PA1 Word Search

Due: Wednesday 1/26 by 11:30PM
Submission: Submit only WordSearch.java to Gradescope. Do not submit the entire project.

Description

Since words were etched in stone, people have loved to do word searches. The game is to search for words in a grid of letters. As you may know, it can be difficult to find every single word in the grid though. So let's write a program to do just that!

Your task is to read in a file representing a grid of characters and a file representing a dictionary of valid words. You will then search through the grid of letters to find all the valid words in the grid.

Important details to note:

- Our program will only search in the vertical and horizontal directions. Do **not** search for words in the diagonal direction.
- Our grids can be any size. The file specifying the grid will have a number as its first line which represents the number of rows, and another number as the second line representing the number of columns. See the example file below. You are guaranteed that both the number of rows and the number of columns will always $\geq$ to 4.
- If a word exists in a grid multiple times, print it out each time you find it.

A word is a valid word if

- It is at least three letters long.
- It can be formed from letters in the grid that are adjacent along a row (left-to-right or right-to-left) or along a column (top-to-bottom or bottom-to-top). Note we are not worrying about diagonal words in this assignment.
- It can be found in the dictionary file.

Expected Behavior

- Read in the dictionary file and the file specifying the grid of letters. **The names of these files will be in command-line arguments 0 and 1 respectively.** The dictionary file name

will be in args[0] and the filename specifying the grid will be in args[1]. For examples discussing command-line arguments see the class examples on github here.

- Read the dictionary file into a Java collection. You will use this collection to check if a word found in the grid is a valid word or not. What data structure makes sense here?
- Read the grid file to create a grid of strings. You know exactly how many rows and columns your grid has before creating it. What data structure makes sense here? Remember that Java collections can store other collections. So you can have a list of lists or a set of sets or an array of arrays for instance.
- Search your grid to find valid words. Matching words from the grid to the dictionary file should be done case-insensitive. i.e. if there exists a word Abe in the dictionary, a sequence of letters in the grid "a", "b", "e" should match that word.
- Collect the legal words found into a list and print them out as indicated below. **The order in which you print them matters.**

Examples

The following is an example of the grid of letters file:

```
6
6
y c o d e j
h s e y p k
l p h b w a
l o b w x z
w o b a a i
p l y y c g
```

In this example, you know you have 6 rows (from the first line) and 6 columns (from the second line). You would then read in the grid. Note that in general the input grid could have a mix of uppercase or lowercase letters. For this grid, and the dictionary.txt included in the starter code, your program would find the below words:

- cod, code, ode, ply (horizontal, L-to-R)
- jed, doc, yes, abo, bow (horizontal, R-to-L)
- spool, pool, way (vertical, top-to-bottom)
- loop, loops (vertical, bottom-to-top)

View the dictionary.txt file in the starter code to see an example of this type of file. Note your program should not print the '(horizontal, L-to-R)' part. It should only print the words one per line **in the same order as above though**. So the expected output for the above example would be:

```
cod
code
ode
ply
jed
doc
yes
abo
bow
spool
pool
way
loop
loops
```

Words are always output in all lowercase as shown above no matter what.

Input Files

We have provided sample input files for you to test your program in the starter code. **It is important to note though that you should test your program on lots of other files.** We certainly will.

Tips and Hints

You will certainly need to think carefully about the strategy for checking for words in a row or column of the grid. And this strategy must output words in the same order that we expect them. Start by thinking about searching from left to right. Consider the first row:

```
y c o d e j
```

Notice that this row contains the words cod, code, and ode. Suppose that the row is represented as the array [”y”, ”c”, ”o”, ”d”, ”e”, ”j”]. A simple way to explore all the possible words (going L to R) in this list would be as follows (the process for the other rows is similar).

- Starting at the first element (i.e., ”y”), check whether the sequence of length 3 starting at that position is a legal word (we start with length 3 because a legal word has to be at least three letters long). Then check for length 4, then for length 5, etc., until you reach the end of the list.
- Now repeat this step, but starting at the second element (i.e., ”c”). Notice that this time you will come to the end of the list one step sooner. Then repeat for the third element, and so on.

Note that the above process can be made easier by first converting the array of individual letters into one concatenated string. This could also lead to some great reuse of code. Then

for all different types of searches you would produce a string representing that sequence of characters that you want to search and you can use one function to check a sequence for any valid words.

Everyone solves problems differently, but I thought it would be helpful to highlight a few functions I used in case you find them helpful as well.

- str.toLowerCase()
- Integer.valueOf(...). When reading in the number of rows and columns, I recommend using Integer.valueOf(scanner.nextLine()) instead of trying to use scanner.nextInt().
- str.split(...)
- str.charAt(...)
- str.substring(... , ...)