University.java

```
1. import java.io.*;
2. import java.util.ArrayList;
3. import java.util.Arrays;
4.
5. public class University {
6.
7.      private static ModuleDescriptor[] moduleDescriptors;
8.
9.      private static Student[] students;
10.
11.     private static Module[] modules = new Module[]{};
12.
13.     private static StudentRecord[] studentRecords;
14.
15.     /**
16.      * @return The number of students registered in the system.
17.      */
18.     public static int getTotalNumberStudents() {
19.             return students.length;
20.     }
21.
22.     /**
23.      * @return The student with the highest gpa.
24.      */
25.     public static Student getBestStudent() {
26.             double bestStudent = 0;
27.             Student theBestStudent = null;
28.             for(Student i:students){
```

```
29.                         if(i.getGpa() > bestStudent){
30.                                 bestStudent = i.getGpa();
31.                                 theBestStudent = i;
32.                         }
33.                 }
34.                 return theBestStudent;
35.         }
36.
37.         /**
38.          * @return The module with the highest average score.
39.          */
40.         public static Module getBestModule() {
41.                 double bestModule = 0;
42.                 Module theBestModule = null;
43.                 for(Module i:modules){
44.                         if(i.getFinalAverageGrade() > bestModule){
45.                                 bestModule = i.getFinalAverageGrade();
46.                                 theBestModule = i;
47.                         }
48.                 }
49.                 return theBestModule;
50.         }
51.
52.         public static void main(String[] args) throws IOException {
53.                 String name;
54.                 String code;
55.                 double[] weights;
56.
57.                 // ------- Create ModuleDescriptors
```

```
58.
59.             ArrayList<ModuleDescriptor> tempModuleList = new ArrayList<>();
60.             String line = "";
61.             String splitBy = ", ";
62.             BufferedReader br = readCSV("module_descriptors.csv");
63.
64.             boolean firstTime = true;  // Ignore first csv line.
65.             while ((line = br.readLine()) != null) {
66.                     if (firstTime) {
67.                             firstTime = false;
68.                             continue;
69.                     }
70.                     String[] descriptors = line.split(splitBy);
71.                     name = descriptors[0];
72.                     code = descriptors[1];
73.                     String tempWeights = descriptors[2];
74.                     weights = stringToArray(tempWeights);
75.                     ModuleDescriptor newModule = new ModuleDescriptor(code, name, weights);
76.                     tempModuleList.add(newModule);
77.             }
78.
79.             // Convert arraylist to array.
80.
81.             moduleDescriptors = tempModuleList.toArray(new ModuleDescriptor[0]);
82.
83.             // -------- Create modules
84.
85.             ArrayList<Module> tempModules = new ArrayList<>();
86.             String newCode;
```

```
87.             int newYear;
88.             byte newTerm;
89.             BufferedReader tr = readCSV("module.csv");
90.             firstTime = true;
91.             while ((line = tr.readLine()) != null) {
92.                     if (firstTime) {
93.                             firstTime = false;
94.                             continue;
95.                     }
96.
97.                     String[] module = line.split(splitBy);
98.                     newYear = Integer.parseInt(module[2]);
99.                     newTerm = convertStringToByte(module[3]);
100.                    newCode = module[1];
101.                    boolean exist = false;
102.                    for(Module i: tempModules){
103.                            if(newCode.equals(i.getCode()) && newTerm == i.getTerm() &&
newYear == i.getYear())
104.                                    exist = true;
105.                    }
106.                    if(exist)
107.                            continue;
108.                    Module newModule = new Module(newYear, newTerm,
findModuleDescriptor(newCode));
109.                    tempModules.add(newModule);
110.                    modules = tempModules.toArray(new Module[0]);
111.            }
112.
113.            int newId;
```

```
114.            String newName;
115.            char newGender;
116.            BufferedReader gr = readCSV("students.csv");
117.            firstTime = true;
118.            ArrayList<Student> newStudentList = new ArrayList<>();
119.            while ((line = gr.readLine()) != null) {
120.                    if (firstTime) {
121.                            firstTime = false;
122.                            continue;
123.                    }
124.                    String[] students = line.split(splitBy);
125.                    newId = Integer.parseInt(students[0]);
126.                    newName = students[1];
127.                    newGender = students[2].charAt(0);
128.                    Student newStudent = new Student(newId, newName, newGender);
129.                    newStudentList.add(newStudent);
130.            }
131.
132.            //Convert arraylist to array.
133.
134.            students = newStudentList.toArray(new Student[0]);
135.
136.            //------ Create student record
137.
138.            ArrayList<StudentRecord> tempStudentRecords = new ArrayList<>();
139.            BufferedReader lr = readCSV("module.csv");
140.            double[] newMarks;
141.            String newCodeCheck;
142.            int newYearCheck;
```

```
143.            byte newTermCheck;
144.            firstTime = true;
145.            while ((line = lr.readLine()) != null) {
146.                    if (firstTime) {
147.                            firstTime = false;
148.                            continue;
149.                    }
150.                    double finalScoreOfStudent;
151.                    String[] studentsRecords = line.split(splitBy);
152.                    String tempMarks = studentsRecords[4];
153.                    newCodeCheck = studentsRecords[1];
154.                    newYearCheck = Integer.parseInt(studentsRecords[2]);
155.                    newTermCheck = convertStringToByte(studentsRecords[3]);
156.                    newMarks = stringToArray(tempMarks);
157.
158.                    // Run the functions created.
159.
160.                    Module newModule = findModule(newYearCheck, newTermCheck,
newCodeCheck);
161.                    Student newStudent = findStudent(Integer.parseInt(studentsRecords[0]));
162.                    finalScoreOfStudent = finalScore(newModule, newMarks);
163.                    StudentRecord studentRecords = new StudentRecord(newStudent, newModule,
newMarks, finalScoreOfStudent);
164.                    tempStudentRecords.add(studentRecords);
165.            }
166.
167.            studentRecords = tempStudentRecords.toArray(new StudentRecord[0]);
168.            findStudentRecordsForModule();
169.
```

```
170.            for(Module i:modules){
171.                    i.updateFinalAverageGrade();
172.            }
173.
174.            for(StudentRecord i:studentRecords){
175.                    i.updateIsAboveAverage();
176.            }
177.
178.            findStudentRecordsForStudent();
179.
180.            //Output the results of required functions.
181.
182.            System.out.println(getTotalNumberStudents());
183.            System.out.println(getBestStudent().getId());
184.            System.out.println(getBestModule().getCode());
185.
186.            for(Student i:students){
187.                    System.out.println(i.printTranscript());
188.            }
189.    }
190.
191.    /**
192.     * Function that finds the student records for each student.
193.     */
194.    public static void findStudentRecordsForStudent(){
195.                    StudentRecord[] studentRecord;
196.                    for(Student i: students){
197.                            ArrayList<StudentRecord> tempStudentRecord= new ArrayList<>();
198.                            for(StudentRecord j:studentRecords){
```

```
199.                                        if(i.getId() == j.getId())
200.                                                tempStudentRecord.add(j);
201.                                }
202.                                studentRecord = tempStudentRecord.toArray(new StudentRecord[0]);
203.                                i.setRecord(studentRecord);
204.                                i.updateGpa();
205.                        }
206.        }
207.
208.        /**
209.         * Function that finds the student records for each module.
210.         */
211.        public static void findStudentRecordsForModule(){
212.                StudentRecord[] modulesRecord;
213.                for(Module i:modules){
214.                        ArrayList<StudentRecord> tempModulesRecord= new ArrayList<>();
215.                        for(StudentRecord j:studentRecords){
216.                                if(j.getCode().equals(i.getCode()) && j.getTerm() == i.getTerm() &&
j.getYear() == i.getYear())
217.                                        tempModulesRecord.add(j);
218.                        }
219.                        modulesRecord = tempModulesRecord.toArray(new StudentRecord[0]);
220.                        i.setRecords(modulesRecord);
221.                }
222.        }
223.
224.        /**
225.         * Function that computes the final score.
226.         * @param newModule - Instance of module for getting the continuous assignment weights.
```

```
227.      * @param marks - Variable that represents the marks obtained for a module.
228.      * @return The sum of the marks.
229.      */
230.     public static double finalScore(Module newModule, double[] marks) {
231.             double[] tempContinuousAssignmentWeights =
newModule.getContinuousAssignmentWeights();
232.             double sum = 0;
233.             for(int i=0;i< marks.length;i++){
234.                     sum = marks[i] * tempContinuousAssignmentWeights[i] + sum;
235.             }
236.             return sum;
237.     }
238.
239.     /**
240.      * Function that finds the module.
241.      * @param year - Year of a module.
242.      * @param term - Term of a module.
243.      * @param code - Code of a module.
244.      * @return Null.
245.      */
246.     public static Module findModule(int year, byte term, String code){
247.             for(Module i: modules){
248.                     if(i.getCode().equals(code) && i.getTerm() == term && i.getYear() == year)
249.                             return i;
250.             }
251.             return null;
252.     }
253.
254.     /**
```

```
255.     * Function that finds the student.
256.     * @param id - Id of a student.
257.     * @return The student with that specific id.
258.     */
259.    public static Student findStudent(int id){
260.            for(Student i:students) {
261.                    if(i.getId() == id){
262.                            return i;
263.                    }
264.            }
265.            return null;
266.    }
267.
268.    /**
269.     * Function that finds the module descriptor.
270.     * @param code - Code of a module.
271.     * @return The module descriptor with that specific code.
272.     */
273.    public static ModuleDescriptor findModuleDescriptor(String code){
274.            for(ModuleDescriptor i:moduleDescriptors){
275.                    if(i.getCode().equals(code)){
276.                            return i;
277.                    }
278.            }
279.            return null;
280.    }
281.
282.    /**
283.     * Convert string to byte.
```

```
284.      * @param str - A string.
285.      * @return The value of that strings as a byte.
286.      */
287.     public static byte convertStringToByte(String str){
288.               return Byte.parseByte(str);
289.     }
290.
291.     /**
292.      * Parsing a CSV file into BufferedReader class constructor.
293.      * @param csv - The csv that we are going to read.
294.      * @return A buffer reader.
295.      */
296.     public static BufferedReader readCSV(String csv) {
297.               BufferedReader br = null;
298.               try {
299.                         br = new BufferedReader(new FileReader(csv));
300.               } catch (IOException e) {
301.                         e.printStackTrace();
302.               }
303.               return br;
304.     }
305.
306.     /**
307.      * Converts a string to an array.
308.      * @param strList - A string.
309.      * @return An array.
310.      */
311.     public static double[] stringToArray(String strList) {
312.               strList = strList.replace("[","").replace("]", "");  // Remove [ ]
```

```
313.                String[] newList = strList.split(",");  // Make string list
314.                return Arrays.stream(newList).mapToDouble(Double::parseDouble).toArray();
315.    }
316. }
317.
```

StudentRecord.java

```
1. public class StudentRecord {
2.
3.      private Student student;
4.
5.      private Module module;
6.
7.      private double[] marks;
8.
9.      private double finalScore;
10.
11.     private Boolean isAboveAverage;
12.
13.     /**
14.      * Constructor method.
15.      * @param newStudent - Instance of Student.
16.      * @param newModule - Instance of Module.
17.      * @param newMarks - Marks of a student for that module.
18.      * @param newFinalScore - Final score obtained.
19.      */
20.     public StudentRecord(Student newStudent, Module newModule, double[] newMarks, double
newFinalScore){
21.             this.student = newStudent;
```

```
22.             this.module= newModule;
23.             this.marks = newMarks;
24.             this.finalScore = newFinalScore;
25.     }
26.
27.     public double getFinalScore() {
28.             return this.finalScore;
29.     }
30.
31.     public int getId(){
32.             return this.student.getId();
33.     }
34.
35.     public int getYear(){
36.             return this.module.getYear();
37.     }
38.
39.     public byte getTerm(){
40.             return this.module.getTerm();
41.     }
42.
43.     public String getCode(){
44.             return this.module.getCode();
45.     }
46.
47.     /**
48.      * Function that updates the truth value of the variable isAboveAverage.
49.      */
50.     public void updateIsAboveAverage(){
```

```
51.                 this.isAboveAverage = this.finalScore > this.module.getFinalAverageGrade();
52.         }
53. }
54.
```

Student.java

```
1. public class Student {
2.
3.          private int id;
4.
5.          private String name;
6.
7.          private char gender;
8.
9.          private double gpa;
10.
11.         private StudentRecord[] records;
12.
13.         /**
14.          * Transcript function.
15.          * @return The transcript of a student.
16.          */
17.         public  String printTranscript() {
18.                 String id = "ID: " + this.id;
19.                 String name = "Name: " + this.name;
20.                 String gpa = "GPA: " + this.gpa;
21.                 StringBuilder studentRecordsTranscript = new StringBuilder();
22.                 for(StudentRecord i:this.records){
```

```
23.                         studentRecordsTranscript.append("|
").append(String.valueOf(i.getYear())).append(" | ")
24.                                         .append(String.valueOf(i.getTerm())).append(" |
").append(i.getCode()).append(" | ")
25.                                         .append(i.getFinalScore()).append(" |\n");
26.             }
27.             return "\n\n"+ id + '\n' + name + '\n' + gpa + "\n\n" + studentRecordsTranscript;
28.     }
29.
30.     /**
31.      * Constructor method.
32.      * @param newId - Id of a student.
33.      * @param newName - Name of a student.
34.      * @param newGender - Gender of a student.
35.      */
36.     public Student(int newId, String newName, char newGender) {
37.             this.id = newId;
38.             this.name = newName;
39.             this.gender = newGender;
40.     }
41.
42.     public int getId(){
43.             return this.id;
44.     }
45.
46.     public double getGpa(){
47.             return this.gpa;
48.     }
49.
```

```
50.     public void setRecord(StudentRecord[] records) {
51.             this.records = records;
52.     }
53.
54.     /**
55.      * Function that computes the gpa of a student.
56.      */
57.     public void updateGpa(){
58.             double average = 0;
59.             double count = 0;
60.             for(StudentRecord i:this.records){
61.                     average = average + i.getFinalScore();
62.                     count++;
63.             }
64.             this.gpa = average/count;
65.     }
66.
67. }
68.
```

ModuleDescriptor.java

```
1. public class ModuleDescriptor {
2.
3.     private String code;
4.
5.     private String name;
6.
7.     private double[] continuousAssignmentWeights;
8.
```

```
9.      /**
10.      *
11.      * @param newCode - Code of a module.
12.      * @param newName - Name of a module.
13.      * @param newContinuousAssignmentWeights - Continuous assignment weights for that
module.
14.      */
15.     public ModuleDescriptor(String newCode, String newName, double[]
newContinuousAssignmentWeights) {
16.             this.code = newCode;
17.             this.name = newName;
18.             this.continuousAssignmentWeights = newContinuousAssignmentWeights;
19.     }
20.
21.     public double[] getContinuousAssignmentWeights() {
22.             return this.continuousAssignmentWeights;
23.     }
24.
25.     public String getCode() {
26.             return this.code;
27.     }
28.
29.
30. }
```

Module.java

```
1. public class Module {
2.
3.      private int year;
```

```
4.
5.      private byte term;
6.
7.      private ModuleDescriptor module;
8.
9.      private StudentRecord[] records;
10.
11.     private double finalAverageGrade;
12.
13.     /**
14.      * Constructor method.
15.      * @param newYear - Year of a module.
16.      * @param newTerm - Term of a module.
17.      * @param newModule - The specific module.
18.      */
19.     public Module(int newYear, byte newTerm, ModuleDescriptor newModule) {
20.             this.year = newYear;
21.             this.term = newTerm;
22.             this.module = newModule;
23.     }
24.
25.     public double[] getContinuousAssignmentWeights() {
26.             return this.module.getContinuousAssignmentWeights();
27.     }
28.
29.     public int getYear(){
30.             return this.year;
31.     }
32.
```

```
33.     public byte getTerm(){
34.             return this.term;
35.     }
36.
37.     public String getCode(){
38.             return this.module.getCode();
39.     }
40.
41.     public void setRecords(StudentRecord[] records) {
42.             this.records = records;
43.     }
44.
45.     /**
46.      * Function that computes the final average grade.
47.      */
48.     public void updateFinalAverageGrade(){
49.             double average = 0;
50.             double count = 0;
51.             for(StudentRecord i:this.records){
52.                     average = average + i.getFinalScore();
53.                     count++;
54.             }
55.             this.finalAverageGrade = average/count;
56.     }
57.
58.     public double getFinalAverageGrade() {
59.             return this.finalAverageGrade;
60.     }
61. }
```

62.