Concurrent Programming

Lab 9

This exercise is about processing a large body of English words, using streams of strings. There is
a list of some 235,000 English words and names in file /usr/share/dict/words on most Unix systems, including MacOS. If you use Windows or do not have the file for some other reason, get it from the course homepage at Words. The exercises below should be solved without any explicit loops (or recursion) as far as possible.

1. Starting from the TestWordStream.java file, complete the readWords method and check that you can read the file as a stream and count the number of English words in it. For the words file on the course homepage the result should be 235,886.
2. Write a stream pipeline to print the first 100 words from the file.
3. Write a stream pipeline to find and print all words that have at least 22 letters.
4. Write a stream pipeline to find and print some word that has at least 22 letters.
5. Write a method boolean isPalindrome(String s) that tests whether a word s is a palindrome: a word that is the same spelled forward and backward. Write a stream pipeline to find all palindromes and print them.
6. Make a parallel version of the palindrome-printing stream pipeline. It is possible to observe whether it is faster or slower than the sequential one?
7. Write a stream pipeline that turns the stream of words into a stream of their lengths, then finds and prints the minimal, maximal and average word lengths.
8. Write a stream pipeline, using method collect and a groupingBy collector from class Collectors, to group the words by length. That is, put all 1-letter words in one group, all 2-letter words in another group, and so on, and print the groups. Optional challenge, easily answered by Java Precisely: Use another overload of groupingBy to compute (and then print) the number of 1-letter words, the number of 2-letter words, and so on.
9. Write a method Map<Character,Integer> letters(String s) that returns a tree map indicating how many times each letter is used in the word s. Convert all letters to lower case. For instance, if s is the word "Persistent" then the tree map will be {e=2, i=1, n=1, p=1, r=1, s=2, t=2}. Now write a stream pipeline that transforms all the English words into the corresponding tree map of letter
counts, and print this for the first 100 words.
10. Use the tree map stream and the reduce method to count the total number of times the letter e is used in the English words. For the words file on the course homepage the result should be 235,886. Words s1 and s2 that have the same tree map of letter counts (by letters(s1).equals(letters(s2)) are anagrams: they use the same letters the same number of times. For instance, "persistent" and "prettiness" are anagrams; both have letter counts {e=2, i=1, n=1, p=1, r=1, s=2, t=2}. Use the collect method on the word stream, groupingBy collector and the letters method to find and print all sets of anagrams in the file of English words. This may take 15–30 seconds to compute. For the words file on the course homepage the result should be 15,287 sets of anagrams, including {a=1, c=1, e=1, r=1}=[Acer, acre, care, crea, race].
11. Try to make a parallel version of the anagram-printing stream pipeline. Is it faster or slower than the sequential one? (If your computer and Java version behaves like mine, this example shows that just slapping on .parallel() does not necessarily lead to more efficient execution).