**策略模式**定义了一系列的算法，并将每一个算法封装起来，而且使它们还可以相互替换。策略模式让算法独立于使用它的客户而独立变化。

举个例子：

出场人物：小菜（菜鸟级程序员），大鸟（骨灰级程序员）

　　要求：做一个商场收银软件，营业员根据客户所购买的商品的单价和数量，向客户收费。

商场收银系统v1.0运行截图以及关键代码

[复制代码](javascript:void(0);)

*public partial class Form1 : Form    {   
 public Form1()*

*{*

*InitializeComponent();*

*}*

*//声明一个double变量total来计算总计*

*double total = 0.0d;*

*private void btnOk\_Click(object sender, EventArgs e)*

*{*

*//声明一个double变量totalPrices来计算每个商品的单价（txtPrice）\*数量(txtNum)后的合计*

*double totalPrices=Convert.ToDouble(txtPrice.Text) \* Convert.ToDouble(txtNum.Text);*

*total = total + totalPrices;//将每个商品合计计入总计*

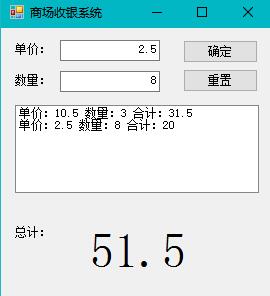
*lbxList.Items.Add("单价："+txtPrice.Text+" 数量："+txtNum.Text+" 合计："+totalPrices.ToString());//在列表框中显示信息*

*lblResult.Text = total.ToString();//在lblResult标签上显示总计数*

*}*

*}  
}*

[复制代码](javascript:void(0);)



现在问题来了：

大鸟：昨天妈妈节商场的所有商品打八折，那么上面的代码需要怎样修改呢？

小菜：上面的totalPrices乘以0.8不就可以了，没什么大不了的。

大鸟：但是过了妈妈节商场老板决定恢复原来的价格，那你是不是又要再改回来？不过再过不久也就六一儿童节了，商场要是打算打七折呢，你怎么办？

小菜：这样呐，我就加一个combox，里面给出打折选项就可以啦。

大鸟笑而不语。

商场收银系统v1.1运行截图以及关键代码

[复制代码](javascript:void(0);)

double total = 0.0d;

private void Form1\_Load(object sender, EventArgs e)

{

cbxType.Items.AddRange(new object[] {"正常收费","打八折","打七折","打五折"});//在combox中添加下拉选项

cbxType.SelectedIndex = 0;

}

private void btnOk\_Click(object sender, EventArgs e)

{

double totalPrices=0d;

switch(cbxType.SelectedIndex)

{

case 0:

totalPrices = Convert.ToDouble(txtPrice.Text) \* Convert.ToDouble(txtNum.Text);

break;

case 1:

totalPrices = Convert.ToDouble(txtPrice.Text) \* Convert.ToDouble(txtNum.Text) \* 0.8;

break;

case 2:

totalPrices = Convert.ToDouble(txtPrice.Text) \* Convert.ToDouble(txtNum.Text) \* 0.7;

break;

case 3:

totalPrices = Convert.ToDouble(txtPrice.Text) \* Convert.ToDouble(txtNum.Text) \* 0.5;

break;

}

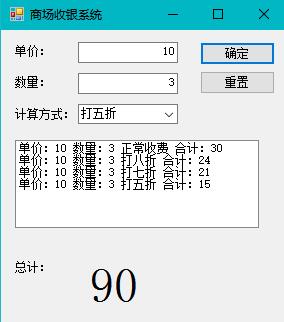
total = total + totalPrices;

lbxList.Items.Add("单价：" + txtPrice.Text + " 数量：" + txtNum.Text

+ " "+cbxType.SelectedItem+ " 合计：" + totalPrices.ToString());

lblResult.Text = total.ToString();  
　　　　　}

[复制代码](javascript:void(0);)



小菜：这样就可以了吧。

大鸟：比之前灵活了些，但是问题也不少：首先Convert.ToDouble()在这里就写了8遍，另外商场决定加大活动力度，满300返100这样的促销算法，你觉得该怎么办？

小菜：那意思就是满300返100，700的话就返200了？写函数就可以了吧？

大鸟：看来之前的简单工厂模式是白学了。

小菜：哦哦哦，这样子呐。那就先写一个父类，在继承它实现多个打折和返利的子类，使用多态，是这样子吧？

大鸟：这样你准备写多少个子类？

小菜：根据商店需求呀，比如9折，8折，6折，满300返100，满400返180......要多少写多少O(∩\_∩)O~~

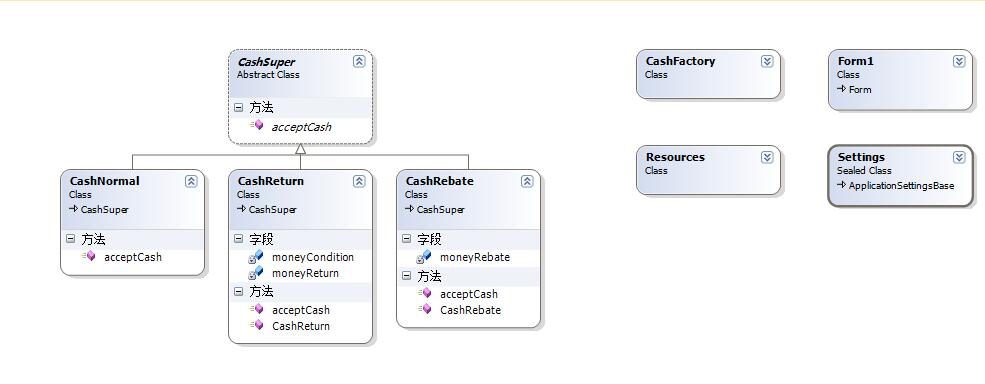
大鸟：小菜没有认真考虑这些促销活动之间的相同点和不同点呐

小菜：(⊙o⊙)哦，好像是有相同的地方，可以分为打折促销，返利促销，正常收费三个模式。这样可以写一个**父类，包含抽象方法：收钱()，然后写三个子类（正常收费，返利收费，打折收费）**分别实现这个收钱的函数，然后使用简单工厂模式，创建**一个收费工厂类**，在里面进行收费的选择，创建不同的收费实例。

大鸟：**面向对象的编程，并不是类越多越好，类的划分是为了封装，但分类的基础是抽象，具有相同属性和功能的对象的抽象集合才是类。**

商场收银系统v1.1运行截图以及关键代码

**代码结构图：**



**现金收费子类：**

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Text;

namespace 商场管理软件

{

//现金收取父类

abstract class CashSuper

{

//抽象方法：收取现金，参数为原价，返回为当前价

public abstract double acceptCash(double money);

}

}

[复制代码](javascript:void(0);)

**正常收费子类：**

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Text;

namespace 商场管理软件

{

//正常收费，继承CashSuper

class CashNormal : CashSuper

{

public override double acceptCash(double money)

{

return money;

}

}

}

[复制代码](javascript:void(0);)

**打折收费子类：**

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Text;

namespace 商场管理软件

{

//打折收费，继承CashSuper

class CashRebate : CashSuper

{

private double moneyRebate = 1d;

//初始化时，必需要输入折扣率,如八折，就是0.8

public CashRebate(string moneyRebate)

{

this.moneyRebate = double.Parse(moneyRebate);

}

public override double acceptCash(double money)

{

return money \* moneyRebate;

}

}

}

[复制代码](javascript:void(0);)

**返利收费子类：**

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Text;

namespace 商场管理软件

{

//返利收费，继承CashSuper

class CashReturn : CashSuper

{

private double moneyCondition = 0.0d;

private double moneyReturn = 0.0d;

//初始化时必须要输入返利条件和返利值，比如满300返100，则moneyCondition为300，moneyReturn为100

public CashReturn(string moneyCondition, string moneyReturn)

{

this.moneyCondition = double.Parse(moneyCondition);

this.moneyReturn = double.Parse(moneyReturn);

}

public override double acceptCash(double money)

{

double result = money;

//若大于返利条件，则需要减去返利值

if (money >= moneyCondition)

result = money - Math.Floor(money / moneyCondition) \* moneyReturn;

return result;

}

}

}

[复制代码](javascript:void(0);)

**现金收费工厂类：**

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Text;

namespace 商场管理软件

{

//现金收取工厂

class CashFactory

{

//根据条件返回相应的对象

public static CashSuper createCashAccept(string type)

{

CashSuper cs = null;

switch (type)

{

case "正常收费":

cs = new CashNormal();

break;

case "满300返100":

CashReturn cr1 = new CashReturn("300", "100");

cs = cr1;

break;

case "打8折":

CashRebate cr2 = new CashRebate("0.8");

cs = cr2;

break;

}

return cs;

}

}

}

[复制代码](javascript:void(0);)

**客户端程序主要部分：**

[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace 商场管理软件

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

//客户端窗体程序（主要部分）

double total = 0.0d;

private void btnOk\_Click(object sender, EventArgs e)

{

//利用简单工厂模式根据下拉选择框，生成相应的对象

CashSuper csuper = CashFactory.createCashAccept(cbxType.SelectedItem.ToString());

double totalPrices = 0d;

//通过多态，可以得到收取费用的结果

totalPrices = csuper.acceptCash(Convert.ToDouble(txtPrice.Text) \* Convert.ToDouble(txtNum.Text));

total = total + totalPrices;

lbxList.Items.Add("单价：" + txtPrice.Text + " 数量：" + txtNum.Text + " "

+ cbxType.SelectedItem + " 合计：" + totalPrices.ToString());

lblResult.Text = total.ToString();

}

private void btnClear\_Click(object sender, EventArgs e)

{

total = 0d;

txtPrice.Text = "0.00";

txtNum.Text = "0";

lbxList.Items.Clear();

lblResult.Text = "0.00";

}

}

}

[复制代码](javascript:void(0);)



小菜：大鸟，这样不管怎样修改，我都可以简单处理了。

大鸟：那我要实现打五折和满500返200的返利活动，你怎么修改？

小菜：在现金工厂中添加打五折和满500返200的case语句，然后在下拉选择框中添加两个选项就可以了。

大鸟：现金工厂？应该是收费对象生成工厂才准确。但是如果促销修改为满100积分加10，当积分达到一定时候就可以领取奖品怎么做？

小菜：有了工厂，何难？添加一个积分算法，构造方法里面有两个参数：条件和返点，继承CashSuper，然后在现金工厂，哦，不对，是收费对象生成工厂里面增加满100积分加10的分支条件，然后在下拉选择框里面添加这个选项就可以了。

大鸟：但是在以后的学习中我们会发现这样的设计模式并不是最好的选择，因为每次维护或者扩展都要修改工厂这个类，以至于代码需要重新编译部署，这样的处理很糟糕，自己去研究一下看看那个设计模式可以用在其中。

小菜（陷入沉思......）

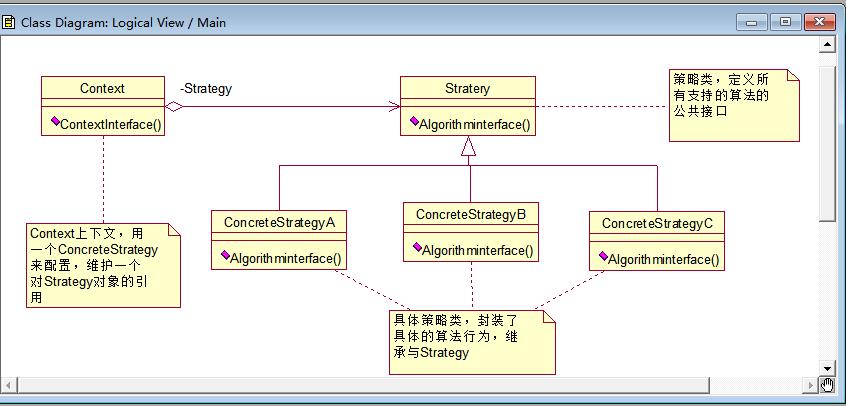
**次日**

小菜：大鸟我知道该使用哪种设计模式了，选择策略模式就很好地解决了这个问题。

**策略模式(Strategy):它定义了算法家族，分别封装起来，让它们之间可以相互替换，此模式让算法的变化，不会影响到使用算法的客户。**

大鸟：你说说看。

小菜：这里是我画的策略模式结构图





[复制代码](javascript:void(0);)

using System;

using System.Collections.Generic;

using System.Text;

namespace 策略模式

{

class Program

{

static void Main(string[] args)

{

Context context;

context = new Context(new ConcreteStrategyA());

context.ContextInterface();

context = new Context(new ConcreteStrategyB());

context.ContextInterface();

context = new Context(new ConcreteStrategyC());

context.ContextInterface();

Console.Read();

}

}

//抽象算法类

abstract class Strategy

{

//算法方法

public abstract void AlgorithmInterface();

}

//具体算法A

class ConcreteStrategyA : Strategy

{

//算法A实现方法

public override void AlgorithmInterface()

{

Console.WriteLine("算法A实现");

}

}

//具体算法B

class ConcreteStrategyB : Strategy

{

//算法B实现方法

public override void AlgorithmInterface()

{

Console.WriteLine("算法B实现");

}

}

//具体算法C

class ConcreteStrategyC : Strategy

{

//算法C实现方法

public override void AlgorithmInterface()

{

Console.WriteLine("算法C实现");

}

}

//上下文

class Context

{

Strategy strategy;

public Context(Strategy strategy)

{

this.strategy = strategy;

}

//上下文接口

public void ContextInterface()

{

strategy.AlgorithmInterface();

}

}

}

[复制代码](javascript:void(0);)