一、概述

设计模式是解决问题的方案，学习现有的设计模式可以做到经验复用。

拥有设计模式词汇，在沟通时就能用更少的词汇来讨论，并且不需要了解底层细节。

二、创建型

1. 单例（Singleton）

意图

确保一个类只有一个实例，并提供该实例的全局访问点。

类图

使用一个私有构造函数、一个私有静态变量以及一个公有静态函数来实现。

私有构造函数保证了不能通过构造函数来创建对象实例，只能通过公有静态函数返回唯一的私有静态变量。

2. 简单工厂（Simple Factory）

意图

在创建一个对象时不向客户暴露内部细节，并提供一个创建对象的通用接口。

类图

简单工厂不是设计模式，更像是一种编程习惯。它把实例化的操作单独放到一个类中，这个类就成为简单工厂类，让简单工厂类来决定应该用哪个具体子类来实例化。

3. 工厂方法（Factory Method）

意图

定义了一个创建对象的接口，但由子类决定要实例化哪个类。工厂方法把实例化推迟到子类。

类图

在简单工厂中，创建对象的是另一个类，而在工厂方法中，是由子类来创建对象。

下图中，Factory 有一个 doSomething() 方法，这个方法需要用到一个产品对象，这个产品对象由 factoryMethod() 方法创建。该方法是抽象的，需要由子类去实现。

4. 抽象工厂（Abstract Factory）

意图

提供一个接口，用于创建 相关的对象家族 。

类图

抽象工厂模式创建的是对象家族，也就是很多对象而不是一个对象，并且这些对象是相关的，也就是说必须一起创建出来。而工厂方法模式只是用于创建一个对象，这和抽象工厂模式有很大不同。

抽象工厂模式用到了工厂方法模式来创建单一对象，AbstractFactory 中的 createProductA() 和 createProductB() 方法都是让子类来实现，这两个方法单独来看就是在创建一个对象，这符合工厂方法模式的定义。

至于创建对象的家族这一概念是在 Client 体现，Client 要通过 AbstractFactory 同时调用两个方法来创建出两个对象，在这里这两个对象就有很大的相关性，Client 需要同时创建出这两个对象。

从高层次来看，抽象工厂使用了组合，即 Cilent 组合了 AbstractFactory，而工厂方法模式使用了继承。

5. 生成器（Builder）

意图

封装一个对象的构造过程，并允许按步骤构造。

6. 原型模式（Prototype）

意图

使用原型实例指定要创建对象的类型，通过复制这个原型来创建新对象。

1. 责任链（Chain Of Responsibility）

意图

使多个对象都有机会处理请求，从而避免请求的发送者和接收者之间的耦合关系。将这些对象连成一条链，并沿着这条链发送该请求，直到有一个对象处理它为止。

2. 命令（Command）

意图

将命令封装成对象中，以便使用命令来参数化其它对象，或者将命令对象放入队列中进行排队，或者将命令对象的操作记录到日志中，以及支持可撤销的操作。

3. 解释器（Interpreter）

意图

为语言创建解释器，通常由语言的语法和语法分析来定义。

4. 迭代器（Iterator）

意图

提供一种顺序访问聚合对象元素的方法，并且不暴露聚合对象的内部表示。

5. 中介者（Mediator）

意图

集中相关对象之间复杂的沟通和控制方式。

6. 备忘录（Memento）

意图

在不违反封装的情况下获得对象的内部状态，从而在需要时可以将对象恢复到最初状态。

7. 观察者（Observer）

意图

定义对象之间的一对多依赖，当一个对象状态改变时，它的所有依赖都会收到通知并且自动更新状态。

主题（Subject）是被观察的对象，而其所有依赖者（Observer）称为观察者。

8. 状态（State）

意图

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所属的类。

9. 策略（Strategy）

意图

定义一系列算法，封装每个算法，并使它们可以互换。

策略模式可以让算法独立于使用它的客户端。

10. 模板方法（Template Method）

意图

定义算法框架，并将一些步骤的实现延迟到子类。

通过模板方法，子类可以重新定义算法的某些步骤，而不用改变算法的结构。