Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Кубанский Государственный Технологический Университет»

Кафедра информационных систем и программирования

Отчет по лабораторной работе №8

По дисциплине «Рефакторинг и работа с унаследованным кодом»

«Рефакторинг и расширение функциональности с использованием поведенческих паттернов проектирования»

Выполнил студент

Группы 21-КБ-ПР2

Жулькин Д.П.

Преподаватель:

Степанова Елизавета Владимировна

Краснодар

2024

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

Задание:

1. Изменить процесс назначения бонусов и скидок в соответствии с предложенными в пункте 2.2 изменениями, применив описанные в теории методы рефакторинга.
2. Проверить согласованность результатов и корректность работы нового варианта программы с результатами старой версии.
3. Оформить отчёт.

Процесс выполнения работы:

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

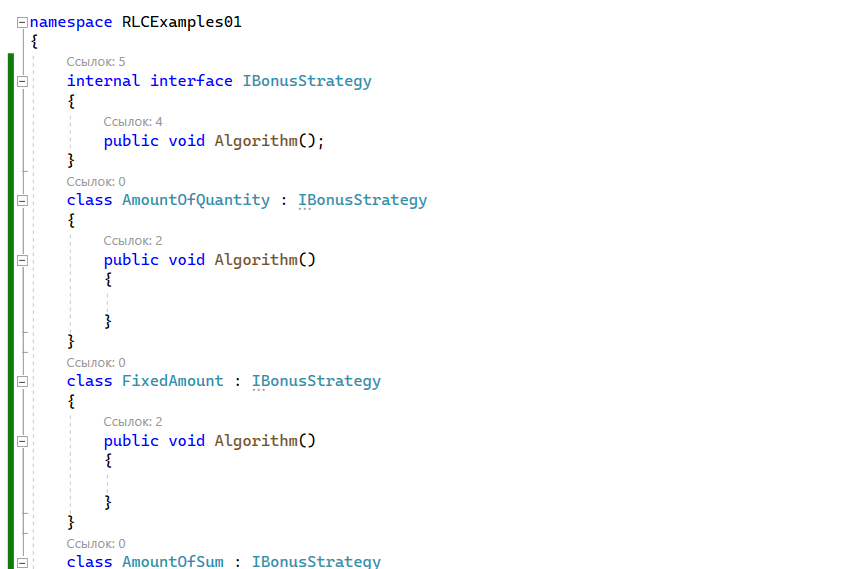


Рисунок 1 – Добавление стратегии с бонусами

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

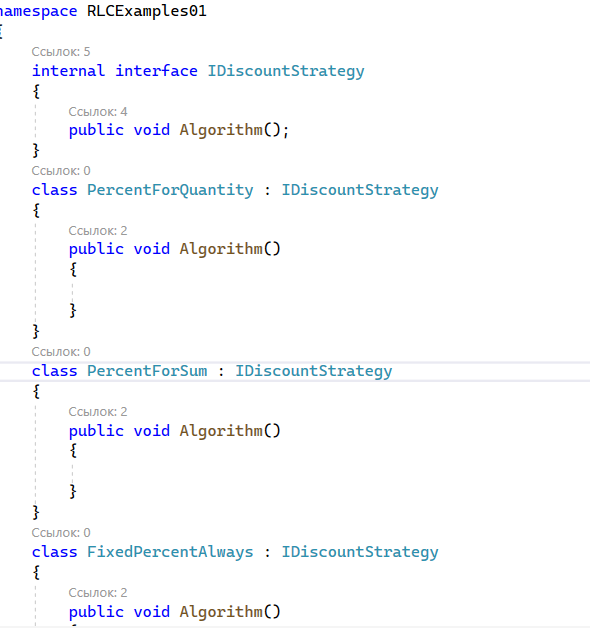


Рисунок 2 – Добавление стратегии со скидками

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

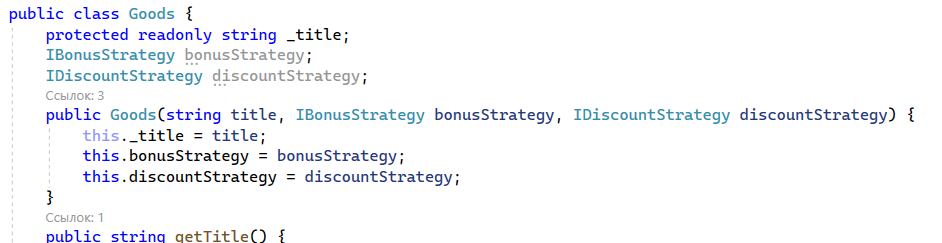


Рисунок 3 – Внедрение новых классов в Goods

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

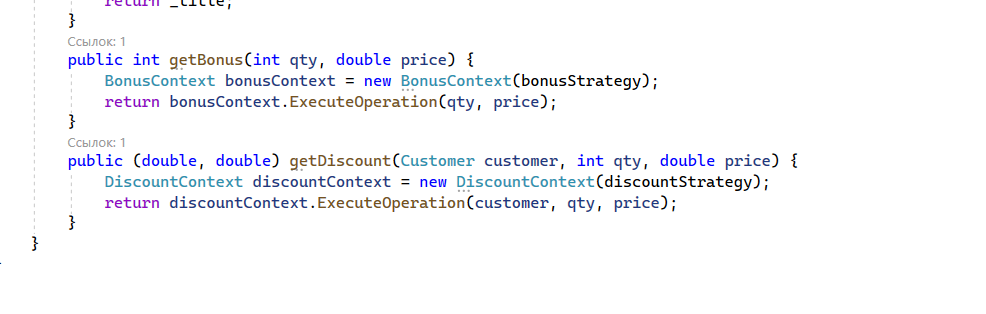


Рисунок 4 – Вызов BonusStrategy и DiscountStrategy

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).

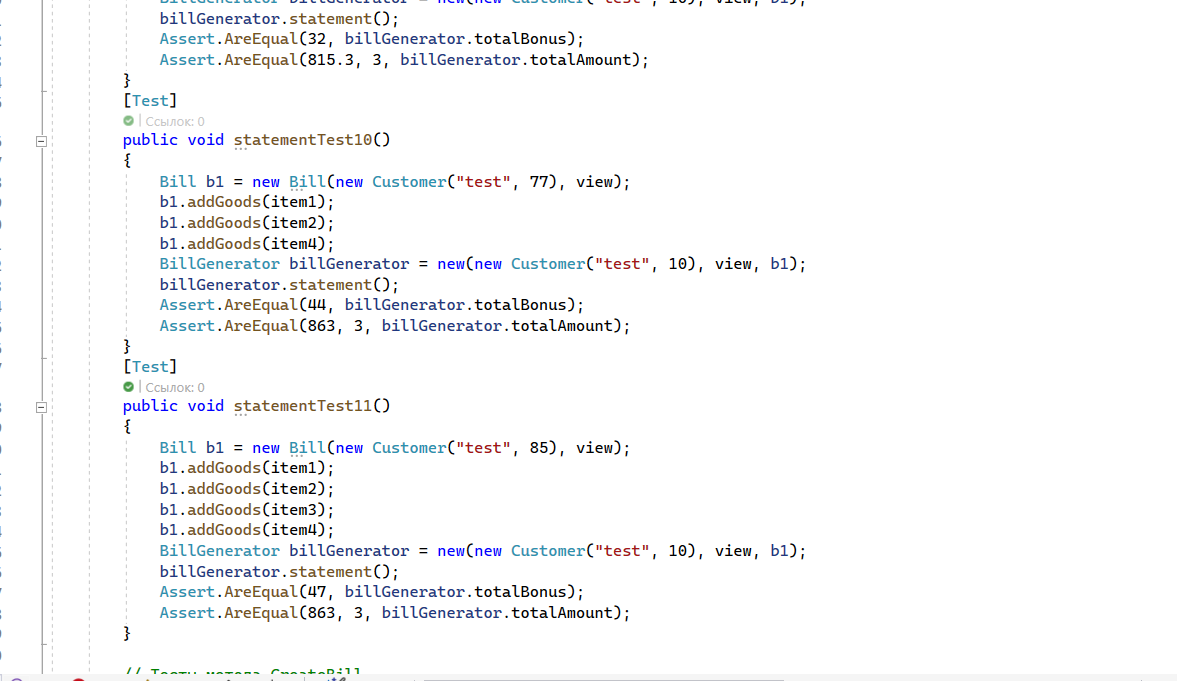


Рисунок 5 – Создание новых тестов

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

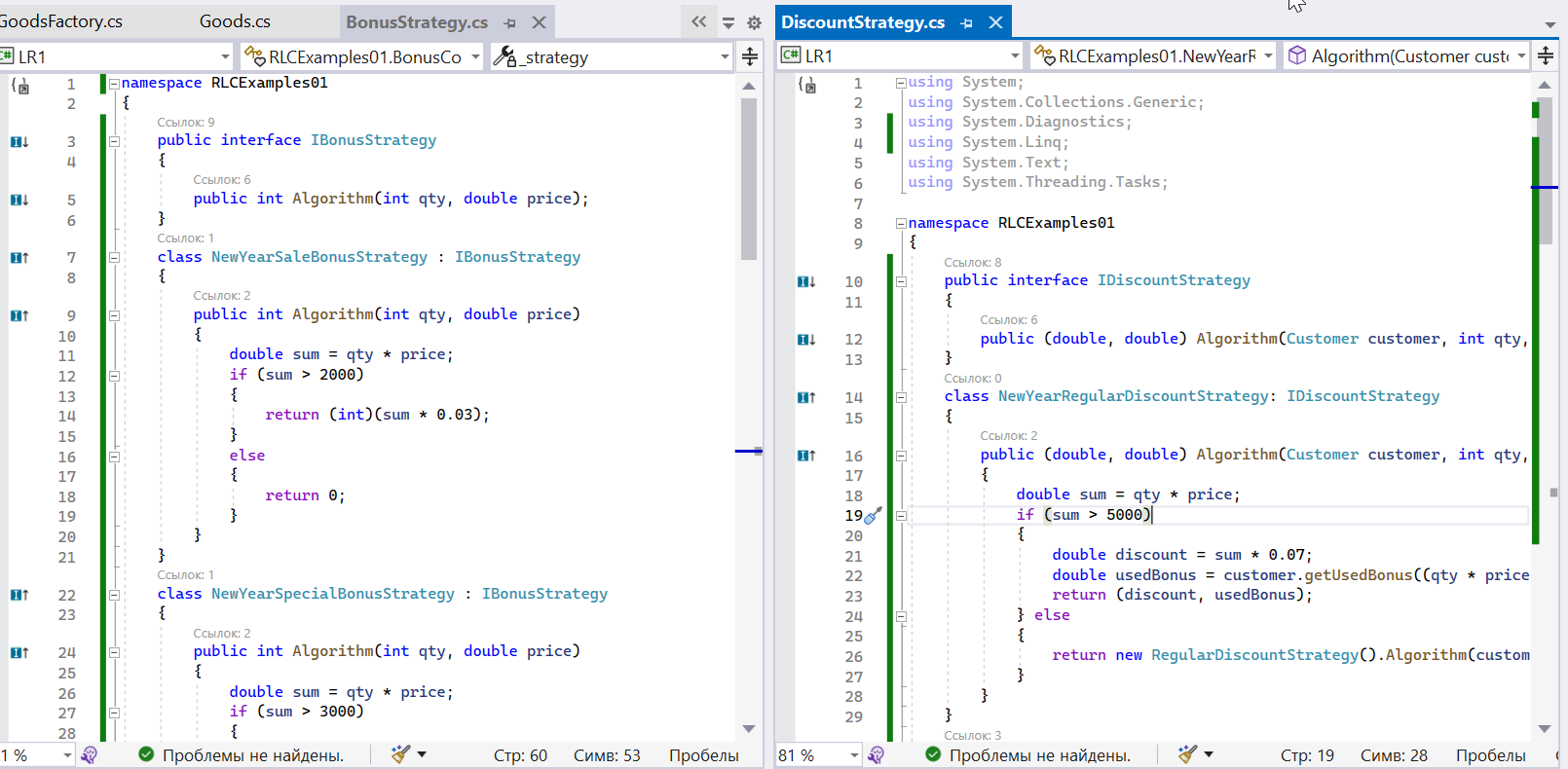


Рисунок 6 – Новый подсчет бонусов и скидок

1. Добавить поддержку файла конфигурации и описание стратегий в нем (формат описания раздела стратегий выбрать самостоятельно).

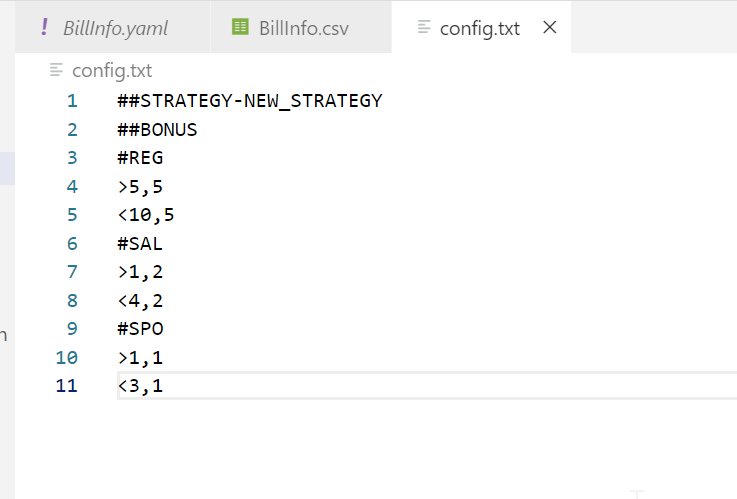


Рисунок 7 – Новый файл конфигурации

1. Изменить процесс создания экземпляра класса Bill и выполнять создание экземпляров классов стратегий с чтением значений процентов скидок из конфигурационного файла.

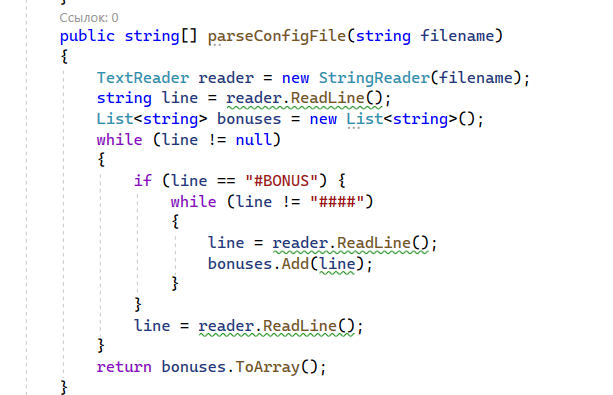


Рисунок 8 – Чтение конфигурационного файла

Вывод: задания выполнены успешно, освоены продвинутые практики рефакторинга.

Листинг:

using RLCExamples01;

namespace LR1

{

public class GoodsFactory

{

static public Goods? Create(string id, string name)

{

if (false) // NEW YEAR

{

if (id == "REG") { return new Goods(name, new RegularBonusStrategy(), new RegularDiscountStrategy()); }

if (id == "SAL") { return new Goods(name, new NewYearSaleBonusStrategy(), new SaleDiscountStrategy()); }

if (id == "SPO") { return new Goods(name, new NewYearSpecialBonusStrategy(), new SpecialDiscountStrategy()); }

}

if (id == "REG") { return new Goods(name, new RegularBonusStrategy(), new RegularDiscountStrategy()); }

if (id == "SAL") { return new Goods(name, new SaleBonusStrategy(), new SaleDiscountStrategy()); }

if (id == "SPO") { return new Goods(name, new SpecialBonusStrategy(), new SpecialDiscountStrategy()); }

return null;

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RLCExamples01

{

public interface IDiscountStrategy

{

public (double, double) Algorithm(Customer customer, int qty, double price);

}

class NewYearRegularDiscountStrategy: IDiscountStrategy

{

public (double, double) Algorithm(Customer customer, int qty, double price)

{

double sum = qty \* price;

if (sum > 5000)

{

double discount = sum \* 0.07;

double usedBonus = customer.getUsedBonus((qty \* price) - discount);

return (discount, usedBonus);

} else

{

return new RegularDiscountStrategy().Algorithm(customer, qty, price);

}

}

}

class RegularDiscountStrategy : IDiscountStrategy

{

public (double, double) Algorithm(Customer customer, int qty, double price)

{

double discount = 0;

double usedBonus = 0;

if (qty > 2)

{

discount = qty \* price \* 0.03;

}

if (qty > 5)

{

usedBonus = customer.getUsedBonus((qty \* price) - discount);

}

return (discount, usedBonus);

}

}

class SaleDiscountStrategy : IDiscountStrategy

{

public (double, double) Algorithm(Customer customer, int qty, double price)

{

double discount = 0;

double usedBonus = 0;

if (qty > 3)

{

discount = qty \* price \* 0.01;

}

return (discount, usedBonus);

}

}

class SpecialDiscountStrategy : IDiscountStrategy

{

public (double, double) Algorithm(Customer customer, int qty, double price)

{

double discount = 0;

double usedBonus = 0;

if (qty > 10)

{

discount = qty \* price \* 0.005;

}

if (qty > 1)

{

usedBonus = customer.getUsedBonus((qty \* price) - discount);

}

return (discount, usedBonus);

}

}

class DiscountContext

{

private IDiscountStrategy \_strategy { get; set; }

public DiscountContext(IDiscountStrategy strategy)

{

\_strategy = strategy;

}

public (double, double) ExecuteOperation(Customer customer, int qty, double price)

{

return \_strategy.Algorithm(customer, qty, price);

}

}

}

namespace RLCExamples01

{

public interface IBonusStrategy

{

public int Algorithm(int qty, double price);

}

class NewYearSaleBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

double sum = qty \* price;

if (sum > 2000)

{

return (int)(sum \* 0.03);

}

else

{

return 0;

}

}

}

class NewYearSpecialBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

double sum = qty \* price;

if (sum > 3000)

{

return (int)(sum \* 0.05);

}

else

{

return 0;

}

}

}

class RegularBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

return (int)(qty \* price \* 0.05);

}

}

class SaleBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

return (int)(qty \* price \* 0.01);

}

}

class SpecialBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

return 0;

}

}

class BonusContext

{

private IBonusStrategy \_strategy { get; set; }

public BonusContext(IBonusStrategy strategy)

{

\_strategy = strategy;

}

public int ExecuteOperation(int qty, double price)

{

return \_strategy.Algorithm(qty, price);

}

}

}

namespace RLCExamples01

{

public interface IBonusStrategy

{

public int Algorithm(int qty, double price);

}

class NewYearSaleBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

double sum = qty \* price;

if (sum > 2000)

{

return (int)(sum \* 0.03);

}

else

{

return 0;

}

}

}

class NewYearSpecialBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

double sum = qty \* price;

if (sum > 3000)

{

return (int)(sum \* 0.05);

}

else

{

return 0;

}

}

}

class RegularBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

return (int)(qty \* price \* 0.05);

}

}

class SaleBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

return (int)(qty \* price \* 0.01);

}

}

class SpecialBonusStrategy : IBonusStrategy

{

public int Algorithm(int qty, double price)

{

return 0;

}

}

class BonusContext

{

private IBonusStrategy \_strategy { get; set; }

public BonusContext(IBonusStrategy strategy)

{

\_strategy = strategy;

}

public int ExecuteOperation(int qty, double price)

{

return \_strategy.Algorithm(qty, price);

}

}

}