```java
import java.util.Arrays;
import java.text.DecimalFormat;

/**
 * Represents a student at the university, contains their id, name, gender(optional),
 * grade point average and their records for each module
 */

public class Student {

    private int id;

    private String name;

    private char gender;

    private double gpa;

    private StudentRecord[] records;


    /**
     * Constructor for student with gender
     *
     * @param id
     * @param name
     * @param gender must be M, F, or X. No gender can also be given using the other constructor
     * @param uni    is passed so that the constructor can check that the id doesn't exist in the university already
     */

    public Student(int id, String name, char gender, University uni) {

        // Checks that the ID is unique for the university it has been submitted for
        if (uni.checkId(id) == true) {
            throw new IllegalArgumentException("ID must be unique");
        }

        //Checks that name isn't empty
        if (name.isEmpty()) {
            throw new IllegalArgumentException("Name cannot be empty");
        }

        //Checks that gender is one of the recognised characters
        else if (gender != 'M' && gender != 'F' && gender != 'X') {
            throw new IllegalArgumentException("Not a valid gender choice");
        }

        // Initializes variables
        this.id = id;
        this.name = name;
        this.gender = gender;

        // Initilizes blank records array, to be added to later
        StudentRecord[] newArray = {};
        this.records = newArray;
    }

    /**
     * Constructor for student without gender
     *
     * @param id
     * @param name
     * @param uni  is passed so that the constructor can check that the id doesn't exist in the university already
     */
    public Student(int id, String name, University uni) {

```

```
67          // Validates that the ID is unique for the university it has been submitted for
68          if (uni.checkId(id) == true) {
69              throw new IllegalArgumentException("ID must be unique");
70          }
71
72          //Validates that name isn't empty
73          if (name.isEmpty()) {
74              throw new IllegalArgumentException("Name cannot be empty");
75          }
76
77
78          // Initializes variables
79          this.id = id;
80          this.name = name;
81
82          // Initilizes blank records array, to be added to later
83          StudentRecord[] newArray = {};
84          this.records = newArray;
85      }
86
87      /**
88       * Adds a new StudentRecord to the records array
89       *
90       * @param newRecord StudentRecord to be added
91       */
92      public void addRecord(StudentRecord newRecord) {
93
94          // Adds new record to records array
95          StudentRecord[] newRecords = Arrays.copyOf(this.records, this.records.length +
            1);
96          newRecords[newRecords.length - 1] = newRecord;
97          this.records = newRecords;
98
99
100         // As new record created, recalculate gpa
101         double total = 0;
102         double count = 0;
103         for (StudentRecord record : records) {
104             double finalScore = record.getFinalScore();
105             total += finalScore;
106             count++;
107         }
108
109         this.gpa = total / count;
110     }
111
112     /**
113      * Returns transcript for student
114      *
115      * @return script formatted script containing student details and their records
116      */
117     public String printTranscript() {
118         //Creates decimal formatter
119         DecimalFormat df2 = new DecimalFormat("#.##");
120
121         //Creates initial part of transcript
122         String script = "University of Knowledge Official Transcript\n\n\nID: "
123                 + Integer.toString(id) + "\nName: " + name + "\nGPA: " + df2.format(gpa)
124                 + "\n\n";
125
126         //Creates line for each student record
127         for (StudentRecord record : records) {
128             Module module = record.getModule();
129             int year = module.getYear();
130             byte term = module.getTerm();
131             String moduleCode = module.getModuleCode();
132             double finalScore = record.getFinalScore();
133
134             script = script + "| " + Integer.toString(year) + " | " +
```

```
                Byte.toString(term) +
135                     " | " + moduleCode + " | " + df2.format(finalScore) + " | \n";
136         }
137
138         return script;
139     }
140
141     public int getId() {
142         return id;
143     }
144
145     public double getGpa() {
146         return gpa;
147     }
148
149     public String getName() {
150         return name;
151     }
152
153     /**
154      * Invokes the setAboveAverage method for each record, to find if the student is
         above average
155      * in that module.
156      */
157     public void setAboveAverage() {
158         for (StudentRecord record : records) {
159             record.setAboveAverage();
160         }
161     }
162
163 }
```