ASSIGNMENT-3

Name - Rajat Rao

Roll no- 2401201067

Course – BCA(AI&DS)

Input-

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


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


class Student {

    private Integer rollNumber;
    private String studentName;
    private Integer[] marks = new Integer[3];

    public Student(Integer rollNumber, String studentName, Integer[] marks)
            throws InvalidMarksException {

        this.rollNumber = rollNumber;
        this.studentName = studentName;
        this.marks = marks;

        validateMarks();
    }


```

```
    public void validateMarks() throws InvalidMarksException {
        for (int i = 0; i < marks.length; i++) {
            if (marks[i] == null || marks[i] < 0 || marks[i] > 100) {
                throw new InvalidMarksException(
                        "Invalid marks for subject " + (i + 1) + ": " + marks[i]
                );
            }
        }
    }

    public double calculateAverage() {
        int sum = 0;
        for (int m : marks) {
            sum += m;
        }
        return sum / 3.0;
    }

    public String getResultStatus() {
        for (int m : marks) {
            if (m < 35) return "Fail";
        }
        return "Pass";
    }

    public Integer getRollNumber() {
        return rollNumber;
    }
    }

    public void displayResult() {
        System.out.println("Roll Number : " + rollNumber);
        System.out.println("Student Name: " + studentName);

        System.out.print(s: "Marks       : ");
        for (int m : marks) {
            System.out.print(m + " ");
        }

        System.out.println("\nAverage     : " + calculateAverage());
        System.out.println("Result      : " + getResultStatus());
    }
}


public class ResultManager {

    private Student[] students = new Student[50];
    private int count = 0;
    private Scanner sc = new Scanner(System.in);


    public void addStudent() {
        try {
```

```
            System.out.print(s: "Enter Roll Number: ");
            int roll = sc.nextInt();
            sc.nextLine();

            System.out.print(s: "Enter Student Name: ");
            String name = sc.nextLine();

            Integer[] marks = new Integer[3];

            for (int i = 0; i < 3; i++) {
                System.out.print("Enter marks for subject " + (i + 1) + ": ");
                marks[i] = sc.nextInt();
            }


            Student s = new Student(roll, name, marks);
            students[count++] = s;

            System.out.println(x: "Student added successfully. Returning to main menu...");

        } catch (InvalidMarksException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (InputMismatchException e) {
            System.out.println(x: "Error: Invalid input type. Please enter numeric values.");
            sc.nextLine();
        } catch (Exception e) {
```

```
            System.out.println("Unexpected error: " + e.getMessage());
        }
    }


    public void showStudentDetails() {
        try {
            System.out.print(s: "Enter Roll Number to search: ");
            int roll = sc.nextInt();

            boolean found = false;

            for (int i = 0; i < count; i++) {
                if (students[i].getRollNumber() == roll) {
                    students[i].displayResult();
                    found = true;
                    break;
                }
            }

            if (!found) {
                System.out.println(x: "Student not found.");
            }

        } catch (InputMismatchException e) {
            System.out.println(x: "Error: Invalid roll number format.");
```

```
            sc.nextLine();
        } catch (Exception e) {
            System.out.println("Unexpected error: " + e.getMessage());
        }
    }

    public void mainMenu() {

        try {
            int choice;

            do {
                System.out.println(x: "\n===== Student Result Management System =====");
                System.out.println(x: "1. Add Student");
                System.out.println(x: "2. Show Student Details");
                System.out.println(x: "3. Exit");
                System.out.print(s: "Enter your choice: ");

                choice = sc.nextInt();

                switch (choice) {
                    case 1:
                        addStudent();
                        break;
```

```
                    case 2:
                        showStudentDetails();
                        break;

                    case 3:
                        System.out.println(x: "Exiting program. Thank you!");
                        break;

                    default:
                        System.out.println(x: "Invalid choice. Try again.");
                }

            } while (choice != 3);

        } finally {
            sc.close();
            System.out.println(x: "Scanner closed. Program terminated.");
        }
    }

    Run | Debug
    public static void main(String[] args) {
        ResultManager rm = new ResultManager();
        rm.mainMenu();
    }
}
```

Output-

```
===== Student Result Management System =====
1. Add Student
2. Show Student Details
3. Exit
Enter your choice: 1
Enter Roll Number: 22
Enter Student Name: Rajat Rao
Enter marks for subject 1: 55
Enter marks for subject 2: 45
Enter marks for subject 3: 89
Student added successfully. Returning to main menu...
```

```
===== Student Result Management System =====
1. Add Student
2. Show Student Details
3. Exit
Enter your choice: 2
Enter Roll Number to search: 22
Roll Number : 22
Student Name: Rajat Rao
Marks       : 55 45 89
Average     : 63.0
Result      : Pass
```

```
===== Student Result Management System =====
1. Add Student
2. Show Student Details
3. Exit
Enter your choice: 3
Exiting program. Thank you!
Scanner closed. Program terminated.
PS C:\Users\rajat\OneDrive\Desktop\rajat java>
```

EXPLANATION –

1. Custom Exception (InvalidMarksException)

This project includes a user-defined exception called InvalidMarksException.
It is used to handle cases where a student enters invalid marks (less than 0 or greater than 100).

This exception class extends the built-in Exception class, making it a checked exception.

When marks are out of valid range, the system intentionally throws this custom exception.

Using custom exceptions makes the program more descriptive and meaningful because it communicates exactly what went wrong.

⭐ 2. Student Class Explanation

The *Student* class represents one student and contains:

Attributes

Roll Number

Student Name

Marks array (for 3 subjects)

Functionality

Validation of Marks
The class validates that each mark lies between 0 and 100.
If not, it throws InvalidMarksException.

Calculate Average
It computes the average of the three subject marks.

Result Status (Pass/Fail)
A student is considered:

Pass if all marks are 35 or above

Fail if any one subject is below 35

Display Result
It prints roll number, name, marks, average, and pass/fail status.

The Student class demonstrates:

Data encapsulation

Custom validation

Use of throw/throws

Checked exception handling

⭐ 3. ResultManager Class (Main System)

This is the most important class because it controls the entire system.
It contains:

1. Array of Student Objects

Stores up to 50 students.

Ensures simple storage without using advanced collections.

2. Scanner Object

Used for user input.

Closed using a finally block to avoid memory leaks.

3. Menu System

A menu-driven interface allows the user to perform operations:

Add Student

Show Student Details

Exit

It uses a loop to continuously show the menu until the user selects Exit.

⭐ 4. Exception Handling in This Project

The project uses multiple types of exceptions:

1. Custom Checked Exception:

InvalidMarksException
Thrown when marks are invalid.

2. Built-in Unchecked Exception:

InputMismatchException
Occurs when user enters text instead of numeric values.

3. General Exception Handling:

A generic catch block handles any unexpected error.

4. try–catch–finally

Used in all major operations.

finally is used to close the Scanner and print a final message.

This demonstrates robust and structured exception handling.

⭐ 5. How the Program Works (Flow Explanation)

The program starts and displays the main menu.

When the user chooses Add Student:

They enter roll number, name, and marks.

Marks are validated.

If valid → student is stored.

If invalid → custom error message is displayed.

When user chooses Show Student Details:

They enter the roll number.

The system searches in the array.

Displays details if found, otherwise shows “Student not found”.

Choosing Exit stops the program.

finally block runs last and closes input resources.

⭐ 6. Concepts Demonstrated

A. Object-Oriented Programming

Classes and Objects

Encapsulation

Modular structure

B. Exception Handling

try

catch

finally

throw

throws

Custom exception

C. Input Validation

Checks for improper marks

Checks for wrong data entry

D. Menu-Driven Program

Repeated user interaction

Loop and conditional logic

⭐ 7. Learning Outcomes

Students completing this project learn to:

Identify, handle, and prevent runtime errors.

Create and use user-defined exceptions.

Understand checked vs unchecked exceptions.

Write clean and modular Java code.

Build real-world applications with strong error-handling logic.

Develop interactive console-based applications.