内部类（重难点）

一、什么是内部类

- **一个类的内部又完整的嵌套了另外一个类.（被嵌套的类叫内部类inner class）**
- **语法:**

```
    class Outer{//外部类
        class inner{}内部类
    }
    class Other{//外部其他类

    }
```

- **类的五大成员:**

1. **属性.**
2. **方法.**
3. **构造器.**
4. **代码块.**
5. **内部类.**

二、内部类总共分四种

- **定义在外部类的局部位置上:**

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static).**
2. **静态内部类(有static).**

```
package com.InnerClass;
/*
* 演示局部内部类的使用*/
public class LocalInnerClass {
    public static void main(String[] args) {
        //演示一遍
        Outer02 outer02 = new Outer02();
        outer02.m1();
    }
}
class Outer02{
    private int n1 = 100;
```

```
    private void m2(){System.out.println("Outer02 m2()");}
    public void m1(){//方法
        //1.局部内部类是定义在外部类的局部位置，通常在方法中
        //3.内部类 (不能)添加 访问修饰符,但是可以用final修饰
        //4.作用域:仅在定义它的 (方法) 或 (代码块) 中
        class Inner02{//局部内部类，若加了final则局部内部类是不能被继承的
            //2.可以直接访问外部类的所有成员,包括私有的
            public void f1(){
                //5.局部内部类可以直接访问外部类的所有成员，如下面的n1 和 m2()
                System.out.println("n1= " + n1);
                m2();//可以访问外部类的所有成员嘛~
            }
        }
        //4.1:局部内部类可以相互继承,除非父类有final
        class Inner03 extends Inner02{}

        //6.外部类想用内部类，只能在方法中创建这个Inner02对象,然后调用方法即可
        Inner02 inner02 = new Inner02();
        inner02.f1();
    }
}
```

- **注意:外部其他类-----不能访问-----局部内部类 (因为局部内部类 地位 相当于一个 局部变量).**

三、匿名内部类（超重要）

- **使用场景**

1. **将匿名内部类当作实参直接传递,简洁高效.**

介绍:

1. **本质是类.**
2. **是内部类.**
3. **该类没有名字（系统取的名字）.**
4. **同时还是一个对象.**

1.基本语法

```
new 类或接口(参数列表){
  类体
};
```

2.匿名内部类 应用&机制

```
package com.InnerClass.AnonymousInnerClass;

public class AnonymousClass01 {
    public static void main(String[] args) {

        Outer04 outer04 = new Outer04();
```

```
        outer04.method();
    }
}

class Outer04 {//外部类
    private int n1 = 10;

    public void method() {
        /*基于接口的匿名内部类
         * 1.需求:使用接口IA,并创建对象
         * 2.传统方式,是一个类,实现该接口,并创建对象
         *
         * 3.若Tiger类只用一次,以后再也不用
         * * 可以使用匿名内部类简化开发*/
        //传统写法在方法内实例化对象,之后在主方法调用该类的方法
        /*Tiger tiger = new Tiger();
        tiger.cry();*/


        /*1.tiger的编译类型是: IA
        * 2.tiger的运行类型是: 匿名内部类(系统分配的名字是:XXXX --->Outer04$1)
        * 匿名内部类的底层:(用完一次就没了)
        *   class XXXX implements IA{
        *         @Override
                public void cry() {
                    System.out.println("匿名内部类在Method()中创建");
                }
        *   }
        * 3.jdk底层在创造匿名内部类 Outer04$1,
        * 马上就创建了Outer04$1实例,并且把地址返回给tiger
        * 4.匿名内部类只能使用一次,但new出来的对象是可以多次使用的*/
        System.out.println("-------------匿名内部类 之 接口的匿名--------------");
        //使用匿名内部类
        IA tiger = new IA(){
            @Override
            public void cry() {
                System.out.println("匿名内部类在Method()中创建");
            }
        };
        System.out.println("tiger的运行类型:"+tiger.getClass());
        //调用cry()
        tiger.cry();

        System.out.println("--------------匿名内部类 之 类的匿名--------------");
        /*new Father("jack"){} ------相当于(底层写法为)----> class Outer04$2
extends Father("jack"){}
        * 表现了创建对象的特征
        * 若是抽象类在实现匿名内部类时必须记得重写抽象方法*/
        Father father = new Father("jack"){//由于Father类构造器需要传承一个name
            //若想要重写一个test()也可以
            @Override
            public void test() {
                System.out.println("重写的Father类的test()");
            }
        };
        System.out.println("father的匿名内部类的名字:"+father.getClass());
        //调用test()
        father.test();
```

```
    }

    interface IA {//接口是不能被new的
        public void cry();//隐藏的public final static
    }

    //Tiger类实现IA接口
/*    class Tiger implements IA {
        @Override
        public void cry() {
            System.out.println("Tiger类实现了接口IA");
        }
    }*/

    class Father {
        public Father(String name) {
            System.out.println("构造器的名字"+name);
        }

        public void test() {
        }
    }
}
```

四,匿名内部类的细节

Detail01:

```
package com.InnerClass.Detail;
/*内部类:(外部类成员的调用细节)
* 1.可以直接访问外部类的所有成员
* 2.不能添加访问修饰符(public,protect,private..),
* 内部类的地位就是一个局部变量
* 3.内部类的作用域:只在定义它的 方法 或 代码块 中
* 4.匿名内部类---访问--->外部类成员[访问方式:直接访问]
* 5.外部其他类 不能访问 匿名内部类(因为其本质是局部变量)
* 6.若 外部类 和 匿名内部类的成员重名时,
*   若想访问匿名内部类 默认是就近原则,
*   若想访问外部成员,则可以使用 (外部类名.this.成员) 去访问*/

public class AnonymousInnerClassDetail01 {
    public static void main(String[] args) {
        OuterClassD1 outerClassD1 = new OuterClassD1(1);
        outerClassD1.method();//调用创建了匿名内部类的方法
    }
}
class OuterClassD1{
    private int n1;

    public OuterClassD1(int n1) {
        this.n1 = n1;
    }
    public void method(){//匿名内部类的创建之地
        new Person(){//类 的 匿名内部类还可以这样写
```

```
            private int n1;
            {//此代码块充当构造器的作用
                n1 = 2;
            }
            //重写一个print()
            @Override
            public void print(String name) {
                super.print(name);
                System.out.println("1.金山词霸");
                System.out.println("访问n1(就近原则):" + n1 + "\n" +
                        "访问外部类成员n1(写法:OuterClassD1.this.n1)" +
OuterClassD1.this.n1);
            }
        }.print("大靓仔");
    }

    public int getN1() {
        return n1;
    }

    public void setN1(int n1) {
        this.n1 = n1;
    }
}


class Person{
    public void print(String name){
        System.out.println("打印目录:");
        System.out.println("打印人:" + name);
    }
}
```

五、匿名内部类的使用实例

Excer01:

```
package com.InnerClass.InnerClassExcercise;
/*匿名内部类最佳使用场景
* 1.当作实参直接传递,简洁高效*/
public class Excer01 {
    public static void main(String[] args) {
        //若只使用一次这个类,可以使用匿名内部类的方法,直接把这个匿名内部类当作f1的实参
        f1(new IL(){
            @Override
            public void Draw() {
                System.out.println("照片中增添了一幅名画(以匿名内部类的方式呈现)");
            }
        });
        f1(new Pictrue());//硬编码,传的参数是IL的徒弟自然可以
    }
    public static void f1(IL il) {
        il.Draw();//需要调用师傅里面的方法,在主方法就不用new了
    }
}
```

```
//接口
interface IL{
    public void Draw();//默认是 public static void
}

//传统写法:实例化一个类--->再在主函数new (俗称硬编码)
class Pictrue implements IL {
    @Override
    public void Draw() {
        System.out.println("照片中增添了一幅名画");
    }
}
```

Excer02:

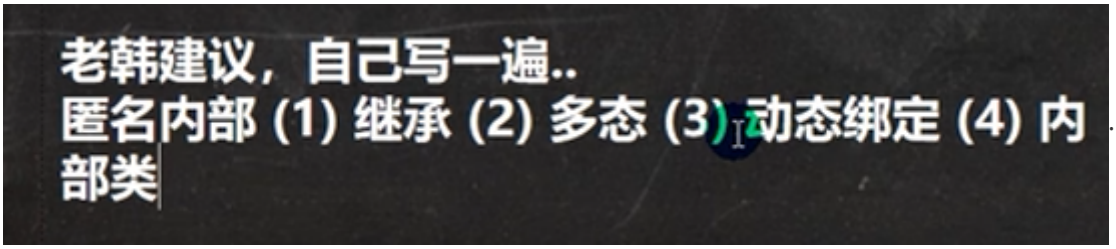
```
package com.InnerClass.InnerClassExcercise;
/*匿名内部类的练习
* 1.有一个铃声接口Bell,里面有一个ring方法
* 2.有一个手机类CellPhone,具有闹钟功能alarmClock,
* 参数就是Bell类型
* 3.测试手机类的闹钟功能,通过匿名内部类作为参数对象,打印:懒猪起床了
* 4.再传入另外一个匿名内部类(对象),打印:伙伴上课了
* */
public class Excer02 {
    public static void main(String[] args) {
        CellPhone cellPhone = new CellPhone();

        cellPhone.alarmclock(new Bell() {
            @Override
            public void ring() {
                System.out.println("懒猪起床了");
            }
        });

        cellPhone.alarmclock(new Bell() {
            @Override
            public void ring() {
                System.out.println("伙伴上课了");
            }
        });//括号里的内容以匿名内部类作为对象,并且书写好要实现的一次性内容

    }
}

//接口Bell
interface Bell{
    void ring();
}
class CellPhone{
    public void alarmclock(Bell bell){
        bell.ring();
    }
}
```



六、成员内部类

```
package com.InnerClass.MemberInnerClass;
/*成员内部类:
* 1.定义在外部类的位置上,并没有static
* 2.可以任意添加访问修饰符,因为他属于成员变量,
* 这与在方法内的内部类不同,因为这属于局部变量
*
* 作用域:与其它成员一样作用于整个类体
*
* 内部类想使用外部类成员:可以直接调用
* 外部类想使用内部类成员:写一个方法创建对象使用其方法
* */

public class MemberInnerClass01 {
    public static void main(String[] args) {
        Outer08 outer08 = new Outer08();
        outer08.print();

        /*外部其他类使用成员内部类的三种方式:*/
        //1.new Inner08相当于Outer08中的一个成员
        Outer08.Inner08 inner08 =  outer08.new Inner08();
        //2.在外部类中写一个返回Inner08的一个方法,再调用
        Outer08.Inner08 inner081 = outer08.getInner08();
    }
}
class Outer08{
    private int n1 = 10;//外部类成员n1
    public String name = "张三";

    public class Inner08{//成员内部类(特点:在外部类的成员位置上,并没有static修饰)
        private int n1 = 20;//内部类成员n1
        public void say(){
            //可以直接调用外部成员
            System.out.println("n1 = " + n1 + "\t名字:" + name);
            /*外部类成员 与 内部类成员 重名时的调用方式*/
            System.out.println("外部类成员n1 = " + Outer08.this.n1 +
                    "\t内部类成员n1 = " + n1);
        }
    }

    public Inner08 getInner08(){
        return new Inner08();
    }

    //创建成员内部类方法(使用)
    public void print(){
        Inner08 inner08 = new Inner08();
        inner08.say();
```

```
    }
}
```

七、静态内部类

```
package com.InnerClass.StaticInnerClass;
/*静态内部类:
* 1.依然是定义在外部类的成员位置上,只是多了static修饰
* 2.可以访问外部类的所以静态成员,不能直接访问非静态成员
* 3.可以任意添加访问修饰符
* 4.作用域:整个外部类
* 5.外部类访问静态内部类的方式就是创建一个方法new一个对象
* 6.重名问题也是一样处理(只不过内部类调用外部类重名的成员语法:外部类名.外部类重名成员的名字)*/


public class StaticInnerClass01 {
    public static void main(String[] args) {
        Outer10 outer10 = new Outer10();
        outer10.m1();
        /*外部其他类,使用静态内部类*/
        //1.
        Outer10.Inner10 inner10 = new Outer10.Inner10();
        inner10.say();
        //2.编写一个方法可以返回静态内部类的实例
        Outer10.Inner10 inner101 = outer10.getInner10();
        inner101.say();
        //3.编写一个静态方法返回静态内部类的实例
        //不用new Outer10
        Outer10.Inner10 inner10_ = Outer10.getInner10_();

    }
}
class Outer10{
    private int n1 = 10;
    private static String name = "张三";

    //加了个访问修饰符public
    public static class Inner10{
        //可以直接访问外部类所有静态成员
        public void say(){
            System.out.println("name: "+ name);
        }
    }
    //第五点例子
    public void m1(){
        Inner10 inner10 = new Inner10();
        inner10.say();
    }

    public Inner10 getInner10(){
        return new Inner10();
    }

    public static Inner10 getInner10_(){
        return new Inner10();
    }
}
```