Student Management Software

Core Objectives

- Presentation: Explain abstract data types and their benefits in software development.
- Application: Create a student management program using algorithms.
- Algorithm Evaluation: Propose and evaluate alternative algorithms.

Key Requirements

- Abstract Data Types (ADTs): Define and explain their role in design, development, and testing.
- Algorithms: Present algorithms in formal notation and their application in the student management program.
- Student Management: Implement features for adding, editing, deleting, sorting, and searching students.
- Ranking: Calculate student rankings based on a given scoring system.
- Algorithm Evaluation: Compare the effectiveness of different algorithms for the task.

Add student function

```java
package StudentManagement;

import java.util.ArrayList;

public class ArrayListAddStudent {  2 usages
    public void addStudent(ArrayList<Student> students, Student objectData){  3 usages
        students.add(objectData);
    }
}
```

Package Declaration:

“package StudentManagement;”:This line specifies that the code belongs to the StudentManagement package. Packages are used to organize Java code into logical units.

Import Statement:

“import java.util.ArrayList;”: This statement imports the ArrayList class from the java.util package. ArrayList is a dynamic data structure that can store a collection of elements.

Class Definition:

“public class ArrayListAddStudent { }”: This line declares a public class named ArrayListAddStudent. Classes are the fundamental building blocks of object-oriented programming in Java.

Method Definition:

“public void addStudent(ArrayList<Student> students, Student objectData) { students.add(objectData); }”: This line declares a public class named ArrayListAddStudent. Classes are the fundamental building blocks of object-oriented programming in Java.

- This method is named addStudent. It takes two parameters:
  - students: An ArrayList that stores objects of the Student class.
  - objectData: A Student object that represents a single student.
- The method adds the objectData object to the students list using the add() method of the ArrayList class.

Student edit function

```
public class ArrayListEditStudent {  2 usages
    public void editStudent(ArrayList<Student> students, int position, Student object){  1 usage
        students.set(position, object);
    }

    public void editStudentById(ArrayList<Student> students, String id, Student data){  1 usage
        for (int i = 0; i < students.size(); i++){
            if(Objects.equals(students.get(i).id, id)){
                students.set(i,data);
            }
        }
    }
}
```

Class Definition:

“public class ArrayListEditStudent { }”:This line declares a public class named ArrayListEditStudent. Classes are the fundamental building blocks of object-oriented programming in Java.

```java
public class ArrayListEditStudent {  2 usages
    public void editStudent(ArrayList<Student> students, int position, Student object){  1 usage
        students.set(position, object);
    }

    public void editStudentById(ArrayList<Student> students, String id, Student data){  1 usage
        for (int i = 0; i < students.size(); i++){
            if(Objects.equals(students.get(i).id, id)){
                students.set(i,data);
            }
        }
    }
}
```

editStudent(ArrayList<Student> students, int position, Student object):

- This method takes three parameters:
  - students: An ArrayList that stores objects of the Student class.
  - position: An integer representing the index of the student to be edited.
  - object: A Student object containing the updated information.
- The method uses the set() method of the ArrayList class to replace the student at the specified position with the object.

```java
public class ArrayListEditStudent {  2 usages
    public void editStudent(ArrayList<Student> students, int position, Student object){  1 usage
        students.set(position, object);
    }

    public void editStudentById(ArrayList<Student> students, String id, Student data){  1 usage
        for (int i = 0; i < students.size(); i++){
            if(Objects.equals(students.get(i).id, id)){
                students.set(i,data);
            }
        }
    }
}
```

editStudentById(ArrayList<Student> students, String id, Student data):

- This method takes three parameters:
  - students: An ArrayList that stores objects of the Student class.
  - id: A string representing the ID of the student to be edited.
  - data: A Student object containing the updated information.
- The method iterates through the students list using a for loop.
- Inside the loop, it checks if the ID of the current student (students.get(i).id) matches the provided id.
- If a match is found, the method uses the set() method to replace the student at the current index (i) with the data object.

Delete student function

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

- This method is named removeStudentById. It takes two parameters:
  - students: An ArrayList that stores objects of the Student class.
  - id: A string representing the ID of the student to be removed.
- The method iterates through the students list using a for loop.
- Inside the loop, it checks if the ID of the current student (students.get(i).id) matches the provided id.

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

- If a match is found, the method uses the remove() method of the ArrayList class to remove the student at the current index (i).
- Optionally, you can add a break statement after removing the student to exit the loop early, as there's no need to continue searching after the target student is found.

Student search function

```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;
    }
}
```

This method is named binarySearch. It takes two parameters:

- students: An ArrayList that stores objects of the Student class.
- id: A string representing the ID of the student to be searched.

- The method implements the binary search algorithm to efficiently find the student with the given ID in the sorted students list.
- Here's a breakdown of the steps:

Initialize left and right pointers to the beginning and end of the list, respectively.

While left is less than or equal to right:

- Calculate the middle index mid.
- Compare the ID of the student at mid with the target id.
    - If they match, return mid as the index of the found student.
    - If the mid ID is less than the target ID, update left to mid + 1 to search the right half.
    - If the mid ID is greater than the target ID, update right to mid - 1 to search the left half.

If the loop completes without finding the student, return -1 to indicate that the student was not found.

Class Student

Instance Variables:

- public String fullName;: This variable stores the full name of the student.
- public String id;: This variable stores the student's ID.
- public double mark;: This variable stores the student's mark or score.
- public String rank;: This variable stores the student's rank based on their mark.

Constructor:

- This constructor is called when a new Student object is created.
- It takes three parameters: id, fullName, and mark.
- The constructor initializes the instance variables with the provided values.

```
public class Student  {  37 usages
    public String fullName;  8 usages
    public String id;  8 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;
```

```java
//getter and setter java for fullname
public String getFullName() { return fullName; }
public void setFullName(String fullName) { this.fullName = fullName; }
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public double getMark() { return mark; }
public void setMark(double mark) { this.mark = mark; }
```

Code Purpose:

This code defines getter and setter methods for the fullName, id, and mark instance variables within the Student class. These methods provide controlled access to the class's internal state, ensuring data integrity and encapsulation.

Breakdown:

Getter Methods:

- public String getFullName(): Retrieves the current value of the fullName instance variable and returns it as a String.
- public String getId(): Retrieves the current value of the id instance variable and returns it as a String.
- public double getMark(): Retrieves the current value of the mark instance variable and returns it as a double.

```java
//getter and setter java for fullname
public String getFullName() { return fullName; }
public void setFullName(String fullName) { this.fullName = fullName; }
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public double getMark() { return mark; }
public void setMark(double mark) { this.mark = mark; }
```

Setter Methods:

- public void setFullName(String fullName): Takes a String parameter, updates the fullName instance variable with the provided value, and doesn't return anything.
- public void setId(String id): Takes a String parameter, updates the id instance variable with the provided value, and doesn't return anything.
- public void setMark(double mark): Takes a double parameter, updates the mark instance variable with the provided value, and doesn't return anything.

```
*/
public static Comparator<Student> IdStudentComparator = new Comparator<~>() {
    public int compare(Student o1, Student o2) {  no usages
        String idStu1 = o1.getId().toUpperCase();
        String idStu2 = o2.getId().toUpperCase();
        return  idStu1.compareTo(idStu2);
    }
};
```

Code Purpose:

This code defines a static comparator named IdStudentComparator that compares two Student objects based on their ID strings, ignoring case sensitivity. Comparators are used to sort collections of objects in a specific order.

Breakdown:

- **Static Declaration:**
  - public static Comparator<Student> IdStudentComparator = new Comparator<Student>() {
  - This line declares a public static field named IdStudentComparator of type Comparator<Student>. The Comparator interface is used to define custom comparison logic for objects.
  - The anonymous class following new Comparator<Student>() provides the implementation for the compare method.

```
public static Comparator<Student> IdStudentComparator = new Comparator<~>() {
    public int compare(Student o1, Student o2) {  no usages
        String idStu1 = o1.getId().toUpperCase();
        String idStu2 = o2.getId().toUpperCase();
        return  idStu1.compareTo(idStu2);
    }
};
```

compare Method:

- public int compare(Student o1, Student o2)
- This method is the core of the comparator. It takes two Student objects as input and returns an integer value to indicate their relative order.
- The method performs the following steps:
  a. Extracts the ID strings from both Student objects using the getId() method and converts them to uppercase using toUpperCase().
  b. Compares the uppercase ID strings using the compareToIgnoreCase() method. This method returns a negative value if idStu1 is less than idStu2, a positive value if idStu1 is greater than idStu2, and 0 if they are equal.
  c. Returns the result of the comparison, which will be used to sort the Student objects based on their IDs.

```java
public static Comparator<Student> FullNameStduComparator = new Comparator<>() {
    public int compare(Student o1, Student o2) {  no usages
        String fullName1 = o1.getFullName().toUpperCase();
        String fullName2 = o2.getFullName().toUpperCase();
        return fullName1.compareTo(fullName2);
    }
};
```

Code Purpose:

This code defines a static comparator named FullNameStduComparator that compares two Student objects based on their full names, ignoring case sensitivity. Comparators are used to sort collections of objects in a specific order.

Breakdown:

Static Declaration:

public static Comparator<Student> FullNameStduComparator = new Comparator<Student>() {

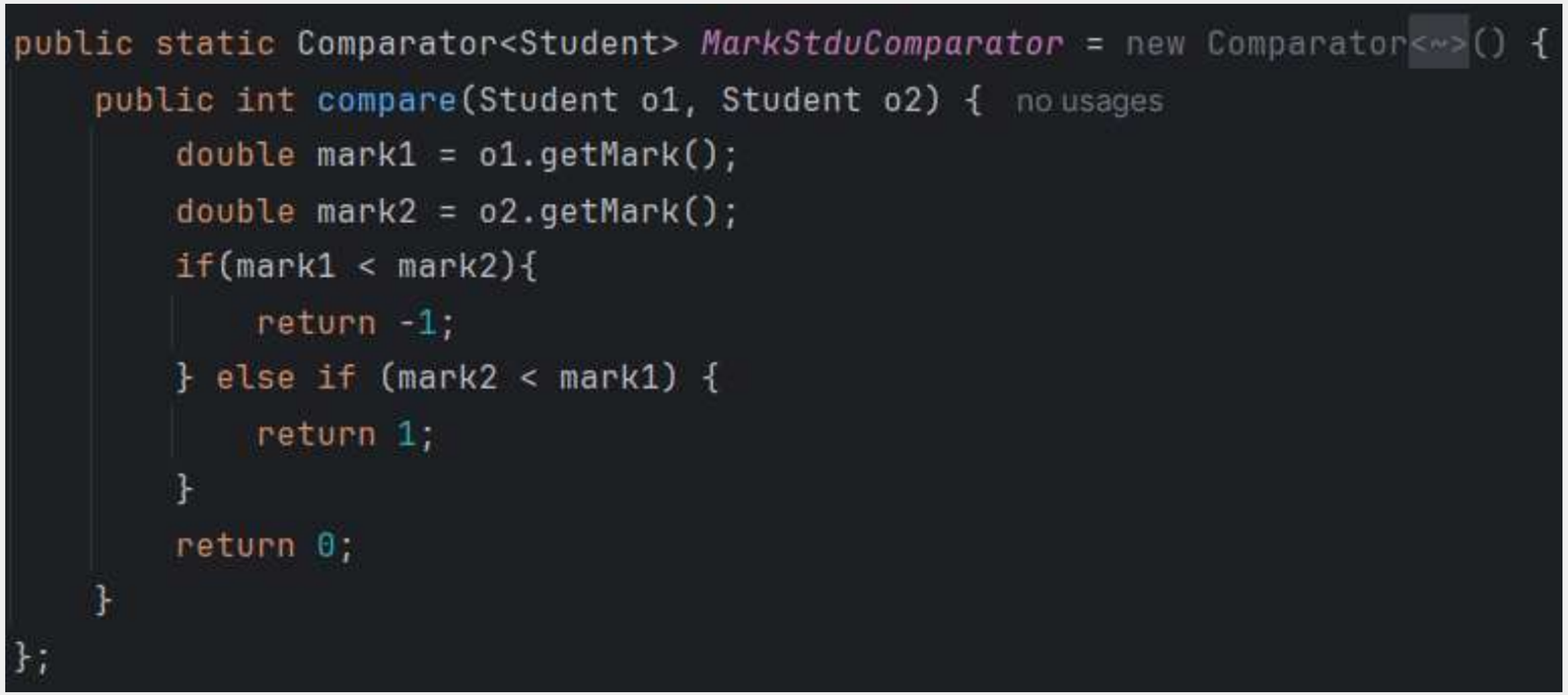
This line declares a public static field named FullNameStduComparator of type Comparator<Student>. The Comparator interface is used to define custom comparison logic for objects.

The anonymous class following new Comparator<Student>() provides the implementation for the compare method.

```
public static Comparator<Student> FullNameStduComparator = new Comparator<~>() {
    public int compare(Student o1, Student o2) {  no usages
        String fullName1 = o1.getFullName().toUpperCase();
        String fullName2 = o2.getFullName().toUpperCase();
        return fullName1.compareTo(fullName2);
    }
};
```

compare Method:

- public int compare(Student o1, Student o2)
- This method is the core of the comparator. It takes two Student objects as input and returns an integer value to indicate their relative order.
- The method performs the following steps:
  a. Extracts the full names from both Student objects using the getFullName() method and converts them to uppercase using toUpperCase().
  b. Compares the uppercase full names using the compareTo() method. This method returns a negative value if fullName1 is less than fullName2, a positive value if fullName1 is greater than fullName2, and 0 if they are equal.
  c. Returns the result of the comparison, which will be used to sort the Student objects based on their full names.



Code Purpose:

This code defines a static comparator named MarkStduComparator that compares two Studen objects based on their marks. Comparators are used to sort collections of objects in a specific order.

```
public static Comparator<Student> MarkStduComparator = new Comparator<~>() {
    public int compare(Student o1, Student o2) {  no usages
        double mark1 = o1.getMark();
        double mark2 = o2.getMark();
        if(mark1 < mark2){
            return -1;
        } else if (mark2 < mark1) {
            return 1;
        }
        return 0;
    }
};
```

Breakdown:

1.Static Declaration:

- public static Comparator<Student> MarkStduComparator = new Comparator<Student>() {
- This line declares a public static field named MarkStduComparator of type Comparator<Student>. The Comparator interface is used to define custom comparison logic for objects.
- The anonymous class following new Comparator<Student>() provides the implementation for the compare method.

```
public static Comparator<Student> MarkStduComparator = new Comparator<~>() {
    public int compare(Student o1, Student o2) {  no usages
        double mark1 = o1.getMark();
        double mark2 = o2.getMark();
        if(mark1 < mark2){
            return -1;
        } else if (mark2 < mark1) {
            return 1;
        }
        return 0;
    }
};
```

compare Method:

- public int compare(Student o1, Student o2)
- This method is the core of the comparator. It takes two Student objects as input and returns an integer value to indicate their relative order.
- The method performs the following steps:
  a. Extracts the marks from both Student objects using the getMark() method.
  b. Compares the marks using simple inequality checks:
    - If mark1 is less than mark2, it returns -1 (indicating o1 should come before o2).
    - If mark2 is less than mark1, it returns 1 (indicating o2 should come before o1).
    - If the marks are equal, it returns 0 (indicating no specific ordering).

```java
@Override  no usages
public String toString() {
    return "[ID = "+ id +" , fullName = " + fullName + ", mark = " + mark + " , rank  = " + rank + " ]";
}
```

Code Purpose:

This code defines a toString() method within the Student class. This method is used to provide a human-readable string representation of a Student object.

Breakdown:

- **Annotation**:
  - @Override
  - This annotation indicates that the toString() method is overriding a method defined in the Object class. Overriding allows you to provide a custom implementation for a method inherited from a superclass.
- **Method Definition**:
  - public String toString()
  - This line declares a public method named toString() that returns a String value.

```java
@Override  no usages
public String toString() {
    return "[ID = "+ id +" , fullName = " + fullName + ", mark = " + mark + " , rank  = " + rank + " ]";
}
```

Return Statement:

- return "[ID = " + id + ", fullName = " + fullName + ", mark = " + mark + ", rank = " + rank + "]";
- This statement constructs a string representation of the Student object by concatenating the values of its instance variables. The string format is enclosed in square brackets and includes the ID, full name, mark, and rank separated by commas.

Main

```java
public class Main {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<~>();
        ArrayListAddStudent st = new ArrayListAddStudent();
        System.out.println("****** Add Student ********");
        st.addStudent(students, new Student( id: "BH001", fullName: "Nguyen Tien Nam",  mark: 8.0));

        st.addStudent(students, new Student( id: "BH002", fullName: "Nguyen Trung Nam",  mark: 7.5));

        st.addStudent(students, new Student( id: "BH003", fullName: "Nguyen Trung An",  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 = "
        }
```

Code Purpose:

This code demonstrates how to create a simple student management system using ArrayList to store student data and various methods to add, display, and potentially perform other operations on the student information.

Breakdown:

- Class Definition:
  - public class Main {
  - This line declares a public class named Main. Classes are the fundamental building blocks of object-oriented programming in Java.
- main Method:
  - public static void main(String[] args) {
  - This is the entry point of the Java application. It's where the program execution begins.
- Creating ArrayLists:
  - ArrayList<Student> students = new ArrayList<>();
  - This line creates an empty ArrayList named students to store Student objects. ArrayList is a dynamic data structure that can efficiently store and manipulate collections of objects.
  - ArrayListAddStudent st = new ArrayListAddStudent();
  - This line creates an instance of the ArrayListAddStudent class, which likely contains methods for adding students to an ArrayList.

- Adding Students:
  - st.addStudent(students, new Student("BH001", "Nguyen Tien Nam", 8.0));
  - This line calls the addStudent method of the st object, passing the students list and a new Student object as arguments. The Student object is created with the specified ID, full name, and mark. This adds the student to the students list.
  - Similar lines add two more students to the list.
- Displaying Students:
  - System.out.println("********* List data of students **********");
  - This line prints a header to the console, indicating that the student data will be displayed.
  - for (Student s : students) {
  - This line starts a for-each loop to iterate over each Student object in the students list.
  - System.out.println("ID = " + s.id + ", fullName = " + s.fullName + ", mark = " + s.mark + ", rank = " + s.rank);
  - This line prints the information of the current Student object, including the ID, full name, mark, and rank. The rank is not calculated in this provided code, but it would likely involve additional logic based on the mark.

```
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);
}
```

Code Purpose:

This code demonstrates how to create a basic student management system using ArrayList to store student data and various methods to edit and remove student information.

Breakdown:

- A**dding Student Data:**
  - The code from the previous response is likely used to add initial student data to the students list.
- E**diting Students:**
  - ArrayListEditStudent edit = new ArrayListEditStudent();
  - This line creates an instance of the ArrayListEditStudent class, which likely contains methods for editing students in an ArrayList.
  - edit.editStudent(students, position: 1, new Student(id: "BH009", fullName: "Teo", mark: 4));
  - This line calls the editStudent method to edit the student at position 1 in the students list with the new information provided.
  - edit.editStudentById(students, id: "BH009", new Student(id: "BH009", fullName: "Ty", mark: 9.0));
  - This line calls the editStudentById method to edit the student with ID "BH009" in the students list with the new information provided.

- **Removing Students:**
  - ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();
  - This line creates an instance of the ArrayListRemoveStudent class, which likely contains methods for removing students from an ArrayList.
  - removeSt.removeStudentById(students, id: "BH009");
  - This line calls the removeStudentById method to remove the student with ID "BH009" from the students list.
- **Displaying Students:**
  - The code uses for loops to iterate over the students list and print the information of each student, including ID, full name, mark, and 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);
}
```

Code Purpose:

This code demonstrates how to perform binary search on a list of Student objects based on their IDs and how to sort the list by ID using a custom comparator.

Breakdown:

- **Binary Search:**
  - ArrayListSearchStudent searchSt = new ArrayListSearchStudent();
  - This line creates an instance of the ArrayListSearchStudent class, which likely contains the binarySearch method for performing binary search.
  - String numberId = "BH001";
  - This line declares a string variable numberId and assigns it the value "BH001", which represents the ID of the student to be searched.
  - int findSt = searchSt.binarySearch(students, numberId);
  - This line calls the binarySearch method of the searchSt object, passing the students list and the numberId as arguments. The method returns the index of the student with the given ID, or -1 if the student is not found.
  - The code then checks if findSt is -1. If it is, the student was not found and a message is printed. Otherwise, the student was found and its ID is printed.

- **Sorting by ID:**
  - Collections.sort(students, Student.IdStudentComparator);
  - This line sorts the students list in ascending order based on the student IDs using the IdStudentComparator.
  - System.out.println("********* After sort **********");
  - This line prints a header to indicate that the students have been sorted.
  - for (Student str : students) {
  - This line starts a for-each loop to iterate over each Student object in the sorted students list.
  - System.out.println(str);
  - This line prints the information of the current Student object. The toString() method of the Student class is likely used to generate a formatted string representation.

```
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);
}
```

Code Purpose:

This code demonstrates how to sort a list of Student objects based on their full names and marks using custom comparators.

Breakdown:

1. Sorting by Full Name:
    - System.out.println("********** Sort Student by Full name **********");
    - This line prints a header to indicate that the students will be sorted by full name.
    - Collections.sort(students, Student.FullNameStduComparator);
    - This line sorts the students list in ascending order based on the student full names using the FullNameStduComparator.
    - System.out.println("********* After sort **********");
    - This line prints a header to indicate that the sorting is complete.
    - for (Student str : students) {
    - System.out.println(str);
    - These lines iterate over the sorted students list and print the information of each student using the toString() method.

- **Sorting by Mark:**
  - System.out.println("********** Sort Student by mark **********");
  - This line prints a header to indicate that the students will be sorted by mark.
  - Collections.sort(students, Student.MarkStduComparator);
  - This line sorts the students list in ascending order based on the student marks using the MarkStduComparator.
  - System.out.println("********* After sort **********");
  - This line prints a header to indicate that the sorting is complete.
  - for (Student str : students) {
  - System.out.println(str);
  - These lines iterate over the sorted students list and print the information of each student using the toString() method.