设计模式原则 - 里氏替换原则

寂然

聊聊继承性

继承优势

- 提高代码的复用性（每个子类有拥有父类的属性和方法）
- 提高代码的可扩展性

继承劣势

- 继承是侵入性的（只要继承，就必须拥有父类的属性和方法）
- 继承机制很大的增加了耦合性

继承其实是一把双刃剑 ==》如何正确合理使用继承呢？==》里氏替换原则

官方定义

里氏替换原则（Liskov Substitution Principle，LSP）是1988年，麻省理工学院一位姓里的女士提出的，

官方定义如下：

> **If for each object o1 of type S there is an object o2 of type T such that for all programs P defined in terms of T,the behavior of P is unchanged when o1 is substituted for o2 then S is a subtype of T.**
>
> 如果对每一个类型为S的对象o1，都有类型为T的对象o2，使得以T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S是类型T的子类型
>
> **Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.**
>
> 所有引用基类的地方必须能透明地使用其子类的对象

基本介绍

> 里氏替换原则通俗的来讲：子类可以扩展父类的功能，但是子类不能修改父类原有的功能
>
> 里氏替换原则就是给继承性的使用制定了规范

案例演示 - 计算器

案例分析

解决方案

```
package com.kkb.designpatternprinciple.liskovsubstitutionprinciple;

//案例演示 - 里氏替换原则
public class LiskovDemo {
    public static void main(String[] args) {

        int result = new Calculator().add(3, 5);
        System.out.println(result);

        int mul = new SuperCalculator().mul(3, 5);
        System.out.println("两数相加之和与100求差的值为" + mul);

    }
}

//需求：现在有一个计算器（父类）可以完成加减乘除，定义其子类，来演示继承可能出现的问题

//创建一个更加基础的基类（定义更加基础的成员或者功能）

class Base{

    //TODO....

}


class Calculator extends Base{

    //定义加法功能
    public int add(int a,int b){

        return a + b;
    }

    //定义减法功能
    public int sub(int a,int b){

        return a - b;
    }
```

```
    //TODO...
}


class SuperCalculator extends Base{

    //变为依赖关系
    private Calculator calculator = new Calculator();


    //增补需求，两数相加再加5

    public int add(int a,int b){

        return a + b + 5;
    }

    //希望两数相加之和与100求差
    public int mul(int a,int b){

        int count = calculator.add(a, b);

        return 100 - count;

    }


}
```

注意事项

一，子类可以实现父类的抽象方法，**但是不能覆盖父类的非抽象方法**

二，子类中可以扩展自己的方法

三，**里氏替换原则并非让我们尽量避免使用继承**

四，里氏替换原则是实现开闭原则的重要方式之一

下节预告

开闭原则