# Head First 设计模式

## 主题

### OO原则

- 封装变化
- 多用组合，少用继承
- 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
- 开闭原则-对扩展开放，对修改关闭
- 依赖倒置原则-依赖抽象，不要依赖具体类
- 最少知识原则：只和你的密友对话
- 好莱坞原则：别找我，我会找你
- 单一责任：类应该只有一个改变的理由

### OO基础

- 抽象
- 封装
- 多态
- 继承

### 设计模式要点

- 让设计模式自然而然的出现在设计中，而不是为了使用而使用
- 设计模式并非僵化的教条；可以根据需要采用或者调整
- 总是使用满足需要的最简单解决方案，不管它用不用模式
- 设计模式的分类，可以帮助思考关系
- 目前大多数的模式都是现有模式的变体，而非新的模式

## 策略模式

模拟鸭子的飞行，叫声行为

定义算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法客户

## 观察者模式

天气数据通知天气板块进行更新 Observable

在对象之间定义一对多的依赖，这样一来，当一个对象发生改变，其他的对象都会收到通知，并自动更新

- 观察者模式定义了对象之间一对多的关系
- 主题（被观察者）用一个共同的接口来更新被观察者
- 观察者与可观察者之间用松耦合的方式进行结合，可观察者不知道观察者实现的细节，只知道观察者实现了观察者接口
- 使用此模式，你可以从被观察者处PUSH或者PULL
- 有多个观察者时不可以依赖特定的通知次序
- JAVA有多种观察者模式的实现，包括了通用的java.util.Observable
- 要注意java.util.Observable实现上所带来的一些问题
- Swing大量使用观察者模式，许多GUI的框架也是如此
- 此模式也被应用在许多地方，JavaBeans、RMI

## 装饰者模式

咖啡种类+调料+杯型  IO包

动态地将责任附加到对象上。想要扩展功能，装饰者提供有利于继承的另一种选择

- 继承属于扩展形式之一，但不见得是达到弹性设计的最佳方案
- 在我们的设计中，应该允许行为可以被扩展，而且不需要修改现有的代码
- 组合和委托可用于在运行时加入新的行为
- 除了继承，装饰者模式也可以让我们扩展行为
- 装饰者模式意味着一堆装饰者类，这些类用于包装具体组件
- 装饰者类反映出被装饰的组件类型（事实上，他们拥有着相同的类型，都经过接口或者继承实现）
- 装饰者可以在被装饰者的前/后加上自己的行为，甚至覆盖被装饰者的行为，从而达到特定的目的
- 你可以用无数个装饰者包装你的组件
- 装饰者一般对组件的客户是透明的，除非客户程序依赖于组件的具体类型
- 装饰者模式会导致设计中出现很多的小对象，如果过度使用，会让程序变得过于复杂

## 工厂模式

### 工厂方法模式

Pizza+NYPizza+ChicagoPizza

定义一个创建对象的接口，但由子类决定实例化的类是哪一个。工厂方法让类把实例化方法推迟到子类

### 抽象工厂模式

Pizza+原料组合工厂

提供一个接口，用来创建相关与依赖对象的家族，而不需要明确指定具体类

- 所有工厂都是用来封装对象的创建
- 简单工厂，虽不是设计模式，但不失为一个简单的方法，可以将客户端程序从具体类中解耦
- 工厂方法使用继承，把对象的创建委托给子类，子类实现工厂方法来创建对象
- 抽象方法使用对象组合，对象的创建被实现在工厂接口所暴露的方法中
- 所有工厂方法都通过减少应用程序与具体类的依赖促进松耦合
- 工厂方法允许类将实例化推迟到子类进行
- 抽象工厂所创建的对象家族，而不需要依赖它们的具体类
- 依赖倒置原则，指导我们避免依赖具体类型，而要尽量依赖抽象
- 工厂是很有威力的技巧，使我们针对抽象编程，而不需要针对具体类编程

## 单件模式

巧克力工厂

确保一个类只有一个实例，并提供一个全局访问点

- 单件模式确保程序中一个类只有一个实例
- 单件模式也提供访问这个实例的全局点
- 在JAVA中实现单件模式，需要私有的构造器，一个静态变量和一个静态方法
- 确定在性能和资源方面的限制，然后小心的选择适当的方案来实现单件模式，以解决多线程的问题
  - 懒汉
  - 饿汉
  - 双重检查加锁
- 如果使用多个类加载器，可能导致单件失效而产生多个实例

## 命令模式

风扇命令+灯命令+宏命令+撤销

将请求封装成对象，以便使用不同的请求、队列、或者日志来参数化其他对象。命令模式也支持可撤销的操作。

当需要将发出请求的对象与执行的对象解耦的时候，使用命令模式

- 在被解耦的两个对象之间是通过命令来沟通的，命令对象封装了接收者和一个动作或者一组动作。
- 调用者通过调用命令对象的execute方法发起请求，这会使得接受者的动作被调用
- 命令模式支持撤销，做法是定义undo方法来回到execute被执行前的状态
- 宏命令是命令的延伸，允许调用多个命令。宏命令也支持撤销操作
- 实际操作时，很常见的使用"聪明"命令对象，也就是直接实现了请求，而不是将工作委托给接收者
- 命令也可以用来实现事务与日志系统

## 适配器模式

鸭子与火鸡之间的行为转换

将一个类的接口，转换成客户期望的另一个接口，适配器让原本不兼容的类型可以合作无间

- 当需要使用一个现有的类而其他接口并不符合你的需要时，就使用适配器
- 适配器改变接口以符合客户的期望
- 实现一个适配器的复杂程度与目标接口的大小和复杂程度而定
- 适配器模式有两种形式：对象适配器与类适配器。类适配器需要用到多重继承
- 适配器将一个对象包装起来以改变其接口；装饰者讲一个对象包装起来以增加新的行为和责任，而改观将一群对象包装起来以简化其接口

## 外观者模式

家庭影院开始与关闭

提供了一个统一接口，用来访问子系统的一群接口。外观定义了一个高层接口，让子系统更容易使用。

- 当需要统一一个很大的接口或者一群复杂的接口对象时，使用外观
- 外观将客户从一个复杂的子系统中解耦
- 实现一个外观，需要将子系统组合进外观中，然后将工作委托给子系统执行
- 可以为子系统实现一个以上的外观

## 模版方法模式

茶与咖啡的冲泡步骤模版

在一个方法中定义一个算法的骨架，而将算法的实现延迟到子类中。模版方法使得子类可以在不改变算法结构的情况下，重新定义算法中的某些步骤。

- 模版方法定义了算法的步骤，把这些步骤的实现推迟到子类
- 模版方法为我们提供了一种代码复用的技巧
- 模版方法的抽象类可以定义具体方法，抽象方法和钩子
- 抽象方法由子类实现
- 钩子是一种方法，它在抽象类中不做或者做默认的事情，子类可以选择是否覆盖它
- 为了防止子类改变模版方法中的算法，可以将模版方法声明为final
- 好莱坞原则告诉我们，将决策权放在高层模块中，以便决定如何与何时调用低层模块
- 你将在真实世界中看到许多模版方法的变体
- 策略模式和模版方法模式都封装算法，一个用组合，一个用继承
- 工厂方法是模版方法的特殊版本

## 迭代器模式

菜单迭代打印

提供一个方法顺序访问一个聚合对象的各个元素，而又不暴露其内部的表示

- 迭代器允许访问聚合的元素，而不需要暴露它的内部结构
- 迭代器将遍历聚合的工作封装到一个对象中
- 当使用迭代器时我们依赖聚合提供遍历
- 迭代器提供了一个通用的接口，让我们遍历聚合的项，当我们编码使用聚合的项时，就可以使用多态机制
- 我们应该努力让一个类分配一个责任

## 组合模式

菜单连环嵌套，迭代+筛选

允许将对象组成树形结构来表现"整体/局部"的层次结构。组合能让客户以一致的方式处理个别对象和对象组合

- 组合模式提供一个结构，可同时包含个别对象和对象组合
- 组合模式允许客户对个别对象以及组合对象一视同仁
- 组合结构内的任意对象成为组件，组件可以是组合，也可以是叶节点
- 在实现组合模式时，有许多设计上的折衷。你要根据需要平衡透明性和安全性

## 状态模式

糖果机状态模型

允许对象在内部状态改变时改变它的行为，对象看起来好像修改了它的类。

- 状态模式允许一个对象基于内部状态而拥有不同的行为
- 和程序状态机不同，状态模式用类来代表状态
- Context会将行为委托给当前状态对象
- 通过将每个状态封装进一个类，我们把以后需要做的任何改变局部化了
- 状态模式和策略模式有相同的类图，但是他们的意图不同
- 策略模式通常会用行为或算法来配置Context类
- 状态模式允许Context随着状态的改变而改变行为
- 状态转换可以由State类或Context类控制
- 使用状态模式通常会导致设计中类的数目大量增加
- 状态类可以被多个Context实例共享

## 代理模式

RMI，动态代理

为另一个对象提供一个替身或占位符以控制对这个对象的访问。（远程代理，虚拟代理，保护代理……）

- 代理模式为另一个对象提供代表，以便控制客户对对象的访问，管理访问的方式有多种
- 远程代理管理客户和远程对象之间的交互
- 虚拟代理控制访问实例化开销大的对象
- 保护代理基于调用者控制对对象方法的访问
- 代理模式有许多变体，例如：缓存代理、同步代理，防火墙代理
- 代理在结构上类似装饰者，但是目的不同
- 装饰者模式为对象加上行为，而代理则是控制访问
- Java内置的代理支持，可以根据需要创建动态代理，并将所有调用分配到所选的处理器
- 就和其他的包装者（wrapper）一样，代理会造成设计中类的增加

## 复合模式

动物学家与鸭子

复合模式结合两个或以上的模式，组成一个解决方案，解决一再发生的一般性问题。

- MVC是复合模式，结合了观察者模式、策略模式和组合模式
- 模型使用观察者模式，以便观察者更新，同时保持两者之间解耦
- 控制器是视图的策略，视图可以使用不同的控制器实现，得到不同的行为
- 视图使用组合模式实现用户界面，用户界面通常组合了嵌套的组件，像面板、框架和按钮
- 这些模式携手合作，把MVC模型的三层解耦，这样可以保持设计干净又有弹性
- 适配器模式用来将新的模型适配成已有的视图和控制器
- Model 2是MVC在WEB上的应用
- 在Model 2中，控制器实现成Servlet，而JSP/HTML实现视图