## Lecture # 3:

**Objective:**

* Class
* Functions
* Inheritance

## Classes:

A class is a construct that enables you to create your own custom types by grouping together variables of other types, methods and events. A class is like a blueprint. It defines the data and behavior of a type.

Classes are declared by using the class keyword, as shown in the following example.

public class Customer

{

//Fields, properties, methods and events go here...

}

The class keyword is preceded by the access level. Because ***public*** is used in this case, anyone can create objects from this class. The name of the class follows the class keyword. The remainder of the definition is the class body, where the behavior and data are defined. Fields, properties, methods, and events on a class are collectively referred to as class members.

Customer object1 = new Customer();

## Constructors in C#

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type. Following example explains the concept of constructor:

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:

## Functions:

Block of code in specific format which may reuse again and again in project when required just writing once.

A function is defined with these characteristics:

* Access level - public, private plus some others
* Return value - void or any type such as int
* Method Name - MyGrade
* Any method parameters defined in the brackets () after the method name

<visibility> <return type> <name>(<parameters>)

{ <function code> }

Let's define a function MyGrade ().

public string MyGrade ()

{

return “Grade A”;

}

## Passing Parameters:

In C#, arguments can be passed to parameters either by value or by reference. Passing by reference enables function members, methods to change the value of the parameters and have that change persist in the calling environment. To pass a parameter by reference, use the ref keyword.

class Program

{

static void Main(string[] args)

{

int arg;

// Passing by value.

// The value of arg in Main is not changed. arg = 4;

squareVal(arg); Console.WriteLine(arg);

// Output: 4

// Passing by reference.

// The value of arg in Main is changed. arg = 4;

squareRef(ref arg); Console.WriteLine(arg);

// Output: 16

}

static void squareVal(int valParameter)

{

valParameter \*= valParameter;

}

// Passing by reference

static void squareRef(ref int refParameter)

{

refParameter \*= refParameter;

}

}

## Function overloading:

Function having same name but different parameters with in same class is known as function overloading.

public static int Compare(string strA, string strB);

public static int Compare(string strA,string strB,bool ignoreCase);

## Inheritance:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

# Base Class:

namespace Inheritance

{

class SuperClass

{

public virtual void doSomeThing()

{

Console.WriteLine("In super class");

}

}

}

# Derived Class:

namespace Inheritance

{

class SubClass:SuperClass

{

public override void doSomeThing()

{

base.doSomeThing(); Console.WriteLine("In sub class");

}

}

}

# Main Class:

namespace Inheritance

{

class Program

{

static void Main(string[] args)

{

SuperClass objsc = new SubClass(); objsc.doSomeThing();

}

}

}

