Rajalakshmi Engineering College

Name: Samyuktha S
Email: 240701701@rajalakshmi.edu.in
Roll no: 240701701
Phone: 6380226314
Branch: REC
Department: CSE - Section 9
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

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 static void validateCreditCard(String cardNumber) throws
InvalidCreditCardException {
      if (cardNumber.length() != 16) {
         throw new InvalidCreditCardException("Invalid credit card number
length.");
      }
      if (!cardNumber.matches("\\d{16}")) {
         throw new InvalidCreditCardException("Invalid credit card number
format.");
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      String cardNumber = scanner.nextLine();
      scanner.close();

      try {
         validateCreditCard(cardNumber);
         System.out.print("Payment information updated successfully!");
      } catch (InvalidCreditCardException e) {
         System.out.print("Error: " + e.getMessage());
      }
   }
}
```

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

2. Problem Statement

Faustus is managing his bank account and wants to create a program to update his account balance based on certain conditions. However, he needs to handle specific scenarios related to invalid inputs and insufficient balances. Faustus wants to update his account balance. He inputs the current balance and the amount to be updated.

The initial account balance should be positive. If Faustus enters a negative initial balance, the program should throw an InvalidAmountException with the message "Invalid amount. Please enter a positive initial balance."If the amount to be updated is negative, the program should check if the subtraction results in a negative balance. If so, it should throw an InsufficientBalanceException with the message "Insufficient balance."If the amount to be updated is positive, it should be added to the current balance, and the new balance should be printed.

Implement a custom exception, InvalidAmountException, and InsufficientBalanceException, to manage his bank account.

Input Format

The first line of input consists of a double value 'd', representing the initial account balance.

The second line of input consists of a double value 'd1', representing the amount to be updated.

Output Format

The output is displayed in the following format:

If the validation passes, print

"Account balance updated successfully! New balance: {new_balance}"

where {new_balance} is the updated account balance.

If the initial bank amount is negative it displays

"Error: Invalid amount. Please enter a positive initial balance."

If the updated amount exceeds the initial account balance in withdrawal it displays

"Error: Insufficient balance."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1000
500
Output: Account balance updated successfully! New balance: 1500.0

Answer

```
import java.util.Scanner;


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


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

class BankAccountManager {

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      double initialBalance = scanner.nextDouble();
      double updateAmount = scanner.nextDouble();
      scanner.close();

      try {
         if (initialBalance < 0) {
            throw new InvalidAmountException("Invalid amount. Please enter a
positive initial balance.");
         }

         double newBalance = initialBalance + updateAmount;

         if (updateAmount < 0 && newBalance < 0) {
            throw new InsufficientBalanceException("Insufficient balance.");
        }

        System.out.print("Account balance updated successfully! New balance: "
+ newBalance);

      } catch (InvalidAmountException e) {
        System.out.print("Error: " + e.getMessage());
      } catch (InsufficientBalanceException e) {
        System.out.print("Error: " + e.getMessage());
      }
    }
}
```

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 PasswordValidator {

   public static void validatePassword(String password) throws
WeakPasswordException, MediumPasswordException {
      if (password.length() < 8) {
         throw new WeakPasswordException("Weak password. It must be at least
8 characters long.");
      }
      boolean hasUpper = false, hasLower = false, hasDigit = false;
      for (char ch : password.toCharArray()) {
         if (Character.isUpperCase(ch)) hasUpper = true;
         else if (Character.isLowerCase(ch)) hasLower = true;
         else if (Character.isDigit(ch)) hasDigit = true;
      }
      if (!(hasUpper && hasLower && hasDigit)) {
         throw new MediumPasswordException("Medium password. It must
include a mix of uppercase letters, lowercase letters, and digits.");
      }
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String password = sc.nextLine();
      sc.close();

      try {
         validatePassword(password);
         System.out.print("Password changed successfully!");
      } catch (WeakPasswordException e) {
         System.out.print("Error: " + e.getMessage());
      } catch (MediumPasswordException e) {
         System.out.print("Error: " + e.getMessage());
      }
   }
}
```

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

4. Problem Statement

Alice is designing a program that requires users to enter positive numbers. She wants to implement a solution that validates whether the entered number is positive. In case the input is not a positive number, she wants to throw a custom exception.

The number should be a positive integer.If this condition is violated, the program should throw a custom exception:InvalidPositiveNumberException with the message "Invalid input. Please enter a positive integer."

Implement a custom exception, InvalidPositiveNumberException , to handle cases where the entered number does not meet the specified criteria.

Input Format

The input consists of an integer value 'n', representing the entered number.

Output Format

The output is displayed in the following format:

If the validation passes, print

"Number {number} is positive."

The {number} represents the entered positive integer.

If the entered number is negative then it displays

"Error: Invalid input. Please enter a positive integer."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 100
Output: Number 100 is positive.

Answer

```
import java.util.Scanner;

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

class PositiveNumberValidator {
   public static void validatePositive(int n) throws
InvalidPositiveNumberException {
      if (n <= 0) {
         throw new InvalidPositiveNumberException("Invalid input. Please enter a
positive integer.");
      }
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      sc.close();

      try {
         validatePositive(n);
         System.out.print("Number " + n + " is positive.");
      } catch (InvalidPositiveNumberException e) {
         System.out.print("Error: " + e.getMessage());
      }
   }
}
```

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