Rajalakshmi Engineering College

Name: Sivaranjanii M
Email: 241901109@rajalakshmi.edu.in
Roll no: 241901109
Phone: 7397469288
Branch: REC
Department: CSE (CS) - Section 2
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

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If the withdrawal is successful, print the amount withdrawn and the current balance, rounded off to one decimal place.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Output: Amount Withdrawn: 150.3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.*;
class InvalidAmountException extends Exception {
   public InvalidAmountException(String message) {
      super(message);
   }
}
class InsufficientFundsException extends Exception {
   public InsufficientFundsException(String message) {
      super(message);
   }
}
public class Main {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double balance = sc.nextDouble();
        double deposit = sc.nextDouble();
        double withdraw = sc.nextDouble();
        try {
            if (deposit <= 0) {
                throw new InvalidAmountException(deposit + " is not valid");
            } else {
                balance += deposit;
            }
            if (withdraw > balance) {
                throw new InsufficientFundsException("Insufficient funds");
            } else{
                balance -= withdraw;
                System.out.printf("Amount Withdrawn: %.1f%n", withdraw);
                System.out.printf("Current Balance: %.1f%n", balance);
            }
} catch (InvalidAmountException e) {
            System.out.println("Error: " + e.getMessage());

} catch (InsufficientFundsException e) {
            System.out.println("Error: " + e.getMessage());
        }
        sc.close();
    }
}
```

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

2. 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.*;
```

```
class InvalidCreditCardException extends Exception {
   public InvalidCreditCardException(String message) {
      super(message);
   }
}
public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String s = sc.nextLine();
      try {
         validateCreditCard(s);
         System.out.println("Payment information updated successfully!");
      } catch (InvalidCreditCardException e) {
         System.out.println("Error: " + e.getMessage());
      }
      sc.close();
   }
   public static void validateCreditCard(String cardNumber) throws
InvalidCreditCardException
{
      if (cardNumber.length() != 16) {
         throw new InvalidCreditCardException("Invalid credit card number
length.");
      }
      for (int i = 0; i < cardNumber.length(); i++) {
         if (!Character.isDigit(cardNumber.charAt(i))) {
            throw new InvalidCreditCardException("Invalid credit card number
format.");
         }
      }
   }
}
```

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 InvalidAmountException extends Exception {
    public InvalidAmountException(String message) {
        super(message);
    }
}
class InsufficientBalanceException extends Exception {
    public InsufficientBalanceException(String message) {
        super(message);
    }
}
public class Main {
    public static void updateBalance(double initialBalance, double
amountToUpdate)
            throws InvalidAmountException, InsufficientBalanceException
{
        if (initialBalance < 0) {
            throw new InvalidAmountException("Error: Invalid amount. Please enter a
positive initial balance.");
        }
        if (amountToUpdate < 0 && (initialBalance + amountToUpdate) < 0)
{
            throw new InsufficientBalanceException("Error: Insufficient balance.");
        }
        double newBalance = initialBalance + amountToUpdate;
        System.out.println("Account balance updated successfully! New balance: " +
newBalance);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double initialBalance = sc.nextDouble();
        double amountToUpdate = sc.nextDouble();
        try {
```

```
            updateBalance(initialBalance, amountToUpdate);
        } catch (InvalidAmountException e) {
            System.out.println(e.getMessage());
} catch (InsufficientBalanceException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

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.util.Scanner;
import java.text.SimpleDateFormat;
import java.text.ParseException;
class InvalidDateOfBirthException extends Exception {
   public InvalidDateOfBirthException(String message) {
      super(message);
   }
}
public class Main {
   public static void validateDateOfBirth(String dateOfBirth) throws
InvalidDateOfBirthException
{
      try {
         SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
         sdf.setLenient(false);
         sdf.parse(dateOfBirth);
} catch (ParseException e) {
         throw new InvalidDateOfBirthException("Invalid date: " + dateOfBirth);
      }
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String dateOfBirth = sc.nextLine();
      try {
         validateDateOfBirth(dateOfBirth);
         System.out.println(dateOfBirth + " is a valid date of birth");
} catch (InvalidDateOfBirthException e) {
         System.out.println(e.getMessage());
      }
   }
}
```

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