```
package com.principle.liskov.improve;


public class Liskov {
    public static void main(String[] args) {
        A a=new A();
        System.out.println("11-3=" + a.func1(11, 3)); System.out.println("1-8=" +
a.func1(1, 8));

        System.out.println("----------------------");
        B b = new B();
        //因为 B 类不再继承 A 类，因此调用者，不会再 func1 是求减法
        //调用完成的功能就会很明确
        System.out.println("11+3=" + b.func1(11, 3));//这里本意是求出 11+3


        System.out.println("1+8=" + b.func1(1, 8));// 1+8
System.out.println("11+3+9=" + b.func2(11, 3));


        //使用组合仍然可以使用到 A 类相关方法
        System.out.println("11-3=" + b.func3(11, 3));// 这里本意是求出 11-3
    }
}

//创建一个更加基础的基类
class Base {
//把更加基础的方法和成员写到 Base 类
}

// A 类
class A extends Base {
    // 返回两个数的差
    public int func1(int num1, int num2) {
        return num1 - num2;
    }
}

// B 类继承了 A


// 增加了一个新功能：完成两个数相加,然后和 9 求和
class B extends Base {
    //如果 B 需要使用 A 类的方法,使用组合关系
    private A a = new A();
```

```
    //这里，重写了 A 类的方法，可能是无意识
    public int func1(int a, int b) {
        return a + b;
    }

    public int func2(int a, int b) {
        return func1(a, b) + 9;
    }

    //我们仍然想使用 A 的方法
    public int func3(int a, int b) {
        return this.a.func1(a, b);
    }
}
```