End Sub
End Class

' Derived class overriding Display method
Public Class AdvancedMathOperations
    Inherits MathOperations
    ' Overriding base class method
    Public Overrides Sub Display()
        Console.WriteLine("Displaying from derived class.")
    End Sub
```

```
' Shadowing method
    Public Shadows Sub Add(num1 As Integer, num2 As Integer)
        Console.WriteLine("Shadowed Sum: " & (num1 + num2))
    End Sub
End Class

' Create objects and invoke methods
Module Program
    Sub Main()
        Dim math As New MathOperations()
        Dim advancedMath As New AdvancedMathOperations()

        math.Add(5, 3) ' Output: Sum: 8
        advancedMath.Add(5, 3) ' Output: Shadowed Sum: 8

        math.Display() ' Output: Displaying from base class.
        advancedMath.Display() ' Output: Displaying from derived class.
    End Sub
End Module
```

- Exception Handling:

Exception handling is a mechanism used to handle runtime errors and unexpected situations in a program gracefully. In VB.NET, exceptions are handled using the `Try`, `Catch`, `Finally`, and `Throw` keywords. Code that might raise an exception is placed inside a `Try` block, and potential exceptions are caught and handled in `Catch` blocks. The `Finally` block is used to clean up resources, and the `Throw` statement is used to raise custom exceptions.

```
' Define a method that may throw an exception
Public Sub Divide(num1 As Integer, num2 As Integer)
    Try
        Dim result As Integer = num1 \ num2
        Console.WriteLine("Result: " & result)
    Catch ex As DivideByZeroException
        Console.WriteLine("Error: Division by zero.")
    Finally
        Console.WriteLine("End of Divide method.")
    End Try
End Sub

' Invoke the method and handle exceptions
Module Program
    Sub Main()
        Divide(10, 2) ' Output: Result: 5, End of Divide method.
        Divide(5, 0) ' Output: Error: Division by zero., End of Divide method.
    End Sub
End Module
```