Project #1 – 100 points

Due Date

Monday, February 13, 11:59pm.

Submission Instructions

1. Zip your project folder and submit the file to Canvas. The zipped file MUST include the following items.
   - **Source folder src,** containing the source code *.java files [60 points]
     - (a) Create a package to organize the source files; use all lowercase letters for the package name; you will **lose 2 points** if this is not done properly.
     - (b) MUST include the **@author** tag in the comment block on top of EVERY Java class and put the name(s) who implemented the Java class, or you will **lose 2 points**.
   - **Test specification**. Use a document editor and write the test cases you used to test the **isValid()** method of the Date class, and the **compareTo()** method of the Student class. [15 points]
   - **Testbed main()** in the following classes implementing the test cases in the test specification above.
     - (a) Date class [10 points]
     - (b) Student class [10 points]
   - **Javadoc** folder in which you must include all the files generated by the Javadoc. [5 points]
2. The submission button on Canvas will disappear after **February 13, 11:59pm**. It is your responsibility to ensure your Internet connection is good for the submission. **You get 0 points** if you do not have a submission on Canvas. **Projects sent through the emails will not be accepted.**

Project Description

Your team will develop a simple software to help a university processes the student roster for tuition purpose. The roster maintains a list of students who are currently registered. The initial phase is to provide a console-based interactive user interface to manage the student roster. The software will read a sequence of line commands entered by the user and write the results on the console.

A line command entered by the user will begin with an operation code followed by the data tokens needed to complete the operations for managing the student roster. An operation code will be represented with a single character or two characters in uppercase letters. The operation code and the data tokens will be delimited by one or more spaces in a line command. The operation code is **case-sensitive**, which means operation codes in lowercase letters are invalid and should be rejected by the software.

The initial requirement for the software is to provide the following functionalities:

(1) Add a student to the roster. A student is uniquely identified by the student's profile, which includes first name, last name, and date of birth. The system also needs to keep track of a student's major, and credits completed. For simplicity, let's assume the university currently includes a short list of majors and schools as listed below. In addition, we assume any student under the age of 16 is not allowed to register.

01:198, CS, SAS
01:640, MATH, SAS
14:332, EE, SOE
04:547, ITI, SC&I
33:136, BAIT, RBS

(2) Remove a student from the roster, given the student's profile.
(3) Display the roster sorted by profile (last name, first name and DOB), OR sorted by standing OR sorted by the school and majors; in ascending order (lexicographical order.) The credits completed is used to classify the standings as follows.

    Freshman, if credit completed is less than 30
    Sophomore, if credit completed is greater or equal to 30 and less than 60
    Junior, if credit completed is greater or equal to 60 and less than 90
    Senior, if credit completed is greater or equal to 90

(4) Change a student's major, given the student' profile and a new major code.
(5) List all the students registered with a specified school, for example, list all the students in SAS.

To provide the above functionalities, the operation codes in line commands are listed below.

- **A** – **add a student** to the roster. Data tokens needed to add a student includes the student's first and last name, date of birth, major and credit completed. Below is an example of a line command for adding a student. You can assume that the user will always enter enough data tokens in the following order: first name, last name, date of birth, major, and the credits completed.

  ```
  A John Doe 1/20/2003  EE 80
  ```

  Each data token is delimited by one or more spaces. The dates shall be given in **mm/dd/yyyy** format. The names and majors are NOT case-sensitive, that is, John Doe and john doe are the same student, if the dates of birth are also the same; EE and ee represent the same major. Human errors are very common. Your software shall not allow the student be added to the roster given the following conditions.

  - Any date of birth that is not a valid calendar date
  - The date of birth is today or a future date
  - A student who is less than 16 years old
  - The major doesn't exist
  - The student is in the roster already
  - Negative number of credit completed

- **R** – **remove the specified student** from the roster, for example,

  ```
  R Paul Siegel 5/1/1999
  ```

  Your software must reject if the user is removing a student who is not in the roster.
- **P** – display the roster sorted by last name, first name, and DOB.
- **PS** – display the roster sorted by standing.
- **PC** – display the roster sorted by school and major.
- **L** – list the students in a specified school, sorted by last name, first name, and DOB. For example, the following line command lists all the students in SAS. The school name is NOT case-sensitive.

  ```
  L SAS
  ```
- **C** – change a student's major. For example, the following line command changes the student's major to BAIT.

  ```
  C Carl Brown 10/7/2004 BAIT
  ```
- **Q** – stop the program execution and display `"Roster Manager terminated."`, so the user knows that your software has stopped running.

Project Requirement

1. You MUST follow the Coding Standard posted on Canvas under Week #1 in the "Modules". **You will lose points** if you are not following the rules.
2. You are responsible for following the Academic Integrity Policy. See the **Additional Note #14** in the syllabus.
3. Test cases for grading are included in the file **Project1TestCases.txt** and the associated output is in the file **Project1ExpectedOutput.txt**. Your project should be able to read the test cases from console in the same order without getting any exceptions and without terminating abnormally. Your program should be able to ignore the empty lines. The graders will run your project with the test cases in **Project1TestCases.txt** and compare your output with the expected output in **Project1ExpectedOutput.txt**. You will **lose 2 points** for each output not matching the expected output, OR for each exception causing the project to terminate abnormally. You MUST create an instance of the Scanner class to read from standard input (System.in), or you will **lose 5 points**.
4. Each source file (.java file) can only include one public Java class, and the file name is the same with the Java class name, or you will lose **-2 points**.
5. Your program MUST handle bad commands; **-2 points** for each bad command not handled, with a **maximum of losing 6 points**.
6. You are not allowed to use any Java library classes, EXCEPT the **Scanner**, **StringTokenizer, Calendar** and **DecimalForamt** class. **You will lose 5 points** FOR EACH additional Java library class imported, with a **maximum of losing 10 points**.
7. You are not allowed to use the Java library class **ArrayList** anywhere in the project, or use any Java library classes from the Java Collections, or **you will get 0 points for this project**.
8. When you import Java library classes, be specific and DO NOT import unnecessary classes or import the whole package. For example, `import java.util.*;`, this will import all classes in the `java.util` package. You will **lose 2 points** for using the asterisk "*" to include all the Java classes in the `java.util` package, or other java packages, with a **maximum of losing 4 points.**
9. You MUST include the Java classes below. **-5 points** for each class missing or NOT used. You should define necessary constant names with UPPERCASE letters and **DO NOT** use MAGIC NUMBERs, or you will **lose 2 points**. A good approach is to use Java enum classes or a public class to define all the constant names and their values. You **CANNOT use System.in** or **System.out** statements in all classes, EXCEPT the interface class RosterManager.java, **-2 points** for each violation, with a **maximum of 10 points off.** You must **always add the @Override** tag for overriding methods, or **-2 points** for each violation.

   (a) **Profile class –** must override the toString(), equals() and compareTo() methoed, or **-2 points each violation**. You cannot change or add the instance variables, or **-2 points each violation.**

   ```
   public class Profile implements Comparable<Profile> {
       private String lname;
       private String fname;
       private Date   dob;   //use the Date class described in (f)
           ...
   }
   ```

   (b) **Student class –** must override the toString(), equals() and compareTo() methods, or **-2 points each**. Two students are equals if they have the same profiles. Can't change or add instance variables, or **-2 points each.**

   ```
   public class Student implements Comparable<Student> {
       private Profile profile;
       private Major   major;             //Major is an enum type
       private int     creditCompleted;
   }
   ```

(c) **Roster class** – this is an array-based linear data structure to hold the list of students. An instance of this class is a growable container with an initial capacity of 4, and automatically grows (increases) the capacity by 4 whenever it is full. The container does not decrease in capacity. You CANNOT change or add instance variables, or **-2 points** for each violation.

```
public class Roster {
    private Student[] roster;
    private int size;

    private int find(Student student) {} //search the given student in roster
    private void grow() {} //increase the array capacity by 4
    public boolean add(Student student){} //add student to end of array
    public boolean remove(Student student){}//maintain the order after remove
    public boolean contains(Student student){} //if the student is in roster
    public void print () {} //print roster sorted by profiles
    public void printBySchoolMajor() {} //print roster sorted by school major
    public void printByStanding() {} //print roster sorted by standing
}
```

- You must implement the methods listed above, and you CANNOT change the signatures of the methods. **-2 points** for each method not implemented, signature changed or not used.
- You CAN use **System.out** ONLY in the **print()** methods.
- The **find()** method searches a student in the list and returns the index if it is found, it **returns -1** if the student is not in the roster. You must define a constant identifier "NOT_FOUND" for the value -1.
- The **remove()** method remove a student from the roster. This method maintains the order of the students in the array after the remove, **-3 points** if this is not done correctly.
- You must use an **in-place sorting algorithm** to sort the list of students in the array. That is, NO additional array can be declared for sorting, or you will **lose 10 points**. You must write code to implement the sorting algorithm. You CANNOT use **Arrays.sort()** or **System.arraycopy()** or any other Java library classes for sorting. You will **lose 10 points** if you do.

(d) **RosterManager class** – this is the **User Interface class** to process the line commands entered on the console and display the results on the console. An instance of this class can process a single line command, or a sequence of line commands. If your project cannot process a sequence of line commands, **you will lose 10 points**. This is an interactive program such that, it displays the results on the console whenever one or more line commands are entered, after the user hits the enter key.

When your project starts running, it shall display "`Roster Manager running...`". In this case, the user would know the software is ready to read the commands. It should continuously process the commands until the "Q" command is read. That is, the "Q" command is the only way to terminate the software normally. Before the software stops running, display "`Roster Manager terminated.`".

You must define a `public void run()` method that includes a while loop to continuously read the line commands until the user quits. You will **lose 5 points** if the run() method is missing. You MUST keep this method **under 40 lines** for readability, or you will **lose 3 points**. You should define necessary instance variables and private methods (helper methods) to handle different commands.

(e) **RunProject1 class** is a driver class to run your Project 1. The main method will call the **run()** method in the RosterManager class.

```
public class RunProject1 {
    public static void main(String[] args) {
        new RosterManager().run();
    }
}
```

(f) **Date class**

```
public class Date implements Comparable<Date> {
    private int year;
    private int month;
    private int day;

    public Date() {} //create an object with today's date (see Calendar class)
    public Date(String date) {} //take "mm/dd/yyyy" and create a Date object
    public boolean isValid() { } //check if a date is a valid calendar date
}
```

- You must implement the constructors and methods listed above. You must override toString(), equals() and compareTo() method, or **lose 2 points** for each violation.
- You CANNOT change or add instance variables, or **-2 points** for each violation.
- The **isValid()** method checks if a date is a valid calendar date.
    - For the month, January, March, May, July, August, October and December, each has 31 days; April, June, September and November, each has 30 days; February has 28 days in a non-leap year, and 29 days in a leap year. DO NOT user **magic numbers** for the months, days and years. Below are some examples for defining the constant identifiers for the constants.

    ```
    public static final int QUADRENNIAL = 4;
    public static final int CENTENNIAL = 100;
    public static final int QUATERCENTENNIAL = 400;
    ```

  To determine whether a year is a leap year, follow these steps:
  Step 1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
  Step 2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
  Step 3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
  Step 4. The year is a leap year.
  Step 5. The year is not a leap year.

    - You MUST design the test cases to test the **isValid()** method. You must write a testbed main and implement the test cases. You must include the 5 test cases for invalid calendar dates in **Project1_testCases.txt**. You must include at least 2 test cases for valid calendar dates. The testbed main for this class **is worth 10 points**. You CAN use System.out in main() to display the test results.

(g) **The enum class** for the majors. You must use an enum class to define the majors, which include a list of the major names, major codes, and school names, or you will **lose 5 points**.

```
public enum Major { }
```

10. You are required to **generate the Javadoc** after you properly commented your code. Your Javadoc must include the documentations for the constructors, private methods, and public methods of all Java classes. Generate the Javadoc in a single folder and include it in the zip file to be submitted to Canvas. **Please double check your Javadoc after you generated it by opening the index.html file**, and ensure the descriptions are NOT EMPTY. You will lose points if any description in the Javadoc is empty. You will **lose 5 points** for not including the Javadoc.