# 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。

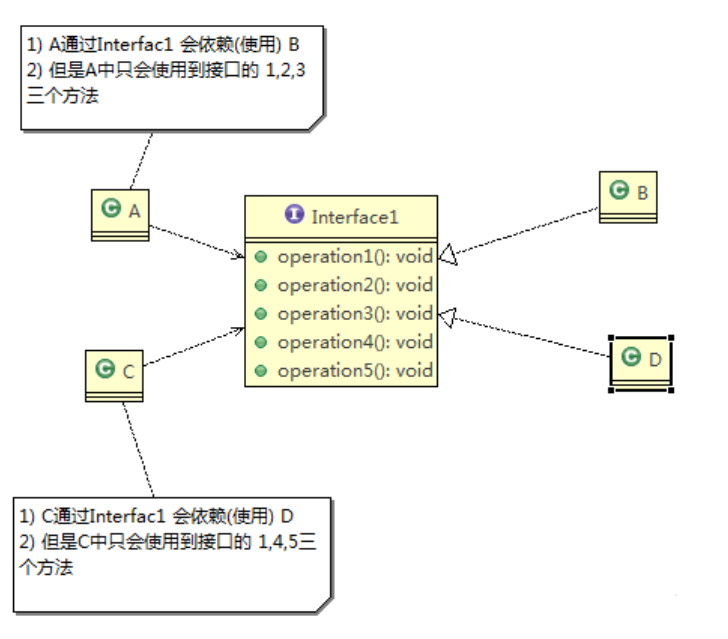
2) 提高类的可读性，可维护性。

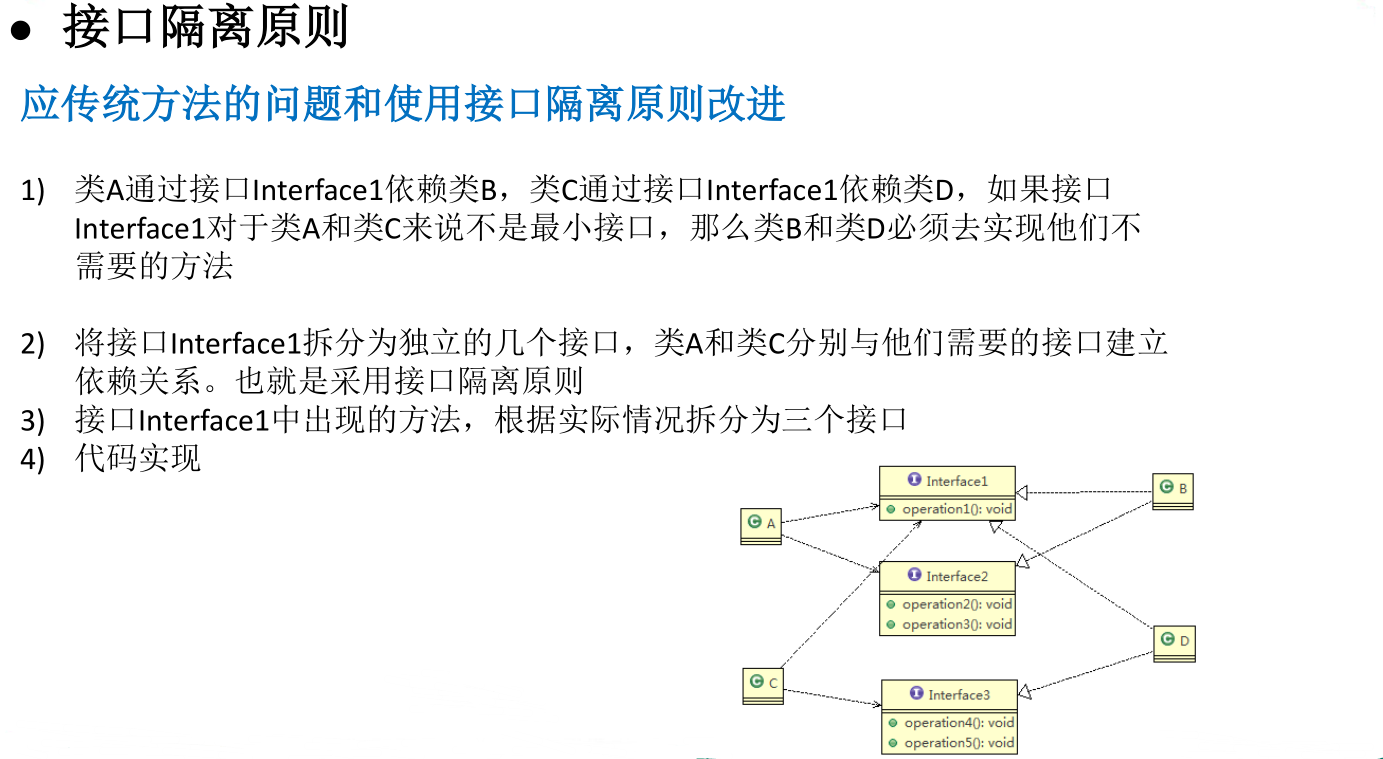
3) 降低变更引起的风险。

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

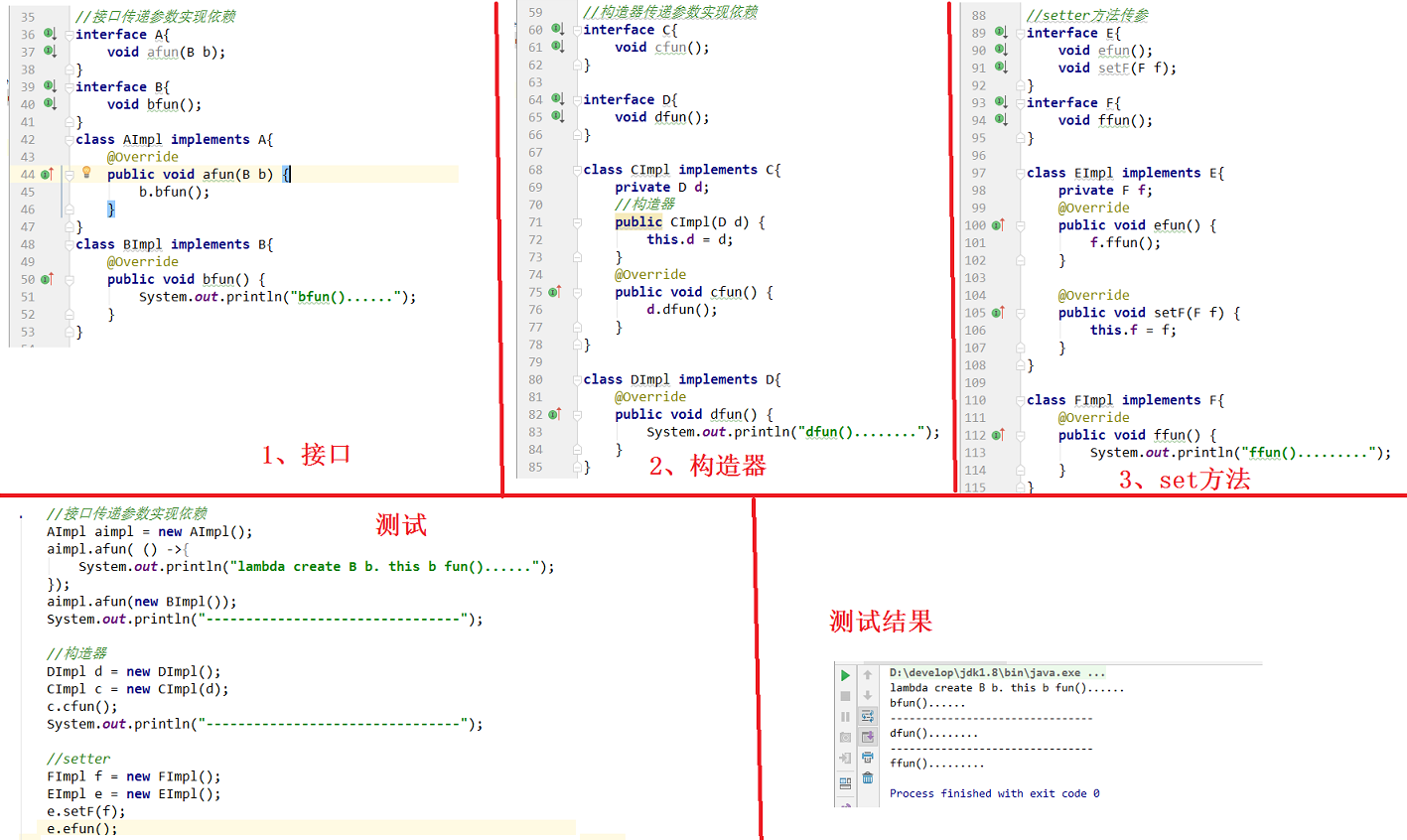
1) 低层模块尽量都要有抽象类或接口，或者两者都有，程序稳定性更好.

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 Liskov Substitution Principle

**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

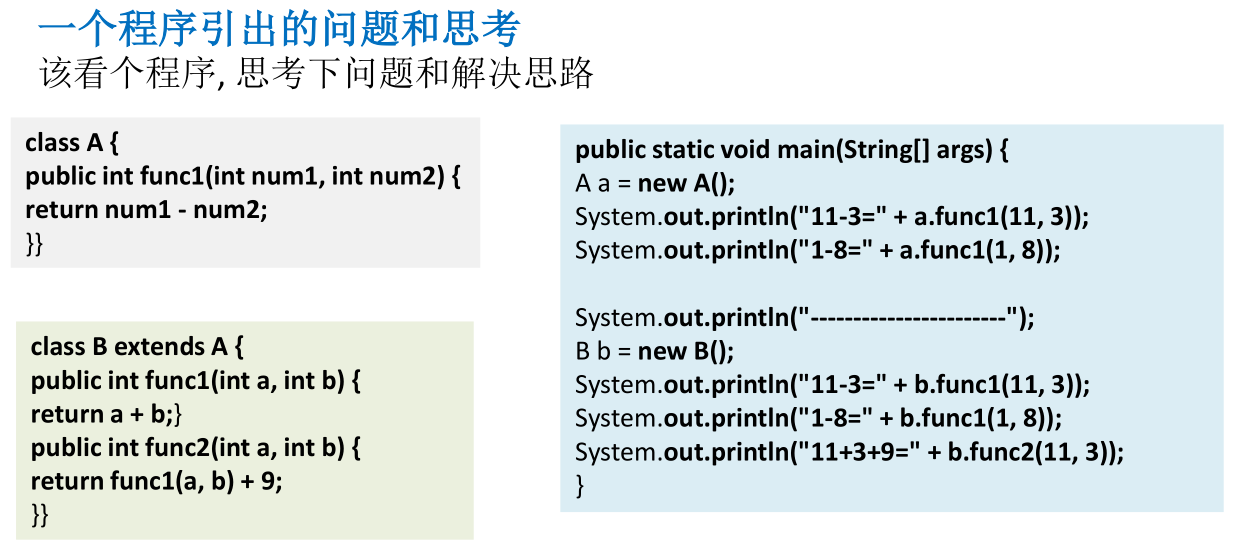
**基本介绍**

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.

## 1.5、开闭原则 Open Closed Principle

1. 开闭原则是编程中 最基础、最重要的设计原则。

2) 一个软件实体如类，模块和函数应该 对扩展开放( 对提供方)，对 修改关闭( 对使用方)。用抽象构建框架，用实现扩展细节。

3) 当软件需要变化时，尽量 通过扩展软件实体的行为来实现变化，而不是 通过修改已有的代码来实现变化。

4) 编程中遵循其它原则，以及使用设计模式的目的就是遵循开闭原则。

## 1.6、迪米特法则 Demeter Principle

1) 一个对象应该对其他对象保持最少的了解

2) 类与类关系越密切，耦合度越大

3) 迪米特法则(Demeter Principle)又叫 最少知道原则，即一个类 对自己依赖的类知道的越少越好。也就是说，对于

被依赖的类不管多么复杂，都尽量将逻辑封装在类的内部。对外除了提供的 public 方法，不对外泄露任何信息

4) 迪米特法则还有个更简单的定义：只与直接的朋友通信

5) 直接的朋友：每个对象都会与其他对象有 耦合关系，只要两个对象之间有耦合关系，我们就说这两个对象之间

是朋友关系。耦合的方式很多，依赖，关联，组合，聚合等。其中，我们称出现 成员变量， 方法参数， 方法返

回值中的类为直接的朋友，而出现在 局部变量中的类不是直接的朋友。也就是说，陌生的类最好不要以局部变量的形式出现在类的内部。

迪米特法则的核心是降低类之间的耦合

但是注意：由于每个类都减少了不必要的依赖，因此迪米特法则只是要求降低类间(对象间)耦合关系， 并不是要求完全没有依赖关系

## 1.7、合成复用原则 Composite Reuse Principle

1) 找出应用中可能需要变化之处，把它们独立出来，不要和那些不需要变化的代码混在一起。

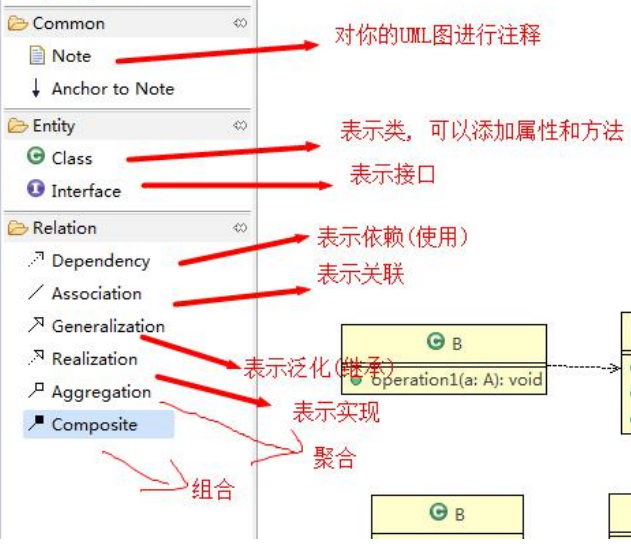
2) 针对接口编程，而不是针对实现编程。

3) 为了交互对象之间的 松耦合设计而努力。

尽量使用合成/聚合的方式，而不是使用继承。



# UML 类图



## 2.1、依赖

1) 类中用到了对方

2) 如果是 类的成员属性 private A a

3) 如果是 方法的返回类型 public A getA()

4) 是方法 接收的参数类型 public void getA(A a)

5) 方法中使用到 方法局部变量

## 2.2、泛化 （继承）

## 2.4、实现

## 2.5、关联

## 2.6、聚合

可以分离

private A a;

## 2.7、组合

不可以分离

private A a = new A();

# 三、设计模式概述和分类

第1层：刚开始学编程不久，听说过什么是设计模式

第2层：有很长时间的编程经验，自己写了很多代码，其中用到了设计模式，但是自己却不知道

第3层：学习过了设计模式，发现自己已经在使用了，并且发现了一些新的模式挺好用的

第4层：阅读了很多别人写的源码和框架，在其中看到别人设计模式，并且能够领会设计模式的精妙和带来的好处。

第 5 层：代码写着写着，自己都没有意识到使用了设计模式，并且熟练的写了出来。

## 设计模式

设计模式是程序员在面对同类软件工程设计问题所总结出来的有用的经验， 模式不是代码，而是 某类问题的通

用解决方案，设计模式（Design pattern）代表了最佳的实践。这些解决方案是众多软件开发人员经过相当长的

一段时间的试验和错误总结出来的。

设计模式的本质提高 软件的维护性，通用性和扩展性，并降低软件的复杂度。

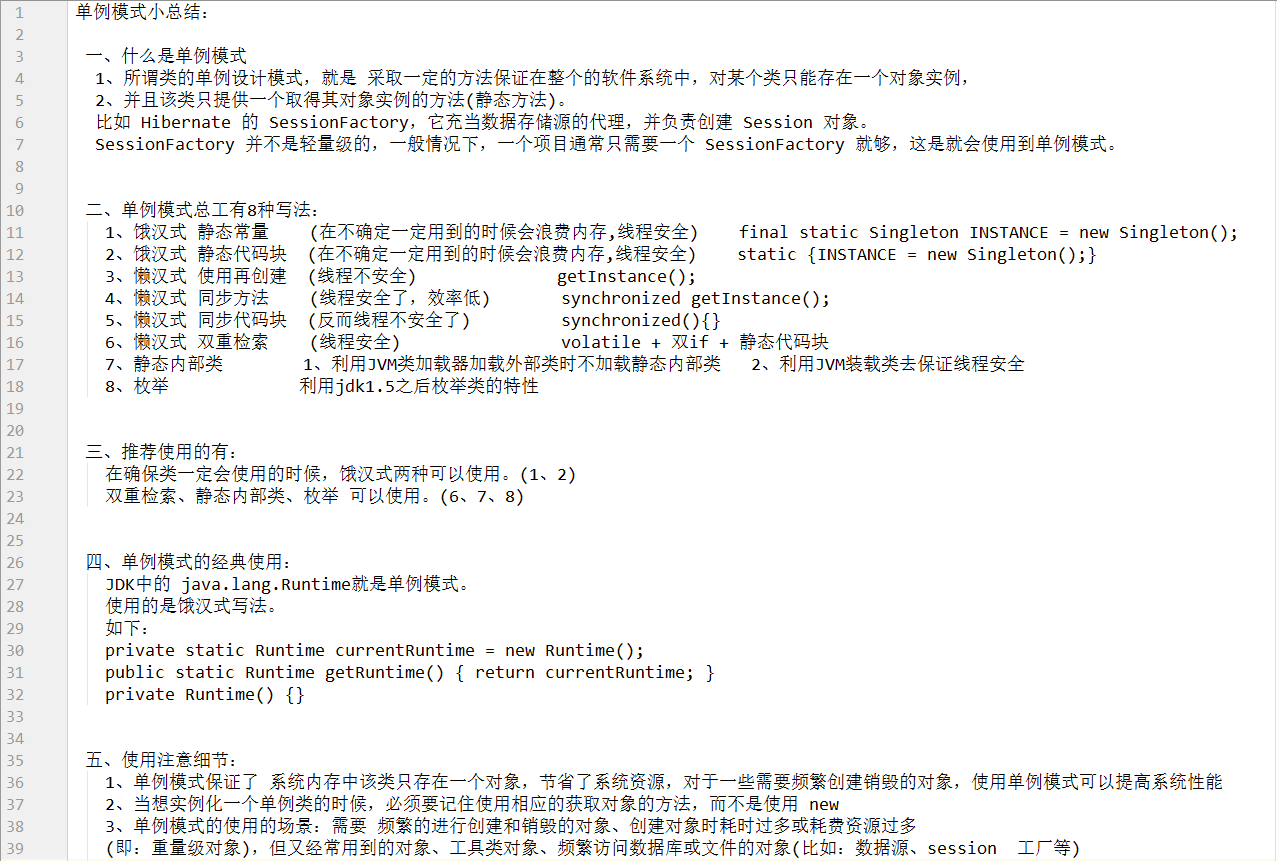
3) << 设计模式>> 是经典的书，作者是 Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides Design（俗

称 “四人组 GOF”）

## 三种类型，共 23 种

1. 创建型模式：单例模式、抽象工厂模式、原型模式、建造者模式、工厂模式。
2. 结构型模式：适配器模式、桥接模式、装饰模式、组合模式、外观模式、享元模式、代理模式。
3. 行为型模式：模版方法模式、命令模式、访问者模式、迭代器模式、观察者模式、中介者模式、备忘录模式、解释器模式（Interpreter 模式）、状态模式、策略模式、职责链模式(责任链模式)。

# 四、单例



五、

六、