C# | 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.

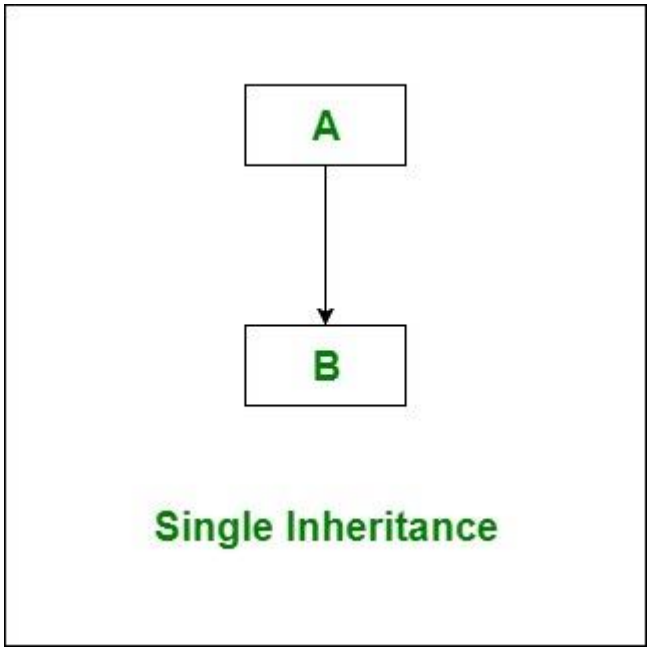
Important terminology:

- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Types of Inheritance in C#

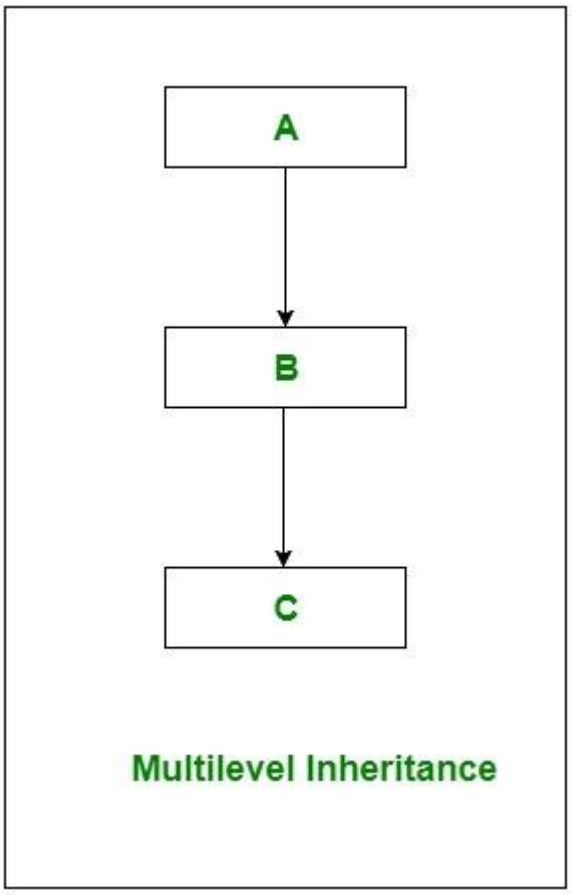
Below are the different types of inheritance which is supported by C# in different combinations.

1. **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.

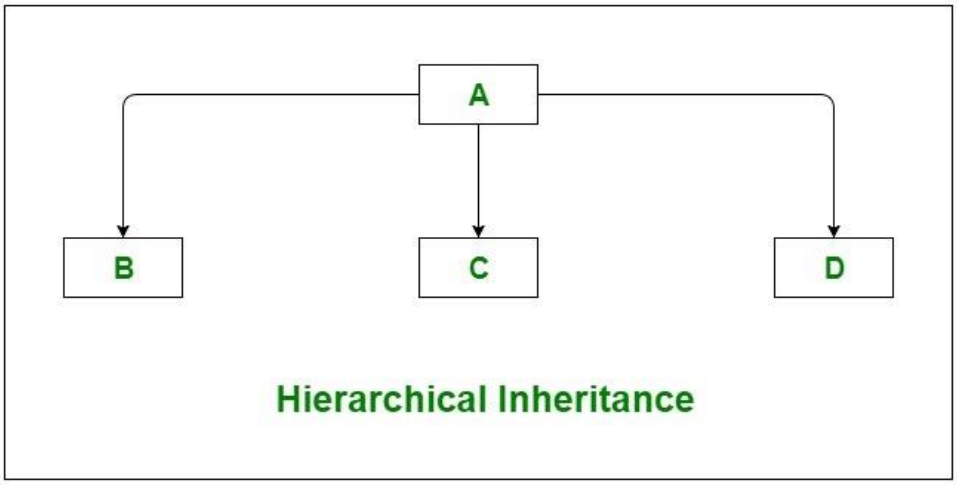


2. **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class

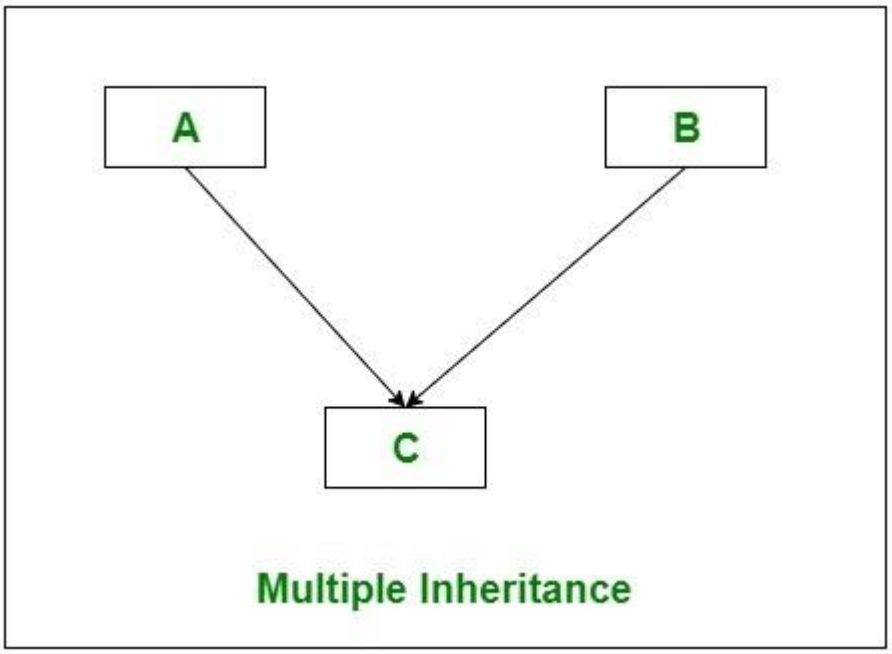
to other class. In below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.

4. 

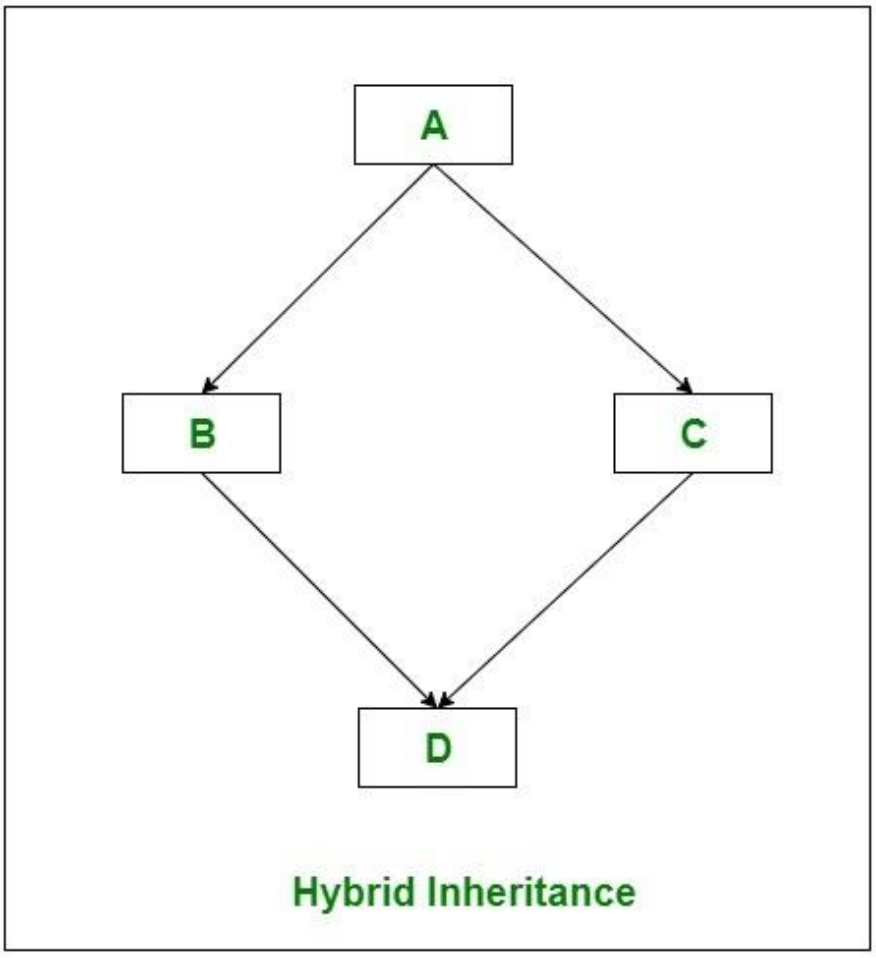
5. **Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In below image, class A serves as a base class for the derived class B, C, and D.

6. 

7. **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.

8. 

9. **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.

10. 

Important facts about inheritance in C#

- **Default Superclass**: Except Object class, which has no superclass, every class has one and only one direct superclass(single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object class.
- **Superclass can only be one:** A superclass can have any number of subclasses. But a subclass can have only **one** superclass. This is because C# does not support multiple inheritance with classes. Although with interfaces, multiple inheritance is supported by C#.
- **Inheriting Constructors:** A subclass inherits all the members (fields, methods) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
- **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the superclass has properties(get and set methods) for accessing its private fields, then a subclass can inherit.

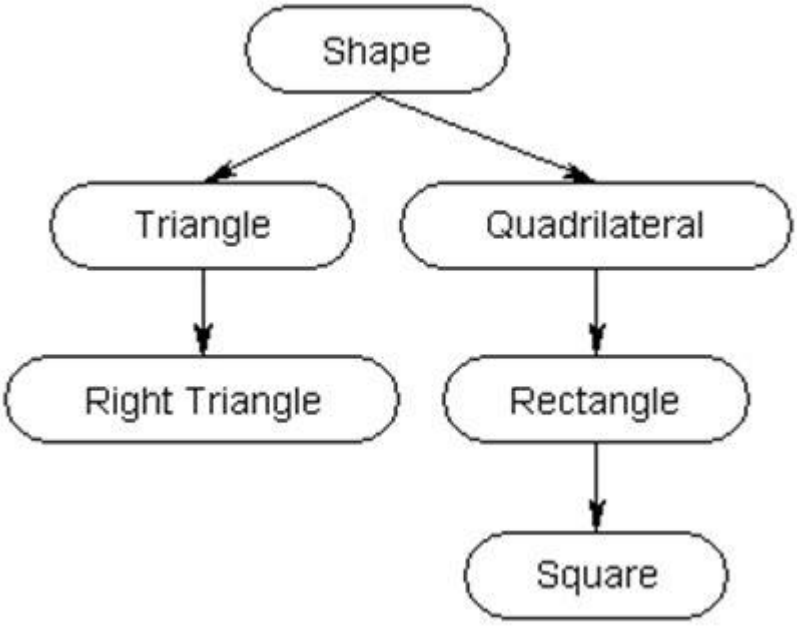
Real-world Example
A scientific calculator is an extended form of a calculator. Here calculator is the parent and scientific calculator is child object.

Programming Example
The process of acquiring the existing functionality of the parent and with the new added features and functionality of a child object.

Inheritance Example

Diagram

The following diagram shows the inheritance of a shape class. Here the base class is Shape and the other classes are its child classes. In the following program we will explain the inheritance of the Triangle class from the Shape class.




```
class Shape
   {
     public double Width;
     public double Height;
     public void ShowDim()
```

```
            {
                Console.WriteLine("Width and height are " +
                      Width + " and " + Height);
        }
    }

    class Triangle : Shape
         {
           public string Style;
          public double Area()
             {
                  return Width * Height / 2;
              }
public void ShowStyle()
          {
         Console.WriteLine("Triangle is " + Style);
          }
    }

    class Driver
         {
        static void Main()
        {
            Triangle t1 new Triangle();
            Triangle t2 new Triangle();
            t1.Width =4.0;
            t1.Height =4.0;
            t1.Style ="isosceles";
            t2.Width =8.0;
            t2.Height =12.0;
```

```
            t2.Style ="right";
            Console.WriteLine("Info for t1: ");
            t1.ShowStyle();
            t1.ShowDim();
            Console.WriteLine("Area is " + t1.Area());
            Console.WriteLine();
            Console.WriteLine("Info for t2: ");
            t2.ShowStyle();
            t2.ShowDim();
            Console.WriteLine("Area is " + t2.Area());
        }
    }
```

DEFINE BASE CLASS CONSTRUCTOR

There is no special rule for defining base class **constructors**. Rules are same as other class constructors. You can define number of constructor as follow

```
class baseclass
{
public baseclass()
        {
        }
public baseclass(string message)
        {
        }
}
```

ACCESS BASE CLASS CONSTRUCTOR IN DERIVED CLASS

If base class has constructor then child class or derived class are required to call the constructor from its base class.

```
class childclass : baseclass
{
        public childclass()
        {
        }
public childclass(string message): base(message)
        {
        }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Inheritance_Constructors
{
        class Program
        {
```

```
        static void Main(string[] args)
        {
        childclass ch = new childclass();
        childclass ch1 = new childclass("Hello Constructor");
        Console.ReadKey();
        }
}
 class baseclass
 {
    public baseclass()
 {
      Console.WriteLine("I am Default Constructors");
    }
public baseclass(string message)
  {
    Console.WriteLine("Constructor Message : " + message);
   }
}
class childclass : baseclass
     {
        public childclass()
        {
        }
public childclass(string message): base(message)
{
        }
        }
}
```