Lab 19 (November 13 or November 14)

Instructions: Complete the exercises below. Be sure to show your code to one of the lab TAs before you leave, so that you can receive credit for this lab. You must also upload a copy of all your source code (`.java`) files through the link on Blackboard by 11:59 PM on Tuesday, November 14.

1. The *Fibonacci sequence* is a well-known integer sequence that starts with the values 0 and 1 (in that order). Every number after that is the sum of the two previous values in the sequence. For example, 0 and 1 are followed by another 1 (0 plus 1 is just 1), then by 2 (1 plus 1), 3 (1 plus 2), etc. Basically, for all n >= 2, term n is equal to (term n-1 plus term n-2).

   The first 10 terms of the Fibonacci sequence are 0, 1, 1, 2, 3, 5, 8, 13, 21, and 34.

   Write a Java program that prompts the user to enter two (positive) integer values: the total length *N* of the Fibonacci sequence to generate (assume that this value will always be greater than or equal to 3), and a number *P* of terms to print (assume this value will always be less than or equal to *N*). Your program should print the last P terms of the first N terms of the Fibonacci sequence. For example, if the user enters 50 for N and 10 for P, your program should calculate the first 50 terms of the Fibonacci sequence, but only print the last 10 terms (basically, the 41^{st} through 50^{th} values).

   Use an ArrayList of integer values (declare its type as ArrayList<Integer>) to store the Fibonacci sequence (don't forget to prime the ArrayList with the initial values 0 and 1); as you calculate each new term, add it to the end of the ArrayList (**Hint:** keep going as long as the ArrayList's size is less than N). When you are finished generating the terms of the sequence, use a new loop to retrieve and print the last P terms of that sequence, in order.

   For example, if N is 20 and P is 5, your program should print out the values 610, 987, 1597, 2584, and 4181 (which are the last 5 values of the first 20 terms of the Fibonacci sequence).

   As a second example, if N is 14 and P is 6, your program should print out 21, 34, 55, 89, 144, 233.

2. The Welsh alphabet consists of the following letters:

   a, b, c, ch, d, dd, e, f, ff, g, ng, h, i, j, l, ll, m, n, o, p, ph, r, rh, s, t, th, u, w, y

   Note that there are eight *digraphs* (two-symbol combinations that count as single letters) in this alphabet: ch, dd, ff, ng, ll, ph, rh, and th. Thus, for example, the town name Llanelli is considered to have only six letters in Welsh (each *ll* counts as a single letter), whereas it would have eight letters in English. Along these lines, ``rhyl'' has 3 letters, and ``cymru'' has 5 letters. In potentially ambiguous cases like *lll*, assume that digraphs precede any single

letters (so *lll* would be interpreted as the digraph *ll* followed by a single *l*, for a total letter count of 2).

Define a Java method `welshLetters()` that takes a lowercase String as its only argument (you may assume that the String only contains the letters listed in the Welsh alphabet above, so it won't contain any q's or z's, for instance). Your method should return an ArrayList of Strings, where each String is the next Welsh letter from the argument.

```
ArrayList<String> welshLetters(String word)
```

For example, `welshLetters("llanelli")` would return an ArrayList with the contents (in order):

```
["ll", "a", "n", "e", "ll", "i"]
```

Write a small Java program that demonstrates that your method works correctly. It should prompt the user to enter a Welsh word and display the total number of Welsh letters (and what they are; remember that you can simply pass an ArrayList to System.out.println() for this purpose). For example:

Please enter a Welsh word: **heddiw**
Welsh letters: 5 ["h", "e", "dd", "i", "w"]

Grading Guidelines: This lab is graded on a scale of 0-3 points, assigned as follows:

0 points: Student is absent or does not appear to have completed any work for the lab

1 point: Student has written only one program, but it does not compile or run at all due to errors.

2 points: Student has written (or attempted to write) both programs, but only one compiles and runs without error.

3 points: Student has written both programs, and they both compile and run correctly, without any apparent errors.