Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

"Кубанский государственный технологический университет"

(ФГБОУ ВО "КубГТУ")

Институт КСиИБ

Кафедра ИСП

**ОТЧЁТ**

по лабораторной работе №8

в рамках освоения дисциплины

**«Рефакторинг и работа с унаследованным кодом»**

**Тема: «Рефакторинг и расширение функциональности с использованием поведенческих паттернов проектирования»**

Обучающегося группы 21-КБ-ПР1

Направление подготовки 09.03.04 Программная инженерия

Студент Кузнецов Вадим Алексеевич

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.

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

//Интерфейс для обработки входных данных

public interface IFileSource

{

public void SetSource(TextReader reader);

public Customer GetCustomer();

public int GetGoodsCount();

public Goods GetNextGood();

public int GetItemsCount();

public Item GetNextItem(Goods[] g);

public void GetNextLine();

}

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

public class YamlFileSource

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

public class YamlFileSource : IFileSource

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

//---Метод для преобразования данных из файла

public static BillGenerator CreateBill(TextReader sr)

{

IFileSource readContent = new YamlFileSource();

}

**5) 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).**

public class BillFactory

{

private IFileSource readContent;

public BillFactory(IFileSource readContent)

{

this.readContent = readContent;

}

}

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

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"));

decimal price = Convert.ToDecimal(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

}

}

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

public class FileSourceFactory

{

public static IFileSource CreateFileSource(string filePath)

{

string extension = Path.GetExtension(filePath).ToLower();

switch (extension)

{

case ".yaml":

return new YamlFileSource();

case ".html":

return new HtmlFileSource();

default:

throw new NotSupportedException($"File extension '{extension}' is not supported.");

}

}

}

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

static void Main(string[] args)

{

string filename = "BillInfo.yaml";

if (args.Length == 1)

filename = args[0];

IFileSource fileSource = FileSourceFactory.CreateFileSource(filename);

using (FileStream fs = new FileStream(filename, FileMode.Open))

using (StreamReader sr = new StreamReader(fs))

{

BillFactory factory = new BillFactory(fileSource);

BillGenerator bill = factory.CreateBill(sr);

string billOutput = bill.GenerateBill();

Console.WriteLine(billOutput);

}

}

**8) 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.**

[Test]

public void htmlFile()

{

using (FileStream fs = new FileStream(nameHtmlSource, FileMode.Open))

using (StreamReader sr = new StreamReader(fs))

{

BillFactory factory = new BillFactory(fileSourceHtml);

BillGenerator bill = factory.CreateBill(sr);

string result = bill.GenerateBill();

Assert.IsTrue(result.Contains("Сумма счета составляет 553,3"));

Assert.IsTrue(result.Contains("Вы заработали 20 бонусных балов"));

}

}

[Test]

public void yamlFile()

{

using (FileStream fs = new FileStream(nameYamlSource, FileMode.Open))

using (StreamReader sr = new StreamReader(fs))

{

BillFactory factory = new BillFactory(fileSourceYaml);

BillGenerator bill = factory.CreateBill(sr);

string result = bill.GenerateBill();

Assert.IsTrue(result.Contains("Сумма счета составляет 553,3"));

Assert.IsTrue(result.Contains("Вы заработали 20 бонусных балов"));

}

}

**До:**

static void Main(string[] args)

{

string filename = "BillInfo.yaml";

if (args.Length == 1)

filename = args[0];

using (FileStream fs = new FileStream(filename, FileMode.Open))

using (StreamReader sr = new StreamReader(fs))

{

ContentFile readContent = new ContentFile();

readContent.SetSource(sr);

// Чтение покупателя

Customer customer = readContent.GetCustomer();

}

}

**После:**

//Интерфейс для обработки входных данных

public interface IFileSource

{

public void SetSource(TextReader reader);

public Customer GetCustomer();

public int GetGoodsCount();

public Goods GetNextGood();

public int GetItemsCount();

public Item GetNextItem(Goods[] g);

public void GetNextLine();

}

public class FileSourceFactory

**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.

**2) 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.

**3) 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).

**Заключение**

В результате выполнения лабораторной работы были изучены техники рефакторинга с использованием поведенческих паттернов проектирования.