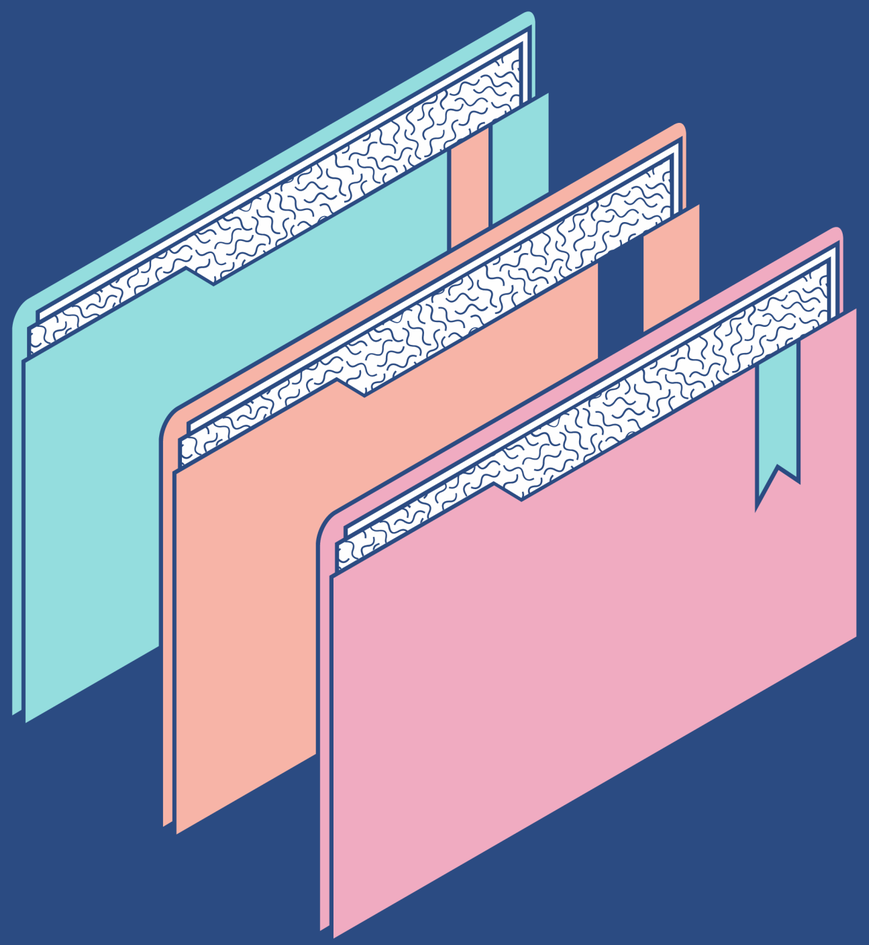
Soft Development
ABK



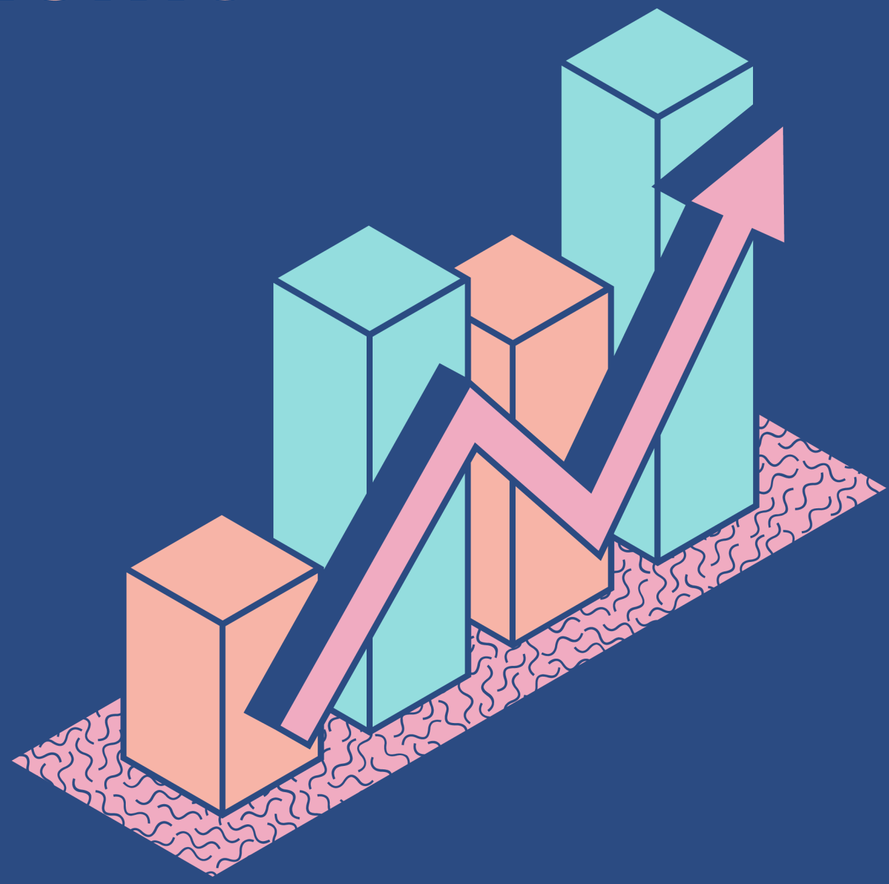
Objective

Develop a student management application that allows users to input student details and rank them based on their marks. The application should include functionalities for adding, editing, deleting, sorting, and searching for student records.

Input Requirements

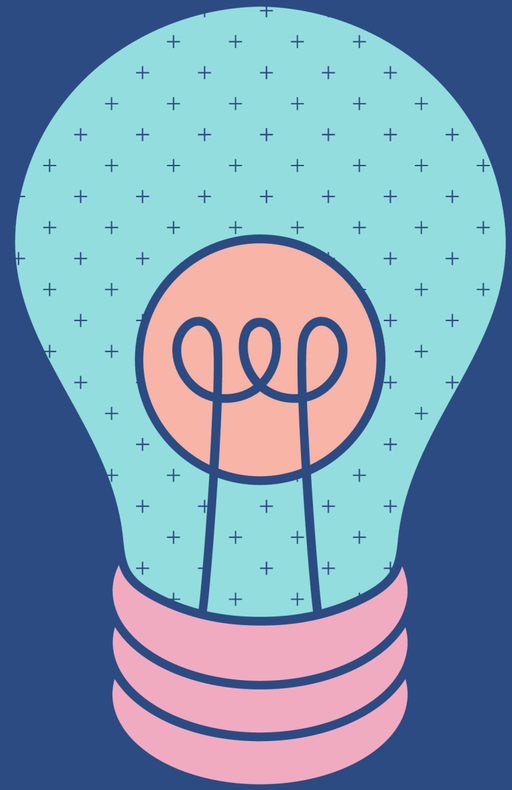
The application must allow the user to enter the following information for each student:

- ID of Student: A unique identifier for each student.
- Name of Student: The full name of the student.
- Marks of Student: The numerical marks obtained by the student.



Student Ranking Table

Marks Rank
[0 – 5.0) Fail
[5.0 – 6.5) Medium
[6.5 – 7.5) Good
[7.5 – 9.0) Very Good
[9.0 – 10.0] Excellent



Output Requirements

For each student, the program will output the following information

- Student ID
- Student's full name
- Marks of Student
- Student ranking
- Add, Edit, Delete, Sort, Search for students by apply algorithms

Functional Requirements

- Add Student: Allow users to input new student records.
- Edit Student: Enable users to modify existing student records.
- Delete Student: Allow users to remove student records.
- Sort Students: Provide functionality to sort students based on marks or names.
- Search Students: Allow users to search for students by ID or name.

Code analysis

Student Class

```java
package StudentManagement;

import java.util.Comparator;

public class Student  {  37 usages
    public String fullName;  8 usages
    public String id;  12 usages
    public double mark;  18 usages
    public String rank;  11 usages

    public Student(String id, String fullName, double mark){  5 usages
        this.id = id;
        this.fullName = fullName;
        this.mark = mark;
        if(this.mark >= 0 && this.mark <5){
            this.rank = "Fail";
        } else if (this.mark >=5 && this.mark < 6.5) {
            this.rank = "Medium";
        } else if (this.mark >= 6.5 && this.mark < 7.5) {
            this.rank = "Good";
        } else if (this.mark >= 7.5 && this.mark < 9) {
            this.rank = "Very Good";
        } else if(this.mark >= 9 && this.mark <= 10){
            this.rank = "Excellent";
        } else {
            this.rank = null;
```

Student Class

```java
public class Student  {  37 usages
    //getter and setter java for fullname
    public String getFullName() { return fullName; }
    public void setFullName(String fullName) { this.fullName = fullName; }
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public double getMark() { return mark; }
    public void setMark(double mark) { this.mark = mark; }


    public static Comparator<Student> IdStudentComparator = new Comparator<Student>() {  1 usage
        public int compare(Student o1, Student o2) {
            String idStu1 = o1.getId().toUpperCase();
            String idStu2 = o2.getId().toUpperCase();
            return  idStu1.compareTo(idStu2);
        }
    };
    public static Comparator<Student> FullNameStduComparator = new Comparator<Student>() {  1 usage
        public int compare(Student o1, Student o2) {
            String fullName1 = o1.getFullName().toUpperCase();
            String fullName2 = o2.getFullName().toUpperCase();
            return fullName1.compareTo(fullName2);
        }
    };

    public static Comparator<Student> MarkStduComparator = new Comparator<Student>() {  1 usage
```

Student Class

```
63          };
64
65          public static Comparator<Student> MarkStduComparator = new Comparator<Student>() {  1 usage
66              public int compare(Student o1, Student o2) {
67                  double mark1 = o1.getMark();
68                  double mark2 = o2.getMark();
69                  if(mark1 < mark2){
70                      return -1;
71                  } else if (mark2 < mark1) {
72                      return 1;
73                  }
74                  return 0;
75              }
76          };
77
78          @Override
79          public String toString() {
80              return "[ID = "+ id +" , fullName = " + fullName + ", mark = " + mark + " , rank  = " + rank + '
81          }
82      }
83
```

Student Class

Structure
Attributes:
- String fullName, String id, double mark, String rank

Constructor:
- Initializes attributes and assigns rank based on the mark.

Methods
Getters and Setters:
- Provide access to private attributes (should be private for better encapsulation).

Comparators:
- IdStudentComparator, FullNameStduComparator, MarkStduComparator for sorting students by ID, name, and mark.

toString():
- Returns a string representation of the student.

Logic
Rank Assignment:
- Uses conditional checks in the constructor—efficient and immediate.

Add Student

Main.java ArrayListAddStudent.java

```java
package StudentManagement;

import java.util.ArrayList;

public class ArrayListAddStudent {  2 usages
    public void addStudent(ArrayList<Student> students, Student objectData) { students.add(objectData); }
}
```

Add Student

Structure
Class Purpose:

- The ArrayListAddStudent class is designed to encapsulate the functionality for adding Student objects to an ArrayList.

Method
addStudent(ArrayList<Student> students, Student objectData):
Parameters:

- ArrayList<Student> students: The list to which the student will be added.
- Student objectData: The student object to be added.

Functionality:

- Uses students.add(objectData) to append the Student object to the end of the list.

Edit Student

```java
package StudentManagement;

import ...

public class ArrayListEditStudent {  2 usages
    public void editStudent(ArrayList<Student> students, int position, Student object){  1 usage
        students.set(position, object);
    }

    public void editStudentById(ArrayList<Student> students, String id, Student data){  1 usage
        for (int i = 0; i < students.size(); i++){
            if(Objects.equals(students.get(i).id, id)){
                students.set(i,data);
            }
        }
    }
}
```

Edit Student

Structure
Class Purpose:

- The ArrayListEditStudent class is designed to handle the editing of Student objects within an ArrayList.

Methods
editStudent(ArrayList<Student> students, int position, Student object):
Parameters:

- ArrayList<Student> students: The list of students.
- int position: The index at which to replace the existing student.
- Student object: The new student object to insert.

Functionality:

- Uses students.set(position, object) to replace the student at the specified index.

editStudentById(ArrayList<Student> students, String id, Student data):
Parameters:

- ArrayList<Student> students: The list of students.
- String id: The unique identifier of the student to edit.
- Student data: The new student object to replace the existing one.

Functionality:

- Iterates through the list to find a student with the matching ID and replaces it with the new student object.

Delete Student

```
package StudentManagement;

import ...

public class ArrayListRemoveStudent {  2 usages
    public void removeStudentById(ArrayList<Student> students, String id){  1 usage
        for (int i = 0; i < students.size(); i++) {
            if(Objects.equals(students.get(i).id, id)){
                students.remove(i);
            }
        }
    }
}
```

Delete Student

Structure
Purpose:

- **Handles the removal of Student objects from an ArrayList based on their unique ID.**

Method
removeStudentById(ArrayList<Student> students, String id):
Parameters:

- **ArrayList<Student> students: The list of students.**
- **String id: The ID of the student to remove.**

Functionality:

- **Iterates through the list and removes students with a matching ID.**

Sort Students

```java
public Student(String id, String fullName, double mark){  5 usages
    this.id = id;
    this.fullName = fullName;
    this.mark = mark;
    if(this.mark >= 0 && this.mark <5){
        this.rank = "Fail";
    } else if (this.mark >=5 && this.mark < 6.5) {
        this.rank = "Medium";
    } else if (this.mark >= 6.5 && this.mark < 7.5) {
        this.rank = "Good";
    } else if (this.mark >= 7.5 && this.mark < 9) {
        this.rank = "Very Good";
    } else if(this.mark >= 9 && this.mark <= 10){
        this.rank = "Excellent";
    } else {
        this.rank = null;
    }
}
```

Sort Students

Structure

Constructor Purpose:

- Initializes a Student object with an ID, full name, and mark, and assigns a rank based on the mark.

Parameters

- String id: Unique identifier for the student.
- String fullName: Full name of the student.
- double mark: Score obtained by the student.

Functionality

Rank Assignment:

The constructor uses a series of conditional statements to assign a rank based on the value of mark:

- Fail: Mark < 5
- Medium: 5 ≤ Mark < 6.5
- Good: 6.5 ≤ Mark < 7.5
- Very Good: 7.5 ≤ Mark < 9
- Excellent: 9 ≤ Mark ≤ 10
- Else: Sets rank to null for invalid marks (e.g., negative values or above 10).

Search Students

```java
package StudentManagement;

import ...

public class ArrayListSearchStudent {  2 usages
    public int binarySearch(ArrayList<Student> students, String id){  1 usage
        int left = 0;
        int right = students.size() - 1;
        while (left <= right){
            int mid = left + (right - left) / 2;
            if (Objects.equals(students.get(mid).id, id)) {
                return mid;
            }
            int compareStr = students.get(mid).id.compareToIgnoreCase(id);
            if(compareStr < 0){
                left = mid + 1;
            } else if (compareStr > 0){
                right = mid - 1;
            }
        }
        return  -1;
    }
}
```

Search Students

Structure
Purpose:

- Implements a binary search algorithm to find a Student object in an ArrayList based on its unique ID.

Method
binarySearch(ArrayList<Student> students, String id):
Parameters:

- ArrayList<Student> students: The list of students (assumed to be sorted by ID).
- String id: The ID of the student to search for.

Functionality:
Implements a binary search algorithm:

- Initializes left and right pointers.

Repeatedly calculates the middle index and compares the ID at that index with the target ID.

- Adjusts pointers based on the comparison until the ID is found or the search space is exhausted.

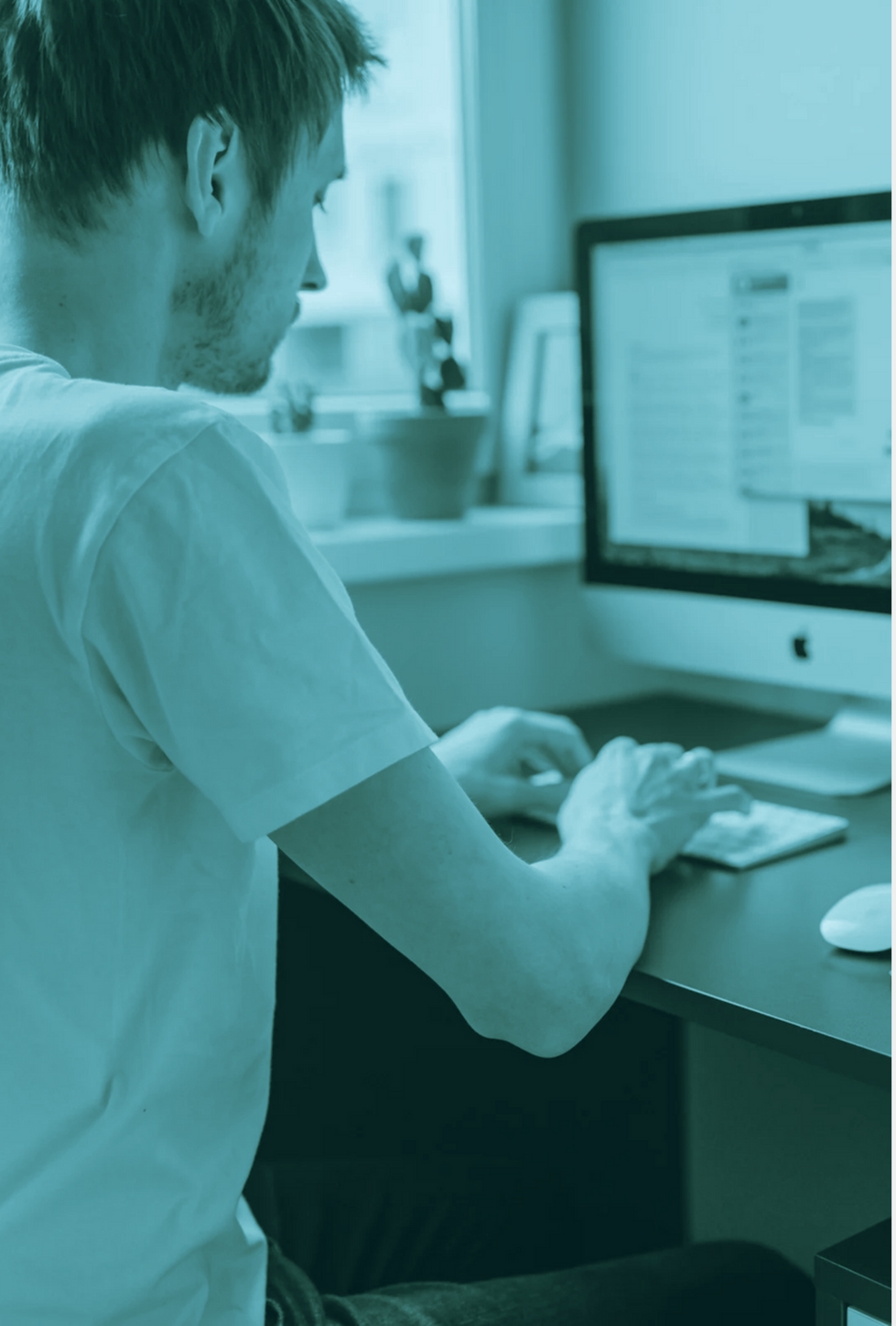
Search Students

Binary Search Algorithm
Concept: Binary search is an efficient algorithm for searching a sorted list. This algorithm divides the list in half and compares the middle value with the target value, thereby reducing the search range.
How it Works:

- Initialization: Set up two indices left and right to determine the search range.
- Iteration: Use a while loop to continue searching when left does not exceed right.
- Calculate Mid Index: Calculate the mid index and compare the student ID at this index with the target ID.

Adjust Range:

- If the ID at the mid index is equal to the target ID, return that index.
- If the ID at the mid index is less than the target ID, adjust left to search in the right half.
- If the ID at the mid index is greater than the target ID, adjust right to search in the left half.



Thanks for Watching