TITLE: STUDENT MANAGEMENT SYSTEM
SUBTITLE: AN OVERVIEW OF THE CODE AND ITS FUNCTIONS
PRESENTED BY: DOAN MANH NGHIA

OBJECTIVE:

TO MANAGE STUDENT DATA USING A STACK-BASED APPROACH IN JAVA.

- Key Features: Add, delete, and display students..

```java
public class Student {
    private String id; // Changed to String
    private String name;
    private double marks;

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

Student Class

The Student class represents a student and contains the student's ID, name, and marks. It also includes a method for calculating the student's rank based on the marks.

Key Components:

- **Attributes:**
  - id: A String that stores the student's ID (can include alphanumeric characters).
  - name: A String that stores the student's name.
  - marks: A double that stores the student's marks.

```
public String getRanking() {
    if (marks < 5.0) return "Fail";
    else if (marks < 6.5) return "Medium";
    else if (marks < 7.5) return "Good";
    else if (marks < 9.0) return "Very Good";
    else return "Excellent";
}
```

getRanking() Method:

This method returns the ranking of the student based on their marks:

```
public String toString() {
    return "Student{" +
            "ID='" + id + '\'' +
            ", Name='" + name + '\'' +
            ", Marks=" + marks +
            ", Ranking=" + getRanking() +
            '}';
}
```

toString() Method:
This method provides a string representation of the student object for easy display.

StudentManagement Class

StudentManagement Class

The StudentManagement class is responsible for managing a collection of Student objects. It allows the user to add, update, delete, search, and display students.

Key Components:

Attributes:students: An instance of the StudentStack class, which is a custom stack used to hold student objects. (We assume StudentStack is a custom implementation of a stack with methods like push(), pop(), etc.)

```
public StudentManagement() {
    this.students = new StudentStack();
}
```

This constructor initializes the students stack, which will hold all the students.

```
public void addStudent(Student student) {
    students.push(student);
}
```

This method adds a new Student object to the stack

updateStudent() Method:

```
public void updateStudent(String id, String newName, double newMarks) {
    StudentStack tempStack = new StudentStack();
    boolean found = false;

    while (!students.isEmpty()) {
        Student student = students.pop();
        if (student.getId().equals(id)) {
            tempStack.push(new Student(id, newName, newMarks));
            found = true;
        } else {
            tempStack.push(student);
        }
    }
```

This method updates an existing student's details based on their ID It uses a temporary stack (tempStack) to search for the student with the matching ID and then updates their information. After the update, the original stack is restored.

deleteStudent() Method:

```
public void deleteStudent(String id) {
    StudentStack tempStack = new StudentStack();
    boolean found = false;

    while (!students.isEmpty()) {
        Student student = students.pop();
        if (!student.getId().equals(id)) {
            tempStack.push(student);
        } else {
            found = true;
        }
    }
```

This method deletes a student by ID. It works similarly to updateStudent(), using a temporary stack to search for the student. If the student is found, they are not pushed back onto the stack

searchStudent() Method:

```
public Student searchStudent(String id) {
    StudentStack tempStack = new StudentStack();
    Student result = null;

    while (!students.isEmpty()) {
        Student student = students.pop();
        if (student.getId().equals(id)) {
            result = student;
        }
        tempStack.push(student);
    }
```

This method searches for a student by ID and returns the student if found. It temporarily pops students from the stack to search for the matching ID and restores the stack afterward.

displayStudents() Method

```
public Student searchStudent(String id) {
    StudentStack tempStack = new StudentStack();
    Student result = null;

    while (!students.isEmpty()) {
        Student student = students.pop();
        if (student.getId().equals(id)) {
            result = student;
        }
        tempStack.push(student);
    }
```

This method displays all students in the stack. It pops each student to print their information and then restores the stack afterward.

STUDENTSTACK CLASS

The StudentStack class is a custom implementation of a stack data structure specifically designed to store Student objects. Since the requirement was to implement this stack without using Java's built-in Stack or List classes, we need to create this stack from scratch using basic Java constructs. Let's break down how the StudentStack class works and the key methods involved.

A stack is a data structure that follows the **Last In, First Out (LIFO)** principle. This means that the last element added to the stack is the first one to be removed.

The StudentStack class should provide the following basic operations:

- **Push**: Add an element (in this case, a Student) to the top of the stack.
- **Pop**: Remove and return the top element from the stack.
- **Peek**: Look at the top element without removing it.
- **isEmpty**: Check if the stack is empty.
- **Size**: (Optional) Get the current number of elements in the stack.

PUSH METHOD

```
public void push(Student student) {
    Node newNode = new Node(student);
    newNode.next = top;
    top = newNode;
}
```

This method adds a new Student to the stack.

- First, it creates a new Node containing the student.
- The new node's next pointer is set to the current top, so it links to the existing stack.
- Then, the top pointer is updated to the new node, effectively making it the new top of the stack.
- The size of the stack is incremented.

POP METHOD

```
public Student pop() {
   if (isEmpty()) throw new IllegalStateException("Stack is empty");
   Student student = top.data;
   top = top.next;
   return student;
}
```

This method removes and returns the student at the top of the stack.
If the stack is empty, it prints a message and returns null.
Otherwise, it retrieves the student from the current top node.
The top is updated to point to the next node (the one below the current top).
The size is decremented.

PEEK METHOD

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

This method returns the student at the top of the stack without removing it.

- If the stack is empty, it returns null.
- Otherwise, it simply returns the student at the top of the stack.

IS EMPTY METHOD

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

This method checks whether the stack is empty by checking if top is null

StudentManagement class

```
// Sort students by their rank using Merge Sort
public void sortStudentsByRank() {
    int size = getStackSize(); // Get the size of the stack
    Student[] studentArray = new Student[size];

    // Transfer stack elements to an array
    for (int i = 0; i < size; i++) {
        studentArray[i] = students.pop();
    }

    // Perform Merge Sort on the array
    mergeSort(studentArray, 0, size - 1);

    // Push sorted elements back into the stack
    for (int i = size - 1; i >= 0; i--) {
        students.push(studentArray[i]);
    }

    System.out.println("Students have been sorted by rank.");
}
```

- Sort student by rank
- Retrieves the stack size using the helper method getStackSize.
- Transfers all students from the stack to an array.
- Sorts the array using the Merge Sort algorithm.
- Pushes the sorted students back into the stack (to maintain stack behavior).