Лабораторная работа №24
Рефакторинг кода

1 Цель работы

1.1 Изучить и применить техники рефакторинга программного кода.

2 Литература

2.1 Фленов, М. Е. Библия C# / М. Е. Фленов. – 5-е изд. – Санкт-Петербург : БХВ-Петербург, 2022. – 464 с. – URL: https://ibooks.ru/bookshelf/380047/reading. – Режим доступа: для зарегистрир. пользователей. – Текст : электронный. – гл.16.

3 Подготовка к работе

3.1 Повторить теоретический материал (см.п.2).

3.2 Изучить описание лабораторной работы.

4 Основное оборудование

4.1 Персональный компьютер.

5 Задание

В файле OrderManagementApp.cs представлен исходный код приложения для рефакторинга. После проверки работоспособности проекта выполнить его рефакторинг по заданию.

5.1 Рефакторинг переменных

5.1.1 Инкапсуляция поля (Encapsulate Field)

В классе "Order" поля являются публичными, что позволяет любому коду изменить их значения напрямую.

Нужно инкапсулировать поле через свойство и добавить проверку, чтобы значение адреса не было пустым, а стоимость — отрицательной.

5.1.2 Замена магического числа (Replace Magic Number with Constant)

В классе "OrderService" метод "CalculateFinalPrice" использует "магические числа".

Нужно создать константы с осмысленным названием, чтобы их назначение было понятно другим разработчикам.

5.1.3 Переименование переменной (Rename variables)

В классе "OrderService" метод "CalculateFinalPrice" использует переменную t и delCost.

Нужно присвоить переменным более осмысленные названия, чтобы их назначение было понятно другим разработчикам.

5.2 Упрощение условных выражений

5.2.1 Упрощение условного выражения (Simplify Conditional Expression)

В методе "CalculateFinalPrice" нужно сделать условное выражение для расчета скидки более компактным с помощью тернарного оператора.

5.2.2 Замена вложенных условий на guard clauses (Replace Nested Conditional with Guard Clauses)

Метод "PrintOrderDetails" проверяет заказ на null и выводит данные, если он не равен "null".

Нужно изменить этот код, чтобы использовать guard clauses и сразу выйти из метода, если заказ равен "null". Перед выходом вынести сообщение, что указанный заказ не найден.

5.3 Замена кода ошибки исключением

5.3.1 Предусмотреть в методе "CalculateFinalPrice" генерацию исключения типа ArgumentException вместо возврата -1 при отсутствии адреса.

5.3.2 Добавить вместо стандартного исключения пользовательское с названием типа OrderException и вызывать его.

5.4 Составление методов и упрощение вызовов

5.4.1 Извлечение метода (Extract Method)

Метод "CalculateFinalPrice" решает несколько задач: расчитывает налог и стоимость доставки, а затем — итоговую стоимость с учётом выполненных расчетов.

Метод "PrintOrderDetails" решает несколько задач: получает заказ по номеру и выводит информацию о нем.

Нужно разделить эти задачи, выделив каждую часть в отдельные методы.

5.4.2 Переименование метода (Rename Method)

Нужно присвоить методу "CalculateFinalPrice" более осмысленное название, чтобы его назначение было понятно другим разработчикам.

5.5 Перемещение функций между объектами

5.5.1 Перемещение метода (Move Method)

Перенести логику расчета стоимости в класс Order.

5.5.2 Извлечение класса (Extract Class)

Перенести логику расчета стоимости доставки в отдельный класс

6 Порядок выполнения работы

6.1 Запустить MS Visual Studio и создать приложение C#. Выполнить все задания из п.5 в проекте LabWork24. При разработке считать, что пользователь ввел данные требуемого типа, остальные возможные ошибки обрабатывать.

6.2 При выполнении заданий использовать минимально возможное количество команд и переменных и выполнять форматирование и рефакторинг кода. Все наименования должны быть на английском языке в PascalCase/camelCase.

6.3 Ответить на контрольные вопросы.

7 Содержание отчета

7.1 Титульный лист

7.2 Цель работы

7.3 Ответы на контрольные вопросы

7.4 Вывод

8 Контрольные вопросы

8.1 Что такое «рефакторинг»?

8.2 Какие группы техник рефакторинга существуют?

8.3 Как выполнить рефакторинг в Visual Studio?