Improving and Measuring Program Efficiency

In this assignment, we'll optimize the implementation of the Document class that you implemented in your last assignment, and then measure how much faster your new implementation is.

Setup

Before you begin this assignment, make sure you check Part 2 in the setup guide to make sure the starter code has not changed since you downloaded it. If you are an active learner, you will have also received an email about any starter code changes. If there have been any changes, follow the instructions in the setup guide for updating your code base before you begin.

1. **(Optional) Download and use our solutions for Document.java and BasicDocument.java**

First, if you are feeling unsure about your Document.java solution, you are welcome to use ours. With Eclipse closed, **move your BasicDocument.java and Document.java files somewhere else, outside of your workspace.** That is, on your file system, you should find these files in the MOOCTextEditor/src/document folder, and then move them somewhere else. This is so that you don't lose them when you download our solution. Then, download our BasicDocument.java and Document.java solution files which are linked from the online version of this assignment and save them back into your MOOCTextEditor/src/document directory on your file system:

2. **Start Eclipse and find the starter code for this assignment**

The starter code for this assignment can be found in the document package. You will be working with the files EfficientDocument.java and DocumentBenchmarking.java. You will need EfficientDocument.java for part 1 and DocumentBenchmarking.java for part 2. Open these two files now.

3. **(If needed) Modify LaunchClass.java so that getDocument returns an EfficientDocument instead of a BasicDocument** (only needed if you changed LaunchClass.java in week 1). If you modified LaunchClass.java so that its getDocument method returned a BasicDocument last week, you should modify it back to returning a document.EfficientDocument for this week's assignment.

Assignment Instructions

This assignment has two parts. The instructions below tell you how to produce the files you need to upload for a grade in each part.

Part 1: Complete the Implementation of EfficientDocument

1. Implement processText() in EfficientDocument.java.

This method should make ONE pass through the tokens list and count the number of words, sentences and syllables in the document. It should store these values in the appropriate member variables so that when the method is over, they never have to be re-calculated.

You should make sure you understand the provided method getTokens, isWord, and the provided first line of code in the processText method. If you don't, or you need some more hints about how to get started, you can find a little more information linked from the online version of these instructions.

IMPORTANT: Notice that countSyllables and isWord **do not** use Matcher objects or regular expressions. This is important. It turns out that the process of creating a Pattern and a Matcher object for each word is slow enough that if you take this approach your EfficientDocument will end up being *slower* than your BasicDocument, because of the overhead of creating and destroying objects in memory. So use the single regex we provide at the start of the processText method, but don't try to use Patterns or Matchers on the individual tokens returned from the first call to getTokens.

2. Implement getNumWords, getNumSyllables, and getNumSentences in EfficientDocument.java

Once you have implemented the processText method, the implementations of these three methods is trivial.

You can again use the method testCase in the Document class to test the correctness of your implementation. We've provided a few test cases for you. You should add more.

Part 2: Benchmarking

Your next task is to determine and plot how much faster EfficientDocument is than BasicDocument in computing a single Flesch score for a document.

1. **Calculate the Big-O running time of your code** to compute the Flesch score for BasicDocument and EfficientDocument (*including the time taken by processText!)* and predict how the running time of calling fleschScore on BasicDocument and EfficientDocument will grow as the document size grows.

2. **Open the file DocumentBenchmarking.java**, which you can find in the document package. You will be adding code to the main method, and you will find the method getStringFromFile useful for reading a specified number of characters from a text file.

3. **Complete the main method** so that it prints a table with one column each for the amount of time it takes to create a BasicDocument or an EfficientDocument, respectively, and call fleschScore on it. Your table should look like the following:

NumberOfChars BasicTime EfficientTime

5000 [basic time 1] [efficient time 1]

10000 [basic time 2] [efficient time 2]

15000 [basic time 3] [efficient time 3]

...

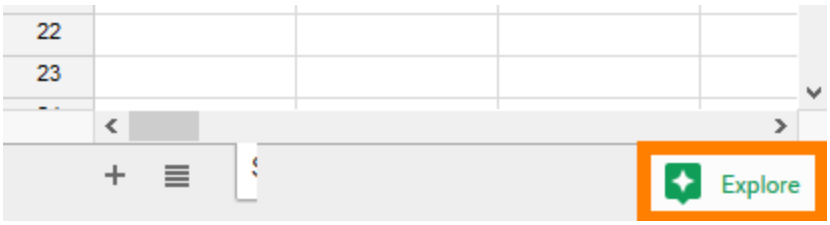
Where the first column in the table prints the number of characters in the test, the second column is the amount of time it takes to create a BasicDocument with that many characters and call fleschScore on it, and the third column is the amount of time it takes to create an EfficientDocument with that many characters and call fleschScore on it. In these timing calculations, you should include the time taken by both constructors (i.e. the time EfficientDocument takes calculating the number of words, syllables and sentences in the document), otherwise it's not a fair comparison at all!

Columns in the table should be separated with a \t character for easy copy and paste in the next section. You can also write this table to a file if you want, but it's not necessary.

You can find some notes and hints about this part here (same document as linked in part 1)

Hints and additional information for Programming Assignment 2.pdf×

4. Plot your results and compare them to your predictions. Copy the output of your program and paste it into a Google spreadsheet (sheets.google.com)**.** It should automatically paste into 3 columns. Then, graph the data using the Explore button in the lower right corner, as shown in the image below.



Do your results look like what you expected them to? If not, consider whether you've done something wrong in your code somewhere and try to fix it. But remember that empirical tests always have some noise, so some deviations from "perfect" data are expected.

****To better understand your results, we strongly encourage you to discuss your results on the discussion forum****

Working with real data is fun, but is best done when you have someone else to talk about it with to make sure you really understand what's going on.

What and How to Submit

1. Create a zip file containing only the files Document.java, EfficientDocument.java and BasicDocument.java To do this, you must locate these files on your computer's filesystem, in the workspace directory you set up for Eclipse.

2. Upload this zip file for BOTH parts 1 and 2. You will upload the same file for both parts, but in part 1 we will grade the correctness of your methods, while in part 2 we will grade their speed (they must be faster than BasicDocument's implementation).

3. Submit! Like last week, grading will take a few minutes, during which time you will see a 0 as your score. Don't worry! Your correct score will refresh once grading is done.

If you get any errors, you can run the EfficientDocumentGrader code we provide and use that class to help you debug your program. If you get errors with part 2, it is likely because your implementation is not efficient enough.

After you pass, make sure you understand your graphs from part 2. We will ask you about them in the quiz that follows this assignment.