Rajalakshmi Engineering College

Name: Jai Shuriya J
Email: 240701200@rajalakshmi.edu.in
Roll no: 240701200
Phone: 8754381941
Branch: REC
Department: CSE - Section 10
Batch: 2028
Degree: B.E - CSE

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 8_CY

Attempt : 1
Total Mark : 40
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

Camila, a user of a social media platform, is looking to change her password to enhance account security. The platform enforces specific rules for password strength to ensure the safety of user accounts. Camila needs a program that prompts her to enter a new password and throws custom exceptions based on the strength of the password.

Password Strength Criteria:

Weak Password:

Length less than 8 characters.Medium Password:

Length 8 or more characters.Missing a mix of uppercase letters, lowercase letters, and digits.

Implement a custom exception, to assist Camila in changing her password

securely. The program should interactively take user input for a new password, categorize its strength, and handle custom exceptions (WeakPasswordException and MediumPasswordException) if the password fails to meet the specified criteria.

Input Format

The input consists of a string s, representing the new password.

Output Format

The output is displayed in the following format:

If the entered password meets the strength criteria, the program outputs

"Password changed successfully!"

If the entered password is weak, the program outputs

"Error: Weak password. It must be at least 8 characters long."

If the entered password is of medium strength, the program outputs

"Error: Medium password. It must include a mix of uppercase letters, lowercase letters, and digits."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: ComplexP@ss1
Output: Password changed successfully!

Answer

```
import java.util.Scanner;

class WeakPasswordException extends Exception {
    public WeakPasswordException(String message) {
        super(message);
    }
}
```

```
class MediumPasswordException extends Exception {
    public MediumPasswordException(String message) {
        super(message);
    }
}

class PasswordChangeSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            String newPassword = scanner.nextLine();

            categorizePassword(newPassword);


            System.out.println("Password changed successfully!");
        } catch (WeakPasswordException | MediumPasswordException e) {

            System.out.println("Error: " + e.getMessage());
        } finally {

            scanner.close();
        }
    }

    private static void categorizePassword(String newPassword) throws
WeakPasswordException, MediumPasswordException {

        if (newPassword.length() < 8) {
            throw new WeakPasswordException("Weak password. It must be at least
8 characters long.");
        } else if (!containsUppercase(newPassword) || !
containsLowercase(newPassword) || !containsDigit(newPassword)) {
            throw new MediumPasswordException("Medium password. It must
include a mix of uppercase letters, lowercase letters, and digits.");
        }
    }

    private static boolean containsUppercase(String password) {
        return !password.equals(password.toLowerCase());
```

```
    }

    private static boolean containsLowercase(String password) {
        return !password.equals(password.toUpperCase());
    }

    private static boolean containsDigit(String password) {
        for (char c : password.toCharArray()) {
            if (Character.isDigit(c)) {
                return true;
            }
        }
        return false;
    }
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Tim was tasked with creating a user profile system that validates the user's date of birth input. The system should throw a custom exception, InvalidDateOfBirthException, if the date is not in the specified format "dd-mm-yyyy" or if it represents an invalid calendar date.

The main method takes user input, validates the date of birth, and prints whether it is valid or not.

Input Format

The input consists of a string, representing the date of birth of the user.

Output Format

The output displays one of the following results:

If the entered date of birth is valid according to the specified format, the program prints:

"[Date] is a valid date of birth"

If the entered date of birth is not valid according to the specified format, the

program prints:

"Invalid date: [Date]"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 01-01-2000

Output: 01-01-2000 is a valid date of birth

Answer

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Scanner;
class InvalidDateOfBirthException extends Exception {
    public InvalidDateOfBirthException(String message) {
        super(message);
    }
}
class UserProfileSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String userInput = "";

        try {
            userInput = scanner.nextLine();
            validateDateOfBirth(userInput);
            System.out.println(userInput + " is a valid date of birth");

        } catch (InvalidDateOfBirthException e) {
            System.out.println(e.getMessage() + ": " + userInput);
        } finally {
            scanner.close();
        }
    }

    private static void validateDateOfBirth(String userInput) throws
InvalidDateOfBirthException {
```

```
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
        dateFormat.setLenient(false);

        try {
            Date dob = dateFormat.parse(userInput);
        } catch (ParseException e) {
            throw new InvalidDateOfBirthException("Invalid date");
        }
    }
}
```

Status : Correct **Marks : 10/10**

3. Problem Statement

In an online shopping cart system, users can apply coupon codes during checkout to avail of discounts. However, to ensure the validity and security of coupon codes, the system enforces specific rules for their format. Your task is to implement a Java program named CouponCodeValidator that takes user input for a coupon code and validates it according to the specified rules.

Rules for Valid Coupon Code:

The coupon code must consist of exactly 10 characters.The coupon code must contain at least one alphabet (uppercase or lowercase) and at least one digit (0-9).Special characters are not allowed in the coupon code.

Implement a custom exception, InvalidCouponException, to handle cases where the entered coupon code does not meet the specified criteria.

Input Format

The input consists of a string s, representing the coupon code.

Output Format

The output is displayed in the following format:

If the entered coupon code meets the specified criteria, the program outputs

"Coupon code applied successfully!"

If the entered coupon code has less than or more than 10 characters it outputs

"Error: Invalid coupon code length. It must be exactly 10 characters."

If the entered coupon code contains only numeric or only alphabets it outputs

"Error: Invalid coupon code format. It must contain at least one alphabet and one digit."

If the entered coupon code contains special characters it outputs

"Error: Coupon code should not contain special characters."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: ABCD123456
Output: Coupon code applied successfully!

Answer

```
import java.util.Scanner;

class InvalidCouponException extends Exception {
    public InvalidCouponException(String message) {
        super(message);
    }
}

class CouponCodeValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            String couponCode = scanner.nextLine();
            validateCouponCode(couponCode);
            System.out.println("Coupon code applied successfully!");
        } catch (InvalidCouponException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
```

```
        scanner.close();
    }
  }

  private static void validateCouponCode(String couponCode) throws
InvalidCouponException {

    if (containsSpecialCharacter(couponCode)) {
      throw new InvalidCouponException("Coupon code should not contain
special characters.");
    }

    if (!couponCode.matches("^(?=.*[a-zA-Z])(?=.*\\d)[a-zA-Z0-9]{10}$")) {
      if (couponCode.length() != 10) {
        throw new InvalidCouponException("Invalid coupon code length. It must
be exactly 10 characters.");
      } else {
        throw new InvalidCouponException("Invalid coupon code format. It
must contain at least one alphabet and one digit.");
      }
    }
  }

  private static boolean containsSpecialCharacter(String str) {

    return str.matches(".*[^a-zA-Z0-9].*");
  }
}
```

Status : Correct **Marks : 10/10**

4. Problem Statement

A company is developing a user registration system that requires users to provide valid email addresses. The development team is implementing an EmailValidator program to ensure that the entered email addresses meet certain criteria using exception handling.

The email address must contain the "@" symbol.The email address must consist of a non-empty username(before "@" symbol) and a non-empty

domain(after "'@" symbol).The domain part of the email address must contain at least one period (".").The email address must not contain leading or trailing spaces.

Implement a custom exception, InvalidEmailException, to fulfill the company's requirements and validate it according to the specified rules.

Input Format

The input consists of a string value 's', which represents the email address.

Output Format

The output is displayed in the following format:

If the entered email address is valid according to the specified rules, the program prints:

"Email address is valid!"

If the entered email address misses the username or domain part or misses "@" symbol or has two or more "@" symbols or misses '.' in the domain part it outputs:

"Error: Invalid email format."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: johndoe@example.com
Output: Email address is valid!

Answer

```
import java.util.Scanner;
class EmailValidator {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      try {
         String emailAddress = scanner.nextLine();
         validateEmailAddress(emailAddress);
```

```
            System.out.println("Email address is valid!");
        } catch (InvalidEmailException | java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void validateEmailAddress(String emailAddress) throws
InvalidEmailException {
        if (!emailAddress.contains("@")) {
            throw new InvalidEmailException("Invalid email format. ");
        }

        String[] parts = emailAddress.split("@");
        if (parts.length != 2 || parts[0].isEmpty() || parts[1].isEmpty() || !
parts[1].contains(".")) {
            throw new InvalidEmailException("Invalid email format. ");
        }
    }
}
class InvalidEmailException extends Exception {
    public InvalidEmailException(String message) {
        super(message);
    }
}
```

Status : Correct **Marks : 10/10**