**创建型模式**

**建造者模式**：使用多个简单的对象一步一步构建一个复杂的对象。

与工厂模式的区别：更加专注于零件装配的顺序。

使用场景：一些基本部件不会变，其组合会经常变化。

例如去吃肯德基，汉堡、薯条、饮料、炸鸡翅等是不变的，其组合经常变化，生成所谓的套餐。

将一个复杂对象的构造和它的表示分离，使同样的构建过程可以创建不同的表示。

**原型模式**：实现一个原型接口，该接口用于创建当前对象的克隆，直接创建对象代价较大时，使用这种模式。

可采用对象实现cloneable接口

**结构型模式**

**桥接模式**：毛笔和颜色两种维度，毛笔抽象类，颜色接口，毛笔抽象类中调用颜色接口。

优点：减少类的个数，如三种毛笔，六种颜色需要18实现类，采用桥接只需3+6种。方便扩展。

是实现Java虚拟机和jdbc等驱动的核心模式之一

**适配器模式**：适配器存在于不匹配的二者之间，用于连接二者，将不匹配变的匹配。

解决问题：两个已有接口不兼容问题，有时不能控制其扩展，常用于现有系统与第三方产品功能的集成

如：实现a接口，b接口

实现a接口，属性化b接口

抽象类实现接口，继承抽象类

类适配器、对象适配器、接口适配器

桥接模式和适配器模式用于设计的不同阶段，桥接用于系统的初步设计，对于有两个独立变化维度的类可以将其分为抽象化和实现化两个角色，使他们可分别变化。

初步设计完成之后，当系统与已有类无法协同工作时，可采用适配器模式。

**门面模式(外观模式):**为子系统中的一组接口提供一个一致的界面，使得子系统调用更易使用。

主要解决问题：降低访问复杂系统的内部子系统时的复杂度。

过滤器模式(标准模式):使用不同的标准过滤一组对象，通过逻辑运算以解耦的方式把他们连接起来，结合多个标准来获得单一标准。

**装饰器模式**：创建一个装饰类，包装原有的类，并在保证方法完整性的前提下，提供额外的功能。

解决问题：扩展一个类时常使用继承，但是继承引入静态特征，且随着扩展功能的增多，子类会膨胀。

装饰器是继承的一个很好替代。

**组合模式：**对象中包含对象，通过组合的方式来布局。

如：员工角色分ceo,经理，职员，CEO对象中组装经理对象，经理对象组装职员。遍历ceo可得所有员工。

**享元模式**：尝试重用现有对象，若未找到匹配的对象，则创建新对象。

场景：如Java中String对象，数据库的数据池

实现方法：用hashmap缓存对象

优点：减少对象创建，节省存储空间

**代理模式：**一个类代表另一个类的功能，某些场景下，一个客户不想或者不能直接引用另一个对象，代理对象可以在客户端和目标对象之间起到中介作用。

场景：火车票不一定要在火车站买，也可以在代售点(代理)

远程代理，防火墙代理

区别适配器：适配器模式主要改变所考虑对象的接口，代理模式不改变所代理类的接口

区别装饰器：装饰器是为了增强功能，代理是为了加以控制。

**行为型模式**

**责任链模式**：责任链上的处理者负责处理请求，客户只需要将请求发送到职责链即可，无需关心请求的处理细节和请求的传递。

比如：qq登录、短信登录、账户密码登录，过滤器使用责任链模式找到合适的登录处理器

**命令模式**：请求以命令的形式包裹在对象中，并传给调用对象，调用对象寻找可以处理该对象的合适对象，并传给他执行。

场景：行为请求者与使用者通常是紧耦合的关系，但某些场合需要对行为进行记录、撤销或重做、事物等处理时，不能抵御变化的紧耦合就不合适了。

角色有生成者、命令、接收者

**解释器模式**：给定一个语言后，解释器模式可以定义出其文法的一种表示，并提供一个解释器。客户端可使用这个解释器来解释这个语言中的句子。

**中介者模式**：降低多个对象和类之间的通信复杂性，提供一个中介类，用以处理不同类之间的通信，松耦合，使代码易于维护。

应用案例：MVC框架，其中c层是M层与V层的中介

**备忘录模式：**保存对象的状态，以便在适当的适合的时候恢复对象。

场景：游戏存档、ctrz+z、数据库的事物、浏览器的后退

观察者模式：当对象间存在一对多关系时，使用观察者模式。如，当一个对象被修改时，自动通知依赖它的对象并更新。