

爪哇语言简单工厂创立性模式介绍

阎宏 www.yesky.com

版权声明

本文版权归作者和天极网所有

作者简介

阎宏, 1987年毕业于中国科技大学近代物理系, 1990年于中科院理论物理所获硕士, 1992年获博士。

从 1992 年到现在, 在美国从事软件研究开发工作。曾任职于汤臣金融 (Thomson Financial), 参与设计开发投资关系网站 www.IRChannel.com (原来叫 www.IRUniverse.com), 第一声 (www.FirstCall.com), 曾在奥本海默基金 (Oppenheimer) 开发股票实时交易系统, 曾在美国阿贡国家实验室从事专家系统的研究。

现在美国花旗银行 (Citibank) 工作, 副总裁级系统工程师。参与公司的网上银行设计与开发工作 (www.citidirect-online.com)。

PDF 文档制作

Java 研究组织

www.javaresearch.org

疾风摩郎

dengke@javaresearch.org

2002 年 7 月 16 日

研究和使用的创立性模式的必要性

面向对象的设计的目的之一, 就是把责任进行划分, 以分派给不同的对象。我们推荐这种划分责任的作法, 是因为它和封装 (Encapsulation) 和分派 (Delegation) 的精神是相符合的。创立性模式把对象的创立过程封装起来, 使得创立实例的责任与使用实例的责任分割开, 并由专门的模块分管实例的创立, 而系统在宏观上不再依赖于对象创立过程的细节。

所有面向对象的语言都有固定的创立对象的办法。爪哇语的办法就是使用 new 操作符。比如:

```
StringBuffer s = new StringBuffer(1000);
```

就创立了一个对象 s, 其类型是 StringBuffer。使用 new 操作符的短处是事先必须明确知道要实例化的类是什么, 而且实例化的责任往往与使用实例的责任不加区分。使用创立性模式将类实例化, 首先不必事先知道每次是要实例化哪一个类, 其次把实例化的责任与使用实例的责任分割开来, 可以弥补直接使用 new 操作符的短处。

而工厂模式就是专门负责将大量有共同接口的类实例化, 而且不必事先知道每次是要实例化哪一个类的模式。

工厂模式有几种形态

工厂模式有以下几种形态：

简单工厂 (Simple Factory) 模式

工厂方法 (Factory Method) 模式, 又称多形性工厂 (Polymorphic Factory) 模式

抽象工厂 (Abstract Factory) 模式, 又称工具箱 (Kit 或 Toolkit) 模式

介绍简单工厂模式

比如说, 你有一个描述你的后花园的系统, 在你的后花园里有各种的花, 但还没有水果。你现在要往你的系统里引进一些新的类, 用来描述下列的水果：

葡萄 Grapes

草莓 Strawberry

苹果 Apple

花和水果最大的不同, 就是水果最终是可以采摘食用的。那么, 很自然的作法就是建立一个各种水果都适用的接口, 这样一来这些水果类作为相似的数据类型就可以和你的系统的其余部分, 如各种的花有所不同, 易于区分。

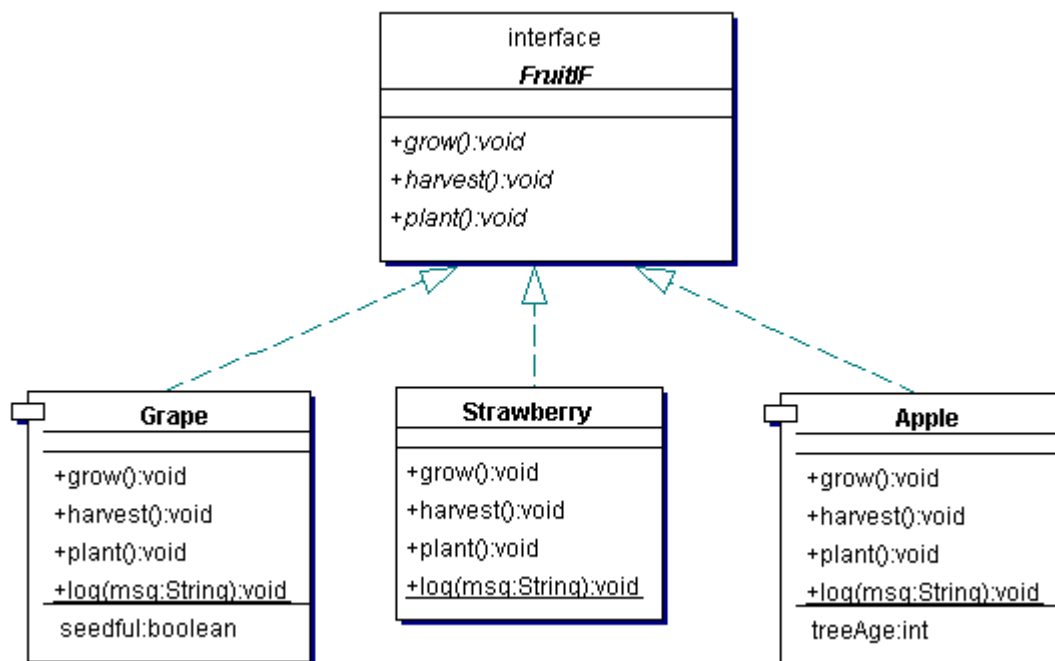


图 1. Grape, Strawberry 和 Apple 是拥有共同接口 FruitIF 的类。

代码清单 1. 接口 FruitIF 的源代码。这个接口确定了水果类必备的方法：种植 plant(), 生长 grow(), 以及收获 harvest()。

```
package com.javapatterns.simplefactory;
```

```
public interface FruitIF
{
    void grow();

    void harvest();

    void plant();

    String color = null;
    String name = null;
}
```

代码清单 2. 类 Apple 的源代码。苹果是多年生木本植物, 因此具备树龄 treeAge 性质。

```
package com.javapatterns.simplefactory;
```

```
public class Apple implements FruitIF
{
    public void grow()
    {
        log("Apple is growing...");
    }

    public void harvest()
    {
        log("Apple has been harvested.");
    }

    public void plant()
    {
        log("Apple has been planted.");
    }

    public static void log(String msg)
    {
        System.out.println(msg);
    }

    public int getTreeAge()
```

```

    {
        return treeAge;
    }

    public void setTreeAge(int treeAge)
    {
        this.treeAge = treeAge;
    }

    private int treeAge;
}

```

代码清单 3. 类 Grape 的源代码。葡萄分为有籽与无籽两种,因此具有 seedful 性质。

```

package com.javapatterns.simplefactory;

public class Grape implements FruitIF
{
    public void grow()
    {
        log("Grape is growing...");
    }

    public void harvest()
    {
        log("Grape has been harvested.");
    }

    public void plant()
    {
        log("Grape has been planted.");
    }

    public static void log(String msg)
    {
        System.out.println(msg);
    }

    public boolean getSeedful()
    {
        return seedful;
    }

    public void setSeedful(boolean seedful)

```

```

{
    this.seedful = seedful;
}

private boolean seedful;
}

```

代码清单 4. 类 Strawberry 的源代码。

```

package com.javapatterns.simplefactory;

public class Strawberry implements FruitIF
{
    public void grow()
    {
        log("Strawberry is growing...");
    }

    public void harvest()
    {
        log("Strawberry has been harvested.");
    }

    public void plant()
    {
        log("Strawberry has been planted.");
    }

    public static void log(String msg)
    {
        System.out.println(msg);
    }
}

```

你作为小花果园的主人兼园丁,也是系统的一部分,自然要由一个合适的类来代表,这个类就是 FruitGardener 类。这个类的结构请见下面的 UML 类图。

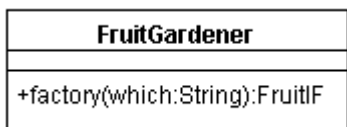


图 2. FruitGardener 类图。

FruitGardener 类会根据要求,创立出不同的水果类,比如苹果 Apple,葡萄 Grape 或草莓 Strawberry 的实例。而如果接到不合法的要求,FruitGardener 类

会给出例外 `BadFruitException`。

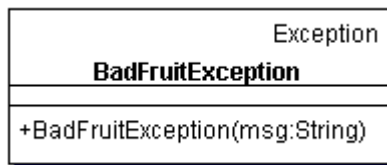


图 3. `BadFruitException` 类图。

代码清单 5. `FruitGardener` 类的源代码。

```
package com.javapatterns.simplefactory;

public class FruitGardener
{
    public FruitIF factory(String which) throws BadFruitException
    {
        if (which.equalsIgnoreCase("apple"))
        {
            return new Apple();
        }
        else if (which.equalsIgnoreCase("strawberry"))
        {
            return new Strawberry();
        }
        else if (which.equalsIgnoreCase("grape"))
        {
            return new Grape();
        }
        else
        {
            throw new BadFruitException("Bad fruit request");
        }
    }
}
```

代码清单 6. `BadFruitException` 类的源代码。

```
package com.javapatterns.simplefactory;

public class BadFruitException extends Exception
{
    public BadFruitException(String msg)
    {
        super(msg);
    }
}
```

```
}
```

在使用时,只须呼叫 FruitGardener 的 factory()方法即可

```
try
{
    FruitGardener gardener = new FruitGardener();

    gardener.factory("grape");
    gardener.factory("apple");
    gardener.factory("strawberry");
    ...
}
catch(BadFruitException e)
{
    ...
}
```

就这样你的小果园一定会有百果丰收啦!

简单工厂模式的定义

总而言之,简单工厂模式就是由一个工厂类根据参数来决定创立出那一种产品类的实例。下面的 UML 类图就精确定义了简单工厂模式的结构。

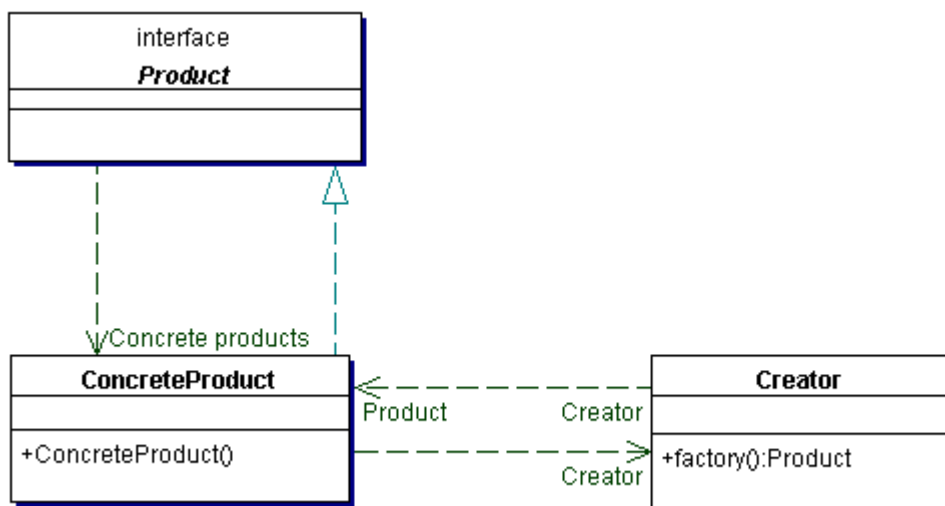


图 4. 简单工厂模式定义的类型图。

```
public class Creator
{
    public Product factory()
    {
```

```

        return new ConcreteProduct();
    }

}

public interface Product
{
}

public class ConcreteProduct implements Product
{
    public ConcreteProduct(){}
}

```

代码清单 7. 简单工厂模式框架的源代码。

简单工厂模式实际上就是我们要在后面介绍的,工厂方法模式的一个简化了的情形。在读者熟悉了本节所介绍的简单工厂模式后,就不难掌握工厂方法模式了。

问答题

1. 在本节开始时不是说,工厂模式就是在不使用 new 操作符的情况下,将.....类实例化的吗,可为什么在具体实现时,仍然使用了 new 操作符呢?

2. 在本节的小果园系统里有三种水果类,可为什么在图 3.(简单工厂模式定义类图)中产品(Product)类只有一种呢?

3. 请使用简单工厂模式设计一个创立不同几何形状,如圆形,方形和三角形实例的描图员(Art Tracer)系统。每个几何图形都要有画出 draw()和擦去 erase()两个方法。当描图员接到指令,要求创立不支持的几何图形时,要提出 BadShapeException 例外。

4. 请简单举例说明描图员系统怎样使用。

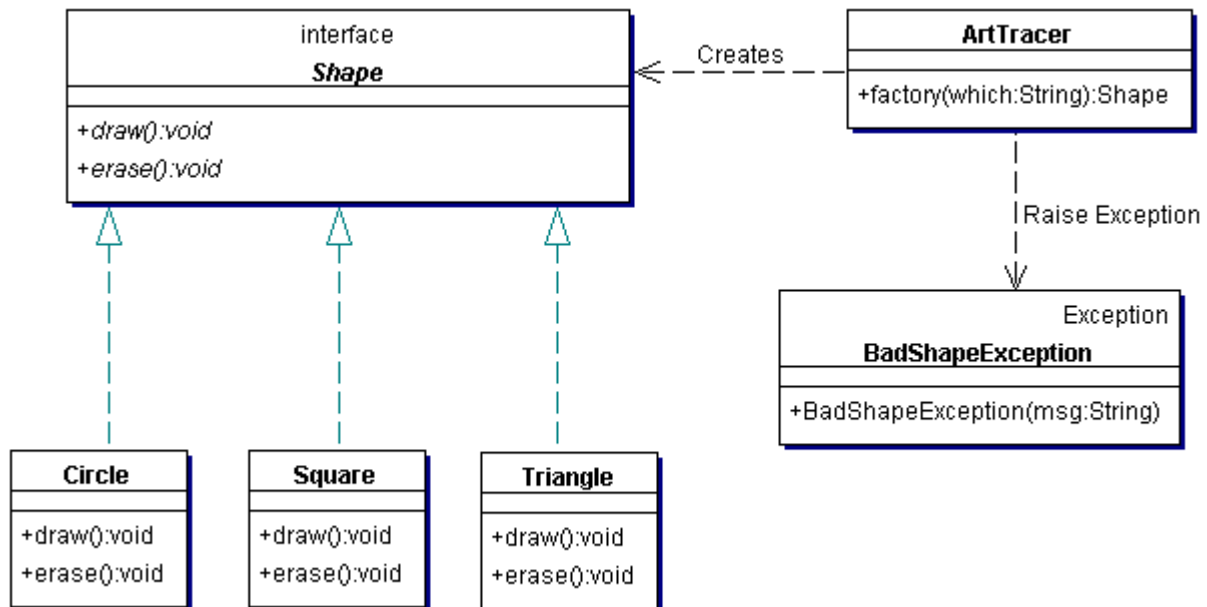
5. 在简单工厂模式的定义(见图 4)中和花果园例子中, factory()方法都是属于实例的,而非静态的或是类的方法。 factory()方法可不可以是静态的方法呢?

问答题答案

1. 对整个系统而言,工厂模式把具体使用 new 操作符的细节包装和隐藏起来。当然只要程序是用爪哇语言写的,爪哇语言的特征在细节里一定会出现的。

2. 图 3.(简单工厂模式定义的类型图),是精减后的框架性类图,用于给出这一模式的准确而精练的定义。产品(Product)类到底会有几种,则要对每个系统作具体分析。

3. 这里给出问题的完整答案。描图员(Art Tracer)系统的 UML 如下



系统的源代码如下

代码清单 8. ArtTracer 类的源代码。

```
package com.javapatterns.simplefactory.exercise;

public class ArtTracer
{
    public Shape factory(String which) throws BadShapeException
    {
        if (which.equalsIgnoreCase("circle"))
        {
            return new Circle();
        }
        else if (which.equalsIgnoreCase("square"))
        {
            return new Square();
        }
        else if (which.equalsIgnoreCase("triangle"))
        {
            return new Triangle();
        }
        else
        {

```

```
        throw new BadShapeException(which);
    }
}
}
```

代码清单 9. Shape 接口的源代码。

```
package com.javapatterns.simplefactory.exercise;

public interface Shape
{
    void draw();

    void erase();
}
```

代码清单 10. Square 类的源代码。

```
package com.javapatterns.simplefactory.exercise;

public class Square implements Shape
{
    public void draw()
    {
        System.out.println("Square.draw()");
    }

    public void erase()
    {
        System.out.println("Square.erase()");
    }
}
```

代码清单 11. Circle 类的源代码。

```
package com.javapatterns.simplefactory.exercise;

public class Circle implements Shape
{
    public void draw()
    {
        System.out.println("Circle.draw()");
    }

    public void erase()
    {
```

```
        System.out.println("Circle.erase()");
    }
}
```

代码清单 12. Triangle 类的源代码。

```
package com.javapatterns.simplefactory.exercise;

public class Triangle implements Shape
{
    public void draw()
    {
        System.out.println("Triangle.draw()");
    }

    public void erase()
    {
        System.out.println("Triangle.erase()");
    }
}
```

代码清单 13. BadShapeException 类的源代码。

```
package com.javapatterns.simplefactory.exercise;

public class BadShapeException extends Exception
{
    public BadShapeException(String msg)
    {
        super(msg);
    }
}
```

描图员(Art Tracer)系统使用方法如下：

```
try
{
    ArtTracer art = new ArtTracer();

    art.factory("circle");
    art.factory("square");
    art.factory("triangle");

    art.factory("diamond");
}
catch(BadShapeException e)
```

```
{  
    ...  
}
```

注意对 ArtTracer 类提出菱形(diamond)请求时,会收到 BadShapeException 例外。

显然 factory()可以是静态的或是类的方法。本文这样介绍简单工厂模式,是为了能方便与后面介绍的工厂方法模式作一比较。