SE06204

MANAGEMENT STUDENT PROJECT

Nguyen Duy Thanh - BH00822

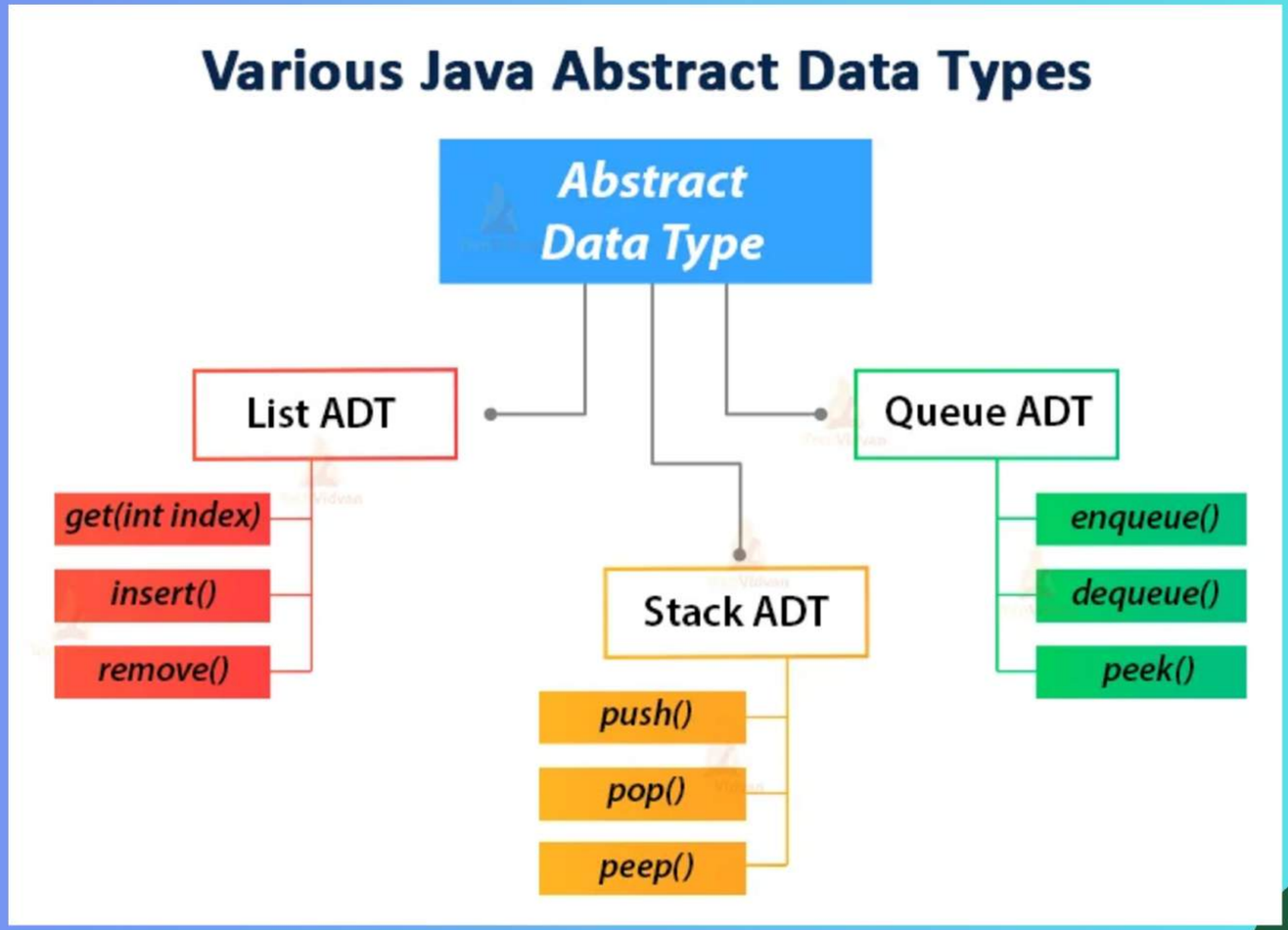
Soft Development ABK

Content

01

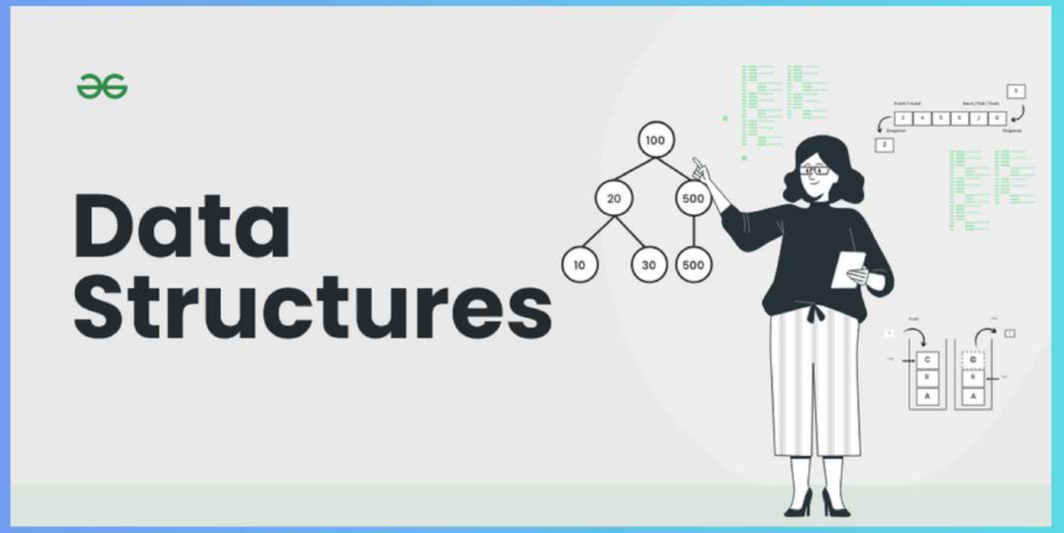
Implementation knowledge

Abstract Data Types (ADTs) represent a way of structuring data in programming that focuses on what operations can be performed on data, rather than how those operations are implemented. The key idea behind ADTs is abstraction—they hide implementation details and focus on the behavior the data structure provides.



01

Implementation knowledge



- Design Specification for Data Structures

• Operations of ADTs: Provide a detailed description of valid operations

that can be performed on the stack (e.g., push, pop, peek) and how these operations apply to your student management application.

• Data Structure Choices: Discuss choices for data structures (e.g., arrays, linked lists) to implement the student management system, highlighting their pros and cons.

01

Implementation knowledge

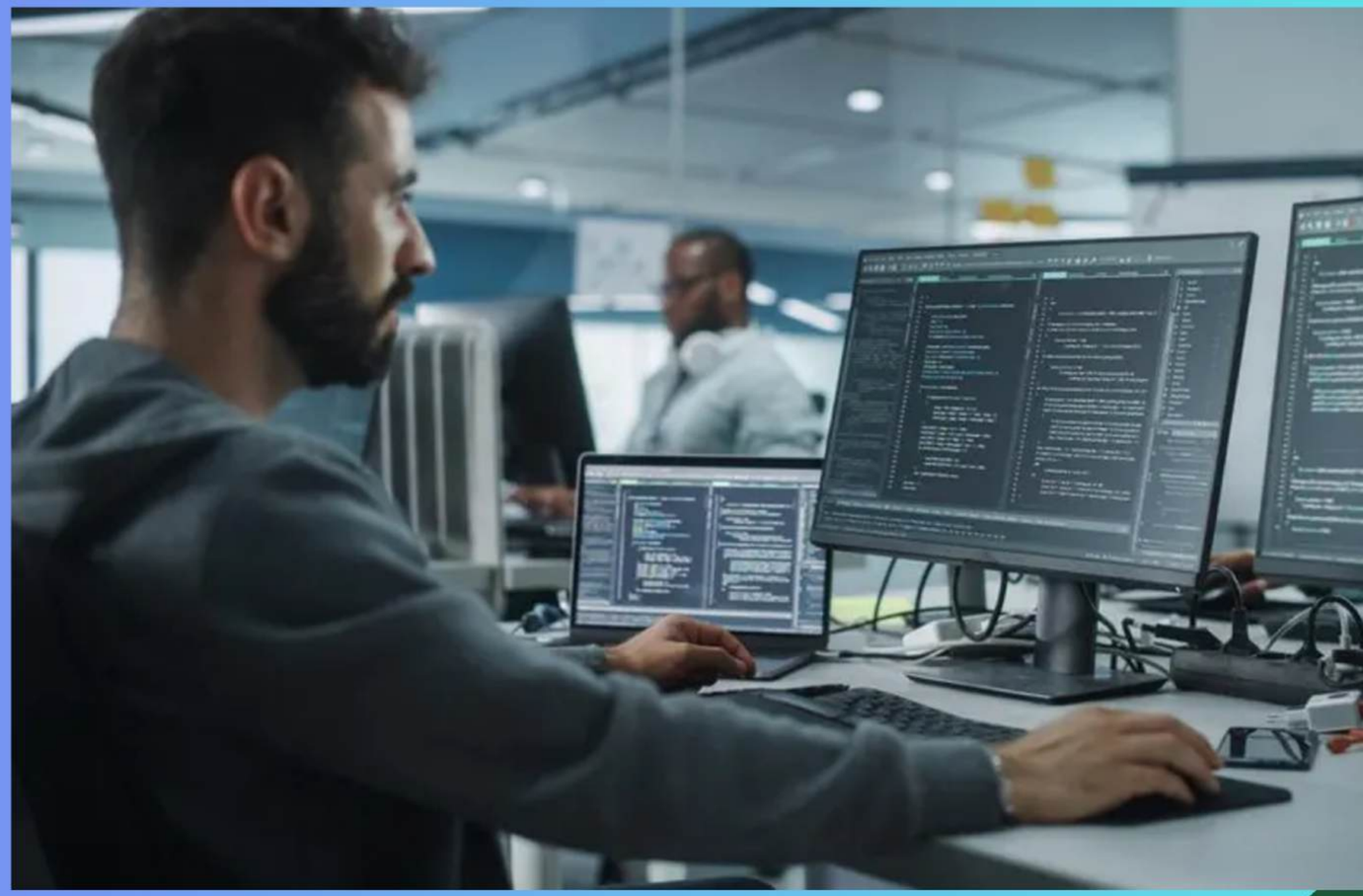
Implementing a Sorting Algorithm

- **Choosing an Algorithm: Choose two sorting algorithms (e.g., Bubble Sort and Quick Sort) and explain their mechanism, performance (time and space complexity), and when each algorithm is best used.**
- **Integrating into an Application: Describe how to sort to rank students based on their scores.**

02

2. Problem Requirements Overview

- **Develop a software application to manage student information, including their ID, name, and marks, while implementing abstract data types (ADTs) to enhance the design, development, and testing processes**



KEY

DATA ENTRY

Allow users to enter the following for multiple students:

- Id of Student
- Name of Student
- Marks of Student

Enable users to specify the number of students to manage

STUDENT RANKING TABLE

Student Ranking Table:

Classify students based on their marks into the following categories:

- [0-5.0): Fail
- [5.0-6.5): Medium
- [6.5-7.5): Good
- [7.5-9.0): Very Good
- [9.0-10.0): Excellent

KEY

OUTPUT DATA

For each student, display:

- Student ID
- Student's full name
- Marks of Student
- Student ranking

CRUD OPERATIONS

Implement functionalities to:

- Add a new student
- Edit existing student information
- Delete a student record
- Sort students by name or marks
- Search for students by ID or name

SOLUTIONS AND TECHNIQUES TO SOLVE

In this project I used ArrayList as solution and applied problem solving technique

USE OF ARRAYLIST IN THE SOLUTION

Dynamic Size

An ArrayList is a dynamic array that can grow and shrink as needed. This is particularly useful for managing student records, as the number of students can vary. Users can add or remove students without needing to define a fixed size beforehand.

Ease of use

ArrayList provides built-in methods for adding, removing, and accessing elements. This simplifies the implementation of CRUD operations:

- Add: Use add() to insert a new student.
- Edit: Access the student by index and modify their details.
- Delete: Remove a student using remove().
- Sort: Utilize methods like Collections.sort() to sort the list based on specific criteria (e.g., marks, names).
- Search: Implement linear search or use Collections.binarySearch() (after sorting) to find students.

Performance

While ArrayList provides average-case constant time complexity for access and amortized constant time for insertions, the performance may vary based on operations. This needs to be considered when designing the application, especially with large datasets.

PROBLEM-SOLVING TECHNIQUES APPLIED

Modular Design

The application is structured in a modular way, separating concerns such as data handling, user interface, and business logic. Each module can be tested independently, improving maintainability and testability.

Encapsulation

Use classes to encapsulate student information, ensuring that related properties (ID, name, marks) are grouped together. This enhances code readability and makes it easier to manage student data.

Data Abtraction

Abstract data types (like ArrayList) are used to hide the complexity of data management. Users interact with simple methods rather than dealing with the underlying data structure directly.

PROBLEM-SOLVING TECHNIQUES APPLIED

Algorithm Selection

FChoosing appropriate algorithms for sorting and searching is crucial. For example, using a simple insertion sort for small datasets or a more efficient sorting algorithm (like quicksort) for larger datasets can greatly affect performance.

Evaluation of Alternatives

As part of your project, propose and evaluate alternative algorithms (e.g., using a LinkedList for frequent insertions/deletions). Compare their performance against the ArrayList in terms of time complexity and memory usage

CODE EXPLANATION

Overview of the Student

Class
Fields:
• public String fullName: Stores the full name of the student.
• public String id: Stores the unique identifier for the student.
• public double mark: Stores the marks obtained by the student. • public String rank: Stores the rank of the student based on their marks.

```java
public class Student  {  37 usages
    public String fullName;  8 usages
    public String id;  12 usages
    public double mark;  18 usages
    public String rank;  11 usages

    public Student(String id, String fullName, double mark){  5 usages
        this.id = id;
        this.fullName = fullName;
        this.mark = mark;
        if(this.mark >= 0 && this.mark <5){
            this.rank = "Fail";
        } else if (this.mark >=5 && this.mark < 6.5) {
            this.rank = "Medium";
        } else if (this.mark >= 6.5 && this.mark < 7.5) {
            this.rank = "Good";
        } else if (this.mark >= 7.5 && this.mark < 9) {
            this.rank = "Very Good";
        } else if(this.mark >= 9 && this.mark <= 10){
            this.rank = "Excellent";
        } else {
            this.rank = null;
        }
    }
```

CODE EXPLANATION

Overview of the Student

Class
Fields:
• public String fullName: Stores the full name of the student.
• public String id: Stores the unique identifier for the student.
• public double mark: Stores the marks obtained by the student. • public String rank: Stores the rank of the student based on their marks.

```java
public class Student  {  37 usages
    public String fullName;  8 usages
    public String id;  12 usages
    public double mark;  18 usages
    public String rank;  11 usages

    public Student(String id, String fullName, double mark){  5 usages
        this.id = id;
        this.fullName = fullName;
        this.mark = mark;
        if(this.mark >= 0 && this.mark <5){
            this.rank = "Fail";
        } else if (this.mark >=5 && this.mark < 6.5) {
            this.rank = "Medium";
        } else if (this.mark >= 6.5 && this.mark < 7.5) {
            this.rank = "Good";
        } else if (this.mark >= 7.5 && this.mark < 9) {
            this.rank = "Very Good";
        } else if(this.mark >= 9 && this.mark <= 10){
            this.rank = "Excellent";
        } else {
            this.rank = null;
        }
    }
```

CODE EXPLANATION

```
60      public static Comparator<Student> IdStudentComparator = new Comparator<Student>() {  1 usage
61          public int compare(Student o1, Student o2) {
62              String idStu1 = o1.getId().toUpperCase();
63              String idStu2 = o2.getId().toUpperCase();
64              return  idStu1.compareTo(idStu2);
65          }
66      };
67      public static Comparator<Student> FullNameStduComparator = new Comparator<Student>() {  1 usage
68          public int compare(Student o1, Student o2) {
69              String fullName1 = o1.getFullName().toUpperCase();
70              String fullName2 = o2.getFullName().toUpperCase();
71              return fullName1.compareTo(fullName2);
72          }
73      };
74
75      public static Comparator<Student> MarkStduComparator = new Comparator<Student>() {  1 usage
76          public int compare(Student o1, Student o2) {
77              double mark1 = o1.getMark();
78              double mark2 = o2.getMark();
79              if(mark1 < mark2){
80                  return -1;
81              } else if (mark2 < mark1) {
82                  return 1;
83              }
84              return 0;
85          }
86      };
```

01 ID Comparator:
Compares students based on their IDs in a case-insensitive manner.

02 Full Name Comparator:
Compares students based on their full names in a case-insensitive manner

03 Marks Comparator:
Compares students based on their marks

CODE EXPLANATION

OVERVIEW OF THE ARRAYLISTADDSTUDENT CLASS

The code snippet you provided appears to be a Java class named ArrayListAddStudent that contains a method addStudent to add student objects to an ArrayList called students.
The key points are:
1.public class ArrayListAddStudent: This declares a public class named ArrayListAddStudent
2. public void addStudent(ArrayList<Student> students, Student objectData): This is a public method named addStudent that takes two parameters:
ArrayList<Student> students: An ArrayList of Student objects. Student objectData: A Student object to be added to the ArrayList.
3.students.add(objectData); This line adds the objectData (a Student object) to the students ArrayList.
The purpose of this class is to provide a way to add Student objects to an ArrayList of Student objects, which could be useful for managing a collection of student data in a software application.

```
4
5   public class ArrayListAddStudent {  2 usages
6   @ >     public void addStudent(ArrayList<Student> students, Student objectData) { students.add(objectData); }
9   }
10
```

CODE EXPLANATION

OVERVIEW OF THE ARRAYLISTEDITSTUDENT CLASS

```
6   public class ArrayListEditStudent {  2 usages
7   @     public void editStudent(ArrayList<Student> students, int position, Student object){  1 usage
8             students.set(position, object);
9         }
10
11  @     public void editStudentById(ArrayList<Student> students, String id, Student data){  1 usage
12            for (int i = 0; i < students.size(); i++){
13                if(Objects.equals(students.get(i).id, id)){
14                    students.set(i,data);
15                }
16            }
17        }
18  }
```

- public void editStudent(ArrayList<Student> students, int position, Student object):
- This method takes three parameters:
- ArrayList<Student> students: An ArrayList of Student objects.
- int position: The index of the student to be edited.
- Student object: The new Student object to replace the existing one.

- It then sets the student at the specified position in the students ArrayList to the new object.
- public void editStudentById(ArrayList<Student> students, String id, Student data):

- This method takes three parameters:
  - ArrayList<Student> students: An ArrayList of Student objects.
  - String id: The ID of the student to be edited.
  - Student data: The new Student object to replace the existing one.

- It then iterates through the students ArrayList and finds the first student whose ID matches the id parameter.
- Once the student is found, it sets the student's data to the new data object.

CODE EXPLANATION

OVERVIEW OF THE ARRAYLISTREMOVESTUDENT CLASS

```java
public class ArrayListRemoveStudent {  2 usages
    public void removeStudentById(ArrayList<Student> students, String id){  1 usage
        for (int i = 0; i < students.size(); i++) {
            if(Objects.equals(students.get(i).id, id)){
                students.remove(i);
            }
        }
    }
}
```

public void removeStudentById(ArrayList<Student> students, String id):

This method takes two parameters:

• ArrayList<Student> students: An ArrayList of Student objects.

• String id: The ID of the student to be removed.

It then iterates through the students ArrayList and finds the first student whose ID matches the id parameter.

Once the student is found, it removes the student from the ArrayList using the remove(i) method, where i is the index of t

student in the ArrayList.

CODE EXPLANATION

OVERVIEW OF THE ARRAYLISTSEARCHSTUDENT CLASS

```java
public class ArrayListSearchStudent {  2 usages
    public int binarySearch(ArrayList<Student> students, String id){  1 usage
        int left = 0;
        int right = students.size() - 1;
        while (left <= right){
            int mid = left + (right - left) / 2;
            if (Objects.equals(students.get(mid).id, id)) {
                return mid;
            }
            int compareStr = students.get(mid).id.compareToIgnoreCase(id);
            if(compareStr < 0){
                left = mid + 1;
            } else if (compareStr > 0){
                right = mid - 1;
            }
        }
        return  -1;
    }
}
```

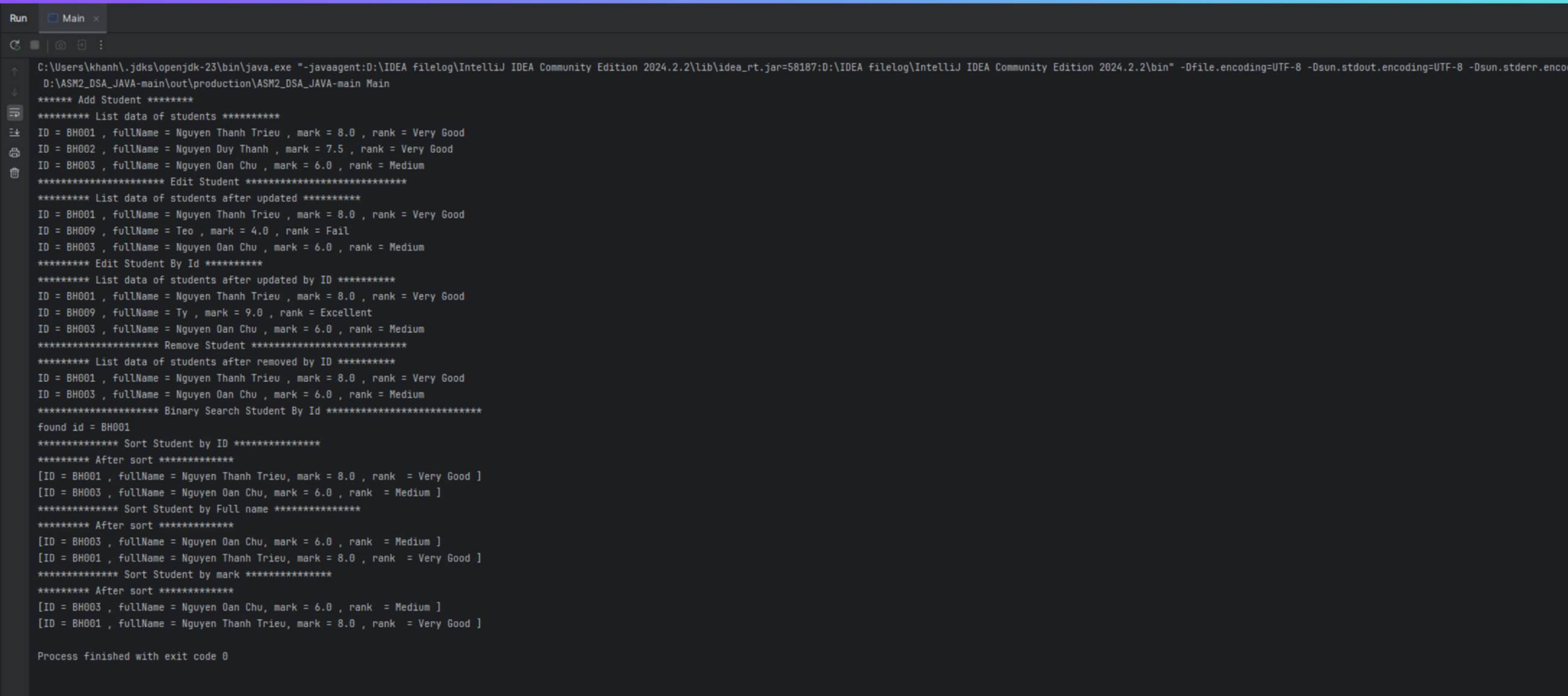
public int binarySearch(ArrayList<Student> students, String id):

- This method takes two parameters:
  - ArrayList<Student> students: An ArrayList of Student objects.
  - String id: The ID of the student to be searched for.
- It uses a binary search algorithm to find the index of the student with the given id in the students ArrayList.
- The algorithm works as follows:
  - It initializes left to 0 and right to the size of the ArrayList minus 1.
  - It then enters a loop that continues as long as left is less than or equal to right.
  - In each iteration, it calculates the middle index mid as (left + right)/2.
  - It then checks if the ID of the student at the mid index matches the id parameter.
    - If it matches, the method returns the mid index.
    - If the ID at mid is less than the id parameter, it updates left to mid + 1.
    - If the ID at mid is greater than the id parameter, it updates right to mid - 1.
  - If the loop completes without finding a match, the method returns -1 to indicate that the student was not found.

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("****** Add Student ********");
        st.addStudent(students, new Student( id: "BH001", fullName: "Nguyen Thanh Trieu",  mark: 8.0));
        st.addStudent(students, new Student( id: "BH002", fullName: "Nguyen Duy Thanh",  mark: 7.5));
        st.addStudent(students, new Student( id: "BH003", fullName: "Nguyen Oan Chu",  mark: 6.0));
        System.out.println("********* List data of students **********");
        for (Student s : students){
            System.out.println("ID = " + s.id +" , fullName = " + s.fullName + " , mark = " + s.mark + " , rank = " + s.rank);
        }
        System.out.println("********************* Edit Student ***************************");
        ArrayListEditStudent edit = new ArrayListEditStudent();
        edit.editStudent(students,  position: 1, new Student( id: "BH009",  fullName: "Teo",  mark: 4));
        System.out.println("********* List data of students after updated **********");
        for (Student s : students){
            System.out.println("ID = " + s.id +" , fullName = " + s.fullName + " , mark = " + s.mark + " , rank = " + s.rank);
        }
        System.out.println("********* Edit Student By Id **********");
        edit.editStudentById(students,  id: "BH009", new Student( id: "BH009",  fullName: "Ty",  mark: 9.0));
        System.out.println("********* List data of students after updated by ID **********");
        for (Student s : students){
            System.out.println("ID = " + s.id +" , fullName = " + s.fullName + " , mark = " + s.mark + " , rank = " + s.rank);
        }
        System.out.println("********************* Remove Student ***************************");
        ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();
        removeSt.removeStudentById(students,  id: "BH009");
        System.out.println("********* List data of students after removed by ID **********");
        for (Student s : students){
            System.out.println("ID = " + s.id +" , fullName = " + s.fullName + " , mark = " + s.mark + " , rank = " + s.rank);
        }
        System.out.println("********************* Binary Search Student By Id ***************************");
        ArrayListSearchStudent searchSt = new ArrayListSearchStudent();
        String numberId = "BH001";
        int findSt = searchSt.binarySearch(students, numberId);
        if(findSt == -1){
            System.out.println("Can not found id = " + numberId);
        } else {
            System.out.println("found id = " + numberId);
        }
        System.out.println("************** Sort Student by ID ***************");
        Collections.sort(students, Student.IdStudentComparator);
        System.out.println("********* After sort *************");
        for (Student str : students){
            System.out.println(str);
        }
        System.out.println("************** Sort Student by Full name ***************");
        Collections.sort(students, Student.FullNameStduComparator);
        System.out.println("********* After sort *************");
        for (Student str : students){
            System.out.println(str);
        }
        System.out.println("************** Sort Student by mark ***************");
        Collections.sort(students, Student.MarkStduComparator);
        System.out.println("********* After sort *************");
        for (Student str : students){
            System.out.println(str);
```

OUTPUT DATA

OUTPUT DATA



Run | Main

```
C:\Users\khanh\.jdks\openjdk-23\bin\java.exe "-javaagent:D:\IDEA filelog\IntelliJ IDEA Community Edition 2024.2.2\lib\idea_rt.jar=58187:D:\IDEA filelog\IntelliJ IDEA Community Edition 2024.2.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.enco
 D:\ASM2_DSA_JAVA-main\out\production\ASM2_DSA_JAVA-main Main
****** Add Student ********
********* List data of students **********
ID = BH001 , fullName = Nguyen Thanh Trieu , mark = 8.0 , rank = Very Good
ID = BH002 , fullName = Nguyen Duy Thanh , mark = 7.5 , rank = Very Good
ID = BH003 , fullName = Nguyen Oan Chu , mark = 6.0 , rank = Medium
********************** Edit Student ****************************
********* List data of students after updated **********
ID = BH001 , fullName = Nguyen Thanh Trieu , mark = 8.0 , rank = Very Good
ID = BH009 , fullName = Teo , mark = 4.0 , rank = Fail
ID = BH003 , fullName = Nguyen Oan Chu , mark = 6.0 , rank = Medium
********* Edit Student By Id **********
********* List data of students after updated by ID **********
ID = BH001 , fullName = Nguyen Thanh Trieu , mark = 8.0 , rank = Very Good
ID = BH009 , fullName = Ty , mark = 9.0 , rank = Excellent
ID = BH003 , fullName = Nguyen Oan Chu , mark = 6.0 , rank = Medium
********************* Remove Student **************************
********* List data of students after removed by ID **********
ID = BH001 , fullName = Nguyen Thanh Trieu , mark = 8.0 , rank = Very Good
ID = BH003 , fullName = Nguyen Oan Chu , mark = 6.0 , rank = Medium
********************* Binary Search Student By Id **************************
found id = BH001
************** Sort Student by ID ***************
********* After sort *************
[ID = BH001 , fullName = Nguyen Thanh Trieu, mark = 8.0 , rank  = Very Good ]
[ID = BH003 , fullName = Nguyen Oan Chu, mark = 6.0 , rank  = Medium ]
************** Sort Student by Full name ***************
********* After sort *************
[ID = BH003 , fullName = Nguyen Oan Chu, mark = 6.0 , rank  = Medium ]
[ID = BH001 , fullName = Nguyen Thanh Trieu, mark = 8.0 , rank  = Very Good ]
************** Sort Student by mark ***************
********* After sort *************
[ID = BH003 , fullName = Nguyen Oan Chu, mark = 6.0 , rank  = Medium ]
[ID = BH001 , fullName = Nguyen Thanh Trieu, mark = 8.0 , rank  = Very Good ]

Process finished with exit code 0
```


GIT ME: TKanH411