UNDERSTANDING DOCUMENT

MODULE - 3

PREPARED BY

PINAL PAMBHAR

Table of Contents

Understanding Classes

Classes are the user-defined data types that represent the **state** and **behaviour** of an object. State represents the properties and **behaviour** is the action that objects can perform.

Types of Classes

1. Abstract class
2. Partial class
3. Sealed class
4. Static class

Abstract Class : An Abstract class is a class that provides a common definition to the subclasses and this is the type of class whose object is not created.

- Abstract classes are declared using the abstract keyword.
- We cannot create an object of an abstract class.
- If you want to use it then it must be inherited in a subclass.
- An Abstract class contains both abstract and non-abstract methods.
- The methods inside the abstract class can either have an implementation or no implementation.
- We can inherit two abstract classes; in this case the base class method implementation is optional.
- An Abstract class has only one subclass.
- Methods inside the abstract class cannot be private.
- If there is at least one method abstract in a class then the class must be abstract.

Partial Class: A Partial class provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled.

  - A partial class is created by using a ***partial*** keyword.
  - This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.
  - When you want to chop the functionality of the class, method, interface, or structure into multiple files, then you should use partial keyword and all the files are mandatory to be available at compile time for creating the final file.
  - The partial modifier can only present instantly before the keywords like struct, class, and interface.
- Every part of the partial class definition should be in the same assembly and namespace, but you can use a different source file name.
- Every part of the partial class definition should have the same accessibility as private, protected, etc.
- If any part of the partial class is declared as an abstract, sealed, or base, then the whole class is declared of the same type.
- The user is also allowed to use nested partial types.

- Dissimilar parts may have dissimilar base types, but the final type must inherit all the base types.

Sealed Class: Sealed classes are used to restrict the users from inheriting the class.

- A class can be sealed by using the sealed keyword.
- The keyword tells the compiler that the class is sealed, and therefore, cannot be extended.
- No class can be derived from a sealed class.
- Access modifiers are not applied to a sealed class.
- To access the sealed members we must create an object of the class.

Static Class: A class with static keyword that contains only static members is defined as static class.

- Static classes are sealed, means you cannot inherit a static class from another class.
- Static class cannot instantiated using a new keyword.
- A static method can only contain static variables and can only access other static items.
- Static items share the resources between multiple users.
- Static members are allocated in a high frequency heap area of the memory.
- The methods of the static class can be called using the class name without creating the instance.

Depth in Classes

Object

- An object is an instance of a class that can be used to access the data members and member functions of a class.
- An object is an entity that has a state and behavior. Here, state means data, and behavior means functionality.
- An object is a runtime entity, it is created at runtime.

Properties

- Property is an extension of the class variable.
- It provides a mechanism to read, write, or change the class variable's value without affecting the external way of accessing it in our applications.
- Properties can contain one or two code blocks called accessors, and those are called a get accessor and set accessor.
- By using get and set accessors, we can change the internal implementation of class variables and expose them without affecting the external way of accessing them based on our requirements.
- With accessors through which the values of the private fields can be read, written, or manipulated.

Following is the **syntax** of defining a property with get and set accessor

```
<access_modifier> <return_type> <property_name>
{
   get
   {
     // return property value
   }
   set
   {
    // set a new value
   }
}
```

- Here, the **get** accessor code block will be executed whenever the property is read, and the code block of **set** accessor will be executed whenever the property is assigned to a new value.

These properties are categorized into three types, those are.

Type	Description
Read-Write	A property that contains a both get and set accessors, then we will call it a read-write property.
Read-Only	A property that contains only get accessor, then we will call it a read-only property.
Write-Only	A property that contains only set accessor, then we will call it a write-only property.

Methods

- Method is a separate code block, and that contains a series of statements to perform particular operations.
- Methods must be declared either in class or struct by specifying the required parameters.
- Methods are useful to improve code reusability by reducing code duplication.

Syntax:

```
class class_name
{
    ...
    ...
    <Access_Specifier> <Return_Type> Method_Name(parameters)
    {
        // Statements to Execute
    }
    ...
    ...
}
```

- **Access_Specifier** - It is used to define an access level, either public or private, etc., to allow other classes to access the method. If we didn't mention any access modifier, then by default, it is private.
- **Return_Type** - It is used to specify the type of value that method wiil return. If the method is not returning any value, then we need to mention void as the return type.
- **Method_Name** - It must be a unique name to identify the method in a class.
- **Parameters** - The method parameters are useful to send or receive data from a method, and these method parameters are enclosed within parentheses and are separated by commas. If no parameters are required for a method, we need to define a method with empty parentheses.

Events

- ❖ Events enable a class or object to notify other classes or objects when something of interest occurs.
- ❖ The class that sends (or raises) the event is called the publisher and the classes that receive (or handle) the event are called subscribers.
- ❖ Events contains a publisher, subscriber, notification, and handler.

Events have the following properties:

- The publisher determines when an event is raised; the subscribers determine what action is taken in response to the event.
- An event can have multiple subscribers. A subscriber can handle multiple events from multiple publishers.
- Events that have no subscribers are never raised.
- Events are typically used to signal user actions such as button clicks or menu selections in graphical user interfaces.
- When an event has multiple subscribers, the event handlers are invoked synchronously when an event is raised.

There are three steps to publish an event

1. Declare a delegate

<Access_Specifier> delegate <Return_Type> Delegate_Name(parameters);

2. Define the event based on delegate

<Access_Specifier> event Delegate_Name Event_Name;

3. Publish the event

Event_Name?.Invoke(parameters);

Scope & Accessibility Modifiers

- All types and type members have an accessibility level.
- The accessibility level controls whether they can be used from other code in your assembly or other assemblies.

1. **public:** The type or member can be accessed by any other code in the same assembly or another assembly that references it. The accessibility level of public members of a type is controlled by the accessibility level of the type itself.
2. **private:** The type or member can be accessed only by code in the same class or struct.
3. **protected:** The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
4. **internal:** The type or member can be accessed by any code in the same assembly, but not from another assembly.
5. **protected internal:** The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
6. **private protected:** The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

Caller's location	**public**	**protected internal**	**protected**	**internal**	**private protected**	**private**
Within the class	✔	✔	✔	✔	✔	✔
Derived class (same assembly)	✔	✔	✔	✔	✔	✕
Non-derived class (same assembly)	✔	✔	✕	✔	✕	✕
Derived class (different assembly)	✔	✔	✔	✕	✕	✕
Non-derived class (different assembly)	✔	✕	✕	✕	✕	✕

Namespace & .Net Library

.NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.

Namespaces	Description
System	It includes all common datatypes, string values, arrays and methods for data conversion.
System.Data, System.Data.Common, System.Data.OleDb, System.Data.SqlClient, System.Data.SqlTypes	These are used to access a database, perform commands on a database and retrieve database.
System.IO, System.DirectoryServices, System.IO.IsolatedStorage	These are used to access, read and write files.
System.Diagnostics	It is used to debug and trace the execution of an application.
System.Net, System.Net.Sockets	These are used to communicate over the Internet when creating peer-to-peer applications.
System.Windows.Forms, System.Windows.Forms.Design	These namespaces are used to create Windows-based applications using Windows user interface components.
System.Web, System.WebCaching, System.Web.UI, System.Web.UI.Design, System.Web.UI.WebControls, System.Web.UI.HtmlControls, System.Web.Configuration, System.Web.Hosting, System.Web.Mail, System.Web.SessionState	These are used to create ASP. NET Web applications that run over the web.
System.Web.Services, System.Web.Services.Description, System.Web.Services.Configuration, System.Web.Services.Discovery, System.Web.Services.Protocols	These are used to create XML Web services and components that can be published over the web.
System.Security, System.Security.Permissions, System.Security.Policy, System.WebSecurity, System.Security.Cryptography	These are used for authentication, authorization, and encryption purpose.
System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath,	These namespaces are used to create and access XML files.

System.Xml.Xsl	

Creating and adding ref. to assemblies

- An Assembly is a basic building block of .Net Framework Application.
- It is basically a compiled code that can be executed by the CLR.
- An Assembly can be a DLL or exe depending upon the project that we choose.

Assemblies are basically of two types.

1. **Private Assembly:** It is an assembly that is used by a single application only.
2. **Shared Assembly:** It is an assembly that is used by more than one projects.

Steps for reference to assembly

- Open solution explorer by (ctrl + alt + L)
- Choose Add Reference
- Browsr of choose available assembly and click ok

Working with collections

- ❖ C# collection types are designed to store, manage and manipulate similar data more efficiently.
- ❖ Data manipulation includes adding, removing, finding, and inserting data in the collection.

Collection types implement the following common functionality:

- Adding and inserting items to a collection
- Removing items from a collection
- Finding, sorting, searching items
- Replacing items
- Copy and clone collections and items
- Capacity and Count properties to find the capacity of the collection and number of items in the collection

.NET supports two types of collections**, generic collections** and **non-generic collections.**

Generic Collection

SYSTEM.COLLECTIONS.GENERIC CLASSES	
Class	**Description**
Dictionary<TKey,TValue>	Represents a collection of key/value pairs that are organized based on the key.
List<T>	Represents a list of objects that can be accessed by index. Provides methods to search, sort, and modify lists.
Queue<T>	Represents a first in, first out (FIFO) collection of objects.
SortedList<TKey,TValue>	Represents a collection of key/value pairs that are sorted by key based on the associated IComparer<T> implementation.
Stack<T>	Represents a last in, first out (LIFO) collection of objects.

Non-Generic Collection

Class	Description
ArrayList	Represents an array of objects whose size is dynamically increased as required.
Hashtable	Represents a collection of key/value pairs that are organized based on the hash code of the key.
Queue	Represents a first in, first out (FIFO) collection of objects.
Stack	Represents a last in, first out (LIFO) collection of objects.