内部类

公告

今天学习内容：

- 内部类
  - 内部类定义及作用
  - 内部类的创建
  - 局部内部类
  - 匿名内部类
  - 静态内部类
  - 常见面试问题
- 课后练习

内部类

内部类（inner class），顾名思义就是定义在另一个类中的类。如下所示：

```
public class OuterClass {
    public class InnerClass {
    }
}
```

相对于的，包含了其他类的类，我们称为外部类或外围类（OuterClass）。

为什么需要使用内部类？它主要有如下一些作用：

1. 内部类的方法可以访问该类定义所在的作用域中的数据，包括私有数据。
2. 内部类对同一个包中的其他类不可见。
3. 当要定义一个回调函数而不想编写大量代码时，可以使用匿名内部类（anonymous inner class）。

内部类之所以可以访问包含它的外围类的属性和方法，实际上是因为每一个内部类对象都持有一个指向包含它的外围类对象的引用。这个外围类的引用是在内部类的构造器中设置。编译器修改了所有内部类的构造器，添加了一个外围类引用的参数，和之前对类实例方法调用时的 this 隐藏参数类似。

在内部类中使用外围类引用的正规语法：`OuterClass.this` 表示外围类引用；如果未使用此前缀时，默认访问内部类定义的属性和方法，内部类不存在相同的属性和方法时则访问外围类的属性和方法。如下代码所示：

```
public class OuterClass {
    private Integer index;
    private String name;

    public void setIndex(Integer index) {
        this.index = index;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public void printName() {
        System.out.println(name);
    }

    public class InnerClass {
        private String name;

        public void setName(String name) {
            this.name = name;
        }

        public void printName() {
            System.out.println(name);
            System.out.println("我的外围类Name=" + OuterClass.this.name);
            System.out.println("我的外围类Index=" + index);
        }
    }

    public static void main(String[] args) {
        OuterClass oc = new OuterClass();
        oc.setIndex(1);
        oc.setName("我是外围类");
        oc.printName();

        OuterClass.InnerClass ic = oc.new InnerClass();
        ic.setName("我是内部类");
        ic.printName();
    }
}
```

内部类的创建

定义了内部类后，我们怎么来创建一个内部类的实例？通常有两种方式：

1. 在包含它的作用域范围内，直接使用 `new` 操作符来构造一个实例，和普通类的构造方式一样。
2. 在包含它的作用域范围外，使用 `外围类对象.new 变量名 内部类类名()` 语法来创建；在作用域范围外，我们这样来引用内部类：`OuterClass.InnerClass`

```
public class OuterClass {
    public class InnerClass {
    }

    public InnerClass createInnerClass() {
        return new InnerClass();
    }

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

```
        OuterClass oc = new OuterClass();

        OuterClass.InnerClass ic1 = oc.createInnerClass();
        OuterClass.InnerClass ic2 = oc.new InnerClass();
    }
}
```

局部内部类

除了在类里定义其他类，我们还可以在方法内，或者代码块内等地方来定义一个类。这些地方定义的类我们称为局部内部类。局部内部类有如下一些特性：

- 不能使用 public、protected 或 private 访问修饰符进行声明，它的作用域被限定在声明这个局部类的块中。
- 和其他内部类相比，局部内部类除了可以访问包含它的外围类外，还可以访问作用域范围内的局部变量。但是，这些局部变量必须被声明为 final。

下边我们使用局部内部类来改写一下前边示例中内部 InnerClass 的 printName 方法：

```
public void printName() {
    String method = "printName";

    class LocalInnerClass {
        public void println() {
            System.out.println(name);
            System.out.println("我的外围类Name=" + OuterClass.this.name);
            System.out.println("我的外围类Index=" + index);
            System.out.println("方法名=" + method);
        }
    }

    new LocalInnerClass().println();
}
```

匿名内部类

对于局部内部类，如果我们只创建这个类的一个实例对象，那么我们就没必要给它命名了。这种没有名字的内部类，我们称为匿名内部类（anonymous inner class）。

下边我们再来改写一下上边的 printName 方法：

```
public interface PrintInterface {
    void println();
}
```

```

```

```
public void printName() {
    String method = "printName";

    new PrintInterface() {
        @Override
        public void println() {
            System.out.println(name);
            System.out.println("我的外围类Name=" + OuterClass.this.name);
            System.out.println("我的外围类Index=" + index);
            System.out.println("方法名=" + method);
        }
    }.println();
}
```

上述匿名内部类的语法格式为：

```
new SuperType(construction parameters) {
    inner class methods and data;
}
```

注意：由于构造器的名字必须和类名相同，而匿名类没有类名，所以匿名类不能有构造器，因此上述语法是将构造器参数传递给父类构造器；而当匿名内部类实现的是接口时就不能有任何构造参数。

双括号初始化（double brace initialization）：我们先来回忆一下什么是初始化代码块？那么对于匿名类中是否也可以有初始化代码块？答案是肯定的，比如我要创建一个 List，然后给它添加一些初始值，下边两段代码是等效的：

```
List<Integer> numbs = new ArrayList();
numbs.add(1);
numbs.add(2);
numbs.add(3);
numbs.add(4);
numbs.add(5);
System.out.println(numbs);

List<Integer> numbs2 = new ArrayList() {{
    add(1);
    add(2);
    add(3);
    add(4);
    add(5);
}};
System.out.println(numbs2);
```

静态内部类

有时我们仅仅只需要把一个类隐藏在另一个类中，而并不需要内部类引用外围类对象，这时只需要将内部类声明为 static 即可。

使用 static 声明的内部类称为静态内部类或者嵌套类。

例如：我们有一个方法，它返回指定的一组数据中的最大值和最小值。代码我们可以这样写，如下所示：

```
public class CalcMinMax {
    public int[] calc(int... numbs) {
        int min = 0;
        int max = 0;

        for (int numb : numbs) {
            if (min > numb) {
                min = numb;
            }
            if (max < numb) {
                max = numb;
            }
        }

        return new int[]{min, max};
    }
}

int[] mm = new CalcMinMax().calc(1, 4, 9, 3, 0);
System.out.println("最小值: " + mm[0]);
System.out.println("最大值: " + mm[1]);
```

上述代码中，我们使用了一个二维数组来表示返回值的最小值和最大值，在没有说明的情况下，如果我们不去看实现代码，我们是不知道到底返回数组中哪个表示最小值，哪个表示最大值。因此我们可以创建一个类 Pair 类来容纳最小值和最大值，而 Pair 名字比较大众，其他程序员也可能使用这个名字，这样就会造成混乱冲突。因此我们可以把它放到类 CalcMinMax 的内部作为它的内部静态类。这样我们改造上述代码如下：

```
public class CalcMinMax {
    public Pair calc(int... numbs) {
        int min = 0;
        int max = 0;

        for (int numb : numbs) {
            if (min > numb) {
                min = numb;
            }
            if (max < numb) {
                max = numb;
            }
        }

        return new Pair(min, max);
    }
```

```
    public static class Pair {
        private int min;
        private int max;

        public Pair(int min, int max) {
            this.min = min;
            this.max = max;
        }

        public int getMin() {
            return min;
        }

        public int getMax() {
            return max;
        }
    }
}

CalcMinMax.Pair pair = new CalcMinMax().calc(1, 4, 9, 3, 0);
System.out.println("最小值: " + pair.getMin());
System.out.println("最大值: " + pair.getMax());
```

如上所示，代码的可读性就更好，直接通过返回值对象的方法名就知道值的确切意思而不用再去翻看代码。

对于静态内部类，我们可以直接使用 `OuterClass.StaticInnerClass` 的方式来访问。

内部类本质

内部类其实是一种编译器现象，与虚拟机无关，编译器将会把内部类编译成文件名为【外围类名 + $ + 内部类名】形式的单独的常规文件，而虚拟机则对此一无所知；而对于匿名内部类，编译器会产生一个数字作为其标识符，因此“内部类名”部分会是一个数字。

我们可以打开前边示例生成的 class 文件目录，看一下这些文件名。

闭包

闭包（closure）是一个可调用的对象，它记录了一些信息，而这些信息来自于创建它的作用域。因此我们可以把内部类看作是面向对象的闭包。

常见面试问题

内部类有哪些种类?

内部类和局部内部类的区别?

内部类的作用?

内部类可以被继承吗?

课后练习

1. 自己编写练习一下课程中的示例和练习。

2. 根据课程内容回答常见面试问题。