Rajalakshmi Engineering College

Name: Adithya B
Email: 240701018@rajalakshmi.edu.in
Roll no: 240701018
Phone: 9444117405
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

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

```
// You are using Java
import java.util.*;

class WeakPasswordException extends Exception {
    public WeakPasswordException(String msg) {
        super(msg);
    }
```

```
}

class MediumPasswordException extends Exception {
    public MediumPasswordException(String msg) {
        super(msg);
    }
}

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

        try {
            validatePassword(password);
            System.out.println("Password changed successfully!");
        }
        catch (WeakPasswordException e) {
            System.out.println("Error: Weak password. It must be at least 8 characters
long.");
        }
        catch (MediumPasswordException e) {
            System.out.println("Error: Medium password. It must include a mix of
uppercase letters, lowercase letters, and digits.");
        }
    }

    public static void validatePassword(String pwd) throws
WeakPasswordException, MediumPasswordException {


        if (pwd.length() < 8) {
            throw new WeakPasswordException("Weak");
        }


        boolean hasUpper = false, hasLower = false, hasDigit = false;

        for (char c : pwd.toCharArray()) {
            if (Character.isUpperCase(c)) hasUpper = true;
            if (Character.isLowerCase(c)) hasLower = true;
            if (Character.isDigit(c))    hasDigit = true;
```

```
        }

        if (!(hasUpper && hasLower && hasDigit)) {
            throw new MediumPasswordException("Medium");
        }
    }
}
```

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

```
// You are using Java
import java.util.*;

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

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

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

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

}

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

3. 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

```
// You are using Java
import java.util.*;

class InvalidDateOfBirthException extends Exception {
    public InvalidDateOfBirthException(String msg) {
        super(msg);
    }
}

public class Main {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String dob = sc.nextLine();

        try {
            validateDOB(dob);
            System.out.println(dob + " is a valid date of birth");
        }
        catch (InvalidDateOfBirthException e) {
            System.out.println("Invalid date: " + dob);
        }
    }

    public static void validateDOB(String dob) throws InvalidDateOfBirthException
{

        if (!dob.matches("\\d{2}-\\d{2}-\\d{4}")) {
            throw new InvalidDateOfBirthException("Invalid format");
        }

        String[] arr = dob.split("-");
        int day = Integer.parseInt(arr[0]);
        int month = Integer.parseInt(arr[1]);
        int year = Integer.parseInt(arr[2]);


        if (month < 1 || month > 12) {
            throw new InvalidDateOfBirthException("Invalid month");
        }

        int[] daysInMonth = { 31, (isLeap(year) ? 29 : 28), 31, 30, 31, 30, 31, 31, 30, 31,
```

```
30, 31 };


        if (day < 1 || day > daysInMonth[month - 1]) {
            throw new InvalidDateOfBirthException("Invalid day");
        }
    }

    private static boolean isLeap(int year) {
        return (year % 400 == 0) || (year % 4 == 0 && year % 100 != 0);
    }
}
```

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

```
// You are using Java
import java.util.*;

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

class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String msg) {
```

```
        super(msg);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        double initialBalance = sc.nextDouble();
        double updateAmount = sc.nextDouble();

        try {
            validateAndUpdate(initialBalance, updateAmount);
        }
        catch (InvalidAmountException e) {
            System.out.println("Error: " + e.getMessage());
        }
        catch (InsufficientBalanceException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    public static void validateAndUpdate(double balance, double change)
            throws InvalidAmountException, InsufficientBalanceException {


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

        if (change < 0) {
            if (balance + change < 0) {
                throw new InsufficientBalanceException("Insufficient balance.");
            } else {
                double newBalance = balance + change;
                System.out.println("Account balance updated successfully! New
balance: " + newBalance);
                return;
            }
        }
```

```
        double newBalance = balance + change;
        System.out.println("Account balance updated successfully! New balance: " +
newBalance);
    }
}
```

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