1 ModuleDescriptor.java

```java
public class ModuleDescriptor {

   private String code;

   private String name;

   private double[] continuousAssignmentWeights;

    //Method: returns the code of the module descriptor
    public String getCode() {
        return this.code;
    }

    //Method: returns wights of module
    public double[] getContinuousAssignmentWeights() {
        return this.continuousAssignmentWeights;
    }
    //Constructor: Create module descriptor object
    public ModuleDescriptor(String code, String name, double[] continuousAssignmentWeights) {
        this.code = code;
        this.name = name;
        this.continuousAssignmentWeights = continuousAssignmentWeights;
    }
}
```

2 Student.java

```java
import java.util.ArrayList;
public class Student {

   private int id;

   private String name;

   private char gender;

   private double gpa;

   private StudentRecord[] records;

    //Method: Returns the student transcript in a string
   public String printTranscript() {
        String transcript;
        ArrayList<String> counted = new ArrayList<>();
        double term = 0;
        double year = 0;
      transcript = "\n" + "\n";
      transcript = transcript + "ID: " + String.valueOf(this.id) + "\n";
      transcript = transcript + "Name: " + String.valueOf(this.name) + "\n";
      transcript = transcript + "GPA: " + String.valueOf(this.gpa) + "\n" + "\n" +"\n";
        for (int i = 0; i < records.length; i++) {
            if (counted.contains(String.valueOf(records[i].getModule().getTerm()) + " " +
                String.valueOf(records[i].getModule().getYear())) == false) {
                term = records[i].getModule().getTerm();
                year = records[i].getModule().getYear();
                for (int j = 0; j < records.length; j++) {
                    if (records[j].getModule().getTerm() == term && records[j].getModule().getYear() == year)
                         {
                        counted.add(String.valueOf(records[i].getModule().getTerm()) + " " +
                            String.valueOf(records[i].getModule().getYear()));
                        transcript = transcript + "| " + String.valueOf(year) + " | " + String.valueOf(term)
                            + " | " + records[j].getModule().getModule().getCode() + " | " +
                            String.valueOf(records[j].getFinalScore()) + " | " + "\n";
                    }
                }
                transcript = transcript + "\n";
            }


        }
        return transcript;
    }


    //Method: Sets the value of an objects record
    public void setRecord(StudentRecord[] records) {
        this.records = records;
    }

    //Method: Returns the student ID of an object
    public int getId() {
        return this.id;
    }

    //Method: Returns the GPA of a student
    public double getGpa() {
        return this.gpa;
    }

    //Method: Returns the student's records
    public StudentRecord[] getRecords() {
        return this.records;
    }

    //Method: Returns the student's GPA
    public void setGpa(double gpa) {
        this.gpa = gpa;
    }

    //Constructor: Create student object
    public Student(int id, String name, char gender) {
        this.id = id;
        this.name = name;
        this.gender = gender;
    }


```

```
}
```

3 StudentRecord.java

```
public class StudentRecord {

   private Student student;

   private Module module;

   private double[] marks;

   private double finalScore;

   private Boolean isAboveAverage;

    //Method: Returns the student
    public Student getStudent() {
        return this.student;
    }

    //Method: Returns the module
    public Module getModule() {
        return this.module;
    }

    //Method: Returns the final score
    public double getFinalScore() {
        return this.finalScore;
    }

    //Method: Sets the boolean of the above average grade
    public void setIsAboveAverage(Boolean isAboveAverage) {
        this.isAboveAverage = isAboveAverage;
    }

    //Constructor: Creates a student record object
    public StudentRecord(Student student, Module module, double[] marks, double finalScore) {
        this.student = student;
        this.module = module;
        this.marks = marks;
        this.finalScore = finalScore;
    }
}
```

4 Module.java

```
public class Module {

   private int year;

   private byte term;

   private ModuleDescriptor module;
```

```java

    private StudentRecord[] records;

    private double finalAverageGrade;

     //Method: Returns the module
     public ModuleDescriptor getModule() {
         return this.module;
     }

     //Method: Returns the year
     public int getYear() {
         return this.year;
     }

     //Method: Returns the term
     public byte getTerm() {
         return this.term;
     }

     //Method: Returns the final average grade
     public double getFinalAverageGrade() {
         return this.finalAverageGrade;
     }

     //Method: Returns the student records
     public StudentRecord[] getRecords() {
         return this.records;
     }

     //Method: Sets the student records
     public void setRecords(StudentRecord[] records) {
         this.records = records;
     }

     //Method: Sets the final Average Grade
     public void setFinalAverageGrade(double finalAverageGrade) {
         this.finalAverageGrade = finalAverageGrade;
     }

     //Method: Sets what is returned when the object is called
     public String toString() {
         return "Code: " + this.module.getCode() + ", Year: " + this.year + ", Term: " + this.term;
     }
     //Constructor: Create module object
     public Module(int year, byte term, ModuleDescriptor module){
         this.year = year;
         this.term = term;
         this.module = module;
     }

 }
```

5 University.java

```java
import java.util.ArrayList;
public class University {

   private ModuleDescriptor[] moduleDescriptors;

   private Student[] students;

   private Module[] modules;

   //Method: Returns the number of students at the university
   public int getTotalNumberStudents() {
      return this.students.length;
   }

   //Method: Returns the best student
   public Student getBestStudent() {
      Student bestStudent = this.students[0];
        for (int i = 0; i < this.students.length; i++) {
            if (this.students[i].getGpa() > bestStudent.getGpa()) {
                bestStudent = this.students[i];
            }
        }
        return bestStudent;
   }

   //Method: Returns the best module
   public Module getBestModule() {
      Module bestModule = this.modules[0];
        for (int i = 0; i < this.modules.length; i++) {
            if (this.modules[i].getFinalAverageGrade() > bestModule.getFinalAverageGrade()) {
                bestModule = this.modules[i];
            }
        }
        return bestModule;
    }

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

    public static void main(String[] args) {
        //Arrays for weights of assesments per module
        double[] ECM0002Weight = {0.1, 0.3, 0.6};
        double[] ECM1400Weight = {0.25, 0.25, 0.25, 0.25};
        double[] ECM1406Weight = {0.25, 0.25, 0.5};
        double[] ECM1410Weight = {0.2, 0.3, 0.5};
        double[] BEM2027Weight = {0.1, 0.3, 0.3, 0.3};
        double[] PHY2023Weight = {0.4, 0.6};

        //Module Descriptor array containing all module objects
        ModuleDescriptor[] moduleDescriptors = {new ModuleDescriptor("ECM0002", "Real World Mathematics",
            ECM0002Weight),
                                             new ModuleDescriptor("ECM1400", "Programming", ECM1400Weight),
                                             new ModuleDescriptor("ECM1406", "Data Structures",
                                                 ECM1406Weight),
                                             new ModuleDescriptor("ECM1410", "Object-Oriented Programming",
                                                 ECM1410Weight),
                                             new ModuleDescriptor("BEM2027", "Information Systems",
                                                 BEM2027Weight),
                                             new ModuleDescriptor("PHY2023", "Thermal Physics",
                                                 PHY2023Weight)};

        //Student array containing all student objects
        Student[] students = {new Student(1000, "Ana", ’F’),
                                    new Student(1001, "Oliver", ’M’),
                                    new Student(1002, "Mary", ’F’),
                                    new Student(1003, "John", ’M’),
                                    new Student(1004, "Noah", ’M’),
                                    new Student(1005, "Chico", ’M’),
                                    new Student(1006, "Maria", ’F’),
                                    new Student(1007, "Mark", ’X’),
                                    new Student(1008, "Lia", ’F’),
                                    new Student(1009, "Rachel", ’F’)};

        //Modules array containing data about all modules and students
        String[][] modules = {{"1000", "ECM1400", "2019", "1", "9, 10, 10, 10"},
                                   {"1001", "ECM1400", "2019", "1", "8, 8, 8, 9"},
                                   {"1002", "ECM1400", "2019", "1", "5, 5, 6, 5"},
                                   {"1003", "ECM1400", "2019", "1", "6, 4, 7, 9"},
                                   {"1004", "ECM1400", "2019", "1", "10, 9, 10, 9"},
                                   {"1005", "PHY2023", "2019", "1", "9, 9"},
                                   {"1006", "PHY2023", "2019", "1", "6, 9"},
                                   {"1007", "PHY2023", "2019", "1", "5, 6"},
                                   {"1008", "PHY2023", "2019", "1", "9, 7"},
                                   {"1009", "PHY2023", "2019", "1", "8, 5"},
                                   {"1000", "BEM2027", "2019", "2", "10, 10, 9.5, 10"},
                                   {"1001", "BEM2027", "2019", "2", "7, 8.5, 8.2, 8"},
                                   {"1002", "BEM2027", "2019", "2", "6.5, 7, 5.5, 8.5"},
                                   {"1003", "BEM2027", "2019", "2", "5.5, 5, 6.5, 7"},
                                   {"1004", "BEM2027", "2019", "2", "7, 5, 8, 6"},
                                   {"1005", "ECM1400", "2019", "2", "9, 10, 10, 10"},
                                   {"1006", "ECM1400", "2019", "2", "8, 8, 8, 9"},
                                   {"1007", "ECM1400", "2019", "2", "5, 5, 6, 5"},
                                   {"1008", "ECM1400", "2019", "2", "6, 4, 7, 9"},
                                   {"1009", "ECM1400", "2019", "2", "10, 9, 8, 9"},
                                   {"1000", "ECM1406", "2020", "1", "10, 10, 10"},
                                   {"1001", "ECM1406", "2020", "1", "8, 7.5, 7.5"},
                                   {"1002", "ECM1406", "2020", "1", "9, 7, 7"},
                                   {"1003", "ECM1406", "2020", "1", "9, 8, 7"},
                                   {"1004", "ECM1406", "2020", "1", "2, 7, 7"},
                                   {"1005", "ECM1406", "2020", "1", "10, 10, 10"},
                                   {"1006", "ECM1406", "2020", "1", "8, 7.5, 7.5"},
                                   {"1007", "ECM1406", "2020", "1", "10, 10, 10"},
                                   {"1008", "ECM1406", "2020", "1", "9, 8, 7"},
                                   {"1009", "ECM1406", "2020", "1", "8, 9, 10"},
                                   {"1000", "ECM1410", "2020", "1", "10, 9, 10"},
                                   {"1001", "ECM1410", "2020", "1", "8.5, 9, 7.5"},
                                   {"1002", "ECM1410", "2020", "1", "10, 10, 5.5"},
                                   {"1003", "ECM1410", "2020", "1", "7, 7, 7"},
                                   {"1004", "ECM1410", "2020", "1", "5, 6, 10"},
                                   {"1005", "ECM0002", "2020", "2", "8, 9, 8"},
                                   {"1006", "ECM0002", "2020", "2", "6.5, 9, 9.5"},
                                   {"1007", "ECM0002", "2020", "2", "8.5, 10, 8.5"},
                                   {"1008", "ECM0002", "2020", "2", "7.5, 8, 10"},
                                   {"1009", "ECM0002", "2020", "2", "10, 6, 10"}};

        //Creating an array called oo_students that contains all module objects
        ArrayList<Module> moduleList = new ArrayList<>();
        int count = 0;
        boolean contained = false;
        double grade = 0;
        for (int i = 0; i < modules.length; i++) {
            for (int j = 0; j < moduleDescriptors.length; j++){
                if (moduleDescriptors[j].getCode() == modules[i][1]) {
                    count = j;
                }
            }
            for (int q = 0; q < moduleList.size(); q++) {
                if (moduleList.get(q).getYear() == Integer.parseInt(modules[i][2]) &&
                    moduleList.get(q).getTerm() == Byte.parseByte(modules[i][3]) &&
                    moduleList.get(q).getModule() == moduleDescriptors[count]) {
                    contained = true;
                }
            }
            if (contained == false) {
                moduleList.add(new Module(Integer.parseInt(modules[i][2]), Byte.parseByte(modules[i][3]),
                    moduleDescriptors[count]));
            }
            contained = false;
        }
        Module[] oo_modules = moduleList.toArray(new Module[0]);

        //Creates a records ArrayList containing all student record objects
        String marks = "";
        ArrayList<Double> marksDouble = new ArrayList<>();
        ArrayList<Double> markProportion = new ArrayList<>();
        ArrayList<StudentRecord> records = new ArrayList<>();
        double finalScore = 0;
        Module temp = oo_modules[1];
        for (int i = 0; i < students.length; i++) {
            for (int j = 0; j < modules.length; j++) {
                if (Integer.parseInt(modules[j][0]) == students[i].getId()) {
                    marks = modules[j][4];
                    String[] stringArray = marks.split(", ");
                    for (int q = 0; q < stringArray.length; q++) {
                        marksDouble.add(Double.parseDouble(stringArray[q]));
                    }
                    for (int q = 0; q < oo_modules.length; q++) {

```

```java
                    if (oo_modules[q].getModule().getCode() == modules[j][1]) {
                        temp = oo_modules[q];
                    }
                }
                double[] weight = temp.getModule().getContinuousAssignmentWeights();
                double[] AssesmentMarks = new double[marksDouble.size()];
                for (int q = 0; q < marksDouble.size(); q++) {
                    markProportion.add(weight[q] * marksDouble.get(q));
                    finalScore = finalScore + (weight[q] * marksDouble.get(q));
                    AssesmentMarks[q] = marksDouble.get(q);
                }
                marksDouble.clear();
                records.add(new StudentRecord(students[i], temp, AssesmentMarks, finalScore));
                finalScore = 0;
            }
        }
    }

    //Assigns all student records to correct students
    int studentID = 0;
    ArrayList<StudentRecord> studentRecordList = new ArrayList<>();
    for (int i = 0; i < students.length; i++) {
        studentID = students[i].getId();
        for (int j = 0; j < records.size(); j++) {
            if (studentID == records.get(j).getStudent().getId()) {
                studentRecordList.add(records.get(j));
            }
        }
        StudentRecord[] studentRecord = studentRecordList.toArray(new StudentRecord[0]);
        students[i].setRecord(studentRecord);
        studentRecordList.clear();
    }

    //Assigns all module records to correct module
    String courseCode = "";
    int year = 0;
    byte term = 0;
    ArrayList<StudentRecord> moduleRecordList = new ArrayList<>();
    for (int i = 0; i < oo_modules.length; i++) {
        courseCode = oo_modules[i].getModule().getCode();
        year = oo_modules[i].getYear();
        term = oo_modules[i].getTerm();
        for (int j = 0; j < records.size(); j++) {
            if (courseCode == records.get(j).getModule().getModule().getCode() && year ==
                 records.get(j).getModule().getYear() && term == records.get(j).getModule().getTerm()) {
                moduleRecordList.add(records.get(j));
            }
        }
        StudentRecord[] moduleRecord = moduleRecordList.toArray(new StudentRecord[0]);
        oo_modules[i].setRecords(moduleRecord);
    }

    //Sets all students GPA
    double total = 0;
    double gpa = 0;
```

```java
        for (int i = 0; i < students.length; i++) {
            total = 0;
            for (int j = 0; j < students[i].getRecords().length; j++) {
                total = total + students[i].getRecords()[j].getFinalScore();
            }
            gpa = total/(students[i].getRecords().length);
            students[i].setGpa(gpa);
        }

        //Sets all modules final average grade
        total = 0;
        double finalAverageGrade = 0;
        for (int i = 0; i < oo_modules.length; i++) {
            total = 0;
            for (int j = 0; j < oo_modules[i].getRecords().length; j++) {
                total = total + oo_modules[i].getRecords()[j].getFinalScore();
            }
            finalAverageGrade = total/(students[i].getRecords().length);
            oo_modules[i].setFinalAverageGrade(gpa);
        }

        //Sets is above average for all students
        for (int i = 0; i < students.length; i++) {
            for (int q = 0; q < students[i].getRecords().length; q++) {
                if (students[i].getRecords()[q].getFinalScore() >
                    students[i].getRecords()[q].getModule().getFinalAverageGrade()) {
                    students[i].getRecords()[q].setIsAboveAverage(true);
                } else {
                    students[i].getRecords()[q].setIsAboveAverage(false);
                }
            }
        }

        // Creating the university object
        University university;
        university = new University(moduleDescriptors, students, oo_modules);

        // Print Reports
        System.out.println("The UoK has " + university.getTotalNumberStudents() + " students.");

        // best module
        System.out.println("The best module is:");
        System.out.println(university.getBestModule());

        // best student
        System.out.println("The best student is:");
        System.out.println(university.getBestStudent().printTranscript());
    }
}
```