1 ModuleDescriptor.java

```java
import java.util.Arrays;
public class ModuleDescriptor {

   private String code;

   private String name;

   private double[] continuousAssignmentWeights;

   //Constructor for a module descriptor
   public ModuleDescriptor (String code, String name, double[] continuousAssignmentWeights) {
      //check if the code is null
      if (code == null) {
         System.err.println("Module descriptor code cannot be null ");
         System.exit(1);
      }
      //check if the name is null
      if (name == null) {
         System.err.println("Module descriptor name cannot be null ");
         System.exit(1);
      }
      this.code = code;
      this.name = name;

      //Check that the continuous assignment weights sum up to 1
      double sum = 0;
      for (double i: continuousAssignmentWeights) {
         //Check that each individual continuous assignment weight is non-negative
         if (i < 0) {
            System.err.println("A continuous assignment weight of a module descriptor with the code " + code
                + " is non-negative.");
            System.exit(1);
         } // no else necessary since program will stop if an error occurs, if not then it will continue as
             normal
         sum += i;
      }
      if (Math.abs(sum - 1) > 0.00001) {
         System.err.println("The continuous assignment weights of a module descriptor with the code " + code
             + " is " + sum + ". The continuous assignment weights must sum up to 1.");
         System.exit(1);
      } // no else necessary since program will stop if an error occurs, if not then it will continue as
          normal
      this.continuousAssignmentWeights = continuousAssignmentWeights;
   }

   //methods:
   //method: get the code of a module
   public String getCode(){
      return code;
   }
   //method: get the name of a module
   public String getName() {
```

```
49        return name;
50     }
51     //method: get the continuous assignment weights of a module
52     public double[] getContinuousAssignmentWeights(){
53        System.out.println(Arrays.toString(continuousAssignmentWeights));
54        return continuousAssignmentWeights;
55     }
56     //method: to string method
57     public String toString() {return "Module descriptor[code= " + code + ", name = " + name + ", continuous
           assignment weights= " + Arrays.toString(continuousAssignmentWeights) + "]";}
58
59  }
```

2 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     //Each students id is checked if it is unique in the university constructor
14
15     //2 Overloaded constructors, 1 being the student before being enrolled to modules, 1 afterwards
16     //Constructor of a new student, therefore they have no GPA or StudentRecord
17     public Student (int id, String name, char gender){
18        //if id is null it throws an error in compilation so no check is necessary
19        this.id = id;
20
21        //check if the name is null
22        if (name == null) {
23           System.err.println("Name input cannot be null ");
24           System.exit(1);
25        }
26        // no else necessary since program will stop if an error occurs, if not then it will continue as normal
27        this.name = name;
28
29        //Check if the gender is either 'M', 'F' or X
30        //If CompareM == 0 then the two values that were compared are equal
31        int compareM = Character.compare('M', gender);
32        int compareF = Character.compare('F', gender);
33        int compareX = Character.compare('X', gender);
34        //check if gender is one of these types
35        if (compareM == 0 || compareF == 0 || compareX == 0) {
36           this.gender = gender;
37        } else {
38
39           // Print a msg describing the problem.
```

```
40          System.err.println("You input gender " + gender + " but the gender of a student must be F, M or
                X.");
41          // This command will stop the execution of the program.
42          System.exit(1);
43       }
44       }
45
46
47    //constructor of a student with an array of student records, gpa calculated using student records
48    public Student (int id, String name, char gender, StudentRecord[] records){
49       //if id is null it throws an error in compilation so no check is necessary
50       this.id = id;
51
52       //check if the name is null
53       if (name == null) {
54          System.err.println("Name input cannot be null ");
55          System.exit(1);
56       }
57       // no else necessary since program will stop if an error occurs, if not then it will continue as normal
58       this.name = name;
59
60       //Check if the gender is either 'M', 'F' or X
61       //If CompareM == 0 then the two values that were compared are equal
62       int compareM = Character.compare('M', gender);
63       int compareF = Character.compare('F', gender);
64       int compareX = Character.compare('X', gender);
65       //check if gender is one of these types
66       if (compareM == 0 || compareF == 0 || compareX == 0) {
67          this.gender = gender;
68       } else {
69          System.err.println("You input gender " + gender + " but the gender of a student must be F, M or
                X.");
70          System.exit(1);
71       }
72
73       //calculate the gpa of a student
74       double gpa = 0;
75       double counter = 0;
76       //Sum up the values of all the final scores in a student record
77       for (StudentRecord i: records) {
78          //Check if the final score of a student in each student record is between 0 and 100
79          if (0 > i.getFinalScore() || i.getFinalScore() > 100){
80             System.err.println("You input final module score " + i.getFinalScore() + " but the finalscore of
                   a student in a module must be between 0 and 100.");
81             System.exit(1);
82          }
83          gpa += i.getFinalScore();
84          counter++;
85       }
86       //Divide the sum of the final scores by the number of student records to give the gpa
87       gpa = gpa/counter;
88       this.gpa = gpa;
89
90       //Check if any student has 2 records for a single module
91       int temp1 = 0;//used as a counter
```

```
92        for (StudentRecord a: records) {
93           if (temp1 == 0) {//stores the 1st student record in the student record array as a
94              for (StudentRecord b: records) {
95                 temp1 ++;
96                 if (temp1 == 2) {//stores the 2nd student record in the student record array as b
97                    for (StudentRecord c: records) {
98                       temp1 ++;
99                       if (temp1 == 5){//stores the 3rd student record in the student record array as c
100                         for (StudentRecord d: records) {
101                            temp1 ++;
102                            if (temp1 == 9) {//stores the 4th student record in the student record array as d
103                               //Compare the memory referenece of each module to each other module
104                               if (a.getModule() == b.getModule() || a.getModule() == c.getModule() ||
                                      a.getModule() == d.getModule() || b.getModule() == c.getModule() ||
                                      b.getModule() == d.getModule() || c.getModule() == d.getModule()) {
105                                  System.err.println("The student with the ID " + id + " has 2 records for a
                                         single module.");
106                                  System.exit(1);
107                               }
108                            }
109                         }
110                      }
111                   }
112                }
113             }
114          }
115       }
116       this.records = records;
117    }
118       //methods:
119       //method: get the name of a student
120       public String getName (){
121          return name;
122       }
123       //method: get the id of a student
124       public int getId (){
125          return id;
126       }
127       //method: get the gender of a student
128       public char getGender(){
129          return gender;
130       }
131
132       //method: get the GPA of a student
133       public double getGPA() {
134          return gpa;
135       }
136
137       //method: set the GPA of a student
138       public void setGPA(double gpa) {
139          this.gpa = gpa;
140       }
141
142       //method: set the GPA of a student
143       public StudentRecord[] getRecords(){
```

```
144         return records;
145     }
146     //method: to string method
147     public String toString(){
148         return "Student[id = " + id + ", name = " + name + ", gender = " + gender + ", gpa = " + gpa + ",
                student records = " + records + "]";
149     }
150 
151     //method: Print the transcript of a student
152     public String printTranscript() {
153         return "University of Knowledge - Official Transcript\n\n" + "ID: " + id +"\nName: " + name + "\nGPA:
                " + gpa + "\n\n" + recordsToString(records);
154     }
155 
156     //method: transform the student record array in a student into a string
157     public String recordsToString (StudentRecord[] records) {
158         StringBuilder sb = new StringBuilder("");
159         int termTemp = 0;
160         for (StudentRecord i: records) {
161             //setting the string arguments equal to the required information
162             String str1 = "| " + i.getModuleYear() + " | ";
163             String str2 = i.getModuleTerm() + " | ";
164             String str3 = i.getModule().getModuleDescriptor().getCode() + " | ";
165             String str4 = i.getFinalScore() + " |";
166             //check if the term of the current and previous modules are the same, if they are not add a new
                    line between them
167             if (Math.abs(i.getModuleTerm() - termTemp) > .1) {
168                 sb.append("\n");
169             }
170             //append the string arguments
171             sb.append(str1);//module year
172             sb.append(str2);//module term
173             sb.append(str3);//module descriptor code
174             sb.append(str4);//module final score
175             sb.append("\n");
176             //Make the tempTerm variable equal to the current term
177             termTemp = i.getModuleTerm();
178         }
179         //delete the last new line character
180         sb.deleteCharAt(sb.length()-1);
181         return "" + sb;
182     }
183 }
```

3 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    //Constructor of a student record
14    public StudentRecord (Student student, Module module, double[] marks, double finalScore, Boolean
          isAboveAverage) {
15       this.student = student;
16       this.module = module;
17       this.marks = marks;
18       if (0 > finalScore || finalScore > 100){
19          System.err.println("You input final module score of " + finalScore + " for the student with an ID
                of " + student.getId() + " but the final score of a student in a module must be between 0 and
                100.");
20          System.exit(1);
21       } // no else necessary since program will stop if an error occurs, if not then it will continue as
             normal
22       this.finalScore = finalScore;
23       this.isAboveAverage = isAboveAverage;
24    }
25
26    //methods:
27    //method: get the module year of a student record
28    public int getModuleYear () {
29       return module.getYear();
30    }
31    //method: get the module term of a student record
32    public byte getModuleTerm () {
33       return module.getTerm();
34    }
35    //method: get the final score of a student record
36    public double getFinalScore() {
37       return finalScore;
38    }
39    //method: get the module of a student record
40    public Module getModule() {
41       return module;
42    }
43
44    //method: get the module of a student record
45    public Student getStudent() {
46       return student;
47    }
48    //method: to string method
49    public String toString () {
50       return "Student record[ " + student + ", " + module + ", marks" + marks + ", final score " +
             finalScore + ", is above average " + isAboveAverage + "]";
51    }
52 }
```

4 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    //Each module's is checked whether it is only offered once per year, per term in the University
          constructor
14
15    //2 Overloaded constructors, 1 being a module before students are enrolled and one afterwards, including
          student records and a final average grade
16    //Constructor of a new module, therefore it will have no students enrolled
17
18    public Module (int year, byte term, ModuleDescriptor module) {
19       this.year = year;
20       this.term = term;
21       this.module = module;
22    }
23
24    //Constructor of a module with students enrolled, therefore they have student records and a final
          average grade
25    public Module (int year, byte term, ModuleDescriptor module, StudentRecord[] records, double
          finalAverageGrade) {
26       this.year = year;
27       this.term = term;
28       this.module = module;
29       this.finalAverageGrade = finalAverageGrade;
30       this.records = records;
31    }
32
33    //methods:
34    //method: get the year of a module
35    public int getYear () {
36       return year;
37    }
38
39    //method: get the term of a module
40    public byte getTerm () {
41       return term;
42    }
43
44    //method: get the module descriptor of a module
45    public ModuleDescriptor getModuleDescriptor(){
46       return module;
47    }
48
49    //method: get the final average grade of a module
50    public double getFinalAverageGrade(){
51       return finalAverageGrade;
52    }
53
```

```
54    //method: set the final average grade of a module
55    public void setFinalAverageGrade(double finalAverageGrade) {
56       this.finalAverageGrade = finalAverageGrade;
57    }
58
59    //method: get the student records of a module
60    public StudentRecord[] getStudentRecords() {
61       return records;
62    }
63    //method: to string method
64    public String toString () {
65       return "Module [year = " + year + ", term = " + term + ", module descriptor code = " +
             module.getCode() + ", final average grade = " + finalAverageGrade +"]";
66    }
67
68 }
```

5 University.java

```
1  import java.util.Arrays;
2  public class University {
3
4     private ModuleDescriptor[] moduleDescriptors;
5
6     private Student[] students;
7
8     private Module[] modules;
9
10 //Constructors for the University class
11 public University (ModuleDescriptor[] moduleDescriptors, Student[] students, Module[] modules) {
12    //Check that each student id is unique
13    for (int i = 0; i < students.length; i++){
14       //Checks each student id with every student id that comes after it in the array, prevents repetitions
15       for (int k = i + 1; k < students.length; k++){
16          if (students[i].getId() == students[k].getId()){
17             System.err.println("The students " + students[i].getName() + " and " + students[k].getName() + "
                   have the same ID. Each student requires a unique ID.");
18             System.exit(1);
19          }
20       }
21    }
22    //Check that each module is only offered once per year, per term
23    for (int i = 0; i < modules.length; i++){
24       //Checks each module with every module that comes after it in the array, prevents repetitions
25       for (int k = i + 1; k < modules.length; k++){
26          if (modules[i].getYear() == modules[k].getYear() && modules[i].getTerm() == modules[k].getTerm() &&
               modules[i].getModuleDescriptor() == modules[k].getModuleDescriptor()){
27             System.err.println("The module " + modules[i].getModuleDescriptor().getCode() + " cannot be
                   offered twice in the same year and term.");
28             System.exit(1);
29          }
30       }
31    }
32    this.moduleDescriptors = moduleDescriptors;
```

```
33      this.students = students;
34      this.modules = modules;
35  }
36
37      /**
38       * @return The number of students registered in the system.
39       */
40      public int getTotalNumberStudents() {
41          //return the length of the array of students, this will be the total number of students in the system
42          return students.length;
43      }
44
45      /**
46       * @return The student with the highest GPA.
47       */
48      public Student getBestStudent() {
49          //initializing a best student in order to be able to set the 1st students gpa as the best student
50          //initialize a temporary best student without a student record so a student record can be added to it
51          Student bestStudenttemp = new Student(0, "bestStudent", 'X');
52          //Create a student record including the best student
53          StudentRecord bestStudentRecord[] = {new StudentRecord (bestStudenttemp, null, new double[] {0}, 0,
                false)};
54          //Initialize the final best student, containing its student record
55          Student bestStudent = new Student(0, "bestStudent", 'X', bestStudentRecord);
56
57          //Iterate through every student in the array students, if a student has a higher gpa than the previous
                highest gpa then set the respective students gpa as the new highest gpa.
58          for (Student i: students) {
59              if (i.getGPA() > bestStudent.getGPA()) {
60                  bestStudent.setGPA(i.getGPA());
61              }
62          }
63          //Iterate through every student in the array of students and if they have the same gpa as the highest
                return that student as being the best student.
64          //If 2 students have the same GPA and the GPA is the highest then the student that comes first in the
                Array will automatically be set as the highest GPA and be returned as the best student.
65          for (Student i: students) {
66              if (Math.abs(i.getGPA() - bestStudent.getGPA()) < .00001){
67                  return i;
68              }
69          }
70          return null;
71      }
72
73      /**
74       * @return The module with the highest average score.
75       */
76      public Module getBestModule() {
77          //Similar to finding the best student this first initializes a module to compare the first module to
                and then it iterates through the array and if a module has a higher final average grade as the
                current highest final average grade, the modules higher final average grade is set as the new
                highest final average grade.
78          Module bestModule = new Module(0, (byte) 0, null, null, 0);
79          for (Module i: modules) {
80              if (i.getFinalAverageGrade() > bestModule.getFinalAverageGrade()) {
```

```
81              bestModule.setFinalAverageGrade(i.getFinalAverageGrade());
82           }
83        }
84        //Similar to finding the best student iterate through an array of modules and if the module has the
              same highest final average grade then return that module.
85        //If 2 modules have the same final average grade and it is the highest then the module that comes
              first in the Array will automatically be set as the highest final average grade and be returned
              as the best module.
86        for (Module i: modules) {
87           if (Math.abs(i.getFinalAverageGrade() - bestModule.getFinalAverageGrade()) < .00001){
88              return i;
89           }
90        }
91        return null;
92     }
93
94     public ModuleDescriptor[] getModuleDescriptor() {
95        return moduleDescriptors;
96     }
97     public Student[] getStudents() {
98        return students;
99     }
100    public Module[] getModules() {
101       return modules;
102    }
103    public String toString (){
104       return "[" + Arrays.toString(students) + " modules are " + Arrays.toString(modules) + "
              moduledsecriptors are " + moduleDescriptors + "]";
105    }
106
107    public static void main(String[] args) {
108
109       //Create an array of module descriptors
110       ModuleDescriptor[] moduleDescriptors = new ModuleDescriptor[6];
111       moduleDescriptors[0] = new ModuleDescriptor("ECM002", "Real World Mathematics", new double[]
              {0.1,0.3,0.6});
112       moduleDescriptors[1] = new ModuleDescriptor("ECM1400", "Programming", new double[]
              {0.25,0.25,0.25,0.25});
113       moduleDescriptors[2] = new ModuleDescriptor("ECM1406", "Data Structures", new double[]
              {0.25,0.25,0.5});
114       moduleDescriptors[3] = new ModuleDescriptor("ECM1410", "Object-Oriented Programming", new double[]
              {0.2,0.3,0.5});
115       moduleDescriptors[4] = new ModuleDescriptor("BEM2027", "Information Systems", new double[]
              {0.1,0.3,0.3,0.3});
116       moduleDescriptors[5] = new ModuleDescriptor("PHY2023", "Thermal Physics", new double[] {0.4,0.6});
117
118       //Check that each module descriptor code is unique
119       for (int i = 0; i < moduleDescriptors.length; i++){
120          //Checks each module descriptor code with every module descriptor code that comes after it in the
                 array, prevents repetitions
121          for (int k = i + 1; k < moduleDescriptors.length; k++){
122             if (moduleDescriptors[i].getCode().equals(moduleDescriptors[k].getCode())){
123                System.err.println("The module descriptors " + moduleDescriptors[i].getName() + " and " +
                       moduleDescriptors[k].getName() + " have the same code, each module descriptor code needs
                       to be unique.");
```

```
            System.exit(1);
        }
    }
}

Student student1 = new Student(1000, "Ana", 'F');
Student student2 = new Student (1001, "Oliver", 'M');
Student student3 = new Student (1002, "Mary", 'F');
Student student4 = new Student (1003, "John", 'M');
Student student5 = new Student (1004, "Noah", 'M');
Student student6 = new Student (1005, "Chico", 'M');
Student student7 = new Student (1006, "Maria", 'F');
Student student8 = new Student (1007, "Mark", 'X');
Student student9 = new Student (1008, "Lia", 'F');
Student student10 = new Student (1009, "Rachel", 'F');

//Create modules that students then enrol onto
Module module1 = new Module (2019, (byte) 1, moduleDescriptors[1]); //ECM1400
Module module2 = new Module (2019, (byte) 1, moduleDescriptors[5]); //PHY2023
Module module3 = new Module (2019, (byte) 2, moduleDescriptors[4]); //BEM2027
Module module4 = new Module (2019, (byte) 2, moduleDescriptors[1]); //ECM1400
Module module5 = new Module (2020, (byte) 1, moduleDescriptors[2]); //ECM1406
Module module6 = new Module (2020, (byte) 1, moduleDescriptors[3]); //ECM1410
Module module7 = new Module (2020, (byte) 2, moduleDescriptors[0]); //ECM002

//Create student records and grades for modules that students enrolled in
//each student took 2 modules, 2 in 2019 and 2 in 2020
//all students took module 5

//student 1
StudentRecord student1Records1= new StudentRecord (student1, module1, new double[] {9,10,10,10}, 97.5,
    true);
StudentRecord student1Records2 = new StudentRecord (student1, module3, new double[] {10,10,9.5,10},
    98.5, true);
StudentRecord student1Records3= new StudentRecord (student1, module5, new double[] {10,10,10}, 100,
    true);
StudentRecord student1Records4= new StudentRecord (student1, module6, new double[] {10,9,10}, 97,
    true);

//student 2
StudentRecord student2Records1= new StudentRecord (student2, module1, new double[] {8, 8, 8, 9}, 82.5,
    true);
StudentRecord student2Records2 = new StudentRecord (student2, module3, new double[] {7, 8.5, 8.2, 8},
    81.1, true);
StudentRecord student2Records3= new StudentRecord (student2, module5, new double[] {8, 7.5, 7.5},
    76.25, false);
StudentRecord student2Records4= new StudentRecord (student2, module6, new double[] {8.5,9,7.5}, 81.5,
    true);

//student 3
StudentRecord student3Records1= new StudentRecord (student3, module1, new double[] {5,5,6,5}, 52.5,
    false);
StudentRecord student3Records2 = new StudentRecord (student3, module3, new double[] {6.5,7.0,5.5,8.5},
    69.5, false);
StudentRecord student3Records3= new StudentRecord (student3, module5, new double[] {9,7,7}, 75, false);
```

```
169         StudentRecord student3Records4= new StudentRecord (student3, module6, new double[] {10,10,5.5}, 77.5,
                false);
170
171         //student 4
172         StudentRecord student4Records1= new StudentRecord (student4, module1, new double[] {6,4,7,9}, 65,
                false);
173         StudentRecord student4Records2 = new StudentRecord (student4, module3, new double[] {5.5,5,6.5,7}, 61,
                false);
174         StudentRecord student4Records3= new StudentRecord (student4, module5, new double[] {9,8,7}, 77.5,
                false);
175         StudentRecord student4Records4= new StudentRecord (student4, module6, new double[] {7,7,7}, 70, false);
176
177         //student 5
178         StudentRecord student5Records1= new StudentRecord (student5, module1, new double[] {10,9,10,9}, 95,
                true);
179         StudentRecord student5Records2 = new StudentRecord (student5, module3, new double[] {7,5,8,6}, 64,
                false);
180         StudentRecord student5Records3= new StudentRecord (student5, module5, new double[] {2,7,7}, 57.5,
                false);
181         StudentRecord student5Records4= new StudentRecord (student5, module6, new double[] {5,6,10}, 78,
                false);
182
183         //student 6
184         StudentRecord student6Records1= new StudentRecord (student6, module2, new double[] {9, 9}, 90, true);
185         StudentRecord student6Records2= new StudentRecord (student6, module4, new double[] {9 ,10, 10, 10},
                97.5, true);
186         StudentRecord student6Records3= new StudentRecord (student6, module5, new double[] {10 ,10, 10}, 100,
                true);
187         StudentRecord student6Records4= new StudentRecord (student6, module7, new double[] {8 ,9, 8}, 83,
                false);
188
189         //student 7
190         StudentRecord student7Records1= new StudentRecord (student7, module2, new double[] {6, 9}, 78, true);
191         StudentRecord student7Records2= new StudentRecord (student7, module4, new double[] {8, 8, 8, 9}, 82.5,
                true);
192         StudentRecord student7Records3= new StudentRecord (student7, module5, new double[] {8, 7.5, 7.5},
                76.25, false);
193         StudentRecord student7Records4= new StudentRecord (student7, module7, new double[] {6.5,9,9.5}, 90.5,
                true);
194
195         //student 8
196         StudentRecord student8Records1= new StudentRecord (student8, module2, new double[] {5, 6}, 56, false);
197         StudentRecord student8Records2= new StudentRecord (student8, module4, new double[] {5,5,6,5}, 52.5,
                false);
198         StudentRecord student8Records3= new StudentRecord (student8, module5, new double[] {10,10,10}, 100,
                true);
199         StudentRecord student8Records4= new StudentRecord (student8, module7, new double[] {8.5,10,8.5}, 89.5,
                true);
200
201         //student 9
202         StudentRecord student9Records1= new StudentRecord (student9, module2, new double[] {9, 7}, 78, true);
203         StudentRecord student9Records2= new StudentRecord (student9, module4, new double[] {6,4,7,9}, 65,
                false);
204         StudentRecord student9Records3= new StudentRecord (student9, module5, new double[] {9,8,7}, 77.5,
                false);
```

```
205         StudentRecord student9Records4= new StudentRecord (student9, module7, new double[] {7.5,8,10}, 91.5,
                true);
206
207         //student 10
208         StudentRecord student10Records1= new StudentRecord (student10, module2, new double[] {8, 5}, 62,
                false);
209         StudentRecord student10Records2= new StudentRecord (student10, module4, new double[] {10,9,8,9}, 90,
                true);
210         StudentRecord student10Records3= new StudentRecord (student10, module5, new double[] {8,9,10}, 92.5,
                true);
211         StudentRecord student10Records4= new StudentRecord (student10, module7, new double[] {10,6,10}, 88,
                false);
212
213         //Create arrays of student records for each student
214         StudentRecord[] studentRecordsstudent1 = {student1Records1, student1Records2, student1Records3,
                student1Records4};
215         StudentRecord[] studentRecordsstudent2 = {student2Records1, student2Records2, student2Records3,
                student2Records4};
216         StudentRecord[] studentRecordsstudent3 = {student3Records1, student3Records2, student3Records3,
                student3Records4};
217         StudentRecord[] studentRecordsstudent4 = {student4Records1, student4Records2, student4Records3,
                student4Records4};
218         StudentRecord[] studentRecordsstudent5 = {student5Records1, student5Records2, student5Records3,
                student5Records4};
219         StudentRecord[] studentRecordsstudent6 = {student6Records1, student6Records2, student6Records3,
                student6Records4};
220         StudentRecord[] studentRecordsstudent7 = {student7Records1, student7Records2, student7Records3,
                student7Records4};
221         StudentRecord[] studentRecordsstudent8 = {student8Records1, student8Records2, student8Records3,
                student8Records4};
222         StudentRecord[] studentRecordsstudent9 = {student9Records1, student9Records2, student9Records3,
                student9Records4};
223         StudentRecord[] studentRecordsstudent10 = {student10Records1, student10Records2, student10Records3,
                student10Records4};
224
225         //Create arrays of student records for each module
226         //students 1-5 took the same modules
227         //students 6-10 took the same modules
228         //all students took module 5
229         StudentRecord[] studentRecordsmodule1 = {student1Records1, student2Records1, student3Records1,
                student4Records1, student5Records1};
230         StudentRecord[] studentRecordsmodule2 = {student6Records1, student7Records1, student8Records1,
                student9Records1, student10Records1};
231         StudentRecord[] studentRecordsmodule3 = {student1Records2, student2Records2, student3Records2,
                student4Records2, student5Records2};
232         StudentRecord[] studentRecordsmodule4 = {student6Records2, student7Records2, student8Records2,
                student9Records2, student10Records2};
233         StudentRecord[] studentRecordsmodule5 = {student1Records3, student2Records3, student3Records3,
                student4Records3, student5Records3, student6Records3, student7Records3, student8Records3,
                student9Records3, student10Records3};
234         StudentRecord[] studentRecordsmodule6 = {student1Records4, student2Records4, student3Records4,
                student4Records4, student5Records4};
235         StudentRecord[] studentRecordsmodule7 = {student6Records4, student7Records4, student8Records4,
                student9Records4, student10Records4};
236
```

```
237         //Initialize students complete with an array of student records, each refering to a module taken
238         //GPA is calculated in the constructor, can be checked with getGPA
239         Student student1Final = new Student(student1.getId(), student1.getName(), student1.getGender(),
                studentRecordsstudent1);
240         Student student2Final = new Student (student2.getId(), student2.getName(), student2.getGender(),
                studentRecordsstudent2);
241         Student student3Final = new Student (student3.getId(), student3.getName(), student3.getGender(),
                studentRecordsstudent3);
242         Student student4Final = new Student (student4.getId(), student4.getName(), student4.getGender(),
                studentRecordsstudent4);
243         Student student5Final = new Student (student5.getId(), student5.getName(), student5.getGender(),
                studentRecordsstudent5);
244         Student student6Final = new Student (student6.getId(), student6.getName(), student6.getGender(),
                studentRecordsstudent6);
245         Student student7Final = new Student (student7.getId(), student7.getName(), student7.getGender(),
                studentRecordsstudent7);
246         Student student8Final = new Student (student8.getId(), student8.getName(), student8.getGender(),
                studentRecordsstudent8);
247         Student student9Final = new Student (student9.getId(), student9.getName(), student9.getGender(),
                studentRecordsstudent9);
248         Student student10Final = new Student (student10.getId(), student10.getName(), student10.getGender(),
                studentRecordsstudent10);
249
250         //Initialize modules with an array of student records, each referring to a different student
251         Module module1Final = new Module (module1.getYear(), module1.getTerm(), module1.getModuleDescriptor(),
                studentRecordsmodule1, 78.5 );
252         Module module2Final = new Module (module2.getYear(), module2.getTerm(), module2.getModuleDescriptor(),
                studentRecordsmodule2, 72.8 );
253         Module module3Final = new Module (module3.getYear(), module3.getTerm(), module3.getModuleDescriptor(),
                studentRecordsmodule3, 74.82 );
254         Module module4Final = new Module (module4.getYear(), module4.getTerm(), module4.getModuleDescriptor(),
                studentRecordsmodule4, 77.5 );
255         Module module5Final = new Module (module5.getYear(), module5.getTerm(), module5.getModuleDescriptor(),
                studentRecordsmodule5, 83.25 );
256         Module module6Final = new Module (module6.getYear(), module6.getTerm(), module6.getModuleDescriptor(),
                studentRecordsmodule6, 80.8 );
257         Module module7Final = new Module (module7.getYear(), module7.getTerm(), module7.getModuleDescriptor(),
                studentRecordsmodule7, 88.5 );
258
259         //create final array of students including student records
260         Student[] students = {student1Final, student2Final, student3Final, student4Final, student5Final,
                student6Final, student7Final, student8Final, student9Final, student10Final};
261
262
263
264         //Create final modules with an array of student records and final grades
265         Module[] modules = {module1Final, module2Final, module3Final, module4Final, module5Final,
                module6Final, module7Final};
266
267         //Create university object containing an array of module descriptors, students and modules
268         University university = new University(moduleDescriptors, students, modules);
269
270         // Print Reports
271         System.out.println("The UoK has " + university.getTotalNumberStudents() + " students.");
272
```

```
273         // best module
274         System.out.println("The best module is:");
275         System.out.println(university.getBestModule());
276 
277         // best student
278         System.out.println("The best student is:");
279         System.out.println(university.getBestStudent().printTranscript());
280     }
281 }
```