

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
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\_MCQ

Attempt : 1  
Total Mark : 15  
Marks Obtained : 13

#### Section 1 : MCQ

1. What will be the output for the following code?

```
import java.io.*;
```

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

```
class Test {  
    public static void main(String[] args) {  
        try {  
            int age = -5;  
            if (age < 0) {  
                throw new NegativeAgeException("Age cannot be negative");  
            }  
        }  
    }  
}
```

```

    }
    } catch (NegativeAgeException e) {
        System.out.println(e.getMessage());
    }
}

```

**Answer**

Age cannot be negative

**Status :** Correct

**Marks :** 1/1

2. What will be the output for the following code?

```

import java.io.*;

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

class Test {
    public static void main(String[] args) {
        try {
            int temperature = 110;
            if (temperature > 100) {
                throw new TemperatureTooHighException("Temperature too
high");
            }
        } catch (TemperatureTooHighException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

**Answer**

Temperature too high

**Status :** Correct

**Marks :** 1/1

3. How do you create an unchecked custom exception?

**Answer**

By extending RuntimeException

**Status : Correct**

**Marks : 1/1**

4. What is the purpose of a custom exception in Java?

**Answer**

To create user-defined exceptions for specific scenarios

**Status : Correct**

**Marks : 1/1**

5. what is the output of the following code?

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

```
class Test {  
    public static void main(String[] args) {  
        try {  
            throw new MyException("Error occurred");  
        } catch (MyException e) {  
            System.out.println(e);  
        }  
    }  
}
```

**Answer**

MyException: Error occurred

**Status : Correct**

**Marks : 1/1**

6. What will be the output for the following code?

```
import java.io.*;
```

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

```
class Test {  
    public static void main(String[] args) {  
        try {  
            int stock = 0;  
            if (stock == 0) {  
                throw new OutOfStockException("Item is out of stock");  
            }  
        } catch (OutOfStockException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

**Answer**

Item is out of stock

**Status :** Correct

**Marks :** 1/1

7. What will be the output for the following code?

```
import java.io.*;
```

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

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

```

try {
    int age = 17;
    if (age < 18) {
        throw new UnderageException("Underage, cannot proceed");
    }
} catch (UnderageException e) {
    System.out.println(e.getMessage());
}
}
}

```

**Answer**

Underage, cannot proceed

**Status :** Correct

**Marks :** 1/1

8. What will be the output for the following code?

```

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

```

```

class Test {
    public static void main(String[] args) {
        try {
            String username = "abc";
            if (username.length() < 5) {
                throw new InvalidUsernameException("Username must be at
least 5 characters long");
            }
        } catch (InvalidUsernameException e) {
            System.out.println(e.getMessage());
        }
    }
}

```

**Answer**

Username must be at least 5 characters long

**Status :** Correct

**Marks :** 1/1

9. What will be the output of the following code?

```
class MyException extends Exception {  
    public MyException() {  
        super("Default Exception Message");  
    }  
}  
  
class Test {  
    public static void main(String[] args) {  
        try {  
            throw new MyException();  
        } catch (MyException e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

**Answer**

Default Exception Message

**Status :** Correct

**Marks :** 1/1

10. Which keyword is used to explicitly throw a custom exception?

**Answer**

throw

**Status :** Correct

**Marks :** 1/1

11. what is the output of the following code?

```
class MyException extends Exception {  
    public MyException(String message) {
```

```

        super(message);
    }
}

class Test {
    static void check() throws MyException {
        throw new MyException("Custom Exception Occurred");
    }

    public static void main(String[] args) {
        try {
            check();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```

**Answer**

Custom Exception Occurred

**Status :** Correct

**Marks :** 1/1

12. What will be the output for the following code?

```

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

class Test {
    public static void main(String[] args) {
        try {
            int age = 15;
            if (age < 18) {
                throw new InvalidVotingAgeException("You are not eligible to
vote");
            }
        }
    }
}

```

```
        System.out.println("Eligible to vote");
    } catch (InvalidVotingAgeException e) {
        System.out.println(e.getMessage());
    }
}
}
```

**Answer**

Exception occurred

**Status :** Wrong

**Marks :** 0/1

13. Which of the following is true about custom exceptions?

**Answer**

Custom exceptions must extend either Exception or RuntimeException

**Status :** Correct

**Marks :** 1/1

14. What will happen if a checked custom exception is thrown inside a method without being caught or declared?

**Answer**

Compilation Error

**Status :** Correct

**Marks :** 1/1

15. What will be the output for the following code?

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

```
class Test {
    public static void main(String[] args) {
        try {
```



```
double balance = -500;  
if (balance < 0) {  
    throw new NegativeBalanceException("Balance cannot be  
negative");  
}  
} catch (NegativeBalanceException e) {  
    System.out.println("Error: " + e.getMessage());  
}  
}  
}
```

**Answer**

null

**Status :** Wrong

**Marks :** 0/1

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 8\_Q1

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

Write a program to validate the email address and display suitable exceptions if there is any mistake.

Create 3 custom exception classes as below

DotException AtTheRateException DomainException

A typical email address should have a "." character, and a "@" character, and also the domain name should be valid. Valid domain names for practice be 'in', 'com', 'net', or 'biz'.

Display Invalid Dot usage, Invalid @ usage, or Invalid Domain message based on email id.

Get the email address from the user, validate the email by checking the

above-mentioned criteria, and print the validity status of the input email address.

### ***Input Format***

The first line of input contains the email to be validated.

### ***Output Format***

The output prints a Valid email address or an Invalid email address along with the suitable exception

If email ends with . or contains not exactly one . after @, it throws:

DotException: Invalid Dot usage

Invalid email address

If @ appears not exactly once, it throws:

AtTheRateException: Invalid @ usage

Invalid email address

If the part after the last dot is not among accepted domains:

DomainException: Invalid Domain

Invalid email address

If all conditions satisfied then print:

Valid email address

Refer to the sample input and output for format specifications.

### **Sample Test Case**

Input: sample@gmail.com

Output: Valid email address

### **Answer**

```
import java.util.*;

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

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

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

public class Main {
    public static void validateEmail(String email)
        throws DotException, AtTheRateException, DomainException {

        int atCount = email.length() - email.replace("@", "").length();
        if (atCount != 1 || email.startsWith("@") || email.endsWith("@") ||
            email.contains("@@")) {
            throw new AtTheRateException("AtTheRateException: Invalid @ usage");
        }

        String[] parts = email.split("@");
        if (parts.length != 2) {
```

```

        throw new AtTheRateException("AtTheRateException: Invalid @ usage");
    }

    String domainPart = parts[1];

    if (!domainPart.contains(".") || email.endsWith(".") || email.startsWith(".") ||
    email.contains("..")) {
        throw new DotException("DotException: Invalid Dot usage");
    }

    int lastDotIndex = email.lastIndexOf(".");
    if (lastDotIndex == -1 || lastDotIndex == email.length() - 1) {
        throw new DotException("DotException: Invalid Dot usage");
    }

    String extension = email.substring(lastDotIndex + 1);
    List<String> validDomains = Arrays.asList("in", "com", "net", "biz");

    if (!validDomains.contains(extension)) {
        throw new DomainException("DomainException: Invalid Domain");
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String email = sc.nextLine();
    try {
        validateEmail(email);
        System.out.println("Valid email address");
    }
    catch (DotException | AtTheRateException | DomainException e) {
        System.out.println(e.getMessage());
        System.out.println("Invalid email address");
    }
    sc.close();
}
}

```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 8\_Q2

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

Elsa, a busy professional, is using a scheduling application to plan her meetings efficiently. The application requires users to input meeting durations in minutes, ensuring that the duration is a positive integer and does not exceed 240 minutes (4 hours). Elsa needs a program to assist her in scheduling meetings securely with proper exception handling.

Create a Java class named ElsaMeetingScheduler. Implement a custom exception: InvalidDurationException for invalid meeting duration entries. Implement the main method to interactively take user input for a meeting duration. Implement the validateMeetingDuration method to validate the meeting duration based on the specified rules and throw a custom exception if the validation fails. Print appropriate success or error messages based on the meeting duration.

Implement a custom exception, `InvalidDurationException`, to handle cases where the entered meeting duration does not meet the specified criteria.

### ***Input Format***

The input consists of an integer value 'n', representing the meeting duration.

### ***Output Format***

The output is displayed in the following format:

If the entered meeting duration meets the specified criteria, the program outputs

"Meeting scheduled successfully!"

If the entered meeting duration is invalid, the program outputs an error message indicating the issue.

"Error: Invalid meeting duration. Please enter a positive integer not exceeding 240 minutes (4 hours)."

Refer to the sample output for formatting specifications.

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

Input: 120

Output: Meeting scheduled successfully!

### ***Answer***

```
import java.util.*;

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

public class Main{

    public static void Checktime(int time) throws InvalidDurationException{
        if ((time <= 0) || (time > 240) ){
            throw new InvalidDurationException ("Error: Invalid meeting duration.
```

Please enter a positive integer not exceeding 240 minutes (4 hours).");

```
}  
}
```

```
public static void main(String args[]){  
    Scanner in = new Scanner(System.in);  
    try{  
        int time = in.nextInt();  
        Checktime(time);  
        System.out.println("Meeting scheduled successfully!");  
    }  
    catch(InvalidDurationException e){  
        System.out.println(e.getMessage());  
    }  
}
```

**Status :** Correct

**Marks :** 10/10



# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 8\_Q3

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

In a user registration system, there is a requirement to implement a username validation module. Users attempting to register must adhere to specific criteria for their usernames to be considered valid.

Your task is to develop a program that takes user input for a desired username and validates it according to the following rules:

The username must not contain any spaces. The username must be at least 5 characters long.

Implement a custom exception, `InvalidUsernameException`, to handle cases where the entered username does not meet the specified criteria.

##### ***Input Format***

The input consists of a string S, representing the desired username.

### **Output Format**

If the username is valid, print "Username is valid: [S]".

If the username is invalid:

1. If the username is short, print "Invalid Username: Username must be at least 5 characters long"
2. If the username contains spaces, print "Invalid Username: Username cannot contain spaces"

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: John

Output: Invalid Username: Username must be at least 5 characters long

### **Answer**

```
import java.util.*;
```

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

```
public class Main{
```

```
    public static void validUsername (String name) throws  
InvalidUsernameException{  
        if (name.contains(" ")){  
            throw new InvalidUsernameException("Invalid Username: Username  
cannot contain spaces");  
        }  
        else if (name.length() < 5){  
            throw new InvalidUsernameException("Invalid Username: Username must  
be at least 5 characters long");  
        }  
    }  
}
```

```
}  
public static void main(String args[]){  
    Scanner in = new Scanner(System.in);  
    try {  
        String name = in.nextLine();  
        validUsername(name);  
        System.out.println("Username is valid: "+name);  
    }  
    catch (InvalidUsernameException e){  
        System.out.println(e.getMessage());  
    }  
}  
}
```

**Status :** Correct

**Marks : 10/10**

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 8\_Q4

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

A local municipality is implementing an online voting system for a community event and wants to ensure that only eligible voters (those aged 18 or older) can participate.

Your task is to develop a program that validates the age of individuals attempting to vote online. If the user's age is below 18, the program should throw a custom exception, `InvalidAgeException`, preventing them from casting their vote. If the input is invalid, catch the appropriate `InputMismatchException` and print the in-built exception message.

##### ***Input Format***

The input consists of an integer representing the age.

##### ***Output Format***

If the age is 18 or older, print "Eligible to vote"

If the age is below 18, print "Exception occurred: InvalidAgeException: Age is not valid to vote"

If there is any other type of exception, print "An error occurred: " followed by the in-built exception message.

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 20

Output: Eligible to vote

### **Answer**

```
import java.util.*;
class InvalidAgeException extends Exception {
    public InvalidAgeException (String message) {
        super(message);
    }
}
public class Main{
    public static void checkage(int age) throws InvalidAgeException{
        if(age <= 18){
            throw new InvalidAgeException ("Exception occurred:
InvalidAgeException: Age is not valid to vote");
        }
    }
    public static void main(String args[]){
        try{
            Scanner in= new Scanner(System.in);
            int age = in.nextInt();
            checkage(age);
            System.out.println("Eligible to vote");
        }
        catch (InvalidAgeException e){
            System.out.println(""+e.getMessage());
        }
    }
}
```

```
        catch (Exception e){  
            System.out.println("An error occurred: "+e);  
        }  
    }  
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
Batch: 2028  
Degree: B.E - CSE (CS)

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 8\_Q5

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

In a file management system, users are required to provide a valid file name when creating new files. The system enforces specific rules for file names to maintain consistency and avoid potential issues. Your task is to implement a Java program named FileNameValidator that takes user input for a file name and validates it according to the specified rules.

Rules for Valid File Name:

The file name must consist of alphanumeric characters (letters and digits) only. The file name must have a minimum length of 3 characters.

Implement a custom exception, FileNameValidator, to handle cases where the entered filename does not meet the specified criteria.

***Input Format***

The input consists of a string S, representing the desired filename.

### **Output Format**

The output is displayed in the following format:

If the entered file name meets the specified criteria, the program outputs

"Valid file name"

If the entered file name does not meet the criteria and triggers the `InvalidFileNameException`, the program outputs

"Error: Invalid file name. It must be alphanumeric and have a minimum length of 3 characters."

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: myfile123

Output: Valid file name

### **Answer**

```
import java.util.*;

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

public class Main {

    public static void validateFileName(String fileName) throws
InvalidFileNameException {
        if (fileName.length() < 3 || !fileName.matches("[A-Za-z0-9]+")) {
            throw new InvalidFileNameException("Error: Invalid file name. It must be
alphanumeric and have a minimum length of 3 characters.");
        }
    }
}
```



```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    try {  
        String fileName = sc.nextLine();  
        validateFileName(fileName);  
        System.out.println("Valid file name");  
    }  
    catch (InvalidFileNameException e) {  
        System.out.println(e.getMessage());  
    }  
    sc.close();  
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
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\_PAH

Attempt : 1  
Total Mark : 40  
Marks Obtained : 40

#### Section 1 : Coding

##### 1. Problem Statement

Enigma is developing a simple web application that takes a user-input URL, validates it, and throws a custom exception `InvalidURLException` if the URL does not start with "http://" or "https://".

The main method prompts the user for input, validates the URL, and prints whether it is valid or not.

##### ***Input Format***

The input consists of a string, representing the URL entered by the user.

##### ***Output Format***

The output displays one of the following results:

If the entered URL is valid according to the specified format, the program prints:

"[URL] is a valid URL"

If the entered URL is not valid according to the specified format, the program prints:

"Invalid URL format: [URL]"

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: `http://www.example.com`

Output: `http://www.example.com is a valid URL`

### **Answer**

```
import java.util.*;
```

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

```
public class Main {
```

```
    public static void validateURL(String url) throws InvalidURLException {  
        if (!(url.startsWith("http://") || url.startsWith("https://"))) {  
            throw new InvalidURLException("Invalid URL format: " + url);  
        }  
    }  
}
```

```
    public static void main(String[] args) {  
        Scanner in = new Scanner(System.in);  
        String url = in.nextLine();  
    }  
}
```

```
try {
    validateURL(url);
    System.out.println(url + " is a valid URL");
}
catch (InvalidURLExceptionFormatException e) {
    System.out.println(e.getMessage());
}
}
```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Daniel is developing a program to verify the age of users. He wants to ensure that the entered age is within a valid range. Write a program to help Daniel implement this age-checking feature using custom exceptions.

Daniel needs a program that takes an integer input representing a person's age. If the age is between 0 and 150 (inclusive), the program should print "Age is valid!". If the age is less than 0 or greater than 150, the program should throw a custom exception (InvalidAgeException) with the message "Invalid age. Please enter an age between 0 and 150."

Implement a custom exception, InvalidAgeException, to handle cases where the entered age does not meet the specified criteria.

### **Input Format**

The input consists of an integer value 'n', representing the age.

### **Output Format**

The output is displayed in the following format:

If the age is valid (between 0 and 150, inclusive), print

"Age is valid!".

If the age is invalid, print

"Error: Invalid age. Please enter an age between 0 and 150."

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 45

Output: Age is valid!

### **Answer**

```
import java.util.Scanner;

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

public class Main{

    public static void ValidAge(int age) throws InvalidAgeException {
        if(age < 0 || age > 150){
            throw new InvalidAgeException ("Error: Invalid age. Please enter an age
between 0 and 150.");
        }
    }

    public static void main (String args[]){
        Scanner in = new Scanner(System.in);
        try{
            int age = in.nextInt();
            ValidAge(age);
            System.out.println("Age is valid!");
        }
        catch (InvalidAgeException e){
            System.out.println(e.getMessage());
        }
    }
}
```

**Status : Correct**

**Marks : 10/10**

### 3. Problem Statement

You are tasked to create a program that defines a custom exception `GradeException`. The program should include a `Student` class with fields for the student's name, age, and grade. Implement a method in the `Student` class that checks the grade, and if the grade is below 40, it should throw a `GradeException`. Otherwise, it should display the student's details.

#### ***Input Format***

The input consists of three parameters in separate lines:

1. A string representing the student's name.
2. An integer representing the student's age.
3. An integer representing the student's grade.

#### ***Output Format***

The output will display the student's details if the grade is valid.

If the grade is below 40, the program will display an error message "Grade is below 40".

Refer to the sample output for formatting specifications.

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

Input: Alice

20

85

Output: Name: Alice

Age: 20

Grade: 85

#### ***Answer***

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

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

```

    }

    public class Main{

        public static void Gradevalid(int grade) throws GradeException{
            if (grade < 40){
                throw new GradeException("Grade is below 40");
            }
        }
    }

```

```

        public static void main(String args[]){
            Scanner in = new Scanner(System.in);
            try{
                String name = in.nextLine();
                int age = in.nextInt();
                int grade = in.nextInt();
                Gradevalid(grade);
                System.out.println("Name: "+name);
                System.out.println("Age: "+age);
                System.out.println("Grade: "+grade);
            }
            catch (GradeException e){
                System.out.println(e.getMessage());
            }
        }
    }

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

An HR software system is being developed to process employee payrolls. During payroll processing, the system must ensure that no employee has a negative salary and that no employee's salary exceeds 2,00,000. If either condition occurs, the system should throw a custom exception.

Create a custom exception InvalidSalaryException and a class Employee that processes salary according to the following rules:

If salary < 0, throw `InvalidSalaryException` with the message: "Salary cannot be negative". If salary > 200000, throw `InvalidSalaryException` with the message: "Salary exceeds threshold limit". Otherwise, display: "Salary processed successfully for <empName>: <salary>".

The payroll processing should always display: "Payroll process completed" at the end, regardless of whether an exception occurs.

### ***Input Format***

The first line of input contains an integer representing the employee ID.

The second line contains a string representing the employee's name.

The third line contains a floating-point number representing the salary of the employee.

### ***Output Format***

If the salary is valid: "Salary processed successfully for <empName>: <salary>"

"Payroll process completed"

If the salary is invalid: "<Exception Message>"

"Payroll process completed"

Refer to the sample output for formatting specifications.

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

Input: 101

Rahul

150000.0

Output: Salary processed successfully for Rahul: 150000.0

Payroll process completed

### ***Answer***

```
import java.util.*;
```

```
class InvalidSalaryException extends Exception{
```



```

    public InvalidSalaryException (String message){
        super(message);
    }
}

public class Main{

    public static void ValidSalary(double salary) throws InvalidSalaryException{
        if (salary < 0 ){
            throw new InvalidSalaryException("Salary cannot be negative");
        }
        else if (salary > 200000){
            throw new InvalidSalaryException("Salary exceeds threshold limit");
        }
    }

    public static void main(String args[]){
        Scanner in = new Scanner(System.in);
        try{
            int id = in.nextInt();
            in.nextLine();
            String name = in.nextLine();
            double salary = in.nextDouble();
            ValidSalary(salary);
            System.out.println("Salary processed successfully for "+name+":
"+salary);
        }
        catch (InvalidSalaryException e){
            System.out.println(e.getMessage());
        }
        finally{
            System.out.println("Payroll process completed");
        }
    }
}

```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: Bharatth S  
Email: 241901019@rajalakshmi.edu.in  
Roll no: 241901019  
Phone: 9962752798  
Branch: REC  
Department: CSE (CS) - Section 1  
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

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.*;
```

// Custom Exception for invalid initial balance

```
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 main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
  
        double initialBalance = sc.nextDouble();  
        double updateAmount = sc.nextDouble();  
  
        try {  
            if (initialBalance < 0) {  
                throw new InvalidAmountException("Error: Invalid amount. Please enter  
a positive initial balance.");  
            }  
  
            if (updateAmount < 0) {  
                double newBalance = initialBalance + updateAmount; // since  
updateAmount is negative  
                if (newBalance < 0) {  
                    throw new InsufficientBalanceException("Error: Insufficient  
balance.");  
                } else {  
                    System.out.println("Account balance updated successfully! New  
balance: " + newBalance);  
                }  
            }  
            else {  
                double newBalance = initialBalance + updateAmount;  
                System.out.println("Account balance updated successfully! New  
balance: " + newBalance);  
            }  
        }  
    }  
}
```

```

    }
    catch (InvalidAmountException e) {
        System.out.println(e.getMessage());
    }
    catch (InsufficientBalanceException e) {
        System.out.println(e.getMessage());
    }

    sc.close();
}
}

```

**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.*;
```

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

```
public class Main {  
    public static void validateEmail(String email) throws InvalidEmailException {  
        email = email.trim();
```

```
        if (!email.contains("@") || email.indexOf('@') != email.lastIndexOf('@')) {  
            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.");  
        }
```

```

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

        try {
            validateEmail(email);
            System.out.println("Email address is valid!");
        } catch (InvalidEmailException e) {
            System.out.println(e.getMessage());
        }

        sc.close();
    }
}

```

**Status :** Correct

**Marks :** 10/10

### 3. 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("Error: " + deposit + " is not valid");
    }
    balance += deposit;
    if (withdraw > balance) {
        throw new InsufficientFundsException("Error: Insufficient funds");
    }
    balance -= withdraw;
    System.out.println("Amount Withdrawn: " + withdraw);
    System.out.printf("Current Balance: %.1f", balance);
}
catch (InvalidAmountException e) {
    System.out.println(e.getMessage());
}
catch (InsufficientFundsException e) {
    System.out.println(e.getMessage());
}

sc.close();
}
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. 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.*;
```

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

public class Main {
    public static void validateCoupon(String code) throws InvalidCouponException
    {
        if (code.length() != 10) {
            throw new InvalidCouponException("Error: Invalid coupon code length. It
            must be exactly 10 characters.");
        }

        boolean hasLetter = false;
        boolean hasDigit = false;

        for (char c : code.toCharArray()) {
            if (Character.isLetter(c)) {
                hasLetter = true;
            } else if (Character.isDigit(c)) {
                hasDigit = true;
            } else {
                throw new InvalidCouponException("Error: Coupon code should not
                contain special characters.");
            }
        }

        if (!hasLetter || !hasDigit) {
            throw new InvalidCouponException("Error: Invalid coupon code format. It
            must contain at least one alphabet and one digit.");
        }
    }

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

        try {
            validateCoupon(code);
            System.out.println("Coupon code applied successfully!");
        } catch (InvalidCouponException e) {
```

```
        System.out.println(e.getMessage());
    }
    sc.close();
}
```

**Status :** Correct

**Marks : 10/10**