Rajalakshmi Engineering College

Name: Ganesh Raghav C
Email: 240701138@rajalakshmi.edu.in
Roll no:
Phone: 6382988302
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

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**

2. Problem Statement

Theo is trying to update his payment information on a subscription-based streaming service. To proceed, the system requires Theo to provide a valid credit card number consisting of 16 digits. However, Theo wants to make sure that the credit card number he enters meets the specified criteria with proper exception handling.

The credit card number must consist of exactly 16 digits.If the entered credit card number does not meet the specified criteria, the program should throw a custom exception, InvalidCreditCardException, and provide Theo with specific error messages:If the length of the credit card number is not 16 digits, the exception message should be: "Invalid credit card number length."If the credit card number contains non-numeric characters, the exception message should be: "Invalid credit card number format."

Implement a custom exception, InvalidCreditCardException, to fulfill Theo's requirements and keep his payment information secure.

Input Format

The input consists of a string value 's', consisting of the 16-digit credit card number.

Output Format

The output is displayed in the following format:

If the entered credit card number is valid, the program should output a success message:

"Payment information updated successfully!"

If the entered credit card has more than 16 digits or less than 16 digits it displays

"Error: Invalid credit card number length."

If the entered 16-digit credit card has non-integers it displays

"Error: Invalid credit card number format."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1234567890123456

Output: Payment information updated successfully!

Answer

```
import java.util.Scanner;

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

class CreditCardValidator {
   public void validateCreditCardNumber(String creditCardNumber) throws
InvalidCreditCardException {
      if (!creditCardNumber.matches("^\\d{16}$")) {
         if (creditCardNumber.length() != 16) {
            throw new InvalidCreditCardException("Invalid credit card number
length.");
         } else {
            throw new InvalidCreditCardException("Invalid credit card number
format.");
         }
      }
   }
}

class CreditCardUpdater {
   private CreditCardValidator validator = new CreditCardValidator();

   public void updateCreditCard() {
      Scanner scanner = new Scanner(System.in);
      try {
         String creditCardNumber = scanner.nextLine();

         validator.validateCreditCardNumber(creditCardNumber);
```

```
            System.out.println("Payment information updated successfully!");
        } catch (InvalidCreditCardException | java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
public class Main {
    public static void main(String[] args) {
        CreditCardUpdater updater = new CreditCardUpdater();
        updater.updateCreditCard();
    }
}
```

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

3. 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 {
        // Check for password strength requirements
        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.");
        }
    }

    // Helper method to check if the password contains uppercase letters
    private static boolean containsUppercase(String password) {
        return !password.equals(password.toLowerCase());
    }

    // Helper method to check if the password contains lowercase letters
    private static boolean containsLowercase(String password) {
        return !password.equals(password.toUpperCase());
    }

    // Helper method to check if the password contains at least one digit
    private static boolean containsDigit(String password) {
        for (char c : password.toCharArray()) {
            if (Character.isDigit(c)) {
                return true;
            }
        }
```

```
        return false;
    }
}
```

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***