Project #2 – 170 points

Due Date

Monday, February 26, @11:59pm.

Submission

1. Zip your project folder and submit the zipped file to Canvas. The zipped file must include the following items.
   - **Source folder src,** containing the Java files (*.java) [**120 points**]
     (a) MUST create a Java package to hold the source files; MUST 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**.
   - **JUnit** test classes
     (a) Date class, test the isValid() method. [**15 points**]
     (b) MemberList class, test the add() and remove() methods [**20 points**]
   - **Class Diagram** [**10 points]**
   - **Javadoc** folder, including all the files generated. [**5 points**]
2. The submission button on Canvas will disappear after **February 26, 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. **Submitting the project through the email will not be accepted**.

Project Description

A fitness club owns the studios at 5 different locations in the central New Jersey area as listed below.

Bridgewater, 08807, Somerset County
Edison, 08837, Middlesex County
Franklin, 08873, Somerset County
Piscataway, 08854, Middlesex County
Somerville, 08876, Somerset County

Your team will develop a software to help the fitness club manages the memberships and class schedules. The software is an interactive system where the users enter the command lines on the terminal and the system displays the information on the terminal after the users hit the enter key. A command line specifies the operation the user wanted to perform and includes the data tokens required to complete the operation. An operation code is a single character or two-character command in uppercase letters. The operation code and the data tokens will be delimited by one or more spaces in a command line. Operation codes are case-sensitive, which means operation codes with lowercase letters should be rejected by the system. The software shall provide the functionality of adding and removing members, managing the attendance of the fitness classes, generating the billing statements, and displaying membership information in a specified order.

The fitness club currently offers 3 membership options – basic, family, and premium. Members must be 18 or older to join. There are 3 types of fitness classes, Pilates, Spinning and Cardio, being offered at different studio locations. Members with the basic membership can only attend the classes offered at the home studio. Members with family and premium memberships can attend the classes offered at any studio location. Anyone holding a guest pass can only attend the fitness classes held at the home studio. The software shall run the membership fee dues for all the members. The fee schedules are listed in the table below.

Project #2 – 170 points

Membership Type	Fee Schedule	Fitness Class Attendance
Basic	• $39.99, per month • billed monthly • additional charge of $10 per class exceeding 4 classes	• home studio only • no guest pass • 4 classes per billing cycle
Family	• $49.99 per month, • billed quarterly (3 months)	• Any studio location for the member • Home studio only for the guests • 1 guest pass at anytime
Premium	• $59.99 per month • billed annually with one month free (pay 11 months)	• Any studio location for the member • Home studio only for the guests • 3 guest passes at anytime

There are 5 instructors teaching the classes, including Jennifer, Kim, Denise, Davis, and Emma. The fitness club will provide the class schedule in a text file. This allows the change of schedule independent to the software. Currently, fitness classes are regularly offered every day in the morning, afternoon, and evening, at 9:30, 14:00 and 18:30, respectively. Each fitness class can be uniquely identified by the class name, instructor, and studio location. The system shall load the class schedule from a text file **classSchedule.txt** and load the historical member information from the text file **memberList.txt.** The system shall support the following operation codes (commands.)

- When **adding** a new member, the system shall create a profile for a new member with the first name, last name, and date of birth. The profile can be used to uniquely identify the member. There are 3 different add commands.
  - **AB**, to add a member with Basic membership, followed by member's first and last name, date of birth, and the location (in city name) of the home studio. Below is a sample command line for adding a member with a Basic membership. The system shall set the expiration date to 1 month later, e.g., if today is 3/31, the expiration date is 4/30, the last day of next month. The expiration date depends on the date when you are adding the member.

    ```
    AB  April  March  3/31/1990 Piscataway
    ```

    The date is given in a mm/dd/yyyy format. The names and locations are NOT case-sensitive, that is, April March and april march are the same person if the dates of birth are also the same; PISCATAWAY and Piscataway represent the same location. Your software shall not add the member to the database if the data tokens in the command line have the following conditions.

    1. Any date that is not a valid calendar date.
    2. The date of birth is today or a future date.
    3. A member being added is less than 18 years old. Note, the system checks the birthday month only, e.g., if today is 2/6/2024, dob 2/28/2006 is considered as eligible, but dob 3/1/2006 is not eligible.
    4. An invalid city name has been entered, that is, the studio location doesn't exist.
    5. The member is already in the member database.
    6. Not enough data tokens to complete the operation.
  - **AF**, to add a member with Family membership to the member database. Check all conditions listed above. The expiration date should be set to expire 3 months later.
  - **AP**, to add a member with Premium membership to the member database. Check all conditions listed above. The expiration date should be set to expire one year later.
- **C**, to cancel the membership and **remove** the member from the member database. Since the system checks the date of birth when adding a member, it doesn't check the dob when removing a member. However, the member being removed may not exist in the database. Sample command line: `C  Paul  Siegel  5/1/1999`

- **S**, display the class schedule with the current attendees. Each fitness class shall include the class name, instructor's name, the time of the class, and the list of members/guests who have registered. For simplicity, we abstract away the complexity of tracking the dates of the class schedule. That is, the system does not maintain the dates of the class schedule, only shows the "current" class schedule.
- **Printing** the list of members in the member database.
  - **PM**, to display the members sorted by member profiles, i.e., last names, then first names, then dates of birth.
  - **PC**, to display the members sorted by county names, then zip codes.
  - **PF**, to print the members with the membership fees. For simplicity, assume the membership fees are for the next billing statement, regardless of the membership expiration dates.
- **R**, to take attendance of a member attending a class and add the member to the class, including the class name, instructor, and member profile, for example:

  ```
  R cardio Jennifer somerville Roy Brooks 12/8/1977
  ```

  The system shall not add the member if the following conditions exist. Check in the order of the data tokens.
  - The class name does not exist.
  - The instructor does not exist.
  - The studio location does not exist.
  - The member is not in the member database.
  - The membership expired.
  - The fitness class in not on the class schedule.
  - Member with Basic membership can only attend the classes held at the home studio.
  - Time conflict – member is currently in a class held at the same time.
  - Member is already added to the class schedule.
- **U** command, to remove a member from a class, for example, `U Pilates Mary Lindsey 12/1/1989`

  Since the system checks the conditions when adding the member, it shall only check if the member is in the class to remove.
- **RG**, to take attendance of a guest attending a class and add the guest to the class. The system does not maintain guest information. Therefore, the system checks all the conditions in the R command, EXCEPT time conflict. In addition, the system checks the restrictions of the associated membership, including the home studio location and the number of guest passes allowed. Below is a sample command line to add a guest of Jonnathan Wei to a class.

  ```
  RG cardio Davis bridgewater Jonnathan Wei 9/21/1992
  ```
- **UG**, to remove the guest from a class: `UG pilates davis Edison Jerry Brown 6/30/1979.`

  The system shall not remove the guest if the guest is not in the class, or the fitness class does not exist.
- **Q**, to stop the program execution and display `"Studio Manager terminated."`, so the user knows that your software has stopped running normally.

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 required to follow the Academic Integrity Policy. See the **Additional Note #14** in the syllabus. If your team uses a repository hosted on a public website, you MUST set the repository to private. Setting it to public is considered as violating the academic integrity policy. The consequences of violation of Academic Integrity Policy are: **(i) your group receives 0 (zero) on the project, (ii) the violation is reported, (iii) a record on your file of this violation.**

3. Test cases for grading are included in the file **Project2TestCases.txt** and the associated output is in the file **Project2ExpectedOutput.txt**. Your project should be able to read the test cases from console in the same order with the test cases provided in **Project2TestCases.txt**, line by line without getting any exceptions. Your program should be able to ignore the empty lines. The graders will run your project with the test cases in **Project2TestCases.txt** and compare your output with the expected output in **Project2ExpectedOutput.txt**. Note that, the expiration dates in **Project2ExpectedOutput.txt** were generated on the day I ran the project. The expiration dates you generated should be different. The data used this project are not associated with any real-world entity and they are created solely for testing purpose.
4. You will **lose 2 points** for each output not matching the expected output, OR for each exception causing your project to terminate abnormally. You MUST use the **Scanner class** to read the command lines from the standard input (**System.in**), DO NOT read from an external file, **except memberList.txt** and **classSchedule.txt**, or you will **lose 5 points**.
5. 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**.
6. Your program MUST handle bad commands; **-2 points** for each bad command not handled, with a **maximum of losing 6 points**.
7. You are NOT allowed to import any Java library classes, EXCEPT the **Scanner**, **File, StringTokenizer, Calendar, DecimalForamt**, and Java **Exception** classes. **You will lose 5 points** for each additional Java library class imported, with a **maximum of losing 10 points**.
8. 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**!
9. 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.**
10. You must define the classes below with the specified inheritance relationships. **-5 points** for each class missing, or incorrect inheritance structure.
    - **Member** class defines the common instance variables and instance methods of the subclasses. See the details of this class in the requirement #12 below.
    - **Basic** class extends the Member class and includes the instance variable `private int numClasses;` to keep track of the number of classes attended so far; must override the bill() method, or **lose 3 points**.
    - **Family** class extends the Member class and includes the instance variable `private boolean guest;` to keep track of the guest pass; must override the bill() method, or **lose 3 points**.
    - **Premium** class extends the Member class and includes the instance variable `private int guestPass;` to keep track of the guest passes; must override the bill() method, or **lose 3 points**.
11. **Polymorphism** is required, i.e., dynamic binding for the equals(), compareTo(), toString(), and the bill() method or you will **lose 10 points.**
12. In addition to the classes in #10, you are required to include the Java classes below, or **-5 points** for each class missing. You must always add the **@Override** tag for overriding methods, **or -2 points** for each violation. All instance variables must be "private". You cannot add additional instance variables or change the method signatures for the methods listed or **-2 points** for each violation. You **CANNOT** perform read or write **with Scanner** or **System.out** in all classes, EXCEPT the user interface class StudioManager.java, and the print() methods in the MemberList and Schedule class or you lose **2 points** for each violation, with a **maximum of 10 points off.** The floating-point numbers must be displayed with 2 decimal places or **-1 point** for each violation.

(a) **Member class.**

```
 public class Member implements Comparable<Member> {
      private Profile  profile;
      private Date     expire;
      private Location homeStudio;

      public double bill() { } //return the next due amount
}
```

- **Location** is a Enum class defining the studio locations listed in Page #1 of this document.
- Reuse the **Date** class in Project 1.
- **Profile** class must be defined as follows. Must override equals(), toString() and compareTo() methods, cannot add or change the instance variables, or **-2 points** each.

```
 public class Profile implements Comparable<Profile>{
     private String fname;
     private String lname;
     private Date   dob;
 }
```

- You must implement **compareTo()** method and override **equals()** and **toString()** method **-2 points** for each violation. The toString() method returns a textual representation of a member in the following format.

```
Jane:Doe:5/1/1996, Membership expires 6/2/2024, Location: EDISON, 08837, MIDDLESEX
```

(b) **MemberList class.** This is an array-based implementation of a linear data structure to hold a list of member objects. An instance of this class is a growable list with an initial array capacity of 4, and it automatically increases the capacity by 4 whenever it is full. The list does not decrease in capacity. An instance of this class can hold a list of members with different types of membership.

```
 public class MemberList {
     private Member [] members; //holds Basic, Family, or Premium objects
     private int       size;    //number of objects in the array

     private int  find(Member member) { }
     private void grow() { }
     public boolean contains(Member member) { }
     public boolean add(Member member) { } //add to end of array
     public boolean remove(Member member) { } //shift up to remove
     public void load(File file) throws IOException { }//from the text file
     public void printByCounty() { } //sort by county then zip code
     public void printByMember() { } //sort by member profile
     public void printFees() //print the array as is with the next due amounts
 }
```

- 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.
- The **find()** method searches a member in the list and returns the index if it is found, it **returns -1** if the member is not in the list. You must define a constant identifier "NOT_FOUND" for the value -1.
- The **remove()** method remove a member from the list. This method maintains the relative order of the members in the list after the remove, **-3 points** if this is not done correctly.
- You must use an **in-place sorting algorithm** to rearrange the order of member objects 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 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.

- **load()** method loads the list of members from a text file.

(c) **Schedule** class. An instance of this class holds a list of fitness classes loaded from the text file **classSchedule.txt.** You cannot add or change the instance variables, and must implement the `load()` method or **-2 points** each violation. You can add constructors and methods.

```
public class Schedule {
    private FitnessClass [] classes;
    private int numClasses;
    public void load(File file) throws IOException { }
}
```

(d) **FitnessClass** class. An instance of this class holds the information of a fitness class and the attendees for members and guests. You cannot add or change the instance variables, or **-2 points each.**

```
public class FitnessClass {
    private Offer      classInfo;
    private Instructor instructor;
    private Location   studio;
    private Time       time;
    private MemberList members;
    private MemberList guests;
}
```

- **Offer** is a Enum class to include 3 types of fitness classes: Pilates, Spinning, and Cardio.
- **Instructor** is a Enum class to include the 5 instructors.
- **Time** is a Enum class. You cannot add change the constant values, or **-2 points** each.

```
public enum Time {
    MORNING   (9, 30),
    AFTERNOON (14, 0),
    EVENING   (18, 30);
}
```

(e) **StudioManager class**

- This is the user interface class to process the command lines. An instance of this class can process a single command line or multiple command lines at a time. **You will lose 10 points** if it cannot process multiple command lines at a time.
- This class should handle all possible Java exceptions thrown by the backend classes, or **-5 points** each violation for handling the exceptions in the backend classes.
- When your project starts running, it shall load the 2 text files first, and then display "`Studio Manager is up running...`". so, the user knows the software is ready to read the command lines. It should continuously process the command lines until the "Q" command is entered. The system displays "`Studio Manager terminated.`" and then terminates the project execution.
- You must define a **`public void`** `run()` method to process the command lines. 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**.

(f) **RunProject2 class** is a driver class to run your Project 2. The main method will call the **run()** method in the StudioManager class.

```
public class RunProject2 {
    public static void main(String[] args) {
        new StudioManager().run(); }

}
```

13. **JUnit test classes.** Write code to test the following methods.
    - **Date** class **– isValid()** method in the Date class, 5 invalid cases, 2 valid cases. You can reuse the test cases you have in the testbed main() from Project 1.
    - **MemberList** class
        - **add()** method **–** 3 true cases and 3 false cases for adding Basic, Family and Premium.
        - **remove()** method **–** 1 true case and 1 false case for removing a random member.
14. **Class Diagram.** Create a class diagram to document your software structure for Project 2. Hand-drawing the diagram is not acceptable, and you'll **lose 10 points**. You can follow the links below to create a class diagram online and save it to your device as a picture, which can be included in your submission. You can also use other UML tools if you like. For each rectangle (class), include ONLY the instance variables and public methods. NO need to include the constants and private methods. DO NOT INCLUDE the JUnit test classes, the Java library classes and RunProject2 class in the diagram.
    - http://draw.io/ → Create New Diagram → UML → Class Diagram
    - https://online.visual-paradigm.com/app/diagrams/ → Create New → Class Diagram
15. 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** in the folder, 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.