```java
public class Module {
    //1. A module must refer to a module descriptor in a specific year and term.
    // It contains an array of student records, and average score.
    //2. The average score of a module is the mean of the student record final scores for that module.
    //3. A module descriptor can only be o↵ered once per year and term.
    // In other words, at most, there will be only one instance of a module with the same descriptor, year, and term.

    private int year;

    private byte term;

    private ModuleDescriptor module;

    private StudentRecord[] records;

    private double finalAverageGrade;

    public Module(ModuleDescriptor m, int y, byte t){
        module = m;
        year = y;
        term = t;

    }

    public int getYear(){
        return year;
    }

    public byte getTerm(){
        return term;
    }

    public void setRecords(StudentRecord[] r){
        records = r;
    }
//way to calculate average frade
}
```

```
 1
 2   // IntelliJ API Decompiler stub source generated
   from a class file
 3   // Implementation of methods is not available
 4
 5 public class Module {
 6     private int year;
 7     private byte term;
 8     private ModuleDescriptor module;
 9     private StudentRecord[] records;
10     private double finalAverageGrade;
11
12     public Module(ModuleDescriptor m, int y, byte t
   ) { /* compiled code */ }
13
14     public int getYear() { /* compiled code */ }
15
16     public byte getTerm() { /* compiled code */ }
17
18     public void setRecords(StudentRecord[] r) { /*
   compiled code */ }
19 }
```

```
 1 public class Student {
 2     //1. A student must have an ID, a name, a
   gender, a GPA, and an array of records for each
   module they have
 3     //been enrolled.
 4     //2. The ID and name cannot be null.
 5     //3. The ID must be unique.
 6     //4. The gender should be represented by one of
    the following characters: 'F', 'M', or 'X'. The
   student can
 7     // also prefer not to disclose this information
   , thus gender can be an empty field.
 8     //5. The grade point average (GPA) is the
   average of student record final scores
 9     // (the explanation of Student Record is below
   in the Ocial Transcript example).
10     //6. The system must be able to generate a
   transcript containing all student records,
11     // grouped by year and term. Implement the
   function public String printTranscript() using the
   following format:
12     //refer to mark scheme sheet
13     private int id;
14
15     private String name;
16
17     private char gender;
18
19     private double gpa;
20
21     private StudentRecord[] records;
22
23     public String printTranscript() {
24         // do something
25         return "";
26     }
27
28     public Student(int i, String n, char g){
29         id = i;
30         name = n;
31         gender = g;
32
33     }
34
```

```
35      public int getId(){
36          return id;
37      }
38
39      public String getName(){
40          return name;
41      }
42
43      public char getGender(){
44          return gender;
45      }
46  }
47
```

```
 1
 2   // IntelliJ API Decompiler stub source generated
   from a class file
 3   // Implementation of methods is not available
 4
 5 public class Student {
 6     private int id;
 7     private java.lang.String name;
 8     private char gender;
 9     private double gpa;
10     private StudentRecord[] records;
11
12     public java.lang.String printTranscript() { /*
   compiled code */ }
13
14     public Student(int i, java.lang.String n, char
   g) { /* compiled code */ }
15
16     public int getId() { /* compiled code */ }
17
18     public java.lang.String getName() { /* compiled
    code */ }
19
20     public char getGender() { /* compiled code */ }
21 }
```

```
 1 public class University {
 2     //1. The university has an array of module
   descriptors, an array of students, and an array of
   modules.
 3     //2. The system must be able to initialise the
   array of module descriptors as shown in Table 1.
 4     //3. The system must be able to initialise the
   array of students as shown in Table 2.
 5     //4. The system must be able to initialise
   modules with their respective students and marks as
    shown in Table 3. 5. In order to generate reports
   , the UoK (University.java) must implement the
   following functions:
 6     //(a) public int getTotalNumberStudents():
   return the number of students registered in the
   system. (b) public Student getBestStudent(): return
    the student with the highest GPA.
 7     //(c) public Module getBestModule(): return the
    module with the highest average score.
 8
 9     private ModuleDescriptor[] moduleDescriptors;
10
11     private Student[] students;
12
13     private Module[] modules;
14
15     int studentRecords[] = new int[10];
16
17     /**
18      * @return The number of students registered in
    the system.
19      */
20     public int getTotalNumberStudents() {
21         // TODO - needs to be implemented
22         return students.length;
23     }
24
25     /**
26      * @return The student with the highest GPA.
27      */
28     public Student getBestStudent() {
29         // TODO - needs to be implemented
30         return null;
31     }
```

```
32
33      /**
34       * @return The module with the highest average
   score.
35       */
36      public Module getBestModule() {
37          // TODO - needs to be implemented
38          return null;
39      }
40
41      public static void main(String[] args) {
42          ModuleDescriptor ECM0002 = new
   ModuleDescriptor("Real World Mathematics", "ECM0002
   ",new double []{0.1,0.3,0.6});
43          ModuleDescriptor ECM1400 = new
   ModuleDescriptor("Programming", "ECM1400",new
   double[]{0.25,0.25,0.25,0.25});
44          ModuleDescriptor ECM1406 = new
   ModuleDescriptor("Data Structures", "ECM1406",new
   double[]{0.25,0.25,0.5});
45          ModuleDescriptor ECM1410 = new
   ModuleDescriptor("Object-Oriented Programming", "
   ECM1410",new double[]{0.2,0.3,0.5});
46          ModuleDescriptor BEM2027 = new
   ModuleDescriptor("Information Systems", "BEM2027",
   new double[]{0.1,0.3,0.3,0.3});
47          ModuleDescriptor PHY2023 = new
   ModuleDescriptor("Thermal Physics", "PHY2023",new
   double[]{0.4,0.6});
48
49          Module math = new Module(ECM0002, 2020, (
   byte) 2);
50          Module programming = new Module(ECM1400,
   2019, (byte) 1);
51          Module dataStructures = new Module(ECM1406
   , 2020, (byte) 2);
52          Module objectOriented = new Module(ECM1410
   , 2020, (byte) 2);
53          Module informationSystems = new Module(
   BEM2027, 2019, (byte) 2);
54          Module thermalPhysics = new Module(PHY2023
   , 2019, (byte) 1);
55          Module programmingTwo = new Module(ECM1400
   , 2020, (byte) 2);
```

```
56
57          Student ana = new Student(1000, "Ana", 'F'
   );
58          Student oliver = new Student(1001, "Oliver"
   , 'M');
59          Student mary = new Student(1002, "Mary", 'F
   ');
60          Student john = new Student(1003, "John", 'M
   ');
61          Student noah = new Student(1004, "Noah", 'M
   ');
62          Student chico = new Student(1005, "Chico",
   'M');
63          Student maria = new Student(1006, "Maria",
   'F');
64          Student mark = new Student(1007, "Mark", 'X
   ');
65          Student lia = new Student(1008, "Lia", 'F'
   );
66          Student rachel = new Student(1009, "Rachel"
   , 'F');
67
68          University university = new University();
69
70          university.moduleDescriptors[0] = ECM0002;
71          university.moduleDescriptors[1] = ECM1400;
72          university.moduleDescriptors[2] = ECM1406;
73          university.moduleDescriptors[3] = ECM1410;
74          university.moduleDescriptors[4] = BEM2027;
75          university.moduleDescriptors[5] = PHY2023;
76          university.moduleDescriptors[6] = ECM1400;
77
78          university.students[0] = ana;
79          university.students[1] = oliver;
80          university.students[2] = mary;
81          university.students[3] = john;
82          university.students[4] = noah;
83          university.students[5] = chico;
84          university.students[6] = maria;
85          university.students[7] = mark;
86          university.students[8] = lia;
87          university.students[9] = rachel;
88
89          university.modules[0] = math;
```

```
 90          university.modules[1] = programming;
 91          university.modules[2] = dataStructures;
 92          university.modules[3] = objectOriented;
 93          university.modules[4] = informationSystems
    ;
 94          university.modules[5] = thermalPhysics;
 95          university.modules[6] = programmingTwo;
 96
 97          int[] codes = {1000, 1001, 1002, 1003,
    1004, 1005, 1006, 1007, 1008, 1009, 1000, 1001,
    1002, 1003, 1004, 1005,
 98          1006, 1007, 1008, 1009, 1000, 1001, 1002,
    1003, 1004, 1005, 1006, 1007, 1008, 1009, 1000,
    1001, 1002, 1003, 1004,
 99          1005, 1006, 1007, 1008, 1009};
100
101         String[] moduleCodes = {"ECM1400", "
    ECM1400","ECM1400", "ECM1400", "PHY2023", "PHY2023
    ", "PHY2023", "PHY2023", "PHY2023",
102                 "BEM2027", "BEM2027", "BEM2027", "
    BEM2027", "BEM2027","ECM1400", "ECM1400", "ECM1400
    ", "ECM1400", "ECM1400", "ECM1406",
103                 "ECM1406", "ECM1406", "ECM1406", "
    ECM1406", "ECM1406", "ECM1406", "ECM1406", "
    ECM1406", "ECM1406", "ECM1410", "ECM1410",
104         "ECM1410", "ECM1410", "ECM1410", "ECM0002"
    , "ECM0002", "ECM0002", "ECM0002", "ECM0002"};
105
106         int[] year = {2019, 2019, 2019, 2019, 2019
    , 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019,
    2019, 2019, 2019, 2019, 2019,
107         2019, 2019, 2020, 2020, 2020, 2020, 2020,
    2020, 2020, 2020, 2020, 2020, 2020, 2020, 2020,
    2020, 2020, 2020, 2020, 2020,
108         2020, 2020, 2020};
109
110         byte[] term = {1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1
    , 1, 1, 1, 1, 1, 1, 1,
111         1, 1, 2, 2, 2, 2, 2};
112
113         double[][] marks = {{9, 10, 10, 10,}, {8,
    8, 8, 9}, {5, 5, 6, 5}, {6, 4, 7, 9}, {10, 9, 10,
    9}, {9, 9}, {6, 9}, {5, 6},
```

```
114          {9, 7}, {8,5}, {10, 10, 9.5, 10}, {7, 8.5
    , 8.2, 8}, {6.5, 7.0, 5.5, 8.5}, {5.5, 5, 6.5, 7
    }, {7, 5, 8, 6},
115          {9, 10, 10, 10}, {8, 8, 8, 9}, {5, 5, 6, 5
    }, {6, 4, 7, 9}, {10, 9, 8, 9}, {10, 10, 10}, {8,
    7.5, 7.5},
116                  {9, 7, 7}, {9, 8, 7}, {2, 7, 7}, {
    10, 10, 10}, {8, 7.5, 7.5}, {10, 10, 10}, {9, 8, 7
    }, {8, 9, 10}, {10, 9, 10},
117                  {8.5, 9, 7.5}, {10, 10, 5.5}, {7,
    7, 7}, {5, 6, 10}, {8, 9, 8}, {6.5, 9, 9.5}, {8.5
    , 10, 8.5}, {7.5, 8,10},
118                  {10, 6, 10}};
119
120
121
122          for(int i = 0; i < university.students.
    length; i++){
123              if(university.students.id[i] == codes[
    i]){
124                  studentRecords.add(students[i]);
125
126              }
127          }
128
129      }
130
131 }
132
133
134
```

```java
public class StudentRecord {
    //1. A student record refers to a student in a module, and contains an array of marks, a final score, and a Boolean
    //to indicate whether the student was above the average.
    //2. The final score is a weighted average calculated based on the arrays of marks and the array of weights of
    //the module descriptor of the respective module.
    //3. Marks and the final score must range between 0 and 100.
    //4. Above the average should be true if the student final score is greater
    // than the average final scores in that particular module.
    //5. A student can only have one record per module.
    private Student student;

    private Module module;

    private double[] marks;

    private double finalScore;

    private Boolean isAboveAverage;

    public StudentRecord(Student s, Module m, double[] r){
        student = s;
        module = m;
        marks = r;
    }

    public Module getModule(){
        return module;
    }

    public Student getStudent(){
        return student;
    }

```

```
34      public double[] getMarks(){
35          return marks;
36      }
37
38      public boolean getIsAboveAverage(){
39          return isAboveAverage;
40      }
41
42
43
44  }
45
```

```
1
2   // IntelliJ API Decompiler stub source generated
  from a class file
3   // Implementation of methods is not available
4
5 public class StudentRecord {
6     private Student student;
7     private Module module;
8     private double[] marks;
9     private double finalScore;
10    private java.lang.Boolean isAboveAverage;
11
12    public StudentRecord() { /* compiled code */ }
13 }
```

```
 1 public class ModuleDescriptor {
 2     //1. A module descriptor must have a code, a
   name, and a double array to store the weights of
   the continuous assessments.
 3     //2. The code and the name can never be null.
 4     //3. The code must be unique.
 5     //4. The Continuous Assessment (CA) weights
   must sum up to 1, and must be non-negative.
 6
 7     private final String code;
 8
 9     private final String name;
10
11     private final double []
   continuousAssignmentWeights;
12
13
14     public ModuleDescriptor(String n, String c,
   double[] w){
15         code = c;
16         name = n;
17         if((checker(w)) == true){
18             continuousAssignmentWeights = w;
19         } else{
20             continuousAssignmentWeights = null;
21             System.out.println("incorrect weights"
   );
22             System.exit(0);
23         }
24
25     }
26
27
28     public boolean checker(double[] weights){
29         //adds to 1 and none are negative
30         double counter = 0.0;
31         for(int i = 0; i < weights.length-1; i++){
32             if(weights[i] < 0) {
33                 return false;
34             }
35             counter += weights[i];
36
37         }
38         if(counter != 1){
```

```
39              return false;
40          }
41          return true;
42
43      }
44
45
46      public String toString(){
47          return "Name:"+name+"Code"+code+"weight"+
    continuousAssignmentWeights;
48      }
49
50      public String getCode(){
51          return code;
52      }
53
54      public String getName(){
55          return name;
56      }
57
58      public double[] getContinuousAssignmentWeights
    (){
59          return continuousAssignmentWeights;
60      }
61
62  }
63  //string code
64  //string name
65  //double [] weights
66  //constructor
67  //setters and getters for each variable
68  //toString
69  //some way of checking if the weights are valid. i
    put this in my descriptor class
```

```
 1
 2   // IntelliJ API Decompiler stub source generated
   from a class file
 3   // Implementation of methods is not available
 4
 5 public class ModuleDescriptor {
 6     private final java.lang.String code;
 7     private final java.lang.String name;
 8     private final double[]
   continuousAssignmentWeights;
 9
10     public ModuleDescriptor(java.lang.String n,
   java.lang.String c, double[] w) { /* compiled code
    */ }
11
12     public boolean checker(double[] weights) { /*
   compiled code */ }
13
14     public java.lang.String toString() { /*
   compiled code */ }
15
16     public java.lang.String getCode() { /* compiled
    code */ }
17
18     public java.lang.String getName() { /* compiled
    code */ }
19
20     public double[] getContinuousAssignmentWeights
   () { /* compiled code */ }
21 }
```