Rajalakshmi Engineering College

Name: Mohit Jha
Email: 241901056@rajalakshmi.edu.in
Roll no:
Phone: 9445934493
Branch: REC
Department: CSE (CS) - Section 1
Batch: 2028
Degree: B.E - CSE (CS)

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

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**

2. 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 PositiveNumberValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            int number = scanner.nextInt();
            validatePositiveNumber(number);
            System.out.println("Number " + number + " is positive.");
        } catch (InvalidPositiveNumberException | java.util.InputMismatchException
e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void validatePositiveNumber(int number) throws
InvalidPositiveNumberException {
```

```
        if (number <= 0) {
            throw new InvalidPositiveNumberException("Invalid input. Please enter a
positive integer.");
        }
    }
}
class InvalidPositiveNumberException extends Exception {
    public InvalidPositiveNumberException(String message) {
        super(message);
    }
}
```

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

3. Problem Statement

Hemanth is designing a banking system for XYZ Bank. The system should allow customers to perform deposit, withdrawal, and balance inquiry operations. Implement exception handling for scenarios involving invalid transaction amounts or insufficient funds.

Create two custom exception classes, InvalidAmountException and InsufficientFundsException, both extending the Exception class.Throw an InvalidAmountException with a message if the deposit amount is less than or equal to zero.Throw an InsufficientFundsException if the withdrawal amount is greater than the available balance.Deduct the withdrawal amount from the balance if the withdrawal is successful.

Assist Hemanth in designing the program.

Input Format

The first line of input consists of a double value B, representing the initial balance.

The second line consists of a double value D, representing the deposit amount.

The third line consists of a double value W, representing the withdrawal amount.

Output Format

If the withdrawal is successful, print the amount withdrawn and the current

balance, rounded off to one decimal place.

If an InvalidAmountException occurs, print "Error: [D] is not valid".

If an InsufficientFundsException occurs, print "Error: Insufficient funds".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1050.1
270.2
150.3
Output: Amount Withdrawn: 150.3
Current Balance: 1170.0

Answer

```
import java.util.Scanner;

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

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

class HDFCBank {
   private double balance;

   public static void main(String[] args) {
      HDFCBank hdfcBank = new HDFCBank();
      hdfcBank.processTransactions();
   }

   public void processTransactions() {
```

```
        Scanner scanner = new Scanner(System.in);

        try {
            balance = scanner.nextDouble();
            double depositAmount = scanner.nextDouble();
            deposit(depositAmount);

            double withdrawAmount = scanner.nextDouble();
            double withdrawnAmount = withdraw(withdrawAmount);

            System.out.printf("Amount Withdrawn: %.1f\n" ,withdrawnAmount);
            balanceEnquiry();
        } catch (InvalidAmountException | InsufficientFundsException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    public void deposit(double amount) throws InvalidAmountException {
        if (amount <= 0) {
            throw new InvalidAmountException(amount + " is not valid");
        }
        balance = balance + amount;
    }

    public double withdraw(double amount) throws InsufficientFundsException {
        if (balance < amount) {
            throw new InsufficientFundsException("Insufficient funds");
        }
        balance = balance - amount;
        return amount;
    }

    public void balanceEnquiry() {
        System.out.printf("Current Balance: %.1f\n" ,balance);
    }
}
```

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

4. 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**