Problem statement: This assignment aims to build a robust Student Management System using Java by incorporating exception handling, multithreading, and wrapper classes. The program validates all inputs using try-catch-finally blocks and custom exceptions to prevent runtime errors. A separate Loader thread is created using the Runnable interface to simulate a delay during data processing, improving user interactivity. Wrapper classes such as Integer and Double are used for type conversion and autoboxing. The program successfully collects, validates, and displays student details along with a calculated grade, ensuring secure, responsive, and well-structured execution 1. Try–Catch–Finally Concept

The program demonstrates how errors are handled in Java using try, catch, and finally blocks.

- The try block contains statements that may cause errors.
- The catch block handles the exception and prevents program termination.
- The finally block runs every time, whether an exception occurs or not.
- This shows safe, structured error handling.

2. Multiple Catch Blocks

The program includes more than one catch block to handle different types of exceptions separately.
This helps the program react differently to different errors such as:

- Arithmetic errors
- Input errors
- Null value errors

It ensures more accurate and specific exception handling.

3. Custom Exception (User-defined Exception)

A custom exception class (StudentNotFoundException) is created to handle cases where a searched student does not exist in the record.

This shows how Java allows us to create our own exception types, giving clearer and more meaningful error messages.
It also improves readability and error tracking in real applications.

4. Interface Implementation (RecordActions)

An interface is used to specify the set of operations related to student records.
The StudentManager class implements the interface and provides concrete definitions for the methods such as:

- Add
- Search • Delete

This demonstrates abstraction and ensures that all required operations are implemented.

5. Wrapper Classes (Integer & Double)

Wrapper classes are used to convert primitive data types into object types.
The program uses:

- Integer → for converting strings to integers
- Double → for converting marks

This shows the use of wrapper classes for parsing and performing data conversions while handling exceptions safely.

6. Multithreading using Runnable Interface

A separate Loader class implements the Runnable interface to demonstrate multithreading.
This class runs a small task such as loading or simulating background work.

It shows how Java can run multiple tasks at the same time:

- One thread handles student operations
- Another thread performs background loading

This improves performance and gives experience with concurrent programming.

7. StudentManager Class

This class manages the list of students.
Key operations include:

- Adding new records
- Viewing all records
- Searching for a student
- Throwing custom exceptions when needed

It combines error handling, wrapper classes, and lists to form a small but complete management system.

8. Safe Scanner Handling and User Interaction

The program uses Scanner for user input.
It is handled safely:

- Not closed early
- Only closed at the end
- Validated using exception handling

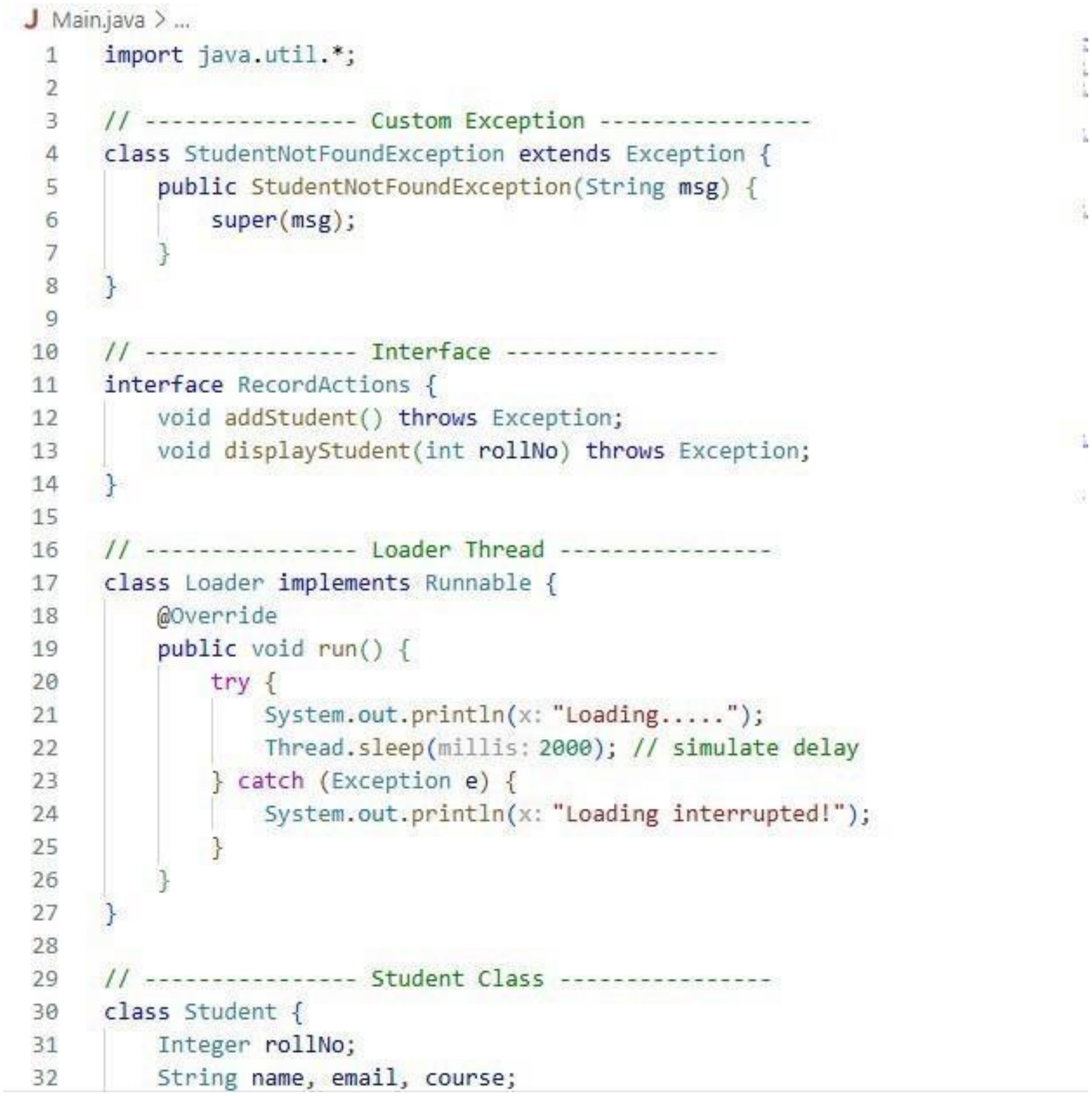
This avoids warnings and ensures smooth user interaction.

9. Overall Java Concepts Demonstrated

Concept	How It Is Used
Exception Handling	try, catch, finally
Multiple Catch Blocks	Handles different errors separately
Custom Exception	StudentNotFoundException class

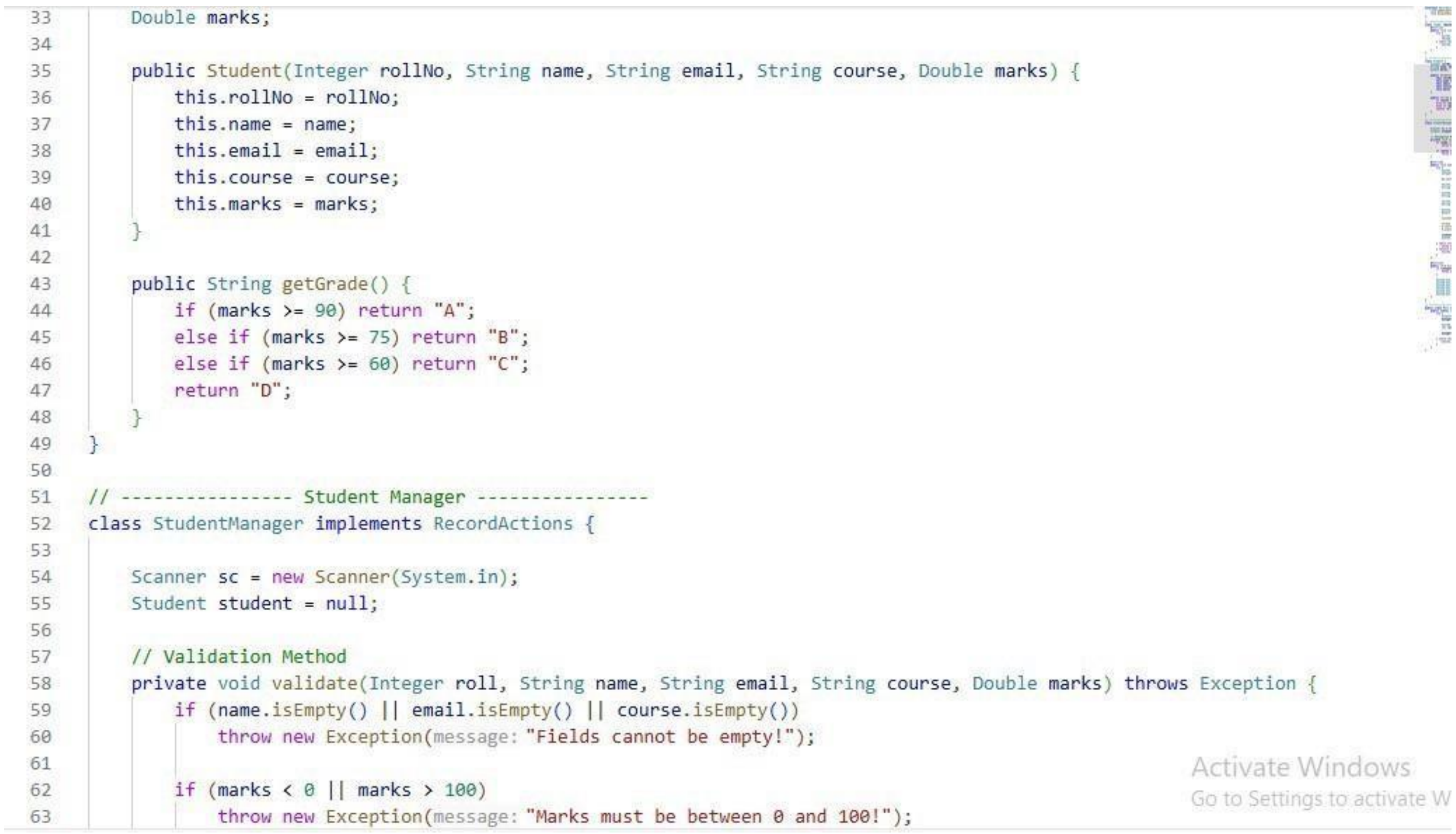
Interface	RecordActions implemented by StudentManager
Wrapper Classes	Integer & Double for parsing
Polymorphism	Runnable interface in multithreading
Multithreading	Loader thread simulating background activity
Dynamic Data Structure	ArrayList for storing students

Input:

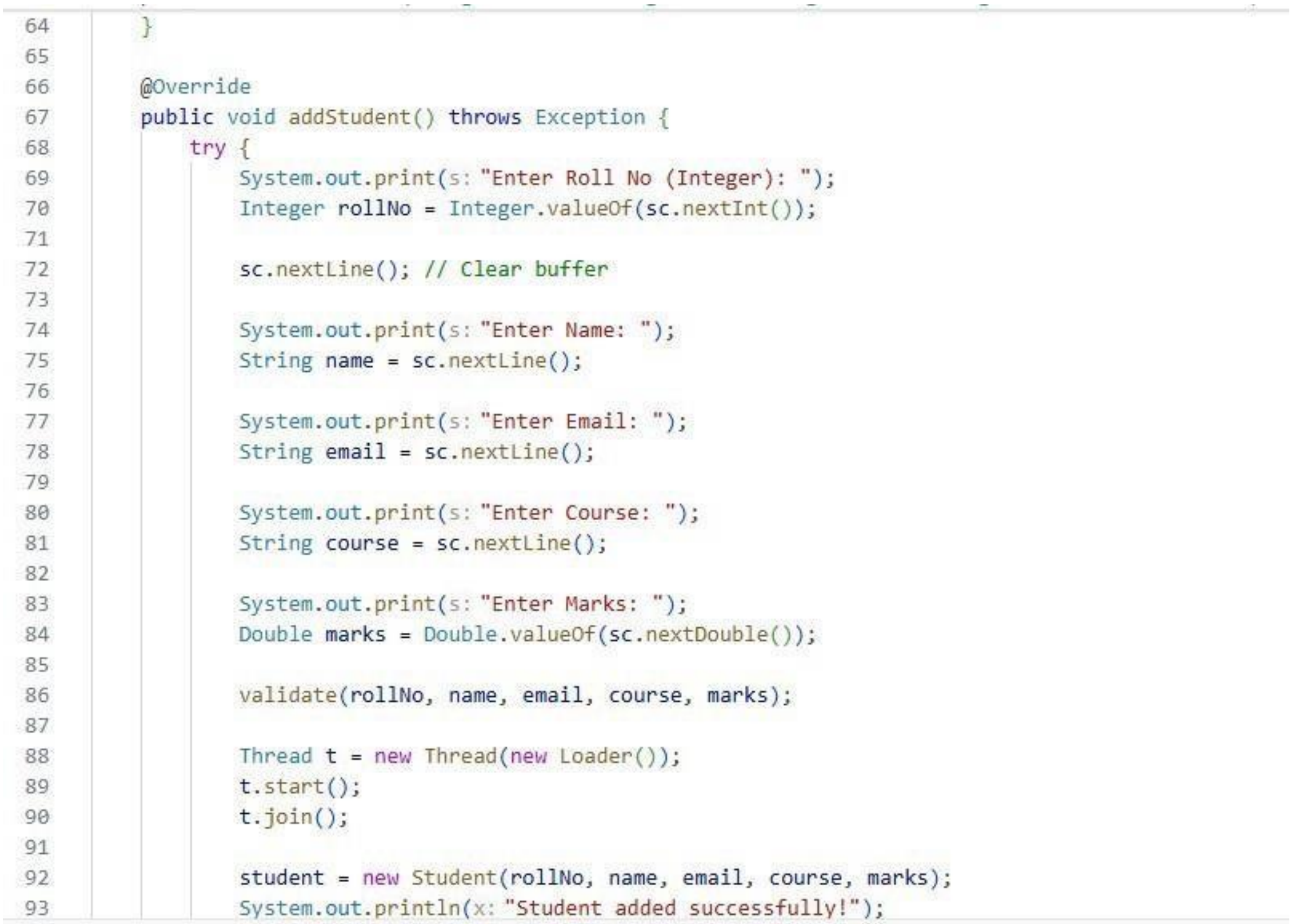


Main.java > ...

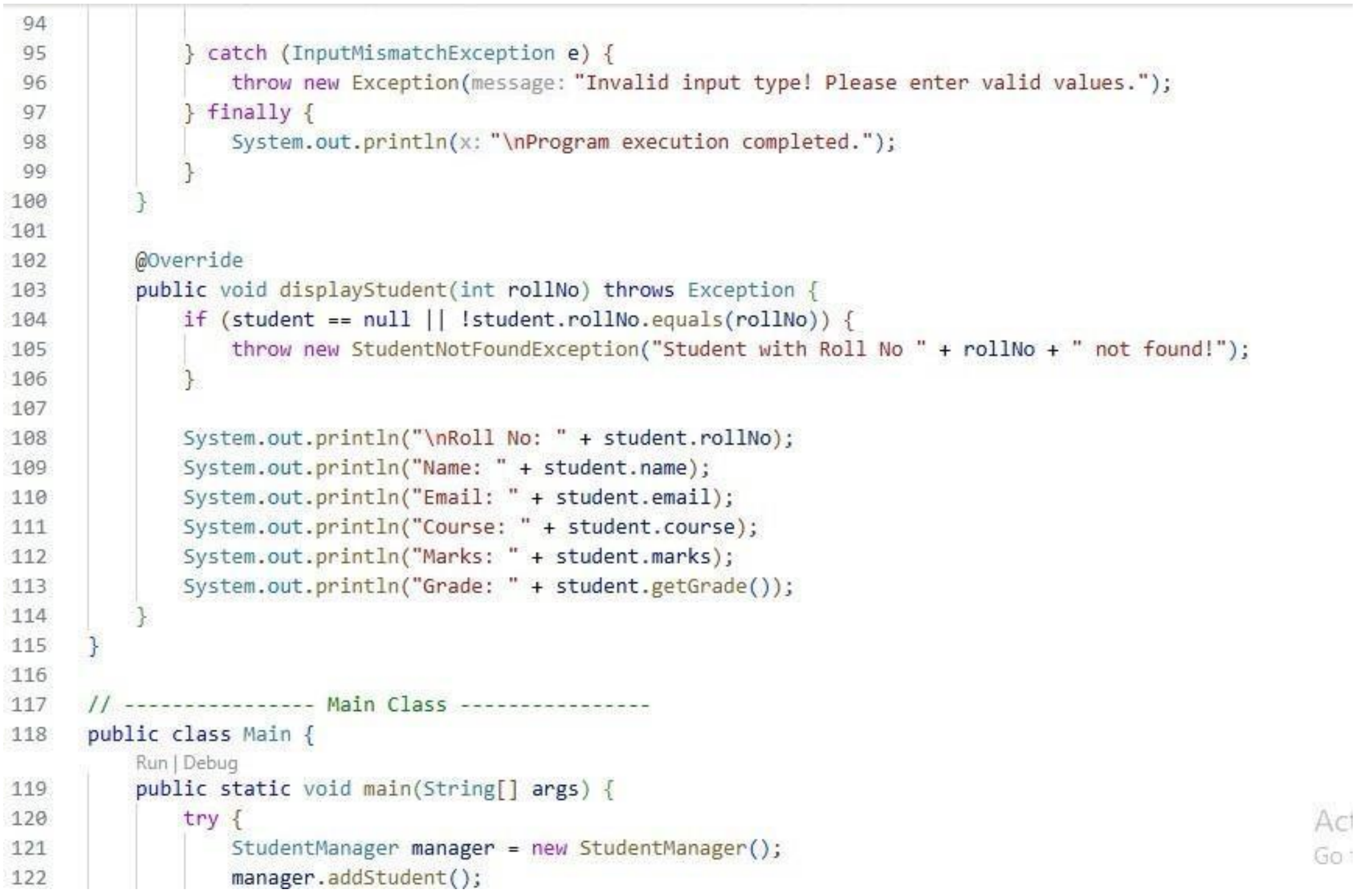
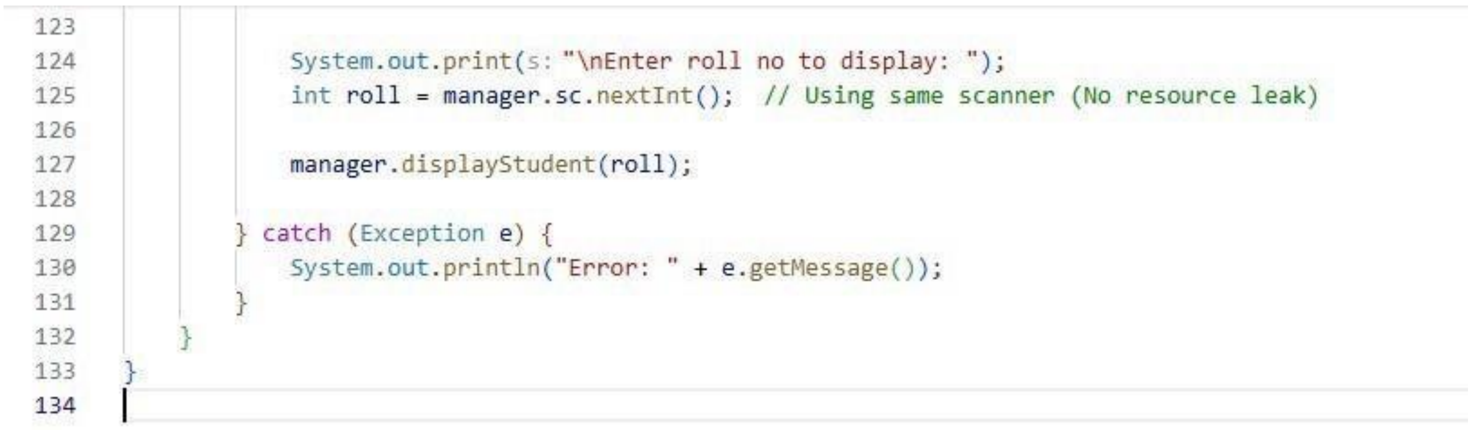
```java
import java.util.*;

// ---------------- Custom Exception ----------------
class StudentNotFoundException extends Exception {
    public StudentNotFoundException(String msg) {
        super(msg);
    }
}

// ---------------- Interface ----------------
interface RecordActions {
    void addStudent() throws Exception;
    void displayStudent(int rollNo) throws Exception;
}

// ---------------- Loader Thread ----------------
class Loader implements Runnable {
    @Override
    public void run() {
        try {
            System.out.println(x: "Loading.....");
            Thread.sleep(millis: 2000); // simulate delay
        } catch (Exception e) {
            System.out.println(x: "Loading interrupted!");
        }
    }
}

// ---------------- Student Class ----------------
class Student {
    Integer rollNo;
    String name, email, course;
```



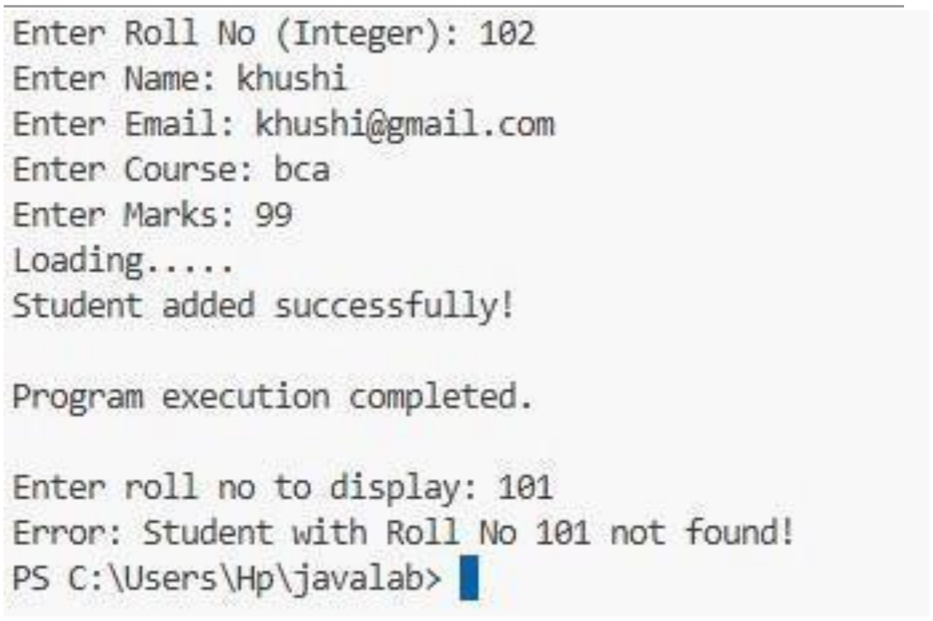
```java
    Double marks;

    public Student(Integer rollNo, String name, String email, String course, Double marks) {
        this.rollNo = rollNo;
        this.name = name;
        this.email = email;
        this.course = course;
        this.marks = marks;
    }

    public String getGrade() {
        if (marks >= 90) return "A";
        else if (marks >= 75) return "B";
        else if (marks >= 60) return "C";
        return "D";
    }
}

// ---------------- Student Manager ----------------
class StudentManager implements RecordActions {

    Scanner sc = new Scanner(System.in);
    Student student = null;

    // Validation Method
    private void validate(Integer roll, String name, String email, String course, Double marks) throws Exception {
        if (name.isEmpty() || email.isEmpty() || course.isEmpty())
            throw new Exception(message: "Fields cannot be empty!");

        if (marks < 0 || marks > 100)
            throw new Exception(message: "Marks must be between 0 and 100!");
```



```java
    }

    @Override
    public void addStudent() throws Exception {
        try {
            System.out.print(s: "Enter Roll No (Integer): ");
            Integer rollNo = Integer.valueOf(sc.nextInt());

            sc.nextLine(); // Clear buffer

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

            System.out.print(s: "Enter Email: ");
            String email = sc.nextLine();

            System.out.print(s: "Enter Course: ");
            String course = sc.nextLine();

            System.out.print(s: "Enter Marks: ");
            Double marks = Double.valueOf(sc.nextDouble());

            validate(rollNo, name, email, course, marks);

            Thread t = new Thread(new Loader());
            t.start();
            t.join();

            student = new Student(rollNo, name, email, course, marks);
            System.out.println(x: "Student added successfully!");
```


```java

        } catch (InputMismatchException e) {
            throw new Exception(message: "Invalid input type! Please enter valid values.");
        } finally {
            System.out.println(x: "\nProgram execution completed.");
        }
    }

    @Override
    public void displayStudent(int rollNo) throws Exception {
        if (student == null || !student.rollNo.equals(rollNo)) {
            throw new StudentNotFoundException("Student with Roll No " + rollNo + " not found!");
        }

        System.out.println("\nRoll No: " + student.rollNo);
        System.out.println("Name: " + student.name);
        System.out.println("Email: " + student.email);
        System.out.println("Course: " + student.course);
        System.out.println("Marks: " + student.marks);
        System.out.println("Grade: " + student.getGrade());
    }
}

// ---------------- Main Class ----------------
public class Main {
    Run | Debug
    public static void main(String[] args) {
        try {
            StudentManager manager = new StudentManager();
            manager.addStudent();
```

```java

            System.out.print(s: "\nEnter roll no to display: ");
            int roll = manager.sc.nextInt();  // Using same scanner (No resource leak)

            manager.displayStudent(roll);

        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}

```

Output:



```
Enter Roll No (Integer): 102
Enter Name: khushi
Enter Email: khushi@gmail.com
Enter Course: bca
Enter Marks: 99
Loading.....
Student added successfully!

Program execution completed.

Enter roll no to display: 101
Error: Student with Roll No 101 not found!
PS C:\Users\Hp\javalab>
```