

## 抽象类

抽象类体现的就是一种模板模式的设计，抽象类作为多个子类的通用模板，子类在抽象类的基础上进行扩展、改造，但子类总体上会保留抽象类的行为方式。

## 接口

接口就是给出一些没有实现的方法,封装到一起,到某个类要使用的时候,在根据具体情况把这些方法写出来。语法:

```
interface 接口名{
    //属性
    //抽象方法
}
class 类名 implements 接口{
    自己属性;
    自己方法;
    必须实现的接口的抽象方法
}
```

**小结:**接口是更加抽象的抽象的类，抽象类里的方法可以有方法体，接口里的所有方法都没有方法体 [jdk7.0]。接口体现了程序设计的多态和高内聚低耦合的设计思想。

特别说明: Jdk8.0后接口类可以有静态方法，默认方法，也就是说接口中可以有方法的具体实现

## 注意细节

1) 接口不能被实例化

2) 接口中所有的方法是 public方法，接口中抽象方法，可以不用abstract 修饰 图示:

```
void aaa();
实际上是 abstract void aa();
```

```
void aaa(){ };
```

3) 一个普通类实现接口,就必须将该接口的所有方法都实现。

4) 抽象类实现接口，可以不用实现接口的方法。

5) 一个类同时可以实现多个接口 [举例]

6) 接口中的属性,只能是final的，而且是 public static final 修饰符。比如: int a=1; 实际上是 public static final int a=1; (必须初始化)

7) 接口中属性的访问形式: 接口名.属性名

8) 接口不能继承其它的类,但是可以继承多个别的接口 [举例]

```
interface A extends B,C{}
```

9) 接口的修饰符 只能是 public 和默认，这点和类的修饰符是一样的。

## 接口和继承的区别

### ➤ 接口和继承解决的问题不同

继承的价值主要在于：解决代码的**复用性和可维护性**。

接口的价值主要在于：设计，设计好各种规范(方法)，让其它类去实现这些方法。即更加的灵活..

### ➤ 接口比继承更加灵活

接口比继承更加灵活，继承是满足 is - a 的关系，而接口只需满足 like - a 的关系。

### ➤ 接口在一定程度上实现代码解耦 [即: 接口规范性+动态绑定机制]

## 接口的多态特性

### 1) 多态参数(前面案例体现) InterfacePolyParameter.java

在前面的Usb接口案例，UsbInterface usb，既可以接收手机对象，又可以接收相机对象，就体现了接口多态(接口引用可以指向实现了接口的类的对象)

### 2) 多态数组 InterfacePolyArr.java

演示一个案例：给Usb数组中，存放 Phone 和 相机对象，Phone类还有一个特有的方法call()，请遍历Usb数组，如果是Phone对象，除了调用Usb接口定义的方法外，还需要调用Phone特有方法 call。

### 3) 接口存在多态传递现象. InterfacePolyPass.java

```
interface AInterface1 {  
    void f1();  
}  
class B implements AInterface1 {  
    @Override  
    public void f1() {  
        System.out.println("f1~~~");  
    }  
}  
class C extends B { }
```

```
C c = new C();  
AInterface1 af = c;
```

```
Usb usbs[] = new Usb[2];  
usbs[0] = new Phone();  
usbs[1] = new Camera();  
for (int i = 0; i < usbs.length; i++) {  
    usbs[i].start();  
    usbs[i].stop();  
    if (usbs[i] instanceof Phone) {  
        ((Phone)usbs[i]).call();  
    }  
}
```

## 内部类

如果定义类在局部位置(方法中/代码块):(1) 局部内部类 (2) **匿名内部类**

定义在成员位置 (1) 成员内部类 (2) 静态内部类

一个类的内部又完整的嵌套了另一个类结构。被嵌套的类称为内部类(inner class)，嵌套其他类的类称为外部类(outer class)。是我们类的第五大成员【思考：类的五大成员是哪些?[属性、方法、构造器、代码块、内部类]】，内部类最大的特点就是可以直接访问私有属性，并且可以体现类与类之间的包含关系，注意:内部类是学习的难点,同时也是重点,后面看底层源码时,有大量的内部类。

## 语法

```
class Outer{ //外部类  
    class Inner{ //内部类  
    }  
}  
class Other{ //外部其他类  
}  
//InnerClass01.java
```



## 内部类分类

➤ 定义在外部类局部位置上（比如方法内）：

- 1) 局部内部类（有类名）
- 2) 匿名内部类（没有类名，重点!!!!!!!）

➤ 定义在外部类的成员位置上：

- 1) 成员内部类（没用static修饰）
- 2) 静态内部类（使用static修饰）

## 局部内部类的使用

说明：局部内部类是定义在外部类的局部位置，比如方法中，并且有类名。

1. 可以直接访问外部类的所有成员，包含私有的
2. 不能添加访问修饰符，因为它的地位就是一个局部变量。局部变量是不能使用修饰符的。但是可以使用final修饰，因为局部变量也可以使用final
3. 作用域：仅仅在定义它的方法或代码块中。

4. 局部内部类---访问---->外部类的成员 [访问方式：直接访问]

5. 外部类---访问---->局部内部类的成员

访问方式：创建对象，再访问(注意：必须在作用域内)

记住:(1)局部内部类定义在方法中/代码块

(2) 作用域在方法体或者代码块中

(3) 本质仍然是一个类

6. 外部其他类---不能访问----->局部内部类（因为局部内部类地位是一个局部变量）

7. 如果外部类和局部内部类的成员重名时，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.this.成员）去访问【演示】

```
System.out.println("外部类的n2=" + 外部类名.this.n2);
```

## 匿名内部类的使用（重要）

//(1) 本质是类(2) 内部类(3) 该类没有名字

(4)同时还是一个对象

说明：匿名内部类是定义在外部类的局部位置，比如方法中，并且没有类名

1. 匿名内部类的基本语法

```
new 类或接口(参数列表){  
    类体  
};
```

2. 匿名内部类的语法比较奇特，请大家注意，因为匿名内部类既是一个类的定义，同时它本身也是一个对象，因此从语法上看，它既有定义类的特征，也有创建对象的特征，对前面代码分析可以看出这个特点，因此可以调用匿名内部类方法。

3. 可以直接访问外部类的所有成员，包含私有的 [案例演示]

4. 不能添加访问修饰符，因为它的地位就是一个局部变量。 [过]

5. 作用域：仅仅在定义它的方法或代码块中。 [过]

6. 匿名内部类---访问---->外部类成员 [访问方式：直接访问]

7. 外部其他类---不能访问----->匿名内部类（因为匿名内部类地位是一个局部变量）

8. 如果外部类和匿名内部类的成员重名时，匿名内部类访问的话，默认遵循就近原则  
如果想访问外部类的成员，则可以使用（外部类名.this.成员）去访问