Class exercises: PoemAnalyzer

========================

PoemAnalyzer

=========================

-wordsArray:String[][]

-rows:int

-cols:int

-------------------------

+PoemAnalyzer()

+PoemAnalyzer(rows:int, cols:int)

+setRow(r:int, strRow:String[]):void

+getWord(r:int, c:int):String

+findLongest():String

+contains(str:String):boolean

+toString():String

-------------------------

The above is called class diagram.

Explanations:

[+,-]varName:type = default-value

- denotes private access

+ denotes public access

[+,-]methodName(parName:type,...):returnType

=================

A class called PoemAnalyzer is designed as shown in the above class diagram.

Write the class following the class diagram and the below guidelines:

- Three private instance variables: wordsArray of the type String[][] and rows and cols of type int.

- **Constructors** - a default constructor with no argument and assigns 0 values to rows and cols,

and a second constructor which takes two arguments for rows and cols, and allocates the appropriate array space.

- Write the method **setRow**, which given a row number and an array of Strings, fills these Strings into the appropriate row in the **wordsArray**. The number of words to put in will be smaller or equal to cols.

\*\*\* Note: If the number of elements in the input array of strings is smaller than the number of columns, fill in the rest of the elements with an empty String.

- Write a method **getWord** which returns the word at location (r,c).

- Write the method **findLongest** which returns the longest word in the poem.

- Write the method **contains**, which returns true if the input String appears in the poem.

- **toString** method which returns the poem printed as in the below example. Do not worry about extra blank spaces at an end of line, or between word to a comma or full stop.

An example to the program performance is given below.

If the below is included in your tester class:

== start

PoemAnalyzer p = new PoemAnalyzer(8,6);

p.setRow(0,new String[] {"And","then","the","day","came",","});

p.setRow(1,new String[] {"when", "the", "risk"});

p.setRow(2,new String[] {"to","remain","tight"});

p.setRow(3,new String[] {"in","a","bud"});

p.setRow(4,new String[] {"was","more","painful"});

p.setRow(5,new String[] {"than","the","risk"});

p.setRow(6,new String[] {"it","took"});

p.setRow(7,new String[] {"to","blossom","."});

System.out.println("\*\*\*\n\n ANAIS NIN \"RISK\" ");

System.out.println(p + "\n\n\*\*\*\n\n");

System.out.println("Word at (row=2, col=1) is " + p.getWord(2,1));

System.out.println("The longest word in the poem: " + p.findLongest());

String str = "buddy";

System.out.println("The word " + str + " appears in the poem: " + p.contains(str));

str = "bud";

System.out.println("The word " + str + " appears in the poem: " + p.contains(str));

=== end

The output would be:

=== start

\*\*\*

ANAIS NIN "RISK"

And then the day came ,

when the risk

to remain tight

in a bud

was more painful

than the risk

it took

to blossom .

\*\*\*

Word at (row=2, col=1) is remain

The longest word in the poem: painful

The word buddy appears in the poem: false

The word bud appears in the poem: true

=== end

=============== End

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Block: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Your solution to PoemAnalyzer:

====================================