程序设计模式作业三

2020年3月2日　　9:56

第三次作业 面向对象设计原则（下）

作业要求：

1、讨论：正方形是否是长方形的子类？（结合里氏代换原则）。

2、结合面向对象程序设计原则，谈谈对类和接口"粒度"的理解。

3、有人将面向对象设计原则简单归为三类：

(1) 封装变化点。

(2) 对接口进行编程。

(3) 多使用组合，而不是继承。

请查阅相关资料，结合本章的内容，谈谈对这三条原则的理解。

- Q1：讨论：正方形是否是长方形的子类？（结合里氏替换原则）
- A1：

首先在这里我们先回顾一下什么是里氏替换原则。里氏替换原则通俗来讲就是在软件中如果能够使用基类对象，了么一定能够使用其子类对象。把基类都替换成它的子类，程序将不会产生任何错误和异常，反过来则不成立，如果一个软件实体不是子类的话，那么它不一定能够使用基类。

```
private int length;
private int width;
public void setLength(int lenght) {
this.length = lenght;
}
public void setWidth(int width) {
this.width= width;
}
public int getArea() {
return this.length * this.width;
}
```

如果说正方形是长方形的子类。为保证正方形的长和宽相等，那对应于长方形的长宽设置就得改成

```
public void setLength(int lenght) {
this.length = lenght;
this.width= lenght;
}
public void setWidth(int width) {
this.length = width;
this.width= width;
}
```

我们都知道长方形的面积等于长与宽的积。那当我们用长方形的时候我们会这样用来计算

```
Rectangle rectangle = new Rectangle();
rectangle.setLength(5);
```

```
rectangle.setWidth(4);
```

思考当我们给正方形的实例给用户会怎么样？（用户只知道长方形而不知道真正的实例是正方形）所以得到的面积

```
rectangle.setLength(5);
rectangle.setWidth(4);
rectangle.getArea();
>>16
```

我们也可以得出，如果子类隐藏了父类的方法，实现不一样，定然会根据对象的静态类型调用，也就定然违反了里式替换原则。继承必须确保超类所拥有的性质在子类中仍然成立。”也就是说，当一个子类的实例应该能够替换任何其超类的实例时，它们之间才具有is-A关系。所以在面向对象中正方形并不是长方形。

- Q2：结合面向对象程序设计原则，谈谈对类和接口“粒度”的理解。
- A2：
  对于这个问题，在本科的软件工程的课程中曾经简单了解过。个人理解为粗粒度容纳的逻辑多；细粒度容纳的逻辑少。
  粗粒度：表示类别级，即仅考虑对象的类别(the type of object)，不考虑对象的某个特
  定实例。比如，用户管理中，创建、删除，对所有的用户都一视同仁，并不区分操作的具体对象实例。
  细粒度：表示实例级，即需要考虑具体对象的实例(the instance of object)，当然，细
  粒度是在考虑粗粒度的对象类别之后才再考虑特定实例。比如，合同管理中，列表、删除，需要区分该合同实例是否为当前用户所创建。
  一般权限的设计是解决了粗粒度的问题，因为这部分具有通用性，而细粒度可以看成业务部分，因为其具有不确定性

- Q3：有人将面向对象设计原则简单归为三类：
  （1）封装变化点。
  （2）对接口进行编程。
  （3）多使用组合，而不是继承。
  请查阅相关资料，结合本章的内容，谈谈对这三条原则的理解
- A3：
  a. 封装变化点。隔离变化点的好处在于，将系统中经常变化的部分和稳定的部分隔离，有助于增加复用性，并降低系统耦合度。很多设计模式的意图中都明显地指出了其对问题的解决方案，学习设计模式的要点是发现其解决方案中封装的变化点。
  b. 对接口进行编程。这里“接口”的含义表示的程序设计语言中的interface ,或者abstract class。对接口编程的一个好处在于客户端程序并不需要了解具体的实现，而只需要了解接口中声明的方法。更大的好处在于能够使用多态性执行动态性的行为。
  c. 多使用组合，而不是继承。Has-a关系要比Is-a关系更好。因为继承是静态行为，

也就是编译时行为。这种设计缺乏灵活度，并且具有比组合更高的耦合度。而组合是动态行为，即运行时行为。可以通过使用组合的方式在设计上获得更高的灵活性。GOF设计模式中将设计模式分为对象设计模式和类设计模式，其中对象设计模式居多，原因就在于对象设计模式多使用组合，通过此获得更好的灵活性。

- *参考：《GOF设计模式》，《设计模式解析》*