CISC 181 Spring 2014
Practice Set 5
Assigned: April 14
Due: April 20 at 11:55PM on Sakai

Practice sets are to be completed individually. You are free to consult other students to help complete the practice sets (see syllabus for collaboration policy). However, keep in mind that each practice set is designed to cover basic material on which you will be quizzed and tested.

This practice set is intended to cover the following major topics:

- Transforming recursive code using a custom linked list into loops over Lists
- Using the List, Set, and Map interfaces for built-in data structures
- Using built-in methods in the java.util.Collections class
- [Optional] Use Scanner with Files for simple input/output
- [Optional] Exceptions

Download the PS5.zip from Sakai and import it to Eclipse using File->Import, Existing Projects into Workspace, Select archive file, Browse and find your downloaded PS5.zip.

Please consult chapters 9, 22, and 23 of your textbook, your class notes, and the online Java documentation linked in Part B for additional examples that will help to solve these problems. Each part of this assignment comes with tests. A good way to start is to read and understand the test cases.

PART A: TRANSFORMING A CUSTOM LINKED LIST INTO LOOPS OVER A LIST

[10 points each] Complete each of the four methods in Pipeline.java using the list property and the PipelineSegment class. Your code should use loops to solve each problem except replaceEnd. The logic itself is the same as the Pipeline problem you worked on earlier in the semester in Practice Set 3.

PART B: USING LISTS, SETS, AND MAPS

You likely want to review the methods and documentation available for each of these core interfaces of the Java Collections Framework:

http://docs.oracle.com/javase/6/docs/api/java/util/List.html
http://docs.oracle.com/javase/6/docs/api/java/util/Set.html
http://docs.oracle.com/javase/6/docs/api/java/util/Map.html

[10 points each] Complete each of the methods in CollectionExercises.java and make sure that the tests in TestCollectionExercises pass. Notes:

- The only method that should mutate the parameters is removeLessThanZero. All other methods should return appropriate new Collection structures.
- Three of the methods (union, intersection, and difference) do not require loops.
- The reverseMapping method is harder to implement than the others.

- The removeLessThanZero method will require you to use an iterator from the List, see:

http://docs.oracle.com/javase/6/docs/api/java/util/List.html#iterator()
http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html

---

PART C [OPTIONAL]: PARSING AND COMPARING SHAKESPEAREAN PLAYS

Completing any or all of this part will count as additional credit towards your practice set grade. You may earn up to 100 points additional credit towards the Practice Set only portion of your course grade (which is 12% of the final course grade). Earning credit past the 12% will not count.

We would like to use our programming skills with Collections to do frequency analysis of literary texts. Before starting this section you will want to look at the short examples and documentation available for a Scanner in your textbook and at:
http://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html

Other than the parse method, none of the other methods require you to write a loop (or other form of repetition). Instead you must find and use appropriate methods in the List, Set, Map, or the Collections class. This will significantly reduce the size of your code and the amount of effort needed to solve each method.

1) Complete the Parser class that will read a given file and keep track of the number of times each word has appeared in the file. This class should have the following methods:
   - **[20 points]** void parse(String filename) throws IOException
     - parses the contents of the file given by filename using a Scanner object
       - note that doing this means that you must handle an IOException in your code. Handle this by allowing your parse method to throw the exception. The tests will catch this exception and call the fail() method (this method is similar to assertEquals but indicates a test failure).
     - updates a Map of Strings to Integers that contains the count for each word found in the file -- the Map must be an object property so that this method can be called multiple times for different files
     - must use the replace method of String to replace all occurrences of punctuation characters with empty Strings (""). Hint: you should use the String's replaceAll method (which produces a new String because Strings are immutable) by giving it this regular expression: "\\p{Punct}" and then the empty String "". See the Java documentation for more details: http://docs.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html
     - **make sure to convert the words to lowercase before updating the Map**
     - **NOTE: it is a valid use case that the parse method is called multiple times on the same Parser and given different file inputs. When this happens the Map of frequencies should *accumulate* all of the occurrences.**
   - **[10 points]** int getCount(String word)
     - gets the count for a given word from the Map property
   - **[20 points]** List<String> getWordsInOrderOfFrequency()

- returns a List of words in order of the frequency (least to greatest) they appear in all of the files that were parsed (use the Map property).
- Hint: you should use the Collections.sort method and have your Parser class implement the Comparator<String> interface (not the Comparable interface!) using your getCount method to compare the counts of two Strings.

--- The TestParser class will test each of the above methods.

2) Add the following methods to your Parser class from question 1. Each of these should take another Parser and return appropriate types of data (consult the tests for method signatures):
   a) **[10 points]** What is the Set of all words that are in common between this Parser and the other Parser?
   b) **[20 points]**What is the total number of words that appear in both top 100 most frequent words lists?

-- Your code must pass the tests in TestParser for the two included Shakespeare plays.

3) **[20 points]** Add a main method to the Parser class that will create Parsers for each of the five Shakespearean plays in the PS5 project. Using the method from question 2b, find which two plays have the most top 100 words in common. **Write a comment in your main method that has your name and the two plays.**

For example comparing Julius Caesar and Romeo and Juliet should find 68 top 100 words in common and comparing Hamlet to Romeo and Juliet gets 72 top 100 words in common. Neither of these is the answer, but you can see that both of these values are used in test_commonWords.

Submit your PS5 project to Sakai by exporting it from Eclipse as an archive.