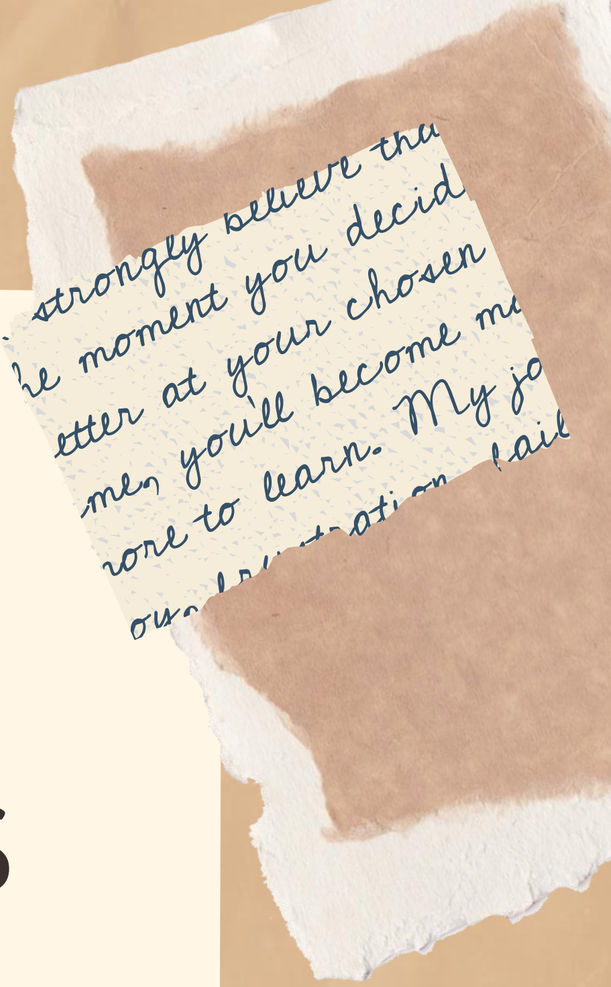
ADVANCED IMPLEMENTATION OF ADTS AND ALGORITHMS



INTRODUCTION

- Briefly describe the purpose of the program:

"This application manages student data using a stack-based data structure."
Outline features such as adding, editing, deleting, searching, sorting, and displaying students

Mention the stack-based approach and why it was chosen.

SYSTEM DESIGN

Class Student : represents a student with attributes **id, name, marks,** and **rank**

StackADT Class : Implements stack operations (**push, pop, peek, size, isEmty.).**

StudentManagement Class : Manages student operations (CRUD and sorting).

Utilizes a StackADT to handle storage.

Main Class : Provides a user interface for interaction.

Uses a menu-driven system for functionality.

CLASS STUDENT

Represents a student with attributes
name, id , marks, rank

Getter Methods
String getId(): Returns the student's ID.
String getName(): Returns the student's name.
d**ouble getMarks():** Returns the student's marks.
String getRank(): Returns the student's rank (calculated based on marks).

```java
public class Student {  38 usages
    private String id;  3 usages
    private String name;  4 usages
    private double marks;  14 usages
    private String rank;  4 usages

    public Student(String id, String name, double marks) {  2 usages
        this.id = id;
        this.name = name;
        this.marks = marks;
        this.rank = calculateRank();
    }
```

```java
    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);
    }
}
```

StackADT Class

push(T data): Adds a new element to the top of the stack

pop(): Removes and returns the element at the top of the stack.

peek() :Returns the element at the top of the stack without removing it.

isEmpty() : Checks if the stack contains any elements. Returns true if top == null, otherwise returns false.

size() : Returns the number of elements currently in the stack.

```
public class StackADT<T> {  23 usages
    private static class Node<T> {  4 usages
        T data;  3 usages
        Node<T> next;  3 usages

        Node(T data) {  1 usage
            this.data = data;
            this.next = null;
        }
    }

    private Node<T> top;  8 usages
    private int size;  4 usages

    public StackADT() {  12 usages
        this.top = null;
        this.size = 0;
    }

    public void push(T data) {  22 usages
        Node<T> newNode = new Node<>(data);
        newNode.next = top;
        top = newNode;
        size++;
    }

    public T pop() {  18 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() {  25 usages
        return top == null;
    }

    public int size() {  no usages
        return size;
    }
}
```

StudentManagement Class

addStudent(String id, String name, double marks)

editStudent(String id, String newName, double newMarks)

deleteStudent(String id)

displayStudents()

searchStudent(String id)

```java
public void addStudent(String id, String name, double marks) {  1 usage
    if (marks < 0 || marks > 10) {
        System.out.println("Invalid Marks: Please enter marks between 0 and 10.");
        return;
    }
    StackADT<Student> tempStack = new StackADT<>();
    while (!studentStack.isEmpty()) {
        Student student = studentStack.pop();
        if (student.getId().equals(id)) {
            System.out.println("Duplicate ID: A student with this ID already exists.");
            tempStack.push(student);
            while (!tempStack.isEmpty()) {
                studentStack.push(tempStack.pop());
            }
            return;
        }
        tempStack.push(student);
    }
    while (!tempStack.isEmpty()) {
        studentStack.push(tempStack.pop());
    }
    Student newStudent = new Student(id, name, marks);
    studentStack.push(newStudent);
    System.out.println("Student added successfully.");
}
```

```java
public void editStudent(String id, String newName, double newMarks) {  1 usage
    StackADT<Student> tempStack = new StackADT<>();
    boolean found = false;
    while (!studentStack.isEmpty()) {
        Student student = studentStack.pop();
        if (student.getId().equals(id)) {
            student.setName(newName);
            student.setMarks(newMarks);
            found = true;
        }
        tempStack.push(student);
    }
    while (!tempStack.isEmpty()) {
        studentStack.push(tempStack.pop());
    }
    if (found) {
        System.out.println("Student updated successfully.");
    } else {
        System.out.println("Student with ID " + id + " not found.");
    }
}
```

Sorting algorithm

sortStudentsBubbleSort()

sortStudentsQuickSort()

sortStudentsMergeSort()

```java
public long sortStudentsBubbleSort() {  3 usages
    if (studentStack.isEmpty()) {
        System.out.println("No students to sort.");
        return 0;
    }
    long startTime = System.nanoTime();

    StackADT<Student> sortedStack = new StackADT<>();
    while (!studentStack.isEmpty()) {
        Student temp = studentStack.pop();
        while (!sortedStack.isEmpty() && sortedStack.peek().getMarks() > temp.getMarks()) {
            studentStack.push(sortedStack.pop());
        }
        sortedStack.push(temp);
    }
    studentStack = sortedStack;

    long endTime = System.nanoTime();
    System.out.println("Students sorted using Bubble Sort.")
    return endTime - startTime;
}
```

```java
public long sortStudentsQuickSort() {  3 usages
    if (studentStack.isEmpty()) {
        System.out.println("No students to sort.");
        return 0;
    }
    long startTime = System.nanoTime();

    StackADT<Student> tempStack = new StackADT<>();
    while (!studentStack.isEmpty()) {
        tempStack.push(studentStack.pop());
    }
    Student[] studentArray = new Student[tempStack.size()];
    int index = 0;
    while (!tempStack.isEmpty()) {
        studentArray[index++] = tempStack.pop();
    }
    quickSort(studentArray, low: 0, high: studentArray.length - 1);
    for (Student student : studentArray) {
        studentStack.push(student);
    }

    long endTime = System.nanoTime();
    System.out.println("Students sorted using Quick Sort.");
    return endTime - startTime;
}
```

Compare Quick Sort and Bubble Sort

Bubble Sort : Time Complexity: $O(n^2)$.

Quick Sort: Time Complexity: O(n log n)

```
Enter your choice: 5
Enter the number of random students to generate: 10000
10000 students have been generated.
```

```
4. Compare Bubble and Quick Sort
Choose sorting method: 4
Students sorted using Bubble Sort.
Students sorted using Quick Sort.
Bubble Sort execution time: 522232800 ms
Quick Sort execution time: 5027700 ms
```

Add and sort student

```
=== Student Management System ===
1. Add Student
2. Edit Student
3. Delete Student
4. Sort Students
5. Generate Random Students
6. Display Students
7. Compare Sorting Performance
8. Exit
Enter your choice:
```

```
Enter your choice: 1
Enter Student ID: 001
Enter Student Name: aaa
Enter Student Marks (0-10): 9
Student added successfully.
```

Before sort

```
Enter your choice: 6
ID          Name                    Marks       Rank
004         acc                     8.00        Very Good
003         abc                     2.00        Fail
002         abb                     6.00        Medium
001         aaa                     9.00        Excellent
```

After sort

```
Enter your choice: 6
ID          Name                    Marks       Rank
001         aaa                     9.00        Excellent
004         acc                     8.00        Very Good
002         abb                     6.00        Medium
003         abc                     2.00        Fail
```

Thanks For Watching