Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 4 з дисципліни
«Основи програмування 2. Модульне програмування»

«Наслідування та поліморфізм»

Варіант 35

Виконав студент ІП-15, Шабанов Метін Шаміль огли
(шифр, прізвище, ім'я, по батькові)

Перевірив Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота 4

Наслідування та поліморфізм
Варіант 35 (1)

Завдання:

1. Створити клас TSystemLinearEquation, який представляє систему лінійних алгебраїчних рівнянь і містить методи для знаходження коренів рівнянь та перевірки того, чи є деякий набір чисел розв'язком системи рівнянь. На основі цього класу створити класи-нащадки, які представляють системи двох та трьох лінійних рівнянь (відповідно з двома та трьома невідомими). Випадковим чином згенерувавши дані для декількох систем двох лінійних рівнянь та декількох систем трьох лінійних рівнянь. Знайти розв'язок даних систем лінійних алгебраїчних рівнянь (обох видів).

Виконання:

C#:

C# Main.cs | C# RandomSystemGenerator.cs | C# TSystemLinearEquation.cs | C# TwoSystemLinearEquation.cs | C# ThreeSystemLinearEquation.cs | C#

```
namespace Lab5_InheritanceAndPolymorphism
{
    class Program
    {
        static void Main(string[] args)
        {
            double[,] firstMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 2);
            TwoSystemLinearEquation firstSystem = new TwoSystemLinearEquation(firstMatrix);
            Printer.PrintSystem(firstSystem);
            double[] firstRoots = firstSystem.MatrixMethod();
            Printer.PrintRoots(firstRoots);

            double[,] secondMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 2);
            TwoSystemLinearEquation secondSystem = new TwoSystemLinearEquation(secondMatrix);
            Printer.PrintSystem(secondSystem);
            double[] secondRoots = secondSystem.KramerMethod();
            Printer.PrintRoots(secondRoots);

            double[,] thirdMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 3);
            ThreeSystemLinearEquation thirdSystem = new ThreeSystemLinearEquation(thirdMatrix);
            Printer.PrintSystem(thirdSystem);
            double[] thirdRoots = thirdSystem.MatrixMethod();
            Printer.PrintRoots(thirdRoots);

            double[,] fourthMatrix = RandomSystemGenerator.GenerateRandomSystem( variablesQuantity: 3);
            ThreeSystemLinearEquation fourthSystem = new ThreeSystemLinearEquation(fourthMatrix);
            Printer.PrintSystem(fourthSystem);
            double[] fourthRoots = fourthSystem.KramerMethod();
            Printer.PrintRoots(fourthRoots);
        }
    }
}
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#

```
using System;

namespace Lab5_InheritanceAndPolymorphism
{
    4 usages
    public class RandomSystemGenerator
    {
        4 usages
        public static double[,] GenerateRandomSystem(int variablesQuantity)
        {
            Random rand = new Random();
            double[,] system = new double[variablesQuantity, variablesQuantity + 1];

            for (int i = 0; i < variablesQuantity; i++)
                for (int j = 0; j < variablesQuantity + 1; j++)
                    system[i, j] = Math.Round(rand.NextDouble() * 9);

            return system;
        }
    }
}
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# Matrix.cs × C# Printer.cs

```
namespace Lab5_InheritanceAndPolymorphism
{
    3 usages  2 inheritors
    public abstract class TSystemLinearEquation
    {
        protected double[,] _coefficients;

        2 usages
        public TSystemLinearEquation(double[,] coefficients)
        {
            _coefficients = coefficients;
        }

        2 usages
        public double[] MatrixMethod()
        {
            Matrix A = new Matrix(_initMatrixA());
            Matrix B = new Matrix(_initMatrixB());

            if (A.MakeInverseMatrix() != null)
            {
                Matrix result = new Matrix(A.MakeInverseMatrix()) * B;
                return _convertToRootsView(result.Matrix1);
            }

            return null;
        }

```

2 usages

```
26          public double[] KramerMethod()
27          {
28              Matrix A = new Matrix(_initMatrixA());
29              Matrix B = new Matrix(_initMatrixB());
30              int size = A.Matrix1.GetLength( dimension: 0);
31              double det = A.FindDeterminant(A.Matrix1);
32
33              if (det ≠ 0)
34              {
35                  double[] roots = new double[size];
36
37                  for (int i = 0; i < roots.Length; i++)
38                  {
39                      Matrix replaced = new Matrix(formNewMatrix(A.Matrix1,  replacingColumn: B.Matrix1, i));
40                      roots[i] = replaced.FindDeterminant(replaced.Matrix1);
41                  }
42
43                  for (int i = 0; i < roots.Length; i++)
44                      roots[i] /= det;
45
46                  return roots;
47              }
48
49              return null;
50          }
```



```
52          private bool checkOnRoots(double[] beleivedRoots)
53          {
54              int size = _coefficients.GetLength( dimension: 1);
55              bool areRoots = true;
56              for (int i = 0; i < size && areRoots; i++)
57              {
58                  double left = 0;
59                  for (int j = 0; j < size - 1 && areRoots; j++)
60                      left += _coefficients[i, j] * beleivedRoots[j];
61
62                  if (left ≠ _coefficients[i, size])
63                      areRoots = false;
64              }
65
66              return areRoots;
67          }
68
69
```

2 usages

```
70          private double[,] _initMatrixA()
71          {
72              int rows = _coefficients.GetLength( dimension: 0);
73              int cols = _coefficients.GetLength( dimension: 1) - 1;
74              double[,] initialized = new double[rows, cols];
75
76              for (int i = 0; i < rows; i++)
77                  for (int j = 0; j < cols; j++)
78                      initialized[i, j] = _coefficients[i, j];
79
80              return initialized;
81          }
```

```
83          private double[,] _initMatrixB()
84          {
85              double[,] B = new double[_coefficients.GetLength( dimension: 0), 1];
86
87              for (int i = 0; i < B.Length; i++)
88                  B[i, 0] = _coefficients[i, _coefficients.GetLength( dimension: 1) - 1];
89
90              return B;
91          }
92
                1 usage
93          private double[,] formNewMatrix(double[,] mainMatrix, double[,] replacingColumn, int replacingIndex)
94          {
95              double[,] tempMatrix = _copyArray(mainMatrix);
96
97              for (int i = 0; i < tempMatrix.GetLength( dimension: 0); i++)
98              {
99                  tempMatrix[i, replacingIndex] = replacingColumn[i, 0];
100             }
101
102             return tempMatrix;
103         }
104
                1 usage
105         private double[] _convertToRootsView(double[,] rootsMatrix)
106         {
107             double[] roots = new double[rootsMatrix.GetLength( dimension: 0)];
108
109             for (int i = 0; i < roots.Length; i++)
110                 roots[i] = rootsMatrix[i, 0];
111
112             return roots;
113         }
                1 usage
115         private double[,] _copyArray(double[,] arr)
116         {
117             double[,] copiedArr = new double[arr.GetLength( dimension: (0)), arr.GetLength( dimension: 1)];
118
119             for (int i = 0; i < copiedArr.GetLength( dimension: 0); i++)
120                 for (int j = 0; j < copiedArr.GetLength( dimension: 1); j++)
121                     copiedArr[i, j] = arr[i, j];
122
123             return copiedArr;
124         }
125
126         public override string ToString()
127         {
128             string linearSystem = "";
129
130             for (int i = 0; i < _coefficients.GetLength( dimension: 0); i++)
131             {
132                 int j = 0;
133                 for (; j < _coefficients.GetLength( dimension: 1) - 2; j++)
134                     linearSystem += $"{_coefficients[i, j]}{(char)(j + 120)} + ";
135
136                 linearSystem += $"{_coefficients[i, j]}{(char)(j + 120)}";
137                 linearSystem += $" = {_coefficients[i, j + 1]}";
138
139                 linearSystem += "\n";
140             }
141
142             return linearSystem;
143         }
144     }
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#

```cs
using System;

namespace Lab5_InheritanceAndPolymorphism
{
    4 usages
    public class TwoSystemLinearEquation : TSystemLinearEquation
    {
        2 usages
        public TwoSystemLinearEquation(double[,] coefficients) : base(coefficients)
        {
            if(coefficients.GetLength( dimension: 0) == 2 && coefficients.GetLength( dimension: 1) == 3)
                _coefficients = coefficients;
            else
                Console.WriteLine("This ain`t a two variables system.");
        }
    }
}
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C#

```cs
using System;

namespace Lab5_InheritanceAndPolymorphism
{
    4 usages
    public class ThreeSystemLinearEquation : TSystemLinearEquation
    {
        2 usages
        public ThreeSystemLinearEquation(double[,] coefficients) : base(coefficients)
        {
            if(coefficients.GetLength( dimension: 0) == 3 && coefficients.GetLength( dimension: 1) == 4)
                _coefficients = coefficients;
            else
                Console.WriteLine("This ain`t a three variables system.");
        }
    }
}
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# M

```cs
using System;

namespace Lab5_InheritanceAndPolymorphism
{
    24 usages  3 exposing APIs
    public class Matrix
    {
        private double[,] _matrix;
        private double[,] _transponated;
        private double[,] _minorsMatrix;

        10 usages
        public Matrix(double[,] matrix)
        {
            _matrix = matrix;
            if (_matrix.GetLength( dimension: 0) == _matrix.GetLength( dimension: 1))
            {
                _transponated = MakeTransponated(_matrix);
                _getMinorsMatrix();
            }
        }

        6 usages
        public double[,] Matrix1 => _matrix;

        public double[,] Transponated => _transponated;
        public double[,] Minors => _minorsMatrix;
```

```
1 usage
public double[,] MakeTransponated(double[,] matrix)
{
    double[,] transponated = new double[matrix.GetLength( dimension: 0), matrix.GetLength( dimension: 1)];

    for (int i = 0; i < matrix.GetLength( dimension: 0); i++)
    {
        for (int j = 0; j < matrix.GetLength( dimension: 1); j++)
        {
            transponated[j, i] = matrix[i, j];
        }
    }

    return transponated;
}

2 usages
public double[,] MakeInverseMatrix()
{
    double det = FindDeterminant(_matrix);

    if (det == 0)
        return null;

    Matrix deltaOnCofactor = new Matrix(_minorsMatrix) * (1/det);

    double[,] inverseMatrix = deltaOnCofactor._transponated;

    return inverseMatrix;
}
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# M

```
private void _getMinorsMatrix()
{
    _minorsMatrix = new double[_matrix.GetLength( dimension: 0), _matrix.GetLength( dimension: 1)];

    for (int i = 0; i < _matrix.GetLength( dimension: 0); i++)
        for (int j = 0; j < _matrix.GetLength( dimension: 1); j++)
            _minorsMatrix[i, j] = Math.Pow(-1, (i + j)) * FindDeterminant(_formMatrix(i, j));
}

1 usage
private double[,] _formMatrix(int indexI, int indexJ)
{
    int size = _matrix.GetLength( dimension: 0) - 1;
    double[,] subMatrix = new double[size, size];

    int counterI = 0;
    int counterJ = 0;

    for (int i = 0; i < _matrix.GetLength( dimension: 0); i++)
    {
        for (int j = 0; j < _matrix.GetLength( dimension: 1); j++)
        {
            if (counterJ ⩾ size)
            {
                counterI++;
                counterJ = 0;
            }

            if (i ≠ indexI && j ≠ indexJ)
            {
                subMatrix[counterI, counterJ] = _matrix[i, j];
                counterJ++;
            }
        }
    }
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# M

4 usages

```
94          public double FindDeterminant(double[,] matrix)
95          {
96              double determinant = 0;
97
98              int size = matrix.GetLength( dimension: 0);
99
100             if (size == 1)
101                 determinant += matrix[0, 0];
102
103             if (size == 2)
104                 determinant += matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0];
105
106             if (size == 3)
107             {
108                 determinant += matrix[0, 0] * matrix[1, 1] * matrix[2, 2];
109                 determinant += matrix[0, 1] * matrix[1, 2] * matrix[2, 0];
110                 determinant += matrix[0, 2] * matrix[1, 0] * matrix[2, 1];
111                 determinant -= matrix[0, 2] * matrix[1, 1] * matrix[2, 0];
112                 determinant -= matrix[0, 1] * matrix[1, 0] * matrix[2, 2];
113                 determinant -= matrix[0, 0] * matrix[1, 2] * matrix[2, 1];
114             }
115
116             return determinant;
117         }
```

C# Main.cs × C# RandomSystemGenerator.cs × C# TSystemLinearEquation.cs × C# TwoSystemLinearEquation.cs × C# ThreeSystemLinearEquation.cs × C# Mat

```
119         public static Matrix operator *(Matrix matrix1, Matrix matrix2)
120         {
121             double[,] table1 = matrix1._matrix;
122             double[,] table2 = matrix2._matrix;
123
124             double[,] resultingTable = new double[table1.GetLength( dimension: 0), table2.GetLength( dimension: 1)];
125
126             for (var i = 0; i < table1.GetLength( dimension: 0); i++)
127             {
128                 for (var j = 0; j < table2.GetLength( dimension: 1); j++)
129                 {
130                     resultingTable[i, j] = 0;
131
132                     for (var k = 0; k < table1.GetLength( dimension: 1); k++)
133                         resultingTable[i, j] += table1[i, k] * table2[k, j];
134                 }
135             }
136
137             return new Matrix(resultingTable);
138         }
139
140         public static Matrix operator *(Matrix matrix1, double number)
141         {
142             double[,] table1 = matrix1._matrix;
143
144             for (int i = 0; i < table1.GetLength( dimension: 0); i++)
145             {
146                 for (int j = 0; j < table1.GetLength( dimension: 1); j++)
147                 {
148                     table1[i, j] *= number;
149                 }
150             }
151
152             return new Matrix(table1);
153         }
154
```

```
155             public static Matrix operator *(Matrix matrix1, double[] number)
156             {
157                 double[,] table1 = matrix1._matrix;
158 
159                 for (int i = 0; i < table1.GetLength( dimension: 0); i++)
160                 {
161                     for (int j = 0; j < table1.GetLength( dimension: 1); j++)
162                     {
163                         table1[j, i] *= number[i];
164                     }
165                 }
166 
167                 return new Matrix(table1);
168             }
169 
170         }
171     }
```

Python:

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
from RandomCoefficientsGenerator import RandSystem
from ThreeVarSystem import ThreeSystemLinearEquation
from TwoVarSystem import TwoSystemLinearEquation
from Printer import SystemsPrinter

first_matrix = RandSystem.generate_random_system(2)
first_sys = TwoSystemLinearEquation(first_matrix)
SystemsPrinter.print_system(first_sys)
first_roots = first_sys.matrix_method()
SystemsPrinter.print_roots(first_roots)

second_matrix = RandSystem.generate_random_system(2)
second_sys = TwoSystemLinearEquation(second_matrix)
SystemsPrinter.print_system(second_sys)
second_roots = second_sys.kramer_method()
SystemsPrinter.print_roots(second_roots)

third_matrix = RandSystem.generate_random_system(3)
third_sys = ThreeSystemLinearEquation(third_matrix)
SystemsPrinter.print_system(third_sys)
third_roots = third_sys.matrix_method()
SystemsPrinter.print_roots(third_roots)

fourth_matrix = RandSystem.generate_random_system(3)
fourth_sys = ThreeSystemLinearEquation(fourth_matrix)
SystemsPrinter.print_system(fourth_sys)
fourth_roots = fourth_sys.matrix_method()
SystemsPrinter.print_roots(fourth_roots)
```

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
from abc import ABC
import numpy.linalg
from numpy import array


class TSystemLinearEquation(ABC):
    def __init__(self, coefficients):
        self.__coefficients = array(coefficients)

    def kramer_method(self):
        A, B = self.__coefficients[:, :-1], self.__coefficients[:, -1]
        roots = []
        determinant = self.count_determinant(A)
        if determinant != 0:
            index = 0
            while index < len(A):
                replaced = self.make_replaced_matrix(A, index, B)
                roots.append(self.count_determinant(replaced))
                roots[index] /= determinant
                index += 1

            return roots

    def matrix_method(self):
        A, B = self.__coefficients[:, :-1], self.__coefficients[:, -1]
        determinant = numpy.linalg.det(A)
        if determinant != 0:
            inverted_matrix = numpy.linalg.inv(A)
            roots = numpy.matmul(inverted_matrix, B)
            return roots

    def count_determinant(self, matrix):
        det = 0
        if len(matrix) == 2:
            det += matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0]
        else:
            det += matrix[0, 0] * matrix[1, 1] * matrix[2, 2]
```

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
                roots.append(self.count_determinant(replaced))
                roots[index] /= determinant
                index += 1

            return roots

    def matrix_method(self):
        A, B = self.__coefficients[:, :-1], self.__coefficients[:, -1]
        determinant = numpy.linalg.det(A)
        if determinant != 0:
            inverted_matrix = numpy.linalg.inv(A)
            roots = numpy.matmul(inverted_matrix, B)
            return roots

    def count_determinant(self, matrix):
        det = 0
        if len(matrix) == 2:
            det += matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0]
        else:
            det += matrix[0, 0] * matrix[1, 1] * matrix[2, 2]
            det += matrix[0, 1] * matrix[1, 2] * matrix[2, 0]
            det += matrix[0, 2] * matrix[1, 0] * matrix[2, 1]
            det -= matrix[0, 2] * matrix[1, 1] * matrix[2, 0]
            det -= matrix[0, 1] * matrix[1, 0] * matrix[2, 2]
            det -= matrix[0, 0] * matrix[1, 2] * matrix[2, 1]
        return det

    def make_replaced_matrix(self, main_matrix, index, replacer):
        temp = main_matrix
        i = 0
        while i < len(temp):
            main_matrix[i, index] = replacer[i]
            i += 1
        return temp
```

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
from numpy import array
from LinearSystem import TSystemLinearEquation


class TwoSystemLinearEquation(TSystemLinearEquation):
    def __init__(self, coefficients):
        super().__init__(coefficients)
        self.__coefficients = array(coefficients)

    def __str__(self):
        lin_system = ''
        for row in self.__coefficients:
            lin_system += f'{row[0]}x + {row[1]}y = {row[2]}\n'

        return lin_system
```

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
from numpy import array
from LinearSystem import TSystemLinearEquation


class ThreeSystemLinearEquation(TSystemLinearEquation):
    def __init__(self, coefficients):
        super().__init__(coefficients)
        self.__coefficients = array(coefficients)

    def __str__(self):
        lin_system = ''
        for row in self.__coefficients:
            lin_system += f'{row[0]}x + {row[1]}y '
            lin_system += f'{row[2]}z = {row[3]}\n'

        return lin_system
```

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
import random


class RandSystem:
    @staticmethod
    def generate_random_system(size):
        lin_system = []
        for i in range(size):
            col = []
            for j in range(size + 1):
                col.append(random.randint(0, 20))
            lin_system.append(col)
        return lin_system
```

main.py | LinearSystem.py | TwoVarSystem.py | ThreeVarSystem.py | RandomCoefficientsGenerator.py | Printer.py

```python
class SystemsPrinter:
    @staticmethod
    def print_system(lin_system):
        print('System:')
        print(f'{lin_system}')

    @staticmethod
    def print_roots(roots):
        if roots is not None:
            i = 0
            for root in roots:
                print(f'{chr(i + 120)} = {round(root, 3)}')
                i += 1
            print('\n')
        else:
            print('Equation has infinite roots')

```

Тестування:

C#:

```
System:
7x + 5y = 3
4x + 7y = 5

x = -0,00475624256837099
y = 0,027348394768133177



System:
3x + 9y = 1
8x + 6y = 1

x = 0,05555555555555555
y = 0,09259259259259259



System:
0x + 3y + 3z = 1
5x + 5y + 5z = 8
2x + 1y + 3z = 4

x = -0,04222222222222223
y = 0,007777777777777772
z = -0,018888888888888886
```

```
System:
4x + 5y + 6z = 9
2x + 0y + 2z = 4
4x + 9y + 6z = 5

x = -1
y = -1
z = 3
```

Python:

```
System:
9x + 14y = 19
11x + 11y = 2

x = -3.291
y = 3.473


System:
17x + 20y = 8
0x + 2y = 2

x = -0.706
y = 0.0


System:
2x + 9y 2z = 3
15x + 5y 15z = 3
16x + 13y 16z = 14

Equation has infinite roots
System:
11x + 16y 5z = 0
16x + 1y 7z = 17
5x + 18y 1z = 9

x = 6.456
y = -0.613
z = -12.241
```