Final Project Report

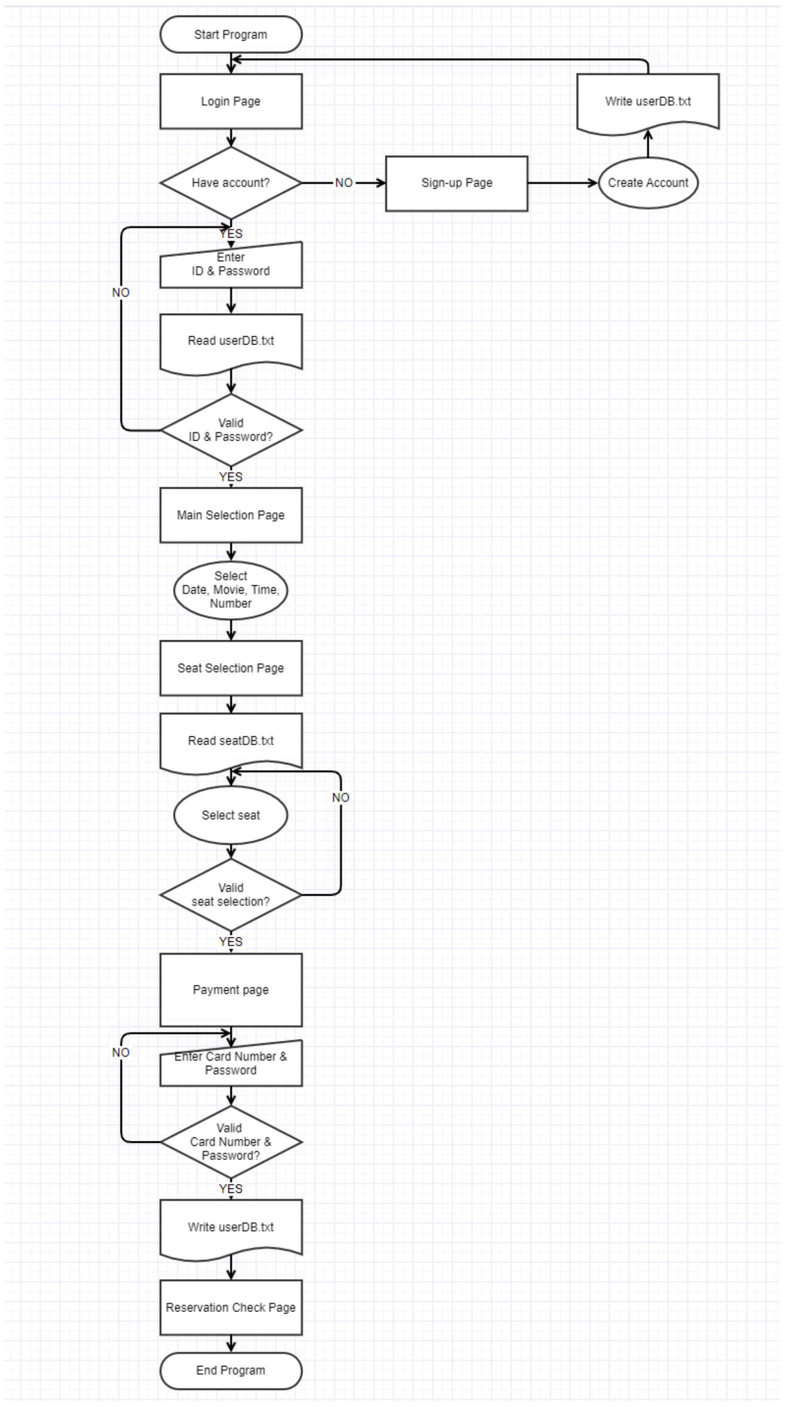
Project Title : Movie Reservation Program

Student Name: Kim Ho Jin
Student ID: 2016314786

1. Briefly describe the project purpose:

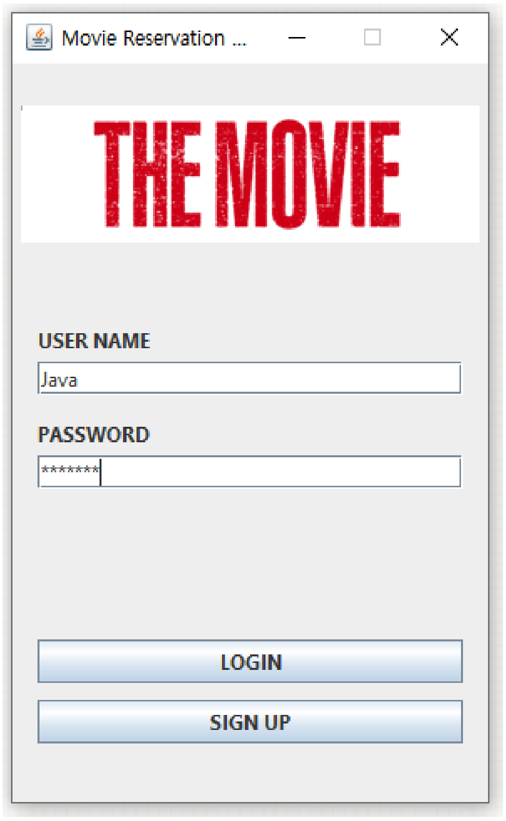
The on-site reservation system is a time consuming task for eveyone and there is a risk of COVID-19 infection. In order to save everyone's time and increase convenience, I created Movie Reservation Program. Using this program, user can check the list and time of screening movies for a week. The program also provides simple information (ex. rating, number of audience...) about movies. The user may select a movie at desired time and then select a seat according to the number of people. After the reservation is confirmed, the mobile movie ticket will be sent to the user's smartphone.

2. Draw the logic flow of the program (with flowchart):



3. Provide screenshots for each screen with brief description:

I. Login Page

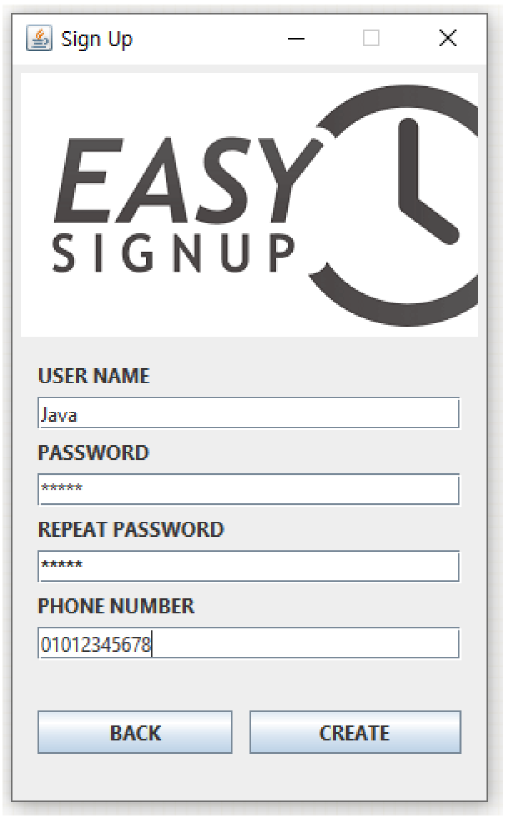


- It implements the program's login page.

- After checking file 'userDB.txt', It determines whether the ID and password entered from the user are correct.

- For security purpose, convert the input password value to * mark.

II. Sign-up page



- It implements the program's sign-up page.

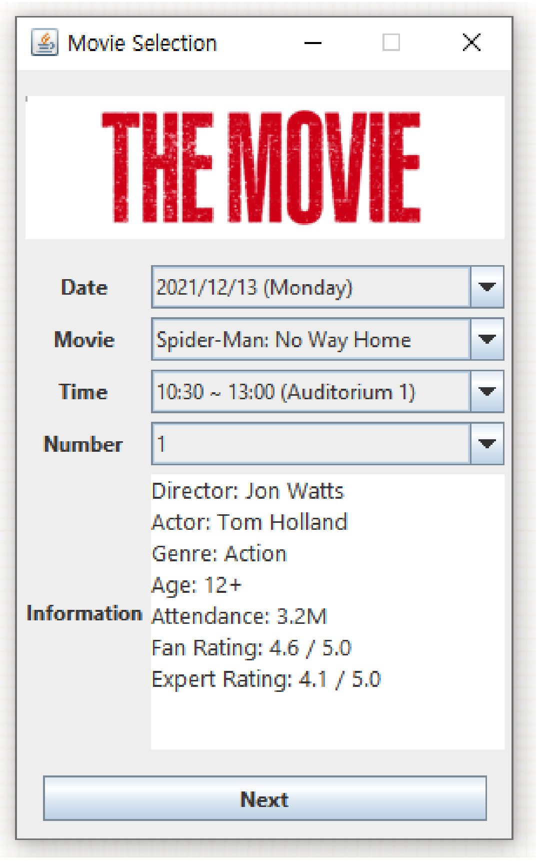
- It creates a new account with a valid ID and password.

- If user have successfully created an account, write the account information in the file 'userDB.txt'.

- It generates messages for unsuitable input.

- For security purpose, convert the input password and repeat password value to * mark.

III. Movie Selection Page



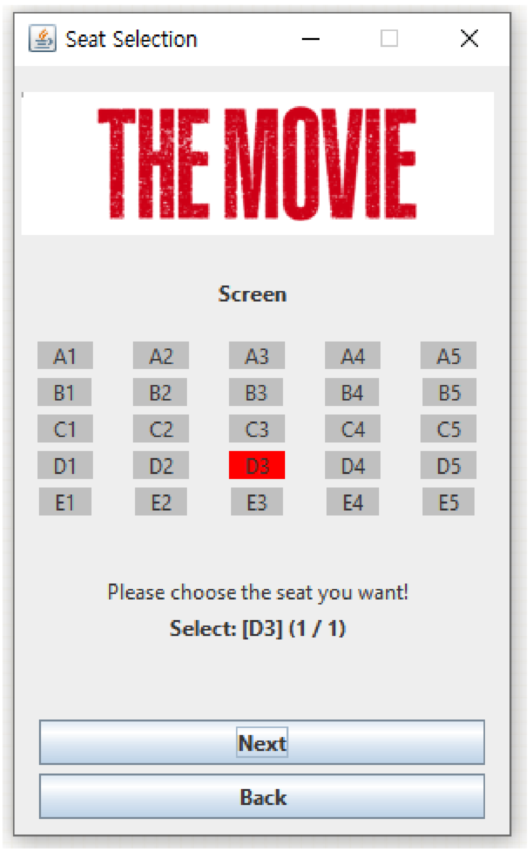
Date: 2021/12/13 (Monday)
Movie: Spider-Man: No Way Home
Time: 10:30 ~ 13:00 (Auditorium 1)
Number: 1
Information:
Director: Jon Watts
Actor: Tom Holland
Genre: Action
Age: 12+
Attendance: 3.2M
Fan Rating: 4.6 / 5.0
Expert Rating: 4.1 / 5.0

- It implements the program's main selection page.

- User selects date, movie, time and number of people in this page.

- The program provides brief movie information (Director, Actor, Genre, Age, Attendance, Rating) that user selected.

IV. Seat Selection Page



Please choose the seat you want!
Select: [D3] (1 / 1)

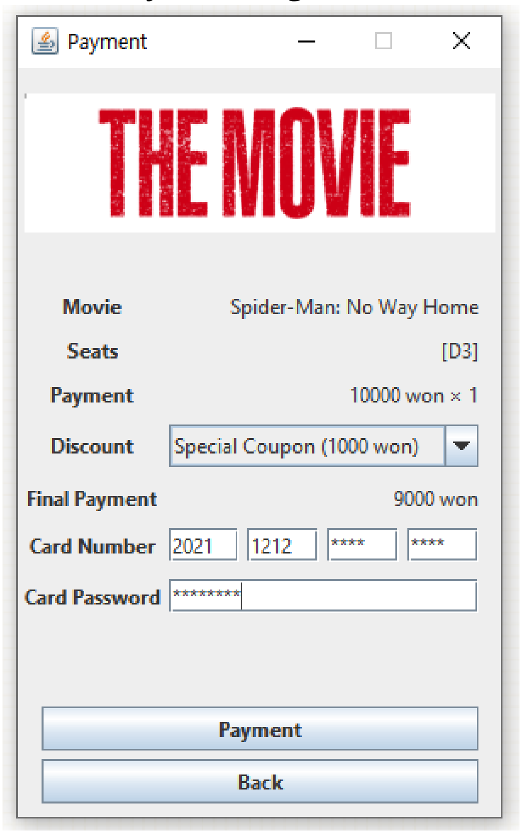
- It implements the program's seat selection page.

- The program identifies the status of seat reservations through file 'seatDB.txt' and provides it to user.

- The user selects seats according to the number of people based on the provided information.

- The seat selected by the user is reflected in real time.

V. Payment Page



Movie	Spider-Man: No Way Home
Seats	[D3]
Payment	10000 won × 1
Discount	Special Coupon (1000 won)
Final Payment	9000 won
Card Number	2021 1212 **** ****
Card Password	********

- It implements the program's payment page.

- The program provides the user with a price for the selected movie.

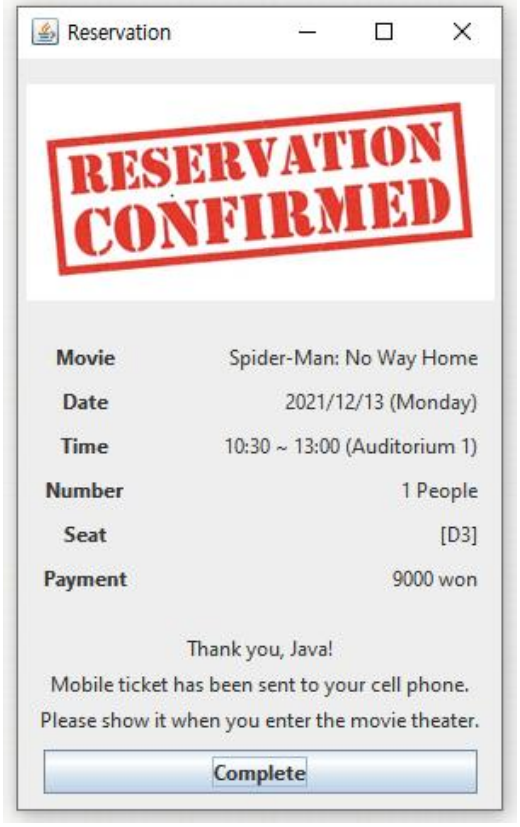
- The final amount depends on the discount option user select.

- The user enters the card number and password to proceed with the payment.

- For security purpose, convert the input card number and password value to * mark.

- It generates messages for unsuitable input.

VI. Reservation Confirm Page



Movie	Spider-Man: No Way Home
Date	2021/12/13 (Monday)
Time	10:30 ~ 13:00 (Auditorium 1)
Number	1 People
Seat	[D3]
Payment	9000 won

Thank you, Java!
Mobile ticket has been sent to your cell phone.
Please show it when you enter the movie theater.

- It implements the program's reservation check page.

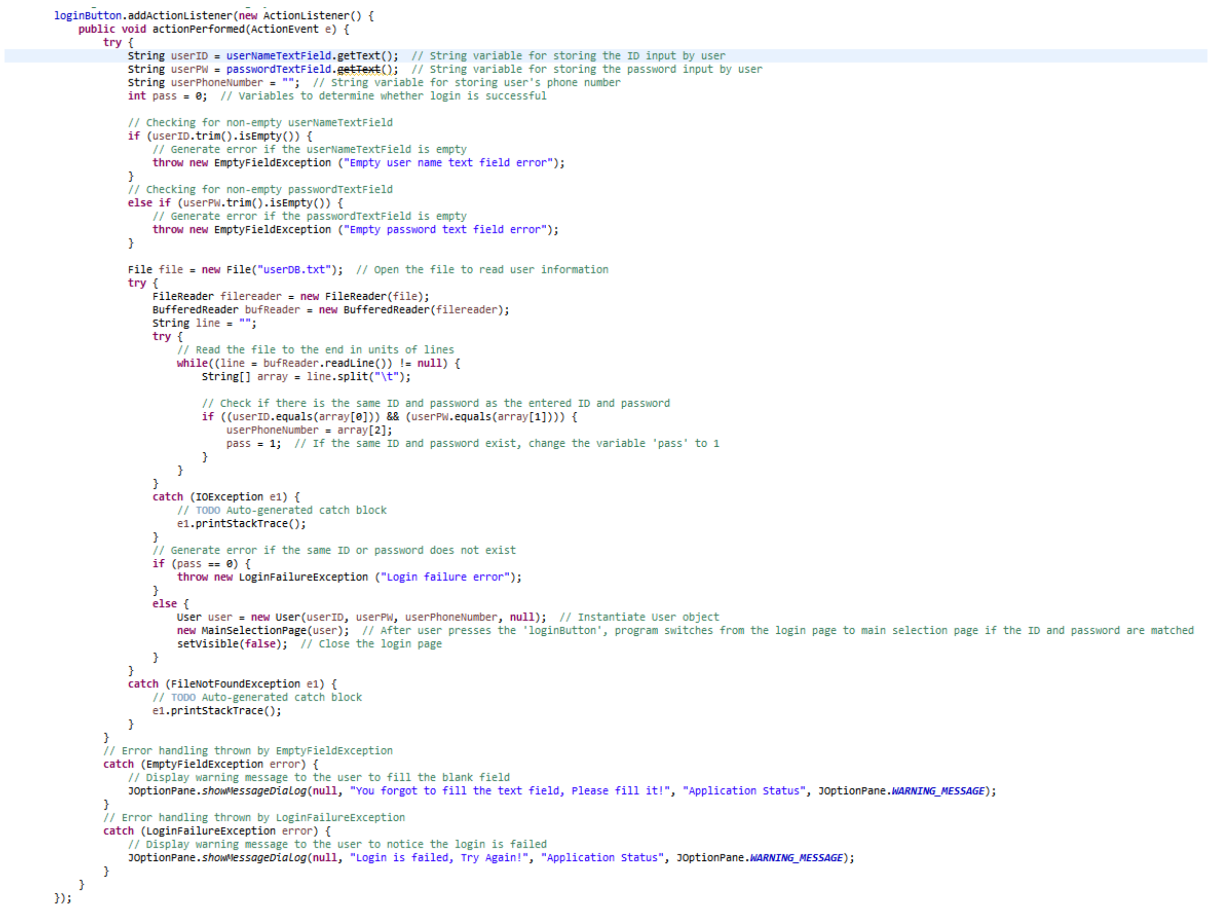
- The program provides the user with overall information on confirmed reservations.

- This allows the user to check whether the reservation has been successful.

- After the reservation is confirmed, mobile movie tickets are sent to the user's smartphone that was used when signing up for membership.

4. Explain the code of the main functionalities

I. Check the validation of login



```
loginButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            String userID = userNameTextField.getText();  // String variable for storing the ID input by user
            String userPW = passwordTextField.~~getText~~();  // String variable for storing the password input by user
            String userPhoneNumber = "";  // String variable for storing user's phone number
            int pass = 0;  // Variables to determine whether login is successful

            // Checking for non-empty userNameTextField
            if (userID.trim().isEmpty()) {
                // Generate error if the userNameTextField is empty
                throw new EmptyFieldException ("Empty user name text field error");
            }
            // Checking for non-empty passwordTextField
            else if (userPW.trim().isEmpty()) {
                // Generate error if the passwordTextField is empty
                throw new EmptyFieldException ("Empty password text field error");
            }

            File file = new File("userDB.txt");  // Open the file to read user information
            try {
                FileReader filereader = new FileReader(file);
                BufferedReader bufReader = new BufferedReader(filereader);
                String line = "";
                try {
                    // Read the file to the end in units of lines
                    while((line = bufReader.readLine()) != null) {
                        String[] array = line.split("\t");

                        // Check if there is the same ID and password as the entered ID and password
                        if ((userID.equals(array[0])) && (userPW.equals(array[1]))) {
                            userPhoneNumber = array[2];
                            pass = 1;  // If the same ID and password exist, change the variable 'pass' to 1
                        }
                    }
                }
                catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                // Generate error if the same ID or password does not exist
                if (pass == 0) {
                    throw new LoginFailureException ("Login failure error");
                }
                else {
                    User user = new User(userID, userPW, userPhoneNumber, null);  // Instantiate User object
                    new MainSelectionPage(user);  // After user presses the 'loginButton', program switches from the login page to main selection page if the ID and password are matched
                    setVisible(false);  // Close the login page
                }
            }
            catch (FileNotFoundException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
        // Error handling thrown by EmptyFieldException
        catch (EmptyFieldException error) {
            // Display warning message to the user to fill the blank field
            JOptionPane.showMessageDialog(null, "You forgot to fill the text field, Please fill it!", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
        // Error handling thrown by LoginFailureException
        catch (LoginFailureException error) {
            // Display warning message to the user to notice the login is failed
            JOptionPane.showMessageDialog(null, "Login is failed, Try Again!", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
    }
});
```

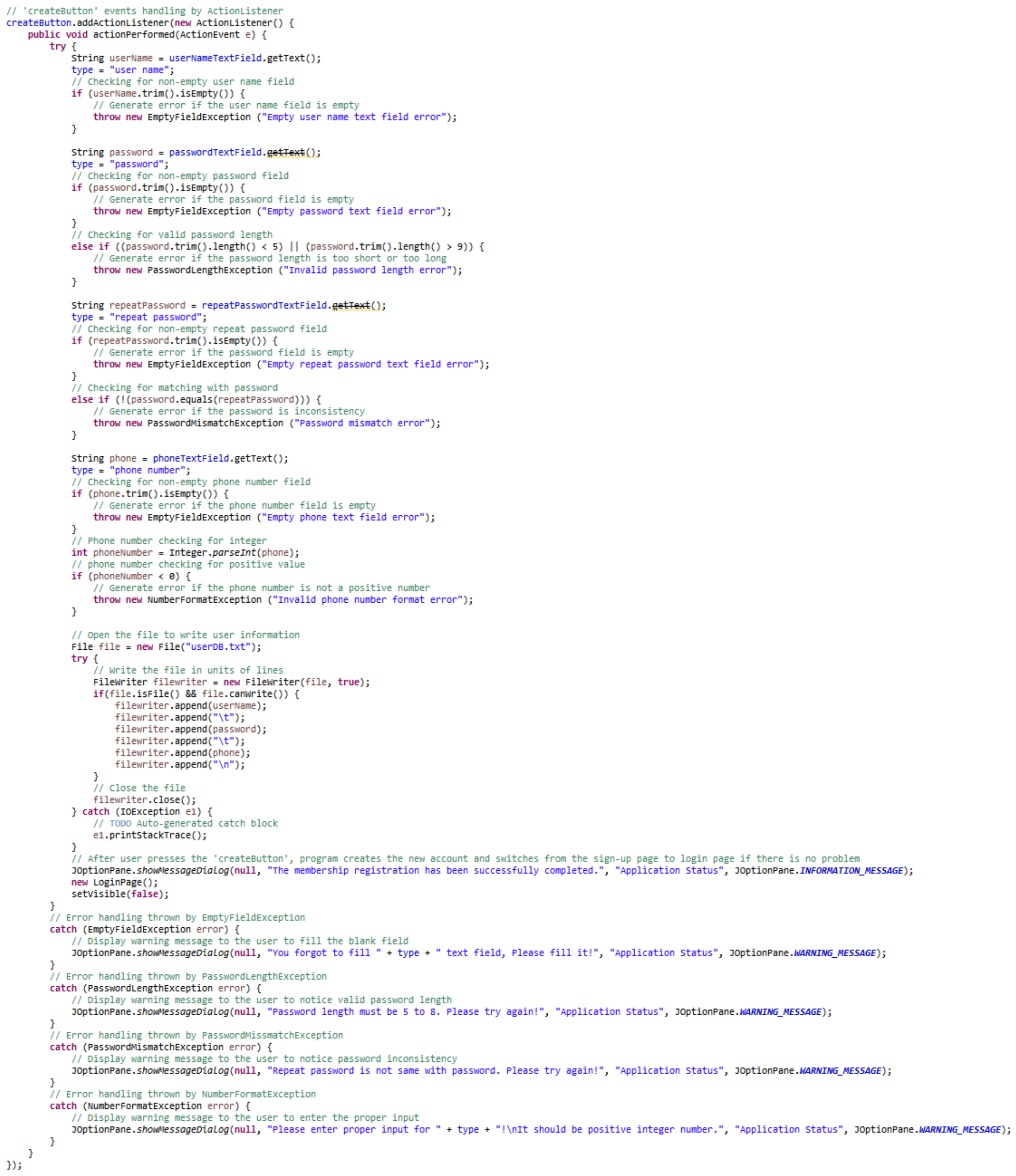
- First, make sure that the ID and password are not blank.

- Open the file 'userDB.txt' to see if there is the same account as the ID and password entered from the user.

- If the same account exists, proceed with the program normally.

- If the same account does not exist, display the warning message and ask for re-entry

II. Creates a new account with a valid ID and password



```java
// 'createButton' events handling by ActionListener
createButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            String userName = userNameTextField.getText();
            type = "user name";
            // Checking for non-empty user name field
            if (userName.trim().isEmpty()) {
                // Generate error if the user name field is empty
                throw new EmptyFieldException ("Empty user name text field error");
            }

            String password = passwordTextField.~~getText~~();
            type = "password";
            // Checking for non-empty password field
            if (password.trim().isEmpty()) {
                // Generate error if the password field is empty
                throw new EmptyFieldException ("Empty password text field error");
            }
            // Checking for valid password length
            else if ((password.trim().length() < 5) || (password.trim().length() > 9)) {
                // Generate error if the password length is too short or too long
                throw new PasswordLengthException ("Invalid password length error");
            }

            String repeatPassword = repeatPasswordTextField.~~getText~~();
            type = "repeat password";
            // Checking for non-empty repeat password field
            if (repeatPassword.trim().isEmpty()) {
                // Generate error if the password field is empty
                throw new EmptyFieldException ("Empty repeat password text field error");
            }
            // Checking for matching with password
            else if (!(password.equals(repeatPassword))) {
                // Generate error if the password is inconsistency
                throw new PasswordMismatchException ("Password mismatch error");
            }

            String phone = phoneTextField.getText();
            type = "phone number";
            // Checking for non-empty phone number field
            if (phone.trim().isEmpty()) {
                // Generate error if the phone number field is empty
                throw new EmptyFieldException ("Empty phone text field error");
            }
            // Phone number checking for integer
            int phoneNumber = Integer.parseInt(phone);
            // phone number checking for positive value
            if (phoneNumber < 0) {
                // Generate error if the phone number is not a positive number
                throw new NumberFormatException ("Invalid phone number format error");
            }

            // Open the file to write user information
            File file = new File("userDB.txt");
            try {
                // Write the file in units of lines
                FileWriter filewriter = new FileWriter(file, true);
                if(file.isFile() && file.canWrite()) {
                    filewriter.append(userName);
                    filewriter.append("\t");
                    filewriter.append(password);
                    filewriter.append("\t");
                    filewriter.append(phone);
                    filewriter.append("\n");
                }
                // Close the file
                filewriter.close();
            } catch (IOException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            // After user presses the 'createButton', program creates the new account and switches from the sign-up page to login page if there is no problem
            JOptionPane.showMessageDialog(null, "The membership registration has been successfully completed.", "Application Status", JOptionPane.INFORMATION_MESSAGE);
            new LoginPage();
            setVisible(false);
        }
        // Error handling thrown by EmptyFieldException
        catch (EmptyFieldException error) {
            // Display warning message to the user to fill the blank field
            JOptionPane.showMessageDialog(null, "You forgot to fill " + type + " text field, Please fill it!", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
        // Error handling thrown by PasswordLengthException
        catch (PasswordLengthException error) {
            // Display warning message to the user to notice valid password length
            JOptionPane.showMessageDialog(null, "Password length must be 5 to 8. Please try again!", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
        // Error handling thrown by PasswordMissmatchException
        catch (PasswordMismatchException error) {
            // Display warning message to the user to notice password inconsistency
            JOptionPane.showMessageDialog(null, "Repeat password is not same with password. Please try again!", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
        // Error handling thrown by NumberFormatException
        catch (NumberFormatException error) {
            // Display warning message to the user to enter the proper input
            JOptionPane.showMessageDialog(null, "Please enter proper input for " + type + "!\nIt should be positive integer number.", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
    }
});
```

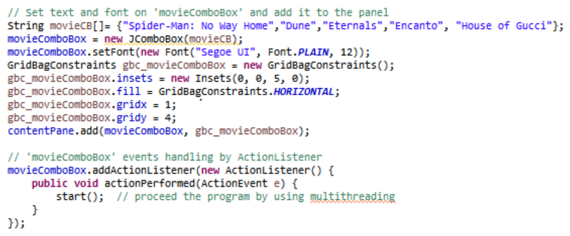
- First, check whether the entered values are valid.

- If all the entered values are valid, create an account and write the information in the file 'userDB.txt'.

- If there exists invalid input value, display the warning message and ask for re-entry

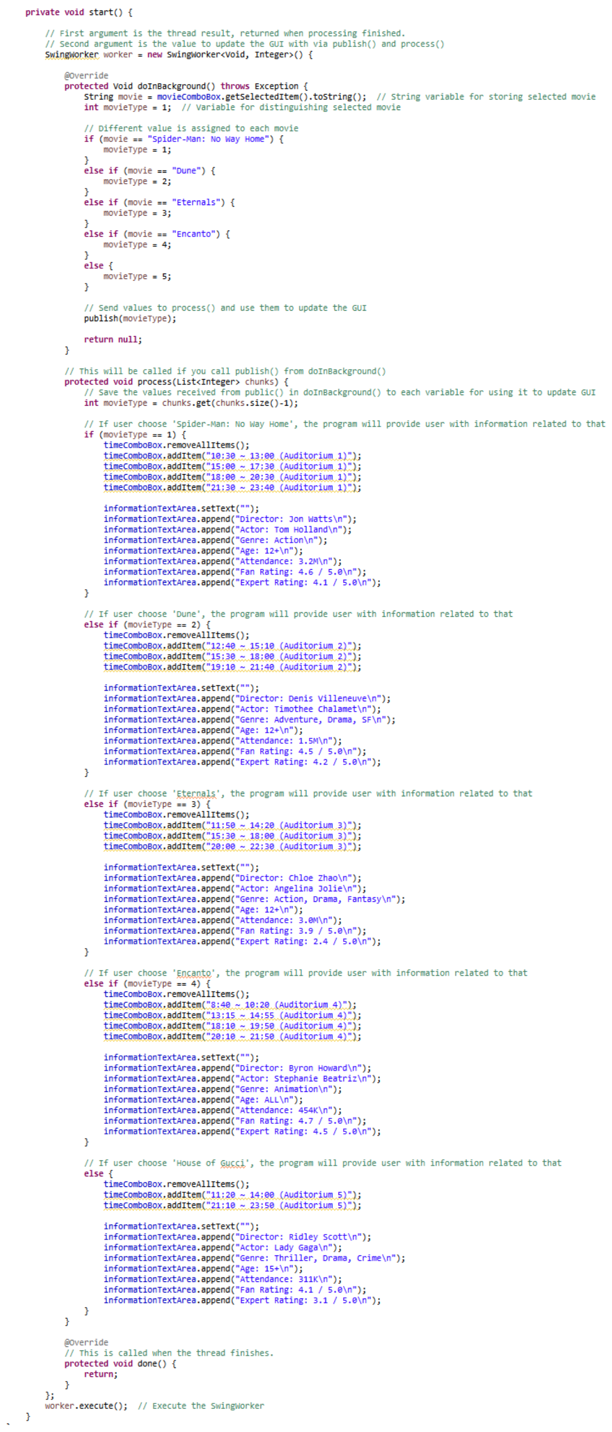
III. Provide the time and information of movie that user select

- When the user selects the movie, program proceeded by multithreading.



```java
// Set text and font on 'movieComboBox' and add it to the panel
String movieCB[]= {"Spider-Man: No Way Home","Dune","Eternals","Encanto", "House of Gucci"};
movieComboBox = new JComboBox(movieCB);
movieComboBox.setFont(new Font("Segoe UI", Font.PLAIN, 12));
GridBagConstraints gbc_movieComboBox = new GridBagConstraints();
gbc_movieComboBox.insets = new Insets(0, 0, 5, 0);
gbc_movieComboBox.fill = GridBagConstraints.HORIZONTAL;
gbc_movieComboBox.gridx = 1;
gbc_movieComboBox.gridy = 4;
contentPane.add(movieComboBox, gbc_movieComboBox);

// 'movieComboBox' events handling by ActionListener
movieComboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        start();  // proceed the program by using multithreading
    }
});
```

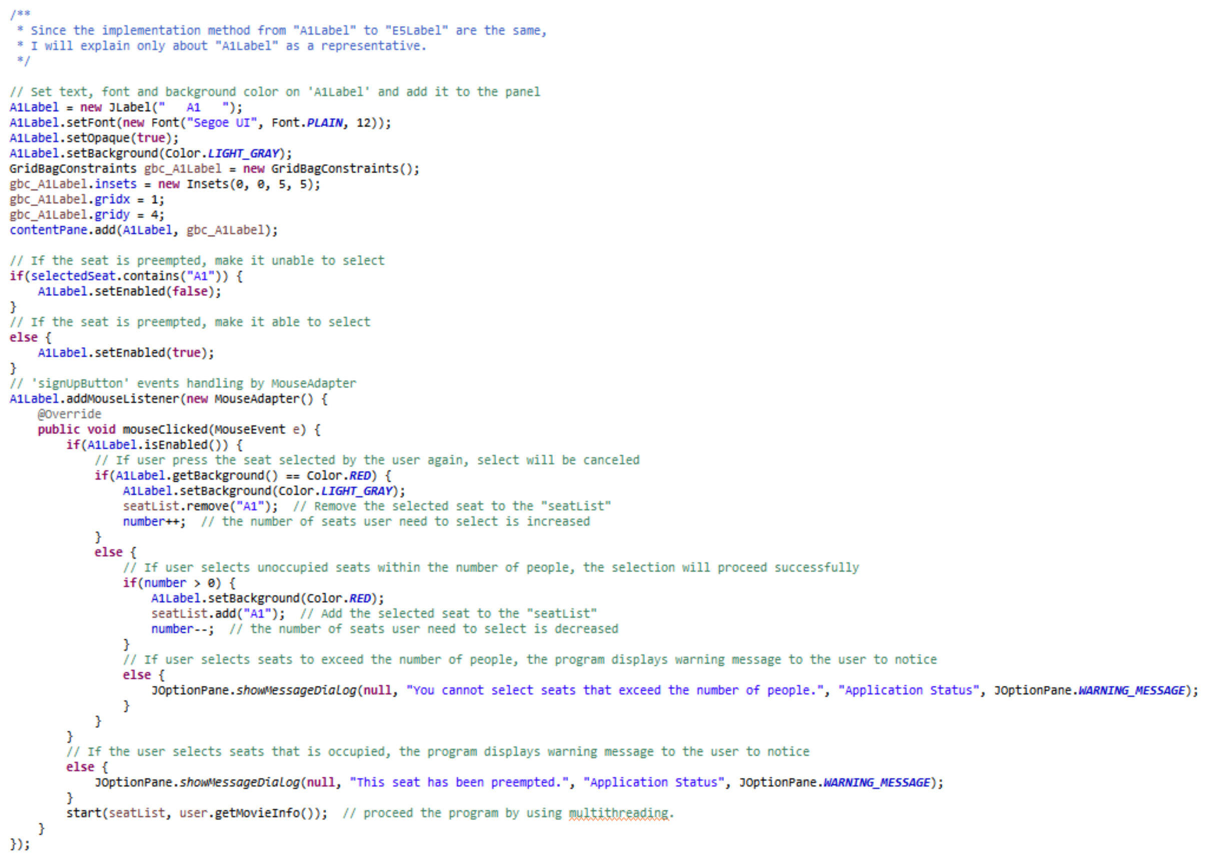
- Through multithreading, users can be provided with the screening time and simple information of the selected movie.



```java
private void start() {

    // First argument is the thread result, returned when processing finished.
    // Second argument is the value to update the GUI with via publish() and process()
    SwingWorker worker = new SwingWorker<Void, Integer>() {

        @Override
        protected Void doInBackground() throws Exception {
            String movie = movieComboBox.getSelectedItem().toString();  // String variable for storing selected movie
            int movieType = 1;  // Variable for distinguishing selected movie

            // Different value is assigned to each movie
            if (movie == "Spider-Man: No Way Home") {
                movieType = 1;
            }
            else if (movie == "Dune") {
                movieType = 2;
            }
            else if (movie == "Eternals") {
                movieType = 3;
            }
            else if (movie == "Encanto") {
                movieType = 4;
            }
            else {
                movieType = 5;
            }

            // Send values to process() and use them to update the GUI
            publish(movieType);

            return null;
        }

        // This will be called if you call publish() from doInBackground()
        protected void process(List<Integer> chunks) {
            // Save the values received from public() in doInBackground() to each variable for using it to update GUI
            int movieType = chunks.get(chunks.size()-1);

            // If user choose 'Spider-Man: No Way Home', the program will provide user with information related to that
            if (movieType == 1) {
                timeComboBox.removeAllItems();
                timeComboBox.addItem("10:30 ~ 13:00 (Auditorium 1)");
                timeComboBox.addItem("15:00 ~ 17:30 (Auditorium 1)");
                timeComboBox.addItem("18:00 ~ 20:30 (Auditorium 1)");
                timeComboBox.addItem("21:30 ~ 23:40 (Auditorium 1)");

                informationTextArea.setText("");
                informationTextArea.append("Director: Jon Watts\n");
                informationTextArea.append("Actor: Tom Holland\n");
                informationTextArea.append("Genre: Action\n");
                informationTextArea.append("Age: 12+\n");
                informationTextArea.append("Attendance: 3.2M\n");
                informationTextArea.append("Fan Rating: 4.6 / 5.0\n");
                informationTextArea.append("Expert Rating: 4.1 / 5.0\n");
            }

            // If user choose 'Dune', the program will provide user with information related to that
            else if (movieType == 2) {
                timeComboBox.removeAllItems();
                timeComboBox.addItem("12:40 ~ 15:10 (Auditorium 2)");
                timeComboBox.addItem("15:30 ~ 18:00 (Auditorium 2)");
                timeComboBox.addItem("19:10 ~ 21:40 (Auditorium 2)");

                informationTextArea.setText("");
                informationTextArea.append("Director: Denis Villeneuve\n");
                informationTextArea.append("Actor: Timothee Chalamet\n");
                informationTextArea.append("Genre: Adventure, Drama, SF\n");
                informationTextArea.append("Age: 12+\n");
                informationTextArea.append("Attendance: 1.5M\n");
                informationTextArea.append("Fan Rating: 4.5 / 5.0\n");
                informationTextArea.append("Expert Rating: 4.2 / 5.0\n");
            }

            // If user choose 'Eternals', the program will provide user with information related to that
            else if (movieType == 3) {
                timeComboBox.removeAllItems();
                timeComboBox.addItem("11:50 ~ 14:20 (Auditorium 3)");
                timeComboBox.addItem("15:30 ~ 18:00 (Auditorium 3)");
                timeComboBox.addItem("20:00 ~ 22:30 (Auditorium 3)");

                informationTextArea.setText("");
                informationTextArea.append("Director: Chloe Zhao\n");
                informationTextArea.append("Actor: Angelina Jolie\n");
                informationTextArea.append("Genre: Action, Drama, Fantasy\n");
                informationTextArea.append("Age: 12+\n");
                informationTextArea.append("Attendance: 3.0M\n");
                informationTextArea.append("Fan Rating: 3.9 / 5.0\n");
                informationTextArea.append("Expert Rating: 2.4 / 5.0\n");
            }

            // If user choose 'Encanto', the program will provide user with information related to that
            else if (movieType == 4) {
                timeComboBox.removeAllItems();
                timeComboBox.addItem("8:40 ~ 10:20 (Auditorium 4)");
                timeComboBox.addItem("13:15 ~ 14:55 (Auditorium 4)");
                timeComboBox.addItem("18:10 ~ 19:50 (Auditorium 4)");
                timeComboBox.addItem("20:10 ~ 21:50 (Auditorium 4)");

                informationTextArea.setText("");
                informationTextArea.append("Director: Byron Howard\n");
                informationTextArea.append("Actor: Stephanie Beatriz\n");
                informationTextArea.append("Genre: Animation\n");
                informationTextArea.append("Age: ALL\n");
                informationTextArea.append("Attendance: 454K\n");
                informationTextArea.append("Fan Rating: 4.7 / 5.0\n");
                informationTextArea.append("Expert Rating: 4.5 / 5.0\n");
            }

            // If user choose 'House of Gucci', the program will provide user with information related to that
            else {
                timeComboBox.removeAllItems();
                timeComboBox.addItem("11:20 ~ 14:00 (Auditorium 5)");
                timeComboBox.addItem("21:10 ~ 23:50 (Auditorium 5)");

                informationTextArea.setText("");
                informationTextArea.append("Director: Ridley Scott\n");
                informationTextArea.append("Actor: Lady Gaga\n");
                informationTextArea.append("Genre: Thriller, Drama, Crime\n");
                informationTextArea.append("Age: 15+\n");
                informationTextArea.append("Attendance: 311K\n");
                informationTextArea.append("Fan Rating: 4.1 / 5.0\n");
                informationTextArea.append("Expert Rating: 3.1 / 5.0\n");
            }
        }

        @Override
        // This is called when the thread finishes.
        protected void done() {
            return;
        }
    };
    worker.execute();  // Execute the SwingWorker
}
```

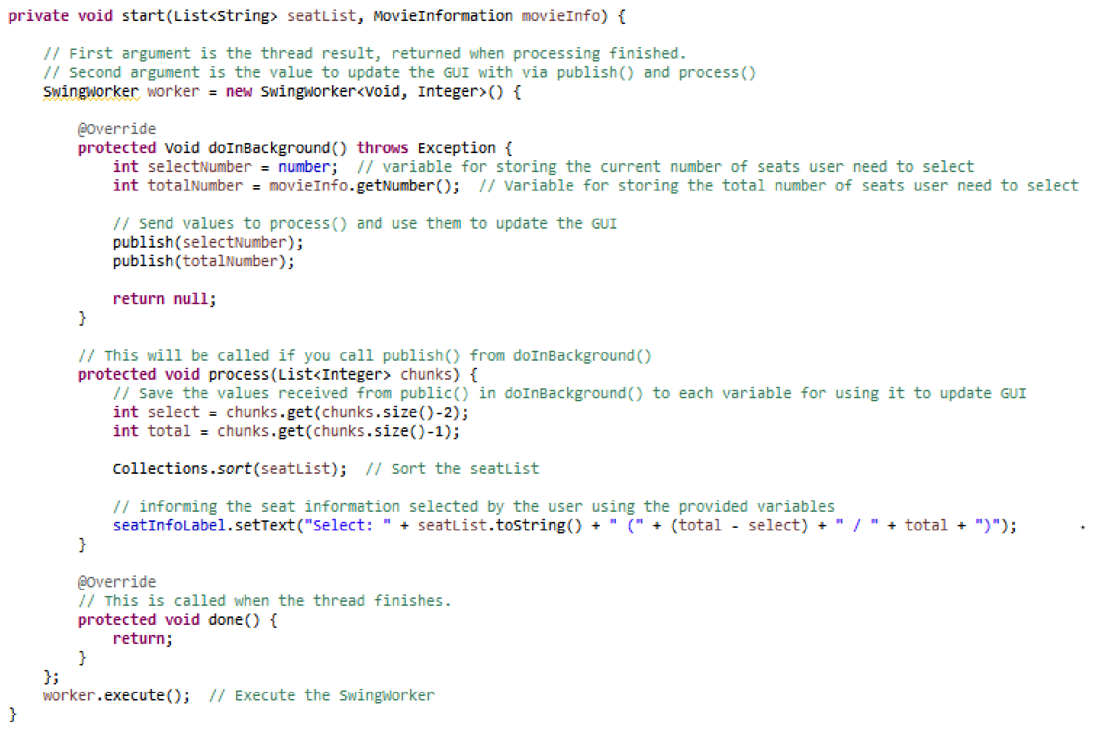
IV. Check the seat status of the movie that user select

- Read the file 'seatDB.txt' and get information about the preoccupied seat.
- Implement seat selection and cancellation functions



```java
/**
 * Since the implementation method from "A1Label" to "E5Label" are the same,
 * I will explain only about "A1Label" as a representative.
 */

// Set text, font and background color on 'A1Label' and add it to the panel
A1Label = new JLabel("   A1   ");
A1Label.setFont(new Font("Segoe UI", Font.PLAIN, 12));
A1Label.setOpaque(true);
A1Label.setBackground(Color.LIGHT_GRAY);
GridBagConstraints gbc_A1Label = new GridBagConstraints();
gbc_A1Label.insets = new Insets(0, 0, 5, 5);
gbc_A1Label.gridx = 1;
gbc_A1Label.gridy = 4;
contentPane.add(A1Label, gbc_A1Label);

// If the seat is preempted, make it unable to select
if(selectedSeat.contains("A1")) {
    A1Label.setEnabled(false);
}
// If the seat is preempted, make it able to select
else {
    A1Label.setEnabled(true);
}
// 'signUpButton' events handling by MouseAdapter
A1Label.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if(A1Label.isEnabled()) {
            // If user press the seat selected by the user again, select will be canceled
            if(A1Label.getBackground() == Color.RED) {
                A1Label.setBackground(Color.LIGHT_GRAY);
                seatList.remove("A1");  // Remove the selected seat to the "seatList"
                number++;  // the number of seats user need to select is increased
            }
            else {
                // If user selects unoccupied seats within the number of people, the selection will proceed successfully
                if(number > 0) {
                    A1Label.setBackground(Color.RED);
                    seatList.add("A1");  // Add the selected seat to the "seatList"
                    number--;  // the number of seats user need to select is decreased
                }
                // If user selects seats to exceed the number of people, the program displays warning message to the user to notice
                else {
                    JOptionPane.showMessageDialog(null, "You cannot select seats that exceed the number of people.", "Application Status", JOptionPane.WARNING_MESSAGE);
                }
            }
        }
        // If the user selects seats that is occupied, the program displays warning message to the user to notice
        else {
            JOptionPane.showMessageDialog(null, "This seat has been preempted.", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
        start(seatList, user.getMovieInfo());  // proceed the program by using multithreading.
    }
});
```

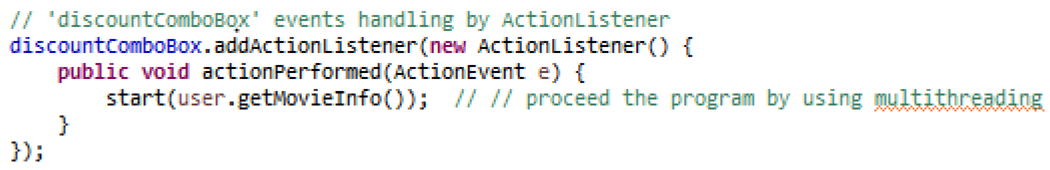
- Program provide information on seat selected by user through multithreading.



```java
private void start(List<String> seatList, MovieInformation movieInfo) {

    // First argument is the thread result, returned when processing finished.
    // Second argument is the value to update the GUI with via publish() and process()
    SwingWorker worker = new SwingWorker<Void, Integer>() {

        @Override
        protected Void doInBackground() throws Exception {
            int selectNumber = number;  // variable for storing the current number of seats user need to select
            int totalNumber = movieInfo.getNumber();  // Variable for storing the total number of seats user need to select

            // Send values to process() and use them to update the GUI
            publish(selectNumber);
            publish(totalNumber);

            return null;
        }

        // This will be called if you call publish() from doInBackground()
        protected void process(List<Integer> chunks) {
            // Save the values received from public() in doInBackground() to each variable for using it to update GUI
            int select = chunks.get(chunks.size()-2);
            int total = chunks.get(chunks.size()-1);

            Collections.sort(seatList);  // Sort the seatList

            // informing the seat information selected by the user using the provided variables
            seatInfoLabel.setText("Select: " + seatList.toString() + " (" + (total - select) + " / " + total + ")");
        }

        @Override
        // This is called when the thread finishes.
        protected void done() {
            return;
        }
    };
    worker.execute();  // Execute the SwingWorker
}
```

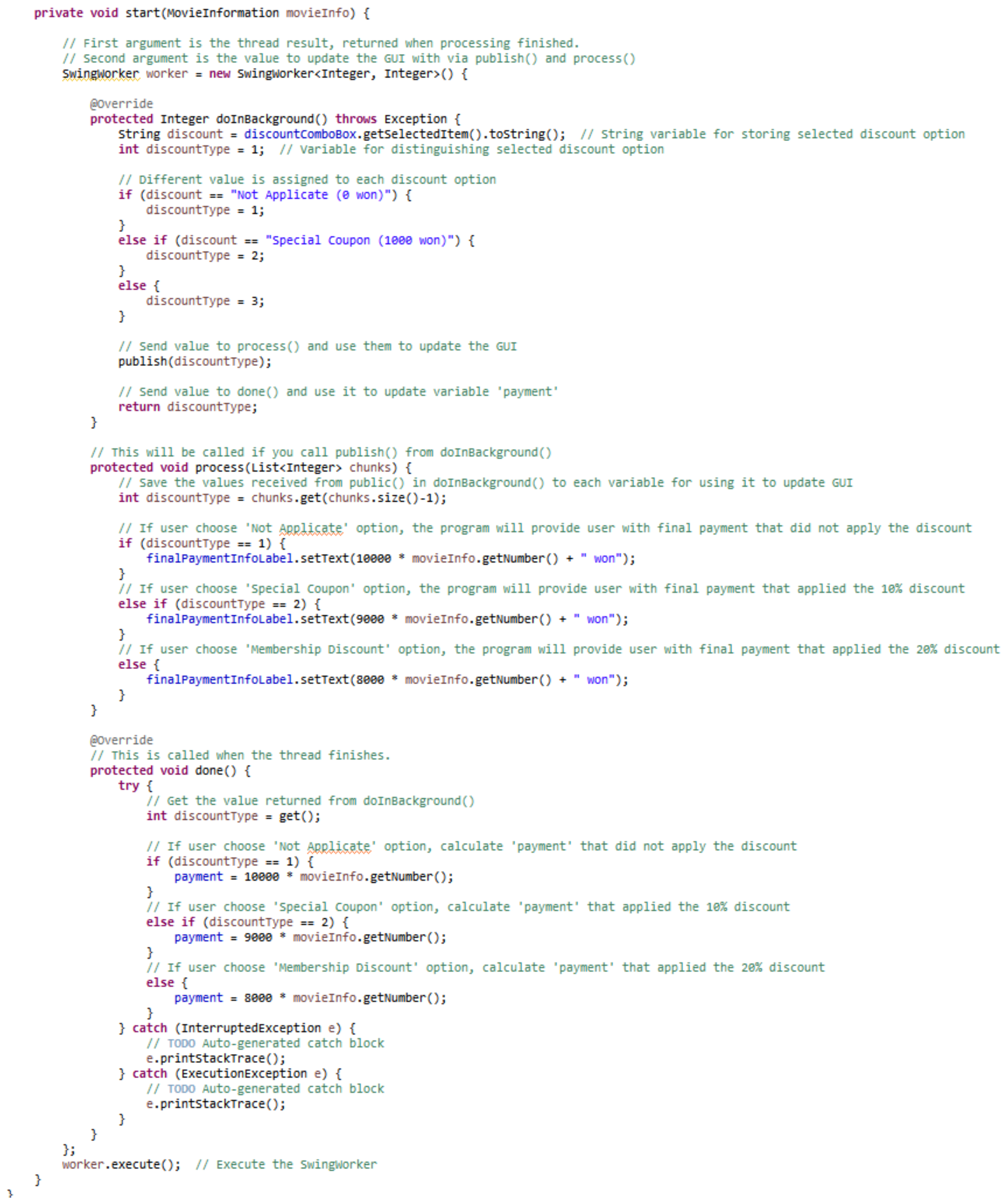
V. Calculate the final amount depends on the discount option user select.

- When the user selects the discount option, program proceeded by multithreading.



```
// 'discountComboBox' events handling by ActionListener
discountComboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        start(user.getMovieInfo());  // // proceed the program by using multithreading
    }
});
```

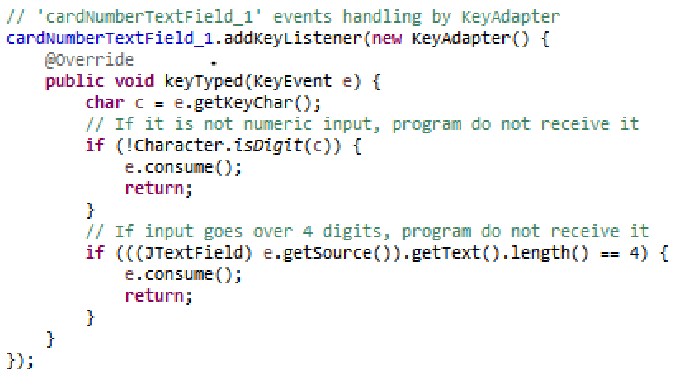
- Through multi-threading, the user may be provided with the final payment to which the discount is applied.



```
    private void start(MovieInformation movieInfo) {

        // First argument is the thread result, returned when processing finished.
        // Second argument is the value to update the GUI with via publish() and process()
        SwingWorker worker = new SwingWorker<Integer, Integer>() {

            @Override
            protected Integer doInBackground() throws Exception {
                String discount = discountComboBox.getSelectedItem().toString();  // String variable for storing selected discount option
                int discountType = 1;  // Variable for distinguishing selected discount option

                // Different value is assigned to each discount option
                if (discount == "Not Applicate (0 won)") {
                    discountType = 1;
                }
                else if (discount == "Special Coupon (1000 won)") {
                    discountType = 2;
                }
                else {
                    discountType = 3;
                }

                // Send value to process() and use them to update the GUI
                publish(discountType);

                // Send value to done() and use it to update variable 'payment'
                return discountType;
            }

            // This will be called if you call publish() from doInBackground()
            protected void process(List<Integer> chunks) {
                // Save the values received from public() in doInBackground() to each variable for using it to update GUI
                int discountType = chunks.get(chunks.size()-1);

                // If user choose 'Not Applicate' option, the program will provide user with final payment that did not apply the discount
                if (discountType == 1) {
                    finalPaymentInfoLabel.setText(10000 * movieInfo.getNumber() + " won");
                }
                // If user choose 'Special Coupon' option, the program will provide user with final payment that applied the 10% discount
                else if (discountType == 2) {
                    finalPaymentInfoLabel.setText(9000 * movieInfo.getNumber() + " won");
                }
                // If user choose 'Membership Discount' option, the program will provide user with final payment that applied the 20% discount
                else {
                    finalPaymentInfoLabel.setText(8000 * movieInfo.getNumber() + " won");
                }
            }

            @Override
            // This is called when the thread finishes.
            protected void done() {
                try {
                    // Get the value returned from doInBackground()
                    int discountType = get();

                    // If user choose 'Not Applicate' option, calculate 'payment' that did not apply the discount
                    if (discountType == 1) {
                        payment = 10000 * movieInfo.getNumber();
                    }
                    // If user choose 'Special Coupon' option, calculate 'payment' that applied the 10% discount
                    else if (discountType == 2) {
                        payment = 9000 * movieInfo.getNumber();
                    }
                    // If user choose 'Membership Discount' option, calculate 'payment' that applied the 20% discount
                    else {
                        payment = 8000 * movieInfo.getNumber();
                    }
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (ExecutionException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        };
        worker.execute();  // Execute the SwingWorker
    }
}
```

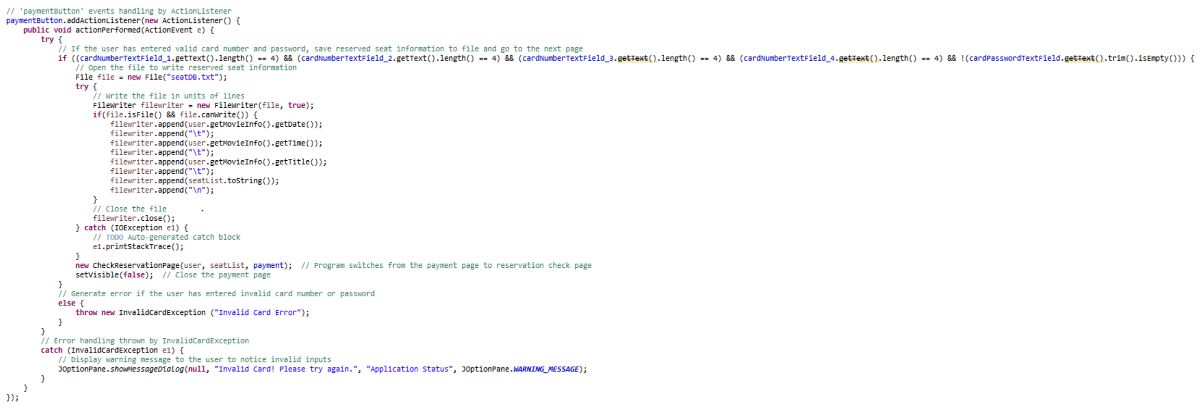
VI. Check the validation of card number and password.

- In the case of card numbers, it is limited to receive only 4 numbers per box.



```java
// 'cardNumberTextField_1' events handling by KeyAdapter
cardNumberTextField_1.addKeyListener(new KeyAdapter() {
    @Override
    public void keyTyped(KeyEvent e) {
        char c = e.getKeyChar();
        // If it is not numeric input, program do not receive it
        if (!Character.isDigit(c)) {
            e.consume();
            return;
        }
        // If input goes over 4 digits, program do not receive it
        if (((JTextField) e.getSource()).getText().length() == 4) {
            e.consume();
            return;
        }
    }
});
```

- If the card number and password are valid, complete the payment and save the reserved movie information to the file 'seatDB.txt'.



```java
// 'paymentButton' events handling by ActionListener
paymentButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // If the user has entered valid card number and password, save reserved seat information to file and go to the next page
            if ((cardNumberTextField_1.getText().length() == 4) && (cardNumberTextField_2.getText().length() == 4) && (cardNumberTextField_3.~~getText~~().length() == 4) && (cardNumberTextField_4.~~getText~~().length() == 4) && !(cardPasswordTextField.~~getText~~().trim().isEmpty())) {
                // Open the file to write reserved seat information
                File file = new File("seatDB.txt");
                try {
                    // Write the file in units of lines
                    FileWriter filewriter = new FileWriter(file, true);
                    if(file.isFile() && file.canWrite()) {
                        filewriter.append(user.getMovieInfo().getDate());
                        filewriter.append("\t");
                        filewriter.append(user.getMovieInfo().getTime());
                        filewriter.append("\t");
                        filewriter.append(user.getMovieInfo().getTitle());
                        filewriter.append("\t");
                        filewriter.append(seatList.toString());
                        filewriter.append("\n");
                    }
                    // Close the file
                    filewriter.close();
                } catch (IOException e1) {
                    // TODO Auto-generated catch block
                    e1.printStackTrace();
                }
                new CheckReservationPage(user, seatList, payment);  // Program switches from the payment page to reservation check page
                setVisible(false);  // Close the payment page
            }
            // Generate error if the user has entered invalid card number or password
            else {
                throw new InvalidCardException ("Invalid Card Error");
            }
        }
        // Error handling thrown by InvalidCardException
        catch (InvalidCardException e1) {
            // Display warning message to the user to notice invalid inputs
            JOptionPane.showMessageDialog(null, "Invalid Card! Please try again.", "Application Status", JOptionPane.WARNING_MESSAGE);
        }
    }
});
```

5. Explain what is included in your project and why it is used (Polymorphism, Inheritance, File I/O, etc)

I. Inheritance

- Inheritance is used so that the 'object User' have 'object MovieInformation'.

II. Exception Handling

- Exception Handling is used to generate messages for unsuitable inputs during login process.

- Exception Handling is used to generate messages for unsuitable inputs during membership registration process.

- Exception Handling is used to generate messages about when the user selects already preoccupied seats or selects seats that does not suit the number of people during the seat selection process.

- Exception Handling is used to generate messages for unsuitable inputs during payment process.

III. Multithreading

- Multithreading is used to immediately provide brief information about the movie selected by the user.

- Multithreading is used to reflect the seats selected by the user in real time.

- Multithreading is used to immediately calculate the final amount depends on the discount option user select.

IV. File I/O

- File 'userDB.txt' is used to store information about the account. After Reading file 'userDB.txt', program determines whether the ID and password entered from the user are correct. If the user have successfully created an account, write the account information in the file 'userDB.txt'

- File 'seatDB.txt' is used to store information about the movie theater seat occupancy status. The program identifies the status of seat reservations through file 'seatDB.txt' and provides it to user. After the user confirms the reservation, the selected seat will be written in 'seatDB.txt'.