Министерство образования Республики Беларусь

Оршанский колледж учреждения образования

«Витебский государственный университет имени П. М. Машерова»

Специальность 5-04-0612-02

«Разработка и сопровождение программного обеспечения информационных систем»

Группа 3ПОИС23

**УЧЕБНАЯ ПРАКТИКА**

**ПО ПРОГРАММИРОВАНИЮ**

с 29 сентября 2025 года по 25 октября 2025 года

**ОТЧЕТ**

Выполнил(а) Шевцова А.Д.

Руководитель практики Алейников М.А.

2025

**СОДЕРЖАНИЕ**

[Модуль 1. Основы C# 4](#_Toc210035478)

[Модуль 2. Объектно-ориентированное программирование (ООП) 4](#_Toc210035479)

[Модуль 3. Задания по делегатам 4](#_Toc210035480)

[Модуль 4. Задания по интерфейсам 4](#_Toc210035481)

[Модуль 5. Графический интерфейс 4](#_Toc210035482)

[Модуль 6. Базы данных 4](#_Toc210035483)

[Модуль 7. Проектная работа 4](#_Toc210035484)

[Модуль 8. Реализация проекта 4](#_Toc210035485)

[8.1. Постановка задачи 4](#_Toc210035486)

[8.1.1. Описание предметной области 4](#_Toc210035487)

[8.1.2. Характеристика решаемой задачи 4](#_Toc210035488)

[8.1.3. Разработка моделей решаемой задачи 4](#_Toc210035489)

[8.2. Проектирование программного модуля 4](#_Toc210035490)

[8.2.1. Сбор исходных материалов 4](#_Toc210035491)

[8.2.2. Проектирование информационной модели 4](#_Toc210035492)

[8.2.3. Описание входных и выходных данных 4](#_Toc210035493)

[8.3. Реализация программного модуля 4](#_Toc210035494)

[8.3.1. Описание диаграммы классов разрабатываемого проекта 4](#_Toc210035495)

[8.3.2 Описание структуры разрабатываемого проекта 4](#_Toc210035496)

[8.3.3. Проектирование и реализация интерфейса программы 4](#_Toc210035497)

[Модуль 9. Совершенствование навыков ООП 4](#_Toc210035498)

[ЗАКЛЮЧЕНИЕ 4](#_Toc210035499)

[Список использованных источников 4](#_Toc210035500)

# Модуль 1. Основы C#

**Цель:** разработать модуль 1.

**Модуль 1.1.**

**Задания. (Вариант 2.)**

1. Напишите программу, которая проверяет, является ли введенное число четным или нечетным, и выводит соответствующее сообщение.
2. Напишите программу, которая запрашивает у пользователя радиус круга и выводит его площадь.
3. Напишите программу, которая запрашивает у пользователя имя и фамилию, а затем выводит их в формате "Фамилия, Имя".
4. Напишите программу, которая создает массив из 20 случайных чисел и находит максимальное и минимальное значение в массиве.
5. Напишите программу, которая запрашивает у пользователя возраст и определяет, может ли он получить водительские права (старше 18 лет).

**Выполнение практических заданий.**

**Выполнение задания 1.**

Чтобы проверить число на четность, необходимо разделить число на 2 и проверить остаток. Если остаток равен 0, то число четное, иначе – нечетное. В программе это проверяется через оператор условия if и else.

**Листинг 1.1.1.** Код программы задания 1.

Console.Write("Введите число: ");

int a = Convert.ToInt32(Console.ReadLine());

if (a % 2 == 0)

{

Console.WriteLine($"{a} - четное");

}

else

{

Console.WriteLine($"{a} - нечетное");

}

Console.WriteLine("Нажмите любую клавишу для выхода...");

Console.ReadKey();

Результат выполнения программы представлен на рисунке 1.1.1. и рисунке 1.1.2

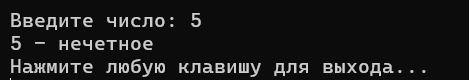
****

Рисунок 1.1.1 – Результат выполнения программы

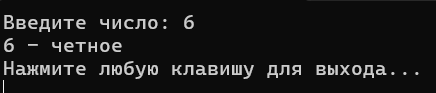


Рисунок 1.1.2 – Результат выполнения программы

**Задание 2.**

Площадь круга можно вычислить по формуле 1.1.1. Для вычислений в программе используется библиотека Math.

(1.1.1)

**Листинг 1.1.2.** Код программы задания 2.

Console.Write("Введите радиус круга: ");

int radius = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Площадь круга равна " + Math.PI \* Math.Pow(radius, 2));

Console.WriteLine("Нажмите любую клавишу для выхода...");

Console.ReadKey();

Результат выполнения программы представлен на рисунке 1.1.3, также на рисунке 1.1.4 показан рисунок результата вычисления сайта https://calc.by/math-calculators/area-circle.html для проверки корректности работы программы.

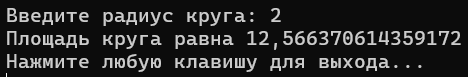


Рисунок 1.1.3 – Результат выполнения программы

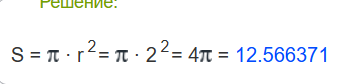


Рисунок 1.1.4 – Результат вычислений на сайте

**Задание 3.**

В программе сначала вводится имя и фамилия в строковый массив. Далее, чтобы отдельно получить имя и фамилию, массив разделяется на два массива по пробелу с помощью Split(). В конце программа выводит сначала вторую часть массива с фамилией, а после первую часть массива с именем.

**Листинг 1.1.3.** Код программы задания 3.

Console.Write("Введите имя и фамилию: ");

string[] str = Console.ReadLine()!.Split(' ');

Console.WriteLine($"Вывод в формате 'Фамилия, Имя': {str[1]}, {str[0]}");

Console.WriteLine("Нажмите любую клавишу для выхода...");

Console.ReadKey();

На рисунке 1.1.5 представлен результат выполнения программы листинга 1.1.3.

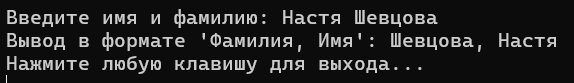


Рисунок 1.1.5 – Результат выполнения программы

**Задание 4.**

В программе сначала задается константа – размер массива. После создается объект целочисленного массива, и объект класса Random. В цикле for массив заполняется случайными значениями от 0 до 100. Массив с случайными значениями выводится. С помощью Array.Sort() массив сортируется. Так как первый элемент и последний элемент отсортированного массива имеет наименьшее и наибольшее значение соответственно, то они и выводятся как минимальный и максимальный элементы.

**Листинг 1.1.4.** Код программы задания 4.

const int size = 30;

int[] numbers = new int[size];

Random rand = new Random();

// Заполнение массива случайными значениями

for (int i = 0; i < size; i++)

numbers[i] = rand.Next(0, 100);

// Вывод массива

Console.WriteLine("Массив случайных чисел: " + string.Join(" ", numbers));

Array.Sort(numbers); // сортировка по возрастанию

int min = numbers[0]; // Первый элемент

int max = numbers[^1]; // Последний элемент

Console.WriteLine($"Минимум: {min}");

Console.WriteLine($"Максимум: {max}");

Console.WriteLine("Нажмите любую клавишу для выхода...");

Console.ReadKey();

Результат выполнения программы представлен на рисунке 1.1.6.

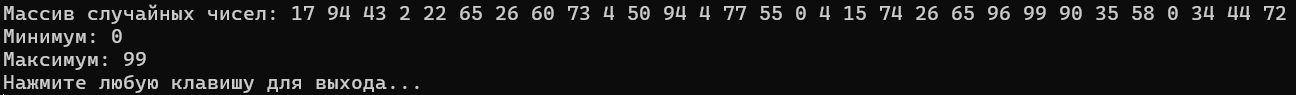


Рисунок 1.1.6 – Результат выполнения программы

**Задание 5.**

Проверка возраста выполняется с помощью тернарного оператора вида [первый операнд - условие] ? [второй операнд] : [третий операнд], где условие – это проверка целочисленной переменной age.

**Листинг 1.1.5.** Код программы задания 5.

Console.Write("Введите возраст: ");

int age = Convert.ToInt32(Console.ReadLine());

Console.WriteLine(age >= 18 ? "Можно получить водительские права" : "Нельзя получить водительские права");

Console.WriteLine("Нажмите любую клавишу для выхода...");

Console.ReadKey();

Результат выполнения программы показан на рисунке 1.1.7 (вводимое значение меньше 18) и рисунке 1.1.8 (вводимое значения больше 18).

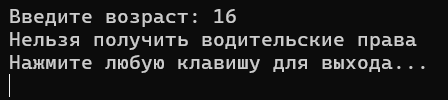


Рисунок 1.1.7 – Результат выполнения программы

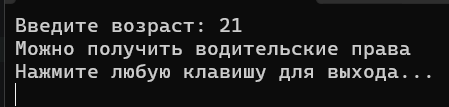


Рисунок 1.1.8 – Результат выполнения программы

**Задания. Модуль 1.2.**

Задача 1. Ввести размер массива N и значения его элементов. Нормировать элементы массива, разделив их на значение максимального по модулю элемента. Вывести значения элементов измененного массива.

Задача 2. Определить и инициализировать целочисленный массив из 10-ти элементов. Ввести целое число и заменить им значение максимального элемента в массиве.

Задача 3. Вычислить К простых чисел. Значение К ввести с клавиатуры. Вывести значения чисел, размещая их по 10 на строке

Задача 4. Определить целочисленный массив из К элементов. Присвоить элементам случайные значения из диапазона [А,В). Найти индексы минимального и максимального элементов массива. Вывести значения элементов, расположенных между найденными (включая найденные)

Задача 5. Определить символьный массив из К элементов. Присвоить элементам случайные значения букв русского алфавита. Создать новый массив, поместив в него только согласные буквы из первого массива. Значение К ввести с клавиатуры. Вывести элементы обоих массивов.

Задача 6. Определить вещественный массив из 10-ти элементов. Присвоить элементам случайные значения из диапазона [–10, 10). Сформировать массив индексов, которые нумеруют элементы первого массива в порядке возрастания их значений

**Выполнение практических заданий.**

**Выполнение задания 1.**

Чтобы нормировать элементы массивы, необходимо разделить каждый элемент на наибольший. Для этого сортируется копия введенного массива. Максимальное значение – последний элемент массива. В цикле происходит деление всех элементов на максимальный.

**Листинг 1.2.1.** Код программы задания 1.

Console.Write("Введите количество элементов массива: ");

int count = Convert.ToInt32(Console.ReadLine()); // Ввод количества элементов в массива

var array = new float[count];

for (int i = 0; i < count; i++)

{

Console.Write($"Введите {i} элемент: ");

array[i] = Convert.ToInt32(Console.ReadLine()); // Ввод значений массива

}

Console.Write("Массив: ");

Console.WriteLine(string.Join(" ", array)); // Вывод заполненного массива

var cpArr = new float[count];

Array.Copy(array, cpArr, count);

Array.Sort(cpArr); // Сортировка массива

var maxNum = cpArr[^1]; // Максимальный элемент

Console.WriteLine("Максимальный элемент: " + maxNum);

for (int i = 0; i < count; i++)

{

array[i] = array[i] / maxNum;

}

Console.Write("Нормированный массив: ");

Console.WriteLine(string.Join(" ", array));

Console.ReadKey();

Результат выполнения программы представлен на рисунке 1.2.1.

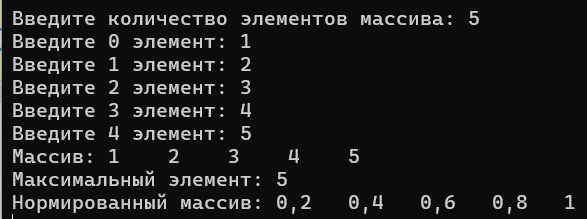


Рисунок 1.2.1 – Результат выполнения программы

**Задание 2.**

Вводится цикл с клавиатуры и число, используемое в замене. Индекс наибольшего элемента находится с помощью IndexOf(). После элемент с найденным индексом заменяется на введенное число.

**Листинг 1.2.2.** Код программы задания 2.

const int count = 10;

var array = new int[count];

// Цикл для ввода значений элементов

for (int i = 0; i < count; i++)

{

Console.Write($"Введите {i} элемент: ");

array[i] = Convert.ToInt32(Console.ReadLine()); // Ввод значения элемента массива

}

Console.Write("Введите число для замены: ");

var number = Convert.ToInt32(Console.ReadLine()); // Ввод числа для замены

var maxIndex = Array.IndexOf(array, array.Max()); // Поиск индекса максимального элемента

array[maxIndex] = number; // Замена числа

Console.WriteLine("Массив с замененным числом: " + string.Join(", ", array)); // Вывод массива

Console.ReadKey();

Результат работы программы показан на рисунке 1.2.2.

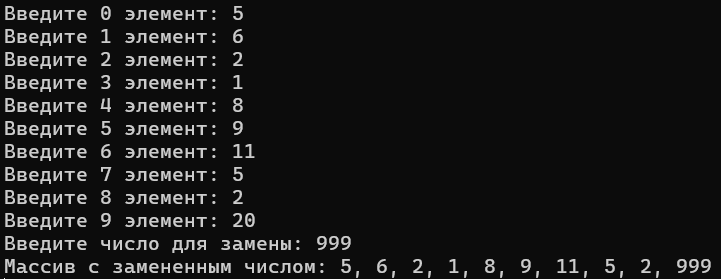


Рисунок 1.2.2. – Результат выполнения программы

**Задание 3.**

Поиск простых чисел происходит в цикле. После нахождения выводятся найденные числа.

**Листинг 1.2.3.** Код программы задания 3.

Console.Write("Введите К: ");

int K = Convert.ToInt32(Console.ReadLine());

int count = 0, num = 2;

// Цикл для поиска простых чисел

while (count < K)

{

bool prime = true;

// i \* i потому что проверяем только до корня от num

for (int i = 2; i \* i <= num; i++)

if (num % i == 0)

{

prime = false; // Делится на что-то кроме 1 и себя - не простое

}

if (prime)

{

Console.Write($"{num, 5}"); // Вывод с шириной 5

count++;

if (count % 10 == 0) Console.WriteLine();

}

num++;

}

Console.ReadKey();

На рисунке 1.2.3 показан результат работы программы листинга 1.2.3.З

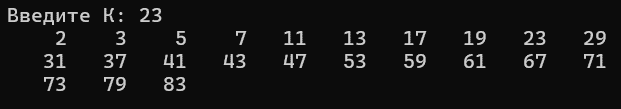


Рисунок 1.2.3 – Результат выполнения программы

**Задание 4.**

Задается массив с помощью случайных значений. Индексы элементов с максимальным и минимальным значением находится используя IndexOf().

**Листинг 1.2.4.** Код программы задания 4.

Console.Write("Введите количество элементов: ");

int K = Convert.ToInt32(Console.ReadLine());

int[] array = new int[K];

var rand = new Random(); // Создание нового объекта класса Random

Console.Write("Введите значение A: ");

int A = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите значение B: ");

int B = Convert.ToInt32(Console.ReadLine());

// Цикл для заполнения массива

for (int i = 0; i < K; i++)

{

array[i] = rand.Next(A, B);

}

Console.WriteLine(string.Join(", ", array));

var minIndex = Array.IndexOf(array, array.Min()); // Индекс минимального элемента

var maxIndex = Array.IndexOf(array, array.Max()); // Индекс максимального элемента

Console.WriteLine($"Минимальное значение: {array.Min()}, его индекс: {minIndex}\n" +

$"Максимальное значение: {array.Max()}, его индекс: {maxIndex}");

var firstIndex = Math.Min(minIndex, maxIndex); // Первый индекс по счету

var secondIndex = Math.Max(minIndex, maxIndex); // второй индекс по счету

Console.Write("Значения в диапозоне: ");

for (int i = firstIndex; i <= secondIndex; i++)

{

Console.Write(array[i] + " ");

}

Console.ReadKey();

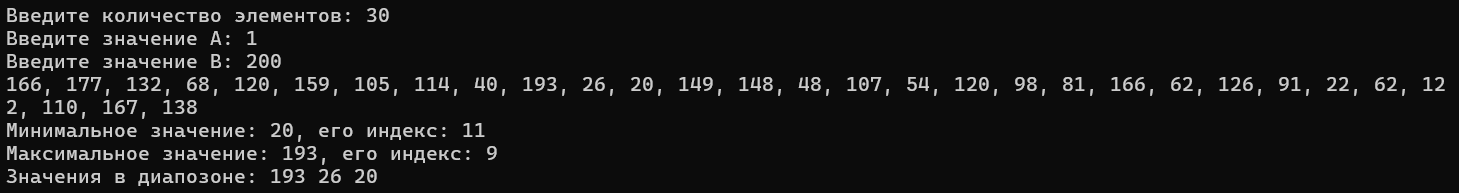


Рисунок 1.2.4 – Результат выполнения программы

**Задание 5.**

Задаются символьные массивы: массив всего русского алфавит и гласных букв. Создается новый символьный массив, который заполняется случайными русскими буквами из соответствующего символьного массива. Далее используя List<> просиходит поиск согласных.

**Листинг 1.2.5.** Код программы задания 5.

Console.Write("Введите количество элементов массива: ");

var K = Convert.ToInt32(Console.ReadLine());

char[] alphabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ".ToCharArray(); // Массив всех букв

char[] vowels = "АЕЁИОУЫЭЮЯ".ToCharArray(); // Массив гласных

Random rand = new Random(); // Создание объекта класса Random

var array = new char[K]; // Массив для случайных букв

// Цикл для заполнения массива случайными буквами

for (int i = 0; i < K; i++)

array[i] = alphabet[rand.Next(0, alphabet.Length)]; // Добавление случайной буквы

List<char> constants = new List<char>(K); // Лист для согласных (чтобы меньше кода было)

// Поиск согласных

foreach (var letter in array)

{

if(!vowels.Contains(letter))

constants.Add(letter);

}

Console.WriteLine("Массив случайных букв: " + string.Join(' ', array)); // Вывод массива со случайными буквами

Console.WriteLine("Лист согласных из массива случайных букв" + string.Join(' ', constants)); // Вывод массива с согласными

Console.ReadKey();

Результат выполнения программы листинга 1.2.5 представлен на рисунке 1.2.5.

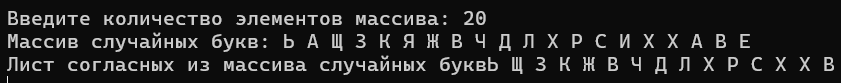


Рисунок 1.2.5 – Результат выполнения программы

**Задание 6.**

Создается массив и заполняется случайными вещественными числами. Далее происходит сортировка индексов по возрастанию значений.

**Листинг 1.2.6.** Код программы задания 6.

const int size = 10;

var array = new float[size];

var indices = new int[size];

var rand = new Random();

// Цикл для заполнения случацными вещественными значениями

for (int i = 0; i < size; i++)

{

array[i] = (float)rand.NextDouble() \* 20 - 10; // [0,1) > [0,20) > [-10,10)

indices[i] = i; // Начальная нумераия

}

// Цикл сортировки индексов по значениям массива

for (int i = 0; i < size - 1; i++)

{

for (int j = i + 1; j < size; j++)

{

if (array[indices[i]] > array[indices[j]])

{

(indices[i], indices[j]) = (indices[j], indices[i]);

}

}

}

Console.WriteLine("Исходный массив: " + string.Join(" ", array.Select(n => n.ToString("F2"))));

Console.WriteLine("Массив индексов по возрастанию значений: " + string.Join(' ', indices));

Console.ReadKey();

На рисунке 1.2.6 показан результат выполнения программы листинга 1.2.6.



Рисунок 1.2.6 – Результат выполнения программы

**Задания. Модуль 1.3.**

Задача 1. Определить функцию (статический метод) для вычисления наибольшего общего делителя двух целых натуральных чисел (Greatest Common Measure). В основной программе, используя функцию, сократить неотрицательную обыкновенную дробь. Дробь вводится с клавиатуры в виде неотрицательного числителя и положительного знаменателя

Задача 2. Присваивая последовательным элементам массива случайные значения от 1 до 9, создать массив с минимальным количеством элементов, сумма которых не превышает заданного пользователем числа.

Задача 3. Сформировав квадратную целочисленную матрицу со случайными значениями элементов, упорядочить ее строки по возрастанию сумм их элементов. Пусть элементы принимают значения в диапазоне от –50 до +50.

**Задание 1.**

Наибольший общий делитель находится с помощью алгоритма Евклида – большее число делится на меньшее, если делится без остатка, то меньшее число и есть НОД. Если остаток есть, то большее число заменяется на остаток от деления.

**Листинг 1.3.1.** Код программы задания 1.

using System;

class Program

{

static int NOD(int a, int b)

{

// Алгоритм Евклида

while (a != 0 && b != 0)

{

if (a > b)

a %= b;

else b %= a;

}

return a + b;

}

static void Main(string[] args)

{

Console.Write("Введите неотрицательный числитель: ");

int a = int.Parse(Console.ReadLine()!);

Console.Write("Введите положительный знаменатель: ");

int b = int.Parse(Console.ReadLine()!);

int nod = NOD(a, b);

Console.WriteLine("Наибольший общий делитель: " + nod);

Console.WriteLine($"Сокращенная дробь: {a/nod} / {b/nod}");

Console.ReadKey();

}

}

На рисунке 1.3.1. представлен результат выполнения программы листинга 1.3.1.

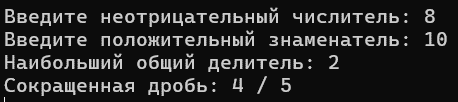


Рисунок 1.3.1 – Результат выполнения программы

**Задание 2.**

Значение максимальной сумме вводится с клавиатуры. Далее пока сумма значений элементов массива, которые генерируются случайно в диапазоне, не привышает введенную сумму – элементы продолжают генерироваться.

**Листинг 1.3.2.** Код программы задания 2.

using System;

class Program

{

static void Main()

{

Console.Write("Введите значение максимальной суммы: ");

int maxSum = Convert.ToInt32(Console.ReadLine());

int sum = 0;

var array = new int[1];

var temp = new int[maxSum];

var count = 0;

var rand = new Random();

while (sum < maxSum)

{

int value = rand.Next(1, 10);

if (sum + value > maxSum) break;

temp[count++] = value;

sum += value;

}

// Создание итогового массива нужной длины

var result = new int[count];

Array.Copy(temp, result, count);

Console.WriteLine("Сформированный массив: " + string.Join(" ", result) + "\nСумма элементов: " + sum);

Console.ReadKey();

}

}

Результат выполнения программы листинга 1.3.2 показан на рисунке 1.3.2.

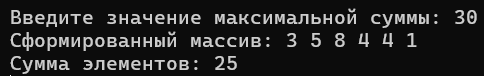


Рисунок 1.3.2 – Результат выполнения программы

**Задание 3.**

Вводится размер квадратной матрицы. Значения двумерного массива случайные в заданном диапозоне. Исходная матрица выводится с помощью функции PrintMatrix(). Далее проихсходит сортировка матрицы по сумме значений в строке. В конце выводится отсортированная по строкам матрица.

**Листинг 1.3.3.** Код программы задания 3.

using System;

class Program

{

static void Main(string[] args)

{

Console.Write("Введите размер квадратной матрицы: ");

var size = Convert.ToInt32(Console.ReadLine());

int[,] matrix = new int[size, size];

var rand = new Random();

for (int i = 0; i < size; i++)

for (int j = 0; j < size; j++)

matrix[i, j] = rand.Next(-50, 51);

Console.WriteLine("Исходная матрица: ");

PrintMatrix(matrix);

//Сортировка по сумме

// Перебирает строки матрицы

for (int i = 0; i < size - 1; i++)

{

// Сравнивает текущую строку i с каждой последующей строкой k

for (int k = i + 1; k < size; k++)

{

// Если сумма одной строки больше суммы другой

if (RowSum(matrix, i) > RowSum(matrix, k))

{

// Поменять местами

for (int j = 0; j < size; j++)

{

(matrix[i, j], matrix[k, j]) = (matrix[k, j], matrix[i, j]);

}

}

}

}

Console.WriteLine("Отсортированная матрица:");

PrintMatrix(matrix);

Console.ReadKey();

}

// Метод для поиска суммы ряда

static int RowSum(int[,] matrix, int row)

{

int sum = 0;

for (int i = 0; i < matrix.GetLength(1); i++)

sum += matrix[row, i];

return sum;

}

// Метод для вывода матрицы

static void PrintMatrix(int[,] matrix)

{

int rows = matrix.GetLength(0);

int cols = matrix.GetLength(1);

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

Console.Write($"{matrix[i, j], 5}");

Console.WriteLine();

}

}

}

Результат выполнения программы листинга 1.3.3 представлен на рисунке 1.3.3.



Рисунок 1.3.3. – Результат выполнения программы

# Модуль 2. Объектно-ориентирование программирование (ООП).

**Задание 1. Создание классов.**

Создайте класс Person, представляющий человека. У этого класса должны быть поля для хранения имени, возраста и адреса. Добавьте методы для установки и получения значений этих полей. Затем создайте объекты этого класса и выведите информацию о них.

**Задание 2. Наследование и полиморфизм.**

Создайте базовый класс Shape, представляющий геометрическую фигуру, и производные классы Circle и Rectangle. В базовом классе определите метод Area(), который возвращает площадь фигуры, и метод Perimeter(), который возвращает периметр фигуры. В производных классах переопределите эти методы для соответствующих фигур (круг и прямоугольник). Создайте объекты всех классов и выведите их площади и периметры.

**Задание 3. Композиция.**

Создайте классы Author и Book. Класс Author должен содержать информацию об авторе (имя и год рождения). Класс Book должен содержать информацию о книге (название, год выпуска и автора). Используйте композицию, чтобы связать объекты Author и Book. Создайте несколько объектов Author и Book и выведите информацию о них.

**Задание 4. Интерфейсы и абстрактные классы.**

Создайте интерфейс IDrawable с методом Draw(), который выводит информацию о рисуемом объекте. Создайте классы Circle, Rectangle и Triangle, реализующие этот интерфейс. Создайте массив объектов, реализующих интерфейс IDrawable, и вызовите метод Draw() для каждого из них.

**Задание 5. События.**

Создайте класс TemperatureSensor, который генерирует событие TemperatureChanged, когда измеренная температура меняется. Создайте класс Thermostat, который подписывается на событие TemperatureChanged и реагирует на изменение температуры, включая или выключая отопление.

**Задания по вариантам. (Вариант 2)**

**Задание 1.**

Создайте класс "Автомобиль" с полями: марка, модель, год выпуска и цена. Реализуйте методы для расчета стоимости автомобиля с учетом скидки и налога на добавленную стоимость.

**Задание 2.**

Создайте структуру с именем student, содержащую поля: фамилия и инициалы, номер группы, успеваемость (массив из пяти элементов). Создать массив из десяти элементов такого типа, упорядочить записи по возрастанию среднего балла. Добавить возможность вывода фамилий и номеров групп студентов, имеющих оценки, равные только 4 или 5.

**Задание 3.**

Создание класса "Геометрическая фигура" (Shape) и его производных: Создайте абстрактный класс "Геометрическая фигура" с методом для вычисления площади. Создайте производные классы, например, "Круг" (Circle), "Прямоугольник" (Rectangle), "Треугольник" (Triangle) и т. д. Реализуйте методы для вычисления площади у каждого класса. Создайте объекты различных классов и протестируйте их функциональность.

**Выполнение практических заданий.**

**Задание 1.**

Создается класс Person с публичными полями Name, Age, Address. Данные вводятся с помощью конструктара класса, выводятся с помощью метода Print().

**Листинг 2.1.** Код программы задания 1.

using System;

class Person

{

public string Name { get; set; }

public int Age { get; set; }

public string Address { get; set; }

// Конструктор класса Person

public Person(string name = "Не введено!", int age = 0, string address = "Не введено!")

{

Name = name;

Age = age;

Address = address;

}

public void Print() => Console.WriteLine($"Имя: {Name}, Возраст: {Age}, Адрес: {Address}");

}

class Program

{

static void Main(string[] args)

{

Console.Write("Введите имя: ");

var name = Console.ReadLine()!;

Console.Write("Введите возраст: ");

var age = int.Parse(Console.ReadLine()!);

Console.Write("Введите адрес: ");

var address = Console.ReadLine()!;

var person = new Person(name, age, address);

person.Print();

Console.ReadKey();

}

}

На рисунке 2.1 показан результат работы программы листинга 2.1.

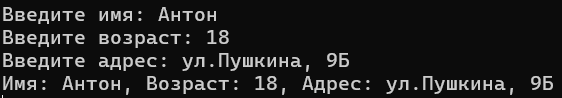


Рисунок 2.1 – Результат выполнения программы

**Задание 2.**

Реализуется базовый класс Shape с виртуальными методами для вычисления площади и периметра. Производные классы Circle и Rectangle переопределяют эти методы. Пользователь вводит параметры фигур, и программа выводит результаты расчётов.

**Листинг 2.2.** Код программы задания 2.

using System;

class Shape

{

public virtual double Area() => 0;

public virtual double Perimeter() => 0;

public void Print() => Console.WriteLine($"Площадь: {Area()}, Периметр: {Perimeter()}");

}

class Circle(double radius): Shape

{

public double Radius { get; set; } = radius;

public override double Area() => Math.PI \* Math.Pow(Radius, 2);

public override double Perimeter() => Math.PI \* Radius \* 2;

}

class Rectangle(double width, double height) : Shape

{

public double Width { get; set; } = width;

public double Height { get; set; } = height;

public override double Area() => Width \* Height;

public override double Perimeter() => (2 \* Width) + (2 \* Height);

}

class Program

{

static void Main(string[] args)

{

Console.Write("Введите радиус для круга: ");

double radius = double.Parse(Console.ReadLine()!);

var circle = new Circle(radius);

circle.Print();

Console.Write("Введите ширину прямоугольника: ");

var width = double.Parse(Console.ReadLine()!);

Console.Write("Введите высоту прямоугольника: ");

var height = double.Parse(Console.ReadLine()!);

var rectangle = new Rectangle(width, height);

rectangle.Print();

Console.ReadKey();

}

}

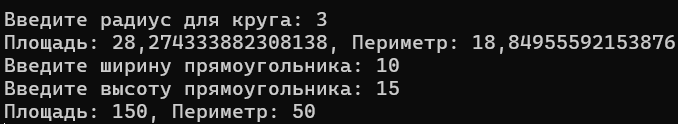


Рисунок 2.2 – Результат выполнения программы

**Задание 3.**

Создаются классы Author Book, связанные через композицию. Книга содержит объект автора. Программа создаёт несколько авторов и книг, затем выводит информацию о каждой книге и её авторе.

**Листинг 2.3.** Код программы задания 3.

class Author

{

public string AuthorName { get; set; }

public double AuthorAge { get; set; }

public Author(string name, int age)

{

AuthorName = name;

AuthorAge = age;

}

public void Print() => Console.WriteLine($"Автор: {AuthorName}, {AuthorAge}");

}

class Book

{

public string Title { get; set; }

public int Year { get; set; }

Author Author { get; set; }

public Book(string title, int year, Author author)

{

Title = title;

Year = year;

Author = author;

}

public void Print()

{

Console.Write($"Название: {Title}, {Year}. ");

Author.Print();

}

}

class Program

{

static void Main(string[] args)

{

Author author1 = new Author("Александр Дюма", 1802);

Author author2 = new Author("Эмили Бронте", 1848);

Book book1 = new Book("Граф Монте-Кристо", 2024, author1);

Book book2 = new Book("Грозовой перевал", 2004, author2);

book1.Print();

book2.Print();

Console.ReadKey();

}

}

Результат выполнения программы листинга 2.3 показан на рисунке 2.3.



Рисунок 2.3 – Результат выполнения программы

**Задание 4.**

Создаётся интерфейс IDrawable с методо Draw() Класс Circle, Rectengle и Triangle реализуют этот интерфейс. В Main создаётся массив фигур, и для каждой вызывается метод Draw().

**Листинг 2.4.** Код программы задания 4.

// Интерфейс

interface IDrawable

{

// Метод по умолчанию

void Draw() => Console.WriteLine("Это фигура");

}

class Circle(float radius): IDrawable

{

public void Draw() => Console.WriteLine($"Это круг с радиусом {radius}");

}

class Rectangle(float width, float height) : IDrawable

{

public void Draw() => Console.WriteLine($"Это квадрат с высотой {height} и шириной {width}");

}

class Triangle(float sideA, float sideB, float sideC) : IDrawable

{

public void Draw() => Console.WriteLine($"Это треугольник со сторонами {sideA}, {sideB}, {sideC}");

}

class Program

{

static void Main(string[] args)

{

// Массив фигур

var shapes = new IDrawable[]

{

new Circle(10),

new Rectangle(10, 20),

new Triangle(15, 20, 25)

};

foreach (var shape in shapes)

{

shape.Draw(); // Вызов метода для предоставления информации о фигуре

}

Console.ReadKey(); // Чтобы консоль не закрылась

}

}

Результат выполнения программы листина 2.4 показан на рисунке 2.4.

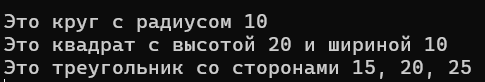


Рисунок 2.4 – Результат выполнения программы

**Задание 5.**

Реализуется класс TemperatureSensor, который генерирует событие при изменении температуры. Класс Thermostat подписывается на это событие и реагирует, включая или выключая отопление в зависимости от температуры.

**Листинг 2.5.** Код программы задания 5.

using System;

class TemperatureSensor

{

// Событие, которое будет вызываться при изменении температуры

public event Action<double>? TemperatureChanged;

// Позволяет задать новую температуру

public void SetTemperature(double temp)

{

// Если есть подписчик, событие вызывается с переданной температурой

TemperatureChanged?.Invoke(temp);

}

}

class Thermostat

{

// Вызывается при смене температуры

public void React(double temp)

{

Console.WriteLine(temp < 10 ? $"Включаем отопление ({temp} градусов)" :$"Выключаем отопление ({temp} градусов)");

}

}

class Program

{

static void Main(string[] args)

{

var sensor = new TemperatureSensor();

var thermostat = new Thermostat();

// thermostat.React подписсывается на событие

sensor.TemperatureChanged += thermostat.React;

sensor.SetTemperature(18);

sensor.SetTemperature(5);

Console.ReadKey();

}

}

Результат выполнения программы показан на рисунке 2.5.



Рисунок 2.5 – Результат выполнения программы

**Выполнение заданий по вариантам.**

**Задание 1.**

Программа создаёт класс автомобиля с полями: марка, модель, год выпуска и цена. Реализованы методы для расчёта стоимости с учётом скидки, налога и их комбинации. В Main() создаётся объект Car, выводится информация и рассчитываются цены по заданным параметрам.

**Листинг 2.6.** Код программы задания 1.

class Car

{

public string Brand {get; set;}

public string Model { get; set; }

public int Year { get; set; }

public double Price { get; set; }

public Car(string brand, string model, int year, double price)

{

Brand = brand;

Model = model;

Year = year;

Price = price;

}

public double GetPriceWithDiscount(double discount)

{

return Price \* (1 - discount / 100);

}

public double GetPriceWithTax(double tax)

{

return Price \* (1 + tax / 100);

}

public double GetFinalPrice(double discount, double tax)

{

double discounted = GetPriceWithDiscount(discount);

return discounted \* (1 + tax / 100);

}

public void PrintInfo()

{

Console.WriteLine($"Автомобиль: {Brand} {Model}, {Year} г., Цена: {Price} руб.");

}

}

class Program

{

static void Main(string[] args)

{

Car car = new Car("Toyota", "Camry", 2020, 30000);

car.PrintInfo();

double discount = 10; // 10%

double tax = 20; // 20%

Console.WriteLine($"Цена со скидкой: {car.GetPriceWithDiscount(discount)} руб.");

Console.WriteLine($"Цена с НДС: {car.GetPriceWithTax(tax)} руб.");

Console.WriteLine($"Итоговая цена (скидка + НДС): {car.GetFinalPrice(discount, tax)} руб.");

Console.ReadKey();

}

}

Результат выполнения программы представлен на рисунке 2.6.

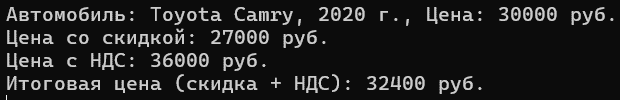


Рисунок 2.6 – Результат выполнения программы

**Задание 2.**

Создаётся структура студента с фамилией, инициалами, номером группы и массивом оценок. Программа выводит всех студентов, находит тех, у кого только оценки 4 и 5, затем сортирует студентов по среднему баллу и выводит отсортированный список.

**Листинг 2.7.** Код программы задания 2.

struct Student

{

public string LastName;

public string Initials;

public string GroupNumber;

public int[] Grades;

// Конструктор

public Student(string lastName, string initials, string groupNumber, int[] grades)

{

LastName = lastName;

Initials = initials;

GroupNumber = groupNumber;

Grades = grades;

}

// Метод для нахождения среднего балла

public int AvgGrade()

{

double sum = 0;

foreach (var grade in Grades)

sum += grade;

return (int)Math.Round(sum / Grades.Length);

}

// Методод для проверки оценок

public bool HasOnlyFoursAndFives()

{

for (int i = 0; i < Grades.Length; i++)

{

if (Grades[i] != 4 && Grades[i] != 5)

return false;

}

return true;

}

// Метод для вывода информации о студенте

public void Print()

{

Console.WriteLine($"{LastName} {Initials}, группа: {GroupNumber}, средний балл: {AvgGrade():F2}");

}

}

class Program

{

static void Main(string[] args)

{

Student[] students =

[

new Student { LastName = "Иванов", Initials = "И.И.", GroupNumber = "101", Grades = new[] {5, 4, 4, 4, 5} },

new Student { LastName = "Петров", Initials = "П.П.", GroupNumber = "102", Grades = new[] {3, 3, 4, 3, 4} },

new Student { LastName = "Сидоров", Initials = "С.С.", GroupNumber = "103", Grades = new[] {5, 5, 5, 5, 5} }

];

// Вывод всей информации о студентах

foreach (var student in students)

student.Print();

// Вывод студентов с оценками 4 и 5

Console.WriteLine("Студенты с оценками только 4 и 5:");

for (int i = 0; i < students.Length; i++)

{

if (students[i].HasOnlyFoursAndFives())

Console.WriteLine($"{students[i].LastName}, группа: {students[i].GroupNumber}");

}

// Вычисляет средний балл всех студентов

for (int i = 0; i < students.Length; i++)

{

for (int j = 0; j < students[i].Grades.Length; j++)

{

students[i].Grades[j] = students[i].AvgGrade();

}

}

// Сортирует студентов по среднему баллу

for (int i = 0; i < students.Length - 1; i++)

{

for (int j = i + 1; j < students.Length; j++)

{

if (students[i].AvgGrade() > students[j].AvgGrade())

(students[i], students[j]) = (students[j], students[i]);

}

}

// Вывод отсортированный массив структуры

Console.WriteLine("Студенты по возрастанию среднего балла:");

for (int i = 0; i < students.Length; i++)

students[i].Print();

Console.ReadKey();

}

}

Результат выполнения программы представлен на рисунке 2.7.

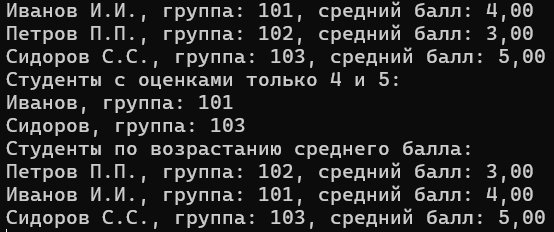


Рисунок 2.7 – Результат выполнения программы

**Задание 3.**

Реализован абстрактный класс геометрической фигуры с методом для вычисления площади. Производные классы Circle, Rectangle и Triangle переопределяют этот метод. В Main() создаются объекты фигур и выводится их описание с рассчитанной площадью.

**Листинг 2.8.** Код программы задания 3.

// Абстрактный класс фигур

abstract class Shape

{

public abstract double Area(); // Метод для нахождения площади

public abstract void Print(); // Метод для вывода информации о фигуре

}

// Класс Круг

class Circle : Shape

{

public double Radius { get; set; } // Радиус

public override double Area() => Math.PI \* 2 \* Radius;

public override void Print() => Console.WriteLine($"Круг с радиусом {Radius} и площадью {Area():F2}");

}

// Класс Прямоугольник

class Rectangle : Shape

{

public double Width { get; set; } // Ширина

public double Height { get; set; } // Высота

public override double Area() => Width \* Height;

public override void Print() => Console.WriteLine($"Прямоугольник с шириной {Width}, и высотой {Height}, и площадью {Area()}");

}

class Triangle : Shape

{

public double Height { get; set; } // Высота

public double Base { get; set; } // Основание

public override double Area() => (Height \* Base) / 2;

public override void Print() => Console.WriteLine($"Треугольник с высотой {Height}, и основанием {Base}, площадью {Area()}");

}

class Program

{

static void Main()

{

var circle = new Circle{Radius = 2};

var rectangle = new Rectangle{Height = 2, Width = 3};

var triangle = new Triangle{Base = 4, Height = 6};

circle.Print();

rectangle.Print();

triangle.Print();

Console.ReadKey();

}

}

Результат выполнения программы представлен на рисунке 2.8.

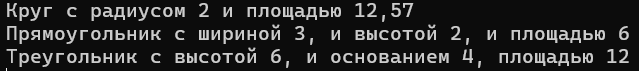


Рисунок 2.8 – Результат выполнения программы

# Модуль 3. Задания по делегатам

**Задания.**

1. Создайте базовый класс "Фигура" с методом для вычисления площади. Затем создайте производные классы для разных геометрических фигур (круг, прямоугольник, треугольник) и используйте делегат для динамического вызова метода вычисления площади.

2. Реализуйте систему событий для мобильного приложения. Создайте класс "Уведомление" с событиями для отправки уведомлений (сообщения, звонки, электронные письма). Зарегистрируйте обработчики событий для разных типов уведомлений.

3. Создайте приложение для управления задачами с использованием делегатов. Пользователь должен иметь возможность добавлять задачи и выбирать делегата для выполнения каждой задачи (например, отправка уведомления или запись в журнал).

4. Разработайте систему фильтрации данных с использованием делегатов. Пользователь должен иметь возможность выбрать фильтр для списка данных (например, фильтр по дате или по ключевым словам).

5. Создайте приложение для сортировки числовых данных. Пользователь должен иметь возможность выбрать метод сортировки (например, сортировка пузырьком или быстрая сортировка) с помощью делегатов.

**Задание 1.**

**Листинг 3.1.** Код программы задания 1.

public delegate double AreaDelegate();

abstract class Shape

{

public abstract double Area();

}

class Circle(double radius) : Shape

{

public double Radius { get; set; } = radius;

public override double Area() => Math.PI \* 2 \* Radius;

}

class Rectangle(double width, double height) : Shape

{

public double Width { get; set; } = width;

public double Height { get; set; } = height;

public override double Area() => Width \* Height;

}

class Triangle(double height, double @base) : Shape

{

public double Height { get; set; } = height;

public double Base { get; set; } = @base;

public override double Area() => (Height \* Base) / 2;

}

class Program

{

static void Main(string[] args)

{

var circle = new Circle(5);

var rectangle = new Rectangle(10, 20);

var triangle = new Triangle(10, 20);

Shape area;

AreaDelegate areaCircle = circle.Area;

AreaDelegate areaRectangle = rectangle.Area;

AreaDelegate areaTriangle = triangle.Area;

Console.WriteLine($"Площадь круга с радиусом {circle.Radius}: {areaCircle():F2}");

Console.WriteLine($"Площадь прямоугольника с шириной {rectangle.Width} и высотой {rectangle.Height}: {areaRectangle():F2}");

Console.WriteLine($"Площадь треугольника с высотой {triangle.Height} и основанием {triangle.Base}: {areaTriangle():F2}");

Console.ReadKey();

}

}

Результат выполнения программы листинга 3.1 показан на рисунке 3.1.

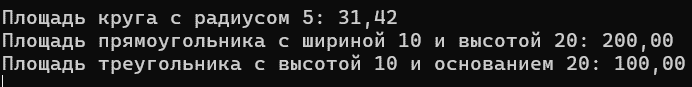


Рисунок 3.1 – Результат выполнения программы

**Задание 2.**

**Листинг 3.2.** Код программы задания 2.

class Notification

{

// События для разных типов уведомлений

public event Action<string>? OnMessageSent;

public event Action<string>? OnCallMade;

public event Action<string>? OnEmailSent;

// Методы для генерации событий

public void SendMessage(string message)

{

Console.WriteLine("Отправка сообщения...");

OnMessageSent?.Invoke(message);

}

public void MakeCall(string number)

{

Console.WriteLine("Набор номера...");

OnCallMade?.Invoke(number);

}

public void SendEmail(string email)

{

Console.WriteLine("Отправка письма...");

OnEmailSent?.Invoke(email);

}

}

class NotificationHandlers

{

public void HandleMessage(string message)

{

Console.WriteLine($"[Сообщение:] {message}");

}

public void HandleCallMade(string number)

{

Console.WriteLine($"Звонок совершен на номер: {number}");

}

public void HandleEmail(string email)

{

Console.WriteLine($"Письмо отправлено на адрес: {email}");

}

}

class Program

{

static void Main(string[] args)

{

var notifier = new Notification();

var handlers = new NotificationHandlers();

notifier.OnMessageSent += handlers.HandleMessage;

notifier.OnCallMade += handlers.HandleCallMade;

notifier.OnEmailSent += handlers.HandleEmail;

handlers.HandleMessage("Текст сообщения");

handlers.HandleCallMade("+375336593341");

handlers.HandleEmail("shevanx21@gmail.com");

Console.ReadKey();

}

}

На рисунке 3.2 показан результат выполнения программы листинга 3.2.

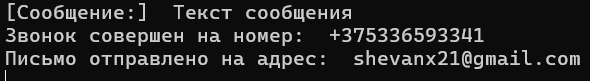


Рисунок 3.2 – Результат выполнения программы

**Задание 3.**

**Листинг 3.3.** Код программы задания 3.

using System.Collections.Generic;

// Делагат для обработки задачи

public delegate void TaskHandler(string taskDesc);

//Класс задачи

class Task(string desc, TaskHandler handler)

{

public string Description { get; set; } = desc;

public TaskHandler Handler { get; set; } = handler;

// Вызывает делегат и передает строку с описанием

public void Execute()

{

Handler(Description);

}

}

class TaskActions

{

public void SendNotification(string taskDesc)

{

Console.WriteLine($"Уведомление: {taskDesc}");

}

public void WriteLog(string taskDesc)

{

Console.WriteLine($"Запись в журнал: {taskDesc}");

}

}

class Program

{

static void Main(string[] args)

{

var actions = new TaskActions();

var tasks = new List<Task>();

while (true)

{

Console.Write("Введите описание задачи или 'выход': ");

string input = Console.ReadLine()!;

if (input.ToLower() == "выход") break;

Console.WriteLine("Выберите обработчик задачи:");

Console.WriteLine("1 — Отправить уведомление");

Console.WriteLine("2 — Записать в журнал");

string choice = Console.ReadLine()!;

TaskHandler handler;

switch (choice)

{

case "1":

handler = actions.SendNotification;

break;

case "2":

handler = actions.WriteLog;

break;

default:

handler = actions.WriteLog; // обработчик по умолчанию

break;

}

tasks.Add(new Task(input, handler));

}

Console.WriteLine("Выполнение задач:");

foreach (var task in tasks)

task.Execute();

Console.ReadKey();

}

}

Результат выполнения программы листинга 3.3 показан на рисунке 3.3.

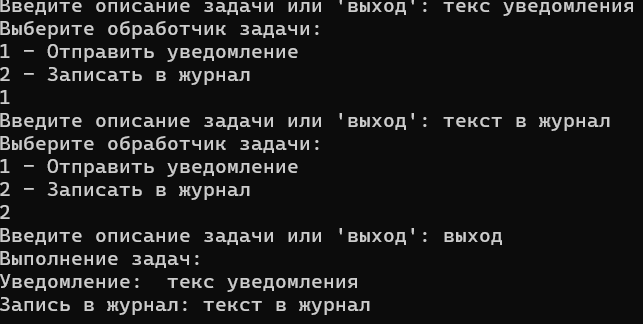


Рисунок 3.3 – Результат выполнения программы

**Задание 4.**

**Листинг 3.4.** Код программы задание 4.

using System.Collections.Generic;

// Делегат для фильтрации с параметром типа TaskItem

public delegate bool DataFilter(TaskItem item);

// Класс задачи

public class TaskItem(string title, DateTime dueDate)

{

public string Title { get; set; } = title;

public DateTime DueDate { get; set; } = dueDate;

public override string ToString()

{

return $"{Title} (до {DueDate:dd.MM.yyyy})";

}

}

class Program

{

private static void Print(List<TaskItem> tasks, DataFilter filter)

{

Console.WriteLine("Отфильтрованные задачи:");

foreach (var task in tasks)

{

if (filter(task))

Console.WriteLine(task);

}

}

static void Main()

{

// Исходные данные

var tasks = new List<TaskItem>

{

new TaskItem("Сдать отчёт", new DateTime(2025, 10, 5)),

new TaskItem("Позвонить клиенту", new DateTime(2025, 10, 3)),

new TaskItem("Обновить сайт", new DateTime(2025, 10, 10)),

new TaskItem("Отправить письмо", new DateTime(2025, 10, 3))

};

foreach (var task in tasks)

{

Console.WriteLine(task);

}

DataFilter? filter = null;

var exit = false;

while (!exit)

{

Console.WriteLine("Выберите фильтр:");

Console.WriteLine("1 — По дате (только задачи на сегодня)");

Console.WriteLine("2 — По ключевому слову (например, 'письмо')");

Console.WriteLine("3 - Выход");

string choice = Console.ReadLine()!;

switch (choice)

{

case "1":

DateTime today = DateTime.Today;

// Присваивает делегату ИСТИНА, если дата задачи совпадает с сегодняшней

filter = item => item.DueDate.Date == today;

Print(tasks, filter);

break;

case "2":

Console.WriteLine("Введите ключевое слово:");

string keyword = Console.ReadLine()!.ToLower();

// Присваивает делегату ИСТИНА, если в названии встречается ключевое слово

filter = item => item.Title.ToLower().Contains(keyword);

Print(tasks, filter);

break;

case "3":

filter = item => true;

exit = true;

break;

default:

Console.WriteLine("Неверный выбор. Показываются все задачи.");

filter = item => true;

break;

}

}

Console.ReadKey();

}

}

Результат выполнения программы представлен на рисунке 3.4.

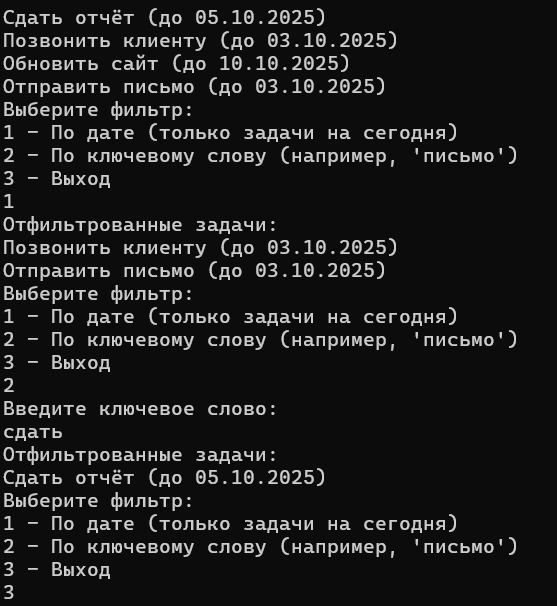


Рисунок 3.4 – Результат выполнения программы

**Задание 5.**

**Листинг 3.5.** Код программы задания 5.

public delegate void SortMethod(int[] array);

// Класс с сортировками

class SortAlgorithms

{

// Пузырьковая сортировка

public void BubbleSort(int[] array)

{

for (int i = 0; i < array.Length - 1; i++)

{

for (int j = 0; j < array.Length - 1 - i; j++)

{

if (array[j] > array[j + 1])

(array[j], array[j + 1]) = (array[j + 1], array[j]);

}

}

}

// Быстрая сортировка

public void QuickSort(int[] array) => QuickSortRecursive(array, 0, array.Length - 1);

private void QuickSortRecursive(int[] array, int left, int right)

{

if (left >= right) return;

int middle = array[(left + right) / 2];

int index = Partition(array, left, right, middle);

QuickSortRecursive(array, left, index - 1);

QuickSortRecursive(array, index, right);

}

// Для разделения на лево и право

private int Partition(int[] array, int left, int right, int middle)

{

// Пока левый указатель не пересёк правый

while (left <= right)

{

// Левый указатель двигается вправо, пока элемент меньше опорного (среднего)

while (array[left] < middle) left++;

// Правый указатель двигается влево, пока элемент больше опорного (среднего)

while (array[right] > middle) right--;

// Если левый указатель меньше или равен правому - элемент с левой части меняется с элементов в правой части

if (left <= right)

{

(array[left], array[right]) = (array[right], array[left]);

left++;

right--;

}

}

return left;

}

}

class Program

{

static void Main(string[] args)

{

// Создание объекта класса

var sorter = new SortAlgorithms();

// Ввод значений

Console.WriteLine("Введите числа через пробел:");

string input = Console.ReadLine()!;

// Разделяет строку на элементы массива по пробелам, удаляя лишние пропуски

string[] parts = input.Split(' ', StringSplitOptions.RemoveEmptyEntries);

// Преобразовывает массив строк в массив целых чисел

int[] numbers = Array.ConvertAll(parts, int.Parse); // ConvertAll применяет int.Parse ко всем элементам

SortMethod? method = null; // Создание переменной делегата

var exit = false;

while (!exit)

{

Console.WriteLine("Выберите метод сортировки:");

Console.WriteLine("1 — Сортировка пузырьком");

Console.WriteLine("2 — Быстрая сортировка");

string choice = Console.ReadLine()!;

switch (choice)

{

case "1":

method = sorter.BubbleSort; // Присваивание делегату ссылки на метод BubbleSort

method?.Invoke(numbers); // Вызов сортировки (если не null)

Console.WriteLine("Отсортированные данные:");

Console.WriteLine(string.Join(" ", numbers));

break;

case "2":

method = sorter.QuickSort; // Присваивание делегату ссылки на метод QuickSort

method?.Invoke(numbers); // Вызов сортировки (если не null)

Console.WriteLine("Отсортированные данные:");

Console.WriteLine(string.Join(" ", numbers));

break;

default:

Console.WriteLine("Неверный выбор. Используется пузырьковая сортировка.");

method = sorter.BubbleSort; // Присваивание делегату ссылки на метод BubbleSort

method?.Invoke(numbers); // Вызов сортировки (если не null)

Console.WriteLine("Отсортированные данные:");

Console.WriteLine(string.Join(" ", numbers));

exit = true;

break;

}

}

Console.ReadKey();

}

}

Результат выполнения программы показан на рисунке 3.5.

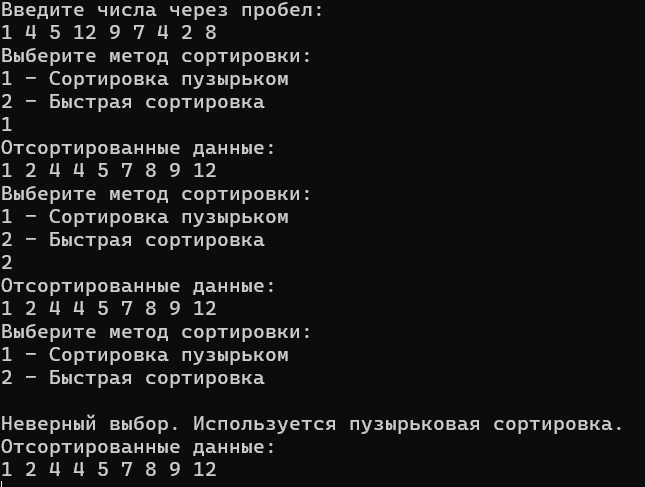


Рисунок 3.5 – Результат выполнения программы

# Модуль 4. Задания по интерфейсам

**Задания.**

1. Создайте интерфейс "Фигура" с методами для вычисления площади и периметра. Затем реализуйте этот интерфейс в классах геометрических фигур (круг, прямоугольник, треугольник).

2. Разработайте приложение для учета продуктов в магазине с использованием интерфейсов. Создайте интерфейс "Товар" с методами для определения стоимости и остатка товара на складе. Реализуйте этот интерфейс в классах различных товаров.

3. Создайте систему учета студентов в университете с помощью интерфейсов. Создайте интерфейс "Студент" с методами для определения среднего балла и получения информации о курсе. Реализуйте этот интерфейс в классах студентов разных курсов.

4. Реализуйте приложение для работы с библиотекой книг с использованием интерфейсов. Создайте интерфейс "Книга" с методами для проверки доступности и выдачи книги. Реализуйте этот интерфейс в классах различных книг.

5. Создайте приложение для рисования на холсте с использованием интерфейсов. Создайте интерфейс "Рисунок" с методами для рисования линий, кругов и прямоугольников. Реализуйте этот интерфейс в классе для работы с холстом.

**Выполнение практических заданий.**

**Задание 1.**

**Листинг 4.1.** Код программы задания 1.

interface IShape

{

void Area();

void Perimeter();

}

class Circle : IShape

{

public double Radius { get; set; }

public void Area() => Console.WriteLine($"Площадь круга с радиусом {Radius}: {Math.PI \* Math.Pow(Radius, 2):F2}");

public void Perimeter() => Console.WriteLine($"Периметр круга с радиусом {Radius}: {Math.PI \* 2 \* Radius:F2}");

}

class Rectangle : IShape

{

public double Width { get; set; }

public double Height { get; set; }

public void Area() => Console.WriteLine($"Площадь прямоугольника с шириной {Width} и высотой {Height}: {Width \* Height}");

public void Perimeter() => Console.WriteLine($"Периметр прямоугольника с шириной {Width} и высотой {Height}: {(Width \* 2) + (Height \* 2)}");

}

class Triangle : IShape

{

public double SideA { get; set; }

public double SideB { get; set; }

public double SideC { get; set; }

private double P => (SideA + SideB + SideC) / 2; // Полупериметр

public void Area() => Console.WriteLine($"Площадь треугольника со сторонами {SideA}, {SideB}, {SideC}: {Math.Sqrt(P \* (P - SideA) \* (P - SideB) \* (P - SideC))}");

public void Perimeter() => Console.WriteLine($"Периметр треугольника со сторонами {SideA}, {SideB}, {SideC}: {SideA + SideB + SideC}");

}

class Program

{

static void Main(string[] args)

{

var circle = new Circle{Radius = 2};

var rectangle = new Rectangle {Width = 5, Height = 4};

var triangle = new Triangle {SideA = 5, SideB = 3, SideC = 4};

circle.Area();

circle.Perimeter();

rectangle.Area();

rectangle.Perimeter();

triangle.Area();

triangle.Perimeter();

Console.ReadKey();

}

}

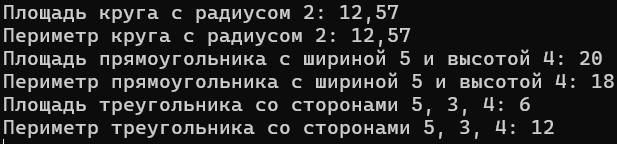


Рисунок 4.1 – Результат выполнения программы

**Задание 2.**

**Листинг 4.2.** Код программы задания 2.

interface IProduct

{

string Name { get; set; }

int Quantity { get; set; }

double Price { get; set; }

ShowStock();

void GetPrice();

}

class Water : IProduct

{

public string Name { get; set; } = "Вода";

public int Quantity { get; set; }

public double Price { get; set; }

public void ShowStock() => Console.WriteLine($"{Name} - {Quantity} шт., {Price} р.");

public void GetPrice() => Console.WriteLine($"Цена всего товара '{Name}' на складе: {Price \* Quantity} р.");

}

class Eggs : IProduct

{

public string Name { get; set; } = "Яйца";

public int Quantity { get; set; }

public double Price { get; set; }

public void ShowStock() => Console.WriteLine($"{Name} - {Quantity} шт., {Price} р.");

public void GetPrice() => Console.WriteLine($"Цена всего товара '{Name}' на складе: {Price \* Quantity} р.");

}

class Program

{

static void Main(string[] args)

{

var water = new Water{Quantity = 1000, Price = 9.79};

var eggs = new Eggs{Quantity = 100, Price = 120.79};

water.ShowStock();

water.GetPrice();

eggs.ShowStock();

eggs.GetPrice();

Console.ReadKey();

}

}

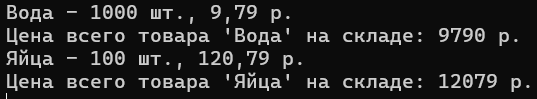


Рисунок 4.2 – Результат выполнения программы

**Задание 3.**

**Листинг 4.3.** Код программы задания 3.

interface IStudent

{

string Name { get; set; }

string Group { get; set; }

int [] Grades { get; set; }

void AvgGrades();

void ShowGrades();

void ShowGroup();

}

class FirstYear : IStudent

{

public string Name { get; set; }

public string Group { get; set; } = "1 курс";

public int [] Grades {get; set;}

public void AvgGrades()

{

var sum = 0;

foreach (var grade in Grades)

sum += grade;

Console.WriteLine($"Средний балл студента '{Name}': {sum / Grades.Length}");

}

public void ShowGrades() => Console.WriteLine($"Отметки студента '{Name}': " + string.Join(", ", Grades));

public void ShowGroup() => Console.WriteLine($"Студент '{Name}' - {Group}");

}

class SecondYear : IStudent

{

public string Name { get; set; }

public string Group { get; set; } = "2 курс";

public int [] Grades {get; set;}

public void AvgGrades()

{

var sum = 0;

foreach (var grade in Grades)

sum += grade;

Console.WriteLine($"Средний балл студента '{Name}': {sum / Grades.Length}");

}

public void ShowGrades() => Console.WriteLine($"Отметки студента '{Name}': " + string.Join(", ", Grades));

public void ShowGroup() => Console.WriteLine($"Студент '{Name}' - {Group}");

}

class Program

{

static void Main(string[] args)

{

List<IStudent> students = new List<IStudent>

{

new FirstYear { Name = "Антон А.А.", Grades = [5, 4, 5, 3, 2, 1] },

new SecondYear { Name = "Сергей С.С.", Grades = [2, 2, 3, 1, 5, 3] }

};

foreach (var student in students)

{

student.ShowGroup();

student.ShowGrades();

student.AvgGrades();

Console.WriteLine();

}

Console.ReadKey();

}

}

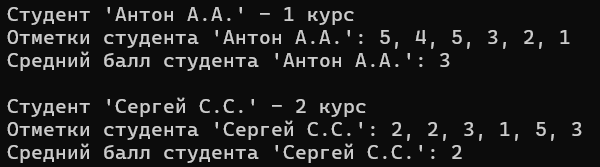


Рисунок 4.3 – Результат выполнения проргаммы

**Задание 4.**

**Листинг 4.4.** Код программы задания 4.

interface IBook

{

string Name { get; set; }

string Author { get; set; }

int Year { get; set; }

int Quantity { get; set; }

double Price { get; set; }

bool IsAvailable();

void GetBook();

}

class Neverwhere : IBook

{

public string Name { get; set; } = "Никогде";

public string Author { get; set; } = "Нил Гейман";

public int Year { get; set; } = 1996;

public int Quantity { get; set; } = 20;

public double Price { get; set; } = 19.99;

public bool IsAvailable()

{

bool isAvailable = (this.Quantity > 0) ? true : false;

Console.WriteLine(isAvailable == true ? $"Книга '{Name}' доступна" : $"Книги '{Name}' нет в наличии");

return isAvailable;

}

public void GetBook()

{

if (this.Quantity > 0)

{

this.Quantity--;

Console.WriteLine($"Книга '{Name}' выдана! Остаток: " + this.Quantity);

}

else

{

Console.WriteLine($"Книги '{Name}' нет в наличии");

}

}

}

class NorwegianWoods : IBook

{

public string Name { get; set; } = "Норвежский лес";

public string Author { get; set; } = "Харуки Мураками";

public int Year { get; set; } = 2023;

public int Quantity { get; set; } = 1;

public double Price { get; set; } = 22.27;

public bool IsAvailable()

{

bool isAvailable = (this.Quantity > 0) ? true : false;

Console.WriteLine(isAvailable ? $"Книга '{Name}' доступна" : $"Книги '{Name}' нет в наличии");

return isAvailable;

}

public void GetBook()

{

if (this.Quantity > 0)

{

this.Quantity--;

Console.WriteLine($"Книга '{Name}' выдана! Остаток: " + this.Quantity);

}

else

{

Console.WriteLine($"Книги '{Name}' нет в наличии");

}

}

}

class Program

{

static void Main(string[] args)

{

var library = new IBook[]

{

new Neverwhere(),

new NorwegianWoods(),

};

foreach (var book in library)

{

book.IsAvailable();

book.GetBook();

book.GetBook();

Console.WriteLine();

}

Console.ReadKey();

}

}

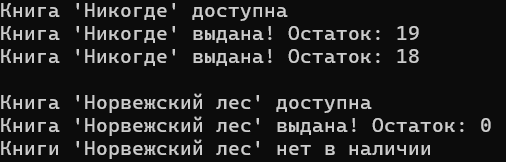


Рисунок 4.4 – Результат выполнения программы

**Задание 5.**

**Листинг 4.5.** Код программы задания 5.

interface IDrawing

{

void DrawLine(int x1, int y1, int x2, int y2);

void DrawRectangle(int x1, int y1, int x2, int y2);

void DrawCircle(int centerX, int centerY, int radius);

}

class Draw : IDrawing

{

public void DrawLine(int x1, int y1, int x2, int y2)

{

Console.WriteLine($"Нарисована линия из точки ({x1};{y1}) в точку ({x2};{y2})");

}

public void DrawRectangle(int x1, int y1, int x2, int y2)

{

Console.WriteLine($"Нарисован прямоугольник из точки ({x1};{y1}) в точку ({x2};{y2})");

}

public void DrawCircle(int centerX, int centerY, int radius)

{

Console.WriteLine($"Нарисован круг с центром в точке ({centerX};{centerY}) и радиусом {radius}");

}

public bool TryParseCoords(string input, int expectedCount, out int[] result)

{

var parts = input.Split(' ', StringSplitOptions.RemoveEmptyEntries);

result = new int[expectedCount];

if (parts.Length < expectedCount) return false;

for (int i = 0; i < expectedCount; i++)

{

if (!int.TryParse(parts[i], out result[i]))

return false;

}

return true;

}

}

class Program

{

static void Main(string[] args)

{

var draw = new Draw();

var exit = false;

while (!exit)

{

Console.WriteLine($"1 - Нарисовать линию \n" +

$"2 - Нарисовать прямоугольник \n" +

$"3 - Нарисовать круг \n" +

$"4 - Выход");

var input = Console.ReadLine()!.Trim();

Console.WriteLine();

string[] coords;

switch (input)

{

case "1":

Console.Write("Введите x1 y1 x2 y2: ");

if (draw.TryParseCoords(Console.ReadLine()!, 4, out int[] line))

draw.DrawLine(line[0], line[1], line[2], line[3]);

else

Console.WriteLine("Ошибка: нужно 4 целых числа.");

break;

case "2":

Console.Write("Введите x1 y1 x2 y2: ");

if (draw.TryParseCoords(Console.ReadLine()!, 4, out int[] rect))

draw.DrawRectangle(rect[0], rect[1], rect[2], rect[3]);

else

Console.WriteLine("Ошибка: нужно 4 целых числа.");

break;

case "3":

Console.Write("Введите centerX centerY radius: ");

if (draw.TryParseCoords(Console.ReadLine()!, 3, out int[] circle))

draw.DrawCircle(circle[0], circle[1], circle[2]);

else

Console.WriteLine("Ошибка: нужно 3 целых числа.");

break;

case "4":

exit = true;

break;

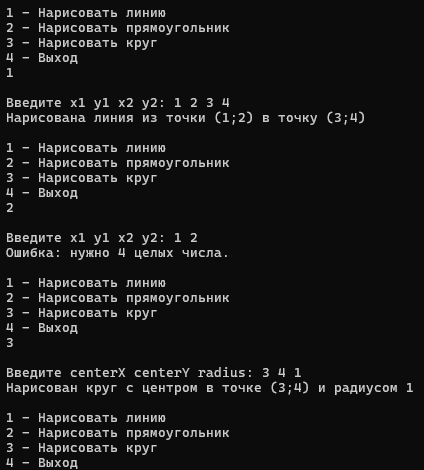
}

Console.WriteLine();

}

}

}

  
Рисунок 4.5 - Результат выполнения программы

# Модуль 5. Графический интерфейс

**Задания.**

1. Создайте графическое приложение для рисования на холсте с использованием Windows Forms. Реализуйте функции рисования линий, кругов и квадратов.

2. Разработайте текстовый редактор с возможностью открытия и сохранения текстовых файлов.

3. Создайте приложение для учета задач с использованием Windows Forms. Пользователь должен иметь возможность добавлять, удалять и отмечать задачи как выполненные.

4. Реализуйте приложение для просмотра изображений. Пользователь должен иметь возможность выбирать изображение для просмотра и масштабировать его.

5. Создайте калькулятор с графическим интерфейсом. Пользователь должен иметь возможность выполнять арифметические операции.

**Выполнение практических заданий**

**Задание 1.**

**Листинг 5.1.** Код программы задания 1.

using System.Drawing.Drawing2D;

namespace Module\_5.\_1;

class Shape

{

public int State; // 1 - линия, 2 - прямоугольник, 3 - круг

public Point Start;

public Point End;

public Rectangle GetBounds()

{

return new Rectangle(

Math.Min(Start.X, End.X),

Math.Min(Start.Y, End.Y),

Math.Abs(End.X - Start.X),

Math.Abs(End.Y - Start.Y));

}

}

public partial class Form1 : Form

{

private List<Shape> shapes = new List<Shape>(); // Список всех нарисованных фигу

private Point startPoint, endPoint; // Точки начала и конца текущей фигуры

private float PenWidth = 3; // Толщина пера

private int \_state = 1; // Текущий выбранный тип фигуры

public Form1()

{

InitializeComponent();

}

private void Form\_Paint(object sender, PaintEventArgs e)

{

Pen pen = new Pen(Color.Black, PenWidth);

foreach (var shape in shapes)

{

switch (shape.State)

{

case 1:

e.Graphics.DrawLine(pen, shape.Start, shape.End);

break;

case 2:

e.Graphics.DrawRectangle(pen, shape.GetBounds());

break;

case 3:

e.Graphics.DrawEllipse(pen, shape.GetBounds());

break;

}

}

pen.Dispose();

}

private void Form\_MouseDown(object sender, MouseEventArgs e)

{

startPoint = e.Location;

}

private void Form\_MouseUp(object sender, MouseEventArgs e)

{

endPoint = e.Location;

shapes.Add(new Shape { State = \_state, Start = startPoint, End = endPoint });

Invalidate();

}

private void btnLine\_Click(object sender, EventArgs e)

{

\_state = 1;

}

private void btnRect\_Click(object sender, EventArgs e)

{

\_state = 2;

}

private void btnCircle\_Click(object sender, EventArgs e)

{

\_state = 3;

}

}



Рисунок 5.1 – Результат выполнения программы

**Задание 2.**

**Листинг 5.2.** Код программы задания 2.

namespace Module\_5.\_2;

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void btnOpen\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text Files (\*.txt)|\*.txt|All Files (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == DialogResult.OK)

textBox1.Text = System.IO.File.ReadAllText(openFileDialog.FileName);

}

private void btnSave\_Click(object sender, EventArgs e)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Text Files (\*.txt)|\*.txt|All Files (\*.\*)|\*.\*";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

System.IO.File.WriteAllText(saveFileDialog.FileName, textBox1.Text);

}

}



Рисунок 5.2 – Результат выполнения программы

**Задание 3.**

**Листинг 5.3.** Код программы задания 3.

using System.Collections.Generic;

namespace Module\_5.\_3;

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void btnAdd\_Click(object sender, EventArgs e)

{

string task = textBox1.Text.Trim();

if (!string.IsNullOrEmpty(task))

{

checkedListBox1.Items.Add(task);

textBox1.Clear();

textBox1.Focus();

}

}

private void btnDel\_Click(object sender, EventArgs e)

{

// Удаляем в обратном порядке чтобы индексы не сдвигались

for (int i = checkedListBox1.Items.Count - 1; i >= 0; i--)

{

if (checkedListBox1.GetItemChecked(i))

{

checkedListBox1.Items.RemoveAt(i);

}

}

}

}

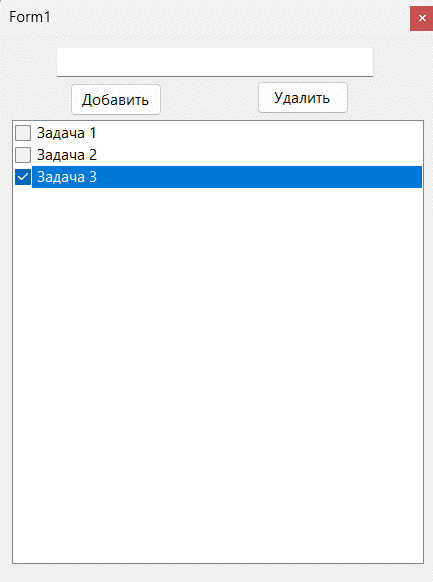


Рисунок 5.3 – Результат выполнения программы

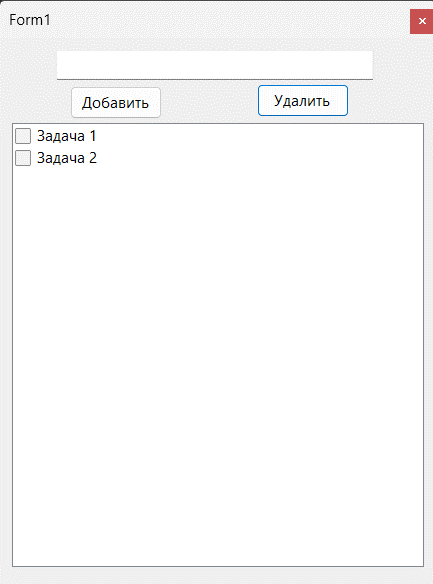


Рисунок 5.4 – Результат выполнения программы

**Задание 4.**

**Листинг 5.4.** Код программы задания 4.

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Module\_5.\_4

{

public partial class Form1 : Form

{

private float scale = 1.0f;

private Image originalImage;

public Form1()

{

InitializeComponent();

InitializeContainer();

}

private void InitializeContainer()

{

panel1.AutoScroll = false; // отключаем прокрутку

pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;

pictureBox1.Location = new Point(0, 0);

panel1.Controls.Add(pictureBox1);

panel1.MouseWheel += Panel1\_MouseWheel;

}

private void btnOpen\_Click(object sender, EventArgs e)

{

openFileDialog1.Filter = "PNG Images (\*.png)|\*.png|JPG Images (\*.jpg)|\*.jpg";

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

originalImage = Image.FromFile(openFileDialog1.FileName);

pictureBox1.Image = originalImage;

pictureBox1.Size = new Size(originalImage.Width, originalImage.Height);

CenterImageInPanel();

scale = 1.0f;

}

}

private void Panel1\_MouseWheel(object sender, MouseEventArgs e)

{

if (originalImage == null || Control.ModifierKeys != Keys.Control)

return;

scale \*= e.Delta > 0 ? 1.1f : 0.9f;

scale = Math.Clamp(scale, 0.1f, 10f);

int newWidth = (int)(originalImage.Width \* scale);

int newHeight = (int)(originalImage.Height \* scale);

pictureBox1.Size = new Size(newWidth, newHeight);

CenterImageInPanel();

((HandledMouseEventArgs)e).Handled = true;

}

private void CenterImageInPanel()

{

if (pictureBox1.Image == null) return;

int x = (panel1.Width - pictureBox1.Width) / 2;

int y = (panel1.Height - pictureBox1.Height) / 2;

pictureBox1.Location = new Point(x, y);

}

}

}

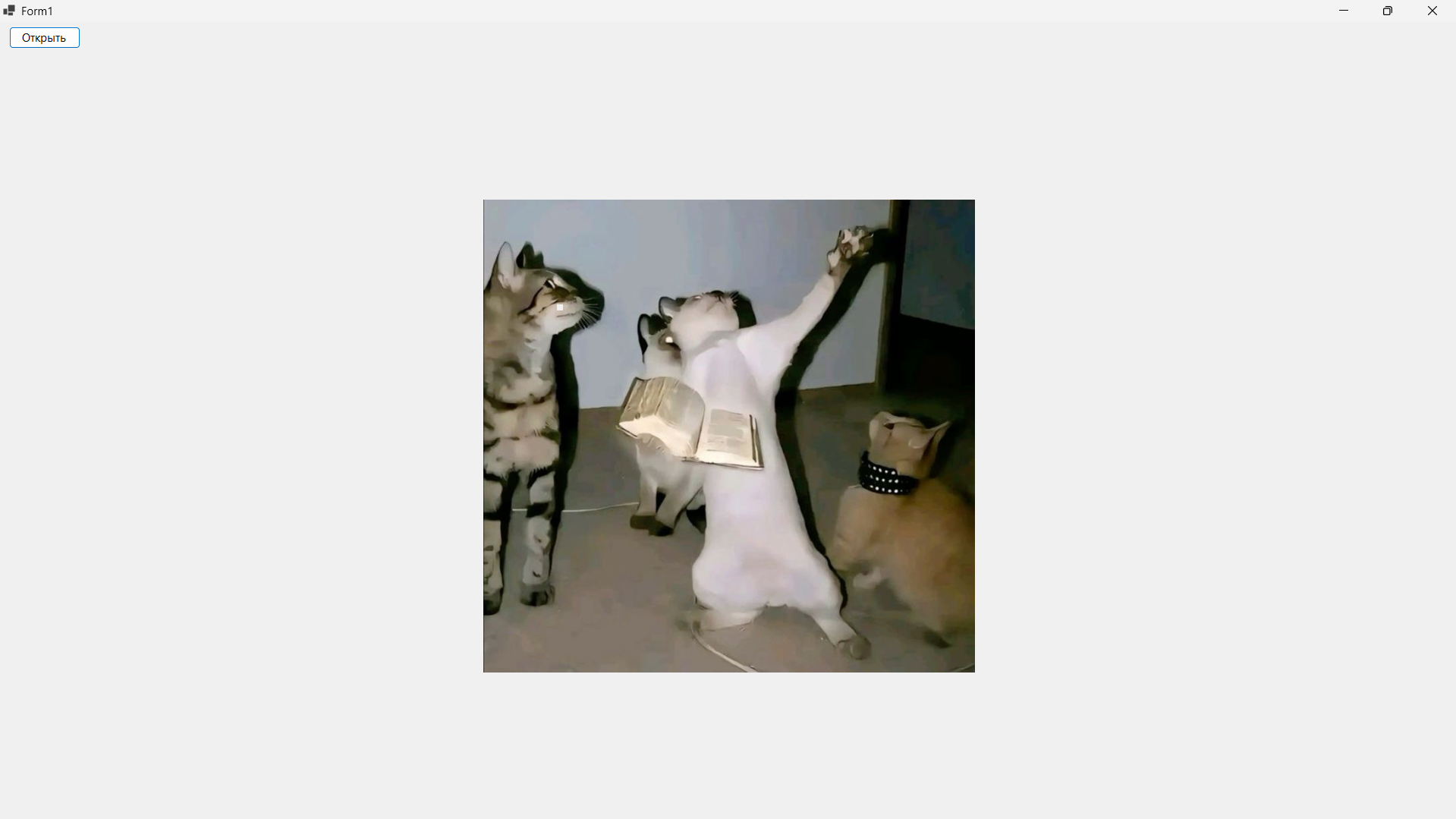


Рисунок 5.5 – Результат выполнения программы

**Задание 5.**

**Листинг 5.5.** Код программы задания 5.

namespace Module\_5.\_5;

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void btnResult\_Click(object sender, EventArgs e)

{

double.TryParse(numberOne.Text, out double numOne);

double.TryParse(numberTwo.Text, out double numTwo);

var sum = numOne + numTwo;

var sub = numOne - numTwo;

var mult = numOne \* numTwo;

var div = numOne / numTwo;

if (btnAdd.Checked)

{

result.Text = sum.ToString();

lbOp.Text = "+";

}

if (btnSub.Checked)

{

result.Text = sub.ToString();

lbOp.Text = "-";

}

if (btnMult.Checked)

{

result.Text = mult.ToString();

lbOp.Text = "\*";

}

if (btnDiv.Checked)

{

if (numTwo != 0)

{

result.Text = div.ToString();

lbOp.Text = "/";

}

else Console.WriteLine("Нельзя делить на 0!");

}

}

}

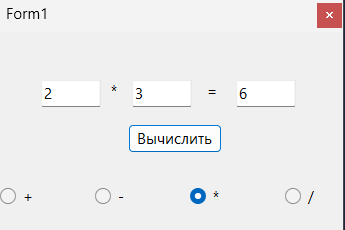


Рисунок 5.6 – Результат выполнения программы

# Модуль 6. Базы данных

**Задания. (Вариант 2)**

Разработайте систему учета задач с использованием базы данных. Пользователь должен иметь возможность добавлять, удалять и обновлять задачи.

**Выполнение практического задания.**

**Задание.**

**Листинг 6.1.** Код программы задания 1.

using Microsoft.Data.Sqlite;

namespace Module\_6;

public partial class Form1 : Form

{

private string \_selectedMenuItem;

private ContextMenuStrip \_contextMenuStrip1;

private readonly string \_path = "C:\\Users\\DKJ\\Desktop\\Praktika\\Modules\\Module-6\\TasksBD.db";

private string \_text;

public Form1()

{

InitializeComponent();

InitializeMenu();

}

// Инициализация контекстного меню

private void InitializeMenu()

{

var toolStripMenuItem1 = new ToolStripMenuItem{Text = "Удалить"};

toolStripMenuItem1.Click += toolStripMenuItem1\_Click;

var toolStripMenuItem2 = new ToolStripMenuItem { Text = "Изменить" };

toolStripMenuItem2.Click += toolStripMenuItem2\_Click;

\_contextMenuStrip1 = new ContextMenuStrip();

\_contextMenuStrip1.Items.AddRange(new ToolStripMenuItem[] {toolStripMenuItem1, toolStripMenuItem2});

listBox1.MouseDown += listBox1\_MouseDown;

}

// Работа со второй формой

private void ShowUpdateDialog()

{

UpdateDialog updateDialog = new UpdateDialog();

if (updateDialog.ShowDialog() == DialogResult.OK)

{

\_text = updateDialog.UpdateText;

}

updateDialog.Dispose();

}

// ОБРАБОТЧИКИ НАЖАТИЙ

// Удалить запись

private void toolStripMenuItem1\_Click(object sender, EventArgs e)

{

DeleteTasks();

}

// Изменить запись

private void toolStripMenuItem2\_Click(object sender, EventArgs e)

{

ShowUpdateDialog();

UpdateTasks();

}

// Открыть контекстное меню

private void listBox1\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right)

{

int index = listBox1.IndexFromPoint(e.Location);

if (index != ListBox.NoMatches)

{

listBox1.SelectedIndex = index; // выделяем элемент

\_selectedMenuItem = listBox1.Items[index].ToString();

\_contextMenuStrip1.Show(listBox1, e.Location); // показываем меню в точке клика

}

else

{

\_contextMenuStrip1.Hide(); // скрываем меню, если клик вне элемента

}

}

}

// Добавить задачу

private void button1\_Click(object sender, EventArgs e)

{

AddTasks();

GetTasks();

}

// РАБОТА С БД

// Получить значения из БД

private void GetTasks()

{

string sqlExpression = "SELECT \* FROM Tasks";

using (var connection =

new SqliteConnection($"Data Source={\_path}"))

{

connection.Open();

SqliteCommand commandReader = new SqliteCommand(sqlExpression, connection);

listBox1.Items.Clear();

using (SqliteDataReader reader = commandReader.ExecuteReader())

{

if (reader.HasRows)

{

while (reader.Read())

{

var id = reader.GetValue(0);

var taskDesc = reader.GetValue(1);

listBox1.Items.Add($"{id}. {taskDesc}");

}

}

}

}

}

// Изменение задачи в БД

private void UpdateTasks()

{

if (listBox1.SelectedItem != null)

{

string selectedItem = listBox1.SelectedItem.ToString();

int dotIndex = selectedItem.IndexOf('.');

if (dotIndex > -1)

{

string idStr = selectedItem.Substring(0, dotIndex);

if (int.TryParse(idStr, out int taskId))

{

string sqlExpression = "UPDATE Tasks SET TaskDesc = @desc WHERE TaskID = @id";

using (var connection = new SqliteConnection($"Data Source={\_path}"))

{

connection.Open();

using (var command = new SqliteCommand(sqlExpression, connection))

{

command.Parameters.AddWithValue("@desc", \_text);

command.Parameters.AddWithValue("@id", taskId);

command.ExecuteNonQuery();

}

}

GetTasks(); // обновить список

}

}

}

}

// Удаление задачи из БД

private void DeleteTasks()

{

if (listBox1.SelectedIndex > -1)

{

if (listBox1.SelectedItem != null)

{

string selectedItem = listBox1.SelectedItem.ToString() ?? string.Empty;

int dotIndex = selectedItem.IndexOf('.');

if (dotIndex > -1)

{

string idStr = selectedItem.Substring(0, dotIndex);

if (int.TryParse(idStr, out int taskId))

{

string sqlExpression = $"DELETE FROM Tasks WHERE TaskID = {taskId}";

using var connection = new SqliteConnection($"Data Source={\_path}");

connection.Open();

using var command = new SqliteCommand(sqlExpression, connection);

command.ExecuteNonQuery();

GetTasks();

}

}

}

}

}

// Добавление задачи в БД

private void AddTasks()

{

string taskDescription = textBox1.Text;

using (var connection =

new SqliteConnection($"Data Source={\_path}"))

{

connection.Open();

SqliteCommand command = connection.CreateCommand();

command.Connection = connection;

command.CommandText = "INSERT INTO Tasks (TaskDesc) VALUES (@desc)";

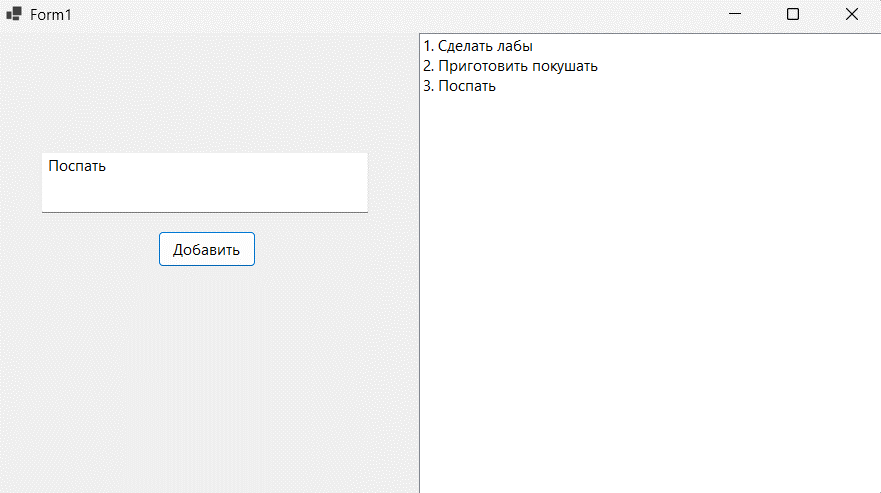
command.Parameters.AddWithValue("@desc", taskDescription);

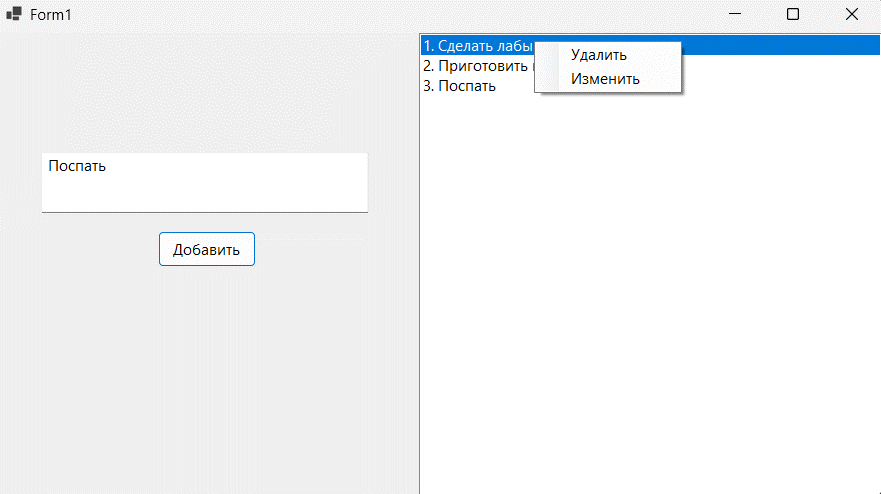
command.ExecuteNonQuery();

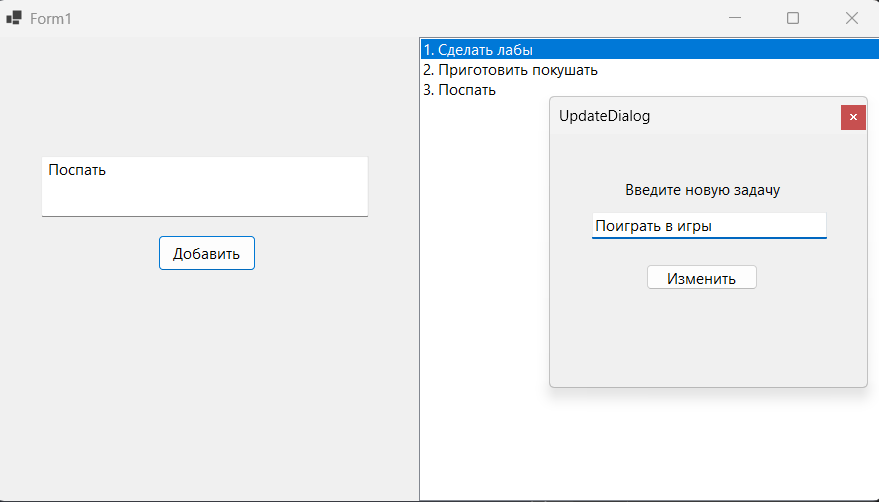
};

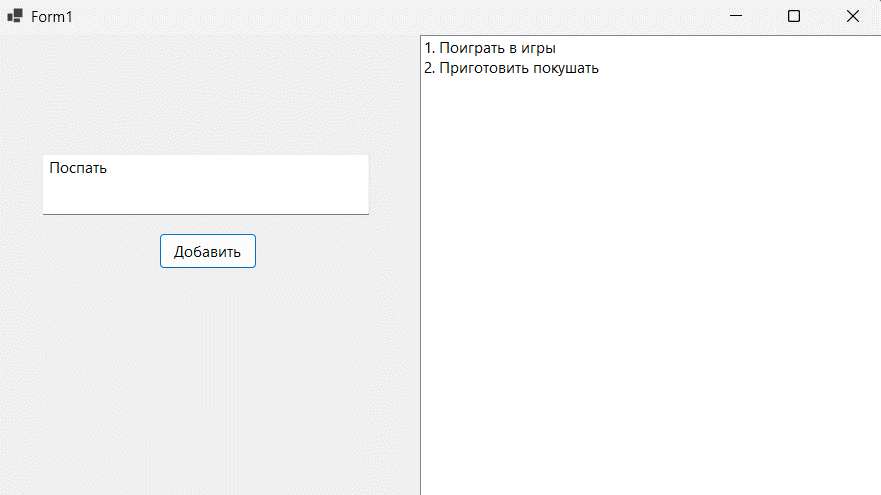
}

}









# Модуль 7. Проектная работа

**Задания.**

1. Выберите проект, который вы разработали в рамках предмета. Основы алгоритмизации и программирования.
2. Сформируйте требования к проекту и создайте план разработки.
3. Разработайте проект, следуя плану и используя полученные знания и навыки.
4. Проведите тестирование и отладку проекта.
5. Подготовьте документацию и презентацию для защиты проекта перед преподавателями.

**Выполнение практических заданий.**

**Задание 1.**

В рамках учебного предмета «Основы алгоритмизации и программирования» была выбрана тема: «Отображение 3D объектов с помощью OpenGL в Delphi». Суть темы заключается в разработке программы с возможностью загрузки и визуализации модели из .obj файла.

**Задание 2.**

Функциональные требования:

1. Визуализация 3D-модели

* рендеринг 3D-модели;
* применение базовых параметров освещения;
* применение текстуры, если она есть в .mtl файле.

1. Управление 3D-моделью

* вращение модели с помощью мыши;
* масштабирование модели с помощью колесика мыши;

1. Добавление 3D-модели из файла

* загрузка модели из .obj файла;
* проверка на наличие .mtl файла;
* проверка на наличие .jpg файла с текстурой;

Для начала, следует выделить, что в самой теме есть определенное правило – не использовать сторонние библиотеки, а только «чистый» OpenGL. Это невозможно, т.к., во-первых, программу необходимо написать на языке программирования C#, и, во-вторых, в C# нет встроенной библиотеки OpenGL. Поэтому разработка будет идти на основе библиотеки OpenTK 3.3.3. Обоснование использования устаревшей версии в том, что в оригинальном проекте по предмету «Основы алгоритмизации и программирования» тоже использовались устаревшие методы OpenGL из-за отсутствия поддержки новых. Чтобы не разрушать и не переписывать логику программы с нуля, было принято решение использовать те же устаревшие методы.

Необходимо выделить основные классы, которые необходимо реализовать в программе.

Game (наследник GameWindow) – главный класс приложения, управляющий его жизненным циклом, отвечает за отображение и рендеринг окна. Данный класс будет иметь свои ключевые методы:

* OnLoad() – инициализация OpenGL, настройка освеещения и материалов.
* OnRenderFrame() – основной цикл рендеринга.
* OnUpdateFrame() – обработка ввода и обновление состояния.
* OnResize() – обработка изменения размера окна.

ObjModel – класс для загрузки и хранения 3D-моделей в формате .obj. Ключевые методы:

* Load() – парсинг .obj файла и создание модели.
* LoadMtl() – загрузка материалов из MTL-файлов.
* GenerateNormals() – вычисление нормалей, если они отсутствуют.

Классы Vertex, Face, Material – структуры данных для представления геометрии, содержат координаты, текстурные координаты, нормали, индексы вершин.

TextureLoader – загрузка OpenGL текстур.

Этапы разработки:

1. Настройка окружения – создание проекта на языке программирования C#, подключение библиотеки OpenTK. Настройка окна рендеринга (GameWindow) и базовая конфигурация OpenGL контекста.
2. Создание системы рендеринга. На данном этапе происходит реализация методов OnLoad(), OnRenderFrame(), OnResize(), настройка матриц и организация буферов цвета и глубины.
3. Написание парсера для файлов формата .obj. Парсер должен считывать вершины, текстурные координаты и нормали в соответствии со структурой .obj файлов, уметь их обрабатывать и разбивать. Необходимо реализовать обработку полигонов с различным количество вершин и добавить поддержку относительных индексов (индексы, которые идут с конца) в формате .obj.
4. Создание системы обработки и загрузки материалов. Здесь будет разработан парсер для файлов типа .mtl. Добавлена возможность загрузки текстурных карт и управления материалами и их свойствами.
5. Реализация рендеринга геометрии – написание метода DrawModel() с поддержкой различных параметров, настройка текстур и материалов для каждого полигона, автоматическая генерация нормалей.
6. Добавление освещения и материалов – настройка источника свет (SetupLighting()), конфигурация свойств материалов (SetupMaterials()).
7. Настройка управления с помощью мыши и клавиатуры. На этом этапе добавляется возможность управление камерой с помощью мыши, а также открытие диалога выбора файлов по нажатию на клавишу буквы «W».

**Задание 3.**

**Листинг 7.1.** Код программы модуля 7.

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

using OpenTK.Input;

using System.Drawing.Imaging;

using PixelFormat = System.Drawing.Imaging.PixelFormat;

using System.Globalization;

namespace Module\_7

{

public static class Program

{

[STAThread]

public static void Main()

{

// Создание и запуск главного игрового окна

using (var game = new Game(800, 600, "3D Model Viewer"))

{

game.Run(60.0);

}

}

}

public class Game : GameWindow

{

private ObjModel model = null;

private float angleX = 0, angleY = 0;

private int mouseX, mouseY;

private float modelScale = 1;

private bool isPressed = false;

public Game(int width, int height, string title)

: base(width, height, GraphicsMode.Default, title)

{

}

protected override void OnLoad(EventArgs e)

{

base.OnLoad(e);

// Настройка базовых параметров OpenGL

GL.ClearColor(0.3f, 0.4f, 0.7f, 1.0f);

GL.Enable(EnableCap.DepthTest);

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Multisample);

GL.Enable(EnableCap.Lighting);

GL.Enable(EnableCap.Light0);

GL.Enable(EnableCap.Normalize);

SetupLighting();

SetupMaterials();

}

private void SetupLighting()

{

// Настройка параметров источника света

float[] ambient = { 0.2f, 0.2f, 0.2f, 1.0f };

float[] diffuse = { 0.8f, 0.8f, 0.8f, 1.0f };

float[] specular = { 1.0f, 1.0f, 1.0f, 1.0f };

GL.Light(LightName.Light0, LightParameter.Ambient, ambient);

GL.Light(LightName.Light0, LightParameter.Diffuse, diffuse);

GL.Light(LightName.Light0, LightParameter.Specular, specular);

}

private void SetupMaterials()

{

// Настройка параметров материала по умолчанию

float[] matAmbient = { 0.3f, 0.3f, 0.3f, 1.0f };

float[] matDiffuse = { 0.6f, 0.6f, 0.6f, 1.0f };

float[] matSpecular = { 0.9f, 0.9f, 0.9f, 1.0f };

float[] matShininess = { 50.0f };

GL.Material(MaterialFace.Front, MaterialParameter.Ambient, matAmbient);

GL.Material(MaterialFace.Front, MaterialParameter.Diffuse, matDiffuse);

GL.Material(MaterialFace.Front, MaterialParameter.Specular, matSpecular);

GL.Material(MaterialFace.Front, MaterialParameter.Shininess, matShininess);

}

protected override void OnResize(EventArgs e)

{

base.OnResize(e);

// Установка области отрисовки

GL.Viewport(0, 0, Width, Height);

// Настройка проекционной матрицы

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

Matrix4 perspective = Matrix4.CreatePerspectiveFieldOfView(

MathHelper.DegreesToRadians(45f),

Width / (float)Height,

0.1f,

100f);

GL.LoadMatrix(ref perspective);

}

protected override void OnUpdateFrame(FrameEventArgs e)

{

base.OnUpdateFrame(e);

// Обработка выхода из приложения

if (Keyboard.GetState().IsKeyDown(Key.Escape))

Exit();

// Обработка загрузки модели по нажатию W

if (Keyboard.GetState().IsKeyDown(Key.W))

{

string path = ShowOpenFileDialog();

if (!string.IsNullOrEmpty(path))

{

model = new ObjModel().Load(path);

}

}

}

protected override void OnRenderFrame(FrameEventArgs e)

{

base.OnRenderFrame(e);

// Очистка буферов цвета и глубины

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

SetupModelView();

// Отрисовка модели, если она загружена

if (model != null)

DrawModel(model);

SwapBuffers();

}

private void SetupModelView()

{

// Настройка матрицы модели-вида

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadIdentity();

GL.Translate(0.0f, 0.0f, -5.0f);

GL.Scale(modelScale, modelScale, modelScale);

GL.Rotate(angleX, 1.0f, 0.0f, 0.0f);

GL.Rotate(angleY, 0.0f, 1.0f, 0.0f);

}

private void DrawModel(ObjModel model)

{

// Отрисовка всех полигонов модели

foreach (var face in model.Faces)

{

// Проверка наличия материала и текстуры для текущего полигона

if (model.Materials.TryGetValue(face.MaterialName, out var mat) && mat.DiffuseTextureID != 0)

{

GL.Enable(EnableCap.Texture2D);

GL.BindTexture(TextureTarget.Texture2D, mat.DiffuseTextureID);

}

else

{

GL.Disable(EnableCap.Texture2D);

}

BeginPrimitive(face.VertexIndices.Length);

// Отрисовка всех вершин полигона

foreach (var index in face.VertexIndices)

{

var vertex = model.Vertices[index];

GL.Normal3(vertex.Normal);

GL.TexCoord2(vertex.TexCoord);

GL.Vertex3(vertex.Position);

}

GL.End();

}

}

private void BeginPrimitive(int vertexCount)

{

// Выбор типа примитива в зависимости от количества вершин

switch (vertexCount)

{

case 3: GL.Begin(PrimitiveType.Triangles); break;

case 4: GL.Begin(PrimitiveType.Quads); break;

default: GL.Begin(PrimitiveType.Polygon); break;

}

}

protected override void OnMouseDown(MouseButtonEventArgs e)

{

// Обработка нажатия левой кнопки мыши для начала вращения

if (e.Button == MouseButton.Left)

{

mouseX = e.X;

mouseY = e.Y;

isPressed = true;

}

}

protected override void OnMouseUp(MouseButtonEventArgs e)

{

// Обработка отпускания кнопки мыши

isPressed = false;

}

protected override void OnMouseMove(MouseMoveEventArgs e)

{

// Обработка вращения модели при зажатой левой кнопке мыши

if (isPressed)

{

angleY += (e.X - mouseX) \* 0.5f;

angleX += (e.Y - mouseY) \* 0.5f;

mouseX = e.X;

mouseY = e.Y;

}

}

protected override void OnMouseWheel(MouseWheelEventArgs e)

{

// Обработка масштабирования модели колесом мыши

float minScale = 0.005f;

float maxScale = 10f;

modelScale \*= e.Delta > 0 ? 1.1f : 1/1.1f;

modelScale = MathHelper.Clamp(modelScale, minScale, maxScale);

}

public string ShowOpenFileDialog()

{

string selectedPath = null;

// Создание диалога выбора файла в отдельном потоке

var thread = new Thread(() =>

{

using (var dialog = new OpenFileDialog())

{

dialog.Filter = "OBJ files (\*.obj)|\*.obj";

dialog.Title = "Select a 3D Model";

if (dialog.ShowDialog() == DialogResult.OK)

{

selectedPath = dialog.FileName;

}

}

});

thread.SetApartmentState(ApartmentState.STA);

thread.Start();

thread.Join();

return selectedPath;

}

}

public class Vertex

{

public Vector3 Position;

public Vector2 TexCoord;

public Vector3 Normal;

public Vertex(Vector3 pos)

{

Position = pos;

TexCoord = Vector2.Zero;

Normal = Vector3.Zero;

}

}

public class Face

{

public int[] VertexIndices;

public string MaterialName;

}

public class Material

{

public string Name;

public Vector3 Ambient = new(0.2f, 0.2f, 0.2f);

public Vector3 Diffuse = new(0.8f, 0.8f, 0.8f);

public Vector3 Specular = new(1.0f, 1.0f, 1.0f);

public string DiffuseTexturePath;

public int DiffuseTextureID;

public string AmbientTexturePath;

public int AmbientTextureID;

public string BumpTexturePath;

public int BumpTextureID;

}

public class ObjModel

{

public List<Vertex> Vertices = new();

public List<Vector2> TexCoords = new();

public List<Vector3> Normals = new();

public List<Face> Faces = new();

public Dictionary<string, Material> Materials = new();

public string CurrentMaterial = null;

public ObjModel Load(string path)

{

var model = new ObjModel();

// Чтение всех строк из OBJ файла

var lines = File.ReadLines(path);

// Обработка каждой строки файла

foreach (var rawLine in lines)

{

// Замена запятых на точки для корректного парсинга чисел

string line = rawLine.Replace(',', '.');

var parts = line.Trim().Split(' ', StringSplitOptions.RemoveEmptyEntries);

if (parts.Length == 0) continue;

ProcessLine(parts, model, path);

}

GenerateNormals(model);

return model;

}

private void ProcessLine(string[] parts, ObjModel model, string path)

{

// Обработка различных типов строк в OBJ файле

switch (parts[0])

{

case "v":

// Обработка вершин (координаты x, y, z)

model.Vertices.Add(new Vertex(new Vector3(

float.Parse(parts[1], CultureInfo.InvariantCulture),

float.Parse(parts[2], CultureInfo.InvariantCulture),

float.Parse(parts[3], CultureInfo.InvariantCulture))));

break;

case "vt":

// Обработка текстурных координат (u, v)

model.TexCoords.Add(new Vector2(

float.Parse(parts[1], CultureInfo.InvariantCulture),

float.Parse(parts[2], CultureInfo.InvariantCulture)));

break;

case "vn":

// Обработка нормалей (nx, ny, nz)

model.Normals.Add(new Vector3(

float.Parse(parts[1], CultureInfo.InvariantCulture),

float.Parse(parts[2], CultureInfo.InvariantCulture),

float.Parse(parts[3], CultureInfo.InvariantCulture)));

break;

case "f":

// Обработка полигонов (индексы вершин)

ProcessFace(parts, model);

break;

case "usemtl":

// Установка текущего материала

model.CurrentMaterial = parts[1];

break;

case "mtllib":

// Загрузка файла материалов

var mtlPath = Path.Combine(Path.GetDirectoryName(path), parts[1]);

LoadMtl(mtlPath, model);

break;

}

}

private void ProcessFace(string[] parts, ObjModel model)

{

var face = new Face();

var vIndices = new List<int>();

// Обработка всех вершин в определении полигона

for (int i = 1; i < parts.Length; i++)

{

// Разбиение на вершины/текстуры/нормали

var tokens = parts[i].Split('/');

int vIdx = int.Parse(tokens[0]) - 1;

vIndices.Add(vIdx);

var vertex = model.Vertices[vIdx];

// Обработка текстурных координат, если они есть

if (tokens.Length > 1 && !string.IsNullOrEmpty(tokens[1]))

{

int tIdx = int.Parse(tokens[1]) - 1;

if (tIdx >= 0 && tIdx < model.TexCoords.Count)

vertex.TexCoord = model.TexCoords[tIdx];

}

// Обработка нормалей, если они есть

if (tokens.Length > 2 && !string.IsNullOrEmpty(tokens[2]))

{

int nIdx = int.Parse(tokens[2]) - 1;

if (nIdx >= 0 && nIdx < model.Normals.Count)

vertex.Normal = model.Normals[nIdx];

}

model.Vertices[vIdx] = vertex;

}

face.VertexIndices = vIndices.ToArray();

face.MaterialName = model.CurrentMaterial;

model.Faces.Add(face);

}

public void LoadMtl(string mtlPath, ObjModel model)

{

// Проверка существования файла материалов

if (!File.Exists(mtlPath)) return;

var lines = File.ReadAllLines(mtlPath);

Material current = null;

// Обработка всех строк в MTL файле

foreach (var raw in lines)

{

var line = raw.Trim();

// Пропуск пустых строк и комментариев

if (line == "" || line.StartsWith("#")) continue;

var parts = line.Split(new[] { ' ', '\t' }, StringSplitOptions.RemoveEmptyEntries);

if (parts.Length < 2) continue;

// Обработка различных параметров материала

switch (parts[0])

{

case "newmtl":

// Создание нового материала

if (current != null && !string.IsNullOrEmpty(current.Name))

model.Materials[current.Name] = current;

current = new Material { Name = parts[1] };

break;

case "Ka": current.Ambient = ParseVec3(parts); break;

case "Kd": current.Diffuse = ParseVec3(parts); break;

case "Ks": current.Specular = ParseVec3(parts); break;

case "map\_Kd":

// Загрузка диффузной текстуры

current.DiffuseTexturePath = Path.Combine(Path.GetDirectoryName(mtlPath), parts[1]);

current.DiffuseTextureID = TextureLoader.LoadTexture(current.DiffuseTexturePath);

break;

case "map\_Ka":

// Загрузка ambient текстуры

current.AmbientTexturePath = Path.Combine(Path.GetDirectoryName(mtlPath), parts[1]);

current.AmbientTextureID = TextureLoader.LoadTexture(current.AmbientTexturePath);

break;

case "map\_bump":

case "bump":

// Загрузка bump текстуры

current.BumpTexturePath = Path.Combine(Path.GetDirectoryName(mtlPath), parts[1]);

current.BumpTextureID = TextureLoader.LoadTexture(current.BumpTexturePath);

break;

}

}

// Добавление последнего материала в словарь

if (current != null && !string.IsNullOrEmpty(current.Name))

model.Materials[current.Name] = current;

}

private Vector3 ParseVec3(string[] parts)

{

// Парсинг вектора из трех компонентов

return new Vector3(

float.Parse(parts[1], CultureInfo.InvariantCulture),

float.Parse(parts[2], CultureInfo.InvariantCulture),

float.Parse(parts[3], CultureInfo.InvariantCulture));

}

private void GenerateNormals(ObjModel model)

{

// Проверка необходимости генерации нормалей

if (model.Vertices.Count == 0 || model.Vertices[0].Normal != Vector3.Zero)

return;

var tempNormals = new Vector3[model.Vertices.Count];

// Вычисление нормалей для каждого полигона

foreach (var face in model.Faces)

{

var v0 = model.Vertices[face.VertexIndices[0]].Position;

var v1 = model.Vertices[face.VertexIndices[1]].Position;

var v2 = model.Vertices[face.VertexIndices[2]].Position;

// Вычисление нормали полигона через векторное произведение

var edge1 = v1 - v0;

var edge2 = v2 - v0;

var faceNormal = Vector3.Cross(edge1, edge2).Normalized();

// Добавление нормали полигона ко всем его вершинам

foreach (var idx in face.VertexIndices)

{

tempNormals[idx] += faceNormal;

}

}

// Нормализация результирующих нормалей вершин

for (int i = 0; i < model.Vertices.Count; i++)

{

var n = tempNormals[i];

model.Vertices[i].Normal = n.Length > 0 ? n.Normalized() : new Vector3(0, 1, 0);

}

}

}

public static class TextureLoader

{

public static int LoadTexture(string filePath)

{

// Проверка существования файла текстуры

if (!File.Exists(filePath))

{

Console.WriteLine("Texture file not found: " + filePath);

return 0;

}

using var bitmap = new Bitmap(filePath);

// Отражение текстуры по Y (для корректного отображения в OpenGL)

bitmap.RotateFlip(RotateFlipType.RotateNoneFlipY);

// Блокировка битов изображения для чтения данных

var data = bitmap.LockBits(

new Rectangle(0, 0, bitmap.Width, bitmap.Height),

ImageLockMode.ReadOnly,

PixelFormat.Format32bppArgb);

// Создание OpenGL текстуры

int textureID = GL.GenTexture();

GL.BindTexture(TextureTarget.Texture2D, textureID);

// Загрузка данных изображения в текстуру

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,

data.Width, data.Height, 0,

OpenTK.Graphics.OpenGL.PixelFormat.Bgra,

PixelType.UnsignedByte, data.Scan0);

bitmap.UnlockBits(data);

// Настройка параметров текстуры

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS, (int)TextureWrapMode.Repeat);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT, (int)TextureWrapMode.Repeat);

return textureID;

}

}

}

**Задание 4.**

Таблица 7.4.1 – Результат теста 1.

|  |  |
| --- | --- |
| Вид теста | Загрузка программы. |
| Что тестируется | Корректный запуск программы на компьютере. |
| Входные данные | Папка с проектом и ресурсами, готовый к запуску файл программы. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор AMD Ryzen 7 7435HS * оперативная память DDR5 24ГБ, 4800 МГц; * жесткий диск SSD 512ГБ; * видеокарта NVIDIA GeForce RTX 4050 6GB * операционная система Windows 11; * OpenGL API 3.3. |
| Шаги теста | Запустить программу. |
| Ожидаемые выходные данные | Отображение окна, отсутствие ошибок в консоли. |
| Ожидаемый результат | Программа запущена без ошибок, окно отображается корректно. |
| Фактический результат | Работает корректно. |
| Обнаруженный дефект | Дефектов не обнаружено. |

Таблица 7.4.2 – Результат теста 2.

|  |  |
| --- | --- |
| Вид теста | Загрузка корректного OBJ файла. |
| Что тестируется | Корректный отображения программы при вводе корректной 3D-модели. |
| Входные данные | OBJ, MTL, JPG файлы. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор AMD Ryzen 7 7435HS * оперативная память DDR5 24ГБ, 4800 МГц; * жесткий диск SSD 512ГБ; * видеокарта NVIDIA GeForce RTX 4050 6GB * операционная система Windows 11; * OpenGL API 3.3. |

Окночание таблицы 7.4.2.

|  |  |
| --- | --- |
| Шаги теста | Запустить программу и загрузить OBJ файл. |
| Ожидаемые выходные данные | Отображение корректной 3D-модели с текстурой. |
| Ожидаемый результат | Модель отображена без ошибок, текстура лежит корректно, свет работает правильно, управление моделью работает. |
| Фактический результат | Модель загрузилась успешно, однако она отобразилась немного за пределами камеры. |
| Обнаруженный дефект | Модель отображается за камерой. |
| Возможная причина дефекта | 3D-модель разрабатывалась не в центре сцены, не учитывается масштаб модели. |
| Предполагаемое решение | Производить моделирование на нулевых координатах сцены. |

Таблица 7.4.3 – Результат теста 3.

|  |  |
| --- | --- |
| Вид теста | Изменение положения камеры. |
| Что тестируется | Корректность отображения при повороте и масшатабировании. |
| Входные данные | Папка с проектом и ресурсами, готовый к запуску файл программы, obj, mtl, jpg файлы. |
| Настройка тестового окружения | Тест выполняется на компьютере со следующими характеристиками:   * процессор AMD Ryzen 7 7435HS * оперативная память DDR5 24ГБ, 4800 МГц; * жесткий диск SSD 512ГБ; * видеокарта NVIDIA GeForce RTX 4050 6GB * операционная система Windows 11; * OpenGL API 3.3. |
| Шаги теста | Запустить программу, открыть .obj файл, приблизить и перевернуть модель. |
| Ожидаемые выходные данные | Отображение окна, отсутствие ошибок в консоли, адекватное отображение модели. |
| Ожидаемый результат | Программа запущена без ошибок, окно отображается корректно, модель тоже. |

Окончание таблицы 7.4.3.

|  |  |
| --- | --- |
| Фактический результат | Работает корректно. |
| Обнаруженный дефект | Дефектов не обнаружено. |

**Задание 5.**

Ниже на рисунках представлена презентация, расположенная в папке с названием «Презентация».



Рисунок 7.5.1 – Слайд 1



Рисунок 7.5.2 – Слайд 2

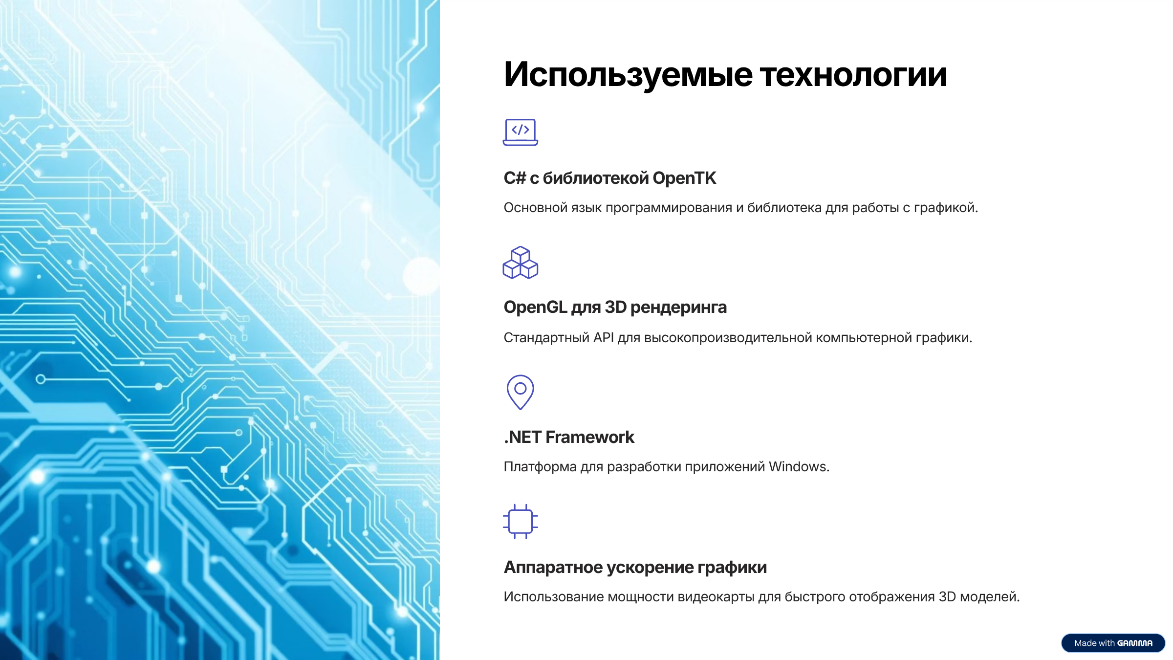


Рисунок 7.5.3 – Слайд 3

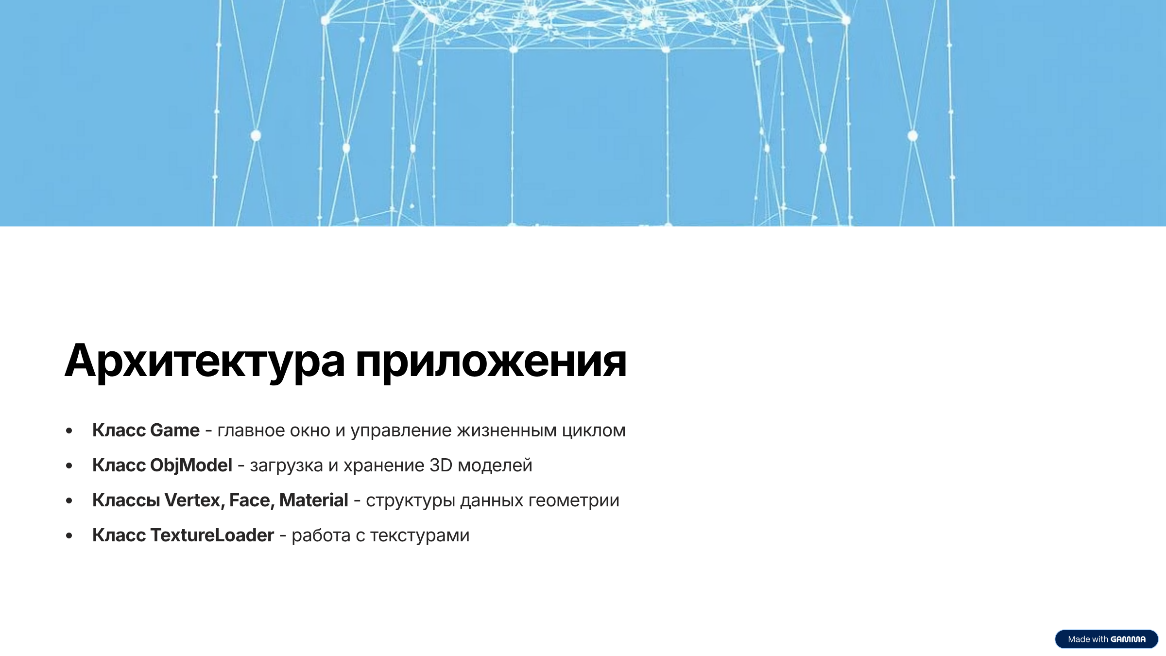


Рисунок 7.5.4 – Слайд 4

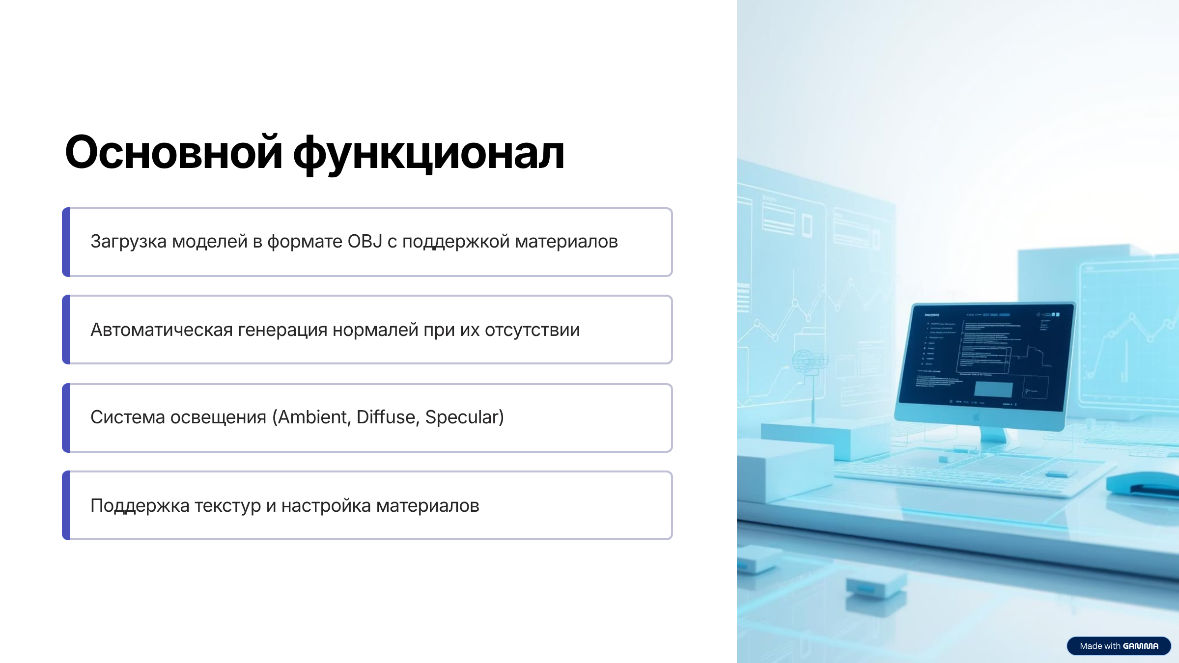


Рисунок 7.5.5 – Слайд 5

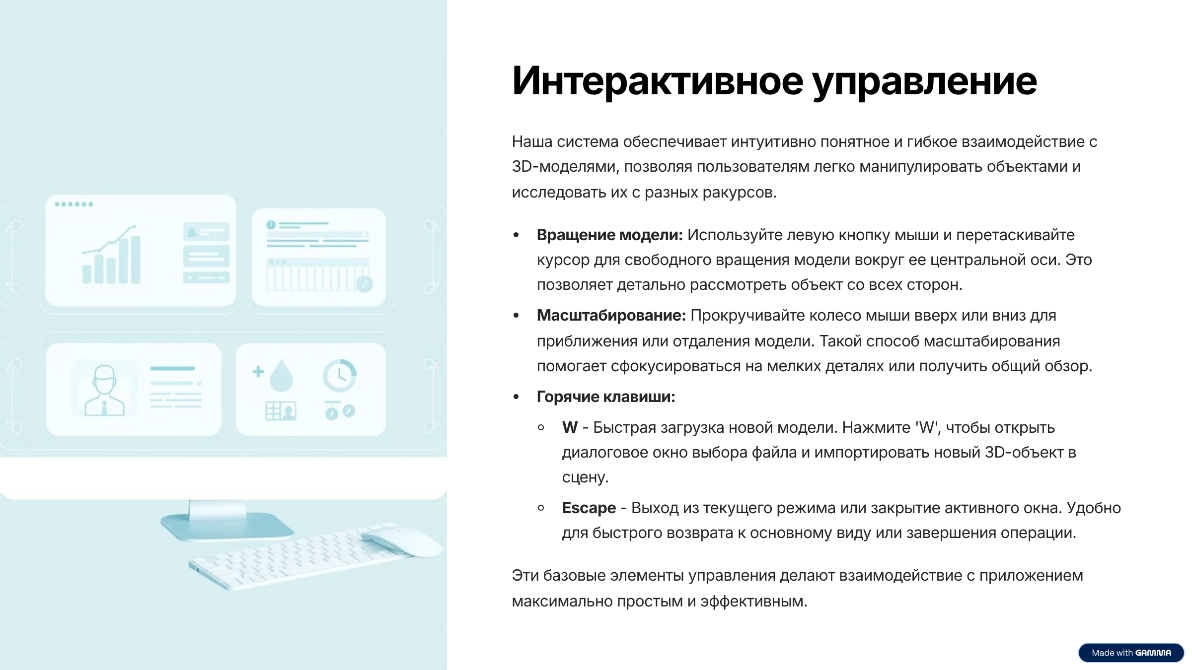


Рисунок 7.5.6 – Слайд 6

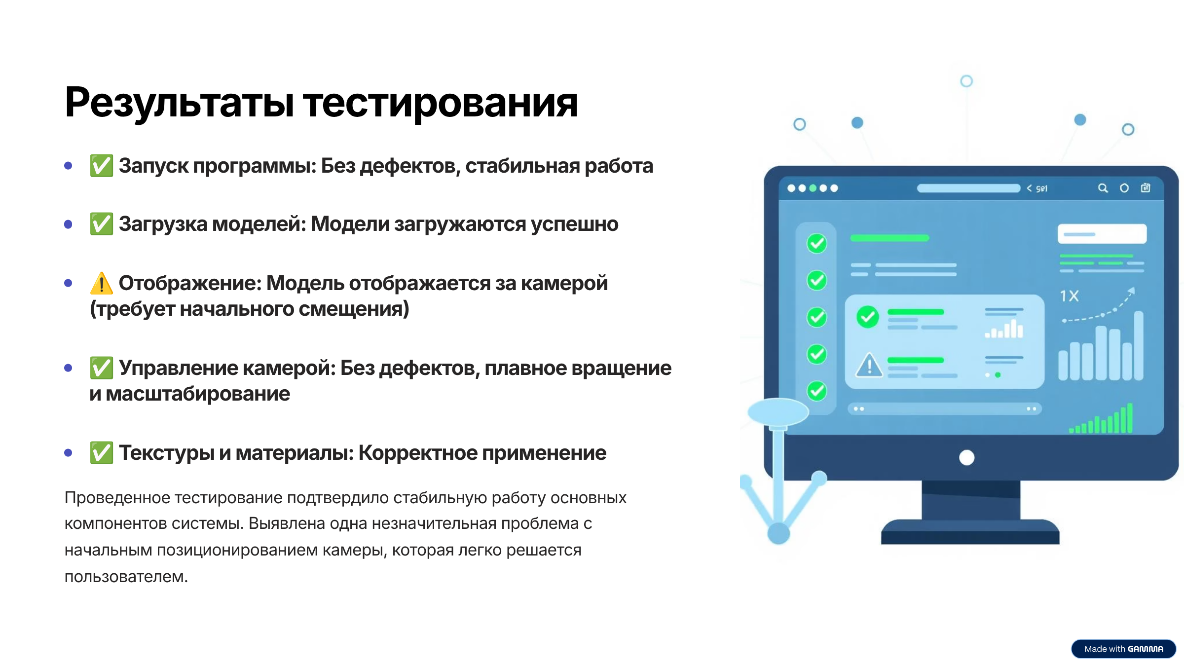


Рисунок 7.5.7 – Слайд 7

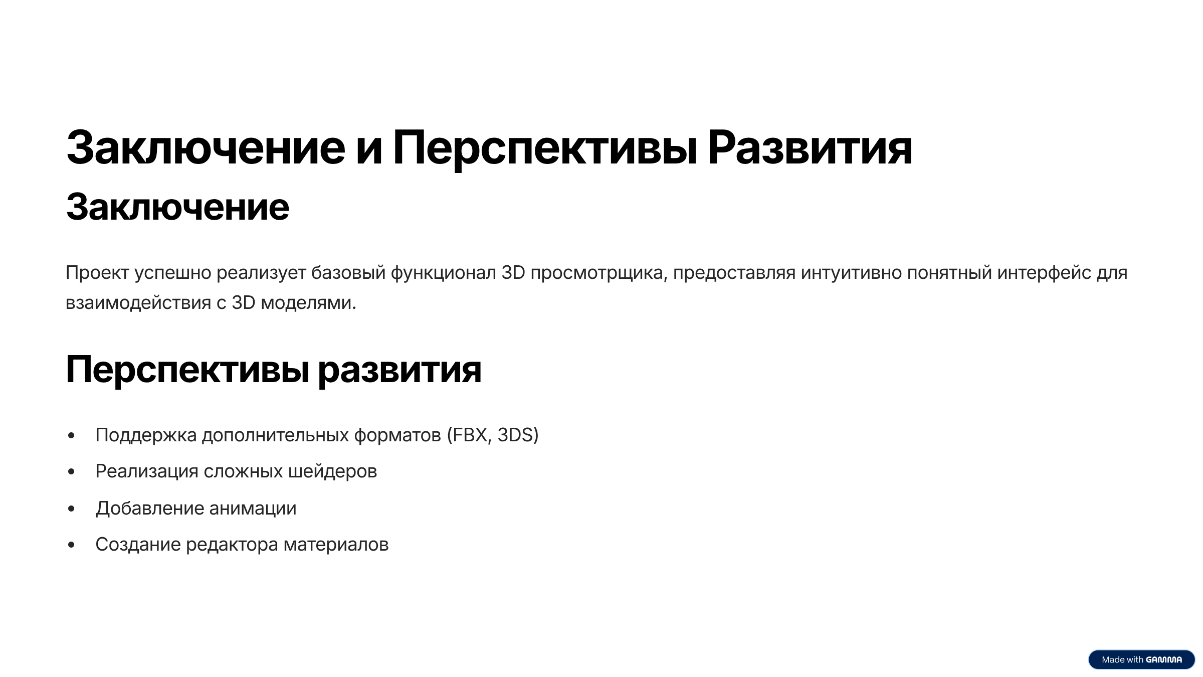


Рисунок 7.5.8 – Слайд 8

# Модуль 8. Реализация проекта

8.1. Постановка задачи

## 8.1.1. Описание предметной области

Предметной областью данного проекта являются программные системы для выполнения математических вычислений, в частности, настольные приложения с графическим интерфейсом пользователя (GUI). Калькулятор, как класс приложений, предназначен для автоматизации рутинных арифметических операций, что позволяет пользователю получать точные результаты быстро и без необходимости ручного подсчета.

В контексте этой работы рассматривается разработка калькулятора, функционирующего как автономное приложение для операционной системы Windows. Его основная сущность – это вычислительный механизм, который обрабатывает числовые данные, введенные пользователем, по заданному алгоритму, соответствующему выбранной арифметической операции. Ключевыми процессами в этой предметной области являются:

1. Ввод данных. Пользователь взаимодействует с элементами управления интерфейса (кнопками, текстовым полем) для передачи чисел и операций программе.
2. Обработка данных. Программа интерпретирует последовательность действий пользователя, формирует математическое выражение и выполняет его вычисление в строгом соответствии с правилами арифметики.
3. Вывод результата. Полученный результат или сообщение об ошибке отображается в элементе управления, предназначенном для этого (например, в текстовом поле).

Важным аспектом предметной области является взаимодействие между пользователем и программой, которое должно быть построено по принципу прямой манипуляции: каждое действие пользователя (нажатие кнопки) немедленно приводит к видимому отклику приложения. Кроме того, предметная область включает в себя обработку особых случаев, таких как деление на ноль, ввод некорректных данных или последовательных операций, что требует реализации механизмов проверки и валидации вводимой информации для обеспечения надежности и предсказуемости работы приложения. Таким образом, разрабатываемый калькулятор представляет собой замкнутую систему, преобразующую пользовательский ввод в конечный числовой результат.

## 8.1.2. Характеристика решаемой задачи

Разрабатываемый калькулятор представляет собой программное решение для выполнения базовых арифметических операций с использованием графического интерфейса. Основная характеристика задачи заключается в необходимости создания устойчивой системы, которая преобразует пользовательский ввод в корректный математический результат.

Ключевой особенностью задачи является требование одновременной реализации нескольких компонентов. Необходимо обеспечить не только математические вычисления, но и интуитивное взаимодействие с пользователем. Это включает обработку последовательности нажатий кнопок, формирование многозначных чисел и корректную интерпретацию математических выражений.

Особое внимание уделяется надежности работы приложения. Система должна корректно обрабатывать исключительные ситуации, такие как деление на ноль или попытки выполнения операций с неполными данными. Для этого предусмотрены механизмы проверки вводимых данных и блокировки некорректных операций.

Архитектура решения основана на объектно-ориентированном подходе, что позволяет четко разделить интерфейс пользователя и вычислительную логику. Такой подход обеспечивает модульность системы и упрощает дальнейшее расширение функциональности.

Важным аспектом является проектирование пользовательского интерфейса, который предоставляет четкую визуальную обратную связь и позволяет легко ориентироваться в возможностях приложения. Интерфейс включает отображение текущей операции, введенных данных и конечного результата вычислений.

## 8.1.3. Разработка моделей решаемой задачи

Для формализации процесса разработки калькулятора была создана модель, описывающая основные компоненты системы и их взаимодействие. Модель включает три ключевых аспекта: математическую логику, пользовательский интерфейс и обработку данных.

Математическая модель основана на объектно-ориентированном подходе, где каждая арифметическая операция представлена в виде отдельного класса, наследуемого от базового абстрактного класса Operation. Это позволяет инкапсулировать логику выполнения каждой операции в отдельном модуле. Модель поддерживает четыре основные операции: сложение, вычитание, умножение и деление. Особое внимание уделено обработке исключительных ситуаций - операция деления включает проверку делителя на ноль с генерацией соответствующего исключения.

Модель управления состоянием описывает жизненный цикл вычислений. В системе отслеживаются следующие состояния:

* ввод первого операнда;
* выбор математической операции;
* выбор второго операнда;
* отображение результата.

Переход между состояниями осуществляется через обработчики событий элементов управления. Модель использует флаг для определения момента завершения предыдущего вычисления и начала нового ввода.

Модель обработки ввода реализует механизм разбора строки выражения на составляющие компоненты. Входная строка разделяется на элементы по пробелам, что позволяет выделить операнды и оператор.

Модель пользовательского интерфейса построена на принципе прямого отображения действий пользователя. Каждое нажатие кнопки немедленно отражается в текстовом поле. Операции управления (очистка, удаление последнего символа) изменяют состояние системы без выполнения вычислений. Визуальная обратная связь обеспечивается через динамическое обновление содержимого текстового поля и обработку ошибочных ситуаций с выводом соответствующих сообщений.

Все модели интегрированы в единую систему, где изменение состояния в одной модели приводит к согласованному обновлению других. Архитектура с использованием словаря operations для связи графических элементов с соответствующими классами операций обеспечивает масштабируемость и простоту добавления новых функций.

8.2. Проектирование программного модуля

## 8.2.1. Сбор исходных материалов

На этапе проектирования программного модуля был проведен сбор и анализ исходных материалов, необходимых для разработки калькулятора. Основное внимание уделялось изучению технических требований и существующих аналогов.

В качестве базовых требований к функциональности были определены:

* выполнение четырех основных арифметических операций (сложение, вычитание, умножение, деление);
* обработка целых и дробных чисел;
* обеспечение последовательного ввода данных;
* визуализация текущего выражения и результата вычислений;
* обработка ошибочных ситуаций, включая деление на ноль;

Для проектирования пользовательского интерфейса были проанализированы стандартные калькулятор операционной системы Windows 11. За основу взята классическая компоновка элементов:

* текстовое поле для отображения вводимых данных и результатов в верхней части формы;
* цифровые кнопки (0-9) и кнопка десятичного разделителя;
* кнопки арифметических операций;
* кнопки управления (равно, очистка, удаление последнего символа);

Технические требования к реализации включали: использование платформы .NET Framework и языка C#; применение Windows Forms для создания графического интерфейса; реализация объектно-ориентированного подхода; обеспечение обработки исключительных ситуаций; поддержка стандартных правил ввода математических выражений.

Особое внимание было уделено проектированию архитектуры приложения. На основе анализа требований принято решение о разделении логики вычислений и пользовательского интерфейса. Математические операции вынесены в отдельные классы, наследуемые от базового абстрактного класса Operation, что соответствует принципам SOLID и обеспечивает возможность расширения функциональности.

ля обработки пользовательского ввода разработана модель последовательного ввода данных, при которой выражение формируется в строковом представлении с последующим парсингом на составляющие элементы перед выполнением вычисления.

## 8.2.2. Проектирование информационной модели

Информационная модель калькулятора построена на объектно-ориентированном подходе и включает два основных компонента: класс формы для управления интерфейсом и иерархию классов для выполнения математических операций.

Основной класс Form1 отвечает за взаимодействие с пользователем и управление состоянием приложения. Он содержит текстовое поле для отображения ввода и результатов, коллекцию операций в виде словаря, связывающего кнопки интерфейса с соответствующими классами операций, а также переменные для хранения текущих операндов, выбранной операции и флага отображения результата.

Иерархия классов операций основана на абстрактном классе Operation, который определяет общий интерфейс для всех математических операций через метод Execute. Конкретные реализации арифметических операций наследуют этот класс и предоставляют собственную логику вычислений. Особое место занимает класс DivOperation, который включает дополнительную проверку на деление на ноль с генерацией исключения.

Модель данных оперирует числовыми значениями типа double для хранения операндов и результатов, а также строковым представлением текущего выражения. Процесс вычисления организован как последовательность этапов: накопление ввода в текстовом поле, разбор выражения на составляющие при нажатии кнопки равенства, выполнение операции через выбранный объект-операцию и обработка возможных исключительных ситуаций.

Такая архитектура обеспечивает четкое разделение ответственности между компонентами. Классы операций инкапсулируют математическую логику, а форма управляет взаимодействием с пользователем и координирует работу всех элементов системы. Это позволяет легко расширять функциональность калькулятора добавлением новых операций без изменения основной логики приложения.

## 8.2.3. Описание входных и выходных данных

Входные данные формируются пользователем через графический интерфейс и включают несколько типов информации. Цифровой ввод состоит из последовательности нажатий кнопок цифр (0-9) и десятичного разделителя, которые формируют строковое представление чисел. Командный ввод осуществляется через кнопки арифметических операцийи функциональные кнопки управления (равно, очистка, удаление символа).

Все входные данные обрабатываются как строковые значения, которые затем преобразуются в числовой формат double при выполнении вычислений. Особенностью входных данных является их последовательный характер - выражения формируются пошагово, с визуальным отображением каждого введенного символа в текстовом поле.

Выходные данные представляют собой результаты работы системы, отображаемые в текстовом поле интерфейса. При успешном выполнении операции выводится числовой результат вычисления в строковом представлении. В случае ошибок (например, деление на ноль) формируется текстовое сообщение, описывающее характер проблемы.

Дополнительным аспектом выходных данных является изменение состояния интерфейса. Система обеспечивает визуальную обратную связь - отображает текущее выражение, выделяет операторы пробелами и своевременно обновляет информацию при выполнении операций очистки или удаления символов.

8.3. Реализация программного модуля

## 8.3.1. Описание диаграммы классов разрабатываемого проекта

Диаграмма классов проекта отражает объектно-ориентированную архитектуру калькулятора и демонстрирует взаимосвязи между основными компонентами системы.

В центре архитектуры находится абстрактный класс Operation, который определяет базовый контракт для всех математических операций. Он содержит абстрактный метод Execute(), предназначенный для выполнения вычислений над двумя операндами типа double.

От базового класса наследуются четыре конкретные реализации:

* AddOperation - реализует операцию сложения;
* SubOperation - реализует операцию вычитания;
* MultOperation - реализует операцию умножения;
* DivOperation - реализует операцию деления с проверкой.

Класс Form1 представляет главную форму приложения и наследует от стандартного класса Form. Он содержит логику пользовательского интерфейса и координацию работы между компонентами. В его обязанности входит:

* обработка событий нажатия кнопок;
* управление текстовым полем для ввода и вывода данных;
* хранение словаря операций для связи кнопок интерфейса с соответствующими классами;
* обработка исключительных ситуаций.

Класс Program является точкой входа в приложение и содержит статический метод Main(), отвечающий за запуск приложения.

Такая архитектура обеспечивает соблюдение принципа открытости/закрытости - система легко расширяется добавлением новых классов операций без изменения существующего кода. Разделение математической логики и интерфейса пользователя повышает сопровождаемость и тестируемость кода.

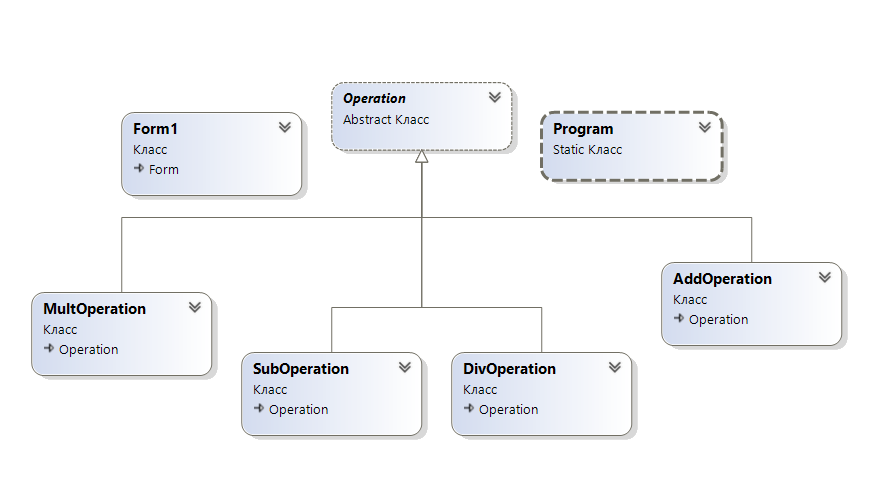


Рисунок 8.3.1 – Диаграмма классов

## 8.3.2 Описание структуры разрабатываемого проекта

Структура проекта калькулятора организована в соответствии с принципами объектно-ориентированного программирования и состоит из нескольких логически связанных компонентов.

Графический интерфейс пользователя реализован в классе Form1, который наследуется от стандартного класса Form. Этот компонент содержит все визуальные элементы: текстовое поле для отображения ввода и результатов, кнопки цифр, арифметических операций и управления. Form1 отвечает за обработку событий мыши и клавиатуры, управление состоянием интерфейса и координацию работы между визуальными элементами и вычислительной логикой.

Вычислительный модуль построен на основе иерархии классов операций. Абстрактный класс Operation определяет общий интерфейс для всех математических операций через метод Execute(). Четыре конкретных класса-наследника (AddOperation, SubOperation, MultOperation, DivOperation) инкапсулируют логику соответствующих арифметических действий. Особенностью класса DivOperation является реализация проверки деления на ноль с генерацией исключения.

Модуль управления данными обеспечивает преобразование между различными форматами представления информации. Он отвечает за конвертацию строкового ввода пользователя в числовые значения типа double, форматирование результатов вычислений для отображения в интерфейсе, а также управление состоянием вычислений через переменные operandA, operandB и selectedOperation.

Координационный центр расположен в классе Form1 и обеспечивает взаимодействие всех модулей. Через словарь operations устанавливается связь между кнопками интерфейса и соответствующими классами операций. Логика обработки последовательности ввода, проверки корректности данных и отображения результатов централизована в обработчиках событий.

Проект имеет четкое разделение ответственности: графический интерфейс занимается только взаимодействием с пользователем, классы операций выполняют математические вычисления, а модуль управления данными обеспечивает корректное преобразование форматов. Такая структура обеспечивает легкость сопровождения и возможности для дальнейшего расширения функциональности.

## 8.3.3. Проектирование и реализация интерфейса программы

Интерфейс программы спроектирован в соответствии с принципами простоты и интуитивной понятности для пользователя. Основное внимание уделено удобству ввода данных и наглядности отображения результатов вычислений.

Основная структура интерфейса включает текстовое поле в верхней части формы, которое отображает текущее математическое выражение и результат вычислений. На рисунке 8.3.1 видно, что выражение отображается полностью с пробелами вокруг оператора для лучшей читаемости.

Блок управления содержит две специализированные кнопки: "Clear" для полной очистки ввода и сброса состояния калькулятора, и "Back" для удаления последнего введенного символа. Эти кнопки расположены в верхней части под текстовым полем.

Цифровой блок организован в виде сетки 4×4. Особенностью является наличие кнопки "00" для быстрого ввода двух нулей, что повышает удобство при работе с большими числами. Кнопка запятой "," используется как десятичный разделитель. Цифры сгруппированы в традиционном порядке: 1-2-3 в первой строке, 4-5-6 во второй, 7-8-9 в третьей, и 0-00 в четвертой.

Блок операций расположен в крайнем правом столбце и содержит основные арифметические действия: деление ("/"), умножение ("\*"), вычитание ("-") и сложение ("+"). Операции распределены по строкам соответственно их приоритету в математических вычислениях.

Визуальное проектирование обеспечивает четкое разделение функциональных зон. Размеры кнопок оптимизированы для комфортного использования, при этом цифровые кнопки имеют квадратную форму, а кнопки операций могут быть немного уже для визуального выделения.

Программная реализация использует единый обработчик для групп кнопок. Особенностью является обработка кнопки "00" как отдельной сущности, которая добавляет два нуля к текущему числу. Кнопка запятой "," программно обеспечивает только один десятичный разделитель в числе.

Интерфейс обеспечивает прямую визуальную обратную связь - пользователь сразу видит введенное выражение в верхнем поле, что позволяет контролировать правильность ввода перед выполнением вычисления.

**Листинг 8.3.1.** Реализация программы «Калькулятор».

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Windows.Forms;

namespace Module\_8

{

public partial class Form1 : Form

{

private double operandA, operandB;

private bool resultShown = false;

private Operation selectedOperation;

private Dictionary<string, Operation> operations = new();

public Form1()

{

InitializeComponent();

// Инициализация операций

operations.Add("btn\_Add", new AddOperation());

operations.Add("btn\_Sub", new SubOperation());

operations.Add("btn\_Mult", new MultOperation());

operations.Add("btn\_Div", new DivOperation());

}

private void btn\_Click(object sender, EventArgs e)

{

Button btn = (Button)sender;

// Если была нажата кнопка результата

if (resultShown)

{

textBox.Text = ""; // Очищается поле вывода

resultShown = false;

}

// Только одну операцию за раз

if (operations.ContainsKey(btn.Name))

{

if (selectedOperation != null)

return; // операция уже выбрана — игнорируем

selectedOperation = operations[btn.Name];

textBox.Text += " " + btn.Text + " ";

}

else

{

textBox.Text += btn.Text;

}

}

private void btn\_Result\_Click(object sender, EventArgs e)

{

// Получение операндов

var parts = textBox.Text.Split(' ', StringSplitOptions.RemoveEmptyEntries);

if (parts.Length < 3 || selectedOperation == null)

return;

if (double.TryParse(parts[0], out operandA) && double.TryParse(parts[2], out operandB))

{

try

{

// Выполнение выбранной операции

double result = selectedOperation.Execute(operandA, operandB);

textBox.Text = result.ToString(CultureInfo.InvariantCulture);

}

catch (DivideByZeroException ex)

{

textBox.Text = "Ошибка: " + ex.Message;

}

}

resultShown = true;

selectedOperation = null;

}

private void btn\_Clear\_Click(object sender, EventArgs e)

{

textBox.Text = "";

operandA = operandB = 0;

selectedOperation = null;

resultShown = false;

}

private void btn\_Back\_Click(object sender, EventArgs e)

{

if (textBox.Text.Length > 0)

textBox.Text = textBox.Text.Substring(0, textBox.Text.Length - 1);

}

}

// Абстрактный класс операций

public abstract class Operation

{

public abstract double Execute(double a, double b);

}

public class AddOperation : Operation

{

public override double Execute(double a, double b) => a + b;

}

public class SubOperation : Operation

{

public override double Execute(double a, double b) => a - b;

}

public class MultOperation : Operation

{

public override double Execute(double a, double b) => a \* b;

}

public class DivOperation : Operation

{

public override double Execute(double a, double b)

{

if (b == 0)

throw new DivideByZeroException("Деление на ноль невозможно");

return a / b;

}

}

}

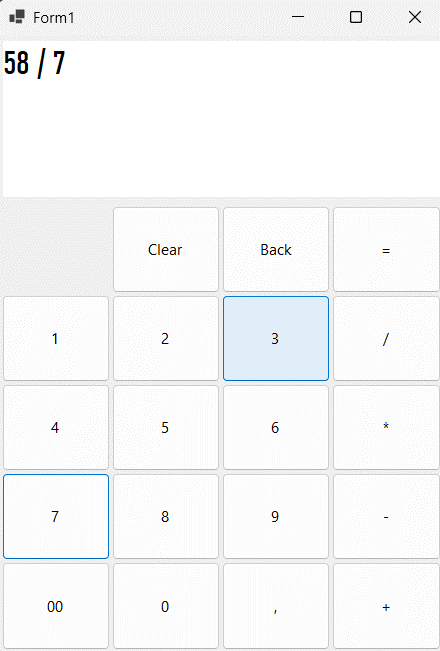


Рисунок 8.3.1 – Ввод примера

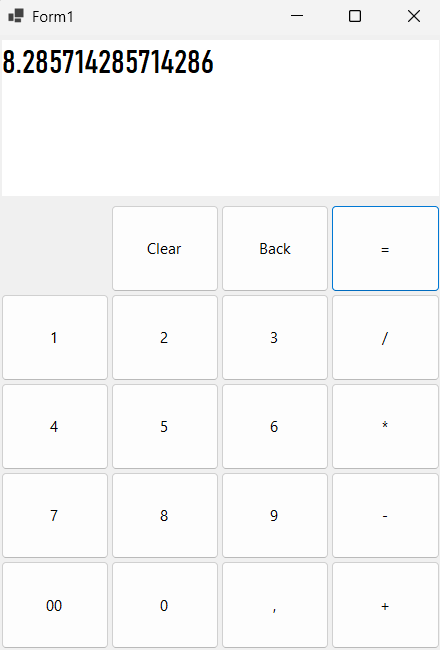


Рисунок 8.3.2 – Результат вычислений

# Модуль 9. Совершенствование навыков ООП

# ЗАКЛЮЧЕНИЕ

# Список использованных источников

Заключение

Список использованных источников

Карпова, И.Б. Основы программирования на языке Object Pascal / И.Б. Карпова. М.: Академия, 2015. 352 с.

Кормен, Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. М.: Вильямс, 2019. 1312 с.

Круг, С. Не заставляйте меня думать / С. Круг. М.: Манн, Иванов и Фербер, 2014. 256 с.

Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин. М.: ДМК Пресс, 2018. 400 с.

Осипов, Д. Delphi. Программирование для Windows, OS X, iOS и Android / Д. Осипов. М.: БХВ-Петербург, 2014. 464 с.

ГОСТ 19.102-77. ЕСПД. Требования к программной документации.

ГОСТ 34.601-90. Автоматизированные системы. Стадии создания.

Boehm, B.W. Экономика программной инженерии / Б.В. Боэм. М.: Мир, 1987. 528 с.

Delphi (software) — Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Delphi\_(%D1%81%D0%BE%D1%84%D1%82%D0%B2%D0%B5%D1%80). Дата доступа: 06.04.2025.

GIF — Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/GIF. Дата доступа: 06.04..2025.

ImageEn component suite for image processing in Delphi [Электронный ресурс]. Режим доступа: https://www.imageen.com/. Дата доступа: 07.04.2025.

Median cut - Wikipedia [Электронный ресурс]. Режим доступа: https://en.wikipedia.org/wiki/Median\_cut. Дата доступа: 10.04.2025.

Р 50.1.028-2001. Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования.

The Median Cut Algorithm for Color Quantization / Gabriel Ytterberg [Электронный ресурс]. Режим доступа: https://medium.com/@gytterberg\_14295/the-median-cut-algorithm-for-color-quantization-cc1128a0c534. Дата доступа: 15.05.2025.

Video Editing Software Market Size, Share & Growth Report by 2033 [Электронный ресурс]. Режим доступа: https://straitsresearch.com/report/video-editing-software-market. Дата доступа: 15.05.20