类与接口（2）：Java 的四种内部类详解

引言

内部类，嵌套在另一个类的里面，所以也称为 嵌套类;

内部类分为以下四种：

- 静态内部类
- 成员内部类
- 局部内部类
- 匿名内部类

一、静态内部类

静态内部类： 一般也称”静态嵌套类 “，在类中用 static 声明的内部类。

因为是 static，所以不依赖于外围类对象实例而独立存在，**静态内部类的可以访问外围类中的所有静态成员，包括 private 的静态成员。**

同时静态内部类可以说是所有内部类中独立性最高的内部类，其创建对象、继承（实现接口）、扩展子类等使用方式与外围类并没有多大的区别。

```
public class OuterClass {//外围类
    public int aa; //实例成员
    private static float f = 1.5f;//private 的静态成员
    static void println() {
        System.out.println("这是静态方法");
    }
    protected static class StaticInnerClass{//protected 的静态内部类
        float a;
        public StaticInnerClass() {
             a = f;// 外围类的 private 静态变量
             println();//外围类的静态方法
        }
    }
}

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

```
        //创建静态内部类的对象
    OuterClass.StaticInnerClass staticInnerClass = new
OuterClass.StaticInnerClass();
   }
}
```

二、成员内部类

成员内部类： 定义在类的内部，而且与成员方法、成员变量同级，即也是外围类的成员之一，因此 成员内部类与外围类是紧密关联的。

注意：

这种紧密关联指的是，成员内部类的对象的创建必须依赖于外围类的对象（即没有外围类对象，就不可能创建成员内部类）。因此，成员内部类有以下 3 个特点：

- 成员内部类可以访问外围类的所有成员，包括私有成员；
- 成员内部类是不可以声明静态成员（包括静态变量、静态方法、静态成员类、嵌套接口），但有个例外---可以声明 static final 的变量， 这是因为编译器对 final 类型的特殊处理，是直接将值写入字节码；
- 成员内部类对象都隐式地保存了一个引用，指向创建它的外部类对象；或者说，成员内部类的入口是由外围类的对象保持着（静态内部类的入口，则直接由外围类保持着）。

成员内部类中的 this，new 关键字：

- 获取外部类对象：OuterClass.this
- 明确指定使用外部类的成员（当内部类与外部类的名字冲突时）：OuterClass.this.成员名
- 创建内部类对象的 new：外围类对象.new

```
//先创建外围类对象
OuterClass outer=new OuterClass();
//创建成员内部类对象
OuterClass.InnerClass inner=outer.new InnerClass();
```

1. 成员内部类的对象创建

我们知道，成员内部类就像外围类的实例成员一样，一定要存在对象才能访问，**即成员内部类必须绑定一个外围类的对象。**上面已经介绍了成员内部类的创建格式了，我们直接看一个例子。

```
public class OuterClass {//外围类
    public int aa; //实例成员
    private static float f = 1.5f;//private 的静态成员
    public void initInnerClass() {
        System.out.println("内部类的初始化方法");
```

```
    }
    public void createInnerClass() {//
        //外围类的成员方法中创建成员内部类对象
        InnerClass innerClass = new InnerClass();
    }
    class InnerClass{//成员内部类
        private double aa; //与外围类的变量 aa 的名字重复
        public InnerClass(){
            this.aa = OuterClass.this.aa + f;//明确指定两个 aa 的所属
            initInnerClass();
        }
    }
}

//其他类
class OtherClass{
    public static void main(String[] args) {
        //其他类中创建成员内部类
        OuterClass oc = new OuterClass();//外部类对象
        //创建内部类对象
        OuterClass.InnerClass innerClass = oc.new InnerClass();
    }
}
```

注意上面的例子中，在外围类的成员方法中创建成员内部类与在其他类中或静态方法中创建成员内部的方式是不一样的。

补充几点：

- 成员内部类可以继续包含成员内部类，而且不管一个内部类被嵌套了多少层，它都能透明地访问它的所有外部类所有成员；
- **成员内部可以继续嵌套多层的成员内部类，但无法嵌套静态内部类**；静态内部类则都可以继续嵌套这两种内部类。

下面的例子是基于上面的例子进行改造：

```
class InnerClass{//成员内部类
        private double aa; //与外围类的变量 aa 的名字重复
        public InnerClass(){
            this.aa = OuterClass.this.aa + f;//明确指定两个 aa 的所属
            initInnerClass();
        }
        public  class InnerInnerCalss2{//成员内部类中的成员内部类
            protected double aa = OuterClass.this.aa;//最外层的外围类的成员变量
```

```
        }//InnerInnerCalss2
    }//InnerClass
```

2. 继承成员内部类

在内部类的访问权限允许的情况下，成员内部类也是可以被继承的。由于成员内部类的对象依赖于外围类的对象，或者说，成员内部类的构造器入口由外围类的对象把持着。因此，**继承了成员内部类的子类必须要与一个外围类对象关联起来。同时，子类的构造器是必须要调用父类的构造器方法，所以也只能通过父类的外围类对象来调用父类构造器。**
下面的例子也是基于上面的例子的，只贴出多出的部分代码。

```
class ChildClass extends OuterClass.InnerClass{
    //成员内部类的子类的构造器的格式
    public ChildClass(OuterClass outerClass) {
        outerClass.super();//通过外围类的对象调用父类的构造方法
    }
}
```

三、局部内部类

局部内部类： 就是在方法、构造器、初始化块中声明的类，在结构上类似于一个局部变量。**因此局部内部类是不能使用访问修饰符。**
局部内部类的**两个访问限制**：

- 对于局部变量，局部内部类只能访问 final 的局部变量。不过，后期 JDK(忘了是 JDK 几了) 局部变量可不用 final 修饰，也可以被局部内部类访问，但你必须时刻记住此局部变量已经是 final 了，不能再改变。
- 对于类的全局成员，局部内部类定义在实例环境中 (构造器、对象成员方法、实例初始化块)，则可以访问外围类的所有成员；但如果内部类定义在静态环境中 (静态初始化块、静态方法)，则只能访问外围类的静态成员。

```
public class OuterClass {
    private int a = 21;
    static {//静态域中的局部内部类
       class LocalClass1{
             //  int z = a; //错误，在静态的作用域中无法访问对象成员
       }
    }
    {//实例初始化块中的局部内部类
        class localClass2{ }
    }
    public OuterClass(){
```

```
        int x = 2;
        final int y = 3;
        // x = 3;//若放开此行注释，编译无法通过，因为局部变量 x 已经是 final 类型
        //构造器中的局部内部类
        class localClass3{
            int z = y; //可以访问 final 的局部变量
            int b = a;//可以访问类的所有成员
             //访问没有用 final 修饰的局部变量
            int c = x;
        }
    }
    public void createRunnable() {
        final int x = 4;
        //方法中的局部内部类
        class LocalClass4 implements Runnable {//
            @Override
            public void run() {
                System.out.println("局部 final 变量："+x);
                System.out.println("对象成员变量："+a);
            }
        }
    }
}
```

四、匿名内部类

匿名内部类： 与局部内部类很相似，只不过匿名内部类是一个没有给定名字的内部类，在创建这个匿名内部类后，便会立即用来创建并返回此内部类的一个对象引用。

作用：匿名内部类用于隐式继承某个类（重写里面的方法或实现抽象方法）或者实现某个接口。

匿名内部类的访问限制： 与局部内部类一样，请参考局部内部类；

匿名内部类的优缺点：

优点： 编码方便快捷；

缺点：

- 只能继承一个类或实现一个接口，不能再继承其他类或其他接口。
- 只能用于创建一次对象实例；

下面的例子是我们创建线程时经常用到的匿名内部类的方式来快速地创建一个对象的例子：

```
class MyOuterClass {
    private int x = 5;
```

```
    void createThread() {
        final int a = 10;
        int b = 189;
        // 匿名内部类继承 Thread 类，并重写 Run 方法
        Thread thread = new Thread("thread-1") {
            int c = x;  //访问成员变量
            int d = a;  //final 的局部变量
            int e = b; //访问没有用 final 修饰的局部变量
            @Override
            public void run() {
                System.out.println("这是线程 thread-1");
            }
        };
        // 匿名内部类实现 Runnable 接口
        Runnable r = new Runnable() {
            @Override
            public void run() {
                System.out.println("线程运行中");
            }
        };
    }
}
```

总结

类 型	访问修饰符	声明静态成员	绑定外围类
静态内部类	四种访问修饰符	可以声明	不绑定
成员内部类	四种访问修饰符	除 final static 的变量外，其余静态成员都不行	绑定
局部内部类	不可以声明	不可以声明	取决于此内部类的声明环境
匿名内部类	不可以声明	不可以声明	取决于此内部类的声明环境