

继承和多态

继承的概念是什么呢？就是一个类可以继承另一个类的属性和方法（成员）

继承是面向对象编程中的一个非常重要的特性。

1.首先我们定义一个子类，给它创建两个构造：

一个无参构造和一个有参构造

定义一个枚举类

在定义的子类中它有自己的属性：

```
//定义一个枚举类
public enum Gender
{
    Male, Female
}

//子类
public class Chinese:Person
{
    public Gender Sex { get; set; }

    //无参构造
    public Chinese()
    {

    }

    //有参构造
    public Chinese(string name, int age, Gender sex)
    {
        this.Name = name;
        this.Age = age;
        this.Sex = sex;
    }
}
```

在定义的父亲类中的属性和构造：

```
//父类
public class Person
{
    public string Name { get; set; }

    public int Age { get; set; }
    //无参构造
    public Person()
    {

    }
    //有参构造
    public Person(string name, int age)
    {
        this.Name = name;
        this.Age = age;
    }
}
```

在 Main 方法中：

```
class Program
{
    static void Main(string[] args)
    {
        Chinese chinese = new Chinese("张三", 18, Gender.Male);
        Console.WriteLine("{0} \t {1}", chinese.Name, chinese.Sex);
        Console.ReadLine();
    }
}
```

同时要注意，在调用子类的带参构造时，我们要想到调用子类构造的时候，没有使用 base 关键字调用父类的指定的构造，默认调用父类的无参构造。

补充一下：

01.base 代表父类对象，如果 base()：调用父类的构造函数

02.base 代表的是方法调用，是不需要参数类型的

03.base(传参顺序和父类构造一致，变量名和子类构造参数一致)

2.在这里补充一点访问修饰符

我们所知道的：public private protected

下面我画一个图来简单的描述一下（√表示可以，×表示不可以）

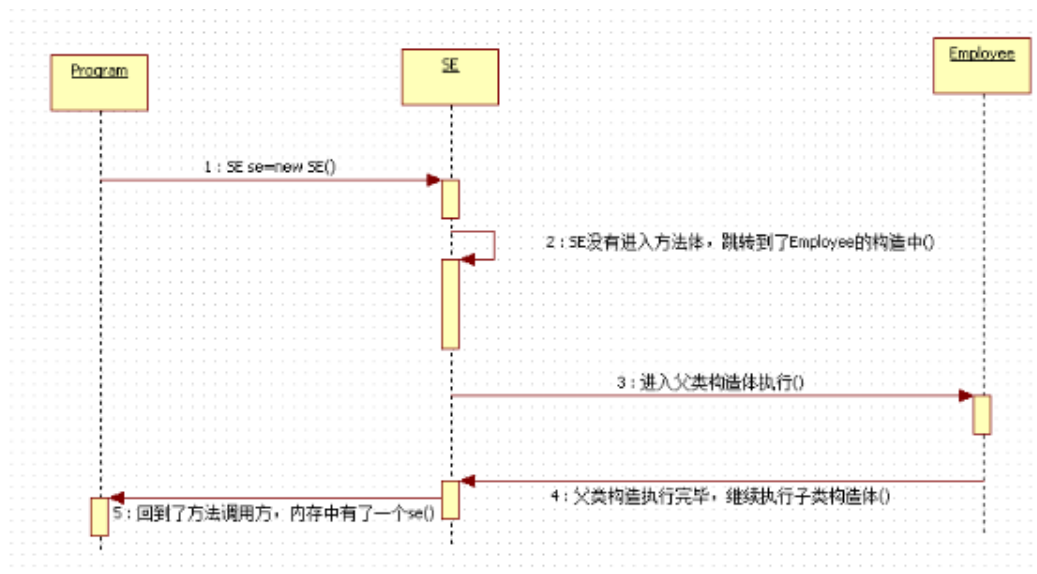
当前类	子类	其他类(Program)	
private	√	×	×
protected	√	√	×
public	√	√	√

总结：我们可以清晰的明白三种访问修饰符对类成员的访问限制强度：

private>protected>public

3.new 子类底层原理图

我简单的用一个图来描述一下：



用文字描述：

- 1) .走到子类构造，不进入构造体
- 2) .转向父类，进入父类构造体执行
- 3) .转回子类构造，执行子类构造体
- 4) .转到 Main，内存中构建出子类对象

4.继承还具有两大特性这个我们也不要忘记了，就是单根性和传递性

单根性指的就是一个子类只有一个父类

传递性就是只要跟父类有继承关系，就可以使用父类的属性和方法

接下来我们讲一讲多态

1.多态是什么呢？字面上的意思就是多种形态

用专业一点的话来说就是指同一个操作作用于不同的对象时，可以有不同的解释，产生不同的执行效果。

我们所接触的方法重载也是多态的一种方式。

如何实现多态呢？不要急下面我来解说

(1) 实现方法的重写

在父类中定义的方法，用 `virtual` 关键字来定义为虚方法

在子类中定义自己的方法，用 `override` 关键字来修饰，实现对父类的方法的重写

(2) 定义父类变量，用子类变量初始化父类变量

是不是觉得抽象，其实我刚开始学习时也是一样的，下面来一个小案例：

//创建一个 Person 父类

```
public class Person
{
    public virtual void SayHello()
    {
        //父类特有的方法

        Console.WriteLine("父类打招呼方法");
    }
}
```

//创建一个 Korea 子类

```
public class Korea:Person //在这里要注意它继承于 Person 类
{
    public override void SayHello()
    {
        Console.WriteLine("金喜善打招呼方法");
    }
}
```

//创建一个 Chinese 类

```
public class Chinese:Person //在这里要注意它继承于 Person 类
```

```

    {
    public override void SayHello()
    {
        Console.WriteLine("你好！");
    }
}

//创建一个 American 类

public class American:Person //在这里要注意它继承于 Person 类
{
    public override void SayHello()
    {
        Console.WriteLine("Hello");
    }
}

```

前面我们也学过泛型了下面我们就用泛型来存储

```

        List<Person> list=new List<Person>();
        Chinese chinese=new Chinese();
        American usa=new American();
        Korea korea=new Korea();
        list.Add(chinese);
        list.Add(usa);
        list.Add(korea);

```

下面我们可以用 foreach 来遍历

方式一：

```

        foreach (Person person in list)
        {
            person.SayHello();
        }

```

方式二：

```

        foreach (Person person in list)
        {
            //方式二：不使用统一调用

            if (person is Chinese)
            {
                Chinese chinese= person as Chinese;
                chinese.SayHello();
            }

            if (person is Korea)
            {
                Korea chinese= person as Korea;
                korea.SayHello();
            }

            if (person is American)
            {
                American chinese= person as American;
                american.SayHello();
            }
        }
    }
}

```

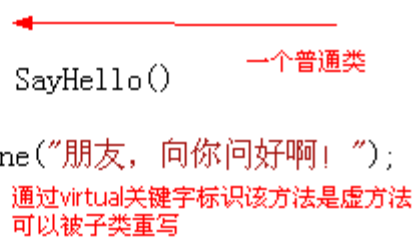
下面我再穿插个示例：

```

public class Person
{
    public virtual void SayHello()
    {
        Console.WriteLine("朋友，向你问好啊！");
    }
}

public class Teacher:Person
{
    public override void SayHello()
    {
        Console.WriteLine("我是老师打招呼方法");
    }
}

```



```
public class Student:Person
{
    public override void SayHello()
    {
        Console.WriteLine("我是学生打招呼方法");
    }
}
```

student继承自
Person

在子类中通过override 关键字对父
类的虚方法进行重写

在 Main 方法中调用