Phase-4

Name : Kotadiya Amisha K.

College : Government Engineering College, Rajkot

Branch : Computer Engineering

Understanding

Module-20

- **Enumerations**

➢ Enumration:

- ✓ In c#, enum is a keyword that is useful to declare an enumeration. In c#, the enumeration is a type that consists of a set of named constants as a list.

- ✓ By using an enumeration, we can group constants that are logically related to each other. For example, the days of the week can be grouped by using enumeration in c#.

- ✓ In c# by default, the first-named constant in the enumerator has a value of 0, and the value of each successive item in the enumerator will be increased by 1. For example, in the above enumeration, Sunday value is 0, Monday is 1, Tuesday is 2, and so forth.

- ✓ If we want to change the default values of an enumerator, then assigning a new value to the first item in the enumerator will automatically assign incremental values to the successive items in an enumerator.

- ✓ In c#, an enumeration can contain only integral data types such as byte, sbyte, short, ushort, int, uint, long or ulong. It's impossible to use an enum with string or any other data types to define enum elements except numeric types.

- ✓ The default type of enumeration element is int (integer). If you want to change the integral type of enum to byte, you need to mention a byte type with a colon (:) after the identifier

- ✓ Syntax:

```
enum enum_name
{
    // enumeration list
}
```

Module-21

- **Handling Exceptions**
  - List, Dictionary etc.

➢ Exception:

- ✓ Exceptions are generated by CLR (common language runtime) or application code. To handle runtime or unexpected errors in applications, c# has provided a built-in exception handling support by using try, catch, and finally blocks.

- ✓ In c#, when an exception is thrown, the CLR (common language runtime) will look for the catch block that handles the exception. If the currently executing method does not contain such a catch block, then the CLR will display an unhandled exception message to the user and stops the program's execution.

- ✓ Syntax:

```
try
{
   // code that may cause exception
}
catch (Exception ex)
{
   // exception handling
}
finally
{
   // clean up resources
}
```

- ✓ try − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

- ✓ catch − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

- ✓ finally − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.

- ✓ throw − A program throws an exception when a problem shows up. This is done using a throw keyword.

- ✓ The following table lists some of the important exception classes available in c#.

Exception	Description
ArgumentException	This exception will be thrown when one of the arguments provided to a method is not valid.
ArgumentNullException	This exception will occur when a null argument is passed to a method that does not accept it as a valid argument.
ArgumentOutOfRangeException	This exception will occur when the value of an argument is outside the allowable range of values.
ArithmeticException	This exception will occur for errors in arithmetic, casting, or conversion operation.
DivideByZeroException	This exception will occur when we try to divide an integral or decimal value by zero.
FormatException	This will occur when the format of an argument is invalid.
IndexOutOfRangeException	This will occur when we try to access an element of an array or collection with an index that is outside of its bounds.
NullReferenceException	This exception will occur when we try to reference an object which is of NULL type.
OutOfMemoryException	This will occur when the program is not having enough memory to execute the code.
OverflowException	This will occur when arithmetic, casting, or conversion operation results in an overflow.
TimeoutException	This will occur when the time allotted for a process or operation has expired.

Module-22

- **Events**

➢ 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. 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. Some other class that accepts this event is called the subscriber class. Events use the publisher-subscriber model.

- ✓ A publisher is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.

- ✓ A subscriber is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.

- ✓ To declare an event inside a class, first of all, you must declare a delegate type for the even as:
  public delegate string BoilerLogHandler(string str);

- ✓ then, declare the event using the event keyword −
  event BoilerLogHandler BoilerEventLog;

- ✓ The preceding code defines a delegate named BoilerLogHandler and an event named BoilerEventLog, which invokes the delegate when it is raised.

Module-23

- **Basic file operations**
  - Create, append, delete, copy, move etc.

➢ Basic File operation:

- ✓ File Handling refers to the various operations like creating the file, reading from the file, writing to the file, appending the file, etc. There are two basic operation which is mostly used in file handling is reading and writing of the file. In C#, System.IO namespace contains lasses which handle input and output streams and provide information about file and directory structure.

- ✓ StreamWriter Class

  The StreamWriter class implements TextWriter for writing character to stream in a particular format. The class contains the following method which are mostly used.

Method	Description
Close()	Closes the current StreamWriter object and stream associate with it.
Flush()	Clears all the data from the buffer and write it in the stream associate with it.
Write()	Write data to the stream. It has different overloads for different data types to write in stream.
WriteLine()	It is same as Write() but it adds the newline character at the end of the data.

- ✓ StreamReader Class

  The StreamReader class implements TextReader for reading character from the stream in a particular format. The class contains the following method which are mostly used.

Method	Description
Close()	Closes the current StreamReader object and stream associate with it.
Peek()	Returns the next available character but does not consume it.
Read()	Reads the next character in input stream and increment characters position by one in the stream
ReadLine()	Reads a line from the input stream and return the data in form of string
Seek()	It is use to read/write at the specific location from a file

- ✓ Copying file

  If there is a need to copy file in our program then we can use File.Copy(sourceFileLoc,destFileLoc).
  We have to provide source file and destination file location.

- ✓ Deleting file

  We use File.Delete(string filePath) in c# to delete a file

- ✓ Appending file

  If there is a need to copy file in our program then we can use File.Copy(sourceFileLoc,destFileLoc).
  We have to provide source file and destination file location.

Module-24

- **Interface & Inheritance**

➢ Interfaces

- ✓ Like abstract classes, interfaces cannot be used to create objects (in the example above, it is not possible to create an "IAnimal" object in the Program class).
- ✓ 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)

Why And When To Use Interfaces?

1) To achieve security - hide certain details and only show the important details of an object (interface).

2) C# does not support "multiple inheritance" (a class can only inherit from one base class). However, it can be achieved with interfaces, because the class can implement multiple interfaces. Note: To implement multiple interfaces, separate them with a comma (see example below).

Syntax for Interface declaration

```
interface  <interface_name >
{
   // declare Events
   // declare indexers
   // declare methods
   // declare properties
}
```

Syntax for Implementing Interface

```
class class_name : interface_name
```

➢ Inheritance

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in C# by which one class is allowed to inherit the features(fields and methods) of another class.

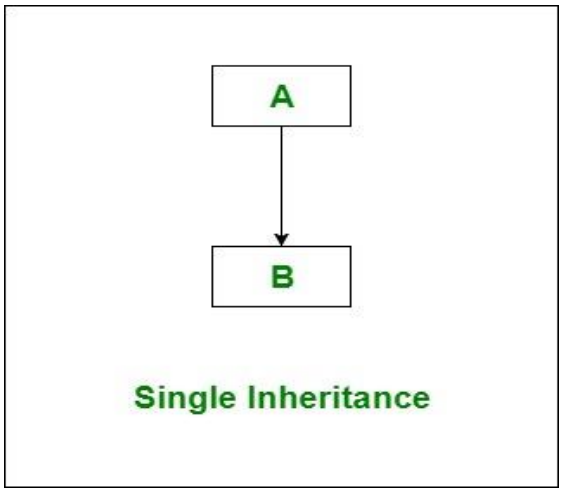
Syntax:

```
class derived-class : base-class
{
  // methods and fields
}
```

Below are the different types of inheritance which is supported by C# in different combinations.

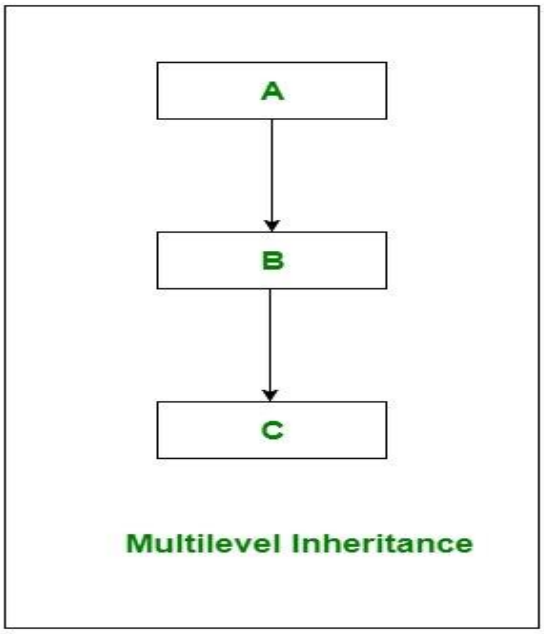
Single Inheritance:

In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



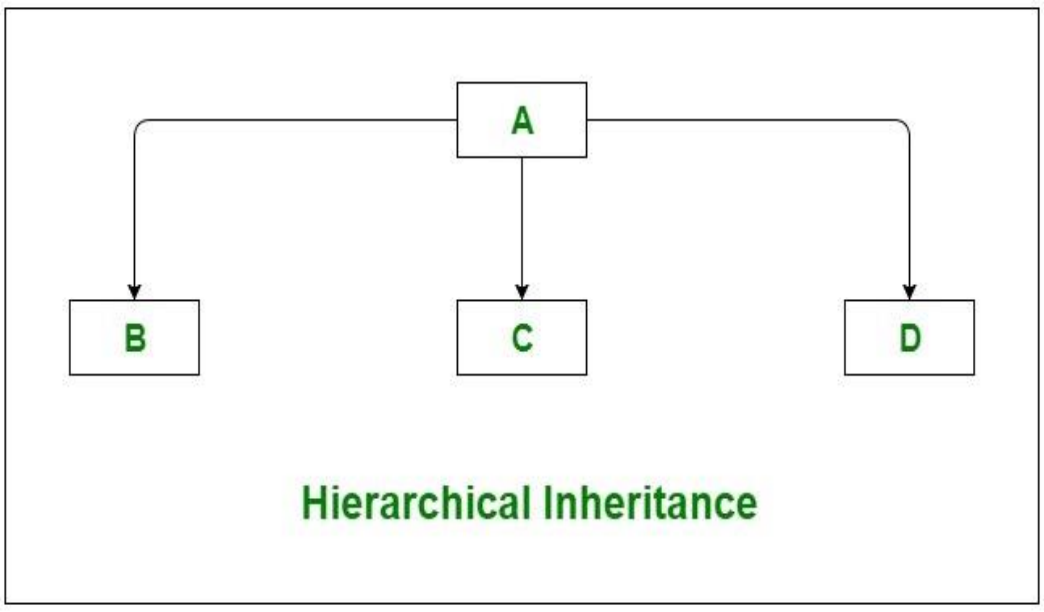
Multilevel Inheritance:

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.



Hierarchical Inheritance:

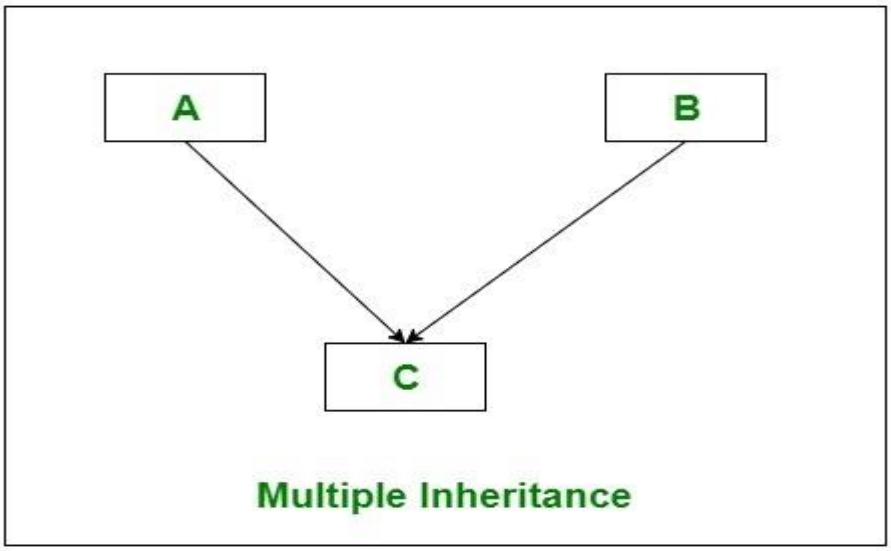
In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In below image, class A serves as a base class for the derived class B, C, and D.



s

Multiple Inheritance(Through Interfaces):

In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that C# does not support multiple inheritance with classes. In C#, we can achieve multiple inheritance only through Interfaces. In the image below, Class C is derived from interface A and B.



Hybrid Inheritance(Through Interfaces): It is a mix of two or more of the above types of inheritance. Since C# doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In C#, we can achieve hybrid inheritance only through Interfaces.