

Object oriented Analysis &Design

面向对象分析与设计

Lecture_15 设计模式引子

主讲: 陈小红

日期:

Architectural style and pattern



图片上传于 [www.myliving.cn 租售情报网](#)

- <http://159.226.47.103/WebAA3/start.html> invalid now
- http://www.sohu.com/a/216457216_100017881
- [http://haokan.baidu.com/v?
pd=wisenatural&vid=14765784896633916505](http://haokan.baidu.com/v?pd=wisenatural&vid=14765784896633916505)

GRASP Patterns

GRASP patterns describe fundamental principles of object design and responsibility assignment. The following is a list of GRASP patterns:

- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

- Polymorphism
- Indirection
- Pure Fabrication
- Protected Variations

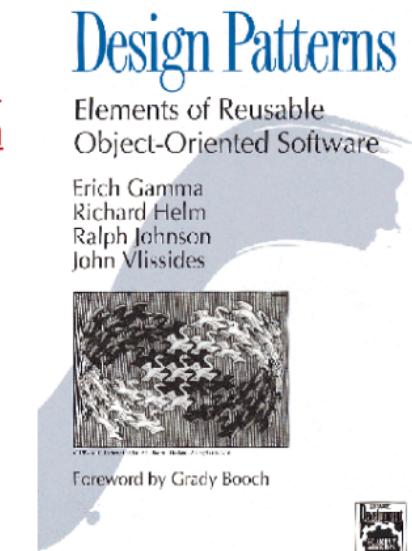


Gang of Four (GoF)

Design Patterns – Elements of Reusable Object-Oriented Software by
Erich Gamma, Richard Helm
Ralph Johnson & John Vlissides

Addison-Wesley, 1995.
(As CD, 1998)

First systematic software pattern description.



ADDISON WESLEY PROFESSIONAL COMPUTING SERIES



Design Patterns GoF Book

Make Objects Object Working Objects Talking to Objects

<u>Purpose</u>				
	<u>Creation</u>	<u>Structure</u>	<u>Behavior</u>	
<u>Class</u>	<u>Factory Method</u>	<u>Adapter</u>	<u>Interpreter</u> <u>Template</u>	
<u>Scope</u>	<u>Object</u>	<u>Abstract Factory</u> <u>Builder</u> <u>Prototype</u> <u>Singleton</u>	<u>Adapter</u> <u>Bridge</u> <u>Composite</u> <u>Decorator</u> <u>Façade</u> <u>Flyweight</u> <u>Proxy</u>	<u>Command</u> <u>Iterator</u> <u>Mediator</u> <u>Memento</u> <u>Observer</u> <u>State</u> <u>Strategy</u> <u>Visitor</u>

面试受挫

- 小菜今年计算机专业大四了，学了不少软件开发方面的东西，也学着编了些小程序，踌躇满志，一心要找一个好单位。当投递了无数份简历后，终于收到了一个单位的面试通知，小菜欣喜若狂。
- 到了人家单位，前台小姐给了他一份题目，上面写着：“请适用C++、Java、C#或VB.NET任意一种面向对象语言实现一个计算器控制台程序，要求输入两个数和运算符号，得到结果。”



```
class Program
{
    static void Main(string[] args)
    {
        Console.Write("请输入数字 A: ");
        string A = Console.ReadLine();
        Console.Write("请选择运算符号(+、 -、 *、 /): ");
        string B = Console.ReadLine();
        Console.Write("请输入数字 B: ");
        string C = Console.ReadLine();
        string D = "";

        if (B == "+")
            D = Convert.ToString(Convert.ToDouble(A) + Convert.ToDouble(C));
        if (B == "-")
            D = Convert.ToString(Convert.ToDouble(A) - Convert.ToDouble(C));
        if (B == "*")
            D = Convert.ToString(Convert.ToDouble(A) * Convert.ToDouble(C));
        if (B == "/")
            D = Convert.ToString(Convert.ToDouble(A) / Convert.ToDouble(C));

        Console.WriteLine("结果是: " + D);
    }
}
```

有什么问题？

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("请输入数字 A: ");
        string A = Console.ReadLine();
        Console.WriteLine("请选择运算符号(+、-、*、/): ");
        string B = Console.ReadLine();
        Console.WriteLine("请输入数字 B: ");
        string C = Console.ReadLine();
        string D = "";
        if (B == "+")
            D = Convert.ToString(Convert.ToDouble(A) + Convert.ToDouble(C));
        if (B == "-")
            D = Convert.ToString(Convert.ToDouble(A) - Convert.ToDouble(C));
        if (B == "*")
            D = Convert.ToString(Convert.ToDouble(A) * Convert.ToDouble(C));
        if (B == "/")
            D = Convert.ToString(Convert.ToDouble(A) / Convert.ToDouble(C));
        Console.WriteLine("结果是: " + D);
    }
}
```

这样命名是非常不规范的

判断分支，你这样的写法，意味着每个条件都要做判断，等于计算机做了三次无用功

如果除数时，客户输入了 0 怎么办，如果用户输入的是字符符号而不是数字怎么办



```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.Write("请输入数字 A: ");
            string strNumberA = Console.ReadLine();
            Console.Write("请选择运算符号(+、-、*、/): ");
            string strOperate = Console.ReadLine();
            Console.Write("请输入数字 B: ");
            string strNumberB = Console.ReadLine();
            string strResult = "";
            switch (strOperate)
            {
                case "+":
                    strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                        + Convert.ToDouble(strNumberB));
                    break;
                case "-":
                    strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                        - Convert.ToDouble(strNumberB));
                    break;
                case "*":
                    strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                        * Convert.ToDouble(strNumberB));
                    break;
                case "/":
                    if (strNumberB != "0")
                        strResult = Convert.ToString(Convert.ToDouble(strNumberA)
                            / Convert.ToDouble(strNumberB));
                    else
                        strResult = "除数不能为 0";
                    break;
                }
            Console.WriteLine("结果是: " + strResult);
            Console.ReadLine();
        }
        catch (Exception ex)
        {
            Console.WriteLine("您的输入有错: " + ex.Message);
        }
    }
}
```

这样的代码是否符合出题人的意图？

- 面向对象！！！

代码无错就是优?

计算机的思维方式

- 满足了当前需求，但是：

程序容易维护吗？

程序容易扩展吗？

程序容易复用吗？

• 活字印刷术，面向对象

喝酒唱歌人生真爽



短歌行



喝酒唱歌人生真爽

对酒当歌人生真爽

对酒当歌人生几何



喝

酒

唱

歌

人

生

真

爽

对

酒

当

歌

人

生

几

何

要改, 只需要改要改之字, 可维护



喝

酒

唱

歌

人

生

真

爽

对

酒

当

歌

人

生

几

何

这些字并非用完这次就无用，完全可以重复使用，可复用



喝

酒

唱

歌

人

生

真

爽

对

酒

当

歌

人

生

几

何

此诗若要加字，只需另刻字加入即可，可扩展
字的排列其实可能说竖排可能说横排，只需将字一动就可，灵活性好



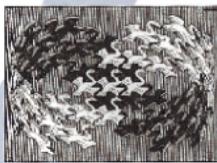
SOFTWARE ENGINEERING INSTITUTE
華東師範大學軟件學院

设计模式

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



Foreword by Grady Booch



ADDISON WESLEY PROFESSIONAL COMPUTING SERIES

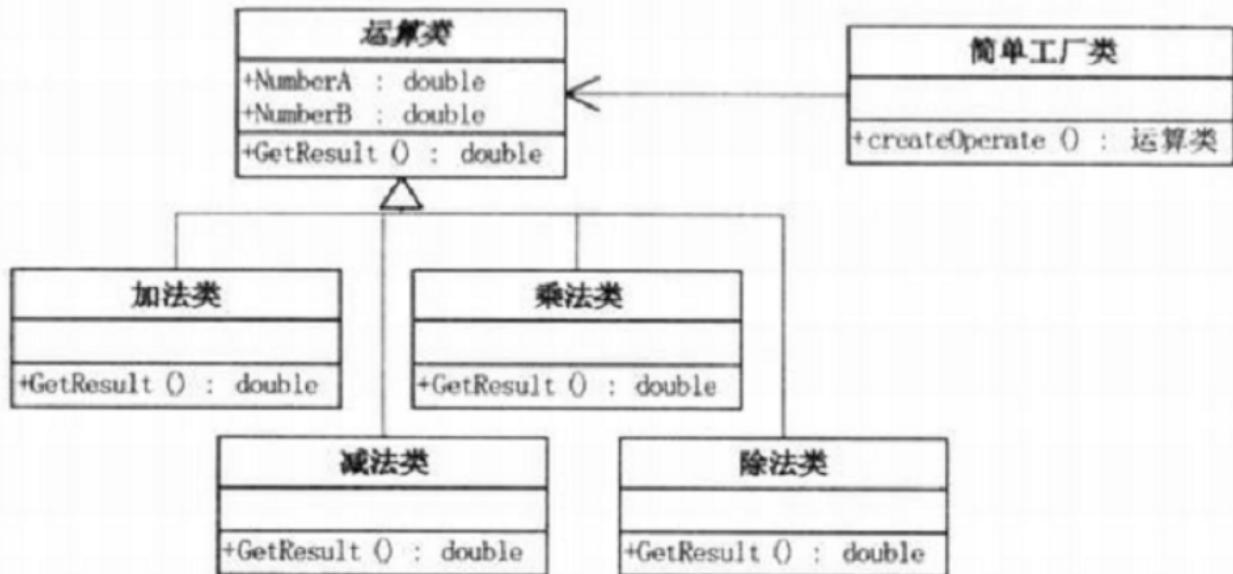
- 为手机做一个计算器的设计，请画出主要的类。
- 如果windows也想用呢？
- 业务逻辑和界面逻辑分开（业务的封装）

如果

- 增加开根号运算?

- 谁来实例化这些类呢？

- 这就是简单工厂模式



代码

```
public class Operation
{
    private double _numberA = 0;
    private double _numberB = 0;

    public double NumberA
    {
        get { return _numberA; }
        set { _numberA = value; }
    }

    public double NumberB
    {
        get { return _numberB; }
        set { _numberB = value; }
    }

    public virtual double GetResult()
    {
        double result = 0;
        return result;
    }
}
```

```
class OperationAdd : Operation
{
    public override double GetResult()
    {
        double result = 0;
        result = NumberA + NumberB;
        return result;
    }
}

class OperationSub : Operation
{
    public override double GetResult()
    {
        double result = 0;
        result = NumberA - NumberB;
        return result;
    }
}
```

加法类，继承运算类

减法类，继承运算类

```
class OperationMul : Operation
{
    public override double GetResult()
    {
        double result = 0;
        result = NumberA * NumberB;
        return result;
    }
}

class OperationDiv : Operation
{
    public override double GetResult()
    {
        double result = 0;
        if (NumberB==0)
            throw new Exception("除数不能为 0。");
        result = NumberA / NumberB;
        return result;
    }
}
```

乘法类，继承运算类

除法类，继承运算类



```
public class OperationFactory
{
    public static Operation createOperate(string operate)
    {
        Operation oper = null;
        switch (operate)
        {
            case "+":
                oper = new OperationAdd();
                break;
            case "-":
                oper = new OperationSub();
                break;
            case "*":
                oper = new OperationMul();
                break;
            case "/":
                oper = new OperationDiv();
                break;
        }
        return oper;
    }
}
```

客户端代码

```
Operation oper;
oper = OperationFactory.createOperate("+");
oper.NumberA = 1;
oper.NumberB = 2;
double result = oper.GetResult();
```

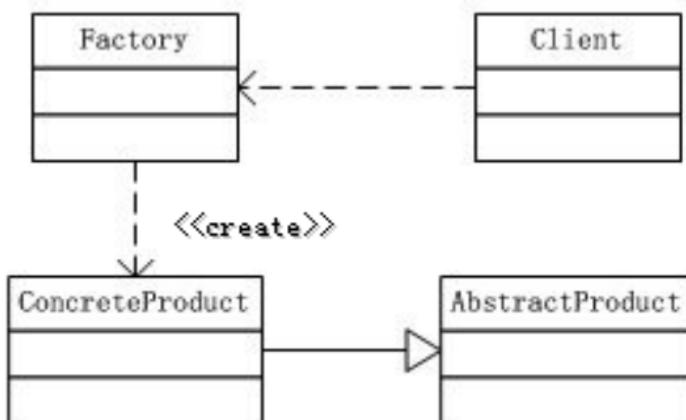
软件的设计原则-开闭原则(Open-Closed Principle, OCP)

Software Entities should be open for extension, but closed for modification.

- --1988, Bertrand Meyer 《Object Oriented Software Construction》
- 通俗理解：软件系统中包含的各种组件，例如模块 Modules、类以及功能等，应该在不修改现有代码的基础上，引入新功能。

简单工厂模式

意图：简单工厂模式根据提供给它的数据，返回几个可能类中的其中一个类的实例。



UML类图

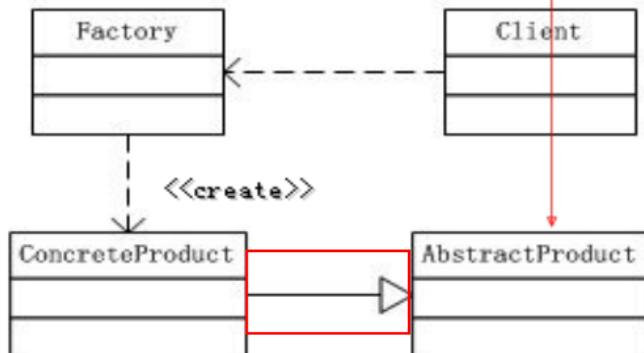
简单工厂模式的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（这些产品类继承自一个父类或接口）的实例

模式参与者

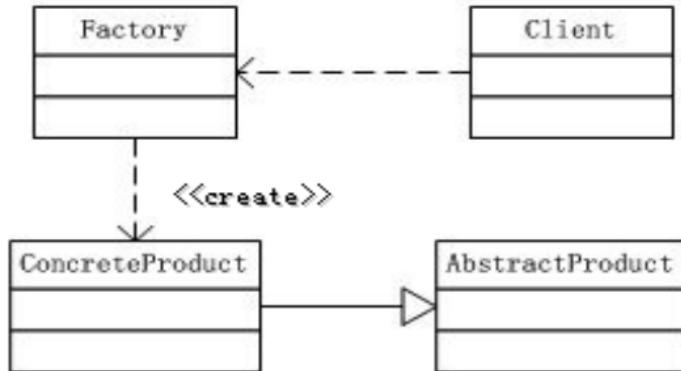
工厂 (Factory) 角色：接受客户端的请求，通过请求负责创建相应的产品对象。

抽象产品 (AbstractProduct) 角色：是工厂模式所创建对象的父类或是共同拥有的接口。可以是**抽象类或接口**。

具体产品 (ConcreteProduct) 对象：工厂模式所创建的对象都是这个角色的实例。



实例： 手机简单工厂



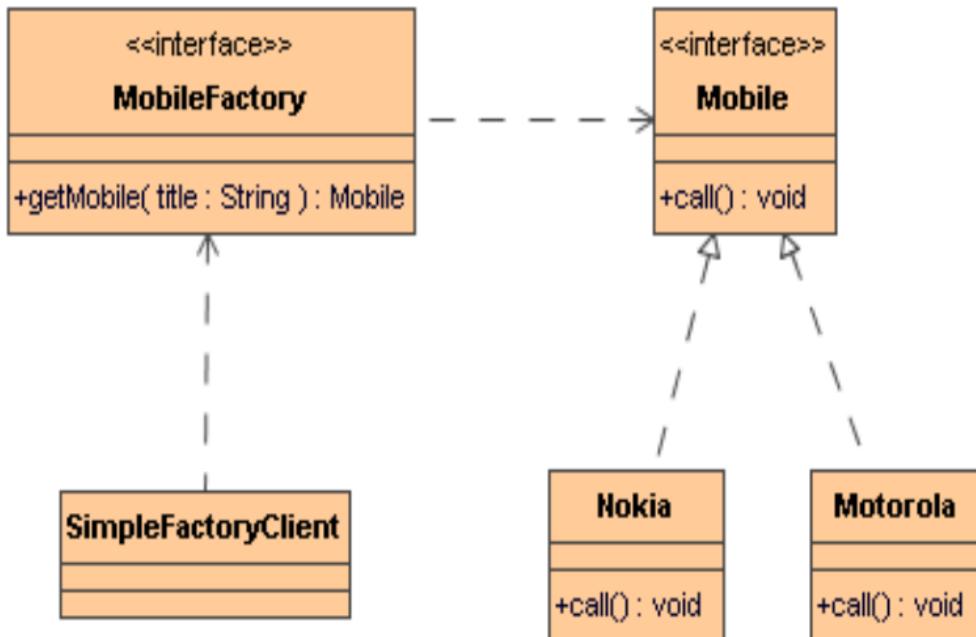
描述：

手机简单工厂可以直接返回两种不同的手机，它
可以生成Motorola手机，也可以生成Nokia手
机。

要求：

使用简单工厂模式设计手机简单工厂，
并给出类图及简单java代码。

类图：



Java代码：

```
//手机接口
public interface Mobile {
    public void call();
}
```

然后定义具体的产品——Nokie和Motorola手机

```
public class Nokia implements Mobile {  
    public void call() {  
        System.out.println("诺基亚手机");  
    }  
}
```

```
public class Motorola implements Mobile {  
    public void call(){  
        System.out.println("摩托罗拉手机");  
    }  
}
```

简单工厂类MobileFactory通过传入参数来确定返回哪种手机的相关信息

```
public class MobileFactory {  
    public Mobile getMobile(String title) throws Exception  
    {  
        if(title.equalsIgnoreCase("nokia")){  
            return new Nokia();  
        }  
        else if(title.equalsIgnoreCase("motorola")){  
            return new Motorola();  
        }  
        else{  
            throw new Exception("no such "+title+" mobile found");  
        }  
    }  
}
```

客户测试程序

```
public class Client {  
    public static void main(String argv[]) {  
        try {  
            MobileFactory mf = new MobileFactory();  
            Mobile m;  
            m=mf.getMobile("nokia");  
            m.call();  
            m=mf.getMobile("motorola");  
            m.call();  
        } catch(Expectation e) {e.printStackTrace();}  
    }  
}
```

输出结果：

诺基亚手机

摩托罗拉手机

优缺点

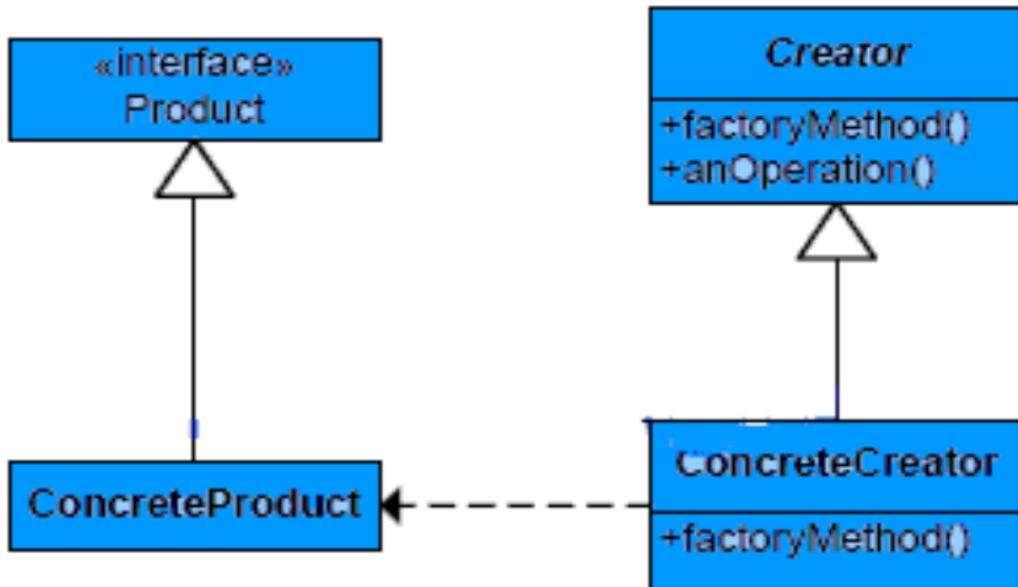
工厂类含有必要的判断逻辑，可以决定在什么时候创建哪一个产品类的实例，客户端可以免除直接创建产品对象的责任，而仅仅“消费”产品。简单工厂模式通过这种做法实现了对责任的分割。

当产品有复杂的多层等级结构时，工厂类只有自己，以不变应万变，就是模式的缺点。因为工厂类集中了所有产品创建逻辑，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，有可能造成工厂逻辑过于复杂，违背了“开放--封闭”原则(OCP)。



- 工厂类



优缺点

- 克服了简单工厂违背开闭原则的缺点，又保持了封装对象创建过程的优点
- 存在的问题
 - 每增加一个产品，就需要加一个产品工厂的类，增加了额外的开发量
 - 由客户端决定实例化哪一个工厂来实现选择类，选择的问题还是存在（修改客户端）

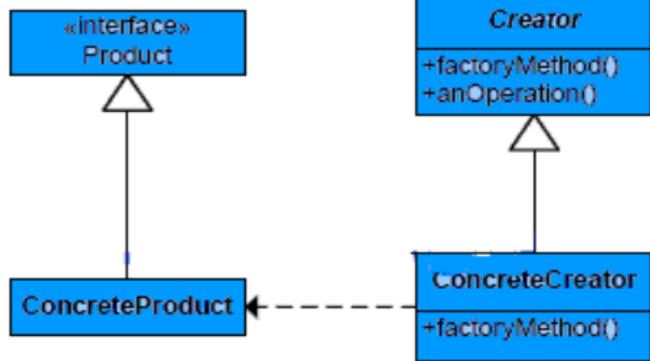
扩展的手机工厂

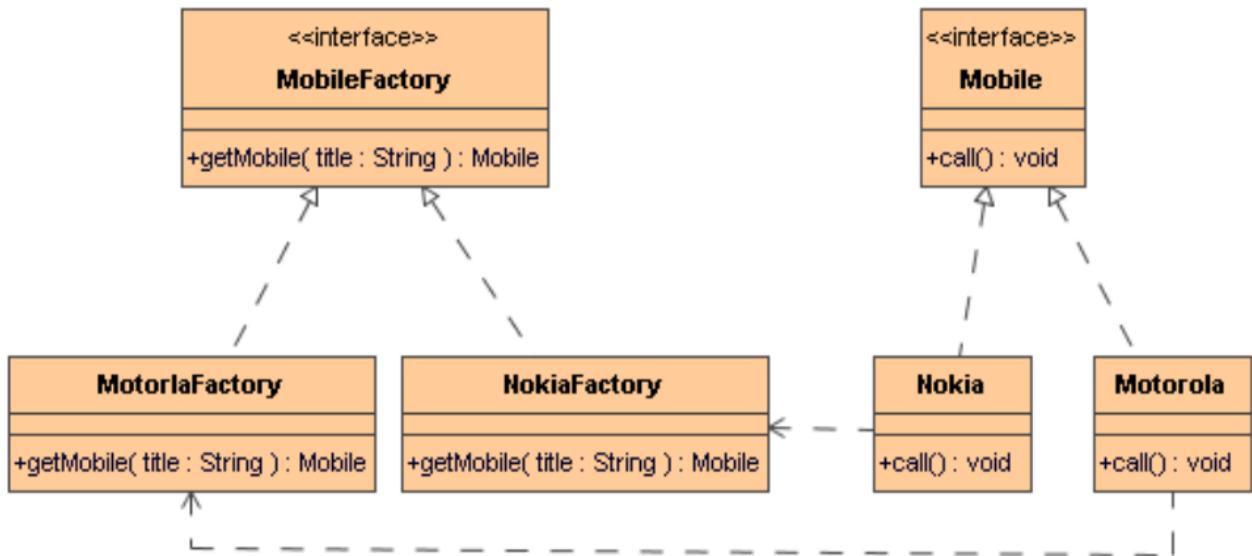
描述：

现实中不同品牌的手机应该由不同的手机厂家生产。

要求：

使用工厂方法对手机简单工厂进行扩展，给出相应的类图和java代码。





Java代码：

//手机接口

```
public interface Mobile {  
    public void call();  
}
```

//手机工厂接口

```
public interface MobileFactory {  
    public Mobile getMobile();  
}
```

然后定义具体的产品——Nokie和Motorola手机

```
public class Nokia implements Mobile {  
    public void call() {  
        System.out.println("诺基亚手机");  
    }  
}
```

```
public class Motorola implements Mobile {  
    public void call(){  
        System.out.println("摩托罗拉手机");  
    }  
}
```

摩托罗拉和诺基亚工厂分别实现了生产手机的方法，分别
返回摩托罗拉手机和诺基亚手机

```
public class MotorolaFactory implements MobileFactory {  
    public Mobile getMobile() {  
        System.out.print("摩托罗拉工厂制造了");  
        return new Motorola();  
    }  
}  
  
public class NokiaFactory implements MobileFactory {  
    public Mobile getMobile() {  
        System.out.print("诺基亚工厂制造了");  
        return new Nokia();  
    }  
}
```

客户测试程序

```
public class Client {  
    public static void main(String argv[]){  
        MobileFactory mf;  
        Mobile m;  
        mf=new MotorolaFactory();  
        m=mf.produceMobile();  
        m.call();  
        mf=new NokiaFactory();  
        m=mf.produceMobile();  
        m.call();  
    }  
}
```

输出结果：

摩托罗拉工厂制造了摩托罗拉手机
诺基亚工厂制造了诺基亚手机

Homework

- 将计算器的简单工厂模式改造成工厂模式。
- 要求有代码，有类图，有截图，最后，根据开闭原则，比较简单工厂与工厂。
- 电子版，12月22日提交

总结

简单工厂

- 工厂类负责创建的对象比较少
- 客户只知道传入工厂类的参数，对于如何创建对象（逻辑）不关心

工厂模式

- 有分角色创建

思考

- 如何解决避免分支判断的问题？（工厂模式）
- 反射+抽象工厂