一、什么是简单工厂模式

简单工厂模式又称为静态工厂模式，**实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（这些产品类继承自一个父类或接口）的实例。**简单工厂模式的创建目标，所有创建的对象都是充当这个角色的某个具体类的实例。

其实就是将一个具体类的实例化交给一个静态工厂方法来执行，它不属于GOF的23种设计模式，但现实中却经常会用到，而且思想也非常简单。

二、简单工厂模式的结构

简单工厂模式包含如下角色：

Factory：工厂角色

Product：抽象产品角色

ConcreteProduct：具体产品角色

工厂角色 (Creator)	**是简单工厂模式的核心，它负责实现创建所有具体产品类的实例。 工厂类可以被外界直接调用，创建所需的产品对象。**
抽象产品角色 (Product)	**是所有具体产品角色的父类，它负责描述所有实例所共有的公共接口。**
具体产品角色 (Concrete Product)	**继承自抽象产品角色，一般为多个，是简单工厂模式的创建目标。 工厂类返回的都是该角色的某一具体产品。**

三、简单工厂模式的应用场景

1.前几天苹果公司刚发布IPhone Xs和iPhone XR，那么问题来了，苹果公司的代工厂到底生产多少种尺寸的手机呢?

由工厂决定生产哪种型号的的手机，苹果公司的工厂就是一个工厂类，是简单工厂模式的核心类。

iPhoneX、iPhoneXs、iphoneXr都是苹果手机，只是型号不同。苹果手机类满足抽象的定义，各个型号的手机类是其具体实现。

2.考虑一个简单的软件应用场景，一个软件系统可以提供多个外观不同的按钮（如圆形按钮、矩形按钮、菱形按钮等），这些按钮都源自同一个基类，不过在继承基类后不同的子类修改了部分属性从而使得它们可以呈现不同的外观，**如果我们希望在使用这些按钮时，不需要知道这些具体按钮类的名字，只需要知道表示该按钮类的一个参数，并提供一个调用**

方便的方法，把该参数传入方法即可返回一个相应的按钮对象，此时，就可以使用简单工厂模式。

在以下情况下可以使用简单工厂模式：
工厂类负责创建的对象比较少：由于创建的对象较少，不会造成工厂方法中的业务逻辑太过复杂。
客户端只知道传入工厂类的参数，对于如何创建对象不关心：客户端既不需要关心创建细节，甚至连类名都不需要记住，只需要知道类型所对应的参数。

四、简单工厂模式和工厂方法模式区别

简单工厂模式：
（1）工厂类负责创建的对象比较少，由于创建的对象较少，不会造成工厂方法中的业务逻辑太过复杂。
（2）客户端只知道传入工厂类的参数，对于如何创建对象并不关心。

工厂方法模式：
（1）客户端不知道它所需要的对象的类。
（2）抽象工厂类通过其子类来指定创建哪个对象。

五、简单工厂模式和策略模式的异同

策略模式和简单工厂模式看起来非常相似，都是通过多态来实现不同子类的选取，这种思想应该是从程序的整体来看得出的。

如果从使用这两种模式的角度来看的话，我们会发现在简单工厂模式中我们只需要传递相应的条件就能得到想要的一个对象，然后通过这个对象实现算法的操作。

而策略模式，使用时必须首先创建一个想使用的类对象，然后将该对象作为参数传递进去，通过该对象调用不同的算法。

在简单工厂模式中实现了通过条件选取一个类去实例化对象，策略模式则将选取相应对象的工作交给模式的使用者，它本身不去做选取工作。

结合下面的代码和下面的释义不难看出，其实两个的差别很微妙,Factory是直接创建具体的对象并用该对象去执行相应的动作，而Context将这个操作交给了Context类，没有创建具体的对象，实现的代码的进一步封装，客户端代码并不需要知道具体的实现过程。

六、简单工厂模式的优缺点

优点：

工厂类是整个模式的关键.包含了必要的逻辑判断,根据外界给定的信息,决定究竟应该创建哪个具体类的对象.

通过使用工厂类,外界可以从直接创建具体产品对象的尴尬局面摆脱出来,仅仅需要负责“消费”对象就可以了。

而不必管这些对象究竟如何创建及如何组织的．明确了各自的职责和权利，有利于整个软件体系结构的优化。

缺点：

由于工厂类集中了所有实例的创建逻辑，违反了开闭原则，将全部创建逻辑集中到了一个工厂类中；

它所能创建的类只能是事先考虑到的，如果需要添加新的类，则就需要改变工厂类了。

当系统中的具体产品类不断增多时候，可能会出现要求工厂类根据不同条件创建不同实例的需求.

这种对条件的判断和对具体产品类型的判断交错在一起，很难避免模块功能的蔓延，对系统的维护和扩展非常不利；

开闭原则定义：一个软件实体如类、模块和函数应该对扩展开放，对修改关闭。

开放-封闭原则的意思就是说，你设计的时候，时刻要考虑，尽量让这个类是足够好，写好了就不要去修改了，如果新需求来，我们增加一些类就完事了，原来的代码能不动则不动。这个原则有两个特性，一个是说“对于扩展是开放的”，另一个是说“对于更改是封闭的”。面对需求，对程序的改动是通过增加新代码进行的，而不是更改现有的代码。这就是“开放-封闭原则”的精神所在。

七、简单工厂模式的实现

首先创建一个"苹果手机"类，定义一个获取手机尺寸的方法

```
public interface Iphone {
   public void getSize();
}
```

苹果手机不同型号的"手机类"。

```
public class IPhoneX implements Iphone{
    @Override
    public void getSize() {
        System.out.println("iphoneX 屏幕尺寸：" + 3.5 + " 英寸");
    }
}

public class IPhoneXs implements Iphone{
    @Override
    public void getSize() {
        System.out.println("iphoneXs 屏幕尺寸：" + 4.5 + " 英寸");
    }
}
```

建立一个"工厂类"生产不同型号的"手机对象"。

```
public class SimpleFactory {
    // 根据不同参数生成不同实例对象。
    public static Iphone create(String model) {
        Iphone phone = null;
        switch (model) {
            case "iphoneX" :
                phone = new IPhoneX();
                break;
            case "iphoneXs" :
                phone = new IPhoneXs();
                break;
            default:
                break;
        }
        return phone;
    }
}
```

最后是客户端测试类

```
public class Client {
    public static void main(String[] args) {
        Iphone iphoneX = SimpleFactory.create("iphoneX");
        iphoneX.getSize();
        Iphone iphoneXs = SimpleFactory.create("iphoneXs");
        iphoneXs.getSize();
    }
}
```

八、总结

创建型模式对类的实例化过程进行了抽象，能够将对象的创建与对象的使用过程分离。
简单工厂模式又称为静态工厂方法模式，它属于类创建型模式。在简单工厂模式中，可以根据参数的不同返回不同类的实例。

简单工厂模式专门定义一个类来负责创建其他类的实例，被创建的实例通常都具有共同的父类。
简单工厂模式包含三个角色：工厂角色负责实现创建所有实例的内部逻辑；抽象产品角色是所创建的所有对象的父类，负责描述所有实例所共有的公共接口；具体产品角色是创建目标，所有创建的对象都充当这个角色的某个具体类的实例。
简单工厂模式的要点在于：当你需要什么，只需要传入一个正确的参数，就可以获取你所需要的对象，而无须知道其创建细节。
简单工厂模式最大的优点在于实现对象的创建和对象的使用分离，将对象的创建交给专门的工厂类负责，但是其最大的缺点在于工厂类不够灵活，增加新的具体产品需要修改工厂类的判断逻辑代码，而且产品较多时，工厂方法代码将会非常复杂。
简单工厂模式适用情况包括：工厂类负责创建的对象比较少；客户端只知道传入工厂类的参数，对于如何创建对象不关心。