

1) Team 53 Group Project work allocation

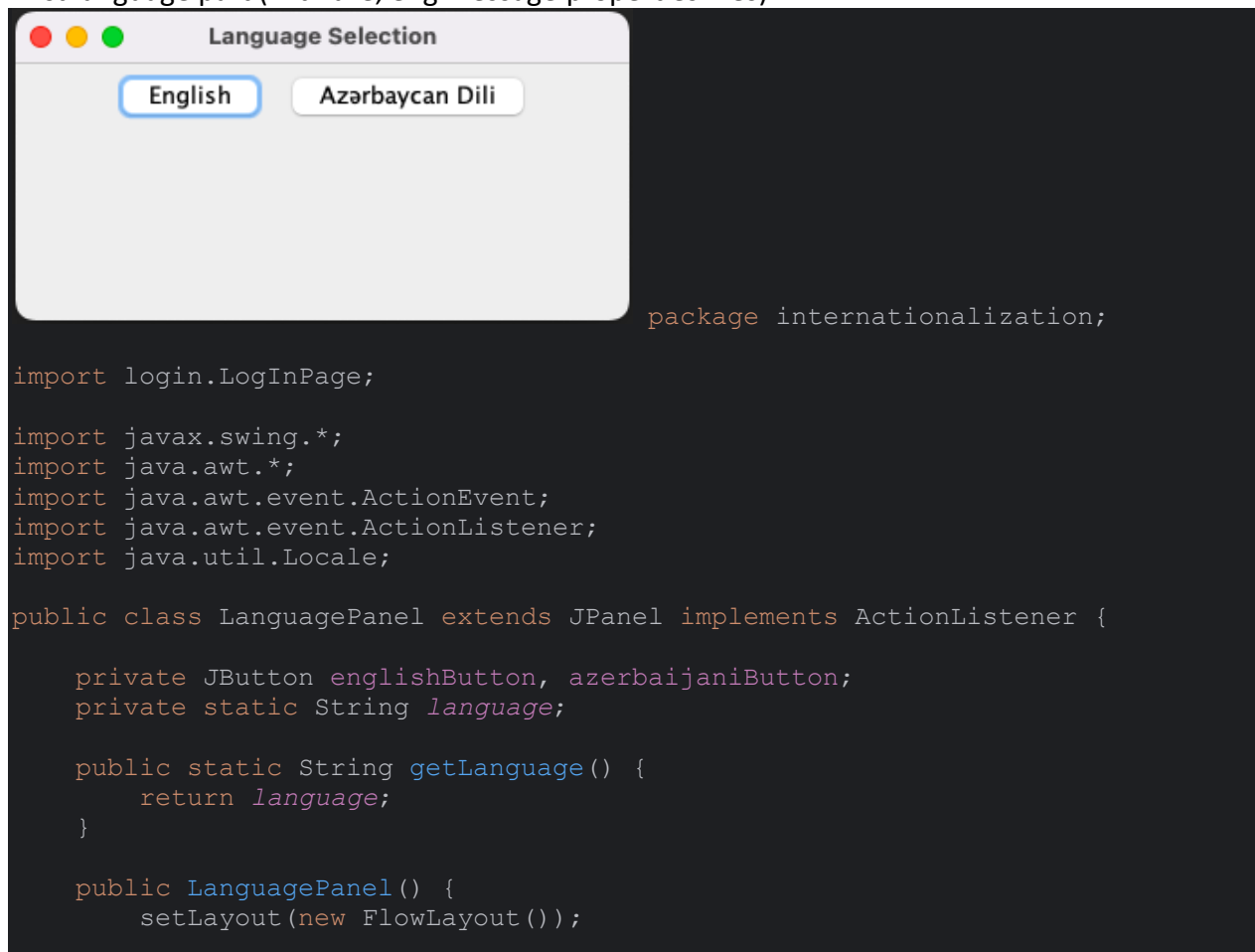
Said Mehraliyev	29%	Created Login and Registration page, added its exceptions in another package called "loginexceptions". Added sorting/filtering system. Added language choice and worked on several main bugs to fix the code.
Farid Guluzada	29%	Initial Data (General/Personal databases) - Created all General library and Personal library panels (and other panels which opens after login) with table views. Wrote main part of reading/writing. Also, some menus(selectors) to offer users switch between panels.
Khayyam Najafli	27%	Wrote mainly CRUD part, in admin and in user. Also worked on README file and github part.
Murad Bakirov	15%	Draw UML diagram, worked on search bars, Admin Panel opening, logout part and user panel for CRUD operations (such as delete edit).

2) Github link of Team 53: <https://github.com/ADA-SITE-CSCI-1202/team-project-team-53>

3) YouTube video link: <https://www.youtube.com/watch?v=eZhv7ZTdNf8>

4) UI Snippets

First language part (with aze, eng message.properties files)



```

        englishButton = new JButton("English");
        azerbaijaniButton = new JButton("Azərbaycan Dili");

        englishButton.addActionListener(this);
        azerbaijaniButton.addActionListener(this);

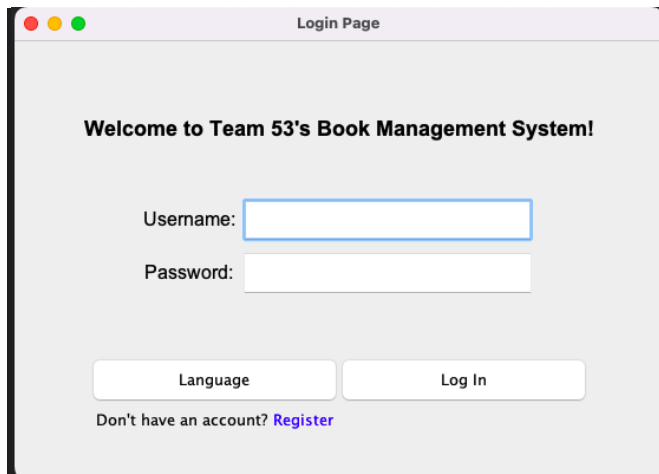
        add(englishButton);
        add(azerbaijaniButton);

        language = Locale.getDefault().getLanguage();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == englishButton) {
            language = "en";
        } else if (e.getSource() == azerbaijaniButton) {
            language = "az";
        }

        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                LoginPage loginPage = new LoginPage();
                loginPage.setVisible(true);
                ((JFrame)
SwingUtilities.getWindowAncestor(LanguagePanel.this)).dispose();
            }
        });
    }
}

```



```

private void
createNewUser(JTextField username, JPasswordField password){
    try{
        String enteredUsername = username.getText();

```

```

        String enteredPassword = new String(password.getPassword()); //
getPassword() method returns char array.
        Map<String, String> credentials =
readCredentials(CREDENTIALS_FILE);

        if (enteredUsername.length() == 0 && enteredPassword.length() ==
0){
            throw new
EmptyUsernamePasswordException(rb.getString("emptyUsernamePassword"));
        }

        else if (enteredUsername.length() < 6 ||
enteredUsername.contains(" ")){
            throw new
InvalidUsernameException(rb.getString("invalidUsername"));
        }

        else if (credentials.containsKey(enteredUsername)){
            throw new
InvalidUsernameException(rb.getString("usernameTaken"));
        }

        else if (!isValidPassword(enteredPassword)){
            throw new
InvalidPasswordException(rb.getString("invalidPassword"));
        }

        else{
            credentials.put(enteredUsername, enteredPassword);
            writeCredentials(CREDENTIALS_FILE, credentials);
            createDatabaseFile(enteredUsername);

            JOptionPane.showMessageDialog(RegistrationPage.this,
rb.getString("accountCreated"), rb.getString("successTitle"),
JOptionPane.INFORMATION_MESSAGE);
        }
    }
    catch (EmptyUsernamePasswordException e){
        JOptionPane.showMessageDialog(RegistrationPage.this,
e.getMessage(), rb.getString("errorTitle"), JOptionPane.ERROR_MESSAGE);
    }
    catch (InvalidUsernameException e){
        JOptionPane.showMessageDialog(RegistrationPage.this,
e.getMessage(), rb.getString("errorTitle"), JOptionPane.ERROR_MESSAGE);
    }
    catch (InvalidPasswordException e){
        JOptionPane.showMessageDialog(RegistrationPage.this,
e.getMessage(), rb.getString("errorTitle"), JOptionPane.ERROR_MESSAGE);
    }
    finally{
        username.setText("");
        password.setText("");
    }
}

private boolean isValidPassword(String enteredPassword){
    boolean hasUppercase = false;

```

```

        boolean hasLowercase = false;
        boolean hasDigit = false;
        boolean hasSpecialCharacter = false;
        boolean isLong = false;
        boolean hasNoSpace = true;
        String specialCharacters = "~`!@#$%^&*()_+-={}[]\\|:;\"'/?/<>,.\";

        if (enteredPassword.length() >= 8) {
            isLong = true;
        }

        for (char c : enteredPassword.toCharArray()) {
            if (Character.isUpperCase(c)) {
                hasUppercase = true;
            }
            else if (Character.isLowerCase(c)) {
                hasLowercase = true;
            }
            else if (Character.isDigit(c)) {
                hasDigit = true;
            }
            else if (specialCharacters.contains(String.valueOf(c))) {
                hasSpecialCharacter = true;
            }
            else if (Character.isWhitespace(c)) {
                hasNoSpace = false;
            }
        }

        return hasUppercase && hasLowercase && hasDigit &&
hasSpecialCharacter && isLong && hasNoSpace;
    }

    public static Map<String, String> readCredentials(String filename) {
        Map<String, String> credentials = new HashMap<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] user = line.split(",");
                credentials.put(user[0], user[1]);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return credentials;
    }

    private void createDatabaseFile(String username) {
        String userDatabaseFile = CREDENTIALS_DIRECTORY + username + ".csv";
        File file = new File(userDatabaseFile);
        try {
            if (file.createNewFile()) {
                System.out.println("Database file created: " +
file.getName());
            } else {
                System.out.println("Database file already exists.");
            }
        }
    }

```

```

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void writeCredentials(String filename, Map<String, String>
credentials) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename))) {
            for (String username : credentials.keySet()) {
                String password = credentials.get(username);
                writer.write(username + "," + password);
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

```

private void
validateAndProcessLogin() {
    try {
        String enteredUsername = username.getText();
        String enteredPassword = new String(password.getPassword());
        Map<String, String> credentials =
RegistrationPage.readCredentials(CREDENTIALS_FILE);

        if (enteredUsername.equals("admin") &&
enteredPassword.equals("admin")) {
            openAdminPanel();
        }

        else if (!(credentials.containsKey(enteredUsername) &&
credentials.get(enteredUsername).equals(enteredPassword))) {
            throw new InvalidCredentials(rb.getString("invalidCredentials"));
        }
    }
}

```

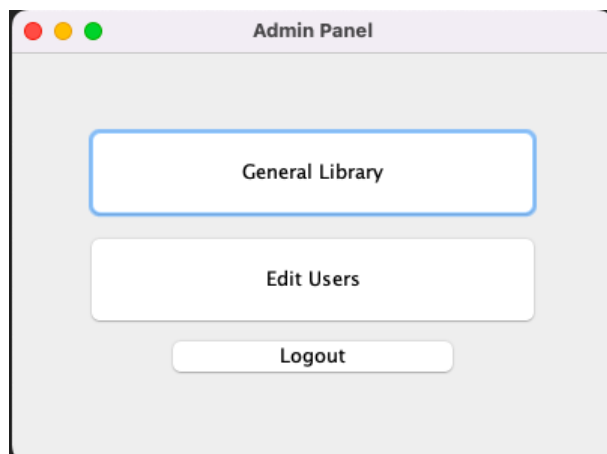
```

    }

    else{
        LogInPage.setUserFileName(enteredUsername);
        openSecondPanel();
    }

} catch (InvalidCredentials ex) {
    JOptionPane.showMessageDialog(LogInPage.this, ex.getMessage(),
rb.getString("errorTitle"), JOptionPane.ERROR_MESSAGE);
}
finally{
    username.setText("");
    password.setText("");
}
}
}

```



This is admin's general library panel which admin can add or remove book or give rating and review.



Edit users panel allows admin to delete user.



The screenshot shows a web application titled "Admin Panel". Inside, there's a section labeled "Edit Users" with a green plus icon. Below it is a table with two columns: "Username" and "Password". The table contains one row with the values "farid2005" and "afz2005Z5". At the bottom of the panel, there is a button labeled "Delete User".

```
private void modelUsers() {
    String[] columnNames = {rb.getString("username"),
rb.getString("password")};
    DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0);

    try (BufferedReader br = new BufferedReader(new
FileReader(credentialsFile))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] data = line.split(",");
            if (data.length >= 2) {
                tableModel.addRow(data);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    userTable = new JTable(tableModel);
    JScrollPane scrollPane = new JScrollPane(userTable);
    add(scrollPane, BorderLayout.CENTER);

    JButton deleteButton = new JButton(rb.getString("deleteUser"));
    deleteButton.addActionListener(e -> deleteSelectedUser());
    add(deleteButton, BorderLayout.SOUTH);
}

private void deleteSelectedUser() {
    int row = userTable.getSelectedRow();
    if (row >= 0) {
        ((DefaultTableModel) userTable.getModel()).removeRow(row);
        saveChangesToFile();
    }
}

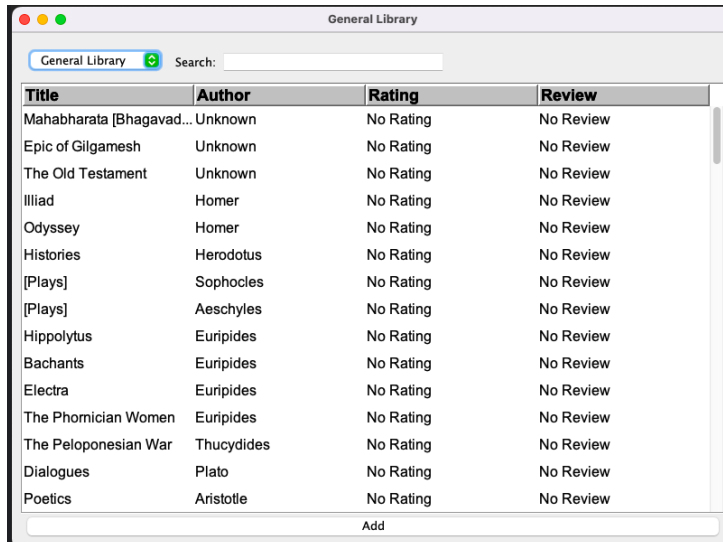
private void saveChangesToFile() {
    try (PrintWriter pw = new PrintWriter(new FileWriter(credentialsFile,
false))) {
        DefaultTableModel model = (DefaultTableModel)
```

```

userTable.getModel();
    for (int i = 0; i < model.getRowCount(); i++) {
        Object username = model.getValueAt(i, 0);
        Object password = model.getValueAt(i, 1);
        pw.println(username + "," + password);
    }
} catch (IOException e) {
    JOptionPane.showMessageDialog(this,
rb.getString("errorUpdatingFile"), rb.getString("errorTitle"),
JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
}
}
}

```

With mainly reading and writing and all others are about swing UI. It firstly reads from original brodsk.csv and then it will work on general.csv



Title	Author	Rating	Review
Mahabharata [Bhagavad...	Unknown	No Rating	No Review
Epic of Gilgamesh	Unknown	No Rating	No Review
The Old Testament	Unknown	No Rating	No Review
Illiad	Homer	No Rating	No Review
Odyssey	Homer	No Rating	No Review
Histories	Herodotus	No Rating	No Review
[Plays]	Sophocles	No Rating	No Review
[Plays]	Aeschyles	No Rating	No Review
Hippolytus	Euripides	No Rating	No Review
Bachants	Euripides	No Rating	No Review
Electra	Euripides	No Rating	No Review
The Phornician Women	Euripides	No Rating	No Review
The Peloponesian War	Thucydides	No Rating	No Review
Dialogues	Plato	No Rating	No Review
Poetics	Aristotle	No Rating	No Review

```

package GeneralDatabase;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class ReadCsv {
    public static List<Book> readCSV(final String filename) throws
IOException {
        List<Book> books = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new
FileReader(filename))) {
            String line;

```

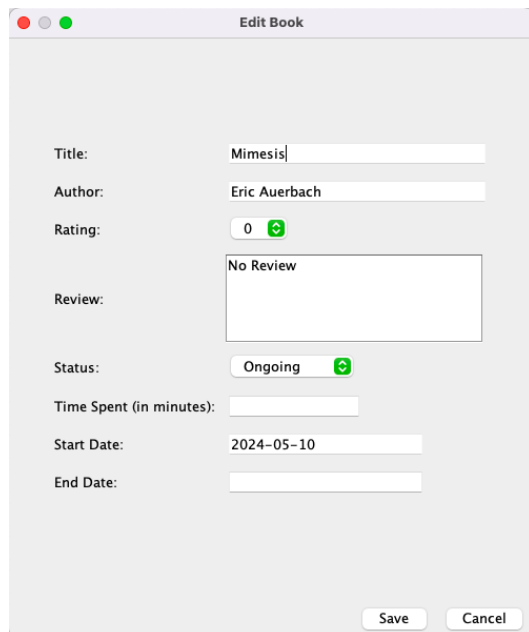


```

        reader.readLine();
        while ((line = reader.readLine()) != null) {
            String[] data =
line.split(", (?=(?:[^\"]*" * \"[^\"]*" * \") * [^\"]*$)");
            String[] titles = data[0].trim().split("\"");
            List<String> bookTitles = new ArrayList<>();
            if (titles.length > 1) {
                String[] splitTitles = titles[1].split(",");
                for (String splitTitle : splitTitles) {
                    bookTitles.add(splitTitle.trim());
                }
            } else {
                bookTitles.add(titles[0].trim());
            }
            String author = data.length == 2 ? data[1].trim() : "";
            books.add(new Book(bookTitles, author));
        }
    }
    return books;
}
}

```

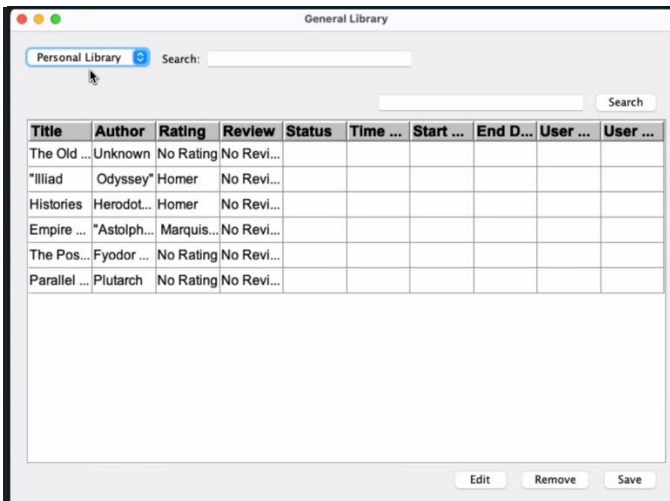
Edit Book is mostly UI and CRUD operations



The screenshot shows a macOS-style window titled "Edit Book". The window contains a form with the following fields and controls:

- Title:** A text input field containing "Mimesis".
- Author:** A text input field containing "Eric Auerbach".
- Rating:** A numeric input field with the value "0" and a green up/down arrow icon.
- Review:** A text area containing the text "No Review".
- Status:** A dropdown menu showing "Ongoing" with a green up/down arrow icon.
- Time Spent (in minutes):** An empty text input field.
- Start Date:** A date input field containing "2024-05-10".
- End Date:** An empty date input field.

At the bottom right of the window, there are two buttons: "Save" and "Cancel".



```

public PersonalBookPanel(String
userCsvFile) {
    currentLocale = new Locale(LanguagePanel.getLanguage());
    rb = ResourceBundle.getBundle("internationalization/messages",
currentLocale);
    this.userCsvFile = userCsvFile;

    model = new DefaultTableModel() {
        @Override
        public boolean isCellEditable(int row, int column) {
            // Allow editing for all columns except for Title, Author,
Rating, and Status
            if (column == 3) { // Review column
                // Allow editing if the current value is "No Review"
                return getValueAt(row,
column).toString().equals(rb.getString("noReview"));
            } else {
                return column != 0 && column != 1 && column != 2 && column !=
4;
            }
        }
    };

    model.addColumn(rb.getString("title"));
    model.addColumn(rb.getString("author"));
    model.addColumn(rb.getString("rating"));
    model.addColumn(rb.getString("review"));
    model.addColumn(rb.getString("status"));
    model.addColumn(rb.getString("timeSpent"));
    model.addColumn(rb.getString("startDate"));
    model.addColumn(rb.getString("endDate"));
    model.addColumn(rb.getString("userRating"));
    model.addColumn(rb.getString("userReview"));

    loadData(LoginPage.getUserFileName());
    JButton editButton = new JButton("Edit");
    editButton.addActionListener(e -> {
        int selectedRow = table.getSelectedRow();
        if (selectedRow != -1) {
            String title = (String) model.getValueAt(selectedRow, 0);
            String author = (String) model.getValueAt(selectedRow, 1);

```

```

String ratingStr = (String) model.getValueAt(selectedRow, 2);
String review = (String) model.getValueAt(selectedRow, 3);
int rating = 0;
if (!ratingStr.equals("No Rating")) {
    try {
        rating = Integer.parseInt(ratingStr);
    } catch (NumberFormatException ex) {
        System.err.println("Error parsing rating: " +
ex.getMessage());
    }
}

EditBookWindow editWindow = new EditBookWindow(title, author,
ratingStr, review, review); // Modified here
editWindow.setVisible(true);
}
});
private void loadData(String userCsvFile) {
    try (Scanner scanner = new Scanner(new File(userCsvFile))) {

        if (scanner.hasNextLine()) {
            scanner.nextLine();
        }
        while (scanner.hasNextLine()) {
            String line = scanner.nextLine();
            String[] data = line.split(",");
            String[] newData = new String[model.getColumnCount()];
            for (int i = 0; i < data.length; i++) {
                if (data[i].equals("null")) {
                    newData[i] = ""; // Replace "null" with an empty string
                } else {
                    newData[i] = data[i];
                }
            }
            // Ensure all columns have data
            for (int i = data.length; i < model.getColumnCount(); i++) {
                newData[i] = "";
            }
            // Check if author field is empty, replace with "Unknown"
            if (newData.length > 1 && newData[1].isEmpty()) {
                newData[1] = rb.getString("unknown");
            }
            // Check if rating field is empty, replace with "No Rating"
            if (newData.length > 2 && newData[2].isEmpty()) {
                newData[2] = rb.getString("noRating");
            }
            // Check if review field is empty, replace with "No Review"
            if (newData.length > 3 && newData[3].isEmpty()) {
                newData[3] = rb.getString("noReview");
            }
            // Check if status field contains text other than "Completed",
            "Ongoing", and "Not Started",
            // replace with an empty string
            if (newData.length > 4 &&
!newData[4].equals(rb.getString("completed")) &&
!newData[4].equals(rb.getString("ongoing")) &&
!newData[4].equals(rb.getString("notStarted"))) {

```

```

        newData[4] = "";
    }
    model.addRow(newData);
}
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
}

private void saveToCSV(String userCsvFile) {
    System.out.println(userCsvFile);
    try (FileWriter writer = new FileWriter(new File(userCsvFile))) { //
        Overwrite mode
        for (int row = 0; row < model.getRowCount(); row++) {
            StringBuilder line = new StringBuilder();
            for (int col = 0; col < model.getColumnCount(); col++) {
                line.append(model.getValueAt(row, col));
                if (col < model.getColumnCount() - 1) {
                    line.append(",");
                }
            }
            writer.write(line.toString() + "\n");
        }
        JOptionPane.showMessageDialog(this, rb.getString("saveSuccessful") +
            userCsvFile, rb.getString("saveSuccessfulTitle"),
            JOptionPane.INFORMATION_MESSAGE);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(this, rb.getString("saveError") +
            userCsvFile, rb.getString("saveErrorTitle"), JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}
}

```