最基本的设计原则

# 接口理解

## 狭义的接口：

Java中定义的接口Interface，其内的方法都是抽象的。

## 广义的接口：

不仅包括Java中定义的接口，广义的接口是超类型，通常用于多态，包括狭义的接口、抽象类和一些基类。

## 要注意狭义接口与抽象类的区别：

Java中规定接口内的所有的方法都是抽象的，而抽象类的内部既可以包括抽象方法，也可以包括具体方法。抽象类需要使用abstract修饰。

## 通常所说的对外提供接口

API中：应用程序接口，Interface。

对外提供接口：可以是狭义的接口，也可以是抽象类或基类。

# 设计原则

设计原则有很多，学习一些常用的设计原则很重要。

## 封装变化原则

## 多用组合，少用继承原则

## 针对接口编程，不针对实现编程

## 为交互对象之间的松耦合设计而努力

## 类应该对扩展开放，对修改关闭

## 依赖抽象，不要依赖具体类

## 最少知识原则：只和密友交谈

# 第一个设计原则：封装变化

把会变化的部分取出并单独“封装”起来，好让其他部分不会受到影响。、

结果：**代码变化引起的不经意后果变少，系统变得更有弹性**。

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。

应用场景：如果每次新的需求一来，都会使某方面的代码发生变化，那么你就可以确定，这部分的代码需要被抽出来，和其他稳定的代码有所区别。

把会变化的部分取出来并单独封装起来，以便以后可以轻便地改动或扩充此部分，而不影响不需要变化的其他部分。

这个设计原则很简单，几乎是每一个设计模式背后的精神所在。所有的设计模式都提供了一套方法让“**系统中的某部分改变不会影响其他部分**”。

# 第二个设计原则：多用组合，少用继承

将两个类结合起来使用，就是**组合**。

使用组合建立系统，具有很大的弹性，不仅可以将算法族封装成类，更可以“在运行时动态地改变行为”，只要将组合的行为对象符合正确的接口标准即可。

# 第三个设计原则：针对接口编程，不针对实现编程

# 第四个设计原则：为交互对象之间的松耦合设计而努力

# 第五个设计原则：类应该对扩展开放，对修改关闭

# 第六个设计原则：依赖抽象，不要依赖具体类

# 第七个设计原则：最少知识原则，只和密友交谈

最少知识原则（**Least Knowledge**），告诉我们要减少对象之间的交互，只留下几个“密友”。

**最少知识原则**：只和你的密友谈话。

具体意思：当你正在设计一个系统，不管是任何对象，你都要注意它所交互的类有哪些，并注意它和这些类是如何交互的。

这个原则希望我们在设计中，不要让太多的类耦合在一起，免得修改系统中一部分，会影响到其他部分。 如果许多类之间相互依赖，那么这个系统就会变成一个易碎的系统，它需要花许多成本维护，也会因为太复杂而不容易被其他人了解。

如System.out.println();就违背了最少知识原则。