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: «Базовые техники рефакторинга.**

**Рефакторинги в рамках одного класса»**

**Лабораторная: 1**

Работу выполнил: Басма Нассиф Хуссейн

Направление подготовки: 09.03.04 Программная инженерия

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

**Цель работы:** изучить базовые техники рефакторинга позволяющие различным образом манипулировать методами программы.

**Контрольные вопросы**

**1) Что такое рефакторинг?**

Изменение внутренней структуры программного обеспечения для упрощения понимания и модификации кода, не затрагивая его поведения.

**2) Каковы цели проведения рефакторинга?**

Рефакторинг позволяет навести порядок в коде и улучшить его структурирование. При этом из него удаляются ненужные фрагменты, и сокращается дублирование кода. Это улучшает в свою очередь понимание и читабельность кода. Все это приводит в конечном итоге к увеличению скорости разработки.

**3) Какие методы рефакторинга составления методом вы знаете?**

* Выделение метода (Extract Method);
* Встраивание метода (Inline Method);
* Перемещение метода (Move Method);
* Введение внешнего метода (Introduce Foreign Method)
* Подъем метода (Pull Up Method)
* Спуск метода (Push Down Method)

**4) В чем заключается рефакторинг «Выделение метода»?**

* Рефакторинг, при котором фрагмент кода превращается в отдельный метод с понятным названием, объясняющим его назначение. Этот подход рекомендуется, когда код слишком длинный или требует много комментариев для понимания. Однако, при использовании этого рефакторинга необходимо быть внимательным к локальным переменным, особенно к временным, и стремиться к их минимизации.

**5) В чем заключается рефакторинг «Встраивание метода»?**

* Рефакторинг, при котором вызов метода заменяется кодом, содержащимся в теле самого метода. Этот подход рекомендуется, когда методы становятся слишком простыми или их тела так же прозрачны, как и названия методов.

**6) В чем суть рефакторинга «Встраивание временной переменной»?**

* Рефакторинг, при котором временная переменная, которой присваивается простое выражение и мешает проведению других рефакторингов, заменяется этим выражением напрямую в коде. Этот подход помогает упростить код и избавиться от лишних переменных, которые могут затруднять чтение и понимание программы.

**7) В чем суть рефакторинга «Замена временной переменной вызовом метода»?**

* Рефакторинг, при котором временная переменная, используемая для хранения значения выражения, преобразуется в метод, который возвращает это значение, и все ссылки на временную переменную заменяются вызовами этого метода.

**8) В чем суть рефакторинга «Введение поясняющей переменной»?**

* Рефакторинга, при котором сложное выражение заменяется временной переменной, которая ясно объясняет его смысл. Это улучшает читаемость кода и делает его более понятным для других разработчиков. Однако стоит помнить, что такой подход может быть заменен выделением метода, что позволит повторно использовать выражение не только в пределах одного метода, а во всем объекте и даже в других объектах.

**9) В чем суть рефакторинга «Расщепление временной переменной»?**

* Рефакторинг "Расщепление временной переменной" заключается в создании отдельной временной переменной для каждого присваивания значения, если переменная не используется в качестве переменной цикла или для накопления результата. Это улучшает читаемость кода и делает его более понятным для других разработчиков, так как каждая переменная теперь отвечает только за одну функцию.

**10) В чем суть рефакторинга «Удаление присваиваний параметрам»?**

* Рефакторинг "Удаление присваиваний параметрам" рекомендуется, когда код присваивает значение параметру, но это значение не возвращается в вызывающий метод. Вместо этого используется временная переменная для хранения промежуточных вычислений, что делает код более чистым и понятным.

**Задание**

1) Перед началом модификации кода написать тестовые наборы для проведения регрессионного тестирования метода statement класса Bill, представленного в примере 1.10.

using РРУК\_01;

namespace regressionTesting

{

public class Tests

{

private Bill bill;

private Customer customer;

[SetUp]

public void Setup()

{

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

bill = new Bill(customer);

}

[Test]

public void EmptyBillTest()

{

var result = bill.statement();

var expectedFooter = "Сумма счета составляет 0\nВы заработали 0 бонусных балов";

Assert.IsTrue(result.Contains(expectedFooter));

}

[Test]

public void BillSingleItem()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 1, 100));

var result = bill.statement();

Assert.IsTrue(result.Contains("Товар 1"));

Assert.IsTrue(result.Contains("Сумма счета составляет 100\n"));

Assert.IsTrue(result.Contains("Вы заработали 5 бонусных балов"));

}

[Test]

public void BIllMultipleItems()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 3, 100));

bill.addGoods(new Item(new Goods("Товар 2", Goods.SPECIAL\_OFFER), 11, 50));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 831"));

Assert.IsTrue(result.Contains("Вы заработали 15 бонусных балов"));

}

[Test]

public void DiscountsBonuses()

{

bill.addGoods(new Item(new Goods("Товар со скидкой", Goods.SALE), 11, 200));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 2178"));

Assert.IsTrue(result.Contains("Вы заработали 22 бонусных балов"));

}

[Test]

public void Statement\_UsingCustomerBonuses\_ReturnsCorrectDiscount()

{

customer.receiveBonus(100);

bill.addGoods(new Item(new Goods("Товар с использованием бонусов", Goods.REGULAR), 6, 100));

var result = bill.statement();

Assert.IsTrue(result.Contains("Вы заработали 30 бонусных балов"));

}

}

}

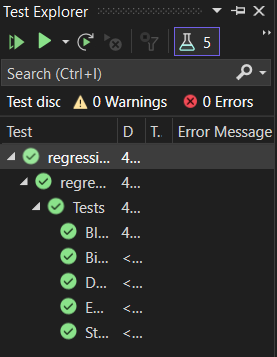


Рисунок 1 – Регрессионное тестирование

2) Тестовые наборы необходимо составлять, опираясь на структуру метода statement методом белого ящика и учитывая, как количество товара и его категория влияют на сумму бонусов и скидку. (Количество тестов должно быть не менее 10-12 штук).

**Набор тестов**

1. Тестирование без товаров
2. Тестирование с одним товарам.
3. Тестирование с товаром SPECIAL\_OFFER с количеством больше 10
4. Тестирование с товаром SALE с количеством больше 3
5. Тестирование с товаром SPECIAL\_OFFER с количеством меньше 10
6. Тестирование с товаром SALE с количеством меньше 3
7. Тестирование с товаром REGULAR с количеством больше 2
8. Тестирование с товаром REGULAR с количеством 2
9. Тестирование c вводом 0 товаров
10. Тестирование с разными товарами
11. Тестирование с максимально возможным количеством товаров

Код:

[Test]

public void test1()

{

var result = bill.statement();

var expectedFooter = "Сумма счета составляет 0\nВы заработали 0 бонусных балов";

Assert.IsTrue(result.Contains(expectedFooter));

}

[Test]

public void test2()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 1, 100));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 100\n"));

Assert.IsTrue(result.Contains("Вы заработали 5 бонусных балов"));

}

[Test]

public void test3()

{

bill.addGoods(new Item(new Goods("Товар 2", Goods.SPECIAL\_OFFER), 11, 50));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 540"));

Assert.IsTrue(result.Contains("Вы заработали 0 бонусных балов"));

}

[Test]

public void test4()

{

bill.addGoods(new Item(new Goods("Товар со скидкой", Goods.SALE), 11, 200));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 2178"));

Assert.IsTrue(result.Contains("Вы заработали 22 бонусных балов"));

}

[Test]

public void test5()

{

bill.addGoods(new Item(new Goods("Товар 2", Goods.SPECIAL\_OFFER), 5, 50));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 240"));

Assert.IsTrue(result.Contains("Вы заработали 0 бонусных балов"));

}

[Test]

public void test6()

{

bill.addGoods(new Item(new Goods("Товар со скидкой", Goods.SALE), 2, 200));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 400"));

Assert.IsTrue(result.Contains("Вы заработали 4 бонусных балов"));

}

[Test]

public void test7()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 3, 100));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 291\n"));

Assert.IsTrue(result.Contains("Вы заработали 15 бонусных балов"));

}

[Test]

public void test8()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 2, 100));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 200\n"));

Assert.IsTrue(result.Contains("Вы заработали 10 бонусных балов"));

}

[Test]

public void test9()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 0, 100));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 0"));

Assert.IsTrue(result.Contains("Вы заработали 0 бонусных балов"));

}

[Test]

public void test10()

{

bill.addGoods(new Item(new Goods("Товар 1", Goods.REGULAR), 6, 100));

bill.addGoods(new Item(new Goods("Товар со скидкой", Goods.SALE), 11, 200));

bill.addGoods(new Item(new Goods("Товар 2", Goods.SPECIAL\_OFFER), 15, 50));

var result = bill.statement();

Assert.IsTrue(result.Contains("Сумма счета составляет 3500"));

Assert.IsTrue(result.Contains("Вы заработали 52 бонусных балов"));

}

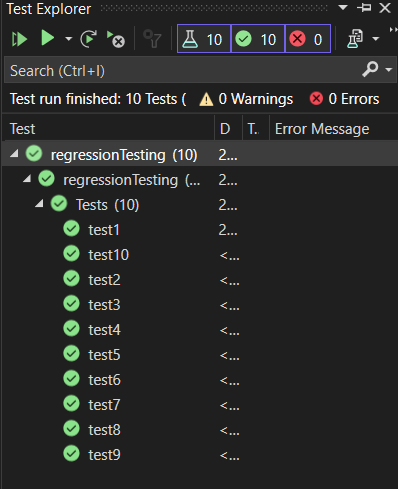


Рисунок 2 – Набор тестов

3) Преобразовать метод statement класса Bill в соответствии с предложенными в пункте 2.4 изменениями, применить описанные в теории методы рефакторинга.

**Порядок выполнение**

1. Тесты выше
2. Выделить части, связанные с форматированием выводимой информации.

public string getHeader()

{

string result = "Счет для " + \_customer.getName();

result += "\t" + "Название" + "\t" + "Цена" +

"\t" + "Кол-во" + "Стоимость" + "\t" + "Скидка" +

"\t" + "Сумма" + "\t" + "Бонус" + "\n";

return result;

}

public string getFooter(string result, double totalAmount, int totalBonus)

{

result = "Сумма счета составляет " +

totalAmount.ToString() + "\n";

result += "Вы заработали " +

totalBonus.ToString() + " бонусных балов";

return result;

}

public string getItemString(string result, Item each, double discount,

double thisAmount, int bonus)

{

result = "\t" + each.getGoods().getTitle() + "\t" +

"\t" + each.getPrice() + "\t" + each.getQuantity() +

"\t" + (each.getQuantity() \* each.getPrice()).ToString() +

"\t" + discount.ToString() + "\t" + thisAmount.ToString() +

"\t" + bonus.ToString() + "\n";

return result;

}

1. Выделим логику получения бонусов в отдельные методы GetBonus и GetDiscount.

public double GetDiscount(Item each)

{

switch (each.getGoods().getPriceCode())

{

case Goods.REGULAR:

if (each.getQuantity() > 2)

return (each.getQuantity() \* each.getPrice()) \* 0.03; // 3%

break;

case Goods.SPECIAL\_OFFER:

if (each.getQuantity() > 10)

return (each.getQuantity() \* each.getPrice()) \* 0.005; // 0.5%

break;

case Goods.SALE:

if (each.getQuantity() > 3)

return (each.getQuantity() \* each.getPrice()) \* 0.01; // 0.1%

break;

}

return 0;

}

public int GetBonus(Item each)

{

switch (each.getGoods().getPriceCode())

{

case Goods.REGULAR:

return(int)(each.getQuantity() \* each.getPrice() \* 0.05);

case Goods.SALE:

return (int)(each.getQuantity() \* each.getPrice() \* 0.01);

}

return 0;

}

1. Далее замечаем, что есть части, связанные с использованием бонуса (UseBonus): они могут быть перенесены в метод GetDiscount.

// Метод для получения скидки

public double GetDiscount(Item each)

{

double discount = 0;

switch (each.getGoods().getPriceCode())

{

case Goods.REGULAR:

if (each.getQuantity() > 2)

discount = (each.getQuantity() \* each.getPrice()) \* 0.03; // 3%

if (each.getQuantity() > 5)

discount += \_customer.useBonus((int)(each.getQuantity() \* each.getPrice()));

break;

case Goods.SPECIAL\_OFFER:

if (each.getQuantity() > 10)

discount = (each.getQuantity() \* each.getPrice()) \* 0.005; // 0.5%

if (each.getQuantity() > 1)

discount += \_customer.useBonus((int)(each.getQuantity() \* each.getPrice()));

break;

case Goods.SALE:

if (each.getQuantity() > 3)

discount = (each.getQuantity() \* each.getPrice()) \* 0.01; // 0.1%

break;

}

return discount;

}

1. Далее упрощаем тело основного цикла

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = GetDiscount(each);

int bonus = GetBonus(each);

//показать результаты

totalAmount += each.getQuantity() \* each.getPrice() - discount;

result += GetItemString(each, discount, totalAmount, bonus);

totalBonus += bonus;

}

1. Замен повторяющего кода на GetSum(each)

public double GetSum(Item each)

{

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

}

1. Выделим метод GetUsedBonus для получения количества потраченных бонусов

//---Метод получения использованных бонусов

public double GetUsedBonus(Item each, double sumWithDiscount)

{

double usedBonus = 0;

switch (each.getGoods().getPriceCode())

{

//Обычный товав

case Goods.REGULAR:

if (each.getQuantity() > 5)

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

break;

//Специальное предложение

case Goods.SPECIAL\_OFFER:

if (each.getQuantity() > 1)

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

break;

}

return usedBonus;

}

1. Исправим блок начисления использованных бонусов в методе GetUsedBonus, который должен начисляться на основе суммы со скидкой, а не от полной суммы (usedBonus = GetUsedBonus(thisAmount – discount)).

double usedBonus = GetUsedBonus(each, sumWithDiscount);

1. Изменим формулу вычисления стоимости

public String statement()

{

double totalAmount = 0;

int totalBonus = 0;

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

String result = GetHeader();

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = GetDiscount(each);

int bonus = GetBonus(each);

//показать результаты

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

result += GetItemString(each, discount, thisAmount, bonus);

totalAmount += thisAmount;

totalBonus += bonus;

}

//добавить нижний колонтитул

result += GetFooter(totalAmount, totalBonus);

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return result;

}

4) Проверить корректность работы нового метода и работоспособность старого метода формирования счета**До:**

public String statement()

{

double totalAmount = 0;

int totalBonus = 0;

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

String result = "Счет для " + \_customer.getName() + "\n";

result += "\t" + "Название" + "\t" + "Цена" +

"\t" + "Кол-во" + "Стоимость" + "\t" + "Скидка" +

"\t" + "Сумма" + "\t" + "Бонус" + "\n";

while (items.MoveNext())

{

double thisAmount = 0;

double discount = 0;

int bonus = 0;

Item each = (Item)items.Current;

//определить сумму для каждой строки

switch (each.getGoods().getPriceCode())

{

case Goods.REGULAR:

if (each.getQuantity() > 2)

discount =

(each.getQuantity() \* each.getPrice()) \* 0.03; // 3%

bonus = (int)(each.getQuantity() \* each.getPrice() \* 0.05);

break;

case Goods.SPECIAL\_OFFER:

if (each.getQuantity() > 10)

discount =(each.getQuantity() \* each.getPrice()) \* 0.005; // 0.5%

break;

case Goods.SALE:

if (each.getQuantity() > 3)

discount =(each.getQuantity() \* each.getPrice()) \* 0.01; // 0.1%

bonus =

(int)(each.getQuantity() \* each.getPrice() \* 0.01);

break;

}

// сумма

thisAmount = each.getQuantity() \* each.getPrice();

// используем бонусы

if ((each.getGoods().getPriceCode() ==

Goods.REGULAR) && each.getQuantity() > 5)

discount +=

\_customer.useBonus((int)(each.getQuantity() \* each.getPrice()));

if ((each.getGoods().getPriceCode() ==

Goods.SPECIAL\_OFFER) && each.getQuantity() > 1)

discount =

\_customer.useBonus((int)(each.getQuantity() \* each.getPrice()));

// учитываем скидку

thisAmount = each.getQuantity() \* each.getPrice() - discount;

//показать результаты

result += "\t" + each.getGoods().getTitle() + "\t" +

"\t" + each.getPrice() + "\t" + each.getQuantity() +

"\t" + (each.getQuantity() \* each.getPrice()).ToString() +

"\t" + discount.ToString() + "\t" + thisAmount.ToString() +

"\t" + bonus.ToString() + "\n";

totalAmount += thisAmount;

totalBonus += bonus;

}

//добавить нижний колонтитул

result += "Сумма счета составляет " +

totalAmount.ToString() + "\n";

result += "Вы заработали " +

totalBonus.ToString() + " бонусных балов";

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return result;

}

**После:**

namespace РРУК\_01

{

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

public Bill(Customer customer)

{

this.\_customer = customer;

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

}

public void addGoods(Item arg)

{

\_items.Add(arg);

}

// Метод для вывода оглавления

public string GetHeader()

{

return "Счет для " + \_customer.getName() + "\n" +

"\t" + "Название" + "\t" + "Цена" +

"\t" + "Кол-во" + "Стоимость" + "\t" + "Скидка" +

"\t" + "Сумма" + "\t" + "Бонус" + "\n";

}

// Метод для вывода результата

public string GetFooter(double totalAmount, int totalBonus)

{

return "Сумма счета составляет " +

totalAmount.ToString() + "\n" +

"Вы заработали " + totalBonus.ToString() +

" бонусных балов";

}

// Метод для вывода списка товаров

public string GetItemString(Item each, double discount,

double totalAmount, int bonus)

{

return "\t" + each.getGoods().getTitle() + "\t" +

"\t" + each.getPrice() + "\t" + each.getQuantity() +

"\t" + (GetSum(each)).ToString() +

"\t" + discount.ToString() + "\t" + totalAmount.ToString() +

"\t" + bonus.ToString() + "\n";

}

// Метод для получения скидки

public double GetDiscount(Item each)

{

double discount = 0;

switch (each.getGoods().getPriceCode())

{

case Goods.REGULAR:

if (each.getQuantity() > 2)

discount = (GetSum(each)) \* 0.03; // 3%

break;

case Goods.SPECIAL\_OFFER:

if (each.getQuantity() > 10)

discount = (GetSum(each)) \* 0.005; // 0.5%

break;

case Goods.SALE:

if (each.getQuantity() > 3)

discount = (GetSum(each)) \* 0.01; // 0.1%

break;

}

return discount;

}

// Метод для получения бонуса

public int GetBonus(Item each)

{

switch (each.getGoods().getPriceCode())

{

case Goods.REGULAR:

return(int)(GetSum(each) \* 0.05);

case Goods.SALE:

return (int)(GetSum(each) \* 0.01);

}

return 0;

}

// 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;

switch (each.getGoods().getPriceCode())

{

//Обычный товав

case Goods.REGULAR:

if (each.getQuantity() > 5)

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

break;

//Специальное предложение

case Goods.SPECIAL\_OFFER:

if (each.getQuantity() > 1)

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

break;

}

return usedBonus;

}

public String statement()

{

double totalAmount = 0;

int totalBonus = 0;

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

String result = GetHeader();

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = GetDiscount(each);

int bonus = GetBonus(each);

//показать результаты

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

result += GetItemString(each, discount, thisAmount, bonus);

totalAmount += thisAmount;

totalBonus += bonus;

}

//добавить нижний колонтитул

result += GetFooter(totalAmount, totalBonus);

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return result;

}

}

}

**Заключение:**

Были изучены базовые техники рефакторинга позволяющие различным образом манипулировать методами программы.