Assignment 2
Avenger Favor (with Linked Lists)

Due Date: Monday Nov 2, before midnight
Weight: 7.5%
This assignment may be completed in groups of 2 to 3. *Each group must submit a statement of contribution that clearly states the contribution from each team member.*

Description

This assignment is a continuation of assignment 1. You are again to help Jon with data analysis on popularity of Avengers. The behaviour of your program should be identical to assignment 1, the only difference is that you cannot use an ArrayList to store your Avenger objects, nor can you use the Collections.sort. You must use an ordered singly linked list of your own implementation instead.

Your finished programs will be run like this:

```
cat input.txt | java -jar A2.jar > output1.txt
```

`input.txt` is any text file containing English text. Your program `A2` will read the standard input, and writes to standard output. The output from your program must follow the instructions given for assignment 1 and must be formatted *exactly* as the provided sample outputs.

Your program reads from the standard input one word at a time, cleans it up, and maintains a list of mentioned avengers and how many times their names or their aliases have occurred in the text. The roster of avengers' aliases and last names is the same as in assignment 1. You can use your own implementation, or the example solution posted for A1 as the starting point, and make sure you follow the implementation details below.

Implementation Details

- Create a class called `A2`. It will have the `main()` method, in which you should instantiate an `A2` object and then call run, and the `A2.run()` that does the bulk of the processing.
- Input is all from standard input. Create a `Scanner` object and use that for all input:

  ```
  Scanner input = new Scanner(System.in);
  ```

  All output should be to standard output. Use `System.out.print();` and `System.out.println();` for all output.
- You have an `Avenger` class from `A1`. It should have the implementation for the `Comparable` interface and overridden the `equals` method, and you should have two `Comparator` classes implemented for it according to the instructions for `A1`.
- **Creating your ordered Linked List:** You should create a linked list class called `SLL`. You can use the `Node` class given out in the tutorial as a starting point. If you create methods in `SLL` that match the ones in `ArrayList`, you can use much of your existing code with little editing.

  Your `Node` class uses a generic type for the data that it stores. However, it should not just be any old class. You want to be able to order your list, so you need a type that implements `Comparable`. That is done like this:

  ```
      public class Node<T extends Comparable<T>>
  ```

  for the generic `Node` class, and

  ```
      public class SLL<T extends Comparable<T>>
  ```

  for the generic `SLL` class. This will restrict `T` to be only classes that implement `Comparable`.

  Your SLL must implement all the required method for adding, finding, and removing items from a list. It should also implement a public `addInOrder` method (more details below).
- After you have created your `SLL` class, you should create your ordered lists. Remember that you are not allowed to use the `ArrayList`. Instead of the `ArrayList<Avenger>` in the previous assignment, you should create four ordered linked lists.
- Initially you should only worry about building a list of avengers in the order they appeared in the input stream. (Hint: consider creating the list by adding each new avenger to the tail)

- For this assignment, instead of sorting the list after the list is created as you did in assignment 1, you must create and maintain the lists in the order required. You need to implement the method `addInOrder()` for this purpose. Refer to the linked list tutorial for further details. The trick for this assignment is to make the `addInOrder` method work with different external `Comparator` objects. (Hint: Consider creating and using a private `compare` method and a private `Comparator` reference as members of your `SLL` class. If your SLL is constructed without any parameter, then you should initialize the internal `Comparator` object reference to null. Otherwise, you should initialize it to the external `Comparator` object passed as parameter to the constructor. The private `compare` method should then make the comparison based on the external `Comparator` if it exists, and based on the data item's internal `Comparable` implementation otherwise.)

- After creating the list of all avengers in the order they appeared in the input stream, I want you to traverse through this list, and create three other ordered lists: one list ordered based on the most popular avenger `Comparator`, another list ordered based on the least popular avenger `Comparator`, and finally a list ordered based on the alphabetical order of their alias (the ordering and tie breaking details are identical to assignment 1).

  Important Note: *You should only ever have one instance of each `Avenger` object, therefor, each `Avenger` object will be part of all four lists.*

- Use these four lists to print out your results as in Assignment 1.

  - The total number of words in the input. This total should not include words that are all digits or punctuation.
  - The total number of avengers mentioned in the input.
  - The list of avengers in the order they appeared in the input stream.
  - The four *most* frequently mentioned avengers ordered from most frequent to least. In case of ties, then in ascending alphabetical order of alias.
  - The four *least* frequent avengers ordered from least to most. In case of ties, then in ascending order of last name length, then again in case of ties, in ascending alphabetical order of last name.
  - All mentioned avengers ordered in alphabetical order of their alias (ascending order).

- **Testing and example files:** These are the same as for Assignment 1. There are four sample input and output pairs named `input1.txt`, `output1.txt` etc. Use these to determine if your program is running correctly. You should use the `diff` command to compare your output to the samples.

```
diff (cat sampleout.txt) (cat myout.txt)
```

If `diff` gives no output, your program is working correctly.

You should also devise a test plan and create a series of test files to fully exercise your program. For example, what about an empty file? or one with only some punctuation in it? Or a single word repeated 100 times? or ...?

I will be evaluating the program against files you have not seen yet that will be meant to *test* your code.

- This is a fairly short assignment. You really only need four classes, `A2`. `Avenger`, `Node` and `SLL` (plus two Comparators). Focus on writing compact, efficient code.

Documentation and Coding Standards

Your program should follow all of the coding rules and guidelines outlined in the document provided. In particular, include `Javadoc` style comments for all classes and substantial methods.

What to Hand In

Hand in a single file, `A2.jar`.

Submit this to the Blackboard drop box provided. The jar should be executable and contain:

1. All of your `.class` and `.java` files.
   *****Make sure you include your java source files, otherwise I cannot mark your code for documentation and style components.***
2. A class called `A2` which has a `main()` method.
3. A class called `Avenger` which implements the Comparable interface.

Grading

A detailed grading rubric is provided at the end of this document.

I will run your program with the command line given above on three text files and compare your output to the specifications.

Outcomes

Once you have completed this assignment you will have:

- Implemented and used an ordered linked list;
- Used the Java Comparable interface;
- Used a Comparator object;
- Used standard input and standard output re-direction.

Rubric

1. Documentation
   (a) Java doc standards 5
   (b) Format of Code 5
   (c) Meets other rules/guidelines 5
2. Testing
   (a) Test file1 10
   (b) Test file2 10
   (c) Test file3 10
3. Follows implementation specifications 30
   - Has the following classes with the required functionality: A2, Avenger, Node, and SLL. (4)
   - Implements and uses a Comparable generic Singly Linked List class correctly. (7)
   - Implements and uses addInOrder() correctly. (7)
   - Constructs four ordered linked list objects correctly. (8)
   - Code is compact and efficient. (4)