1 Part 1

1.1 Specifications

The overall task is to write, and test, a scaled-down student database for a university, containing the following information:

- Every student has:
  - a `student number` (a seven digit integer)
  - a `family name`
  - `given name(s)`
  - a `degree` (one of `"Science"`, `"Arts"` or `"Medicine"`)
  - `results` for each topic
- Each topic result contains:
  - a `topic code` (8 characters, e.g. `COMP2008`)
  - the grade (`"FL"`,`"PS"`,`"CR"`,`"DN"` or `"HD"`)
  - a `mark` (0 - 100), *which is optional*
- Medicine students also have a list of (zero or more) `prizes`
- Arts students have a `major` and a `minor`

You are required to read in the student information from a file in the following format:

- Each line of input is either a student details line, a topic result line, or a comment
- Each data field in each line is separated by commas (like an Excel .csv file)
- The fields in a student details line are in the following order:
  - a character `'A'`, `'M'` or `'S'` representing the degree (arts, medicine or science respectively)
  - the `student number`
  - the `family name`
  - the `given names` (separated by spaces)
  - a (possibly empty) list of `prizes`, separated by commas (only if the degree is medicine)
  - the `major` (only if the degree is arts)
  - the `minor` (only if the degree is arts)
- Topic results appear in the following order (separated by commas):
  - the letter `'R'` (upper case)
  - the student number
  - the topic code
  - the grade
  - the mark (if it exists)
- Prizes for medical students are as follows:
  - an input line beginning with a `'P'` indicates a new prize

- ➢ the second field is the name of the prize
- ➢ the third field is the topics that the prize is awarded for
    - e.g. `"MMED2"` means the prize is awarded for performance in all topics with a code beginning with `MMED2`
- ➢ the fourth field, *min*, is the minimum number of topics attempted which match the third field
- ➢ the prize is awarded to the medical student who has the highest average mark in their best *min* matching topics
    - e.g. for the input line `"P,Neuroscience 1 Prize,MMED1904,1"`, the prize goes to the student with the highest mark in `MMED1904`
    - e.g. for the input line `"P,3rd Year Medicine Prize,MMED3,5"`, the prize goes to the student with the highest average mark in their best 5 `MMED3xxx` topics
    - a prize description may appear anywhere within the input file
    - prizes should be calculated and added to the students' prize lists in the order the prize descriptions appear, and after any prizes in the student details line
    - there will be **no more than 10 prizes**

You are required to print, and write to file, an academic record for each student, with the following requirements:

- ■ students appear in the order that the student details lines appear in the input file
- ■ topic results for each student appear in the same order as they appear in the input file
- ■ each academic record contains the following lines of output, in order:
    - ➢ the string `"Academic record for"` followed by the given name(s), the family name and the student number in parentheses (each separated by spaces)
    - ➢ the string `"Degree:  "` followed by the degree
    - ➢ the string `"Major:  "` followed by the major (if applicable)
    - ➢ the string `"Minor:  "` followed by the minor (if applicable)
    - ➢ for each prize, the string `"Prize:  "` followed by the prize name
    - ➢ for each topic result, the topic code, grade and mark (if given), separated by spaces
    - ➢ a blank line, to indicate the end of that student's record

You may make the following assumptions:

- ■ each student record occurs before any of that student's results
- ■ no student attempts more than 24 topics
- ■ no student is awarded more than 10 prizes
- ■ there are no more than 1000 students
- ■ any line not beginning with `‘A’`, `‘M’`, `‘P’`, `‘R’` or `‘S’` is a comment, and should be ignored

A sample input file and corresponding output are shown below.

```
// Sample input file

// Student details
S,9800123,Smith,John Paul
M,9821012,Jones,Mary,Chemistry Prize 1998
A,9987654,Howard,John,Politics,Economics

// Topic results
R,9821012,BIOL1000,HD,89
R,9821012,CHEM1001,HD,92
R,9800123,COMP1000,PS,55
R,9821012,COMP1000,DN,75
R,9800123,COMP1001,DN,77
R,9800123,HIST1234,HD
R,9821012,PHYS1010,HD,93
R,9800123,PSYC0123,FL,42
```

Figure 1: A Sample Input File

```
Academic record for John Paul Smith (9800123)
Degree: Science
COMP1000 PS 55
COMP1001 DN 77
HIST1234 HD
PSYC0123 FL 42

Academic record for Mary Jones (9821012)
Degree: Medicine
Prize: Chemistry Prize 1998
BIOL1000 HD 89
CHEM1001 HD 92
COMP1000 DN 75
PHYS1010 HD 93

Academic record for John Howard (9987654)
Degree: Arts
Major: Politics
Minor: Economics
```

Figure 2: Output for Sample Input

1.2 Deliverables

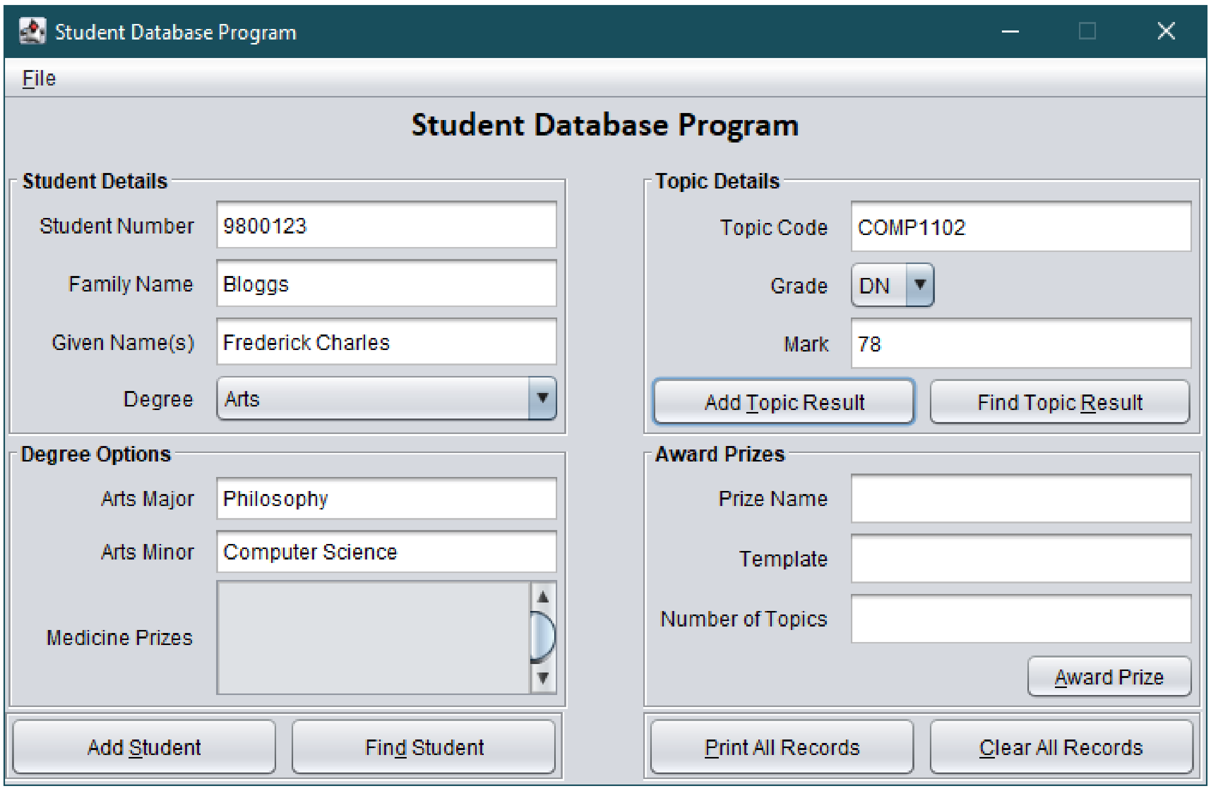
- Design and document a test strategy against these specifications.
- Execute the test strategy using the application you have built. Note that this will involve prioritising which tests are carried out and how much time is spent on the tests.
- Report on the outcome of the testing.

You will be expected to design your solution with extension and future additions in mind. To that end an important part of the assignment will be the model on which you build and test. You are free to develop the solution as you see fit, however it must conform to the specifications. As well as the JUnit tests that will be written, you will also provide a report on the coverage of your tests. By running the tests with coverage you will be able to see where parts of your code have not been tested and write tests accordingly.

2 Part 2

2.1 Specifications

For this part of the assignment, you will write a graphical interface to the student database developed in Part 1, see section 1, of the assignment. The interface may look something similar to the following (**not exactly the same, be creative**), however you are free to design this anyway you wish:



Student Database Program

Student Details	
Student Number	9800123
Family Name	Bloggs
Given Name(s)	Frederick Charles
Degree	Arts
Degree Options	
Arts Major	Philosophy
Arts Minor	Computer Science
Medicine Prizes	
Topic Details	
Topic Code	COMP1102
Grade	DN
Mark	78
Award Prizes	
Prize Name	
Template	
Number of Topics	

Figure 3: Sample Database GUI

The functionality is as follows (this functionality must be included):

Add Student:
adds a new student to the database with the details as shown in the left hand side of the image above. No topic results are stored. If the student already exists in the database then an appropriate message should be displayed, the fields will be cleared, and no details will be stored.

If the student is successfully added, then the button label should change to **Enter New Student**. Now if the button is pressed all student details are cleared ready for a new student to be added and the button label returns to **Add Student**.

Find Student:
looks up the student database for a student with an ID number matching the one in the student ID box, and updates the rest of the student details fields to the values previously entered for that student. Topic details are not updated. If no student with this ID exists in the database, then an appropriate message is displayed.

Add Topic Result:
stores the topic details for the student matching the student ID number on the top left of Figure 3. The student matching the Student ID should have been added to the database previously. If all mandatory fields are not present (Topic Code and Grade), then an appropriate message should be displayed and no results should be added.

Find Topic Result:
looks up the student database for a result matching the student ID and topic code fields, and updates the rest of the topic details fields accordingly. If no matching topic is found, the mark and grade fields are cleared.

Award Prize:
calls `awardPrize(...)` in the `StudentDatabase` classs using the details entered in the Prize Name, Template and Number of Topics fields. The outcome should be the same as that from Part 1

Print All Records:
calls `printRecords()` in the `StudentDatabase` class.

Clear All Records:
calls `clearRecords()` in the `StudentDatabase` class.

You will need a working version of Part 1 of the assignment (your own version preferably), see section 1 for specifications. If you do not have a fully functional version then please contact me to arrange a copy that can be used.

2.1.1 Stage 1

- First you should complete the layout of the components on the GUI. Feel free to vary the layout, colours, and sizes of components if you wish.
- Once this is completed you should then implement the "listener" classes which is the event handling code. This code will store or retrieve information from the student database when one of the seven buttons is pressed.
- You may or may not include uploading of data via file (although the capacity to do so will still exist from Assignment 1).

2.1.2 Stage 2

- update your test strategy to incorporate integration testing of the GUI, and also "black-box testing", against the specifications detailed above, for the system when it is complete. This may also involve some regression testing to ensure that the original code still performs as it did.
- Execute the updated test strategy
- Report on the outcome of the testing