МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Информационной безопасности

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Модели безопасности компьютерных систем»
Тема: Модернизация матрицы доступа.

Студенты гр. 1361 _____ Горбунова Д.А.
_____ Кравцов И.Ю.
Преподаватель _____ Шкляр Е.В.

Санкт-Петербург
2024

ТЕОРЕТИЧЕСКАЯ СПРАВКА

Матрица доступа (МД) – это готовая модель, позволяющая регламентировать доступ к информационным ресурсам компании, но основании которой можно оценить состояние и структуру защиты данных в информационных системах. В матрице четко устанавливаются права для каждого субъекта по отношению ко всем объектам информации.

Визуально это можно представить в качестве некого массива данных со множеством ячеек, которые формируются пересечением строки, указывающей на субъект и столбика, указывающего на объект. Получается, что при таком подходе к управлению доступом ячейка содержит определенную запись, характерную для пары субъект-объект и указывает на режим доступа, разрешенный или запрещенный, или его характеристику для каждого конкретного случая. В матрице доступа к информационным ресурсам столбец отождествляется с перечнем контроля доступа, строка выполняет роль профиля доступа присущего объекту.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Задача: доработать программу администратора из ЛР2, чтобы изменять матрицу доступа посредством команд grant, create, remove.

Технические требования:

1) Команда grant выдаёт заданному набору субъектов права доступа на заданный набор объектов.

2) Команда create создаёт новый субъект с доступом к заданному набору объектов (набор может быть пустым). Если субъект уже существует, create работает как grant. Если некоторых объектов из заданного набора не существует, их нужно автоматически создать.

3) Команда remove удаляет у заданного набора субъектов права на заданный набор объектов.

4) Предусмотрите режимы grant_all и remove_all, чтобы выдать или забрать у заданного набора субъектов права доступа на все существующие объекты.

5) В коде программы grant, create и remove должны быть реализованы как отдельные функции и принимать соответствующие аргументы.

6) Предусмотрите обработку ошибок.

Старые функции из ЛР2 должны работать:

- Ввод имён субъектов и объектов. Имена регистрозависимы, то есть G и g — разные объекты. Предусмотрите обработку ошибок при вводе.

- Сохранение и загрузка матрицы доступа из файла;

- Отображение матрицы доступа в окне программы и возможность её интерактивного изменения без перезагрузки программы.

Для проверки используем программу пользователя из ЛР2.

Важно: обе программы остаются оконными с пользовательским интерфейсом. Язык программирования любой. Все необходимые для работы текстовые поля должны быть подписаны. Консольные приложения в этой работе не принимаются.

ХОД РАБОТЫ

1) К программе администратора, которая была создана в прошлой лабораторной работе были добавлены следующие функции: создаёт новый субъект с доступом к заданному набору объектов (*create access right*), выдать или забрать у заданного набора субъектов права доступа на все существующие объекты (*grant all access right* и *remove all access right*, соотвественно), а также переделана функция *grant access right*, которая теперь может выдаёт заданному набору субъектов права доступа на заданный набор объектов, и *remove access right*, которая удаляет у заданного набора субъектов права на заданный набор объектов. Обновленный интерфейс программы представлен на рисунке 1.

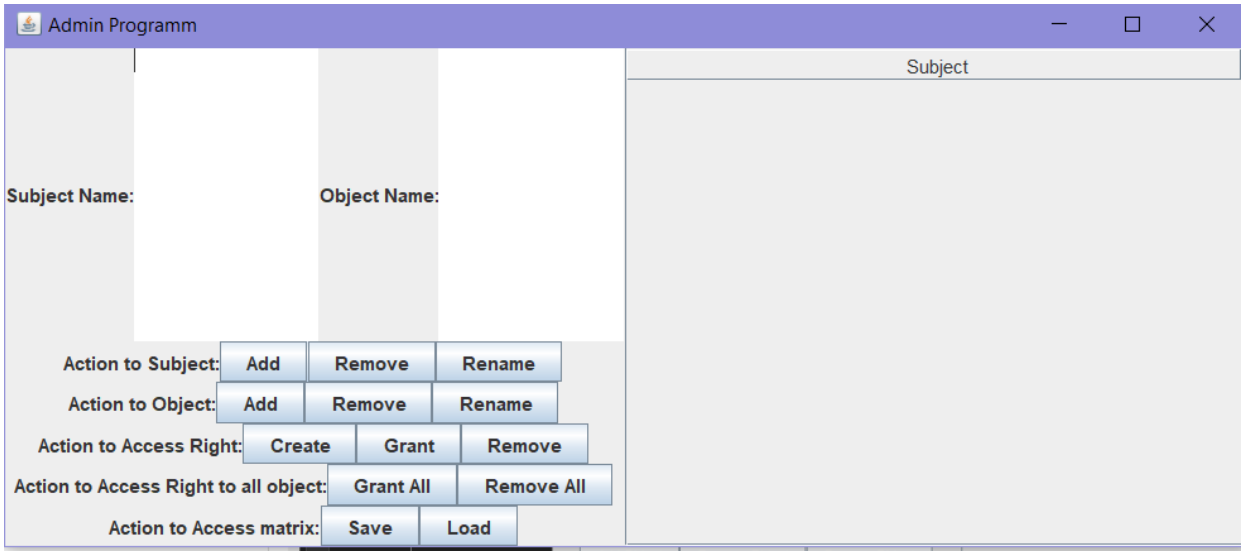


Рисунок 1 – Интерфейс программы администратора

Далее рассмотрим каждую из новых функций в отдельности. Начнем с функции *create access right*. Данная функция позволяет создавать новые субъекты с доступом к заданному набору объектов. Для этого нужно ввести имена субъектов, которые нужно создать, разделив имена переносом строки, и соответственно с объектами, далее нажав на кнопку «Create», таблица прав доступа справа обновится, выполнив задачу и выдав всем права доступа. Результат работы функции представлен на рисунке 2 и 3.

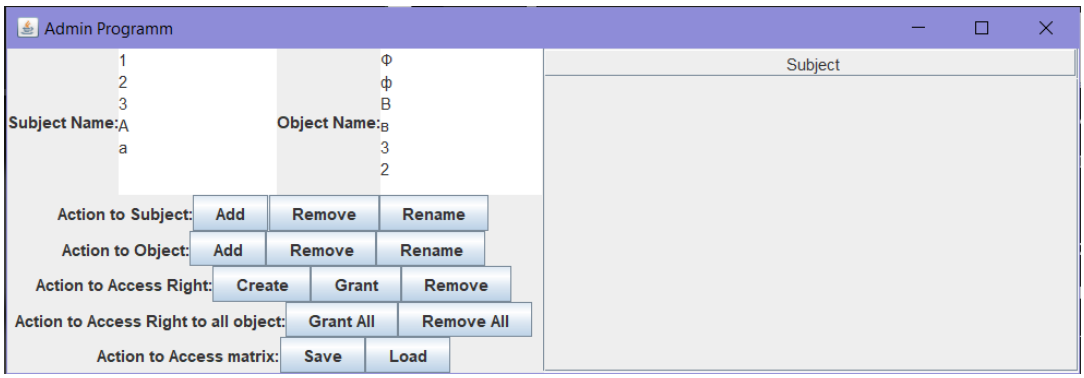


Рисунок 2 – Работа функции *Create*

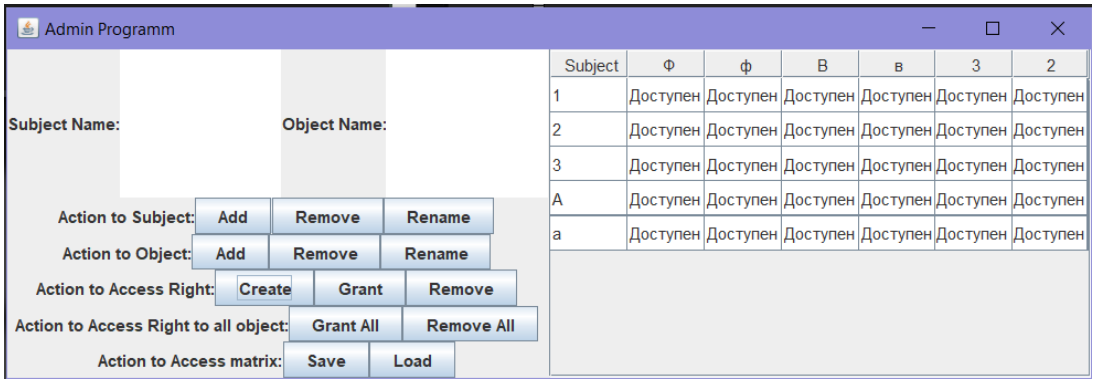


Рисунок 3 – результат работы функции *Create*

Функция *remove all access right* позволяет забрать у заданного набора субъектов права доступа на все существующие объекты. Для этого достаточно ввести нужные нам субъекты в поле разделив их переносом строки и нажать кнопку *Remove All.* Результат представлен на рисунке 4 и 5.

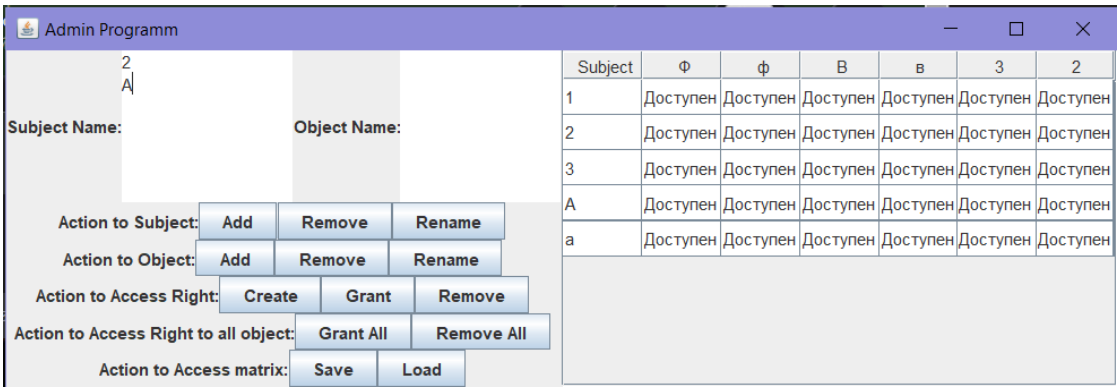


Рисунок 4 – результат работы функции *Remove All*

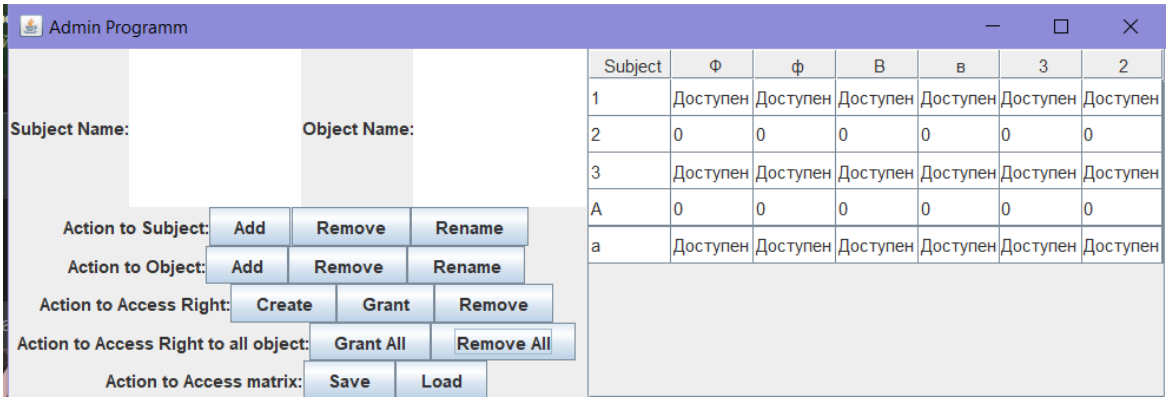


Рисунок 5 – результат работы функции *Remove All*

Функция *grant all access right* позволяет выдать заданному набору субъектов права доступа на все существующие объекты. Для этого достаточно ввести нужные нам субъекты в поле разделив их переносом строки и нажать кнопку *Grant All.* Результат представлен на рисунке 6 и 7.

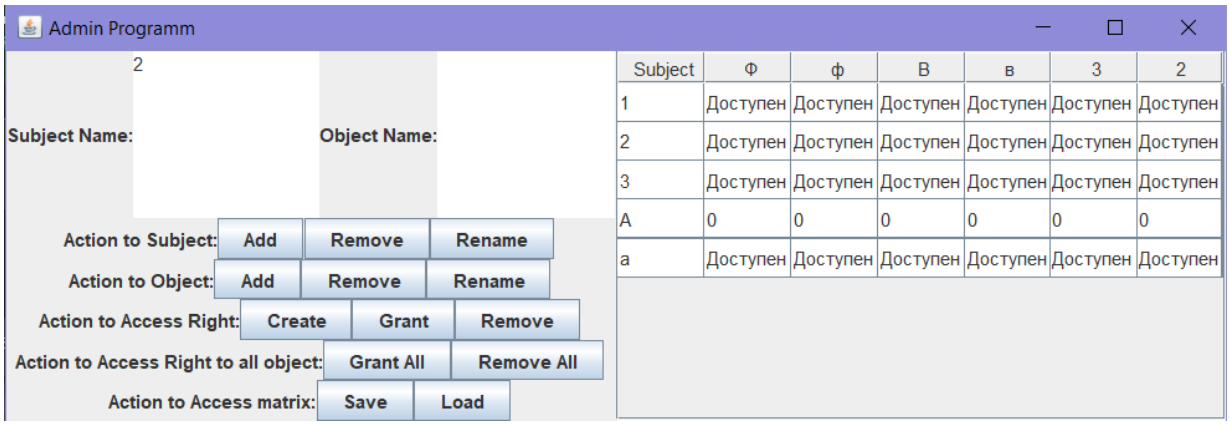


Рисунок 6 – результат работы функции *Grant All*.

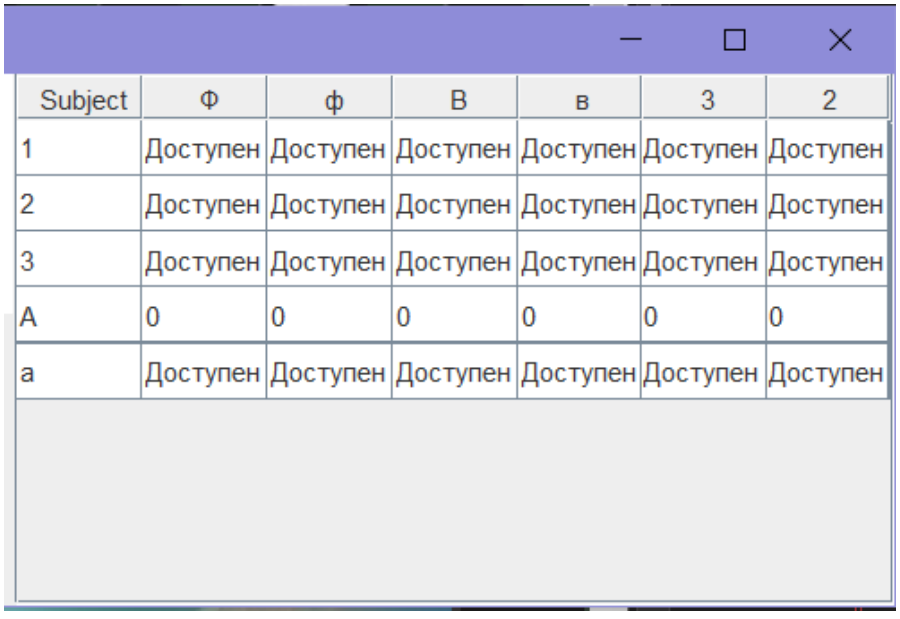
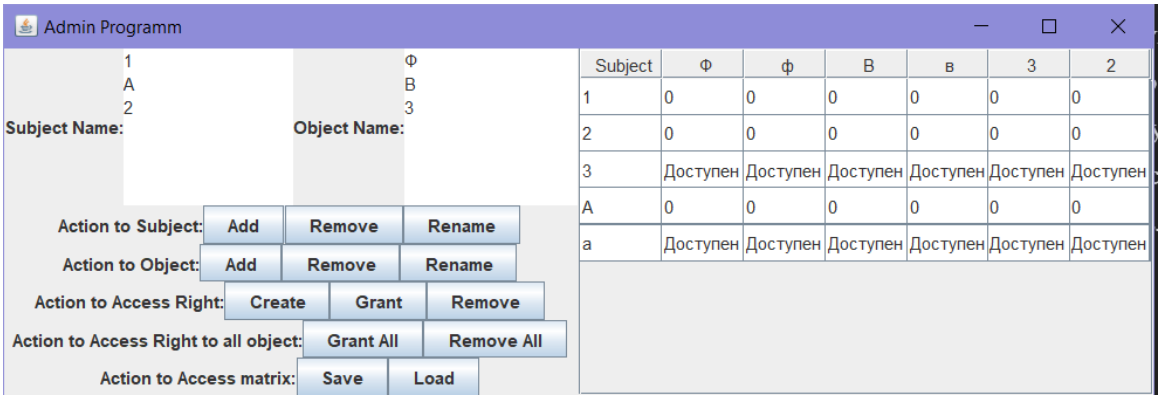


Рисунок 7 – результат работы функции *Grant All*.

Функция *grant access right* позволяет теперь выдавать заданному набору субъектов права доступа на заданный набор объектов. Для этого достаточно ввести нужные нам субъекты в поле разделив их переносом строки и аналогично сделать со списком объектов, после чего нужно нажать кнопку *Grant.* Результат представлен на рисунке 8 и 9.



Subject	Ф	ф	В	в	3	2
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	Доступен	Доступен	Доступен	Доступен	Доступен	Доступен
A	0	0	0	0	0	0
a	Доступен	Доступен	Доступен	Доступен	Доступен	Доступен

Рисунок 8 – результат работы функции *Grant*.

Subject	Ф	ф	В	в	3	2
1	Доступен	0	Доступен	0	Доступен	0
2	Доступен	0	Доступен	0	Доступен	0
3	Доступен	Доступен	Доступен	Доступен	Доступен	Доступен
A	Доступен	0	Доступен	0	Доступен	0
a	Доступен	Доступен	Доступен	Доступен	Доступен	Доступен

Рисунок 9 – результат работы функции *Grant*.

Функция *remove access right* позволяет теперь забрать у заданного набора субъектов права доступа на заданный набор объектов. Для этого достаточно ввести нужные нам субъекты в поле разделив их переносом строки и нажать кнопку *Remove.* Результат представлен на рисунке 10 и 11.

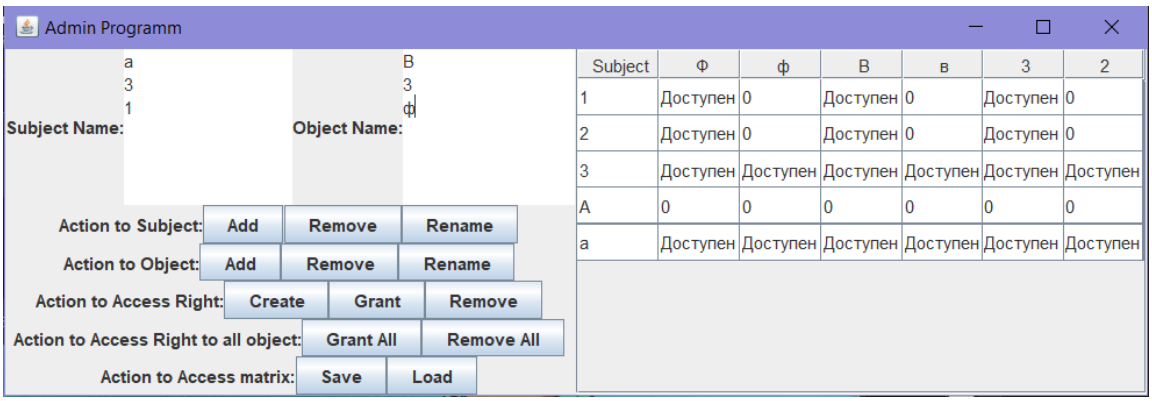


Рисунок 10 – результат работы функции *Remove*.

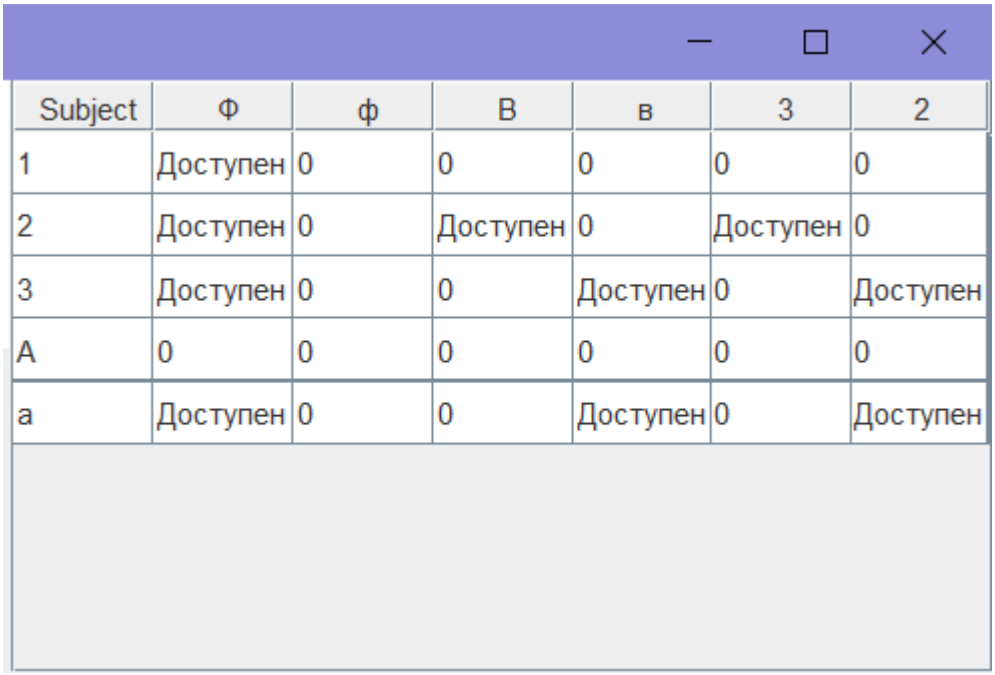


Рисунок 11 – результат работы функции *Remove*.

2) Была реализована программа пользователя с графическим интерфейсом. Интерфейс предоставлен на рисунке 12.

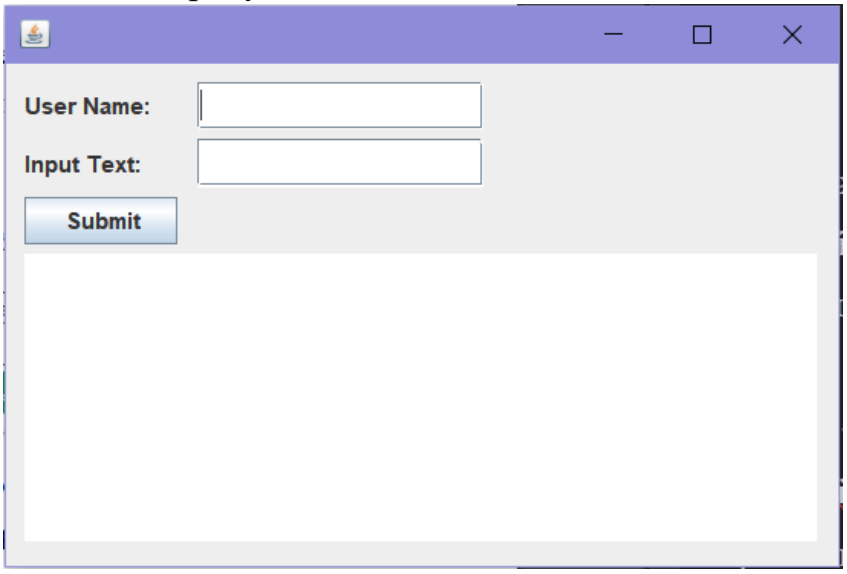


Рисунок 12 – Интерфейс программы

- Теперь введём имя пользователя и строку, права доступа в которой надо выделить для пользователя (рисунок 13).

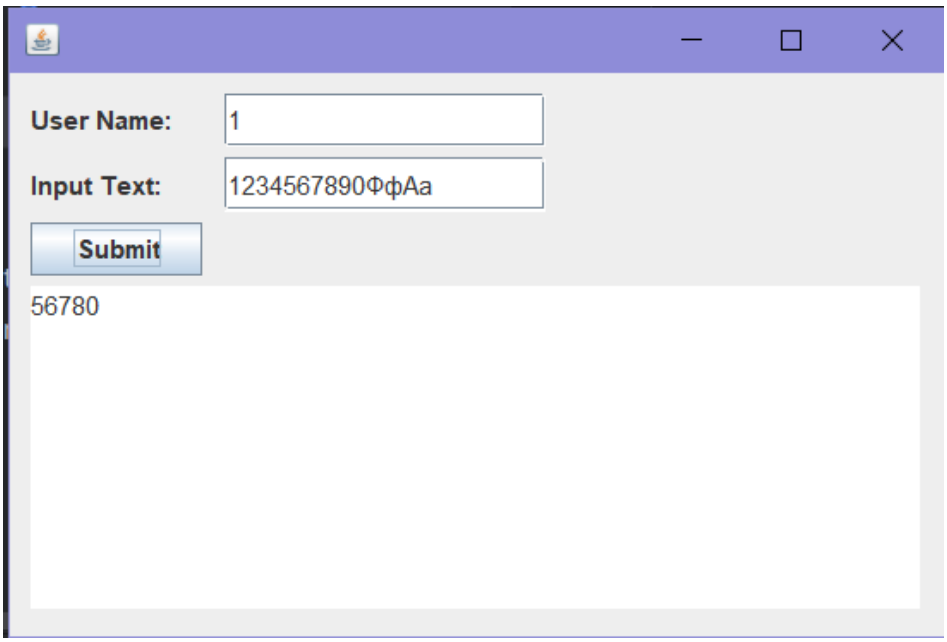


Рисунок 13 – Проверка прав доступа User.

Как видно, программа пользователя выводит только те объекты введенной строки, которые совпадают с объектами, на которые имеют права данные субъекты.

Так же в ходе тестирования были выявлены и решены следующие проблемы: создание и выдача прав доступа субъектам и объектам, длина названия которых более одного символа, в программе User вывод доступных объектов не совпадает с введенной строкой.

ВЫВОДЫ.

В ходе данной лабораторной работы были реализованы две программы. Первая программа создаёт, изменяет и удаляет объекты, субъекты, а также выдает, удаляет и создает доступ субъектов к отдельным или всем объектам. Вторая программа проверяет наличие прав доступа у введённого субъекта на введенные в виде строки объекты. Также в ходе лабораторной работы была рассмотрена модель матрицы доступа.

КОД ПРИЛОЖЕНИЯ.

1. Программа пользователя.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;

public class UserProgram {
    private JFrame frame;
    private JTextField userNameField;
    private JTextField inputTextField;
    private JTextArea filteredTextArea;
    private JButton submitButton;

    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            try {
                UserProgram window = new UserProgram();
                window.frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
    public UserProgram() {
        initialize();
    }

    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel lblUserName = new JLabel("User Name:");
        lblUserName.setBounds(10, 10, 80, 25);
        frame.getContentPane().add(lblUserName);

        userNameField = new JTextField();
        userNameField.setBounds(100, 10, 150, 25);
        frame.getContentPane().add(userNameField);
        userNameField.setColumns(10);
```

```
            JLabel lblInputText = new JLabel("Input Text:");
            lblInputText.setBounds(10, 40, 80, 25);
            frame.getContentPane().add(lblInputText);

            inputTextField = new JTextField();
            inputTextField.setBounds(100, 40, 150, 25);
            frame.getContentPane().add(inputTextField);
            inputTextField.setColumns(10);

            submitButton = new JButton("Submit");
            submitButton.setBounds(10, 70, 80, 25);
            frame.getContentPane().add(submitButton);

            filteredTextArea = new JTextArea();
            filteredTextArea.setBounds(10, 100, 414, 150);
            frame.getContentPane().add(filteredTextArea);
            submitButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    String userName = userNameField.getText();
                    String inputText = inputTextField.getText();
                    StringBuilder filteredText = filterText(userName,
inputText);
                    filteredTextArea.setText(filteredText.toString());
                }
            });
        }
        private StringBuilder filterText(String userName, String inputText) {
            String accessRightsFilePath = "access_rights.txt";
            StringBuilder filteredText = new StringBuilder();
            try {
                List<String> lines =
Files.readAllLines(Paths.get(accessRightsFilePath));
                for (String line : lines) {
                    String[] parts = line.split(":");
                    if (parts.length == 2 && parts[0].equals(userName)) {
                        var objects = List.of(parts[1].split("-"));
                        for (char c : inputText.toCharArray()) {
                            String charStr = String.valueOf(c);
                            if (objects.contains(charStr)) {
                                filteredText.append(charStr);
                            }
                        }
                        break;
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }
            return filteredText;
        }
    }
```

2. Программа администратора.

```
    import dto.MyObject;
    import dto.Subject;
    import tools.AccessTableModel;
    import tools.IoTools;

    import javax.swing.*;
    import javax.swing.table.TableColumn;
    import java.awt.*;
    public class AccessControlGUI extends JFrame {
        private JTextArea subjectNameField, objectNameField;
        private JButton addSubjectButton, addObjectButton;
        private JButton grantAccessRightButton, removeAccessRightButton,
createSubjectWithAccessButton;
        private JButton grantAllAccessButton, removeAllAccessButton;
        private JButton saveAccessMatrixButton, loadAccessMatrixButton;
        private JButton removeSubjectButton, removeObjectButton;
        private JButton renameSubjectButton, renameObjectButton;
        private JTable accessTable;
        private AccessTableModel tableModel;

        public AccessControlGUI() {
            setLayout(new GridLayout(1, 2));
            setSize(800, 600);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setTitle("Admin Programm");

            subjectNameField = new JTextArea();
            objectNameField = new JTextArea();

            addSubjectButton = new JButton("Add");
            removeSubjectButton = new JButton("Remove");
            renameSubjectButton = new JButton("Rename");

            addObjectButton = new JButton("Add");
            removeObjectButton = new JButton("Remove");
            renameObjectButton = new JButton("Rename");

            createSubjectWithAccessButton = new JButton("Create");
            grantAccessRightButton = new JButton("Grant");
            grantAllAccessButton = new JButton("Grant All");
            removeAccessRightButton = new JButton("Remove");
            removeAllAccessButton = new JButton("Remove All");
```

```
            saveAccessMatrixButton = new JButton("Save");
            loadAccessMatrixButton = new JButton("Load");

            tableModel = new AccessTableModel();
            accessTable = new JTable(tableModel);
            setTableBounds();

            tableModel.addTableModelListener(e -> refreshTable());

            JPanel panelInput = new JPanel();
            panelInput.setLayout(new BoxLayout(panelInput, BoxLayout.X_AXIS));
            panelInput.add(new JLabel("Subject Name:"));
            panelInput.add(subjectNameField);
            panelInput.add(new JLabel("Object Name:"));
            panelInput.add(objectNameField);

            JPanel PanelSubject = new JPanel();
            PanelSubject.setLayout(new BoxLayout(PanelSubject,
BoxLayout.X_AXIS));
            PanelSubject.add(new JLabel("Action to Subject:"));
            PanelSubject.add(addSubjectButton);
            PanelSubject.add(removeSubjectButton);
            PanelSubject.add(renameSubjectButton);

            JPanel panelObject = new JPanel();
            panelObject.setLayout(new BoxLayout(panelObject,
BoxLayout.X_AXIS));
            panelObject.add(new JLabel("Action to Object:"));
            panelObject.add(addObjectButton);
            panelObject.add(removeObjectButton);
            panelObject.add(renameObjectButton);

            JPanel panelAccessRight = new JPanel();
            panelAccessRight.setLayout(new BoxLayout(panelAccessRight,
BoxLayout.X_AXIS));
            panelAccessRight.add(new JLabel("Action to Access Right:"));
            panelAccessRight.add(createSubjectWithAccessButton);
            panelAccessRight.add(grantAccessRightButton);
            panelAccessRight.add(removeAccessRightButton);

            JPanel panelAccessRight2 = new JPanel();
            panelAccessRight2.setLayout(new BoxLayout(panelAccessRight2,
BoxLayout.X_AXIS));
            panelAccessRight2.add(new JLabel("Action to Access Right to all
object:"));
            panelAccessRight2.add(grantAllAccessButton);
            panelAccessRight2.add(removeAllAccessButton);

            JPanel panelMatrix = new JPanel();
```

```
            panelMatrix.setLayout(new BoxLayout(panelMatrix,
BoxLayout.X_AXIS));
            panelMatrix.add(new JLabel("Action to Access matrix:"));
            panelMatrix.add(saveAccessMatrixButton);
            panelMatrix.add(loadAccessMatrixButton);

            Box buttonBox = Box.createVerticalBox();
            buttonBox.add(panelInput);
            buttonBox.add(PanelSubject);
            buttonBox.add(panelObject);
            buttonBox.add(panelAccessRight);
            buttonBox.add(panelAccessRight2);
            buttonBox.add(panelMatrix);

            add(buttonBox, BorderLayout.WEST);
            add(new JScrollPane(accessTable), BorderLayout.EAST);

            addSubjectButton.addActionListener(e -> addSubject());
            removeSubjectButton.addActionListener(e -> removeSubject());
            renameSubjectButton.addActionListener(e -> renameSubject());

            addObjectButton.addActionListener(e ->addObject());
            removeObjectButton.addActionListener(e -> removeObject());
            renameObjectButton.addActionListener(e -> renameObject());

            createSubjectWithAccessButton.addActionListener(e -> {
                String[] subjectNames =
subjectNameField.getText().split("\n");
                String[] objectNames = objectNameField.getText().split("\n");
                createSubjectWithAccess( subjectNames, objectNames);
                subjectNameField.setText("");
                objectNameField.setText("");
                refreshTable();
            });
            grantAccessRightButton.addActionListener(e -> {
                String[] subjectNames =
subjectNameField.getText().split("\n");
                String[] objectNames = objectNameField.getText().split("\n");
                grantAccessRight(subjectNames, objectNames);
                subjectNameField.setText("");
                objectNameField.setText("");
                refreshTable();
            });
            grantAllAccessButton.addActionListener(e -> grantAllAccess());
            removeAccessRightButton.addActionListener(e -> {
                String[] subjectNames =
subjectNameField.getText().split("\n");
                String[] objectNames = objectNameField.getText().split("\n");
                removeAccessRight(subjectNames, objectNames);
```

```
                subjectNameField.setText("");
                objectNameField.setText("");
                refreshTable();
            });
            removeAllAccessButton.addActionListener(e -> removeAllAccess());

            saveAccessMatrixButton.addActionListener(e -> {
                IoTools.saveAccessMatrixToFile(tableModel,
"access_rights.txt");
            });
            loadAccessMatrixButton.addActionListener(e -> {
                IoTools.loadAccessMatrixFromFile(tableModel,
"access_rights.txt");
                refreshTable();
            });
        }

        public Subject createSubject(String subjectName) {
            Subject newSubject = new Subject(subjectName);

           if ( tableModel.addSubject(newSubject) == 0){
            return newSubject;}
           else return null;
        }
        public MyObject createObject(String objectName) {
            MyObject newObject = new MyObject(objectName);

            if (tableModel.addObject(newObject) == 0){
            return newObject;}
            else {return null;}
        }
        private void syncColumnModel() {
            var columnModel = accessTable.getColumnModel();
            var model = accessTable.getModel();
            while (columnModel.getColumnCount() > 0) {
                columnModel.removeColumn(columnModel.getColumn(0));
            }
            for (int i = 0; i < model.getColumnCount(); i++) {
                var col = new TableColumn(i);
                col.setHeaderValue(model.getColumnName(i));
                columnModel.addColumn(col);
            }
        }
        private void refreshTable(){
            syncColumnModel();
            accessTable.revalidate();
            accessTable.repaint();
        }
```

```
        private void setTableBounds() {
            accessTable.setRowHeight(25);
            for (int i = 0; i < accessTable.getColumnCount(); i++) {

accessTable.getColumnModel().getColumn(i).setPreferredWidth(100);
            }
        }

        /**Action to Subject*/
        private void addSubject() {
            String subjectName = subjectNameField.getText();
            if (subjectName.length() == 1){
                if (subjectName.isEmpty()) {
                    JOptionPane.showMessageDialog(null,
                            "Subject name cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if (tableModel.subjectExists(subjectName)) {
                    JOptionPane.showMessageDialog(null,
                            "Subject name already exists.", "Error",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
                tableModel.addSubject(new Subject(subjectName));
                subjectNameField.setText("");
                refreshTable();}
        }
        private void removeSubject() {
            String subjectName = subjectNameField.getText();
            if (subjectName.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Subject name cannot be
empty.", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }
            tableModel.removeSubject(subjectName);
            subjectNameField.setText("");
            refreshTable();
        }
        private void renameSubject() {
            JFrame renameSubjectFrame = new JFrame("Rename Subject");
            renameSubjectFrame.setSize(300, 150);
            renameSubjectFrame.setLayout(new GridLayout(3, 2));

renameSubjectFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

            JLabel oldNameLabel = new JLabel("Old Name:");
            JTextField oldNameField = new JTextField();
            JLabel newNameLabel = new JLabel("New Name:");
```

```
            JTextField newNameField = new JTextField();
            JButton renameButton = new JButton("Rename");

            renameButton.addActionListener(e -> {
                String oldName = oldNameField.getText();
                String newName = newNameField.getText();
                if (!oldName.isEmpty() && !newName.isEmpty() &&
newName.length() <= 1) {
                    tableModel.renameSubject(oldName, newName);
                    renameSubjectFrame.dispose();
                } else {
                    JOptionPane.showMessageDialog(null, "Both fields must be
filled.", "Error", JOptionPane.ERROR_MESSAGE);
                }

                refreshTable();
            });

            renameSubjectFrame.add(oldNameLabel);
            renameSubjectFrame.add(oldNameField);
            renameSubjectFrame.add(newNameLabel);
            renameSubjectFrame.add(newNameField);
            renameSubjectFrame.add(renameButton);
            renameSubjectFrame.setVisible(true);
        }

        /**Action to Object*/
        private void addObject() {
            String objectName = objectNameField.getText();
            if (objectName.length() == 1){
                if (objectName.isEmpty()) {
                    JOptionPane.showMessageDialog(null,
                            "Object name cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
                if (tableModel.objectExists(objectName)) {
                    JOptionPane.showMessageDialog(null,
                            "Object name already exists.", "Error",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
                tableModel.addObject(new MyObject(objectName));
                objectNameField.setText("");
                refreshTable();}
        }
        private void removeObject() {
            String objectName = objectNameField.getText();
            if (objectName.isEmpty()) {
```

```
                JOptionPane.showMessageDialog(null,
                        "Object name cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }
            tableModel.removeObject(objectName);
            objectNameField.setText("");
            refreshTable();
        }
        private void renameObject() {
            JFrame renameObjectFrame = new JFrame("Rename Object");
            renameObjectFrame.setSize(300, 150);
            renameObjectFrame.setLayout(new GridLayout(3, 2));

renameObjectFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

            JLabel oldNameLabel = new JLabel("Old Name:");
            JTextField oldNameField = new JTextField();
            JLabel newNameLabel = new JLabel("New Name:");
            JTextField newNameField = new JTextField();
            JButton renameButton = new JButton("Rename");

            renameButton.addActionListener(e -> {
                String oldName = oldNameField.getText();
                String newName = newNameField.getText();
                if (!oldName.isEmpty() && !newName.isEmpty()&&
newName.length() <= 1) {
                    tableModel.renameObject(oldName, newName);
                    renameObjectFrame.dispose();
                } else {
                    JOptionPane.showMessageDialog(null, "Both fields must be
filled.", "Error", JOptionPane.ERROR_MESSAGE);
                }

                refreshTable();
            });

            renameObjectFrame.add(oldNameLabel);
            renameObjectFrame.add(oldNameField);
            renameObjectFrame.add(newNameLabel);
            renameObjectFrame.add(newNameField);
            renameObjectFrame.add(renameButton);
            renameObjectFrame.setVisible(true);

        }

        /**Action to Access Right*/
        public void createSubjectWithAccess(String[] subjectNames, String[]
objectNames) {
```

```
            if (subjectNames.length == 0 || objectNames.length == 0) {
                JOptionPane.showMessageDialog(null,
                        "Subject or object names cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
                return;
            }

            for (String subjectName: subjectNames){
                Subject subject = tableModel.findSubjectByName(subjectName);
                if (subject == null) {
                    subject = createSubject(subjectName);
                    if (subject == null) continue;
                }

                for (String objectName : objectNames) {
                    MyObject myObject =
tableModel.findObjectByName(objectName);
                    if (myObject == null) {
                        myObject = createObject(objectName);
                        if (myObject == null) continue;
                    }
                    tableModel.grantAccessRight(subject, myObject);
                }
            }
        }
        private void grantAccessRight(String[] subjectNames, String[]
objectNames){
            if (subjectNames.length == 0 || objectNames.length == 0) {
                JOptionPane.showMessageDialog(null,
                        "Subject or object names cannot be empty.",
"Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            for (String subjectName : subjectNames) {
                Subject subject =
tableModel.findSubjectByName(subjectName.trim());
                if (subject == null) {
                    JOptionPane.showMessageDialog(null,
                            "Subject not found: " + subjectName, "Error",
JOptionPane.ERROR_MESSAGE);
                    continue;
                }

                for (String objectName : objectNames) {
                    MyObject myObject =
tableModel.findObjectByName(objectName.trim());
                    if (myObject == null) {
                        JOptionPane.showMessageDialog(null,
```

```
                                        "Object not found: " + objectName,
"Error", JOptionPane.ERROR_MESSAGE);
                                continue;
                            }
                            tableModel.grantAccessRight(subject, myObject);
                        }
                    }
                }
            private void removeAccessRight(String[] subjectNames, String[]
objectNames){
                if (subjectNames.length == 0 || objectNames.length == 0) {
                    JOptionPane.showMessageDialog(null,
                            "Subject or object names cannot be empty.", "Error",
JOptionPane.ERROR_MESSAGE);
                    return;
                }
                for (String subjectName : subjectNames) {
                    Subject subject =
tableModel.findSubjectByName(subjectName.trim());
                    if (subject == null) {
                        JOptionPane.showMessageDialog(null,
                                "Subject not found: " + subjectName, "Error",
JOptionPane.ERROR_MESSAGE);
                        continue;
                    }

                    for (String objectName : objectNames) {
                        MyObject myObject =
tableModel.findObjectByName(objectName.trim());
                        if (myObject == null) {
                            JOptionPane.showMessageDialog(null,
                                    "Object not found: " + objectName, "Error",
JOptionPane.ERROR_MESSAGE);
                            continue;
                        }

                        tableModel.removeAccessRight(subjectName, objectName);
                    }
                }
            }

            public void grantAllAccess() {
                String[] subjectNames = subjectNameField.getText().split("\n");
                MyObject[] allObjects = tableModel.getAllObjects();

                for (String subjectName : subjectNames) {
                    Subject subject = tableModel.findSubjectByName(subjectName);
                    if (subject == null) {
                        Subject newSubject = new Subject(subjectName);
```

```
                    tableModel.addSubject(newSubject);
                    subject = newSubject;
                }

                for (MyObject object : allObjects) {
                    tableModel.grantAccessRight(subject, object);
                }
            }
            subjectNameField.setText("");
            objectNameField.setText("");
            refreshTable();
        }
        public void removeAllAccess() {
            String[] subjectNames = subjectNameField.getText().split("\n");
            MyObject[] allObjects = tableModel.getAllObjects();

            for (String subjectName : subjectNames) {
                Subject subject = tableModel.findSubjectByName(subjectName);
                if (subject != null) {
                    for (MyObject object : allObjects) {
                        tableModel.removeAccessRight(subject.getName(),
object.getName());
                    }
                }
            }
            subjectNameField.setText("");
            objectNameField.setText("");
            refreshTable();
        }

        public static void main(String[] args) {
            AccessControlGUI gui = new AccessControlGUI();
            gui.setVisible(true);
        }
    }
```

3. Вспомогательные классы

```
package tools;

import dto.AccessRight;
import dto.MyObject;
import dto.Subject;

import javax.swing.*;
import javax.swing.table.AbstractTableModel;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

```
public class AccessTableModel extends AbstractTableModel {
    private List<Subject> subjects;
    private List<MyObject> myObjects;
    private Set<AccessRight> accessRights;

    public AccessTableModel() {
        this.subjects = new ArrayList<>();
        this.myObjects = new ArrayList<>();
        this.accessRights = new HashSet<>();
    }

    public int addSubject(Subject subject) {
        if (subject.getName().length() > 1){
            JOptionPane.showMessageDialog(null,
                    "Subject "+subject.getName()+" name is long.",
"Error", JOptionPane.ERROR_MESSAGE);
            return 1;
        }else{
        subjects.add(subject);
        return 0;}
    }
    public int addObject(MyObject myObject) {
        if (myObject.getName().length() > 1){
            JOptionPane.showMessageDialog(null,
                    "Object "+myObject.getName()+" name is long.",
"Error", JOptionPane.ERROR_MESSAGE);
            return 1;
        }else{
        myObjects.add(myObject);
        return 0;}
    }
    public void grantAccessRight(Subject subject, MyObject myObject) {
        AccessRight accessRight = new AccessRight(subject, myObject);
        accessRights.add(accessRight);
    }
    public void removeSubject(String name) {
        subjects.removeIf(subject -> subject.getName().equals(name));
    }
    public void removeObject(String name) {
        myObjects.removeIf(myObject -> myObject.getName().equals(name));
    }
    public void removeAccessRight(String subjectName, String objectName) {
        accessRights.removeIf(accessRight ->
                accessRight.getSubject().getName().equals(subjectName) &&

accessRight.getObject().getName().equals(objectName)
        );
    }
```

```
        public MyObject[] getAllObjects() {
            return myObjects.toArray(new MyObject[0]);
        }
        @Override
        public int getRowCount() {
            return subjects.size();
        }
        @Override
        public int getColumnCount() {
            return myObjects.size() + 1;
        }
        @Override
        public String getColumnName(int column) {
            if (column == 0) {
                return "Subject";
            }
            return myObjects.get(column - 1).getName();
        }
        @Override
        public String getValueAt(int rowIndex, int columnIndex) {
            if (columnIndex == 0) {
                return subjects.get(rowIndex).getName();
            }
            String subjectName = subjects.get(rowIndex).getName();
            String objectName = myObjects.get(columnIndex - 1).getName();
            return findAccessRight(subjectName, objectName);
        }
        @Override
        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return false;
        }
        @Override
        public void setValueAt(Object aValue, int rowIndex, int columnIndex)
{}
        public List<Subject> getSubjects() {return subjects;}
        public List<MyObject> getObjects() {return myObjects; }
        public Set<AccessRight> getAccessRights() {return accessRights;}
        public String findAccessRight(String subjectName, String objectName) {
            for (AccessRight accessRight : accessRights) {
                if (accessRight.getSubject().getName().equals(subjectName) &&
                        accessRight.getObject().getName().equals(objectName))
{ return "Доступен"; }
            }
            return "0";
        }
        public Subject findSubjectByName(String name) {
            for (Subject subject : subjects) {
                if (subject.getName().equals(name)) {
```

```
                return subject;
            }
        }
        return null;
    }
    public MyObject findObjectByName(String name) {
        for (MyObject myObject : myObjects) {
            if (myObject.getName().equals(name)) {
                return myObject;
            }
        }
        return null;
    }
    public boolean subjectExists(String name) {
        for (Subject subject : subjects) {
            if (subject.getName().equals(name)) { return true; }
        }
        return false;
    }
    public boolean objectExists(String name) {
        for (MyObject myObject : myObjects) {
            if (myObject.getName().equals(name)) {
                return true;
            }
        }
        return false;
    }
    public void renameSubject(String oldName, String newName) {
        Subject subjectToRename = findSubjectByName(oldName);
        if (subjectToRename != null) {
            if (!subjectExists(newName)) {
                subjectToRename.setName(newName);
                updateSubjectReferences(oldName, newName);
            } else { System.out.println("A subject with the new name
already exists."); }
        } else { System.out.println("Subject not found."); }
    }
    public void renameObject(String oldName, String newName) {
        MyObject objectToRename = findObjectByName(oldName);
        if (objectToRename != null) {
            if (!objectExists(newName)) {
                objectToRename.setName(newName);
                updateObjectReferences(oldName, newName);
            } else { System.out.println("An object with the new name
already exists."); }
        } else { System.out.println("Object not found."); }
    }
    private void updateSubjectReferences(String oldName, String newName) {
```

```
            for (AccessRight accessRight : accessRights) {
                if (accessRight.getSubject().getName().equals(oldName)) {
                    accessRight.getSubject().setName(newName);
                }
            }
        }
        private void updateObjectReferences(String oldName, String newName) {
            for (AccessRight accessRight : accessRights) {
                if (accessRight.getObject().getName().equals(oldName)) {
                    accessRight.getObject().setName(newName);
                }
            }
        }
    }
    package tools;

    import dto.MyObject;
    import dto.Subject;

    import javax.swing.*;
    import java.io.*;
    import java.util.ArrayList;
    import java.util.List;

    public class IoTools {
        private IoTools() {}
        public static void saveAccessMatrixToFile(AccessTableModel tableModel,
String fileName) {
            try (BufferedWriter writer = new BufferedWriter(new
FileWriter(fileName))) {
                List<Subject> subjects = tableModel.getSubjects();
                List<MyObject> myObjects = tableModel.getObjects();
                for (Subject subject : subjects) {
                    List<String> accessibleObjects = new ArrayList<>();
                    for (MyObject myObject : myObjects) {
                        String accessRight =
tableModel.findAccessRight(subject.getName(), myObject.getName());
                        if (!"0".equals(accessRight)) {
                            accessibleObjects.add(myObject.getName());
                        } else {
                            accessibleObjects.add(myObject.getName() + "(нет
доступа)");
                        }
                    }
                    String objectsAccess = String.join("-",
accessibleObjects);
                    writer.write(subject.getName() + ": " + objectsAccess +
"\n");
                }
```

```
                JOptionPane.showMessageDialog(null,
                        "Matrix is save in " + fileName);

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        public static void loadAccessMatrixFromFile(AccessTableModel
tableModel, String fileName) {
            try (BufferedReader reader = new BufferedReader(new
FileReader(fileName))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    String[] parts = line.split(":");
                    if (parts.length == 2) {
                        String subjectName = parts[0];
                        String[] objects = parts[1].split("-");
                        Subject subject =
tableModel.findSubjectByName(subjectName);
                        if (subject == null) {
                            subject = new Subject(subjectName);
                            tableModel.getSubjects().add(subject);
                        }

                        for (var obj : objects) {
                            String objectName = obj.replace("(нет доступа)",
"").trim();
                            MyObject myObject =
tableModel.findObjectByName(objectName);
                            if (myObject == null) {
                                myObject = new MyObject(objectName);
                                tableModel.getObjects().add(myObject);
                            }
                            if (!obj.contains("(нет доступа)")) {
                                tableModel.grantAccessRight(subject,
myObject);
                            }
                        }
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    package dto;
    import java.util.Objects;
```

```
public class AccessRight {
    private Subject subject;
    private MyObject myObject;

    public AccessRight(Subject subject, MyObject myObject) {
        this.subject = subject;
        this.myObject = myObject;
    }

    public Subject getSubject() {
        return subject;
    }

    public MyObject getObject() {
        return myObject;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        AccessRight that = (AccessRight) o;
        return subject.getName().equals(that.subject.getName())
                && myObject.getName().equals(that.myObject.getName());
    }

    @Override
    public int hashCode() {
        return Objects.hash(subject.getName(), myObject.getName());
    }
}

package dto;

public class MyObject {
    private String name;
    public MyObject(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public String toString() {return name;}
```

```
}

package dto;
public class Subject {
    private String name;
    public Subject(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public String toString() {return name;}

}
```