day11_final、权限、内部类

一.final关键字

学习了继承后，我们知道，子类可以在父类的基础上改写父类内容，比如，方法的覆盖重写。那么我们能不能随意的继承API中提供的类，改写其内容呢？显然这是不合适的。为了避免这种随意改写的情况，Java中提供了final关键字，用于修饰不可改变内容。final 关键字可以用来修饰以下内容：

- 类：被修饰的类，不能被继承。
- 方法：被修饰的方法，不能被子类重写。
- 变量：被修饰的变量，不能被重新赋值。

1.修饰类

```
public final class 类名称 /*extends Object*/ {
    // 任何一个类都默认继承object类
}
```

被final关键字修饰的类，不能被其他类所继承，当然该类所有的成员方法都无法进行覆盖重写。(因为没有子类)

2.修饰方法

```
修饰符 final 返回值类型 方法名称(参数列表) {
    // ...
}
```

被final关键字修饰的方法，不能被子类覆盖重写。而且对于类和方法来说，abstract、final关键字不能同时使用，因为二者相互矛盾。

3.修饰变量

(1).局部变量——基本类型：被final修饰的基本类型局部变量，只能赋值一次，不能再更改。

(2).局部变量——引用类型：被final修饰的引用类型局部变量，只能指向一个对象，地址不能再更改。但不影响对象内部的成员变量值的修改。

(3).成员变量：被final关键字修饰的成员变量的内容也是不可改变的。由于成员变量具有默认值，所以用了final修饰后，必须要对成员变量进行手动赋值。手动赋值的方式有两种：（但是二者只能选其一）

- 法一：直接赋值（直接在定义成员变量时赋值）
- 法二：通过构造方法赋值

#.注意：被final关键字修饰的变量，由于其内容不可改变，所以效果上和常量一样。一般要使用完全大写的字母，用下划线进行分隔单词来命名final变量。(类比：接口中的常量就是使用final关键字修饰)

二.权限修饰符

1.类的权限修饰符

类的权限修饰符只能使用 public、(default) 进行修饰，不可使用其他的权限修饰符。

a. *public class 类名{ }*：类名称必须与文件名称完全一致；被 public 修饰的类可以被其他包访问。(不在同一个package下的类要进行导包操作)

b. *class 类名{ }*：类名称可以与文件名称不一致；被 (default) 修饰的类只能在本包之内使用。(在本包之外，使用导包操作也不可以使用)

2.成员的权限修饰符(成员变量、成员方法、构造方法)

使用不在同一个package下的类要进行导包的操作，但是即使使用import语句把这个类给导入了，也不一定能使用这个类的全部内容，要看其中成员的权限修饰符、两个类的包位置关系。下表就展示了不同权限修饰符的访问能力。(此表首先要保证类能被访问到)

在Java中提供了四种权限修饰符来修饰成员，四种修饰符的权限大小关系为： public > protected > (default) > private

	public	protected	default(空的)	private
同一类中	√	√	√	√
同一包中(子类、无关类)	√	√	√	×
不同包的子类	√	√	×	×
不同包中的无关类	√	×	×	×

#.注意:

a.类的权限修饰符决定了该类是否能被其他类访问。在类能被访问到的基础上，类中成员的权限修饰符决定了该成员是否能被其他类访问。

b.编写代码时，如果没有特殊的考虑，建议这样使用权限修饰符:

- 类使用 public ，方便被其他包里的类访问
- 成员变量使用 private ，封装保护属性
- 成员方法使用 public ，方便调用方法
- 构造方法使用 public ，方便创建对象

三.内部类

在描述事物时，若一个事物内部还包含其他事物，就可以使用内部类这种结构，即一个类的内部包含另一个类。例如：身体和心脏的关系、汽车和发动机的关系。内部类主要分为以下几种:

1.成员内部类

成员内部类是定义在类中、方法外(成员位置)的类，相当于是外部类的一个成员。定义格式如下:

```
修饰符  class 外部类名称 {

    修饰符  class 内部类名称 {
        // …
    }

    // …

}
```

(1).成员内部类的访问特点:

a.可以直接在内部类中访问外部类的所有成员，包括私有成员。[内用外，随意访问]

b.外部类要访问内部类的成员，必须要先建立内部类的对象。 [外用内，需要内部类对象]

- 在外部类中：创建内部类对象访问内部类成员
- 在其它类中：创建内部类对象访问内部类成员。格式如下:

```
外部类名称.内部类名称 对象名 = new 外部类名称().new 内部类名称();
```

(2).成员内部类的访问权限:

由于成员内部类是外部类的一个成员，所以可以使用public、protected、(default)、private四种权限修饰符来修饰。此时可参照上述 "成员权限访问表"，得到成员内部类的访问权限如下。

a.在外部类中：无论内部类是何种权限修饰符，均可以访问到内部类

b.在其它类中：根据内部类的权限修饰符、类的包位置关系，并参照上表即可确定能否使用该内部类。

*小拓展：成员内部类里面的成员的访问权限又是如何呢？也可参照上表得出结果。（前提是能访问到该内部类）

(3).成员内部类的变量重名问题:

2.局部内部类

如果一个类是定义在方法内部的，那么就是一个局部内部类。"局部"的含义：只有在当前方法的局部范围内才能使用它，出了这个方法就不能用了。定义格式如下:

```
修饰符  class 外部类名称 {

    修饰符 返回值类型 外部类方法名称(参数列表) {

        class 局部内部类名称 {
            // ...
        }

    }

}
```

(1).局部内部类的使用:

由于局部内部类是定义在外部类方法里面的，所以局部内部类只能在这个方法内部使用。只能在方法内部创建 "局部内部类对象" 使用。

(2).局部内部类的访问权限:

局部内部类不能使用任何权限修饰符，但是局部内部类里面的成员可以使用四种权限修饰符，并且里面的成员无论使用何种权限修饰符，都能被方法内部的局部内部类对象访问到。

(3).局部内部类的final问题：

如果局部内部类希望访问到所在方法的局部变量，那么这个局部变量必须是 "有效final" 的。什么是"有效final"呢：局部变量被final修饰、或者不被final修饰但此变量的值后面不会再改变。（从Java 8+开始，只要方法内的局部变量后面不会再改变，那么它就默认被final修饰）

那么为什么它必须是"有效final"的呢？原因如下：new出来的局部内部类对象在堆内存当中，局部变量随着方法进栈在栈内存中产生，随着方法出栈而消失，但是new出来的对象会在堆内存中持续存在，直到被垃圾回收。如果局部变量消失的时候，堆内存中的对象还想使用局部变量怎么办呢？其实为了解决这个问题，堆内存中的对象一开始就会将局部变量的值复制一份出来放在堆内存中进行使用，所以这才要求局部变量后面不能再改变。

3.匿名内部类

当我们使用一个抽象父类、一个接口时，要做以下几步操作：

- 第一步：定义一个子类来继承抽象父类、实现接口
- 第二步：重写抽象父类、接口中的抽象方法
- 第三步：创建子类对象，调用重写后的方法

我们的目的只是为了调用子类中重写的方法，那么能不能简化一下，把以上步骤合并成一步呢？匿名内部类就是这样的快捷方式。其定义格式如下：

```
new 抽象父类名称()  {
  // 覆盖重写所有抽象方法
};
```

```
new 接口名称() {
  // 覆盖重写所有抽象方法
};
```

如果抽象父类的子类、接口的实现类只需要使用唯一的一次，那么就可以省略掉该子类的定义，而改为使用匿名内部类。匿名内部类是抽象父类的子类、接口的实现类的简化替代形式，就是在方法的局部范围内创建一个一次性的匿名的局部内部类，并且这个内部类必须要重写抽象父类、接口中所有的抽象方法，所以匿名内部类也是局部内部类的一种。

(1).new代表创建了一个对象，所以匿名内部类的本质是"抽象父类、接口的一个匿名实现类的对象"

(2).一般会使用抽象父类引用、接口引用来指向匿名内部类的对象，即多态。此时使用多态的方式调用父类、接口中的方法，就可以访问到匿名内部类重写后的方法。

#.注意：所有的内部类都仍然是一个独立的类，虽然没有独立的.java文件，但是在编译之后会内部类会被编译成独立的.class文件，文件名前冠以外部类的类名和$符号 。比如，Body$Heart.class，可在相应的 out 文件夹中查看。

四.引用类型用法总结

基本类型可以作为成员变量、方法的参数、方法的返回值的数据类型，那么引用类型也是可以的。

1.class 作为成员变量的数据类型

类作为成员变量的数据类型时，对它进行赋值的操作，实际上，是赋给它该类的一个对象。

2.interface 作为成员变量的数据类型

接口作为成员变量的数据类型时，对它进行赋值的操作，一般是赋给它该接口的一个子类对象。接口的子类对象被接口类型的引用所接收，形成多态。

3.interface 作为方法参数和返回值类型

接口作为方法的参数、返回值的数据类型时，一般传递与返回的都是该接口的子类对象。此时子类对象的传递与返回，都是使用接口类型来接收，形成多态。

五.综合案例——发红包[界面版]