Project #1 – 120 points

Due Date and Time

Monday, February 12, by 11:59pm.

Submission

(1) Zip your project folder and submit the zipped file to Canvas.
  (a) Windows: right click the folder and select Compress to Zip file
  (b) Mac/Linux: right click the folder and select Compress

(2) The zip file must include the following grading items.
- **Source folder src,** containing the Java files (*.java) [80 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**.
- **Test specification**, include the test cases to test the **isValid()** method of the **Date class**, and the **compareTo()** method of the **Artist class**. [15 points]
- **Testbed main()** in the following Java classes implementing the test cases in your test specification.
  (a) Date class [15 points]
  (b) Artist class [5 points]
- **Javadoc** folder named **doc** to include all the files generated by the Javadoc. [5 points]

(3) The submission button on Canvas will disappear after **February 12, 11:59pm**. It is your responsibility to ensure your Internet connection/speed is good for submitting the project on time. **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 software "Collection Manager" to manage an album collection. For each album, the system shall keep track of the title, release date, artist, genre, and ratings. The software shall provide the following functionality.

1. Add an album to the collection.
2. Remove an album from the collection.
3. Rate an album in the collection, with a scale of 1 to 5.
4. Display the albums in the collection sorted by release dates, by genre, or by rating.

The software is an interactive system where the users enter command lines on the terminal, and the system immediately generates responses and output the results on the terminal. That is, when the user hits the enter key, the system reads the data entered, process the data, and immediately output the results.

A command line always begins with a command and followed by additional data tokens delimited by commas. Commands are in uppercase letter(s) and **case-sensitive**, which means the commands in lowercase letters are invalid. However, the data tokens are NOT case-sensitive, which means "FEARLESS" and "fearless", "POP" and "pop", "TAYLOR SWIFT" and "taylor swift" are equivalent. Below is a list of commands the software must support.

- **A** command; to **add an album** to the collection. The user shall enter the information about the album in the following sequence: the title, artist's name and date of birth, genre, and release date. Below is an example of a command line for adding an album to the collection.

```
A,Fearless,Taylor Swift,12/13/1989,pop,11/8/2008
```

The software classifies the albums with 4 genres: Pop, Classical, Jazz and Country. Other genres will be classified as "Unknown". You can assume that the user will always enter enough data tokens if a valid command is entered. The date of birth and release date shall be given in a **mm/dd/yyyy** format. Your software shall check the following conditions before adding an album to the calendar. Refer to the expected output for the error messages to display.

1. The date of birth of an artist is not a valid calendar date.
2. The date of birth of an artist is today or a future date.
3. The date of birth of an artist is before the year of 1900.
4. The release date is not a valid calendar date.
5. The release date is today or a future date.
6. The release date is before the year of 1900.
7. An album with the same title and artist is already in the collection.

- **D** command, to **remove the album** from the collection, for example,

  ```
  D,Blue,April,1/11/2015
  ```

  The system shall use the title and artist to uniquely identify an album in the collection. Since the system checks the dates when adding an album, it doesn't check the date when removing an album. However, the album being removed might not exist in the collection.
- **R** command, to rate an album with a scale of 1 to 5, for example,

  R,Fearless,Taylor Swift,12/13/1989,5

  The system shall use the title and artist to identify an album in the collection and add the rating. The system shall check if an invalid rating number has been entered.
- **PD** command, to display all the albums in the collection, sorted by the release dates. If two albums are released on the same day, sort by the titles in lexicographical order.
- **PG** command, to display all the albums in the collection, sorted by the genres in lexicographical order. If two albums have the same genre, sort by the artists. If two artists have the same name, sort by their dates of birth.
- **PR** command, to display all the albums in the collection, sorted by the average ratings. If two albums have the same average rating, sort by the titles in lexicographical order.
- **Q** command, to stop the execution of the software and display `"Collection Manager terminated."`

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 **Project1TestCases.txt.** The associated output generated from the test cases is in **Project1ExpectedOutput.txt**. The source of the test data is from allmusic.com. Some of the test data are modified for testing purpose.
4. Your project should be able to read the test cases from **System.in** in the same sequence with the sequence in **Project1TestCases.txt**, line by line without getting any exceptions. Your project should be able to ignore the empty lines between the test cases. 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 your project to terminate abnormally. You MUST use **Scanner class** to read the command lines from the standard input (**System.in**), DO NOT read from an external file, 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.
7. 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**.
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 include the Java classes below. **-5 points** for each class missing or NOT used. You should define necessary constant names in 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 constants.

    You **CANNOT use System.in** or **System.out** statements in ALL classes, EXCEPT the interface class CollectionManager.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) **Album class**

```java
public class Album {
    private String title;
    private Artist artist;
    private Genre  genre;
    private Date   released;
    private Rating ratings; //a linked list of ratings

    public void rate(int star) { } //add a rating to the linked list
    public double avgRatings() { } //compute the average ratings
}
```

    - The instance variable `ratings` contains the reference to the head of a singly linked list of ratings. You must implement the `rate()` method to add a rating to the linked list, and implement the `avgRatings()` method to compute the average rating by traversing the linked list.
    - You can add necessary constants, constructors, and methods. However, you CANNOT change or add instance variables. **-2 points** for each violation.
    - You MUST override **equals()** and **toString()** methods. You must add the **@Override** tags. **-2 points** for each violation. The **equals()** method returns true if the titles and artists of the albums are the same. Titles are NOT case-sensitive. The **toString()** method returns a textual representation of an album in the following formats. The asterisks are used to display the rating scales, e.g., *(2) means there are 2 ratings with the scale of 1 star. The `repeat()` method a `String` can be used to repeat the "*".

```
[Fearless] Released 11/8/2008 [Taylor Swift:12/13/1989] [Pop] Rating: *(2)**(1)***(2)****(2)*****(3)(average rating: 3.30)

[Queen of Hearts] Released 1/1/1989 [Joan Baez:1/9/1941] [Unknown] Rating: none
```

(b) **Rating class**. This class defines a node in a singly linked list that maintains the list of ratings. You can add constructors and methods, but you CANNOT add/change the instance variables, or **-2 points off** each violation.

```
public class Rating {
    private int    star;
    private Rating next;
}
```

(c) **Genre class**

- A Enum class to include Pop, Country, Classical, Jazz, and Unknown.

(d) **Collection class**

- This is an array-based implementation of a linear data structure to hold the list of albums. A new album is always added to the end of the array. 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.

```
public class Collection {
    private Album[] albums; //list of albums
    private int     size;   //number of albums in the list

    private int find(Album album) //helper method
    private void grow() //helper method to increase the capacity by 4
    public boolean contains(Album album)
    public boolean add(Album album) //false if the album exists
    public boolean remove(Album album) //false if the album doesn't exist
    public void rate(Album album, int rating)
    public void printByDate()  //sort by release date, then title
    public void printByGenre() //sort by genre, then artist
    public void printByRating()//sort by average rating, then title
}
```

- You can add necessary constants, constructors, and methods. However, you CANNOT change or add instance variables. **-2 points** for each violation.
- You MUST implement and use the methods listed above; you CANNOT change the signatures of the methods. **-2 points** for each violation.
- You CAN use **System.out** ONLY in the three **print()** methods listed above.
- The **find()** method searches an album in the list and returns the index if it is found, it **returns -1** if it is not in the list. You must define a constant identifier "NOT_FOUND" for the value -1.
- The **remove()** method delete an album from the collection. This method maintains the relative order of the events in the array after the deletion, i.e., shift every element in the array up one position. **-3 points** if this is not done correctly.
- You must use an "in-place" sorting algorithm to implement the sorting, i.e., the order of the objects in the array will be rearranged after the sorting without using an additional array. You CANNOT use **Arrays.sort()** or **System.arraycopy()** or any other Java library classes or utilities for sorting. You must write the code yourself for sorting. You will **lose 10 points** for the violation.

(e) **CollectionManager 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.

- When your software starts running, it shall display `"Collection Manager is up running."`. Next, it will read and process the command lines continuously until the "Q" command is entered. Before the software stops running, display `"Collection Manager terminated"`.
- You must define a `run()` method that includes a while loop to continuously read the command lines until a "Q" command is entered. 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 can define necessary instance variables and private helper methods for handling the commands.

(f) **Date class**

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

    public  boolean isValid() //check if the date is a valid calendar date
}
```

- You can add necessary constants, constructors, and methods, however, you CANNOT change or add instance variables, **-2 points** for each violation.
- You MUST implement Comparable Interface and implement the **compareTo()** method, or **lose 2 points.**
- You MUST include a testbed main() in this class, or you **lose 15 points**. You CAN use **System.out** in the testbed main().
- You MUST implement the **isValid()** method to check 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. You can use Calendar class or defined constant names. Below are some examples for defining the constant names.

    ```
    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 thoroughly test the **isValid()** method. Follow the instructions under the "Test Design" section in the **Coding Standard** posted on Canvas. You must implement the test cases in the testbed main **or lose 15 points**.

(g) **Artist class**

```
public class Artist implements Comparable<Artist> {
    private String name;
    private Date   born;
}
```

- You can add necessary constants, constructors, and methods. However, you CANNOT change or add instance variables, **-2 points** for each violation.

- You MUST implement Comparable Interface and implement the `compareTo()` method, or **lose 2 points.** Compare the name of the artist first, then date of birth. Names are NOT case-sensitive.
- You MUST override **equals()** and **toString()** methods. You must add the **@Override** tags. **-2 points** for each violation.

(h) **RunProject1 class.** This is a driver class to run your software.

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

11. **Test Specification;** You must use the table template in the Coding Standard, **or you will get 0 points for this part. -1 point** for each test case missing.
    (a) Date class – design five invalid and two valid test cases for testing the `isValid()` method,
    (b) Artist class – design test cases for testing the `compareTo()` method; two test cases return -1, two test cases return 1, and one test case returns 0.
12. **Testbed main()** is the main() method in each Java class to exercise and test the public methods within the same class. You MUST implement the test cases designed in the Test Specification.
    (a) Date class – 7 test cases, **-2 points** for each test case missing.
    (b) Artist class – 5 test cases, **-1 point** for each test case missing.
13. You are required to **generate the Javadoc** after you properly comment your code. Set the scope to 'private" so your Javadoc will include the documentations for the private data members, constructors, private 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. **You are responsible to double check your Javadoc folder after you generated it. Submitting an empty folder will result in 0 points for this part.** Open the Javadoc folder and look for the **index.html** file. Double click the file and check every links to ensure all comments 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.