# 继承及多态性

# 继承

继承是面向对象程序设计中一个重要的特性，通过继承可以实现派生类和基类资源共享。

实现继承表示一个类型派生于一个基类型，拥有基类型所有的字段成员和函数方法成员。

实现继承时派生类型的每个方法都会采用基类型的代码实现，除非在派生类型中指定代码重写基类型的代码。

派生类型可以在基类型的基础之上添加功能。

# 基类与派生类的构造函数的关系

*1.基类中有无参构造函数，派生类没有无参构造函数：派生类将隐式地继承此无参构造函数， new 时执行基类构造函数。*

*2.基类中有无参构造函数，派生类也有自己的无参构造函数：将先执行基类的，后执行派生类的。*

*3.基类中没有无参构造函数，派生类有无参构造函数：执行派生类的。*

*4.基类中没有无参构造函数，派生类没有无参构造函数：*

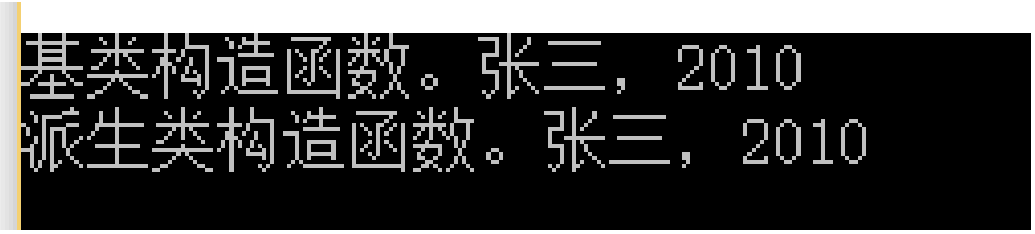
*5.如果基类中定义了无参构造函数，在派生类中可以自定义有参构造函数。（执行顺序：基类无参构造函数，派生类有参构造函数）*

*6.如果基类定义了有参构造函数，则此构造函数必须被显式继承：在派生类中实现构造函数时，提供一个将参数传递给基类构造函数的途径，以保证在基类初始化时能获得必需的数据，这时使用 base 关键字（执行顺序：基类构造函数，派生类构造函数）。例：*

```
Class  Personclass
   { public  Personclass(string T, string M)
        {  Console.WriteLine(" 基类构造函数。 {0} ， {1}", T,M);   }
   }

class Studentclass : Personclass
   {
      public Studentclass(string Name, string Num) : base (Name,Num)
      {  Console.WriteLine(" 派生类构造函数。 {0} ， {1}", Name,Num); }
   }

class Program
   {
      static void Main(string[] args)
      {   Studentclass s = new Studentclass(" 张三 ", "2010");
          Console.Read();
   } }
```



# 抽象类和抽象方法

C# 中可以使用 abstract 关键字把类和方法定义为抽象类和抽象方法。

**抽象类**具有如下特点：

- 抽象类只能做父类
- 抽象类不能实例化，即不能用 new 关键字来产生属于抽象类的对象
- 抽象类是普通类的特例。抽象类除了可以象普通类一样具有字段、属性、普通方法外，还可以（不是必须）包含抽象方法。

抽象类声明语法格式如下所示：

```
abstract  class  类名 { 类成员 }
```

**抽象方法**具有如下特点：

- 抽象方法没有方法内容，只有一个方法名和参数列表。
- 抽象方法不能执行代码，必须在非抽象的派生类中重写抽象方法。
- 在方法的返回类型前加 abstract 。
- 抽象方法的内容由它的继承类根据自身情况重写。重写时把 abstract 替换成 override 。

# 继承抽象类

抽象类不能进行实例化。只有在派生类继承抽象类并重写所有基类中没有实现功能的方法之后，才能实例化非抽象的派生类。

如何实现重写抽象方法：在派生类的相应方法前加 override 修饰符。

```
public  abstract   class  BaseClass          //  抽象类
  {
     public  abstract  void  Multiplication();
  }

public class Derived1:BaseClass // 继承于 BaseClass 的 Derived1 类
 {
     public  override  void  Multiplication()    // 实现基类的 Multiplication() 方法
     {
        for (int i = 1; i <= 9; i++)
        {
           for (int j = 1; j <= i; j++)
                Console.Write("{0}*{1}={2} ", i, j, i * j);
           Console.Write("\n");
        }
   }  }
```

# 虚方法和虚属性

C# 中可以在派生类中重写基类的虚方法，在调用方法时会选择调用对象类型的合适方法。 C# 中的虚方法需使用 virtual 显式地声明。

虚方法与抽象方法的区别在于：虚方法有自己的执行代码和方法签名，抽象方法只有方法签名没有方法体。

**例：** virtual 显式声明 print() 虚方法，这样任何派生类方法都可重写 print()

```
class Class3
{
    public  virtual  void  print()
     {     Console.Write(" 显示声明虚拟方法 ");     }
}
```

虚属性： 和声明虚拟方法相似，只需在属性前加 virtual 关键字：

```
class Class3
{
  public  virtual  string  getName { get; set; }
  public  virtual  string  getPwd{get;set;}
}
```

# 隐藏方法

当派生类和基类拥有形式相同的方法时（方法名、参数列表以及返回类型相同），派生类将隐藏基类方法，即在派生类中可以创建与基类完全相同的方法但是执行的过程却不同。这需要在派生类中使用 new 关键字显式地声明（不加 new 会报警，但不报错）。

**例**：派生类中定义 F() 方法时使用 new 关键字隐藏了基类中的 F() 方法。

```
public  class  HideBase          // HideBase 基类
{
    public void F()
    {    Console.Write(" 需要被隐藏的基类方法 ");    }
}

public  class  Hide : HideBase          // Hide 派生类
{
   new public void  F()                 // 隐藏基类 F() 方法
    {    Console.Write(" 派生类隐藏基类的方法 ");      }
}
```

# 调用基类

派生类并不能继承基类的所有成员，例如构造方法、析构方法和静态方法以及那些在基类中定义拒绝访问的成员。当需要调用基类的这些成员时，可以通过 base 关键字访问。

**例**：派生类中使用 base.<methodName>() 调用基类的方法。例如在派生类的方法中返回基类方法返回值的 80% ：

```
public class MyBase    // MyBase 基类
{
   public MyBase() { }
   public virtual int getNum()
     {        return 100;       }
}

public class MyDerived : MyBase    // MyDerived 派生类
{
  public override int getNum()
  {
     return Convert.ToInt32(base.getNum()*0.8); //base 调用基类方法
 } }
```

# 实现类的多态

多态是面向对象的特征之一。通过继承实现的不同对象调用相同的方法，表现出不同的行为，称之为多态。面向对象通过继承实现多态。

**例**：先创建一个基类：

```
public class TestBook
{    public virtual void Read()
     {    Console.Write(" 图书 ");       }
}
```

然后分别创建不同的类继承 TestBook 类的方法，实现多态的效果：

```
public class CBook:TestBook
{
   public override void Read()
   {    Console.Write(" 儿童图书 ");      }
}
public  class EBook:TestBook
{
    public override void Read()
    {     Console.Write(" 教育图书 ");     }
}
```