Programming Exercise: Breaking the Caesar Cipher

Before attempting these assignments, you should type in, compile, and understand the example programs from the lesson, including: 1) counting the twenty most common words from Shakespeare's plays, 2) counting the resulting random rolls of dice (how many 2's, 3's, 4's, etc), and 3) automatic decryption of the Caesar Cipher using statistical letter occurrences.

Assignment 1: Word lengths

You will write a program to figure out the most common word length of words from a file. To solve this problem you will need to keep track of how many words from a file are of each possible length. You should group all words of length 30 or more together, and you should not count basic punctuation that are the first or last characters of a group of characters.
Specifically, you should do the following:

- Create a new class called WordLengths.
- Write a void method **countWordLengths** that has two parameters, a FileResource named **resource** and an integer array named **counts**. This method should read in the words from **resource** and count the number of words of each length for all the words in **resource**, storing these counts in the array **counts**.
    - For example, after this method executes, **counts[k]** should contain the number of words of length **k**.
    - If a word has a non-letter as the first or last character, it should not be counted as part of the word length. For example, the word `And,` would be considered of length 3 (the comma is not counted), the word `"blue-jeans"` would be considered of length 10 (the double quotes are not counted, but the hyphen is). Note that we will miscount some words, such as `"Hello,"` which will be counted as 6 since we don't count the double quotes but will count the comma, but that is OK as there should not be many words in that category.
    - For any words equal to or larger than the last index of the **counts** array, count them as the largest size represented in the **counts** array.

    - You may want to consider using the `Character.isLetter` method that returns true if a character is a letter, and false otherwise. For example, `Character.isLetter('a')` returns `true`, and `Character.isLetter('-')` returns `false`.
- Write a void method **testCountWordLengths** that creates a FileResource so you can select a file, and creates a **counts** integer array of size 31. This method should call **countWordLengths** with a file and then print the number of words of each length. Test it on the small file **smallHamlet.txt** shown below.
- Write a method **indexOfMax** that has one parameter named **values** that is an integer array. This method returns the index position of the largest element in **values**. Then add code to the method **testCountWordLengths** to call **indexOfMax** to determine the most common word length in the file. For example, calling **indexOfMax** after calling **countWordLengths** on the file **smallHamlet.txt** should return 3.

First test your program on a small file, such as this simple file shown called **smallHamlet.txt**:

```
Laer. My necessaries are embark'd. Farewell.
And, sister, as the winds give benefit
```

Note this file has words that are:

2 words of length 2: My as

3 words of length 3: are And the

2 words of length 4: Laer give

1 word of length 5: winds

1 word of length 6: sister

1 word of length 7: benefit

2 words of length 8: embark'd Farewell

1 word of length 11: necessaries

Assignment 2: Caesar Cipher Two Keys Decrypt

You should start by writing the decryption method explained in the lesson that decrypts a message that was encrypted with one key, using statistical letter frequencies of English text. Then you will add code to be able to decrypt a message that was encrypted with two keys, using ideas from the single key decryption method and the encryption with two keys method from the program you wrote in the last lesson.

Idea for two keys decrypt method. Recall that in using two keys, **key1** and **key2**, **key1** was used to encrypt every other character, starting with the first, of the String, and **key2** was used to encrypt every other character, starting with the second. In order to decrypt the encrypted String, it may be easier to split the String into two Strings, one String of all the letters encrypted with **key1** and one String of all the letters encrypted with **key2**. Then use the algorithm from the lesson to determine the key for each String, and then use those keys and the two key encryption method to decrypt the original encrypted message.

For example, if the encrypted message was `"Qbkm Zgis"`, then you would split this String into two Strings: `"Qk gs"`, representing the characters in the odd number positions and `"bmZi"` representing the characters in the even number positions. Then you would get the key for each half String and use the two key encryption method to find the message. Note this example is so small it likely won't find the keys, but it illustrates how to take the Strings apart.

A sample file to test your program that is small with lots of e's is called **wordsLotsOfEs.txt** and shown here:

```
Just a test string with lots of eeeeeeeeeeeeeeeees
```

And the same file encrypted using keys 23 and 2 is called **wordsLotsOfEsEncrypted.txt** and is shown here:

```
Gwpv c vbuq pvokki yfve iqqu qc bgbgbgbgbgbgbgbgbu
```

Specifically, you should do the following.

- Complete the decryption method shown in the lesson by creating a CaesarBreaker class with the methods **countLetters**, **maxIndex**, and **decrypt**. Recall that the **decrypt**

method creates a CaesarCipher object in order to use the **encrypt** method you wrote for the last lesson. *Make sure that your CaesarCipher class is in the same folder as CaesarBreaker!* You may want to use the following code as part of your **decrypt** method.

```
CaesarCipher cc = new CaesarCipher();
String message = cc.encrypt(encrypted, 26 - key);
```

Write a **testDecrypt** method in the CaesarBreaker class that prints the decrypted message, and make sure it works for encrypted messages that were encrypted with one key.

- Write the method **halfOfString** in the CaesarBreaker class that has two parameters, a String parameter named **message** and an int parameter named **start**. This method should return a new String that is every other character from message starting with the start position. For example, the call `halfOfString("Qbkm Zgis", 0)` returns the String `"Qk gs"` and the call `halfOfString("Qbkm Zgis", 1)` returns the String `"bmZi"`. Be sure to test this method with a small example.
- Write the method **getKey** in the CaesarBreaker class that has one parameter, a String **s**. This method should call **countLetters** to get an array of the letter frequencies in String **s** and then use **maxIndex** to calculate the index of the largest letter frequency, which is the location of the encrypted letter 'e', which leads to the key, which is returned.
- Write the method **decryptTwoKeys** in the CaesarBreaker class that has one parameter, a String parameter named **encrypted** that represents a String that was encrypted with the two key algorithm discussed in the previous lesson. This method attempts to determine the two keys used to encrypt the message, prints the two keys, and then returns the decrypted String with those two keys. More specifically, this method should:
    - Calculate a String of every other character starting with the first character of the encrypted String by calling **halfOfString**.
    - Calculate a String of every other character starting with the second character of the encrypted String.
    - Then calculate the key used to encrypt each half String.

- You should print the two keys found.
- Calculate and return the decrypted String using the **encryptTwoKeys** method from your CaesarCipher class, again making sure it is in the same folder as your CaesarBreaker class.