# UNDERSTANDMENTS & CONCLUSIONS

NANDINI BHATTACHARYA

## MODULE – 3

### 14. Understanding Classes

- A class is like a blueprint of a specific object.
- 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.
- 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
- The default access modifier of methods and variables is Private.
- Directly inside the namespaces declarations of private classes are not allowed.

#### 14.1 Class types

- **static -** It is the type of class that cannot be instantiated; 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.

- **Sealed -** 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

- **Abstract -** 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.

### 15. Depth in Classes

- **Objects -** An object is an instance of class which is used to call class methods and functionalities. A class is also called as definition of object.
- Ex – Business objbusiness = new Business ();
- We created class Business and defined its object to be objbusiness.

- **Properties -** C# properties provides flexible mechanism to expose private fields of a class in a public way a get accessor is used to return the value of the field and set is used to assign value to the field.
- They are special methods called **accessors**. There are 2 types of accessors get and set.
- Properties can be read only, read-write, write-only too.
- It is not possible to have a same name for property and variable.
- **Get Accessor** specifies the read only property and help us to access the

value of the field publicly.
- **Set Accessor** specifies the write only property and help us to assign the value of the private field and also returns a single value.

  **Syntax** - <access modifier><return-type><property-name>
  ```
  {
  get
          {
                  //code
          }
  set
          {
                  //code
          }
  }
  ```

- **Methods -** Methods are members of the class which are implement the logical tasks which are performed with the help of method called by object of the class or by class itself if method is static.
- If a method doesn't return any value then the return types must be void.
- The signature or name of the method should be unique inside a class.
- The signature of a method means the name of the method along with the Parameters, Modifiers and Data type of the Parameters.
- There are also different types of parameters a method can get like value type, reference type, out, named, default.
- Syntax - <access modifier><return-type><method-name>(parameters)
  ```
  {
  //code
  }
  ```

- **Constructors -** A constructor is a special method of the class with the name same as of class. It is invoked when an instance of that class is created. It's code is executed when an object is created.
- 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.
- Access modifiers can be used in constructor declaration to control its access i.e. which other class can call the constructor.
- Types of Constructors –
  Default Constructor
  Parameterized Constructor
  Copy Constructor
  Private Constructor
  Static Constructor

- **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.
- Event is raised in a class with a delegate in the same or some other class. The class which contains event is called publisher class. The class which accepts event is called subscriber class. This forms publisher - subscriber model. In the publisher class the event is raised after defining the delegate and defining the event itself.
- The publisher class object calls the event and it is notified to other objects as it consists of event definition.
- The subscriber class object contains the event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

**16. Scope & Accessibility Modifiers**

- It defines the accessibility of a class and its members by other classes or outside that particular class.

- **public** – This provides access to members even outside the assembly or namespace ,it provides accessibility to the entire program by giving a reference of public class.
  Syntax – public <TypeName> <Name>

- **protected** – The access is granted to only subclasses inside or outside of the same namespace or assembly which derived from the following class.
  Syntax – protected <TypeName> <Name>

- **internal** – The access is only granted in the current namespace that is any class declared internal is accessible by all other classes in the same namespace.
  Syntax – internal <TypeName> <Name>

- private – The access is granted only within the class which contains the private members not outside of that class.
  Syntax – private <TypeName> <Name>

**17. Namespace & .Net Library**

- 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
- 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.
- . Net Class Library is a base class library containing namespaces, interfaces, classes, value-types used in .Net Applications.
- **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 collection that are built to work together and form a logical functionality unit. It is in the form of executable(.exe) or dynamic link library (.dll). They maybe created in the form of class library project and can contain one or more files in it.
- **Adding ref. to assemblies** – Ways to add reference to an assembly :
  1. To add specific DLL from the path instead of referring to class library projects.
  2. To add DLL from the bin folder of the class library project made of the same solution.

3. To add assembly file reference from the .Net class library.

- 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**

- 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