Assignment 1

Object

To become familiar with Ragged Array Structures (array of arrays) and using 2-D array random indexing.

Back Ground

A word search puzzle consists of a board and a set of words. The board is usually laid out as a square of characters. In the assignment this square will be 25x25. The word set which is given requires the player to find the word within the board. Words may be represented forward, backward, up, down and on any of the 4 diagonals.

The Assignment

You are given a word puzzle which you must solve. The data file **wordSearch.dat** consists of 21 words followed by a 25x25 word board. You are to read the words into a ragged array. This array will consist of 21 rows, where each row represents 1 word. The board should be read into a 25x25 char matrix. For each word in the word list, identify it's location in the board. For output, print the board with just the found words.

The solution to the puzzle is given below. This also represents the desired output from your program.

```
Found   21
I
 N             PECNEREFER
  T  R      C O
   E  E    S I
    RL T  O N
     FI UC T   T      C
L     AN PE   C      O
I      CKRM  E      L
S       EE OJ      L
T         DBC     ENTER
S D       O      CP
  R             T  A
  A           KI    C  S
  O   Y       OS     KS
  B    R     N  I    AA
  Y     O       ED  L  G
  E      M       VDC    E
  K   Y   E     EEIR
     H A   M     CPTA
      E R   N     NYAH
       A R   I     ETM
        P A   A     IBI
         S     M     CUR
```

SSP

PROTOTYPE

How you should proceed with the solution and things to think about.

1. Create the data structures for the ragged array to hold the word list. Each word in the list will represent a right sized char array. The board is simply a 25x25 char matrix.
2. Read the data file loading both the word list and the board. When reading the word list consider reading a word as a String, and then using the toCharArray method to convert the string to a right sized character array.
3. Write a small test routine which prints both of these data structures to a displayer or system output. It is nice to know that your program works correctly up to this point. The output should match the input.

4. Notice that the output consists of only those words found in the board. Once you have found a word, copy that word into a second 25x25 board, we can call this Ouput. That board would start out as having each location blank, which would be the initialization state. Once a word from the word list is found, you know the starting location of that word, the direction, and the word from the word list which must be copied over.

5. When searching the board which is indexed as board[i][j], a direction to search modifies the indexing by a variable x. E.g. to search forward, board is indexed as board[i][j+x]. Where x is in the range 0 to length-1 of the word. If we wanted to search diagonally back and up, it would look like board[i-x][j-x] and so on. There will be 8 directions in total.

6. Write a search routine, which, for each char in board, will search all 8 directions for each word in the list. It might look something like this.

   ```
   For each word in the list
       For each i in the board
           For each j in the board
               Check if word[x] matches board[i][j+x] //Search forward
   ```

   Each word can only exist once in board. You will also note that this will get repetitive since the above must be repeated for each direction.

   Note as well, that if at any point the if statement fails, the search in that direction was not successful and can be terminated, progressing onto the next char in board.

7. Consider using a 2nd 25x25 matrix only for output. Once a word is found, write it into this output matrix. Note from above, once you have found a word, the base of that word is located at [i][j] with a known direction and the word.

8. Printing the output matrix will give the above output. The output matrix should be initialized to blank characters.

Keep the problem manageable by implementing each search direction one at a time. E.g. get it working for finding words spelled forward first, then implement each additional direction 1 at a time. This allows for easy debugging.

Dealing with a matrix such as this will require keeping a close tab on upper and lower bounds (Exceptions such as ArrayOutOfBounds). Use the filter method see below, which will cut down the logic required. Have it return a '*' which is non alpha and thus will cause a fail when matching to a character within a word.

Filter method

```
private char getChar(int i, int j)
```

can be used to index the board matrix, implementing bounds checking. If a valid character can be returned then that character is returned, otherwise some non-alpha character can be returned which will result in a failed match. This simplifies the comparison algorithm.

Submission

To Brightspace.

Include the complete IntelliJ project directory. The markers will be running your code under IntelliJ, ensure it words.

Zip the project using 7zip prior to submission.