CSCI 235 Spring 2020

Lab 11 **Apr. 21**

Connecting to a VNC display

You will use a lab account for a VNC display. The lab account already started a VNC session, so you don't need to do Step 3 or 4 of Lab 7's Task #1. Two steps that you need to do are:

1. Create a tunnel from your machine to a VNC display (Step 5 of Lab 7's Task #1).
    a. A cslab machine number and a display number will be provided to you.
    b. For example, if the given machine is **cslab10**, and the display number is **12**, the command for creating a tunnel to the display will be:
        - Mac Terminal:
          `ssh -L 5901:localhost:5912 your_loginID@cslab10.wheaton.edu`
        - Windows PuTTY:
          Host Name: `cslab10.wheaton.edu`
          For Connection, SSH, Tunnels: `5901` for the "Source port" and `localhost:5912` for the "Destination".
    c. The CS system will prompt for your (ID and) password as usual.

2. Start a VNC viewer on your machine (Step 6 of Lab 7's Task #1).
    a. The given display's VNC password will be provided. You need to use this password to connect to the display.

Tips:

- If you don't see the menu bar at the top of the CS desktop window, right-click the desktop screen to get Applications menu.
- When you are done with the desktop, close the desktop and connection by doing:
    - For TightVNC, use the X button at the top corner of the window.
    - For Mac, in Screen Sharing menu at the top, select Connection, and select Close.

===

Java Collections

Documentation and styling of your code will be part of your grade.

Create a new directory and cd to it. Then copy starting files from the lab directory to your current directory.

```
$ cp /cslab/class/csci235/labs/lab11/* .
```

You should have two Java files to work on, one class file, and three files that you can use as input for testing.

You will write a program that takes a piece of text and counts the frequency of the words in it. For example, in this sentence, the word *times* occurs 5 times, *occurs* also occurs 5 times, *and* occurs 3 times, and *for*, *example*, *in*, *this*, *sentence*, *the*, *word*, *each* and *also* each occurs 2 times.

Your program will read the text on which to do the word counting from a file. You have been provided with the component that reads in from a file. You will run the program as follows:
`$ java FileProcessor filename` (for example, `$ java FileProcessor Ps132`)

The program reads in a piece of text stored in the file and prints the results to the screen. You will complete two Java files (`FileProcessor.java` and `SIPair.java`) through the small steps below. **Refer to the CollectionAPI document for necessary collection methods.**

Step #1: Open `FileProcessor.java`, and inspect the code. The main method calls `FileAux.filesToWords()` from the provided class file, which takes the name of a file, opens and reads that file, and returns an `ArrayList<String>` of the words it contains (ignoring punctuation, spaces, etc.).

Compile the file, and run it with one of the provided files as input. Which statement generates the output? What method has been used to print the output?

Step #2: Now delete the line that prints out the `ArrayList` object. Uncomment the code block (enclosed by /* and */) for Step 2, and fill in the loop so that it adjusts the `HashMap` to account for the current word. Make the word lowercase: we do not want to consider capitalization in the word counting. The loop will read every word in the `ArrayList` object and count its occurrences.

- See how Iterator has been used for the ArrayList object, `words`.
- `it.next()` will return a word of the ArrayList because it has been defined on `words`.

Then, the pair of word (as Key) and occurrences (as Value) will be inserted into the `HashMap`, *tally*. In other words, you need to write the code for the following steps by using appropriate instance methods:

- Get a word from the iterator, and make the word lowercase (use toLowerCase()).
- Check if the word is already in the map, *tally*.
    - If it is in the map,
        - get the word's current number of occurrences, and increase the number by 1.
        - Put this updated pair back to the map.
    - Else, store a new pair of the word and the occurrence (namely, 1) to the map.

Compile and test the program.

Step #3: Delete the line that prints the `HashMap`, `tally` (which is `System.out.println(tally);` at line 38). You will write the results, one word per line. Uncomment the code block for Step 3 (don't uncomment yet the statement, `ArrayList<SIPair> pairs…`), and fill in the `for` loop. The loop should print out a word followed by its frequency, one word per line. Find the statement that declares the iterator within the `for` loop, and answer to the questions:

- Does `HashMap` provide `iterator()` method?
- What will be returned by `keySet()`?
    - This method returns a set of Key values in the map. For example, assume that a HashMap object, `myMap` contains three pairs as follows:
      {apple=3, orange=1, banana=5}
      `myMap.keySet()` will return the set of the keys: [apple, orange, banana]
- What does the following statement do?
  `Iterator<String> it = tally.keySet().iteractor()`
    - Since `keySet()` returns the keys of `tally`, the iterator object, `it` can be used to get each key of `tally`.
- What will be then stored at `s` from the following statement?
  `String s = it.next();`

Write the body of the loop. Refer to the CollectionAPI document, and use appropriate HashMap methods. Compile and test the program.

The frequency information would be much more useful if it were in an order. Moreover, since we are more likely to be interested in knowing what the most frequent words are, we would like it *backwards sorted*. For example, for the file, *Ps132,*

```
will: 13
for: 12
i: 12
the: 12
your: 10
and: 9
to: 8
my: 7
a: 6
lord: 6
of: 6
ever: 5
his: 5
one: 5
…
```

In order to do this, you will store all the word/frequency pairs in "an `ArrayList` of pairs" and sort them. Then, you will first need a Java class that represents those pairs.

Step #4: Open the file `SIPair.java`. This is for a class which represents a pair containing a String object and an int (see the instance variables).
Fix the `toString()` method so that it returns the string followed by a colon and space, then the number (as seen in the figure).

Back in `FileProcessor.java`, uncomment the line 45 that creates an `ArrayList` of these pairs (`SIPair`). Also, uncomment the line further down that prints `pairs`.

- See the types of `ArrayList`. `pairs` will hold objects of `SIPair`.
- On the other hand, `words` is an `ArrayList` object that holds String objects.

Now comment out the body of the loop you wrote in Step 3, and write new code for the body so that instead of printing each pair of String and int, it makes a new `SIPair` object out of them and adds that object to the `ArrayList` object, `pairs`.

- Namely, instead of using the `HashMap` object (`tally`), you are creating `SIPair` objects and storing them to the `ArrayList` object, `pairs`.

Make sure to understand how `SIPair` objects are created and stored in `pairs`. Compile and run the program. The output should be now in the default format of printing the contents of `ArrayList`.

- Which statement prints the output?

Step #5: Java comes with the `Collections.sort()` method in the class `Collections`, which we can use to order our list. This method will operate on an `ArrayList` object as long as the type of element stored in the object implements an interface, `Comparable`. This interface has a single method, `compareTo()` which allows us to compare two items of the same type so that the `sort()` method knows what order to put them in.

Open `SIPair.java` which implements the `Comparable` interface (see the class heading), and fill in the `compareTo()` method. This method is defined as returning zero if the two objects are equal, a negative value if the calling object (`this`) is "less than" the parameter `other`, and a positive value if the calling object is "greater than" `other`. Because we are sorting, "less than" means "belonging before." We want *higher* frequency first; so we should get a negative value if the frequency of `this` is larger. If two words have the same frequency, we break the tie by putting them in alphabetical order based on the words themselves (for example, "ever", "his", and "one" in the above figure), which we can get by **calling `compareTo()` method on String object**.

Write the body of `compareTo`. If necessary, write a simple main method in `SIPair.java` to make a few pairs and see the results of comparing them. If your `compareTo` works, move to the next step.

Step #6: Back in `FileProcessor.java`, uncomment the line 57 that calls `Collections.sort(pairs)`. This method should work now because `SIPair` (the type of `pairs`) has the appropriate code in `compareTo()`. Compile and test the program.

Step #7: Replace the statement that prints `pairs` (`System.out.println (pairs)`) with a loop that will write each element in `pairs` on a line by itself. Make use of the `toString()` method you wrote in Step 4. Then, the program should print the output similar to the one in the figure.

Compile and test everything with provided testing files, i.e., *Ps132*.

Make sure to understand how the Collection classes and SIPair work together to produce the output.

Important: After you email your files to the TA, delete your lab folder for the next team. Your display will be available during the scheduled lab hours only.

What to submit

- Email `SIPair.java` and `FileProcessor.java` to your TA:
  - 8:30 am session: to Drew at drew.smith@my.wheaton.edu
  - 1:15 pm session: to Brian at brian.drown@my.wheaton.edu
  - **In the subject line**, put "CSCI 235 Lab 11 (*the first initial of your first name and your last name*; *the first initial of the partner's first name and the last name*)"
    - For example, CSCI 235 Lab 11 (HKim; JSmith)