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?**

Путем устранения лишних уровней наследования, объединяя функциональность подклассов и суперклассов в один класс, или удаляя неиспользуемые классы. Это делается для упрощения структуры кода, уменьшения сложности и улучшения читаемости. Процесс может включать в себя перемещение методов и полей вверх или вниз по иерархии, удаление классов, которые не добавляют уникального поведения, или замену наследования делегированием.

1. **Для чего используется паттерн Strategy?**

Для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм независимо от объектов-клиентов, которые его используют. Паттерн Strategy применяется, когда есть несколько родственных классов, которые отличаются только поведением, или когда необходимо изменять поведение объектов на стадии выполнения, или же когда в коде присутствует множество условных операторов, выбор которых зависит от одного и того же типа условий.

1. **Опишите преимущества использования паттерна «стратегия» при необходимости динамического изменения поведения?**

• Гибкость и расширяемость: Легко добавлять новые стратегии и изменять существующие без изменения контекста, который их использует.

• Уменьшение условных операторов: Код становится более чистым и легче для понимания, так как избавляется от множества условных конструкций за счет использования полиморфизма.

• Инкапсуляция алгоритмов: Каждый алгоритм заключается в свой собственный класс, что улучшает модульность и помогает изолировать код, связанный с выполнением определенных задач.

• Взаимозаменяемость поведения: Позволяет динамически изменять алгоритмы поведения объектов во время выполнения программы, что делает систему более гибкой.

• Принцип открытости/закрытости: Система остается открытой для расширения (можно легко добавлять новые стратегии), но закрытой для изменений (не требуется изменять существующий код при добавлении новых алгоритмов).

**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.**

// interface for bonus strategy

public interface IBonusStrategy

{

int CalculateBonus(int quantity, double price);

}

// bonus strategy for regular goods

public class RegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

if (quantity > 2)

return (int)(quantity \* price \* 0.05); // 5%

return 0;

}

}

// bonus strategy for sale goods

public class SaleBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

return (int)(quantity \* price \* 0.01); // 1%

}

}

// bonus strategy for special goods

public class SpecialBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

return 0;

}

}

// bonus strategy for new year regular goods

public class NewYearRegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

var total = quantity \* price;

if (total > 5000)

return (int)(total \* 0.07); // 7%

if (quantity > 2)

return (int)(quantity \* price \* 0.05); // 5%

return 0;

}

}

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

// interface for discount strategy

public interface IDiscountStrategy

{

double CalculateDiscount(int quantity, double price);

}

// discount strategy for regular goods

public class RegularDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

if (quantity > 2)

return (quantity \* price) \* 0.03; // 3%

return 0;

}

}

// discount strategy for sale goods

public class SaleDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

if (quantity > 3)

return (quantity \* price) \* 0.01; // 0.1%

return 0;

}

}

// discount strategy for regular goods

public class SpecialDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

if (quantity > 10)

return (quantity \* price) \* 0.005; // 0.5% скидка

return 0;

}

}

// discount strategy for new year sale goods

public class NewYearSaleDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

var total = quantity \* price;

if (total > 2000)

return total \* 0.03; // 3%

if (quantity > 3)

return (quantity \* price) \* 0.01; // 0.1%

return 0;

}

}

// discount strategy for new year special goods

public class NewYearSpecialDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

var total = quantity \* price;

if (total > 3000)

return total \* 0.05; // 5%

return 0;

}

}

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

public class Goods

{

protected String title;

protected int \_priceCode;

protected IBonusStrategy bonusStrategy;

protected IDiscountStrategy discountStrategy;

public Goods(string title,

IBonusStrategy bonusStrategy,

IDiscountStrategy discountStrategy)

{

this.title = title;

this.bonusStrategy = bonusStrategy;

this.discountStrategy = discountStrategy;

}

}

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

// method to get bonus

public virtual int GetBonus(int quantity, double price)

{

return bonusStrategy.CalculateBonus(quantity, price);

}

// method to get discount

public virtual double GetDiscount(int quantity, double price)

{

return discountStrategy.CalculateDiscount(quantity, price);

}

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

public class BillFactory

{

private IFileSource readContent;

public BillFactory(IFileSource readContent)

{

this.readContent = readContent;

}

// Method to convert info from config file

public BillGenerator CreateBill(TextReader sr, string config = "RegularSettings.json", IView view = null)

{

if (view == null)

{

view = new TxtView();

}

int strategyType;

var configPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, config);

if (!File.Exists(configPath))

{

throw new FileNotFoundException("Конфигурационный файл не найден.", configPath);

}

string configJson = File.ReadAllText(configPath);

var settings = JsonSerializer.Deserialize<ConfigSettings>(configJson);

if (settings.Season == "NewYears")

{

strategyType = 1;

}

else

{

strategyType = 0;

}

readContent.SetSource(sr, strategyType);

// Чтение покупателя

Customer customer = readContent.GetCustomer();

BillGenerator b = new BillGenerator(customer, view);

//Чтение количества продуктов

int goodsQty = readContent.GetGoodsCount();

Goods[] g = new Goods[goodsQty];

//Чтение каждого продукта

for (int i = 0; i < g.Length; i++)

{

g[i] = readContent.GetNextGood();

}

//Чтение количества товаров

int itemsQty = readContent.GetItemsCount();

//Чтение каждого товара

for (int i = 0; i < itemsQty; i++)

{

b.addGoods(readContent.GetNextItem(g));

}

return b;

}

// Class for configuration

public class ConfigSettings

{

public string Season { get; set; }

}

}

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

public class GoodsFactory

{

public Goods Create(string type, string title, int strategyType)

{

IBonusStrategy bonusStrategy;

IDiscountStrategy discountStrategy;

switch (strategyType)

{

case 0: // default strategies

switch (type)

{

case "REG":

bonusStrategy = new RegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy();

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy();

discountStrategy = new SaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy();

discountStrategy = new SpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

case 1: // new year strategies

switch (type)

{

case "REG":

bonusStrategy = new NewYearRegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy(); // Предполагается, что для RegularGoods скидка остаётся такой же как в обычный период

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy(); // Бонусы для товаров со скидкой остаются неизменными даже в новогодний период

discountStrategy = new NewYearSaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy(); // Предполагается, что для SpecialGoods бонусы не изменяются

discountStrategy = new NewYearSpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

default:

throw new ArgumentException("Неизвестный тип стратегии");

}

return new Goods(title, bonusStrategy, discountStrategy);

}

}

1. **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.**

Удаляем все наследники Goods

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

**До:**

public class RegularGoods : Goods

{

public RegularGoods(string title) : base(title)

{

}

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

{

return (int)(\_quantity \* \_price \* 0.05);

}

// Метод для получения скидки

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

{

if (\_quantity > 2)

return (\_quantity \* \_price) \* 0.03; // 3%

return 0;

}

}

public class SaleGoods : Goods

{

public SaleGoods(string title) : base(title)

{

}

// Метод для получения бонуса

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

{

return (int)(\_quantity \* \_price \* 0.01);

}

// Метод для получения скидки

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

{

if (\_quantity > 3)

return (\_quantity \* \_price) \* 0.01; // 0.1%

return 0;

}

}

public class SpecialGoods : Goods

{

public SpecialGoods(string title) : base(title)

{

}

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

{

return 0;

}

// Метод для получения скидки

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

{

if (\_quantity > 10)

return (\_quantity \* \_price) \* 0.005; // 0.5%

return 0;

}

}

public class BillFactory

{

private IFileSource contentFile;

// Параметризованный конструктор, принимающий IFileSource

public BillFactory(IFileSource contentFile)

{

this.contentFile = contentFile;

}

// Метод для преобразования данных из файла

public BillGenerator CreateBill(TextReader sr)

{

contentFile.SetSource(sr);

Customer customer = contentFile.GetCustomer();

IView view = new TxtView();

BillGenerator b = new BillGenerator(customer, view);

// read goods count

int goodsQty = contentFile.GetGoodsCount();

Goods[] g = new Goods[goodsQty];

for (int i = 0; i < g.Length; i++)

{

g[i] = contentFile.GetNextGood();

}

// read items count

int itemsQty = contentFile.GetItemsCount();

for (int i = 0; i < itemsQty; i++)

{

b.addGoods(contentFile.GetNextItem(g));

}

return b;

}

}

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public IView View

{

get { return view; }

set { view = value; }

}

public Bill(Customer customer, IView view)

{

this.\_customer = customer;

this.\_items = new List<Item>();

this.view = view;

}

public void addGoods(Item arg)

{

\_items.Add(arg);

}

// Метод для вычисления суммы

public double GetSum(Item each)

{

return each.getQuantity() \* each.getPrice();

}

//---Метод получения использованных бонусов

public double GetUsedBonus(Item each, double sumWithDiscount)

{

double usedBonus = 0;

if (each.getGoods().GetType() == typeof(RegularGoods))

{

if (each.getQuantity() > 5)

usedBonus = \_customer.useBonus((int)(sumWithDiscount));

} else if (each.getGoods().GetType() == typeof(SpecialGoods))

{

if (each.getQuantity() > 1)

usedBonus = \_customer.useBonus((int)(sumWithDiscount));

}

return usedBonus;

}

public BillSummary Process()

{

BillSummary billSummary = new BillSummary();

List<Item>.Enumerator items = \_items.GetEnumerator();

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = each.GetDiscount();

int bonus = each.GetBonus();

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

ItemSummary itemSummary = new ItemSummary(

name: each.getGoods().getTitle(),

price: each.getPrice(),

quantity: each.getQuantity(),

sum: sumWithDiscount,

discount: discount,

bonus: bonus

);

billSummary.ItemSummaries.Add(itemSummary);

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

return billSummary;

}

}

public class Goods

{

protected String \_title;

protected int \_priceCode;

public Goods(String title)

{

\_title = title;

}

public void setPriceCode(int arg)

{

\_priceCode = arg;

}

public String getTitle()

{

return \_title;

}

// Метод для получения бонуса

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

{

return 0;

}

// Метод для получения скидки

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

{

return 0;

}

}

public class GoodsFactory

{

public Goods Create(string type, string title)

{

switch (type)

{

case "REG":

return new RegularGoods(title);

case "SAL":

return new SaleGoods(title);

case "SPO":

return new SpecialGoods(title);

default:

throw new ArgumentException("Неизвестный тип товара");

}

}

}

public class HtmlView : IView

{

// Метод для вывода оглавления

public string GetHeader(Customer \_customer)

{

throw new NotImplementedException();

}

// Метод для вывода результата

public string GetFooter(double totalAmount, int totalBonus)

{

throw new NotImplementedException();

}

// Метод для вывода списка товаров

public string GetItemString(Item each, double discount,

double totalAmount, int bonus)

{

throw new NotImplementedException();

}

}

public class YamlFileSource : IFileSource

{

private TextReader reader;

string line;

string[] result;

// get source

public void SetSource(TextReader reader)

{

this.reader = reader;

}

// read customer info

public Customer GetCustomer()

{

// read customer

GetNextLine();

result = line.Split(':');

string name = result[1].Trim();

// read bonus

GetNextLine();

result = line.Split(':');

int bonus = Convert.ToInt32(result[1].Trim());

return new Customer(name, bonus);

}

// returns goods count

public int GetGoodsCount()

{

GetNextLine();

result = line.Split(':');

return Convert.ToInt32(result[1].Trim());

}

// reads each product

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

GetNextLine();

result = line.Split(':');

result = result[1].Trim().Split();

string type = result[1].Trim();

return factory.Create(type, result[0]);

}

// reads the amount of products

public int GetItemsCount()

{

GetNextLine();

result = line.Split(':');

return Convert.ToInt32(result[1].Trim());

}

// reads each item

public Item GetNextItem(Goods[] g)

{

GetNextLine();

result = line.Split(':');

result = result[1].Trim().Split();

int gid = Convert.ToInt32(result[0].Trim());

double price = Convert.ToDouble(result[1].Trim());

int qty = Convert.ToInt32(result[2].Trim());

return new Item(g[gid - 1], qty, price);

}

// skip comments

public void GetNextLine()

{

do

{

line = reader.ReadLine();

}

while (line.StartsWith('#'));

}

}

public class HtmlFileSource : IFileSource

{

private TextReader reader;

private string line;

// выбор типа файла

public void SetSource(TextReader reader)

{

this.reader = reader;

}

// read customer info

public Customer GetCustomer()

{

string name = ExtractValue("customerName");

int bonus = Convert.ToInt32(ExtractValue("customerBonus"));

return new Customer(name, bonus);

}

// get goods count

public int GetGoodsCount()

{

return Convert.ToInt32(ExtractValue("goodsCount"));

}

// next product

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

string type = ExtractValue("goodType");

string name = ExtractValue("goodName");

return factory.Create(type, name);

}

// get items amount

public int GetItemsCount()

{

return Convert.ToInt32(ExtractValue("itemsCount"));

}

// get next item

public Item GetNextItem(Goods[] g)

{

int gid = Convert.ToInt32(ExtractValue("itemId"));

double price = Convert.ToDouble(ExtractValue("itemPrice"));

int qty = Convert.ToInt32(ExtractValue("itemQty"));

return new Item(g[gid - 1], qty, price);

}

// searches for lines with given first and last tags to get the text in between

private string ExtractValue(string tagName)

{

string startTag = $"<{tagName}>";

string endTag = $"</{tagName}>";

string value = "";

while ((line = reader.ReadLine()) != null)

{

if (line.Contains(startTag))

{

int startIndex = line.IndexOf(startTag) + startTag.Length;

int endIndex = line.IndexOf(endTag);

value = line.Substring(startIndex, endIndex - startIndex).Trim();

break;

}

}

return value;

}

public void GetNextLine()

{

// Этот метод не используется в текущей реализации, но должен быть определен

}

}

**После:**

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public IView View

{

get { return view; }

set { view = value; }

}

public Bill(Customer customer, IView view)

{

this.\_customer = customer;

this.\_items = new List<Item>();

this.view = view;

}

public void addGoods(Item arg)

{

\_items.Add(arg);

}

// Метод для вычисления суммы

public double GetSum(Item each)

{

return each.getQuantity() \* each.getPrice();

}

//---Метод получения использованных бонусов

public int GetUsedBonus(Item each, double sumWithDiscount)

{

int usedBonus = 0;

switch (each.getGoods().bonusStrategy.GetType())

{

//Обычный товав

case Type t when (t == typeof(RegularBonusStrategy) || t == typeof(NewYearRegularBonusStrategy)):

if (each.getQuantity() > 5)

usedBonus = \_customer.useBonus((int)(sumWithDiscount));

break;

//Специальное предложение

case Type t when t == typeof(SpecialBonusStrategy):

if (each.getQuantity() > 1)

usedBonus = \_customer.useBonus((int)(sumWithDiscount));

break;

}

return usedBonus;

}

public BillSummary Process()

{

BillSummary billSummary = new BillSummary();

List<Item>.Enumerator items = \_items.GetEnumerator();

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = each.GetDiscount();

int bonus = each.GetBonus();

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

ItemSummary itemSummary = new ItemSummary(

name: each.getGoods().getTitle(),

price: each.getPrice(),

quantity: each.getQuantity(),

sum: sumWithDiscount,

discount: discount,

bonus: bonus

);

billSummary.ItemSummaries.Add(itemSummary);

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

return billSummary;

}

}

public class BillFactory

{

private IFileSource readContent;

public BillFactory(IFileSource readContent)

{

this.readContent = readContent;

}

// Method to convert info from config file

public BillGenerator CreateBill(TextReader sr, string config = "RegularSettings.json", IView view = null)

{

if (view == null)

{

view = new TxtView();

}

int strategyType;

var configPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, config);

if (!File.Exists(configPath))

{

throw new FileNotFoundException("Конфигурационный файл не найден.", configPath);

}

string configJson = File.ReadAllText(configPath);

var settings = JsonSerializer.Deserialize<ConfigSettings>(configJson);

if (settings.Season == "NewYears")

{

strategyType = 1;

}

else

{

strategyType = 0;

}

readContent.SetSource(sr, strategyType);

// Чтение покупателя

Customer customer = readContent.GetCustomer();

BillGenerator b = new BillGenerator(customer, view);

//Чтение количества продуктов

int goodsQty = readContent.GetGoodsCount();

Goods[] g = new Goods[goodsQty];

//Чтение каждого продукта

for (int i = 0; i < g.Length; i++)

{

g[i] = readContent.GetNextGood();

}

//Чтение количества товаров

int itemsQty = readContent.GetItemsCount();

//Чтение каждого товара

for (int i = 0; i < itemsQty; i++)

{

b.addGoods(readContent.GetNextItem(g));

}

return b;

}

// Class for configuration

public class ConfigSettings

{

public string Season { get; set; }

}

}

public class Goods

{

protected String title;

protected int \_priceCode;

public IBonusStrategy bonusStrategy;

protected IDiscountStrategy discountStrategy;

public Goods(string title,

IBonusStrategy bonusStrategy,

IDiscountStrategy discountStrategy)

{

this.title = title;

this.bonusStrategy = bonusStrategy;

this.discountStrategy = discountStrategy;

}

public String getTitle()

{

return title;

}

// method to get bonus

public virtual int GetBonus(int quantity, double price)

{

return bonusStrategy.CalculateBonus(quantity, price);

}

// method to get discount

public virtual double GetDiscount(int quantity, double price)

{

return discountStrategy.CalculateDiscount(quantity, price);

}

}

public class HtmlFileSource : IFileSource

{

private TextReader reader;

private string line;

private int strategyType;

// выбор типа файла

public void SetSource(TextReader reader, int strategyType)

{

this.reader = reader;

this.strategyType = strategyType;

}

// read customer info

public Customer GetCustomer()

{

string name = ExtractValue("customerName");

int bonus = Convert.ToInt32(ExtractValue("customerBonus"));

return new Customer(name, bonus);

}

// get goods count

public int GetGoodsCount()

{

return Convert.ToInt32(ExtractValue("goodsCount"));

}

// next product

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

string type = ExtractValue("goodType");

string name = ExtractValue("goodName");

return factory.Create(type, name, strategyType);

}

// get items amount

public int GetItemsCount()

{

return Convert.ToInt32(ExtractValue("itemsCount"));

}

// get next item

public Item GetNextItem(Goods[] g)

{

int gid = Convert.ToInt32(ExtractValue("itemId"));

double price = Convert.ToDouble(ExtractValue("itemPrice"));

int qty = Convert.ToInt32(ExtractValue("itemQty"));

return new Item(g[gid - 1], qty, price);

}

// searches for lines with given first and last tags to get the text in between

private string ExtractValue(string tagName)

{

string startTag = $"<{tagName}>";

string endTag = $"</{tagName}>";

string value = "";

while ((line = reader.ReadLine()) != null)

{

if (line.Contains(startTag))

{

int startIndex = line.IndexOf(startTag) + startTag.Length;

int endIndex = line.IndexOf(endTag);

value = line.Substring(startIndex, endIndex - startIndex).Trim();

break;

}

}

return value;

}

public void GetNextLine()

{

// Этот метод не используется в текущей реализации, но должен быть определен

}

}

public class GoodsFactory

{

public Goods Create(string type, string title, int strategyType)

{

IBonusStrategy bonusStrategy;

IDiscountStrategy discountStrategy;

switch (strategyType)

{

case 0: // default strategies

switch (type)

{

case "REG":

bonusStrategy = new RegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy();

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy();

discountStrategy = new SaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy();

discountStrategy = new SpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

case 1: // new year strategies

switch (type)

{

case "REG":

bonusStrategy = new NewYearRegularBonusStrategy();

discountStrategy = new RegularDiscountStrategy(); // Предполагается, что для RegularGoods скидка остаётся такой же как в обычный период

break;

case "SAL":

bonusStrategy = new SaleBonusStrategy(); // Бонусы для товаров со скидкой остаются неизменными даже в новогодний период

discountStrategy = new NewYearSaleDiscountStrategy();

break;

case "SPO":

bonusStrategy = new SpecialBonusStrategy(); // Предполагается, что для SpecialGoods бонусы не изменяются

discountStrategy = new NewYearSpecialDiscountStrategy();

break;

default:

throw new ArgumentException("Неизвестный тип товара");

}

break;

default:

throw new ArgumentException("Неизвестный тип стратегии");

}

return new Goods(title, bonusStrategy, discountStrategy);

}

}

public class HtmlView : IView

{

// header

public string GetHeader(Customer \_customer)

{

return $"<html><head><title>Чек для {\_customer.getName()}</title></head><body>" +

$"<h1>Чек для {\_customer.getName()}</h1>" +

"<table border=\"1\"><tr><th>Название</th><th>Цена</th><th>Кол-во</th><th>Стоимость</th><th>Скидка</th><th>Сумма</th><th>Бонус</th></tr>";

}

// table content

public string GetItemString(Item each, double discount, double thisAmount, int bonus)

{

return $"<tr><td>{each.getGoods().getTitle()}</td>" +

$"<td>{each.getPrice()}</td>" +

$"<td>{each.getQuantity()}</td>" +

$"<td>{each.getQuantity() \* each.getPrice()}</td>" +

$"<td>{discount}</td>" +

$"<td>{thisAmount}</td>" +

$"<td>{bonus}</td></tr>";

}

// results

public string GetFooter(double totalAmount, int totalBonus)

{

return $"</table><p>Сумма счета составляет {totalAmount}</p>" +

$"<p>Вы заработали {totalBonus} бонусных балов</p></body></html>";

}

}

// interface for bonus strategy

public interface IBonusStrategy

{

int CalculateBonus(int quantity, double price);

}

// bonus strategy for regular goods

public class RegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

return (int)(quantity \* price \* 0.05);// 5%

}

}

// bonus strategy for sale goods

public class SaleBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

return (int)(quantity \* price \* 0.01); // 1%

}

}

// bonus strategy for special goods

public class SpecialBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

return 0;

}

}

// bonus strategy for new year regular goods

public class NewYearRegularBonusStrategy : IBonusStrategy

{

public int CalculateBonus(int quantity, double price)

{

var total = quantity \* price;

if (total >= 5000)

return (int)(total \* 0.07); // 7%

if (quantity > 2)

return (int)(quantity \* price \* 0.05); // 5%

return 0;

}

}

// interface for discount strategy

public interface IDiscountStrategy

{

double CalculateDiscount(int quantity, double price);

}

// discount strategy for regular goods

public class RegularDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

if (quantity > 2)

return (quantity \* price) \* 0.03; // 3%

return 0;

}

}

// discount strategy for sale goods

public class SaleDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

if (quantity > 3)

return (quantity \* price) \* 0.01; // 0.1%

return 0;

}

}

// discount strategy for regular goods

public class SpecialDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

if (quantity > 10)

return (quantity \* price) \* 0.005; // 0.5% скидка

return 0;

}

}

// discount strategy for new year sale goods

public class NewYearSaleDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

var total = quantity \* price;

if (total > 2000)

return total \* 0.03; // 3%

if (quantity > 3)

return (quantity \* price) \* 0.01; // 0.1%

return 0;

}

}

// discount strategy for new year special goods

public class NewYearSpecialDiscountStrategy : IDiscountStrategy

{

public double CalculateDiscount(int quantity, double price)

{

var total = quantity \* price;

if (total > 3000)

return total \* 0.05; // 5%

return 0;

}

}

public class YamlFileSource : IFileSource

{

private TextReader reader;

string line;

string[] result;

private int strategyType;

// get source

public void SetSource(TextReader reader, int strategyType)

{

this.reader = reader;

this.strategyType = strategyType;

}

// read customer info

public Customer GetCustomer()

{

// read customer

GetNextLine();

result = line.Split(':');

string name = result[1].Trim();

// read bonus

GetNextLine();

result = line.Split(':');

int bonus = Convert.ToInt32(result[1].Trim());

return new Customer(name, bonus);

}

// returns goods count

public int GetGoodsCount()

{

GetNextLine();

result = line.Split(':');

return Convert.ToInt32(result[1].Trim());

}

// reads each product

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

GetNextLine();

result = line.Split(':');

result = result[1].Trim().Split();

string type = result[1].Trim();

return factory.Create(type, result[0], strategyType);

}

// reads the amount of products

public int GetItemsCount()

{

GetNextLine();

result = line.Split(':');

return Convert.ToInt32(result[1].Trim());

}

// reads each item

public Item GetNextItem(Goods[] g)

{

GetNextLine();

result = line.Split(':');

result = result[1].Trim().Split();

int gid = Convert.ToInt32(result[0].Trim());

double price = Convert.ToDouble(result[1].Trim());

int qty = Convert.ToInt32(result[2].Trim());

return new Item(g[gid - 1], qty, price);

}

// skip comments

public void GetNextLine()

{

do

{

line = reader.ReadLine();

}

while (line.StartsWith('#'));

}

}

**Заключение:**

Были изучены техники рефакторинга с использованием поведенческих паттернов проектирования.