Rajalakshmi Engineering College

Name: DINESH K V
Email: 241501048@rajalakshmi.edu.in
Roll no: 241501048
Phone: 7708632555
Branch: REC
Department: AI & ML - Section 1
Batch: 2028
Degree: B.E - AI & ML

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

```
// You are using Java
import java.util.Scanner;
class InvalidEmailException extends Exception {
    public InvalidEmailException(String message) {
        super(message);
    }
}
class EmailValidator {
    public void validateEmail(String email) throws InvalidEmailException {
        email = email.trim();
        if (!email.contains("@") || email.startsWith("@") || email.endsWith("@")) {
            throw new InvalidEmailException("Error: Invalid email format.");
        }
        String[] parts = email.split("@");
        if (parts.length != 2 || parts[0].isEmpty() || parts[1].isEmpty()) {
            throw new InvalidEmailException("Error: Invalid email format.");
        }
        if (!parts[1].contains(".")) {
            throw new InvalidEmailException("Error: Invalid email format.");
        }
        System.out.println("Email address is valid!");
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String email = sc.nextLine();
        EmailValidator validator = new EmailValidator();
        try {
            validator.validateEmail(email);
        } catch (InvalidEmailException e) {
            System.out.println(e.getMessage());
        }
        sc.close();
    }
}
```

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.util.Scanner;
import java.text.SimpleDateFormat;
import java.text.ParseException;
class InvalidDateOfBirthException extends Exception {
   public InvalidDateOfBirthException(String message) {
      super(message);
   }
}
class DateValidator {
   public void validateDate(String date) throws InvalidDateOfBirthException {
      SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
      sdf.setLenient(false);
      try{
         sdf.parse(date);
         System.out.println(date + " is a valid date of birth");
      }
      catch(ParseException e) {
         throw new InvalidDateOfBirthException("Invalid date: " + date);
      }
   }
}
```

```
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String date = sc.nextLine();
    DateValidator validator = new DateValidator();
    try {
      validator.validateDate(date);
    } catch (InvalidDateOfBirthException e) {
      System.out.println(e.getMessage());
    }
    sc.close();
  }
}
```

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

3. 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 validateCard(String cardNumber) throws
InvalidCreditCardException {
      if (cardNumber.length() != 16) {
         throw new InvalidCreditCardException("Error: Invalid credit card number
length.");
      }
      for (int i = 0; i < cardNumber.length(); i++) {
         if (!Character.isDigit(cardNumber.charAt(i))) {
            throw new InvalidCreditCardException("Error: Invalid credit card number
format.");
```

```
            }
        }
        System.out.println("Payment information updated successfully!");
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String cardNumber = sc.nextLine();
        CreditCardValidator validator = new CreditCardValidator();
        try {
            validator.validateCard(cardNumber);
        } catch (InvalidCreditCardException e) {
            System.out.println(e.getMessage());
        }
        sc.close();
    }
}
```

Status : Correct **_Marks : 10/10_**

4. 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 BankAccount {
   private double balance;
   public BankAccount(double balance) throws InvalidAmountException {
      if (balance < 0) {
         throw new InvalidAmountException("Error: Invalid amount. Please enter a
positive initial balance.");
      }
      this.balance = balance;
   }
   public void updateBalance(double amount) throws
InsufficientBalanceException {
      if (amount < 0) {
         if (balance + amount < 0) {
            throw new InsufficientBalanceException("Error: Insufficient balance.");
         } else {
            balance += amount;
         }
      } else {
         balance += amount;
      }
      System.out.println("Account balance updated successfully! New balance: " +
balance);
   }
}
public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      double initialBalance = sc.nextDouble();
      double updateAmount = sc.nextDouble();
      try {
         BankAccount account = new BankAccount(initialBalance);
         account.updateBalance(updateAmount);
      } catch (InvalidAmountException | InsufficientBalanceException e) {
         System.out.println(e.getMessage());
      }
      sc.close();
```

```
    }
}
```

Status : Correct *Marks : 10/10*