|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет «Радиотехнический»**

**Кафедра «Системы обработки информации и управления»**

Лабораторная работа № 1

по дисциплине «Базовые компоненты интернет технологий»

Выполнил:

студент(ка) группы № РТ5-31Б

Пакало Александр Сергеевич

подпись, дата

Проверил:

преподаватель каф. ИУ5  
Гапанюк Юрий Евгеньевич

подпись, дата

2020 г.

Оглавление

[Описание задания. 3](#_Toc59478314)

[Диаграмма классов. 3](#_Toc59478315)

[Текст программы. 5](#_Toc59478316)

[Lab\_1 5](#_Toc59478317)

[BiquadraticEquations.cs 5](#_Toc59478318)

[RootsResult.cs 6](#_Toc59478319)

[Lab\_1.Roots 7](#_Toc59478320)

[NoRoots.cs 7](#_Toc59478321)

[OneRoot.cs 7](#_Toc59478322)

[TwoRoots.cs 7](#_Toc59478323)

[ThreeRoots.cs 7](#_Toc59478324)

[FourRoots.cs 8](#_Toc59478325)

[Lab\_1.EquationSolvers 8](#_Toc59478326)

[EquationSolver.cs 8](#_Toc59478327)

[QuadraticEquationSolver.cs 9](#_Toc59478328)

[BiquadraticEquationSolver.cs 10](#_Toc59478329)

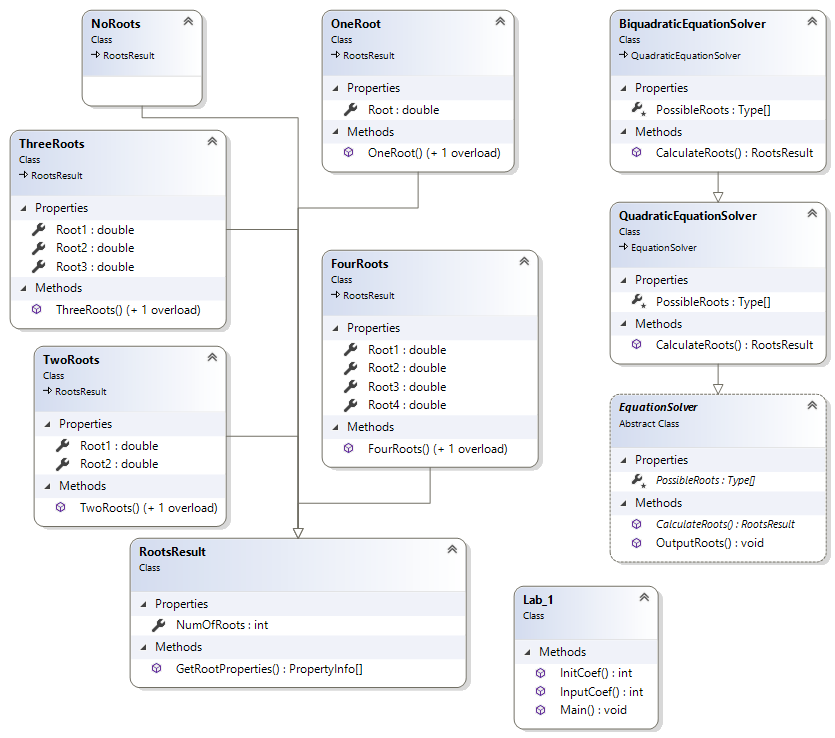
[Анализ результатов. 11](#_Toc59478330)

# Описание задания.

Разработать программу для решения биквадратного уравнения – <https://ru.wikipedia.org/wiki/%D0%A3%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%87%D0%B5%D1%82%D0%B2%D1%91%D1%80%D1%82%D0%BE%D0%B9_%D1%81%D1%82%D0%B5%D0%BF%D0%B5%D0%BD%D0%B8#%D0%91%D0%B8%D0%BA%D0%B2%D0%B0%D0%B4%D1%80%D0%B0%D1%82%D0%BD%D0%BE%D0%B5_%D1%83%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5>

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Программа осуществляет ввод с клавиатуры коэффициентов А, В, С, вычисляет дискриминант и корни уравнения (в зависимости от дискриминанта).
3. Если коэффициент А, В, С введен некорректно, то необходимо проигнорировать некорректное значение и ввести коэффициент повторно.
4. Первой строкой программа выводит ФИО разработчика и номер группы.
5. Корни уравнения выводятся зеленым цветом. Если корней нет, то сообщение выводится красным цветом.
6. ДОПОЛНИТЕЛЬНОЕ ТРЕБОВАНИЕ. Коэффициенты А, В, С задаются в виде параметров командной строки. Если они не указаны, то вводятся с клавиатуры в соответствии с пунктом 2. Проверка из пункта 3 в этом случае производится для параметров командной строки без повторного ввода с клавиатуры.

# Диаграмма классов.



False

# Текст программы.

## Lab\_1

### BiquadraticEquations.cs

using System;

using System.Collections.Generic;

using System.Reflection;

using Lab\_1;

using Lab\_1.Roots;

using Lab\_1.EquationSolvers;

namespace Equations

{

class Lab\_1

{

static public int InitCoef(string defaultArg, string coefName)

{

try

{

return Convert.ToInt32(defaultArg);

}

catch (System.IndexOutOfRangeException) // В параметрах консоли нет такого элемента.

{

return InputCoef(coefName);

}

catch (System.FormatException) // Элемент был неправильно введён в параметрах консоли.

{

return InputCoef(coefName);

}

}

static public int InputCoef(string coefName)

{

try

{

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

return Convert.ToInt32(Console.ReadLine()); // Если исключений нет, выходим из рекурсивного цикла.

}

catch (System.FormatException e)

{

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine(e.Message);

Console.ForegroundColor = ConsoleColor.Black;

Console.WriteLine("Введите коэффициент ещё раз.");

Console.ResetColor();

return InputCoef(coefName); // Рекурсивно вызываем функцию, пока пользователь не введёт правильно коэффициент.

}

}

static public void Main(string[] args)

{

Console.WriteLine("Автор программы:\nПакало Александр Сергеевич РТ5-31Б");

int a, b, c; // Коэффициенты уравнения.

Console.WriteLine("Введите коэффициенты уравнений вида ax^2 + bx + c, ax^4 + bx^2 + c.");

a = InitCoef(args[0], "a");

b = InitCoef(args[1], "b");

c = InitCoef(args[2], "c");

Console.WriteLine("Корни квадратного уравнения:");

QuadraticEquationSolver quadEq = new QuadraticEquationSolver();

RootsResult quadRoots = quadEq.CalculateRoots(a, b, c);

quadEq.OutputRoots(quadRoots);

Console.WriteLine("Корни биквадратного уравнения:");

BiquadraticEquationSolver biquadEq = new BiquadraticEquationSolver();

RootsResult biquadRoots = biquadEq.CalculateRoots(a, b, c);

biquadEq.OutputRoots(biquadRoots);

}

}

}

### RootsResult.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Reflection;

using System.Text.RegularExpressions;

namespace Lab\_1

{

class RootsResult

{

public int NumOfRoots

{

get

{

Type resultType = this.GetType(); // Тип класса, в котором вызывается геттер.

PropertyInfo[] rootProperties = resultType.GetProperties();

return rootProperties.Length;

}

}

/// <summary>

/// Метод отбирает из свойств данного класса свойства, обеспечивающих доступ к корням уравнения.

/// </summary>

/// <returns>Массив со свойствами, обеспечивающими доступ к корням уравнения.</returns>

public PropertyInfo[] GetRootProperties()

{

Type T = this.GetType(); // Тип класса, в котором вызывается метод GetRootProperties.

PropertyInfo[] properties = T.GetProperties(); // Массив всех свойств данного класса.

string pattern = @"^Root\d\*$";

Regex rgx = new Regex(pattern);

IEnumerable<PropertyInfo> rootProperties = properties.Where(property => rgx.IsMatch(property.Name)); // Отфильтровали массив, получив необходимый набор свойств.

return rootProperties.ToArray();

}

}

}

## Lab\_1.Roots

### NoRoots.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab\_1.Roots

{

class NoRoots : RootsResult { }

}

### OneRoot.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab\_1.Roots

{

class OneRoot : RootsResult

{

public double Root { get; private set; }

public OneRoot(double root)

{

Root = root;

}

public OneRoot(List<double> roots) : this(roots[0])

{

}

}

}

### TwoRoots.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab\_1.Roots

{

class TwoRoots : RootsResult

{

public double Root1 { get; private set; }

public double Root2 { get; private set; }

public TwoRoots(double root1, double root2)

{

Root1 = root1;

Root2 = root2;

}

public TwoRoots(List<double> roots) : this(roots[0], roots[1])

{

}

}

}

### ThreeRoots.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab\_1.Roots

{

class ThreeRoots : RootsResult

{

public double Root1 { get; private set; }

public double Root2 { get; private set; }

public double Root3 { get; private set; }

public ThreeRoots(double root1, double root2, double root3)

{

Root1 = root1;

Root2 = root2;

Root3 = root3;

}

public ThreeRoots(List<double> roots) : this(roots[0], roots[1], roots[2])

{

}

}

}

### FourRoots.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab\_1.Roots

{

class FourRoots : RootsResult

{

public double Root1 { get; private set; }

public double Root2 { get; private set; }

public double Root3 { get; private set; }

public double Root4 { get; private set; }

public FourRoots(double root1, double root2, double root3, double root4)

{

Root1 = root1;

Root2 = root2;

Root3 = root3;

Root4 = root4;

}

public FourRoots(List<double> roots) : this(roots[0], roots[1], roots[2], roots[3])

{

}

}

}

## Lab\_1.EquationSolvers

### EquationSolver.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Reflection;

namespace Lab\_1.EquationSolvers

{

abstract class EquationSolver

{

abstract protected Type[] PossibleRoots { get; }

abstract public RootsResult CalculateRoots(int a, int b, int c);

public void OutputRoots(RootsResult result)

{

Type resultType = result.GetType();

if (resultType.Name.ToString() == "NoRoots")

{

Console.ForegroundColor = ConsoleColor.DarkRed;

Console.WriteLine("Корней нет!");

Console.ResetColor();

return;

}

PropertyInfo[] rootProperties = result.GetRootProperties();

Console.ForegroundColor = ConsoleColor.Green;

foreach (PropertyInfo rootProperty in rootProperties)

{

var root = rootProperty.GetValue(result, null);

Console.WriteLine(root);

}

Console.ResetColor();

}

}

}

### QuadraticEquationSolver.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Lab\_1.Roots;

namespace Lab\_1.EquationSolvers

{

class QuadraticEquationSolver : EquationSolver

{

override protected Type[] PossibleRoots

{

get

{

Type[] possibleRoots =

{

typeof(NoRoots),

typeof(OneRoot),

typeof(TwoRoots)

};

return possibleRoots;

}

}

override public RootsResult CalculateRoots(int a, int b, int c)

{

RootsResult result;

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

if (discriminant < 0)

{

result = new NoRoots();

}

else if (discriminant == 0)

{

result = new OneRoot(-b / 2 \* a);

}

else

{

result = new TwoRoots(

(-b - Math.Sqrt(discriminant)) / 2 \* a,

(-b + Math.Sqrt(discriminant)) / 2 \* a

);

}

return result;

}

}

}

### BiquadraticEquationSolver.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Reflection;

using Lab\_1.Roots;

namespace Lab\_1.EquationSolvers

{

class BiquadraticEquationSolver : QuadraticEquationSolver

{

override protected Type[] PossibleRoots

{

get

{

Type[] possibleRoots =

{

typeof(NoRoots),

typeof(OneRoot),

typeof(TwoRoots),

typeof(ThreeRoots),

typeof(FourRoots)

};

return possibleRoots;

}

}

override public RootsResult CalculateRoots(int a, int b, int c)

{

// Подсчитываем корни уравнения, как если бы оно было квадратным.

RootsResult quadResult = base.CalculateRoots(a, b, c);

PropertyInfo[] rootProperties = quadResult.GetRootProperties(); // Список свойств, обеспечивающих доступ к корням уравнения.

// Перебираем корни.

List<double> roots = new List<double>(); // Список, необходимый для хранения новых корней.

foreach (PropertyInfo rootProperty in rootProperties)

{

if (rootProperty.Name.ToString() == "NumOfRoots") continue; // Количество корней нам сейчас не нужно обрабатывать.

double root = (double)rootProperty.GetValue(quadResult, null); // Получаем значение текущего свойства.

if (root < 0) continue;

if (root == 0)

{

roots.Add(0);

continue;

}

roots.Add(Math.Sqrt(root));

roots.Add(-Math.Sqrt(root));

}

if (roots.Count == 0)

{

return new NoRoots();

}

RootsResult result = (RootsResult)Activator.CreateInstance(PossibleRoots[roots.Count], roots);

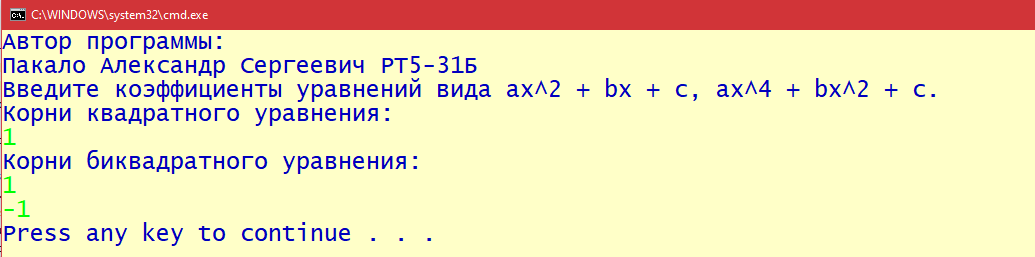
return result;

}

}

}

# Анализ результатов.



Скриншот 1 (демонстрация работы программы).