STUDENT MANAGEMENT SOFTWARE

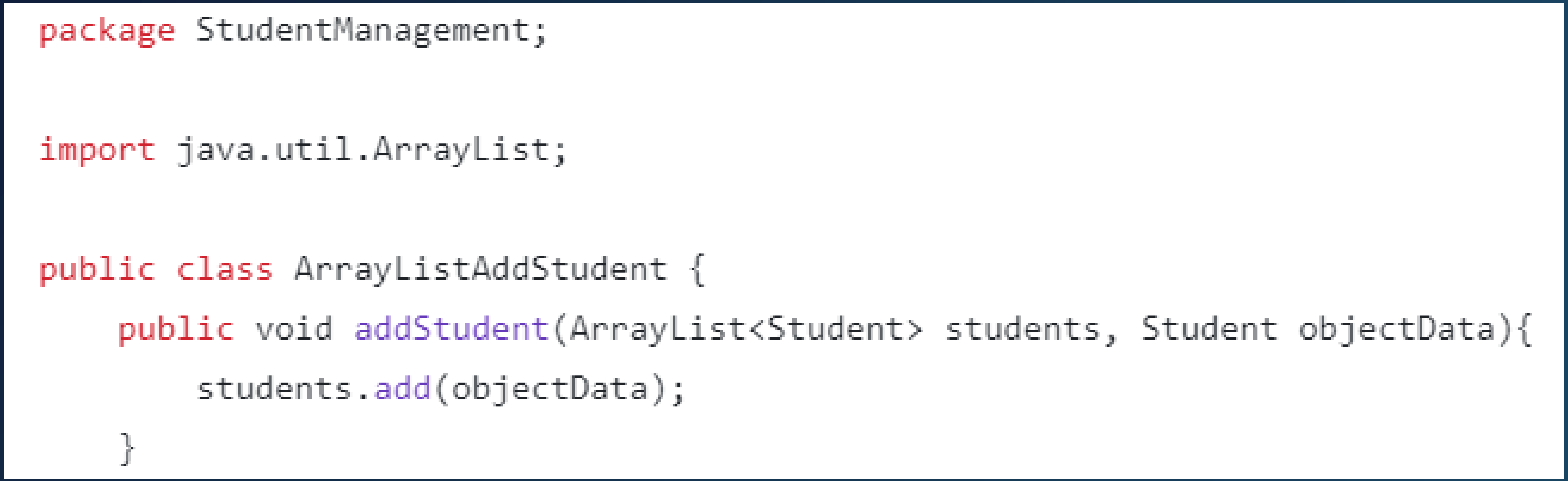
CORE OBJECTIVES

- **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 {
    public void addStudent(ArrayList<Student> students, Student objectData){
        students.add(objectData);
    }
```

- **package StudentManagement;**

This line declares that the class belongs to the StudentManagement package. Packages are used to organize classes and prevent naming conflicts.

- **import java.util.ArrayList;**

This line imports the ArrayList class from the java.util package. ArrayList is a dynamic array that can grow or shrink as needed. This import makes it so you don't have to use the full java.util.ArrayList every time you refer to it.

- **public class ArrayListAddStudent { ... }**

This line declares a public class named ArrayListAddStudent. The public keyword means this class is accessible from other classes, even those in different packages. The code within the curly braces {} defines the class's members.

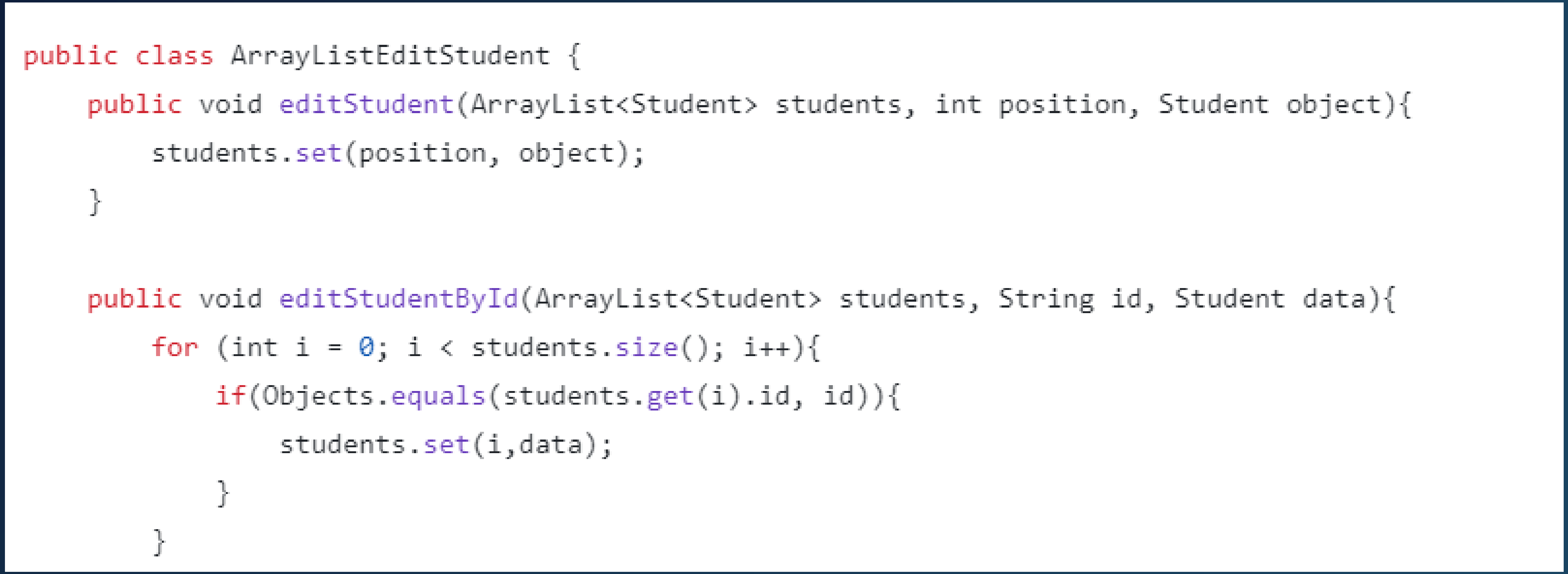
- **public void addStudent(ArrayList<Student> students, Student objectData) { ... }**

This declares a public method named addStudent. Let's analyze its parts:
public: The method is accessible from other classes.
void: The method doesn't return any value.
ArrayList<Student> students: This is the first parameter. It's an ArrayList that holds objects of the Student class (presumably a custom class defining student attributes). This parameter represents the list of students to which we'll add a new student.
Student objectData: This is the second parameter. It represents the Student object to be added to the list.

- **students.add(objectData);**

This line is inside the addStudent method. It uses the add() method of the ArrayList to append the objectData (the new student) to the end of the students list.

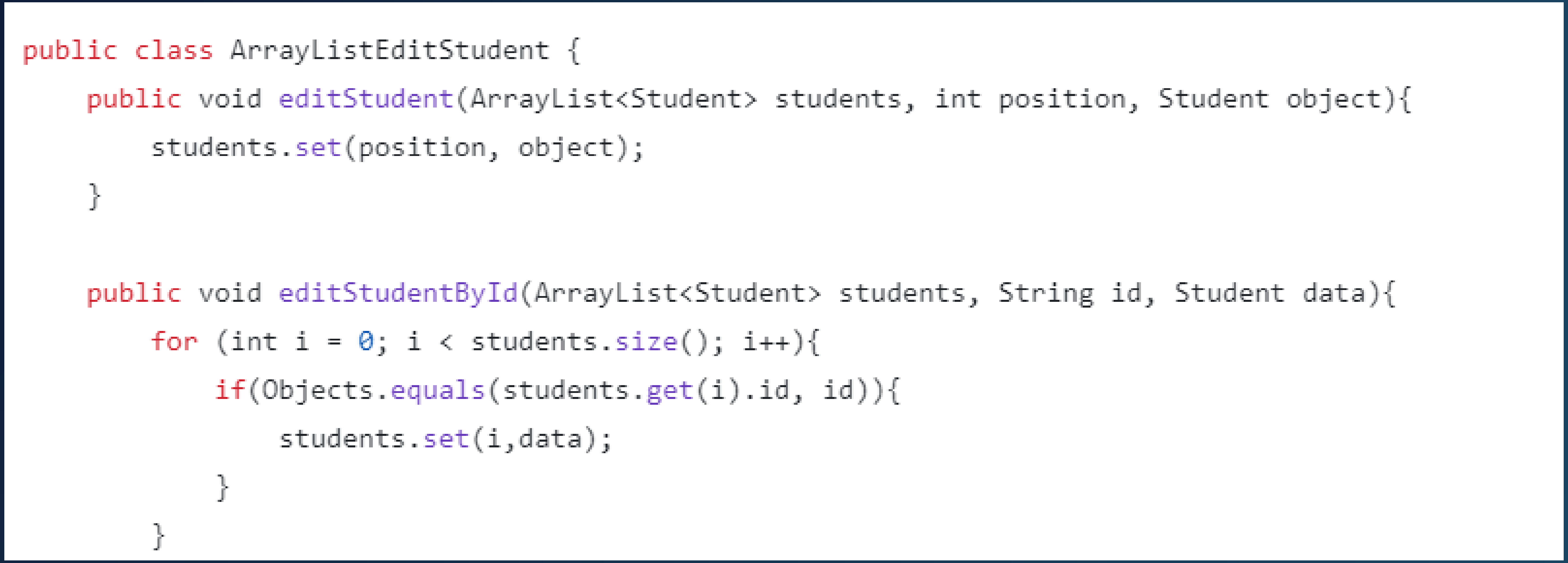
Student edit function



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

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

Class declaration:

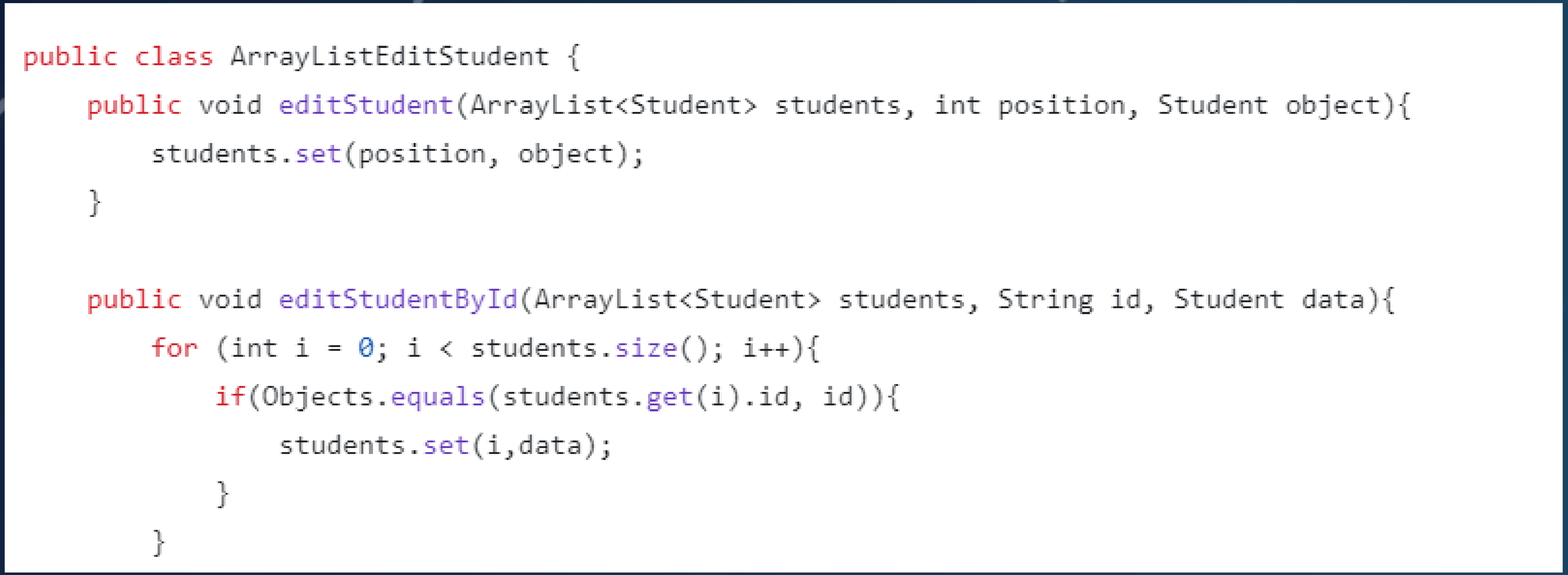
- public class **ArrayListEditStudent**: Declare a public class named ArrayListEditStudent. This class will contain methods to perform modifications to the student list.



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

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

editStudent method:

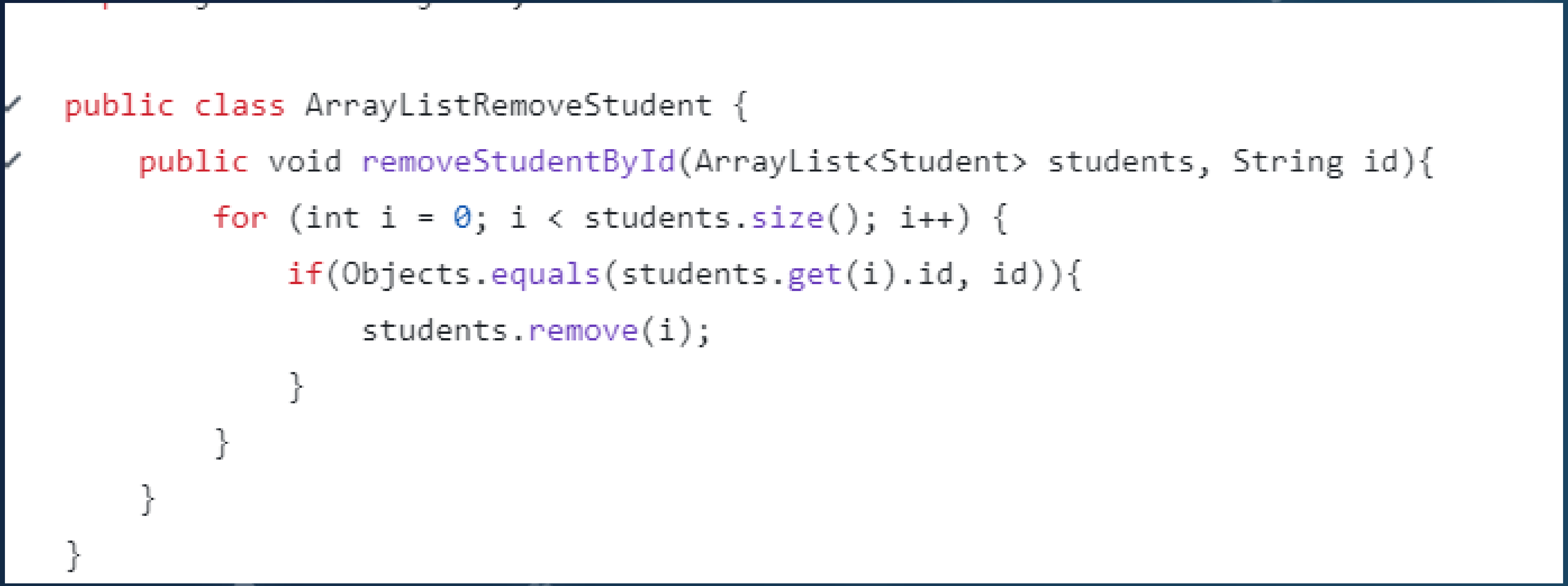
- **public void editStudent:** This is a public method that does not return a value (void).
- **ArrayList<Student> students:** The first parameter is an ArrayList containing Student objects. This is the list of students that we want to modify.
- **int position:** The second parameter is the position in the list that we want to change.
- **Student object:** The last parameter is the new Student object that we want to assign to the given position.
- **students.set(position, object):** This line replaces the object at position in the students list with the new object.


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

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

EditStudentById method:

- **public void editStudentById:** Similar to the editStudent method, this is also a public method that does not return a value.
- **ArrayList<Student> students:** Also an ArrayList containing Student objects.
- **String id:** This parameter is a string representing the ID of the student we want to find and modify.
- **Student data:** The new Student object that we want to assign to the student with the matching ID.
- **for (int i = 0; i < students.size(); i++):** This loop iterates through each element in the students list.
- **if (Objects.equals(students.get(i).id, id)):** Checks if the current student ID is equal to the required ID. If so, then modify.
- **students.set(i, data):** Replaces the current student object with the new data object.

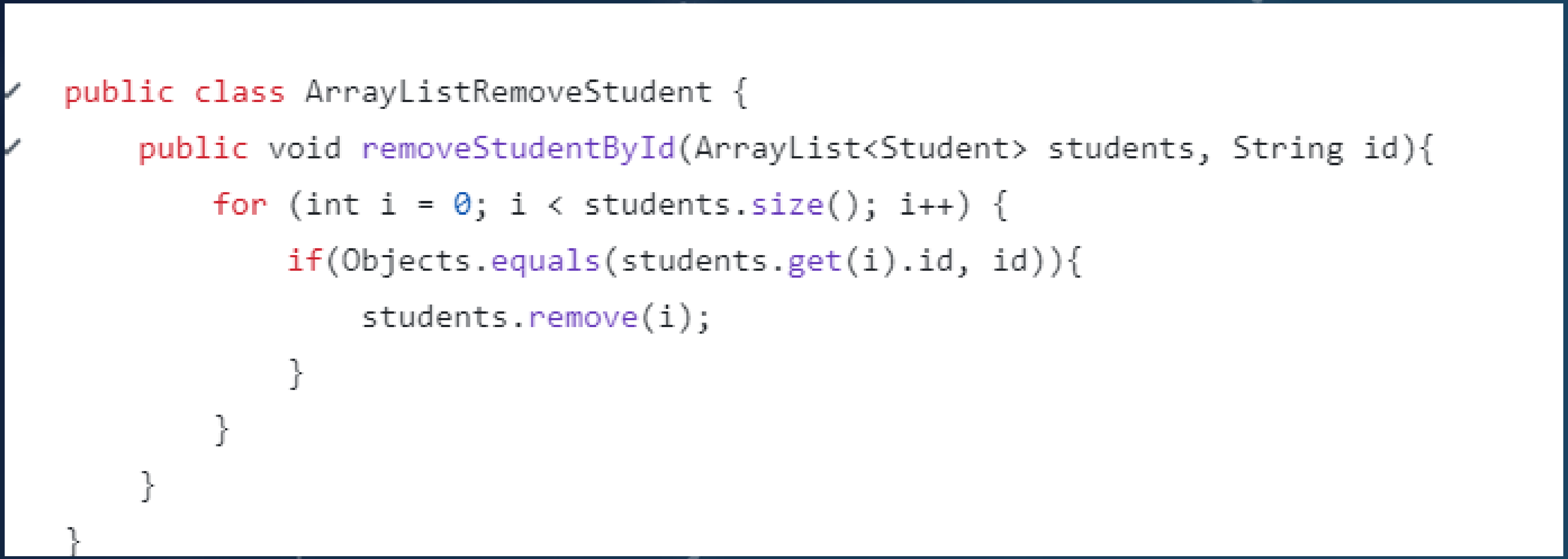
Delete student function



```
public class ArrayListRemoveStudent {
    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);
            }
        }
    }
}
```

Class declaration:

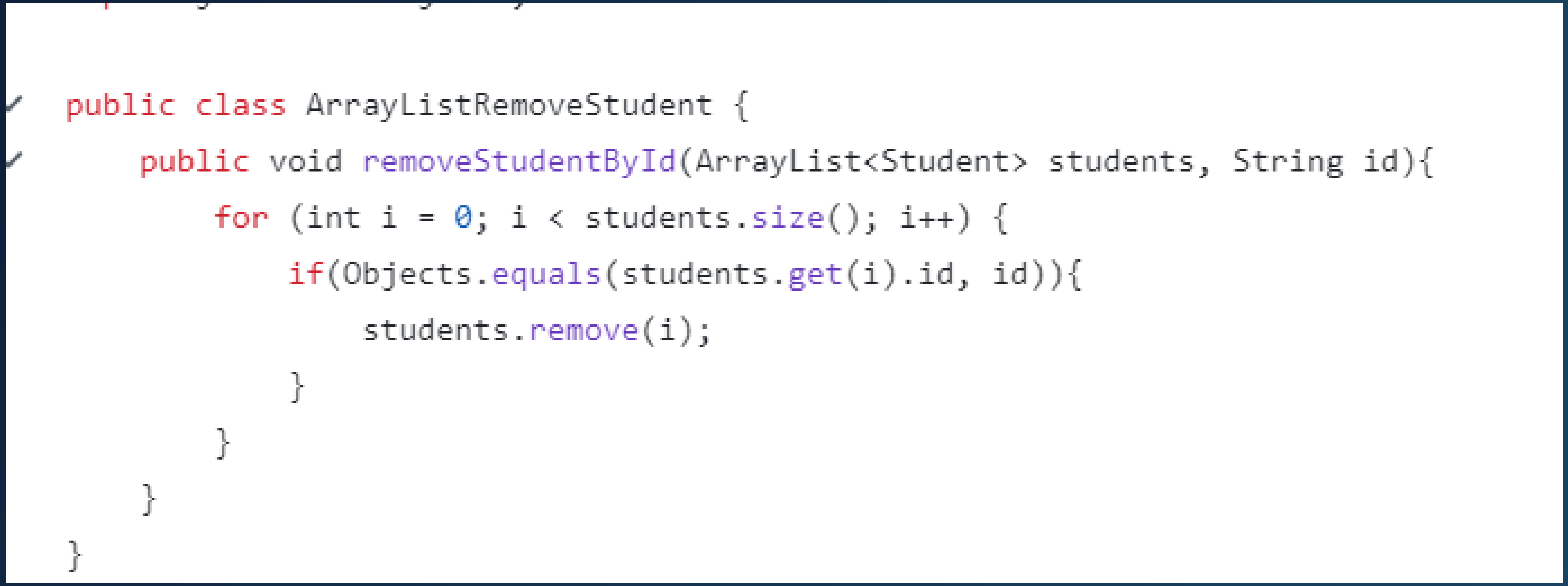
- public class ArrayListRemoveStudent: Similar to the previous code, we declare a public class named ArrayListRemoveStudent. This class will contain the method to delete the student.



```
public class ArrayListRemoveStudent {
    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);
            }
        }
    }
}
```

removeStudentById method:

- **public void removeStudentById:** This is a public method that does not return a value.
- **ArrayList<Student> students:** This parameter is a list of Student objects that we want to delete.
- **String id:** This parameter is the ID of the student that we want to delete.

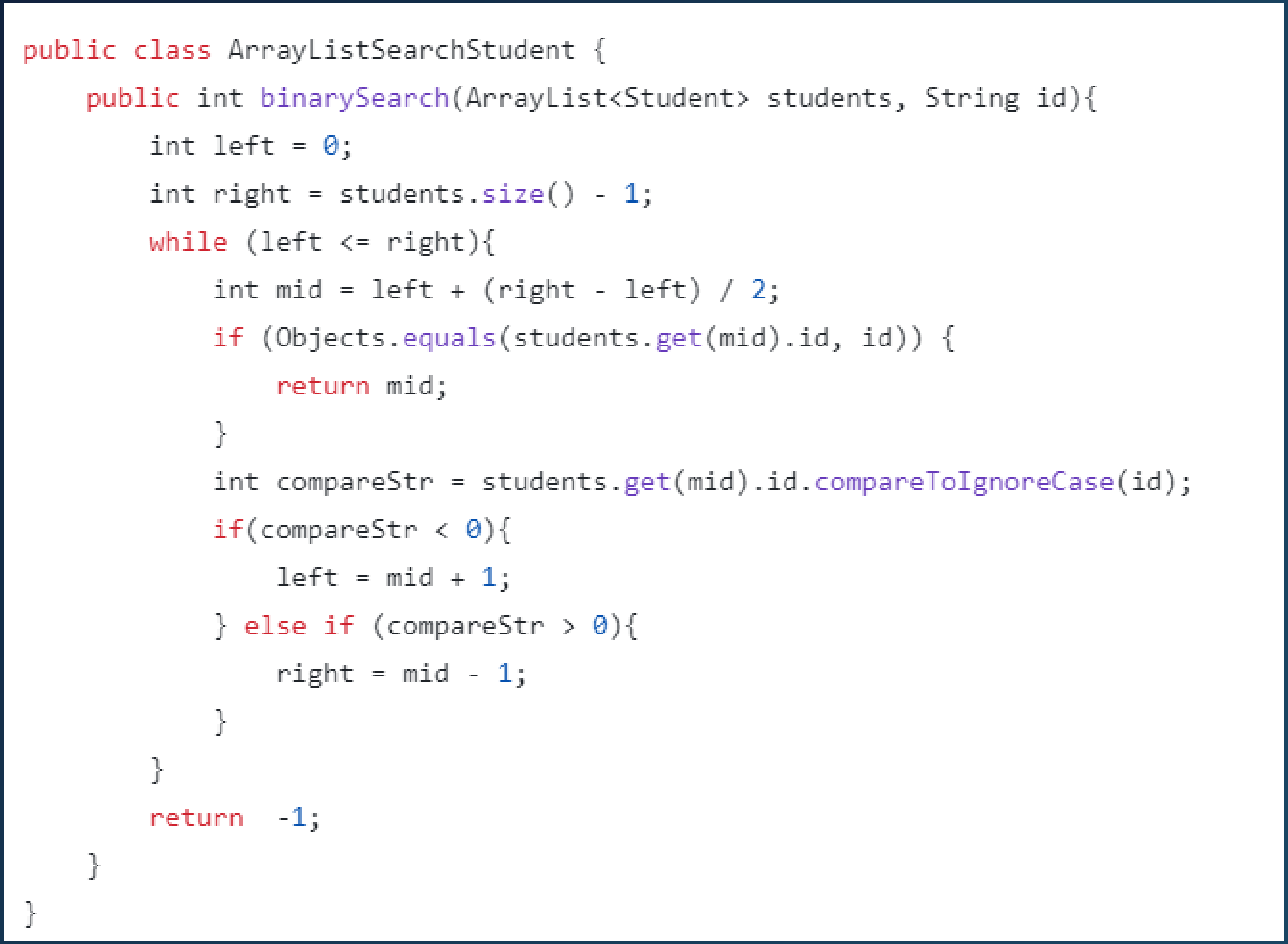


```
public class ArrayListRemoveStudent {
    public void removeStudentById(ArrayList<Student> students, String id){
        for (int i = 0; i < students.size(); i++) {
            if(Objects.equals(students.get(i).id, id)){
                students.remove(i);
            }
        }
    }
}
```

Loops and conditions:

- for (int i = 0; i < students.size(); i++): This loop iterates through each element in the students list.
- if (Objects.equals(students.get(i).id, id)): Checks if the current student ID is equal to the required ID. If so, deletes it.
- students.remove(i): Deletes the element at position i in the list.

Student search function



```java
public class ArrayListSearchStudent {
    public int binarySearch(ArrayList<Student> students, String id){
        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;
    }
}
```

Class declaration:

```java
public class ArrayListSearchStudent {
```

- Declare a public class named ArrayListSearchStudent to contain methods related to searching for students.

binarySearch method:

```java
public int binarySearch(ArrayList<Student> students, String id){
```

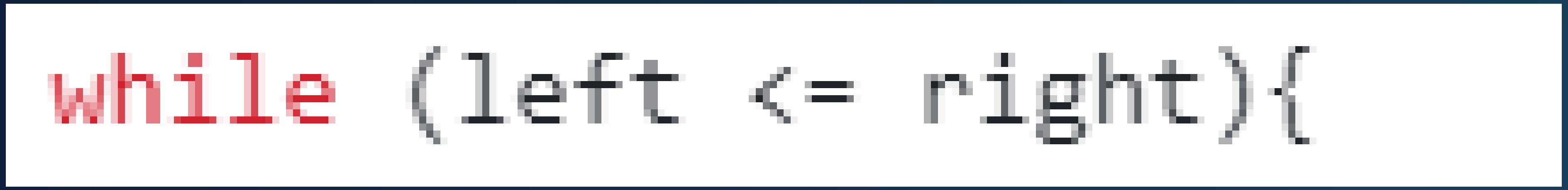
- This method takes two parameters:
  - **students**: A list of **Student** objects sorted by ID.
  - **id**: The ID of the student to find.
- The method returns the index of the student found in the list, or -1 if not found.

Initialize variables:

```java
int left = 0;

int right = students.size() - 1;
```

- **left**: Index of the first element in the search range.
- **right**: Index of the last element in the search range.

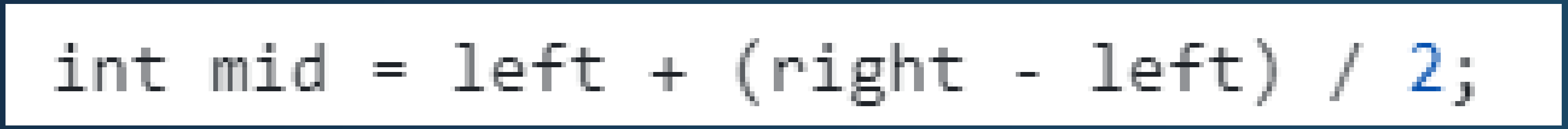
While loop:



```
while (left <= right){
```

- **This loop will continue until the remaining search range is empty (left > right).**

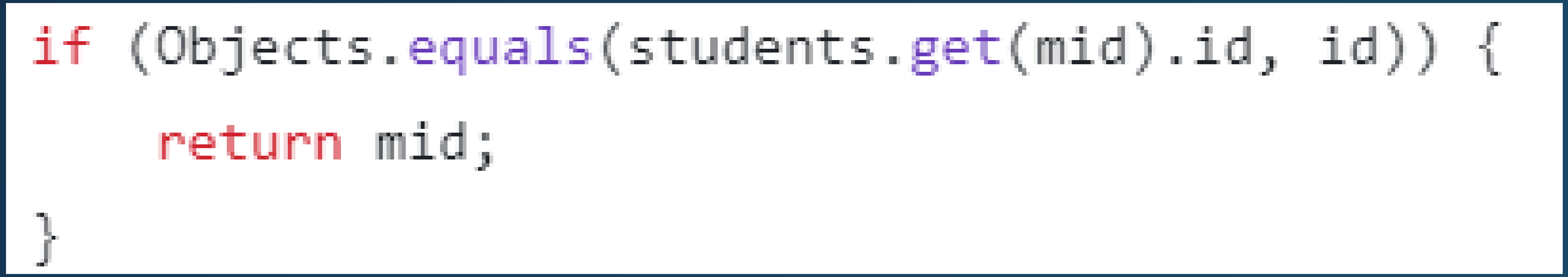
Calculate the average value:



```
int mid = left + (right - left) / 2;
```

- Calculates the index of the element in the middle of the current search range.

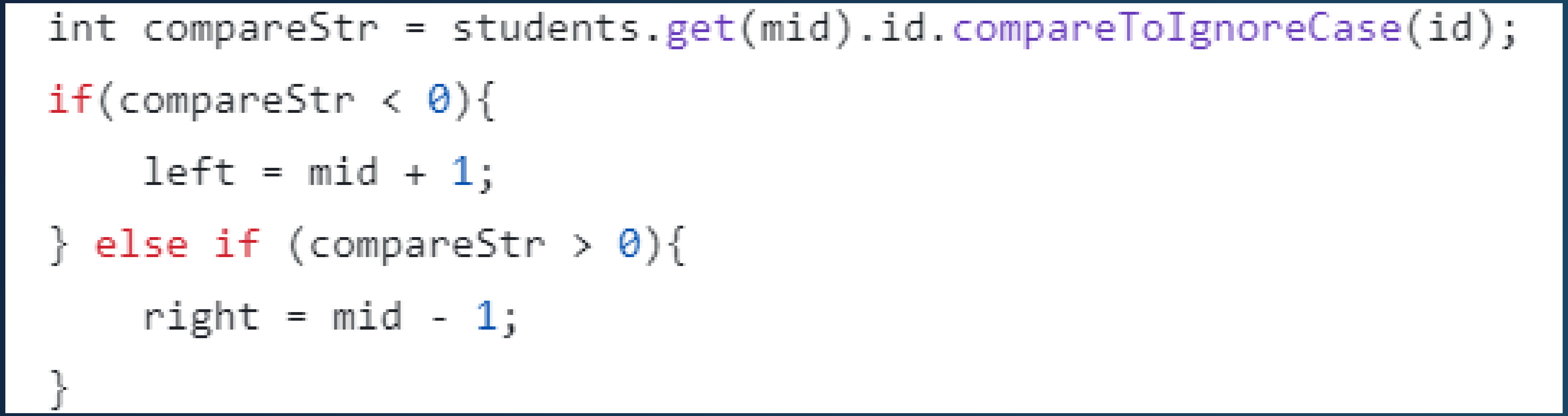
Compare and return results:



```
if (Objects.equals(students.get(mid).id, id)) {
    return mid;
}
```

- If the student ID in the middle position is equal to the ID to be found, return that index.

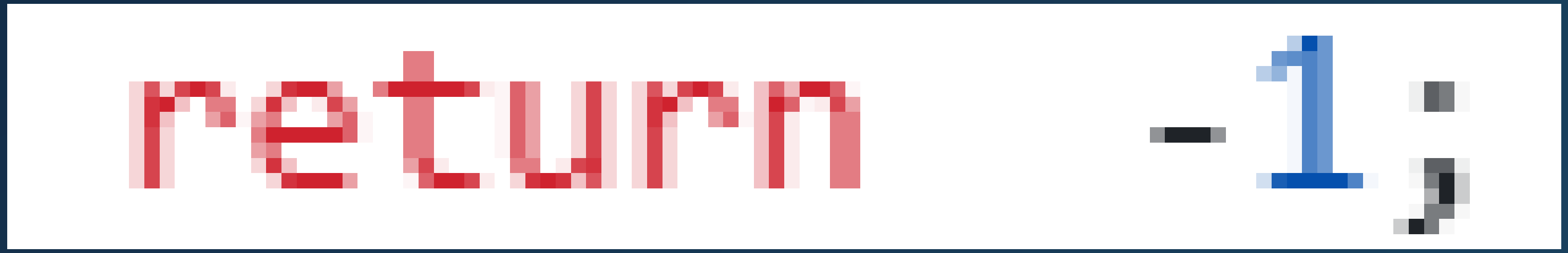
Adjust search range:



```
int compareStr = students.get(mid).id.compareToIgnoreCase(id);
if(compareStr < 0){
    left = mid + 1;
} else if (compareStr > 0){
    right = mid - 1;
}
```

- Compare the ID of the middle student with the ID you want to find.
- If the ID of the middle student is smaller than the ID you want to find, narrow the search range to the right half of the list.
- If the ID of the middle student is larger than the ID you want to find, narrow the search range to the left half of the list.

If not found:

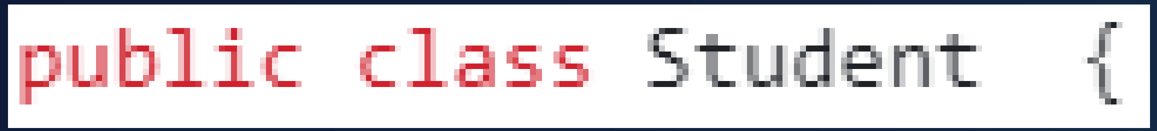


```
return -1;
```

- If the loop ends without finding any students, return -1.

Class Student :

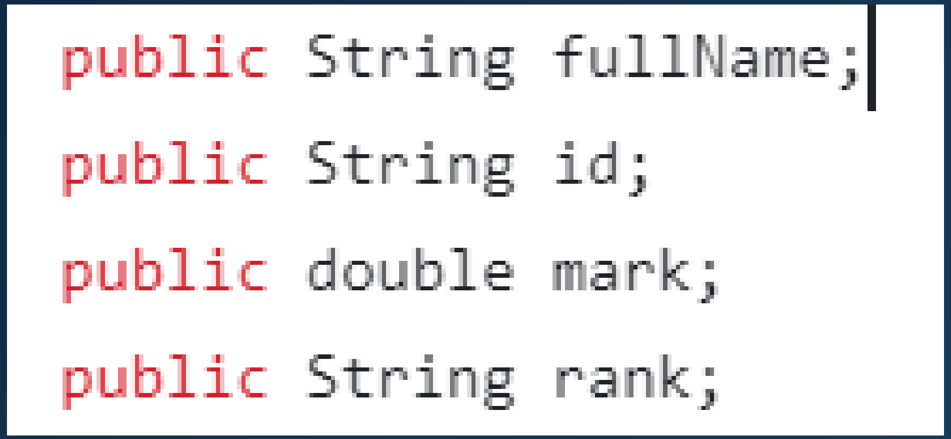
- **Class declaration:**



```java
public class Student  {
```

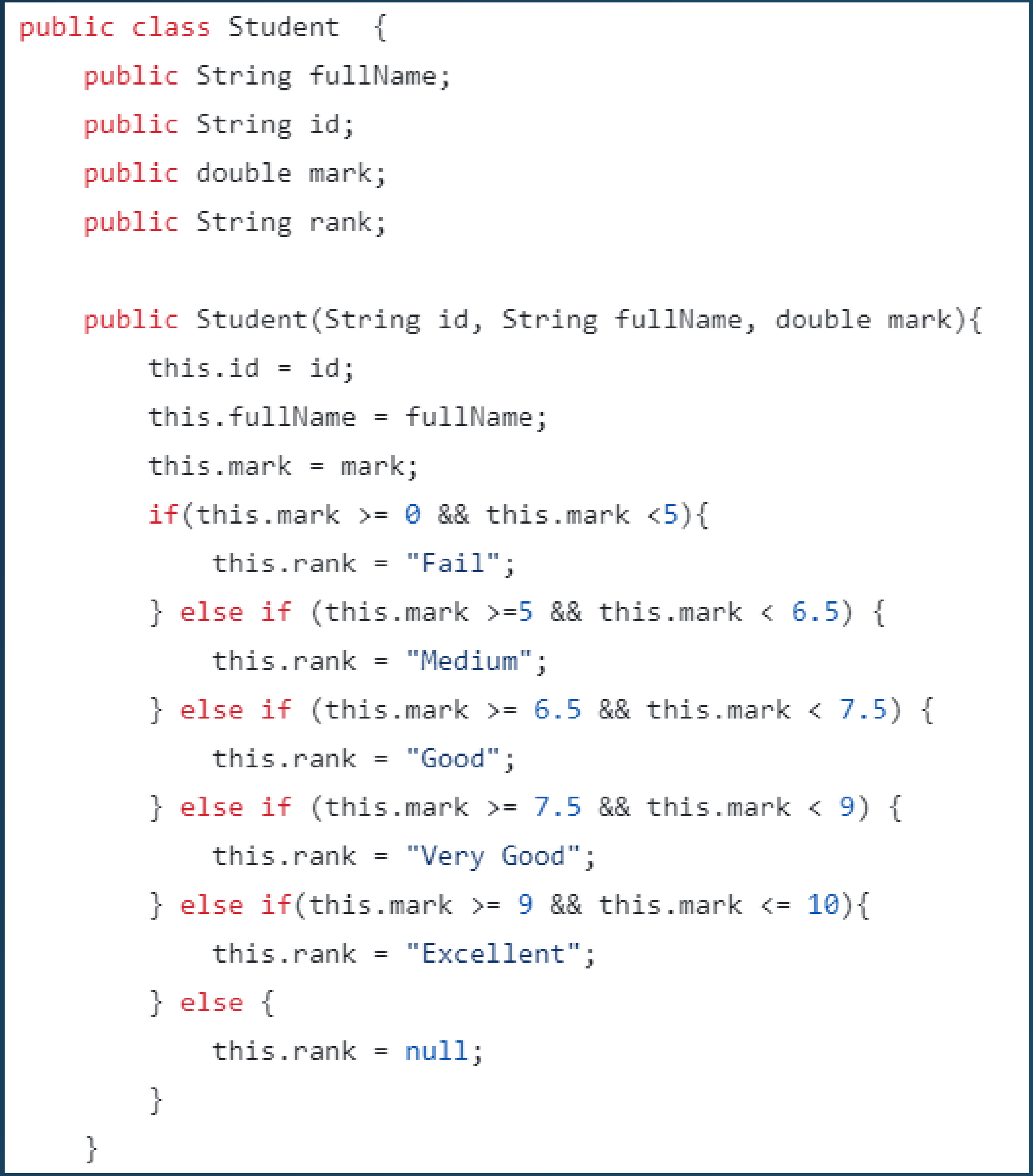
  - This line declares a public class named Student. A public class means that the class can be accessed from anywhere in the program.

- **Property declaration:**



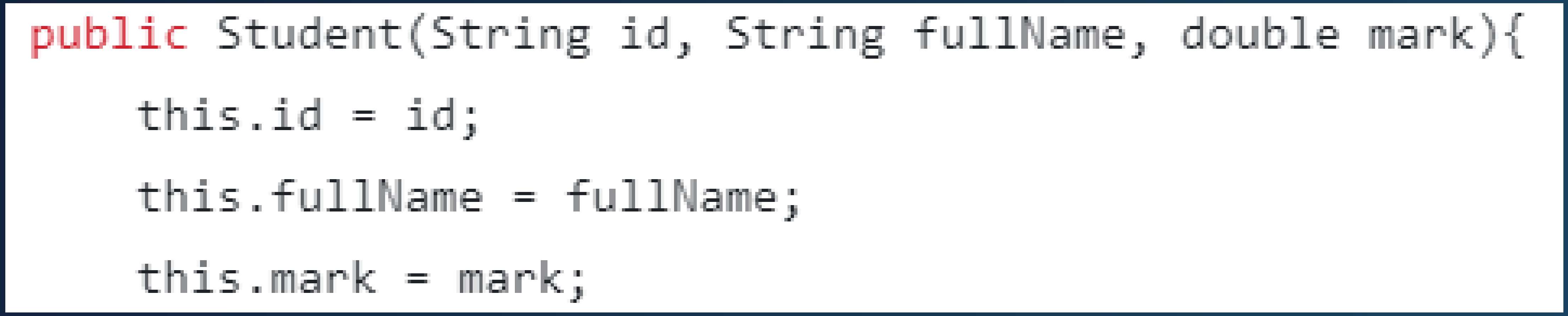
```java
public String fullName;
public String id;
public double mark;
public String rank;
```

- The **Student** class has four properties:
  - **fullName**: String containing the full name of the student.
  - **id**: String containing the student's ID number.
  - **mark**: Double containing the student's score.
  - **rank**: String containing the student's ranking based on the score.



```java
public class Student  {
    public String fullName;
    public String id;
    public double mark;
    public String rank;

    public Student(String id, String fullName, double mark){
        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;
        }
    }
```

Constructor:



```
public Student(String id, String fullName, double mark){
    this.id = id;
    this.fullName = fullName;
    this.mark = mark;
```

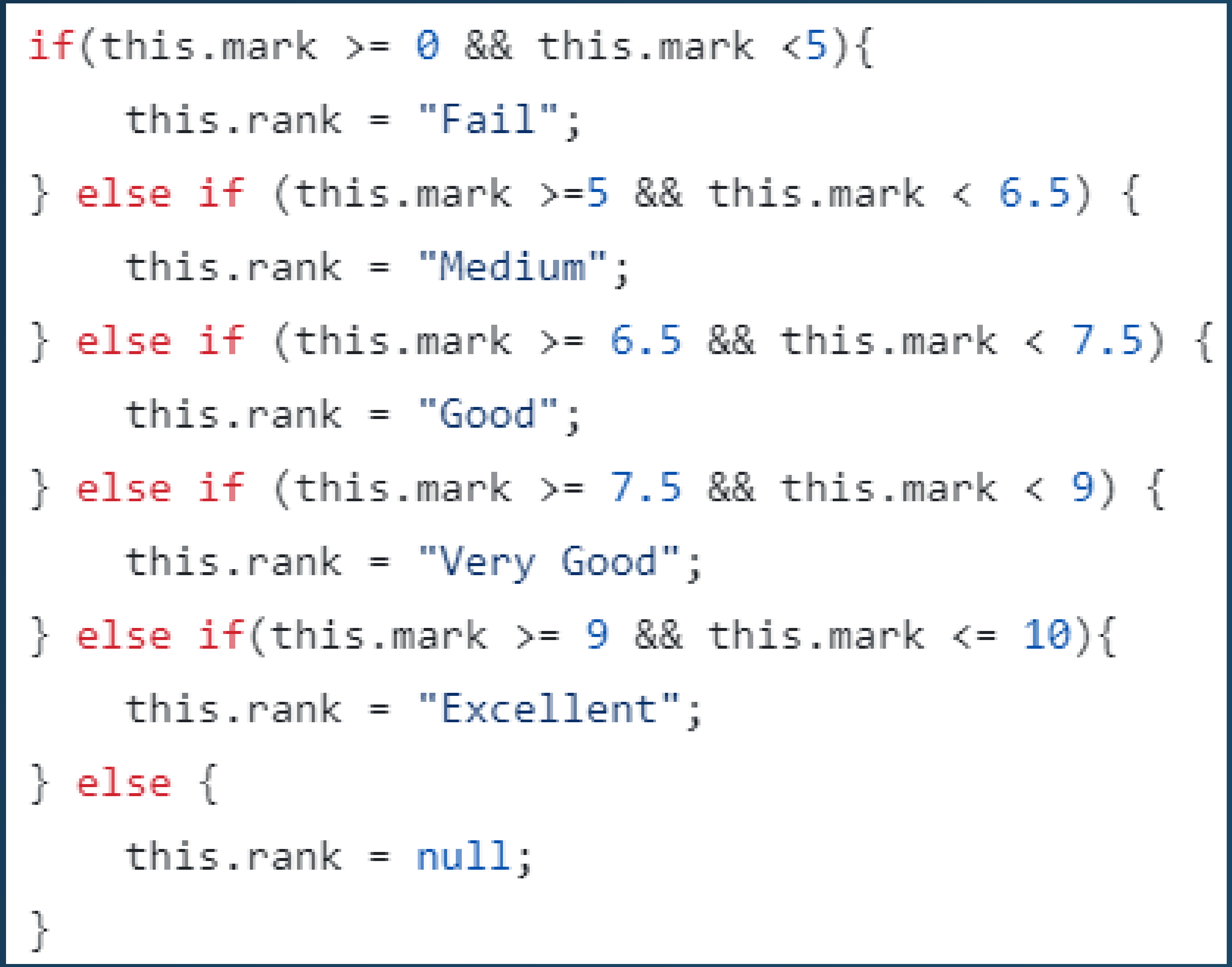
- This constructor is called when a new object of the Student class is created. It takes three parameters: student ID, full name, and score. These parameters are assigned to the corresponding properties of the object.

Classification:

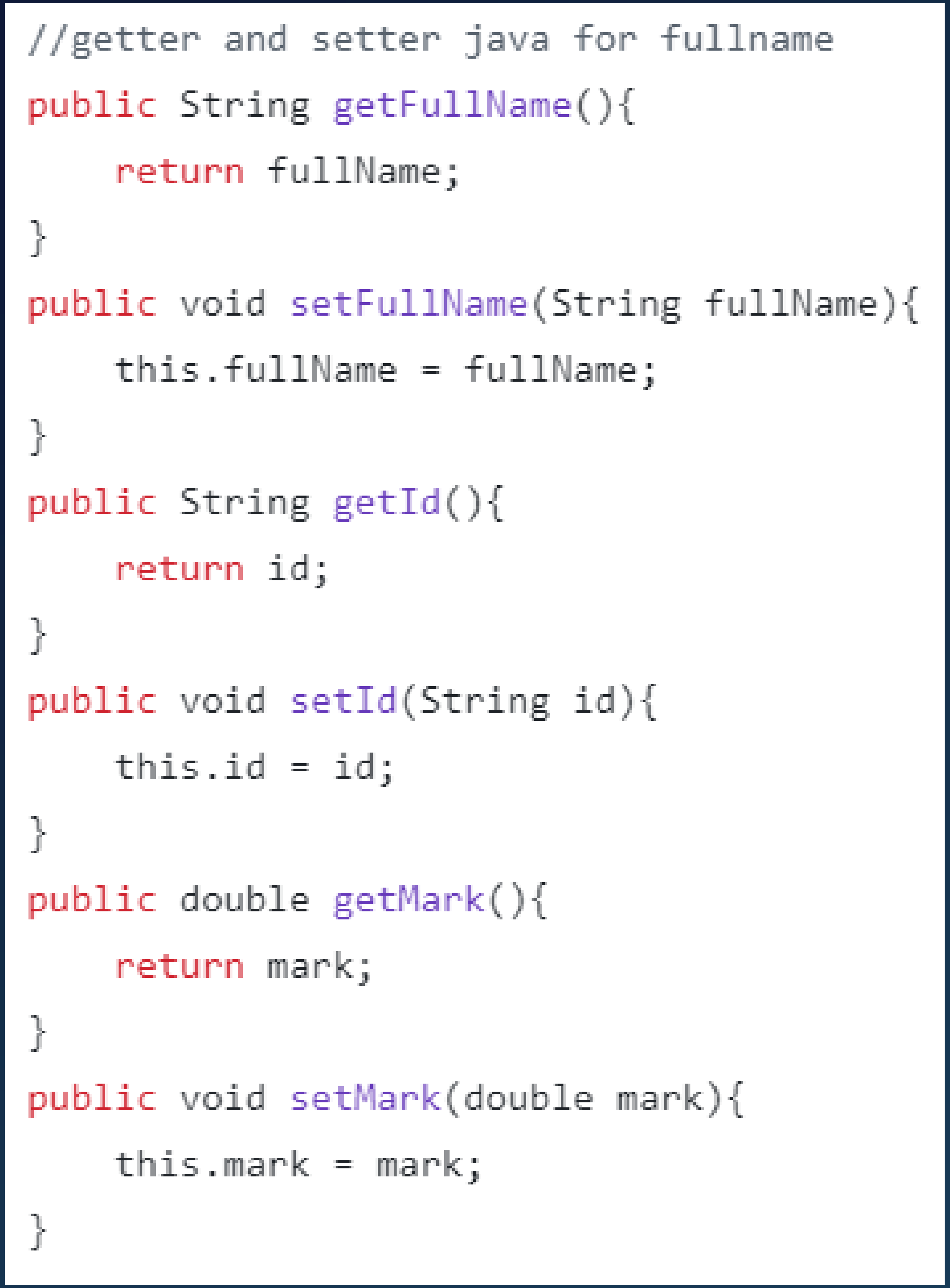
- The code inside the constructor performs the calculation and assigning of grades to students based on their scores. The logic for calculating grades is implemented through **if-else** statements.

Explain the logic of the ranking:

- Condition: Each condition checks whether the score falls within a certain rating range.
- Assigning a value: If the condition is true, the corresponding rating is assigned to the **rank** attribute.
- Handling special cases: If the score does not fall within any of the specified ranges, the rating is assigned **null.**



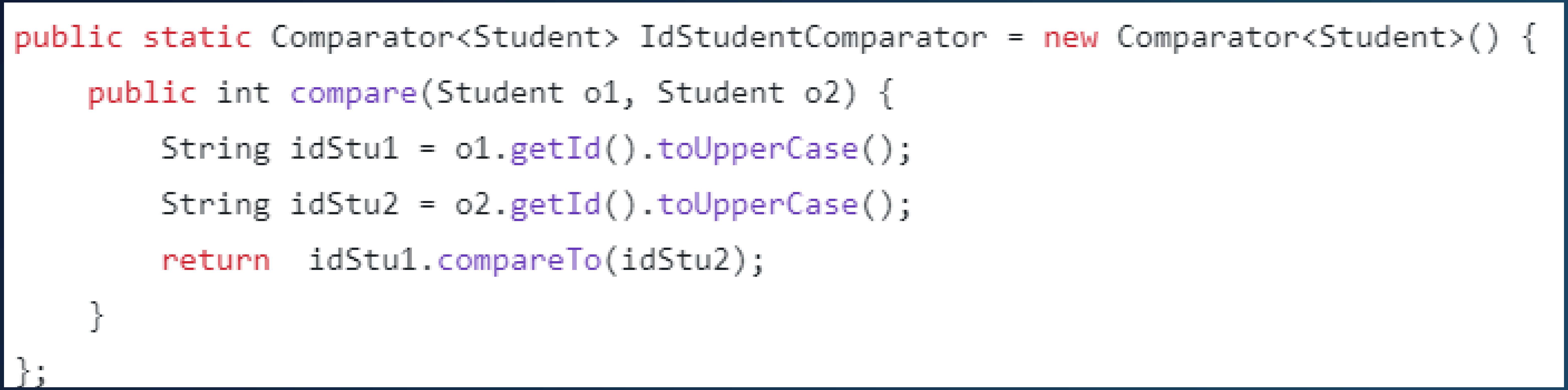
```
if(this.mark >= 0 && this.mark <5){
    this.rank = "Fail";
} else if (this.mark >=5 && this.mark < 6.5) {
    this.rank = "Medium";
} else if (this.mark >= 6.5 && this.mark < 7.5) {
    this.rank = "Good";
} else if (this.mark >= 7.5 && this.mark < 9) {
    this.rank = "Very Good";
} else if(this.mark >= 9 && this.mark <= 10){
    this.rank = "Excellent";
} else {
    this.rank = null;
}
```



```
//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;
}
```

The code in the image is a simple implementation of Java getter and setter methods for the fields fullName, id, and mark. Here's an explanation of each part:

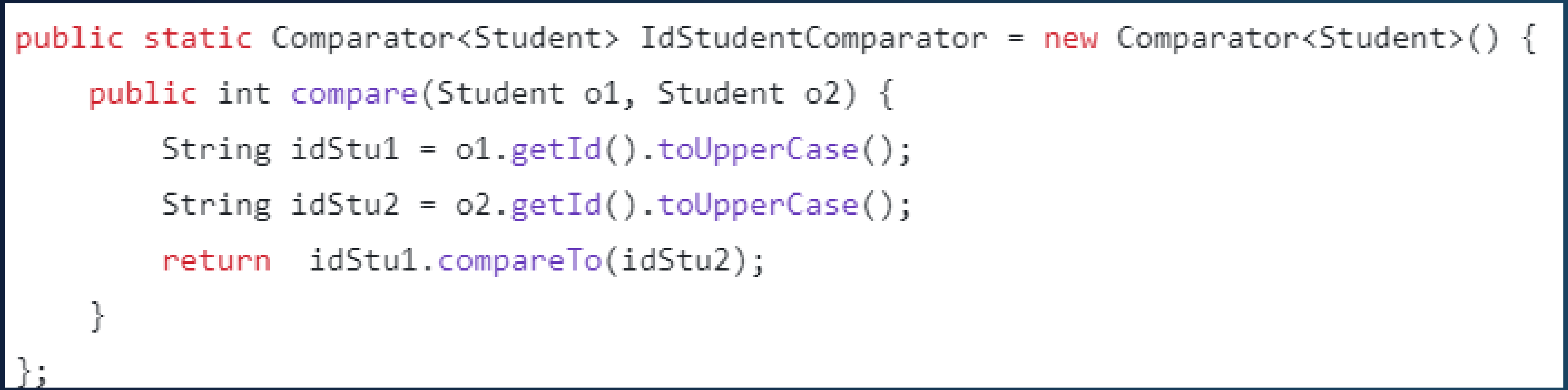
- **getFullName() and setFullName():**
  - These methods manage access to and modification of the **fullName** attribute.
  - **getFullName()** returns the current value of **fullName**.
  - **setFullName(String fullName)** assigns a new value to **fullName**.
- **getId() and setId():**
  - These methods are for handling the **id** field.
  - **getId()** returns the current value of the **id** attribute.
  - **setId(String id)** allows updating the value of **id.**
- **getMark() and setMark():**
  - These methods control access to and modification of the **mark** attribute.
  - **getMark()** returns the current value of **mark**, which is a **double.**
  - **setMark(double mark)** sets a new value to the **mark** attribute.



```
public static Comparator<Student> IdStudentComparator = new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
        String idStu1 = o1.getId().toUpperCase();
        String idStu2 = o2.getId().toUpperCase();
        return  idStu1.compareTo(idStu2);
    }
};
```

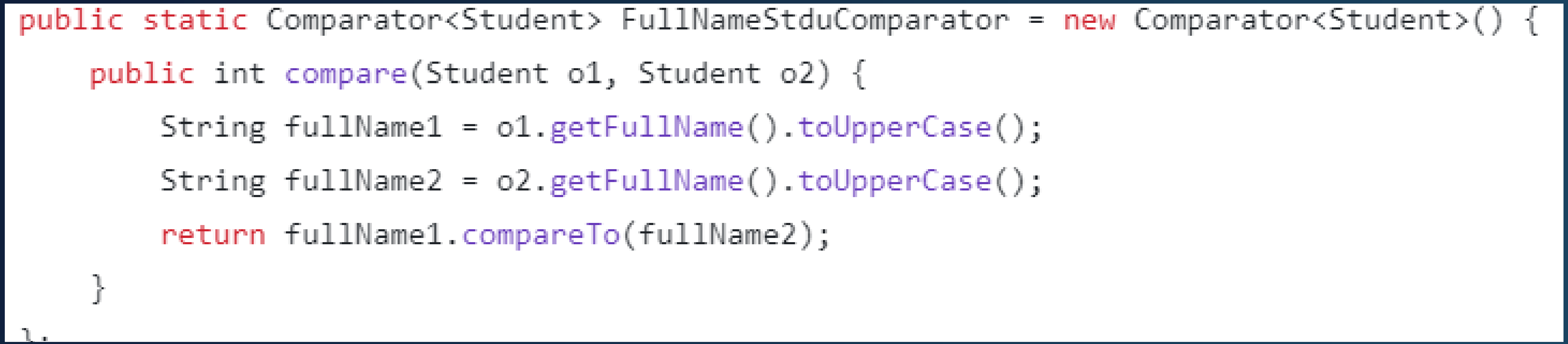
This code defines a comparator for comparing two Student objects based on their id values. Here's an explanation of each part:

- **public static Comparator<Student> IdStudentComparator:**
  - This creates a static comparator object that can be used to compare **Student** objects. It is declared as **Comparator<Student>,** meaning it works specifically for **Student** objects.
- **new Comparator<Student>() { ... }:**
  - This creates an anonymous class that implements the **Comparator<Student>** interface, defining the logic for comparing two **Student** objects.



```
public static Comparator<Student> IdStudentComparator = new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
        String idStu1 = o1.getId().toUpperCase();
        String idStu2 = o2.getId().toUpperCase();
        return  idStu1.compareTo(idStu2);
    }
};
```

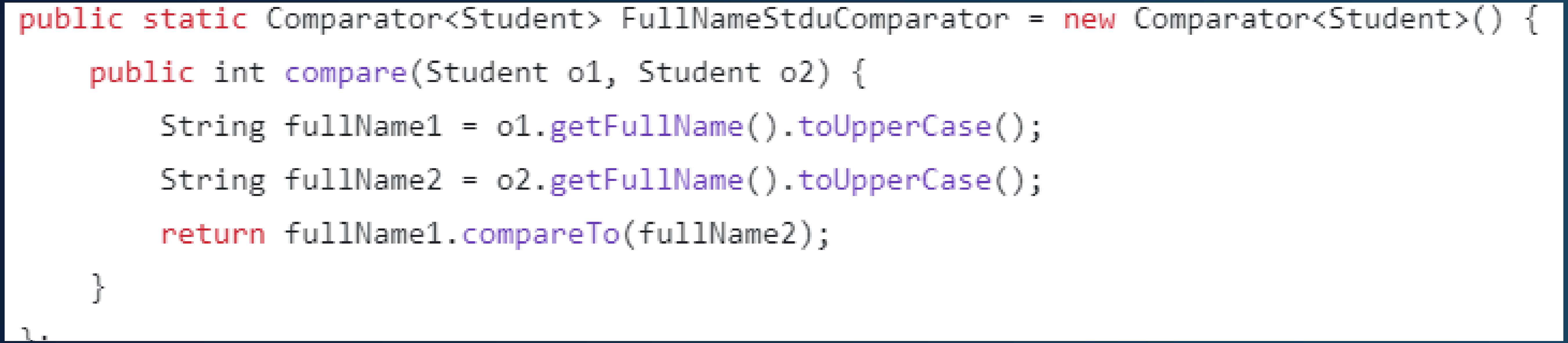
- **compare(Student o1, Student o2):**
  - This method implements the comparison logic.
  - It takes two **Student** objects **(o1 and o2)** as parameters and compares them based on their **id** fields.
- **o1.getId().toUpperCase() and o2.getId().toUpperCase():**
  - These lines retrieve the **id** of each student and convert it to **uppercase** to ensure case-insensitive comparison.
- **idStu1.compareTo(idStu2):**
- This compares the two **id** strings **(idStu1 and idStu2)**.
- The **compareTo** method returns:
  - A negative value if **idStu1** is lexicographically smaller than **idStu2**.
  - **0** if they are equal.
  - A positive value if **idStu1** is larger than **idStu2**.



```
public static Comparator<Student> FullNameStduComparator = new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
        String fullName1 = o1.getFullName().toUpperCase();
        String fullName2 = o2.getFullName().toUpperCase();
        return fullName1.compareTo(fullName2);
    }
};
```

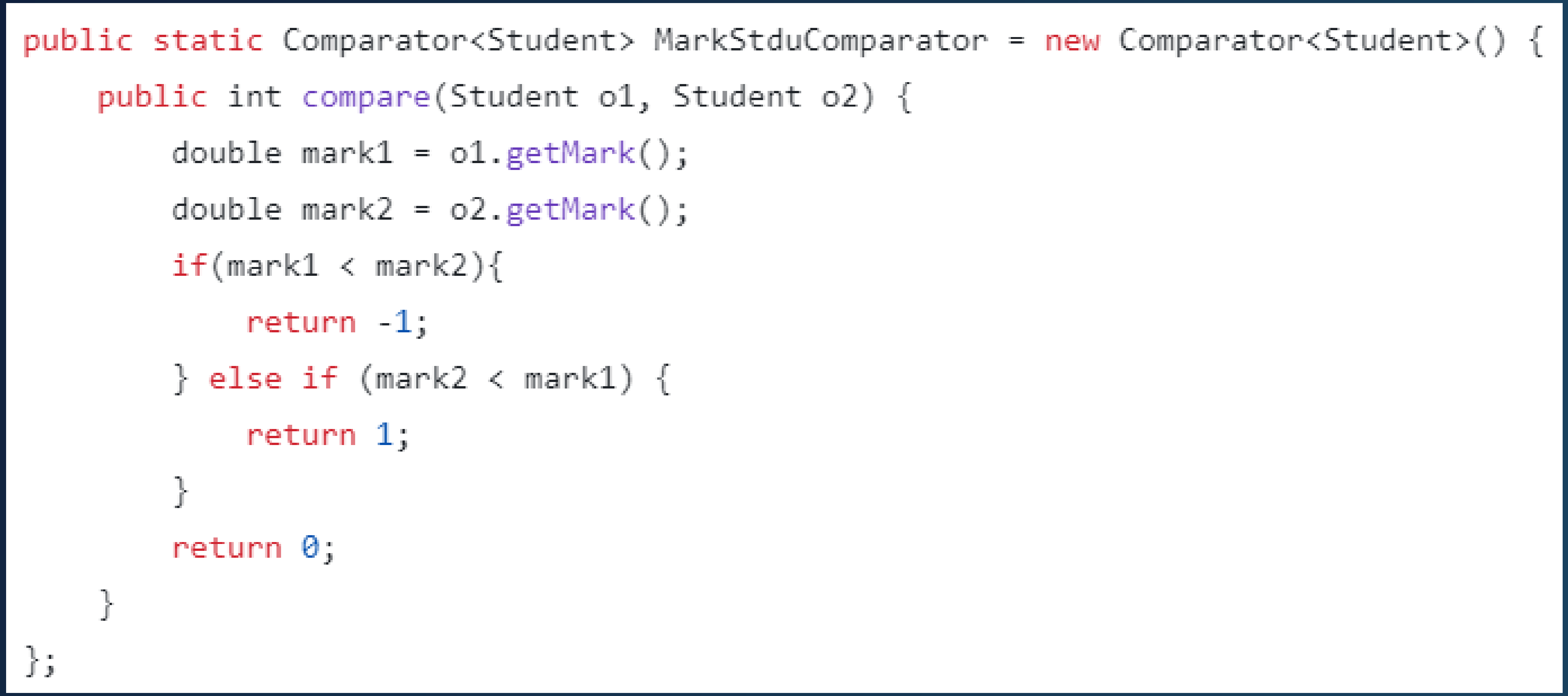
The code shown defines a comparator for comparing two Student objects based on their fullName values. Here's an explanation of each part:

- **public static Comparator<Student> FullNameStdComparator:**
    - This defines a static comparator for comparing **Student** objects based on their **fullName**. It is of type **Comparator<Student>**, which means it works specifically for comparing **Student** objects.
- **new Comparator<Student>() { ... }:**
    - This is an anonymous class that implements the **Comparator<Student>** interface. It provides the logic required to compare two **Student** objects based on their full names.



```
public static Comparator<Student> FullNameStduComparator = new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
        String fullName1 = o1.getFullName().toUpperCase();
        String fullName2 = o2.getFullName().toUpperCase();
        return fullName1.compareTo(fullName2);
    }
};
```

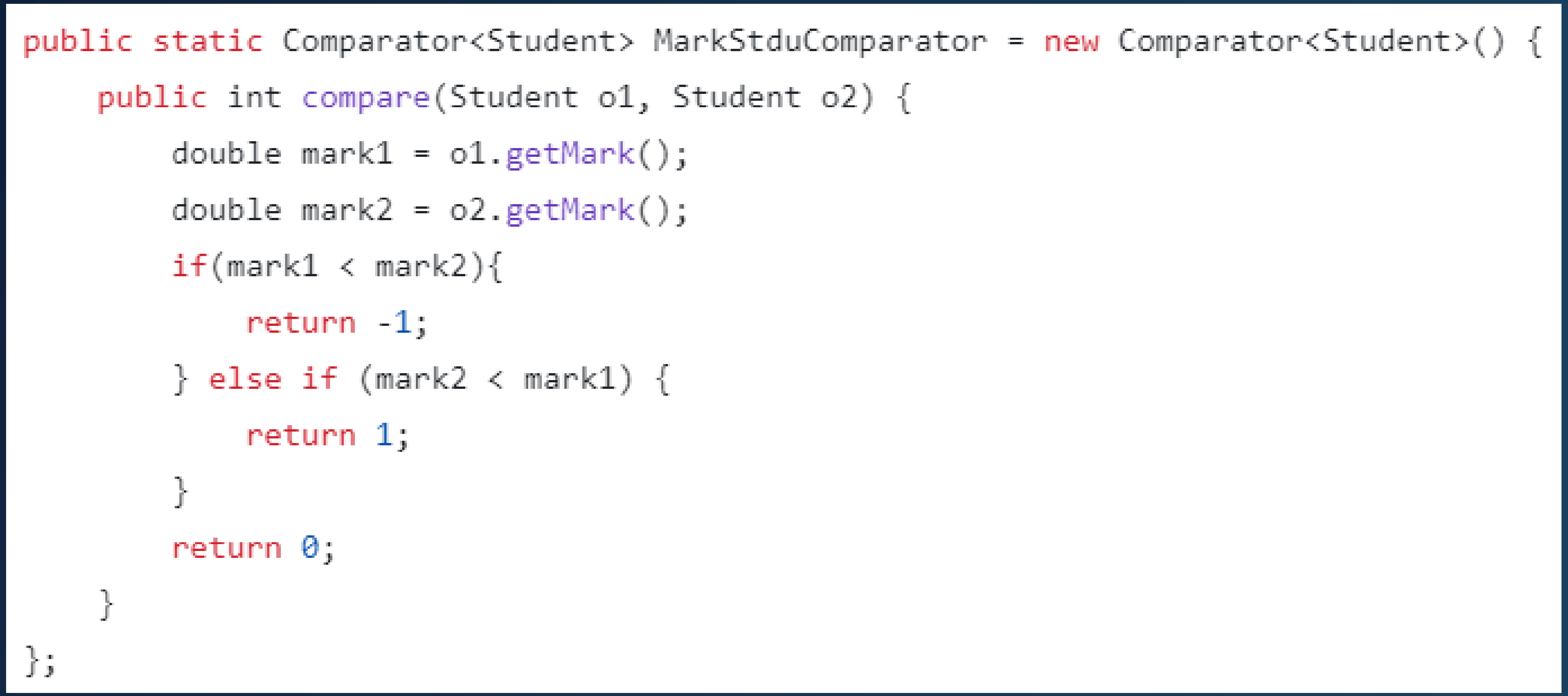
- **compare(Student o1, Student o2):**
  - This method contains the logic to compare two **Student** objects. It takes two **Student** objects, **o1** and **o2**, as input parameters and compares their fullName fields.
- **o1.getFullName().toUpperCase() and o2.getFullName().toUpperCase():**
  - These lines retrieve the **fullName** of each **Student** object and convert them to **uppercase**. This ensures that the comparison is **case-insensitive** (so "John" and "john" are treated as equal).
- **fullName1.compareTo(fullName2):**
  - This compares the two **fullName** strings (**fullName1** and **fullName2**).
  - The **compareTo** method returns:
  - A negative value if **fullName1** is lexicographically smaller than **fullName2**.
  - **0** if they are equal.
  - A positive value if **fullName1** is larger than **fullName2**.



```
public static Comparator<Student> MarkStduComparator = new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
        double mark1 = o1.getMark();
        double mark2 = o2.getMark();
        if(mark1 < mark2){
            return -1;
        } else if (mark2 < mark1) {
            return 1;
        }
        return 0;
    }
};
```

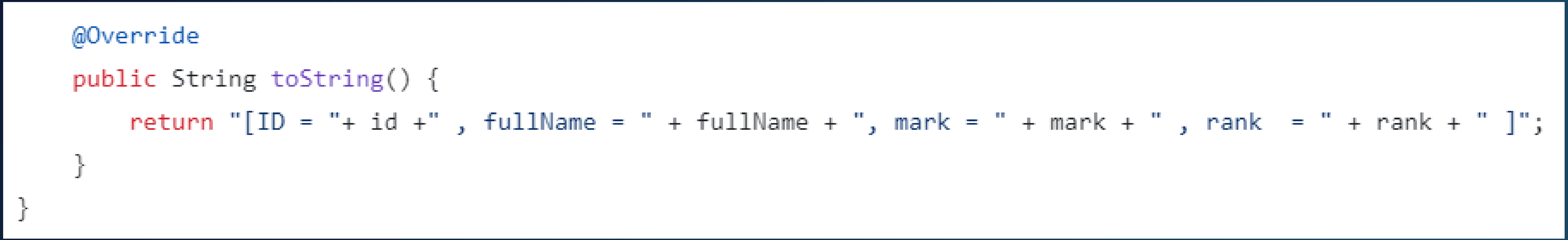
This code defines a class called ComparatorStudent with a static method compare(Student o1, Student o2). The purpose of this method is to compare two Student objects based on their marks. Here's a breakdown of the code:

- **public static Comparator<Student> MarkStduComparator = new Comparator<Student>() { ... }:**
    - This line creates a new **Comparator<Student>** object and assigns it to the **MarkStduComparator** static field. The Comparator interface is used to define a custom comparison logic for objects.
- **public int compare(Student o1, Student o2) { ... }:**
    - This is the implementation of the compare**(Student o1, Student o2)** method, which is part of the **Comparator** interface. This method takes two **Student** objects as input and returns an integer value that represents the comparison result.



```java
public static Comparator<Student> MarkStduComparator = new Comparator<Student>() {
    public int compare(Student o1, Student o2) {
        double mark1 = o1.getMark();
        double mark2 = o2.getMark();
        if(mark1 < mark2){
            return -1;
        } else if (mark2 < mark1) {
            return 1;
        }
        return 0;
    }
};
```

- **double mark1 = o1.getMark(); and double mark2 = o2.getMark();:**
  - These lines retrieve the marks of the two **Student** objects and store them in the **mark1** and **mark2** variables, respectively.
- **if (mark1 < mark2) { return -1; } else if (mark2 < mark1) { return 1; }:**
  - This part of the code compares the marks of the two **Student** objects. If the mark of **o1** is less than the mark of **o2**, it **returns -1**, indicating that **o1** is less than **o2**. If the mark of o2 is less than the mark of **o1**, it returns 1, indicating that **o1** is greater than **o2**.
- **return 0;:** If the marks of the two **Student** objects are equal, the method returns **0**, indicating that they are equal.

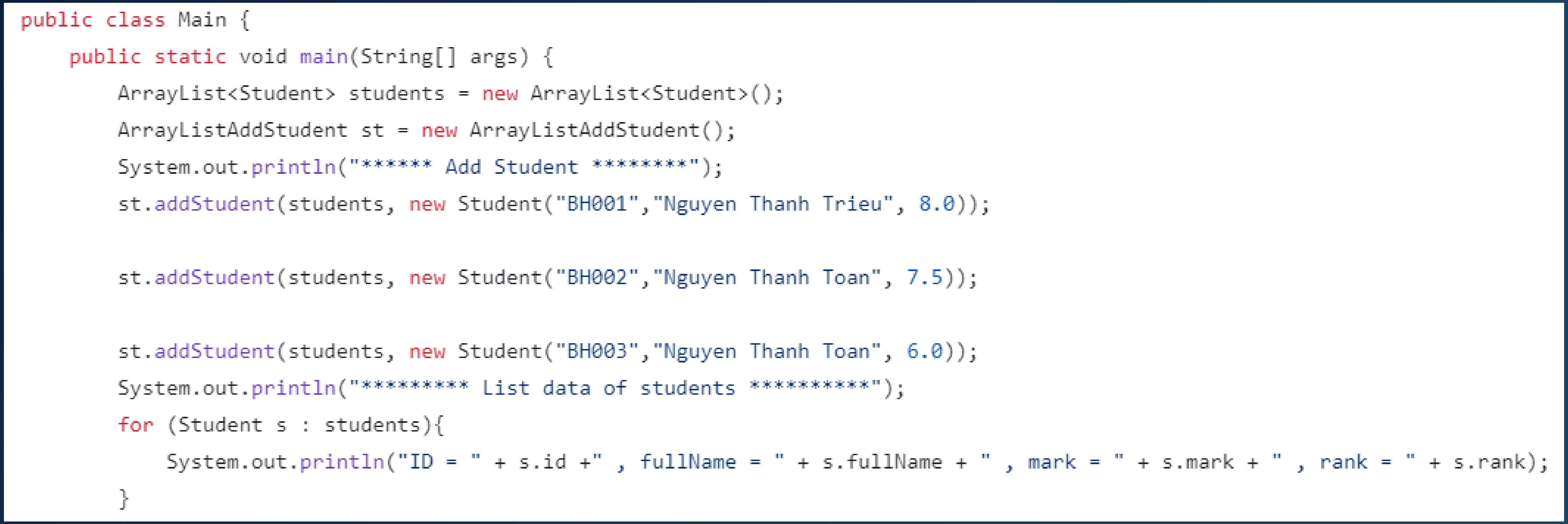


```
    @Override
    public String toString() {
        return "[ID = "+ id +" , fullName = " + fullName + ", mark = " + mark + " , rank  = " + rank + " ]";
    }
}
```

The provided code defines a public method named toString() within a class called @Override. The purpose of this method is to return a string representation of an object in a specific format. Here's a breakdown of the code:

- **public String toString() {:**
  - This line declares the toString() method, which is a standard method in Java that is used to provide a string representation of an object.
- **return "[ID = " + id + " , fullName = " + fullName + ", mark = " + mark + ", rank = " + rank + "]";:**
  - This line constructs and returns a string that includes the values of several properties of the object, such as id, fullName, mark, and rank. The string is formatted to include labels for each of these properties, enclosed within square brackets.

MAIN

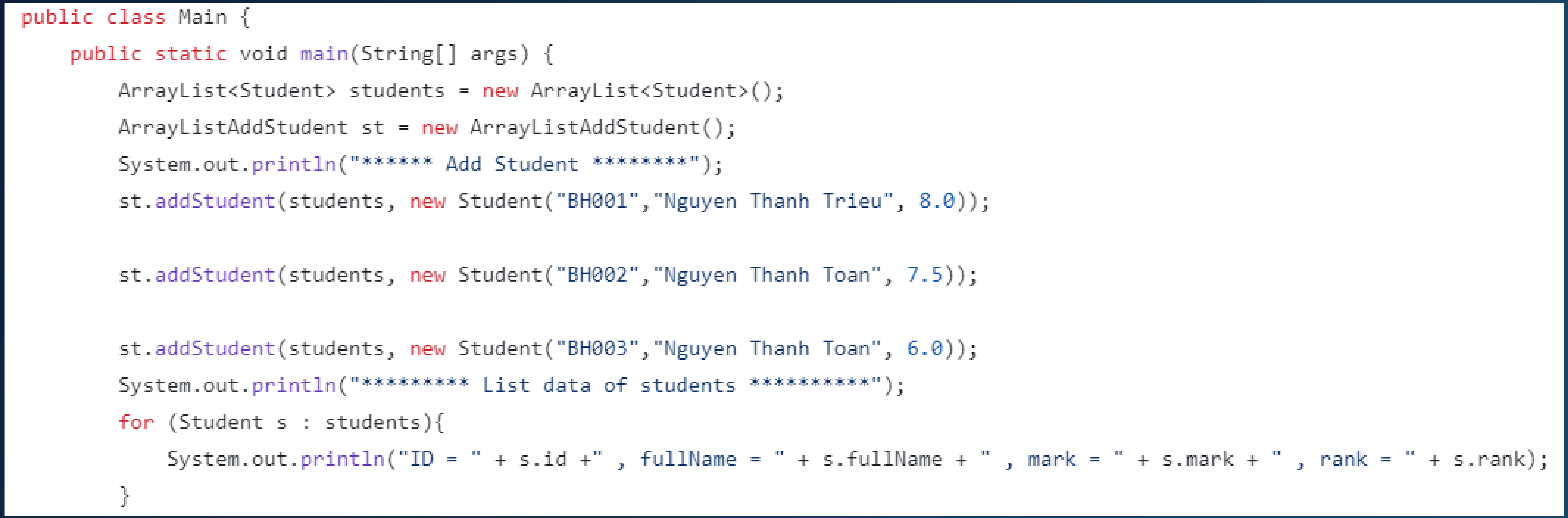


```java
public class Main {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<Student>();
        ArrayListAddStudent st = new ArrayListAddStudent();
        System.out.println("****** Add Student ********");
        st.addStudent(students, new Student("BH001","Nguyen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

        st.addStudent(students, new Student("BH002","Nguyen Thanh Toan", 7.5));

        st.addStudent(students, new Student("BH003","Nguyen Thanh Toan", 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);
        }
```

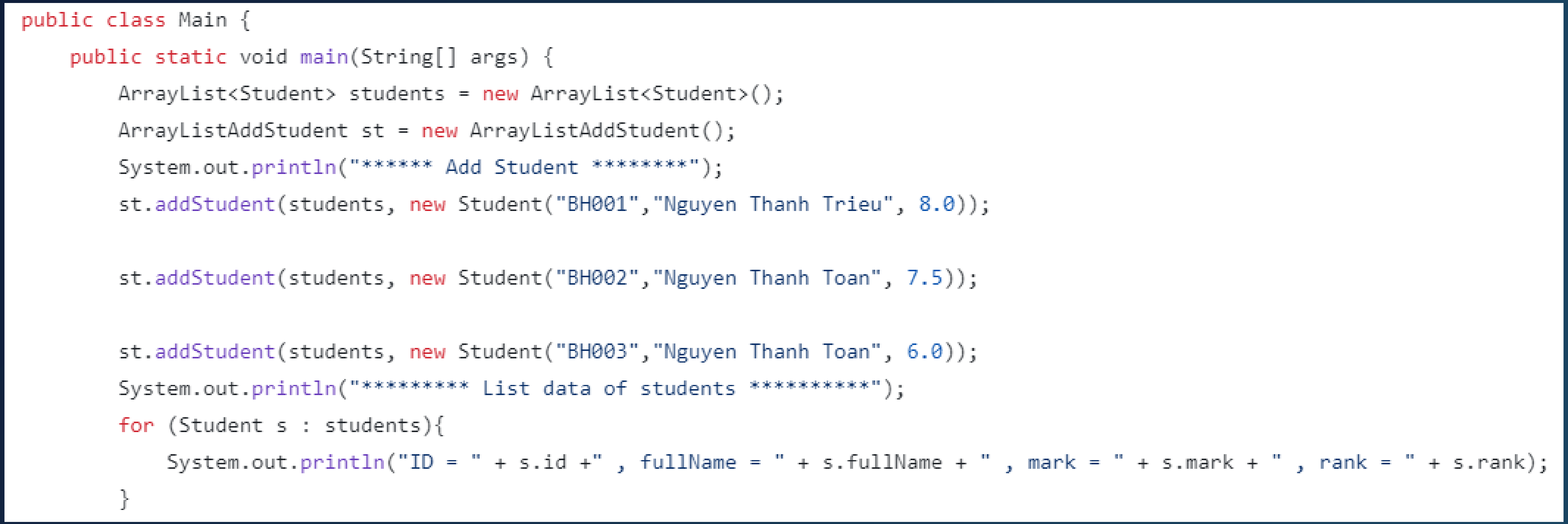
The purpose of this code is to create and manipulate a list of Student objects.

Here's a breakdown of the code:

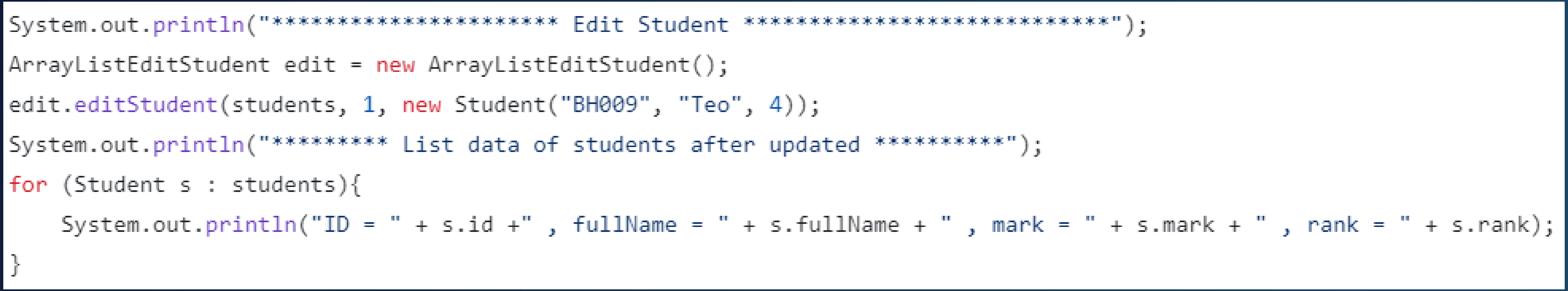
- **public class Main {:** This line declares the Main class.
- **public static void main(String[] args) {:** This line defines the main() method, which is the entry point of the program.
- **ArrayList<Student> students = new ArrayList<>();:** This line creates a new ArrayList of Student objects and assigns it to the students variable.


```
public class Main {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<Student>();
        ArrayListAddStudent st = new ArrayListAddStudent();
        System.out.println("****** Add Student ********");
        st.addStudent(students, new Student("BH001","Nguyen Thanh Trieu", 8.0));

        st.addStudent(students, new Student("BH002","Nguyen Thanh Toan", 7.5));

        st.addStudent(students, new Student("BH003","Nguyen Thanh Toan", 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);
        }
```

- **ArrayListAddStudent st = new ArrayListAddStudent();:** This line creates a new **ArrayListAddStudent** object and assigns it to the st variable. This object likely has methods to add new **Student** objects to the students list.
- **st.addStudent(students, new Student("BH001", "Nguyen Thanh Trieu", 8.0));:** This line calls the **addStudent()** method of the st object, passing the **students** list and a new Student object as arguments. This adds a new student with the ID "BH001", name "Nguyen Thanh Trieu", and mark 8.0 to the students list.


```
public class Main {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<Student>();
        ArrayListAddStudent st = new ArrayListAddStudent();
        System.out.println("****** Add Student ********");
        st.addStudent(students, new Student("BH001","Nguyen Thanh Trieu", 8.0));

        st.addStudent(students, new Student("BH002","Nguyen Thanh Toan", 7.5));

        st.addStudent(students, new Student("BH003","Nguyen Thanh Toan", 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);
        }
```

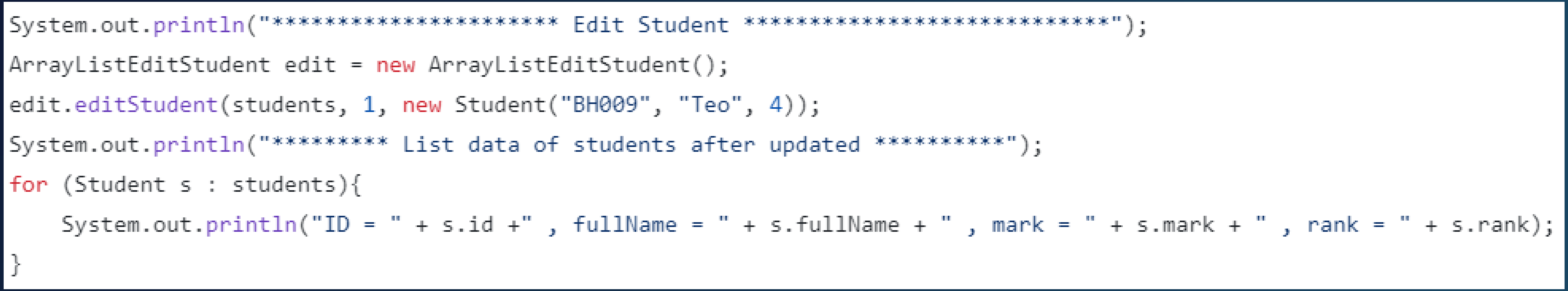
- **st.addStudent(students, new Student("BH002", "Nguyen Thanh Toan", 7.5));:** This line adds another student with the ID "BH002", name "Nguyen Thanh Toan", and mark 7.5 to the students list.
- **st.addStudent(students, new Student("BH003", "Nguyen Thanh Toan", 6.0));:** This line adds a third student with the ID "BH003", name "Nguyen Thanh Toan", and mark 6.0 to the students list.
- **System.out.println("********* List data of students *********");:** This line prints a message to the console.
- **for (Student s : students) { ... }:** This is a "for-each" loop that iterates over the students list and prints the ID, fullName, mark, and rank of each student.



```
System.out.println("********************** Edit Student ****************************");
ArrayListEditStudent edit = new ArrayListEditStudent();
edit.editStudent(students, 1, new Student("BH009", "Teo", 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);
}
```

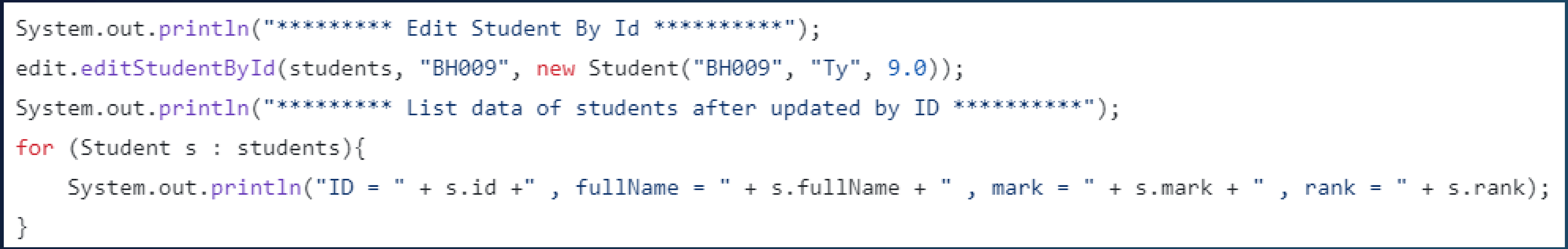
- **System.out.println("******************** Edit Student ********************");:** This line prints a message to the console indicating that the student list is about to be edited.
- **ArrayListEditStudent edit = new ArrayListEditStudent();:** This line creates a new ArrayListEditStudent object and assigns it to the edit variable. This object likely has methods to edit the Student objects in the students list.
- **edit.editStudent(students, 1, new Student("BH009", "Teo", 4.0));:** This line calls the editStudent() method of the edit object. It takes three arguments:
  - students: the list of Student objects to be edited
  - 1: the index of the student to be edited (in this case, the second student in the list)
  - new Student("BH009", "Teo", 4.0): a new Student object with the updated information (ID "BH009", name "Teo", and mark 4.0)

This line effectively updates the second student in the students list with the new information.



```java
System.out.println("********************** Edit Student *****************************");
ArrayListEditStudent edit = new ArrayListEditStudent();
edit.editStudent(students, 1, new Student("BH009", "Teo", 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);
}
```

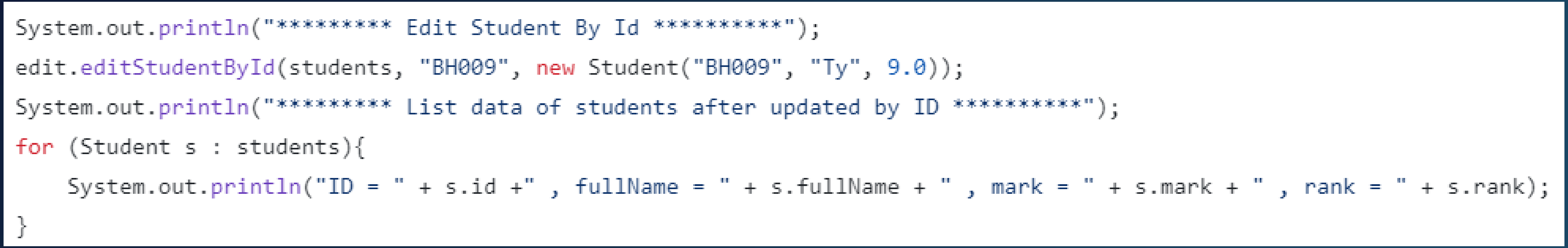
- **System.out.println("********* List data of students after updated **********");:** This line prints a message to the console indicating that the list of students has been updated.
- **for (Student s : students) { ... }:** This is the same "for-each" loop as in the previous code, which iterates over the students list and prints the ID, fullName, mark, and rank of each student.



```
System.out.println("********* Edit Student By Id **********");
edit.editStudentById(students, "BH009", new Student("BH009", "Ty", 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("********* Edit Student By Id *********");:** This line prints a message to the console indicating that a student will be edited by their ID.
- **edit.editStudentById(students, "BH009", new Student("BH009", "Ty", 9.0));:** This line calls the editStudentById() method of the edit object. It takes three arguments:
  - students: the list of Student objects to be edited
  - "BH009": the ID of the student to be edited
  - new Student("BH009", "Ty", 9.0): a new Student object with the updated information (ID "BH009", name "Ty", and mark 9.0)

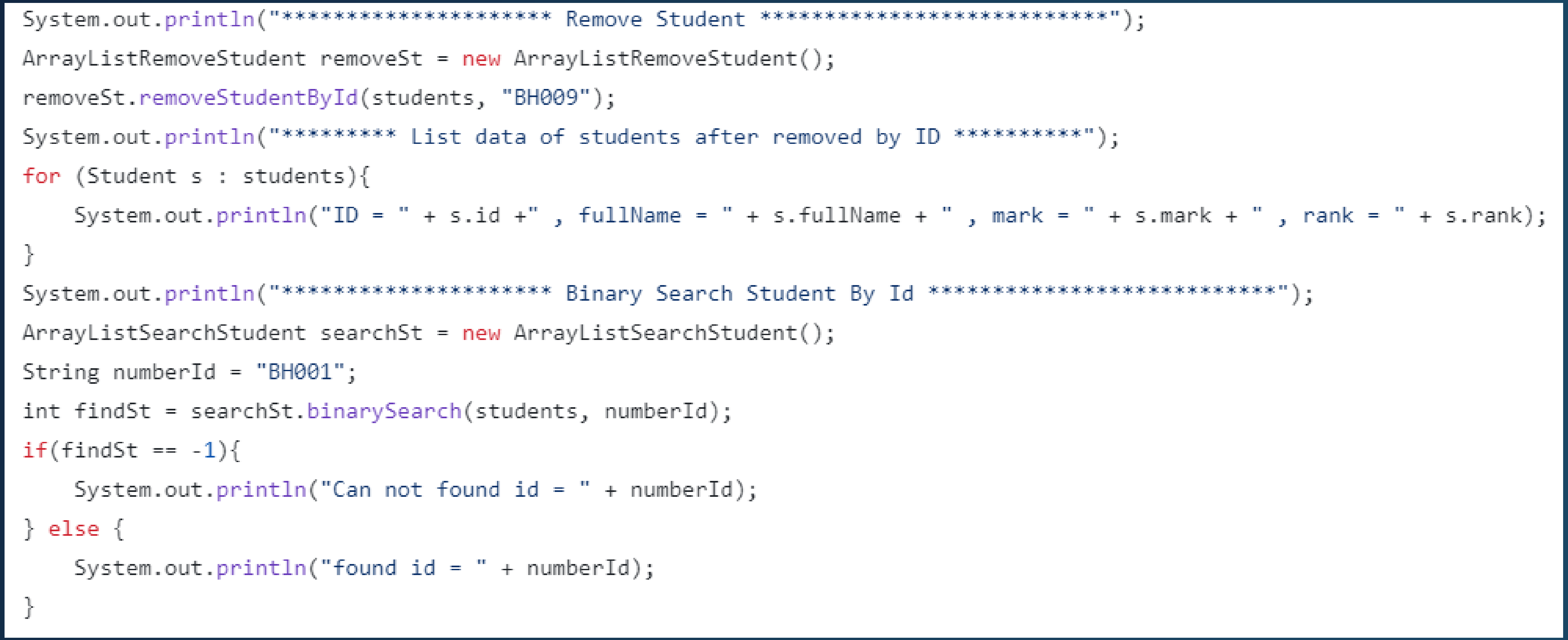
This line effectively updates the student in the students list with the ID "BH009" with the new information.



```
System.out.println("********* Edit Student By Id **********");
edit.editStudentById(students, "BH009", new Student("BH009", "Ty", 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("********* List data of students after updated by ID *********");:** This line prints a message to the console indicating that the list of students has been updated.
- **for (Student s : students) { ... }:** This is the same "for-each" loop as in the previous code, which iterates over the students list and prints the ID, fullName, mark, and rank of each student.

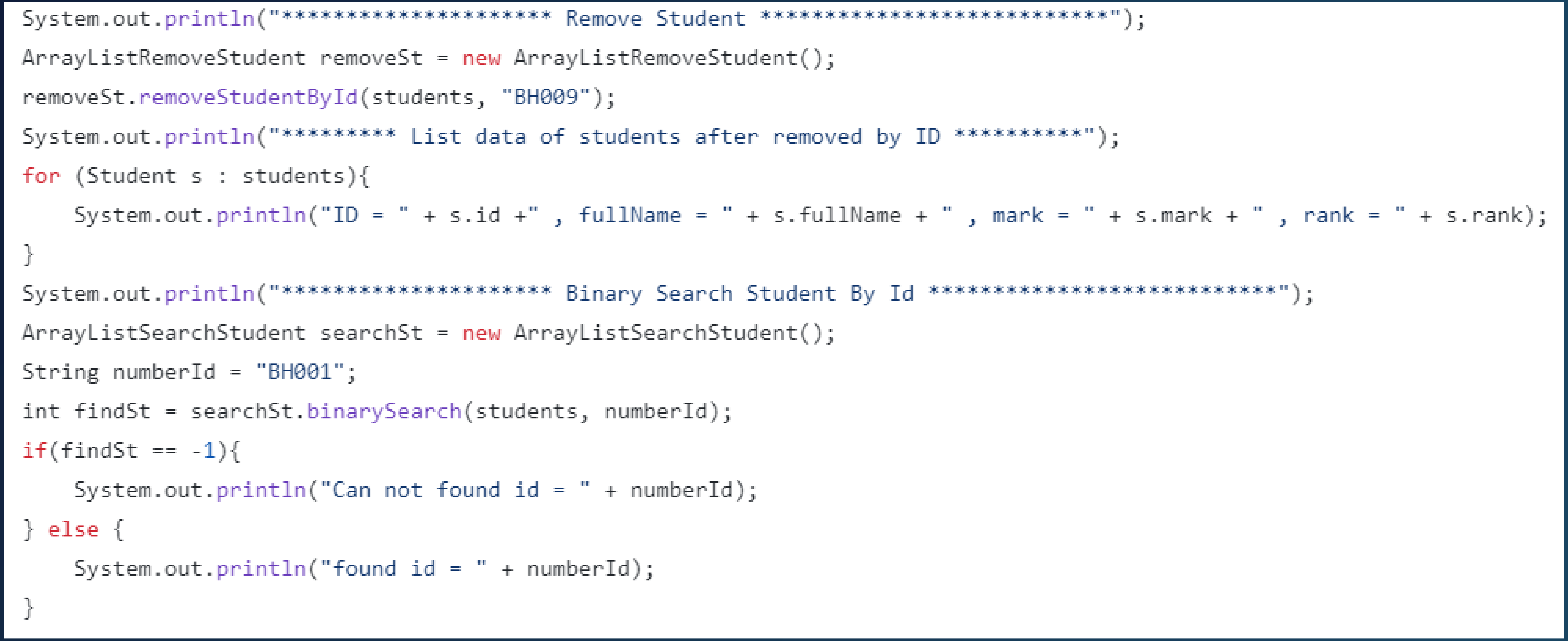
Removing a Student by ID:



```java
System.out.println("********************* Remove Student ***************************");
ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();
removeSt.removeStudentById(students, "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("******************** Remove Student ********************");:** This line prints a message to the console indicating that a student will be removed.
- **ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();:** This line creates a new ArrayListRemoveStudent object and assigns it to the removeSt variable. This object likely has methods to remove Student objects from the students list.
- **removeSt.removeStudentById(students, "BH009");:** This line calls the removeStudentById() method of the removeSt object. It takes two arguments:
  - students: the list of Student objects to be modified
  - "BH009": the ID of the student to be removed

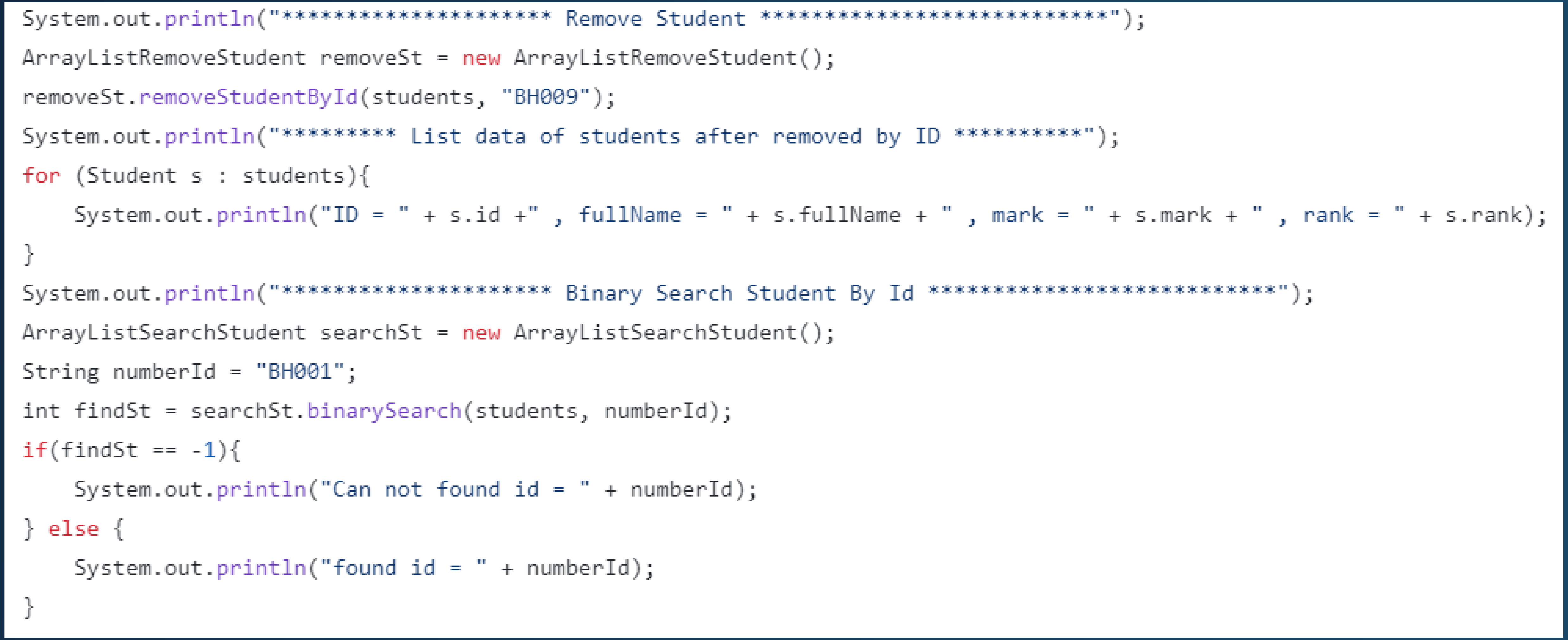
This line effectively removes the student with the ID "BH009" from the students list.



```
System.out.println("******************** Remove Student **************************");
ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();
removeSt.removeStudentById(students, "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);
}
```

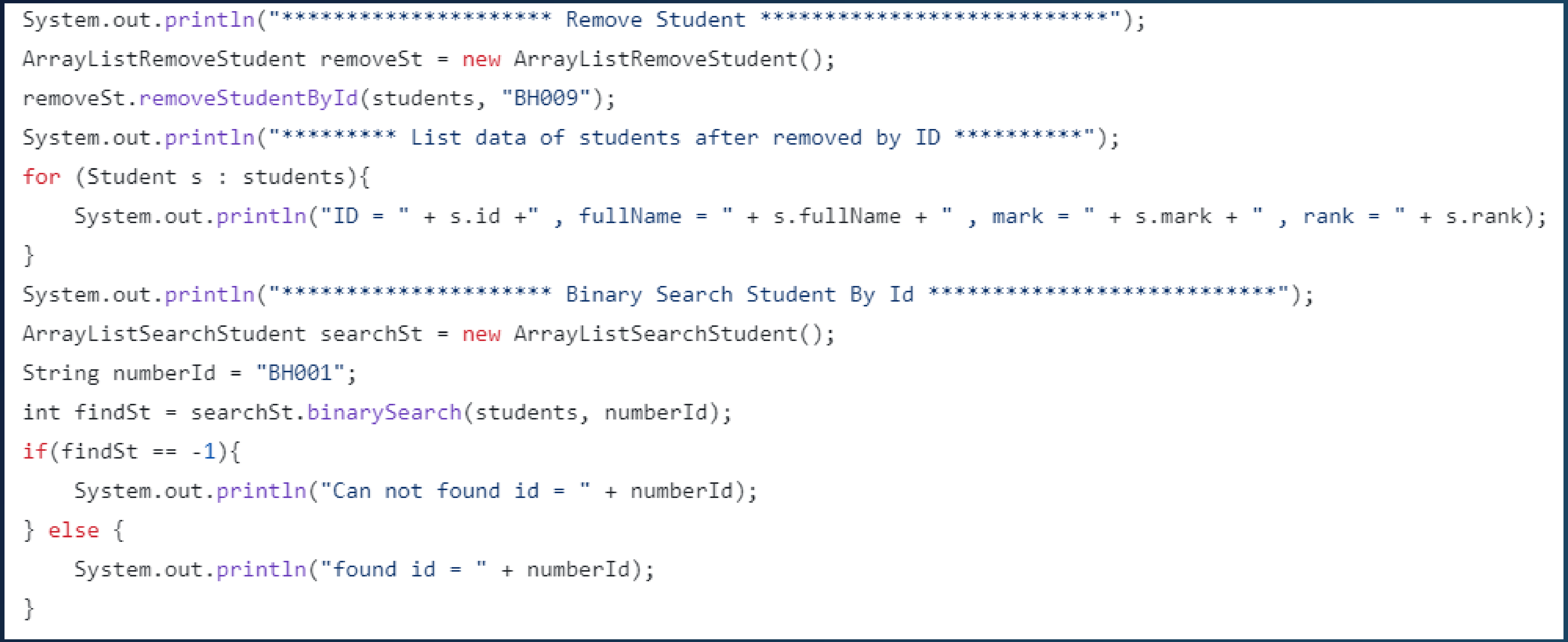
- **System.out.println("********* List data of students after removed by ID *********");:** This line prints a message to the console indicating that the list of students has been updated after the removal.
- **for (Student s : students) { ... }:** This is the same "for-each" loop as in the previous code, which iterates over the students list and prints the ID, fullName, mark, and rank of each student.

Binary Search for a Student by ID:



```
System.out.println("********************* Remove Student ***************************");
ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();
removeSt.removeStudentById(students, "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("******************** Binary Search Student By Id *********************");:** This line prints a message to the console indicating that a student will be searched for by their ID using binary search.
- **ArrayListSearchStudent searchSt = new ArrayListSearchStudent();**: This line creates a new ArrayListSearchStudent object and assigns it to the searchSt variable. This object likely has methods to perform binary search on the students list.
- **String numberId = "BH001";:** This line assigns the string value "BH001" to the numberId variable, which represents the ID of the student to be searched for.

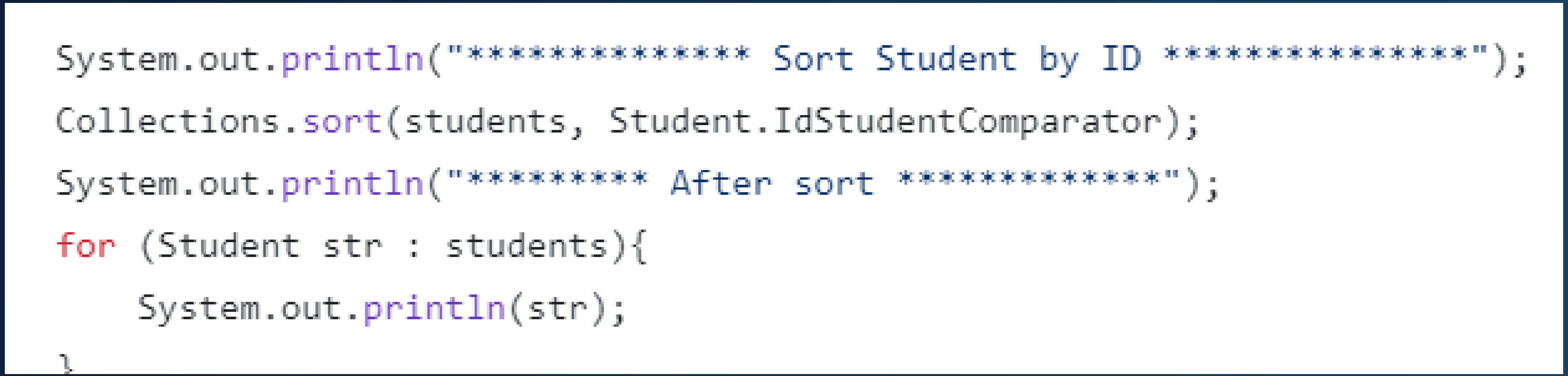


```
System.out.println("********************* Remove Student ***************************");
ArrayListRemoveStudent removeSt = new ArrayListRemoveStudent();
removeSt.removeStudentById(students, "BH009");
System.out.println("********* List data of students after removed by ID **********");
for (Student 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);
}
```

- **int findSt = searchSt.binarySearch(students, numberId);:** This line calls the binarySearch() method of the searchSt object. It takes two arguments:
    - students: the list of Student objects to be searched
    - numberId: the ID of the student to be searched for

The method returns the index of the student with the matching ID, or -1 if the student is not found.

- **if (findSt == -1) { ... } else { ... }:** This is an if-else statement that checks the value of findSt. If it's -1, it means the student was not found, and the code prints a message indicating that. If the value is not -1, it means the student was found, and the code prints a message indicating the found ID.



```
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);
}
```

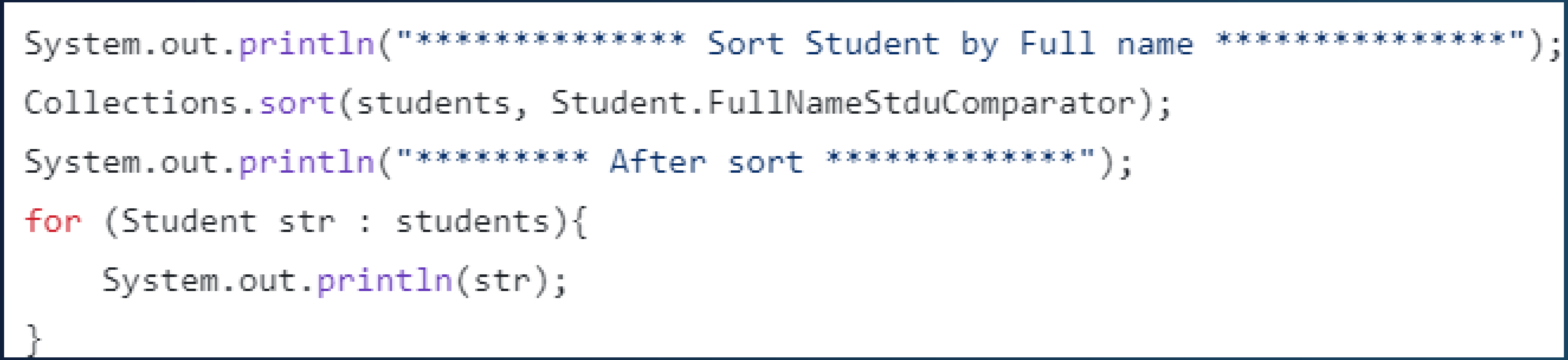
This code snippet demonstrates how to sort a list of Student objects by their ID using the Collections.sort() method.

Here's a breakdown of the code:

- **System.out.println("*************** Sort Student by ID ***************");:** This line prints a message to the console indicating that the students will be sorted by their ID.
- **Collections.sort(students, Student.IdStudentComparator);:** This line calls the sort() method of the Collections class. It takes two arguments:
  - **students:** the list of Student objects to be sorted
  - **Student.IdStudentComparator:** a static Comparator object defined in the Student class, which is used to compare Student objects based on their ID.

This line sorts the students list in ascending order based on the student ID.

- **System.out.println("********* After sort ***********");:** This line prints a message to the console indicating that the sorting process is complete.
- **for (Student str : students) { ... }:** This "for-each" loop iterates over the sorted students list and prints the details of each student.



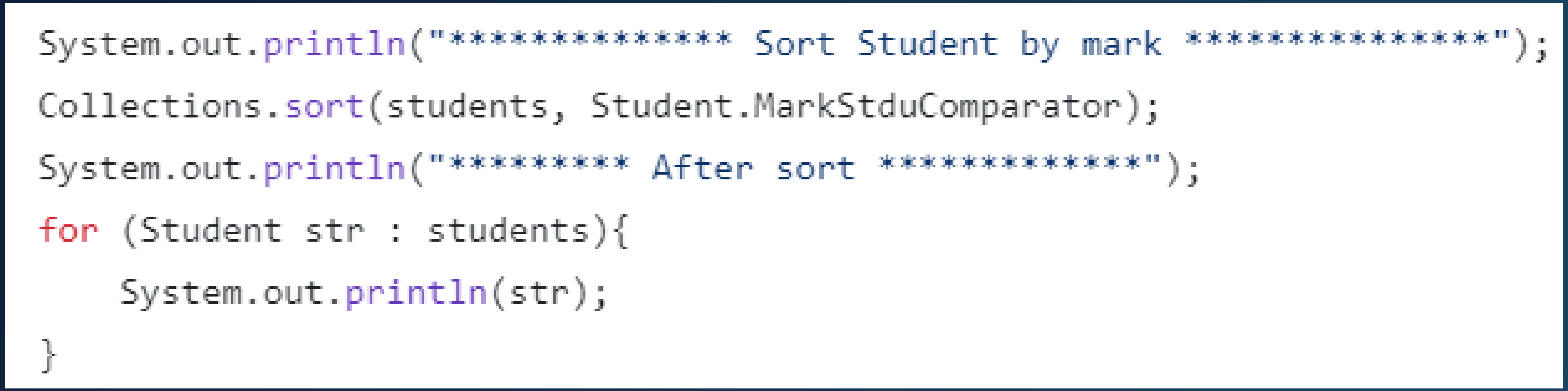
```java
System.out.println("************** Sort Student by Full name ***************");
Collections.sort(students, Student.FullNameStduComparator);
System.out.println("********* After sort *************");
for (Student str : students){
    System.out.println(str);
}
```

This code snippet demonstrates how to sort a list of Student objects by their full name using the Collections.sort() method.
Here's a breakdown of the code:

- **System.out.println("*************** Sort Student by Full name **************");:** This line prints a message to the console indicating that the students will be sorted by their full name.
- **Collections.sort(students, Student.FullNameStduComparator);:** This line calls the sort() method of the Collections class. It takes two arguments:
    - students: the list of Student objects to be sorted
    - Student.FullNameStduComparator: a static Comparator object defined in the Student class, which is used to compare Student objects based on their full name.

This line sorts the students list in ascending order based on the student's full name.

- **System.out.println("********* After sort ***********");:** This line prints a message to the console indicating that the sorting process is complete.
- **for (Student str : students) { ... }:** This "for-each" loop iterates over the sorted students list and prints the details of each student.



```
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);
}
```

This code snippet demonstrates how to sort a list of Student objects by their marks using the Collections.sort() method.
Here's a breakdown of the code:

- **System.out.println("*************** Sort Student by mark ***************");:** This line prints a message to the console indicating that the students will be sorted by their marks.
- **Collections.sort(students, Student.MarkStduComparator);:** This line calls the sort() method of the Collections class. It takes two arguments:
  - **students:** the list of Student objects to be sorted
  - **Student.MarkStduComparator:** a static Comparator object defined in the Student class, which is used to compare Student objects based on their marks.

This line sorts the students list in ascending order based on the student's marks.

- **System.out.println("********* After sort ***********");:** This line prints a message to the console indicating that the sorting process is complete.
- **for (Student str : students) { ... }:** This "for-each" loop iterates over the sorted students list and prints the details of each student.