1 ModuleDescriptor.java

```java
// The ModuleDescriptor class:
public class ModuleDescriptor {

   // 3 instance attributes
   private String code;
   private String name;
   private double[] continuousAssignmentWeights;

   // Methods

   //checks that the weights of the assesments are valid
   public boolean checkWeights(double[] continuousAssignmentWeights) {
      double totalWeights = 0.0;
      for(int i = 0; i< continuousAssignmentWeights.length; i++){
         if (continuousAssignmentWeights[i] > 0.0) {
            // Check weights are not negative.
            totalWeights += continuousAssignmentWeights[i];
         }
      }
      if (totalWeights == 1.0) {
         // Check weights sum up to 1.
         return true;
      } else {
         return false;
      }
   }

   // turns weights into string for printing
   public String weightsToString() {
      String weightString = "[";
      for (int i = 0; i < continuousAssignmentWeights.length; i++) {
            weightString += continuousAssignmentWeights[i];
            if (i < (continuousAssignmentWeights.length - 1)) {
               weightString += ", ";
            }
      }
      weightString += "]";
      return weightString;
   }

   // toString Method:
   public String toString(){
     return "ModuleDescriptor[code="+code+",name="+name+",CAWeights="+weightsToString()+"]";
   }

   // Getters
   public String getCode() {
      return code;
   }

   public String getName() {
      return name;
```

```
53    }
54
55    public double[] getContinuousAssignmentWeights() {
56       return continuousAssignmentWeights;
57    }
58
59    // Constructor
60    public ModuleDescriptor(String name, String code, double[] continuousAssignmentWeights) {
61       if (code != "") {
62          // Check code isn't null.
63          this.code = code;
64       }
65       if (name != "") {
66          // Check name isn't null.
67          this.name = name;
68       }
69       // check weights
70       boolean weightsFine = checkWeights(continuousAssignmentWeights);
71       if (weightsFine == true) {
72          this.continuousAssignmentWeights = continuousAssignmentWeights;
73          }
74    }
75
76 }
```

2 Student.java

```
1  // The Student class:
2  public class Student {
3
4     // 6 instance attributes
5     private int id;
6     private String name;
7     private char gender = ' ';
8     private double gpa = 0.0;
9     private StudentRecord[] records;
10    private int numberOfModules = 0;
11
12
13    // Methods
14
15    // prints transcript using the format outlined within the requirements in section 1 of the CA
16    public String printTranscript() {
17       System.out.println("    University of Knowledge - Official Transcript");
18       System.out.println("");
19       System.out.println("");
20       System.out.println("ID: " + id);
21       System.out.println("Name: " + name);
22       System.out.println("GPA: " + gpa);
23       System.out.println("");
24       for (int i = 0; i < records.length; i++) {
25            System.out.println("| " +records[i].getModule().getYear()+" |
                 "+records[i].getModule().getTerm()+" |
                 "+records[i].getModule().getModuleDescriptor().getCode()+" | "+records[i].getFinalScore()+"
```

```
                |");
26            if ((i+1) < records.length) {
27               if (records[i].getModule().getTerm() < records[i+1].getModule().getTerm()) {
28                  System.out.println("");
29               } else if (records[i].getModule().getYear() < records[i+1].getModule().getYear()) {
30                  System.out.println("");
31               }
32            }
33      }
34      return "";
35   }
36
37   // checks gender is one of the valid options to be inputted
38   public void setGender(char gender) {
39      if (gender == 'F') {
40         this.gender = gender;
41      } else if (gender == 'M') {
42         this.gender = gender;
43      } else if (gender == 'X') {
44         this.gender = gender;
45      }
46   }
47
48   // adds student record
49   public void addRecord(StudentRecord record) {
50      numberOfModules = records.length + 1;
51      StudentRecord[] records2;
52      records2 = new StudentRecord[numberOfModules];
53      for (int i = 0; i < records.length; i++) {
54            records2[i] = records[i];
55      }
56      records2[numberOfModules - 1] = record;
57      records = new StudentRecord[numberOfModules];
58      records = records2;
59      calculateGpa();
60   }
61
62   // prints records (for testing)
63   public void printRecords() {
64      for (int i = 0; i < records.length; i++) {
65            System.out.println(records[i]);
66      }
67   }
68
69   // calculates gpa by finding the average grade
70   public void calculateGpa() {
71      double totalScore = 0.0;
72      for (int i = 0; i < records.length; i++) {
73            totalScore += records[i].getFinalScore();
74      }
75      this.gpa = totalScore / records.length;
76   }
77
78
79   // Getters
```

```
80     public double getGpa() {
81        return gpa;
82     }
83
84     public int getId() {
85        return id;
86     }
87
88     public String getName() {
89        return name;
90     }
91
92     // toString Method:
93     public String toString(){
94       return "Student[id="+id+",name="+name+",gender="+gender+",gpa="+gpa+"]";
95     }
96
97     // Constructor
98     public Student(int id, String name, char gender) {
99        this.id = id;
100       this.name = name;
101       setGender(gender);
102       records = new StudentRecord[0];
103    }
104 }
```

3 StudentRecord.java

```
1  // The StudentRecord class:
2  public class StudentRecord {
3
4     // 5 instance attributes
5     private Student student;
6     private Module module;
7     private double[] marks;
8     private double finalScore;
9     private Boolean isAboveAverage;
10
11    // Methods
12
13    // calculates finalScore by multiplying marks by weights
14    public double calculateFinalScore() {
15       double[] continuousAssignmentWeights;
16       continuousAssignmentWeights = module.getWeights();
17       for (int i = 0; i < continuousAssignmentWeights.length; i++) {
18            finalScore += (continuousAssignmentWeights[i] * marks[i]);
19       }
20       return finalScore;
21    }
22
23    // calculates if final score is greater than the avarage for the class
24    public boolean calculateifAboveAverage() {
25       if (finalScore > module.getFinalAverageGrade()) {
26          return true;
```

```
27        } else {
28           return false;
29        }
30     }
31
32     // Getters
33     public double getFinalScore() {
34        return finalScore;
35     }
36
37     public boolean getIsAboveAverage() {
38        return isAboveAverage;
39     }
40
41     public Module getModule() {
42        return module;
43     }
44
45     // toString Method:
46     public String toString(){
47       return
           "StudentRecord[student="+student+",module="+module+",finalScore="+finalScore+",isAboveAverage="+isAboveAverag
48     }
49
50     // Constructor
51     public StudentRecord(Student student, Module module, double[] marks) {
52        this.student = student;
53        this.module = module;
54        this.marks = marks;
55        this.finalScore = calculateFinalScore();
56        this.isAboveAverage = calculateifAboveAverage();
57     }
58
59  }
```

4 Module.java

```
1  // The Module class:
2  public class Module {
3
4     // 6 instance attributes
5     private int year;
6     private byte term;
7     private ModuleDescriptor module;
8     private StudentRecord[] records;
9     private double finalAverageGrade = 0.0;
10    private int numberOfRecords = 0;
11
12    // Methods
13
14    // calculates finalAverageGrade by finding the sum of grades in the module and dividing by number og
          records
15    public double setFinalAverageGrade() {
16       double totalScore = 0.0;
```

```
17        for (int i = 0; i < records.length; i++) {
18              totalScore += records[i].getFinalScore();
19        }
20        return totalScore / records.length;
21     }
22
23     // adds record
24     public void addRecord(StudentRecord record) {
25        numberOfRecords = records.length + 1;
26        StudentRecord[] records2;
27        records2 = new StudentRecord[numberOfRecords];
28        for (int i = 0; i < records.length; i++) {
29              records2[i] = records[i];
30        }
31        records2[numberOfRecords - 1] = record;
32        records = new StudentRecord[numberOfRecords];
33        records = records2;
34        this.finalAverageGrade = setFinalAverageGrade();
35     }
36
37     // prints records (for testing)
38     public void printRecords() {
39        for (int i = 0; i < records.length; i++) {
40              System.out.println(records[i]);
41        }
42     }
43
44     // Getters
45     public double[] getWeights() {
46        return module.getContinuousAssignmentWeights();
47     }
48
49     public double getFinalAverageGrade() {
50        return finalAverageGrade;
51     }
52
53     public int getYear() {
54        return year;
55     }
56
57     public byte getTerm() {
58        return term;
59     }
60
61     public ModuleDescriptor getModuleDescriptor() {
62        return module;
63     }
64
65     // toString Method:
66     public String toString(){
67       return
          "Module[year="+year+",term="+term+",ModuleDescriptor="+module+",finalAverageGrade="+finalAverageGrade+"]";
68     }
69
70     // Constructor
```

```
71     public Module(int year, byte term, ModuleDescriptor module) {
72        this.year = year;
73        this.term = term;
74        this.module = module;
75        records = new StudentRecord[0];
76     }
77
78  }
```

5 University.java

```
1   public class University {
2
3      // 3 instance attributes
4      private ModuleDescriptor[] moduleDescriptors;
5      private Student[] students;
6      private Module[] modules;
7
8      //returns number of students in the university
9      public int getTotalNumberStudents() {
10        return students.length;
11     }
12
13     //returns student with the highest gpa
14     public Student getBestStudent() {
15        double highestGpa = 0.0;
16        int bestStudent = 0;
17        for (int i = 0; i < students.length; i++) {
18           double score = students[i].getGpa();
19           if (score > highestGpa) {
20              highestGpa = score;
21              bestStudent = i;
22           }
23        }
24        return students[bestStudent];
25     }
26
27     //returns module with the highest average grade
28     public Module getBestModule() {
29        double highestScore = 0.0;
30        int bestModule = 0;
31        for (int i = 0; i < modules.length; i++) {
32           double score = modules[i].getFinalAverageGrade();
33           if (score > highestScore) {
34              highestScore = score;
35              bestModule = i;
36           }
37        }
38        return modules[bestModule];
39     }
40
41     // for testing purposes
42     public void printModuleDescriptors() {
43        for (int i = 0; i < moduleDescriptors.length; i++) {
```

```
44          System.out.println(moduleDescriptors[i]);
45          }
46      }
47
48      // for testing purposes
49      public void printStudents() {
50          for (int i = 0; i < students.length; i++) {
51          System.out.println(students[i]);
52          }
53      }
54
55      // for testing purposes
56      public void printModules() {
57          for (int i = 0; i < modules.length; i++) {
58          System.out.println(modules[i]);
59          }
60      }
61
62      // adds student record to module and student
63      public void addStudentRecord(Student student, Module module, double[] marks) {
64          StudentRecord record;
65          record = new StudentRecord(student, module, marks);
66          student.addRecord(record);
67          module.addRecord(record);
68      }
69
70      // adds all student records
71      public void addRecords() {
72          addStudentRecord(students[0], modules[0], new double[]{9,10,10,10});
73          addStudentRecord(students[1], modules[0], new double[]{8,8,8,9});
74          addStudentRecord(students[2], modules[0], new double[]{5,5,6,5});
75          addStudentRecord(students[3], modules[0], new double[]{6,4,7,9});
76          addStudentRecord(students[4], modules[0], new double[]{10,9,10,9});
77          addStudentRecord(students[5], modules[1], new double[]{9,9});
78          addStudentRecord(students[6], modules[1], new double[]{6,9});
79          addStudentRecord(students[7], modules[1], new double[]{5,6});
80          addStudentRecord(students[8], modules[1], new double[]{9,7});
81          addStudentRecord(students[9], modules[1], new double[]{8,5});
82          addStudentRecord(students[0], modules[2], new double[]{10,10,9.5,10});
83          addStudentRecord(students[1], modules[2], new double[]{7,8.5,8.2,8});
84          addStudentRecord(students[2], modules[2], new double[]{6.5,7.0,5.5,8.5});
85          addStudentRecord(students[3], modules[2], new double[]{5.5,5,6.5,7});
86          addStudentRecord(students[4], modules[2], new double[]{7,5,8,6});
87          addStudentRecord(students[5], modules[3], new double[]{9,10,10,10});
88          addStudentRecord(students[6], modules[3], new double[]{8,8,8,9});
89          addStudentRecord(students[7], modules[3], new double[]{5,5,6,5});
90          addStudentRecord(students[8], modules[3], new double[]{6,4,7,9});
91          addStudentRecord(students[9], modules[3], new double[]{10,9,8,9});
92          addStudentRecord(students[0], modules[4], new double[]{10,10,10});
93          addStudentRecord(students[1], modules[4], new double[]{8,7.5,7.5});
94          addStudentRecord(students[2], modules[4], new double[]{9,7,7});
95          addStudentRecord(students[3], modules[4], new double[]{9,8,7});
96          addStudentRecord(students[4], modules[4], new double[]{2,7,7});
97          addStudentRecord(students[5], modules[4], new double[]{10,10,10});
98          addStudentRecord(students[6], modules[4], new double[]{8,7.5,7.5});
```

```
        addStudentRecord(students[7], modules[4], new double[]{10,10,10});
        addStudentRecord(students[8], modules[4], new double[]{9,8,7});
        addStudentRecord(students[9], modules[4], new double[]{8,9,10});
        addStudentRecord(students[0], modules[5], new double[]{10,9,10});
        addStudentRecord(students[1], modules[5], new double[]{8.5,9,7.5});
        addStudentRecord(students[2], modules[5], new double[]{10,10,5.5});
        addStudentRecord(students[3], modules[5], new double[]{7,7,7});
        addStudentRecord(students[4], modules[5], new double[]{5,6,10});
        addStudentRecord(students[5], modules[6], new double[]{8,9,8});
        addStudentRecord(students[6], modules[6], new double[]{6.5,9,9.5});
        addStudentRecord(students[7], modules[6], new double[]{8.5,10,8.5});
        addStudentRecord(students[8], modules[6], new double[]{7.5,8,10});
        addStudentRecord(students[9], modules[6], new double[]{10,6,10});
    }

    // Constructor
    public University(ModuleDescriptor[] moduleDescriptors, Student[] students, Module[] modules) {
        this.moduleDescriptors = moduleDescriptors;
        this.students = students;
        this.modules = modules;
    }

    public static void main(String[] args) {
        //data from csv files
        ModuleDescriptor[] setModuleDescriptors;
        setModuleDescriptors = new ModuleDescriptor[6];
        setModuleDescriptors[0] = new ModuleDescriptor("Real World Mathematics", "ECM0002", new
            double[]{0.1,0.3,0.6});
        setModuleDescriptors[1] = new ModuleDescriptor("Programming", "ECM1400", new
            double[]{0.25,0.25,0.25,0.25});
        setModuleDescriptors[2] = new ModuleDescriptor("Data Structures", "ECM1406", new
            double[]{0.25,0.25,0.5});
        setModuleDescriptors[3] = new ModuleDescriptor("Object-Oriented Programming", "ECM1410", new
            double[]{0.2,0.3,0.5});
        setModuleDescriptors[4] = new ModuleDescriptor("Information Systems", "BEM2027", new
            double[]{0.1,0.3,0.3,0.3});
        setModuleDescriptors[5] = new ModuleDescriptor("Thermal Physics", "PHY2023", new double[]{0.4,0.6});
        Student[] setStudents;
        setStudents = new Student[10];
        setStudents[0] = new Student(1000, "Ana", 'F');
        setStudents[1] = new Student(1001, "Oliver", 'M');
        setStudents[2] = new Student(1002, "Mary", 'F');
        setStudents[3] = new Student(1003, "John", 'M');
        setStudents[4] = new Student(1004, "Noah", 'M');
        setStudents[5] = new Student(1005, "Chico", 'M');
        setStudents[6] = new Student(1006, "Maria", 'F');
        setStudents[7] = new Student(1007, "Mark", 'X');
        setStudents[8] = new Student(1008, "Lia", 'F');
        setStudents[9] = new Student(1009, "Rachel", 'F');
        Module[] setModules;
        setModules = new Module[7];
        setModules[0] = new Module(2019, (byte)1, setModuleDescriptors[1]);
        setModules[1] = new Module(2019, (byte)1, setModuleDescriptors[5]);
        setModules[2] = new Module(2019, (byte)2, setModuleDescriptors[4]);
        setModules[3] = new Module(2019, (byte)2, setModuleDescriptors[1]);
```

```
149         setModules[4] = new Module(2020, (byte)1, setModuleDescriptors[2]);
150         setModules[5] = new Module(2020, (byte)1, setModuleDescriptors[3]);
151         setModules[6] = new Module(2020, (byte)2, setModuleDescriptors[0]);
152         // create instance of university called uok (University of Knowledge)
153         University uok;
154         uok = new University(setModuleDescriptors, setStudents, setModules);
155         uok.addRecords();
156 
157         //just for me to test data was entered correctly
158         //uok.printModuleDescriptors();
159         //uok.printStudents();
160         //uok.printModules();
161 
162         //find best module
163         System.out.println("The best module is: ");
164         System.out.println();
165         System.out.println(uok.getBestModule());
166         System.out.println();
167 
168         //find number of students
169         System.out.println("The number of students at the University are: ");
170         System.out.println();
171         System.out.println(uok.getTotalNumberStudents());
172         System.out.println();
173 
174         //find best student
175         System.out.println("The best of these students is: ");
176         System.out.println();
177         Student theBestStudent;
178         theBestStudent = uok.getBestStudent();
179         System.out.println(theBestStudent);
180         System.out.println();
181         theBestStudent.printTranscript();
182 
183 
184     }
185 }
```