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 перед добавлением**

**функциональности, схожей с уже существующей»**

**Лабораторная: 7**

Работу выполнил: Басма Нассиф Хуссейн

Направление подготовки: 09.03.04 Программная инженерия

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

**Цель работы:** изучить процесс рефакторинга перед добавлением новой функциональности, схожей с уже существующей.

**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.

**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.**

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.**

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();

}

1. **Переименуем ContentFile в YamlFileSource для более четкого именования.**

public class YamlFileSource

1. **Добавим интерфейс IFileSource в список реализуемых в классе YamlFileSource.**

public class YamlFileSource : IFileSource

1. **Заменим все использования ContentFile на соответствующий интерфейс IFileSource**

// BILLFACTORY

public BillGenerator CreateBill(TextReader sr)

{

IFileSource contentFile = new YamlFileSource();

}

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

// BILLFACTORY

private IFileSource contentFile;

// Параметризованный конструктор, принимающий IFileSource

public BillFactory(IFileSource contentFile)

{

this.contentFile = contentFile;

}

1. **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"));

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

}

}

1. **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.");

}

}

}

1. **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.GetBIll();

Console.WriteLine(billOutput);

}

}

1. **Добавить тесты для новой функциональности.**

private BillGenerator bill;

private Customer customer;

private BillFactory billFactoryHtml;

private BillFactory billFactoryYaml;

private string nameHtmlSource = "BillInfo.html";

private string nameYamlSource = "BillInfo.yaml";

IFileSource fileSourceHtml;

IFileSource fileSourceYaml;

[SetUp]

public void Setup()

{

fileSourceHtml = FileSourceFactory.CreateFileSource(nameHtmlSource);

billFactoryHtml = new BillFactory(fileSourceHtml);

fileSourceYaml = FileSourceFactory.CreateFileSource(nameYamlSource);

billFactoryYaml = new BillFactory(fileSourceYaml);

customer = new Customer("Тестовый клиент",10);

IView view = new TxtView();

bill = new BillGenerator(customer, view);

}

// ЛАБА 7

[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.GetBIll();

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.GetBIll();

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

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

}

}



Рисунок 1 – Результат тестирование

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

**До:**

// Program

using System.Text;

namespace РРУК\_01

{

public class Program

{

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))

{

BillFactory factory = new BillFactory();

BillGenerator bill = factory.CreateBill(sr);

string billOutput = bill.GetBIll();

Console.WriteLine(billOutput);

}

}

}

}

// YamlFileSource (бывший ContentFile)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace РРУК\_01

{

public class ContentFile

{

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('#'));

}

}

}

**После:**

// UNITTEST

private BillGenerator bill;

private Customer customer;

private BillFactory billFactoryHtml;

private BillFactory billFactoryYaml;

private string nameHtmlSource = "BillInfo.html";

private string nameYamlSource = "BillInfo.yaml";

IFileSource fileSourceHtml;

IFileSource fileSourceYaml;

[SetUp]

public void Setup()

{

fileSourceHtml = FileSourceFactory.CreateFileSource(nameHtmlSource);

billFactoryHtml = new BillFactory(fileSourceHtml);

fileSourceYaml = FileSourceFactory.CreateFileSource(nameYamlSource);

billFactoryYaml = new BillFactory(fileSourceYaml);

customer = new Customer("Тестовый клиент",10);

IView view = new TxtView();

bill = new BillGenerator(customer, view);

}

// ЛАБА 7

[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.GetBIll();

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.GetBIll();

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

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

}

}

// FileSourceFactory

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.");

}

}

}

// HtmlFileSource

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

}

}

// Program

static void Main(string[] args)

{

string filename = "BillInfo.html";

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.GetBIll();

Console.WriteLine(billOutput);

}

}

// IFileSource

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();

}

// YamlFileSource

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('#'));

}

}

**Заключение:**

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