Student Management System

Project Overview

Objective:

- Manage student records including ID, name, marks, and rank.
- Implement key functionalities: Add, Edit, Delete, Search, Display, and Sort.

Key Features:

- Console-based menu for user interaction.
- Stack-based data structure for storing student records.
- Sorting algorithms for efficient data organization

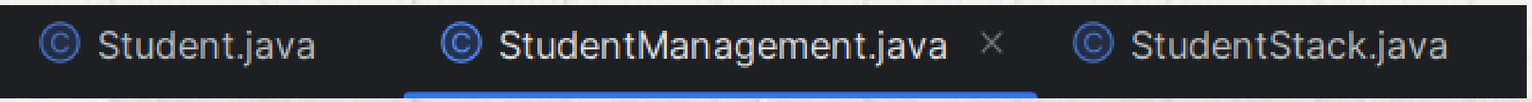
System Architecture

Components:

1. **Student Class: Encapsulates student attributes and methods.**
2. **StudentStack Class: Implements stack data structure for storage.**
3. **StudentManagement Class: Handles core logic and user interface.**

Relationships:

- **StudentManagement interacts with StudentStack and Student classes to process data.**



Class: Student

Purpose:

Represents a student with attributes:

- **id: Unique identifier.**
- **name: Student name.**
- **marks: Scores (0–10).**
- **rank: Derived from marks.**

Key Methods:

calculateRank(): Classifies performance (Fail, Medium, Good, Very Good, Excellent).

toString(): Displays student details in formatted output.

```
    private String calculateRank() {  2 usages
        if (marks >= 0 && marks < 5.0) {
            return "Fail";
        } else if (marks >= 5.0 && marks < 6.5) {
            return "Medium";
        } else if (marks >= 6.5 && marks < 7.5) {
            return "Good";
        } else if (marks >= 7.5 && marks < 9.0) {
            return "Very Good";
        } else if (marks >= 9.0 && marks <= 10.0) {
            return "Excellent";
        } else {
            return "Invalid Marks";
        }
    }

    @Override
    public String toString() { return String.format("%-10s %-20s %-10.2f %-15s", id, name, marks, rank); }
```

Class: StudentStack

Purpose:

- Implements stack (LIFO) to store Student objects.

Key Operations:

- push(Student student): Add a student.
- pop(): Remove and return the top student.
- peek(): View the top student without removal.
- isEmpty(): Check if the stack is empty.

Traversal:

- displayStudents(): Print all student records
- from top to bottom.

```java
public StudentStack() {  9 usages
    this.top = null;
    this.size = 0;
}

public void push(T data) {  19 usages
    Node<T> newNode = new Node<>(data);
    newNode.next = top;
    top = newNode;
    size++;
}

public T pop() {  17 usages
    if (isEmpty()) {
        throw new IllegalStateException("Stack is empty.");
    }
    T data = top.data;
    top = top.next;
    size--;
    return data;
}

public T peek() {  1 usage
    if (isEmpty()) {
        throw new IllegalStateException("Stack is empty.");
    }
    return top.data;
}

public boolean isEmpty() { return top == null; }
```

Class: StudentManagement

+ Core Functionalities:

Add: Insert a student with unique ID and valid marks.

Edit: Update student details.

Delete: Remove a student by ID.

Search: Locate a student by ID.

+ Sort: Organize students by marks using:

Bubble Sort.

Quick Sort.

Merge Sort.

Display: Show all student records in stack order.

```
public void sortStudentsBubbleSort() {  1 usage
    if (studentStack.isEmpty()) {
        System.out.println("No students to sort.");
        return;
    }
ublic void sortStudentsQuickSort() {  1 usage
   StudentStack<Student> tempStack = new StudentStack<>();
   while (!studentStack.isEmpty()) {
       tempStack.push(studentStack.pop());
   }
 public void sortStudentsMergeSort() {  1 usage
    StudentStack<Student> tempStack = new StudentStack<>();
    while (!studentStack.isEmpty()) {
        tempStack.push(studentStack.pop());
    }
public void editStudent(String id, String newName, double newMarks) {  1 usage
   StudentStack<Student> tempStack = new StudentStack<>();
 public void deleteStudent(String id) {  1 usage
    StudentStack<Student> tempStack = new StudentStack<>();
    boolean found = false;
public void searchStudent(String id) {  1 usage
    StudentStack<Student> tempStack = new StudentStack<>();
```

Sorting Algorithms

+Implemented Algorithms:
1 Bubble Sort:
Time Complexity: $O(n^2)$.
In-place sorting; simple but inefficient for large datasets.

2 Quick Sort:
Time Complexity: O(n log n) (average case).
Divide-and-conquer approach; efficient but unstable.

3 Merge Sort:
Time Complexity: O(n log n).
Stable and efficient for linked structures.

Purpose:
Sort students by marks in ascending order.

Console Menu

Features:

Intuitive menu with numbered options for actions.

Input validation for IDs, names, and marks.

Handles invalid inputs gracefully.

```
Student Management System
1. Add Student
2. Edit Student
3. Delete Student
4. Sort Students
5. Search Student
6. Display Students
7. Exit
Enter your choice:
```

Example Scenarios

Adding Students:

Input: ID: 1 , Name: Nam, Marks: 9.0.

Output: Student added successfully.

Sorting Students:

Use Merge Sort.

Output (Ascending Marks):

```
Enter your choice: 1
Enter Student ID: 1
Enter Student Name: Nam
Enter Student Marks (0-10): 9
Student added successfully.
```

before sort

```
Enter your choice: 6
ID          Name                    Marks       Rank
3           hung                    10.00       Excellent
2           huy                     5.00        Medium
1           Nam                     9.00        Excellent
```

after sort

```
Students sorted using Merge Sort.
Enter your choice: 6
ID          Name                    Marks       Rank
3           hung                    10.00       Excellent
1           Nam                     9.00        Excellent
2           huy                     5.00        Medium
```

Algorithm Analysis

+ Benchmark Results:

Dataset: 100,000 random entries.

Sorting Execution Times:

Bubble Sort:10123901900

Quick Sort: 11227200

Merge Sort: 14952200

Run time Insertion: 1748536900

+ Takeaway:

Quick Sort is faster for most cases.

Merge Sort is more stable for linked lists.

```
Run time Bubble: 10123901900
Run time Quick: 11227200
Run time Merge: 14952200
Run time Insertion: 1748536900


Process finished with exit code 0
```