大话设计模式

# 面向对象

通过规划业务逻辑（包含未来即将使用到的），构建一个可归纳主干，可扩展子业务，可修改主业务和子业务，可灵活删除业务的系统；

脱离具体的业务逻辑，从多种系统使用情况，归纳业务逻辑，根据此抽象出的业务逻辑，创建可归纳此业务逻辑的系统，或是模块；

活字印刷术

提高工作效率，通过重复利用工作成果；将劳动成果分割开来；

1. 可维护：在未来修改内容的时候可以在原有基础上进行修改，达到重复使用部分劳动成果的作用
2. 可复用：将生产的劳动成果重复的使用；
3. 可扩展：当分割后的劳动成果无法满足当前需求的时候，可以添加同类的劳动成果，达到要求；
4. 灵活性好：在根据需求进行使用的时候非常方便，修改简单；

通过封装 继承 多态把程序的耦合度降低；

1. 业务的封装：

将不同的业务封装到不同的类或方法中，而不是混乱的交织在一起；

1.1 良好的封装能够减少耦合

1.2 类内部的实现可以自由修改

1.3类具有清晰的对外接口；

1. 继承

两个对象a b，如果说b是a，那么b可以集成a；

is a的关系，这种关系是不能颠倒的，b除了集成了a，还拥有自己的具体的定义；

如果子类继承父类1.子类拥有父类非private的属性和功能；2.子类具有自己的属性和功能；3.子类对父类的方法的重写；4.构造方法不能被继承，可以被调用；

优点1.公共部分放在子类，实现代码共享；2.使修改和扩展变得轻松；

缺点 1.父类变，子类不得不变；2.继承破坏包装，父类实现细节暴露给子类，增加了耦合性，建立强耦合关系；

1. 多态

表示不同的对象可以执行相同的操作，但要通过他们自己的实现代码来执行；

几点注意：1.子类以父类的身份出现；2.子类在工作的时候以自己的方式实现；3.子类以父类的身份出现时，子类特有的属性和方法不可以使用；4.使用override关键字重写；

多台原理：当方法被调用时，无论对象是否被转换为其父类，都只有位于对象继承链最末端的方法实现会被调用。也就是说虚方法是按照其运行时类型而非编译时类型进行动态绑定调用的。

紧耦合 vs 松耦合

根据需求和业务逻辑，将同类业务适当细分为子业务，变为可扩展，可修改（不影响其他子业务），可变更运算顺序；

对象：对象是一个自包含的实体，用一组可识别的特性和行为来标识；

类：具有相同的属性和功能的对象的抽象的集合；

实例：就是一个真实的对象；

实例化：就是创建对象的过程，用new关键字；

构造方法：对类进行初始化，与类同名，无返回值，在new的时候调用；

Oop :针对对象来编程；

方法重载：

属性与修饰符：

重构：在原有的开放的接口上，进行私有的逻辑的改动，甚至是类的业务逻辑的改动；

抽象类：1.抽象类不能被实例化;2.抽象方法是必须被子类重写的方法;3.如果类中包含抽象方法,那么类就必须定义为抽象类,不论是否还包含其他一般方法;4.抽象类拥有尽可能多的共同的代码,拥有尽可能少的数据;

抽象类在一个以继承关系形成的等级结构里面，树叶节点应当是具体类，而树枝节点均英但是抽象类

接口：接口是把隐式公共方法和属性组合起来，以封装特定功能的一个集合。一旦类实现了接口，类就可以支持接口所指定的所有属性和成员。声明接口在语法上与声明抽象类完全相同，但不允许提供接口中任何成员的执行方式。

抽象类和接口的比较：

形态上区分

抽象类 一些成员实现，接口不行

抽象类的成员可被子类部分实现，接口需要完全实现

一个类只能继承一个抽象类，但接口可以有多个

1. 类是对对象的抽象 ；抽象类是对类的抽象； 接口是对行为的抽象；
2. 如果行为跨越不同类的对象，可使用接口；对于一些相似的类对象，用继承抽象类
3. 从设计角度上讲，抽象类是从子类中发现了公共的东西，泛化出父类，然后子类继承父类，接口是根本不知道子类的存在，方法如何实现还不确认，预先定义；
4. 接口就像比赛中举办方的比赛项目制定，比什么；抽象类就像对参赛者进行分类抽象；

# 简单工厂模式

根据传入的定义好的规则参数，返回不同的对象；对于不同的对象，对外界有相同的操作接口，也可由自己公开的操作方法和属性；

# 策略模式

案例：收银台促销手段的实现；

他定义了算法家族，分别封装起来，让他们之间可以相互替换，次模式让算法的变化，不会影响到使用算法的客户。

将满足某业务的各种操作方式，看作一个需求；用策略模式管理这种需求；

结构图

Context：结合简单工厂模式，使业务模块与调用者减少类之间的耦合；暴露给调用者；

Strategy:抽象类，将各种具体算法或行为抽象出来。

Strategya:某具体算法，

优点

1. 策略模式是一种定义一系列算法的方法，从概念上来讲，所有这些算法完成的都是相同的工作，只是实现不同，它可以以相同的方式，调用所有的算法，减少了各种算法与使用算法类之间的耦合
2. Strategy类层次为context定义了一系列的可重用的算法或行为。继承有助于析取出这些算法的公共功能。
3. 简化测试单元

# 单一职责原则

案例：手机集合了多种功能，摄像不清晰

定义：就一个类而言，应该仅有一个引起他变化的原因；如果能够想到多余一个动机去改变一个类，那么这个类就具有多个职责；

优点：

1. 一个类承担的责任过多，当修改一个职责时，可能会消弱或抑制其他职责的能力。
2. 当发生变化后，设计会遭到意想不到的破坏；
3. 软件设计真正要做的许多内容，就是发现职责并把那些职责相互分离。