链滴

设计模式 |10 模板方法模式

作者：qq692310342
原文链接：https://ld246.com/article/1569466211024
来源网站：链滴

开头说几句

博主的博客地址： https://www.jeffcc.top/

博主学习设计模式用的书是Head First的《设计模式》，强烈推荐配套使用！

什么是模板方法模式

权威定义：

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。

博主的理解：

模板方法其实就是封装了不变的部分，对外开放了对变化的部分的拓展，通过使用final修饰的方法达实现的一个算法的骨架，并且将算法的需要被延迟加载的方法修饰为abstract方法。同时也可以使用子的思路来进一步拓展。

模板方法模式与工厂模式

回顾工厂方法模式定义：工厂方法模式定义了一个创建对象的接口，但是由子类决定实例化的类是哪个。工厂方法让实例化延迟到了子类。

相似之处：大家都是将实现延迟到了子类，并且将不变的部分封装在父类，并且都是通过抽象类的继方式实现的。

不同之处：模板方法是通过final修饰的算法骨架实现的，而我们的工厂方法模式是通过定义的不变部到顶层工厂中，并没有限制子类的对于父类的工厂的修改。

设计原则

1. 依赖倒置原则，要依赖抽象，不要依赖具体类，具体做法是需要高层组件（工厂）和底层组件（实类）之间不要有太多的依赖关系，而是可以通过一个共同的抽象类（工厂产生的对象）来实现依赖倒。

2. 多用组合，少用继承。

3. 针对接口编程，而不是针对实现编程。

4. 为交互对象之间的松耦合设计而努力。

5. 类应该对外开放拓展，对修改关闭。

6. 依赖抽象，不依赖具体类。

7. 最少知识原则。

8. 好莱坞原则：别来找我，我会找你的；意思是将决策权交给高层模块中，以决定什么时候调用底层块，在模板方法中体现就是使用倒钩的方法，让底层的组件将自己倒挂在高层超类中，而是否调用是过顶层决定的。

设计要点

1. 巧用钩子，好莱坞原则；

2. 算法核心要通过final修饰以防被修改；

3. 必须子类实例化的方法需要通过abstract修饰；

设计实例

设计背景

设计一个制作奶茶的模板方法，并且实例化几款具体的奶茶制作；

设计思路

首先我们需要将制作奶茶的大体步骤抽取出来，作为核心算法：

1、增加配料（延迟到子类）

2、加水（公共部分）

3、搅拌机搅拌（公共部分）

4、加冰（使用钩子，不是每个都要加 但是默认加冰）

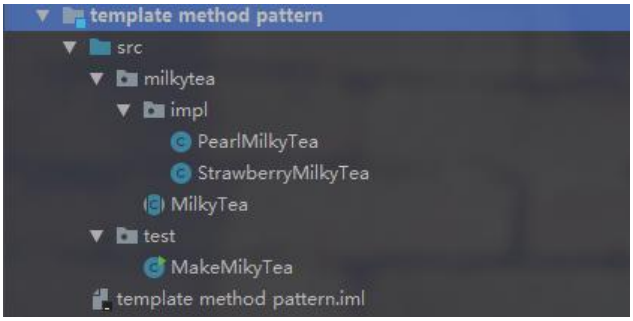
5、包装机包装（公共部分）

6、摇匀（公共部分）

7、提醒奶茶制作成功（延迟到子类）

项目类图

项目结构



奶茶模板方法

```
package milkytea;

/**
 * 奶茶的制作超类 模板方法
 */
public abstract class MilkyTea {

    /**
     * 抽象出一层final的制作奶茶的算法
     */
    public final void createMilkyTea() {
        addIngredients();
        addWater();
        stir();
        //钩子函数的体现 好莱坞原则的体现
        if(addIceOrNot()){
            addIce();
        }
        packaging();
        shakeUp();
        makeSuccess();
    }

    /**
     * 添加配料的方法
     * 延迟到子类实现
     */
    public abstract void addIngredients();

    /**
     * 添加水
```

```
     * 公共部分
     * 父类提供实现
     */
    public void addWater() {
        System.out.println("奶茶加水成功！");
    }

    /**
     * 搅拌
     * 公共部分
     * 父类提供实现
     */
    public void stir() {
        System.out.println("奶茶搅拌成功！");
    }

    /**
     * 加冰
     * 使用钩子（在算法中体现）
     */
    public void addIce() {
        System.out.println("奶茶加冰成功！");
    }

    /**
     * 使用一个方法来让子类实现，是否加冰
     *
     * @return true 加冰 false 不加
     */
    public boolean addIceOrNot() {
        return true;
    }

    /**
     * 包装
     * 公共部分
     * 父类提供实现
     */
    public void packaging() {
        System.out.println("奶茶包装成功！");
    }

    /**
     * 摇匀
     * 公共部分
     * 父类提供实现
     */
    public void shakeUp() {
        System.out.println("奶茶摇匀成功！");
    }

    /**
     * 制作成功的提醒
     * 延迟到子类
```

```
     */
    public abstract void makeSuccess();

}
```

珍珠奶茶实现

```
package milkytea.impl;

import milkytea.MilkyTea;

/**
 * 制作珍珠奶茶
 */
public class PearlMilkyTea extends MilkyTea {
    /**
     * 添加原料
     */
    @Override
    public void addIngredients() {
        System.out.println("珍珠奶茶添加：珍珠+椰果");
    }

    @Override
    public void makeSuccess() {
        System.out.println("珍珠奶茶制作成功！");
    }
}
```

草莓奶茶实现

```
package milkytea.impl;

import milkytea.MilkyTea;

/**
 * 制作草莓奶茶
 */
public class StrawberryMilkyTea extends MilkyTea {
    /**
     * 添加原料
     */
    @Override
    public void addIngredients() {
        System.out.println("草莓奶茶添加：草莓+果冻");
    }

    //体现钩子 重写不加冰
    @Override
    public boolean addIceOrNot() {
        return false;
```

```
    }

    @Override
    public void makeSuccess() {
        System.out.println("草莓奶茶制作成功！");
    }
}
```

测试

```
package test;

import milkytea.MilkyTea;
import milkytea.impl.PearlMilkyTea;
import milkytea.impl.StrawberryMilkyTea;

/**
 * 测试做奶茶
 */
public class MakeMikyTea {
    public static void main(String[] args) {
        //创建珍珠奶茶实例
        MilkyTea peralMilkyTea = new PearlMilkyTea();
        //创建草莓奶茶实例
        MilkyTea strawberryMilkyTea = new StrawberryMilkyTea();

        //做奶茶
        peralMilkyTea.createMilkyTea();
        System.out.println("----------------------------");
        strawberryMilkyTea.createMilkyTea();

    }
}
```

输出

```
珍珠奶茶添加：珍珠+椰果
奶茶加水成功！
奶茶搅拌成功！
奶茶加冰成功！
奶茶包装成功！
奶茶摇匀成功！
珍珠奶茶制作成功！
----------------------------
草莓奶茶添加：草莓+果冻
奶茶加水成功！
奶茶搅拌成功！
奶茶包装成功！
奶茶摇匀成功！
草莓奶茶制作成功！
```

Process finished with exit code 0

回到定义

相信大家看完上面写的小demo可以很明显的看出模板方法的特点了：核心算法通过final修饰防止被改，算法中不变的部分由父类实现，变化的部分封装成abstract方法延迟到子类实现，钩子函数的巧利用实现了子类管理是否使用父类的组件，在这里我们的父类是一个底层组件，子类是属于高层组件所以我们实现了好莱坞原则。

END

2019年9月26日10:50:01