# 七大设计原则

## 单一职责 Single Responsibility

1) 降低类的复杂度，一个类只负责一项职责。

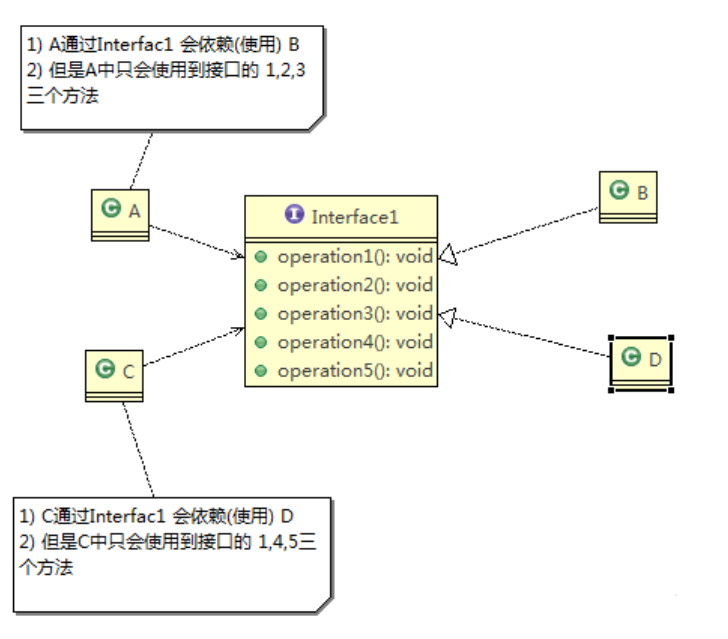
2) 提高类的可读性，可维护性。

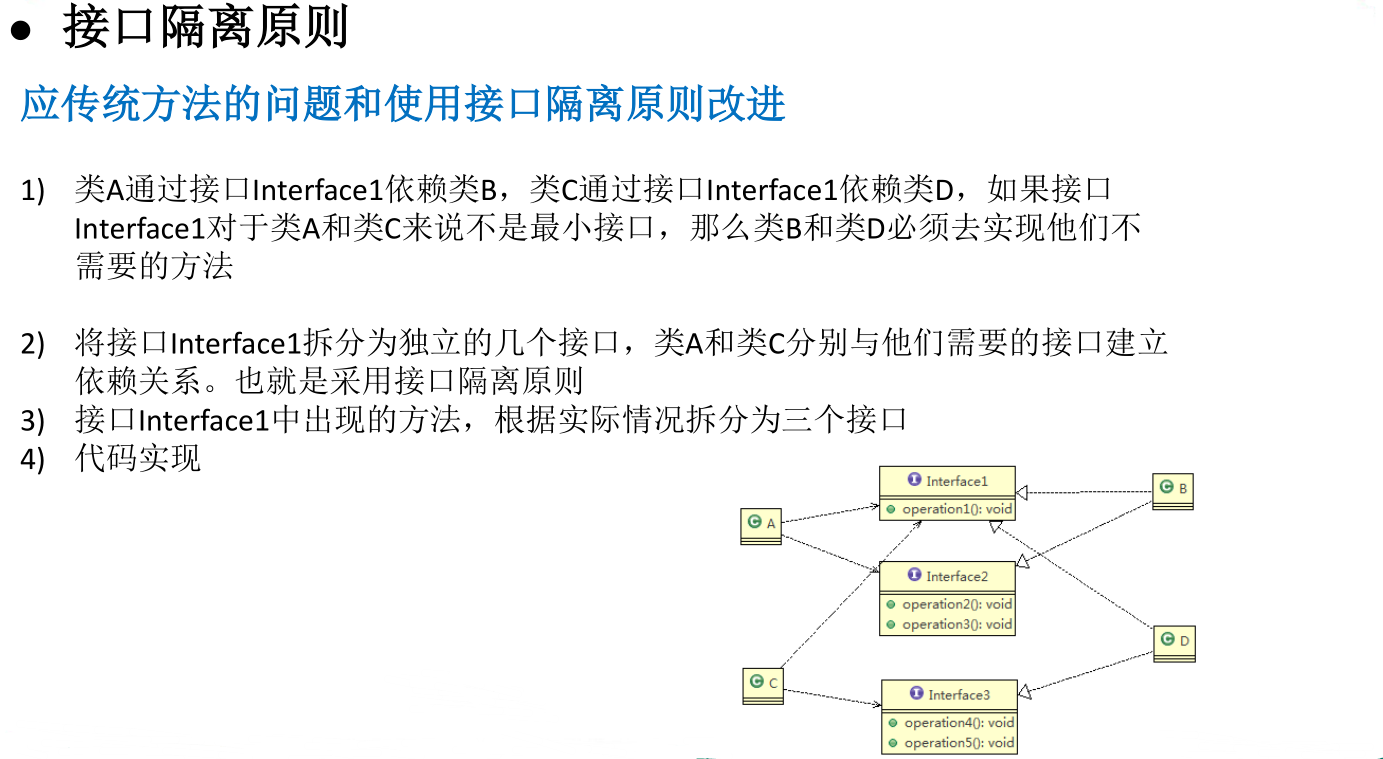
3) 降低变更引起的风险。

4) 通常情况下， 我们应当遵守单一职责原则，只有逻辑足够简单，才可以在代码级违反单一职责原则；只有类中方法数量足够少，可以在方法级别保持单一职责原则。

## 接口隔离原则 Interface Segregation Principle

拆接口。一个类对另一个类的依赖应该建立在最小的接口上。





## 依赖倒转原则 Dependence Inversion Princple

1) 高层模块不应该依赖低层模块，二者都应该依赖其抽象。

2) 抽象不应该依赖细节，细节应该依赖抽象。

3) 依赖倒转(倒置)的中心思想是面向接口编程。

4) 依赖倒转原则是基于这样的设计理念：相对于细节的多变性，抽象的东西要稳定的多。以抽象为基础搭建的架构比以细节为基础的架构要稳定的多。在java中，抽象指的是接口或抽象类，细节就是具体的实现类。

5) 使用接口或抽象类的目的是制定好规范，而不涉及任何具体的操作，把展现细节的任务交给他们的实现类去完成。

细节

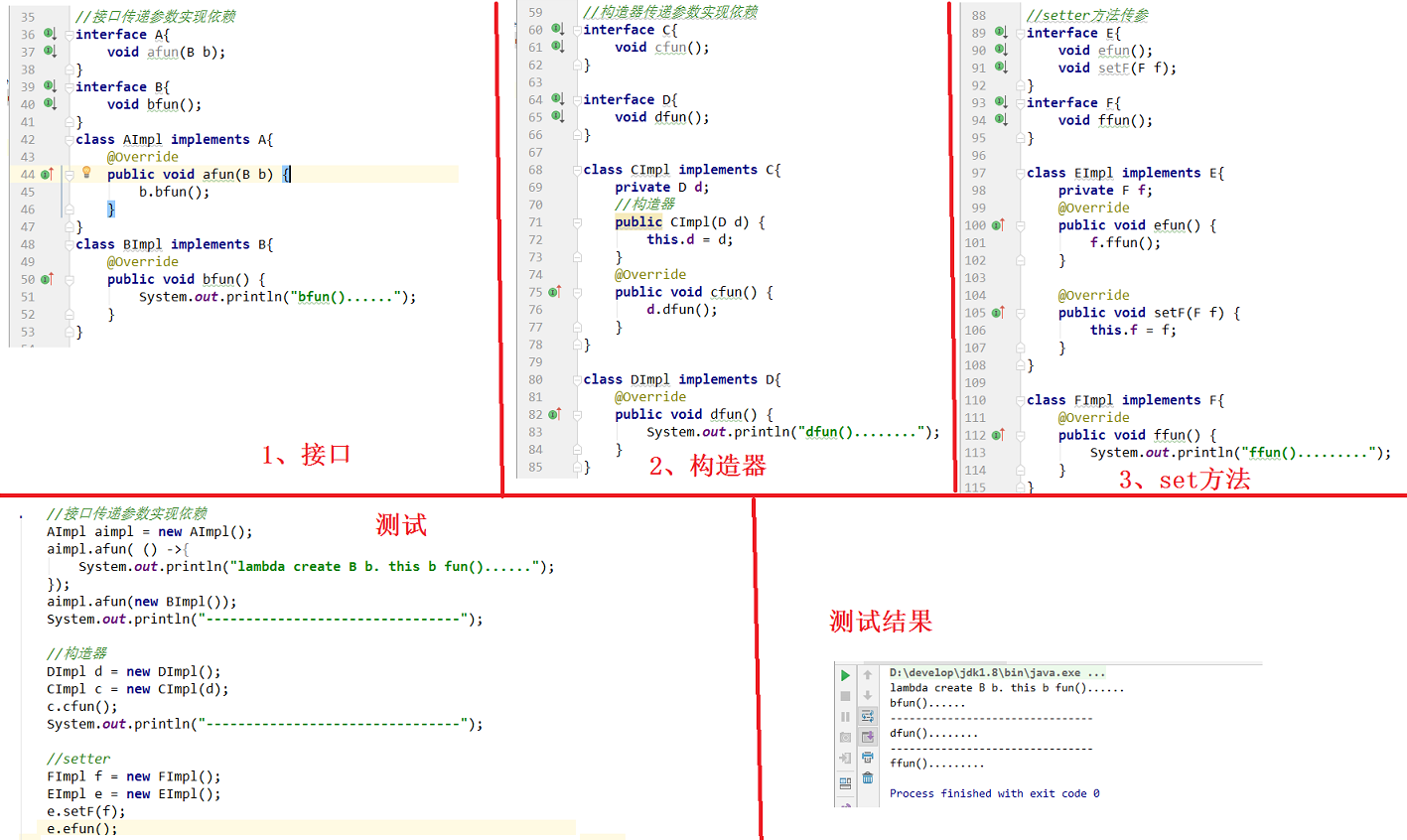
1) 低层模块尽量都要有抽象类或接口，或者两者都有，程序稳定性更好.

2) 变量的声明类型尽量是抽象类或接口, 这样我们的变量引用和实际对象间，就存在一个缓冲层，利于程序扩展和优化。

3) 继承时遵循里氏替换原则。

依赖倒转的时候，一般有三种传递方式。

接口、构造方法、setter方法。



## 1.4里氏替换原则

**OO 中的继承性的思考和说明**

1) 继承包含这样一层含义：父类中凡是已经实现好的方法，实际上是在设定规范和契约，虽然它不强制要求所有的子类必须遵循这些契约，但是如果子类对这些已经实现的方法任意修改，就会对整个继承体系造成破坏。

2) 继承在给程序设计带来便利的同时，也带来了弊端。比如使用继承会给程序带来侵入性，程序的可移植性降低，增加对象间的耦合性，如果一个类被其他的类所继承，则当这个类需要修改时，必须考虑到所有的子类，并且父类修改后，所有涉及到子类的功能都有可能产生故障。

1. 问题提出：在编程中，如何正确的使用继承? =>里氏替换原则

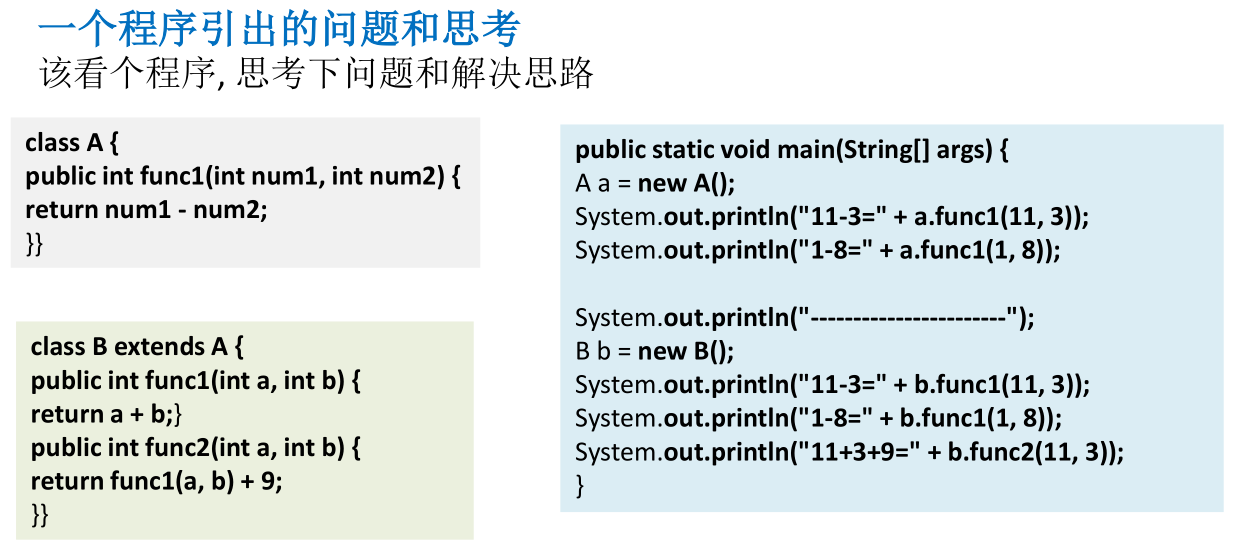
**基本介绍**

1) 里氏替换原则(Liskov Substitution Principle)在1988年，由麻省理工学院的以为姓里的女士提出的。

2) 如果对每个类型为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。换句话说，所有引用基类的地方必须能透明地使用其子类的对象。

3) 在使用继承时，遵循里氏替换原则，在子类中 尽量不要重写父类的方法。

4) 里氏替换原则告诉我们，继承实际上让两个类耦合性增强，在适当的情况下，通过聚合，组合，依赖来解决问题



**解决方法**

1) 我们发现原来运行正常的相减功能发生了错误。原因就是类B无意中重写了父类的方法，造成原有功能出现错误。在实际编程中，我们常常会通过重写父类的方法完成新的功能，这样写起来虽然简单，但整个继承体系的复用性会比较差。特别是运行多态比较频繁的时候

2) 通用的做法是：原来的父类和子类都继承一个更通俗的基类，原有的继承关系去掉，采用依赖，聚合，组合等关系代替.