```java
//this class contains info about the modules available at the university such as the year, term , module descriptor info, records and final average grade for the module.

public class Module {

    private int year;

    private byte term;

    private ModuleDescriptor module;

    private StudentRecord[] records;

    private double finalAverageGrade;

    //overloaded constructor containing arguments ( year, term, records, finalAverageGrade, and module)
    public Module(ModuleDescriptor module,
                  int year,
                  byte term,
                  StudentRecord[] records,
                  double finalAverageGrade){
        this.module = module;
        this.year = year;
        this.term = term;
        this.records = records;
        this.finalAverageGrade = finalAverageGrade;
    }
    //overloaded constructor containing arguments ( year, term and module and finalAverageGrade)
    public Module(int year, byte term, ModuleDescriptor module,double finalAverageGrade ){
        this.module = module;
        this.year = year;
        this.term = term;
        this.finalAverageGrade = finalAverageGrade;

    }
    //overloaded constructor containing arguments ( year, term and module)
    public Module(int year, byte term, ModuleDescriptor module ) {
```

```
37          this.module = module;
38          this.year = year;
39          this.term = term;
40      }
41      // public getters for private instance
42      public ModuleDescriptor getModule(){
43          return module;
44      }
45      public int getYear(){
46          return year;
47      }
48      public byte getTerm(){
49          return term;
50      }
51      public StudentRecord[] getRecords(){
52          return records;
53      }
54      public double getFinalAverageGrade(){
55          return finalAverageGrade;
56      }
57
58      //method to calculate the final average grade
59      public double calculateFinalAverageGrade(){
60          double finalAverageGrade = 0.0;
61          Student student = null;
62          for(StudentRecord record : records)
63              student = record.getStudent();
64              finalAverageGrade += student.getGpa();
65          double average = finalAverageGrade /records
    .length;
66          return average;
67      }
68
69      //public setter for private instance
70      public void setFinalAverageGrade(double
    finalAverageGrade) {
71          this.finalAverageGrade = finalAverageGrade;
72      }
73
74      //toString method to print out info about a
    particular module
75      @Override
76      public String toString() {
77          return "Module: |" +
```

```
78                  "year=" + year +
79                  "| term=" + term +
80                  "| module=" + module.getCode() +
81                  "| finalAverageGrade=" +
   finalAverageGrade +
82                  '|';
83      }
84  }
85
```

```java
//this class contains the students enrolled at the university, it stores valid info about their id, name, gender, gpa and records.

public class Student {

    private int id;

    private String name;

    private char gender;

    private double gpa;

    private StudentRecord[] records;

    //overloaded constructor containing arguments ( id, name, gender, records and gpa)
    public Student(int id, String name, char gender , StudentRecord[] records, double gpa){
        setId(this.id);
        setName(this.name);
        setGender(this.gender);
        this.gpa = gpa;
        this.records = records;
    }
    //overloaded constructor containing arguments ( id, name and gender)
    public Student(int mId, String mName, char gender)
    {
        setId(mId);
        setName(mName);
        setGender(this.gender);
    }

    // public getters for private instance

    public int getId(){
        return id;
    }
    public String getName(){
        return name;
    }
```

```
39      public char getGender(){
40          return gender;
41      }
42      public double getGpa(){
43          return gpa;
44      }
45      public StudentRecord[] getRecord(){
46          return records;
47      }
48
49      // public setters for private instances with
   required constraints
50
51      //set ID so it is not null and is unique
52      public void setId(int mId){
53          //checks if it is not equal to null
54          if(mId != 0) {
55              id = mId;
56          }else {
57              System.out.println("ID was null");
58              System.exit(1);
59          }
60
61      }
62
63      //set gender to F,M,X or empty
64      public void setGender(char gender){
65          if(gender == 'M' || gender == 'F' || gender
    == 'X' || gender == '\0'){
66              this.gender = gender;
67          }else{
68              System.out.println("Gender was not
   allowed character: F,M,X or blank");
69              System.exit(1);
70          }
71      }
72
73      //set name to something that is not null
74      public void setName(String mName){
75          if(mName == null){
76              System.out.println("Name null");
77              System.exit(1);
78          }else{
79              name = mName;
```

```
 80         }
 81     }
 82 
 83     public void setRecords(StudentRecord[] records
    ){
 84         this.records = records;
 85     }
 86     public void setGpa(double gpa){
 87         this.gpa = gpa;
 88     }
 89 
 90     // prints info about a student in order of
    year and term
 91     public String printTranscript() {
 92         System.out.println("\t\t University of
    Knowledge - Official Transcript \n\n ID: " + getId
    () +"\n Name: "+ getName() +"\n GPA:" + getGpa
    () + "\n" );
 93 
 94 
 95         for (StudentRecord record:records){
 96             if (record.getModule().getYear() ==
    2019){
 97                 if (record.getModule().getTerm
    () == 1){
 98                     System.out.println("| "+
    record.getModule().getYear() + " | " + record.
    getModule().getTerm() + " | " + record.getModule
    ().getModule().getCode() + " | " + record.
    getFinalScore() + " |");
 99                 }
100             }
101         }
102         System.out.println(" ");
103         for (StudentRecord record:records){
104             if (record.getModule().getYear() ==
    2019){
105                 if (record.getModule().getTerm
    () == 2){
106                     System.out.println("| "+
    record.getModule().getYear() + " | " + record.
    getModule().getTerm() + " | " + record.getModule
    ().getModule().getCode() + " | " + record.
    getFinalScore() + " |");
```

```
107                 }
108             }
109         }
110         System.out.println(" ");
111         for (StudentRecord record:records){
112             if (record.getModule().getYear() ==
    2020){
113                 if (record.getModule().getTerm
    () == 1){
114                     System.out.println("| "+
    record.getModule().getYear() + " | " + record.
    getModule().getTerm() + " | " + record.getModule
    ().getModule().getCode() + " | " + record.
    getFinalScore() + " |");
115                 }
116             }
117         }
118         System.out.println(" ");
119         System.out.println(" ");
120         for (StudentRecord record:records){
121             if (record.getModule().getYear() ==
    2020){
122                 if (record.getModule().getTerm
    () == 2){
123                     System.out.println("| "+
    record.getModule().getYear() + " | " + record.
    getModule().getTerm() + " | " + record.getModule
    ().getModule().getCode() + " | " + record.
    getFinalScore() + " |");
124                 }
125             }
126         }
127         System.out.println(" ");
128 
129         return "";
130     }
131 }
```

```java
public class University {

    private ModuleDescriptor[] moduleDescriptors;

    private Student[] students;

    private Module[] modules;

    private StudentRecord[] fullRecords;

    //constructor for the university class containing module descriptors students and modules
    public University(ModuleDescriptor[] moduleDescriptors, Student[] students, Module[] modules ){
        this.moduleDescriptors = moduleDescriptors;
        this.students = students;
        this.modules = modules;
    }


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

    /**
     * @return The student with the highest GPA.
     */
    public Student getBestStudent() {
        double highestGpa = 0.0;
        Student bestStudent = students[0];
        for (Student student : students){
            if (student.getGpa()>highestGpa){
                highestGpa = student.getGpa();
                bestStudent = student;
            }
        }

        return bestStudent;
```

```
41      }
42
43      /**
44       * @return The module transcript with the
    highest average score.
45      **/
46      public String getBestModule() {
47          double max = 0.0;
48          Module bestModule = modules[0];
49          for (int i = 0; i < modules.length; i++){
50              if (max < modules[i].
    getFinalAverageGrade())
51              {
52                  max = modules[i].
    getFinalAverageGrade();
53                  bestModule = modules[i];
54              }
55          }
56
57          return bestModule.toString();
58      }
59
60      public static void main(String[] args) {
61
62      //initialise module
63      ModuleDescriptor[] moduleDescriptor = new
    ModuleDescriptor[6];
64      moduleDescriptor[0] = new ModuleDescriptor("
    ECM0002", "Real World Mathematics", new double[] {0
    .1, 0.3, 0.6});
65      moduleDescriptor[1] = new ModuleDescriptor("
    ECM1400", "Programming", new double[] {0.25, 0.25,
    0.25, 0.25});
66      moduleDescriptor[2] = new ModuleDescriptor("
    ECM1406", "Data Structures", new double[] {0.25, 0.
    25, 0.5});
67      moduleDescriptor[3] = new ModuleDescriptor("
    ECM1410", "Object-Oriented Programming", new double
    [] {0.2, 0.3, 0.5});
68      moduleDescriptor[4] = new ModuleDescriptor("
    BEM2027", "Information Systems", new double[] {0.1
    , 0.3, 0.3, 0.3});
69      moduleDescriptor[5]= new ModuleDescriptor("
    PHY2023", "Thermal Physics", new double[] {0.4, 0.6
```

```
69 });
70
71     //initialise students
72     Student[] students = new Student[10];
73     students[0] = new Student(1000, "Ana",'F');
74     students[1] = new Student(1001, "Oliver", 'M'
   );
75     students[2] = new Student(1002, "Mary", 'F');
76     students[3] = new Student(1003, "John", 'M');
77     students[4] = new Student(1004, "Noah", 'M');
78     students[5] = new Student(1005, "Chico", 'M');
79     students[6] = new Student(1006, "Maria", 'F');
80     students[7] = new Student(1007, "Mark", 'x');
81     students[8] = new Student(1008, "Lia", 'F');
82     students[9] = new Student(1009, "Rachel", 'F'
   );
83
84     //initialise modules
85     Module[] module = new Module[7];
86     module[0] = new Module(2019, (byte)1,
   moduleDescriptor[1], 7.85);
87     module[1] = new Module(2019, (byte)1,
   moduleDescriptor[5], 7.280);
88     module[2] = new Module(2019, (byte)2,
   moduleDescriptor[4], 7.482);
89     module[3] = new Module(2019, (byte)2,
   moduleDescriptor[1],7.75);
90     module[4] = new Module(2020, (byte)1,
   moduleDescriptor[2],8.325);
91     module[5] = new Module(2020, (byte)1,
   moduleDescriptor[3],8.08);
92     module[6] = new Module(2020, (byte)2,
   moduleDescriptor[0], 8.85);
93
94     //initialise student records
95     StudentRecord[] allRecords = new StudentRecord
   [40];
96     allRecords[0] = new StudentRecord(students[0
   ], module[0], new double[]{9, 10, 10, 10});
97     allRecords[1] = new StudentRecord(students[1
   ], module[0], new double[] {8, 8, 8, 9});
98     allRecords[2]= new StudentRecord(students[2],
   module[0], new double[]{5, 5, 6, 5});
99     allRecords[3]= new StudentRecord(students[3],
```

```
 99 module[0], new double[]{6, 4, 7, 9});
100     allRecords[4]= new StudentRecord(students[4],
    module[0], new double[]{10, 9, 10, 9});
101     allRecords[5] = new StudentRecord(students[5
    ], module[1], new double[]{9,9});
102     allRecords[6] = new StudentRecord(students[6
    ], module[1], new double[]{6, 9});
103     allRecords[7] = new StudentRecord(students[7
    ], module[1], new double[]{5, 6});
104     allRecords[8] = new StudentRecord(students[8
    ], module[1], new double[]{9, 7});
105     allRecords[9] = new StudentRecord(students[9
    ], module[1], new double[]{8,5});
106     allRecords[10] = new StudentRecord(students[0
    ], module[2], new double[]{10, 10, 9.5, 10});
107     allRecords[11] = new StudentRecord(students[1
    ], module[2], new double[]{7, 8.5, 8.2, 8});
108     allRecords[12] = new StudentRecord(students[2
    ], module[2],new double[]{6.5, 7.0, 5.5, 8.5});
109     allRecords[13] = new StudentRecord(students[3
    ], module[2], new double[]{5.5, 5,6.5, 7 });
110     allRecords[14] = new StudentRecord(students[4
    ], module[2], new double[]{7, 5, 8, 6});
111     allRecords[15] = new StudentRecord(students[5
    ], module[3], new double[]{9, 10, 10, 10});
112     allRecords[16] = new StudentRecord(students[6
    ], module[3], new double[]{8, 8, 8, 9});
113     allRecords[17] = new StudentRecord(students[7
    ], module[3], new double[]{5, 5, 6, 5});
114     allRecords[18] =new StudentRecord(students[8
    ], module[3], new double[]{6, 4, 7, 9});
115     allRecords[19] =new StudentRecord(students[9
    ], module[3], new double[]{10, 9, 8, 9});
116     allRecords[20] = new StudentRecord(students[0
    ], module[4], new double[]{10, 10, 10});
117     allRecords[21] = new StudentRecord(students[1
    ], module[4], new double[]{8, 7.5, 7.5});
118     allRecords[22] = new StudentRecord(students[2
    ], module[4], new double[]{9,7,9});
119     allRecords[23] = new StudentRecord(students[3
    ], module[4], new double[]{9, 8, 7});
120     allRecords[24] = new StudentRecord(students[4
    ], module[4], new double[]{2, 7, 7});
121     allRecords[25] = new StudentRecord(students[5
```

```
121 ], module[4], new double[]{10, 10, 10});
122     allRecords[26] = new StudentRecord(students[6
    ], module[4], new double[]{8, 7.5, 7.5});
123     allRecords[27] = new StudentRecord(students[7
    ], module[4], new double[]{10, 10, 10});
124     allRecords[28] = new StudentRecord(students[8
    ], module[4], new double[]{9, 8, 7});
125     allRecords[29] = new StudentRecord(students[9
    ], module[4], new double[]{8, 9, 10});
126     allRecords[30] = new StudentRecord(students[0
    ], module[5], new double[]{10, 9 , 10});
127     allRecords[31] = new StudentRecord(students[1
    ], module[5], new double[]{8.5, 9, 7.5});
128     allRecords[32] = new StudentRecord(students[2
    ], module[5], new double[]{10, 10, 5.5});
129     allRecords[33] = new StudentRecord(students[3
    ], module[5], new double[]{7, 7 ,7});
130     allRecords[34] = new StudentRecord(students[4
    ], module[5], new double[]{5, 6, 10});
131     allRecords[35] = new StudentRecord(students[5
    ], module[6], new double[]{8, 9, 8});
132     allRecords[36] = new StudentRecord(students[6
    ], module[6], new double[]{6.5, 9, 9.5});
133     allRecords[37] = new StudentRecord(students[7
    ], module[6], new double[]{8.5, 10, 8.5});
134     allRecords[38] = new StudentRecord(students[8
    ], module[6], new double[]{7.5, 8, 10});
135     allRecords[39] = new StudentRecord(students[9
    ], module[6], new double[]{10, 6, 10});
136
137     //initialise university
138     University universityOfKnowledge = new
    University(moduleDescriptor, students, module);
139
140     //calculate final scores for every student in
    each module and check if they are above average
141     for (StudentRecord record : allRecords){
142         record.setIsAboveAverage(record.
    checkAboveAverage());
143         record.setFinalScore(record.
    calculateFinalScore());
144     }
145
146     //add student records to students
```

```
147     for(Student student : students){
148         int i = 0;
149         StudentRecord[] myRecords = new
    StudentRecord[4];
150         for (StudentRecord record: allRecords){
151             if (record.getStudent() == student){
152                 myRecords[i] = record;
153                 i++;
154             }
155         }
156         student.setRecords(myRecords);
157     }
158 
159     //calculating the gpa of students
160     for(Student student: students){
161         int i = 0;
162         student.setGpa(0);
163         double x = 0;
164         while (i < 4){
165             x += student.getRecord()[i].
    getFinalScore()/4;
166             i++;
167         }
168         student.setGpa(x);
169     }
170 
171 
172     //Print Reports
173     System.out.println("The UoK has " +
    universityOfKnowledge.getTotalNumberStudents() +
    " students.");
174 
175     // best module
176     System.out.println("The best module is:");
177     System.out.println(universityOfKnowledge.
    getBestModule());
178 
179     // best student
180     System.out.println("The best student is:");
181     universityOfKnowledge.getBestStudent().
    printTranscript();
182     }
183 }
184 
```

```java
//this class contains the students records, such as th modules they took, the marks they received, their final score and whether they were above the class average

public class StudentRecord {

    private Student student;

    private Module module;

    private double[] marks;

    private double finalScore;

    private Boolean isAboveAverage;

    //constructor containing arguments (
    public StudentRecord(Student student, Module module, double[] marks){
        this.student = student;
        this.module = module;
        this.marks = marks;
    }

    //public getters for private instances
    public Student getStudent(){
        return student;
    }
    public Module getModule(){
        return module;
    }
    public double[] getMarks(){
        return marks;
    }

    //method to calculate the final score of a student
    public double calculateFinalScore(){
        double[] weights = module.getModule().getContinuousAssignmentWeight();
        double calculatedScore;
        calculatedScore = 0.0;
        int i = 0;
```

```
39          for (double mark: marks){
40              calculatedScore += mark*weights[i];
41              i ++;
42          }
43          return calculatedScore;
44      }
45
46      //public getter for private instance
47      public double getFinalScore(){
48          return finalScore;
49      }
50
51      //public setters for private instance
52      public void setFinalScore(double finalScore){
53          this.finalScore = finalScore;
54      }
55
56      //checks whether the student achieved above
   average in the module
57      public boolean checkAboveAverage(){
58          double grade = 0;
59          grade = module.getFinalAverageGrade();
60          if (grade > finalScore){
61              isAboveAverage = true;
62          }
63          else{
64              isAboveAverage = false;
65          }
66          return isAboveAverage;
67      }
68
69      //public getter for private instance
70      public Boolean getAboveAverage() {
71          return isAboveAverage;
72      }
73
74      //public setter for private instance
75      public void setIsAboveAverage(Boolean
   isAboveAverage) {
76          this.isAboveAverage = isAboveAverage;
77      }
78
79  }
80
```

```java
//this class contains module descriptors such as the module code, name and the weight of each assignment

public class ModuleDescriptor {

    private String code;

    private String name;

    private double[] continuousAssignmentWeights;

    /*constructor containing 3 arguments (course code, course name
    and array of weights of the continuous assessments)*/
    public ModuleDescriptor(String mCode, String mName, double[] mContinuousAssignmentWeights) {
        setCode(mCode);
        setName(mName);
        setContinuousAssignmentWeight(mContinuousAssignmentWeights) ;
    }

    //public setters for private instances

    //verifying that name is not null
    public void setCode(String mCode) {
        if (mCode == null) {
            System.out.println("Code null");
            System.exit(1);
        } else {
            code = mCode;
        }
    }

    //verifying that name is not null
    public void setName(String mName) {
        if (mName == null) {
            System.out.println("Name null");
            System.exit(1);
        } else {
            name = mName;
        }
```

```
39      }
40      //setting up continuous assignment weights so
    they are non negative and sum to 1
41      public void setContinuousAssignmentWeight(
    double[] mContinuousAssignmentWeights) {
42          for (double mContinuousAssignmentWeight :
    mContinuousAssignmentWeights)
43              if (!(mContinuousAssignmentWeight <= 0
    )) {
44                  double sum = 0;
45                  for (double assignmentWeight :
    mContinuousAssignmentWeights) {
46                      sum = sum + assignmentWeight;
47                  }
48                  if (sum != 1) {
49                      System.out.println("Invalid
    weighing");
50                      System.exit(1);
51                  } else {
52                      continuousAssignmentWeights =
    mContinuousAssignmentWeights;
53                  }
54              } else {
55                  System.out.println("Weights are
    negative");
56                  System.exit(1);
57              }
58
59      }
60      // public getters for private instance
61      public double[] getContinuousAssignmentWeight
    () {
62          return continuousAssignmentWeights;
63      }
64      public String getCode(){
65          return code;
66      }
67
68      public String getName(){
69          return name;
70      }
71  }
```