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 №6

в рамках освоения дисциплины

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

**Тема: «Рефакторинг с использованием порождающих 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 порождающих паттернов проектирования.

**Задание:**

1) Преобразовать процесс создания объектов в методе Main класса Program 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) Перед началом преобразований необходимо добавить тесты для проверки создания объектов на основе заданного файла. Однако код создания объекта совершенно не приспособлен для тестирования, так как взаимодействует непосредственно с файлом для ввода и консолью для вывода. Поэтому нам необходимо выполнить некоторые предварительные преобразования для того, чтобы привести этот код в состояние, пригодное для тестирования. Для этого можно воспользоваться рефакторингом «Выделение метода».**

**Выделим почти весь код основного метода Main так, чтобы он возвращал экземпляр класса Bill (вставив в Main только определение имени файла, его создание и вывод строки из метода statement на экран). Назовем его CreateBill и параметризуем TextReader для организации доступа к строкам файла.**

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))

{

BillGenerator bill = CreateBill(sr);

string billOutput = bill.GenerateBill();

Console.WriteLine(billOutput);

}

}

//---Метод для преобразования данных из файла

public static BillGenerator CreateBill(TextReader sr)

{

// read customer

string line = sr.ReadLine();

string[] result = line.Split(':');

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

// read bonus

line = sr.ReadLine();

result = line.Split(':');

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

Customer customer = new Customer(name, bonus);

IView view = new TxtView();

BillGenerator b = new BillGenerator(customer, view);

// read goods count

line = sr.ReadLine();

result = line.Split(':');

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

Goods[] g = new Goods[goodsQty];

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

{

// Пропустить комментарии

do

{

line = sr.ReadLine();

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

result = line.Split(':');

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

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

switch (type)

{

case "REG":

g[i] = new RegularGoods(result[0]);

break;

case "SAL":

g[i] = new SaleGoods(result[0]);

break;

case "SPO":

g[i] = new SpecialGoods(result[0]);

break;

}

}

// read items count

// Пропустить комментарии

do

{

line = sr.ReadLine();

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

result = line.Split(':');

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

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

{

// Пропустить комментарии

do

{

line = sr.ReadLine();

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

result = line.Split(':');

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

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

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

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

b.addGoods(new Item(g[gid - 1], qty, price));

}

return b;

}

}

**После этого необходимо добавить тесты фиксирующие работу текущего парсинга файлов с использованием StringReader в качестве источника данных.**

[Test]

public void OneGoodTestCreateBill()

{

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 1\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Cola REG\r\n" +

"ItemsTotalCount: 1\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 80 20";

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = Program.CreateBill(sr);

var result = billGenerator.GenerateBill();

Assert.IsTrue(result.Contains("Сумма счета составляет 1542,00"));

Assert.IsTrue(result.Contains("Вы заработали 80 бонусных балов"));

}

}

[Test]

public void TestBillGeneratorCreation()

{

string testData = "CustomerName: Test\r\n" +

"CustomerBonus: 10\r\n" +

"GoodsTotalCount: 3\r\n" +

"# ID: NAME TYPE(REG/SAL/SPO)\r\n" +

"1: Cola REG\r\n" +

"2: Pepsi SAL\r\n" +

"3: Fanta SPO\r\n" +

"ItemsTotalCount: 3\r\n" +

"# ID: GID PRICE QTY\r\n" +

"1: 1 65 6\r\n" +

"2: 2 50 3\r\n" +

"3: 3 35 1";

using (StringReader sr = new StringReader(testData))

{

BillGenerator billGenerator = Program.CreateBill(sr);

var result = billGenerator.GenerateBill();

Assert.IsTrue(result.Contains("Сумма счета составляет 553,30"));

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

}

}

**2) Создадим новый класс GoodsFactory (в отдельном файле в проекте).**

public class GoodsFactory

{

}

**3) Создадим параметризованный метод Create в классе GoodsFactory, и определим в нем процесс создания экземпляров наследников класса Goods.**

**Поскольку в дальнейшем нам будет необходима поддержка различных форматов входных файлов, то для доступа к информации, хранимой в файле необходимо выделить код обработки файла в отдельный класс.**

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");

}

}

**5) Создадим новый класс ContentFile (в отдельном файле в проекте).**

public class ContentFile

{

}

**6) Определим интерфейс нового класса. Нам потребуются методы SetSource, GetCustomer, GetGoodsCount, GetNextGood, GetItemsCount, GetNextItem, которые будут последовательно вызываться из метода CreateBill, содержащего последовательность вызовов методов.**

**Создание экземпляров наследников класса Goods возложено на метод Create класса GoodsFactory.**

//Класс отвечающий за чтение и обработку

public class ContentFile

{

private TextReader reader;

string line;

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

line = reader.ReadLine();

parts = line.Split(':');

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

// Чтение бонусов

line = reader.ReadLine();

parts = line.Split(':');

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

return new Customer(name, bonus);

}

//---Метод отвечающий за чтение количества продуктов

public int GetGoodsCount()

{

line = reader.ReadLine();

parts = line.Split(':');

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

}

//---Метод отвечающий за чтение каждого продукта

public Goods GetNextGood()

{

GoodsFactory factory = new GoodsFactory();

do

{

line = reader.ReadLine();

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

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()

{

// Пропуск комментариев

do

{

line = reader.ReadLine();

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

parts = line.Split(':');

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

}

//---Метод отвечающий за чтение каждого товара

public Item GetNextItem(Goods[] g)

{

// Пропуск комментариев

do

{

line = reader.ReadLine();

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

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);

}

}

**7) Так как на данном этапе нет необходимости в различных форматах файлов, то на данном у нас будет иметься только одна реализация, работающая с YAML файлами.**

**8) В классе ContentFile также можно создать вспомогательные методы для работы с файлом (например, получение следующей строки с пропуском комментариев GetNextLine).**

//---Метод отвечающий за считывание строки

public void GetNextLine()

{

// Пропуск комментариев

do

{

line = reader.ReadLine();

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

}

**9) В основной части программы, внутри выделенного для тестирования метода, заменим создание объекта Bill на создание ContentFile и получение от него объекта Bill.**

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);

}

}

**10) Создадим новый класс BillFactory (в отдельном файле в проекте) перенеся туда код CreateBill.**

public class BillFactory

{

//---Метод для преобразования данных из файла

public static BillGenerator CreateBill(TextReader sr)

{

ContentFile readContent = new ContentFile();

readContent.SetSource(sr);

// Чтение покупателя

Customer customer = readContent.GetCustomer();

IView view = new TxtView();

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));

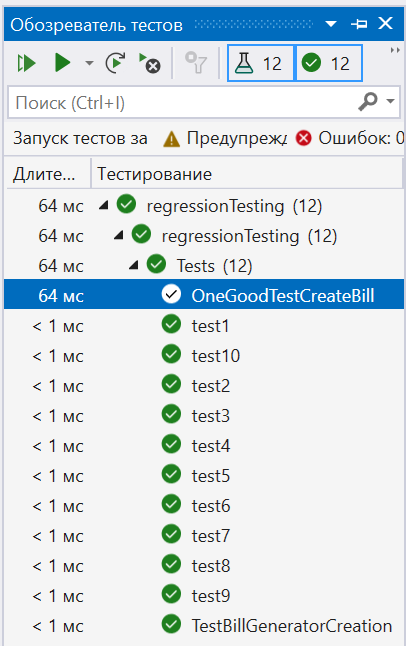
}

return b;

}

}

**11) Протестируем вновь создаваемые объекты с помощью уже существующих тестов.**

****

**До:**

static void Main(string[] args)

{

string filename = "BillInfo.yaml";

if (args.Length == 1)

filename = args[0];

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

StreamReader sr = new StreamReader(fs);

// read customer

string line = sr.ReadLine();

string[] result = line.Split(':');

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

// read bonus

line = sr.ReadLine();

result = line.Split(':');

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

Customer customer = new Customer(name, bonus);

IView view = new TxtView();

BillGenerator b = new BillGenerator(customer, view);

// read goods count

line = sr.ReadLine();

result = line.Split(':');

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

Goods[] g = new Goods[goodsQty];

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

{

// Пропустить комментарии

do

{

line = sr.ReadLine();

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

result = line.Split(':');

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

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

switch (type)

{

case "REG":

g[i] = new RegularGoods(result[0]);

break;

case "SAL":

g[i] = new SaleGoods(result[0]);

break;

case "SPO":

g[i] = new SpecialGoods(result[0]);

break;

}

}

// read items count

// Пропустить комментарии

do

{

line = sr.ReadLine();

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

result = line.Split(':');

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

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

{

// Пропустить комментарии

do

{

line = sr.ReadLine();

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

result = line.Split(':');

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

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

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

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

b.addGoods(new Item(g[gid - 1], qty, price));

}

string bill = b.GenerateBill();

Console.WriteLine(bill);

}

**После:**

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);

}

}

//Класс отвечающий за чтение и обработку

public class ContentFile

{

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 определения типа файла

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");

}

}

}

**Контрольные вопросы**

**1) Что такое паттерн проектирования?**

Это повторяемое решение часто встречающейся проблемы в контексте архитектуры программного обеспечения. Паттерны помогают стандартизировать решения, упростить процесс разработки и улучшить коммуникацию между разработчиками за счёт общего словаря.

**2) Для чего используется паттерн Factory method?**

Используется для создания объектов без спецификации конкретных классов. Паттерн определяет интерфейс создания объекта, но позволяет подклассам изменять тип создаваемых объектов. Это обеспечивает гибкость и расширяемость кода, поскольку добавление новых классов продуктов не требует изменений в клиентском коде.

**3) Как проверять результаты работы программы в случае ее неприспособленности для автоматизированного тестирования?**

Для проверки результатов работы программы, не приспособленной для автоматизированного тестирования, можно использовать следующие подходы:

* **Ручное тестирование**: Прямое взаимодействие с программой для проверки её функциональности и поведения.
* **Логирование**: Вывод информации о работе программы в логи для последующего анализа.
* **Отладка**: Использование отладчиков для пошагового выполнения программы и наблюдения за изменениями в её состоянии.
* **Exploratory Testing**: Исследовательское тестирование, при котором тестировщик активно исследует программу в поисках ошибок без заранее подготовленных тестовых сценариев.
* **Код-ревью и парное программирование**: Ручная проверка кода другими разработчиками может помочь выявить ошибки и улучшить качество программы.

**Заключение**

В результате выполнения лабораторной работы были изучены техники рефакторинга позволяющие выполнять отделение модели предметной области от непосредственно процесса генерации чека.