CSD 3464 – ASSIGNMENT 04 (Question 19)

Overview:

The following question is an extension to *Absolute Java* (6th Ed.)'s Chapter 06 Programming Project Q19 (pg. 425). Please follow the instructions included in **this** document and implement the following Java files:

⇒ WordGame.java *(Contains a `main()` method)*

The above classes/files should be inside a package called q*19*.

Instructions:

This assignment question differs from your last couple of assignments as you will be developing a program inside a single class called `WordGame` which contains your `main` method. Additionally, you will develop a set of `static` methods that your main program will utilize to produce the required output.

As outlined in `Absolute Java` (6th Ed.) Chapter 06 Programming Project Q19 the objective is given the `words.txt` text file and a word provided by the user via keyboard determine which words, if any, inside of `words.txt` can be constructed using only the letters in the user provided word; for example, given the word `SWIMMING` from the user possible words that can be constructed include `SWIM`, `WIN`, `MIMING`, etc. **Note**: `SINGING` cannot be constructed from `SWIMMING` as it lacks enough `N` and `G` characters.

A clear way to go about solving this problem is to generate a histogram or character frequency chart to represent words; for example, `SWIMMING` could be represented as:

G	I	M	N	S	W
1	2	2	1	1	1

We can further generalize our solution by creating an array of size 26 where index 0 would hold the number of `'A'` characters in a word and index 25 would hold the number of `'Z'` characters.

A-F	G	H	I	J-L	M	N	O-R	S	T-V	W	X-Z
	1		2		2	1		1		1	

As generating this 26 element array is essential to solving the problem a function named `generateHistogram` has already been provided to you. The function takes a single `word` as a `String` and returns a 26 element array correctly populated with `int` values representing the frequency of each letter in the provided `word`.

Next you are to implement a `static void` method called `displayHist` that takes the String `word` ands displays the word followed by its histogram representation. For instance, calling `displayHist("SWIMMING")` should display:

```
SWIMMING -> G|1 -> I|2 -> M|2 -> N|1 -> S|1 -> W|1
```

- Method will make a call to `generateHistogram` to get the 26-element array

The third step to solve this problem is to you must implement a `static` method called `canBeGeneratedFrom` that takes two words (`String word1, String word2`) and returns a `boolean` result indicating if `word1` can be constructed from `word2`.

- This method should make two (2) calls to `generateHistogram` to get the 26 element arrays for **both** words.
- Use the two arrays mentioned above to determine if `word1` can be built from the letters in `word2`.

The final step in this question is to complete the series of `//TODO` blocks inside of the `main` method to make your program produce the output shown below

```
********USER WORD*******
BOOKKEEPER -> B|1 -> E|3 -> K|2 -> O|2 -> P|1 -> R|1
********MATCHES**********
BEE -> B|1 -> E|2
BEEP -> B|1 -> E|2 -> P|1
BEEPER -> B|1 -> E|3 -> P|1 -> R|1
BEER -> B|1 -> E|2 -> R|1
BOER -> B|1 -> E|1 -> O|1 -> R|1
BOO -> B|1 -> O|2
BOOK -> B|1 -> K|1 -> O|2
BOOKER -> B|1 -> E|1 -> K|1 -> O|2 -> R|1
BOOKKEEPER -> B|1 -> E|3 -> K|2 -> O|2 -> P|1 -> R|1
BOOR -> B|1 -> O|2 -> R|1
BOP -> B|1 -> O|1 -> P|1
BORE -> B|1 -> E|1 -> O|1 -> R|1
BRO -> B|1 -> O|1 -> R|1
BROKE -> B|1 -> E|1 -> K|1 -> O|1 -> R|1
BROOK -> B|1 -> K|1 -> O|2 -> R|1
EKE -> E|2 -> K|1
```

```java
/**
 * Class that allows user to play a word game that determines
 * all possible words from words.txt that can be constructed
 * from a word provided by the user via keyboard
 */
public class WordGame
{

     /**
      * Main method used as the program driver
      * @param args Command line arguments
      */
      public static void main(String[] args)
     {
        String userWord;
        String fileWord;

        //TODO: Read a String from the user and store it in userWord.

        System.out.println("******USER WORD******");

        //TODO: Display the word along with its histogram (call one of
        //the below static functions to do this).

        System.out.println("******FOUND MATCHES******");
        //TODO: Go line by line through the words.txt reading the word
        //on each line into fileWord. If fileWord can be constructed
        //using only the letters in userWord display the fileWord
        //along with its histogram.
     }
     /**
      * Generate histogram containing the occurrence of letters A-Z
      * @param word English word with no spaces or special characters
      * @return A 26 element array containing histogram of character
      * frequency in given word
      */
     public static int[] generateHistogram(String word)
     {
        /* Index 0 is A, 1 is B, ... , 25 is Z */
        int[] charHist = new int[26];

        /* Removes any characters that are not alphabetical
           characters */
        word = word.replaceAll("[^a-zA-Z]", "").toUpperCase();

        for(int i = 0; i < word.length(); i++) {
          charHist[word.charAt(i) - 'A'] += 1; //'A' is 65 in ASCII
        }

        return charHist;
     }
```

```
    /**
     * Display histogram representation of provided word
     * @param word Word containing alphabetical characters
     */
    public static void displayHist(String word)
    {
        // insert code here
    }

    /**
     * Returns the boolean true or false depending on whether word1
     * can be built from the letters in word2
     * @param word1 Word you which to construct
     * @param word2 Word that contains the letters you have available
     * @return true if word1 can be built from word2, false
     * otherwise
     */
    public static boolean canBeGeneratedFrom(String word1, String
    word2)
    {
        // insert code here
    }

}
```