Project 2

# 1. Database of Students

In this project, you will follow the program specifications given below, to write a Java program that represents the database of student records. Your program should be able to accept commands from the user, extract necessary information from the database and display it to the user and update the database with the new information. Single student record contains information about one particular student and his/her courses. Your program should make it possible to import the student records from the .txt file to build the database.

# 2. Learning Goals

This project has the following learning goals.

1. Write a program that conforms to a specified user interface.
2. Gain experience with File I/O.
3. Practice employing good software design principles.
4. Implement basic exception handling.
5. Implement an entire program from scratch (no skeleton files!).
6. Learn to write code that follows the commenting and the style guidelines.

# 3. Getting Started

There are no skeleton files for this project. The program design is up to you, although we do require that your **main() method be inside a file called Database.java**.

Before writing a single line of code, design your program on paper by identifying, describing and naming the various classes your program will use. Next, begin identifying data fields and methods for each class. Wait until you have a good idea of what classes you will create and what their responsibilities will be, before writing code.

50% of your grade is for the correctness of the code (i.e. whether your program works as expected). Other 50% of your grade will depend on how well you have designed your classes (whether classes are relevant to this problem, having constructors, private data fields correctly initialized in the constructors, public getters and setters, toString methods, equals method, and other relevant public / private methods).

We require that your program can be executed in the following way as explained below.

1. java Database <inputFile>  
   In this mode of execution, the user will be able to interact with your program.

Your program should allow the user to be able to interact with the database. Your main method in the class Database should accept one argument which will be the name of the .txt file with information about the student records. For example, if filename is **“input.txt”** then your program should accept this argument and access the file name inside main by accessing the command line argument args[0].

In this file each line represents a single student record:

Katy Adams,1995,CS,302,3,CS,367,3

Bill Stewart,1996,CS,354,3,CS,367,3,math,620,4

Each line starts with the name of the student which can contain white spaces, uppercase and lowercase letters. Then, there is a year of birth of the student which is a single integer value. After that there is an information about the courses of this student. Each course is described through course department, the number of the course and number of credits student can get by taking this course. All these values are separated by comma. **There are no whitespaces after or before the commas.**

By reading this input file your program should initialize your database by saving these student records to refer to them later in the program.

After initializing the database, it should prompt the user to choose from the menu of options that will be shown in the sample runs later. After interaction of the user with the database, user can choose whether or not to export (i.e. save) the updated database to a new output file. If user decides to save the updated student records to the file output.txt for example, then output.txt should have the same format as input.txt (with the updated student details). Program terminates when user decides to do so by choosing the exit option.

### Menu options available for the user:

DATABASE OF STUDENTS!

=====================

Choose from the menu of options:

1. Print the student details.

2. Search for a student.

3. Add a student record.

4. Delete a student record.

5. Update a student record.

6. Exit the database.

Enter your option:

### Handling different options:

**1. Print the student details.**

This option prints all the students in the current database with the information about name, ID, age and courses (course department, course number and number of credits for it).

**2. Search for a student.**

When this option is chosen, user is asked based on what field the search should be made: ID, name, age or a particular course. This option if chosen displays the result of search to the console. Searching based on ID should return only one student, because IDs are unique and are generated by your program automatically. If no such student exists with given ID, an error message is printed. For searching based on the name, your program should display all the students that contain this word in their name fields. For example, if we type “anna”, then it is possible that names like “Anna”, “anna”, “Anna Clarke” should be considered as a match. So, this search can return a list of students. For search based on a course, user should be asked to enter course department name, course number and number of credits. Courses like “cs 302 3” and “CS 302 3” are considered as the same course. However, these two courses are different: “cs 302 3” and “cs 302 4” because number of credits are not matching. Again, this search should return a list of students who are taking this class.

**3. Add a student record.**

This option adds a new student to the database. Initially, when you add a student to the empty database you should generate IDs automatically and they should be unique. However, if you delete some student from the database, you should keep track of which IDs you can reuse again. Imagine, you have 3 students with IDs 1, 2 and 3. If second student is deleted, then id 2 becomes vacant for use again. So, next time if you add a new student you should assign an ID=2, not ID=4. Only, if there are no IDs already available, you should generate a new one. Having duplicate students is fine. So, you can have students with the same name, same age and courses, however their IDs should be different.

**4. Delete a student record.**

Remove a student with a particular ID. This option removes a student given the ID as input, or prints an error message to the console that there is no student with the given ID. Again, keep track of removed IDs to use again.

**5. Update a student record.**

This option updates the student record with the specific ID. If there is no student with the given ID, print an error message. This option should allow to update particular field of the student record: name, year of birth or the list of courses. In case of updating the list of courses, there is a choice of either adding one or more course(s) or removing one or more course(s) from the list of courses. For removing the course from the list of courses, all fields like department name (**ignore case for this field**), course number and number of credits should match. For adding the course, you should check whether this course is already in the list of courses. If the course to be added is already in the list of courses that this student is taking, then the course shouldn’t not be added to the student’s list of courses. A student cannot take 2 courses with the same department name and course number (even though the credits may vary). For example, if a student already takes the course (CS 302 with 3 credits), the same student cannot take the same course with different credits (e.g. CS 302 with 4 credits).

**6. Exit the database.**

This option should exit the database. Program should terminate. Otherwise, if options 1-6 are chosen, the menu options should appear again to ask the user for the next command.

### User input validation

There is a minimal user input validation in this program. You can assume user always enters valid options for each step. For example, no strings are given as input when integer is asked to be entered. The few cases when you need to deal with user input validation are the following:

1. If we try to remove a student who doesn’t exist in the database (i.e. student ID is not in the list of the student IDs that are currently in the database), then your program should display an error message.
2. If we try to delete a course that a student is not taking, then your program should display an error message.
3. If we try to add the course that student is already taking, display an error message.

Also, there are some acknowledgement / error messages that should be printed after each step. You can see more details about the acknowledgement / error messages in the sample runs later. Of course, you should also deal with the errors when files with the names passed as command line arguments to the main method do not exist. However, for the output.txt if there is no such file, you should create one, or overwrite the existing one.

# 4. About Copying Code and Academic Misconduct

*Do* ***not*** *post your project solutions (or drafts) on any publicly accessible web sites. This specifically includes GitHub. It is academic misconduct to post your solution.*  
Just a note to remind everyone what is **OK** and what is **not OK** in terms of talking to other people in class about this and future projects.

It is **definitely OK** to:

* discuss the project in general terms (e.g. How do we check if the game is being played the first time so that we can ask the players to enter their names?)
* discuss about how to implement a particular logic (e.g. What should I do to make sure that the players can play the game multiple times.)
* help others debug their code and find problems
* ask the TA or instructor or both for as much help as you need!

It is **NOT OK** to:

* share your code with other students in the class (Oh you want your game to be played multiple times, well here is my code and it works so you can just use/copy that)
* ask other students / someone outside the class to write code for you
* bug someone else for a lot of help (particularly if they are already done!)

**Discovery of any inappropriate code sharing will lead to harsh penalties for all involved parties.** This draconian policy is in place to protect the bulk of you who have put in the hard work on the project.

The penalty for academic misconduct on this project will be a failing grade in the course. This penalty is significantly more harsh than if you simply do not do the project. You will gain much more by doing the project than by copying, possibly modifying, and turning in someone else's effort.