# 简单工厂模式简介

## 概述

1. 简单工厂模式并属于23种设计模式之一，但它通常作为学习其他工厂模式（工厂方法模式、抽象工厂模式））的基础。
2. 简单工厂模式的定义如下：

为抽象类或接口定义一个工厂类，在工厂类中定义一个静态方法，这个方法最少接收一个参数，在方法的内部会根据传入参数的不同返回不同子类的实例，这些实例通常具有共同的父类。

1. 简单工厂模式的要点在于当用户需要什么时，只需要传入一个正确的参数就可以获取所需要的对象，而不需知道其创建细节。

## 结构



简单工厂模式工厂角色、抽象产品角色、具体产品角色3个角色组成。

1. Factory（工厂角色）：工厂角色指的是工厂类，它是简单工厂模式的核心，负责实现创建所有产品实例的内部逻辑；通常在工厂类中定义一个静态方法，该方法的返回值是抽象产品角色，该方法最少接收一个参数，在方法内部根据这个参数的不同返回不同的子类实例。
2. Product（抽象产品角色）：抽象产品角色是一个抽象类或接口，在工厂类中静态方法的返回值就是此类型，在工厂类静态方法中，根据传入的参数不同，返回该抽象产品的子类实例。
3. ConcreteProduct（具体产品角色）：具体产品角色是简单工厂模式的创建目标，每一个具体产品角色都继承或实现了抽象产品角色。

## 技巧

1. 如果一个抽象类或接口有很多的子类，可以使用简单工厂模式；如果以抽象类或接口并没有很多子类，内部简单，也无需为其提供工厂类。
2. 如果多个子类拥有共同的代码，则可以将这些公共代码移至到抽象类中，对于不同的，则在抽象类中定义抽象方法，让子类实现；而如果这些子类没有共同的代码，则可以将其父定义为一个接口，这些子类实现接口，覆写抽象方法就可以了。

# 实现与简化

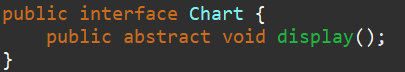
## 实例

实例说明：某软件公司基于Java语言开发一套图表库，该图表库（Chart）可以为应用系统提供多种不同外观的图表，例如柱状图（HistogramChart）、饼状图（PieChart）、折现图（LineChart）等。该软件公司图标库设计人员希望为应用系统开发人员提供一套灵活易用的图表库，通过设置不同的参数即可得到不同类型的图表。

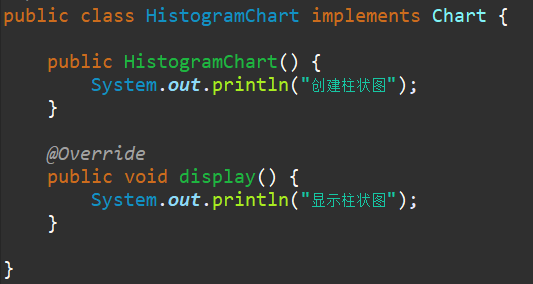


使用简单工厂模式实现：

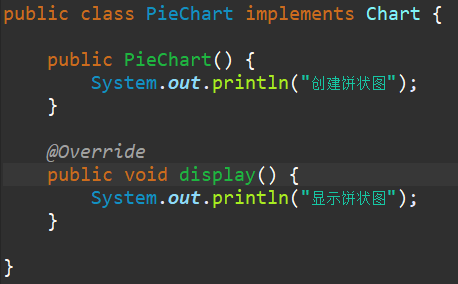
1. Chart：图表接口，充当抽象产品类。



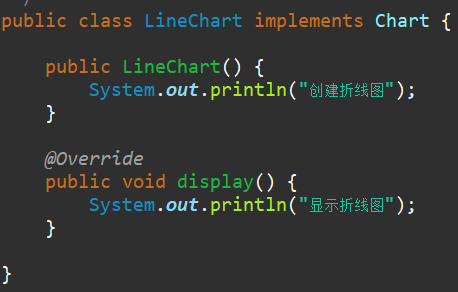
1. HistogramChart：柱状图类，充当具体产品类（也就是抽象产品类的子类）



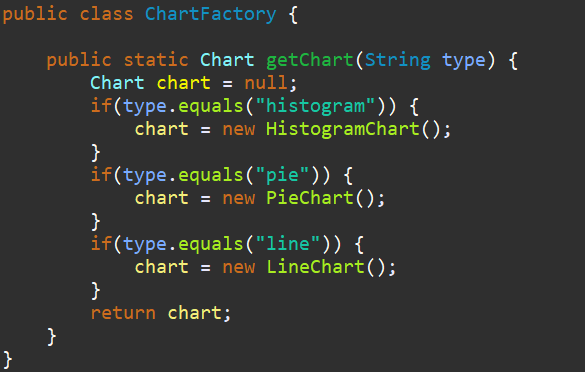
1. PieChart：饼状图类，充当具体产品类（抽象产品类的子类）



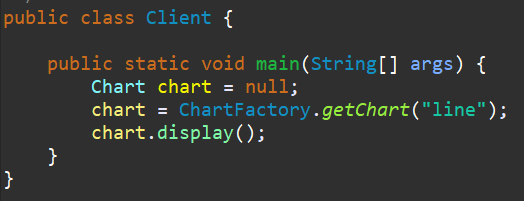
1. LineChart：折线图类，充当具体产品类（抽象产品类的子类）



1. ChartFactory：图表工厂类，充当工厂类，对应着一个抽象产品类



1. 编写客户端测试类，进行测试：



## 简化

有时候，为了简化简单工厂模式，可以将抽象产品类和工厂类合并，将工厂类中的静态工厂方法移至抽象产品类中，而不再单独为其定义一个工厂类。

# 优缺点与使用环境

## 优点

1. 工厂类中的静态方法中包含了必要的逻辑判断，可以决定在什么时候创建一个插件类的实例，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，实现了对象创建和使用的分离。
2. 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，对于一些复杂的类型，通过简单工厂模式可以在一定程度上减少使用者的记忆量。

## 缺点

1. 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。
2. 使用简单工厂模式势必会增加系统中类的个数（因为需要定义工厂类），增加了系统的复杂度和理解难度。
3. 如果抽象产品类添加了新产品（定义了新子类）就需要修改工厂类中静态方法中的逻辑代码，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。
4. 简单工厂模式由于使用了静态工厂方法，造成工厂类无法形成基于继承的等级结构。（有待思考）

## 适用环境

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，此时也可以使用简单工厂模式。