Smit Vora

Learnings and Conclusion

Module-3

Topics

14. Understanding Classes

14.1. Class types and usage

14.1.1. There are 4 types of classes abstract, sealed, static, partial.

14.1.2. It is used for representing different states and behaviour of objects used in them and also for performing tasks.

14.2. Static

14.2.1. It is created using “static” keyword.

14.2.2. A static class cannot be inherited.

14.2.3. It cannot be instantiated.

14.2.4. Its methods can be called directly by classname.

14.2.5. It is always sealed.

14.2.6. Syntax –

```
static class Information
{
}
```

14.3. Abstract

14.3.1. It is created using “abstract” keyword.

14.3.2. It cannot be instantiated.

14.3.3. If there is an abstract method in class it has to be abstract.

14.3.4. It doesn’t support multiple inheritance.

14.3.5. It can have both abstract and non-abstract methods.

14.3.6. It can be inherited for its abstract methods to be implemented.

14.3.7. Methods in abstract class cannot be private.

14.3.8. **Syntax** –

abstract class Information

{

}

14.4. Sealed

14.4.1. Sealed class objects cannot be inherited.

14.4.2. To obtain its objects we must instantiate it.

14.4.3. Access modifiers are not applied to sealed class.

14.4.4. **Syntax** -

sealed class Information

{

}

15. Depth in Classes

15.1. A class is a logical collection of similar kinds of objects. It is one of the most fundamental types in C#. It is basically a data structure that is a combination of Methods, Functions, and Fields. It provides the definition for the dynamic instances i.e. objects that need to be created for the class.

15.2. **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.

15.2.1. Ex – Business business = new Business ();

15.2.2. We created class Business and defined its object to be business.

15.3. **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.

15.3.1. **Characteristics about Constructor -**

15.3.2. A constructor can not be abstract, final, and synchronized.

15.3.3. Within a class, you can create only one static constructor.

15.3.4. A constructor doesn't have any return type, not even void.

15.3.5. A static constructor cannot be a parameterized constructor.

15.3.6. A class can have any number of constructors.

15.3.7. Access modifiers can be used in constructor declaration to control its access i.e. which other class can call the constructor.

15.3.8. **Types of Constructors –**

15.3.8.1. Default Constructor

15.3.8.2. Parameterized Constructor

15.3.8.3. Copy Constructor

15.3.8.4. Private Constructor

15.3.8.5. Static Constructor

15.3.9. **Syntax(Default)-**

```
class Business
{
    Public Business(){}
}
// A class business is created and its constructor is called by instantiating it.
Business business = new Business();
```

15.4. **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.

15.4.1. If a method doesn't return any value then the return types must be void.

15.4.2. 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.

15.4.3. There are also different types of parameters a method can get like value type, reference type, out, named, default.

15.4.4. **Syntax -** <access modifier><return-type><method-name>(parameters)

```
{
        //code
}
```

15.5. **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.

15.5.1. They are special methods called accessors. There are 2 types of accessors get and set.

15.5.2. Properties can be read only, read-write, write-only too.

15.5.3. It is not possible to have a same name for property and variable.

15.5.4. **Get Accessor** specifies the read only property and help us to access the value of the field publicly.

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

15.5.6. **Syntax -** <access modifier><return-type><property-name>

```
{
        get
        {
                //code
        }
        set
        {
                //code
        }
}
```

15.6. **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.

15.7. 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.

15.8. The publisher class object calls the event and it is notified to other objects as it consists of event definition.

15.9. The subscriber class object contains the event handler. The delegate in the publisher class invokes the method(event handler) of the subscriber class.

15.10. **Example –**

```
public delegate void VideoEncodedEventHandler(object source, EventArgs
args);
public event VideoEncodedEventHandler VideoEncoded;
```

16. **Scope Of Accessibility Modifiers** – It defines the accessibility of a class and its members by other classes or outside that particular class.

16.1. **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.

16.1.1. **Syntax** –

public <TypeName> <Name>

16.2. **protected** – The access is granted to only subclasses inside or outside of the same namespace or assembly which derived from the following class.

16.2.1. **Syntax** –

protected <TypeName> <Name>

16.3. **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.

16.3.1. **Syntax** –

internal <TypeName> <Name>

16.4. **private** – The access is granted only within the class which contains the private members not outside of that class.

16.4.1. **Syntax** –

private <TypeName> <Name>

17. **Namespace & .Net Library -** .Net Class Library is a base class library containing namespaces, interfaces, classes, value-types used in .Net Applications.

17.1. **They provide us with functions like-**

17.2. Database access queries.

17.3. Web services creation and operability.

17.4. Exception Handling

17.5. I/O operations

17.6. Predefined and User-defined datatypes.

17.7. Code Diagnostics

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,	These are used to create ASP. NET Web applications that run over the web.

System.Web.Hosting, System.Web.Mail, System.Web.SessionState	
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.

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.
    18.1. **Adding ref. to assemblies** – Ways to add reference to an assembly :
        18.1.1. To add specific DLL from the path instead of referring to class library projects.
        18.1.2. To add DLL from the bin folder of the class library project made of the same solution.
        18.1.3. To add assembly file reference from the .Net class library.
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. 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

19.1. There are two types Generic and Non-Generic types of collections. The key difference between them being that in Generic type same type of values are stored and in Non-Generic different types of values can be stored. Both don't have a fixed size and we can add and remove elements.

19.2. **ArrayList(Non-generic) and List(Generic)** – ArrayList is like array except it doesn't have a fixed size. Any no. of elements can be stored.

19.3. **HashTable(Non-generic) and Dictionary(Generic) –** Similar to ArrayList except it has a combination of key and value.

19.4. **SortedList(Non-generic) and SortedList (Generic) –** Represents the data as, a key and value pair.Arranges the items in sorted order.

19.5. **Stack(Non-generic) and Stack (Generic) and Queue(Non-generic) and Queue (Generic) -** Same as Stack and Queue data structures.

Generic Collections	Description
List<T>	Generic List<T> contains elements of specified type. It grows automatically as you add elements in it. **Syntax –** List<T> listName = new List<T>();
Dictionary<TKey,TValue>	Dictionary<TKey,TValue> contains key-value pairs. **Syntax –** Dictionary<TKey,TValue> dictionaryName= Dictionary<TKey,TValue> ();
SortedList<TKey,TValue>	SortedList stores key and value pairs. It automatically adds the elements in ascending order of key by default. **Syntax –** SortedList<TKey,TValue> sortedlistName = new SortedList<TKey,TValue> ();
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. **Syntax –** Queue<T> queueName = new Queue <T>();
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. **Syntax –** Stack<T> stackName = new Stack <T>();

Non-Generic Collections	Description
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. **Syntax –** ArrayList arraylistName = new ArrayList();
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. **Syntax –** SortedList sortedlistName = new SortedList ();
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.

	Syntax – Stack stackName = new 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. **Syntax –** Queue queueName = new Queue();
Hashtable	Hashtable stores key and value pairs. It retrieves the values by comparing the hash value of the keys. **Syntax –** HashTable hashtableName = new HashTable();