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 №4

в рамках освоения дисциплины

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

**Тема: «Базовые техники рефакторинга. Обмен частей между классами»**

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 позволяющие выполнять отделение представления от остального кода.

**Задание:**

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

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

**1) Создаем иерархию классов представлений interface IView (с набором методов GetHeader/GetFooter/GetItemString)**

public interface IView

{

string GetHeader();

string GetItemString();

string GetFooter();

}

public class HtmlView : IView

{

public string GetFooter()

{

throw new NotImplementedException();

}

public string GetHeader()

{

throw new NotImplementedException();

}

public string GetItemString()

{

throw new NotImplementedException();

}

}

public class TxtView : IView

{

public string GetFooter()

{

throw new NotImplementedException();

}

public string GetHeader()

{

throw new NotImplementedException();

}

public string GetItemString()

{

throw new NotImplementedException();

}

}

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

public class TxtView : IView

{

//---Метод для получения оглавлений чека

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 получения списка товаров

public string GetItemString(Item each, double discount, double thisAmount, int bonus)

{

return "\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"; ;

}

//---Метод для получения нижнего колонтитула чека

public string GetFooter(double totalAmount, int totalBonus)

{

return "Сумма счета составляет " +

totalAmount.ToString() + "\n" +

"Вы заработали " + totalBonus.ToString() +

" бонусных балов";

}

}

**3) Создаем методы GetHeader/GetFooter/GetItemString для HtmlView**

public class HtmlView : IView

{

public string GetHeader(Customer \_customer)

{

throw new NotImplementedException();

}

public string GetItemString(Item each, double discount, double thisAmount, int bonus)

{

throw new NotImplementedException();

}

public string GetFooter(double totalAmount, int totalBonus)

{

throw new NotImplementedException();

}

}

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

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

IView view;

public Bill(Customer customer, IView view)

{

this.\_customer = customer;

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

this.view = view;

}

**6) Заменяем в тексте метода statement все вхождения**

**GetHeader/GetFooter/GetItemString на p.GetHeader/GetFooter/GetItemString**

//---Добавляем оглавления

string header = view.GetHeader(\_customer);

//---Вызов метода выводящего все товары

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

//---Добавляем нижний колонтитул

string footer = view.GetFooter(totalAmount, totalBonus);

**7) Заменяем создание класса Bill**

**Bill b = new Bill(customer); На IView view = new TxtView(); Bill b = new Bill(customer, view);**

[SetUp]

public void Setup()

{

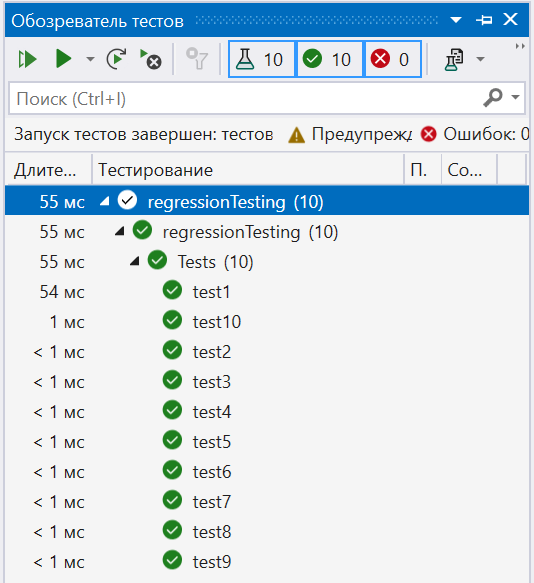
customer = new Customer("Тестовый клиент",10);

IView view = new TxtView();

bill = new Bill(customer, view);

}

**8) Все тесты должны проходить**

****

**9) Создаем свойство View для динамической замены представления.**

**10) Переименовать метод statement в более отражающий суть (например, GetBill/GenerateBill/GetOutput)**

public String GenerateBill()

{

double totalAmount = 0;

int totalBonus = 0;

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

//---Добавляем оглавления

string header = view.GetHeader(\_customer);

string itemString="";

while (items.MoveNext())

{

Item each = items.Current;

//------Определить сумму для каждой строки

//---Получаем скидку

double discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

double itemSum = GetSum(each);

double sumWithDiscount = itemSum - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

//---Вызов метода выводящего все товары

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

//---Добавление суммы и Накопление бонусов

totalAmount += thisAmount;

totalBonus += bonus;

}

//---Добавляем нижний колонтитул

string footer = view.GetFooter(totalAmount, totalBonus);

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return header + itemString + footer;

}

**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 получения оглавлений чека

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 получения списка товаров

public string GetItemString(Item each, double discount, double thisAmount, int bonus)

{

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

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

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

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

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

}

//---Метод для получения нижнего колонтитула чека

public string GetFooter(double totalAmount, int totalBonus)

{

return "Сумма счета составляет " +

totalAmount.ToString() + "\n" +

"Вы заработали " + totalBonus.ToString() +

" бонусных балов";

}

//---Метод получения использованных бонусов

public double GetUsedBonus(Item each,double sumWithDiscount)

{

double usedBonus = 0;

switch (each.getGoods().GetType())

{

//Обычный товав

case Type t when t == typeof(RegularGoods):

if (each.getQuantity() > 5)

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

break;

//Специальное предложение

case Type t when t == typeof(SpecialGoods):

if (each.getQuantity() > 1)

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

break;

}

return usedBonus;

}

//---Метод получения цены с учётом количества

public double GetSum(Item each)

{

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

}

public String statement()

{

double totalAmount = 0;

int totalBonus = 0;

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

//---Добавляем оглавления

string header = GetHeader(); //заменить result

string itemString="";

while (items.MoveNext())

{

Item each = items.Current;

//------Определить сумму для каждой строки

//---Получаем скидку

double discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

double itemSum = GetSum(each);

double sumWithDiscount = itemSum - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

//---Вызов метода собирающего все товары

itemString += GetItemString(each, discount+usedBonus, thisAmount, bonus);

//---Добавление суммы и Накопление бонусов

totalAmount += thisAmount;

totalBonus += bonus;

}

//---Добавляем нижний колонтитул

string footer = GetFooter(totalAmount, totalBonus);

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return header + itemString + footer;

}

}

**После:**

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

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 получения использованных бонусов

public double GetUsedBonus(Item each,double sumWithDiscount)

{

double usedBonus = 0;

switch (each.getGoods().GetType())

{

//Обычный товав

case Type t when t == typeof(RegularGoods):

if (each.getQuantity() > 5)

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

break;

//Специальное предложение

case Type t when t == typeof(SpecialGoods):

if (each.getQuantity() > 1)

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

break;

}

return usedBonus;

}

//---Метод получения цены с учётом количества

public double GetSum(Item each)

{

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

}

public String GenerateBill()

{

double totalAmount = 0;

int totalBonus = 0;

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

//---Добавляем оглавления

string header = view.GetHeader(\_customer);

string itemString="";

while (items.MoveNext())

{

Item each = items.Current;

//------Определить сумму для каждой строки

//---Получаем скидку

double discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

double itemSum = GetSum(each);

double sumWithDiscount = itemSum - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

//---Вызов метода выводящего все товары

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

//---Добавление суммы и Накопление бонусов

totalAmount += thisAmount;

totalBonus += bonus;

}

//---Добавляем нижний колонтитул

string footer = view.GetFooter(totalAmount, totalBonus);

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return header + itemString + footer;

}

}

public interface IView

{

string GetHeader(Customer \_customer);

string GetItemString(Item each, double discount, double thisAmount, int bonus);

string GetFooter(double totalAmount, int totalBonus);

}

public class TxtView : IView

{

//---Метод для получения оглавлений чека

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 получения списка товаров

public string GetItemString(Item each, double discount, double thisAmount, int bonus)

{

return "\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"; ;

}

//---Метод для получения нижнего колонтитула чека

public string GetFooter(double totalAmount, int totalBonus)

{

return "Сумма счета составляет " +

totalAmount.ToString() + "\n" +

"Вы заработали " + totalBonus.ToString() +

" бонусных балов";

}

}

public class HtmlView : IView

{

public string GetHeader(Customer \_customer)

{

throw new NotImplementedException();

}

public string GetItemString(Item each, double discount, double thisAmount, int bonus)

{

throw new NotImplementedException();

}

public string GetFooter(double totalAmount, int totalBonus)

{

throw new NotImplementedException();

}

}

**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.

**Заключение**

В результате выполнения лабораторной работы были изучены техники рефакторинга позволяющие выполнять отделение представления от остального кода.