CSci 1933 Assignment 3

This assignment is due on **Friday, November 13th, 2015** at **11:55 PM**

General:

1. **You are supposed to work in group of 2 (unless exempted by the instructor), otherwise you will lose 20% of your grade.**
2. **Some of the files/code we gave you are not needed at all in this assignment. Please don't change them if you are not asked to.**
3. **Likewise, there are some methods which are commented as TODOs. If they are not mentioned in the instruction, you don't need to implement them.**
4. **When an assignment description tells you to give something a particular name, you must give it exactly that name. Failure to do so may result in a very low or zero score due to inability to properly run your code.**
5. **Your program must output its results in exactly the format we describe, with no additional text being written. If you insert additional output statements to aid in debugging, make sure they are removed before you submit.**

You have the options 1) to work with the previous teammate or 2) to find a new teammate. If you have difficulties finding a teammate, please post to Student Forum on Moodle site **no later than Nov. 10th**.

1. Overview

1.1. In this assignment, you will implement the infrastructure for a sortable "leaderboard" for a nation-wide game. Participants in the game completed various tasks and received scores for each task. For the purposes of the assignment, we will assume the competition lasted a full year. Participants could begin doing tasks -- and therefore getting scores -- as soon as the year began and could continue for the whole year.

1.2. Information about each participant includes their name, where they are from (a US state), and an entry for each task they completed. Each entry consists of a score and the month and day on which the entry was submitted.

1.3. You will implement several techniques to create and sort a leaderboard of participants. First, you will define a class for a `Participant` (details below) that implements the `Comparable` interface, just as you have done in a previous lab. However, in this assignment, you'll want to be able to sort the list of participants in more than one way. To enable this, you also will implement a `Comparator`. A `Comparator` is a class that defines more ways of sorting objects in addition to the order defined by their `compareTo` method. For more information about the `Comparator` class, see the Java documentation for `Comparator`.

2. File setup

2.1. We provide a number of different files to get you started:

- DateFilter.java - the DateFilter class. Please import this into your src/ folder.
- Entry.java - the Entry class. Please import this into your src/ folder.
- Filter.java - the Filter interface. Please import this into your src/ folder.
- Leaderboard.java - the Leaderboard class. Please import this into your src/ folder.
- LeaderboardReader.java - a class that reads data from a file, and returns the required Objects. Please import this into your src/ folder.
- Participant.java - the Participant class. Please import this into your src/ folder.
- ScoreFilter.java - the ScoreFilter class. Please import this into your src/ folder.
- SortedList.java - the SortedList class. Please import this into your src/ folder.
- NameComparator.java - the NameComparator class. Please import this into your src/ folder.
- participants.dat - the data file, which defines participants, and their attributes. Please create a folder "`dat/`" in your project root and import this file into that folder.
- build.xml - the Gradle build file. Please import this into your project folder.

2.2. The `LeaderboardReader` class operates similarly to `TweetReader` and other `Reader` classes from previous assignments. It follows the same general structure for accessing records from the data file:

```
while advance() returns true
   process a record
```

However, the `LeaderboardReader` class is different from previous `Reader` classes in one key way: it can read and return different types of objects (e.g., a `Participant` or an `Entry`). Therefore, you will need logic in your `Leaderboard` class (specifically, in the `readParticipantData()` method) to check the type of the record returned and implement the behavior appropriate for that type.

2.3. We provide you with a participants.dat file, which contains the data that you will be interacting with. The structure of this file is as follows:

```
Ani,MN
37,3,14
45,5,27
80,11,12
14,12,1

Bob,WI
49,1,25
52,2,14
20,4,19
70,6,29
75,6,30
79,8,23
```

```
90,10,31

Caroline,MN
72,4,13
74,5,25
83,11,27
```

In particular, note that each `Participant` (and the associated `Entries`) are separated by a blank line.

In addition to these instructions, we have provided `javadoc` documentation for the classes we provided and that you must complete. When we refer to "the javadoc" in these instructions, we are referring to what javadoc folder provided with this assignment. The html files in `javadoc` folder can be viewed in your browser.

3. Part 1

3.1. For Part 1, your tasks are:

- implement the `SortedList` class
- implement the `compareTo()` and `computeAndSetAverageScore()` methods in the `Participant` class
- implement the `readParticipantData()` method in the `Leaderboard` class
- call `readParticipantData()` and print the leaderboard in the `main()` method of the `Leaderboard` class

3.2. `SortedList` implementation

3.2.1. Before reading data for `Participants`, you must implement the `SortedList` class. To see precisely what you need to do, please refer to the the javadoc for `SortedList` and the stub method definitions in `SortedList.java`. You must implement each of these methods.

3.2.2. There are various ways to implement a sorted list. You will implement `SortedList` with a linked list of `Node` objects, as we have done in previous Labs. A `Node` is just the same as you have seen in Lab: it must contain a data field and a reference to the next node in the linked list. However, the way you add entries to a `SortedList` is different. By definition, elements are added to maintain their natural order as defined by the `compareTo` method of their class. Again, you must implement each of the methods in the javadoc for `SortedList`.

3.2.3. The algorithm for properly inserting an item into a sorted list can be found in the textbook. It includes a description of the algorithm, the code, and a deeper description of how a sorted linked-list works. This can be found on 4th edition of the book starting from page 482. If you use the textbook as

part of your implementation, please note it in the comments above your code as follows:

```
// based on the implementation provided in the 4th Edition of Data Structures and Abstractions with Java, on page 482.
```

Note It may be useful to write a unit test for `SortedList` to ensure that their implementation is working correctly. (The unit test is not required in your submission.)

3.3. `Participant.compareTo()` implementation

3.3.1. Once you have completed implementing your `SortedList`, implement the `compareTo()` method in the `Participant` class. It will make the later portion of Part-1 easier. Remember that `compareTo()` returns:

- -1 if this item is less than the item it's being compared against
- 0 if this item is equal to the item it's being compared to
- 1 if this item is greater than the item it's being compared against.

You can refer to the javadoc for `Participant` for the method signature and more specific implementation details.

3.4. `readParticipantData()` implementation

3.4.1. Once you've completed your `compareTo()` implementation, you can now start working on populating the `SortedList` that you implemented previously. You will do this by implementing the `readParticipantData()` method and using the `LeaderboardReader` class that is provided. Remember that `LeaderboardReader` acts similarly to `TweetReader` from past assignments, but that the `getRecord()` method may return multiple types of `Objects`.

3.4.2. Therefore, you will need to check the type of the `Object` that is being returned. You must insert each `Entry` associated with a given `Participant` to that `Participant`'s `listOfEntries`. An empty `String` object returned will denote the end of a given `Participant` as in the file format above.

3.4.3. Once you have properly read the `Entries` and added them to the corresponding `Participant`, you will then need to compute an average score of all the `Entries`, for each `Participant`. You will do this by implementing the `computeAndSetAverageScore()` method within the `Participant` class. Please refer to the javadoc for `Participant` for more details about this method.

3.4.4. Make sure you call `computeAndSetAverageScore()` within the `readParticipantData()` method, to ensure that each `Participant` has an average score prior to inserting the `Participant` into the `SortedList`.

This will look something like:

- `while advance()`
  - `process a record`
  - `if record instanceof Participant`
    - `tempParticipant = object`
  - `if record instanceof Entry`
    - `tempParticipant.addEntry(object)`
  - `if record instanceof String`
    - `tempParticipant.computeAndSetAverageScore()`
    - `sortedList.add(tempParticipant)`

3.5. Print the output

Now that you have properly inserted each `Participant` into the correct position in the `SortedList`, please print your list in the `main()` method of `Leaderboard.java`. You will need to iterate through the list returned from the `toList()` method that you implemented for your `SortedList`. Remember that there is a `toString()` method defined in the `Participant` class. Please print "`sort_by_scores:`" (without quotation) on the first line of the output. See the sample output:

```
sort_by_scores:
Name: Brent, Average score: 100
Name: Henry, Average score: 99
Name: Saurav, Average score: 98.34
Name: Allen, Average score: 97.23
…
```

4. Part 2

Now, consider how you would go about re-ordering the Leaderboard list by different mechanisms. The `compareTo()` method you implemented in Part-1 provides the natural order for an object, but there are cases where you may want to order objects based on different criteria. Java provides a mechanism to do this, called a `Comparator`.

4.1. Using a `Comparator`

As stated at the beginning of the write-up, a `Comparator` is a class that defines a `compare()` method, as specified by the `Comparator` interface that this class is implementing. The mechanics of this `compare()` method are left to the student, but in general, the return values of `compare()` act in a very similar way to those in the `compareTo()` methods you have seen previously. However, in order to use the `Comparator` method when sorting, you will make a call to `Collections.sort()` in a similar way to:

```
Collections.sort(myList, new myComparator());
```

Implement a new class called `NameComparator` that sorts names in lexicographically DESCENDING order (the same order as String.compareTo() does). Refer to the lecture from November, 04 and online documentation for how to implement a Comparator.

4.2. Print the output

Upon completing `NameComparator`, create a new list (by calling the `toList()` method from Part-1 in the `main()` method of your `Leaderboard` class). Call `Collections.sort()` method (similar to the way listed above, with the `NameComparator` as a parameter), and print out the the new, sorted list of `Participants`. Verify that the leaderboard is now sorted alphabetically. Note that this should sort in DESCENDING order. To differentiate from the list you print out in 3.5, please print "`sort_by_names:`" (without quotation) on the first line of the list. See sample output:

```
sort_by_names:
Name: Allen, Average score: 97.23
Name: Brent, Average score: 100
Name: Henry, Average score: 99
Name: Saurav, Average score: 98.34
...
```

5. Submission

Before you submit your solution, **you must create a text file called `group.txt` in your `src/` directory**. In this file, put the names and x.500 ID's of the members of your group.

Put `build.gradle` file in the project root and run command `gradle tar`. This will create a tar.gz file for submission. Please make sure the `src/` and `dat/` files are in the created tar.gz file.

Submit the tar.gz to the **Lecture Moodle Site.**

---End of Assignment 3---