Chapter 7
Adding Responsibilities to Problem Domain Classes

Objectives
In this chapter, you will:
• Write a new problem domain class definition
• Create custom methods
• Write class variables and methods
• Write overloaded methods
• Work with exceptions

Writing a New Problem Domain Class Definition

• Slip class
  – Attributes
    • slipId
    • slipWidth
    • slipLength
  – Custom method
    • LeaseSlip

Writing a New Problem Domain Class Definition

• Class definition of Slip
  – Class header
    Public Class Slip
  – Attribute definition statements
    Private slipId As Integer
    Private slipWidth As Integer
    Private slipLength As Integer

Writing a New Problem Domain Class Definition

• Class definition of Slip (Continued)
  – Parameterized constructor
    • Enables attributes to be automatically populated whenever an instance is created
    Public Sub New(ByVal aSlipId As Integer, ByVal aSlipWidth As Integer, ByVal aSlipLength As Integer)
    • Argument data types must be assignment compatible with parameter data types
    SetSlipId(aSlipId)
    SetSlipWidth(aSlipWidth)
    SetSlipLength(aSlipLength)
Writing a New Problem Domain Class Definition

- Class definition of Slip (Continued)
  - TellAboutSelf method
    - Invokes the three getters
    - Concatenates the returned values
    - Returns the result
  
  ```vbnet
  Public Function TellAboutSelf() As String
    Dim info As String
    info = "Slip: Id = " & GetSlipId() & " & GetSlipWidth() & " & " & GetSlipLength()
    Return info
  End Function
  ```

Creating Custom Methods

- Custom methods
  - Process data
- Accessor methods
  - Store and retrieve attribute values
- LeaseSlip method
  - A custom method
  - Computes the lease fee for a slip
  - Fees range from $800 to $1,500 depending on the slip’s width

```vbnet
Public Function LeaseSlip() As Single
  Dim fee As Single
  Select Case slipWidth
    Case 10
      fee = 800
    Case 12
      fee = 900
    Case 14
      fee = 1100
    Case 16
      fee = 1500
    Case Else
      fee = 0
  End Select
  Return fee
End Function
```

Writing Class Variables and Methods

- Instance variables and methods
  - Each instance has a copy
- Class variables and methods
  - Shared by all instances of the class
  - Each instance does not have its own copy
  - Declared using the keyword Shared
Writing Class Variables and Methods

• Variable numberOfSlips can be used to track the total number of slips
• numberOfSlips variable
  – Initialized to 0
  – Private accessibility
    • Will be accessed only by methods within the Slip class
  
  Private Shared numberOfSlips As Integer = 0

Writing Class Variables and Methods

• Code added to the Slip constructor for the numberOfSlips variable
  'increment shared attribute
  numberOfSlips += 1
  – Increments numberOfSlips each time a slip instance is created

GetNumberOfSlips() method

'shared (class) method
Public Shared Function GetNumberOfSlips() As Integer
  Return numberOfSlips
End Function

Writing Overloaded Methods

• Method signature: Name and parameter list
• A class definition can contain several methods with the same name, as long as their parameter lists differ
  – VB .NET identifies a method by its signature, not only by its name
• Overloaded method: a method that has the same name as another method in the same class, but a different parameter list

Overloading a Constructor

• Problem
  – Most slips at Bradshaw Marina are 12 feet wide and 25 feet long, but a few have different widths and lengths
  – Current Slip constructor requires three arguments: slipId, slipWidth, and slipLength
Overloading a Constructor

• Solution
  – Write a second Slip constructor that has only a single parameter for slipId
  – Include statements to assign the default values of 12 and 25 to slipWidth and slipLength, respectively

• Constants can be used for the default width and length values:
  'constants
  Private Const DEFAULT_SLIP_WIDTH As Integer = 12
  Private Const DEFAULT_SLIP_LENGTH As Integer = 25

• Code for second constructor:
  '1-parameter constructor
  'Overloads keyword not used with constructors
  Public Sub New(ByVal aSlipId As Integer)
    'invoke 3-parameter constructor, passing default values
    Me.New(aSlipId, DEFAULT_SLIP_WIDTH, DEFAULT_SLIP_LENGTH)
  End Sub

• VB .NET determines which constructor to invoke by the argument list
  – If the argument consists of three values
    • Original constructor with three parameters is executed
  – If the argument consists of a single value
    • New constructor with one parameter is invoked

Overloading a Custom Method

• Problem
  – Bradshaw Marina permits a discounted lease fee under certain conditions

• Solution
  – A second version of the LeaseSlip method that accepts a value for the percentage discount
  • Will overload the original LeaseSlip method

• Two LeaseSlip methods in Slip, each with a different signature
  – Original LeaseSlip has an empty parameter list
  – Second LeaseSlip has a parameter variable named aDiscountPercent
Overloading a Custom Method

- Header for new method
  - Must contain the keyword Overloads

  ```vbnet
  Public Overloads Function LeaseSlip(ByVal aDiscountPercent As Single) As Single
  ```

Overloading a Custom Method

- New method:

  ```vbnet
  'overloaded custom method LeaseSlip if discount requested
  Public Overloads Function LeaseSlip(ByVal aDiscountPercent As Single) As Single
    'invoke LeaseSlip() to get fee
    Dim fee As Single = Me.LeaseSlip()
    'calculate and return discount fee
    Dim discountFee As Single = fee * (100 - aDiscountPercent) / 100
    Return discountFee
  End Function
  ```

Overloading a Custom Method

- VB .NET will execute the appropriate method depending on the argument list
  - If no argument is passed
    - Original LeaseSlip method is invoked
  - If an argument is coded
    - Overridden method is executed

Working with Exceptions

- An exception
  - Used to notify the programmer of errors, problems, and other unusual conditions that may occur while the system is running
  - An instance of the Exception class or one of its subclasses

- VB .NET uses five keywords to deal with exceptions
  - Try
    - Used by the client
  - Catch
    - Used by the client
  - Finally
    - Used by the client
  - End Try
    - Used by the client
    - Throw
      - Used by the client
      - Used by the server
Data Validation for slipId

- slipId value should be within the range of 1 through 50
- SetSlipId method
  - Data validation logic used to verify that slipId values are in the range of 1 through 50
  - Create and throw an exception instance if a value is outside the valid range

- Constant MAXIMUM_NUMBER_OF_SLIPS
  - Simplifies future maintenance: should Bradshaw Marina decide to have docks with more than 50 slips
  Private Const MAXIMUM_NUMBER_OF_SLIPS As Integer = 50

Data Validation for slipId

- If statement
  - Determines if aSlipId is within valid range
  - If a value is outside acceptable range
    - An instance of Exception class is created and thrown

Public Sub SetSlipId(ByVal aSlipId As Integer)
' reject slipId if < 0 or > maximum
If aSlipId < 1 Or aSlipId > MAXIMUM_NUMBER_OF_SLIPS Then
  Throw New Exception("Slip Id not between 1 and " 
  & MAXIMUM_NUMBER_OF_SLIPS)
Else
  slipId = aSlipId
End If
End Sub

Data Validation for slipWidth

- Code must be added to the SetSlipWidth method to verify that the width parameter is a valid width value: 10, 12, 14, or 16
- VALID_SLIP_WIDTHS
  - An integer array
  - Stores valid width values

Private Shared ReadOnly VALID_SLIP_WIDTHS As Integer() = {10, 12, 14, 16}

Data Validation for slipWidth

- Validation logic
  - Will iterate the array, seeking a match between the parameter value received (aSlipWidth) and an array value
  - If a matching value is found
    - Parameter is valid
  - If the end of the array is reached without finding a matching value
    - Parameter contains an invalid value
    - An exception is created and thrown
Data Validation for slipWidth

- **Validation logic:**
  ```vbnet
dim validWidth as boolean = false
'Search for a valid width
Dim i as integer
For i = 0 To VALID_SLIP_WIDTHS.Length - 1
  If aSlipWidth = VALID_SLIP_WIDTHS(i) Then validWidth = true
Next i
```

- **If statement to test validWidth**
  - **If true**
    - Width attribute is populated with parameter value
    - An instance of Exception is created and thrown
  - **If not true**
    - If a valid width found, set value
    - Else, throw exception
      ```vbnet
      if validWidth then
        slipWidth = aSlipWidth
      Else 'else throw exception
        throw new exception("Invalid Slip Width")
      End If
      ```

Catching Exceptions

- **If a method that might throw an exception is invoked, the invoking code must be prepared to catch the exception**
  - .NET CLR will terminate processing if an exception is thrown and not caught
- **Try block structure**
  - Keyword Try
  - Code containing invoking statement or statements
  - Keyword End Try

- **Catch block**
  - Receives and deals with an exception
  - Must specify a parameter variable to receive a reference to the exception instance

```vbnet
Dim aSlip as slip
Try 'force an exception with invalid slipID (150)
aSlip = New Slip(150, 10, 25)
Console.WriteLine(aSlip.TellAboutSelf())
Catch theException as exception
  Console.WriteLine("An exception was caught: " & theException.ToString())
End Try
```

- **Finally block**
  - Used to add statements that are to be executed whether or not an exception was caught

```vbnet
Finally
Console.WriteLine("Finally block always executes")
```

Summary

- **Custom methods:** methods which process data
- **Class variables and methods are associated with the class instead of individual instances**
- A new instance is given a copy of all class variables and access to instance methods
- A new instance does not get a copy of class variables and methods
- A method signature consists of the method name and its parameter list
- VB .NET identifies a method by its signature
Summary

• An overloaded method has the same name as another method in the same class, but a different signature
• Exceptions notify the programmer of errors, problems, and other unusual conditions that may occur while the system is running
• Keywords dealing with exceptions: Try, Catch, Finally, End Try, and Throw
• CLR terminates processing if an exception is thrown and not caught