RKIT Module – 3

Name: Adhyaru Keyur D.

Understanding Classes

Class

- ✓ 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 (member function which defines actions) into a single unit.
- ✓ In C#, classes support polymorphism, inheritance and also provide the concept of derived classes and base classes.

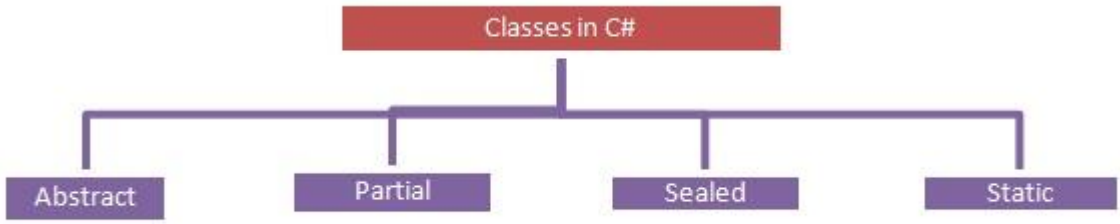
Declaration of class

- ✓ Generally, a class declaration contains only keyword class, followed by an identifier(name) of the class.
- ✓ class declarations can include these components – Modifiers, Keyword class, Base Class, Interfaces and Body.

Some Key points about classes

- ✓ Classes are reference types that hold the object created dynamically in a heap.
- ✓ All classes have a base type of System.Object.
- ✓ The default access modifier of a class is Internal.
- ✓ The default access modifier of methods and variables is Private.
- ✓ Directly inside the namespaces declarations of private classes are not allowed.

Types of 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:

- ✓ 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.
  - 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.
  - 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.
  - 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.

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 Constructor

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Private Constructor
5. Static Constructor

Default Constructor

- ✓ A constructor with no parameters is called a default constructor.
- ✓ 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.
- ✓ It is the implementation of a singleton class pattern.
- ✓ use private constructor when we have only static members.
- ✓ Using private constructor, prevents the creation of the instances of that 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.
- ✓ It can't be called directly.
- ✓ When it is executing then the user has no control.
- ✓ It does not take access modifiers or any parameters.
- ✓ It is called automatically to initialize the class before the first instance created.

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 behaviour. Here, state means data and behaviour 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. For example, interrupts.
- ✓ Events are used for inter-process communication.

Using Delegates with Events

- ✓ 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. The class containing the event is used to publish the event. This is called the publisher class. Some other class that accepts this event is called the subscriber class. Events use the publisher-subscriber model.
- ✓ 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.

Points to Publishing an Event

1. First need to declare a delegate, that is the contract between the publisher and the subscriber.
2. Define the event based on the delegate.
3. Publish the event.

Scope & Accessibility Modifiers

- ✓ Access Modifiers are keywords that define the accessibility of a member, class or datatype in a program.
- ✓ These are mainly used to restrict unwanted data manipulation by external programs or classes.
- ✓ There are 4 access modifiers (public, protected, internal, private) which defines the 6 accessibility levels as follows:

Access Modifiers	Public	Protected	Internal	Protected Internal	Private	Private Protected
Entire Program	YES	NO	NO	NO	NO	NO
Containing Class	YES	YES	YES	YES	YES	YES
Current Assembly	YES	NO	YES	YES	NO	NO
Derived Types	YES	YES	NO	YES	NO	NO
Derived types within current assembly	YES	YES	YES	YES	No	YES

Public:

- ✓ Access is granted to the entire program.
- ✓ This means that another method or another assembly which contains the class reference can access these members or types.
- ✓ This access modifier has the most permissive access level in comparison to all other access modifiers.

Protected:

- ✓ Access is limited to the class that contains the member and derived types of this class.
- ✓ It means a class which is the subclass of the containing class anywhere in the program can access the protected members.

Internal:

- ✓ Access is limited to only the current Assembly, that is any class or type declared as internal is accessible anywhere inside the same namespace.
- ✓ It is the default access modifier in C#.

Note: In the same code if you add another file, the class Complex will not be accessible in that namespace and compiler gives an error.

Private:

- ✓ Access is only granted to the containing class.
- ✓ Any other class inside the current or another assembly is not granted access to these members.

Private Protected:

- ✓ Access is granted to the containing class and its derived types present in the current assembly.
- ✓ This modifier is valid in C# version 7.2 and later

Namespace & .Net Library

What is Namespace?

- namespaces are used to logically arrange classes, structs, interfaces, enums and delegates.
- The namespaces can be nested. That means one namespace can contain other namespaces also.
- The .NET framework already contains number of standard namespaces like System, System.Net, System.IO etc.
- In addition to these standard namespaces the user can define their own namespaces.

Declaring a Namespace

Syntax:

```
namespace <namespace_name>
{
        // Classes and/or structs and/or enums etc.
}
```

- **Note: namespace** keyword is used to define namespace
- It is not possible to use any access specifiers like private, public etc with a namespace declaration.
- The namespaces in C# are implicitly have public access and this is not modifiable.
- Default it provide internal access.

Creating Aliases

- **Developers can create Aliases of the namespace.**

Example:

```
using con = System.Console; // Create an alias
class MyClient
{
        public static void Main()
        {
                con.WriteLine("namespace demo");
        }
}
```

Standard Namespaces in .NET

- 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.

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.

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.
- ✓ 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.
    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.