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 модели предметной области»**

**Лабораторная: 5**

Работу выполнил: Басма Нассиф Хуссейн

Направление подготовки: 09.03.04 Программная инженерия

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

**Цель работы:** изучить техники рефакторинга позволяющие выполнять отделение модели предметной области от непосредственно процесса генерации чека.

**Контрольные вопросы**

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

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

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

**Задание**

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

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

public class BillSummary

{

public decimal TotalAmount;

public decimal TotalDiscount;

public string CustomerName;

public int TotalBonus;

public BillSummary(decimal TotalAmount, decimal TotalDiscount,

string CustomerName, int TotalBonus)

{

this.TotalAmount = TotalAmount;

this.TotalDiscount = TotalDiscount;

this.CustomerName = CustomerName;

this.TotalBonus = TotalBonus;

}

}

public class ItemSummary

{

public string Name;

public decimal Price;

public double Quantity;

public decimal Sum;

public decimal Discount;

public int Bonus;

public ItemSummary(string name, decimal price, double quantity,

decimal sum, decimal discount, int bonus)

{

Name = name;

Price = price;

Quantity = quantity;

Sum = sum;

Discount = discount;

Bonus = bonus;

}

}

1. **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 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;

}

}

1. **В классе BillGenerator делаем в нем делаем ссылку на Bill**

private Bill bill;

1. **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;

double totalDiscount = 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;

totalDiscount += discount;

totalBonus += bonus;

}

//добавить нижний колонтитул

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

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return new BillSummary(totalAmount, totalDiscount, \_customer.getName(), totalBonus);

}

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

public BillSummary Process()

{

double totalAmount = 0;

int totalBonus = 0;

double totalDiscount = 0;

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

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;

totalAmount += thisAmount;

totalDiscount += discount;

totalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(totalBonus);

return new BillSummary(totalAmount, totalDiscount, \_customer.getName(), totalBonus);

}

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

public BillSummary Process()

{

BillSummary billSummary = new BillSummary();

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

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;

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

billSummary.CustomerName = \_customer.getName();

return billSummary;

}

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

public BillSummary Process()

{

BillSummary billSummary = new BillSummary();

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

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = each.GetDiscount();

int bonus = each.GetBonus();

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double thisAmount = sumWithDiscount - usedBonus;

ItemSummary itemSummary = new ItemSummary(

name: each.getGoods().getTitle(),

price: (decimal)each.getPrice(),

quantity: each.getQuantity(),

sum: (decimal)sumWithDiscount,

discount: (decimal)discount,

bonus: bonus

);

billSummary.ItemSummaries.Add(itemSummary);

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

return billSummary;

}

1. **Удаляем лишний неиспользуемый код в классе Bill.**
2. **В классе BillGenerator в начале метода бывшего метода statement получаем экземпляр BillSummary из метода Process класса Bill.**

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

BillSummary billSummary = bill.Process();

1. **Удаляем весь код связанный с вычислениями**
2. **Заменяем все места использования локальных переменных (totalAmount/totalDiscount и т.д.) на использование полей BillSummary и ItemSummary.**

public String GetBIll()

{

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

foreach (Item item in \_items)

{

bill.addGoods(item);

}

BillSummary billSummary = bill.Process();

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

String result = view.GetHeader(\_customer);

int index = 0; // Initialize an index for parallel access

foreach (var itemSummary in billSummary.ItemSummaries)

{

if (index < \_items.Count) // Check to avoid IndexOutOfRangeException

{

Item each = \_items[index]; // Access the corresponding Item

result += view.GetItemString(

each, // Pass the Item object

itemSummary.Discount,

itemSummary.Sum,

itemSummary.Bonus);

index++; // Move to the next item

}

}

result += view.GetFooter(billSummary.TotalAmount, billSummary.TotalBonus);

return result;

}

1. **Удаляем лишний неиспользуемый код в классе BillGenerator**

public class BillGenerator

{

private List<Item> \_items;

private Customer \_customer;

private IView view;

public void addGoods(Item arg)

{

\_items.Add(arg);

}

public String GetBIll()

{

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

foreach (Item item in \_items)

{

bill.addGoods(item);

}

BillSummary billSummary = bill.Process();

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

String result = view.GetHeader(\_customer);

int index = 0; // Initialize an index for parallel access

foreach (var itemSummary in billSummary.ItemSummaries)

{

if (index < \_items.Count) // Check to avoid IndexOutOfRangeException

{

Item each = \_items[index]; // Access the corresponding Item

result += view.GetItemString(

each, // Pass the Item object

itemSummary.Discount,

itemSummary.Sum,

itemSummary.Bonus);

index++; // Move to the next item

}

}

result += view.GetFooter(billSummary.TotalAmount, billSummary.TotalBonus);

return result;

}

}

1. **Все тесты должны проходить**



Рисунок 1 – Результат тестирование

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

**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 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;

}

}

**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 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 BillSummary Process()

{

BillSummary billSummary = new BillSummary();

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

while (items.MoveNext())

{

Item each = (Item)items.Current;

//определить сумму для каждой строки

double discount = each.GetDiscount();

int bonus = each.GetBonus();

double sumWithDiscount = GetSum(each) - discount;

double usedBonus = GetUsedBonus(each, sumWithDiscount);

double 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);

billSummary.TotalAmount += thisAmount;

billSummary.TotalDiscount += discount;

billSummary.TotalBonus += bonus;

}

//Запомнить бонус клиента

\_customer.receiveBonus(billSummary.TotalBonus);

return billSummary;

}

}

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);

}

public String GetBIll()

{

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

foreach (Item item in \_items)

{

bill.addGoods(item);

}

BillSummary billSummary = bill.Process();

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

String result = view.GetHeader(\_customer);

int index = 0; // Initialize an index for parallel access

foreach (var itemSummary in billSummary.ItemSummaries)

{

if (index < \_items.Count) // Check to avoid IndexOutOfRangeException

{

Item each = \_items[index]; // Access the corresponding Item

result += view.GetItemString(

each, // Pass the Item object

itemSummary.Discount,

itemSummary.Sum,

itemSummary.Bonus);

index++; // Move to the next item

}

}

result += view.GetFooter(billSummary.TotalAmount, billSummary.TotalBonus);

return result;

}

}

public class BillSummary

{

public double TotalAmount;

public double TotalDiscount;

public string CustomerName;

public int TotalBonus;

public List<ItemSummary> ItemSummaries { get; } = new List<ItemSummary>();

public BillSummary(double TotalAmount, double TotalDiscount,

string CustomerName, int TotalBonus)

{

this.TotalAmount = TotalAmount;

this.TotalDiscount = TotalDiscount;

this.CustomerName = CustomerName;

this.TotalBonus = TotalBonus;

}

public BillSummary() { }

}

public class ItemSummary

{

public string Name;

public double Price;

public double Quantity;

public double Sum;

public double Discount;

public int Bonus;

public ItemSummary(string name, double price, double quantity,

double sum, double discount, int bonus)

{

Name = name;

Price = price;

Quantity = quantity;

Sum = sum;

Discount = discount;

Bonus = bonus;

}

}

**Заключение:**

Были изучены техники рефакторинга позволяющие выполнять отделение модели предметной области от непосредственно процесса генерации чека.