Project #2

Computer Science 2334
Spring 2010

User Request:

"Create a Sortable and Searchable Award Winner Information System."

Milestones:

1.	Use keyboard input to get information from the user.	*5 points*
2.	Use text file I/O to read and write text files.	*10 points*
3.	Create a class to store award information, one to store information on people, one to store information on people who have won awards, and one to store a collection of award-winning people. Note that you should create any additional classes (abstract and/or concrete) and/or interfaces you deem necessary to arrive at a good design.	*10 points*
4.	Implement both the **Comparable** and **Comparator** interfaces to compare one person's information to another's.	*10 points*
5.	Use a list to store, retrieve, and display information related to people as described below.	*15 points*
6.	Use the `sort()` and `binarySearch()` methods from the **Collections** class to sort and search for information related to the description below.	*20 points*
►	Develop and use a proper design. (See Milestone 3, above.)	*15 points*
►	Use proper documentation and formatting.	*15 points*

Description:

For this project, as with Project #1, you will put together several techniques and concepts learned in CS 1323 and some new techniques to make an application that searches a large collection of information on people who have won awards. We will call this application a "sortable and searchable award information system" or "SSAIS," since the primary features of this system are its sortability and searchability. Note that much of the code you write for this program could be reused in more complex applications, as we will see in later assignments.

Your software will first ask the user for the name of the data file where the awardee information is stored. This should be done using the technique described in the section below on reading input from the keyboard. It will then read in the specified data file and store the information into the SSAIS. Each entry in the data file contains a person's last name, first name, zero or more middle names, and zero or more awards that person has won. The award information itself will consist of the award name and year for each award. The format of this file is described below under Input Format.

Once the information is loaded from the file, your program will enter a loop where it asks the user for criteria on which to sort the data. As with the file name, this information should come from the keyboard using the technique described below. The possible sorting options the user can enter are 'F' for first name, 'L' for last name, 'FL' for first name then last name, 'LF' for last name then first name, 'N' for number of awards, and 'R' for random. Alternately, the user may choose to enter 'PC' for print to console, 'PF' to print to a file, or 'S' for search. If the user chooses either print option, the award information will be printed in its current order (whatever that may be, given the last sort option) and in the format described below under Output Format. (If the print option is 'PF' the user will also be prompted for an output file name.) If the user chooses search, he or she will be prompted for the first and last name of an award winner on which to search and your program will search for and display the information on that one award winner. (You may assume that the combination of first name and last name is unique in the SSAIS.) The final option available to the user is 'Q' for quit. If the user chooses quit, your program will thank him or her for running the program and exit without errors.

Learning Objectives:

Sorting and Searching:

Sorting information can be useful to users because the output may be organized in a way that makes it easier to use. It can also be useful to software developers because it can improve the efficiency of their software. Consider finding personal information based on first and last name. If the data structure holding the personal data is unsorted, you need to do a linear search through it to find an entry. However, if the data structure is sorted based on name, you can do a binary search instead. A binary search will, in most cases, take far fewer comparisons to find the desired entry than a linear search.

To observe this efficiency gain, you will measure the amount of time the system uses to find an award winner based on name, given the ordering of the data in the SSAIS. **Do not count the time the system uses for reading in the data or carrying out other activities, like waiting for the user to provide input.** Sort the data using one ordering (such as 'F') then search for five different names, record the values for search times, repeat this for each possible ordering of the data, then present them in a simple table using the format shown below. (You may need to adjust spacing for long names.)

```
                 1.             2.             3.             4.             5.
Last Name:
First Name:
Order        Search Time    Search Time    Search Time    Search Time    Search Time
-----        -----------    -----------    -----------    -----------    -----------
 F
 L
 FL
 LF
 N
 R
```

Note that some of the sort options are not related to the search terms that the user will be providing or do not define a unique ordering. For example, the number of awards (one of the possible sort options) is not related to the name of the person searched for (the search term). Moreover, the ordering defined by this sort option is not unique. (Several people may have the won the same number of awards, so the ordering of these awardees based on this criterion is arbitrary.) For these situations, you will have no choice but to search the collection linearly. On the other hand, when the sort option is related to the search terms and does define a unique ordering, a binary search is preferred and should be used. For which sort option(s) is it appropriate to use a binary search (F, L, FL, LF, N, or R)? *Explain your answers.* For which sort option(s) is it *not* appropriate to use a binary search? *Explain your answers.*

Put the table of data and your answers to these questions into milestones.txt under milestone 5.

Note that the 'FL' sort option sorts based on first name then last name which means that the names Joey Ramone, Johnny Ramone, and Johnny B. Goode would be ordered: (1) Joey Ramone, (2) Johnny B. Goode, and (3) Johnny Ramone because the first name Joey comes before the first name Johnny and the last name Goode comes before the last name Ramone. Ordering these same names based on the 'LF' sort option would give the order (1) Johnny B. Goode, (2) Joey Ramone, and (3) Johnny Ramone because the last name Goode comes before the last name Ramone and the first name Joey comes before the first name Johnny. (Note that the middle initial has no effect in either ordering.)

Note that each collection can have at most one *natural ordering*. You should determine an appropriate natural ordering for the award winners and define the `compareTo()` method of the award winner class (which must implement **Comparable**) to use that ordering. The other sort options will need to be implemented using `compare()` methods that come from implementing **Comparator**.

Input/Output Formats:

Input Format

Each entry in the awardee data file contains a person's last name, first name, zero or more middle names, and information on zero or more awards the person has won. The award information itself consists of an award name and an award year for each award. Note that the information on each award should be stored in an object of the award class, and each award object should then belong to an award winner. Note also that the award winner class should be derived from a more general person class and should allow for each award winner to have an arbitrary number of awards.

In the awardee data file each line contains all of the information on a single award winner. Within each line, the information is ordered last name, first name, then any middle names, where each name is separated from the next by a comma and a space, and the final name is followed by a semi-colon and a space, followed by the information on each award in the order award name then award year where the parts of a single award are separated by a comma and a space and the awards are separated from each other by a semi-colon and a space. For example:

```
Marley, Robert, Nesta; UN Peace Medal, 1978; Order of Merit, 1981
```

This line gives information on a person with the last name "Marley," first name "Robert," and middle name "Nesta." It shows that he has won two awards: (1) a UN Peace Medal in1978 and (2) the Order of Merit in 1981. If a person has won no awards, there will be no semi-colon and no award information.

Output Format:

The text written out for each award winner must conform to the following output format.

Line 1: First name, middle name(s), last name.
Line 2: First award name, year (if any).
Line 3: Second award name, year (if any).
Line 4: Third award name, year (if any).
…
Line n: Last award name, year (if any).
Line n+1: Blank line.

Sample Output:

```
Robert Nesta Marley
UN Peace Medal, 1978
Order of Merit, 1981

```

Implementation Issues:

File I/O:

To perform output to a file, use the **FileWriter** class with the **BufferedWriter** class as follows.

```
FileWriter outfile = new FileWriter("output.txt");
BufferedWriter bw = new BufferedWriter(outfile);
bw.write("This is a test -- beep.");
bw.newLine();
bw.close();
```

When you have finished writing to a file, you must remember to close it, or the file won't be saved. If you fail to close the file, it will be empty!

Remember to add '`throws IOException`' to the signature of any method that uses a **FileWriter** or **BufferedWriter** or that directly or indirectly calls a method that performs File I/O.

Reading Input from the Keyboard:

In order to get the personal information from the user, you need to read input from the Keyboard. This can be done using the **InputStream** member of the **System** class, that is named "`in`". When this input stream is wrapped with a **BufferedReader** object, the `readLine()` method of the **BufferedReader** class can be used to read and store all of the characters typed by the user into a **String**. Note that `readLine()` will *block* until the user presses the Enter key, that is, the method call to `readLine()` will not return until the user presses the Enter key.

The following code shows how to wrap and read strings from `System.in` using an **InputStreamReader** and a **BufferedReader**.

```
BufferedReader inputReader = new BufferedReader(
                                new InputStreamReader( System.in ) );
System.out.print( "Type some input here: " );
String input = inputReader.readLine();
System.out.println( "You typed: " +  input );
```

You need to add '`throws IOException`' to the signature of any method that uses or that directly or indirectly calls a method that uses a **BufferedReader** or **InputStreamReader**.

How to Complete this Project:

1.Revise your UML design from the lab session. Be sure to clearly write your name and the names of your group members and "Project 1" on the cover sheet. **Remember to not include any personally identifying information on your project other than on your cover sheet.** Make sure to keep a copy of your UML when you turn it in.

2.Create the classes and methods specified in your design, but do not put code in the methods. Add the required documentation to your classes and methods as specified in the documentation requirements posted on the class website. This is called "stubbing" your classes and methods.

3.Run your stubbed Java files through Javadoc as described in the Lab #2 slides. This will create a set of HTML files in a directory named "docs" under your project directory.

4.Submit your UML design, stub code, and Javadoc as your initial design. (See below for due dates and requirements regarding submission of paper and electronic copies of project components.)

5.Implement the design you have developed by coding each method you have defined as well as any others you have left out of your design. As you do this, make sure to modify and annotate the changes to your design on your UML and properly document all new code. A good approach to the implementation of your project is to follow the project's milestones in the order they have been supplied.

6.Test your program and fix any bugs.

7.Once you have completed the project and are ready to submit it for grading, create a new set of Javadoc files using Eclipse and inspect them to make sure your detailed design is properly documented in your source code. (Actually, it is a good practice to keep your Javadocs up to date as you develop your software so that they can be of use to you and your team members and to the instructor and/or TA if you come to office hours for help.)

8. Submit all parts of your completed project. (See below for due dates and requirements regarding submission of paper and electronic copies of project components.)

Extra Credit Features:

You may extend this project with more search features for an extra 5 points of credit. For example, you could think of ways to enable a wider range of sort and display options, such as ways to find and display all awardees who won prizes in the same year.

To receive the full five points of extra credit, your extended features must be novel (unique) and must involve effort in the design of the extra features and their integration into the project and the actual coding of the features. Also, you must indicate on your final UML design which portions of the design support the extra feature(s); and you must include a write-up of the feature(s) in your milestones.txt file. The write-up must indicate what each feature is, how it works, how it is unique, and the write-up must cite any outside resources used.

Due Dates and Notes:

An electronic copy of your revised design including stub code and detailed Javadoc are due on **Wednesday, February 24th**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your revised UML design on *engineering paper* or a hardcopy using UML layout software at the **beginning of lab on Thursday, February 25th**.

An electronic copy of the final version of the project is due on **Wednesday, March 3rd**. Submit the project archive following the steps given in the submission instructions **by 9:00pm**. Submit your final UML design on *engineering paper* or a hardcopy using UML layout software, a hardcopy of the cover page for your project, and a hardcopy of the milestones.txt file at the **beginning of lab on Thursday, March 4th**.

You are not allowed to use the **StringTokenizer** class. Instead you must use `String.split()` and a regular expression that specifies the delimiters you wish to use to "tokenize" or split each line of the file.

You may write your program from scratch or may start from programs for which the source code is freely available on the web or through other sources (such as friends or student organizations). If you do not start from scratch, you must give a complete and accurate accounting of where all of your code came from and indicate which parts are original or changed, and which you got from which other source. Failure to give credit where credit is due is academic fraud and will be dealt with accordingly.

As noted in the syllabus, you are required to work on this programming assignment in a group of at least two people. It is your responsibility to find other group members and work with them. The group should turn in only one (1) hard copy and one (1) electronic copy of the assignment. Both the electronic and hard copies should contain the names and student ID numbers of **all** group members on the cover sheet. If your group composition changes during the course of working on this assignment (for example, a group of five splits into a group of two and a separate group of three), this must be clearly indicated in your cover sheet, including the names and student ID numbers of everyone involved and details of when the change occurred and who accomplished what before and after the change.

Each group member is required to contribute equally to each project, as far as is possible. You must thoroughly document which group members were involved in each part of the project. For example, if you have three functions in your program and one function was written by group member one, the second was written by group member two, and the third was written jointly and equally by group members three and four, your cover sheet must clearly indicate this division of labor. Giving improper credit to group members is academic misconduct and grounds for penalties in accordance with school policies.