

## COMPUTER SCIENCE DEPARTMENT

**CS0053**

(PROGRAMMING TOOLS AND TECHNIQUES)

EXERCISE

4

GUI-Based Password Creation Module with Validation

|  |  |
| --- | --- |
| Name: John Paul L. Besagas | Professor: Dr. Beau Habal |
| Date Performed : 10/7/2024 | Date Submitted: 10/7/2024 |

1. **OBJECTIVES**

At the end of this exercise, students must be able to:

Cognitive

1. Understand the topics they have learned from lesson 4.

.

Psychomotor:

1. Apply exception handling and assertion.
2. Construct a simple JUnit Testing.
3. Create a GUI-based Java program using exception handling.

Affective

1. Appreciate the concept behind this exercise.
2. **BACKGROUND INFORMATION**

In order to accomplish this exercise, the student must have a clear understanding of the following topics:

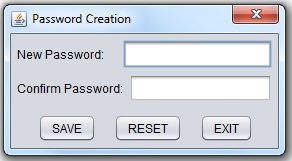
* Java Exception Classes
* try-catch statements
* Import packages or libraries

1. **LABORATORY PROCEDURE**

1. Create a new program.

Program Name: PasswordCreation.java

2. Design your layout as shown below



3. Requirements

- The program will force the user to input correct value.

* The program will display notification for successful and unsuccessful attempt.
* The program must apply the password criteria:

- must be at least 8 characters

- must have at least one char both lower and uppercase, one number and one special character.

* Apply coding conventions.
* The program must be free from any errors.

|  |
| --- |
| import javax.swing.\*;  import java.awt.\*;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.util.prefs.Preferences;  public class PasswordCreation {      JFrame framePasswordCreation;      JPasswordField fieldNewPassword, fieldConfirmPassword;      JButton buttonPasswordCreation, buttonClear, buttonCloseButton;      Font fontDisplay = new Font("Arial", Font.PLAIN, 16);      Preferences preferences = Preferences.userRoot().node(this.getClass().getName());      public PasswordCreation() {          framePasswordCreation = new JFrame("Password Creation");          framePasswordCreation.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);          framePasswordCreation.setSize(400, 300);          framePasswordCreation.getContentPane().setBackground(new Color(60, 63, 65));          framePasswordCreation.setLayout(new BorderLayout());          JPanel mainPanel = new JPanel();          mainPanel.setBackground(new Color(60, 63, 65));          mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y\_AXIS));          fieldNewPassword = new JPasswordField();          fieldConfirmPassword = new JPasswordField();          fieldNewPassword.setForeground(Color.WHITE);          fieldNewPassword.setBackground(new Color(40, 40, 40));          fieldConfirmPassword.setForeground(Color.WHITE);          fieldConfirmPassword.setBackground(new Color(40, 40, 40));          fieldNewPassword.setCaretColor(Color.WHITE);          fieldConfirmPassword.setCaretColor(Color.WHITE);          JLabel labelNewPassword = new JLabel("New Password:");          JLabel labelConfirmPassword = new JLabel("Confirm Password:");          labelNewPassword.setForeground(Color.WHITE);          labelConfirmPassword.setForeground(Color.WHITE);          JPanel formPanel = new JPanel(new GridLayout(2, 2, 10, 10));          formPanel.setBackground(new Color(60, 63, 65));          formPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));          formPanel.add(labelNewPassword);          formPanel.add(fieldNewPassword);          formPanel.add(labelConfirmPassword);          formPanel.add(fieldConfirmPassword);          JPanel buttonPanel = new JPanel();          buttonPanel.setBackground(new Color(60, 63, 65));          buttonPasswordCreation = new JButton("SAVE");          buttonPasswordCreation.setPreferredSize(new Dimension(110, 30));          buttonPasswordCreation.setBackground(new Color(30, 144, 255));          buttonPasswordCreation.setForeground(Color.WHITE);          buttonPasswordCreation.setFocusPainted(false);          buttonClear = new JButton("RESET");          buttonClear.setPreferredSize(new Dimension(110, 30));          buttonClear.setBackground(new Color(30, 144, 255));          buttonClear.setForeground(Color.WHITE);          buttonClear.setFocusPainted(false);          buttonCloseButton = new JButton("EXIT");          buttonCloseButton.setPreferredSize(new Dimension(110, 30));          buttonCloseButton.setBackground(new Color(220, 20, 60));          buttonCloseButton.setForeground(Color.WHITE);          buttonCloseButton.setFocusPainted(false);          buttonPanel.add(buttonPasswordCreation);          buttonPanel.add(buttonClear);          buttonPanel.add(buttonCloseButton);          buttonPasswordCreation.addActionListener(new ActionListener() {              public void actionPerformed(ActionEvent e) {                  String newPassword = new String(fieldNewPassword.getPassword());                  String confirmPassword = new String(fieldConfirmPassword.getPassword());                  String savedPassword = preferences.get("savedPassword", "");                  try {                      validatePasswords(newPassword, confirmPassword, savedPassword);                      preferences.put("savedPassword", newPassword);                      JOptionPane.showMessageDialog(framePasswordCreation, "Password saved successfully.");                      preferences.putBoolean("loggedIn", true);                      framePasswordCreation.dispose();                      new PasswordCreation();                  } catch (PasswordMatch exception) {                      JOptionPane.showMessageDialog(framePasswordCreation, exception.getMessage());                  } catch (PasswordCriteria exception) {                      JOptionPane.showMessageDialog(framePasswordCreation, exception.getMessage());                  } catch (Exception exception) {                      JOptionPane.showMessageDialog(framePasswordCreation, "Error: " + exception.getMessage());                  }              }          });          buttonClear.addActionListener(new ActionListener() {              public void actionPerformed(ActionEvent e) {                  clearPasswordCreation();              }          });          buttonCloseButton.addActionListener(new ActionListener() {              public void actionPerformed(ActionEvent e) {                  framePasswordCreation.dispose();              }          });          mainPanel.add(formPanel);          mainPanel.add(buttonPanel);          framePasswordCreation.add(mainPanel, BorderLayout.CENTER);          framePasswordCreation.setLocationRelativeTo(null);          framePasswordCreation.setVisible(true);      }      class PasswordMatch extends Exception {          public PasswordMatch(String message) {              super(message);          }      }      class PasswordCriteria extends Exception {          public PasswordCriteria(String message) {              super(message);          }      }      // password with exceptions      public void validatePasswords(String newPassword, String confirmPassword, String savedPassword)              throws PasswordMatch, PasswordCriteria {          if (!newPassword.equals(confirmPassword)) {              throw new PasswordMatch("Passwords do not match. Try again.");          }          if (newPassword.equals(savedPassword)) {              throw new PasswordMatch(                      "The new password cannot be the same as the current password. Please try a different password.");          }          if (!checkPassword(newPassword)) {              throw new PasswordCriteria(                      "Invalid password. Password must be at least 8 characters, must have at least one char both lower and uppercase, one number, and one special character.");          }      }      // password criteria      public boolean checkPassword(String password) {          return password.length() >= 8 &&                  password.matches(".\*[a-z].\*") &&                  password.matches(".\*[A-Z].\*") &&                  password.matches(".\*[0-9].\*") &&                  password.matches(".\*[!@#$%^&\*(),.?\":{}|<>].\*");      }      public void clearPasswordCreation() {          fieldNewPassword.setText("");          fieldConfirmPassword.setText("");      }      public static void main(String[] args) {          new PasswordCreation();      }  } |

1. **QUESTION AND ANSWER**
2. What happen when the program encounter runtime errors?

**I have defined custom exceptions PasswordMatch and PasswordCriteria to handle runtime errors such as mismatched passwords or failure to meet password criteria. These exceptions ensure that specific issues related to password validation are handled gracefully by displaying appropriate error messages to the user, rather than crashing the program. Additionally, I have a generic Exception block in place to catch any unforeseen errors and display the error message to the user via a dialog box, allowing the program to continue running without abruptly terminating.**

1. How do you able to apply the exception handling?

**I implemented exception handling by creating custom exceptions and using a try-catch block to handle password validation. The validatePasswords method throws custom exceptions PasswordMatch and PasswordCriteria for password mismatch or criteria violations. In the ActionListener for the password creation button, I use a try-catch structure to handle these exceptions and display specific error messages based on the exception type. If any exception is thrown during password validation, it is caught, and the user is notified through a message dialog box, while the program continues to run smoothly.**

1. What are the anticipated exception types in your program?

**I have implemented three types of exceptions for password validation in my program. The first, PasswordMatch, is thrown when the input passwords don't match or when the new password is the same as the current saved password. The second, PasswordCriteria, is triggered when the new password fails to meet specific security requirements, such as being at least eight characters long and containing a mix of upper and lowercase letters, a number, and a special character. Finally, I use a general Exception to catch any other unforeseen runtime errors that are not directly related to password validation.**

1. **QUESTION AND ANSWER**

|  |  |
| --- | --- |
| Department | Computer Science |
| Subject Code | CSSSPEC2 |
| Description | Programming Tools and Techniques |
| Term/Academic Year | 1st Term SY 2016-2017 |

|  |  |
| --- | --- |
| Topic | Handling Exceptions and Assertions with JUnit |
| Lab Activity No | 4 |
| Lab Activity | **GUI-Based Password Creation Module with Validation** |
| CLO | **1, 2** |

**Note: The following rubrics/metrics will be used to grade students’ output in the lab exercise 4.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Criteria | Exceptional | Acceptable | Amateur | Unsatisfactory |
| Specifications  (40%) | The program works and meets all of the specifications. (40) | The program works and produces the correct results and displays them correctly. It also meets most of the other specifications. (35-39) | The program produces correct results but does not display them correctly. (30-34) | The program is producing incorrect results. (20-29) |
| Design  (15 %) | The design is exceptionally attractive. Program is "user-friendly" with informative and consistent prompts and messages. (15) | The design is fairly attractive. Program is "user-friendly" with informative and consistent prompts and messages. (13-14) | The design is fairly attractive. Program is not "user-friendly" but still provide informative and consistent prompts and messages. (10-12) | The design is unattractive and not user-friendly (8-9) |
| Efficiency (20%) | The code is extremely efficient without sacrificing readability and understanding. (20) | The code is fairly efficient without sacrificing readability and understanding. (17-19) | The code is brute force and unnecessarily long. (14-16) | The code is huge and appears to be patched together. (10-13) |
| Readability  (10 %) | The code is exceptionally well organized and very easy to follow. (10) | The code is fairly easy to read. (8-9) | The code is readable only by someone who knows what it is supposed to be doing. (6-7) | The code is poorly organized and very difficult to read. (4-5) |
| Delivery  (15%) | The program was delivered on time. (15) | The program was delivered within a day of the due date. (13-14) | The code was within 2 days of the due date. (10-12) | The code was within a week of the due date. (8-9) |
| Total: 100% |  |  |  |  |