Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра Информатики

Дисциплина «Программирование»

**ОТЧЕТ**

к лабораторной работе №7

на тему:

**«Наследование»**

БГУИР 6-05-0612-02 34

|  |
| --- |
| Выполнил студент группы 353505  Хорошко Кирилл Николаевич |
|  |
| (дата, подпись студента) |
| Проверил ассистент каф. Информатики  РОМАНЮК Максим Валерьевич |
|  |
| (дата, подпись преподавателя) |

Минск 2024

# 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 !=. (d) если уравнение не имеет корней, уравнение = false. Преобразовать в число (a) и назад (ax2 ) – в обоих случаях явно.

# 2 Выполнение работы

Для выполнения данной работы был разработан класс quadric, который содержал поля a, b, c коэффициенты квадратного уравнения. Был реализован метод решения квадратных уравнений. Были перегружены операторы, можно слаживать уравнения, отнимать, сравнивать, слаживать с числом, делить, умножать и остальные операции с числом.

# Вывод

В ходе лабораторной работы мною было изучено наследование в языке C#. Мною были углублены знания по работе с классами в объектно-ориентированным языком программирования C#. Были рассмотрены обращения к классу-родителю из класса-наследник для вызова методов класса-родителя. Была изучена схема создания UML-диаграмм, которые позволяют отображать отношения между классами, что позволяет более простым способом их анализировать. Было установлено, что при создании объекта класса-наследника вызывается конструктор класса-родителя. Конструктор класса-родителя, который будет вызываться можно установить с помощью ключевого слова “base”.

public class quadric

{

public double a;

public double b;

public double c;

public quadric(double a, double b, double c)

{

this.a = a;

this.b = b;

this.c = c;

solve();

}

private double x1;

private double x2;

bool solved = false;

private void solve()

{

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

if (D < 0)

{

solved = false;

return;

}

if (D == 0)

{

solved = true;

x1 = -b / (2 \* a);

x2 = x1;

return;

}

if (D > 0)

{

solved = true;

D = Math.Sqrt(D);

x1 = (-b - D) / 2 / a;

x2 = (-b + D) / 2 / a;

return;

}

}

public double X1

{

get

{

if (solved)

{

return x1;

}

else

{

return -9999999999999;

}

}

}

public double X2

{

get

{

if (solved)

{

return x2;

}

else

{

return -9999999999999;

}

}

}

public static quadric operator + (quadric a, quadric b)

{

return new quadric(a.a + b.a, a.b + b.b, a.c + b.c);

}

public static quadric operator -(quadric a, quadric b)

{

return new quadric(a.a - b.a, a.b - b.b, a.c - b.c);

}

public static bool operator ==(quadric a, quadric b)

{

if (a.a == b.a && a.b == b.b && a.c == b.c)

{

return true;

}

else return false;

}

public static bool operator !=(quadric a, quadric b)

{

if (a.a == b.a && a.b == b.b && a.c == b.c)

{

return false;

}

else return true;

}

public static quadric operator +(quadric a, int val)

{

return new quadric(a.a, a.b, a.c + val);

}

public static quadric operator -(quadric a, int val)

{

return new quadric(a.a, a.b, a.c - val);

}

public static quadric operator \*(quadric a, int val)

{

return new quadric(a.a \* val, a.b \* val, a.c \* val) ;

}

public static quadric operator /(quadric a, int val)

{

return new quadric(a.a / val, a.b / val, a.c / val);

}

public static quadric operator ++(quadric a)

{

return new quadric(a.a, a.b, a.c + 1);

}

public static quadric operator --(quadric a)

{

return new quadric(a.a, a.b, a.c - 1);

}

public string print()

{

return "a = " + a + " b = " + b + " c = " + c;

}

}

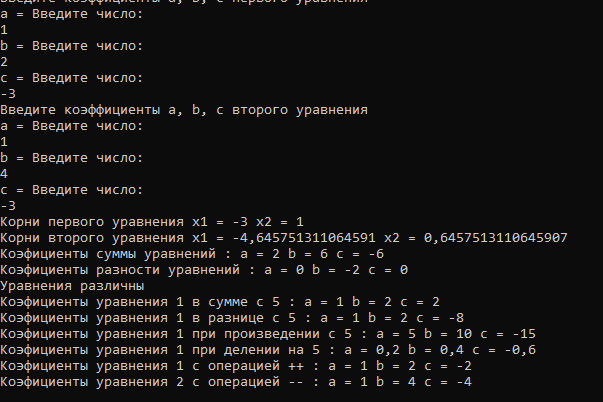


Рисунок 1 — Результат работы программы

# Вывод

В ходе лабораторной работы мною была изучена одна из парадигм ООП, а именно полиморфизм, суть которого заключается в возможности переопределения существующей реализации методов как внутри класса, так и в классах наследниках. Было укреплено понимание свойств и авто свойств. Было на практике отработано переопределение методов и операторов. Была отработана генерация собственных исключений, которая помогает дополнять логику проверки данных получаемых и обрабатываемых программой. Так же была изучена схема создания библиотек классов, которая позволяет подключать ранее разработанные классы к проекту.