Chapter 6

Writing a Problem Domain Class Definition

Objectives

In this chapter, you will:
• Learn VB .NET naming conventions
• Develop a problem domain (PD) class definition
• Define attributes
• Write methods and properties
• Test a PD class

Objectives

In this chapter, you will:
• Create an instance
• Write a constructor method
• Write a TellAboutSelf method
• Write a Tester class as a Form

VB .NET Naming Conventions

• Class names
  – Start with a capital letter
  – Examples: Customer, Boat
• Attribute names
  – Begin with a lowercase character
  – Subsequent words are capitalized
  – Examples: address, phoneNo
• Method names
  – Begin with an uppercase character
  – Subsequent words are capitalized
  – Examples: GetPhoneNo, SetAddress, ComputeLease

Developing a PD Class Definition

• Bradshaw Marina system PD classes include
  – Customer
  – Boat
  – Slip
  – Dock
• Class definition
  – Code that represents a class
  – Contains attributes and methods of the object

Developing a PD Class Definition

• Customer class
  – Represents marina’s customers
  – Has attributes for customer’s
  • Name
  • Address
  • Telephone number
Developing a PD Class Definition

Class Definition Structure

- Structure of a class definition
  - Class header
  - Attribute definitions
  - Method code
- Class header
  - Line of code that identifies the class and some of its characteristics

Class Definition Structure

- Class header for the Customer definition:

  Public Class Customer

- Keyword Public indicates that the class has public accessibility
- Keyword Class indicates that this line of code is a class header
- Customer establishes the class name

Defining Attributes

- Attributes: defined by declaring variables for each attribute
- An attribute definition
  - Written in the same way as a variable is declared, except:
    - Keyword Private is used instead of Dim
  - Customer attributes are defined as follows:
    - attributes
      - Private name As String
      - Private address As String
      - Private phoneNo As String

Defining Attributes

- When defining attributes, a variable’s accessibility can be:
  - Public: allows any class to access the variable directly
  - Private: prohibits direct access; variable is accessible only within the class where it is defined
  - Protected: allows subclasses to have direct access
  - Friend: permits classes within the same assembly to have access
    - Assembly: a collection of one or more projects deployed as an application
  - Accessor methods can be invoked by other classes to access attribute values

Writing Methods and Properties

- In OO systems, objects interact by one object sending a message to another object to invoke a method
  - Client object: object sending the message
    - Sends a message invoking a server method
    - Can send values in the form of arguments
  - Server object: object receiving the message
    - Performs the requested task
    - May return a value to the client
Writing Methods and Properties

Methods are written using procedures.

1. **VB.NET** has two types of procedures: Sub procedures and Function procedures.
   - A Sub procedure does not return a value.
   - A Function procedure returns a value.

2. A Sub procedure definition:
   - A procedure header followed by one or more statements.
   
   ```
   Sub procedurename (parameter list)
   method statements
   End Sub
   ```

3. A Function procedure definition:
   - A procedure header followed by one or more statements.
   - A Function procedure returns a value.

   ```
   Function procedurename (parameter list) As datatype
   method statements
   End Function
   ```

Defining Attributes

1. **Accessor methods**
   - Often called standard methods.
   - Typically not shown on class diagrams.

2. **Custom methods**
   - Perform other functions.
   - Shown on class diagrams.

3. Two types of accessor methods:
   - Get accessor methods or getters:
     - Retrieve, or get, attribute values.
     - Named with the prefix “Get” followed by the attribute name.
   - Set accessor methods or setters:
     - Change, or set, attribute values.
     - Named with the prefix “Set” followed by the attribute name.

4. A property:
   - Can be used to set and get attribute values.
   - Similar to a method, but appears as an attribute to a client.
   - Begins with a header indicating a property definition.
   - Ends with End Property.
Testing a PD Class

- A tester class simulates the way a client might send messages
- For example: TesterOne class can invoke methods in the Customer class definition
  - TesterOne is the client and Customer is the server
- TesterOne class
  - Startup object for the project
  - Main method begins execution when it is loaded

Creating an Instance

- In TesterOne
  - An instance of the Customer class is created
  - A variable with data type Customer references this instance
- Creating an instance of a class:
  ```
  Dim firstCustomer As Customer = New Customer() ' create instance
  ```
  - The variable firstCustomer points to the newly created Customer instance

Creating an Instance

- Attributes of the Customer instance initially have no values
- Instance attributes can be populated by
  - Setter methods
  - Properties
- Example of using properties
  - CustomerName property can be used to populate the name attribute
  ```
  ' use property to populate name
  firstCustomer.CustomerName = "Eleanor"
  ```

Creating an Instance

- Example of using setter methods
  ```
  ' invoke set accessors to populate attributes
  firstCustomer SetName("Eleanor")
  firstCustomer.SetAddress("Atlanta")
  firstCustomer.SetPhoneNo("123-4567")
  ```
Creating an Instance

- Code to retrieve attribute values and display them:
  - Define variables to contain attribute values retrieved
    Dim customerName, customerAddress, customerPhoneNo As String
    customerName = firstCustomer.GetName()
    customerAddress = firstCustomer.GetAddress()
    customerPhoneNo = firstCustomer.GetPhoneNo()
  - Display the retrieved attribute values
    Console.WriteLine("The name is " + customerName)
    Console.WriteLine("The address is " + customerAddress)
    Console.WriteLine("The phone is " + customerPhoneNo)

Creating Multiple Instances

- A tester class can create more than one instance of a class
- For example: TesterTwo will create three instances using the Customer class definition
- To create three instances, three reference variables are needed:
  Dim firstCustomer, secondCustomer, thirdCustomer As Customer

Creating Multiple Instances

- Code to create three instances of the Customer class:
  firstCustomer = New Customer()
  secondCustomer = New Customer()
  thirdCustomer = New Customer()

Creating Multiple Instances

- Code to invoke setters to populate the attributes:
  firstCustomer.SetName("Eleanor") ' populate first instance
  firstCustomer.SetAddress("Atlanta")
  firstCustomer.SetPhoneNo("123-4567")
  secondCustomer.SetName("Mike") ' populate second instance
  secondCustomer.SetAddress("Boston")
  secondCustomer.SetPhoneNo("467-1234")
  thirdCustomer.SetName("JoAnn") ' populate third instance
  thirdCustomer.SetAddress("St. Louis")
  thirdCustomer.SetPhoneNo("765-4321")

Creating Multiple Instances

- Each instance has
  - Its own identity
  - Its own attribute values
  - The ability to respond to messages
- Statements to retrieve and display each customer’s name:
  - Display names of all three customers
    Console.WriteLine(firstCustomer.GetName())
    Console.WriteLine(secondCustomer.GetName())
    Console.WriteLine(thirdCustomer.GetName())

Writing a Constructor Method

- Constructor
  - A method that is automatically invoked whenever an instance of a class is created using the keyword New
  - Named New
  - Written as a Sub procedure; cannot return a value
- Default constructor
  - Created by VB .NET if the programmer does not write a constructor
  - Does not do anything
  - Consists of only a header and an End Sub statement
Writing a Constructor Method

- Parameterized constructor
  - Created by the programmer
  - Can contain a parameter list to receive arguments that are used to populate the instance attributes

Writing a Constructor Method

- Parameterized constructor for Customer:
  `constructor (3 parameters)
  Public Sub New(ByVal aName As String, ByVal anAddress As String, ByVal aPhoneNo As String)
  setName(aName)
  setAddress(anAddress)
  setPhoneNo(aPhoneNo)
  End Sub
- This constructor invokes setter methods to populate attributes

Writing a Constructor Method

- Alternative design: constructor assigns values directly to attribute variables instead of invoking setter methods
  `Public Sub New(ByVal aName As String, ByVal anAddress As String, ByVal aPhoneNo As String)
  name = aName
  address = anAddress
  phoneNo = aPhoneNo
  End Sub`

Writing a TellAboutSelf Method

- Good design: changes in one class should be insulated from outside classes to reduce maintenance requirements
- To accomplish this, a TellAboutSelf method can be used
  - Can be invoked to retrieve all of the attribute values for an instance
  - Places all the values in a String instance
  - Returns the String instance to the invoking client
  - Should have public accessibility
  - Should be written as a Function procedure

Writing a TellAboutSelf Method

- TellAboutSelf method for Customer
  `TellAboutSelf method
  Public Function TellAboutSelf() As String
  Dim info As String
  info = "Name = " & GetName() & ", Address = " & GetAddress() & ", Phone No = " & GetPhoneNo()
  Return info
  End Function`

Writing a Tester Class as a Form

- In previous examples
  - Tester classes were written as modules
  - A console application was used to run them
- Another approach to writing tester classes: making them GUIs
- Form
  - A visible GUI object
  - Can have push buttons and other GUI objects
Writing a Tester Class as a Form

When a button is clicked on a form, an event is created
• Event procedure or event handler
  – A Sub procedure
  – Executes when an event occurs

Summary
• VB .NET has naming conventions for classes, methods, and variables
• Problem domain class definitions are written for each PD class
• Class definition: code that represents a class
• Accessor methods provide access to attribute values
  – Set accessories, or setters, store values
  – Get accessories, or getters, retrieve values
• A property can also provide access to attributes
• Client objects invoke server methods to perform tasks
• A constructor, a special method, is automatically invoked whenever a class is instantiated
• TellAboutSelf, a custom method, retrieves all instance attribute values and returns them in a string