**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П.О.СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальности 6-05-0611-01 Информационные системы

и технологии (в игровой индустрии)

ЛАБОРАТОРНАЯ РАБОТА №7

по дисциплине: «Объектно-ориентированное проектирование и программирование»

на тему: **«Наследование классов»**

Выполнил: студент гр. ИТИ-21

Спиридонов Д.А.

Принял: ассистент

Панарин К.А.

Гомель 2024

# **Задание:**

1. Необходимо решить задачу, согласно варианта (таблица 1).
2. При создании классов руководствоваться Code Convention.
3. Весь код должен содержать элементы документирования.
4. При реализации иерархии классов использовать механизм наследования.
5. Разработать модульные тесты для верификации созданных классов.
6. Классы должен быть размещены в библиотеке классов.
7. Модульные тесты - в отдельном проекте.
8. В отдельном проекте реализовать интерфейс WPF.
9. WPF-приложение должно обеспечивать ввод, редактирование и просмотр данных.

Таблица 1

|  |  |
| --- | --- |
| Вариант | Условие задачи |
| 14 | 1.1 Создать иерархию классов для учета выполняемых работ шиномонтажной мастерской 1.2 Мастерская выполняет следующие виды работ: замена шин, ремонт проколов, балансировка колёс 1.3 По каждой работе известна дата выполнения, для какого автомобиля она выполнялась 1.4 Внести информацию не менее, чем о 10 работах 1.5 Вывести информацию о среднем количестве выданного вида работ автомобилям 1.6 Для заданной марки автомобиля найти наиболее часто выполняемую работу 1.7 За указанный период по каждому виду работ вывести суммарную стоимость 1.8 Проводить некоторые скидки видов работ ("разные-скидки") - в этом нельзя проделывать и изменять методы, реализующие п. 1.5-1.7 |

**Ход работы:**

1. Разработка класса *ExamResult.cs* и перегрузки операций

На рисунке 1 представлен результат создания проекта с названием *TravelLibrary.c.* Класс отвечает за хранение информации и добавление поездок пользователя.

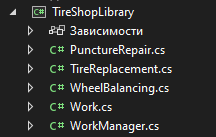


Рисунок 1 – класс *TravelLibrary.cs*

В коде реализована перегрузка оператора сложения для суммирования оценок двух студентов и перегрузка оператора вычитания для нахождения разности оценок двух студентов.

Листинг кода класса представлен в приложении А.

1. Разработка класса *Program.cs*

На рисунке 2 представлен результат создания класса с названием *Program.cs.* Это основной класс программы для ввода и обработки результатов экзаменов.

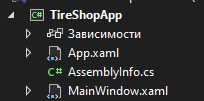


Рисунок 2 – проект *Windows Forms* *TravelApp.cs*

1. Разработка 15 исключительных модульных тестов.

На рисунке 3 представлен результат создания проекта с названием *TravelLibrary.Tests.c.* Это класс, отвечающий за полное модульное тестирование программы. В данном классе описано 15 модульных тестов.

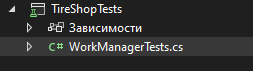


Рисунок 3 – проект модульных тестов *TravelLibrary.Tests.cs*

На рисунке 4 представлен результат создания всех необходимых проектов и классов.

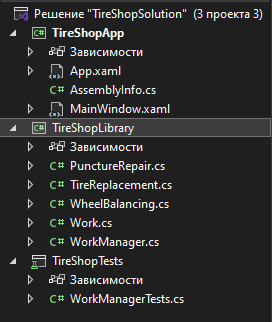


Рисунок 4 – Все описанные проекты, модульные тесты и классы в проекте решений

1. Верификация программы.

На рисунке 5 представлен результат прохождения всех модульных тестов, описанных в классе *CarNumberExtractorTests.cs*. Все тесты оказались успешными, что говорит о корректной работе программы.

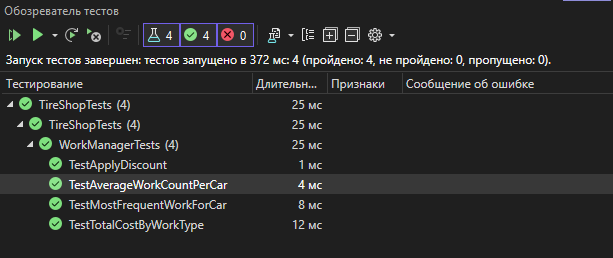


Рисунок 5 – Результаты выполнения модульных тестов

На рисунке 6 представлен результат выполнения программы пользователем.

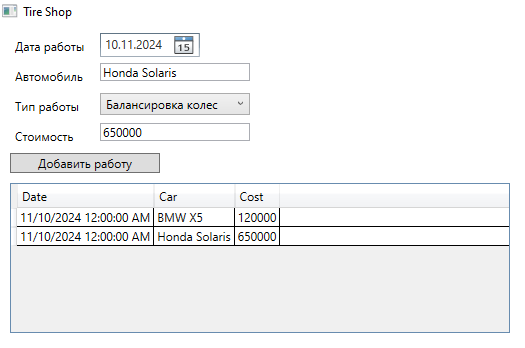


Рисунок 6 – Результаты выполнения программы в WPF

Листинг кода программ представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы были изучены методы и механизмы инкапсуляции и организации доступа к элементам объекта.

**ПРИЛОЖЕНИЕ А  
(обязательное)  
Листинг кода**

Файл *Program*.*cs*

using System;

using System.Collections.Generic;

using System.Text;

using CarNumberExtractorLib;

namespace CarNumberExtractorConsoleApp

{

class Program

{

static void Main(string[] args)

{

List<StringBuilder> reports = new List<StringBuilder>();

Console.WriteLine("Введите отчёты о ДТП (введите пустую строку для завершения ввода):");

while (true)

{

string input = Console.ReadLine();

if (string.IsNullOrWhiteSpace(input))

{

break;

}

reports.Add(new StringBuilder(input));

}

var extractor = new CarNumberExtractor();

var carNumbers = extractor.ExtractCarNumbers(reports);

Console.WriteLine("\nАвтомобильные номера и количество их упоминаний:");

foreach (var carNumber in carNumbers)

{

Console.WriteLine($"{carNumber.Key}: {carNumber.Value}");

}

}

}

}Файл *ExamResult*.*cs*

using System;

namespace ExamResultsLibrary

{

/// <summary>

/// Класс для хранения информации о результатах сдачи экзамена.

/// </summary>

public class ExamResult

{

public string StudentName { get; }

public int Score { get; }

public ExamResult(string studentName, int score)

{

StudentName = studentName;

Score = score;

}

/// <summary>

/// Перегрузка оператора сложения для суммирования оценок двух студентов.

/// </summary>

public static double operator +(ExamResult a, ExamResult b)

{

return (a.Score + b.Score) / 2.0; // Возвращает средний балл

}

/// <summary>

/// Перегрузка оператора вычитания для нахождения разности оценок двух студентов.

/// </summary>

public static int operator -(ExamResult a, ExamResult b)

{

return a.Score - b.Score; // Возвращает разность оценок

}

}

}

Файл *ExamModuleTest*.*cs*

using Microsoft.VisualStudio.TestTools.UnitTesting;

using ExamResultsLibrary;

namespace ExamResultsTests

{

[TestClass]

public class ExamModuleTest

{

public TestContext TestContext { get; set; }

[TestInitialize]

public void Setup()

{

// Код, который выполнится перед каждым тестом

}

[TestCleanup]

public void Cleanup()

{

TestContext.WriteLine("Тест завершен.");

}

[TestMethod]

public void TestAverageScore()

{

var student1 = new ExamResult("Student A", 85);

var student2 = new ExamResult("Student B", 95);

double average = (student1.Score + student2.Score) / 2;

Assert.AreEqual(90.0, average);

TestContext.WriteLine("TestAverageScore завершен успешно.");

}

[TestMethod]

public void TestScoreDifference()

{

var student1 = new ExamResult("Student A", 85);

var student2 = new ExamResult("Student B", 95);

int difference = student1.Score - student2.Score;

Assert.AreEqual(-10, difference);

TestContext.WriteLine("TestScoreDifference завершен успешно.");

}

[TestMethod]

public void TestSumScores()

{

var student1 = new ExamResult("Student A", 70);

var student2 = new ExamResult("Student B", 80);

int sum = student1.Score + student2.Score;

Assert.AreEqual(150, sum);

TestContext.WriteLine("TestSumScores завершен успешно.");

}

[TestMethod]

public void TestNegativeScoreDifference()

{

var student1 = new ExamResult("Student A", 60);

var student2 = new ExamResult("Student B", 75);

int difference = student1.Score - student2.Score;

Assert.AreEqual(-15, difference);

TestContext.WriteLine("TestNegativeScoreDifference завершен успешно.");

}

[TestMethod]

public void TestEqualScores()

{

var student1 = new ExamResult("Student A", 88);

var student2 = new ExamResult("Student B", 88);

Assert.AreEqual(student1.Score, student2.Score);

TestContext.WriteLine("TestEqualScores завершен успешно.");

}

[TestMethod]

public void TestHigherScore()

{

var student1 = new ExamResult("Student A", 92);

var student2 = new ExamResult("Student B", 85);

Assert.IsTrue(student1.Score > student2.Score);

TestContext.WriteLine("TestHigherScore завершен успешно.");

}

[TestMethod]

public void TestLowerScore()

{

var student1 = new ExamResult("Student A", 78);

var student2 = new ExamResult("Student B", 82);

Assert.IsTrue(student1.Score < student2.Score);

TestContext.WriteLine("TestLowerScore завершен успешно.");

}

[TestMethod]

public void TestNonZeroScore()

{

var student = new ExamResult("Student A", 67);

Assert.AreNotEqual(0, student.Score);

TestContext.WriteLine("TestNonZeroScore завершен успешно.");

}

[TestMethod]

public void TestMaxPossibleScore()

{

var student = new ExamResult("Student A", 100);

Assert.AreEqual(100, student.Score);

TestContext.WriteLine("TestMaxPossibleScore завершен успешно.");

}

[TestMethod]

public void TestMinPossibleScore()

{

var student = new ExamResult("Student A", 0);

Assert.AreEqual(0, student.Score);

TestContext.WriteLine("TestMinPossibleScore завершен успешно.");

}

[TestMethod]

public void TestScoreBoundaryAbove()

{

var student = new ExamResult("Student A", 101);

Assert.IsTrue(student.Score > 100, "Score should not be greater than 100");

TestContext.WriteLine("TestScoreBoundaryAbove завершен успешно.");

}

[TestMethod]

public void TestScoreBoundaryBelow()

{

var student = new ExamResult("Student A", -1);

Assert.IsTrue(student.Score < 0, "Score should not be less than 0");

TestContext.WriteLine("TestScoreBoundaryBelow завершен успешно.");

}

[TestMethod]

public void TestValidScoreRange()

{

var student = new ExamResult("Student A", 50);

Assert.IsTrue(student.Score >= 0 && student.Score <= 100);

TestContext.WriteLine("TestValidScoreRange завершен успешно.");

}

[TestMethod]

public void TestInvalidScore()

{

var student = new ExamResult("Student A", 110);

Assert.IsTrue(student.Score > 100, "Score is invalid");

TestContext.WriteLine("TestInvalidScore завершен успешно.");

}

[TestMethod]

public void TestZeroScore()

{

var student = new ExamResult("Student A", 0);

Assert.AreEqual(0, student.Score);

TestContext.WriteLine("TestZeroScore завершен успешно.");

}

}

}