# 内部类

一般情况下， 类与类之间是相互独立的，内部类的意思是打破这样的独立，让一个类成为另一个类的内部信息，和成员变量，成员方法同等级别。

> 为什么要使用内部类?

采用内部类这个技术，可以隐藏细节和内部结构，封装性更好，让程序的结构更加合理

## 非静态内部分类

```
package test;

public class OuterClass {
    //成员变量
    private String outerName;
    //成员方法
    private void dsiplay(){
        System.out.println("OuterClass display");
        System.out.println(outerName);
    }
    //内部类
    public class InnerClass{
        private String innerName;

        public InnerClass() {
            this.innerName = "inner Class";
        }

        public void display(){
            System.out.println("InnerClass display");
            System.out.println(innerName);
        }
    }

    public static void main(String[] args) {
        OuterClass outerClass = new OuterClass();
        outerClass.dsiplay();
        OuterClass.InnerClass innerClass = outerClass.new InnerClass();//使用的内部类方法
        innerClass.display();
    }
}
```

```
OuterClass display
null
InnerClass display
inner Class
```

非静态内部类的使用就是将内部类当做外部类的一个成员变量/成员方法来使用，所以必须依赖于外部类的对象才能调用，用法和成员变量/成员方法是一模一样的。

基本的内部类还可以在一个方法中定义。

```
package test;

public class OuterClass {
    //成员变量
    private String outerName;
    //成员方法
    private void dsiplay(){
         class InnerClass{//内部类
            public void display(){
                System.out.println("InnerClass display");
            }
        }
        InnerClass innerClass = new InnerClass();
         innerClass.display();
    }
    public static void main(String[] args) {
        OuterClass outerClass = new OuterClass();
        outerClass.dsiplay();
    }
}
```

# 静态内部类

静态内部类的构造不需要依赖于外部类的对象，类中的所以静态组件都不需要依赖于任何对象，可以直接通过类本身进行构造。

```
package test;

public class OuterClass {
    //成员变量
    private String outerName;
    //成员方法
    private void dsiplay(){
        System.out.println("outerclass display");
    }
    //静态内部类
    public static class InnerClass{
        private String innerName;
        public InnerClass(){
            innerName="inner class";
        }
        public void display(){
            System.out.println("Innerclass display");
            System.out.println(innerName);
        }
    }

    public static void main(String[] args) {
        OuterClass outerClass = new OuterClass();
        outerClass.dsiplay();
        InnerClass innerClass = new InnerClass();
```

```
        innerClass.display();
    }
}
```

```
outerclass display
Innerclass display
inner class
```

# 匿名内部类

定义接口

```
package test;

public interface MyInterface {
    public void test();
}
```

使用

```
package test;

public class Test {
    public static void main(String[] args) {
        MyInterface myInterface = new MyInterface() {
            @Override
            public void test() {
                System.out.println("test");
            }
        };
        myInterface.test();
    }
}
```

使用内部类也可以实现接口

```
package test;

public class Test {
    public class MyImplement implements MyInterface{
        @Override
        public void test() {
            System.out.println("test");
        }
    }
    public static void main(String[] args) {
        Test test = new Test();
        MyImplement myImplement = test.new MyImplement();
        myImplement.test();
    }
}
```