Inheritance

Course Code : CS-127

Course Title : Object Oriented Programming

Semester : 2nd

C# Inheritance

- In c#, Inheritance is one of the primary concept of object-oriented programming (OOP) and it is used to inherit the properties from one class (base) to another (child) class.

- The inheritance will enable us to create a new class by inheriting the properties from other classes to reuse, extend and modify the behavior of other class members based on the requirements.

- In c# inheritance, the class whose members are inherited is called a base (parent) class and the class that inherits the members of base (parent) class is called a derived (child) class.

Advantage of C# Inheritance

"***Code reusability:*** Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class"

C# Single Level Inheritance

Inheriting Fields and Methods

- When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields and methods .

Single Level Inheritance Example

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication60
{
   class Employee
   {
      public float salary = 4000;
   }
   class Programmer : Employee
   {
      public float bonus = 3433;
   }

   class Program
   {
      static void Main(string[] args)
      {
         Programmer pr = new Programmer();
         Console.WriteLine(" Salary :  "+pr.salary);
         Console.WriteLine(" Bonus : "+pr.bonus);
         Console.Read();
      }
   }
}
```

Access Modifiers in C#

Accessibility / keyword	With in a class	Inheritance (In derived class)	Within same Application (Outside a class)	Anywhere Outside the Application
Private	Yes	No	No	No
Protected	Yes	Yes	No	No
Internal	Yes	Yes	Yes	No
Protected Internal	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

Order Of Execution Of The Constructors

- To demonstrate the order of execution of the constructors, examine the following sample code. The first class is the main Program class within a console application.

- It simply instantiates a MySubclass object. MyBaseClass is a class with a single constructor that outputs a message to the console. MySubclass inherits from MyBaseClass and also outputs to the console when constructed.

- Note the order in which the messages appear.

```
class Program
{
    static void Main()
    {
        MySubclass test = new MySubclass();
    }
}
class MyBaseClass
{
    public MyBaseClass()
    {
        Console.WriteLine("MyBaseClass constructor called.");
    }
}
class MySubclass : MyBaseClass
{
    public MySubclass()
    {
        Console.WriteLine("MySubclass constructor called.");
    }
}

/* OUTPUT
MyBaseClass constructor called.
MySubclass constructor called.
 */
```

Order Of Execution Of The Constructors

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication60
{
   class User
   {
      public static string name;
      public string location;
      public void displayuserinfo()
      {
         Console.WriteLine(" Name : "+name);
         Console.WriteLine(" Location : "+location);
      }
      public User()
      {
         Console.WriteLine("Parent (Base) class
constructor invoked...");
      }
   }
```

```
class details : User
   {
      public int salary;
      public void displaysalary()
      {
         Console.WriteLine(" Salary : " + salary);
      }
      public details(int number)
      {
         Console.WriteLine(" Child Class Constructor invoked..." +
number);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         details d = new details(387564);
         details.name = "Ali Ahmed";
         d.location = "Karachi";
         d.salary = 2324;
         d.displayuserinfo();
         d.displaysalary();
         Console.Read();
      }
   }
}
```

C# Multi-Level Inheritance

- When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C#.

- Inheritance is transitive so the last derived class acquires all the members of all its base classes.

- Let's see the example of multi level inheritance in C#.

- For example, suppose if class **C** is derived from class **B**, and class **B** is derived from class **A**, then class **C** inherits the members declared in both class **B** and class **A**.

```
public class A
{
// Implementation
}
public class B : A
{
// Implementation
}
public class C : B
{
// Implementation
}
```

- class **C** is derived from class **B**, and class **B** is derived from class **A**, then class **C** inherits the members declared in both class **B** and class **A**. This is how we can implement **multi-level** inheritance in our applications.

Multi-Level Inheritance Example

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication60
{
   class A
   {
      public string name;
      public void displayname()
      {
         Console.WriteLine(" Name : "+name);
      }
   }
   class B : A
   {
      public string location;
      public void displaylocation()
      {
         Console.WriteLine("Location : "+location);
      }
   }
   class C : B
   {
      public int salary;
      public void displaysalary()
      {
         Console.WriteLine("Salary : "+salary);
      }}
   class D : C
      {
         public int age;
         public void displayage()
         {
            Console.WriteLine(" Age : "+age);
         }
      }
      class Program
      {
         static void Main(string[] args)
         {
            D d = new D();
            d.name = "Ikrama";
            d.location = "Islamabad";
            d.salary = 3533;
            d.age = 65;
            d.displayname();
            d.displaysalary();
            d.displayage();
            d.displaylocation();
            Console.Read();
         }
      }
   }
```

The sealed Keyword

If we don't want other classes to inherit from a class, use the sealed keyword:

If you try to access a `sealed` class, C# will generate an error:

```
sealed class Vehicle
{
  ...
}

class Car : Vehicle
{
  ...
}
```

The error message will be something like this:

```
'Car': cannot derive from sealed type 'Vehicle'
```