

# Slide Assignment 2 Data Structure and Algorithm

U untitled14 Version control

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

```
1 import java.util.InputMismatchException;
2 import java.util.Random;
3 import java.util.Scanner;
4
5 public class Main {
6     private static Stack studentStack; 9 usages
7     private static Random random = new Random(); 2 usages
8
9     public static void main(String[] args) {
10         Scanner scanner = new Scanner(System.in);
11
12         // Initialize the stack for managing students
13         System.out.print("Specify the number of students to manage: ");
14         int numberOfStudents = scanner.nextInt();
15         studentStack = new Stack(numberOfStudents);
16
17         // Main program loop
18         while (true) {
19             try {
20                 displayMenu();
21                 System.out.print("Select an option by entering the corresponding number: ");
22                 int choice = scanner.nextInt();
23
24                 switch (choice) {
25                     case 1:
26                         addStudent(scanner);
27                         break;
28                     case 2:
29                         editStudent(scanner);
30                         break;
31                     case 3:
32                         deleteStudent(scanner);
33                         break;
34                     case 4:
35                         searchStudent(scanner);
36                         break;
37                     case 5:
38                         studentStack.sortStudentsQuick();
39                         break;
40                     case 6:
41                         studentStack.sortStudentsBubble();
42
```

untitled14 > src > Main > main

U untitled14 Version control

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

```
5     public class Main {
9         public static void main(String[] args) {
42             break;
43             case 7:
44                 studentStack.displayStudents();
45             break;
46             case 8:
47                 generateRandomStudents(scanner);
48             break;
49             case 9:
50                 System.out.println("Closing the program. Goodbye!");
51                 scanner.close();
52                 return;
53             default:
54                 System.out.println("Option not recognized. Please select a valid choice.");
55         }
56     } catch (InputMismatchException e) {
57         System.out.println("Input error! Ensure you enter a numeric value.");
58         scanner.next(); // Clear invalid input
59     } catch (Exception e) {
60         System.out.println("An unexpected problem occurred: " + e.getMessage());
61     }
62
63 }
64
65 private static void displayMenu() { 1 usage
66     System.out.println("\n--- Student Records Management System ---");
67     System.out.println("1. Register a New Student");
68     System.out.println("2. Update Student Information");
69     System.out.println("3. Remove a Student Record");
70     System.out.println("4. Find a Student");
71     System.out.println("5. Organize Students (Quick Sort)");
72     System.out.println("6. Organize Students (Bubble Sort)");
73     System.out.println("7. Show All Students");
74     System.out.println("8. Add Randomly Generated Students");
75     System.out.println("9. Exit the Program");
76 }
77
78 @ private static void addStudent(Scanner scanner) { 1 usage
79     try {
80         System.out.print("Provide a unique student ID: ");
81         int id = scanner.nextInt();
```

untitled14 > src > Main > main

U

untitled14

Version control

Main.java

addStudent.java

Stack.java

Student.java

5

public class Main {

78

private static void addStudent(Scanner scanner) { 1 usage

81

int id = scanner.nextInt();

82

if (id < 0) {

83

System.out.println("The ID cannot be less than zero. Please try again.");

84

return;

85

}

86

87

scanner.nextLine(); // Consume newline left by nextInt()

88

String name = getValidName(scanner);

89

90

System.out.print("Enter the student's marks: ");

91

double marks = scanner.nextDouble();

92

if (marks < 0) {

93

System.out.println("Marks must be a positive value. Please re-enter.");

94

return;

95

}

96

97

Student student = new Student(id, name, marks);

98

studentStack.push(student);

99

System.out.println("Student has been successfully added.");

100

} catch (InputMismatchException e) {

101

System.out.println("Invalid data detected. Please enter correct values.");

102

scanner.next(); // Clear invalid input

103

}

104

}

105

106

@ private static void editStudent(Scanner scanner) { 1 usage

107

try {

108

System.out.print("Specify the ID of the student to update: ");

109

int id = scanner.nextInt();

110

if (id < 0) {

111

System.out.println("The ID must be a non-negative number. Try again.");

112

return;

113

}

114

115

scanner.nextLine(); // Consume newline left by nextInt()

116

String name = getValidName(scanner);

117

118

System.out.print("Input the updated marks for the student: ");

119

double marks = scanner.nextDouble();

120

if (marks < 0) {

untitled14

>

src

>

Main

>

main

U

untitled14

Version control

Main.java

addStudent.java

Stack.java

Student.java

5

public class Main {

106

private static void editStudent(Scanner scanner) { 1 usage

120

if (marks < 0) {

121

System.out.println("Marks cannot be negative. Please input a valid number.");

122

return;

123

}

124

125

studentStack.editStudent(id, name, marks);

126

System.out.println("Student details have been successfully updated.");

127

} catch (InputMismatchException e) {

128

System.out.println("Error: Input must be in the correct format. Please try again.");

129

scanner.next(); // Clear invalid input

130

}

131

}

132

133

@ private static void deleteStudent(Scanner scanner) { 1 usage

134

System.out.print("Enter the ID of the student to remove: ");

135

int id = scanner.nextInt();

136

studentStack.deleteStudent(id);

137

System.out.println("The student record has been deleted.");

138

}

139

140

@ private static void searchStudent(Scanner scanner) { 1 usage

141

System.out.print("Provide the student ID to locate: ");

142

int id = scanner.nextInt();

143

Student foundStudent = studentStack.searchStudent(id);

144

if (foundStudent != null) {

145

System.out.printf("Student Found! ID: %d, Name: %s, Marks: %.2f, Rank: %s\n",

146

foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());

147

} else {

148

System.out.println("No student with this ID exists in the records.");

149

}

150

}

151

152

@ private static void generateRandomStudents(Scanner scanner) { 1 usage

153

System.out.print("How many random students should be created? ");

154

int count = scanner.nextInt();

155

156

for (int i = 0; i < count; i++) {

157

int id = random.nextInt( bound: 1000); // Random ID between 0 and 999

158

String name = "AutoStudent" + (i + 1); // Naming convention for generated students

159

double marks = 1 + (10 - 1) \* random.nextDouble(); // Marks between 1.0 and 10.0

untitled14

>

src

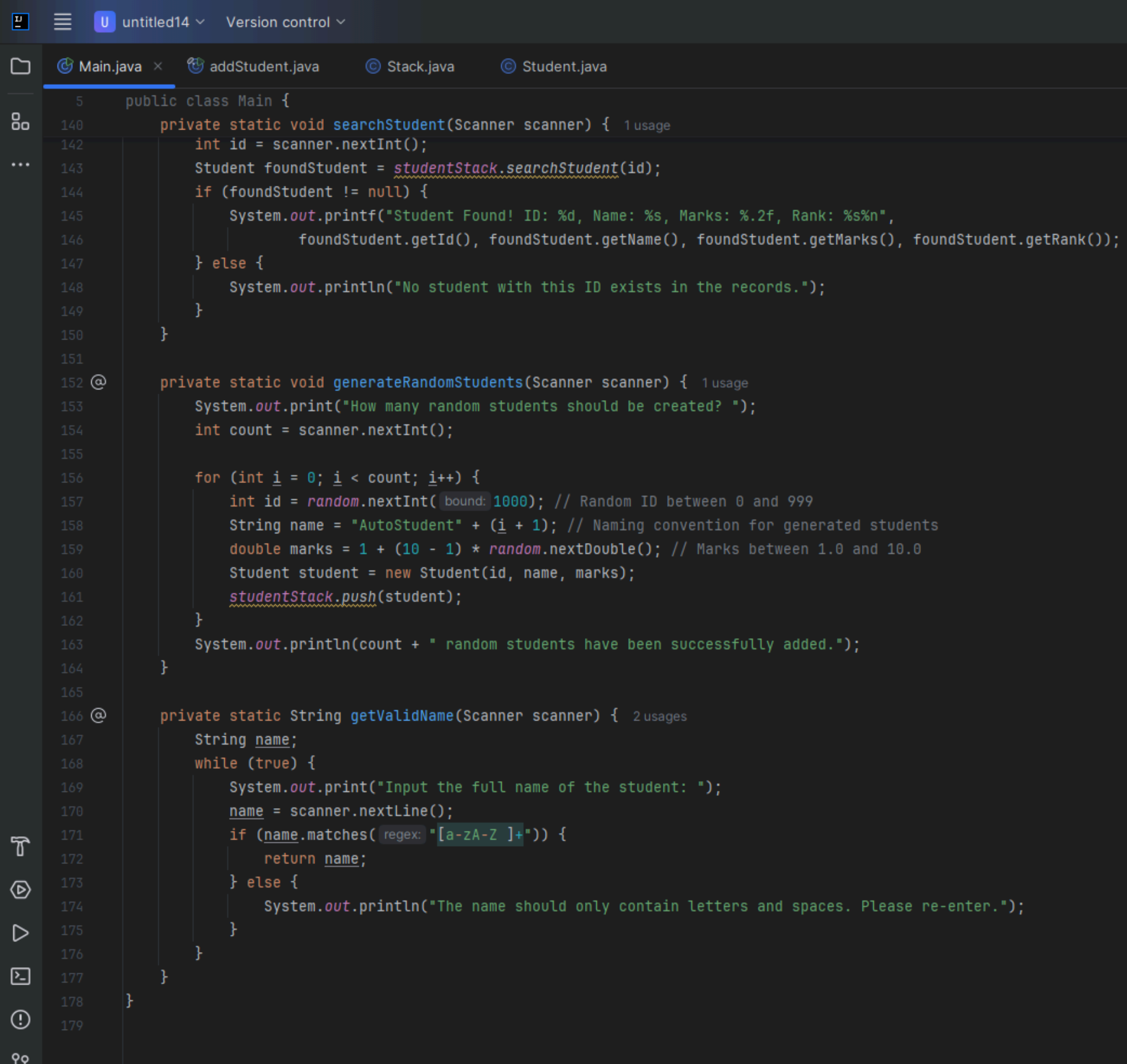
>

Main

>

main





```
5 public class Main {
140     private static void searchStudent(Scanner scanner) { 1 usage
142         int id = scanner.nextInt();
143         Student foundStudent = studentStack.searchStudent(id);
144         if (foundStudent != null) {
145             System.out.printf("Student Found! ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
146                 foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
147         } else {
148             System.out.println("No student with this ID exists in the records.");
149         }
150     }
151
152     @ private static void generateRandomStudents(Scanner scanner) { 1 usage
153         System.out.print("How many random students should be created? ");
154         int count = scanner.nextInt();
155
156         for (int i = 0; i < count; i++) {
157             int id = random.nextInt( bound: 1000); // Random ID between 0 and 999
158             String name = "AutoStudent" + (i + 1); // Naming convention for generated students
159             double marks = 1 + (10 - 1) * random.nextDouble(); // Marks between 1.0 and 10.0
160             Student student = new Student(id, name, marks);
161             studentStack.push(student);
162         }
163         System.out.println(count + " random students have been successfully added.");
164     }
165
166     @ private static String getValidName(Scanner scanner) { 2 usages
167         String name;
168         while (true) {
169             System.out.print("Input the full name of the student: ");
170             name = scanner.nextLine();
171             if (name.matches( regex: "[a-zA-Z ]+")) {
172                 return name;
173             } else {
174                 System.out.println("The name should only contain letters and spaces. Please re-enter.");
175             }
176         }
177     }
178 }
179 }
```

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.

U

untitled14

Version control

Main.java

addStudent.java

Stack.java

Student.java

1

import java.util.InputMismatchException;

2

import java.util.Scanner;

3

4

private static void addStudent(Scanner scanner) { no usages

5

try {

6

System.out.print("Enter student ID: ");

7

int id = getValidIntegerInput(scanner);

8

if (id <= 0) {

9

System.out.println("Invalid ID. ID must be a positive number.");

10

return;

11

}

12

13

scanner.nextLine(); // Consume newline character

14

System.out.print("Enter student name: ");

15

String name = scanner.nextLine().trim();

16

if (name.isEmpty()) {

17

System.out.println("Invalid Name. Name cannot be empty.");

18

return;

19

}

20

21

System.out.print("Enter student marks: ");

22

double marks = getValidDoubleInput(scanner);

23

if (marks < 0 || marks > 10) {

24

System.out.println("Invalid Marks. Marks must be between 0 and 10.");

25

return;

26

}

27

28

Student student = new Student(id, name, marks);

29

Stack.push(student);

30

System.out.println("Student added successfully.");

31

} catch (InputMismatchException e) {

32

System.out.println("Invalid input. Please enter valid data.");

33

scanner.nextLine(); // Clear invalid input

34

} catch (Exception e) {

35

System.out.println("An unexpected error occurred while adding the student: " + e.getMessage());

36

}

37

}

38

39

private static void editStudent(Scanner scanner) { no usages

40

try {

41

System.out.print("Enter student ID to edit: ");

42

int id = getValidIntegerInput(scanner);

untitled14 > src > addStudent

U

untitled14

Version control

Main.java

addStudent.java

Stack.java

Student.java

4

private static void addStudent(Scanner scanner) { no usages

39

private static void editStudent(Scanner scanner) { no usages

40

try {

41

System.out.print("Enter student ID to edit: ");

42

int id = getValidIntegerInput(scanner);

43

if (id <= 0) {

44

System.out.println("Invalid ID. ID must be a positive number.");

45

return;

46

}

47

48

scanner.nextLine(); // Consume newline character

49

System.out.print("Enter new student name: ");

50

String name = scanner.nextLine().trim();

51

if (name.isEmpty()) {

52

System.out.println("Invalid Name. Name cannot be empty.");

53

return;

54

}

55

56

System.out.print("Enter new student marks: ");

57

double marks = getValidDoubleInput(scanner);

58

if (marks < 0 || marks > 10) {

59

System.out.println("Invalid Marks. Marks must be between 0 and 10.");

60

return;

61

}

62

63

Stack.editStudent(id, name, marks);

64

System.out.println("Student updated successfully.");

65

} catch (InputMismatchException e) {

66

System.out.println("Invalid input. Please enter valid data.");

67

scanner.nextLine(); // Clear invalid input

68

} catch (Exception e) {

69

System.out.println("An unexpected error occurred while editing the student: " + e.getMessage());

70

}

71

}

72

73

private static void deleteStudent(Scanner scanner) { no usages

74

try {

75

System.out.print("Enter student ID to delete: ");

76

int id = getValidIntegerInput(scanner);

77

if (id <= 0) {

78

System.out.println("Invalid ID. ID must be a positive number.");

79

return;

untitled14 > src > addStudent



U

untitled14

Version control

Main.java

addStudent.java

Stack.java

Student.java

```
4 private static void addStudent(Scanner scanner) { no usages
73 private static void deleteStudent(Scanner scanner) { no usages
79     return;
80 }
81
82     Stack.deleteStudent(id);
83     System.out.println("Student deleted successfully.");
84 } catch (InputMismatchException e) {
85     System.out.println("Invalid input. Please enter a valid integer ID.");
86     scanner.nextLine(); // Clear invalid input
87 } catch (Exception e) {
88     System.out.println("An unexpected error occurred while deleting the student: " + e.getMessage());
89 }
90 }
91
92 private static void searchStudent(Scanner scanner) { no usages
93     try {
94         System.out.print("Enter student ID to search: ");
95         int id = getValidIntegerInput(scanner);
96         if (id <= 0) {
97             System.out.println("Invalid ID. ID must be a positive number.");
98             return;
99         }
100
101         Student foundStudent = Stack.searchStudent(id);
102         if (foundStudent != null) {
103             System.out.printf("Found: ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
104                 foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
105         } else {
106             System.out.println("No student found with the given ID.");
107         }
108     } catch (InputMismatchException e) {
109         System.out.println("Invalid input. Please enter a valid integer ID.");
110         scanner.nextLine(); // Clear invalid input
111     } catch (Exception e) {
112         System.out.println("An unexpected error occurred while searching for the student: " + e.getMessage());
113     }
114 }
115
116 @ private static int getValidIntegerInput(Scanner scanner) { 4 usages
117     while (true) {
118         try {
119             return scanner.nextInt();
120         } catch (InputMismatchException e) {
121             System.out.print("Invalid input. Please enter a valid integer: ");
122             scanner.nextLine(); // Clear invalid input
123         }
124     }
125 }
126
127 @ private static double getValidDoubleInput(Scanner scanner) { 2 usages
128     while (true) {
129         try {
130             return scanner.nextDouble();
131         } catch (InputMismatchException e) {
132             System.out.print("Invalid input. Please enter a valid number: ");
133             scanner.nextLine(); // Clear invalid input
134         }
135     }
136 }
137
138 public void main() {
139 }
140
```

untitled14 > src > addStudent

U

untitled14

Version control

Main.java

addStudent.java

Stack.java

Student.java

```
4 private static void addStudent(Scanner scanner) { no usages
92 private static void searchStudent(Scanner scanner) { no usages
103     System.out.printf("Found: ID: %d, Name: %s, Marks: %.2f, Rank: %s%n",
104         foundStudent.getId(), foundStudent.getName(), foundStudent.getMarks(), foundStudent.getRank());
105     } else {
106         System.out.println("No student found with the given ID.");
107     }
108 } catch (InputMismatchException e) {
109     System.out.println("Invalid input. Please enter a valid integer ID.");
110     scanner.nextLine(); // Clear invalid input
111 } catch (Exception e) {
112     System.out.println("An unexpected error occurred while searching for the student: " + e.getMessage());
113 }
114 }
115
116 @ private static int getValidIntegerInput(Scanner scanner) { 4 usages
117     while (true) {
118         try {
119             return scanner.nextInt();
120         } catch (InputMismatchException e) {
121             System.out.print("Invalid input. Please enter a valid integer: ");
122             scanner.nextLine(); // Clear invalid input
123         }
124     }
125 }
126
127 @ private static double getValidDoubleInput(Scanner scanner) { 2 usages
128     while (true) {
129         try {
130             return scanner.nextDouble();
131         } catch (InputMismatchException e) {
132             System.out.print("Invalid input. Please enter a valid number: ");
133             scanner.nextLine(); // Clear invalid input
134         }
135     }
136 }
137
138 public void main() {
139 }
140
```

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.



```
untitled14  Version control  ▾
Main.java  addStudent.java  Stack.java  Student.java  ×
1  public class Student { 17 usages
2      private int id; 3 usages
3      private String name; 3 usages
4      private double marks; 7 usages
5
6      public Student(int id, String name, double marks) { 4 usages
7          if (id <= 0) {
8              throw new IllegalArgumentException("Invalid ID. Must be a positive number.");
9          }
10         if (name == null || name.trim().isEmpty()) {
11             throw new IllegalArgumentException("Invalid Name. Name cannot be empty.");
12         }
13         if (marks < 0 || marks > 10) {
14             throw new IllegalArgumentException("Invalid Marks. Must be between 0 and 10.");
15         }
16         this.id = id;
17         this.name = name.trim();
18         this.marks = marks;
19     }
20
21     public int getId() { return id; }
22
23
24     public String getName() { return name; }
25
26
27
28     public double getMarks() { return marks; }
29
30
31
32
33     public String getRank() { 4 usages
34         if (marks < 5.0) {
35             return "Fail";
36         } else if (marks < 6.5) {
37             return "Medium";
38         } else if (marks < 7.5) {
39             return "Good";
40         } else if (marks < 9.0) {
41             return "Very Good";
42         } else {
43             return "Excellent";
44         }
45     }
46
47     @Override
48     public String toString() {
```

```
47     @Override
48     public String toString() {
49         return "Student ID: " + id + ", Name: " + name + ", Marks: " + marks + ", Ranking: " + getRank();
50     }
51 }
52 |
untitled14 > src > Student
```

## 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.

U

untitled14

Version control

Stack.java

1import java.util.Arrays;  
2  
3public class Stack { 6 usages  
4 private static Student[] *students*; 18 usages  
5 private static int *top*; 12 usages  
6 private static int *size*; 2 usages  
7  
8 public Stack(int size) { 1 usage  
9 if (size <= 0) {  
10 throw new IllegalArgumentException("The stack size must be a positive number.");  
11 }  
12 *this.size* = size;  
13 *students* = new Student[size];  
14 *top* = -1;  
15 }  
16  
17 public static void push(Student student) { 3 usages  
18 if (*top* >= *size* - 1) {  
19 System.out.println("Unable to add more students. The stack is at full capacity.");  
20 return;  
21 }  
22 *students*[++*top*] = student;  
23 System.out.println("The student has been successfully added to the stack.");  
24 }  
25  
26 public static void editStudent(int id, String name, double marks) { 2 usages  
27 if (id <= 0) {  
28 System.out.println("Error: Student ID must be greater than zero.");  
29 return;  
30 }  
31  
32 for (int *i* = 0; *i* <= *top*; *i*++) {  
33 if (*students*[*i*].getId() == id) {  
34 try {  
35 *students*[*i*] = new Student(id, name, marks);  
36 System.out.println("Student details have been successfully updated.");  
37 } catch (IllegalArgumentException e) {  
38 System.out.println("Failed to update student: " + e.getMessage());  
39 }  
40 return;  
41 }  
42 }  
43 }  
44  
45 public static void deleteStudent(int id) { 2 usages  
46 if (id <= 0) {  
47 System.out.println("Error: The student ID must be a positive number.");  
48 return;  
49 }  
50  
51 for (int *i* = 0; *i* <= *top*; *i*++) {  
52 if (*students*[*i*].getId() == id) {  
53 for (int *j* = *i*; *j* < *top*; *j*++) {  
54 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
55 }  
56 *students*[*top*--] = null; // Clear the last position  
57 System.out.println("Student has been successfully removed.");  
58 return;  
59 }  
60 }  
61 System.out.println("No record found for the student ID " + id + ".");  
62 }  
63  
64 @  
65 public static Student searchStudent(int id) { 2 usages  
66 if (id <= 0) {  
67 System.out.println("Invalid input: The ID must be a positive integer.");  
68 return null;  
69 }  
70  
71 for (int *i* = 0; *i* <= *top*; *i*++) {  
72 if (*students*[*i*].getId() == id) {  
73 return *students*[*i*];  
74 }  
75 }  
76 System.out.println("Student with the ID " + id + " could not be located.");  
77 return null;  
78 }  
79  
80 public void sortStudentsQuick() { 1 usage  
81 long startTime = System.nanoTime();  
82 }  
83  
84 public static void editStudent(int id, String name, double marks) { 2 usages  
85 if (id <= 0) {  
86 System.out.println("Error: Student ID must be greater than zero.");  
87 return;  
88 }  
89  
90 for (int *i* = 0; *i* <= *top*; *i*++) {  
91 if (*students*[*i*].getId() == id) {  
92 try {  
93 *students*[*i*] = new Student(id, name, marks);  
94 System.out.println("Student details have been successfully updated.");  
95 } catch (IllegalArgumentException e) {  
96 System.out.println("Failed to update student: " + e.getMessage());  
97 }  
98 return;  
99 }  
100 }  
101 }  
102  
103 public static void deleteStudent(int id) { 2 usages  
104 if (id <= 0) {  
105 System.out.println("Error: The student ID must be a positive number.");  
106 return;  
107 }  
108  
109 for (int *i* = 0; *i* <= *top*; *i*++) {  
110 if (*students*[*i*].getId() == id) {  
111 for (int *j* = *i*; *j* < *top*; *j*++) {  
112 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
113 }  
114 *students*[*top*--] = null; // Clear the last position  
115 System.out.println("Student has been successfully removed.");  
116 return;  
117 }  
118 }  
119 System.out.println("No record found for the student ID " + id + ".");  
120 }  
121  
122 @  
123 public static Student searchStudent(int id) { 2 usages  
124 if (id <= 0) {  
125 System.out.println("Invalid input: The ID must be a positive integer.");  
126 return null;  
127 }  
128  
129 for (int *i* = 0; *i* <= *top*; *i*++) {  
130 if (*students*[*i*].getId() == id) {  
131 return *students*[*i*];  
132 }  
133 }  
134 System.out.println("Student with the ID " + id + " could not be located.");  
135 return null;  
136 }  
137  
138 public void sortStudentsQuick() { 1 usage  
139 long startTime = System.nanoTime();  
140 }  
141  
142 public static void editStudent(int id, String name, double marks) { 2 usages  
143 if (id <= 0) {  
144 System.out.println("Error: Student ID must be greater than zero.");  
145 return;  
146 }  
147  
148 for (int *i* = 0; *i* <= *top*; *i*++) {  
149 if (*students*[*i*].getId() == id) {  
150 try {  
151 *students*[*i*] = new Student(id, name, marks);  
152 System.out.println("Student details have been successfully updated.");  
153 } catch (IllegalArgumentException e) {  
154 System.out.println("Failed to update student: " + e.getMessage());  
155 }  
156 return;  
157 }  
158 }  
159 }  
160  
161 public static void deleteStudent(int id) { 2 usages  
162 if (id <= 0) {  
163 System.out.println("Error: The student ID must be a positive number.");  
164 return;  
165 }  
166  
167 for (int *i* = 0; *i* <= *top*; *i*++) {  
168 if (*students*[*i*].getId() == id) {  
169 for (int *j* = *i*; *j* < *top*; *j*++) {  
170 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
171 }  
172 *students*[*top*--] = null; // Clear the last position  
173 System.out.println("Student has been successfully removed.");  
174 return;  
175 }  
176 }  
177 System.out.println("No record found for the student ID " + id + ".");  
178 }  
179  
180 @  
181 public static Student searchStudent(int id) { 2 usages  
182 if (id <= 0) {  
183 System.out.println("Invalid input: The ID must be a positive integer.");  
184 return null;  
185 }  
186  
187 for (int *i* = 0; *i* <= *top*; *i*++) {  
188 if (*students*[*i*].getId() == id) {  
189 return *students*[*i*];  
190 }  
191 }  
192 System.out.println("Student with the ID " + id + " could not be located.");  
193 return null;  
194 }  
195  
196 public void sortStudentsQuick() { 1 usage  
197 long startTime = System.nanoTime();  
198 }  
199  
200 public static void editStudent(int id, String name, double marks) { 2 usages  
201 if (id <= 0) {  
202 System.out.println("Error: Student ID must be greater than zero.");  
203 return;  
204 }  
205  
206 for (int *i* = 0; *i* <= *top*; *i*++) {  
207 if (*students*[*i*].getId() == id) {  
208 try {  
209 *students*[*i*] = new Student(id, name, marks);  
210 System.out.println("Student details have been successfully updated.");  
211 } catch (IllegalArgumentException e) {  
212 System.out.println("Failed to update student: " + e.getMessage());  
213 }  
214 return;  
215 }  
216 }  
217 }  
218  
219 public static void deleteStudent(int id) { 2 usages  
220 if (id <= 0) {  
221 System.out.println("Error: The student ID must be a positive number.");  
222 return;  
223 }  
224  
225 for (int *i* = 0; *i* <= *top*; *i*++) {  
226 if (*students*[*i*].getId() == id) {  
227 for (int *j* = *i*; *j* < *top*; *j*++) {  
228 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
229 }  
230 *students*[*top*--] = null; // Clear the last position  
231 System.out.println("Student has been successfully removed.");  
232 return;  
233 }  
234 }  
235 System.out.println("No record found for the student ID " + id + ".");  
236 }  
237  
238 @  
239 public static Student searchStudent(int id) { 2 usages  
240 if (id <= 0) {  
241 System.out.println("Invalid input: The ID must be a positive integer.");  
242 return null;  
243 }  
244  
245 for (int *i* = 0; *i* <= *top*; *i*++) {  
246 if (*students*[*i*].getId() == id) {  
247 return *students*[*i*];  
248 }  
249 }  
250 System.out.println("Student with the ID " + id + " could not be located.");  
251 return null;  
252 }  
253  
254 public void sortStudentsQuick() { 1 usage  
255 long startTime = System.nanoTime();  
256 }  
257  
258 public static void editStudent(int id, String name, double marks) { 2 usages  
259 if (id <= 0) {  
260 System.out.println("Error: Student ID must be greater than zero.");  
261 return;  
262 }  
263  
264 for (int *i* = 0; *i* <= *top*; *i*++) {  
265 if (*students*[*i*].getId() == id) {  
266 try {  
267 *students*[*i*] = new Student(id, name, marks);  
268 System.out.println("Student details have been successfully updated.");  
269 } catch (IllegalArgumentException e) {  
270 System.out.println("Failed to update student: " + e.getMessage());  
271 }  
272 return;  
273 }  
274 }  
275 }  
276  
277 public static void deleteStudent(int id) { 2 usages  
278 if (id <= 0) {  
279 System.out.println("Error: The student ID must be a positive number.");  
280 return;  
281 }  
282  
283 for (int *i* = 0; *i* <= *top*; *i*++) {  
284 if (*students*[*i*].getId() == id) {  
285 for (int *j* = *i*; *j* < *top*; *j*++) {  
286 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
287 }  
288 *students*[*top*--] = null; // Clear the last position  
289 System.out.println("Student has been successfully removed.");  
290 return;  
291 }  
292 }  
293 System.out.println("No record found for the student ID " + id + ".");  
294 }  
295  
296 @  
297 public static Student searchStudent(int id) { 2 usages  
298 if (id <= 0) {  
299 System.out.println("Invalid input: The ID must be a positive integer.");  
300 return null;  
301 }  
302  
303 for (int *i* = 0; *i* <= *top*; *i*++) {  
304 if (*students*[*i*].getId() == id) {  
305 return *students*[*i*];  
306 }  
307 }  
308 System.out.println("Student with the ID " + id + " could not be located.");  
309 return null;  
310 }  
311  
312 public void sortStudentsQuick() { 1 usage  
313 long startTime = System.nanoTime();  
314 }  
315  
316 public static void editStudent(int id, String name, double marks) { 2 usages  
317 if (id <= 0) {  
318 System.out.println("Error: Student ID must be greater than zero.");  
319 return;  
320 }  
321  
322 for (int *i* = 0; *i* <= *top*; *i*++) {  
323 if (*students*[*i*].getId() == id) {  
324 try {  
325 *students*[*i*] = new Student(id, name, marks);  
326 System.out.println("Student details have been successfully updated.");  
327 } catch (IllegalArgumentException e) {  
328 System.out.println("Failed to update student: " + e.getMessage());  
329 }  
330 return;  
331 }  
332 }  
333 }  
334  
335 public static void deleteStudent(int id) { 2 usages  
336 if (id <= 0) {  
337 System.out.println("Error: The student ID must be a positive number.");  
338 return;  
339 }  
340  
341 for (int *i* = 0; *i* <= *top*; *i*++) {  
342 if (*students*[*i*].getId() == id) {  
343 for (int *j* = *i*; *j* < *top*; *j*++) {  
344 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
345 }  
346 *students*[*top*--] = null; // Clear the last position  
347 System.out.println("Student has been successfully removed.");  
348 return;  
349 }  
350 }  
351 System.out.println("No record found for the student ID " + id + ".");  
352 }  
353  
354 @  
355 public static Student searchStudent(int id) { 2 usages  
356 if (id <= 0) {  
357 System.out.println("Invalid input: The ID must be a positive integer.");  
358 return null;  
359 }  
360  
361 for (int *i* = 0; *i* <= *top*; *i*++) {  
362 if (*students*[*i*].getId() == id) {  
363 return *students*[*i*];  
364 }  
365 }  
366 System.out.println("Student with the ID " + id + " could not be located.");  
367 return null;  
368 }  
369  
370 public void sortStudentsQuick() { 1 usage  
371 long startTime = System.nanoTime();  
372 }  
373  
374 public static void editStudent(int id, String name, double marks) { 2 usages  
375 if (id <= 0) {  
376 System.out.println("Error: Student ID must be greater than zero.");  
377 return;  
378 }  
379  
380 for (int *i* = 0; *i* <= *top*; *i*++) {  
381 if (*students*[*i*].getId() == id) {  
382 try {  
383 *students*[*i*] = new Student(id, name, marks);  
384 System.out.println("Student details have been successfully updated.");  
385 } catch (IllegalArgumentException e) {  
386 System.out.println("Failed to update student: " + e.getMessage());  
387 }  
388 return;  
389 }  
390 }  
391 }  
392  
393 public static void deleteStudent(int id) { 2 usages  
394 if (id <= 0) {  
395 System.out.println("Error: The student ID must be a positive number.");  
396 return;  
397 }  
398  
399 for (int *i* = 0; *i* <= *top*; *i*++) {  
400 if (*students*[*i*].getId() == id) {  
401 for (int *j* = *i*; *j* < *top*; *j*++) {  
402 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
403 }  
404 *students*[*top*--] = null; // Clear the last position  
405 System.out.println("Student has been successfully removed.");  
406 return;  
407 }  
408 }  
409 System.out.println("No record found for the student ID " + id + ".");  
410 }  
411  
412 @  
413 public static Student searchStudent(int id) { 2 usages  
414 if (id <= 0) {  
415 System.out.println("Invalid input: The ID must be a positive integer.");  
416 return null;  
417 }  
418  
419 for (int *i* = 0; *i* <= *top*; *i*++) {  
420 if (*students*[*i*].getId() == id) {  
421 return *students*[*i*];  
422 }  
423 }  
424 System.out.println("Student with the ID " + id + " could not be located.");  
425 return null;  
426 }  
427  
428 public void sortStudentsQuick() { 1 usage  
429 long startTime = System.nanoTime();  
430 }  
431  
432 public static void editStudent(int id, String name, double marks) { 2 usages  
433 if (id <= 0) {  
434 System.out.println("Error: Student ID must be greater than zero.");  
435 return;  
436 }  
437  
438 for (int *i* = 0; *i* <= *top*; *i*++) {  
439 if (*students*[*i*].getId() == id) {  
440 try {  
441 *students*[*i*] = new Student(id, name, marks);  
442 System.out.println("Student details have been successfully updated.");  
443 } catch (IllegalArgumentException e) {  
444 System.out.println("Failed to update student: " + e.getMessage());  
445 }  
446 return;  
447 }  
448 }  
449 }  
450  
451 public static void deleteStudent(int id) { 2 usages  
452 if (id <= 0) {  
453 System.out.println("Error: The student ID must be a positive number.");  
454 return;  
455 }  
456  
457 for (int *i* = 0; *i* <= *top*; *i*++) {  
458 if (*students*[*i*].getId() == id) {  
459 for (int *j* = *i*; *j* < *top*; *j*++) {  
460 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
461 }  
462 *students*[*top*--] = null; // Clear the last position  
463 System.out.println("Student has been successfully removed.");  
464 return;  
465 }  
466 }  
467 System.out.println("No record found for the student ID " + id + ".");  
468 }  
469  
470 @  
471 public static Student searchStudent(int id) { 2 usages  
472 if (id <= 0) {  
473 System.out.println("Invalid input: The ID must be a positive integer.");  
474 return null;  
475 }  
476  
477 for (int *i* = 0; *i* <= *top*; *i*++) {  
478 if (*students*[*i*].getId() == id) {  
479 return *students*[*i*];  
480 }  
481 }  
482 System.out.println("Student with the ID " + id + " could not be located.");  
483 return null;  
484 }  
485  
486 public void sortStudentsQuick() { 1 usage  
487 long startTime = System.nanoTime();  
488 }  
489  
490 public static void editStudent(int id, String name, double marks) { 2 usages  
491 if (id <= 0) {  
492 System.out.println("Error: Student ID must be greater than zero.");  
493 return;  
494 }  
495  
496 for (int *i* = 0; *i* <= *top*; *i*++) {  
497 if (*students*[*i*].getId() == id) {  
498 try {  
499 *students*[*i*] = new Student(id, name, marks);  
500 System.out.println("Student details have been successfully updated.");  
501 } catch (IllegalArgumentException e) {  
502 System.out.println("Failed to update student: " + e.getMessage());  
503 }  
504 return;  
505 }  
506 }  
507 }  
508  
509 public static void deleteStudent(int id) { 2 usages  
510 if (id <= 0) {  
511 System.out.println("Error: The student ID must be a positive number.");  
512 return;  
513 }  
514  
515 for (int *i* = 0; *i* <= *top*; *i*++) {  
516 if (*students*[*i*].getId() == id) {  
517 for (int *j* = *i*; *j* < *top*; *j*++) {  
518 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
519 }  
520 *students*[*top*--] = null; // Clear the last position  
521 System.out.println("Student has been successfully removed.");  
522 return;  
523 }  
524 }  
525 System.out.println("No record found for the student ID " + id + ".");  
526 }  
527  
528 @  
529 public static Student searchStudent(int id) { 2 usages  
530 if (id <= 0) {  
531 System.out.println("Invalid input: The ID must be a positive integer.");  
532 return null;  
533 }  
534  
535 for (int *i* = 0; *i* <= *top*; *i*++) {  
536 if (*students*[*i*].getId() == id) {  
537 return *students*[*i*];  
538 }  
539 }  
540 System.out.println("Student with the ID " + id + " could not be located.");  
541 return null;  
542 }  
543  
544 public void sortStudentsQuick() { 1 usage  
545 long startTime = System.nanoTime();  
546 }  
547  
548 public static void editStudent(int id, String name, double marks) { 2 usages  
549 if (id <= 0) {  
550 System.out.println("Error: Student ID must be greater than zero.");  
551 return;  
552 }  
553  
554 for (int *i* = 0; *i* <= *top*; *i*++) {  
555 if (*students*[*i*].getId() == id) {  
556 try {  
557 *students*[*i*] = new Student(id, name, marks);  
558 System.out.println("Student details have been successfully updated.");  
559 } catch (IllegalArgumentException e) {  
560 System.out.println("Failed to update student: " + e.getMessage());  
561 }  
562 return;  
563 }  
564 }  
565 }  
566  
567 public static void deleteStudent(int id) { 2 usages  
568 if (id <= 0) {  
569 System.out.println("Error: The student ID must be a positive number.");  
570 return;  
571 }  
572  
573 for (int *i* = 0; *i* <= *top*; *i*++) {  
574 if (*students*[*i*].getId() == id) {  
575 for (int *j* = *i*; *j* < *top*; *j*++) {  
576 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
577 }  
578 *students*[*top*--] = null; // Clear the last position  
579 System.out.println("Student has been successfully removed.");  
580 return;  
581 }  
582 }  
583 System.out.println("No record found for the student ID " + id + ".");  
584 }  
585  
586 @  
587 public static Student searchStudent(int id) { 2 usages  
588 if (id <= 0) {  
589 System.out.println("Invalid input: The ID must be a positive integer.");  
590 return null;  
591 }  
592  
593 for (int *i* = 0; *i* <= *top*; *i*++) {  
594 if (*students*[*i*].getId() == id) {  
595 return *students*[*i*];  
596 }  
597 }  
598 System.out.println("Student with the ID " + id + " could not be located.");  
599 return null;  
600 }  
601  
602 public void sortStudentsQuick() { 1 usage  
603 long startTime = System.nanoTime();  
604 }  
605  
606 public static void editStudent(int id, String name, double marks) { 2 usages  
607 if (id <= 0) {  
608 System.out.println("Error: Student ID must be greater than zero.");  
609 return;  
610 }  
611  
612 for (int *i* = 0; *i* <= *top*; *i*++) {  
613 if (*students*[*i*].getId() == id) {  
614 try {  
615 *students*[*i*] = new Student(id, name, marks);  
616 System.out.println("Student details have been successfully updated.");  
617 } catch (IllegalArgumentException e) {  
618 System.out.println("Failed to update student: " + e.getMessage());  
619 }  
620 return;  
621 }  
622 }  
623 }  
624  
625 public static void deleteStudent(int id) { 2 usages  
626 if (id <= 0) {  
627 System.out.println("Error: The student ID must be a positive number.");  
628 return;  
629 }  
630  
631 for (int *i* = 0; *i* <= *top*; *i*++) {  
632 if (*students*[*i*].getId() == id) {  
633 for (int *j* = *i*; *j* < *top*; *j*++) {  
634 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
635 }  
636 *students*[*top*--] = null; // Clear the last position  
637 System.out.println("Student has been successfully removed.");  
638 return;  
639 }  
640 }  
641 System.out.println("No record found for the student ID " + id + ".");  
642 }  
643  
644 @  
645 public static Student searchStudent(int id) { 2 usages  
646 if (id <= 0) {  
647 System.out.println("Invalid input: The ID must be a positive integer.");  
648 return null;  
649 }  
650  
651 for (int *i* = 0; *i* <= *top*; *i*++) {  
652 if (*students*[*i*].getId() == id) {  
653 return *students*[*i*];  
654 }  
655 }  
656 System.out.println("Student with the ID " + id + " could not be located.");  
657 return null;  
658 }  
659  
660 public void sortStudentsQuick() { 1 usage  
661 long startTime = System.nanoTime();  
662 }  
663  
664 public static void editStudent(int id, String name, double marks) { 2 usages  
665 if (id <= 0) {  
666 System.out.println("Error: Student ID must be greater than zero.");  
667 return;  
668 }  
669  
670 for (int *i* = 0; *i* <= *top*; *i*++) {  
671 if (*students*[*i*].getId() == id) {  
672 try {  
673 *students*[*i*] = new Student(id, name, marks);  
674 System.out.println("Student details have been successfully updated.");  
675 } catch (IllegalArgumentException e) {  
676 System.out.println("Failed to update student: " + e.getMessage());  
677 }  
678 return;  
679 }  
680 }  
681 }  
682  
683 public static void deleteStudent(int id) { 2 usages  
684 if (id <= 0) {  
685 System.out.println("Error: The student ID must be a positive number.");  
686 return;  
687 }  
688  
689 for (int *i* = 0; *i* <= *top*; *i*++) {  
690 if (*students*[*i*].getId() == id) {  
691 for (int *j* = *i*; *j* < *top*; *j*++) {  
692 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
693 }  
694 *students*[*top*--] = null; // Clear the last position  
695 System.out.println("Student has been successfully removed.");  
696 return;  
697 }  
698 }  
699 System.out.println("No record found for the student ID " + id + ".");  
700 }  
701  
702 @  
703 public static Student searchStudent(int id) { 2 usages  
704 if (id <= 0) {  
705 System.out.println("Invalid input: The ID must be a positive integer.");  
706 return null;  
707 }  
708  
709 for (int *i* = 0; *i* <= *top*; *i*++) {  
710 if (*students*[*i*].getId() == id) {  
711 return *students*[*i*];  
712 }  
713 }  
714 System.out.println("Student with the ID " + id + " could not be located.");  
715 return null;  
716 }  
717  
718 public void sortStudentsQuick() { 1 usage  
719 long startTime = System.nanoTime();  
720 }  
721  
722 public static void editStudent(int id, String name, double marks) { 2 usages  
723 if (id <= 0) {  
724 System.out.println("Error: Student ID must be greater than zero.");  
725 return;  
726 }  
727  
728 for (int *i* = 0; *i* <= *top*; *i*++) {  
729 if (*students*[*i*].getId() == id) {  
730 try {  
731 *students*[*i*] = new Student(id, name, marks);  
732 System.out.println("Student details have been successfully updated.");  
733 } catch (IllegalArgumentException e) {  
734 System.out.println("Failed to update student: " + e.getMessage());  
735 }  
736 return;  
737 }  
738 }  
739 }  
740  
741 public static void deleteStudent(int id) { 2 usages  
742 if (id <= 0) {  
743 System.out.println("Error: The student ID must be a positive number.");  
744 return;  
745 }  
746  
747 for (int *i* = 0; *i* <= *top*; *i*++) {  
748 if (*students*[*i*].getId() == id) {  
749 for (int *j* = *i*; *j* < *top*; *j*++) {  
750 *students*[*j*] = *students*[*j* + 1]; // Shift elements  
751 }  
752 *students*[*top*--] = null; // Clear the last position  
753 System.out.println("Student has been successfully removed.");  
754 return;  
755 }  
756 }  
757 System.out.println("No record found for the student ID " + id + ".");  
758 }  
759  
760 @  
761 public static Student searchStudent(int id) { 2 usages  
762 if (id <= 0) {  
763 System.out.println("Invalid input: The ID must be a positive integer.");  
764 return null;  
765 }  
766  
767 for (int *i* = 0; *i* <= *top*; *i*++) {  
768 if (*students*[*i*].getId() ==



untitled14Version control

Main.javaaddStudent.javaStack.javaStudent.java

3

public class Stack { 6 usages

80 public void sortStudentsQuick() { 1 usage

81 long startTime = System.nanoTime();

82 quickSort(students, low: 0, top);

83 long endTime = System.nanoTime();

84 long durationNanoseconds = (endTime - startTime);

85 System.out.println("Students have been sorted by ID using Quick Sort in " + durationNanoseconds + " nanoseconds.");

86 }

87

88 private void quickSort(Student[] array, int low, int high) { 3 usages

89 if (low < high) {

90 int pi = partition(array, low, high);

91

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

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

94 }

95 }

96

97

98 @ private int partition(Student[] array, int low, int high) { 1 usage

99 int pivot = array[high].getId();

100 int i = (low - 1);

101

102 for (int j = low; j < high; j++) {

103 if (array[j].getId() <= pivot) {

104 i++;

105 swap(array, i, j);

106 }

107 }

108 swap(array, i: i + 1, high);

109 return i + 1;

110 }

111

112 @ private void swap(Student[] array, int i, int j) { 3 usages

113 Student temp = array[i];

114 array[i] = array[j];

115 array[j] = temp;

116 }

117

118 public void sortStudentsBubble() { 1 usage

119 long startTime = System.nanoTime();

120 bubbleSort();

121 long endTime = System.nanoTime();

122 long durationNanoseconds = (endTime - startTime);

123 System.out.println("Students have been sorted by ID using Bubble Sort in " + durationNanoseconds + " nanoseconds.");

124 }

125

126 private void bubbleSort() { 1 usage

127 int n = top + 1;

128 for (int i = 0; i < n - 1; i++) {

129 for (int j = 0; j < n - i - 1; j++) {

130 if (students[j].getId() > students[j + 1].getId()) {

131 swap(students, j, j: j + 1);

132 }

133 }

134 }

135 }

136

137 public void displayStudents() { 1 usage

138 if (top == -1) {

139 System.out.println("The stack is empty. No student records to show.");

140 return;

141 }

142 for (int i = 0; i <= top; i++) {

143 System.out.printf("Student ID: %d | Name: %s | Marks: %.2f | Rank: %s\n",

144 students[i].getId(), students[i].getName(), students[i].getMarks(), students[i].getRank());

145 }

146 }

147 }

148

untitled14 > src > Stack

untitled14Version control

Main.javaaddStudent.javaStack.javaStudent.java

3

public class Stack { 6 usages

110 }

111

112 @ private void swap(Student[] array, int i, int j) { 3 usages

113 Student temp = array[i];

114 array[i] = array[j];

115 array[j] = temp;

116 }

117

118 public void sortStudentsBubble() { 1 usage

119 long startTime = System.nanoTime();

120 bubbleSort();

121 long endTime = System.nanoTime();

122 long durationNanoseconds = (endTime - startTime);

123 System.out.println("Students have been sorted by ID using Bubble Sort in " + durationNanoseconds + " nanoseconds.");

124 }

125

126 private void bubbleSort() { 1 usage

127 int n = top + 1;

128 for (int i = 0; i < n - 1; i++) {

129 for (int j = 0; j < n - i - 1; j++) {

130 if (students[j].getId() > students[j + 1].getId()) {

131 swap(students, j, j: j + 1);

132 }

133 }

134 }

135 }

136

137 public void displayStudents() { 1 usage

138 if (top == -1) {

139 System.out.println("The stack is empty. No student records to show.");

140 return;

141 }

142 for (int i = 0; i <= top; i++) {

143 System.out.printf("Student ID: %d | Name: %s | Marks: %.2f | Rank: %s\n",

144 students[i].getId(), students[i].getName(), students[i].getMarks(), students[i].getRank());

145 }

146 }

147 }

148

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.