OO前三次作业总结

# 知识点总结

## 我理解的封装、继承、多态

几乎所有的教科书在讲到面向对象的特点的时候，都会说提到这三个词。这三次实验几乎也是严格遵循这三个特征来设计的。查阅过很多资料，大家说的几乎也都是面向对象的好，面向对象要强于面向过程等等诸如此类的论调。其实我觉得这几个词其实并没有那么重要，网上其他人的说法也没有那么重要，真正重要的还是自己对于这样一种编程思想的认识和理解。下面是我自己对于这些思想的一些理解。

先来谈谈封装。我觉得封装算是OOP相对于面向过程编程最大的区别了，也是所有面向对象的抽象基础。封装最根本的目的有两个，一是为了保护数据，二是为了把一些比较零散的属性划归为一个对象，更加符合我们在生活中对于事物的认知。

谈到保护数据，无外乎就是用到了我们又爱又恨的private, protected, default, public声明了。在用以上四个词修饰类或者类中的属性方法的时候，他们的使用范围就受到了限定。

下面斜体是教科书一样的无聊念白，不爱看的可跳过：

Private是类内权限，它只能生命类内的方法属性，不能修饰外部类，被他修饰的方法或属性只能在类内使用； protected是父子类权限，类中的方法和属性只能在子类中使用，同样不能修饰外部类；默认（default）为包权限，不做声明的变量可以在同包的类中使用，包外的；public为公共权限，用它修饰的方法在包内包外的所有类中均可见。

使用这些权限修饰符的好处就在于我们可以轻松地分清哪些属性是仅在类里面被调用的，哪些需要公开给其他类使用，并且可以有效地防止数据被篡改，因为限定的区域外根本就没有这些属性的访问权限。变量们的作用域各不相同，比如一个循环结构里面的i作为计数数字肯定是封装在方法里面的，而电梯调度中电梯自身的楼层一定要有外部获得的方法。为了方便使用而将所有属性设为public也是不可取的，因为这样就失去了封装的本来意义。这里就有一个需要避免的误区了。虽然private声明的变量很“安全”,不会被轻易篡改，但是当外部类需要频繁获取一个属性的时候，不妨将其设为public。将所有属性设为private，然后添加所有类的Getter & Setter方法是一种为了面向对象而面向对象的行为，在我看来这是不明智的。借用知乎上的一个对于良好封装的定义：

真正的封装是，经过深入的思考，做出良好的抽象，给出“完整且最小”的接口，并使得内部细节可以对外透明。对外透明的意思是，外部调用者可以顺利的得到自己想要的任何功能，完全意识不到内部细节的存在；而不是外部调用者为了完成某个功能、却被碍手碍脚的private声明弄得火冒三丈；最终只能通过怪异、复杂甚至奇葩的机制，才能更改他必须关注的细节——而且这种访问往往被实现的如此复杂，以至于稍不注意就会酿成大祸  
https://www.zhihu.com/question/20275578

接着我们来谈继承。个人认为继承最重要的一点是提高了代码的复用，为将来程序的扩展打下了基础，节省了开发时间。正是因为有了继承这一特征，我们才能够不断地构建越来越复杂的对象。从头写一个1000行的financialOfficer类可能会让我们感到有些压力，但是如果我们从300行的Person核心类写起，事情可能就会看上去稍稍简单一些。我们可以不断地为这个软件系统增添更多的组件，而这一切都是通过继承来实现的。当我们新写出来的类出现Bug时，我们也只需要将精力集中在新类的代码中而无需顾及其父类的方法，当然前提是父类已经经过了严格的测试。通过继承，我们可以忽略编程中的很多细节，从而更加专注于我们当前对象所需要实现的事情。需要注意的是，java并不支持多继承，但是可以通过接口来实现多继承的效果。（虽然接口都是抽象类）java中所有类的子类为Object类，一般在创建新类的时候都要重写Object类的toString和equals方法。子类可以继承父类中的所有方法，如果子类不喜欢父类方法的实现，可以对该方法进行重写。但是需要注意的是，方法重写必须保证方法的名称和参数完全相同，否则不构成重写。重写后的方法的访问权限必须不低于原有的父类方法的权限，否则也不能够形成重写。子类可以通过super关键字来调用父类的方法，super最常用于子类的构造方法中。继承在给我们带来极大便利的同时，还有着一些隐患。首先，新类代码需要在父类的代码上实现，那么当我们不断地添加新类的子类的时候，我们就有可能会离最早的父类非常遥远。比如：

Son extends parent, grandson extends son, grand\_grandson extends grandson

这里的parent可能只有很少的几百行代码，而grand\_grandson类中却可能拥有上万行代码。这个机制可能会导致一种现象——我们越早书写的类，就越与我们的核心功能无关，以至于这些类的作用就被弱化掉了。为了避免这种现象，我们需要在程序设计之初想清楚自己的程序架构。尽早地创建尽可能多的class往往不是解决问题的好方法，那恰恰会大大加大程序逻辑混乱的程度。

最后我们来聊多态。个人认为多态机制是面向对象中最复杂的机制，但同时也是极大地加深了面向对象抽象层次的一个机制。我自己对于多态的理解其实也不是特别深刻，所以这里就少说一点。

*多态是指相同的输入可以根据发送对象的不同而产生不同的效果。*当然这句话是从网上摘来的，我觉得这是句屁话，因为它完全不能够体现多态的效果及原理。多态来源于继承，他是面向对象归一化方法的体现。老师上课对于多态举得例子是：同一个F1键，在Word里面就显示Word的帮助文档，在Eclipse下就显示Eclipse的帮助文档。他们都可以看作软件的子类，接受到的输入相同，但是却拥有不同的相应结果。多态实现需要三个必要条件：继承，重载，父类引用指向子类对象。

我更倾向于用一个例子来阐释多态的含义，比如下面的这个程序。

**class** A {  
 **public** String show(D d){  
 **return** (**"A and D"**);  
 }  
 **public** String show(A a){  
 **return** (**"A and A"**);  
 }  
}  
  
**class** B **extends** A{  
 **public** String show(B b){  
 **return** (**"B and B"**);  
 }  
 **public** String show(A a){  
 **return** (**"B and A"**);  
 }  
}  
**class** C **extends** B{}  
**class** D **extends** B{}  
  
**public class** learnPoly {  
 **public static void** main(String[] args){  
 A a1 = **new** A();  
 A a2 = **new** B();  
 B b = **new** B();  
 C c = **new** C();  
 D d = **new** D();  
 System.***out***.println(a1.show(b));  
 System.***out***.println(a1.show(c));  
 System.***out***.println(a1.show(d));  
 System.***out***.println(a2.show(b));  
 System.***out***.println(a2.show(c));  
 System.***out***.println(a2.show(d));  
 System.***out***.println(b.show(b));  
 System.***out***.println(b.show(c));  
 System.***out***.println(b.show(d));  
 }  
}

很简单的一个程序，有兴趣的读者可以尝试计算一下主函数的9条指令的执行结果。答案为：

A and A

A and A

A and D

B and A

B and A

A and D

B and B

B and B

A and D

如果你感觉三观收到了颠覆，那很正常。因为我特喵的折腾这个程序想了一晚上2333.如果你的预期和结果完全一致，那么我只想说你真的很厉害。至于解析我打算在本博客的最后给予公布。

## 接口

接口和抽象类有些类似，但是又有所不同。首先和类继承不同的是，接口可以由多个类同时实现，一个类也可以实现多个接口。这就极大地丰富了java类的扩展性。通常来说，接口是类的某种性质的抽象，也就是说它包括了这类类的某种性质。比如汽车和人都可以运动。那么它们都可以实现Movable接口，因为这是他们所共有的属性。但是人和汽车的运动是不同，所以他们需要各自重写接口中的抽象方法。比如Movable中有speed，direction等属性，只需要各自重写就能够体现出各个体之间的差异，但是我们又很好理解汽车和人类运动之间的相似性。类的继承关系一般可以看作是一种“纵向“的，因为他的属性由爸爸传给儿子再由儿子传给孙子，这是一种严格的一对多关系。但是接口的实现更像是横向的扩展，比如同样是父类的儿子，但是他们的差异可以通过继承的接口不同来进一步地加以体现。同时，当需要使用接口中相应的方法的时候，也可以通过新建匿名类重写方法的方式来使用。这种方法在Android编程中经常被利用（比如Listener接口）。

## 对象的可变与不可变性

我觉得这算是比较零碎的知识点了。不可变（immutable）对象，指的是对象在创建完成之后就不会被修改，例如我们最常见的字符串String 就是如此。只要字符串被创建完成，其内容就不会发生变化了。至于”a” + “b”这种操作只是生成了一个新的字符串，而没有改变原有的“a”和”b“。这一点还是需要记住的。可变对象当然就是和不可变对象相对的了，个人认为我们在程序中操作的大多是可变对象。

关于对象的可变和不可变，在程序设计之初就应该被考虑清楚。例如在完成作业2的时候，生成的电梯请求Request可以看作是不可变的，因为请求一旦生成，他的类型、目标楼层、时间等属性就不会发生变化了，但是保存请求的队列RQueue应该是可变的，因为它要负责请求的增删改查，是我们所需要操作的主要类之一。分析清楚对象的可变性有利于我们梳理类之间的关系，并把重要方法尽量安插在可变的对象之中。

# 前三次作业的分析和收获

前三次的程序还算是比较简单，总得来说没有碰到技术上的瓶颈。

第一次的程序由于没有考虑到第一个括号前的符号失了分，总体的设计架构和PPT中基本相同，因为当初是照着课本建的类，加上程序比较简单，因此相似度很高，也没有完全体现面向对象的好处。

第二次程序没有被扣分，其实第二次的程序还是比较容易实现的，我把Request类设为了不可变，并新建了一个RQueue类作为操作类。在电梯类中新建了方法用来改变电梯的属性。控制类负责读取队列并调度电梯。

第三次是因为捎带队列的某一种捎带情况没考虑全而丢了分。这次作业才有点儿面向对象编程的意思，虽然周围有很多人表示类都要重写的了。但是好在第二次程序功能设计的比较合理，其他类的代码修改的比较少。最终设计的结果和图中有些许不同，我并没有把Carry的请求当作Request的子类，而是直接在队列中进行了操作。我至今仍认为把Request区别对待是没有意义的，只要在调度时加以区分即可。

这些都是比较细节的地方，今后自查的时候还是需要再精细一些。《软件测试的艺术》中说程序走查是找Bug的最全的方法，也许今后可以和室友试一试。

关于收获，我觉得最大的收获是学会使用正则表达式了。虽然前三次作业中都没有使用正则表达式，但是啃代码学到的东西总是能够给人深刻的印象。我打算将第三次作业中的输入方法改成正则识别，相信这样一定能够进一步压缩代码量。

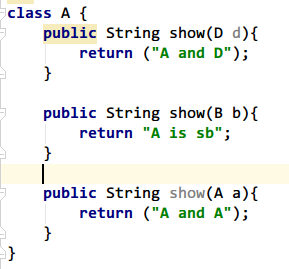
# 问题的解析

解：a1为A声明的A对象，a2为A声明的B对象，b,c,d为B,C,D对象。

首先前三个应该是比较好理解的：

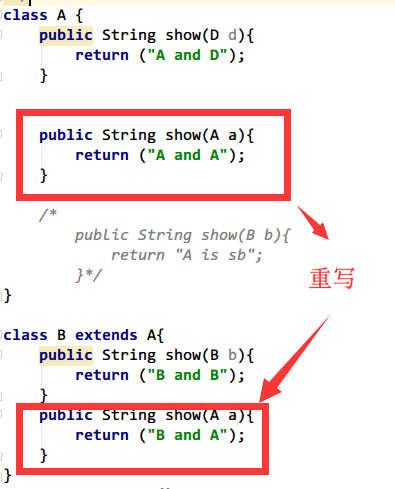
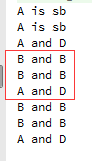
对于1~3：

1. 实例化对象为A，参数为B。由于B是A的子类，且A类中没有显式的B为参数的方法，因此执行的是A中的show(A a)方法。倘若我们在A类中加入一个新的show方法，其参数类型为B，那么将会有不同的效果。



1. 同1，只不过多了一步继承而已
2. 实例化A对象，参数为D，直接调用A中的show(D d)方法

对于4~6：要注意显示是声明了一个A类的变量a2并在内存中创建了一个B类的对象。B类重载了A中的show(A a)方法.

1. 实例化对象依然是A，参数是B。由于A类中没有B为参数的方法，应该将B向上转型执行的是A类中的show(A a)方法。但是由于该方法被B类重写了，因此最终显示的是B类中的show（A a）方法。如下图：  
     
   如果我们此时在A类中新建一个show(B b)的方法，结果又会发生改变，最终执行的将是B类中的show(B b)方法，结果就会发生改变。  
   
2. 同4.
3. 直接调A中的方法，没啥可说的。

7~9：其实这三个又很简单了，只需要弄懂前三个就能懂了

1. b为声明的B类对象，因此输入参数为B的时候自动调用B中的Show（B b）
2. 同7
3. 执行从A继承过来的show(D d)方法。如果A中没有show(D d)方法，那么执行结果将和7、8相同。