Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра Информатики
Дисциплина «Программирование»

ОТЧЕТ
к лабораторной работе №7
на тему:
«ПЕРЕГРУЗКА ОПЕРАТОРОВ»
БГУИР 6-05-0612-02 113

Выполнил студент группы 453503
ХАЛАМОВ Николай Андрееевич

(дата, подпись студента)

Проверил ассистент каф. Информатики
РОМАНЮК Максим Валерьевич

(дата, подпись преподавателя)

Минск 2025

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Задание 1. Вариант 4. Класс квадратное уравнение. Поля – int (a, b, c). Добавить метод нахождения корней. Перегрузить +, -, ++, --, * на число, / на число. Сравнить на == и !=. Если уравнение не имеет корней, уравнение = false. Преобразовать в число (a) и назад (ax^2) – в обоих случаях явно.

Для заданной задачи реализовать следующее:

– спроектировать класс согласно варианту индивидуального задания. Для класса использовать отдельный модуль;

– спроектировать конструкторы и свойства с контролем корректности вводимых значений;

– перегрузить метод toString();

– добавить индексирование для получения полей класса

– перегрузить математические операции (имеющие смысл для объектов класса), инкремент и декремент (изменить поля на 1), отношения (==, !=, <, >), true и false, преобразования типа;

– создать несколько объектов класса. Продемонстрировать использование конструкторов и свойств;

– продемонстрировать работу всех методов и операций.

2 ВЫПОЛНЕНИЕ РАБОТЫ

Для выполнения задания был создан класс QuadraticEquation, в котором были добавлены приватные поля для коэффициентов при переменных и свойства для доступа к ним с контролем значений, где коэффициент при x^2 не мог быть равен нулю. Реализованы конструкторы. Был добавлен индексатор для получения коэффициентов при переменных и перегружен метод toString() для вывода уравнения как строки. Мы перегрузили все нужные операции для удобного использования их с обьектами класса.

```
namespace Overloading
{
    internal class QuadraticEquation
    {

        private int a, b, c;

        public QuadraticEquation(int a, int b, int c)
        {
            this.a = a;
            this.b = b;
            this.c = c;
        }

        public QuadraticEquation()
        {
```

```
            a = 1;
            b = -2;
            c = 1;
        }
        public int A
        {
            get => a;
            set
            {
                if (value == 0)
                {
                    throw new ArgumentException("Коэффициент при x^2 не может
быть равен нулю, так как тогда уравнение не будет квадратным");
                }
                a = value;
            }
        }

        public int B
        {
            get => b;
            set => b = value;
        }

        public int C
        {
            get => c;
            set => c = value;
        }

        public List<Double> FindRoots()
        {
            List<double> roots = new List<double>();

            double discriminant = b * b - 4 * a * c;

            if (discriminant < 0)
            {
                return roots;
            }
            else if (discriminant == 0)
            {
                roots.Add(-b / (2.0 * a));
                return roots;
            }
            else
            {
                double sqrtDiscriminant = Math.Sqrt(discriminant);
                roots.Add((-b - sqrtDiscriminant) / (2.0 * a));
                roots.Add((-b + sqrtDiscriminant) / (2.0 * a));
                return roots;
            }
        }
            public override string ToString()
        {

            string partA ="",partB="", partC="";
            if (A==0)
            {
                partA = "";
            }
            else if (A==1)
            {
```

```
                partA = "x^2";
            }
            else if (A == -1)
            {
                partA = "-x^2";
            }
            else
            {
                partA = $"{A}x^2";
            }

            if (B == 1)
            {
                partB = $" + x";
            }
            else if (B == -1)
            {
                partB = $" - x";
            }
            else if (B>0)
            {
                 partB = $" + {B}x";
            }
            else if (B<0)
            {
                 partB = $" - {-B}x";
            }
            if (C > 0)
            {
                 partC = $" + {C}";
            }
            else if (C<0)
            {
                 partC = $" - {-C}";
            }
            return $"{partA}{partB}{partC} = 0";
        }

        public int this[int index]
        {
            get
            {
                switch (index)
                {
                    case 0: return A;
                    case 1: return B;
                    case 2: return C;
                    default: throw new IndexOutOfRangeException("Индекс должен
быть 0 (A), 1 (B) или 2 (C)");
                }
            }
            set
            {
                switch (index)
                {
                    case 0: A = value; break;
                        case 1: B = value; break;
                        case 2: C = value; break;
                        default: throw new IndexOutOfRangeException("Индекс
должен быть 0 (A), 1 (B) или 2 (C)");
                }
            }
        }
```

```
        public static QuadraticEquation operator +(QuadraticEquation eq1,
QuadraticEquation eq2)
        {
            return new QuadraticEquation(eq1.A + eq2.A, eq1.B + eq2.B, eq1.C +
eq2.C);
        }

        public static QuadraticEquation operator -(QuadraticEquation eq1,
QuadraticEquation eq2)
        {
            return new QuadraticEquation(eq1.A-eq2.A,eq1.B-eq2.B, eq1.C-eq2.C);
        }

        public static QuadraticEquation operator *(QuadraticEquation eq,int
number)
        {
            return new QuadraticEquation(eq.A*number,eq.B*number, eq.C*number);
        }

        public static QuadraticEquation operator /(QuadraticEquation eq, int
number)
        {
            if (number == 0)
                throw new DivideByZeroException("Деление на ноль невозможно");
            return new QuadraticEquation(eq.A / number, eq.B / number, eq.C /
number);
        }

        public static QuadraticEquation operator ++(QuadraticEquation eq)
        {
            return new QuadraticEquation(eq.A + 1, eq.B + 1, eq.C + 1);
        }

        public static QuadraticEquation operator --(QuadraticEquation eq)
        {
            return new QuadraticEquation(eq.A - 1, eq.B - 1, eq.C - 1);
        }

        public static bool operator ==(QuadraticEquation eq1, QuadraticEquation
eq2)
        {
            return eq1.A==eq2.A && eq1.B==eq2.B && eq1.C==eq2.C;
        }

        public static bool operator !=(QuadraticEquation eq1, QuadraticEquation
eq2)
        {
            return !(eq1==eq2);
        }
        public static bool operator true(QuadraticEquation eq)
        {
            return eq.FindRoots().Count>0;
        }

        public static bool operator false(QuadraticEquation eq)
        {
            return eq.FindRoots().Count == 0;
        }

        public static explicit operator int (QuadraticEquation eq)
        {
            return eq.A;
```

```
        }
        public static explicit operator QuadraticEquation(int a)
        {
            return new QuadraticEquation(a, 0, 0);
        }

        public override bool Equals(object? obj)
        {
            return obj is QuadraticEquation other
                && A == other.A
                && B == other.B
                && C == other.C;
        }

        public override int GetHashCode()
        {
            return HashCode.Combine(A, B, C);
        }
    }
}
```

Работу всех реализованных в классе QuadraticEquation методов и операцией продемонстрируем в классе Program, создав 3 обьекта класса и проводя над ними изменения и выводя результаты в консоль.

```
using System;

namespace Overloading
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("=== Демонстрация работы класса QuadraticEquation
===");

            // 1. Создание объектов (демонстрация конструкторов)
            Console.WriteLine("\n1. Создание объектов:");
            var eq1 = new QuadraticEquation(1, -5, 6); // Конструктор с
параметрами
            var eq2 = new QuadraticEquation();         // Конструктор по
умолчанию (x^2 -2x +1 =0)
            var eq3 = (QuadraticEquation)3;            // Явное преобразование
(3x^2 =0)

            Console.WriteLine($"eq1: {eq1}");
            Console.WriteLine($"eq2: {eq2}");
            Console.WriteLine($"eq3: {eq3}");

            // 2. Демонстрация свойств
            Console.WriteLine("\n2. Работа свойств:");
            eq1.B = -3; // Изменение коэффициента b
            try
            {
                eq1.A = 0; // Попытка задать a=0 (выбросит исключение)
            }
            catch (ArgumentException ex)
            {
                Console.WriteLine($"Ошибка: {ex.Message}");
            }
            Console.WriteLine($"Изменённое eq1: {eq1}");
```

```
            // 3. Демонстрация индексатора
            Console.WriteLine("\n3. Работа индексатора:");
            Console.WriteLine($"eq1[0] (коэф. a): {eq1[0]}");
            Console.WriteLine($"eq1[1] (коэф. b): {eq1[1]}");
            Console.WriteLine($"eq1[2] (коэф. c): {eq1[2]}");
            eq1[2] = 10; // Изменение коэффициента с через индексатор
            Console.WriteLine($"Изменённое eq1: {eq1}");

            // 4. Демонстрация методов
            Console.WriteLine("\n4. Работа методов:");
            Console.WriteLine($"Корни eq1: [{string.Join(", ",
eq1.FindRoots())}]");
            Console.WriteLine($"Корни eq2: [{string.Join(", ",
eq2.FindRoots())}]");
            Console.WriteLine($"Корни eq3: [{string.Join(", ",
eq3.FindRoots())}]");

            // 5. Демонстрация операторов
            Console.WriteLine("\n5. Работа операторов:");
            // Математические операторы
            Console.WriteLine($"eq1 + eq2: {eq1 + eq2}");
            Console.WriteLine($"eq1 - eq2: {eq1 - eq2}");
            Console.WriteLine($"eq1 * 2: {eq1 * 2}");
            Console.WriteLine($"eq1 / 2: {eq1 / 2}");
            // Инкремент и декремент
            Console.WriteLine($"++eq1: {++eq1}");
            Console.WriteLine($"--eq1: {--eq1}");
            // Операторы сравнения
            Console.WriteLine($"eq1 == eq2: {eq1 == eq2}");
            Console.WriteLine($"eq1 != eq2: {eq1 != eq2}");
            // true/false
            Console.WriteLine($"if(eq1): {(eq1 ? "Есть корни" : "Нет
корней")}");
            Console.WriteLine($"if(eq2): {(eq2 ? "Есть корни" : "Нет
корней")}");
            // Преобразование типов
            Console.WriteLine($"(int)eq1: {(int)eq1}");
            Console.WriteLine($"(int)eq2: {(int)eq2}");
            Console.WriteLine($"(int)eq3: {(int)eq3}");
        }
    }
}
```

Результат работы программы продемонстрирован ниже (см. рисунок 1).

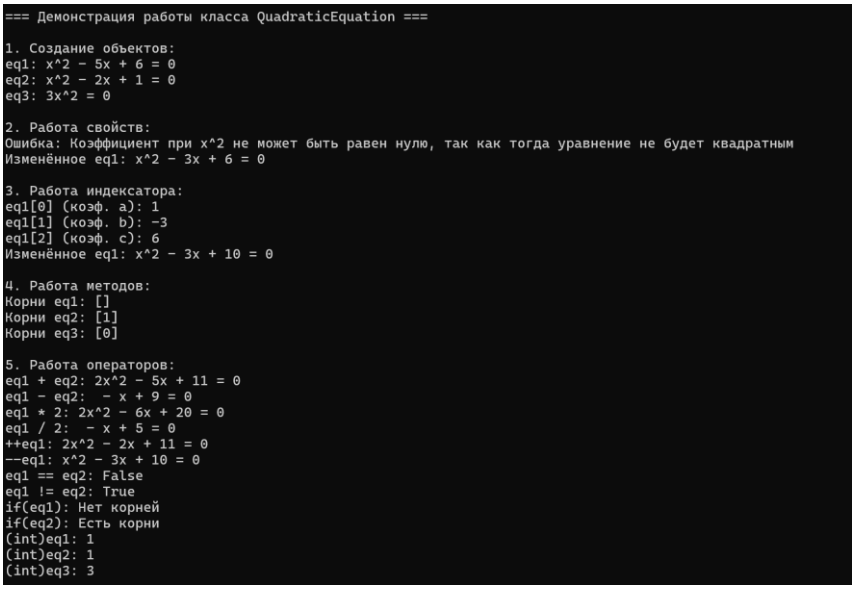

Рисунок 1 – Результат работы программы

ВЫВОД

В ходе лабораторной работы были изучены принципы перегрузки операторов и методов, а также работа с индексаторами.