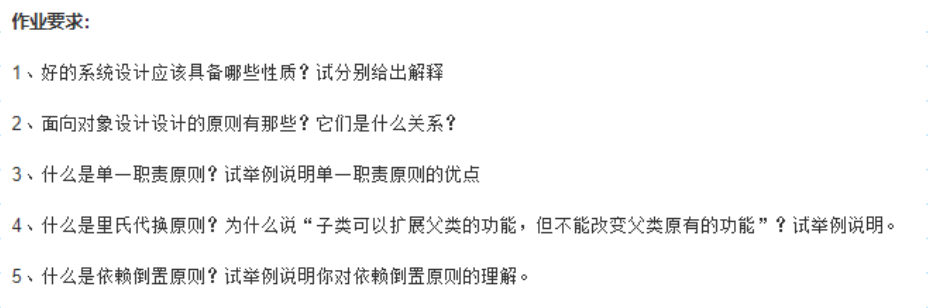
程序设计模式作业二

2020年2月23日 21:13



- Q1:好的设计系统应该具备哪些性质？试分别给出解释（PPT P3）

  A1：

  a. 在本科学习的软件工程中，曾学习到构件系统中的构件应该是高内聚，低耦合的，但构件之间应当有若干种关系，例如继承关系（即一个构件可从其他构件那里继承其功能）；可以发送消息给其他构件；可以与其他构件联合，支持协同工作。对于一个设计良好的系统，也应当具有高内聚、低耦合的特征。

  b. 在面向对象设计原则来看，设计系统应该具有可维护性和可复用性，一个好的设计系统应该具有可扩展性、灵活性和可插入性。

     i. 可复用性or重用性的优点：

     提高开发效率、质量，节约开发成本，恰当的复用还可以改善系统的可维护性。当然，面向对象设计复用的目标在于实现支持可维护性的复用。遵循这些设计原则可以有效的提高系统的复用性，同时提高系统的可维护性。

     ii. 重构是在不改变系统的功能的基础上，比如通过调整程序代码改善软件的质量、性能，使其程序的设计模式和架构更趋于合理，提高了扩展性和维护性。

- Q2:面向对象设计的原则有哪些?他们是什么关系?(PPT P8)

  A2:

  a. 面向对象的设计原则有七个，包括：开闭原则、里氏替换原则、迪米特原则（最少知道原则）、单一职责原则、接口隔离原则、依赖倒置原则、组合/聚合复用原则。

  b. 七大原则之间并不是相互孤立的，彼此间存在着一定关联，一个可以是另一个原则的加强或是基础。违反其中的某一个，可能同时违反了其余的原则。

  开闭原则是面向对象的可复用设计的基石。其他设计原则是实现开闭原则的手段和工具。

  一般地，可以把这七个原则分成了以下两个部分：

  设计目标：开闭原则、里氏替换原则、迪米特原则

  设计方法：单一职责原则、接口隔离原则、依赖倒置原则、组合/聚合复用原则

- Q3:什么是单一职责原则,为什么说"子类可以扩展父类的功能,但不能改变父类原有的功能"?试举

例说明（PPT P23）

A3:

a. 单一职责原则定义：

定义1：如果对每一个类型为 T1的对象 o1，都有类型为 T2 的对象o2，使得以 T1定义的所有程序 P 在所有的对象 o1 都代换成 o2 时，程序 P 的行为没有发生变化，那么类型 T2 是类型 T1 的子类型。

定义2：所有引用基类的地方必须能透明地使用其子类的对象。

通俗简单的说就是：子类可以扩展父类的功能，但不能改变父类原有的功能

b. "子类可以扩展父类的功能,但不能改变父类原有的功能"这里提到便是里氏替换原则。

i. 子类可以实现父类的抽象方法，但不能覆盖父类的非抽象方法。

ii. 子类中可以增加自己特有的方法。

iii. 当子类的方法重载父类的方法时，方法的前置条件（即方法的输入/入参）要比父类方法的输入参数更宽松。

iv. 当子类的方法实现父类的方法时（重载/重写或实现抽象方法）的后置条件（即方法的输出/返回值）要比父类更严格或相等。

E.G

```
class A{
   public int func1(int a, int b){
      return a-b;
   }
}
public class Client{
   public static void main(String[] args){
      A a = new A();
      System.out.println("100-50="+a.func1(100, 50));
      System.out.println("100-80="+a.func1(100, 80));
   }
}
```

运行结果：

100-50=50

100-80=20

后来，我们需要增加一个新的功能：完成两数相加，然后再与100求和，由类B来负责。

即类B需要完成两个功能：

两数相减。

两数相加，然后再加100。

由于类A已经实现了第一个功能【两数相减】，所以类B继承类A后，只需要再完成第二个功能【两数相加，然后再加100】就可以了

```
class B extends A{
   public int func1(int a, int b){
      return a+b;
   }
```

```
    public int func2(int a, int b){
        return func1(a,b)+100;
    }
}
public class Client{
    public static void main(String[] args){
        B b = new B();
        System.out.println("100-50="+b.func1(100, 50));
        System.out.println("100-80="+b.func1(100, 80));
        System.out.println("100+20+100="+b.func2(100, 20));
    }
}
```

类B完成后，运行结果：

100-50=150

100-80=180

100+20+100=220

我们发现原本运行正常的相减功能发生了错误。**原因就是类B在给方法起名时无意中重写了父类的方法**，造成所有运行相减功能的代码全部调用了类B重写后的方法，造成原本运行正常的功能出现了错误。

- Q4：什么是单一职责原则？试举例说明单一职责原则的优点

A4:

a. 单一职责原则的定义为：一个对象应该只包含单一的职责，并且该职责被完整的封装在一个类中；另一种定义为：就一个类而言应该仅有一个引起他变化的原因。

b. 单一职责原则的优点

i. 类的复杂性降低，实现什么职责都有清晰明确的定义；

ii. 可读性提高，复杂性降低；

iii. 可维护性提高

iv. 提高系统可维护性

- Q5：什么是依赖倒置原则？试举例你对依赖倒置原则的理解。

A5：

a. 依赖倒置原则定义为：

i. 高层模块不应该依赖于低层模块，二者都应该依赖于抽象

ii. 抽象不应该依赖于细节，细节应该依赖于抽象

iii. 针对接口编程，不要针对实现编程。

b. 理解：

i. 依赖：在程序设计中，如果一个模块a使用/调用了另一个模块b，我们称模块a依赖模块b。

ii. 高层模块与低层模块：往往在一个应用程序中，我们有一些低层次的类，这些类实现了一些基本的或初级的操作，我们称之为低层模块；另外有一

些高层次的类，这些类封装了某些复杂的逻辑，并且依赖于低层次的类，这些类我们称之为高层模块。

iii. 依赖倒置：面向对象程序设计相对于面向过程（结构化）程序设计而言，依赖关系被倒置了。因为传统的结构化程序设计中，高层模块总是依赖于低层模块。