一、设计问题

实验一 Strategy模式的应用

某公司专门销售各种打印机，销售打印机时都有一定的折扣让利给顾客，但折扣计算的方法有很多种，如：不打折；每台减扣固定的金额；按售价的5%打折等等，且折扣计算方法可能发生变化。现在要开发该公司的销售系统，请你设计一个方案，实现打印机销售时的折扣计算，你的方案应该能够使得在销售打印机（即使是同一种打印机）时可以灵活的选用折扣计算方法来给出打印机的实际售价，增加折扣计算方法时不需要修改原有的程序代码。

请用strategy模式设计解决方案，给出Java源程序，绘制相应的类图，编写实验报告。系统使用图形用户界面，用户在图形用户界面选择打印机品牌型号，折扣计算方法，再单击确定按钮系统就计算出打印机的实际售价。

二、问题分析与模式选用

实现打折功能有多种算法或者策略，我们可以根据环境或者条件的不同选择不同的算法或者策略来完成该功能。

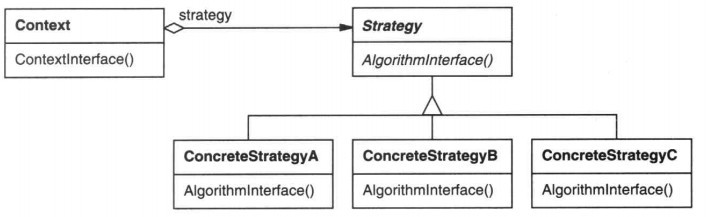
一种常用的方法是硬编码(Hard Coding)在一个类中，如需要提供多种查找算法，可以将这些算法写到一个类中，在该类中提供多个方法，每一个方法对应一个具体的查找算法；当然也可以将这些查找算法封装在一个统一的方法中，通过if…else…或者case等条件判断语句来进行选择。这两种实现方法我们都可以称之为硬编码，如果需要增加一种新的查找算法，需要修改封装算法类的源代码；更换查找算法，也需要修改客户端调用代码。在这个算法类中封装了大量查找算法，该类代码将较复杂，维护较为困难。如果我们将这些策略包含在客户端，这种做法更不可取，将导致客户端程序庞大而且难以维护，如果存在大量可供选择的算法时问题将变得更加严重。

如何让算法和对象分开来，使得算法可以独立于使用它的客户而变化？

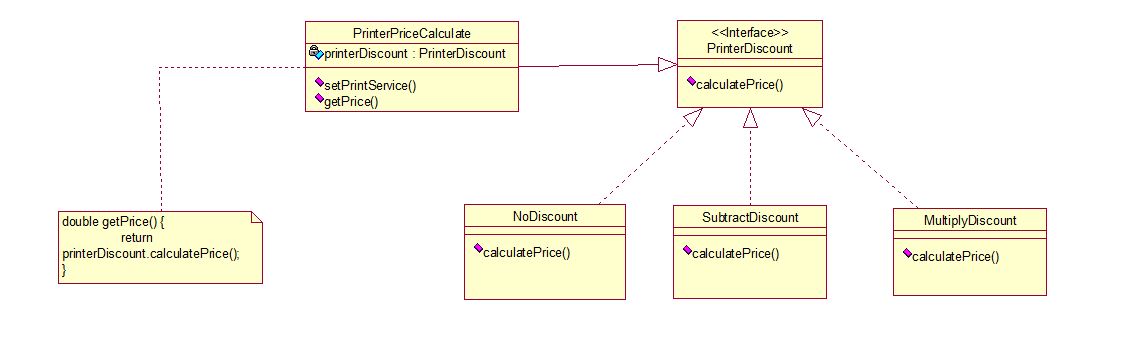
解决方案：

策略模式：定义一系列的算法,把每一个算法封装起来, 并且使它们可相互替换。本模式使得算法可独立于使用它的客户而变化。策略模式把对象本身和运算规则区分开来，其功能非常强大，因为这个设计模式本身的核心思想就是面向对象编程的多形性的思想。

策略模式UML类图：



三、设计方案



定义抽象策略类PrinterDiscount

具体策略类NoDiscount、SubtractDiscount、MultiplyDiscount

环境类PrinterPriceCalculate

在使用时只需实例化PrinterDiscount，再设置具体策略，即可通过PrinterDiscount计算折扣价。如果要增加新的策略，只需实现PrinterDiscount接口重写calculatePrice函数即可

**四、运行结果及效果分析**





五、实现源代码

以附件形式上传源代码