

# Rajalakshmi Engineering College

Name: Sruthi Dinesh  
Email: 240701536@rajalakshmi.edu.in  
Roll no: 240701536  
Phone: 7845725087  
Branch: REC  
Department: CSE - Section 7  
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

In an online shopping cart system, users can apply coupon codes during checkout to avail of discounts. However, to ensure the validity and security of coupon codes, the system enforces specific rules for their format. Your task is to implement a Java program named `CouponCodeValidator` that takes user input for a coupon code and validates it according to the specified rules.

##### Rules for Valid Coupon Code:

The coupon code must consist of exactly 10 characters. The coupon code must contain at least one alphabet (uppercase or lowercase) and at least one digit (0-9). Special characters are not allowed in the coupon code.

Implement a custom exception, `InvalidCouponException`, to handle cases where the entered coupon code does not meet the specified criteria.

### ***Input Format***

The input consists of a string *s*, representing the coupon code.

### ***Output Format***

The output is displayed in the following format:

If the entered coupon code meets the specified criteria, the program outputs

"Coupon code applied successfully!"

If the entered coupon code has less than or more than 10 characters it outputs

"Error: Invalid coupon code length. It must be exactly 10 characters."

If the entered coupon code contains only numeric or only alphabets it outputs

"Error: Invalid coupon code format. It must contain at least one alphabet and one digit."

If the entered coupon code contains special characters it outputs

"Error: Coupon code should not contain special characters."

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: ABCD123456

Output: Coupon code applied successfully!

### ***Answer***

```
import java.util.Scanner;
```

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

```

class CouponCodeValidator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            String couponCode = scanner.nextLine();
            validateCouponCode(couponCode);
            System.out.println("Coupon code applied successfully!");
        } catch (InvalidCouponException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void validateCouponCode(String couponCode) throws
    InvalidCouponException {

        if (containsSpecialCharacter(couponCode)) {
            throw new InvalidCouponException("Coupon code should not contain
special characters.");
        }

        if (!couponCode.matches("(?=[a-zA-Z])(?=.*\\d)[a-zA-Z0-9]{10}$")) {
            if (couponCode.length() != 10) {
                throw new InvalidCouponException("Invalid coupon code length. It must
be exactly 10 characters.");
            } else {
                throw new InvalidCouponException("Invalid coupon code format. It
must contain at least one alphabet and one digit.");
            }
        }
    }

    private static boolean containsSpecialCharacter(String str) {

        return str.matches(".*[^a-zA-Z0-9].*");
    }
}

```

**Status :** Correct

**Marks :** 10/10

## 2. 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

### 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 PasswordChangeSystem {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            String newPassword = scanner.nextLine();

            categorizePassword(newPassword);

            System.out.println("Password changed successfully!");
        } catch (WeakPasswordException | MediumPasswordException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void categorizePassword(String newPassword) throws
    WeakPasswordException, MediumPasswordException {
        if (newPassword.length() < 8) {
            throw new WeakPasswordException("Weak password. It must be at least
            8 characters long.");
        } else if (!containsUppercase(newPassword) || !
        containsLowercase(newPassword) || !containsDigit(newPassword)) {
            throw new MediumPasswordException("Medium password. It must
            include a mix of uppercase letters, lowercase letters, and digits.");
        }
    }

    private static boolean containsUppercase(String password) {
        return !password.equals(password.toLowerCase());
    }
}

```

```

private static boolean containsLowercase(String password) {
    return !password.equals(password.toUpperCase());
}

private static boolean containsDigit(String password) {
    for (char c : password.toCharArray()) {
        if (Character.isDigit(c)) {
            return true;
        }
    }
    return false;
}
}

```

**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 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