链滴

《Head First 设计模式》：模板方法模式

作者：jingqueyimu

原文链接：https://ld246.com/article/1598794448836

来源网站：链滴

正文

一、定义

模板方法模式在一个方法中定义一个算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可在不改变算法结构的情况下，重新定义算法中的某些步骤。

要点：

- 模板方法定义了一个算法的步骤，每个步骤都被一个方法所代表，而这几个方法的具体实现可由子提供。
- 模板方法可确保算法的结构保持不变，同时由子类提供部分实现。

二、实现步骤

1、创建一个抽象类，并定义模板方法

模板方法一般声明为 final，以免子类改变算法的步骤。

抽象类中，可以声明一些钩子方法，子类视情况决定要不要覆盖它们。钩子的存在，可以让子类有能对算法的不同点进行挂钩，使得模板方法更具有弹性。

```
/**
 * 抽象类
 */
public abstract class AbstractClass {

    /**
     * 模板方法
```

```
     */
    public final void templateMethod() {
        // 公共步骤，由抽象类实现
        commonStep();
        // 依赖于子类的步骤，由子类实现
        step1();
        step2();
        step3();
        // 钩子方法，由子类决定要不要覆盖
        hook();
    }

    /**
     * 步骤1
     */
    public abstract void step1();

    /**
     * 步骤2
     */
    public abstract void step2();

    /**
     * 步骤3
     */
    public abstract void step3();

    /**
     * 公共步骤
     */
    private void commonStep() {
        System.out.println("I'm common step!");
    }

    /**
     * 钩子方法
     */
    public void hook() {
        // 可以空实现，也可以提供默认实现
    }
}
```

2、创建具体子类，并提供算法步骤的具体实现

（1）具体子类A

```
/**
 * 具体子类A
 */
public class ConcreteClassA extends AbstractClass{

    @Override
    public void step1() {
```

```
        System.out.println("I'm step A1!");
    }

    @Override
    public void step2() {
        System.out.println("I'm step A2!");
    }

    @Override
    public void step3() {
        System.out.println("I'm step A3!");
    }

    @Override
    public void hook() {
        System.out.println("I'm hook step A!");
    }
}
```

（2）具体子类B

```
/**
 * 具体子类B
 */
public class ConcreteClassB extends AbstractClass{

    @Override
    public void step1() {
        System.out.println("I'm step B1!");
    }

    @Override
    public void step2() {
        System.out.println("I'm step B2!");
    }

    @Override
    public void step3() {
        System.out.println("I'm step B3!");
    }
}
```

3、通过使用不同的子类，来确定具体的算法步骤

```
public class Test {

    public static void main(String[] args) {
        AbstractClass classA = new ConcreteClassA();
        classA.templateMethod();
        System.out.println();
        AbstractClass classB = new ConcreteClassB();
        classB.templateMethod();
    }
```

```
}
```

三、举个栗子

1、背景

《星巴兹咖啡师傅训练手册》规定了，准备星巴兹饮料时，必须精确地遵循下面的冲泡法：

星巴兹咖啡冲泡法：

1. 把水煮沸
2. 用沸水冲泡咖啡
3. 把咖啡倒进杯子
4. 加糖和牛奶

星巴兹茶冲泡法：

1. 把水煮沸
2. 用沸水浸泡茶叶
3. 把茶倒进杯子
4. 加柠檬

现在，让我们扮演“代码师傅”，写一些代码来创建咖啡和茶。

2、实现

（1）创建咖啡因饮料抽象类，并定义冲泡步骤的方法

```
/**
 * 咖啡因饮料抽象类
 */
public abstract class CaffeineBeverage {

    /**
     * 准备冲泡法（模板方法）
     */
    final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        if (customerWantsCondiments()) {
            addCondiments();
        }
    }

    /**
     * 冲泡（依赖于子类的步骤）
     */
    abstract void brew();
```

```
    /**
     * 添加调料（依赖于子类的步骤）
     */
    abstract void addCondiments();

    /**
     * 烧水（公共步骤）
     */
    void boilWater() {
        System.out.println("Boiling water");
    }

    /**
     * 把饮料倒进杯子（公共步骤）
     */
    void pourInCup() {
        System.out.println("Pouring into cup");
    }

    /**
     * 顾客是否想要添加调料（钩子方法）
     */
    boolean customerWantsCondiments() {
        return true;
    }
}
```

（2）创建具体的饮料，并提供具体冲泡步骤的实现

```
/**
 * 咖啡
 */
public class Caffee extends CaffeineBeverage{

    @Override
    void brew() {
        System.out.println("Dripping Coffee through filter");
    }

    @Override
    void addCondiments() {
        System.out.println("Adding Sugar and Milk");
    }

    @Override
    boolean customerWantsCondiments() {
        String answer = askCustomer();
        if (answer.toLowerCase().startsWith("y")) {
            return true;
        } else {
            return false;
        }
    }
```

```
    /**
     * 询问顾客
     */
    private String askCustomer() {
        System.out.print("Would you like milk and suger with your coffee (y/n)?");
        String answer = null;
        Scanner scanner = new Scanner(System.in);
        if (scanner.hasNext()) {
            answer = scanner.nextLine();
        }
        scanner.close();
        if (answer == null) {
            return "no";
        }
        return answer;
    }
}

/**
 * 茶
 */
public class Tea extends CaffeineBeverage{

    @Override
    void brew() {
        System.out.println("Steeping the tea");
    }

    @Override
    void addCondiments() {
        System.out.println("Adding Lemon");
    }
}
```

（3）冲泡饮料

```
public class Test {

    public static void main(String[] args) {
        // 茶
        System.out.println("\nMaking tea...");
        CaffeineBeverage tea = new Tea();
        tea.prepareRecipe();
        // 咖啡
        System.out.println("\nMaking coffee...");
        CaffeineBeverage caffee = new Caffee();
        caffee.prepareRecipe();
    }
}
```