Concordia University
Comp 249 – Fall 2018
Object-Oriented Programming II
Assignment 4 – Due 10:00 <u>AM</u> - Saturday December 1, 2018
<u>Important Note:</u>
Since the demos will be held between the 1st and 5th of December, no late assignment submissions will be accepted after 10:00 AM (morning) of December 1st .

Purpose: The purpose of this assignment is to allow you practice File I/O, ArrayList, and Linked Lists.

Part 1: ArrayList & Other subjects

In 1935, the American linguist George Zipf noticed something very peculiar with the books he was reading. Whenever he would count the words in his books, he noticed that most of the words appeared only once and a small number of words appeared very frequently. In addition, this phenomenon seemed to hold true for any text and in any language. Zipf analysed this a little further, and noticed that if you sort the words by frequency and plot the frequency of words versus their rank in the sorted list, you will always get a graph similar to the one below, where a few words have a high frequency (the peak on the left), and most words appear only once (the long tail on the right).
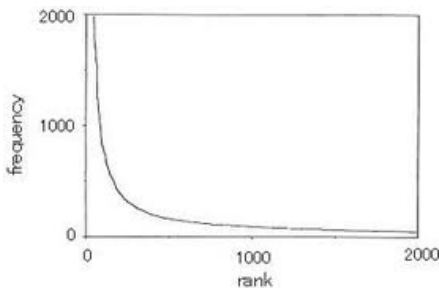
In technical terms:
- Words that appear frequently (the peak on the left) and that happen to be short are called *stop-words*.
- Words that only appear once (the tail on the right) are called *happax legomena*.

Today, this behaviour of language is known as Zipf's law and is used tremendously in the Search Engine industry. In fact, it turns out that Zipf's law is also applicable to many other domains and applications such as monitoring traffic (on the Internet or on the highway), predicting financial activities, and even analysing animal behaviour...

Write a program to verify Zipf's law with your favourite text. Specifically, your program must ask the user for an input text file (please handle potential I/O errors properly), and count how many words the file contains and display the following statistics:

- Display each word in the text along with its rank and frequency.
  You can assume that a word is defined as anything that the method `Scanner.next()` returns and only contains letters. For example "U2", "data-base" and "hi!" should not be counted as words.
  Your program does not have to be case-sensitive. For example, the words "hi" and "Hi" can be considered as same words.
  The list of rank/word/frequency must be displayed in descending order of frequency, and all words with the same frequency must be displayed in alphabetical order (uppercases before lowercases).
- Display the total number of word tokens and word types.
  The number of word tokens refers to the total number of words in the text (the number of times the method `Scanner.next()` returned a `String`); whereas the number of word types refers to the number of different words in the text. For example, if the word "the" appears 30 times, it will count as 30 word tokens, but only 1 word type.
- Display statistics on the *happax legomena*:
  - the number of happax,
  - the percentage of happax (nb happax ÷ nb of word types), and
  - how many of the tokens in the text they account for (nb happax ÷ nb of word tokens).
- Display statistics on the stops words:
  - the number of stop words,
  - the percentage of stop words (nb stop words ÷ nb of word types), and

- how many of the tokens in the text they account for (total frequency of stop stops ÷ nb of word tokens).

For this assignment, you can assume that a stop word is a word that has a length of 4 characters or less and that appears at least 10 times in the text.

Here is an example of how your program should behave (the file jokes.txt is available with this assignment):

```
Enter input file:   jokes.txt


-----------------------------------
RANK           FREQ          WORD
-----------------------------------
1              17            a
2              10            A
3              7             in
4              7             was
5              5             the
6              5             you
7              4             The
8              4             When
9              4             are
10             4             is
11             4             of
12             4             who
13             4             your
14             3             and
...
34             1             Bakers
35             1             Count
36             1             Did
...
51             1             about
52             1             all
53             1             backward
54             1             baseball
55             1             batteries
...
174            1             why
175            1             will
176            1             wondered
177            1             write
178            1             writes


-----------------------------------


Nb of word tokens: 268
Nb of word types: 178

Nb of Happax: 145
% of Happax: 81%
Happax account for: 54% of the text

Nb of stop words: 6
% of stop words: 3%
Stop words account for: 19% of the text
```

Part 2: Linked Lists

Write a program to manipulate Cellular Phones using linked lists.

I) The **CellPhone** class has the following attributes: a serialNum (long type), a brand (String type), a year (int type, which indicates manufacturing year) and a price (double type). It is assumed that brand name is always recorded as a single word (i.e. Motorola, SonyEricsson, Panasonic, etc.). It is also assumed that all cellular phones follow one system of assigning serial numbers, regardless of their different brands, so no two cell phones may have the same serial number.

You are required to write the implementation of the CellPhone class. Beside the usual mutator and accessor methods (i.e. getPrice(), setYear()) the class must have the following:

(a) Parameterized constructor that accepts four values and initializes *serialNum*, *brand*, *year* and *price* to these passed values;

(b) Copy constructor, which takes two parameters, a CellPhone object and a long value. The newly created object will be assigned all the attributes of the passed object, with the exception of the serial number. *serialNum* is assigned the value passed as the second parameter to the constructor. It is always assumed that this value will correspond to the unique serial number rule;

(c) clone() method. This method will prompt the user to enter a new serial number, then creates and returns a clone of the calling object with the exception of the serial number, which is assigned the value entered by the user;

(d) Additionally, the class should have a toString() and an equals() methods. Two cell phones are equal if they have the same attributes, with the exception of the serial number, which could be different.

II) The file **Cell_Info.txt**, which one of its versions is provided with this assignment, has the information of various cell phone objects. The file may have zero or more records. The information stored in this file is always assumed to be correct and following the unique serial number rule. A snapshot of the contents of the Cell_info.txt file is shown in Figure 1 below.

Figure 1: Cell_info.txt

III) The **CellList** class has the following:

(a) An inner class called CellNode. This class has the following:

    i.   Two private attributes: an object of CellPhone and a pointer to a CellNode object;

    ii. A default constructor, which assigns both attributes to null;

iii. A parameterized constructor that accepts two parameters, a `CellPhone` object and a `CellNode` object, then initializes the attributes accordingly;

iv. A copy constructor;

v. A `clone()` method;

vi. Other mutator and accessor methods.

(b) A private attribute called `head`, which should point to the first node in this list object;

(c) A private attribute called `size`, which always indicates the current size of the list (how many nodes are in the list);

(d) A default constructor, which creates an empty list;

(e) A copy constructor, which accepts a `CellList` object and creates a copy of it;

(f) A method called `addToStart()`, which accepts one parameter, an object from `CellPhone` class. The method then creates a node with that passed object and inserts this node at the head of the list;

(g) A method called `insertAtIndex()`, which accepts two parameters, an object from `CellPhone` class, and an integer representing an index. If the index is not valid (a valid index must have a value between `0` and `size-1`), the method must throw a `NoSuchElementException` and terminate the program. If the index is valid, then the method creates a node with the passed `CellPhone` object and inserts this node at the given index. The method must properly handle all special cases;

(h) A method called `deleteFromIndex()`, which accepts one integer parameter representing an index. Again, if the index is not valid, the method must throw a `NoSuchElementException` and terminate the program. Otherwise; the node pointed by that index is deleted from the list. The method must properly handle all special cases;

(i) A method called `deleteFromStart()`, which deletes the first node in the list (the one pointed by `head`). All special cases must be properly handled.

(j) A method called `replaceAtIndex()`, which accepts two parameters, an object from `CellPhone` class, and an integer representing an index. If the index is not valid, the method simply returns; otherwise the object in the node at the passed index is to be replaced by the passed object;

(k) A method called `find()`, which accepts one parameter of type long representing a serial number. The method then searches the list for a node with a cell phone with that serial number. If such an object is found, then the method returns a pointer to that node where the object is found; otherwise, the method returns `null`. The method must keep track of how many iterations were made before the search finally finds the phone or concludes that it is not in the list;

(l) A method called `contains()`, which accepts one parameter of type long representing a serial number. The method returns `true` if the an object with that serial number is in the list; otherwise, the method returns `false`;

(m) A method called `showContents()`, which displays the contents of the list, in a similar fashion to what is shown in Figure 2 below.

(n) A method called `equals()`, which accepts one parameter of type `CellList`. The method returns `true` if the two lists contain similar objects; otherwise the method returns `false`. Recall that two `CellPhone` objects are equal if they have the same values with the exception of the serial number, which can, and actually is expected to be, different.

```
eclipse-workspace - CellListUtilization/src/CellListUtilization.java - Eclipse
File  Edit  Source  Refactor  Navigate  Search  Project  Run  Window  Help

Console ✕
<terminated> CellListUtilization [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 12, 2018, 11:44:43 PM)

The current size of the list is 23. Here are the contents of the list
=================================================================
[1119000: SonyEricsson 347.94$ 2009] ---> [2389076: BlackBerry 564.22$ 2010] ---> [5909887: Apple 726.99$ 2017] --->
[9873330: Nokia 677.9$ 2010] ---> [5555902: SonyEricsson 177.11$ 2007] ---> [8006832: Motorola 423.22$ 2008] --->
[6987612: HTC 577.25$ 2009] ---> [2999900: Siemens 457.28$ 2006] ---> [7333403: BenQ 659.0$ 2009] --->
[5890779: Motorola 457.28$ 2007] ---> [8888902: Samsung 810.35$ 2018] ---> [5000882: Apple 977.27$ 2016] --->
[1119002: Motorola 457.28$ 2008] ---> [9675654: Nokia 388.0$ 2009] ---> [6699001: Lenovo 237.29$ 2012] --->
[2887460: Siemens 457.28$ 2009] ---> [7559090: Pansonic 290.9$ 2005] ---> [2887685: Motorola 569.28$ 2012] --->
[89076: Sharp 564.22$ 2009] ---> [1989000: Nokia 237.24$ 2006] ---> [4900088: LG 232.99$ 2017] --->
[2787985: Acer 572.2$ 2013] ---> [3890909: Samsung 857.28$ 2015] ---> X
```

Figure 2: Sample of Displaying the Contents of a CellList

➔ Finally, here are some general rules that you must consider when implementing the above methods:

- Whenever a node is added or deleted, the list size must be adjusted accordingly;

- All special cases must be handled, whether or not the method description explicitly states that;

- All `clone()` and copy constructors must perform a deep copy; no shallow copies are allowed;

- If any of your methods allows a privacy leak, you must clearly place a comment at the beginning of the method 1) indicating that this method may result in a privacy leak 2) explaining the reason behind the privacy leak. Please keep in mind that you are not required to implement these proposals;

IV) Now, you are required to write a public class called **CellListUtilization**. In the `main()` method, you must do the following:

(a) Create at least two empty lists from the `CellList` class;

(b) Open the `Cell_Info.txt` file, and read its contents line by line. Use these records to initialize one of the `CellList` objects you created above. You can simply use the `addToStart()` method to insert the read objects into the list. However, the list should not have any duplicate records, so if the input file has duplicate entries, which is the case in the file provided with the assignment for instance, your code must handle this case so thast each record is inserted in the list only once;

(c) Show the contents of the list you just initialized;

(d) Prompt the user to enter a few serial numbers and search the list that you created from the input file for these values. Make sure to display the number of iterations performed;

(e) Following that, you must create enough objects to test each of the constructors/methods of your classes. The details of this part are left as open to you. You can do whatever you wish as long as your methods are being tested including some of the special cases.

General Guidelines When Writing Programs:
- Include the following comments at the top of your source codes
  ```
  // -----------------------------------------------------
  // Assignment (include number)
  // Question: (include question/part number, if applicable)
  // Written by: (include your name and student id)
  // -----------------------------------------------------
  ```
- In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
- Include comments in your program describing the main steps in your program.
- Display a welcome message which includes your name(s).

- Display clear prompts for users when you are expecting the user to enter data from the keyboard.

- All output should be displayed with clear messages and in an easy to read format.
- End your program with a closing message so that the user knows that the program has terminated.

JavaDoc Documentation:
Documentation for your program must be written in **javaDoc**.
In addition, the following information must appear at the top of each file:

```
Name(s) and ID(s)      (include full names and IDs)
COMP249
Assignment #           (include the assignment number)
Due Date               (include the due date for this assignment)
```

Submitting Assignment 3

- For this assignment, you are allowed to work individually, or in a group of a maximum of 2 students (i.e. you and one other student). You and your teammate must however be in the same section of the course. Groups of more than 2 students = zero mark for all members!
- Only electronic submissions will be accepted. Zip together the source codes.
- Students will have to submit their assignments (one copy per group) using the Moodle/EAS system (please check for your section submission). Assignments must be submitted in the right DropBox/folder of the assignments. **Assignments uploaded to an incorrect DropBox/folder will not be marked and result in a zero mark. No resubmissions will be allowed. Your instructor will indicate whether you should upload your assignment to EAS (**https://fis.encs.concordia.ca/eas/**) or to Moodle.**
- Naming convention for zip file: Create one zip file, containing all source files and produced documentations for your assignment using the following naming convention:
    The zip file should be called *a#_StudentName_StudentID*, where # is the number of the assignment and *StudentName/StudentID* is your name and ID number respectively. Use your "official" name only - no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student 12345678 would submit a zip file named like: *a1_Mike-Simon_123456.zip*. if working in a group, the name should look like: *a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip*.
- Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded and all others will be disregarded.
- If working in a team, only one of the members can upload the assignment. Do **NOT** upload the file for each of the members!

⇨ Important: Following your submission, a demo is required (please refer to the courser outline for full details). The marker will inform you about the demo times. Please notice that failing to demo your assignment will result in zero mark regardless of your submission.

Evaluation Criteria for Assignment 3 (10 points)

Part 1 (4 points)	
Correct Implementation	2 pts
Proper Use & Testing of the Code	2 pts
Part 2 (6 points)	
Design and Corretness of Classes	3.5 pts
Proper and Sufficient Testing of Your Methods	2 pts
Privay Leak Comments and Proposals to Avoid Them	0.5 pts