1. 设计模式常用七大原则
   1. 单一职责

比如userDao就只操作user表

* 1. 接口隔离

尽量让接口小型化,拆的细一些,把不相关的方法隔离开

* 1. 依赖倒转

有几种依赖的方式,一是直接在consumer的方法里传参(类or接口),二是把依赖的接口变成consumer的成员变量,可以使用构造器或者setter方法来初始化.依赖的尽量使用接口,这样扩展性强很多

* 1. 里氏替换

尽量不要重写父类里的方法.或者尽量不用使用继承,而是使用组合,依赖,聚合

* 1. 开闭

对扩展开放对修改关闭,自己可以扩展,别人不能修改

* 1. 迪米特法则

直接朋友: A与B两个类 如果在A中，B是成员变量或者方法参数或者方法返回值，那么就是直接朋友。如果是在A的某个方法里new一个B用，那么就是陌生的。不建议这样用。 这个法则的核心就是降低类之间的耦合

* 1. 合成复用

1. UML建模
2. 单例模式
   1. 饿汉式

私有化构造器，然后在静态变量后面直接new对象，或者写到静态代码块里

Private A(){};

Private static final A instance = new A();

/\*

Static{

Instance = new A();

}

\*/

Public static A getInstance(){

Return instance;

}

总结：在类装载的时候初始化对象，不会有线程安全问题，但是没有懒加载，容易造成内存浪费

* 1. 懒汉式

Private A(){};

Private static A instance;

Public static A getInstance(){

If(null == instance){

Instance = new A();

}

Return instance;

}

优点： 懒加载

缺点：线程不安全

线程安全的懒汉

Private A(){};

Private static A instance;

Public static synchornized A getInstance(){

If(null == instance){

Instance = new A();

}

Return instance;

}

* 1. 双重检查可以避免懒加载和线程安全问题（略）
  2. 静态内部类

Public class A{

Private A(){};

Private static class AHolder{

Private static final A instance = new A();

}

Public static A getInstance(){

Return AHolder.instance;

}

}

因为在装在外部类的时候内部类不会被装载，所以实现了懒加载而且线程安全

* 1. 枚举（推荐）

1. 工厂模式
   1. 简单工厂

Public class Factory{

Public Pizza createPizza(){

Return new Pizza();

}

}

当多个客户端都都需要创建某一类对象时，可以把创建对象的逻辑写在工厂里，客户端只需要依赖工厂，从而降低重复代码，也可以解耦客户端与类之间的关系

* 1. 工厂方法模式

没有工厂类了，直接在client端程序去写一个抽象工厂方法，然后子类去实现它，写不不同的逻辑

* 1. 抽象工厂模式

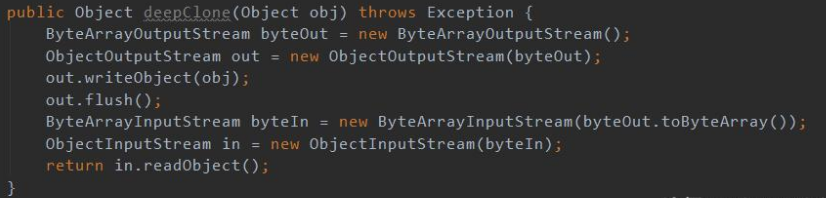
这个其实是简单工厂和工厂方法的结合，只不过这个更符合单一职责原则，还是要新创建一个工厂类，不和客户端业务代码耦合。

不同的工厂实现类生产不同的对象，比如小米工厂生产小米手机，华为工厂生产华为手机，但是他们都是继承手机工厂

1. 原型模式
   1. 重写clone方法然后实现cloneable接口
   2. 浅拷贝：当一个对象里有引用对象，copy外层对象时候，只是copy了外层对象，里层对象还是指向之前对象的引用。 这样的话修改新对象里的那个对象的值时候，就会影响之前对象

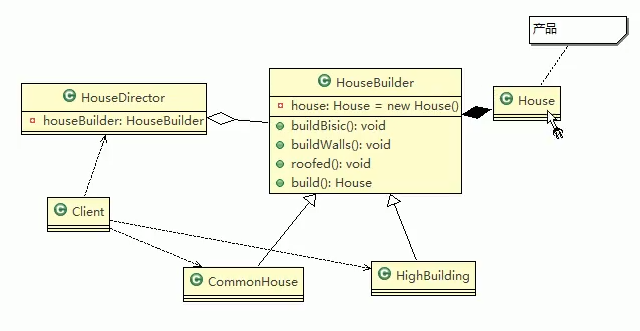
深拷贝：里层外层对象都是新的 ==后是false

* 1. 深拷贝的实现方式
     1. 重写clone方法，把里面的类也clone一遍
     2. 通过对象的序列化实现



1. 建造者模式

核心思想就是把产品的建造过程和产品本身分开。 可以把复杂的建造过程隐藏起来，客户用起来方便



建造者模式的核心在于在builder中的每一个setter方法都返回本身的builder

Public class header{

Private String a;

Private String b;

Private String c;

Private String d;

Private String e;

}

Public class HeaderBuilder{

Private String a;

Private String b;

Private String c;

Private String d;

Private String e;

Public HeaderBuilder seta(String a){

This. a = a;

Return this;

}

….

Public Header buildHeader(){

Return new Header(a, b….);

}

}

1. 适配器模式
   1. 类适配器

当客户端的某个方法使用target接口的一个方法

Class client{

Public void a(){

Target.method2();

}

}

但是我们想要在调用方法里使用另外一个类的方法，但是这个类却没有实现target接口，在这里我们可以创建一个新的adapter类实现target接口并继承想用的那个类，在method2方法里调用那个类的方法。这样就可以在不改变client类代码的前提下，也使用到了想要用的类里的方法

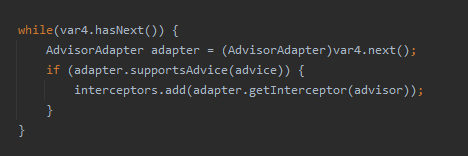
* 1. 对象适配器

这是常用的一种方法，代替了上一个方法里的继承，而是使用组合或者聚合的方法来调用adptee里的方法

* 1. 接口适配器（不常用）
  2. 在springmvc的HandlerAdapter中使用了适配器模式

在doDispatch方法中，根据调用request进来的handler也就是比如注解的controller，实现接口的controller，请求静态资源，servlet。根据不同的handler来获取不同的HandlerAdapter。然后下面代码统一执行HandlerAdapter接口的handle方法，这个方法的入参是handler，在handle方法里调用handler的具体执行方法。这样直接使用了适配器的接口在client代码中，所有的子类adapter都可以用

在AOP的DefaultAdvisorAdapterRegistry类中，需要根据advisor注册interceptor，这里获取getInterceptor也是适配器模式



在这里，不同的advisor会有不同的AdvisorAdapter，不同的AdvisorAdapter也有不同的getInterceptoer实现方法