Chapter 12a
OOP: Creating Classes and Using a Business Tier

Chapter Objectives

- Understand how a three-tier application separates the user interface from the business logic and data
- Differentiate between a class and an object
- Create a class that has properties and methods
- Declare object variables and assign values to the properties with a constructor or property methods

Designing Your Own Class

- Analyze the characteristics (variables) and behaviors (methods) needed by a new object
- Example
  - A form gathers price and quantity of a product (presentation tier)
  - Design a class to perform calculation of extended price (business tier)**VERY IMPORTANT
    - Price, quantity, and extended price are stored in private variables in the class
    - Variables are accessed through property methods
Designing Your Own Class – cont’d

- The presentation tier form instantiates the business tier class
- Pass the price and quantity to it through property procedures or...
- Invokes a method and passes arguments to the method’s parameter list to calculate the extended price
- Display the extended price on the form by retrieving it from a property method

Creating Properties in a Class

- Define private member variables inside your class
  - Stores the values for the properties of the class
  - Declare all variables as `private` or `protected`, not `public`, which violates the rules of encapsulation
  - Protected variables and methods behave as `private` but are available in classes that inherit from the class
- Use property `set` and `get` methods to pass values between a class and the class where objects of the class are instantiated

Class Methods

- Create methods of a new class by coding public methods within the class
- Methods declared with the `private` keyword are available only within the class
- Methods declared with the `public` keyword are available to external objects created from this class or other classes
- Methods declared with the `protected` keyword behave as private within the class and any class that inherits from it
### Constructor

- Executes automatically when an instance (object) of the class is created.
- Ideal location for initialization tasks such as setting the initial values of variables and properties.
- Constructor must be public because the objects created must execute this method.
- If no constructor is written for a class, the compiler creates an implicit default constructor with an empty parameter list.

### Overloading the Constructor

- **Overloading** means that two methods have the same name but a different list of arguments (the signature).
- Create overloaded methods in a class:
  - Give the same name to multiple methods.
  - An empty constructor:
    ```
    public ClothingSale()
    ```
  - A constructor that passes arguments to the class:
    ```
    public ClothingSale(string productNumberString, int quantityInteger, decimal discountRateDecimal)
    ```

### Parameterized Constructor

- A constructor that requires arguments.
- Allows arguments to be passed when creating a new object from the class.
- Assign incoming values to the properties of the class:
  - Assign an argument to the property name, which then uses the `set` method.
  - Validation is often performed in the `set` methods.
Creating a New Class – Step-by-Step

- Add a new class to a project
  - Choose Project/Add Class
  - In the Add New Item dialog box, select Class
  - Name the class and click Add
- Define the class properties
  - Declare private class-level variables to hold the property values of the new class
  - Write the property methods with a public get and a private set method
- Write the constructor
- Code a method

Property Methods with Mixed Access Levels

- Set a property statement as public and then assign a private get or set method

```csharp
public string EmployeeID
{
  get
  {
    return employeeIDString;
  }
  private set
  {
    employeeIDString = value;
  }
}
```

Example – Creating a New Object Using a Class

- Creating a new class does not create any objects of the class
  - Similar to creating a new tool for the toolbox
- Generally a two-step operation
  - Declare the variable
  - Instantiate the new object using the new keyword
- Can declare and instantiate in the same statement
  ```csharp
  ClothingSale aClothingSale = new ClothingSale();
  ```
Creating a New Object Using a Class – cont’d

- If an object variable is to be used in multiple methods, declare the object at class level
- Instantiate the object inside of a method at the time the object is needed
  - Use a `try/catch` if converting and passing values entered by a user
- Pass values for the arguments at instantiation when using a parameterized constructor

Instance Variables versus Static Variables

- Instance variables and properties
  - Separate memory location for each instance of the object
  - Each object created from the class has its own set of properties
  - Also called `instance members`
  - An instance member has one copy for each instance or object of the class
- Static variable, property or method
  - Exists, or is available, for all objects of the class
  - Used for a count or total of all objects of the class
  - Also called static members
  - A static member has one copy for all objects of the class
  - Access static members without instantiating an object

Creating Static Members

- Use the `static` keyword to create a static member
- Make properties for static members read-only
  - Allows values to be retrieved, but not set directly
- A `static` keyword on a private class-level variable is required
- The `static` keyword on a property method is optional
  - Allows a property to be retrieved without creating an instance of the class
Inheritance

- A new class can be based on another class
  - Can inherit from
    - One of the existing .NET classes
    - One of your own classes
  - The inheritance clause must follow the class header prior to any comments

Inheriting Properties and Methods

- When writing code for a derived class, all public and protected data members and methods of the base class can be referenced
- Use the `protected` keyword to declare elements that are accessible only within their own class or any class derived from that class

Setting the Startup Form

- Visual Studio by default begins execution of C# applications with the Program.cs file
- The first form added to a project is the startup form
- To change the startup form, modify the `Application.Run` method in the Program.cs file
  ```csharp
  Application.Run(new MainForm());
  ```
Adding an Existing Class to a Project

- Two ways to include an existing form or other class in a project
  1. Reference the file in its original location
  2. Move or copy the file into the project folder (best way)

- Select *Project/Add Existing Item* to add the file to the project
  - Or, right-click the project name in Solution Explorer and select from the context menu