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: «Сложные техники рефакторинга.**

**Выделение представления»**

**Лабораторная: 4**

Работу выполнил: Басма Нассиф Хуссейн

Направление подготовки: 09.03.04 Программная инженерия

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

**Цель работы:** изучить техники рефакторинга позволяющие выполнять отделение представления от остального кода.

**Контрольные вопросы**

**1) В чем особенность выделения представлений при проведении сложных рефакторингов?**

Особенность выделения представлений при проведении сложных рефакторингов заключается в возможности отделения логики приложения от его пользовательского интерфейса, что упрощает понимание, модификацию и тестирование кода. Это позволяет фокусироваться на изменении бизнес-логики без риска случайного нарушения пользовательского интерфейса, а также облегчает обновление UI без влияния на основную логику программы.

**2) Как проверять корректность работы программы в случае отсутствия готовых тестов?**

В случае отсутствия готовых тестов для проверки корректности работы программы рекомендуется:

* **Разработать тесты:** начать с написания тестов, основанных на текущем поведении программы, чтобы обеспечить регрессионное тестирование.
* **Использовать ручное тестирование:** временно применять ручное тестирование для проверки ключевых функций программы.
* **Применить пошаговое тестирование:** разбить рефакторинг на маленькие шаги и после каждого шага проводить тестирование измененных частей программы.
* **Использовать инструменты статического анализа:** они могут помочь обнаружить потенциальные ошибки и уязвимости в коде без выполнения программы.
* **Проводить Exploratory Testing (исследовательское тестирование):** Этот подход предполагает активный поиск ошибок путем исследования программы без заранее определенных сценариев тестирования.
* **Использовать логирование и отладку:** Добавление логирования в критические части системы может помочь отследить и верифицировать поведение программы в процессе рефакторинга.

**Задание**

**1) Преобразовать методы и классы программы в соответствии с предложенными в пункте 2.2 изменениями, применив описанные в теории методы рефакторинга.**

1. **Создаем иерархию классов представлений interface IView (снабором методов GetHeader/GetFooter/GetItemString)**

public interface IView

{

string GetHeader(Customer \_customer);

string GetFooter(double totalAmount, int totalBonus);

string GetItemString(Item each, double discount,

double totalAmount, int bonus);

}

public class HtmlView : IView

{

public string GetHeader(Customer \_customer)

{

throw new NotImplementedException();

}

public string GetFooter(double totalAmount, int totalBonus)

{

throw new NotImplementedException();

}

public string GetItemString(Item each, double discount,

double totalAmount, int bonus)

{

throw new NotImplementedException();

}

}

public class TxtView : IView

{

public string GetHeader(Customer \_customer)

{

throw new NotImplementedException();

}

public string GetFooter(double totalAmount, int totalBonus)

{

throw new NotImplementedException();

}

public string GetItemString(Item each, double discount,

double totalAmount, int bonus)

{

throw new NotImplementedException();

}

}

1. **Переносим методы GetHeader/GetFooter/GetItemString из Bill в TxtView.**

public class TxtView : IView

{

public string GetHeader(Customer \_customer)

{

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

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

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

}

}

1. **Создаем методы GetHeader/GetFooter/GetItemString для HtmlView**

public class HtmlView : IView

{

// Метод для вывода оглавления

public string GetHeader(Customer \_customer)

{

throw new NotImplementedException();

}

// Метод для вывода результата

public string GetFooter(double totalAmount, int totalBonus)

{

throw new NotImplementedException();

}

// Метод для вывода списка товаров

public string GetItemString(Item each, double discount,

double totalAmount, int bonus)

{

throw new NotImplementedException();

}

}

1. **Параметрезуем конструктор Bill новым параметром IView view и создаем для его хранения поле IView view в классе.**

private List<Item> \_items;

private Customer \_customer;

private IView \_view;

public Bill(Customer customer, IView view)

{

this.\_customer = customer;

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

this.\_view = view;

}

1. **Заменяем в тексте метода statement все вхождения**

String result = view.GetHeader(\_customer);

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

result += view.GetFooter(totalAmount, totalBonus);

1. **Заменяем создание класса Bill**

public void Setup()

{

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

IView view = new TxtView();

bill = new Bill(customer, view);

}

1. **Все тесты должны проходить**



Рисунок 1 – Результаты тестирование

1. **Создаем свойство View для динамической замены представления.**
2. **Переименовать метод statement в более отражающий суть (например, GetBill/GenerateBill/GetOutput)**

public String GetBIll()

**2) Проверить корректность работы нового метода и работоспособность старого метода формирования счета.**

**До:**

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 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;

if (each.getGoods().GetType() == typeof(RegularGoods))

{

if (each.getQuantity() > 5)

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

} else if (each.getGoods().GetType() == typeof(SpecialGoods))

{

if (each.getQuantity() > 1)

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

}

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 = each.GetDiscount();

int bonus = each.GetBonus();

//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;

}

}

**После:**

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public Bill(Customer customer, IView view)

{

this.\_customer = customer;

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

this.view = view;

}

public void addGoods(Item arg)

{

\_items.Add(arg);

}

// 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;

if (each.getGoods().GetType() == typeof(RegularGoods))

{

if (each.getQuantity() > 5)

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

} else if (each.getGoods().GetType() == typeof(SpecialGoods))

{

if (each.getQuantity() > 1)

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

}

return usedBonus;

}

public String GetBIll()

{

double totalAmount = 0;

int totalBonus = 0;

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

String result = view.GetHeader(\_customer);

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = each.GetDiscount();

int bonus = each.GetBonus();

//показать результаты

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

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

totalAmount += thisAmount;

totalBonus += bonus;

}

//добавить нижний колонтитул

result += view.GetFooter(totalAmount, totalBonus);

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return result;

}

}

**Заключение:**

Были изучены техники рефакторинга позволяющие выполнять отделение представления от остального кода.