# 什么是设计模式

设计模式(Design Pattern)是一套被反复使用、多数人知晓的、经过 分类编目的、代码设计经验的总结，使用设计模式是为了可重用代码、让代码更容易被他人理解并且保证代码可靠性。

设计模式一般包含模式名称、问题、目的、解决方案、效果等组成要素，其中关键要素是模式名称、问题、解决方案和效果

# **设计模式的作用**

* 复用解决方案
* 保证代码的可靠性
* 大多数设计模式还能使软件更容易修改和维护。

# **设计模式的原则**

设计模式有7大原则：

* 单一职责原则 (Single Responsibility Principle, SRP) ：一个类只负责一个功能领域中的相应职责，当然，我们可以进行变形，让方法职责单一；
* 接口隔离原则 (Interface Segregation Principle, ISP) ：使用多个专门的接口，而不使用单一的总接口，说白了，接口应该尽可能的小。
* 依赖倒转原则 (Dependence Inversion Principle, DIP)：抽象不应该依赖于细节，细节应该依赖于抽象 ，强调的是面向接口编程；
* 里氏代换原则 (Liskov Substitution Principle, LSP) ：所有引用基类对象的地方能够透明地使用其子类的对象，LSP和继承的关系非常密切；
* 开闭原则 (Open-Closed Principle, OCP) ： 软件实体应对扩展开放，而对修改关闭。
* 合成复用原则 (Composite Reuse Principle, CRP) ：尽量使用对象组合，而不是继承来达到复用的目的
* 迪米特法则 (Law of Demeter, LoD)：一个软件实体应当尽可能少地与其他实体发生相互作用，其中门面模式就很好的展现了迪米特法则；

# **设计模式分类**

## 创建型模式（6个）

* 简单工厂模式
* 工厂方法模式
* 抽象工厂模式
* 单例模式
* 原型模式
* 建造者模式

## 结构型模式（7个）

* 适配器模式
* 桥接模式
* 组合模式
* 装饰模式
* 外观模式
* 享元模式
* 代理模式

## 行为型模式（11个）

* 责任链模式
* 命令模式
* 解释器模式
* 迭代器模式
* 中介者模式
* 备忘录模式
* 观察者模式
* 状态模式
* 策略模式
* 模板方法模式
* 访问者模式

# **简单工厂模式**

## 定义

简单工厂模式(Simple Factory Pattern)：定义一个工厂类，它可以根据参数的不同返回不同类的

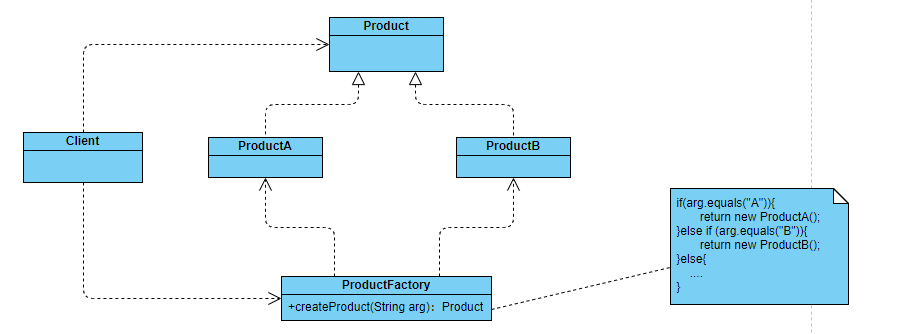
实例，被创建的实例通常都具有共同的父类。因为在简单工厂模式中用于创建实例的方法是

静态(static)方法，因此简单工厂模式又被称为静态工厂方法(Static Factory Method)模式。

## 类图

### 标准类图：

为了让工厂能够统一生产对象，这些对象通常会有一个公共的父类，这个类多半被定义成为抽象类或者是接口（当然这个不是必须的），如下Product，然后让各个具体的类去具体实现或者继承Product。



### 简化类图

为了简化简单工厂，可以将抽象的产品类和工厂类合并在一起，形成如下比较简单的方式，由于工厂需要被实例化，否则不能使用，所以Product不再是抽象类或者接口。



## 总结

### 优点

* 工厂类包含必要的判断逻辑，可以决定在什么时候创建哪一个产品类的实例，客户端可以免除直接创建产品对象的职责，而仅仅“消费”产品，简单工厂模式实现了对象创建和使用的分离。
* 客户端无须知道所创建的具体产品类的类名，只需要知道具体产品类所对应的参数即可，对于一些复杂的类名，通过简单工厂模式可以在一定程度减少使用者的记忆量。
* 可以通过配置文件参数的方式，在不修改代码的情况下传入arg，从而灵活的构建出Product对象；

### 缺点

* 由于工厂类集中了所有产品的创建逻辑，职责过重，一旦不能正常工作，整个系统都要受到影响。
* 系统扩展困难，一旦添加新产品就不得不修改工厂逻辑，在产品类型较多时，有可能造成工厂逻辑过于复杂，不利于系统的扩展和维护。
* 简单工厂模式由于使用了静态工厂方法，造成工厂角色无法形成基于继承的等级结构。
* 对于工厂类而已，每增加一种产品，就需要更改逻辑违背了开闭原则

### 适用场景

* 当需要根据不同的参数创建不同的对象，且他们有共同的抽象方法，只是具体表现行为不同的时候；
* 创建的对象相对较少，这样不会让工厂方法业务过于复杂；

## 开源项目案例

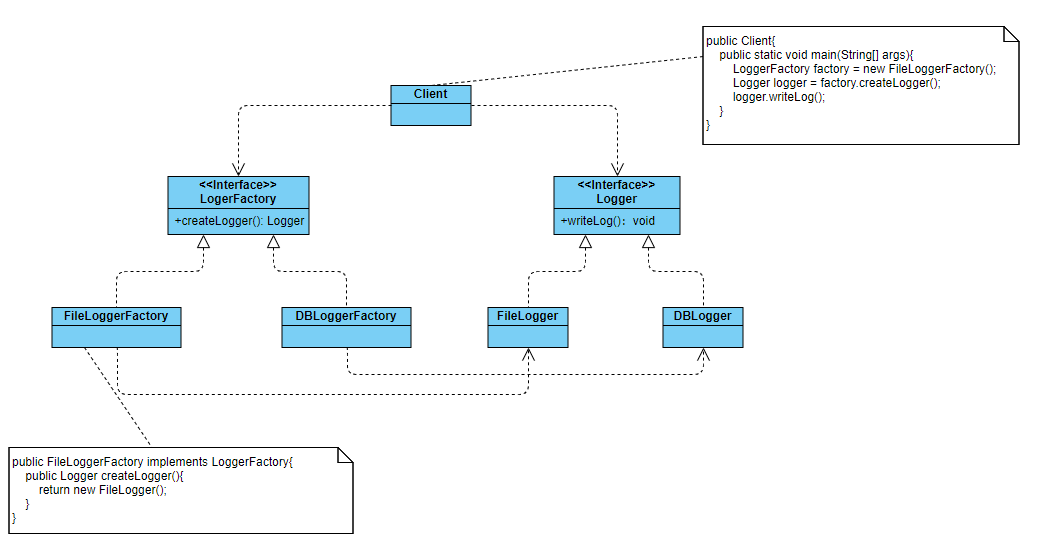
# **工厂方法模式**

## 定义

工厂方法模式(Factory Method Pattern)：定义一个用于创建对象的接口，让子类决定将哪一个 类实例化。工厂方法模式让一个类的实例化延迟到其子类。工厂方法模式又简称为工厂模式 (Factory Pattern)，又可称作虚拟构造器模式(Virtual Constructor Pattern)或多态工厂模式 (Polymorphic Factory Pattern)

工厂方法模式提供一个抽象工厂接口来声明抽象工厂方法，而由其子类来具体实现工厂方法，创建具体的产品对象。

## 类图



## 总结

### 优点

* 客户端只需要关心工厂类抽象和产品抽象，不需要关心具体的实现细节；
* 当有新的产品加入的时候，无需修改抽象工厂类和抽象产品类以及客户端，扩展性强；

### 缺点

* 每新增加一种产品，就需要新增一个工厂类，会导致类的数量很多；

### 适用场景

同一个抽象的实现不是很多，且相对固定，对于不同的场景，可能选择不同的实现的时候，比较适合使用工厂方法模式；

## 开源项目案例

# **抽象工厂模式**

## 定义

抽象工厂模式(Abstract Factory Pattern)：提供一个创建一系列相关或相互依赖对象的接口，而

无须指定它们具体的类。抽象工厂模式又称为Kit模式，

其实说白了，就是对工厂方法模式的一种优化，可以减少工厂类的数据；

同一个工厂类创建出来的对象被称为同一个产品族的；

## 类图

## 总结

### 优点

* 抽象工厂模式隔离了具体类的生成，使得客户并不需要知道什么被创建；
* 当新增同一个产品族的时候，无需更改原有抽象；

### 缺点

* 增加新的产品等级结构麻烦，需要对原有系统进行较大的修改，甚至需要修改抽象层代码

### 适用场景

* 当系统有类似于产品族结构的时候，上面类图所示的结构，有多个产品族，每个产品族有多个产品；
* 产品类型相对固定的时候

## 开源项目案例

# **单例模式**

## 定义

单例模式(Singleton Pattern)：确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类称为单例类，它提供全局访问的方法。

单例模式有三个要点：一是某个类只能有一个实例；二是它必须自行创建这个实例；三是它 必须自行向整个系统提供这个实例。

## 类图

## 核心代码

### 饿汉式

|  |
| --- |
| public class Singleton{  private static Singleton instance = new Singleton();  private Singleton(){}  public static Singleton getInstance(){  return instance;  }  } |

这种方式是上来就直接创建，然而，如果这个类用不到的话，那么这个创建是没有意义的；

### 懒汉式

|  |
| --- |
| public class Singleton{  private static Singleton instance = null;  private Singleton(){}  public synchronized static Singleton getInstance(){  if(instance == null){  instance = new Singleton();  }  return instance;  }  } |

把对象的创建延迟到第一次使用的时候，需要加锁，否则，会存在多线程同时创建，产生多个对象；

### 双重锁校验

### 静态内部类实现

### 枚举实现

## 总结

### 优点

* 单例模式提供了对唯一实例的受控访问。因为单例类封装了它的唯一实例，所以它可以严格控制客户怎样以及何时访问它。
* 由于在系统内存中只存在一个对象，因此可以节约系统资源；

### 缺点

* 由于单例模式中没有抽象层，因此单例类的扩展有很大的困难
* 单例类的职责过重，在一定程度上违背了“单一职责原则”。因为单例类既充当了工厂角色，提供了工厂方法，同时又充当了产品角色

### 适用场景

* 系统只需要一个实例对象
* 客户调用类的单个实例只允许使用一个公共访问点

## 开源项目案例