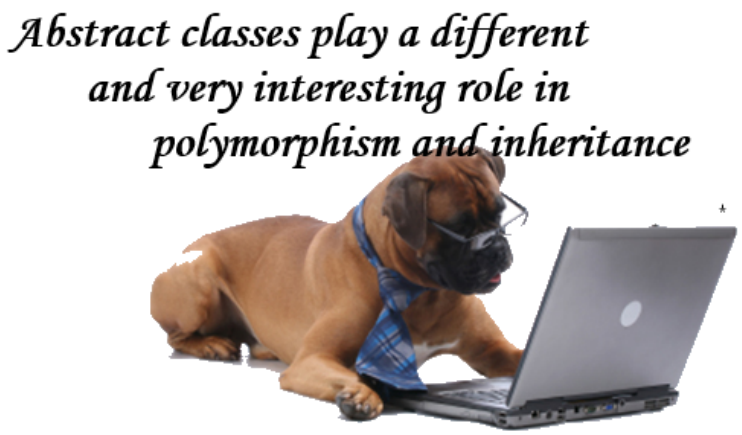
Diving in OOP (Day 4) : Polymorphism and Inheritance (All about Abstract classes in C#)

1. Introduction:

We learnt a lot about polymorphism and Inheritance. In this article of the series "Diving in OOP", we'll discuss about the most hot and exciting topic of OOP in C# i.e. Abstract Classes. The concept of Abstract classes is same for any other language, but in c# we deal with it in a bit different way. Abstract classes play a different and very interesting role in polymorphism and inheritance. We'll cover all the aspects of abstract classes with our hands-on lab and theory as an explanation to what output we get. We'll also list down points to remember at the end of the article.



Pre-requisites:

Wonder, we are dealing with the fourth part of our learning objective. Now my only expectation with my readers is to enjoy the series.

2. Roadmap:

Let's recall our road map ,

1. Diving in OOP (Day 1): Polymorphism and Inheritance (Early Binding/Compile Time Polymorphism).
2. Diving in OOP (Day 2): Polymorphism and Inheritance (Inheritance).
3. Diving in OOP (Day 3): Polymorphism and Inheritance (Dynamic Binding/Run Time polymorphism).
4. **Diving in OOP (Day 4) : Polymorphism and Inheritance (All about Abstarct classes in C#).**
5. Diving in OOP (Day 5) : Access Modifiers in C#.
6. Diving in OOP (Day 6) : All about Properties and Indexers.

3. Abstract Classes :

Let's get the definition from MSDN,

" The abstract keyword enables you to create classes and class members that are incomplete and must be implemented in a derived class. An abstract class cannot be instantiated. The purpose of an abstract class is to provide a common definition of a base class that multiple derived classes can share. For example, a class library may define an abstract class that is used as a parameter to many of its functions, and require programmers using that library to provide their own implementation of the class by creating a derived class. Abstract classes may also define abstract methods. This is accomplished by adding the keyword abstract before the return type of the method."

4. **Abstract classes in action:**

Add a console application named **"InheritanceAndPolymorphism"** in your visual studio.You'll get a class named Program.cs,just add one more class named ClassA.cs, note that the ClassA should be marked abstract, and the following code to ClassA.cs and Program.cs,

```
using System;

namespace InheritanceAndPolymorphism
{
    public abstract class ClassA
    {

    }

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

}
```

Compile the code,

Output:

Compile time error: Cannot create an instance of the abstract class or interface 'InheritanceAndPolymorphism.ClassA'

Point to remember: We cannot create an object of abstract class using new keyword.

Now we go into understanding the concept. No power can stop abstract keyword to be written before a class. It acts as a modifier to the class. We cannot create an object of abstract class using new keyword. Seems that the class is useless for us as we cannot use it for other practical purposes as we use to do.

5. Non abstract method definition in abstract class:

Let's add some code to our abstract class,

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }
}


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

We again see the error that we encountered earlier. Again it reminds that we cannot use new if we have already used an abstract modifier.

6. Abstract class acting as a base class:

Let's add one more class now,

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA
/// </summary>
public class ClassB:ClassA
{

}

/// <summary>
```

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

We get no error ☺. A class can be derived from abstract class. Creating an object of ClassB does not gives us any error.

Point to remember: A class can be derived from an abstract class.

Point to remember: A class derived from an abstract class can create an object.

7. Non abstract method declaration in abstract class :

Another scenario,

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }

    public void YYY();
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA.
/// </summary>
public class ClassB:ClassA
{

}

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

We just declared a method named YYY() in our abstract class ClassA.
Compile the code, we get,

Output:

Compile time error: 'InheritanceAndPolymorphism.ClassA.YYY()' must declare a body because it is not marked abstract, extern, or partial

InheritanceAndPolymorphism is the namespace I used for my console application so you can ignore that, no need to confuse with the logic.

In the above code, we just added a method declaration in the abstract class. An abstract method indicates that the actual definition or code of the method is created somewhere else. The method prototype declared in abstract class must also be declared abstract as per the rules of C#.

8. **Abstract method declaration in abstract class:**

Just make the method YYY() as abstract in ClassA

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }

   abstract public void YYY();
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA.
/// </summary>
public class ClassB:ClassA
{

}

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

Output:

Compiler error: 'InheritanceAndPolymorphism.ClassB' does not implement inherited abstract member 'InheritanceAndPolymorphism.ClassA.YYY()'

Point to remember: If we declare any method as abstract in our abstract class, then it's the responsibility of the derived class to provide the body of that abstract method, unless a body is provided for that abstract method, we cannot create an object of that derived class.

In above mentioned scenario, we declared method YYY() as abstract in ClassA. Since ClassB derives from ClassA, now it becomes the responsibility of ClassB to provide the body of that abstract method, else we cannot create an object of ClassB.

9. **Abstract method implementation in derived class:**

Now provide a body of method YYY() in ClassB, let's see what happens,

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }

   abstract public void YYY();
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA.
/// </summary>
public class ClassB:ClassA
{
    public void YYY()
    {

    }
}

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

Every thing seems fine now, but no ☹. Compile the code, what we get,

Output:

Two compile time errors this time,

Compile time error: 'InheritanceAndPolymorphism.ClassB' does not implement inherited abstract member 'InheritanceAndPolymorphism.ClassA.YYY()'

Compile time warning: 'InheritanceAndPolymorphism.ClassB.YYY()' hides inherited member 'InheritanceAndPolymorphism.ClassA.YYY()'. To make the current member override that implementation, add the override keyword. Otherwise add the new keyword.

We have been continuously trying to compile our code, but no success till now. The compiler error indicates clearly that both of our base and derived class contains the same method named YYY().
If both our derived class and base class contains the method with the same name, always an error occurs. The only way to overcome this error is derived class explicitly add the modifier override to its method signature. We have already discussed such scenarios in our previous parts of the articles of Diving in OOP series.

Let's add the override keyword before derived class method YYY().

```
/// <summary>
    /// Abstract class ClassA
    /// </summary>
    public abstract class ClassA
    {
        public int a;
        public void XXX()
        {

        }

       abstract public void YYY();
    }

    /// <summary>
    /// Derived class.
    /// Class derived from abstract class ClassA.
    /// </summary>
    public class ClassB:ClassA
    {
        public override void YYY()
        {

        }
```

```
}

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

We get no warning or error now ☺.

10. Abstract method implementation in derived class with different return type:

Let's just change the return type of the method YYY() in derived class,

```
/// <summary>
  /// Abstract class ClassA
  /// </summary>
  public abstract class ClassA
  {
      public int a;
      public void XXX()
      {

      }

     abstract public void YYY();
  }

  /// <summary>
  /// Derived class.
  /// Class derived from abstract class ClassA.
  /// </summary>
  public class ClassB:ClassA
  {
      public override int YYY()
      {

      }
  }
```

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

We changed return type of method YYY from void to int in derived class.Compile the code.

Output:

Compile time error: 'InheritanceAndPolymorphism.ClassB.YYY()': return type must be 'void' to match overridden member 'InheritanceAndPolymorphism.ClassA.YYY()'

Therefore one more constraint.

Point to remember: When we override an abstract method from a derived class, we cannot change the parameters passed to it or the return type irrespective of the number of methods declared as abstract in abstract class.

Let's see the implementation of second line mentioned in "point to remember",

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }

   abstract public void YYY();
   abstract public void YYY1();
   abstract public void YYY2();
   abstract public void YYY3();
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA.
/// </summary>
public class ClassB:ClassA
{
    public override int YYY()
    {

    }
}
```

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

Compiler error:

'InheritanceAndPolymorphism.ClassB' does not implement inherited abstract member 'InheritanceAndPolymorphism.ClassA.YYY3()'

'InheritanceAndPolymorphism.ClassB' does not implement inherited abstract member 'InheritanceAndPolymorphism.ClassA.YYY2()'

'InheritanceAndPolymorphism.ClassB' does not implement inherited abstract member 'InheritanceAndPolymorphism.ClassA.YYY1()'

If we implement these three methods in derived class, we'll get no error.

Point to remember : An abstract class means that the class is incomplete and cannot be directly used. An abstract class can only be used as a base class for other classes to derive from.

11. Variable initialization in abstract class:

Therefore as seen earlier, we get a error if we use a new keyword on an abstract class. If we do not Initialize a variable in an abstract class like we used a, it will automatically have a default value of 0 which is what the compiler kept warning us about. We can initialize int variable a of ClassA to any value we wish.The variables in abstract class act similar to that in any other normal class.

12. Power of abstract class:

Whenever a class remains incomplete i.e. we do not have the code for some methods, we mark those methods abstract and the class is marked abstract as well . And so we can compile our class without any error or blocker.

Any other class can then derive from our abstract class but they have to implement the abstract i.e. our incomplete methods from abstract class.
Abstract therefore enables us to write code for a part of the class and allows the others (derived classes) to complete the rest of the code.

13. Abstract method in non abstract class :

Let's take another code block,

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public class ClassA
{
    public int a;
    public void XXX()
    {

    }

   abstract public void YYY();
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA.
/// </summary>
public class ClassB:ClassA
{
    public override void YYY()
    {

    }
}

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

Compile the code,

Output:

Compiler error: 'InheritanceAndPolymorphism.ClassA.YYY()' is abstract but it is contained in non-abstract class 'InheritanceAndPolymorphism.ClassA'

We just removed abstract keyword from class ClassA.The error clearly conveys a message that if a single method is marked abstract in a class, then the class will have to be abstract as well.

Point to remember: If a class has even a single abstract method, then the class has to be declared abstract as well.

Point to remember: An abstract method also cannot use the modifiers such as static or virtual.

We can only have the abstract method in an abstract class. Any class that derives from abstract class has to give implementation to its abstract method. By default the modifier new gets added to th derived class method, that makes it a new/different method.

14. Abstract base method:

```
/// <summary>
/// Abstract class ClassA
/// </summary>
public abstract class ClassA
{
    public int a;
    public void XXX()
    {

    }

   abstract public void YYY();
}

/// <summary>
/// Derived class.
/// Class derived from abstract class ClassA.
/// </summary>
public class ClassB:ClassA
{
    public override void YYY()
    {
         base.YYY();
    }
}

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

Output:

Compile time error : Cannot call an abstract base member: 'InheritanceAndPolymorphism.ClassA.YYY()'

We cannot call the method YYY() from the base class ClassA as it does not carry any implementation/code along with it and has also been declared abstract. Common sense prevails ☺ and C# off course does not allow us to call a method that does not contain code.

15. Abstract class acting as derived as well as base class:

Let's modify our code a bit, and prepare our class structure something as follows,

```
/// <summary>
/// Base class ClassA
/// </summary>
public class ClassA
{
    public virtual void XXX()
    {
        Console.WriteLine("ClassA XXX");
    }
}

/// <summary>
/// Derived abstract class.
/// Class derived from base class ClassA.
/// </summary>
public abstract class ClassB:ClassA
{
    public new abstract void XXX();
}

public class ClassC:ClassB
{
    public override void XXX()
    {
        System.Console.WriteLine("ClassC XXX");
    }
}

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

Compile the code, and run.

Output:

ClassA XXX
ClassC XXX

We created a base class named ClassA that is not abstract and added a virtual method XXX to it.Since the method is non abstract but marked virtual so it has to be overridden in its deriving class.We added one more class named ClassB and marked that class abstract, note that this class is derived from ClassA. So this class has a choice to override the method marked as virtual in base class.But we'll do something different and tricky, We marked XXX method in this derived class as new abstract, and did not give any body to this method. Now what? We will add one more class ClassC, that will derive from ClassB. Class C has no choice but to override the method XXX. Therefore we override the method XXX in Class C.
In main method we created two objects `ClassA classA = new ClassC(); and ClassB classB = new ClassC();` First object looks like that of ClassC but refers to ClassA and second one again seems to be like ClassC but refers to ClassB.

In case of classA.XXX() will definitely first look into the class ClassA. Here it finds the method XXX marked as virtual. These kind of scenarios we have already taken n number of times in our earlier articles where we discussed about run time polymorphism . C# will then crawl over to class ClassB. Here it gets shocked that the method XXX() is abstract i.e. there is no code or implementation for method XXX() and also that it is a method marked as new, thus severing all links with the base class. And so flow halts and all and the method XXX() from ClassA gets executed.
In the case of b.XXX()(), since the method is new, the links to the base class gets broken, we are left with no choice but to invoke the method from ClassC as it says override.

We cannot replace the modifier new with the keyword override for the method XXX() in abstract class ClassB. Let's replace the override modifier in ClassC with "new" like,

```
public class ClassC:ClassB
    {
       public new void XXX()
       {
           System.Console.WriteLine("ClassC XXX");
       }
    }
```

Output:

Compile time error: 'InheritanceAndPolymorphism.ClassC' does not implement inherited abstract member 'InheritanceAndPolymorphism.ClassB.XXX()'

The error indicates that as there is no code for the method XXX. Remember the XXX() of class ClassA has nothing to do at all with that of ClassB and ClassC.
Also there is one more point to remember,

Point to remember: Virtual methods run slower that non virtual methods.

16. Can abstract class be sealed?

Let's take this final question into our consideration, let's test this too with an example,

```
/// <summary>
/// sealed abstract class ClassA
/// </summary>
public sealed abstract class ClassA
{
    public abstract void XXX()
    {
```

```
        Console.WriteLine("ClassA XXX");
    }
}

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

Compile the code.

Output:

Compile time error: 'InheritanceAndPolymorphism.ClassA': an abstract class cannot be sealed or static

And so we et two point to remember.

Point to remember : Abstract class cannot be sealed class.

Point to remember : Abstract class cannot be a static class.

17. Points to remember:

Let's sum up all the points to remember,

1. We cannot create an object of abstract class using new keyword.
2. A class can be derived from an abstract class.
3. Class derived from an abstract class can create an object.
4. If we declare any method as abstract in our abstract class, then it's the responsibility of the derived class to provide the body of that abstract method, unless a body is provided for that abstract method, we cannot create an object of that derived class.
5. When we override an abstract method from a derived class, we cannot change the parameters passed to it or the return type irrespective of the number of methods declared as abstract in abstract class.
6. An abstract class means that the class is incomplete and cannot be directly used. An abstract class can only be used as a base class for other classes to derive from.
7. If a class has even a single abstract method, then the class has to be declared abstract as well.
8. An abstract method also cannot use the modifiers such as static or virtual.

9. Virtual methods run slower that non virtual methods.
10. Abstract class cannot be sealed class.
11. Abstract class cannot be a static class.

18. Conclusion:

With this article we complete our understanding on Inheritance and polymorphism. We have covered almost all the aspects of Polymorphism and Inheritance. Abstract classes are one of my favorite's so I just wanted to take them separately.I hope my readers enjoyed my this article too and learnt about abstract classes in C#.
In my upcoming articles of the series we'll be discussing about other OOP features in the C# way with full hands-on lab and lot of discussion.

Keep coding and enjoy reading 

Also do not forget to rate/comment/like my article if it helped you by any means, this helps me to get motivated and encourages me to write more and more.

Read more:

- Learning MVC series.
- C# and Asp.Net Questions (All in one)
- MVC Interview Questions
- C# and Asp.Net Interview Questions and Answers
- Web Services and Windows Services Interview Questions

For more technical articles you can reach out to **A Practical Approach**.