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 №7

в рамках освоения дисциплины

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

**Тема: «Рефакторинг перед добавлением функциональности, схожей с уже существующей»**

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 процесс рефакторинга перед добавлением новой функциональности, схожей с уже существующей.

**Задание:**

1) Преобразовать процесс парсинга файлов в соответствии с предложенными в пункте 2.1 изменениями, применив описанные в теории методы рефакторинга.

2) Добавить поддержку нового формата входных файлов.

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

**1) Перед началом добавления нового формата файла необходимо выделить общий интерфейс для всех источников IFileSource, добавив в него основные методы из ContentFile, используемы для создания класса Bill в методе CreateBill из BillFactory.**

//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) Переименуем ContentFile в YamlFileSource для более четкого именования.**

public class YamlFileSource

**3) Добавим интерфейс IFileSource в список реализуемых в классе YamlFileSource.**

public class YamlFileSource : IFileSource

**4) Заменим все использования ContentFile на соответствующий интерфейс IFileSource**

//---Метод для преобразования данных из файла

public static BillGenerator CreateBill(TextReader sr)

{

IFileSource readContent = new YamlFileSource();

}

**5) Параметризуем конструктор BillFactory интерфейсом IFileSource и передадим в него YamlFileSource при создании BillFactory.**

public class BillFactory

{

private IFileSource readContent;

public BillFactory(IFileSource readContent)

{

this.readContent = readContent;

}

}

**6) Создадим еще одну реализацию интерфейса IFileSource, обеспечивающую поддержку альтернативного входного формата (формат файла и его парсинг реализовать самостоятельно).**

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

}

}

**7) Создать фабрику (FileSourceFactory) создающую экземпляры IFileSource на основе расширения передаваемых файлов.**

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.");

}

}

}

**8) Заменить в классе Program в Main процесс создания парсеров файлов на FileSourceFactory.**

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);

}

}

**10) Добавить тесты для новой функциональности.**

[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);

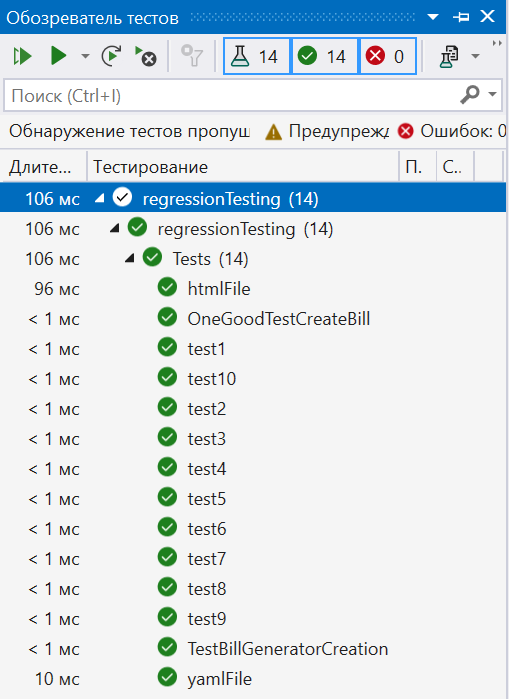
string result = bill.GenerateBill();

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();

IView view = new TxtView();

BillGenerator billGenerator = 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++)

{

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

}

string billOutput = billGenerator.GenerateBill();

Console.WriteLine(billOutput);

}

}

**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

{

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.");

}

}

}

//Класс отвечающий за чтение и обработку Yaml файлов

public class YamlFileSource : IFileSource

{

private TextReader reader;

private string line;

private string[] parts;

//---Метод отвечающий за получение источника

public void SetSource(TextReader reader)

{

this.reader = reader;

}

//---Метод отвечающий за чтение покупателя

public Customer GetCustomer()

{

// Чтение покупателя

GetNextLine();

parts = line.Split(':');

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

// Чтение бонусов

GetNextLine();

parts = line.Split(':');

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

return new Customer(name, bonus);

}

//---Метод отвечающий за чтение количества продуктов

public int GetGoodsCount()

{

GetNextLine();

parts = line.Split(':');

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

}

//---Метод отвечающий за чтение каждого продукта

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

GetNextLine();

parts = line.Split(':');

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

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

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

}

//---Метод отвечающий за чтение количества товаров

public int GetItemsCount()

{

GetNextLine();

parts = line.Split(':');

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

}

//---Метод отвечающий за чтение каждого товара

public Item GetNextItem(Goods[] g)

{

GetNextLine();

parts = line.Split(':');

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

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

decimal price = Convert.ToDecimal(parts[1].Trim());

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

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

}

//---Метод отвечающий за считывание строки

public void GetNextLine()

{

// Пропуск комментариев

do

{

line = reader.ReadLine();

} while (line.StartsWith("#"));

}

}

//Класс отвечающий за чтение и обработку Html файлов

public class HtmlFileSource : IFileSource

{

private TextReader reader;

private string line;

//---Метод отвечающий за получение источника

public void SetSource(TextReader reader)

{

this.reader = reader;

}

//---Метод отвечающий за чтение покупателя

public Customer GetCustomer()

{

string name = ExtractValue("customerName");

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

return new Customer(name, bonus);

}

//---Метод отвечающий за чтение количества продуктов

public int GetGoodsCount()

{

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

}

//---Метод отвечающий за чтение каждого продукта

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

string type = ExtractValue("goodType");

string name = ExtractValue("goodName");

return factory.Create(type, name);

}

//---Метод отвечающий за чтение количества товаров

public int GetItemsCount()

{

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

}

//---Метод отвечающий за чтение каждого товара

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);

}

//---Метод поиска строки с первым и последним тегами, чтобы получить внутренний текст

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

}

}

[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"));

}

}

**Контрольные вопросы**

**1) В чем особенность подготовки к включению новой функциональности схожей с существующей?**

В анализе и рефакторинге текущего кода для обеспечения его расширяемости и поддержки повторного использования. Это включает в себя выявление общих шаблонов, поведений и структур данных, которые можно абстрагировать или параметризировать, чтобы новая функциональность могла быть добавлена с минимальными изменениями в коде. Такой подход уменьшает дублирование кода, упрощает тестирование и обслуживание системы, а также ускоряет процесс разработки за счет повторного использования существующих компонентов.

**2) Для чего необходимо явно выделять скрытые абстракции при подготовке к добавлению новой функциональности.**

Для создания более гибкой и масштабируемой архитектуры приложения. Это позволяет лучше организовать код, сделать его более понятным и удобным для внесения изменений. Выделение абстракций способствует отделению интерфейса от реализации, что дает возможность изменять поведение программы без изменения её основного кода, а также облегчает добавление новых функциональностей, которые могут работать с различными реализациями этих абстракций. В итоге, это приводит к повышению модульности и уменьшению связанности компонентов системы.

**Заключение**

В результате выполнения лабораторной работы был изучен процесс рефакторинга перед добавлением новой функциональности, схожей с уже существующей.