Module 4

1. Enumeration

- C# enum is a value type with a set of related named constants often referred as an enumerator list. The C# enum keyword is used to declare an enumeration. It is a primitive data type, which is user-defined. Enums type can be an integer (float, int, byte, double etc.) but if you use beside int, it has to be cast.

- Enums are enumerated data type in C#.
- Enums are not for end-users, they are meant for developers.
- Enums are strongly typed constants. They are strongly typed, i.e., an enum of one type may not be implicitly assigned to an enum of another type even though the underlying value of their members are the same.
- Enumerations (enums) make your code more readable and understandable.
- enum values are fixed. enum can be displayed as a string and processed as an integer.
- The default type is int, and the approved types are byte, sbyte, short, ushort, uint, long, and ulong.
- Every enum type automatically derives from System.Enum and thus we can use System.Enum methods on enums.
- Enums are value types and are created on the stack and not on the heap.

2. Handling Exception:

- Exception handling in C#, suppoted by the try catch and finaly block is a mechanism to detect and handle run-time errors in code.
- The exceptions are anomalies that occur during the execution of a program. They can be because of user, logic or system errors.
- C# provides three keywords try, catch and finally to implement exception handling. The try encloses the statements that might throw an exception whereas catch handles an exception if one exists.
- The finally can be used for any cleanup work that needs to be done.
- If any exception occurs inside the try block, the control transfers to the appropriate catch block and later to the finally block.
- But in C#, both catch and finally blocks are optional. The try block can exist either with one or more catch blocks or a finally block or with both catch and finally blocks.
- If there is no exception occurred inside the try block, the control directly transfers to finally block.
- In C#, exceptions are nothing but objects of the type Exception. The Exception is the ultimate base class for any exceptions in C#.
- A try block can throw multiple exceptions, which can handle by using multiple catch blocks.

- By providing a catch block without brackets or arguments, we can catch all exceptions occurred inside a try block.
- Some of the exceptions are:

- System.OutOfMemoryException
- System.NullReferenceException
- Syste.InvalidCastException
- Syste.ArrayTypeMismatchException
- System.IndexOutOfRangeException
- System.ArithmeticException
- System.DevideByZeroException
- System.OverFlowException

3. Event:

- C# and .NET supports event driven programming via delegates. Delegates and events provide notifications to client applications when some state changes of an object.

- It is an encapsulation of idea that "Something happened". Events and Delegates are tightly coupled concept because event handling requires delegate implementation to dispatch events.

- The class that sends or raises an event is called a Publisher and class that receives or handle the event is called "Subscriber".

- Event Handlers in C# return void and take two parameters.
- The First parameter of Event - Source of Event means publishing object.
- The Second parameter of Event - Object derived from EventArgs.
- The publishers determines when an event is raised and the subscriber determines what action is taken in response.
- An Event can have so many subscribers.
- Events are basically used for the single user action like button click.
- If an Event has multiple subscribers then event handlers are invoked synchronously.

4. File operations:

- A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream.

- The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the input stream and the output stream. The input stream is used for reading data from file (read operation) and the output stream is used for writing into the file (write operation).

- The System.IO namespace has various classes that are used for performing numerous operations with files, such as creating and deleting files, reading from or writing to a file, closing a file etc.

- The parent class of file processing is stream. Stream is an abstract class, which is used as the parent of the classes that actually implement the necessary operations.

Class Name	Description
FileStream	It is used to read from and write to any location within a file
BinaryReader	It is used to read primitive data types from a binary stream
BinaryWriter	It is used to write primitive data types in binary format
StreamReader	It is used to read characters from a byte Stream
StreamWriter	It is used to write characters to a stream.
StringReader	It is used to read from a string buffer
StringWriter	It is used to write into a string buffer
DirectoryInfo	It is used to perform operations on directories
FileInfo	It is used to perform operations on files

5. Interface & Inheritance:

- INHERITANCE:

- Acquiring the properties of one class into another class is called inheritance. Inheritance provides reusability by allowing us to extend an existing class.
- Base class - is the class from which features are to be inherited into another class.
- Derived class - it is the class in which the base class features are inherited.
- Basically there are 4 types of inheritance:

Single inheritance:

- It is the type of inheritance in which there is one base class and one derived class.

Multilevel inheritance :

- When one class is derived from another derived class then this type of inheritance is called multilevel inheritance.

Hierarchical inheritance :

- This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.

Multiple inheritance :

- C# does not support multiple inheritances of classes. To overcome this problem we can use interfaces.

INTERFACE:

- Inheritance allows creating classes that are derived from other classes, so that they automatically include some of its "parent's" members, plus its own.

- An interface looks like a class, but has no implementation. The only thing it contains are declarations of events, indexers, methods and/or properties.
- The reason interfaces only provide declarations is because they are inherited by structs and classes, that must provide an implementation for each interface member declared.

- Like **abstract classes**, interfaces **cannot** be used to create objects Interface methods do not have a body - the body is provided by the "implement" class.
- On implementation of an interface, you must override all of its methods
- Interfaces can contain properties and methods, but not fields/variables
- Interface members are by default abstract and public
- An interface cannot contain a constructor (as it cannot be used to create objects)