File - C:\Users\nishi\Desktop\NEW JAVA\ECM1400 CA1 OOP\src\com\company\Main.java

```java
package com.company;
//PLEASE NOTE : IF THERE IS AN ISSUE RUNNING THE CODE PLEASE CONTACT ME AT
//NA510@EXETER.AC.UK OR NISHILAMIN1213@GMAIL.COM
//SO WE CAN TRY AND RECTIFY THE ISSUE
//THANKS FROM NISHIL

public class Main {

    public static void main(String[] args) {
    // write your code here
    }
}

```

File - C:\Users\nishi\Desktop\NEW JAVA\ECM1400 CA1 OOP\src\com\company\Module.java

```java
package com.company;

import java.util.Arrays;

public class Module {

    private int year;
    private byte term;
    private ModuleDescriptor module;
    private StudentRecord[] records;
    private double finalAverageGrade;
    private static Module[] createdModules = new Module[]{};

    /**
     * Constructor for Module, Used to create a new Module when called
     * @param moduleDesc - Module Descriptor to base this module off
     * @param year - Year this module takes place
     * @param term - Term this module takes place
     */
    public Module(ModuleDescriptor moduleDesc, int year, byte term){
        Module[] modules = Module.createdModules;
        // Assume that the module is valid
        boolean notValid = false;

        // Make sure no other modules exist with the same descriptor, term and year
        for(Module existingModule:modules){
            // Iterate through the existing modules and check the descriptor, year and term
            if (existingModule.getModuleDescriptor() == moduleDesc && existingModule.getTerm() == term && existingModule.getYear() == year){
                notValid=true;
            }
```

```
31          }
32
33          // If the data is all valid, then finish off instantiating the object, otherwise
    throw an error.
34          if(notValid==true){
35              throw new IllegalArgumentException();
36          }else{
37              // Set variables to the correct data
38              this.year = year;
39              this.term = term;
40              this.module = moduleDesc;
41              this.records = new StudentRecord[]{};
42              Module.addModule(this);
43          }
44      }
45
46      /**
47       * This code is used to re calculate the final Average grade for this module
48       */
49      public void resetAverage(){
50          // Set the counter and total to 0
51          int counter = 0;
52          double total = 0.00;
53
54          // Iterate through all the records belonging to the module
55          for (StudentRecord rec:this.records){
56              // Add the final score of the record to the total
57              total += rec.GetFinalScore();
58              // increment the counter
59              counter++;
60          }
```

```
61          // Set the new final average grade to the total/counter (the average)
62          this.finalAverageGrade = total/counter;
63
64          //Reset all students boolean values
65          for (StudentRecord rec:this.records){
66              rec.AboveAvgCheck();
67          }
68      }
69
70      /**
71       * GETTER METHOD FOR YEAR
72       * @return the year for this module
73       */
74      public int getYear(){
75          return this.year;
76      }
77
78      /**
79       * GETTER METHOD FOR CODE
80       * @return the Module Code for this module
81       */
82      public String getCode(){
83          return this.getModuleDescriptor().getCode();
84      }
85
86      /**
87       * GETTER METHOD FOR TERM
88       * @return the Term value for this module
89       */
90      public byte getTerm(){
91          return this.term;
```

```
 92      }
 93
 94      /**
 95       * GETTER METHOD FOR FINAL AVERAGE GRADE
 96       * @return The Average Grade of all students on this module
 97       */
 98      public double getFinalAverageGrade(){
 99          return this.finalAverageGrade;
100      }
101
102      /**
103       * GETTER METHOD FOR RECORDS
104       * @return An array of all the StudentRecords for this module
105       */
106      public StudentRecord[] getRecords() {
107          return records;
108      }
109
110      /**
111       * GETTER METHOD FOR THE MODULE DESCRIPTOR
112       * @return The module descriptor for this module
113       */
114      public ModuleDescriptor getModuleDescriptor(){
115          return this.module;
116      }
117
118      /**
119       * METHOD TO ADD A RECORD TO THE METHODS ARRAY OF STUDENT RECORDS
120       * @param record - the StudentRecord we are adding to the records array
121       */
122      public void addRecord(StudentRecord record){
```

```
123         // Create new array of length one greater than the current array and make it a copy
     of the original array
124         this.records = Arrays.copyOf(this.records, this.records.length + 1);
125         // Add the new element to the last index of the new array (which should be empty)
126         this.records[this.records.length - 1] = record;
127     }
128 
129     /**
130      * STATIC METHOD TO ADD A MODULE TO THE CLASS ARRAY OF CREATED MODULES
131      * @param m - the Module to add to the static variable belonging to the class
132      * this keeps track of all previously created modules to ensure no duplicates are made
133      */
134     public static void addModule(Module m){
135         // Create new array of length one greater than the current array and make it a copy
     of the original array
136         Module.createdModules = Arrays.copyOf(Module.createdModules, Module.createdModules.
    length + 1);
137         // Add the new element to the last index of the new array (which should be empty)
138         Module.createdModules[Module.createdModules.length - 1] = m;
139     }
140 
141     @Override
142     public String toString() {
143         // ALLOWS THE OUTPUT OF MODULE OBJECTS
144         return "Module[" +
145                 "code="+ getModuleDescriptor().getCode()+
146                 ", year=" + year +
147                 ", term=" + term +
148                 ", finalAverageGrade=" + finalAverageGrade +
149                 ']';
150     }
```

```
151 }
152
```

File - C:\Users\nishi\Desktop\NEW JAVA\ECM1400 CA1 OOP\src\com\company\Student.java

```java
package com.company;

import java.util.Arrays;

public class Student {

    private int id;
    private String name;
    private char gender;
    private double gpa;
    private StudentRecord[] records;
    private static Student[] createdStudents = new Student[]{};

    /**
     * Constructor for Student, Used to create a new Student when called
     * @param id - ID of the student we are creating
     * @param name - Name of student we are creating
     * @param gender - Gender of student we are creating
     */
    public Student(int id, String name, char gender){
        Student[] students = Student.createdStudents;
        boolean not_valid = false;
        // Constructor for Student

        // ensure neither ID nor name are null (ID cannot be null due to code design)
        if(name==null){not_valid = true;}

        // check ID is not taken by any other student
        for(Student item:students){
            if(item.id == id){not_valid = true;}
        }
```

```
32
33          // ensure the gender is valid(M,F,X or Empty char)
34          if (!(gender == 'M' | gender == 'F' | gender == 'X' | gender =='\0')){not_valid =
    true;}
35
36          // If the data is all valid, then finish off instantiating the object, otherwise
    throw an error.
37          if (not_valid){
38              throw new IllegalArgumentException();
39          }else{
40              // Set variables to the correct data
41              this.id = id;
42              this.name = name;
43              this.gender = gender;
44              this.gpa = 0;
45              this.records = new StudentRecord[]{};
46              Student.addStudent(this);
47          }
48      }
49
50      /**
51       * Calculates the GPA of the student and saves it into the private attribute gpa
52       */
53      public void calcGPA(){
54          //If there are no records, then set the GPA to 0, as it cannot be null
55          if (this.records.length == 0){
56              this.gpa = 0;
57          }else{
58              //calculate GPA from records in this.records
59              // set total and counter to 0
60              double total = 0;
```

```
61              int counter = 0;
62              // Iterate through all the students records
63              for (StudentRecord record : this.records){
64                  // Add the records final score to the total
65                  total += record.GetFinalScore();
66                  // increment the counter
67                  counter++;
68              }
69              // Set the new GPA to total/counter (the average)
70              this.gpa = total/counter;
71          }
72      }
73
74      /**
75       * Enrolls the Student to a Module given the Module and the Marks the student has
   achieved
76       */
77      public void enrollTo(Module m, double[] marks) {
78          //marks is an array of doubles as found in the csv file and from inputs.
79
80          // Create the new StudentRecord
81          StudentRecord myRecord = new StudentRecord(this, m);
82
83          // Add the marks to the record
84          myRecord.AddMarks(marks);
85
86          //Add the record to the module and this student
87          this.addRecord(myRecord);
88          m.addRecord(myRecord);
89      }
90
```

```
 91      /**
 92       * GETTER METHOD FOR THE GPA
 93       * @return the GPA of this student
 94       */
 95      public double getGPA(){
 96          return this.gpa;
 97      }
 98
 99      /**
100       * Method that sorts the records into ascending order of data
101       * @return an array of all the ORDERED student records
102       */
103      public StudentRecord[] getOrderedRecords() {
104          // Make a copy of records
105
106          // Make an empty StudentRecord array of the same length as the length of the
    studenst records array
107          StudentRecord[] copy = new StudentRecord[this.records.length];
108
109          // Copy over all the records into the copy array
110          int index=0;
111          for(StudentRecord r: this.records){
112              copy[index] = r;
113              index++;
114          }
115
116          //Initialize an array res, for the result
117          StudentRecord[] res;
118
119          if (this.records.length == 1) {
120              // If the length of records is 1 then make res the records array as there is no
```

```
120  order
121              res = this.records;
122          } else {
123              res = new StudentRecord[]{};
124              while(res.length != this.records.length){
125                  // While the res array isnt full
126                  // Set pos and counter to 0, and earliest to the first value in copy
127                  StudentRecord earliest = copy[0];
128                  int pos=0;
129                  int counter = 0;
130                  // For the rest of the records in copy
131                  for (StudentRecord r : copy) {
132                      // If the year is less than the earliest
133                      if (r.getModule().getYear() < earliest.getModule().getYear()) {
134                          // make earliest the current record
135                          earliest = r;
136                          // set position to the current counter value
137                          pos = counter;
138                      }else if (r.getModule().getYear() == earliest.getModule().getYear()) {
139                          // If the years of earliest and the current vale are equal, check
    the terms
140                          if (r.getModule().getTerm() < earliest.getModule().getTerm()) {
141                              // If the current term is less than the earliest term
142                              // make earliest the current record
143                              earliest = r;
144                              // set position to the current counter value
145                              pos = counter;
146                          }
147                      }
148                      // Increment counter
149                      counter++;
```

```
150                 }
151                 // Add earliest to result
152                 res = StudentRecord.append(res, earliest);
153                 // remove earliest from copy using the pos
154                 copy = StudentRecord.remove(copy, pos);
155             }
156         }
157         // Return the res array
158         return res;
159     }
160 
161     /**
162      * STATIC METHOD TO ADD A STUDENT TO THE CLASS ARRAY OF CREATED STUDENTS
163      * @param s - the Student to add to the static variable belonging to the class
164      */
165     public static void addStudent(Student s){
166         // Create new array of length one greater than the current array and make it a copy
     of the original array
167         Student.createdStudents = Arrays.copyOf(Student.createdStudents, Student.
    createdStudents.length + 1);
168         // Add the new element to the last index of the new array (which should be empty)
169         Student.createdStudents[Student.createdStudents.length - 1] = s;
170     }
171 
172     /**
173      * METHOD TO ADD A RECORD TO THE METHODS ARRAY OF STUDENT RECORDS
174      * @param record- StudentRecord we are adding to the students array of records
175      */
176     public void addRecord(StudentRecord record){
177         // Create new array of length one greater than the current array and make it a copy
     of the original array
```

```
178         this.records = Arrays.copyOf(this.records, this.records.length + 1);
179         // Add the new element to the last index of the new array (which should be empty)
180         this.records[this.records.length - 1] = record;
181         this.calcGPA();
182     }
183 
184     /**
185      * METHOD TO PRINT THE TRANSCRIPT OF A GIVEN STUDENT AS REQUIRED
186      * @return the String of the generated transcript of the correct layout
187      */
188     public String printTranscript(){
189         String res = "University of Knowledge - Official Transcript";
190         res += "\n\n\n";
191         res += ("ID: "+this.id + "\n");
192         res += ("Name: "+this.name+ "\n");
193         res += ("GPA: "+this.gpa+ "\n\n");
194         //iterate through this.StudentRecords and generate ordered string
195         StudentRecord[] orderedRecords = this.getOrderedRecords();
196         for (int i=0;i<orderedRecords.length;i++){
197             res += orderedRecords[i];
198             res += "\n";
199             //if the next record has a different year or term, then add another \n
200             if(i<orderedRecords.length-1){
201                 if (orderedRecords[i].getModule().getYear() != orderedRecords[i+1].
    getModule().getYear()){
202                     res += "\n";
203                 }else if (orderedRecords[i].getModule().getTerm() != orderedRecords[i+1].
    getModule().getTerm()){
204                     res += "\n";
205                 }
206             }
```

```
207             }
208             return res;
209         }
210 }
211
```

File - C:\Users\nishi\Desktop\NEW JAVA\ECM1400 CA1 OOP\src\com\company\University.java

```java
package com.company;

import java.util.Arrays;

public class University {

    public ModuleDescriptor[] moduleDescriptors = new ModuleDescriptor[]{};
    public Student[] students = new Student[]{};
    public Module[] modules = new Module[]{};

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

    /**
     * @return The student with the highest GPA.
     */
    public Student getBestStudent() {
        Student best;
        if (this.students.length > 1){
            // FIND BEST STUDENT (IF THERE ARE >1 BEST STUDENTS, THE BEST ONES ARE STORED IN AN ARRAY AT THE END
            // OF THE FUNCTION HOWEVER NOT RETURNED AS THE RETURN TYPE FOR THE FUNCTION WAS STUDENT, NOT ARRAY, AND
            // I DIDNT WANT TO CHANGE THIS AS IT WAS ALREADY GIVEN)

            // Create empty array for the result
            Student[] multipleBestStudents = new Student[]{this.students[0]};
```

```
30              // assume the best student is the first one
31              best = this.students[0];
32
33              // Iterate through the rest of the students in the array of Students
34              for(Student s:this.students){
35                  if (s.getGPA() > best.getGPA()){
36                      // If the current student in the array has a better GPA:
37
38                      // make best, the current student
39                      best = s;
40                      // reset the best students array to hold the current students
41                      multipleBestStudents = new Student[]{s};
42                  }
43                  if (s.getGPA() == best.getGPA()){
44                      // Otherwise, if the current students in the array has an equal GPA
45                      // to the current best students:
46
47                      // Add this current student to the best students array
48                      // Create new array of length one greater than the current array and
   make it a copy of the original array
49                      multipleBestStudents = Arrays.copyOf(multipleBestStudents,
   multipleBestStudents.length + 1);
50                      // Add the new element to the last index of the new array (which should
   be empty)
51                      multipleBestStudents[multipleBestStudents.length - 1] = s;
52                  }
53              }
54              // return the value in best (the best student)
55              return best;
56          }else if(this.students.length == 1){
57              // If there is only one student, its the best by default
```

```
58              return this.students[0];
59          }else{
60              // If there are no students, there can be no best student
61              return null;
62          }
63
64      }
65
66      /**
67       * @return The module with the highest average score.
68       */
69      public Module getBestModule() {
70          Module best;
71          if (this.modules.length > 1){
72              // FIND BEST MODULE (IF THERE ARE >1 BEST MODULES, THE BEST ONES ARE STORED IN
    AN ARRAY AT THE END
73              // OF THE FUNCTION HOWEVER NOT RETURNED AS THE RETURN TYPE FOR THE FUNCTION WAS
    MODULE, NOT ARRAY, AND
74              // I DIDNT WANT TO CHANGE THIS AS IT WAS ALREADY GIVEN)
75
76              // Create empty array for the result
77              Module[] multipleBestModules = new Module[]{this.modules[0]};
78              // assume the best module is the first one
79              best = this.modules[0];
80
81              // Iterate through the rest of the modules in the array of Modules
82              for(Module m:this.modules){
83                  if (m.getFinalAverageGrade() > best.getFinalAverageGrade()){
84                      // If the current module in the array has a better final average grade:
85
86                      // make best, the current module
```

```
 87                      best = m;
 88                      // reset the best modules array to hold the current module
 89                      multipleBestModules = new Module[]{m};
 90                  } else if (m.getFinalAverageGrade() == best.getFinalAverageGrade()){
 91                      // Otherwise, if the current module in the array has an equal final
    average grade
 92                      // to the current best module:
 93
 94                      // Add this current Module to the best modules array
 95                      // Create new array of length one greater than the current array and
    make it a copy of the original array
 96                      multipleBestModules = Arrays.copyOf(multipleBestModules,
    multipleBestModules.length + 1);
 97                      // Add the new element to the last index of the new array (which should
     be empty)
 98                      multipleBestModules[multipleBestModules.length - 1] = m;
 99                  }
100              }
101
102              // return the value in best (the best module)
103              return best;
104          }else if(this.modules.length == 1){
105              // If there is only one module, its the best by default
106              return this.modules[0];
107          }else{
108              // If there are no modules, there can be no best module
109              return null;
110          }
111      }
112
113      /**
```

```
114      * Main Class for the University
115      */
116     public static void main(String[] args) {
117         //Create the new University under the variable university
118         University university = new University();
119 
120         //CREATE ALL NEW STUDENTS AND ADD THEM TO THE ARRAY
121         university.addStudent(new Student(1000, "Ana", 'F'));
122         university.addStudent(new Student(1001, "Oliver", 'M'));
123         university.addStudent(new Student(1002, "Mary", 'F'));
124         university.addStudent(new Student(1003, "John", 'M'));
125         university.addStudent(new Student(1004, "Noah", 'M'));
126         university.addStudent(new Student(1005, "Chico", 'M'));
127         university.addStudent(new Student(1006, "Maria", 'F'));
128         university.addStudent(new Student(1007, "Mark", 'X'));
129         university.addStudent(new Student(1008, "Lia", 'F'));
130         university.addStudent(new Student(1009, "Rachel", 'F'));
131 
132         //CREATE ALL MODULE DESCRIPTORS AND ADD THEM TO THE ARRAY
133         university.addModuleDescriptor(new ModuleDescriptor("ECM0002", "Real World
    Mathematics", new double[]{0.1, 0.3, 0.6}));
134         university.addModuleDescriptor(new ModuleDescriptor("ECM1400", "Programming", new
    double[]{0.25, 0.25, 0.25, 0.25}));
135         university.addModuleDescriptor(new ModuleDescriptor("ECM1406", "Data Structures",
    new double[]{0.25, 0.25, 0.5}));
136         university.addModuleDescriptor(new ModuleDescriptor("ECM1410", "Object-Orientated
    Programming", new double[]{0.2, 0.3, 0.5}));
137         university.addModuleDescriptor(new ModuleDescriptor("BEM2027", "Information Systems
    ", new double[]{0.1, 0.3, 0.3, 0.3}));
138         university.addModuleDescriptor(new ModuleDescriptor("PHY2023", "Thermal Physics",
    new double[]{0.4, 0.6}));
```

```
139
140         //CREATE ALL MODULES
141         university.addModule(new Module(university.moduleDescriptors[1], 2019, (byte) 1));
142         university.addModule(new Module(university.moduleDescriptors[5], 2019,(byte) 1));
143         university.addModule(new Module(university.moduleDescriptors[4], 2019,(byte) 2));
144         university.addModule(new Module(university.moduleDescriptors[1], 2019,(byte) 2));
145         university.addModule(new Module(university.moduleDescriptors[2], 2020,(byte) 1));
146         university.addModule(new Module(university.moduleDescriptors[3], 2020,(byte) 1));
147         university.addModule(new Module(university.moduleDescriptors[0], 2020,(byte) 2));
148
149         //ENROLL ALL STUDENTS TO THE CORRECT MODULE and pass in array of marks
150         university.students[0].enrollTo(university.modules[0], new double[]{9,10,10,10});
151         university.students[1].enrollTo(university.modules[0], new double[]{8,8,8,9});
152         university.students[2].enrollTo(university.modules[0], new double[]{5,5,6,5});
153         university.students[3].enrollTo(university.modules[0], new double[]{6,4,7,9});
154         university.students[4].enrollTo(university.modules[0], new double[]{10,9,10,9});
155         university.students[5].enrollTo(university.modules[1], new double[]{9,9});
156         university.students[6].enrollTo(university.modules[1], new double[]{6,9});
157         university.students[7].enrollTo(university.modules[1], new double[]{5,6});
158         university.students[8].enrollTo(university.modules[1], new double[]{9,7});
159         university.students[9].enrollTo(university.modules[1], new double[]{8,5});
160         university.students[0].enrollTo(university.modules[2], new double[]{10,10,9.5,10});
161         university.students[1].enrollTo(university.modules[2], new double[]{7,8.5, 8.2, 8
    });
162         university.students[2].enrollTo(university.modules[2], new double[]{6.5, 7, 5.5, 8.
    5});
163         university.students[3].enrollTo(university.modules[2], new double[]{5.5, 5, 6.5, 7
    });
164         university.students[4].enrollTo(university.modules[2], new double[]{7,5,8,6});
165         university.students[5].enrollTo(university.modules[3], new double[]{9,10,10,10});
166         university.students[6].enrollTo(university.modules[3], new double[]{8,8,8,9});
```

```
167         university.students[7].enrollTo(university.modules[3], new double[]{5,5,6,5});
168         university.students[8].enrollTo(university.modules[3], new double[]{6,4,7,9});
169         university.students[9].enrollTo(university.modules[3], new double[]{10,9,8,9});
170         university.students[0].enrollTo(university.modules[4], new double[]{10,10,10});
171         university.students[1].enrollTo(university.modules[4], new double[]{8,7.5,7.5});
172         university.students[2].enrollTo(university.modules[4], new double[]{9,7,7});
173         university.students[3].enrollTo(university.modules[4], new double[]{9,8,7});
174         university.students[4].enrollTo(university.modules[4], new double[]{2,7,7});
175         university.students[5].enrollTo(university.modules[4], new double[]{10,10,10});
176         university.students[6].enrollTo(university.modules[4], new double[]{8,7.5,7.5});
177         university.students[7].enrollTo(university.modules[4], new double[]{10,10,10});
178         university.students[8].enrollTo(university.modules[4], new double[]{9,8,7});
179         university.students[9].enrollTo(university.modules[4], new double[]{8,9,10});
180         university.students[0].enrollTo(university.modules[5], new double[]{10,9,10});
181         university.students[1].enrollTo(university.modules[5], new double[]{8.5, 9, 7.5});
182         university.students[2].enrollTo(university.modules[5], new double[]{10,10,5.5});
183         university.students[3].enrollTo(university.modules[5], new double[]{7,7,7});
184         university.students[4].enrollTo(university.modules[5], new double[]{5,6,10});
185         university.students[5].enrollTo(university.modules[6], new double[]{8,9,8});
186         university.students[6].enrollTo(university.modules[6], new double[]{6.5, 9, 9.5});
187         university.students[7].enrollTo(university.modules[6], new double[]{8.5, 10, 8.5});
188         university.students[8].enrollTo(university.modules[6], new double[]{7.5, 8,10});
189         university.students[9].enrollTo(university.modules[6], new double[]{10,6,10});
190
191         //TESTING OUTPUTS AS REQUIRED
192         // Print Reports
193         System.out.println("The UoK has " + university.getTotalNumberStudents() + "
    students.");
194
195         // best module
196         System.out.println("The best module is:");
```

```
197         System.out.println(university.getBestModule());
198
199         // best student
200         System.out.println("The best student is:");
201         System.out.println(university.getBestStudent().printTranscript());
202     }
203
204     /**
205      * Add Module to the universities Module Array
206      * @param m - m is the Module we are adding to the Module[] modules
207      */
208     public void addModule(Module m){
209         // Create new array of length one greater than the current array and make it a copy
    of the original array
210         this.modules = Arrays.copyOf(this.modules, this.modules.length + 1);
211         // Add the new element to the last index of the new array (which should be empty)
212         this.modules[this.modules.length - 1] = m;
213     }
214
215     /**
216      * Add Student to the universities Student Array
217      * @param s - s is the Student we are adding to the Student[] students
218      */
219     public void addStudent(Student s){
220         // Create new array of length one greater than the current array and make it a copy
    of the original array
221         this.students = Arrays.copyOf(this.students, this.students.length + 1);
222         // Add the new element to the last index of the new array (which should be empty)
223         this.students[this.students.length - 1] = s;
224     }
225
```

```
226     /**
227      * Add Module Descriptor to the universities Module Descriptor Array
228      * @param d - d is the ModuleDescriptor we are adding to the ModuleDescriptor[]
    moduleDescriptors
229      */
230     public void addModuleDescriptor(ModuleDescriptor d){
231         // Create new array of length one greater than the current array and make it a copy
     of the original array
232         this.moduleDescriptors = Arrays.copyOf(this.moduleDescriptors, this.
    moduleDescriptors.length + 1);
233         // Add the new element to the last index of the new array (which should be empty)
234         this.moduleDescriptors[this.moduleDescriptors.length - 1] = d;
235     }
236 }
237
```

File - C:\Users\nishi\Desktop\NEW JAVA\ECM1400 CA1 OOP\src\com\company\StudentRecord.java

```java
package com.company;

import java.util.Arrays;

public class StudentRecord {

    private Student student;
    private Module module;
    private double[] marks;
    private double finalScore;
    private Boolean isAboveAverage;

    /**
     * Constructor for StudentRecord, Used to create a new StudentRecord when called
     * @param student - Student we are creating the record for
     * @param module - Module we are creating the record for
     */
    public StudentRecord(Student student, Module module){
        //Make sure this same student doesnt have another record for this module
        for(StudentRecord rec:module.getRecords()){
            if(student == rec.student){
                throw new IllegalArgumentException();
            }
        }

        // Save the Required data
        this.student = student;
        this.module = module;
    }

    /**
```

```
32      * Allows the user to add marks to the record
33      * @param marks - Array of marks for the student
34      * Adds Marks to the studentRecord
35      */
36     public void AddMarks(double[] marks){
37         // Assume values are valid
38         boolean notValid = false;
39         // Get the weights from the module this record is for
40         double[] weights = module.getModuleDescriptor().getContinuousAssignmentWeights();
41         double total = 0.00;
42 
43         if(!(weights.length == marks.length)){
44             //unable to calculate final score due to non 1-1 relation between weightings and
    marks
45             System.out.println("WEIGHTS NOT EQUAL TO LENGTH");
46             notValid = true;
47         }else{
48             //calculate total AND check conditions
49             for(int i=0;i<=weights.length-1;i++){
50                 total += weights[i] * marks[i];
51                 // Condition to make sure marks are not <0 or >100
52                 if(marks[i] < 0 || marks[i] > 100){
53                     System.out.println("MARKS LESS THAN 0 OR GREATER THAN 100");
54                     notValid = true;
55                 }
56             }
57         }
58 
59         // make temporary variable finalScore equal to toal
60         double finalScore = total;
61         // Check the validity of finalScore
```

```
62          if (notValid == true || finalScore<0 || finalScore>100){
63              throw new IllegalArgumentException();
64          }else{
65              // Save Data
66              this.finalScore = finalScore;
67              this.marks = marks;
68          }
69          //UPDATE STUDENTS GPA
70          module.resetAverage();
71      }
72
73      /**
74       * Checks if the Students final score is above or below the module average
75       */
76      public void AboveAvgCheck(){
77          if(this.finalScore <= module.getFinalAverageGrade()){
78              this.isAboveAverage = false;
79          }else{
80              this.isAboveAverage = true;
81          }
82      }
83
84      /**
85       * GETTER METHOD FOR THE MODULE THAT THE RECORD IS FOR
86       * @return the Module belonging to the record
87       */
88      public Module getModule() {
89          return module;
90      }
91
92      /**
```

```java
     * GETTER METHOD FOR THE FINAL SCORE
     * @return The final score of this record
     */
    public double GetFinalScore() {
        return this.finalScore;
    }

    /**
     * STATIC METHOD TO ADD A RECORD TO AN EXISTING STUDENTRECORD ARRAY
     * @param array - Array of records we are adding to
     * @param record - Record we are adding to the array
     * @return the new array of records with the added record
     */
    public static StudentRecord[] append(StudentRecord[] array, StudentRecord record){
        // Create new array of length one greater than the current array and make it a copy
 of the original array
        StudentRecord[] res = Arrays.copyOf(array, array.length + 1);
        // Add the new element to the last index of the new array (which should be empty)
        res[res.length - 1] = record;
        return res;
    }

    /**
     * STATIC METHOD TO REMOVE A RECORD TO AN EXISTING STUDENTRECORD ARRAY
     * @param array - Array of records we are removing from
     * @param pos - Position of the array we want to remove
     * @return - New Array with the removed value
     */
    public static StudentRecord[] remove(StudentRecord[] array, int pos){
        // Copy all items to a new array of length one less than the original array,
 skipping
```

```
122          // the position which is specified
123          StudentRecord[] res = new StudentRecord[]{};
124          for(int i=0;i<array.length;i++){
125              if(i!=pos){
126                  res = StudentRecord.append(res, array[i]);
127              }
128          }
129          return res;
130      }
131
132      @Override
133      public String toString() {
134          Module m = this.module;
135          String res = "| ";
136          res += m.getYear() + " | ";
137          res += m.getTerm() + " | ";
138          res += m.getCode() + " | ";
139          res += this.finalScore + " |";
140          return res;
141      }
142  }
143
```

File - C:\Users\nishi\Desktop\NEW JAVA\ECM1400 CA1 OOP\src\com\company\ModuleDescriptor.java

```java
package com.company;

import java.util.Arrays;

public class ModuleDescriptor {

    private String code;
    private String name;
    private double[] continuousAssignmentWeights;
    private static ModuleDescriptor[] createdDescriptors = new ModuleDescriptor[]{};

    /**
     * Constructor for the ModuleDescriptor we are creating
     * @param code - Module Code for the module we are creating
     * @param name - Name for the module we are creating
     * @param continuousAssignmentWeights - Assignment Weights for the module we are creating
     */
    public ModuleDescriptor(String code, String name, double[] continuousAssignmentWeights){
        ModuleDescriptor[] moduleDescriptors = ModuleDescriptor.createdDescriptors;
        // Assume the parameters are valid
        boolean not_valid = false;
        // Constructor for ModuleDescriptor

        // ensure neither code nor name are null
        if(code==null | name==null){not_valid = true;}

        //check code is not used in another module
        for(ModuleDescriptor item:moduleDescriptors){
            if(item.code == code){not_valid = true;}
        }
```

```
31
32          // ensure that the total weightings dont exceed 1.00
33          double total_weight = 0;
34          for(double val:continuousAssignmentWeights){
35              total_weight += val;
36              if (val < 0) {not_valid = true;}
37          }
38          if (total_weight > 1.00){not_valid = true;}
39
40          // If the module is not valid, then throw and exception
41          if (not_valid){
42              throw new IllegalArgumentException();
43          }else{
44              // Save the required data
45              this.code = code;
46              this.name = name;
47              this.continuousAssignmentWeights = continuousAssignmentWeights;
48              ModuleDescriptor.addModuleDescriptor(this);
49          }
50      }
51
52      /**
53       * GETTER METHOD FOR THE MODULE CODE
54       * @return the code for this Module Descriptor
55       */
56      public String getCode(){
57          return this.code;
58      }
59
60      /**
61       * GETTER METHOD FOR THE ARRAY OF ASSIGNMENT WEIGHTS
```

```
62       * @return the array of weights for this module descriptor
63       */
64      public double[] getContinuousAssignmentWeights(){
65          return continuousAssignmentWeights;
66      }
67
68      /**
69       * Adds the new descriptor we are creating to a static variable
70       * to allow us to look for identical copies
71       * @param d - The descriptor we are creating
72       */
73      public static void addModuleDescriptor(ModuleDescriptor d){
74          ModuleDescriptor.createdDescriptors = Arrays.copyOf(ModuleDescriptor.
    createdDescriptors, ModuleDescriptor.createdDescriptors.length + 1);
75          ModuleDescriptor.createdDescriptors[ModuleDescriptor.createdDescriptors.length - 1
    ] = d;
76      }
77
78
79
80  }
81
```