CSC 115: Fundamentals of Programming II
Assignment #3: Recursion and backtracking

Due date

Wednesday, March 7th, 2018 at 9:00 am via submission to conneX.

How to hand in your work

Submit files through the Assignment #3 link at the CSC 115 conneX site. Please make sure you follow all the required steps (including confirming your submission).

Learning outcomes

We will step aside for one assignment from our intensive focus on ADTs. When you have completed this assignment, you will more familiar with:

- How to a write a non-trivial *recursive method*.
- How to use a recursive method *to implement a backtracking algorithm*.

Word game

Problem description

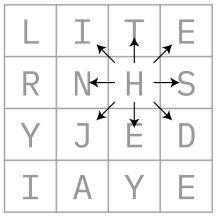
The world is full of word games, many of them involving a word search on a gameboard. One such involves a four-by-four random arrangement of English letters and finding words in it. Although we will not implement the full game, instead we concentrate on perhaps the trickiest part: *verifying a given word is on the gameboard*.

Consider diagram below (and ignore for now how it was created):

rows \ columns	0	1	2	3
0	L	I	T	E
1	R	N	H	S
2	Y	J	E	D
3	I	A	Y	E

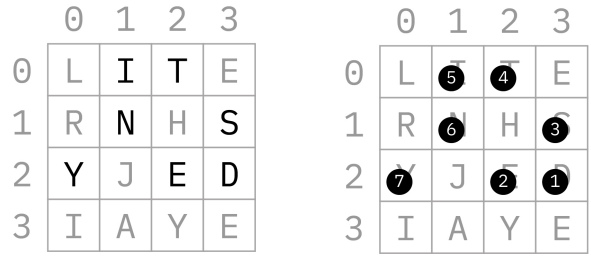
Individual letters on this board may be identified by their row-and-column numbers. For example, the uppercase letter 'H' is at the row 1 and column 2, i.e. 'H' is at(1, 2).

Using this same notation, 'L' is at (0, 0), 'D' is at (2, 3), the two 'I's are at (0, 1) and (3, 0), etc. There are also eight letters adjacent to the lettersquare containing 'H':



Notice "adjacent" with respect to a letter also means squares lying on a diagonal in addition to those above, below, to the right, and to the left. The eight neighbours of 'H' are (clockwise from 12 o'clock) 'T', 'E', 'S', 'D', 'E', 'J', 'N', and 'I'.

There are many words on this board. One of them is "DESTINY". (Another is "LINTSEED"; how long do you need to find it?!) The diagram below shows the letters in "DESTINY" alongside the actual sequence:



There is an imaginary line connecting the letters in the sequence. The line is also permitted to cross over itself (although this does not actually in our example). *However, a lettersquare may appear **at most once** in a sequence.*

Your main task for this assignment is to write a recursive backtracking algorithm that provided a gameboard determines whether or not a given word can be found. A word is considered found if there exists a sequence of (row, column) letter-squares generated by the algorithm which forms the letter-sequence that is the same as the word; for the gameboard above and the word "DESTINY" this sequence is:

```
[(2,3), (2,2), (1,3), (0,2), (0,1), (1,1), (2,0)]
```

Programming requirements

You will complete source code for the *GameBoard* class. The class as given to you already has some crucial functionality:

- The first constructor has one parameter – an array of strings – and uses that parameter to initialize the game board. *This method has been written for you.*

- The second constructor has one parameter – a long integer – and uses that parameter to initialize a random-number generator whose values will guide the construction of a board. *This method has been written for you.*

The latter constructor, when provided the same parameter value, will *always* produce the same random board. (Please read the code comments in that constructor if you want to learn more about how it actually uses random numbers to create a board.)

- You can ignore the method named *shuffle()*.

- The method *toString()* creates a human-readable version of the created gameboard. *This method has been written for you.*

Your task is to complete the following two methods and add other private methods of your own if you need them:

- *public String isWord(String word)*: Using the parameter *word*, determine whether or not that word can be found as a sequence within the gameboard. The method must do this by, in part, calling *findPath* (mentioned below). If no sequence is found, the method returns *null*. If there is a sequence of letters, then this sequence of (row, column) values is returned as a string (i.e., in the format shown at the bottom of the previous page). ***Important***: *This method must not print out the sequence!*

- *private Boolean findPath(String wordToFind, Stack<String> pathSoFar, int row, int col)*: A method implementing the actual recursive-backtracking algorithm. The meaning and use of a variable like *pathSoFar* will be described in lectures.

You are also provided with two more classes, one of which is complete, and the other which you yourself must complete. Both of these depend upon *GameBoard.java*.

- *WordSearch.java* (*This has been written for you.*): The program first prompts the user for an integer, i.e., value for random seed. The program then repeatedly requests a word and attempts to find the word on generated gameboard. If successful, the (row, col) sequence is printed, otherwise "no path" is printed. To end the program, the exclamation mark ("!") is given as the word. A transcript showing one run of this program is on the next page.

- *FindAllWord.java* (for you to complete): Given a random seed and the name of a file with words (one line per word in the file), print out all words that are on the gameboard generated by the seed, and beside each word print the sequence of lettersquares on the board producing the word. Words in the file *not* on the board *produce no output*. **Important: You must not use a depth-first search for this function.** A transcript of a session with a completed version of this program is shown on this page. Several word-list text files are provided with this assignment (*longwords.txt*; *shortwords.txt*; *veryshortwords.txt*).

```
# Transcript of a WordSearch session
$ java WordSearch
Random seed? 4005
     0 1 2 3
     -------
  0: L I T E
  1: R N H S
  2: Y J E D
  3: I A Y E

Word to search (enter ! to exit)? destiny
[(2,3), (2,2), (1,3), (0,2), (0,1), (1,1), (2,0)]
Word to search? collusion
no path
Word to search? !
```

```
# Transcript of a FindAllWords session (completed code)
$ java FindAllWords 123 longwords.txt
     0 1 2 3
     -------
  0: A O C W
  1: L G E Z
  2: P O V K
  3: L I S T

ALPIST: [(0,0), (1,0), (2,0), (3,1), (3,2), (3,3)]
COLOVE: [(0,2), (0,1), (1,0), (2,1), (2,2), (1,2)]
EGOIST: [(1,2), (1,1), (2,1), (3,1), (3,2), (3,3)]
PLOSIVE: [(2,0), (1,0), (2,1), (3,2), (3,1), (2,2), (1,2)]
VOLAGE: [(2,2), (2,1), (1,0), (0,0), (1,1), (1,2)]
```

Testing *GameBoard*

Within *GameBoard.java* is a *main()* method already containing a little bit of code. As you write your solution, you should initially avoid working with a fully-random board and instead use the constructor to create a much simpler board.

- You can begin by the simple case of searching a simple gameboard for a single-letter word.
- Once single-letter words are working, move on to two-letter words. Experiment with paths of an expected word (including letters on a diagonal).
- The trickiest bit of programming will involve backtracking for words longer than two letters. Construct gameboards to test for cases where all letters in a word *except for the last letter* can be found as a sequence; your code must properly terminate.

- Other things to consider: use the String methods *toUpperCase()* and *toLowerCase()* as needed; think carefully about how your code handles letters that lie on a board edge; recognize that objects references given as method parameters can result in changes to those objects, such that the changes are visible outside the method; beware of adding to a sequence a lettersquare already visited in the sequence.

Files to submit:

a. *GameBoard.java*
b. *FindAllWords.java*

Grading

- Submissions are expected to follow all of the requirements detailed in the specification (assuming you are already logged into conneX, the link is `http://bit.ly/2GuuVy1`).
- The coding conventions, specified in the "Coding Conventions for CSC 115" document on conneX (link is `http://bit.ly/2D4f8oh`) must be followed.