King Abdulaziz University
Faculty of Computing and Information Technology
Computer Science Department

CPCS204, 1st Term 2017 (Fall 2016)
Program 1: *FCIT Registrar*
Assigned: Thursday, September 22nd, 2016
Due: Thursday, October 6th, 2016

Purpose:

1. The first purpose of this program is for you to REVIEW your JAVA (202 & 203).
2. The second purpose of this program is for you to review file I/O (input/output).
3. Finally, you are to implement binary search & sort algorithms in the program.

Read Carefully:

- This program is worth 6% of your final grade.
- **WARNING**: This is an individual project; you must solve it by yourself. Any form of cheating will result in receiving **– 4%** (less than zero) in the program.
- The deadline for this project is by **11:59 PM on Thursday, October 6, 2016.**
  - **Note:** once the clock becomes 11:59PM, the submission will be closed! Therefore, in reality, you must submit by 11:58 and 59 seconds.
- LATE SUBMISSION: you are allowed to make a late submission, but there is a penalty. If you submit **within 24 hours** of the due date (so on Friday by 11:59PM), you will receive a 25% deduction. If you submit within 48 hours of the due date (so on Saturday by 11:59PM), you will receive a 50% deduction.
- NetBeans Project Name: `FCIT_Registrar`
- Package: please use the following package name: `fcit_registrar`
- Your package MUST be named the same in order to receive full credit.
- **Blackboard Submission**:
  - This project must be submitted online via blackboard
  - The source file(s) of your program should be zipped up. You must name the zip file using the following naming convention:
    SectionNumber_StudentID_ProgramNumber.zip
    Example: *EA_ 1110348_ P1.**zip***

Program 1: FCIT Registrar

Objective

Practice all concepts from CPCS-202 and CPCS-203, with a focus on making classes and creating objects from these classes. Specifically, you will practice making an array of objects and manipulating/using the data inside the array. Finally, and VERY important is that this program requires you to read from a file and write to a file (practice with File I/O). This is very important because all programs during the semester will use File I/O.

The Problem

Your job is to create a basic course registration system for FCIT CS Department. This system will allow students to register for courses and will keep track of all students, all courses, and all registrations. During the program, the following things can occur:

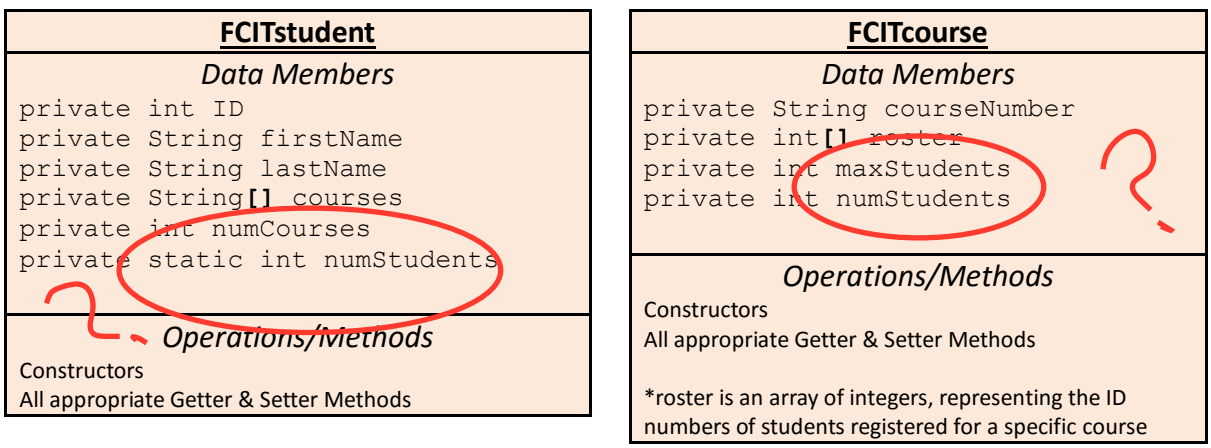
- New students can be enrolled in the system.
- New courses can be opened in the system.
- Courses can be expanded to hold more students.
- Courses can be deleted/removed from the system.
- Students can register for courses (ADD a course to their schedule). The maximum load one student can take is six courses in the semester.
- Students can remove courses (DROP) from their schedule.
- And more!

You will use File I/O to read input from a file and then print the output to a file. To be clear, you will read COMMANDS from an input file. Example commands are OPENCOURSE, EXPANDCOURSE, ENROLLSTUDENT, COURSEADD, COURSEDROP, etc. Then, depending on the command, you will either open a new course, expand the capacity of a course, enroll students into the system, add or drop courses from a student schedule, and more. But instead of printing to the console window (screen), you will print to an output file.

**Sample input and output files have been provided for you on Blackboard.*

Implementation

For this program, you will create **two** Classes (UML diagram shown below):



You will use the `FCITstudent` class to create objects of type `FCITstudent`, and you will use the `FCITcourse` class to create objects of type `FCITcourse`, which are used to represent the courses available in the FCIT Registration System.

The first two lines of the input file are as follows:
Line 1: the maximum number of courses that can be used in the registration system.
Line 2: the maximum number of students allowed to enroll in the registration system.

You must read these values into appropriate variables (such as maxCourses and maxStudents).

Next, you will use these new values to make two new arrays:

1. An array of `FCITstudent` object ***references***
2. An array of `FCITcourse` object ***references***

- *See pages 71-78 of Chapter 8 slides (from 203) for help on array of object references*

To help you, here is the code to do this:

```
FCITcourse[] courses = new FCITcourse[maxCourses];
FCITstudent[] students = new FCITstudent[maxStudents];
```

What do these lines do? They create an array of **references**. Note: each reference has a default value of **`null`**. Until now, we have NOT created any objects. For example, `FCITstudent` objects are only created when we see the command `ADDSTUDENT`. At that time, a new `FCITstudent` object will be created and the reference for that object will be saved in the appropriate location of the array. Note, while we do give you the maximum number of possible courses allowed in the system, the largest course number allowed is 499 (such as CPCS-499). Therefore, there is a much easier way of modeling the array of courses…

Input File Specifications

You will read in input from a file, "`FCIT_Registrar.in`". Have this AUTOMATED. Do not ask the user to enter "**`FCIT_Registrar.in`**". You should read in this automatically.

> ***Note:*** a **`*.in`** file is just a regular text file. Also, a `*.out` file is a regular text file. Do not be scared by this. Instead of calling the input and output file as input.txt and output.txt, it is better for those files to have a good name. Therefore, the input file will be called `FCIT_Registrar.in` and the output file will be called `FCIT_Registrar.out`.

The first line of the file will be an integer, *d*, representing the maximum number of courses allowed in the FCIT Registrar. The second line of the file will be an integer representing the maximum possible number of students allowed to enroll in the FCIT Registrar. The third line of the input file will be an integer, *n*, representing the number of commands in the input file. Each of the following *n* lines will have a command, which is then followed by relevant data as described below (and this relevant data will be on the same line as the command).

The commands you will have to implement are as follows:

- **OPENCOURSE** – Adds (opens) a new course to the FCIT Registrar. The command will be followed by the following information all on the same line: a String representing the course number (such as CPCS-202), and an integer representing the initial maximum capacity of the course. Note that before opening a new course, you should make sure that the course is not already open. If it is already open, you should print an error message (see output examples). Also, only CPCS courses are used to make it easy.

- **EXPANDCOURSE** – Expands the maximum capacity of an already open course. The command will be followed by the following information all on the same line: a String representing the course number (such as CPCS-202), and an integer representing the new maximum capacity of the course. Note that before changing the capacity of a course, you should first make sure that the course is already open. Additionally, you should confirm that the new capacity is, in fact, bigger than the previous capacity. If the course is not open or if the new capacity is not larger than the previous capacity, then an error message should be printed (see output examples).

- **DELETECOURSE** – Deletes a course from the FCIT Registrar. For example, maybe an instructor is not available to teach the course; therefore, the course will be deleted, even though some students may already be registered for the course. The command will be followed by the following information all on the same line: a String representing the course number (such as CPCS-202). Note that before deleting a course, you must confirm that the course exists and is open. If the course is not open, then an error message should be printed (see output examples). Additionally, be careful when deleting a course. Why? Some students may have already registered for the course. Therefore, before deleting the course, you must remove the course from the schedule of all students who are currently registered for the course. Once this is done, you can then delete the course.

- **ENROLLSTUDENT** – Adds a new student to the FCIT Registrar. The command will be followed by the following information all on the same line: ID, an integer representing the student ID number; firstName, a string representing the student first name; and lastName, another string for student last name.

  When you read the ENROLLSTUDENT command, you must make a new object of type `FCITstudent`. Next, you must scan the additional information (listed above) and save this information into the correct data members of the new object. Finally, you must save the reference of this object inside the appropriate index of the student array.

  ***Note**: this array **must** stay in sorted order based on the ID of the student. So the student with the smallest ID value will be at index 0, the student with the next smallest at index 1, and so on. Therefore, when you save the object reference into the array, you must first find the correct sorted index. After you find the correct insertion location, if there is no object at that location, then you can easily save the object reference at this index. BUT, IF there is an object at the index that you find, you must **SHIFT** that object, **AND** all objects to the right of it, one space to the right. This is how you make a new space in the array for the new object.

what is best ~~way for~~ shifting!

Example: if your array already has 6 student object references, from index 0 to index 5. And now you want to add a new student object reference at index 4 (based on the sorted location). Before you can add the new student object reference at index 4, you must SHIFT the student object references at index 4 and index 5 to index 5 and index 6. Basically you need to move them one position over to the right. Now, you have an empty spot at index 4 for the new student object reference.

- **COURSEADD** - Adds a new course to the student's schedule. The command will be followed by the following information all on the same line: ID, an integer representing the student's ID number and a String representing the course number of the course the student is adding. As with previous commands, there are several things to check here. Is the student already enrolled in the system? Meaning, is he/she even allowed to register for courses? Is the course open? Has the student already registered for the maximum of six courses? Meaning, is the student even allowed to register for more courses? And does the course still have room for additional student registrations or has it already reached maximum capacity? All of these situations should be checked. If one is present, an error message should be printed. See output for examples. If none of the situations are present, the course should be added to the student's schedule, and the student should be added to the official course roster of that course. *Note that the String array inside the object of each student (representing the registered courses) is not sorted. You simply add courses (Strings) to the next available spot in the array. *Also, note that the int array, called roster, inside the object of each course (representing a listing of students registered for that particular course) is also not sorted. These int ID number are simply added to the next available spot in the int array.

- **COURSEDROP** – Drops a course from the student's schedule. The command will be followed by the following information all on the same line: ID, an integer representing the student's ID number and a String representing the course number of the course the student is dropping. As with previous commands, there are several things to check here. Is the student already enrolled in the system? Meaning, is he/she even allowed to drop courses? Is the course open? Perhaps the course is not even open and the command is a mistake. Has the student actually registered for the course they are requesting to drop? Perhaps, the request is a mistake and the course ID number is wrong. Meaning, perhaps the student is not even registered in the course to begin with! All of these situations should be checked. If one is present, an error message should be printed. See output for examples. If none of the situations are present, the course should be dropped from the student's schedule, and the student should be removed from the official course roster of that course. *Note, be careful when removing the course from the student schedule and when removing the student from the course roster. For example, when you remove a course from the student schedule, there will be a "hole" in the array. Therefore, in order to "fill the hole", you will need to shift all remaining elements to the left. The same thing is true for the student roster. For example, if you are removing student ID 1123456 from the roster, and if the student is found at index 6, this means that you must SHIFT indices 6 and larger one space to the left in order to fill the hole.

- **DELTESTUDENT** – Removes/deletes a student from the FCIT Registrar system. The command will be followed by the following information all on the same line: an int representing the ID of the student to be deleted. Note that the student must be enrolled in the system in order to be deleted. If the student is not in the system, print an error message (see output). Also, before deleting a student, the student must be removed from

the course roster of each course he/she is already registered in. Once the student is removed from the course roster of each course they are registered in, you can then delete the student (see output for clarification).

- **PRINTDETAILSCOURSE** – Prints details of a specific course. The command will be followed by the following information all on the same line: a string representing the course number whose details should be printed. If the course is not even open, an error should print (see output). Otherwise, print the course details (see output).

- **PRINTDETAILSSTUDENT** – Prints details of a specific student. The command will be followed by the following information all on the same line: an int representing the ID number of a student whose details should be printed. If the student is not found in the system, an error should print (see output). Otherwise, print the student details (see output). **Note: you MUST use Binary Search when searching for a student for this command AND you must print the indices viewed during the search. See output for clarification.

- **PRINTCOURSES** – Prints a listing of all courses open in the system. There will be no other information on the line. See output for printing possibilities.

- **PRINTSTUDENTS** – Prints a listing of all enrolled students. There will be no other information on the line. See output for printing possibilities.

See output for different printing possibilities.

To implement these commands, you should make many methods to help you. Here are **some** example methods:

```
void openCourse(courses, in, out);
void expandCourse(courses, in, out);
void deleteCourse(courses, students, in, out);
void enrollStudent(students, in, out);
```

These methods can be in your main program. They can be part of the FCITstudent class, or they can be part of the FCITbook class. YOU are the programmer! The choice is up to you and your style for how you want to use these methods and where you want to use them. Also, you may want to use more methods to help you do certain tasks as needed in the program.

Output Format

Your program must output to a file, called "**`FCIT_Registrar.out`**". **You must follow the program specifications exactly.** You will lose points for formatting errors and spelling.

Sample and Output Files

A sample input file, with corresponding output file, can be found on Blackboard (the files have too many lines to paste here).

NOTE: These files do NOT test every possible scenario. That is the job of the programmer to come think up the possible cases and test for them accordingly. You can be sure that our graders

will grade with very large input files, which are intended to, ideally, test all possible cases. It is recommended that you spend time thinking of various test cases and build your own input file for testing purposes.

*****WARNING*****

Your program MUST adhere to the EXACT format shown in the sample output file (spacing capitalization, use of dollar signs, periods, punctuation, etc). The graders will use very large input files, resulting in very large output files. As such, the graders will use text comparison programs to compare your output to the correct output. If, for example, you have two spaces between in the output when there should be only one space, this will show up as an error even through you may have the program correct. You WILL get points off if this is the case, which is why this is being explained in detail. Minimum deduction will be 10% of the grade, as the graders will be forced to go to text editing of your program in order to give you an accurate grade. Again, your output MUST ADHERE EXACTLY to the sample output.

Grading Details

Your program will be graded upon the following criteria:

1) Adhering to the implementation specifications listed on this write-up.
2) Your algorithmic design.
3) Correctness.
4) **Use of Classes, Objects, and Arrays of Objects. If your program is missing these elements, you will get a zero. Period.**
5) The frequency and utility of the comments in the code, as well as the use of white space for easy readability. (We're not kidding here. If your code is poorly commented and spaced and works perfectly, you could earn as low as 80-85% on it.)
6) Compatibility to the **newest version** of NetBeans. (If your program does not compile in NetBeans, you will get a large deduction from your grade.)
7) Your program should include a header comment with the following information: your name, **email**, course number, section number, assignment title, and date.
8) Your output MUST adhere to the EXACT output format shown in the sample output file.

Deliverables

You should submit a zip file with THREE files inside:

1. `FCITstudent.java`
2. `FCITcourse.java`
3. `FCIT_Registrar.java` (this is your main program)

**These files MUST be zipped up inside a zip file, and the file MUST be named according to the directions on Page 1 of this PDF.

***These three files should all be INSIDE the same package called **`fcit_registrar`**. If they are not in this specific package, you will lose points.

NOTE: your name, ID, section number AND EMAIL should be included as comments in all files!

Points will be deducted for now following these simply instructions! Please be careful.

Suggestions:

- Read AND fully understand this document BEFORE starting the program!
- Next, start by making the two classes to create student and book objects:
  - `FCITstudent` and `FCITcourse`
- So make those two classes, with all the data members, constructors, and accessor/mutator methods.
- Now, after you have made these two classes, make your main program file:
  - `FCIT_Registrar.java`
- This is your main program that will read the commands from the file, process the commands, and print to the output file.
- Use the `IOexample.java` program to help you create your main to read from a file and write to a file.
- Finally, one by one, implement the commands (OPENCOURSE, EXPANDCOURSE, ENROLLSTUDENT, etc).

Hope this helps.

Final suggestion: START EARLY!