# 基础

开闭原则:尽量不修改原来代码，基础上扩展

## 多态

A imp P

B imp P

P p = new A();

P p = new B();

# 设计模式

## 单例模式

返回固定的一个实例

饿汉

先new好了

懒汉

创建再new，volatile 指令重排，静态块上锁，内外双重判断

静态内部类

外部类加载时并不需要立即加载内部类，内部类不被加载则不去初始化INSTANCE，故而不占内存

枚举

## 策略模式

Comparable：实体类实现接口，必须实现比较方法。

Comparator：策略模式，在Comparable设计基础上扩展，写一个比较器实现Compartor的比较方法，在排序方法中传入T[] 和Compartor 作为参数使用，更为灵活。

## 工厂模式

只要是返回对象的方法或者类都是工厂

使用接口和抽象方法都可以，语义问题，形容词用接口，名词用抽象类

简单工厂：能返回多种的

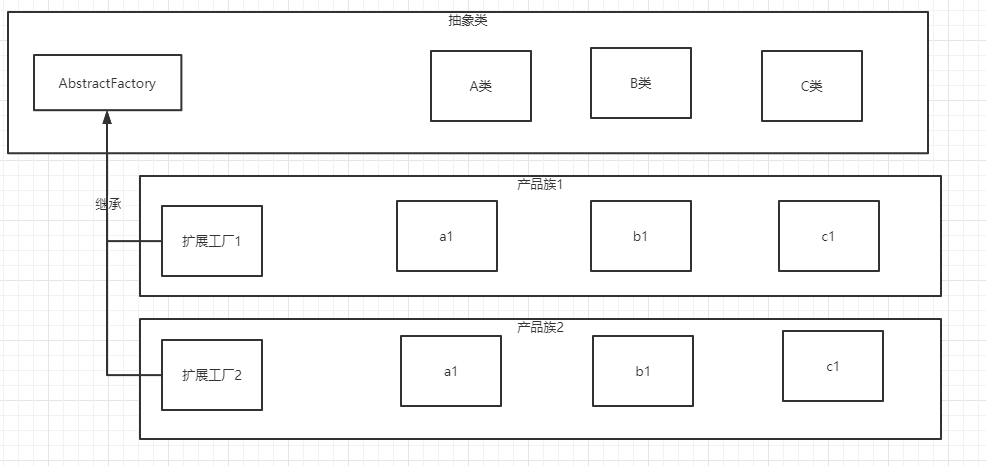
静态工厂：单例

工厂方法模式：

产品维度扩展使用接口类接收，传入不同参数，返回不同对象

抽象工厂：

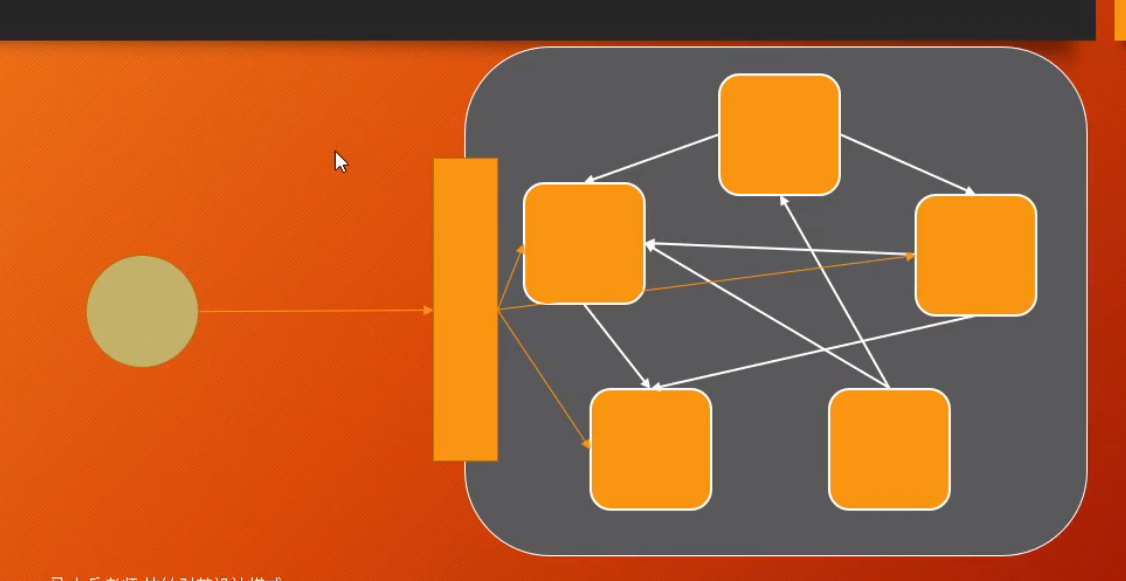
产品族扩展，AbstractFactory定义三个方法返回A B C，具体工厂类去实现三个方法返回不同的产品



bean工厂

## 门面Façade （对外）

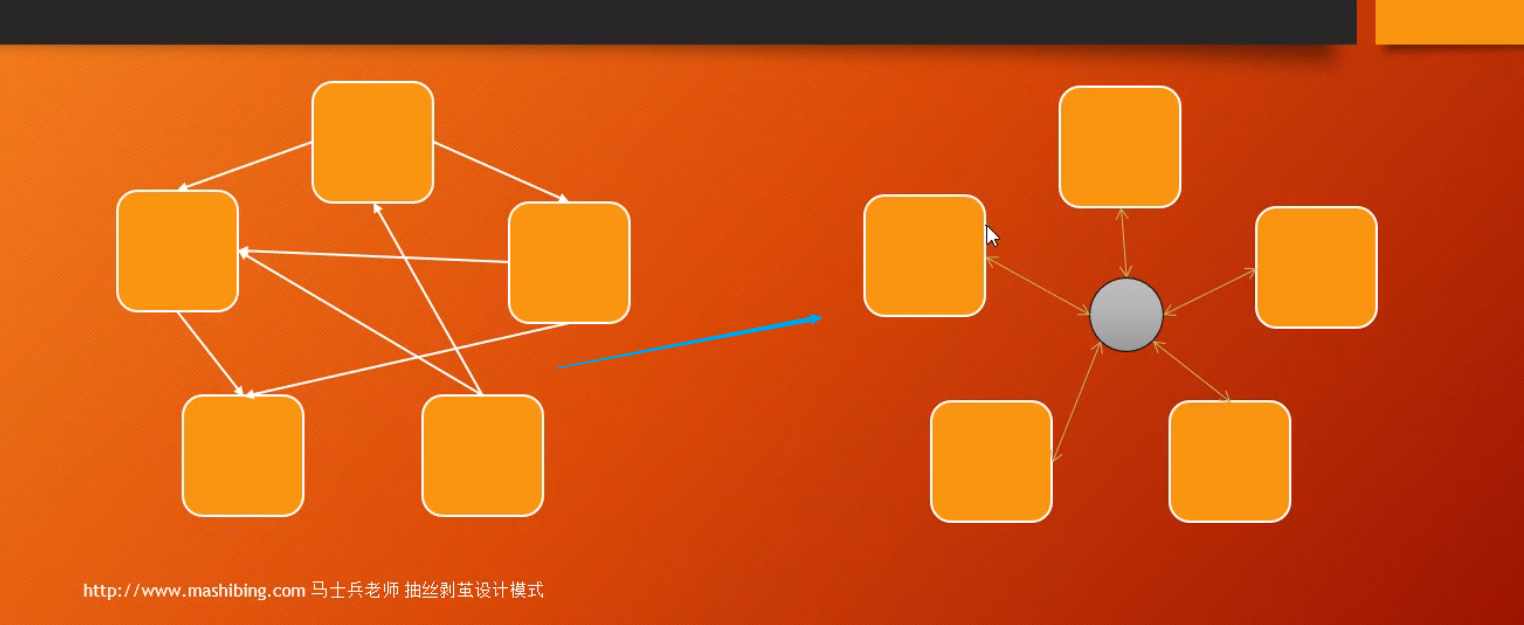
门面模式，通过一个门面，帮助你协调业务关系



## 调停者Mediator（门面内部）

内部协调关系，解耦

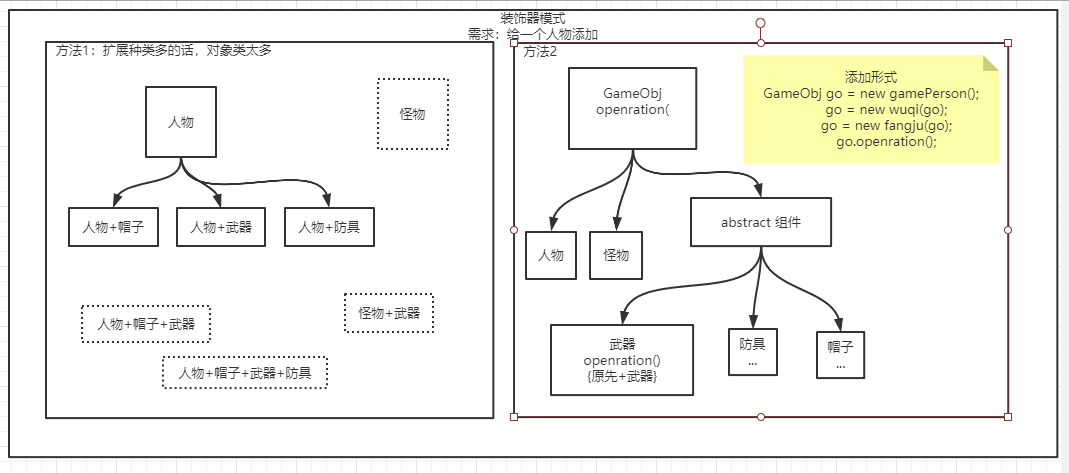
例子：消息中间件



## 装饰器Decorator

实现给一个人物+帽子+武器+防具

兼容怪物+帽子+武器+防具



玩家的例子

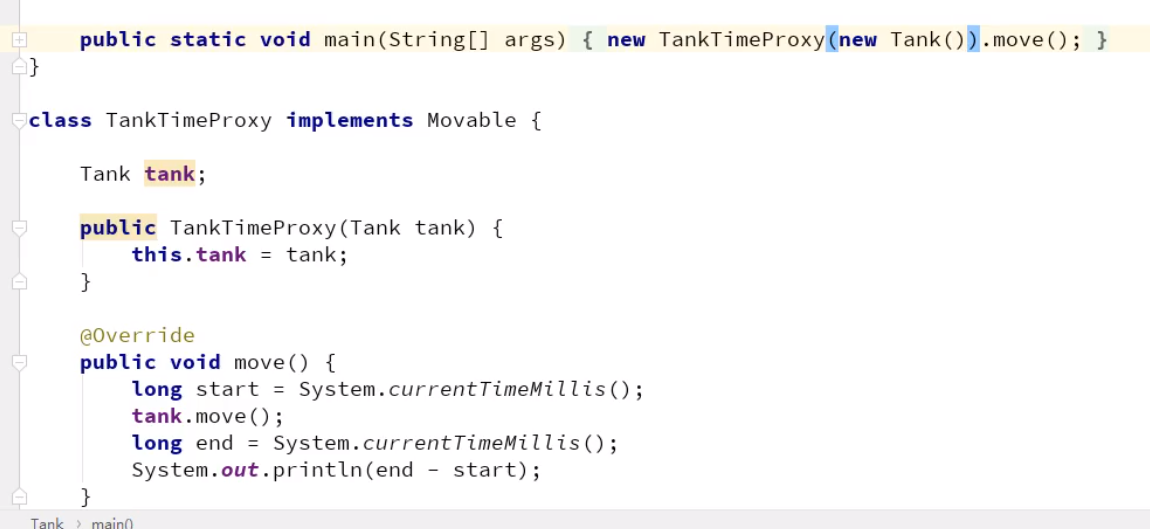
*/\*\*  
 \* 装饰者模式  
 \* 开闭原则  
 \* 给人物添加武器防具等  
 \*/*public class test {  
 public static void main(String[] args) {  
 GameObj go = new gamePerson();  
 go = new wuqi(go);  
 go = new fangju(go);  
 go.openration();  
 }  
}  
  
  
interface GameObj{  
 void openration();  
}  
  
class gamePerson implements GameObj{  
 @Override  
 public void openration() {  
 System.*out*.println("我是人");  
 }  
}  
  
class gameMonster implements GameObj{  
 @Override  
 public void openration() {  
 System.*out*.println("我是怪物");  
 }  
}  
  
abstract class gameDecorator implements GameObj{  
 GameObj go;  
  
 public gameDecorator(GameObj go) {  
 this.go = go;  
 }  
}  
  
class wuqi extends gameDecorator{  
 public wuqi(GameObj go) {  
 super(go);  
 }  
  
 @Override  
 public void openration() {  
 go.openration();  
 System.*out*.println("加上了武器");  
 }  
}  
  
class fangju extends gameDecorator{  
 public fangju(GameObj go) {  
 super(go);  
 }  
  
 @Override  
 public void openration() {  
 go.openration();  
 System.*out*.println("加上了防具");  
 }  
}

## 代理Proxy

记录时间，日志

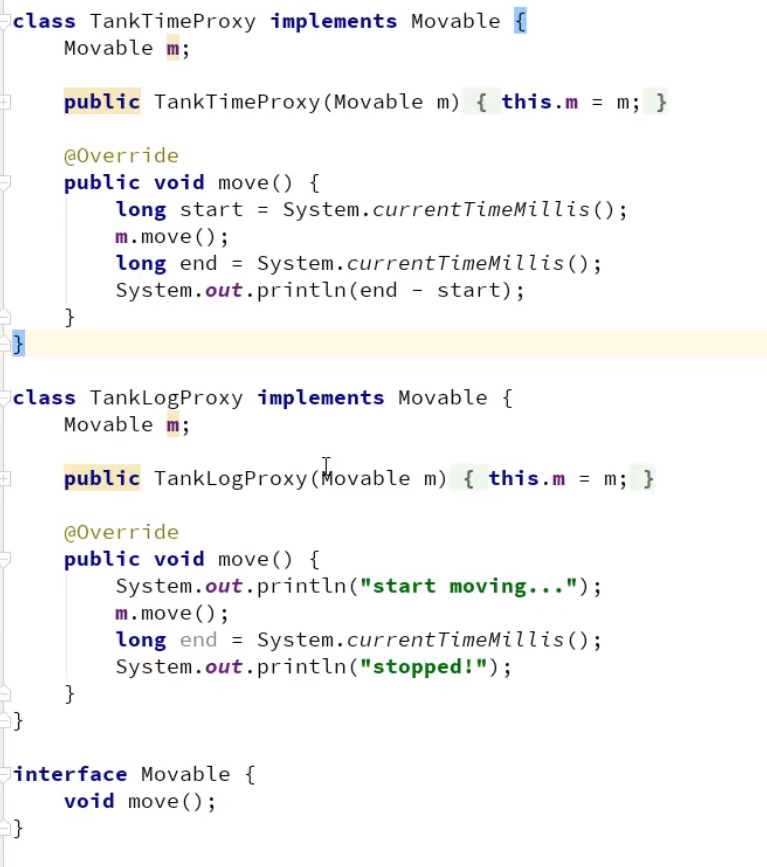
### 静态

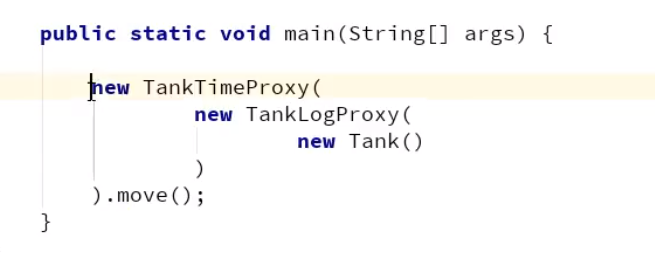
记录时间



记录时间和日志

嵌套





### 动态

## 责任链

## TemplateMethod模板方法

一个方法自动调用了多个方法

具体步骤：

继承并调用父类方法，

同时自动调用了很多方法，

子类可以直接重写改动这些方法，会自动被调用，不需要在调用。

