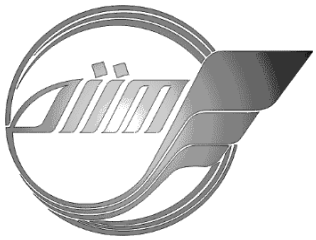
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



Дніпровський національний університет
залізничного транспорту імені академіка В. Лазаряна

Кафедра «Комп'ютерні інформаційні технології»

Лабораторна робота № 15

з дисципліни «Об'єктно-орієнтоване програмування»

на тему: «Робота з файловими потоками Java.»

Виконав:
студент гр.ПЗ1911
Сафонов Д. Є.
Прийняла:
Демидович І.М.

Дніпро, 2021

Тема. Робота з файловими потоками Java.

Завдання. Написати об'єктно-орієнтовану програму на мові Java, яка використовує файлові потоки для збереження та зчитування матриці з файлу. Індивідуальні завдання взяти з попередньої лабораторної роботи.

Індивідуальне завдання.

Діаграма класів.

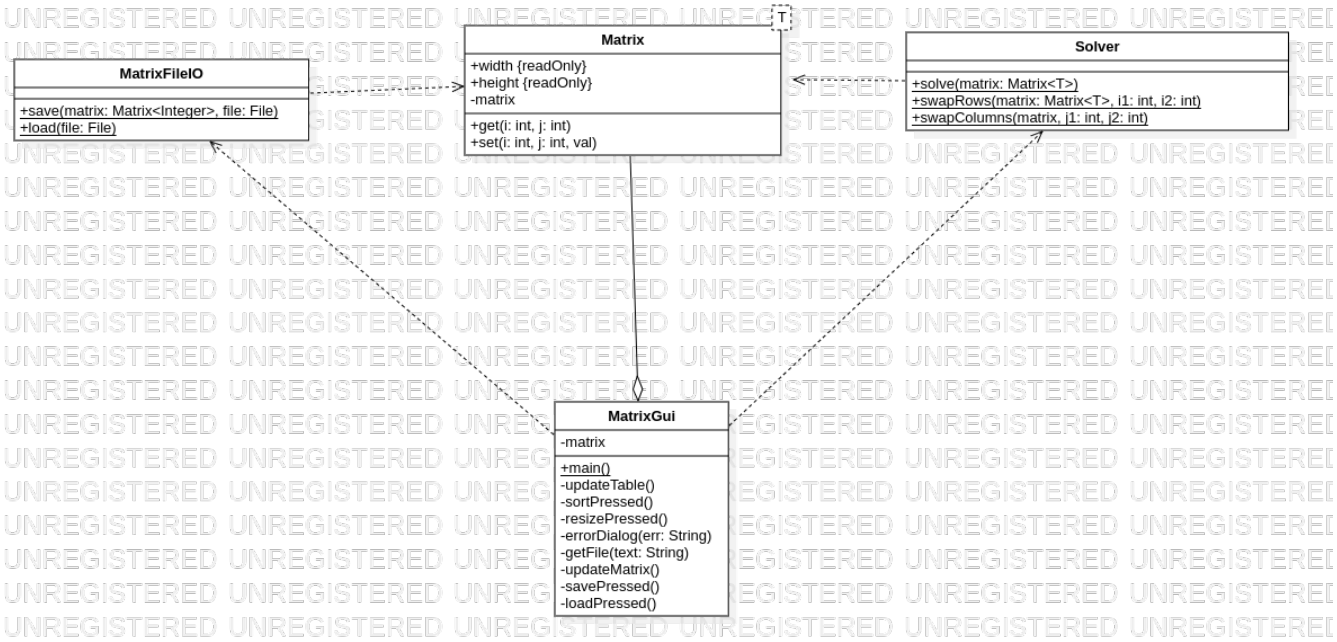


Рисунок 1

Опис файлових потоків, які використовуються в програмі.

BufferedReader - зчитує символи з потоку, буферизує їх щоб забезпечити ефективне читання рядків.
FileReader - клас для читання текстових файлів.
FileWriter - клас для запису текстових файлів.

Опис патерну Декоратор та його застосування в роботі з файлами.

Патерн декоратор є способом додавання нового функціоналу (операцій) до вже існуючого класу і його об'єктів за допомогою інкапсуляції об'єктів(або посилань на них) вже існуючого класу в новому "Декораторі". Цей клас повинен реалізовувати всі інтерфейси існуючого, однак може змінювати їх поведінку (в цьому і є мета таких класів) або додавати нові методи.

Зазвичай перевизначені методи не просто мають подібну поведінку, але також засновані на методах основного класу і просто їх доповнюють.

Використаний у програмі клас BufferedReader можна вважати декоратором для класу Reader, він додає нові методи для зчитування рядків, але ці операції основані на операціях зчитування символу. FileReader можна вважати декоратором для класу InputStreamReader, який спеціалізується на читанні текстових файлів. FileWriter можна вважати декоратором класу FileWriter, який спеціалізується на записі текстових файлів.

Текст програми.

“MatrixGui.java”

```
import java.io.File;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.filechooser.FileSystemView;
import com.dazzlemon.oop15.*;

public class MatrixGui extends javax.swing.JFrame {
    public MatrixGui() {
        initComponents();
        //
        updateTable();
        //
    }

    private void updateTable() {
        var m = new Integer[matrix.height][matrix.width];
        for (int i = 0; i < matrix.height; i++) {
            m[i] = new Integer[matrix.width];
            for (int j = 0; j < matrix.width; j++) {
                m[i][j] = matrix.get(i, j);
            }
        }

        var names = new String[matrix.width];
        for (int j = 0; j < matrix.width; j++) {
            names[j] = String.valueOf(j);
        }
        table.setModel(new javax.swing.table.DefaultTableModel(m, names));
        table.setShowGrid(true);
        table.setTableHeader(null);
        jScrollPane1.setViewportView(table);
    }

    private void sortButtonActionPerformed(java.awt.event.ActionEvent evt) {
        updateMatrix();

        var indices = Solver.solve(matrix);
            var x = indices.x;
            var y = indices.y;

        Solver.swapRows(matrix, 0, x);
```

```
            Solver.swapColumns(matrix, 0, y);

        updateTable();
    }

    private void resizeButtonActionPerformed(java.awt.event.ActionEvent evt) {
        int w = Integer.parseInt(columnsText.getText());
        int h = Integer.parseInt(rowsText.getText());
        var newMatrix = new Matrix<>(h, w, 0);
        for (int i = 0; i < h && i < matrix.height; i++) {
            for (int j = 0; j < w && j < matrix.width; j++) {
                var mij = Integer.parseInt(table
                                    .getModel()
                                    .getValueAt(i, j)
                                    .toString());
                newMatrix.set(i, j, mij);
            }
        }
        matrix = newMatrix;
        updateTable();
    }

    private void updateMatrix() {
        for (int i = 0; i < matrix.height; i++) {
            for (int j = 0; j < matrix.width; j++) {
                var mij = Integer.parseInt(table
                                    .getModel()
                                    .getValueAt(i, j)
                                    .toString());
                matrix.set(i, j, mij);
            }
        }
    }

    private void errorDialog(String err) {
        JOptionPane.showMessageDialog(
                new JFrame(),
                err,
                "Error",
                JOptionPane.ERROR_MESSAGE
        );
    }

    private void saveButtonActionPerformed(java.awt.event.ActionEvent evt) {
        var file = getFile("save");
        if (file == null) {
            return;
```

```
        }
        if (!file.getName().endsWith(".txt")) {
            errorDialog("INCORRECT FILE EXTENSION!");
            return;
        }
        if (!file.exists()) {
            try {
                file.createNewFile();
            } catch (IOException e) {
                errorDialog("ERROR OCCURED WHILE CREATING FILE!");
                return;
            }
        }
        if (!file.canWrite()) {
            errorDialog("CAN\'T WRITE TO THAT FILE!");
            return;
        }

        updateMatrix();
        try {
            MatrixFileIO.matrixSave(matrix, file);
        } catch (IOException e) {
            errorDialog("ERROR OCCURED WHILE WRITING TO THE FILE!");
        }
    }

    private void loadButtonActionPerformed(java.awt.event.ActionEvent evt) {
        var file = getFile("load");
        if (file == null) {
            return;
        }
        if (!file.getName().endsWith(".txt")) {
            errorDialog("INCORRECT FILE EXTENSION!");
            return;
        }
        if (!file.exists()) {
            errorDialog("FILE DOESN\'T EXIST!");
            return;
        }
        if (!file.canRead()) {
            errorDialog("CAN\'T READ THAT FILE!");
            return;
        }
        try {
            matrix = MatrixFileIO.matrixLoad(file);
            updateTable();
        } catch (IOException e) {
```

```
            errorDialog("ERROR OCCURED WHILE READING THE FILE!");
        } catch (Exception e) {
            errorDialog(e.getMessage());
        }
    }

    private File getFile(String text) {
        var j = new JFileChooser(FileSystemView
                                 .getFileSystemView()
                                 .getHomeDirectory());
        var f = new FileNameExtensionFilter("txt", "txt");
        j.setFileFilter(f);
        j.setAcceptAllFileFilterUsed(false);
        var r = j.showDialog(null, text);
        if (r == JFileChooser.APPROVE_OPTION) {
            return j.getSelectedFile();
        }
        return null;
    }

    private Matrix<Integer> matrix = new Matrix<Integer>(4, 4, 0);
}
```

“MatrixFileIO.java”

```
package com.dazzlemon.oop15;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class MatrixFileIO {
    public static void matrixSave(Matrix<Integer> matrix, File file)
            throws IOException {
        var fw = new FileWriter(file);
        //write size (rows cols)
        var buffer = "%d %d\n"
                . formatted(matrix.height, matrix.width);
        fw.write(buffer);
        //write elements
        for (int i = 0; i < matrix.height; i++) {
            for (int j = 0; j < matrix.width; j++) {
                buffer = "%d "
                        . formatted(matrix.get(i, j));
                fw.write(buffer);
            }
            fw.append('\n');
        }
        fw.flush();
    }


    public static Matrix<Integer> matrixLoad(File file)
            throws IOException, Exception {
        var br = new BufferedReader(new FileReader(file));
        var line = br.readLine();
        var words = line.split("\\s");
        if (words.length != 2) {
            throw new Exception("incorrect amount tokens on size line");
        }
        var h = Integer.parseInt(words[0]);
        var w = Integer.parseInt(words[1]);
        var matrix = new Matrix<Integer>(h, w, 0);
        for (int i = 0; i < h; i++) {
            line = br.readLine();
            if (line == null) {
                throw new Exception(
                    "incorrect amount of lines: %d instead of %d"
```

```
                    .formatted(i, h)
            );
        }
        words = line.split("\\s");
        if (words.length != w) {
            throw new Exception(
                "incorrect amount of elements on line " + (i + 1)
            );
        }
        for (int j = 0; j < w; j++) {
            var mij = Integer.parseInt(words[j]);
            matrix.set(i, j, mij);
        }
    }
    if (br.ready()) {
        throw new Exception("too many lines");
    }
    return matrix;
  }
}
```

“Matrix.java”

```
package com.dazzlemon.oop15;

import java.util.ArrayList;
import java.util.List;

public class Matrix <T> {
   public final int width;
   public final int height;
   private List<List<T>> matrix;


   public Matrix(int height, int width, T filler) {
         this.height = height;
         this.width  = width;

         this.matrix = new ArrayList<>(height);
         for (int i = 0; i < height; i++) {
         this.matrix.add(i, new ArrayList<>(width));
         for (int j = 0; j < width; j++) {
                  this.matrix.get(i).add(filler);
         }
         }
   }


   public void set(int i, int j, T val) {
         this.matrix.get(i).set(j, val);
   }

   public T get(int i, int j) {
         return this.matrix.get(i).get(j);
   }
}
```

“Solver.java”

```
package com.dazzlemon.oop15;

import java.awt.Point;

public class Solver {
   public static <T extends Comparable<T>> Point solve(Matrix<T> matrix) {
         int imax = 0;
         int jmax = 0;
         for (int i = 0; i < matrix.height; i++) {
         for (int j = 0; j < matrix.width; j++) {
                  if (matrix.get(i, j).compareTo(matrix.get(imax, jmax)) > 0) {
               imax = i;
               jmax = j;
                  }
         }
         }
         return new Point(imax, jmax);
   }

   public static <T> void swapRows(Matrix<T> matrix, int i1, int i2) {
      for (int j = 0; j < matrix.width; j++) {
         var tmp = matrix.get(i1, j);
         matrix.set(i1, j, matrix.get(i2, j));
         matrix.set(i2, j, tmp);
         }
   }

   public static <T> void swapColumns(Matrix<T> matrix, int j1, int j2) {
      for (int i = 0; i < matrix.height; i++) {
         var tmp = matrix.get(i, j1);
         matrix.set(i, j1, matrix.get(i, j2));
         matrix.set(i, j2, tmp);
         }
   }
}
```

Приклад роботи програми.

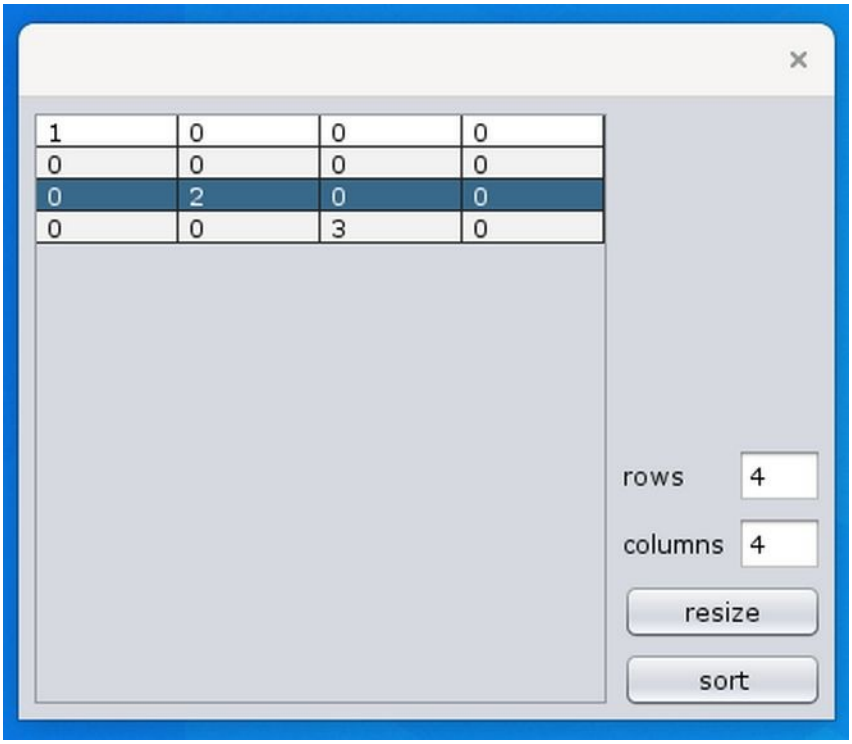


Рисунок 2(Тест 1. Підготовка)

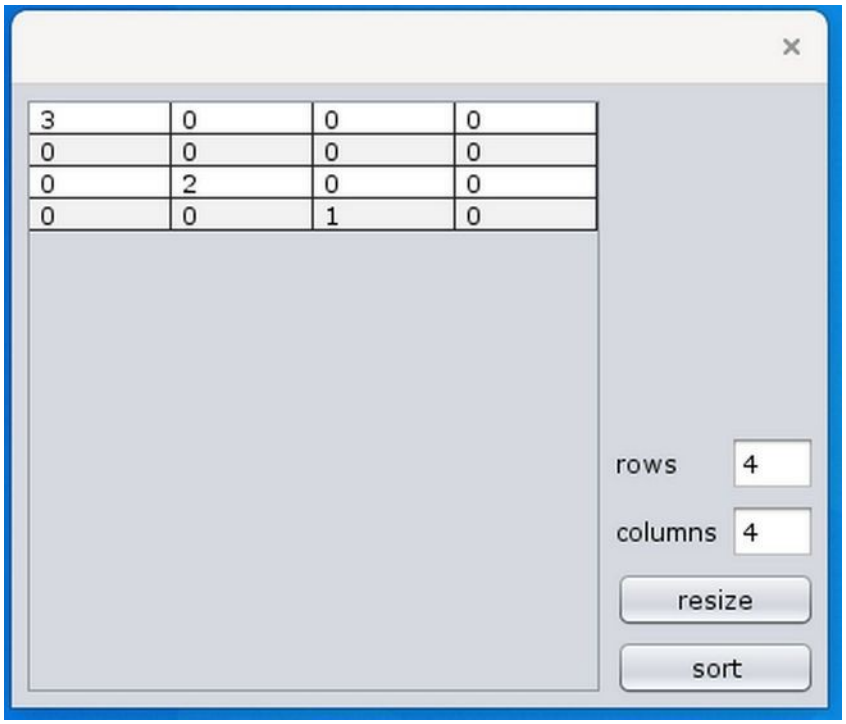


Рисунок 3(Тест 1. Стовпці та Рядки були замінені так, що найбільший єлемент знаходиться за індексами 0, 0)

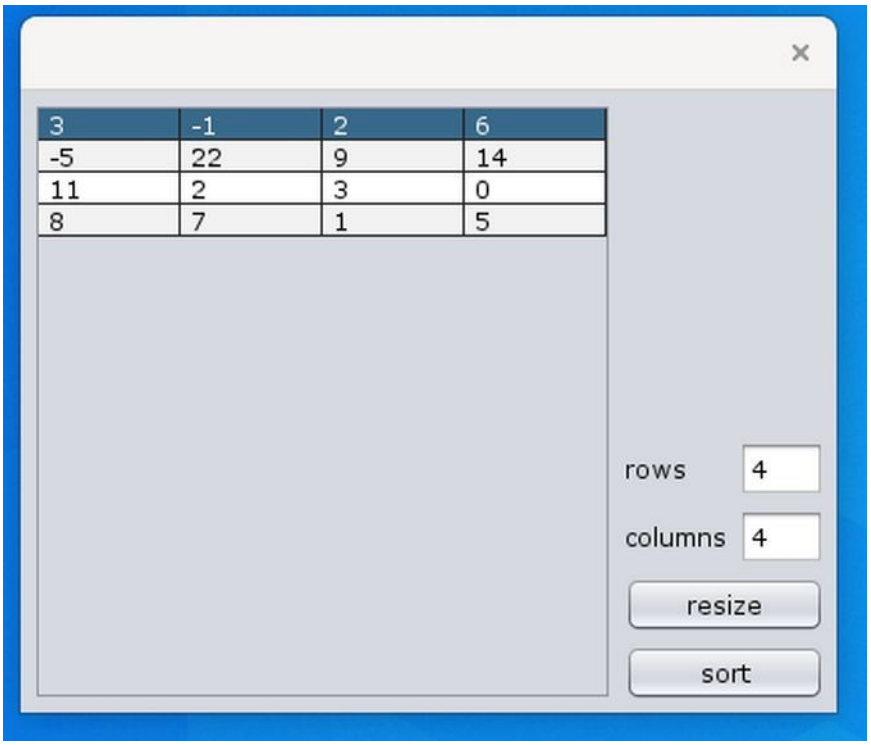


Рисунок 4(Тест 2. Підготовка)

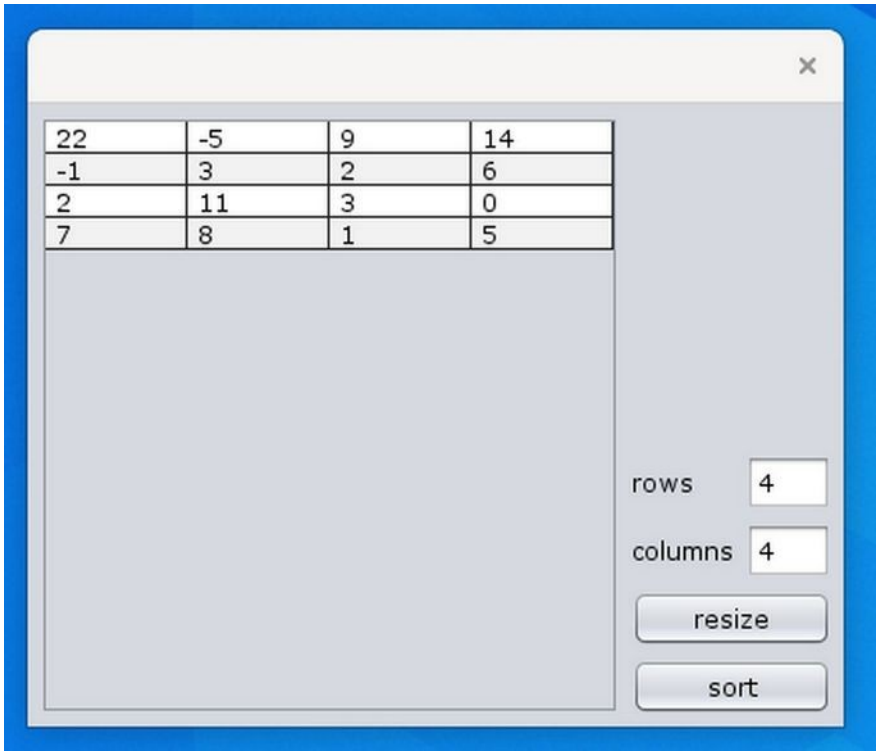


Рисунок 5(Тест 2. Найбільший елемент за індексом 0, 0)

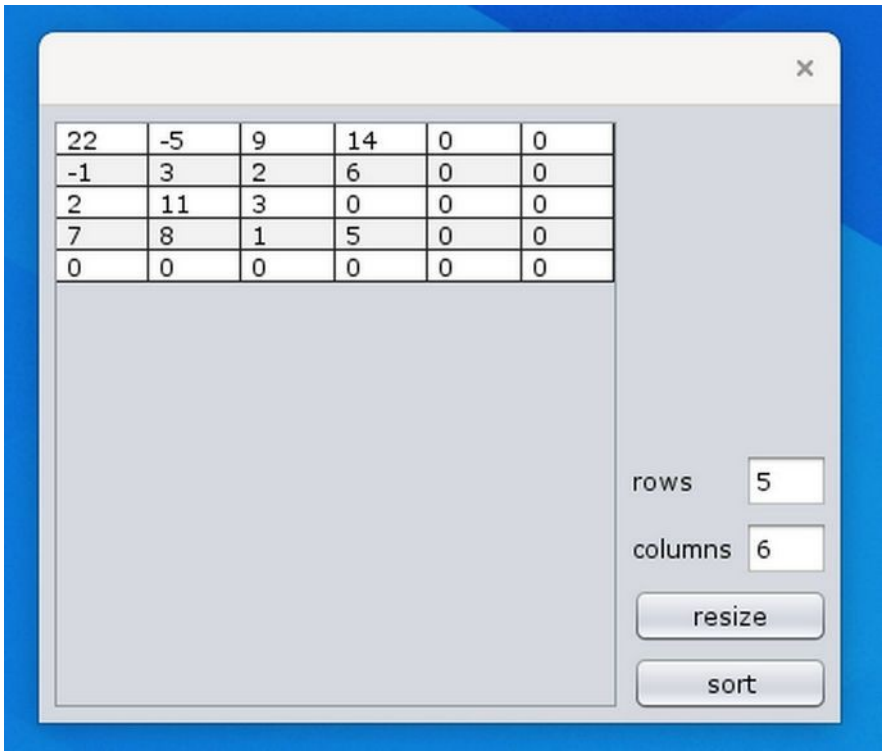


Рисунок 6(збільшення розміру)

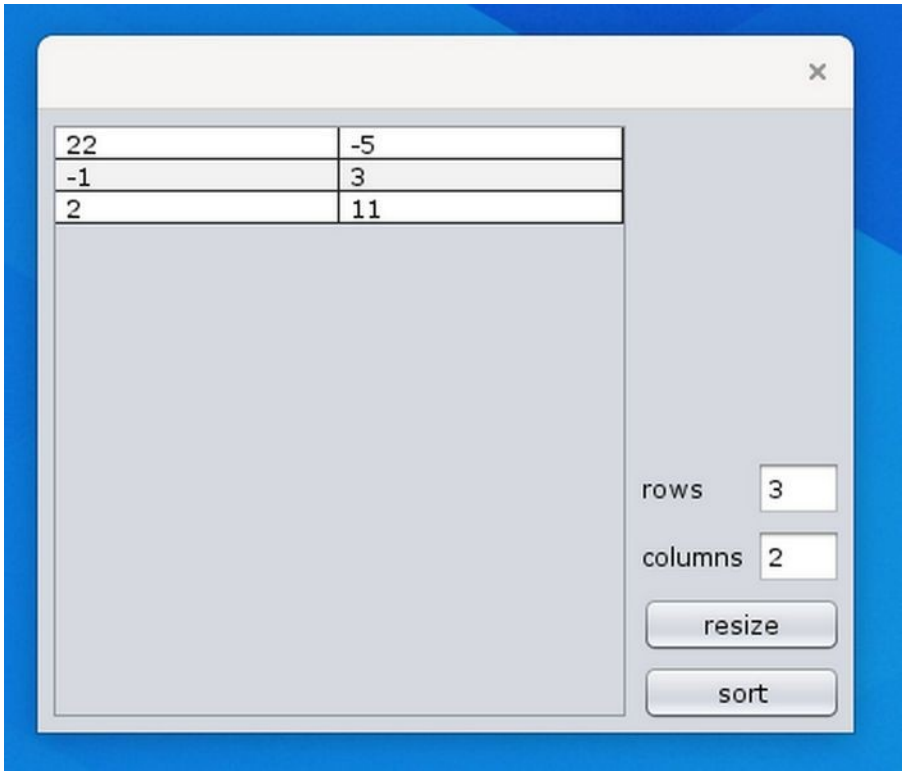


Рисунок 7(зменьшення розміру)

Перші 6 скріншотів були скопійовані з попередньої лабораторної роботи через те, що та частина функціоналу яку вони показують ніяк не змінилась.

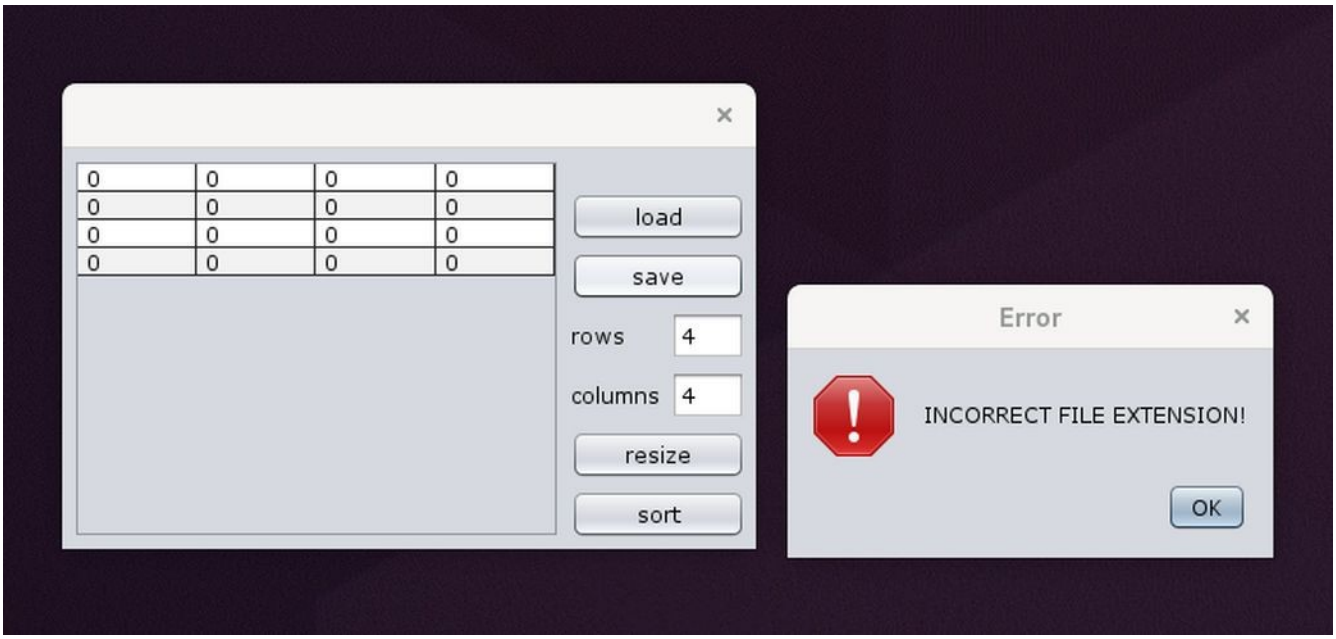


Рисунок 8(Приклад вікна помилки при зчитуванні або записі у файл)

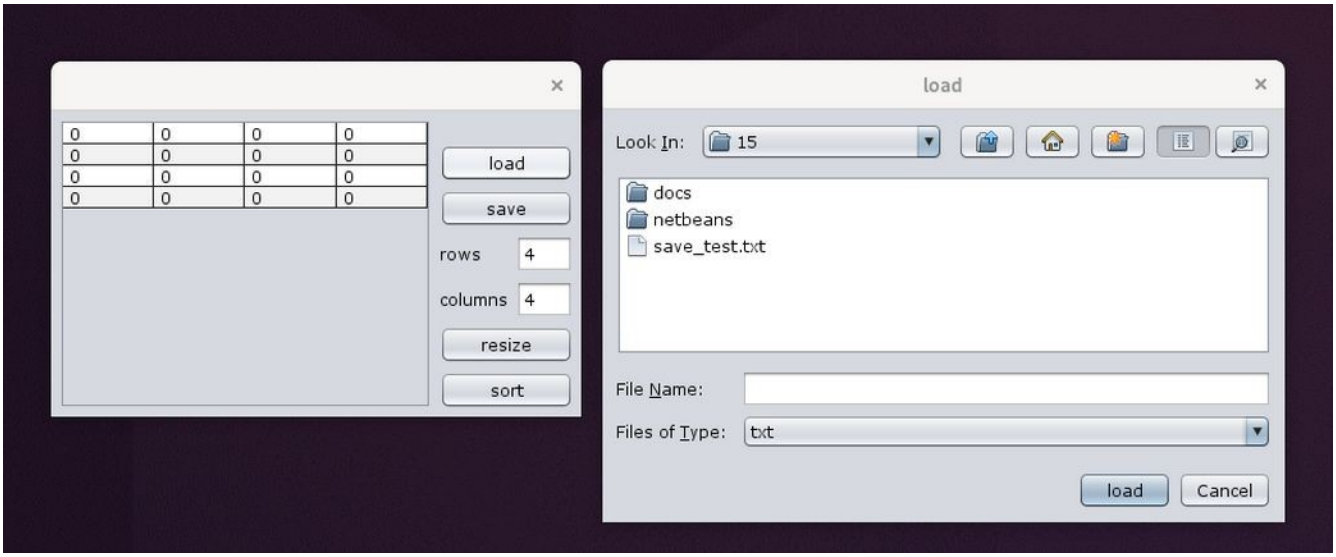


Рисунок 9(Приклад вікна вибору файлу)

Рисунок 10(Приклад матриці збереженої у файлі)

Не були продемонстровані усі варіації помилок, та вибору файлів, бо їх було б забагато, а різниці між ними майже немає.

Висновки.

Розроблена програма заснована на програмах із лабораторних робіт 13 та 14. Класи Matrix<T>, який представляє абстрактну матрицю; Solver, який вирішує поставлене завдання із 13ої ЛР не були змінені. Клас MatrixGui притерпів невеликих змін, а саме:

- були додані методи:
  - updateMatrix - функціонал цієї функції вже був присутній у версії з 14ої ЛР, але знадобилося повторне використання, тож код був винесений у функцію.
  - errorDialog - Виведення вікна із помилкою.
  - getFile - Виведення вікна для вибору файлу.
- та обробники подій для кнопок збереження та завантаження матриці до та з файлу відповідно.

Для збереження був обраний текстовий формат, щоб файли можна було редагувати не тільки у цій програмі, а й в текстових редакторах. Матриця зберігається у файлі в наступному форматі:

- Розмір матриці(Висота та Ширина через пробіл).
- Рядки матриці.

При некоректному форматі викидаються виключення(у програмі вони оброблені, і викликають діалогові вікна із повідомленням про помилку):

- У рядку із розмірами не 2 токени
- У якомусь із рядків кількість токенів не збігається з очікуваною
- Недостатньо рядків
- Забагато рядків
- Токен не може бути зведений до цілого числа