## Understanding Classes

- A class is a user-defined blueprint or prototype from which objects are created. 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.
- Declaration of class: a class declaration contains only keyword class, followed by an identifier(name) of the class. But there are some optional attributes that can be used with class declaration according to the application requirement. In general, class declarations can include these components, in order:

- Modifiers: A class can be public or internal etc. By default modifier of class is *internal*.
- Keyword class: A *class* keyword is used to declare the type class.
- Class Identifier: The variable of type class is provided. The name of class should begin with a initial letter which should be capitalized by convention.
- Base class or Super class: The name of the class's parent (superclass), if any, preceded by the *: (colon)*. This is optional.
- Interfaces: A comma-separated list of interfaces implemented by the class, if any, preceded by the : (colon). A class can implement more than one interface. This is optional.
- Body: The class body is surrounded by { } (curly braces).
- To access the class members, you use the dot (.) operator.
- The dot operator links the name of an object with the name of a member.
- Member Functions and Encapsulation:
  - A member function of a class is a function that has its definition or its prototype within the class definition similar to any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object.

    - Member variables are the attributes of an object and they are kept private to implement encapsulation. These variables can only be accessed using the public member functions.

- Constructor:
    - A class constructor is a special member function of a class that is executed whenever we create new objects of that class.
    - A constructor has exactly the same name as that of class and it does not have any return type.
    - A default constructor does not have any parameter but if you need, a constructor can have parameters. Such constructors are called parameterized constructors. This technique helps you to assign initial value to an object at the time of its creation
- Destructor:
    - A destructor is a special member function of a class that is executed whenever an object of its class goes out of scope. A destructor has exactly the same name as that of the class with a prefixed tilde (~) and it can neither return a value nor can it take any parameters.
    - Destructor can be very useful for releasing memory resources before exiting the program. Destructors cannot be inherited or overloaded.
- Static:
    - static means something which cannot be instantiated. You cannot create an object of a static class and cannot access static members using an object.
    - classes, variables, methods, properties, operators, events, and constructors can be defined as static using the static modifier keyword.
    - In static class all members of it also static
    - You cannot create an object of the static class; therefore the members of the static class can be accessed directly using a class name like ClassName.MemberName.
- Sealed:

  - 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.
  - A method can also be sealed, and in that case, the method cannot be overridden. However, a method can be sealed in the classes in which they have been inherited. If you want to declare a method as sealed, then it has to be declared as virtual in its base class.

- Abstract:
  - Abstraction  is the process to hide the internal details and showing only the functionality. The abstract modifier indicates the incomplete implementation.
  - The keyword abstract is used before the class or method to declare the class or method as abstract. Also, the abstract modifier can be used with indexers, events, and properties.
  - A method which is declared abstract, has no “body” and declared inside the abstract class only. An abstract method must be implemented in all non-abstract classes using the override keyword. After overriding the abstract method is in the non-Abstract class.
  - An Abstract class is never intended to be instantiated directly. This class must contain at least one abstract method.
  - An abstract class is an incomplete class or special class we can’t be instantiated. The purpose of an abstract class is to provide a blueprint for derived classes and set some rules what the derived classes must implement when they inherit an abstract class

## Depth in Classes

- Object:
  - It is a basic unit of Object-Oriented Programming and represents the real-life entities. A typical C# program creates many objects, which as you know, interact by invoking methods. An object consists of :

- State: It is represented by attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.
- When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.
- The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

- Methods:
  - method is invoked to execute the code
  - it describes behavior of the program
  - it has access modifier private or public
  - Access modifier : A class can be public or internal etc. By default modifier of class is internal.
  - Return Type : A method may or may not return a value. When it does not return a value, we use return type void otherwise return keyword is used to return the value.
  - Method name: It is a unique and case sensitive identifier of a method.
  - Parameter list: It is an optional component enclose in parentheses, used to pass data from method.
  - Method Body: It is a set of instructions required to complete the activity.
  - These method name and parameters together are called method signature. Every method must have unique method signature as it allow C# compiler to distinguish between various methods.

- Field and Properties

    - Field: A variable declared inside a class or struct is defined by fields. Fields can be marked as public, private, protected, internal or optionally as static.

    - Property: A property is similar to field except that a property has a special syntax to control what happens when someone reads the data. These are get and set accessors. These are like methods and are contained inside property declaration. Get is used to return the property value and set is used to assign the value.

    - Syntax for property:

```
Private string _name;
Public string name{
        get{
                return  _name;
        }
        set{
                _name=value;
        }
}
```

- Events:
    - Event is another member that we can have in class. It gives a way for object to signal state change. A good example of event is in Graphical User Interface in which it notifies when user does something with controls.
    - 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.
    - 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. A publisher class object invokes the event and it is notified to other objects.

  - Some other class that accepts this event is called the subscriber class. The delegate in the publisher class invokes the method of the subscriber class.
  - Events use the publisher-subscriber model.

## Accessibility Modifiers

- Access modifiers are used to specify the scope of accessibility of a member of a class or type of the class itself. There are six different types of access modifiers.
- Public :
  - Objects that implement public access modifiers are accessible from everywhere in a project without any restrictions.
- Private:
  - Objects that implement private access modifier are accessible only inside a class or a structure. As a result, we can't access them outside the class they are created.
- Protected:
  - The protected keyword implies that the object is accessible inside the class and in all classes that derive from that class.
- Internal:
  - For Internal keyword, the access is limited exclusively to classes defined within the current project assembly.
- Protected Internal:
  - The protected internal access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assemblies.
- Private Protected :

- The private protected access modifier is a combination of the private and protected keywords. We can access members inside the containing class or in a class that derives from a containing class, but only in the same assembly(project). Therefore, if we try to access it from another assembly, we will get an error.

## Namespace & .Net Library

- NET Framework Class Library is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.
- It contains thousands of classes that supports the following functions.
  - Base and user-defined data types
  - Support for exceptions handling
  - input/output and stream operations
  - Communications with the underlying system
  - Access to data
  - Ability to create Windows-based GUI applications
  - Ability to create web-client and server applications
  - Support for creating web services

| 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, System.Xml.Xsl** | These namespaces are used to create and access XML files. |

## Creating and adding ref. to assemblies

- Normally, manually adding assembly references to your project is not required, since Visual Studio is able to automatically add the appropriate references to the project as you place Infragistics tools onto the design surface. However, there will be times when you want or need to manually add a reference to an assembly to your project. Adding an assembly to your Visual Studio project is simple.

- right click on the References or Dependencies node in Solution Explorer and choose Add Reference. You can also right-click on the project node and select Add > Reference.
- We can add a reference to the following types of components and services:
    - .NET class libraries or assemblies
    - UWP apps
    - COM components
    - Other assemblies or class libraries of projects in the same solution
    - Shared projects
    - XML Web services

## 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.
- System.Collections namespace contains the non-generic collection types and System.Collections.Generic namespace includes generic collection types.

- Generic Collections :
    - List<T> : Generic List<T> contains elements of specified type. It grows automatically as you add elements in it.

    - Dictionary<TKey,TValue> : Dictionary<TKey,TValue> contains key-value pairs.
    - SortedList<TKey,TValue> : SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default.
    - Queue<T> : Queue<T> stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection.
    - Stack<T> : Stack<T> stores the values as LIFO (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values.
    - Hashset<T> : Hashset<T> contains non-duplicate elements. It eliminates duplicate elements.

- Non - Generic Collections :
    - ArrayList : ArrayList stores objects of any type like an array. However, there is no need to specify the size of the ArrayList like with an array as it grows automatically.
    - SortedList : SortedList stores key and value pairs. It automatically arranges elements in ascending order of key by default. C# includes both, generic and non-generic SortedList collection.
    - Stack : Stack stores the values in LIFO style (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. C# includes both, generic and non-generic Stack.
    - Queue : Queue stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. C# includes generic and non-generic : Queue.
    - Hashtable : Hashtable stores key and value pairs. It retrieves the values by comparing the hash value of the keys.
    - BitArray : BitArray manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0).