策略模式设计样例及特点

概念：

策略模式是一种行为型模式，它将对象和行为分开，将行为定义为 一个行为接口 和 具体行为的实现。策略模式最大的特点是行为的变化，行为之间可以相互替换。每个if判断都可以理解为就是一个策略。

特点：

策略模式把对象本身和行为区分开来，因此我们整个模式也分为三个部分。

1、抽象策略类(Strategy):策略的抽象，行为方式的抽象

2、具体策略类(ConcreteStrategy):具体的策略实现，每一种行为方式的具体实现。

3、环境类(Context):用来封装具体行为,操作策略的上下文环境。

举例：

这里举个简单的例子,来理解开发中运用策略模式的场景。

有一个打车软件,现在有三种计费模式给 用户 选择，1、拼车 2、快车 3、豪车这个时候用户就是对象，三种计费方式就是行为，可以根据不同的行为计算不同不通的费用。

（1）传统实现方式

/\*\*

\* @Description: 这里只展示计费最终费用示例

\*

\* @param type 计费类型

\* @param originalPrice 原始价格

\*/

public Double calculationPrice(String type, Double originalPrice) {

//拼车计费

if (type.equals("pc")) {

return originalPrice \* 0.5;

}

//快车计费

if (type.equals("kc")) {

return originalPrice \* 1;

}

//豪车计费

if (type.equals("hc")) {

return originalPrice \* 2;

}

return originalPrice;

}

传统的实现方式，通过传统if代码判断。这样就会导致后期的维护性非常差。当后期需要新增计费方式，还需要在这里再加上if(),也不符合设计模式的开闭原则。

（2）策略模式实现

抽象策略类：

/\*\*

\* 出行策略接口

\*/

public interface PriceStrategy {

/\*\*

\* @param originalPrice 原始价格

\* @return 计算后的价格

\*/

Double countPrice(Double originalPrice);

}

具体策略实现类：

/\*\*

\* @Description: 拼车的计费方式

\*/

public class PcStrategy implements PriceStrategy {

@Override

public Double countPrice(Double originalPrice) {

return originalPrice \* 0.5;

}

}

/\*\*

\* @Description: 快车的计费方式

\*/

public class KcStrategy implements PriceStrategy {

@Override

public Double countPrice(Double originalPrice) {

return originalPrice \* 1;

}

}

/\*\*

\* @Description: 拼车的计费方式

\*/

public class HcStrategy implements PriceStrategy {

@Override

public Double countPrice(Double originalPrice) {

return originalPrice \* 2;

}

}

环境类：

也叫做上下文类或环境类，起承上启下封装作用。

/\*\*

\* 负责和具体的策略类交互

\* 这样的话，具体的算法和直接的客户端调用分离了，使得算法可以独立于客户端独立的变化。

\* 如果使用spring的依赖注入功能，还可以通过配置文件，动态的注入不同策略对象，动态的切换不同的算法.

\*/

public class PriceContext {

/\*\*

\* 出行策略接口

\*/

private PriceStrategy riceStrategy;

/\*\*

\* 构造函数注入

\*/

public PriceContext(PriceStrategy riceStrategy) {

this.riceStrategy = riceStrategy;

}

/\*\*

\* 计算价格

\*/

public Double countPrice(Double originalPrice) {

return riceStrategy.countPrice(originalPrice);

}

}

测试类：

public static void main(String[] args) {

//具体行为策略

PriceStrategy pcStrategy = new PcStrategy();

PriceStrategy kcStrategy = new KcStrategy();

PriceStrategy hcStrategy = new HcStrategy();

//用户选择不同的策略

PriceContext pcContext = new PriceContext(pcStrategy);

PriceContext kcContext = new PriceContext(kcStrategy);

PriceContext hcContext = new PriceContext(hcStrategy);

System.out.println("拼车价格 = " + pcContext.countPrice(10D));

System.out.println("快车价格 = " + kcContext.countPrice(10D));

System.out.println("豪车价格 = " + hcContext.countPrice(10D));

}