Inheritance and Polymorpism

Asma Kanwal
Lecturer
Department of Computer Science
GC University, Lahore

CLASS INHERITANCE

Inheritance allows you to define a new class that in corporates and extends an already declared class.

- An existing class, called the base class, as the basis for a new class, called the derived class. The members of the derived class consist of the following:
  - The members in its own declaration
  - The members of the base class
- A derived class is said to extend its base class, because it includes the members of the base class plus any additional functionality provided in its own declaration.
- A derived class cannot delete any of the members it has inherited.

```
                    Class-base specification
                              ↓
class OtherClass : SomeClass
{                  ↑       ↑
   ...           Colon   Base class
}
```

Accessing the Inherited Members

```
class SomeClass
  {
    public string Field1 = "base class field ";
    public void Method1( string value ) {
      Console.WriteLine("Base class --
Method1:     {0}", value);
    }
  }

  class OtherClass: SomeClass
  {
    public string Field2 = "derived class field";
    public void Method2( string value ) {
      Console.WriteLine("Derived class --
Method2:  {0}", value);
    }
  }
```

```
class Program
  {
    static void Main() {
      OtherClass oc = new OtherClass();

      oc.Method1( oc.Field1 );      // Base
method with base field
      oc.Method1( oc.Field2 );      // Base
method with derived field
      oc.Method2( oc.Field1 );      // Derived
method with base field
      oc.Method2( oc.Field2 );      // Derived
method with derived field
    }
  }
```

Masking Members of a Base Class

- To mask an inherited data member, declare a new member of the same type and with the same name.
- To mask an inherited function member, declare a new function member with the same signature.
- To let the compiler know that purposely masking an inherited member, use the new modifier.
- You can also mask static members.

```
class SomeClass
{
    public string Field1;
    ...
}

class OtherClass : SomeClass
{
    new public string Field1;
    ↑
  Keyword
```

EXAMPLE

```
class SomeClass
  {
     public string Field1 = "SomeClass Field1";
     public void   Method1(string value)
        { Console.WriteLine("SomeClass.Method1:
{0}", value); }
  }

  class OtherClass : SomeClass
  {
new public string Field1 = "OtherClass Field1";  //
Mask the base member.
     new public void   Method1(string value)         //
Mask the base member.
        { Console.WriteLine("OtherClass.Method1:
{0}", value); }
  }
```

```
class Program
  {
     static void Main()
     {
        OtherClass oc = new OtherClass();
// Use the masking member.
        oc.Method1(oc.Field1);                    //
Use the masking member.
     }
  }
```

Base Access

```
class SomeClass {
    public string Field1 = "Field1 -- In the base class";
  }

  class OtherClass : SomeClass {

    new public string Field1 = "Field1 -- In the derived class";

public void PrintField1()
    {
      Console.WriteLine(Field1);
      Console.WriteLine(base.Field1);
}
  }
```

```
class Program {
    static void Main()
    {
      OtherClass oc = new OtherClass();
      oc.PrintField1();
    }
  }
```

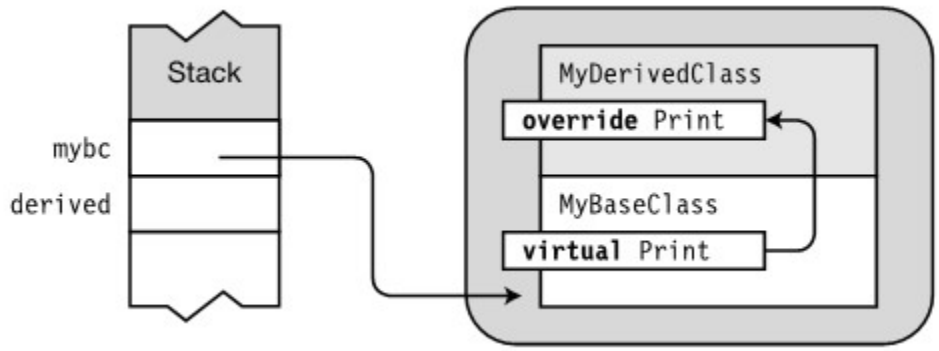
Using References to a Base Class

```
class MyBaseClass
  {
    public void Print()
    {
      Console.WriteLine("This is the
base class.");
    }
  }

  class MyDerivedClass : MyBaseClass
  {
    new public void Print()
    {
      Console.WriteLine("This is the
derived class.");
    }
  }
```

```
class Program
  {
    static void Main()
    {
      MyDerivedClass derived = new
MyDerivedClass();
      MyBaseClass mybc =
(MyBaseClass)derived;


          derived.Print();          // Call
Print from derived portion.
          mybc.Print();             // Call
Print from base portion.
    }
  }
```

Virtual and Override Methods

- The method in the derived class and the method in the base class each have the same signature and return type.
- The method in the base class is labeled virtual.
- The method in the derived class is labeled override



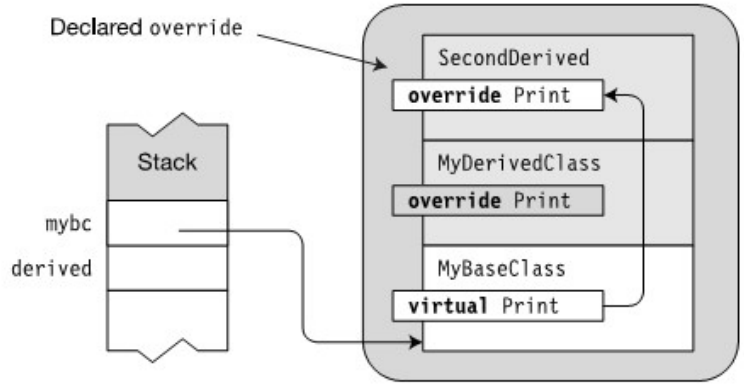
EXAMPLE

```
class MyBaseClass
  {
    virtual public void Print()
    {
      Console.WriteLine("This is the base class.");
    }
  }

  class MyDerivedClass : MyBaseClass
  {
    override public void Print()
    {
      Console.WriteLine("This is the derived class.");
    }
  }
```

```
class Program
  {
    static void Main()
    {
      MyDerivedClass derived = new MyDerivedClass();
      MyBaseClass mybc = (MyBaseClass)derived;

      derived.Print();
      mybc.Print();
    }
  }
```

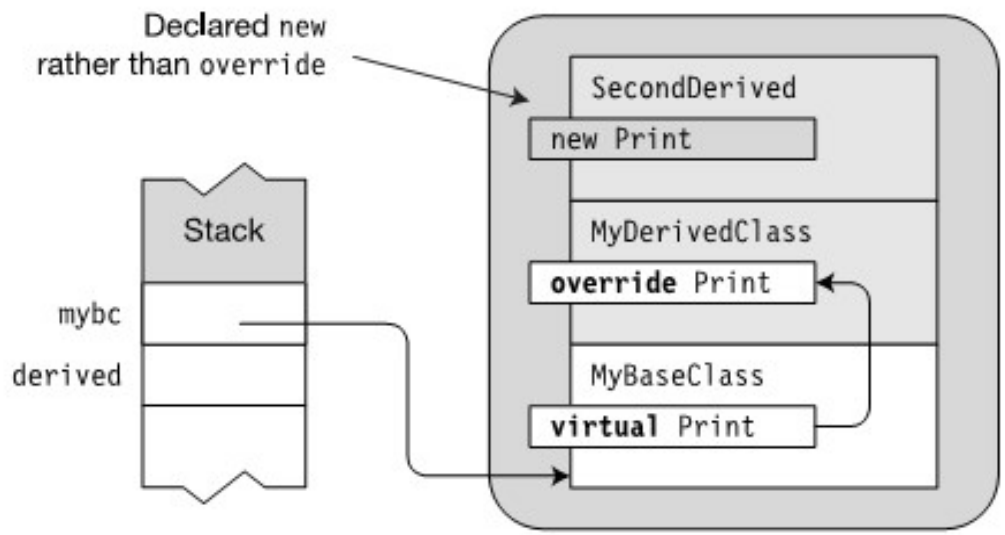
Overriding a Method Marked override

```
class MyBaseClass                                   // Base class
  {
    virtual public void Print()
    { Console.WriteLine("This is the base class."); }
  }

  class MyDerivedClass : MyBaseClass                //
Derived class
  {
    override public void Print()
    { Console.WriteLine("This is the derived class."); }
  }

  class SecondDerived : MyDerivedClass              //
Most-derived class
  {
override public void Print() {
        Console.WriteLine("This is the second derived
class.");
}
```




```
class Program
  {
    static void Main()
    {
      SecondDerived derived = new SecondDerived(); //
Use SecondDerived.
      MyBaseClass mybc = (MyBaseClass)derived;     //
Use MyBaseClass.

      derived.Print();
      mybc.Print();
    }
  }
```

Declaring Print with new



Abstract Members

An abstract member is a function member that is designed to be overridden. An abstract member has the following characteristics:

- It must be a function member. That is, fields and constants cannot be abstract members.
- It must be marked with the abstract modifier.
- It must not have an implementation code block. The code of an abstract member is represented by a semicolon.

```
abstract public void PrintStuff(string s);
```

Abstract Classes

Abstract classes are designed to be inherited from. An abstract class can only be used as the base class of another class.

- Cannot create instances of an abstract class.
- An abstract class is declared using the abstract modifier.

EXAMPLE

```
abstract class AbClass     // Abstract class
  {
     public void IdentifyBase()
// Normal method
     { Console.WriteLine("I am AbClass"); }
abstract public void IdentifyDerived();       //
Abstract method
  }


  class DerivedClass : AbClass
// Derived class
  {
override public void IdentifyDerived()
// Implementation of
     { Console.WriteLine("I am DerivedClass");
}          // abstract method
  }
```

```
class Program
  {
    static void Main()
    {
      // AbClass a = new AbClass();        //
Error.  Cannot instantiate
      // a.IdentifyDerived();  // an abstract
class.


      DerivedClass b = new DerivedClass(); //
Instantiate the derived class.
      b.IdentifyBase();   // Call the inherited
method.
      b.IdentifyDerived();  // Call the "abstract"
method.
    }
  }
```

Sealed Classes

- A sealed class can be instantiated only as a stand-alone class object-it cannot be used as a base class.
- A sealed class is labeled with the sealed modifier.

```
sealed class MyClass
  {

}
```

Static Classes

- The class itself must be marked static.
- All the members of the class must be static.
- The class can have a static constructor, but it cannot have an instance constructor, since you cannot create an instance of the class.
- Static classes are implicitly sealed. That is, you cannot inherit from a static class.

Static Classes

```
static public class MyMath
  {
    public static float PI = 3.14f;
    public static bool IsOdd(int x)
            { return x % 2 == 1; }

    public static int Times2(int x)
                { return 2 * x; }
  }
```

```
class Program
  {
    static void Main( )
    {
        int val = 3;
      Console.WriteLine("{0} is
odd is {1}.", val,
MyMath.IsOdd(val));
      Console.WriteLine("{0} * 2 =
{1}.",    val,  MyMath.Times2(val));
    }
  }
```