Programming Assignment for Module 2: Improving and Measuring Program Efficiency

NOTE: The submission instructions have been updated. Please see the submission instructions below.

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 from the following links and save them back into your MOOCTextEditor/src/document directory on your file system:

BasicDocument.java

Document.java

Additionally, if last week you changed LaunchClass.java's getDocument method to return a BasicDocument object, change it back so that it creates and returns a document.EfficientDocument object.

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 and Submission 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

You might have already noticed that what you did in fleschScore is inefficient, because we have to pass through the whole document each time we count the syllables, words and sentences in it. Instead, we could have calculated all three in just one pass through the document, and stored these values for later so we wouldn't have to recalculate them if someone asked for them again!

EfficientDocument does just that. It uses three member variables to store the number of sentences, words and syllables in the document. It sets the value of these variables during the processText() method, which it runs from the constructor. Your job is to implement this method so that EfficientDocument works correctly.

Implement processText()

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.

Notice that we have given you the pattern string to use to split the text. Make sure you understand what this regular expression will split the text into. You might find it useful to print out the list of tokens for some examples to help you get started.

Notice also that we've provided a method that checks to see if a given string is a word.

boolean isWord(String s)

The idea behind this method is that it takes a string that either contains sentence ending punctuation, or does not contain sentence ending punctuation. You'll be able to use this method for counting the number of words AND the number of sentences. (If you are keeping either words OR sentence ending punctuation, if a token isn't a word, what is it?) *Pay special attention to the case where the last sentence does not end in a sentence-ending punctuation.* It should still be counted as a sentence, but you'll need to handle that case carefully...

You'll also find the method countSyllables in the Document class useful. Hopefully you implemented this last time, but if you didn't, we've provided it for you in our solution.

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.

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.

What and how to submit part 1

When you have tested your code and are ready to submit this part, you should 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. You can name this file whatever you like, but we recommend mod2.zip.

Once you have created the zip file upload this file for grading. Note that because we are extracting and running your files to grade them, grading will take a minute or two. While the grading is taking place, you will see a score of 0 displayed. This does not mean you got a 0! Your correct score will refresh once grading is done.

Part 2: Benchmarking

Your 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]

...

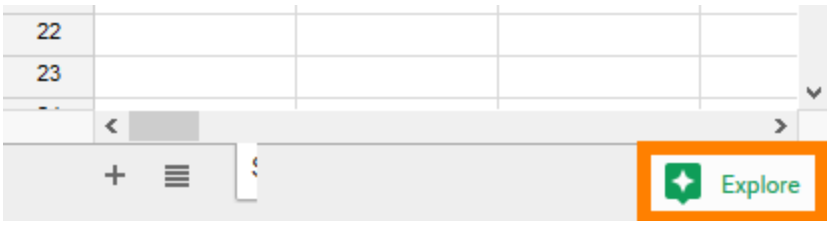
Where the first column in the table prints the number of characters in the test, the second column is the amount of time it takes to create a BasicDocument with that many characters and call fleschScore on it, and the third column is the amount of time it takes to create an EfficientDocument with that 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.

Notes and hints on this part:

- The main method in the starter code provides pseudocode for this part in the comments.
- You can print the time in any units you want, but recall that the unit of System.nanoTime() is nanoseconds, so to convert these to seconds you can divide by 1000000000 (10^9). Be sure to use type long to store the results of System.nanoTime()! ints don't give you room to store big enough numbers.
- Your data will be too noisy if you create only one BasicDocument and one EfficientDocument for each size, so we recommend you use a loop to create and call fleschScore on many BasicDocument and EfficientDocument objects and measure the total time for the whole loop. This is what we have provided the 'trials' variable for: it is the number of trials that you do for each type of document at each size. Don't worry about dividing the total time by the number of trials. You can just output the total time. Play around with different values of trials until you get data that is smooth.

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 for part 2

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. You can name this file whatever you like, but we recommend mod2.zip.

Once you have created the zip file upload this file for grading. Note that because we are extracting and running your files to grade them, grading will take a minute or two. While the grading is taking place, you will see a score of 0 displayed. This does not mean you got a 0! Your correct score will refresh once grading is done.

Finally, you will answer a few questions about the running time of your documents on the quiz, which follows this programming assignment. Once you pass the graded portion of this assignment and have completed all of the steps above, you're ready to take the quiz!