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 №5

в рамках освоения дисциплины

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

**Тема: «Сложные техники рефакторинга.**

**Выделение модели предметной области»**

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 модели предметной области от непосредственно процесса генерации чека.

**Задание:**

1) Преобразовать методы и классы программы в соответствии с предложенными в пункте 2.1 изменениями, применив описанные в теории методы рефакторинга.

2) Проверить корректность работы новых классов и работоспособность формирования счета.

**1) Создаем представления модели предметной области BillSummary и ItemSummary, добавляя в них необходимые для отображения поля.**

public class ItemSummary

{

public string Name { get; set; }

public decimal Price { get; set; }

public double Quantity { get; set; }

public decimal Sum { get; set; }

public decimal Discount { get; set; }

public int Bonus { get; set; }

}

public class BillSummary

{

public decimal TotalAmount { get; set; }

public decimal TotalDiscount { get; set; }

public string CustomerName { get; set; }

public int TotalBonus { get; set; }

public List<ItemSummary> Items { get; set; }

}

**2) Разделяем класс Bill: создаем класс BillGenerator и Копируем код бывшего класса Bill в BillGenerator (по сути, должна получиться копия оригинального Bill).**

public class BillGenerator

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public IView View

{

get { return view; }

set { view = value; }

}

public BillGenerator(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;

}

}

**3) В классе BillGenerator делаем ссылку на Bill**

public class BillGenerator

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

private Bill \_bill;

public IView View

{

get { return view; }

set { view = value; }

}

public BillGenerator(Customer customer, IView view, Bill bill)

{

this.\_customer = customer;

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

this.view = view;

\_bill = bill;

}

**4) Переименовываем бывший метод statement в классе Bill в метод Process. Данный метод должен возвращать BillSummary в качестве результата.**

public BillSummary Process()

{

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;

}

**5) Убираем из метода Process все аспекты, касающиеся генерации чека (вызовы view и т.д.).**

public BillSummary Process()

{

double totalAmount = 0;

int totalBonus = 0;

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

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;

//---Добавление суммы и Накопление бонусов

totalAmount += thisAmount;

totalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return header + itemString + footer;

}

**6) Создаем внутри Process экземпляр класса BillSummary и заменяем использование локальных переменных (totalAmount/totalDiscount и т.д.) на использование полей BillSummary.**

public BillSummary Process()

{

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

var billSummary = new BillSummary{};

while (items.MoveNext())

{

Item each = items.Current;

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

//---Получаем скидку

decimal discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

decimal itemSum = GetSum(each);

decimal sumWithDiscount = itemSum - discount;

int usedBonus = GetUsedBonus(each, sumWithDiscount);

decimal thisAmount = sumWithDiscount - usedBonus;

//---Добавление суммы и Накопление бонусов

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

billSummary.CustomerName=\_customer.getName();

return billSummary;

}

**7) Заменяем в методе Process все используемые для генерации Item переменные на поля ItemSummary и добавляем полученные элементы в BillSummary.**

public BillSummary Process()

{

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

var billSummary = new BillSummary { CustomerName = \_customer.getName() };

while (items.MoveNext())

{

Item each = items.Current;

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

//---Получаем скидку

decimal discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

decimal itemSum = GetSum(each);

decimal sumWithDiscount = itemSum - discount;

int usedBonus = GetUsedBonus(each, sumWithDiscount);

decimal thisAmount = sumWithDiscount - usedBonus;

ItemSummary itemSummary = new ItemSummary

{

Name = each.getGoods().getTitle(),

Price = each.getPrice(),

Quantity = each.getQuantity(),

Sum = sumWithDiscount,

Discount = discount,

Bonus = bonus

};

billSummary.ItemSummaries.Add(itemSummary);

//---Добавление суммы и Накопление бонусов

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

return billSummary;

}

**8) Удаляем лишний неиспользуемый код в классе Bill.**

**9) В классе BillGenerator в начале метода бывшего метода statement получаем экземпляр BillSummary из метода Process класса Bill.**

public String GenerateBill()

{

Bill bill = new Bill(\_customer, view);

BillSummary summary = bill.Process();

decimal 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

decimal discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

decimal itemSum = GetSum(each);

decimal sumWithDiscount = itemSum - discount;

int usedBonus = GetUsedBonus(each, sumWithDiscount);

decimal 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;

}

**10) Удаляем весь код связанный с вычислениями**

**11) Заменяем все места использования локальных переменных (totalAmount/totalDiscount и т.д.) на использование полей BillSummary и ItemSummary.**

public String GenerateBill()

{

Bill bill = new Bill(\_customer, view);

foreach (Item item in \_items)

{

bill.addGoods(item);

}

BillSummary billSummary = bill.Process();

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

//---Добавляем оглавления

string headerAndItemString = view.GetHeader(\_customer);

int index = 0; // индекс продукта

foreach (var itemSummary in billSummary.ItemSummaries)

{

if(index < \_items.Count) // Проверка на IndexOutOfRangeException

{

Item each = \_items[index];

//Добавляем товары в чек

headerAndItemString += view.GetItemString(each,

itemSummary.Discount,

itemSummary.Sum,

itemSummary.Bonus);

index++;

}

}

//---Добавляем нижний колонтитул

string footer = view.GetFooter(billSummary.TotalAmount, billSummary.TotalBonus);

return headerAndItemString + footer;

}

**12) Удаляем лишний неиспользуемый код в классе BillGenerator**

public class BillGenerator

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

private Bill \_bill;

public IView View

{

get { return view; }

set { view = value; }

}

public BillGenerator(Customer customer, IView view)

{

this.\_customer = customer;

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

this.view = view;

}

//---Метод для генерации чека

public String GenerateBill()

{

Bill bill = new Bill(\_customer, view);

foreach (Item item in \_items)

{

bill.addGoods(item);

}

BillSummary billSummary = bill.Process();

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

//---Добавляем оглавления

string headerAndItemString = view.GetHeader(\_customer);

int index = 0; // индекс продукта

foreach (var itemSummary in billSummary.ItemSummaries)

{

if(index < \_items.Count) // Проверка на IndexOutOfRangeException

{

Item each = \_items[index];

//Добавляем товары в чек

headerAndItemString += view.GetItemString(each,

itemSummary.Discount,

itemSummary.Sum,

itemSummary.Bonus);

index++;

}

}

//---Добавляем нижний колонтитул

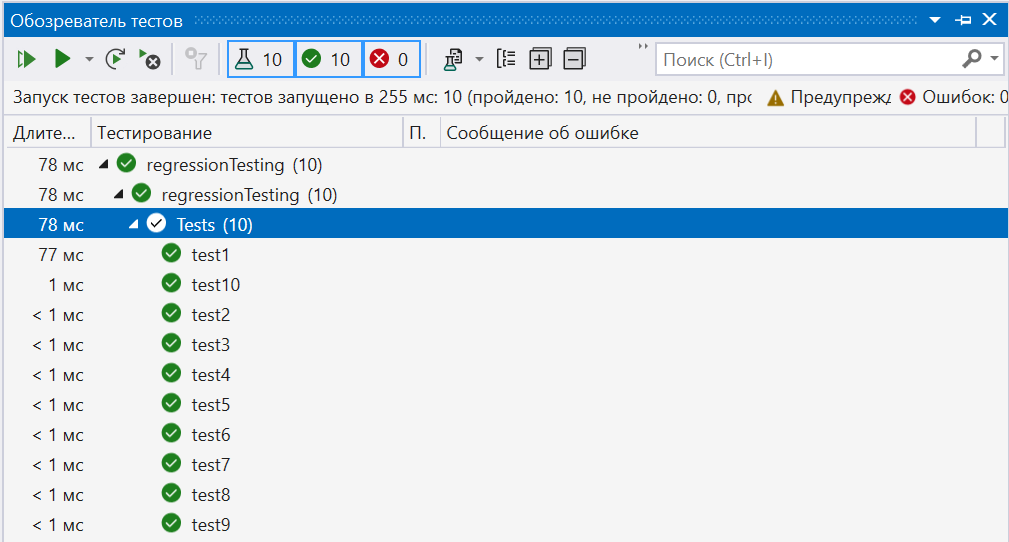
string footer = view.GetFooter(billSummary.TotalAmount, billSummary.TotalBonus);

return headerAndItemString + footer;

}

}

**13) Все тесты должны проходить**

****

**До:**

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public IView View

{

get { return view; }

set { view = value; }

}

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;

}

}

**После:**

public class Bill

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public IView View

{

get { return view; }

set { view = value; }

}

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 int GetUsedBonus(Item each, decimal sumWithDiscount)

{

int 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 decimal GetSum(Item each)

{

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

}

public BillSummary Process()

{

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

var billSummary = new BillSummary { CustomerName = \_customer.getName(),

ItemSummaries=new List<ItemSummary>()};

while (items.MoveNext())

{

Item each = items.Current;

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

//---Получаем скидку

decimal discount = each.GetDiscount();

//---Получаем бонусы

int bonus = each.GetBonus();

//---Учитываем скидку и бонусы

decimal itemSum = GetSum(each);

decimal sumWithDiscount = itemSum - discount;

int usedBonus = GetUsedBonus(each, sumWithDiscount);

decimal thisAmount = sumWithDiscount - usedBonus;

ItemSummary itemSummary = new ItemSummary

{

Name = each.getGoods().getTitle(),

Price = each.getPrice(),

Quantity = each.getQuantity(),

Sum = sumWithDiscount,

Discount = discount,

Bonus = bonus

};

billSummary.ItemSummaries.Add(itemSummary);

//---Добавление суммы и Накопление бонусов

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

return billSummary;

}

}

public class BillSummary

{

public decimal TotalAmount { get; set; }

public decimal TotalDiscount { get; set; }

public string CustomerName { get; set; }

public int TotalBonus { get; set; }

public List<ItemSummary> ItemSummaries { get; set; }

}

public class ItemSummary

{

public string Name { get; set; }

public decimal Price { get; set; }

public double Quantity { get; set; }

public decimal Sum { get; set; }

public decimal Discount { get; set; }

public int Bonus { get; set; }

}

public class BillGenerator

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

private Bill \_bill;

public IView View

{

get { return view; }

set { view = value; }

}

public BillGenerator(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 генерации чека

public String GenerateBill()

{

Bill bill = new Bill(\_customer, view);

foreach (Item item in \_items)

{

bill.addGoods(item);

}

BillSummary billSummary = bill.Process();

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

//---Добавляем оглавления

string headerAndItemString = view.GetHeader(\_customer);

int index = 0; // индекс продукта

foreach (var itemSummary in billSummary.ItemSummaries)

{

if(index < \_items.Count) // Проверка на IndexOutOfRangeException

{

Item each = \_items[index];

//Добавляем товары в чек

headerAndItemString += view.GetItemString(each,

itemSummary.Discount,

itemSummary.Sum,

itemSummary.Bonus);

index++;

}

}

//---Добавляем нижний колонтитул

string footer = view.GetFooter(billSummary.TotalAmount, billSummary.TotalBonus);

return headerAndItemString + footer;

}

}

**Контрольные вопросы**

**1) В чем особенность выделения модели предметной области при проведении сложных рефакторингов?**

Особенность выделения модели предметной области при сложных рефакторингах заключается в возможности централизовать и изолировать бизнес-логику и правила домена от других аспектов приложения, таких как пользовательский интерфейс или инфраструктура. Это упрощает тестирование и поддержку бизнес-логики, а также уменьшает риск внесения ошибок при изменении кода, так как изменения в одной части системы меньше влияют на другие части.

**2) Для чего необходимо отделять модель предметной области от модели представления?**

Отделение модели предметной области от модели представления необходимо для обеспечения гибкости и масштабируемости приложения. Это позволяет разработчикам изменять пользовательский интерфейс или способ отображения данных без риска повлиять на основную бизнес-логику. Такой подход также облегчает реализацию различных представлений для одних и тех же данных, поддерживает чистоту кода и упрощает его тестирование.

**3) Что такое Data Transfer Objects?**

Это простые объекты, которые содержат данные и используются для передачи информации между подсистемами приложения, например, между слоем бизнес-логики и слоем представления или между клиентом и сервером. DTO не содержат бизнес-логику, кроме возможности сериализации и десериализации, и предназначены для уменьшения количества вызовов методов, необходимых для передачи данных, упаковывая множество данных в один объект.

**Заключение**

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