SEMESTER PROJECT

SCHOOL MANAGEMENT SYSTEM

Feature:

Login

Logout

Student Manage

Teacher Manage

Attendence Manage

Exam/Grade Manage

Section/Class/Timetable Manage

Fee Manage

Library Manage

Transport Manage

Notification/Event

Online Fees Payment

Student Performance

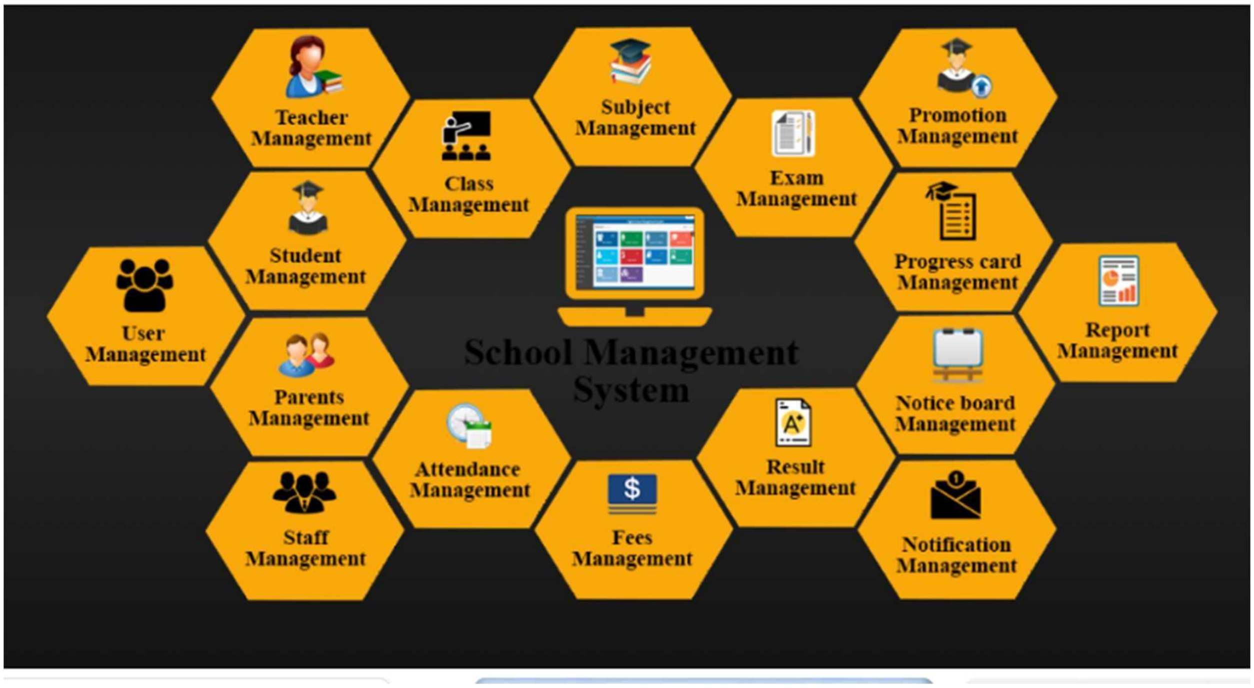
RFID Entry

Discipline Entry

School 3D Maps

Discussion Group

Hostel Management System



✓ WEEK 3 — ASSIGNMENT TASK 3

"The Coding Glow-Up: Data Structures Edition"

🎯 Objective (exact wording you can use)

In Week 3, the goal is to bring the School Management System to life by building the core features using data structures, testing them, improving them using the Kaizen approach, organizing tasks with the Eisenhower Matrix, completing GitHub activities, and engaging with peers through LinkedIn. A short in-class activity is also part of this week.

TASK A: Build Core DS Features – (Week 3)

“In this task, you have to create at least 3 major features for your School Management System using Data Structures.”

- It can not include Graphical User Interface.
- It can not include Database Connection.
- You only need to create DS-based logic (using ArrayList, HashMap, Queue, Stack, LinkedList, or a 2D Array).

➢ Feature 1: Student Management

This feature allows the system to store, update, and manage student information efficiently.
Using an **ArrayList** helps in storing all student objects in order, while a **HashMap** makes searching fast by mapping each student's ID to their record.

Include:

– Add Student
– Delete Student
– Search Student
(Use: ArrayList + HashMap)

➢ Feature 2: Attendance System

This feature keeps track of whether a student is present or absent on a given day.
A **HashMap** is used to store attendance records, making it easy to update or check attendance for any specific student.

Include:

– Mark Attendance
– Check Present/Absent
(Use: HashMap)

➢ Feature 3: Library / Book Queue

This feature manages book issuing in the order requests are received.
A **Queue** ensures that the first student who requests a book is the first one to receive it (FIFO – First In, First Out).

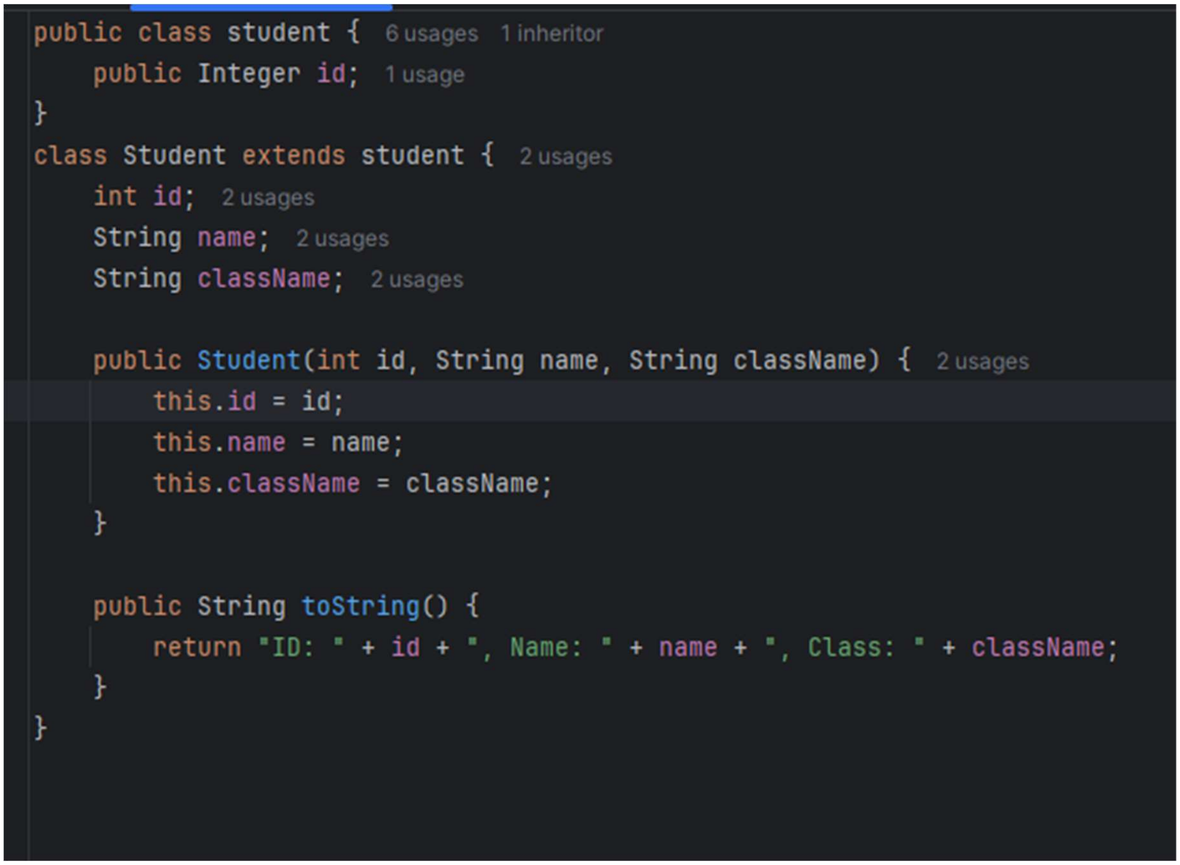
Include:

– Issue Book
– Return Book
– Display queue
(Use: Queue)

Folder Structure

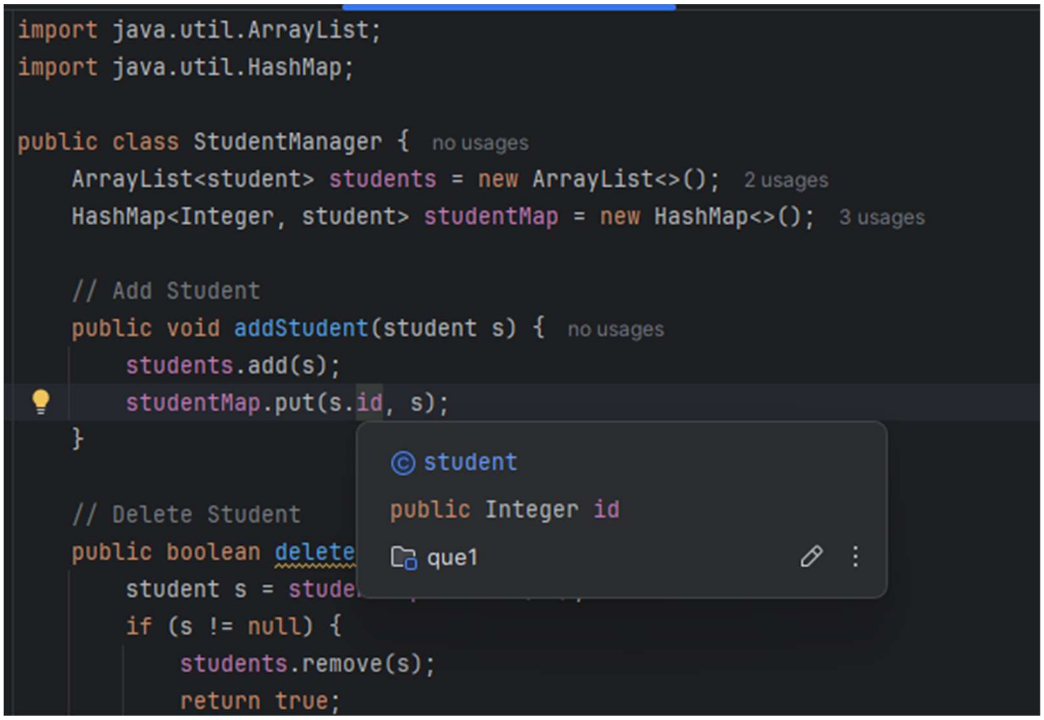
```
src/
   Student.java
   StudentManager.java
   AttendanceManager.java
   LibraryManager.java
   Main.java
```

Student.Java

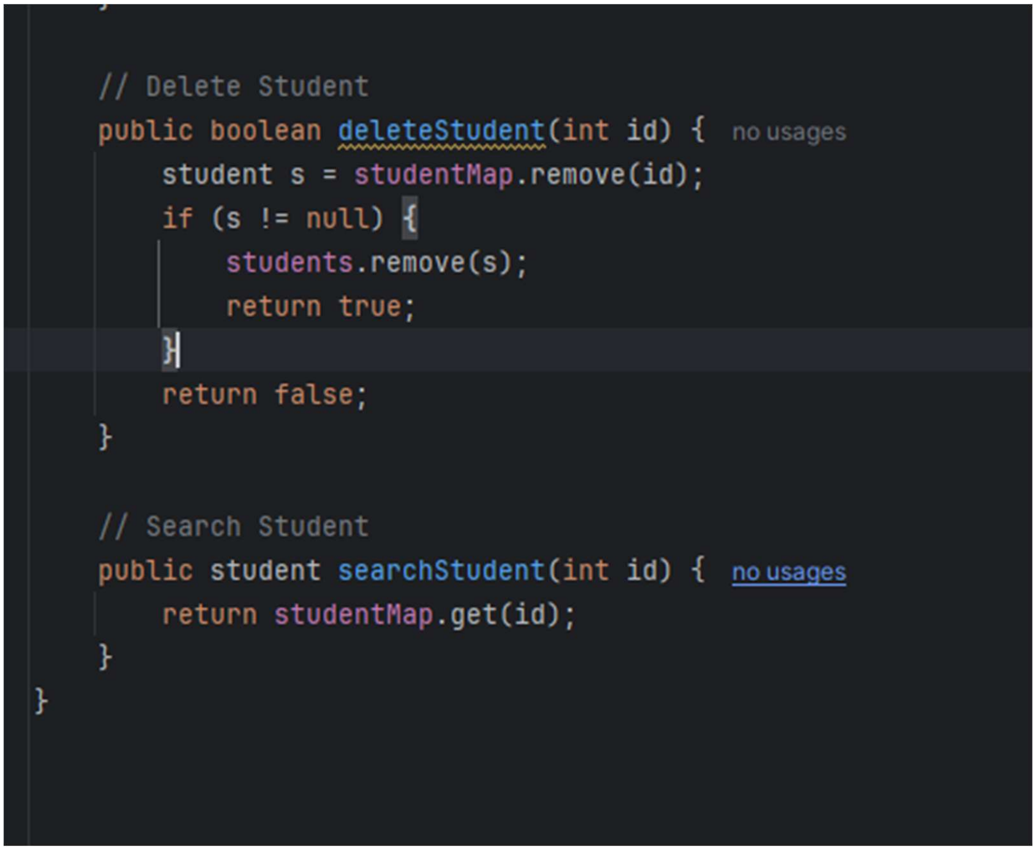


```java
public class student {  6 usages   1 inheritor
    public Integer id;  1 usage
}
class Student extends student {  2 usages
    int id;  2 usages
    String name;  2 usages
    String className;  2 usages

    public Student(int id, String name, String className) {  2 usages
        this.id = id;
        this.name = name;
        this.className = className;
    }

    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Class: " + className;
    }
}
```

StudentManager.java

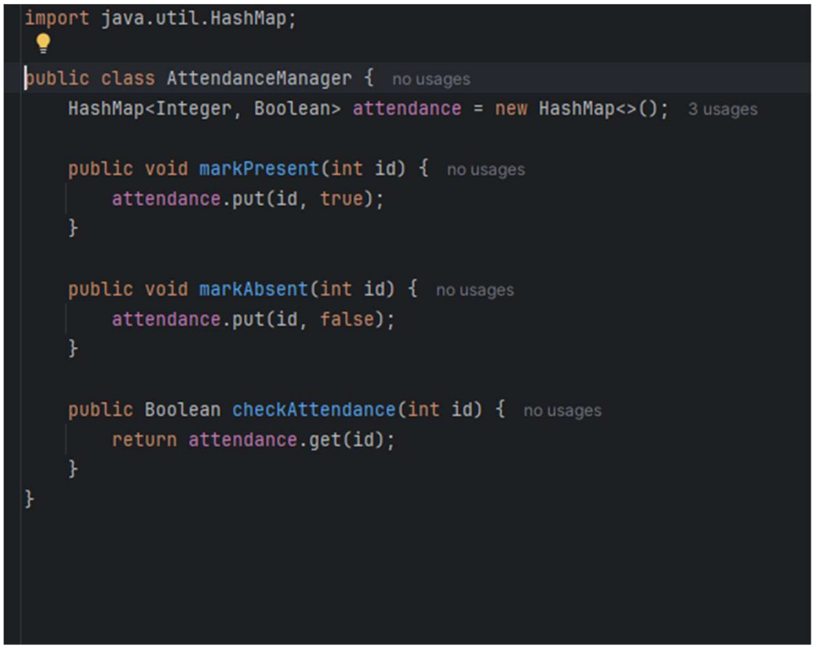


```
import java.util.ArrayList;
import java.util.HashMap;

public class StudentManager {  no usages
    ArrayList<student> students = new ArrayList<>();  2 usages
    HashMap<Integer, student> studentMap = new HashMap<>();  3 usages

    // Add Student
    public void addStudent(student s) {  no usages
        students.add(s);
        studentMap.put(s.id, s);
    }

    // Delete Student
    public boolean delete
        student s = stude
        if (s != null) {
            students.remove(s);
            return true;
```



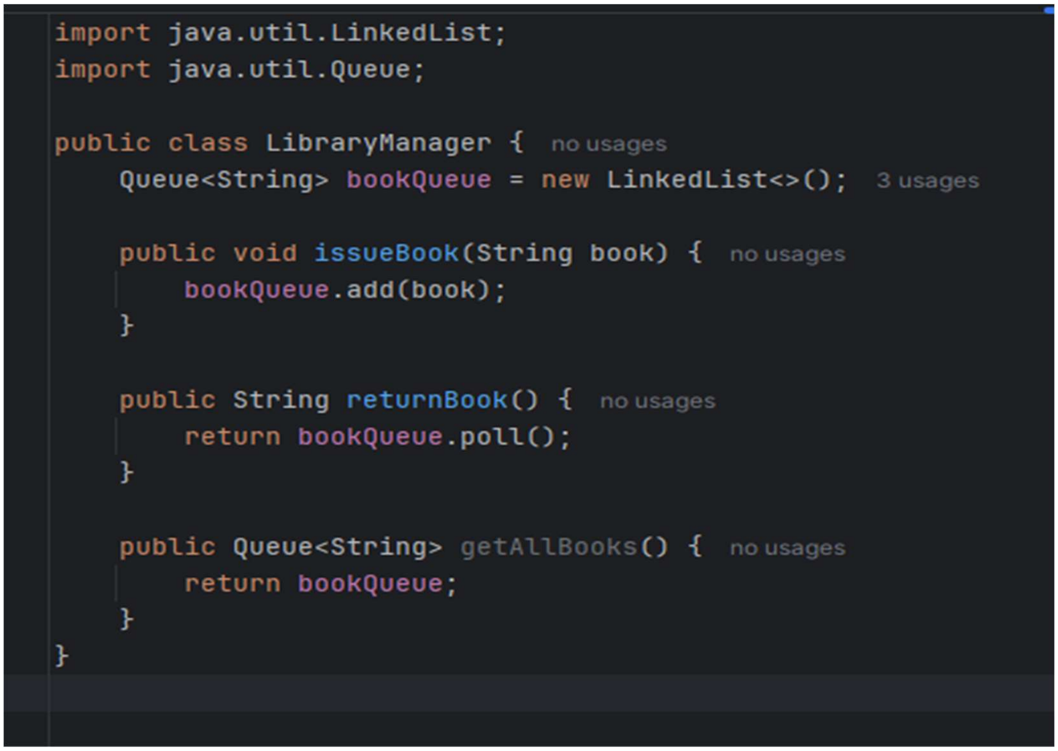
```
    // Delete Student
    public boolean deleteStudent(int id) {  no usages
        student s = studentMap.remove(id);
        if (s != null) {
            students.remove(s);
            return true;
        }
        return false;
    }

    // Search Student
    public student searchStudent(int id) {  no usages
        return studentMap.get(id);
    }
}
```

AttendanceManager.java



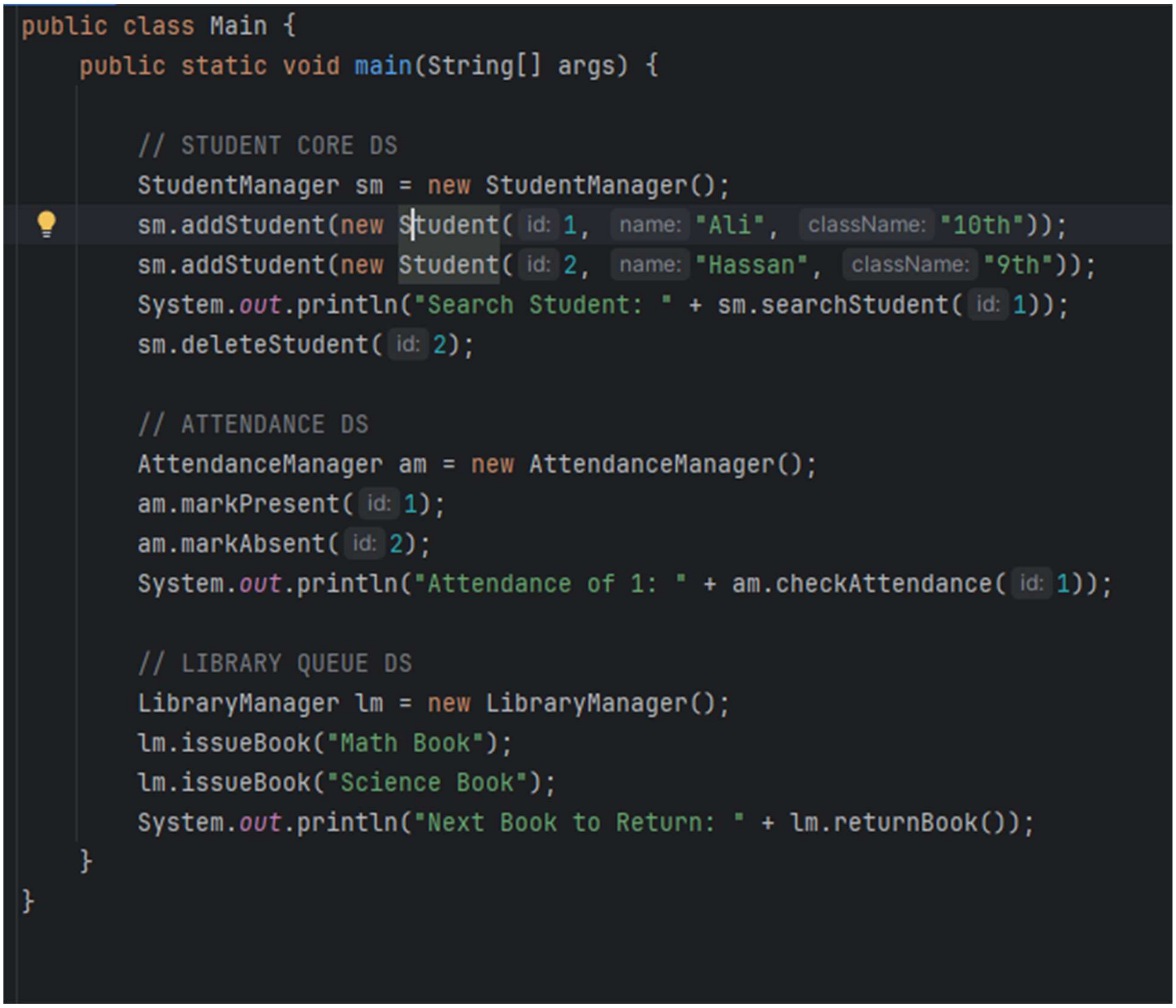
```
import java.util.HashMap;

public class AttendanceManager {  no usages
    HashMap<Integer, Boolean> attendance = new HashMap<>();  3 usages

    public void markPresent(int id) {  no usages
        attendance.put(id, true);
    }

    public void markAbsent(int id) {  no usages
        attendance.put(id, false);
    }

    public Boolean checkAttendance(int id) {  no usages
        return attendance.get(id);
    }
}
```

LibraryManager.java



```
import java.util.LinkedList;
import java.util.Queue;

public class LibraryManager {  no usages
    Queue<String> bookQueue = new LinkedList<>();  3 usages

    public void issueBook(String book) {  no usages
        bookQueue.add(book);
    }

    public String returnBook() {  no usages
        return bookQueue.poll();
    }

    public Queue<String> getAllBooks() {  no usages
        return bookQueue;
    }
}
```

Main.Java



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

        // STUDENT CORE DS
        StudentManager sm = new StudentManager();
        sm.addStudent(new Student( id: 1,  name: "Ali",  className: "10th"));
        sm.addStudent(new Student( id: 2,  name: "Hassan",  className: "9th"));
        System.out.println("Search Student: " + sm.searchStudent( id: 1));
        sm.deleteStudent( id: 2);

        // ATTENDANCE DS
        AttendanceManager am = new AttendanceManager();
        am.markPresent( id: 1);
        am.markAbsent( id: 2);
        System.out.println("Attendance of 1: " + am.checkAttendance( id: 1));

        // LIBRARY QUEUE DS
        LibraryManager lm = new LibraryManager();
        lm.issueBook("Math Book");
        lm.issueBook("Science Book");
        System.out.println("Next Book to Return: " + lm.returnBook());
    }
}
```

TASK B: Testing & Output Verification — What You Must Do

Also take on feature used on project in Task A:

1. **Student Management (Add/Delete/Search)**
2. **Attendance System**
3. **Library Queue System**

Also test on 3 feature include on project:

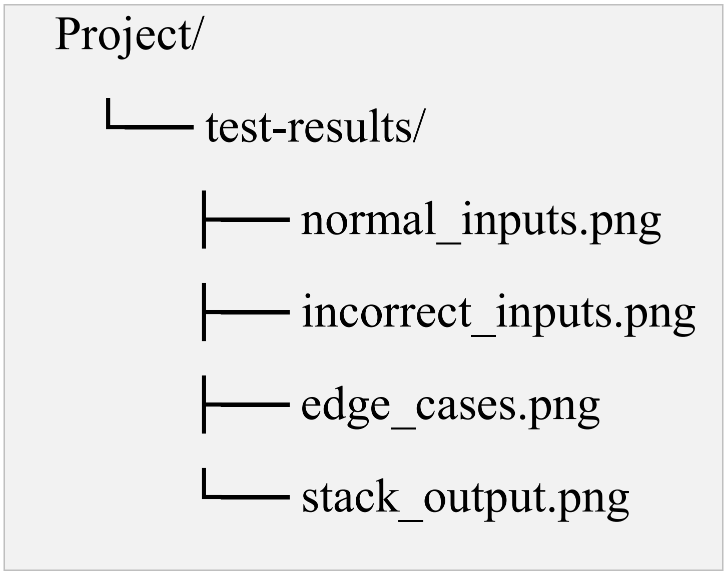
✓ **Normal Inputs:**
(Valid data that a user would normally enter)

✓ **Incorrect Inputs:**
(Invalid data — such as wrong IDs or empty strings)

✓ **Edge Cases:**
(Extreme or boundary conditions — such as ID = 0, an empty queue, or deleting a student who does not exist)

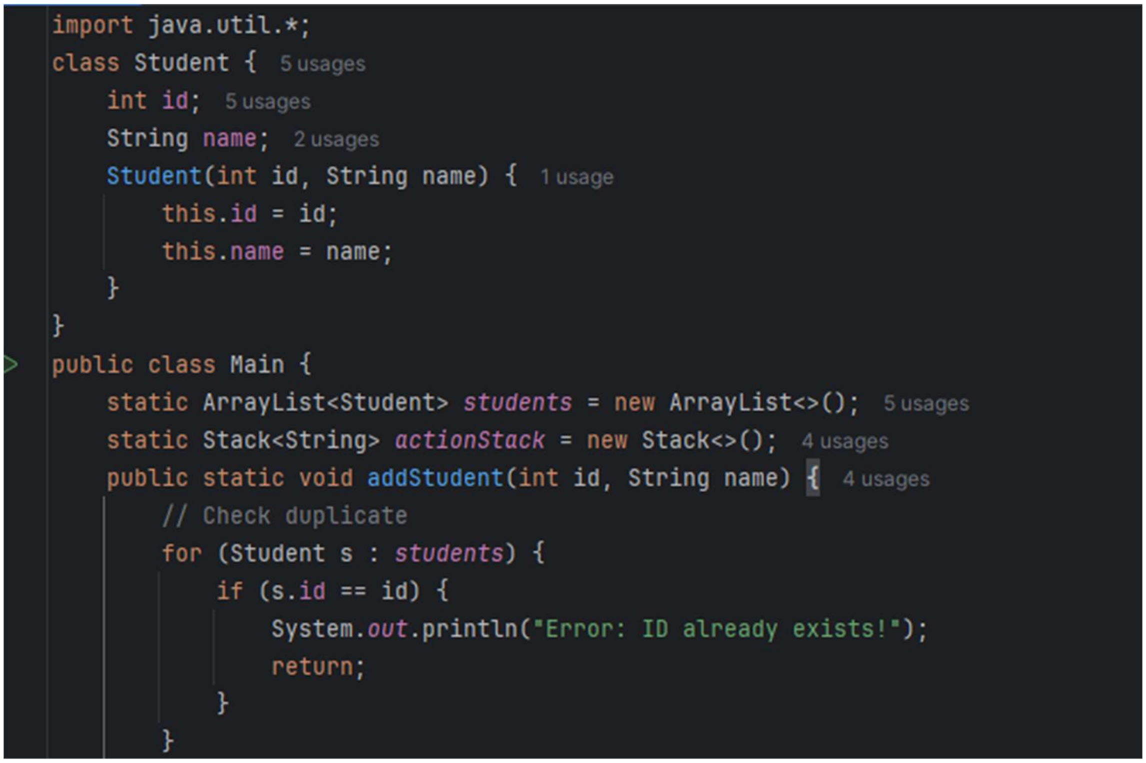
And you must place the screenshots and console logs of these tests in **one folder**.

Folder Structure.

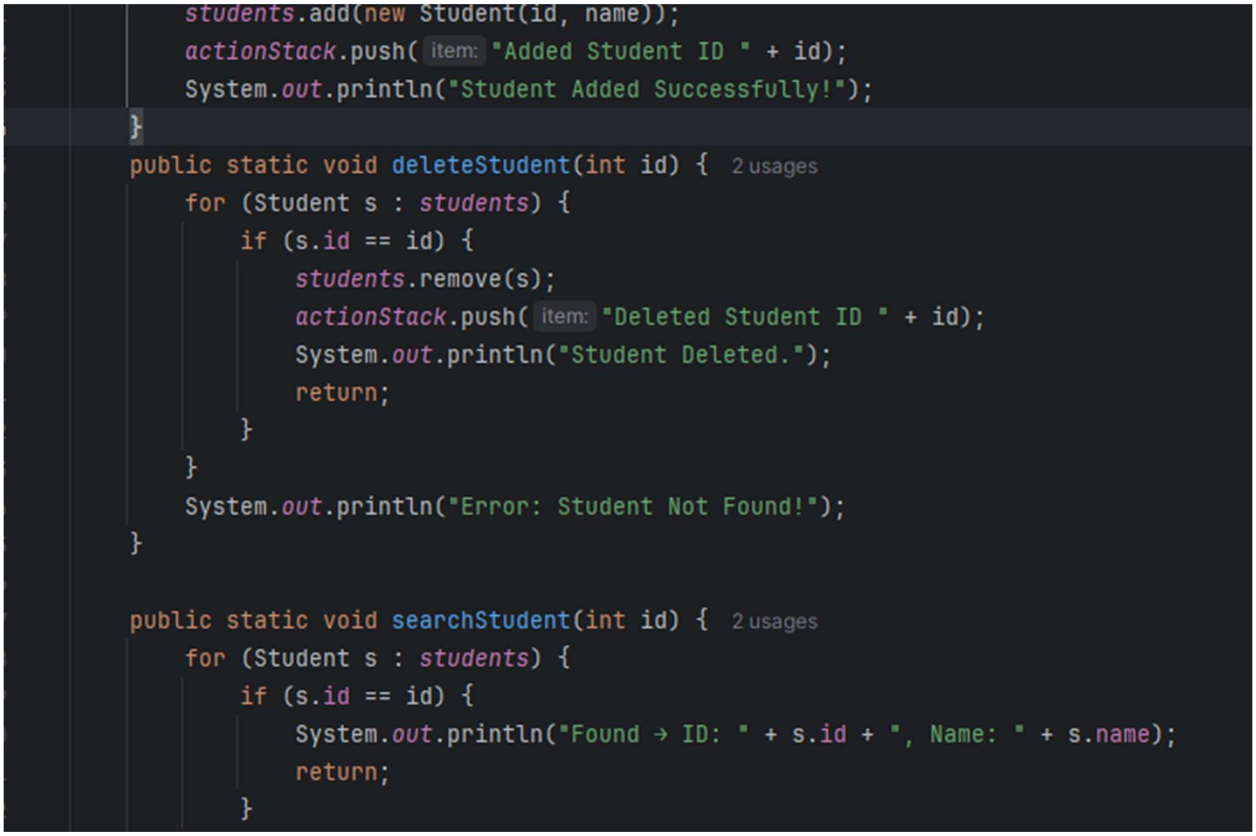


```
Project/
  └── test-results/
        ├── normal_inputs.png
        ├── incorrect_inputs.png
        ├── edge_cases.png
        └── stack_output.png
```

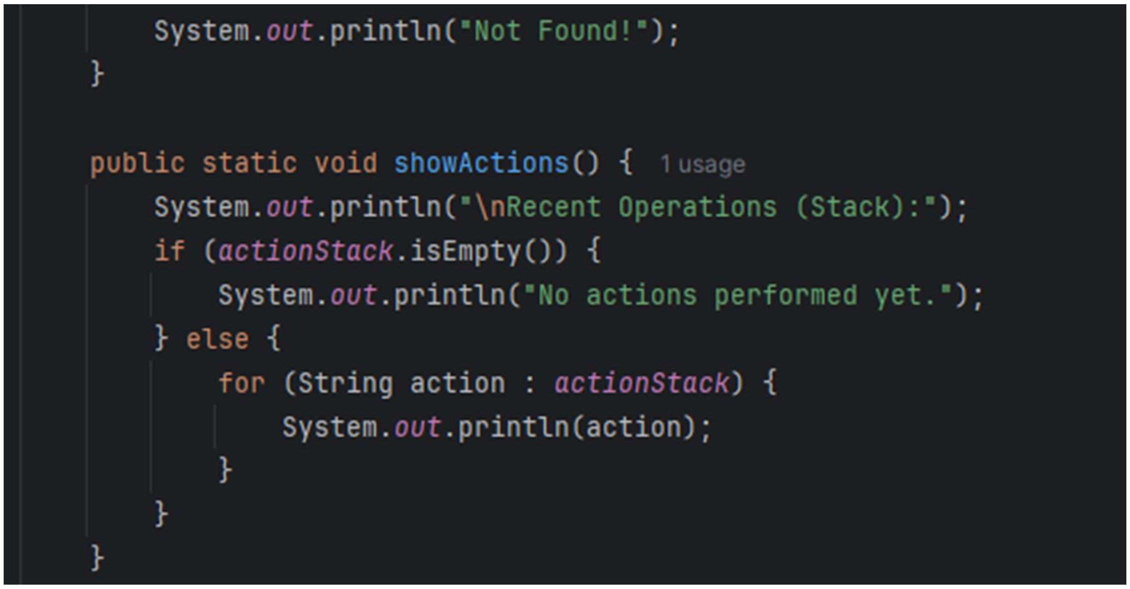
Main.java



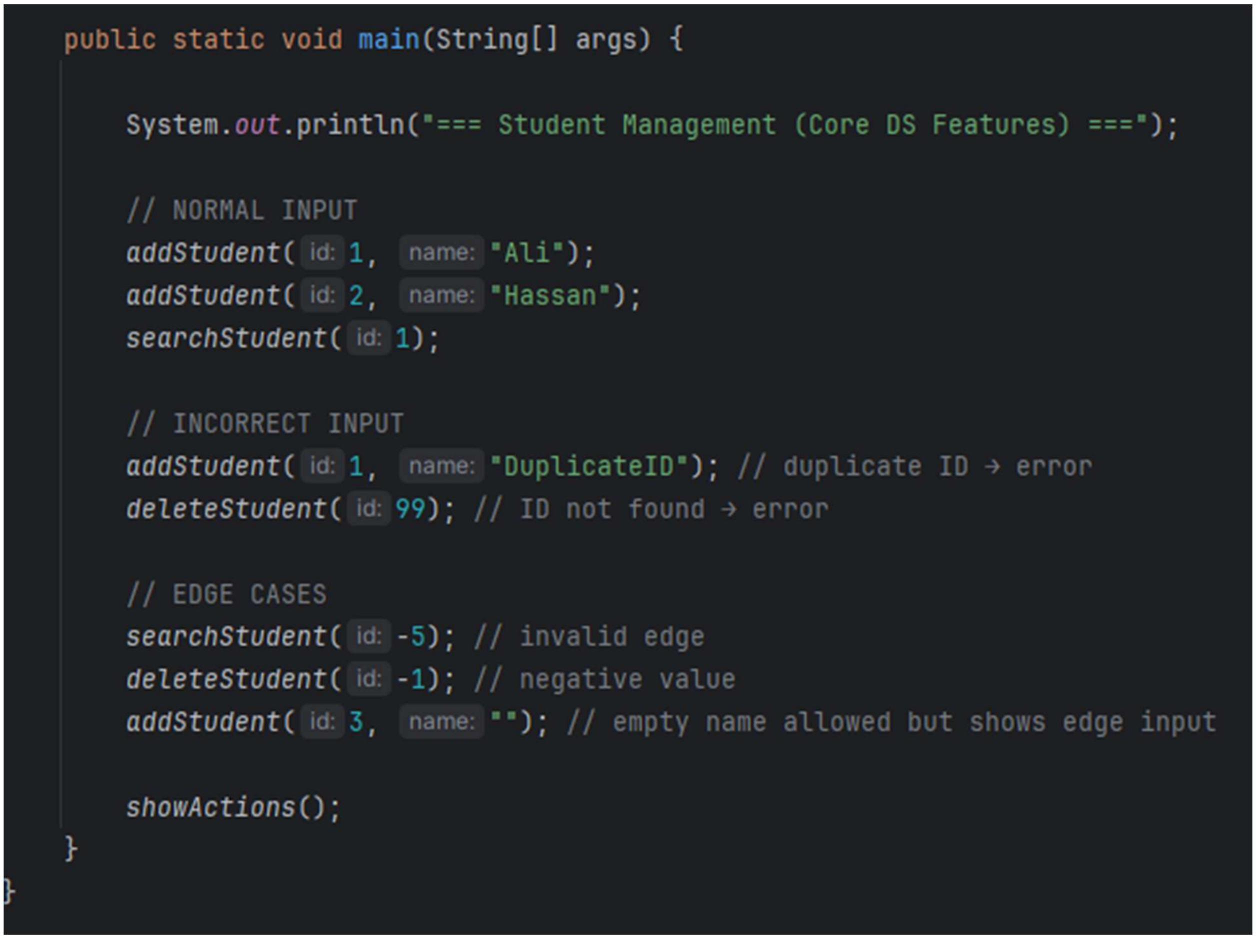
```java
import java.util.*;
class Student {  5 usages
    int id;  5 usages
    String name;  2 usages
    Student(int id, String name) {  1 usage
        this.id = id;
        this.name = name;
    }
}
public class Main {
    static ArrayList<Student> students = new ArrayList<>();  5 usages
    static Stack<String> actionStack = new Stack<>();  4 usages
    public static void addStudent(int id, String name) {  4 usages
        // Check duplicate
        for (Student s : students) {
            if (s.id == id) {
                System.out.println("Error: ID already exists!");
                return;
            }
        }
```



```java
        students.add(new Student(id, name));
        actionStack.push( item: "Added Student ID " + id);
        System.out.println("Student Added Successfully!");
    }
    public static void deleteStudent(int id) {  2 usages
        for (Student s : students) {
            if (s.id == id) {
                students.remove(s);
                actionStack.push( item: "Deleted Student ID " + id);
                System.out.println("Student Deleted.");
                return;
            }
        }
        System.out.println("Error: Student Not Found!");
    }

    public static void searchStudent(int id) {  2 usages
        for (Student s : students) {
            if (s.id == id) {
                System.out.println("Found → ID: " + s.id + ", Name: " + s.name);
                return;
            }
```

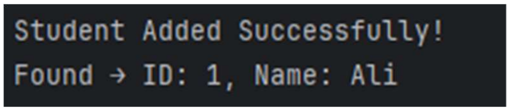


```
        System.out.println("Not Found!");
    }

    public static void showActions() {  1 usage
        System.out.println("\nRecent Operations (Stack):");
        if (actionStack.isEmpty()) {
            System.out.println("No actions performed yet.");
        } else {
            for (String action : actionStack) {
                System.out.println(action);
            }
        }
    }
```



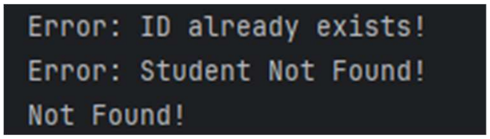
```
    public static void main(String[] args) {

        System.out.println("=== Student Management (Core DS Features) ===");

        // NORMAL INPUT
        addStudent( id: 1,  name: "Ali");
        addStudent( id: 2,  name: "Hassan");
        searchStudent( id: 1);

        // INCORRECT INPUT
        addStudent( id: 1,  name: "DuplicateID"); // duplicate ID → error
        deleteStudent( id: 99); // ID not found → error

        // EDGE CASES
        searchStudent( id: -5); // invalid edge
        deleteStudent( id: -1); // negative value
        addStudent( id: 3,  name: ""); // empty name allowed but shows edge input

        showActions();
    }
}
```

- **Normal Input:**



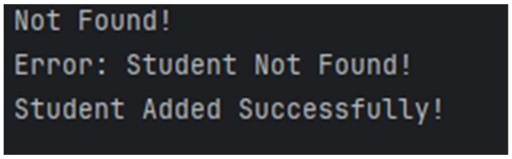
```
Student Added Successfully!
Found → ID: 1, Name: Ali
```

- **Incorrect Input:**



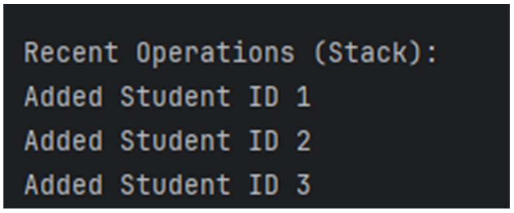
```
Error: ID already exists!
Error: Student Not Found!
Not Found!
```

- **Edge Cases:**



```
Not Found!
Error: Student Not Found!
Student Added Successfully!
```

- **Stack Output (Recent Actions):**



```
Recent Operations (Stack):
Added Student ID 1
Added Student ID 2
Added Student ID 3
```

Task C: Kaizen Improvement Review:

“Kaizen” is a Japanese word that means **continuous, small, step-by-step improvement** in any system, process, or team.

Kaizen Improvement Review

A **Kaizen Improvement Review** is a meeting or evaluation where you check:

Create Table Kaizen Improvement Review also include:

- What was built
- Problem / Limitation
- Improvement idea
- Team member responsible
- Expected impact

What Was Built	Problem / Limitation Found	Improvement Idea	Team Member Responsible	Expected Impact
Student Add/Delete/Search (Core DS Feature)	Duplicate IDs accepted, weak validation	Add strong ID and name validation	Ali	Prevents incorrect data and improves reliability
Action Stack (Recent Activities)	Shows only action text, no timestamp	Add timestamps using LocalDateTime	Hassan	Better tracking and auditing
Edge Case Handling	Negative ID accepted, empty names allowed	Block negative ID, add name check	Faizan	Cleaner and valid student records
Error Messages	Simple and unclear	Add descriptive and user-friendly messages	Subhan	Better user experience
Code Structure	All logic in Main class	Divide into StudentManager + Validation classes	Ali	Clean and maintainable code
Testing Logs	No saved logs for testing	Add file-based logging	Subhan	Easier debugging and documentation
Search Operation	Linear search slow for large data	Replace with HashMap	Hassan	Fast and efficient searching
Delete Operation	Error on invalid index	Pre-check before removal	Faizan	Prevents runtime exceptions
Console UI	Not user-friendly	Upgrade to GUI (Swing)	Ali	Better usability
No Database	Data not saved permanently	Add MySQL database	Subhan	Permanent storage & scalability

Task D: Eisenhower Matrix Planning:

“Eisenhower Matrix Planning is a time-management method that organizes tasks based on their urgency and importance.”

Create an Eisenhower Matrix with four quadrants:

- **Urgent + Important**
- **Important + Not urgent**
- **Urgent + Not important**
- **Not urgent + Not important**

Also Using Eisenhower Matrix on project in Four quadrants:

Quadrant	Task Example
Urgent + Important (Q1)	Student login issue fix, Add/Delete student validation
Important + Not urgent (Q2)	Add GUI dashboards, MySQL database integration, Improve attendance report
Urgent + Not important (Q3)	Update README, Code commenting, File structure organization
Not urgent + Not important (Q4)	Add fancy animations, Colorful UI, Extra optional features