Programming Assignment for Module 1: How Easy to Read is Your Writing?

NOTE: The submission instructions have changed. You will now submit your code in a zip file. Please review the submission instructions below.

In this first programming assignment, which is the first part of the project you will develop throughout this course, you will write code to analyze the reading level of a piece of text. The Flesch Readability Score is a measure of the reading complexity of a piece of text. Developed by author Rudolf Flesch, it is a measure that approximates how easy a piece of text is to read based on the number of sentences, words and syllables in that text. Higher scores indicate text that is simple to read, while lower scores indicate more complex text.

In this programming assignment you will write the back-end code for calculating Flesch readability scores, and then you will integrate this back-end capability with the front-end application we provide that you will be adding to throughout this course.

Getting Set Up

1. (If not already done) Download and set up the starter code

Before you begin this programming assignment, you need to download and set up the starter code. You should have already followed the instructions for setting up Java and Eclipse . Make sure you have the most recent version of the starter code. Because this course is so new and the starter and grading code is relatively complex, we occasionally need to update the starter code with bug fixes. The zip file on the set up page is annotated with the date it was uploaded, so if there's a new version since you downloaded it, you should grab it now. You are welcome to use any IDE of your choice, but we provide the starter code as an Eclipse project, so you'll probably find it easiest to work in Eclipse.

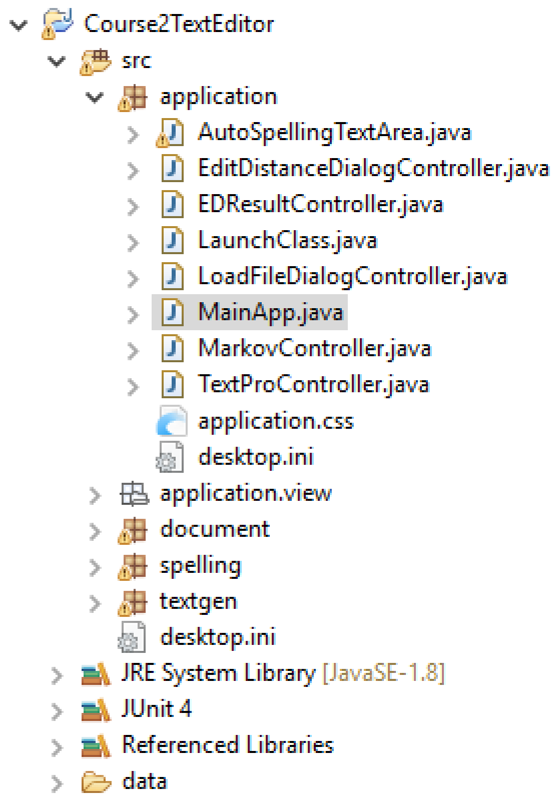
Make sure you've followed these setup instructions even if you've previously worked with Eclipse, and even if you took the first course in our series, because you'll need to make sure you have some additional pieces installed and set up (e.g. Java 1.8 latest version, JUnit).

2. Verify that the front-end runs

Make sure the project you downloaded and set up for this course is open in eclipse.

As you saw in Mia's demo video, all of the functionality you will implement in this course will be integrated into a text editor application. You can already run this application, but it won't do much.

Run the text editor by running the MainApp class inside the application package. You can do this by expanding the application package in the package explorer and then selecting the MainApp.java file and running it.



```
Course2TextEditor
  src
    application
      AutoSpellingTextArea.java
      EditDistanceDialogController.java
      EDResultController.java
      LaunchClass.java
      LoadFileDialogController.java
      MainApp.java
      MarkovController.java
      TextProController.java
      application.css
      desktop.ini
    application.view
    document
    spelling
    textgen
    desktop.ini
  JRE System Library [JavaSE-1.8]
  JUnit 4
  Referenced Libraries
  data
```

You will see a text editor window open. You can type in the text window and load text using the "Load" button. None of the other buttons work yet (you can click them, but they don't do anything). But they will soon...

3. Orient yourself to the starter code

In this project, you will write a class to represent a text Document. The Document class will be capable of calculating and returning properties of the text, including its Flesch readability score. Before you begin coding, you should get comfortable with the class hierarchy (if you need a refresher on inheritance and class hierarchies in Java, check out the first course in our specialization).

Open the starter code for this assignment by expanding the document package in the Package Explorer window. There you will see several files, but the two that are relevant to this assignment are Document.java and BasicDocument.java. Open these by double-clicking on them.

Notice that Document is an abstract class. BasicDocument will implement the abstract methods in the Document class using the guidelines below described in the parts below.

Assignment and Submission Details

This assignment is divided into two parts. You will submit a separate file for each part, as described below.

Part 1: Implement the missing methods in BasicDocument.java

In this class, countSyllables, countWords, and countSentences each calculate the number of syllables, words, and sentences, respectively, each time the method is called. **It should do this by processing the entire text of the document each time the method is called. It should NOT store this information after it calculates it**, but rather just return it. This class is not supposed to be efficient.

To facilitate auto-grading, you must use the following definitions of what constitutes a syllable, word, and sentence and make sure you are consistent with our test cases/examples in the main method.

public int getNumSentences()

This method counts and returns the number of sentences in the document. A sentence is any sequence of characters ending in one or more end of line punctuation marks, or the last sequence of characters in a document, even if they don't end with an end of line punctuation mark. An end of line punctuation mark includes a period (.), an exclamation point (!) and a question mark (?). That's it. Notice that we provide a main method that gives you several examples/test cases.

public int getNumWords()

This method counts and returns the number of words in the document (the stored text). A word is any contiguous sequence of one or more alphabetical characters. We will completely ignore numbers when we count words, and you can assume you will not have any strings that combine numbers and letters. Again, check our examples in the main method.

public int getNumSyllables()

This method counts and returns the total number of syllables in the document (the stored text). To count the number of syllables in a word, you should use the following rule:

Each contiguous sequence of one or more vowels is a syllable, *with the following exception*: a lone "e" at the end of a word is not considered a syllable unless the word has no other syllables. You should consider y a vowel.

Under these rules the words “the”, “fly”, “yes”, “cave” and “double” all have 1 syllable, but "segue" has two syllables. Notice that this isn't exactly correct (“double” actually has 2 syllables), but it's close enough for our purposes. Here are some more examples with the number of syllables your method should return to help you: "contiguous" (3 syllables), "sleepy" (2 syllables), "obvious" (2 syllables), "toga" (2 syllables). Notice that our rules get a lot wrong, especially when you have more than 2 vowels in a row, but these are the rules we will test you against.

Testing your code:

We have provided a method named testCase in the Document class. This method takes a Document object (which can be a BasicDocument object…) as well as the expected number of syllables, words, and sentences in that document and tests to make sure each method is correct, printing messages accordingly. You should make use of this method and add your own tests in the main method we provide. We've provided some tests to get you started.

Hints for this part:

- Notice the helper method getTokens in the Document superclass. It takes a string which is a regular expression representing the regular expression of the “tokens” it’s trying to find. Consider passing a different pattern for each method. If you get really stuck here there is a support video that can help.
- It is possible to write regex's to count the number of syllables directly (we did it with a combination of 3, but it might also be possible to write a single one), but this is not necessarily the best approach. If you find yourself doing mental gymnastics to come up with a single regex to count syllables directly, that's usually an indication that there's a simpler solution (hint: consider a loop over characters--see the next hint below). Just because a piece of code (e.g. a regex) is shorter does not mean it is always better.
- Consider making a helper method in the Document class (i.e. the superclass) that counts the number of syllables in a single word. This will come in useful in next week's assignment as well. We have a stub method already in Document.java. You should not need to create countWords or countSentences methods in Document.java because these methods will not be useful next week. For reasons of efficiency, which we'll mention more in next week’s assignment, you should not create Matcher or Pattern objects inside your countSyllables method. Just use a loop to loop through the characters in the string and write your own logic for counting syllables.

Explanation for sample test case

- "many??? Senteeeeeeeeeences are"

For this string, getNumSentences() will return 2 because there is one sequence of characters ending with a series of end of line punctuation marks, as well as the last sequence of characters which also counts as a sentence (despite not ending with an end of line punctuation mark.)

getNumWords() will return 3 words because there are three sequences of contiguous alphabetical characters (many, Senteeeeeeeeeences, are).

getNumSyllables() should evaluate to 6 syllables because there are six contiguous sequences of vowels which fit our definition of a syllable. They are:

- many: 2 (a, y)
- Senteeeeeeeeeences: 3 (e, eeeeeeeeee, e)
- are: 1 (a) -- note the lone e on the end does not count in this case.

We have provided several other test cases in the main method in BasicDocument. If you have questions about number of sentences, words or syllables, post in the discussion forum and we or your peers are happy to help you understand them.

Making your code work with the text editor application (optional, but encouraged)

Currently, the text editor application is not set up to use the BasicDocument class (it uses EfficientDocument, which you will implement next week). So if you want your text editor application to calculate the Flesch score (after you finish part 2), you must change one line of code in the file LaunchClass.java, which is in the application package. In the method

public document.Document getDocument(String text)

change "document.EfficientDocument" to "document.BasicDocument". Remember to change this back next week!

What and how to submit

When you have tested your code and are ready to submit this part, you should create a zip file containing only Document.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 mod1.zip.

Once you have created the zip file, upload this file for grading. You must upload the zip file containing both Document.java and BasicDocument.java. **DO NOT upload either file separately.** 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. Upload this file as your submission for part 1 and click submit to see your grading feedback! If there are any errors, fix them and then try submitting again. You can see exactly what test cases we used by looking at the code for the grader and checking the file the grader is reading from.

Part 1 Bonus (purely optional, just for fun):

Our REGEX are pretty naive. For example, the word 7.5 causes real problems for our expressions as a "." usually denotes the end of a sentence. If you want to learn more about REGEX, you can improve upon our approach. But if you want to do this, put it in a separate class (like ImprovedDocument) and make sure you don't use it to produce your grading output.

Part 2: Implement the getFleschScore method in Document.java

Now fill in the method getFleschScore() in Document.java to calculate the Flesch Score for the text in the document. You should use the following formula, and make calls to the getNumSyllables, getNumWords, and getNumSentences you just implemented.

$$Flesch\ score = 206.835 - 1.015\left(\frac{\#\ words}{\#\ sentences}\right) - 84.6\left(\frac{\#\ syllables}{\#\ words}\right)$$

You should test your code by calculating the Flesch score by hand on some very basic documents and then calling your method from main to make sure it's giving the same output.

What and how to submit

To submit part 2, you should again create a zip file containing Document.java and BasicDocument.java You can again name this file whatever you like, but we again recommend mod1.java. At this point you can actually submit this same zip file for both parts (since it now contains solution code for both parts). Upload this zip file as your submission for part 2 (and part 1 if you want, but if you've already passed part 1, that's not necessary). You must upload this zip file containing both documents.

Part 3 (nothing to submit): Have fun with calculating the readability of text that you find

Now that you've implemented the Flesch score, your GUI interface will automatically include this functionality. Try running it again, and playing around with calculating the Flesch score of various documents, either that you have produced or that you find on the web. We'd love to see fun things you discover on the discussion board!