# Module – 3

## 14. Understanding Classes

- A class is like a blueprint of a specific object.
- Class and Object are the basic concepts of Object-Oriented Programming which revolve around the real-life entities.
- Basically, a class combines the fields and methods into a single unit.
- In C#, classes support polymorphism, inheritance and also provide the concept of derived classes and base classes.

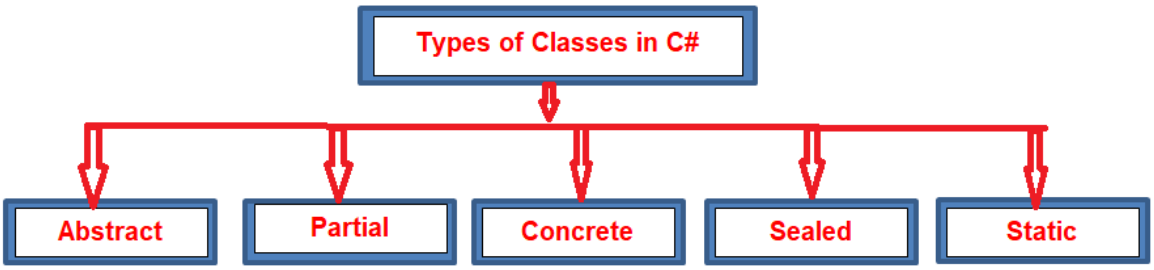
**Declaring Classes**

Classes are declared by using the class keyword followed by a unique identifier, as shown in the following example:

```
[access modifier] [class] [identifier]
public class Customer
{
// Fields, properties, methods and events etc.
}
```

- The class keyword is preceded by the access level. Because public is used in this case, anyone can create instances of this class.
- The name of the class follows the class keyword. The name of the class must be a valid C# identifier name.
- The remainder of the definition is the class body, where the behavior and data are defined.
- Fields, properties, methods, and events on a class are collectively referred to as class members.

## Types of Class




### Abstract class

- 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 Classes

It is a type of class that allows dividing their properties, methods and events into multiple source files and at compile time these files are combined into a single class.

The following are some key points:

- All the parts of the partial class must be prefixed with the partial keyword.
- If you seal a specific part of a partial class then the entire class is sealed, the same as for an abstract class.
- Inheritance cannot be applied on partial classes.
- The classes that are written in two class files are combined together at run time.

**Sealed Class**

A Sealed class is a class that cannot be inherited and used to restrict the properties.

The following are some key points:

- A Sealed class is created using the sealed keyword.
- Access modifiers are not applied to a sealed class.
- To access the sealed members we must create an object of the class.

**Static Class**

It is the type of class that cannot be instantiated, in other words we cannot create an object of that class using the new keyword, such that class members can be called directly using their class name.

The following are some key points:

- Created using the static keyword.
- Inside a static class only static members are allowed, in other words everything inside the static class must be static.
- We cannot create an object of the static class.
- A Static class cannot be inherited.
- It allows only a static constructor to be declared.
- The methods of the static class can be called using the class name without creating the instance.

**Concrete Class**

The default class is a concrete class. The class keyword is used to define classes And usually they are simply referred to as classes. Concrete classes depict the conceptual representation of real-world objects.

- Classes have properties called attributes.
- Attributes are implemented as global and instance variables.
- Methods in the classes represent or define the behavior of these classes.
- Methods and attributes of classes are called the members of the class.
- Typically, encapsulation is achieved by making the attributes private, while creating public methods that can be used to access those attributes.
- An object is the instance of a class.

## 15. Depth in classes

Object:

- Object is a real-world entity, for example, chair, car, pen, mobile, laptop etc.
- In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.
- Object is a runtime entity; it is created at runtime.
- Object is an instance of a class. All the members of the class can be accessed through object.

Properties:

- Property in C# is a member of a class that provides a flexible mechanism for classes to expose private fields.
- Properties are an extension of fields and are accessed using the same syntax.
- They use accesses through which the values of the private fields can be read, written or manipulated.
- Properties do not name the storage locations. Instead, they have accesses that read, write, or compute their values

Events:

- Events are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications.
- Applications need to respond to events when they occur.
- The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class.

A publisher is an object that contains the definition of the event and the delegate. The event delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.

A subscriber is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

Constructor

- Constructor of a class must have the same name as the class name in which it resides.
- A constructor cannot be abstract, final, and Synchronized.
- Within a class, you can create only one static constructor.
- A constructor doesn't have any return type, not even void.
- A static constructor cannot be a parameterized constructor.
- A class can have any number of constructors.

Types of Constructors

- Default Constructor
- Parameterized Constructor
- Copy Constructor
- Private Constructor
- Static Constructor

Default Constructor

- A constructor with no parameters is called a default constructor.
- A default constructor has every instance of the class to be initialized to the same values.
- The default constructor initializes all numeric fields to zero and all string and object fields to null inside a class.

Parameterized Constructor

- A constructor having at least one parameter is called as parameterized constructor.
- It can initialize each instance of the class to different values.

Copy Constructor

- This constructor creates an object by copying variables from another object.
- Its main use is to initialize a new instance to the values of an existing instance.

Private Constructor

- If a constructor is created with private specifier is known as Private Constructor.
- It is not possible for other classes to derive from this class and also it's not possible to create an instance of this class.

Static Constructor

- Static Constructor has to be invoked only once in the class and it has been invoked during the creation of the first reference to a static member in the class.
- A static constructor is initialized static fields or data of the class and to be executed only once.

## 16.Scope & Accessibility Modifiers

C# Access modifiers or specifiers are the keywords that are used to specify accessibility or scope of variables and functions in the C# application.

C# provides five types of access specifiers.

1. Public
2. Protected
3. Internal
4. Protected internal
5. Private

**public modifier**

The public keyword is an access modifier for types and type members. Public access is the most permissive access level.

There are no restrictions on accessing public members.

**Accessibility**

- Can be accessed by objects of the class
- Can be accessed by derived classes

**private modifier**

Private access is the least permissive access level.

Private members are accessible only within the body of the class or the struct in which they are declared.

**Accessibility**

- Cannot be accessed by object
- Cannot be accessed by derived classes

**protected modifier**

A protected member is accessible from within the class in which it is declared, and from within any class derived from the class that declared this member.

A protected member of a base class is accessible in a derived class only if the access takes place through the derived class type.

**Accessibility**

- Cannot be accessed by object
- By derived classes

**Protected Internal modifier**

Variable or function declared protected internal can be accessed in the assembly in which it is declared. It can also be accessed within a derived class in another assembly.

## 17. Namespace & .Net Library.

**Namespace:**

- Namespaces are used to organize the classes.
- It helps to control the scope of methods and classes in larger .Net programming projects.
- It provides a way to keep one set of names different from other sets of names.
- The biggest advantage of using namespace is that the class names which are declared in one namespace will not clash with the same class names declared in another namespace.

Syntax:

```
Namespace
        {
                // Classes and/or structs and/or enums etc.
        }
```

**.net library:**

- The Class Library contains program code, data, and resources that can be can used by other programs and are easily implemented into other Visual Studio projects.
- In visual studio we can implement code in project type as class library and then it can be referred by other programs or even in same program.
- Simple program that demonstrates use of class library is shown below.
- I have prepared one class library project and another console application which uses method specified in the class library.

**Standard Namespaces in .NET**

The following are some of the standard namespaces in .NET framework.

- System: Contain classes that implement basic functionalities like mathematical operations, data conversions etc.
- System.IO: Contains classes used for file I/O operations.
- System.Net: Contains class wrappers around underlying network protocols.
- System.Collections: Contains classes that implement collections of objects such as lists, hashtable etc.
- System.Data: Contains classes that make up ADO.NET data access architecture.
- System,Drawing: Contains classes that implement GUI functionalities.
- System.Threading: Contains classes that are used for multithreading programming.
- System.Web: Classes that implement HTTP protocol to access web pages.
- System.Xml: Classes that are used for processing XML data.

## 18.Creating and adding ref. to assemblies

- An Assembly is a basic building block of .Net Framework applications.
- It is basically a compiled code that can be executed by the CLR.
- An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality.
- An Assembly can be a DLL or exe depending upon the project that we choose.

Assemblies are basically the following two types:

1. Private Assembly
2. Shared Assembly

Private Assembly

- It is an assembly that is being used by a single application only.
- Suppose we have a project in which we refer to a DLL so when we build that project that DLL will be copied to the bin folder of our project.
- That DLL becomes a private assembly within our project. Generally, the DLLs that are meant for a specific project are private assemblies.

Shared Assembly

- Assemblies that can be used in more than one project are known to be a shared assembly.
- Shared assemblies are generally installed in the GAC. Assemblies that are installed in the GAC are made available to all the .Net applications on that machine.

## 19. Working with collections

- Collections standardize the way of which the objects are handled by your program.
- In other words, it contains a set of classes to contain elements in a generalized manner.
- With the help of collections, the user can perform several operations on objects like the store, update, delete, retrieve, search, sort etc.
- NET supports two types of collections

    1. Generic Collection
    2. Non-Generic Collection

- Generic collections with work generic data type.
- In non-generic collections, each element can represent a value of a different type. The collection size is not fixed. Items from the collection can be added or removed at runtime.
- Non-generic: ArrayList, HashTable, SortedList, Stack, Queue
- Generic: List, Dictionary, SortedList, Stack, Queue.