Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГТУ»)**

**Институт компьютерных систем и информационной безопасности**

**Кафедра информационных систем и программирования**

**ОТЧЁТ**

**Дисциплина: Тестирование и отладка программного обеспечения**

**Тема: «Рефакторинг и расширение**

**функциональности с использованием поведенческих паттернов проектирования»**

**Лабораторная: 8**

Работу выполнил: Басма Нассиф Хуссейн

Направление подготовки: 09.03.04 Программная инженерия

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. В. Степанова

**Цель работы:** изучить техники рефакторинга с использованием поведенческих паттернов проектирования.

**Контрольные вопросы**

1. **Как выполняется свертывание иерархии классов?**
2. **Для чего используется паттерн Strategy?**
3. **Опишите преимущества использования паттерна «стратегия» при необходимости динамического изменения поведения?**

**Задание**

**1) Изменить процесс назначения бонусов и скидок в соответствии с предложенными в пункте 2.2 изменениями, применив описанные в теории методы рефакторинга.**

1. Создать новые классы для реализации паттерна стратегия для бонусов и реализовать их в соответствии с документацией.

public interface IBonusStrategy

{

int CalculateBonus(int totalAmount, int bonusTotalAmount);

}

public class DefaultBonusStrategy : IBonusStrategy

{

// without bonus

public int CalculateBonus(int totalAmount, int bonusTotalAmount)

{

return bonusTotalAmount;

}

}

public class FirstBonusStrategy : IBonusStrategy

{

// with bonus

public int CalculateBonus(int totalAmount, int bonusTotalAmount)

{

double bonus = 0;

if (totalAmount >= 5000)

{

bonus = totalAmount \* 0.07;

return (int)(bonus + bonusTotalAmount);

}

return bonusTotalAmount;

}

}

1. Создать новые классы для реализации паттерна стратегия для скидок и реализовать их в соответствии с документацией.

public interface IDiscountStrategy

{

double CalculateDiscount(int totalAmount, int discountTotalAmount);

}

public class DefaultDiscountStrategy : IDiscountStrategy

{

// No discount

public double CalculateDiscount(int totalAmount, int discountTotalAmount)

{

return discountTotalAmount;

}

}

public class FirstDiscountStrategy : IDiscountStrategy

{

// 3% discount for more than 2000 units

public double CalculateDiscount(int totalAmount, int discountTotalAmount)

{

double discount = 0;

if (totalAmount >= 2000)

{

discount = totalAmount \* 0.03;

return discount + discountTotalAmount;

}

return discountTotalAmount;

}

}

public class SecondDiscountStrategy : IDiscountStrategy

{

// 5% discount for more than 3000 units

public double CalculateDiscount(int totalAmount, int discountTotalAmount)

{

double discount = 0;

if (totalAmount >= 3000)

{

discount = totalAmount \* 0.05;

return discount + discountTotalAmount;

}

return discountTotalAmount;

}

}

1. Внедрить в класс Goods поля типов BonusStrategy и DiscountStrategy и добавить возможность их передачи через конструктора.

protected IBonusStrategy \_bonusStrategy;

protected IDiscountStrategy \_discountStrategy;

public Goods(String title)

{

\_title = title;

}

1. Изменить процесс определения скидки и бонуса путем вызова соответствующих методов классов BonusStrategy и DiscountStrategy.

public void SetStrategy(int totalAmount)

{

// automatically chooses the needed strategy for discount

if (totalAmount >= 2000 && totalAmount < 3000)

{

this.\_discountStrategy = new FirstDiscountStrategy();

}

else if (totalAmount >= 3000)

{

this.\_discountStrategy = new SecondDiscountStrategy();

}

else this.\_discountStrategy = new DefaultDiscountStrategy();

// automatically chooses the needed strategy for bonus

if (totalAmount >= 5000)

{

this.\_bonusStrategy = new FirstBonusStrategy();

}

else this.\_bonusStrategy = new DefaultBonusStrategy();

}

1. Добавить тесты для проверки новых вариантов подсчета скидок и бонусов (На этом этапе воспользуемся подходом TDD (разработка через тестирование): сначала напишем тесты, а потом код).

[Test]

public void FirstBonusStrategy()

{

var goods = new RegularGoods("Test");

int quantity = 100;

double price = 50;

int totalAmount = (int)(quantity \* price);

goods.SetStrategy(totalAmount);

int bonus = goods.GetBonus(quantity, price);

Assert.That(bonus, Is.EqualTo(600));

}

[Test]

public void DefaultBonusStrategy()

{

var goods = new SaleGoods("Test");

int quantity = 100;

double price = 40;

int totalAmount = (int)(quantity \* price);

goods.SetStrategy(totalAmount);

int bonus = goods.GetBonus(quantity, price);

Assert.That(bonus, Is.EqualTo(40));

}

[Test]

public void DefaultDiscountStrategyTest()

{

var goods = new RegularGoods("Test");

int quantity = 100;

double price = 10;

int totalAmount = (int)(quantity \* price);

goods.SetStrategy(totalAmount);

int discount = (int)goods.GetDiscount(quantity, price);

Assert.That(discount, Is.EqualTo(30));

}

[Test]

public void FirstDiscountStrategyTest()

{

var goods = new SaleGoods("Test");

int quantity = 100;

double price = 20;

int totalAmount = (int)(quantity \* price);

goods.SetStrategy(totalAmount);

int discount = (int)goods.GetDiscount(quantity, price);

Assert.That(discount, Is.EqualTo(80));

}

[Test]

public void SecondDiscountStrategyTest()

{

var goods = new SpecialGoods("Test");

int quantity = 100;

double price = 40;

int totalAmount = (int)(quantity \* price);

goods.SetStrategy(totalAmount);

int discount = (int)goods.GetDiscount(quantity, price);

Assert.That(discount, Is.EqualTo(220));

}

1. Изменить процесс подсчёта бонусов и скидок, отделив его от основанного цикла подсчета суммы, согласовав его с написанными ранее тестами.

// RegularGoods

public class RegularGoods : Goods

{

public RegularGoods(string title) : base(title)

{

}

public override int GetBonus(int \_quantity, double \_price)

{

int totalAmount = (int)(\_quantity \* \_price);

int bonusTotalAmount = (int)(\_quantity \* \_price \* 0.05);

return \_bonusStrategy.CalculateBonus(totalAmount, bonusTotalAmount);

}

// Метод для получения скидки

public override double GetDiscount(int \_quantity, double \_price)

{

int totalAmount = (int)(\_quantity \* \_price);

int discountTotalAmount = (int)(\_quantity \* \_price);

if (\_quantity > 2)

{

discountTotalAmount = (int)(\_quantity \* \_price \* 0.03); // 3%

return \_discountStrategy.CalculateDiscount(totalAmount, discountTotalAmount);

}

return \_discountStrategy.CalculateDiscount(totalAmount, discountTotalAmount);

}

}

// SaleGoods

public class SaleGoods : Goods

{

public SaleGoods(string title) : base(title)

{

}

// Метод для получения бонуса

public override int GetBonus(int \_quantity, double \_price)

{

int totalAmount = (int)(\_quantity \* \_price);

int bonusTotalAmount = (int)(\_quantity \* \_price \* 0.01);

return \_bonusStrategy.CalculateBonus(totalAmount, bonusTotalAmount);

}

// Метод для получения скидки

public override double GetDiscount(int \_quantity, double \_price)

{

int totalAmount = (int)(\_quantity \* \_price);

int discountTotalAmount = (int)(\_quantity \* \_price);

if (\_quantity > 3)

{

discountTotalAmount = (int)(\_quantity \* \_price \* 0.01); // 0.1%

return \_discountStrategy.CalculateDiscount(totalAmount, discountTotalAmount);

}

return \_discountStrategy.CalculateDiscount(totalAmount, discountTotalAmount);

}

}

// SpecialGoods

public class SpecialGoods : Goods

{

public SpecialGoods(string title) : base(title)

{

}

public override int GetBonus(int \_quantity, double \_price)

{

int totalAmount = (int)(\_quantity \* \_price);

int bonusTotalAmount = (int)(\_quantity \* \_price);

return \_bonusStrategy.CalculateBonus(totalAmount, bonusTotalAmount);

}

// Метод для получения скидки

public override double GetDiscount(int \_quantity, double \_price)

{

int totalAmount = (int)(\_quantity \* \_price);

int discountTotalAmount = (int)(\_quantity \* \_price);

if (\_quantity > 10)

{

discountTotalAmount = (int)(\_quantity \* \_price \* 0.005);

return \_discountStrategy.CalculateDiscount(totalAmount, discountTotalAmount);

}

return \_discountStrategy.CalculateDiscount(totalAmount, discountTotalAmount);

}

}

1. Добавить поддержку файла конфигурации и описание стратегий в нем (формат описания раздела стратегий выбрать самостоятельно).
2. Изменить процесс создания экземпляра класса Bill и выполнять создание экземпляров классов стратегий с чтением значений процентов скидок из конфигурационного файла.
3. При желании можно свернуть иерархию классов товаров, однако это потребует множества изменений в первоначальных тестах. Данные тесты оказались хрупкими, хотя в самом коде программы потребуется изменить только одну часть в классе BillBuilder, т.е. полученный код оказался связным и достаточно гибким.

**2) Проверить согласованность результатов и корректность работы нового варианта программы с результатами старой версии.**

**До:**

**После:**

**Заключение:**

Были изучены техники рефакторинга с использованием поведенческих паттернов проектирования.