```java
import java.util.Arrays;

public class Module {
  private int year;
  private byte term;
  private ModuleDescriptor module;
  private StudentRecord[] records;
  private double finalAverageGrade;

  public int getYear() { return year; }
  public byte getTerm() { return term; }
  public ModuleDescriptor getModule() { return module; }
  public StudentRecord[] getRecords() { return records; }
  public double getFinalAverageGrade() { return finalAverageGrade; }

  /**
   * Sets the array of student records for each student that takes this module
   * @param records array of student records for each student that takes this module
   */
  public void setRecords(StudentRecord[] records) {
    for (int i=0; i<records.length; i++){
      for (int j=0; j<records.length; j++){
        if (records[i].equals(records[j]) && i != j){
          System.err.println("""
              Error has occurred.\s
              CheckList:
              . A module descriptor can only be offered once per year and term""");
          System.exit(1);
        }
      }
    }
    this.records = records;
    setFinalAverageGrade();
  }

  /**
   * The final average grade is the mean of the student record final scores for this module
   */
  public void setFinalAverageGrade() {
    for (StudentRecord record : records) finalAverageGrade += record.getFinalScore();
    finalAverageGrade /= records.length;
  }

  /**
   * Module constructor which sets the initial information about the module
   * @param year The Year the module is taken in
   * @param term The term the module is taken in
   * @param module The module description
   */
  public Module(int year, byte term, ModuleDescriptor module){
    this.year = year;
    this.term = term;
    this.module = module;
  }

  @Override
  public String toString() {
    return "Module{" +
        "year=" + year +
                ", term=" + term +
                ", module=" + module +
                ", records=" + java.util.Arrays.toString(records) +
                ", finalAverageGrade=" + finalAverageGrade +
                '}';
    }
}

```

```java
import java.util.Arrays;

public class Student {

  private int id;
  private String name;
  private char gender;
  private double gpa;
  private StudentRecord[] records;

  public int getId() { return id; }
  public String getName() { return name; }
  public char getGender() { return gender; }
  public double getGpa() { return gpa; }
  public StudentRecord[] getRecords() { return records; }


  /**
   * Sets GPA for student
   */
  public void setGpa() {
    for (StudentRecord record : records) {
      gpa += record.getFinalScore();
    }
    gpa /= records.length;
  }

  /**
   * Sets the array of student records for each module the student takes
   * @param records Array of student records for each module the student takes
   */
  public void setRecords(StudentRecord[] records) {
    for (int i=0; i<records.length; i++){
      for (int j=0; j<records.length; j++){
        if (records[i].equals(records[j]) && i != j){
          System.err.println("""
              Error has occurred.\s
              CheckList:
              . A student can only have one record per module""");
          System.exit(1);
        }
      }
    }
    this.records = records;
    orderRecords();
    setGpa();
  }

  /**
   * Orders the student records by year and term
   */
  public void orderRecords() {
    for (int i=0; i<records.length-1; i++){
      if ((records[i].getModule().getYear() > records[i+1].getModule().getYear
()) || ((records[i].getModule().getYear() == records[i+1].getModule().getYear
()) && records[i].getModule().getTerm() > records[i+1].getModule().getTerm
()))) {
        StudentRecord temp = records[i];
        records[i] = records[i + 1];
        records[i + 1] = temp;
      }
    }
  }

  /**
   * This generates a transcript containing all student records, grouped by year and term.
   * @return This returns the transcript as a string
   */
  public String printTranscript() {
    final int RL = records.length;
    String[] studentDetails = new String[RL];
    StringBuilder studentRecord = new StringBuilder();

    for (int i = 0; i < RL-1; i++) {
      if ((records[i].getModule().getYear() != records[i+1].getModule().getYear()) || ((records[i].getModule().getYear() == records[i+1].getModule().getYear()) && (records[i].getModule().getTerm() != records[i+1].getModule().getTerm()))) {
        studentDetails[i] = "| " + records[i].getModule().getYear() + " | " + records[i].getModule().getTerm() +
            " | " + records[i].getModule().getModule().getCode() + " | " + records[i].getFinalScore() + " |\n\n";
      } else {studentDetails[i] = "| " + records[i].getModule().getYear() + " | " + records[i].getModule().getTerm() +
          " | " + records[i].getModule().getModule().getCode() + " | " + records[i].getFinalScore() + " |\n";}
    }
    studentDetails[RL-1] = "| " + records[RL-1].getModule().getYear() + " | " + records[RL-1].getModule().getTerm() +
        " | " + records[RL-1].getModule().getModule().getCode() + " | " + records[RL-1].getFinalScore() + " |";
    for (String studentDetail : studentDetails) { studentRecord.append(studentDetail); }

    return "University of Knowledge - Official Transcript\n\n \nID: %d\nName: %s\nGPA: %s\n\n%s".formatted(id, name, gpa, studentRecord);
  }

  /**
   * Student constructor which sets the initial information about the student
   * @param id Student id
   * @param name Student name
   * @param gender Student gender
   */
  public Student(int id, String name, char gender){
    if ("MFX".contains(Character.toString(gender)) && id > 0  && !name.isEmpty()){
      this.id = id;
      this.name = name;
      this.gender = gender;
    } else {
      System.err.println("Error has occurred. \n" +
                 "CheckList:\n" +
                 ". ID and name cannot be null\n" +
                 ". ID must be unique\n" +
                 ". Gender must be 'M', 'F', 'X' or empty");
      System.exit(1);
    }
  }

  /**
   * Student constructor which sets the initial information about the student (without gender)
   * @param id Student id
   * @param name Student name
```

```
     */
    public Student(int id, String name){
      if (id > 0 && !name.isEmpty()){
        this.id = id;
        this.name = name;
      } else {
        System.err.println("Error has occurred. \n" +
                   "CheckList:\n" +
                   " . ID and name cannot be null\n" +
                   " . ID must be unique");
        System.exit(1);
      }
    }


    //@Override
    public String toString() {
      return "Student{" +
          "id=" + id +
          ", name='" + name + '\'' +
          ", gender=" + gender +
          ", gpa=" + gpa +
          ", records=" + Arrays.toString(getRecords()) +
          '}';
    }
  }

```

```java
import java.util.Arrays;

public class University {
    private ModuleDescriptor[] moduleDescriptors;
    private Student[] students;
    private Module[] modules;

    public void setModuleDescriptors(ModuleDescriptor[] moduleDescriptors) { this.moduleDescriptors = moduleDescriptors; }
    public void setStudents(Student[] students) { this.students = students; }
    public void setModules(Module[] modules) { this.modules = modules; }

    public ModuleDescriptor[] getModuleDescriptors() { return moduleDescriptors; }
    public Module[] getModules() { return modules; }
    public Student[] getStudents() { return students; }

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

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

    /**
     * @return The module with the highest average score.
     */
    public String getBestModule() {
        Module best = modules[0];
        for (Module module:modules) if (module.getFinalAverageGrade() > best.getFinalAverageGrade()) best = module;
        return "Year: %d, Term: %s, Module code: %s, Average grade: %s".formatted(best.getYear(), best.getTerm(), best.getModule().getCode(), best.getFinalAverageGrade());
    }

    /**
     * This method checks for duplicate student IDs
     * @param students Array of students
     * @param student Current student being initialised
     */
    public void duplicateId(Student[] students, Student student){
        for (Student individual : students) {
            if (individual != null && student.getId() == individual.getId() && !student.getName().equals(individual.getName())) {
                System.err.println("Error has occurred.\nTwo students cannot have the same ID");
                System.exit(1);
            }
        }
    }

    /**
```

```java
     * This method checks for duplicate module descriptor codes
     * @param moduleDescriptors Array of module descriptors
     * @param moduleDescriptor Current module descriptor being initialised
     */
    public void duplicateCode(ModuleDescriptor[] moduleDescriptors, ModuleDescriptor moduleDescriptor){
        for (ModuleDescriptor module : moduleDescriptors){
            if (module != null && moduleDescriptor.getCode().equals(module.getCode()) && !moduleDescriptor.getName().equals(module.getName())) {
                System.err.println("Error has occurred.\nTwo module descriptors cannot have the same code");
                System.exit(1);
            }
        }
    }

    /**
     * This initialises all the data
     * @param args
     */
    public static void main(String[] args) {
        University university = new University();

        Student[] students = new Student[10];

        Module[] modules = new Module[7];

        ModuleDescriptor[] moduleDescriptors = new ModuleDescriptor[6];

        // initialises module descriptors
        university.duplicateCode(moduleDescriptors, moduleDescriptors[0] = new ModuleDescriptor("ECM0002","Real World Mathematics", new double[] {0.1, 0.3, 0.6}));
        university.duplicateCode(moduleDescriptors, moduleDescriptors[1] = new ModuleDescriptor("ECM1400","Programming", new double[] {0.25, 0.25, 0.25, 0.25}));
        university.duplicateCode(moduleDescriptors, moduleDescriptors[2] = new ModuleDescriptor("ECM1406","Data Structures", new double[] {0.25, 0.25, 0.5}));
        university.duplicateCode(moduleDescriptors, moduleDescriptors[3] = new ModuleDescriptor("ECM1410","Object-Oriented Programming", new double[] {0.2, 0.3, 0.5}));
        university.duplicateCode(moduleDescriptors, moduleDescriptors[4] = new ModuleDescriptor("BEM2027","Information Systems ", new double[] {0.1, 0.3, 0.3, 0.3}));
        university.duplicateCode(moduleDescriptors, moduleDescriptors[5] = new ModuleDescriptor("PHY2023","Thermal Physics", new double[] {0.4, 0.6}));

        // initialises students
        university.duplicateId(students, students[0] = new Student(1000, "Ana", 'F'));
        university.duplicateId(students, students[1] = new Student(1001, "Oliver", 'M'));
        university.duplicateId(students, students[2] = new Student(1002, "Mary", 'F'));
        university.duplicateId(students, students[3] = new Student(1003, "John", 'M'));
        university.duplicateId(students, students[4] = new Student(1004, "Noah", 'M'));
        university.duplicateId(students, students[5] = new Student(1005, "Chico", 'M'));
        university.duplicateId(students, students[6] = new Student(1006, "Maria", 'F'));
        university.duplicateId(students, students[7] = new Student(1007, "Mark", 'X'));
        university.duplicateId(students, students[8] = new Student(1008, "Lia", 'F'));
        university.duplicateId(students, students[9] = new Student(1009, "Rachel", 'F'));

        // initialises module
        modules[0] = new Module(2019, (byte) 1, moduleDescriptors[1]);
        modules[1] = new Module(2019, (byte) 1, moduleDescriptors[5]);
        modules[2] = new Module(2019, (byte) 2, moduleDescriptors[4]);
        modules[3] = new Module(2019, (byte) 2, moduleDescriptors[1]);
        modules[4] = new Module(2020, (byte) 1, moduleDescriptors[2]);
        modules[5] = new Module(2020, (byte) 1, moduleDescriptors[3]);
        modules[6] = new Module(2020, (byte) 2, moduleDescriptors[0]);

        // initialises student records
        StudentRecord anaECM1400 = new StudentRecord(students[0], modules[0], new double[] {9, 10, 10, 10});
        StudentRecord oliverECM1400 = new StudentRecord(students[1], modules[0], new double[] {8, 8, 8, 9});
        StudentRecord maryECM1400 = new StudentRecord(students[2], modules[0], new double[] {5, 5, 6, 5});
        StudentRecord johnECM1400 = new StudentRecord(students[3], modules[0], new double[] {6, 4, 7, 9});
        StudentRecord noahECM1400 = new StudentRecord(students[4], modules[0], new double[] {10, 9, 10, 9});

        StudentRecord chicoPHY2023 = new StudentRecord(students[5], modules[1], new double[] {9, 9});
        StudentRecord mariaPHY2023 = new StudentRecord(students[6], modules[1], new double[] {6, 9});
        StudentRecord markPHY2023 = new StudentRecord(students[7], modules[1], new double[] {5, 6});
        StudentRecord liaPHY2023 = new StudentRecord(students[8], modules[1], new double[] {9, 7});
        StudentRecord rachelPHY2023 = new StudentRecord(students[9], modules[1], new double[] {8, 5});

        StudentRecord anaBEM2027 = new StudentRecord(students[0], modules[2], new double[] {10, 10, 9.5, 10});
        StudentRecord oliverBEM2027 = new StudentRecord(students[1], modules[2], new double[] {7, 8.5, 8.2, 8});
        StudentRecord maryBEM2027 = new StudentRecord(students[2], modules[2], new double[] {6.5, 7.0, 5.5, 8.5});
        StudentRecord johnBEM2027 = new StudentRecord(students[3], modules[2], new double[] {5.5, 5, 6.5, 7});
        StudentRecord noahBEM2027 = new StudentRecord(students[4], modules[2], new double[] {7, 5, 8, 6});

        StudentRecord chicoECM1400 = new StudentRecord(students[5], modules[3], new double[] {9, 10, 10, 10});
        StudentRecord mariaECM1400 = new StudentRecord(students[6], modules[3], new double[] {8, 8, 8, 9});
        StudentRecord markECM1400 = new StudentRecord(students[7], modules[3], new double[] {5, 5, 6, 5});
        StudentRecord liaECM1400 = new StudentRecord(students[8], modules[3], new double[] {6, 4, 7, 9});
        StudentRecord rachelECM1400 = new StudentRecord(students[9], modules[3], new double[] {10, 9, 8, 9});

        StudentRecord anaECM1406 = new StudentRecord(students[0], modules[4], new double[] {10, 10, 10});
        StudentRecord oliverECM1406 = new StudentRecord(students[1], modules[4], new double[] {8, 7.5, 7.5});
        StudentRecord maryECM1406 = new StudentRecord(students[2], modules[4], new double[] {9, 9, 7});
        StudentRecord johnECM1406 = new StudentRecord(students[3], modules[4], new double[] {9, 8, 7});
        StudentRecord noahECM1406 = new StudentRecord(students[4], modules[4], new double[] {2, 7, 7});
        StudentRecord chicoECM1406 = new StudentRecord(students[5], modules[4], new double[] {10, 10, 10});
        StudentRecord mariaECM1406 = new StudentRecord(students[6], modules[4], new double[] {8, 7.5, 7.5});
        StudentRecord markECM1406 = new StudentRecord(students[7], modules[4], new double[] {10, 10, 10});
        StudentRecord liaECM1406 = new StudentRecord(students[8], modules[4], new double[] {9, 8, 7});
        StudentRecord rachelECM1406 = new StudentRecord(students[9], modules[4], new double[] {8, 9, 10});

        StudentRecord anaECM1410 = new StudentRecord(students[0], modules[5], new double[] {10, 9, 10});
        StudentRecord oliverECM1410 = new StudentRecord(students[1], modules[5], new double[] {8.5, 9, 7.5});
        StudentRecord maryECM1410 = new StudentRecord(students[2], modules[5], new double[] {10, 10, 5.5});
        StudentRecord johnECM1410 = new StudentRecord(students[3], modules[5], new double[] {7, 7, 7});
        StudentRecord noahECM1410 = new StudentRecord(students[4], modules[5], new double[] {5, 6, 10});

        StudentRecord chicoECM0002 = new StudentRecord(students[5], modules[6], new double[] {8, 9, 8});
        StudentRecord mariaECM0002 = new StudentRecord(students[6], modules[6], new double[] {6.5, 9, 9.5});
        StudentRecord markECM0002 = new StudentRecord(students[7], modules[6], new double[] {8.5, 10, 8.5});
        StudentRecord liaECM00002 = new StudentRecord(students[8], modules[6], new double[] {7.5, 8, 10});
        StudentRecord rachelECM0002 = new StudentRecord(students[9], modules[6], new double[] {10, 6, 10});

        // sets the student records of each module each student takes
        students[0].setRecords(new StudentRecord[] {anaECM1400, anaBEM2027, anaECM1406, anaECM1410});
        students[1].setRecords(new StudentRecord[] {oliverECM1400, oliverBEM2027, oliverECM1406, oliverECM1410});
        students[2].setRecords(new StudentRecord[] {maryECM1400, maryBEM2027, maryECM1406, maryECM1410});
        students[3].setRecords(new StudentRecord[] {johnECM1400, johnBEM2027, johnECM1406, johnECM1410});
        students[4].setRecords(new StudentRecord[] {noahECM1400, noahBEM2027, noahECM1406, noahECM1410});
        students[5].setRecords(new StudentRecord[] {chicoPHY2023, chicoECM1400, chicoECM1406, chicoECM0002});
        students[6].setRecords(new StudentRecord[] {mariaPHY2023, mariaECM1400, mariaECM1406, mariaECM0002});
        students[7].setRecords(new StudentRecord[] {markPHY2023, markECM1400, markECM1406, markECM0002});
        students[8].setRecords(new StudentRecord[] {liaPHY2023, liaECM1400, liaECM1406, liaECM00002});
        students[9].setRecords(new StudentRecord[] {rachelPHY2023, rachelECM1400, rachelECM1406, rachelECM0002});

        // sets the student records of each student that take each module
        modules[0].setRecords(new StudentRecord[]{anaECM1400, oliverECM1400, maryECM1400, johnECM1400, noahECM1400});
        modules[1].setRecords(new StudentRecord[]{chicoPHY2023, mariaPHY2023, markPHY2023, liaPHY2023, rachelPHY2023});
        modules[2].setRecords(new StudentRecord[]{anaBEM2027, oliverBEM2027, maryBEM2027, johnBEM2027, noahBEM2027});
        modules[3].setRecords(new StudentRecord[]{chicoECM1400, mariaECM1400, markECM1400, liaECM1400, rachelECM1400});
        modules[4].setRecords(new StudentRecord[]{anaECM1406, oliverECM1406, maryECM1406, johnECM1406, noahECM1406, chicoECM1406,mariaECM1406, markECM1406, liaECM1406, rachelECM1406});
        modules[5].setRecords(new StudentRecord[]{anaECM1410, oliverECM1410, maryECM1410, johnECM1410, noahECM1410});
        modules[6].setRecords(new StudentRecord[]{chicoECM0002, mariaECM0002, markECM0002, liaECM00002, rachelECM0002});

        // sets data to university object
        university.setModuleDescriptors(moduleDescriptors);
        university.setStudents(students);
        university.setModules(modules);

        // sets if a student is above average in the module
        for (int i=0; i<university.getStudents().length; i++)
            for (int j = 0; j < university.getStudents()[i].getRecords().length; j++)
                university.getStudents()[i].getRecords()[j].setAboveAverage();

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

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

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

```java
import java.util.Arrays;

public class StudentRecord {
  private Student student;
  private Module module;
  private double[] marks;
  private double finalScore;
  private Boolean isAboveAverage;

  public Student getStudent() { return student; }
  public Module getModule() { return module; }
  public double[] getMarks() { return marks; }
  public double getFinalScore() { return finalScore; }
  public Boolean getAboveAverage() { return isAboveAverage; }

  /**
   * Sets the final score the student achieved in a particular module
   */
  public void setFinalScore() {
    for (int i=0; i<marks.length; i++){
      finalScore += marks[i] * getModule().getModule().getContinuousAssignmentWeights()[i];
    }
  }

  /**
   * Calculates if the student is above average in a particular module
   */
  public void setAboveAverage() {
    isAboveAverage = finalScore > module.getFinalAverageGrade();
  }

  /**
   * Student constructor which sets the initial information about the student record
   * @param student A student
   * @param module A particular module the student takes
   * @param marks The marks the student got for each assessment in that module
   */
  public StudentRecord(Student student, Module module, double[] marks){
    for (double mark : marks)
      if (!(mark >= 0 && mark <= 10)) {
        System.err.println("""
            Error has occurred.\s
            CheckList:
             . Marks must range between 0 and 10""");
        System.exit(1);
      }
    this.student = student;
    this.module = module;
    this.marks = marks;
    setFinalScore();
    }

  @Override
  public String toString() {
    return "StudentRecord{" +
        "student=" + student.getName() +
        ", module=" + module.getModule().getName() +
        ", marks=" + java.util.Arrays.toString(marks) +
        ", finalScore=" + finalScore +
        ", isAboveAverage=" + isAboveAverage +
        '}';
    }
}

```

```java
import java.util.ArrayList;
import java.util.Arrays;

public class ModuleDescriptor {
    private String code;
    private String name;
    private double[] continuousAssignmentWeights;

    public String getCode() { return code; }
    public String getName() { return name; }
    public double[] getContinuousAssignmentWeights() { return continuousAssignmentWeights; }

    /**
     * Module descriptor constructor which sets the information about the modules
     * @param code Module descriptor code
     * @param name Module descriptor name
     * @param continuousAssignmentWeights Module descriptor continuous assignment weights
     */
    public ModuleDescriptor(String code, String name, double[] continuousAssignmentWeights) {
        double sum = 0;
        for (double weight:continuousAssignmentWeights){ sum += weight; }
        if (!(code.isEmpty() || name.isEmpty()) && sum == 1) {
            this.code = code;
            this.name = name;
            this.continuousAssignmentWeights = continuousAssignmentWeights;
        } else {
            System.err.println("""
                    Error ha occurred.\s
                    CheckList:
                     . Code and name cannot be null
                     . Code must be unique
                     . Continuous Assessment weights must sum up to 1, and must be non-negative""");
            System.exit(1);
        }
    }

    @Override
    public String toString() {
        return "ModuleDescriptor{" +
               "code='" + code + '\'' +
               ", name='" + name + '\'' +
               ", continuousAssignmentWeights=" + Arrays.toString(continuousAssignmentWeights) +
               '}';
    }
}

```