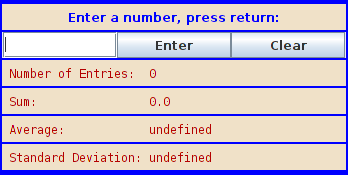
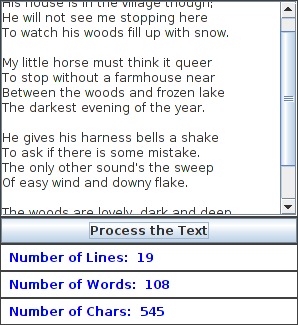
Exercise 6.7:

[Exercise 5.2](http://math.hws.edu/javanotes/c5/ex2-ans.html) involved a class, [StatCalc.java](http://math.hws.edu/javanotes/source/chapter5/StatCalc.java), that could compute some statistics of a set of numbers. Write a GUI program that uses the StatCalc class to compute and display statistics of numbers entered by the user. The panel will have an instance variable of type StatCalc that does the computations. The panel should include a JTextField where the user enters a number. It should have four labels that display four statistics for the numbers that have been entered: the number of numbers, the sum, the mean, and the standard deviation. Every time the user enters a new number, the statistics displayed on the labels should change. The user enters a number by typing it into the JTextField and pressing return. There should be a "Clear" button that clears out all the data. This means creating a new StatCalc object and resetting the displays on the labels. My panel also has an "Enter" button that does the same thing as pressing the return key in the JTextField. (Recall that a JTextField generates an ActionEvent when the user presses return, so your panel should register itself to listen for ActionEvents from the JTextField as well as the buttons.) Here is a picture of my solution to this problem:



Exercise 6.8:

Write a program that has a JTextArea where the user can enter some text. Then program should have a button such that when the user clicks on the button, the panel will count the number of lines in the user's input, the number of words in the user's input, and the number of characters in the user's input. This information should be displayed on three labels. Recall that if textInput is a JTextArea, then you can get the contents of the JTextArea by calling the function textInput.getText(). This function returns a String containing all the text from the text area. The number of characters is just the length of this String. Lines in the String are separated by the new line character, '\n', so the number of lines is just the number of new line characters in the String, plus one. Words are a little harder to count. [Exercise 3.4](http://math.hws.edu/javanotes/c3/ex4-ans.html) has some advice about finding the words in a String. Essentially, you want to count the number of characters that are first characters in words. Don't forget to put your JTextArea in aJScrollPane, and add the scroll pane to the container, not the text area. Scrollbars should appear when the user types more text than will fit in the available area. Here is a picture of my solution:



Exercise 10.5:

An example in [Subsection 10.4.2](http://math.hws.edu/javanotes/c10/s4.html#generics.4.2) concerns the problem of making an index for a book. A related problem is making a concordance for a document. A concordance lists every word that occurs in the document, and for each word it gives the line number of every line in the document where the word occurs. All the subroutines for creating an index that were presented in [Subsection 10.4.2](http://math.hws.edu/javanotes/c10/s4.html#generics.4.2) can also be used to create a concordance. The only real difference is that the integers in a concordance are line numbers rather than page numbers.

Write a program that can create a concordance. The document should be read from an input file, and the concordance data should be written to an output file. You can use the indexing subroutines from[Subsection 10.4.2](http://math.hws.edu/javanotes/c10/s4.html#generics.4.2), modified to write the data to TextIO instead of to System.out. (You will need to make these subroutines static.) The input and output files should be selected by the user when the program is run. The sample program [WordCount.java](http://math.hws.edu/javanotes/source/chapter10/WordCount.java), from [Subsection 10.4.4](http://math.hws.edu/javanotes/c10/s4.html#generics.4.4), can be used as a model of how to use files. That program also has a useful subroutine that reads one word from input.

As you read the file, you want to take each word that you encounter and add it to the concordance along with the current line number. Keeping track of the line numbers is one of the trickiest parts of the problem. In an input file, the end of each line in the file is marked by the newline character, '\n'. Every time you encounter this character, you have to add one to the line number. WordCount.java ignores ends of lines. Because you need to find and count the end-of-line characters, your program cannot process the input file in exactly the same way as does WordCount.java. Also, you will need to detect the end of the file. The function TextIO.peek(), which is used to look ahead at the next character in the input, returns the value TextIO.EOF at end-of-file, after all the characters in the file have been read.

Because it is so common, don't include the word "the" in your concordance. Also, do not include words that have length less than 3.