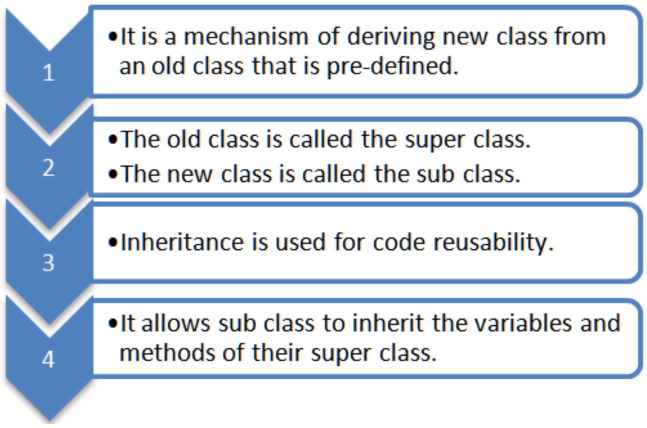
OOP (Object Oriented Programming) with C#: Inheritance –by Akhil Mittal

Table of Contents

Introduction

In our first part of the article, we learned about different scenarios of method overloading and did lots of interesting hands on too. My article in the second part of the series will focus solely on inheritance concept in OOP. Let’s define Inheritance using some bullet points,



1
•It is a mechanism of deriving new class from an old class that is pre-defined.
2
•The old class is called the super class.
•The new class is called the sub class.
3
•Inheritance is used for code reusability.
4
•It allows sub class to inherit the variables and methods of their super class.

Inheritance in Action:

OK. Let’s do some hands on. Create a console application and name it **InheritanceAndPolymorphism.** Add a class named ClassA and a class named ClassB, with the following code,

ClassA:

```
class ClassA
 {

 }
```

ClassB:

```
class ClassB
{
    public int x = 100;
    public void Display1()
    {
        Console.WriteLine("ClassB Display1");
    }
    public void Display2()
    {
        Console.WriteLine("ClassB Display2");
    }
}
```

We see class ClassA is empty and we added two methods in class ClassB i.e. Display1 and Display2. We also have a variable x declared and defined with a value 100.
Now in the main method of Program.cs, write the following code,

Program.cs:

```
class Program
 {
     static void Main(string[] args)
     {

         ClassA a = new ClassA();
         a.Display1();
     }
 }
```

If we run the code, we immediately result in the compile time error.

Error: 'InheritanceAndPolymorphism.ClassA' does not contain a definition for 'Display1' and no extension method 'Display1' accepting a first argument of type 'InheritanceAndPolymorphism.ClassA' could be found (are you missing a using directive or an assembly reference?)

i.e. too obvious, we don't have definition of Display1 method in ClassA, and nor can we access the same method using ClassA instance because it is not derived from any such class like ClassB that contains Display1 method. The class ClassA does not contain any code or variable defined . An empty class does not throw any error as we are able to instantiate an object that looks like a (instance of ClassA). The error
comes about because the class ClassA has no method called Display1. However the class ClassB has a method named Display1.Guess how fun it could be if we are allowed to access all the code of classB from ClassA itself.

Just derive the class ClassA from ClassB using : operator as code shown below,

ClassA:

```
  class ClassA:ClassB
    {

    }
```

ClassB:

```
class ClassB
    {
        public int x = 100;
        public void Display1()
        {
            Console.WriteLine("ClassB Display1");
        }
        public void Display2()
        {
            Console.WriteLine("ClassB Display2");
        }
    }
```

Program.cs:

```
class Program
    {
        static void Main(string[] args)
        {
            ClassA a = new ClassA();
            a.Display1();
            Console.ReadKey();
        }
    }
```

And now run the code as it was, we get an output now,

Output:

```
ClassB Display1
```

i.e. now ClassA can access the inherited public methods of ClassB. The error vanishes and the Display1 in ClassB gets invoked. If after the name of a class we specify **: ClassB** i.e. the name of another class, a lot changes at once. ClassA is now said to have been derived from ClassB. What that means is all the code we wrote in ClassB can now be accessed and used in ClassA. It is if we actually wrote all the code that is contained in ClassB in ClassA. If we had created an instance that looks like that of ClassB, everything that the instance could do, now an instance of ClassA can also do. But we have not written a line of code in ClassA. We are made to believe that ClassA has one variable x and two functions Display1 and Display2 as ClassB contains these two functions. Therefore we enter into the concepts of inheritance where ClassB is the base class, ClassA the derived class.

Child inherit properties from father
Derived class inherits properties from its base class

Let’s take another scenario. Suppose we get into a situation where ClassA also have a method of same name as of in ClassB. Let’s define a method Derive1 in ClassA too, so our code for classA becomes,

```
class ClassA:ClassB
    {
        public void Display1()
        {
            System.Console.WriteLine("ClassA Display1");
        }
    }

ClassB:

class ClassB
    {
        public int x = 100;
        public void Display1()
        {
            Console.WriteLine("ClassB Display1");
        }
        public void Display2()
        {
            Console.WriteLine("ClassB Display2");
        }
    }
```

Now if we run the application using following code snippet for Program.cs class,

```
class Program
    {
        static void Main(string[] args)
        {
            ClassA a = new ClassA();
            a.Display1();
            Console.ReadKey();
        }
    }
```

The question is what will happen now? What will be the output? Will there be any output or any compilation error. Ok, let's run it,

We get Output,

```
ClassA Display1
```

But did you notice one thing, we also got a warning when we run the code,

Warning: 'InheritanceAndPolymorphism.ClassA.Display1()' hides inherited member 'InheritanceAndPolymorphism.ClassB.Display1()'. Use the new keyword if hiding was intended.

Point to remember: *No one can stop a derived class to have a method with the same name already declared in its base class.*

So, ClassA undoubtedly can contain Display1 method, that is already defined with the same name in ClassB. When we invoke a.Display1(), C# first checks whether the class ClassA has a mehtod named Display1. If it does not find it, it checks in the base class. Earlier Display1 method was only available in the base class ClassB and hence got executed. Here since it is there in ClassA, it gets called from ClassA and not ClassB.

Point to remember: *Derived classes get a first chance at execution, then the base class.*

The reason for this is that the base class may have a number of mehtods and for various reasons we may not be satisfied with what they do. We should have the full right to have our copy of the method to be called. In other words the derived classes methods **override** the ones defined in the base class.

What happens if we call base class Display1 method too with base keyword in derived class i.e. by using base.Display1(), so our ClassA code will be,

ClassA:

```
  class ClassA:ClassB
    {
        public void Display1()
        {
            Console.WriteLine("ClassA Display1");
            base.Display1();
        }
    }
```

```
ClassB:

class ClassB
    {
        public int x = 100;
        public void Display1()
        {
            Console.WriteLine("ClassB Display1");
        }
        public void Display2()
        {
            Console.WriteLine("ClassB Display2");
        }
    }
```

Program.cs:

```
class Program
    {
        static void Main(string[] args)
        {
            ClassA a = new ClassA();
            a.Display1();
            Console.ReadKey();
        }
    }
```

Output:

```
ClassA Display1
ClassB Display1
```

We see here first our ClassA Display1 method is called and then ClassB Display1 method.

Now if you want the best of both the classes , you may want to call the base classes (ClassB) Display1 first and then yours or vice versa. To achieve this, C# gives you a free keyword, called base. The keyword base can be used in any of derived class. It means call the method off the base class. Thus base.Display1 will call the method Display1 from ClassB the base class of ClassA as defined earlier.

Point to remember: *A reserved keyword named "base" can be used in derived class to call the base class method.*

What if we call Display2 method from base class, with an instance of derived class ClassA?

```
   /// <summary>
   /// ClassB: acting as base class
   /// </summary>
   class ClassB
    {
        public int x = 100;
        public void Display1()
        {
            Console.WriteLine("ClassB Display1");
        }
        public void Display2()
        {
            Console.WriteLine("ClassB Display2");
        }
    }

    /// <summary>
    /// ClassA: acting as derived class
    /// </summary>
    class ClassA : ClassB
    {
        public void Display1()
        {
            Console.WriteLine("ClassA Display1");
            base.Display2();
        }
    }

    /// <summary>
    /// Program: used to execute the method.
    /// Contains Main method.
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {
            ClassA a = new ClassA();
            a.Display1();
            Console.ReadKey();
        }
    }
```

Output:

```
ClassA Display1
ClassB Display2
```

In the above code we only made just a small change, base.Display1 was replaced by base.Display2.In this particular scenario method Display2 from the class ClassB gets called. Base is usually a very general purpose. It lets us access members of the base class from the derived class as explained earlier. We cannot use base in ClassB as ClassB is not derived from any class as per our code. So its done that the base keyword can only be used in derived classes ☺.

Let take another case,

```
   /// <summary>
   /// ClassB: acting as base class
   /// </summary>
   class ClassB
    {
        public int x = 100;
        public void Display1()
        {
            Console.WriteLine("ClassB Display1");
        }
    }

    /// <summary>
    /// ClassA: acting as derived class
    /// </summary>
    class ClassA : ClassB
    {
        public void Display2()
        {
            Console.WriteLine("ClassA Display2");
        }
    }
    /// <summary>
    /// Program: used to execute the method.
    /// Contains Main method.
    /// </summary>
    class Program
    {
        static void Main(string[] args)
        {
            ClassB b = new ClassB();
            b.Display2();
            Console.ReadKey();
        }
    }
```

<u>Output:</u>

Error: 'InheritanceAndPolymorphism.ClassB' does not contain a definition for 'Display2' and no extension method 'Display2' accepting a first argument of type 'InheritanceAndPolymorphism.ClassB' could be found (are you missing a using directive or an assembly reference?)

Point to remember: *Inheritance does not work backwards.*

So we got an error. Since we see, ClassA is derived from ClassB i.e. ClassB is base class. Therefore class ClassA can use all the members of class ClassB. Inheritance does not have backwords compatibility,whatever members ClassA contains do not permeate upwards to ClassB. When we tried to access Display2 method of classA from the instance of class ClassB ,it cannot give it to class ClassB and thus an error occurs.

Point to remember: *Except constructors and destructors,a class inherits everything from its base class .*

If a class ClassC is derived from class ClassB, which in turn has been derived from class ClassA, then ClassC will inherit all the members declared in ClassB and also of ClassA. This is called transitive concept in inheritance. A derived class may inherit all the members of the base class but it cannot remove members off that base class. A derived class can however hide members of the base class by creating methods by the same name. The original member/method of the base class remains unmodified and unaffected by whatever happens in the derived class. It remains unchanged in the base class, i.e. simply not visible in the derived class.

A class member could be of two type i.e. either a static member that directly belongs to a class or an instance member that is accessed through instance of that class and belongs to that particular instance only. Instance member is accessible only through the object of the class and not directly by the class. The default member declared in the class are non static, we just have to make them static by using static keyword.
All classes derive from a common base class named object .So Object is the mother of all classes.

If we do not derive any class from any other class, its responsibility of c# to add :object by itself to the class definition.Object is the only class that is not derived from any other class.It is the ultimate base class for all the classes.

Suppose ClassA is derived from ClassB as in our case, but ClassB is not derived from any class,

```
   public class ClassB
    {
    }

    public class ClassA : ClassB
    {
    }
```

C# automatically adds :object to ClassB i.e., the code at compile time becomes,

```
    public class ClassB:object
    {
    }

    public class ClassA : ClassB
    {
    }
```

But as per theory we say ClassB is the direct base class of ClassA,so the classes of ClassA are ClassB and object.

Let's go for another case,

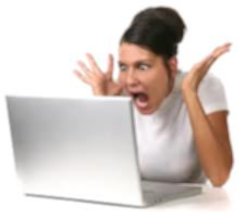
```
  public class ClassW : System.ValueType
    {
    }

    public class ClassX : System.Enum
    {
    }

    public class ClassY : System.Delegate
    {
    }

    public class ClassZ : System.Array
    {
    }
```

Here we have defined four classes, each derive from a built in class in c#,lets run the code.

We get so many compile time errors,

Errors:
'InheritanceAndPolymorphism.ClassW' cannot derive from special class 'System.ValueType'
'InheritanceAndPolymorphism.ClassX' cannot derive from special class 'System.Enum'
'InheritanceAndPolymorphism.ClassY' cannot derive from special class 'System.Delegate'
'InheritanceAndPolymorphism.ClassZ' cannot derive from special class 'System.Array'

Don't be scared,



Did you notice the word **special class.** Our classes defined can not inherit from special built in classes in c#.

Point to remember: *In inheritance in C#, custom classes can not derive from special built in c# classes like System.ValueType, System.Enum,System.Delegate,System.Array etc.*

One more case,

```
  public class ClassW
    {
    }

    public class ClassX
    {
    }

    public class ClassY : ClassW, ClassX
    {
    }
```

In the above mentioned case, we see three classes, ClassW, ClassX and ClassY.ClassY is derived from ClassW and ClassX.Now if we run the code, what would we get?

Compile time Error: `Class 'InheritanceAndPolymorphism.ClassY' cannot have multiple base classes: 'InheritanceAndPolymorphism.ClassW' and 'ClassX'`

So one more **Point to remember:** *A class can only be derived from one class in C#.C# does not support multiple inheritance by means of class*.*

**Multiple inheritance in C# can be accomplished by the use of Interfaces,we are not discussing about interfaces in this article.*

We are not allowed to derive from more than one class, thus every class can have only one base class.

Another case,

Suppose we try to write a code as below,

```
 public class ClassW:ClassY
  {
  }

  public class ClassX:ClassW
  {
  }

  public class ClassY :  ClassX
  {
  }
```

Code is quite readable and simple, ClassW is derived from ClassY, ClassX is derived from ClassW, and ClassY in turn is derived from ClassX.So no problem of multiple inheritance, so our code should build successfully, let’s compile the code.What do we get? Again a compile time error,

Error: Circular base class dependency involving 'InheritanceAndPolymorphism.ClassX' and 'InheritanceAndPolymorphism.ClassW'

Point to remember:*Circular dependency is not allowed in inheritance in C#.ClassX is derived from ClassW which was derived from ClassY and ClassY was again derived from ClassX, which caused circular dependency in three classes, that is logically impossible.*

Equalizing the Instances/Objects:

Let's directly start with a real case,

ClassB:
```
public class ClassB
    {
        public int b = 100;
    }
```

ClassA:
```

    public class ClassA
    {
        public int a = 100;
    }
```

Program.cs:
```

    /// <summary>
    /// Program: used to execute the method.
    /// Contains Main method.
    /// </summary>
    public class Program
    {
        private static void Main(string[] args)
        {
            ClassB classB = new ClassB();
            ClassA classA = new ClassA();
            classA = classB;
            classB = classA;
        }
    }
```

We are here trying to equate two objects or two instances of two different classes.Let's compile the code, We get compile time error,

Error:
Cannot implicitly convert type 'InheritanceAndPolymorphism.ClassB' to 'InheritanceAndPolymorphism.ClassA'
Cannot implicitly convert type 'InheritanceAndPolymorphism.ClassA' to 'InheritanceAndPolymorphism.ClassB'

InheritanceAndPolymorphism is the namespace that I used for my console application, so no need to be scared of that word, just ignore it.

C# works on rules, it will never allow you to equate objects of different independent classes to each other.Therefore we can not equate an object classA of ClassA to classB of ClassB or vice versa.No matter the classes contains similar structure and their variables are initialized to similar integer value, even if we do,

```
public class ClassB
 {
     public int a = 100;
 }

 public class ClassA
 {
     public int a = 100;
 }
```

I just changed int b of ClassB to int a.In this case too, to equate an object is not allowed and not possible.

C# is also very particular if it comes with dealing with data types.

There is however one way to to this. By this way which we'll discuss, one of the errors will disappear. The only time we are allowed to equate dissimilar data types is only when we derive from them ☺.Check out the code mentioned below. Let's discuss this in detail,when we create an object of ClassB by declaring new, we are creating two objects at one go, one that looks like ClassB and the other that looks like object i.e. derived from Object class (i.e. ultimate base class). All classes in C# are finally derived from object. Since ClassA is derived from ClassB,when we declare new ClassA, we are creating 3 objects, one that looks like ClassB, one that looks like ClassA and finally that looks like object class.

```
public class ClassB
 {
     public int b = 100;
 }

 public class ClassA:ClassB
 {
     public int a = 100;
 }

 /// <summary>
 /// Program: used to execute the method.
 /// Contains Main method.
 /// </summary>
 public class Program
 {
     private static void Main(string[] args)
     {
         ClassB classB = new ClassB();
```

```
            ClassA classA = new ClassA();
            classA = classB;
            classB = classA;
        }
    }
```

```
We just derived ClassA from ClassB, this is something we can do, we learned a lot about this in this
article.Now compile the code, we get,
```

`Error: Cannot implicitly convert type 'InheritanceAndPolymorphism.ClassB' to 'InheritanceAndPolymorphism.ClassA'. An explicit conversion exists (are you missing a cast?)`

Like I mentioned C# is very particular about objects equating.

Thus when we write `classA = classB`, classA looks like ClassA,ClassB and object and as a looks like ClassB, there is a match at ClassB.Result? No error :-) . Even though classB and classA have the same values, using classB we can only access the members of ClassB, even though had we used classA we could access ClassA also. We have devalued the potency of classB . The error occurs at classA = classB, because the class ClassB is less/smaller than the class ClassA . The class ClassA has ClassB and more. We cannot have a larger class on the right and a smaller class on the left. classB only represents a ClassB whereas classA expects a ClassA which is a ClassA and ClassB.

Point to remember: We can only and only equate the dissimilar objects if they are derived from each other. We can equate an object of a base class to a derived class but not vice versa.

Another code snippet,

```
public class ClassB
    {
        public int b = 100;
    }

    public class ClassA:ClassB
    {
        public int a = 100;
    }

    /// <summary>
    /// Program: used to execute the method.
    /// Contains Main method.
    /// </summary>
    public class Program
    {
        private static void Main(string[] args)
        {
            ClassB classB = new ClassB();
            ClassA classA = new ClassA();
            classB=classA;
            classA = (ClassA)classB;
        }
    }
```

Although we violated a C# rule of equating objects, we did not get any compiler error because of the cast we did to the object . A **()** is called a cast. Within the brackets the name of the class is put. A cast basically proves to be a great leveller. When we intend to write classA = classB, C# expects the right hand side of the equal to to be a classA i.e. a ClassA instance . But it finds classB i.e. a ClassB instance. So when we apply the cast , we actually try to convert instance of ClassB to instance of ClassA. This approach satisfies the rules of C# on only equating similar objects type. Remember it is only for the duration of the line that classB becomes a ClassA and not a ClassB.

Now if we remove ClassB as a base class to class ClassA as in following code, and try to typecast classA to ClassB object,

```
 public class ClassB
  {
      public int b = 100;
  }

  public class ClassA // Removed ClassB as base class
  {
      public int a = 100;
  }

  /// <summary>
  /// Program: used to execute the method.
  /// Contains Main method.
  /// </summary>
  public class Program
  {
      private static void Main(string[] args)
      {
          ClassB classB = new ClassB();
          ClassA classA = new ClassA();
          classB = (ClassB)classA;
          classA = (ClassA)classB;
      }
  }
```

Output:
Error:
Cannot convert type 'InheritanceAndPolymorphism.ClassA' to 'InheritanceAndPolymorphism.ClassB'
Cannot convert type 'InheritanceAndPolymorphism.ClassB' to 'InheritanceAndPolymorphism.ClassA'

*'InheritanceAndPolymorphism' : Namespace I used in my application , so ignore that.

So we see that casting only works if one of the two classes is derived from one another. We can not cast any two objects to each other.

One last example,

```
  /// <summary>
  /// Program: used to execute the method.
  /// Contains Main method.
  /// </summary>
  public class Program
  {
      private static void Main(string[] args)
      {
          int integerA = 10;
          char characterB = 'A';
          integerA = characterB;
          characterB = integerA;
      }
```

```
    }
```

We run the code,

Output:

Error:Cannot implicitly convert type 'int' to 'char'. An explicit conversion exists (are you missing a cast?)

Point to remember: *We can not implicitly convert an int to char,but char can be converted to int.*

Conclusion:

In this part of our article series we learned about inheritance. We took various scenarios and practical examples back to back to understand the concept deeply. In my next article we'll be discussing about run time polymorphism. Inheritance plays very important role in run time polymorphism.
Lets list down our all the point to remember,

1. No one can stop a derived class to have a method with the same name already declared in its base class.
2. Derived classes get a first chance at execution, then the base class.
3. A reserved keyword named "base" can be used in derived class to call the base class method.
4. Inheritance does not work backwards.
5. Except constructors and destructors,a class inherits everything from its base class .
6. In inheritance in C#, custom classes can not derive from special built in c# classes like System.ValueType, System.Enum,System.Delegate,System.Array etc.
7. A class can only be derived from one class in C#.C# does not support multiple inheritance by means of class.
8. Circular dependency is not allowed in inheritance in C#.ClassX is derived from ClassW which was derived from ClassY and ClassY was again derived from ClassX, which caused circular dependency in three classes, that is logically impossible.
9. We can only and only equate the dissimilar objects if they are derived from each other. We can equate an object of a base class to a derived class but not vice versa.
10. We can not implicitly convert an int to char,but char can be converted to int.

You can read about compile time polymorphism in my first article of the series. Keep coding and learning

.

About Author

Akhil Mittal works as a Sr. Analyst in **Magic Software** and have an experience of more than 8 years in C#.Net. He is a codeproject and a c-sharpcorner MVP (Most Valuable Professional). Akhil is a B.Tech (Bachelor of Technology) in Computer Science and holds a diploma in Information Security and Application Development from CDAC. His work experience includes Development of Enterprise Applications using C#, .Net and Sql Server, Analysis as well as Research and Development. His expertise is in web application development. He is a MCP (Microsoft Certified Professional) in Web Applications (MCTS-70-528, MCTS-70-515) and .Net Framework 2.0 (MCTS-70-536).