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.**

**Обмен частей между классами»**

**Лабораторная: 2**

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 между классами программы.

**Контрольные вопросы**

**1) В чем заключается рефакторинг «Перемещение метода»?**

Заключается в переносе метода в класс, где он более уместен из-за более тесной связи с данными этого класса. Это делается для улучшения структуры кода, упрощения взаимодействия между классами и повышения читабельности. Если метод больше взаимодействует с данными другого класса, его следует переместить в этот класс. Процесс включает в себя анализ зависимостей метода, определение наиболее подходящего класса для его размещения, копирование тела метода в новый класс с необходимой адаптацией и замену исходного метода делегированием вызова или его удалением.

**2) В чем заключается рефакторинг «Перемещение поля»?**

Предполагает перенос поля из одного класса в другой, если это поле чаще используется или будет использоваться вторым классом. Этот процесс включает в себя создание нового поля в целевом классе, перенаправление всех обращений к этому полю на новое местоположение, и удаление поля из исходного класса после переадресации для обеспечения целостности и актуальности ссылок в коде. Такой рефакторинг позволяет лучше организовать данные в системе, упростить доступ к данным и повысить гибкость и расширяемость архитектуры программного обеспечения.

**3) В чем суть рефакторинга «Введение внешнего метода»?**

Применяется, когда необходимо добавить новый метод к классу-серверу, но нет возможности изменить его исходный код. Вместо этого, новый метод реализуется в классе-клиенте, принимая экземпляр класса-сервера как аргумент. Этот подход помогает расширить функциональность класса без его изменения, избегая дублирования кода при многократном использовании нового метода в клиентском коде. Если потребность в таких внешних методах возникает часто, рекомендуется рассмотреть другие рефакторинги, например «Введение локального расширения», для более структурированного и удобного управления расширениями функциональности.

**4) В чем суть рефакторинга «Введение локального расширения»?**

Применяется, когда необходимо добавить новые методы к существующему классу, который нельзя модифицировать. Решение заключается в создании нового класса, который либо наследует существующий класс (подкласс), либо оборачивает его (wrapper), предоставляя дополнительные методы. Этот подход позволяет расширять функциональность без изменения оригинального кода, поддерживая чистоту и организованность кодовой базы и облегчая повторное использование кода. Локальное расширение полезно, когда требуется более одного нового метода, объединяя новые функции в одном месте и избегая разбросанности кода.

**5) В чем суть рефакторинга «Сокрытие делегирования»?**

Заключается в добавлении методов в класс-сервер для скрытия его внутренних делегирований, уменьшая тем самым зависимость клиентских классов от структуры и взаимодействия внутренних объектов сервера. Это улучшает инкапсуляцию, сокращает связанность компонентов системы и облегчает последующие изменения, поскольку модификация внутренних отношений класса не влияет на клиентские классы.

**6) В чем суть рефакторинга «Удаление посредника»?**

Предполагает устранение классов, которые выполняют только функцию делегирования, позволяя клиентам напрямую обращаться к делегату. Это уменьшает количество ненужных промежуточных вызовов, упрощает структуру программы и повышает её прозрачность, делая взаимодействие между объектами более очевидным.

**Задание**

**1) Преобразовать метод statement класса Bill, и перенести методы получения скидки и начисления бонусов в класс Goods в соответствии с предложенными в пункте 2.2 изменениями, применив описанные в теории методы рефакторинга.**

**Порядок выполнение**

1) Переносим (копируем) методы GetBonus в Goods делая их public.public class Goods

{

public const int REGULAR = 0;

public const int SALE = 1;

public const int SPECIAL\_OFFER = 2;

private String \_title;

private int \_priceCode;

public Goods(String title, int priceCode)

{

\_title = title;

\_priceCode = priceCode;

}

public int getPriceCode()

{

return \_priceCode;

}

public void setPriceCode(int arg)

{

\_priceCode = arg;

}

public String getTitle()

{

return \_title;

}

// Метод для получения бонуса

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;

}

// 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%

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%

break;

}

return discount;

}

}

2) Заменяем в statement GetBonus(each) на each.GetGoods().GetBonus(each)

//определить сумму для каждой строки

double discount = each.getGoods().GetDiscount(each);

int bonus = each.getGoods().GetBonus(each);

3) Создаем proxy-метод (сокрытие делегирования) GetBonus() в классе Item для избавления от GetGoods().// Метод для сокрытия делегирования GetBonus

public int GetBonus(Item each)

{

return \_Goods.GetBonus(each);

}

// Метод для сокрытия делегирования GetDiscount

public double GetDiscount(Item each)

{

return \_Goods.GetDiscount(each);

}

4) Так как обрабатываемый объект класса Item: each и передаваемый параметр в GetBonus(each) суть одно и тоже, то можно удалить each из списка параметров, а внутри метода заменить его на this.

// Метод для сокрытия делегирования GetBonus

public int GetBonus()

{

return \_Goods.GetBonus(this);

}

// Метод для сокрытия делегирования GetDiscount

public double GetDiscount()

{

return \_Goods.GetDiscount(this);

}

5) Далее можно избавиться от this заменой его 2-мя параметрами:

// Метод для сокрытия делегирования GetBonus

public int GetBonus()

{

return \_Goods.GetBonus(\_quantity, \_price);

// Метод для сокрытия делегирования GetDiscount

}

public double GetDiscount()

{

return \_Goods.GetDiscount(\_quantity, \_price);

}

// GOODS

// Метод для получения бонуса

public int GetBonus(int \_quantity,double \_price)

{

switch (\_priceCode)

{

case Goods.REGULAR:

return (int)(\_quantity \* \_price \* 0.05);

case Goods.SALE:

return (int)(\_quantity \* \_price \* 0.01);

}

return 0;

}

// Метод для получения скидки

public double GetDiscount(int \_quantity, double \_price)

{

double discount = 0;

switch (\_priceCode)

{

case Goods.REGULAR:

if (\_quantity > 2)

discount = (\_quantity \* \_price) \* 0.03; // 3%

break;

case Goods.SPECIAL\_OFFER:

if (\_quantity > 10)

discount = (\_quantity \* \_price) \* 0.005; // 0.5%

break;

case Goods.SALE:

if (\_quantity > 3)

discount = (\_quantity \* \_price) \* 0.01; // 0.1%

break;

}

return discount;

}

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

**До:**

// BILL

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;

}

// GOODS

public class Goods

{

public const int REGULAR = 0;

public const int SALE = 1;

public const int SPECIAL\_OFFER = 2;

private String \_title;

private int \_priceCode;

public Goods(String title, int priceCode)

{

\_title = title;

\_priceCode = priceCode;

}

public int getPriceCode()

{

return \_priceCode;

}

public void setPriceCode(int arg)

{

\_priceCode = arg;

}

public String getTitle()

{

return \_title;

}

}

// ITEM

// Класс, представляющий данные о чеке

public class Item

{

private Goods \_Goods;

private int \_quantity;

private double \_price;

public Item(Goods Goods, int quantity, double price)

{

\_Goods = Goods;

\_quantity = quantity;

\_price = price;

}

public int getQuantity()

{

return \_quantity;

}

public double getPrice()

{

return \_price;

}

public Goods getGoods()

{

return \_Goods;

}

}

**После:**

// BILL

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;

}

// GOODS

public class Goods

{

public const int REGULAR = 0;

public const int SALE = 1;

public const int SPECIAL\_OFFER = 2;

private String \_title;

private int \_priceCode;

public Goods(String title, int priceCode)

{

\_title = title;

\_priceCode = priceCode;

}

public int getPriceCode()

{

return \_priceCode;

}

public void setPriceCode(int arg)

{

\_priceCode = arg;

}

public String getTitle()

{

return \_title;

}

// Метод для получения бонуса

public int GetBonus(int \_quantity,double \_price)

{

switch (\_priceCode)

{

case Goods.REGULAR:

return (int)(\_quantity \* \_price \* 0.05);

case Goods.SALE:

return (int)(\_quantity \* \_price \* 0.01);

}

return 0;

}

// Метод для получения скидки

public double GetDiscount(int \_quantity, double \_price)

{

double discount = 0;

switch (\_priceCode)

{

case Goods.REGULAR:

if (\_quantity > 2)

discount = (\_quantity \* \_price) \* 0.03; // 3%

break;

case Goods.SPECIAL\_OFFER:

if (\_quantity > 10)

discount = (\_quantity \* \_price) \* 0.005; // 0.5%

break;

case Goods.SALE:

if (\_quantity > 3)

discount = (\_quantity \* \_price) \* 0.01; // 0.1%

break;

}

return discount;

}

}

// ITEM

// Класс, представляющий данные о чеке

public class Item

{

private Goods \_Goods;

private int \_quantity;

private double \_price;

public Item(Goods Goods, int quantity, double price)

{

\_Goods = Goods;

\_quantity = quantity;

\_price = price;

}

public int getQuantity()

{

return \_quantity;

}

public double getPrice()

{

return \_price;

}

public Goods getGoods()

{

return \_Goods;

}

// Метод для сокрытия делегирования GetBonus

public int GetBonus()

{

return \_Goods.GetBonus(\_quantity, \_price);

// Метод для сокрытия делегирования GetDiscount

}

public double GetDiscount()

{

return \_Goods.GetDiscount(\_quantity, \_price);

}

}

**Заключение:**

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