Module 3 Theory

*** For detailed theory/explanation along with examples/execution code , refer to the commented code in the practicals of the module ***

1) When we define a **class**, we define a blueprint for a data type. This does not actually define any data, but it does define what the class name means. That is, what an object of the class consists of and what operations can be performed on that object. Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces.

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 (from design perspective) and they are kept private to implement encapsulation. These variables can only be accessed using the public member functions.

2) Create a new project for **Class library**. Add the code in Class1.cs (created by default) . Build the class library project . Add reference of it's DLL file in our console app . Add "using ClassLibrary1;" statement at the top of our code in console app. Use it's classes in the console app as they were defined in the same file or namespace.

3) For **importing other namespace**, create a new project for Console App. Add the code in Program.cs (created by default) . Build the console app project . Add reference of it's EXE file in our console app . Add "using CA2=ConsoleApp2;" statement at the top of our code in console app. Alias is created so that we can specify full name of classes defined in ConsoleApp2 whenever we use that classes in our console app. Alias makes the task easier whenever namespace has long names. Moreover, using full name of classes helps to avoid collisions in name of classes.

4) **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:

- public
- protected
- internal
- protected internal
- private
- private protected

5) **Collection** classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.

Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types in C#.

The following are the various commonly used classes of the System.Collection namespace --

- ArrayList - It represents ordered collection of an object that can be indexed individually.It is basically an alternative to an array. However, unlike array you can add and remove items from a list at a specified position using an index and the array resizes itself automatically. It also allows dynamic memory allocation, adding, searching and sorting items in the list.
- Hashtable - It uses a key to access the elements in the collection.A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.
- SortedList - It uses a key as well as an index to access the items in a list.A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key , it is a Hashtable. The collection of items is always sorted by the key value.
- Stack - It represents a last-in, first out collection of object.It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.
- Queue - It represents a first-in, first out collection of object.It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called enqueue and when you remove an item, it is called deque.

6) **Reflection** objects are used for obtaining type information at runtime. The classes that give access to the metadata of a running program are in the System.Reflection namespace.

The System.Reflection namespace contains classes that allow you to obtain information about the application and to dynamically add types, values, and objects to the application.

Reflection has the following applications −

- It allows view attribute information at runtime.
- It allows examining various types in an assembly and instantiate these types.
- It allows late binding to methods and properties
- It allows creating new types at runtime and then performs some tasks using those types.

7) **Properties** are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called Fields. Properties are an extension of fields and are accessed using the same syntax. They use accessors through which the values of the private fields can be read, written or manipulated.

Properties do not name the storage locations. Instead, they have accessors that read, write, or compute their values.

8) An **indexer** allows an object to be indexed such as an array. When you define an indexer for a class, this class behaves similar to a virtual array. You can then access the instance of this class using the array access operator ([ ]).

9) A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

10) **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.

11) Delegates are used to reference any methods that has the same signature as that of the delegate. In other words, you can call a method that can be referenced by a delegate using that delegate object.

Anonymous methods provide a technique to pass a code block as a delegate parameter. Anonymous methods are the methods without a name, just the body.

You need not specify the return type in an anonymous method; it is inferred from the return statement inside the method body.