Chapter 8
Understanding Inheritance and Interfaces

Objectives
In this chapter, you will:
• Implement the Boat generalization/specification class hierarchy
• Understand abstract and final classes and the MustInherit and NotInheritable keywords
• Override a superclass method
• Understand private versus protected access

Implementing the Boat Generalization/Specification Hierarchy
• Boat class
  – Stores information about a boat’s
  – State registration number
  – Length
  – Manufacturer
  – Model year

Testing the Boat Superclass with a Windows Form
• A Windows application and a Windows form with buttons can be used to test all problem domain classes
• Windows form testers
  – Accomplish the same objective as a class module
  – To systematically test each problem domain class to demonstrate that all functionality works as intended

Implementing the Boat Generalization/Specification Hierarchy
• Boat class
  – Parameterized constructor
  • Accepts values for all four attributes
  – Eight standard accessor methods
  • Four setter methods
  • Four getter methods
  – TellAboutSelf method

Objectives
In this chapter, you will:
• Explore the Lease subclasses and abstract methods
• Understand and use interfaces
• Use custom exceptions
• Understand the Object class and inheritance

Testing the Boat Superclass with a Windows Form
• A Windows application and a Windows form with buttons can be used to test all problem domain classes
• Windows form testers
  – Accomplish the same objective as a class module
  – To systematically test each problem domain class to demonstrate that all functionality works as intended

Implementing the Boat Generalization/Specification Hierarchy
• Boat class
  – Parameterized constructor
  • Accepts values for all four attributes
  – Eight standard accessor methods
  • Four setter methods
  • Four getter methods
  – TellAboutSelf method
Using the Inherits Keyword to Create the Sailboat Subclass

- Generalization/specialization hierarchy
  - Superclass
    - Includes attributes and methods that are common to specialized subclasses
  - Instances of the subclasses
    - Inherit attributes and methods of the superclass
    - Include additional attributes and methods

Using the Inherits Keyword to Create the Sailboat Subclass

- Superclass
  - Boat class
    - Four attributes and eight accessor methods
- Subclasses
  - Sailboat
    - Three additional attributes
  - Powerboat
    - Two additional attributes

Using the Inherits Keyword to Create the Sailboat Subclass

- Inherits keyword
  - Used in the class header to implement a subclass
  - Indicates which class the new class is extending
  - Example:
    - Class header to define the Sailboat class as a subclass of Boat:
      ```
      Public Class Sailboat
      Inherits Boat
      ```

Using the Inherits Keyword to Create the Sailboat Subclass

- Class definition of a subclass
  - Includes any attributes and methods in addition to those inherited from the superclass
- Sailboat constructor
  - Accepts seven parameters
    - Four for attributes defined in the Boat superclass
    - Three for additional attributes defined in Sailboat

Using the Inherits Keyword to Create the Sailboat Subclass

- Sailboat constructor
  - MyBase.New call
    - Used to set attributes for registration number, length, manufacturer, and year
    - Invokes constructor of the superclass
    - Must be the first statement in the constructor
    - Required unless the superclass includes a default constructor without parameters
- Sailboat set accessor methods
  - Used to set the three remaining attribute values

Figure 5.7 - Generalization/specialization hierarchy for Boat classes
Adding a Second Subclass – Powerboat

- Powerboat class
  - Extends the Boat class
  - Declares two attributes:
    - numberEngines
    - fuelType
  - Constructor expects six parameters
    - Four required by Boat
    - Two additional attributes for Powerboat

- Once the boats are created
  - Four getter methods inherited from Boat
    - Can be invoked for both sailboats and powerboats
  - Three additional getter methods of Sailboat
    - Not present in powerboats
  - Two additional getter methods of Powerboats
    - Not present in sailboats

Understanding Abstract and Final Classes

- Concrete classes
  - Classes that can be instantiated
  - Examples
    - Sailboat class
    - Powerboat class

Using the MustInherit Keyword

- Abstract class
  - Not intended to be instantiated
  - Only used to extend into subclasses
  - Used to facilitate reuse
- MustInherit keyword
  - Used in class header to declare an abstract class
  - Example:
    - Class header to make the Boat class abstract:
      ```csharp
      Public MustInherit Class Boat
      ```

Using the NotInheritable Keyword

- A Final class
  - A class that cannot be extended
  - Created for security purposes or efficiency
  - Created using the NotInheritable keyword
  - Example
    - Class header for the Powerboat class:
      ```csharp
      Public NotInheritable Class Powerboat
      Inherits Boat
      ```

Overriding a Superclass Method

- Method overriding
  - Method in subclass will be invoked instead of method in the superclass if both methods have the same signature
  - Allows the subclass to modify the behavior of the superclass
Overriding a Superclass Method

- Method overriding vs. method overloading
  - Overloading
    - Two or more methods in the same class have the same name but a different return type or parameter list
  - Overriding
    - Methods in both the superclass and subclass have the same signature

Overriding the Boat TellAboutSelf Method

- TellAboutSelf method
  - Demonstrates method overriding
  - Used to make it easier to get information about an instance of the class
- Overridable keyword
  - Included in method header to make a method overridable

Overriding the Boat TellAboutSelf Method

- To override the TellAboutSelf method
  - Use the same method signature in the subclass, except for using
    - Overrides keyword in place of Overridable keyword
- Once a method is overridden
  - Statements in the subclass method control what system does when a sailboat instance is asked to tell about itself
  - Method in Boat is ignored

Overriding and Invoking a Superclass Method

- Sometimes the programmer needs to override a method by extending what the method does
- For example
  - Powerboat TellAboutSelf method invokes the superclass method using
    - MyBase keyword
    - Superclass method name

Overriding, Polymorphism, and Dynamic Binding

- Polymorphism
  - Objects of different classes can respond to the same message in their own way
- Dynamic binding
  - Used to resolve which method to invoke when the system runs and finds more than one method with the same name
  - Provides flexibility when adding new subclasses that override superclass methods

Understanding Private Versus Protected Access

- Declaring attributes as private
  - Done by using Private keyword
  - No other object can directly read or modify the values
    - Other objects must use methods of the class to get or set values
    - Ensures encapsulation and information hiding
  - Limits the ability of an instance of a subclass to directly access attributes defined by the superclass
Understanding Private Versus Protected Access

- Declaring attributes as Protected
  - Done by using Protected keyword
  - Values can be directly accessed by subclasses

- Local variable
  - Accessible only to statements within a method where it is declared
  - Exists only as long as the method is executing
  - Does not need to be declared as private, protected, friend, or public

Understanding Private Versus Protected Access

- Methods
  - Private methods
    - Can only be invoked from a method within the class
  - Private methods
    - Cannot be invoked even by subclass methods
  - Protected methods
    - Can be invoked by a method in a subclass

Introducing the Lease Subclasses and Abstract Methods

- Lease class
  - Subclasses
    - AnnualLease
    - DailyLease
  - Attributes
    - Amount: a numeric value
    - Start date: a reference variable
    - End date: a reference variable
  - Defined as abstract
    - Includes MustInherit keyword in header

Introducing the Lease Subclasses and Abstract Methods

- Lease class constructor
  - Accepts one parameter
    - A reference to a DateTime instance for start date of the lease
  - Sets end date to Nothing
  - Sets amount of the lease to zero
  - Subclasses set end date and calculate amount depending on type of the lease

Introducing the Lease Subclasses and Abstract Methods

- Lease class constructor
  - Accepts one parameter
    - A reference to a DateTime instance for start date of the lease
  - Sets end date to Nothing
  - Sets amount of the lease to zero
  - Subclasses set end date and calculate amount depending on type of the lease

Adding an Abstract Method to Lease

- Sometimes it is desirable to require all subclasses to include a method
- All Lease subclasses need a CalculateFee method
  - Subclasses are responsible for determining what the lease amount will be
  - Necessary for polymorphism
Adding an Abstract Method to Lease

- Abstract method
  - A method without any statements that must be overridden by all subclasses
  - Declared by using MustOverride keyword in method header
  - For example
    - CalculateFee method of the Lease class

Implementing the AnnualLease Subclass

- AnnualLease subclass attributes
  - balanceDue
    - The amount of the annual lease that remains unpaid
  - payMonthly
    - A Boolean
    - Indicates whether monthly payments will be made for the annual lease

Implementing the AnnualLease Subclass

- If payMonthly is true
  - balanceDue is initially set to eleven-twelfths of the lease amount
- If payMonthly is false
  - balanceDue will be zero

Implementing the DailyLease Subclass

- DailyLease
  - A subclass of the Lease class
  - Used when a customer leases a slip for a short time
  - Additional attribute
    - Number of days of the lease
      - Calculated based on the start date and end date

Understanding and Using Interfaces

- An interface
  - Defines abstract methods and constants that must be implemented by classes that use the interface
  - Can be used to ensure that an instance has a defined set of methods

Understanding and Using Interfaces

- Component-based development
  - Components interact in a system using a well-defined interface
  - Components might be built using a variety of technologies
- Interfaces
  - Define how components can be used
  - Play an important role in developing component-based systems
Understanding and Using Interfaces

- A VB .NET class
  - Can inherit from only one superclass
  - Can implement one or more interfaces
- Interfaces allow VB .NET subclasses a form of multiple inheritance
- Multiple inheritance
  - Ability to inherit from more than one class
  - A subclass is part of two or more generalization/specialization hierarchies

Creating a VB .NET Interface

- Interface
  - Name (by convention)
    - Begins with a capital letter "I"
  - Header
    - Uses Interface keyword, followed by the interface name
  - Methods
    - Include no code

Creating a VB .NET Interface

- A class can implement an interface by adding the following to the class header:
  - Implements keyword
  - Name of the interface

Implementing More Than One Interface

- To implement more than one interface
  - Use Implements keyword in the class header
  - Separate multiple interface names by commas
  - For example:

```plaintext
Public Class DailyLease
  Inherits Lease
  Implements ILeaseInterface, ICompanyInterface
```

Using Custom Exceptions

- NotInheritable keyword
  - Any class that is not declared NotInheritable can be extended
- Custom exception
  - An exception written specifically for an application
  - An example of extending a built-in class

Defining LeasePaymentException

- LeasePaymentException
  - A custom exception
  - Created by defining a class that extends the Exception class
  - Designed for use by the AnnualLease class
  - Thrown if payment is invalid
Throwing a Custom Exception

- **RecordLeasePayment** method
  - A custom method
  - Records a payment
  - Expects to receive the amount of the payment
  - Throws a LeasePaymentException instance if payment amount is not valid

Understanding the Object Class and Inheritance

- **Object class**
  - Extended by all classes in VB .NET
  - Defines basic functionality that other classes need in VB .NET
- **ToString** method
  - A method of the Object class
  - Inherited by all classes
  - By default, returns a string representation of the class name
  - Overridden by other classes to provide more specific information

Understanding the Object Class and Inheritance

- Some other methods of the Object class
  - `Equals`
  - `GetHashCode`
  - `GetType`
  - `ReferenceEquals`
  - The Default New constructor

Summary

- Generalization/specialization hierarchies show superclasses and subclasses
- The subclass inherits attributes and methods of the superclass
- An abstract class: a class that is not instantiated; exists only to serve as a superclass
- A final class: a class that cannot be extended

Summary

- An abstract method in a superclass must be implemented by all the subclasses
- An interface can be used to require that classes contain specific methods
- Custom exceptions can provide detailed information following an exception
- In VB .NET, the Object class is the superclass of all classes