Rajalakshmi Engineering College

Name: RANNESH KHUMAR B R
Email: 240701422@rajalakshmi.edu.in
Roll no: 2116240701
Phone: 9042350670
Branch: REC
Department: CSE - Section 8
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

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**

2. 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**

3. 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 BalanceUpdater {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      try {
         double currentBalance = scanner.nextDouble();
         if (currentBalance < 0) {
            throw new InvalidAmountException("Invalid amount. Please enter a
positive initial balance.");
         }
         double updateAmount = scanner.nextDouble();
         updateBalance(currentBalance, updateAmount);
```

```
            System.out.println("Account balance updated successfully! New balance:
" + (currentBalance + updateAmount));
        } catch (InvalidAmountException | InsufficientBalanceException |
java.util.InputMismatchException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void updateBalance(double currentBalance, double
updateAmount)
            throws InvalidAmountException, InsufficientBalanceException {
        if (updateAmount < 0) {
            if (currentBalance + updateAmount < 0) {
                throw new InsufficientBalanceException("Insufficient balance.");
            }
        } else {
            currentBalance += updateAmount;
        }
    }
}
class InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}

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

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

4. 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**