University.java

```java
import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;

public class University {

    private static ModuleDescriptor[] moduleDescriptors;

    private static Student[] students;

    private static Module[] modules = new Module[]{};

    private static StudentRecord[] studentRecords;

    /**
     * @return The number of students registered in the system.
     */
    public static int getTotalNumberStudents() {
        return students.length;
    }

    /**
     * @return The student with the highest gpa.
     */
    public static Student getBestStudent() {
        double bestStudent = 0;
        Student theBestStudent = null;
        for(Student i:students){
            if(i.getGpa() > bestStudent){
                bestStudent = i.getGpa();
                theBestStudent = i;
            }
        }
        return theBestStudent;
    }

    /**
     * @return The module with the highest average score.
     */
    public static Module getBestModule() {
        double bestModule = 0;
        Module theBestModule = null;
        for(Module i:modules){
            if(i.getFinalAverageGrade() > bestModule){
                bestModule = i.getFinalAverageGrade();
                theBestModule = i;
            }
        }
        return theBestModule;
    }

    public static void main(String[] args) throws IOException {
        String name;
        String code;
        double[] weights;

        // ------- Create ModuleDescriptors
```

```

ArrayList<ModuleDescriptor> tempModuleList = new ArrayList<>();
String line = "";
String splitBy = ", ";
BufferedReader br = readCSV("module_descriptors.csv");

boolean firstTime = true;  // Ignore first csv line.
while ((line = br.readLine()) != null) {
	if (firstTime) {
		firstTime = false;
		continue;
	}
	String[] descriptors = line.split(splitBy);
	name = descriptors[0];
	code = descriptors[1];
	String tempWeights = descriptors[2];
	weights = stringToArray(tempWeights);
	ModuleDescriptor newModule = new ModuleDescriptor(code, name, weights);
	tempModuleList.add(newModule);
}

// Convert arraylist to array.

moduleDescriptors = tempModuleList.toArray(new ModuleDescriptor[0]);

// -------- Create modules

ArrayList<Module> tempModules = new ArrayList<>();
String newCode;
```

```
int newYear;
byte newTerm;
BufferedReader tr = readCSV("module.csv");
firstTime = true;
while ((line = tr.readLine()) != null) {
    if (firstTime) {
        firstTime = false;
        continue;
    }

    String[] module = line.split(splitBy);
    newYear = Integer.parseInt(module[2]);
    newTerm = convertStringToByte(module[3]);
    newCode = module[1];
    boolean exist = false;
    for(Module i: tempModules){
        if(newCode.equals(i.getCode()) && newTerm == i.getTerm() && newYear == i.getYear())
            exist = true;
    }
    if(exist)
        continue;
    Module newModule = new Module(newYear, newTerm, findModuleDescriptor(newCode));
    tempModules.add(newModule);
    modules = tempModules.toArray(new Module[0]);
}

int newId;
```

```
String newName;
char newGender;
BufferedReader gr = readCSV("students.csv");
firstTime = true;
ArrayList<Student> newStudentList = new ArrayList<>();
while ((line = gr.readLine()) != null) {
        if (firstTime) {
                firstTime = false;
                continue;
        }
        String[] students = line.split(splitBy);
        newId = Integer.parseInt(students[0]);
        newName = students[1];
        newGender = students[2].charAt(0);
        Student newStudent = new Student(newId, newName, newGender);
        newStudentList.add(newStudent);
}

//Convert arraylist to array.

students = newStudentList.toArray(new Student[0]);

//------ Create student record

ArrayList<StudentRecord> tempStudentRecords = new ArrayList<>();
BufferedReader lr = readCSV("module.csv");
double[] newMarks;
String newCodeCheck;
int newYearCheck;
```

```
    byte newTermCheck;
    firstTime = true;
    while ((line = lr.readLine()) != null) {
        if (firstTime) {
            firstTime = false;
            continue;
        }
        double finalScoreOfStudent;
        String[] studentsRecords = line.split(splitBy);
        String tempMarks = studentsRecords[4];
        newCodeCheck = studentsRecords[1];
        newYearCheck = Integer.parseInt(studentsRecords[2]);
        newTermCheck = convertStringToByte(studentsRecords[3]);
        newMarks = stringToArray(tempMarks);

        // Run the functions created.

        Module newModule = findModule(newYearCheck, newTermCheck, newCodeCheck);
        Student newStudent = findStudent(Integer.parseInt(studentsRecords[0]));
        finalScoreOfStudent = finalScore(newModule, newMarks);
        StudentRecord studentRecords = new StudentRecord(newStudent, newModule, newMarks, finalScoreOfStudent);
        tempStudentRecords.add(studentRecords);
    }

    studentRecords = tempStudentRecords.toArray(new StudentRecord[0]);
    findStudentRecordsForModule();

```

```
		for(Module i:modules){
			i.updateFinalAverageGrade();
		}

		for(StudentRecord i:studentRecords){
			i.updateIsAboveAverage();
		}

		findStudentRecordsForStudent();

		//Output the results of required functions.

		System.out.println(getTotalNumberStudents());
		System.out.println(getBestStudent().getId());
		System.out.println(getBestModule().getCode());

		for(Student i:students){
			System.out.println(i.printTranscript());
		}
	}

	/**
	 * Function that finds the student records for each student.
	 */
	public static void findStudentRecordsForStudent(){
			StudentRecord[] studentRecord;
			for(Student i: students){
				ArrayList<StudentRecord> tempStudentRecord= new ArrayList<>();
				for(StudentRecord j:studentRecords){
```

```
                        if(i.getId() == j.getId())
                            tempStudentRecord.add(j);
                    }
                    studentRecord = tempStudentRecord.toArray(new StudentRecord[0]);
                    i.setRecord(studentRecord);
                    i.updateGpa();
                }
    }

    /**
     * Function that finds the student records for each module.
     */
    public static void findStudentRecordsForModule(){
            StudentRecord[] modulesRecord;
            for(Module i:modules){
                ArrayList<StudentRecord> tempModulesRecord= new ArrayList<>();
                for(StudentRecord j:studentRecords){
                    if(j.getCode().equals(i.getCode()) && j.getTerm() == i.getTerm() &&
j.getYear() == i.getYear())
                        tempModulesRecord.add(j);
                }
                modulesRecord = tempModulesRecord.toArray(new StudentRecord[0]);
                i.setRecords(modulesRecord);
            }
    }

    /**
     * Function that computes the final score.
     * @param newModule - Instance of module for getting the continuous assignment weights.
     * @param marks - Variable that represents the marks obtained for a module.
     * @return The sum of the marks.
     */
    public static double finalScore(Module newModule, double[] marks) {
        double[] tempContinuousAssignmentWeights = newModule.getContinuousAssignmentWeights();
        double sum = 0;
        for(int i=0;i< marks.length;i++){
            sum = marks[i] * tempContinuousAssignmentWeights[i] + sum;
        }
        return sum;
    }

    /**
     * Function that finds the module.
     * @param year - Year of a module.
     * @param term - Term of a module.
     * @param code - Code of a module.
     * @return Null.
     */
    public static Module findModule(int year, byte term, String code){
        for(Module i: modules){
            if(i.getCode().equals(code) && i.getTerm() == term && i.getYear() == year)
                return i;
        }
        return null;
    }

    /**
```

```
 * Function that finds the student.
 * @param id - Id of a student.
 * @return The student with that specific id.
 */
public static Student findStudent(int id){
		for(Student i:students) {
			if(i.getId() == id){
				return i;
			}
		}
		return null;
}

/**
 * Function that finds the module descriptor.
 * @param code - Code of a module.
 * @return The module descriptor with that specific code.
 */
public static ModuleDescriptor findModuleDescriptor(String code){
		for(ModuleDescriptor i:moduleDescriptors){
			if(i.getCode().equals(code)){
				return i;
			}
		}
		return null;
}

/**
 * Convert string to byte.
 * @param str - A string.
 * @return The value of that strings as a byte.
 */
public static byte convertStringToByte(String str){
        return Byte.parseByte(str);
}

/**
 * Parsing a CSV file into BufferedReader class constructor.
 * @param csv - The csv that we are going to read.
 * @return A buffer reader.
 */
public static BufferedReader readCSV(String csv) {
        BufferedReader br = null;
        try {
                br = new BufferedReader(new FileReader(csv));
        } catch (IOException e) {
                e.printStackTrace();
        }
        return br;
}

/**
 * Converts a string to an array.
 * @param strList - A string.
 * @return An array.
 */
public static double[] stringToArray(String strList) {
        strList = strList.replace("[","").replace("]", "");  // Remove [ ]
```

```java
        String[] newList = strList.split(",");  // Make string list
        return Arrays.stream(newList).mapToDouble(Double::parseDouble).toArray();
    }
}

```

StudentRecord.java

```java
public class StudentRecord {

    private Student student;

    private Module module;

    private double[] marks;

    private double finalScore;

    private Boolean isAboveAverage;

    /**
     * Constructor method.
     * @param newStudent - Instance of Student.
     * @param newModule - Instance of Module.
     * @param newMarks - Marks of a student for that module.
     * @param newFinalScore - Final score obtained.
     */
    public StudentRecord(Student newStudent, Module newModule, double[] newMarks, double newFinalScore){
        this.student = newStudent;
        this.module= newModule;
        this.marks = newMarks;
        this.finalScore = newFinalScore;
}

public double getFinalScore() {
        return this.finalScore;
}

public int getId(){
        return this.student.getId();
}

public int getYear(){
        return this.module.getYear();
}

public byte getTerm(){
        return this.module.getTerm();
}

public String getCode(){
        return this.module.getCode();
}

/**
 * Function that updates the truth value of the variable isAboveAverage.
 */
public void updateIsAboveAverage(){
        this.isAboveAverage = this.finalScore > this.module.getFinalAverageGrade();
    }
}

```

Student.java

```
public class Student {

    private int id;

    private String name;

    private char gender;

    private double gpa;

    private StudentRecord[] records;

    /**
     * Transcript function.
     * @return The transcript of a student.
     */
    public  String printTranscript() {
        String id = "ID: " + this.id;
        String name = "Name: " + this.name;
        String gpa = "GPA: " + this.gpa;
        StringBuilder studentRecordsTranscript = new StringBuilder();
        for(StudentRecord i:this.records){
                studentRecordsTranscript.append("| ").append(String.valueOf(i.getYear())).append(" | ")
                        .append(String.valueOf(i.getTerm())).append(" | ").append(i.getCode()).append(" | ")
                        .append(i.getFinalScore()).append(" |\n");
            }
            return "\n\n"+ id + '\n' + name + '\n' + gpa + "\n\n" + studentRecordsTranscript;
    }

    /**
     * Constructor method.
     * @param newId - Id of a student.
     * @param newName - Name of a student.
     * @param newGender - Gender of a student.
     */
    public Student(int newId, String newName, char newGender) {
            this.id = newId;
            this.name = newName;
            this.gender = newGender;
    }

    public int getId(){
            return this.id;
    }

    public double getGpa(){
            return this.gpa;
    }

```

```java
    public void setRecord(StudentRecord[] records) {
        this.records = records;
    }

    /**
     * Function that computes the gpa of a student.
     */
    public void updateGpa(){
        double average = 0;
        double count = 0;
        for(StudentRecord i:this.records){
            average = average + i.getFinalScore();
            count++;
        }
        this.gpa = average/count;
    }

}

```

ModuleDescriptor.java

```java
public class ModuleDescriptor {

    private String code;

    private String name;

    private double[] continuousAssignmentWeights;

    /**
     *
     * @param newCode - Code of a module.
     * @param newName - Name of a module.
     * @param newContinuousAssignmentWeights - Continuous assignment weights for that module.
     */
    public ModuleDescriptor(String newCode, String newName, double[] newContinuousAssignmentWeights) {
        this.code = newCode;
        this.name = newName;
        this.continuousAssignmentWeights = newContinuousAssignmentWeights;
    }

    public double[] getContinuousAssignmentWeights() {
        return this.continuousAssignmentWeights;
    }

    public String getCode() {
        return this.code;
    }


}
```

<u>Module.java</u>

```
public class Module {

    private int year;

    private byte term;

    private ModuleDescriptor module;

    private StudentRecord[] records;

    private double finalAverageGrade;

    /**
     * Constructor method.
     * @param newYear - Year of a module.
     * @param newTerm - Term of a module.
     * @param newModule - The specific module.
     */
    public Module(int newYear, byte newTerm, ModuleDescriptor newModule) {
        this.year = newYear;
        this.term = newTerm;
        this.module = newModule;
    }

    public double[] getContinuousAssignmentWeights() {
        return this.module.getContinuousAssignmentWeights();
    }

    public int getYear(){
        return this.year;
    }

    public byte getTerm(){
        return this.term;
    }

    public String getCode(){
        return this.module.getCode();
    }

    public void setRecords(StudentRecord[] records) {
        this.records = records;
    }

    /**
     * Function that computes the final average grade.
     */
    public void updateFinalAverageGrade(){
        double average = 0;
        double count = 0;
        for(StudentRecord i:this.records){
            average = average + i.getFinalScore();
            count++;
        }
        this.finalAverageGrade = average/count;
    }

    public double getFinalAverageGrade() {
        return this.finalAverageGrade;
    }
}
```

62.