《Head First 设计模式》：工厂方法模式

作者：jingqueyimu
原文链接：https://ld246.com/article/1595252503688
来源网站：链滴

正文

一、定义

工厂方法模式定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实化推迟到子类。

PS：在设计模式中，“实现一个接口”泛指实现某个超类型（可以是类或接口）的某个方法。

要点：

- 通过子类来创建具体的对象。客户只需要知道他们所使用的抽象类型即可。
- 由子类决定要实例化的类是哪一个，是指在编写创建者类时，不需要知道实际创建的产品是哪一个选择了使用哪个创建者子类，自然就决定了实际创建的产品是什么。
- 对象统一由定义好的工厂方法来创建。

二、实现步骤

1、创建产品抽象类

```
/**
 * 产品抽象类
 */
public abstract class Product {

    String name;

    public String getName() {
```

```
        return name;
    }
}
```

2、创建具体的产品，并继承产品抽象类

（1）产品A1

```
/**
 * 产品A1
 */
public class ConcreteProductA1 extends Product {

    public ConcreteProductA1() {
        name = "ConcreteProductA1";
    }
}
```

（2）产品A2

```
/**
 * 产品A2
 */
public class ConcreteProductA2 extends Product {

    public ConcreteProductA2() {
        name = "ConcreteProductA2";
    }
}
```

（3）产品B1

```
/**
 * 产品B1
 */
public class ConcreteProductB1 extends Product {

    public ConcreteProductB1() {
        name = "ConcreteProductB1";
    }
}
```

（4）产品B2

```
/**
 * 产品B2
 */
public class ConcreteProductB2 extends Product {

    public ConcreteProductB2() {
        name = "ConcreteProductB2";
```

```
    }
}
```

3、创建创建者抽象类，并定义用来创建产品的工厂方法

创建者一般为需要用到产品的类，需要的产品则通过类中的工厂方法创建。

```
/**
 * 创建者抽象类
 */
public abstract class Creator {

    /**
     * 创建产品（工厂方法）
     */
    protected abstract Product createProduct(String productType);
}
```

4、创建具体的创建者，并继承创建者抽象类

具体的创建者需要实现创建产品的工厂方法。

（1）创建者1

```
/**
 * 创建者1
 */
public class ConcreteCreator1 extends Creator {

    @Override
    protected Product createProduct(String productType) {
        // 由具体的创建者（子类）决定创建哪个类的对象
        if ("A".equals(productType)) {
            return new ConcreteProductA1();
        } else if ("B".equals(productType)) {
            return new ConcreteProductB1();
        }
        return null;
    }
}
```

（2）创建者2

```
/**
 * 创建者2
 */
public class ConcreteCreator2 extends Creator {

    @Override
    protected Product createProduct(String productType) {
        // 由具体的创建者（子类）决定创建哪个类的对象
        if ("A".equals(productType)) {
```

```
            return new ConcreteProductA2();
        } else if ("B".equals(productType)) {
            return new ConcreteProductB2();
        }
        return null;
    }
}
```

5、创建者通过工厂方法创建产品

```
public class Test {

    public static void main(String[] args) {
        // 创建者1
        Creator creator1 = new ConcreteCreator1();
        // 创建者2
        Creator creator2 = new ConcreteCreator2();

        // 通过工厂方法创建产品
        Product product = creator1.createProduct("A");
        System.out.println("创建者1创建产品A：" + product.getName());
        product = creator2.createProduct("A");
        System.out.println("创建者2创建产品A：" + product.getName());
    }
}
```

三、举个栗子

1、背景

假设你有一个披萨店，出售多种类型的披萨：芝士披萨、蛤蜊披萨、素食披萨等。由于经营有成，你算推广自己的加盟店。

为了确保加盟店运营的质量，你希望加盟店能够采用固定的制作流程。但是由于区域的差异，每家加店都可能想要提供不同风味的披萨（比如纽约、芝加哥、加州），因此又必须允许加盟店能够自由地作该区域的风味。

2、实现

披萨店子类通过实现创建披萨方法来决定要创建什么风味的披萨。

（1）创建披萨抽象类

```
/**
 * 披萨抽象类
 */
public abstract class Pizza {

    /**
     * 名称
     */
```

```
    String name;
    /**
     * 面团
     */
    String dough;
    /**
     * 酱料
     */
    String sauce;
    /**
     * 佐料
     */
    ArrayList<String> toppings = new ArrayList<>();

    void prepare() {
        System.out.println("Preparing " + name);
        System.out.println("Tossing dough...");
        System.out.println("Adding souce...");
        System.out.println("Adding toppings: ");
        for (int i = 0; i < toppings.size(); i++) {
            System.out.println(" "+ toppings.get(i));
        }
    }

    /**
     * 烘烤
     */
    void bake() {
        System.out.println("Bake for 25 minutes at 350");
    }

    /**
     * 切片
     */
    void cut() {
        System.out.println("Cutting the pizza into diagonal slices");
    }

    /**
     * 装盒
     */
    void box() {
        System.out.println("Place pizza in official PizzaStore box");
    }

    public String getName() {
        return name;
    }
}
```

（2）创建不同风味、不同类型的披萨

```
/**
 * 纽约风味的芝士披萨
```

```
 */
public class NYStyleCheesePizza extends Pizza {

    public NYStyleCheesePizza() {
        name = "NY Style Sauce and Cheese Pizza";
        dough = "Thin Crust Dough";
        sauce = "Marinara Sauce";
        toppings.add("Grated Reggiano Cheese");
    }
}

/**
 * 纽约风味的蛤蜊披萨
 */
public class NYStyleClamPizza extends Pizza {

    public NYStyleClamPizza() {
        name = "NY Style Sauce Clam Pizza";
        dough = "Thin Crust Dough";
        sauce = "Marinara Sauce";
        toppings.add("Fresh Clams");
    }
}

/**
 * 芝加哥风味的芝士披萨
 */
public class ChicagoStyleCheesePizza extends Pizza {

    public ChicagoStyleCheesePizza() {
        name = "Chicago Style Deep Dish Cheese Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";
        toppings.add("Shredded Mozzarella Cheese");
    }

    void cut() {
        System.out.println("Cutting the pizza into square slices");
    }
}

/**
 * 芝加哥风味的蛤蜊披萨
 */
public class ChicagoStyleClamPizza extends Pizza {

    public ChicagoStyleClamPizza() {
        name = "Chicago Style Clam Pizza";
        dough = "Extra Thick Crust Dough";
        sauce = "Plum Tomato Sauce";
        toppings.add("Frozen Clams");
    }

    void cut() {
```

```
        System.out.println("Cutting the pizza into square slices");
    }
}
```

（3）创建披萨店抽象类

```
/**
 * 披萨店抽象类
 */
public abstract class PizzaStore {

    /**
     * 订购披萨
     */
    public Pizza orderPizza(String type) {
        Pizza pizza = createPizza(type);
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
        return pizza;
    }

    /**
     * 创建披萨（工厂方法）
     */
    protected abstract Pizza createPizza(String type);
}
```

（4）创建不同风味的披萨店

```
/**
 * 纽约风味披萨店
 */
public class NYStylePizzaStore extends PizzaStore {

    @Override
    protected Pizza createPizza(String type) {
        Pizza pizza = null;
        if ("cheese".equals(type)) {
            pizza = new NYStyleCheesePizza();
        } else if ("clam".equals(type)) {
            pizza = new NYStyleClamPizza();
        }
        return pizza;
    }
}

/**
 * 芝加哥风味披萨店
 */
public class ChicagoStylePizzaStore extends PizzaStore {
```

```
    @Override
    protected Pizza createPizza(String type) {
        Pizza pizza = null;
        if ("cheese".equals(type)) {
            pizza = new ChicagoStyleCheesePizza();
        } else if ("clam".equals(type)) {
            pizza = new ChicagoStyleClamPizza();
        }
        return pizza;
    }
}
```

（5）使用不同风味的披萨店订购披萨

```
public class Test {

    public static void main(String[] args) {
        // 纽约风味披萨店
        PizzaStore nyStore = new NYStylePizzaStore();
        // 芝加哥风味披萨店
        PizzaStore chicagoStore = new ChicagoStylePizzaStore();

        // 订购芝士披萨
        Pizza pizza = nyStore.orderPizza("cheese");
        System.out.println("Ethan ordered a " + pizza.getName() + "\n");
        pizza = chicagoStore.orderPizza("cheese");
        System.out.println("Joel ordered a " + pizza.getName() + "\n");
    }
}
```