Slide Assignment 2 Data Structure and Algorithm

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
import java.util.InputMismatchException;
import java.util.Random;
import java.util.Scanner;

public class Main {
    private static Stack studentStack;  9 usages
    private static Random random = new Random();  2 usages

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Initialize the stack for managing students
        System.out.print("Specify the number of students to manage: ");
        int numberOfStudents = scanner.nextInt();
        studentStack = new Stack(numberOfStudents);

        // Main program loop
        while (true) {
            try {
                displayMenu();
                System.out.print("Select an option by entering the corresponding number: ");
                int choice = scanner.nextInt();

                switch (choice) {
                    case 1:
                        addStudent(scanner);
                        break;
                    case 2:
                        editStudent(scanner);
                        break;
                    case 3:
                        deleteStudent(scanner);
                        break;
                    case 4:
                        searchStudent(scanner);
                        break;
                    case 5:
                        studentStack.sortStudentsQuick();
                        break;
                    case 6:
                        studentStack.sortStudentsBubble();
                        break;
```

untitled14 > src > Main > main

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Main {
    public static void main(String[] args) {
                        break;
                    case 7:
                        studentStack.displayStudents();
                        break;
                    case 8:
                        generateRandomStudents(scanner);
                        break;
                    case 9:
                        System.out.println("Closing the program. Goodbye!");
                        scanner.close();
                        return;
                    default:
                        System.out.println("Option not recognized. Please select a valid choice.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Input error! Ensure you enter a numeric value.");
                scanner.next(); // Clear invalid input
            } catch (Exception e) {
                System.out.println("An unexpected problem occurred: " + e.getMessage());
            }
        }
    }

    private static void displayMenu() {  1 usage
        System.out.println("\n--- Student Records Management System ---");
        System.out.println("1. Register a New Student");
        System.out.println("2. Update Student Information");
        System.out.println("3. Remove a Student Record");
        System.out.println("4. Find a Student");
        System.out.println("5. Organize Students (Quick Sort)");
        System.out.println("6. Organize Students (Bubble Sort)");
        System.out.println("7. Show All Students");
        System.out.println("8. Add Randomly Generated Students");
        System.out.println("9. Exit the Program");
    }

    private static void addStudent(Scanner scanner) {  1 usage
        try {
            System.out.print("Provide a unique student ID: ");
            int id = scanner.nextInt();
```

untitled14 > src > Main > main

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Main {
    private static void addStudent(Scanner scanner) { 1 usage
            int id = scanner.nextInt();
            if (id < 0) {
                System.out.println("The ID cannot be less than zero. Please try again.");
                return;
            }

            scanner.nextLine(); // Consume newline left by nextInt()
            String name = getValidName(scanner);

            System.out.print("Enter the student's marks: ");
            double marks = scanner.nextDouble();
            if (marks < 0) {
                System.out.println("Marks must be a positive value. Please re-enter.");
                return;
            }

            Student student = new Student(id, name, marks);
            studentStack.push(student);
            System.out.println("Student has been successfully added.");
        } catch (InputMismatchException e) {
            System.out.println("Invalid data detected. Please enter correct values.");
            scanner.next(); // Clear invalid input
        }
    }

    private static void editStudent(Scanner scanner) { 1 usage
        try {
            System.out.print("Specify the ID of the student to update: ");
            int id = scanner.nextInt();
            if (id < 0) {
                System.out.println("The ID must be a non-negative number. Try again.");
                return;
            }

            scanner.nextLine(); // Consume newline left by nextInt()
            String name = getValidName(scanner);

            System.out.print("Input the updated marks for the student: ");
            double marks = scanner.nextDouble();
            if (marks < 0) {
```

untitled14 > src > Main > main

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Main {
    private static void editStudent(Scanner scanner) { 1 usage
            if (marks < 0) {
                System.out.println("Marks cannot be negative. Please input a valid number.");
                return;
            }

            studentStack.editStudent(id, name, marks);
            System.out.println("Student details have been successfully updated.");
        } catch (InputMismatchException e) {
            System.out.println("Error: Input must be in the correct format. Please try again.");
            scanner.next(); // Clear invalid input
        }
    }

    private static void deleteStudent(Scanner scanner) { 1 usage
        System.out.print("Enter the ID of the student to remove: ");
        int id = scanner.nextInt();
        studentStack.deleteStudent(id);
        System.out.println("The student record has been deleted.");
    }

    private static void searchStudent(Scanner scanner) { 1 usage
        System.out.print("Provide the student ID to locate: ");
        int id = scanner.nextInt();
        Student foundStudent = studentStack.searchStudent(id);
        if (foundStudent != null) {
            System.out.printf("Student Found! ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
                    foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
        } else {
            System.out.println("No student with this ID exists in the records.");
        }
    }

    private static void generateRandomStudents(Scanner scanner) { 1 usage
        System.out.print("How many random students should be created? ");
        int count = scanner.nextInt();

        for (int i = 0; i < count; i++) {
            int id = random.nextInt( bound: 1000); // Random ID between 0 and 999
            String name = "AutoStudent" + (i + 1); // Naming convention for generated students
            double marks = 1 + (10 - 1) * random.nextDouble(); // Marks between 1.0 and 10.0
```

untitled14 > src > Main > main

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Main {
    private static void searchStudent(Scanner scanner) {  1 usage
        int id = scanner.nextInt();
        Student foundStudent = studentStack.searchStudent(id);
        if (foundStudent != null) {
            System.out.printf("Student Found! ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
                    foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
        } else {
            System.out.println("No student with this ID exists in the records.");
        }
    }

    private static void generateRandomStudents(Scanner scanner) {  1 usage
        System.out.print("How many random students should be created? ");
        int count = scanner.nextInt();

        for (int i = 0; i < count; i++) {
            int id = random.nextInt( bound: 1000); // Random ID between 0 and 999
            String name = "AutoStudent" + (i + 1); // Naming convention for generated students
            double marks = 1 + (10 - 1) * random.nextDouble(); // Marks between 1.0 and 10.0
            Student student = new Student(id, name, marks);
            studentStack.push(student);
        }
        System.out.println(count + " random students have been successfully added.");
    }

    private static String getValidName(Scanner scanner) {  2 usages
        String name;
        while (true) {
            System.out.print("Input the full name of the student: ");
            name = scanner.nextLine();
            if (name.matches( regex: "[a-zA-Z ]+")) {
                return name;
            } else {
                System.out.println("The name should only contain letters and spaces. Please re-enter.");
            }
        }
    }
}
```

The given picture is about coding for a Java program that functions as a Student Records Management System

1.Initialization and Setup:

- The Main class initializes a Scanner object for user input and a Random object for generating random numbers.
- It also initializes a custom Stack data structure to manage student records.

2.Main Program Flow:

- The program operates in a loop where it continuously displays a menu of options for interacting with student records.
- Users can choose options like adding a new student, editing student information, deleting a student record, searching for a student, sorting students using Quick Sort or Bubble Sort, displaying all students, adding randomly generated students, and exiting the program.

3.Handling User Input:

- The program uses the Scanner object to capture user input for various operations.
- It includes robust error handling to manage scenarios where users input incorrect data types or formats.

4. Student Operations:
    - Adding a Student (addStudent):Users can add a new student by providing a unique ID, name, and marks. Input validation checks ensure the ID is non-negative and marks are positive.
    - Editing a Student (editStudent):Users can update an existing student's information by specifying the student's ID, name, and marks.
    - Deleting a Student (deleteStudent):Users can delete a student record by entering the student's ID.
    - Searching for a Student (searchStudent):Users can search for a student by providing their ID. If found, the program displays the student's details.
    - Generating Random Students (generateRandomStudents):Users can add a specified number of randomly generated student records to the stack.
5. Utility Methods:
    - Displaying the Menu (displayMenu):Prints out a clear menu of options for the user to choose from.
    - Validating Name Input (getValidName):Ensures that the entered name contains only letters and spaces.
6. Error Handling:
- The program catches exceptions like InputMismatchException to handle input errors gracefully and guide users to enter correct values.
- Program Termination:
- Users can exit the program by selecting the appropriate option, which closes the Scanner object and gracefully terminates the program.
7. Program Termination:
- Users can exit the program by selecting the appropriate option, which closes the Scanner object and gracefully terminates the program.

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
import java.util.InputMismatchException;
import java.util.Scanner;

private static void addStudent(Scanner scanner) {  no usages
    try {
        System.out.print("Enter student ID: ");
        int id = getValidIntegerInput(scanner);
        if (id <= 0) {
            System.out.println("Invalid ID. ID must be a positive number.");
            return;
        }

        scanner.nextLine(); // Consume newline character
        System.out.print("Enter student name: ");
        String name = scanner.nextLine().trim();
        if (name.isEmpty()) {
            System.out.println("Invalid Name. Name cannot be empty.");
            return;
        }

        System.out.print("Enter student marks: ");
        double marks = getValidDoubleInput(scanner);
        if (marks < 0 || marks > 10) {
            System.out.println("Invalid Marks. Marks must be between 0 and 10.");
            return;
        }

        Student student = new Student(id, name, marks);
        Stack.push(student);
        System.out.println("Student added successfully.");
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter valid data.");
        scanner.nextLine(); // Clear invalid input
    } catch (Exception e) {
        System.out.println("An unexpected error occurred while adding the student: " + e.getMessage());
    }
}

private static void editStudent(Scanner scanner) {  no usages
    try {
        System.out.print("Enter student ID to edit: ");
        int id = getValidIntegerInput(scanner);
```

untitled14 > src > addStudent

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
private static void addStudent(Scanner scanner) {  no usages
private static void editStudent(Scanner scanner) {  no usages
    try {
        System.out.print("Enter student ID to edit: ");
        int id = getValidIntegerInput(scanner);
        if (id <= 0) {
            System.out.println("Invalid ID. ID must be a positive number.");
            return;
        }

        scanner.nextLine(); // Consume newline character
        System.out.print("Enter new student name: ");
        String name = scanner.nextLine().trim();
        if (name.isEmpty()) {
            System.out.println("Invalid Name. Name cannot be empty.");
            return;
        }

        System.out.print("Enter new student marks: ");
        double marks = getValidDoubleInput(scanner);
        if (marks < 0 || marks > 10) {
            System.out.println("Invalid Marks. Marks must be between 0 and 10.");
            return;
        }

        Stack.editStudent(id, name, marks);
        System.out.println("Student updated successfully.");
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter valid data.");
        scanner.nextLine(); // Clear invalid input
    } catch (Exception e) {
        System.out.println("An unexpected error occurred while editing the student: " + e.getMessage());
    }
}

private static void deleteStudent(Scanner scanner) {  no usages
    try {
        System.out.print("Enter student ID to delete: ");
        int id = getValidIntegerInput(scanner);
        if (id <= 0) {
            System.out.println("Invalid ID. ID must be a positive number.");
            return;
```

untitled14 > src > addStudent

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
    private static void addStudent(Scanner scanner) {  no usages
    private static void deleteStudent(Scanner scanner) {  no usages
            return;
        }

        Stack.deleteStudent(id);
        System.out.println("Student deleted successfully.");
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter a valid integer ID.");
        scanner.nextLine(); // Clear invalid input
    } catch (Exception e) {
        System.out.println("An unexpected error occurred while deleting the student: " + e.getMessage());
    }
}

private static void searchStudent(Scanner scanner) {  no usages
    try {
        System.out.print("Enter student ID to search: ");
        int id = getValidIntegerInput(scanner);
        if (id <= 0) {
            System.out.println("Invalid ID. ID must be a positive number.");
            return;
        }

        Student foundStudent = Stack.searchStudent(id);
        if (foundStudent != null) {
            System.out.printf("Found: ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
                    foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
        } else {
            System.out.println("No student found with the given ID.");
        }
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter a valid integer ID.");
        scanner.nextLine(); // Clear invalid input
    } catch (Exception e) {
        System.out.println("An unexpected error occurred while searching for the student: " + e.getMessage());
    }
}

private static int getValidIntegerInput(Scanner scanner) {  4 usages
    while (true) {
        try {
```

untitled14 > src > addStudent

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
    private static void addStudent(Scanner scanner) {  no usages
    private static void searchStudent(Scanner scanner) {  no usages
            System.out.printf("Found: ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
                    foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
        } else {
            System.out.println("No student found with the given ID.");
        }
    } catch (InputMismatchException e) {
        System.out.println("Invalid input. Please enter a valid integer ID.");
        scanner.nextLine(); // Clear invalid input
    } catch (Exception e) {
        System.out.println("An unexpected error occurred while searching for the student: " + e.getMessage());
    }
}

private static int getValidIntegerInput(Scanner scanner) {  4 usages
    while (true) {
        try {
            return scanner.nextInt();
        } catch (InputMismatchException e) {
            System.out.print("Invalid input. Please enter a valid integer: ");
            scanner.nextLine(); // Clear invalid input
        }
    }
}

private static double getValidDoubleInput(Scanner scanner) {  2 usages
    while (true) {
        try {
            return scanner.nextDouble();
        } catch (InputMismatchException e) {
            System.out.print("Invalid input. Please enter a valid number: ");
            scanner.nextLine(); // Clear invalid input
        }
    }
}

public void main() {
}
```

untitled14 > src > addStudent

1. addStudent(Scanner scanner) Method:
    - This method enables the user to add a new student to the system.
    - It prompts the user to enter the student's ID, name, and marks.
    - Input validations are performed to ensure that the ID is positive, the name is not empty, and the marks are within the range of 0 to 10.
    - If the input is invalid, appropriate error messages are displayed.
    - If the input is valid, a new Student object is created and pushed onto a Stack data structure, representing the student records.
2. editStudent(Scanner scanner) Method:
    - This method allows the user to edit an existing student's information.
    - The user is asked to provide the ID of the student to be edited, along with the new name and marks.
    - Similar input validations are performed as in the addStudent method.
    - If the student with the provided ID exists, their information is updated in the system.
3. deleteStudent(Scanner scanner) Method:
    - This method facilitates the deletion of a student record.
    - The user is prompted to enter the ID of the student to be deleted.
    - If the student with the specified ID is found, it is removed from the system.
4. searchStudent(Scanner scanner) Method:
    - This method allows users to search for a student by their ID.
    - After entering the ID, the system searches for the student.
    - If the student is found, their details (ID, name, marks, and rank) are displayed. Otherwise, a message indicating that the student was not found is shown.
5. Input Validation Methods:
    - getValidIntegerInput(Scanner scanner) and getValidDoubleInput(Scanner scanner) methods ensure valid integer and double inputs, respectively. They handle cases where users input incorrect data types.
6. Error Handling:
    - The code includes try-catch blocks to handle exceptions like InputMismatchException that may occur during user input.
    - If invalid input is provided, the code prompts the user to enter the correct type of data.
7. main() Method:
    - The code snippet does not contain a complete main method. A main method is typically the entry point of a Java program.

```java
public class Student {  17 usages
    private int id;  3 usages
    private String name;  3 usages
    private double marks;  7 usages

    public Student(int id, String name, double marks) {  4 usages
        if (id <= 0) {
            throw new IllegalArgumentException("Invalid ID. Must be a positive number.");
        }
        if (name == null || name.trim().isEmpty()) {
            throw new IllegalArgumentException("Invalid Name. Name cannot be empty.");
        }
        if (marks < 0 || marks > 10) {
            throw new IllegalArgumentException("Invalid Marks. Must be between 0 and 10.");
        }
        this.id = id;
        this.name = name.trim();
        this.marks = marks;
    }

    public int getId() { return id; }

    public String getName() { return name; }

    public double getMarks() { return marks; }

    public String getRank() {  4 usages
        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";
        }
    }

    @Override
    public String toString() {
        return "Student ID: " + id + ", Name: " + name + ", Marks: " + marks + ", Ranking: " + getRank();
    }
}
```

1. Student Class:
   - The Student class encapsulates the attributes and behavior of a student, including ID, name, marks, and methods to access and manipulate this information.
2. Attributes:
   - id: Represents the unique identifier of the student.
   - name: Stores the name of the student.
   - marks: Holds the marks obtained by the student.
3. Constructor:
   - The class defines a constructor that initializes a Student object with the provided ID, name, and marks.
   - Input validations are performed within the constructor to ensure that the provided values meet certain criteria:
     - ID must be a positive number (> 0).
     - Name cannot be null or an empty string after trimming.
     - Marks must be within the range of 0 to 10.
   - If any of these conditions are violated, an IllegalArgumentException is thrown with an appropriate error message.

4. Getter Methods:

- getId(), getName(), getMarks(): These methods allow access to the student's ID, name, and marks respectively.
- getRank(): Calculates and returns the ranking based on the student's marks:
  - "Fail" for marks less than 5.0.
  - "Medium" for marks between 5.0 and 6.5.
  - "Good" for marks between 6.5 and 7.5.
  - "Very Good" for marks between 7.5 and 9.0.
  - "Excellent" for marks 9.0 and above.

5. toString() Method:

- The toString() method is overridden to provide a string representation of the Student object.
- It returns a formatted string containing the student's ID, name, marks, and their ranking obtained using the getRank() method.

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
import java.util.Arrays;

public class Stack { 6 usages
    private static Student[] students; 18 usages
    private static int top; 12 usages
    private static int size; 2 usages

    public Stack(int size) { 1 usage
        if (size <= 0) {
            throw new IllegalArgumentException("The stack size must be a positive number.");
        }
        this.size = size;
        students = new Student[size];
        top = -1;
    }

    public static void push(Student student) { 3 usages
        if (top >= size - 1) {
            System.out.println("Unable to add more students. The stack is at full capacity.");
            return;
        }
        students[++top] = student;
        System.out.println("The student has been successfully added to the stack.");
    }

    public static void editStudent(int id, String name, double marks) { 2 usages
        if (id <= 0) {
            System.out.println("Error: Student ID must be greater than zero.");
            return;
        }

        for (int i = 0; i <= top; i++) {
            if (students[i].getId() == id) {
                try {
                    students[i] = new Student(id, name, marks);
                    System.out.println("Student details have been successfully updated.");
                } catch (IllegalArgumentException e) {
                    System.out.println("Failed to update student: " + e.getMessage());
                }
                return;
            }
        }
```

untitled14 > src > Stack

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Stack { 6 usages
    public static void editStudent(int id, String name, double marks) { 2 usages
        }
        System.out.println("No student found with the ID " + id + ".");
    }

    public static void deleteStudent(int id) { 2 usages
        if (id <= 0) {
            System.out.println("Error: The student ID must be a positive number.");
            return;
        }

        for (int i = 0; i <= top; i++) {
            if (students[i].getId() == id) {
                for (int j = i; j < top; j++) {
                    students[j] = students[j + 1]; // Shift elements
                }
                students[top--] = null; // Clear the last position
                System.out.println("Student has been successfully removed.");
                return;
            }
        }
        System.out.println("No record found for the student ID " + id + ".");
    }

    public static Student searchStudent(int id) { 2 usages
        if (id <= 0) {
            System.out.println("Invalid input: The ID must be a positive integer.");
            return null;
        }

        for (int i = 0; i <= top; i++) {
            if (students[i].getId() == id) {
                return students[i];
            }
        }
        System.out.println("Student with the ID " + id + " could not be located.");
        return null;
    }

    public void sortStudentsQuick() { 1 usage
        long startTime = System.nanoTime();
```

untitled14 > src > Stack

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Stack {  6 usages
    public void sortStudentsQuick() {  1 usage
        long startTime = System.nanoTime();
        quickSort(students, low: 0, top);
        long endTime = System.nanoTime();
        long durationNanoseconds = (endTime - startTime);
        System.out.println("Students have been sorted by ID using Quick Sort in " + durationNanoseconds + " nanoseconds.");
    }

    private void quickSort(Student[] array, int low, int high) {  3 usages
        if (low < high) {
            int pi = partition(array, low, high);

            quickSort(array, low, high: pi - 1);
            quickSort(array, low: pi + 1, high);
        }
    }


    private int partition(Student[] array, int low, int high) {  1 usage
        int pivot = array[high].getId();
        int i = (low - 1);

        for (int j = low; j < high; j++) {
            if (array[j].getId() <= pivot) {
                i++;
                swap(array, i, j);
            }
        }
        swap(array, i: i + 1, high);
        return i + 1;
    }

    private void swap(Student[] array, int i, int j) {  3 usages
        Student temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public void sortStudentsBubble() {  1 usage
        long startTime = System.nanoTime();
        bubbleSort();
```

untitled14 > src > Stack

untitled14 Version control

Main.java addStudent.java Stack.java Student.java

```java
public class Stack {  6 usages
    }

    private void swap(Student[] array, int i, int j) {  3 usages
        Student temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public void sortStudentsBubble() {  1 usage
        long startTime = System.nanoTime();
        bubbleSort();
        long endTime = System.nanoTime();
        long durationNanoseconds = (endTime - startTime);
        System.out.println("Students have been sorted by ID using Bubble Sort in " + durationNanoseconds + " nanoseconds.");
    }

    private void bubbleSort() {  1 usage
        int n = top + 1;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (students[j].getId() > students[j + 1].getId()) {
                    swap(students, j, j: j + 1);
                }
            }
        }
    }

    public void displayStudents() {  1 usage
        if (top == -1) {
            System.out.println("The stack is empty. No student records to show.");
            return;
        }
        for (int i = 0; i <= top; i++) {
            System.out.printf("Student ID: %d | Name: %s | Marks: %.2f | Rank: %s%n",
                    students[i].getId(), students[i].getName(), students[i].getMarks(), students[i].getRank());
        }
    }
}
```

untitled14 > src > Stack

1. Stack Class:
    - The Stack class maintains an array of Student objects as a stack data structure.
2. Attributes:
    - students: An array to store Student objects.
    - top: An integer representing the index of the top element in the stack.
    - size: The maximum size of the stack.
3. Constructor:
    - Initializes the stack with a given size. If the size is less than or equal to 0, an IllegalArgumentException is thrown.
4. push Method:
    - Adds a Student object to the stack if there is space available. If the stack is full, a message indicating the stack is at capacity is printed.
5. editStudent Method:
    - Edits the details of a student with a given ID by creating a new Student object with updated information. If the student is not found, a message is displayed.

6. deleteStudent Method:
    - Deletes a student with a given ID from the stack by shifting elements in the array. If the student is not found, a corresponding message is printed.
7. searchStudent Method:
    - Searches for a student with a given ID in the stack and returns the Student object if found. If not found, a message is displayed.
8. Sorting Methods:
    - sortStudentsQuick and sortStudentsBubble methods are provided for sorting students based on their IDs using Quick Sort and Bubble Sort algorithms respectively.
    - quickSort and associated methods are used for Quick Sort implementation.
    - bubbleSort method is used for Bubble Sort implementation.
9. Utility Methods:
    - swap: Swaps two Student objects within an array.
    - displayStudents: Displays the details of all students in the stack, including ID, name, marks, and rank.
10. Output Handling:

- Various messages are printed to the console to inform about the success or failure of operations like adding, editing, deleting, and searching students.