ICT/KTH
08-Oct-2010/FK (updated)

id1006 Java Programming

Assignment 3 - Flesch metric and commandline parsing

It is recommended that you submit this work no later than Tuesday, 12 October 2010. Solution examples will be presented on 13 October.

This assignment is part of the individual student examination on the course id1006 Java Programming. The assignment is to be done by an individual student. When the assignment has been approved, it corresponds to 1/5th credit of the 7.5 credits (hp) given for the completed course.

HOW TO SUBMIT THE COMPLETED ASSIGNMENT

The completed work is delivered electronically in the form of an email addressed to

java.assignments@fc.dsv.su.se

In an emergency, send the email to fki@kth.se.

Send one email per assignment. Since there are three programming tasks and one essay in this course, a student is expected to submit a total of four emails.

The subject of the email must contain the text 'id1006 Assignment xxx' or 'id1006 Essay' etc.

The body of the email must contain the submitting student's name and optional civic registration number (Sv. 'personnummer'). The sender id on the email is NOT sufficient identification.

The body of the email must also contain all additional information needed to identify the submitted work and the context in which it is being submitted. For example, a re-submission.

Submitted files (e.g. program sources) should be adjoined to the email as one or more attachments.

SOURCE CODE is to be submitted as PLAIN TEXT, ie files that can be compiled by the Java standard development kit (javac). All files necessary to build the program or programs must be submitted together. If you send an archive, let it be ZIP.

Typeset documents (e.g. Ms Word, Open Office, LaTex etc) MUST be submitted in PDF, the Portable Document Format by Adobe. This is currently the optimal way to guarantee cross-platform readability of electronic documents.

Submitted work is expected to be carefully prepared, annotated, commented and above all original. Where it is not, quotes, citations, and references are to be CLEARLY indicated. Images, graphics and other multimedia products can only be incorporated into the submitted work with the permission of the copyright holder, and the permission must be expressed in the submitted work.

Submitted work will be tested for originality.

The three programming assignments for the fall 2010 instance of the course are all related. Together, they create a simple and extensible application for estimating the readability of english text.

A readability index (of which there are several) is language dependent, statistical and computable. They are usually constructed by computing a ratio between the average number of words per sentence, and the proportion of complicated words to simple words. As a result, they usually return a single figure, like 30, or 14.2.

One famous readability metric for english is the Flesch index:

$$\text{Flesch} = 206.876 - \left(1.015 * \frac{O}{P}\right) - \left(84.6 * \frac{Y}{O}\right)$$

where

- O = the number of words
- P = the number of sentences
- Y = the number of syllables

in the text.

The Flesch metric is a weighted sum of the average number of words per sentence, and the average number of syllables per word. Since more syllables requires more letters in a word, it is quite similar to the Lix metric. The major difference, however, is that Flesch is continuous over word length, while Lix uses a threshold for long words at six characters. Also, while the Lix index increases with more difficult texts, the Flesch index is a negative sum, and thus decreases with lower readability.

A possible interpretation of the Flesch reading ease score is

0 - 29	Very difficult
30 - 49	Difficult
50 - 59	Fairly difficult
60 - 69	Standard
70 - 79	Fairly easy
80 - 89	Easy
90 - 100	Very easy

[Source: http://rfptemplates.technologyevaluation.com/readability-scores/flesch-reading-ease-readability-score.html]

Assignment 3 - FleschMeter.java, SyllableCounter.java, and FleschTest.java

The Flesch TextMeter - FleschMeter.java

In order to complete this assignment, you must have completed assignment 1, because you will need its compiled classes in order to run. See below for how to organize the assignment files in separate folders and set up the CLASSPATH environment variable.

The third assignment consists of writing three new source files, a TextMeter implementation called FleschMeter.java, a syllable counter called SyllableCounter.java and a new main program called FleschTest.java.

The FleschMeter class should compute the Flesch readability index as expressed above. The property map should contain these properties:

words	the number of words
syllables	the number of syllables (in the whole text)
sentences	the number of sentences
Flesch	the Flesch ease of reading score

The easiest way to go about this, is to copy the file SimpleMeter.java from assignment 1, or LixMeter.java from assignment 2, rename it to FleschMeter.java, and then make all the necessary changes. Remember to change comments too.

The class FleschMeter should of course implement interface TextMeter.

The Syllable counter - SyllableCounter.java

While it is very hard to create a perfect algorithm for finding syllables in english words, the following approximation makes a surprisingly good job. It works like this:

1. If the length of the word is three letters or less, return a count of 1.

2.1 If the word ends with "ES" or "ED", remove the last two letters from the string,

2.2 else if the word ends with "LE", leave the string intact,

2.3 else if the word ends with "E", remove the last letter from the string.

3. Count the number of vowel groups in the remaining string. A vowel group is defined as one or more vowels that are adjacent to each other. Return this count as the value of the function.

For example:

Word	Syllable count
"aa"	1
"aeiouy"	1
"monsoon"	2
"avalanche"	3
"kisses"	1
"snapple"	2
"opened"	2

Here is an outline of the SyllableCounter class:

```
public class SyllableCounter {
  ...
  public static int nofSyllables (String s) {
    ...
  }
}
```

Note that since the method is a pure function - it does not need to remember anything between calls - we can create it as a static function. That means we do not need to create instances of class SyllableCounter in order to use it, we can call nofSyllables directly, using the syntax:

```
SyllableCounter.nofSyllables (String)
```

Here are some hints for the implementation of the syllable counter function:

* Convert the string to uppercase before going to work on it. That makes it easier to compare the suffixes.

* Add a method

  ```
  protected static boolean isVowel (char c)
  ```

  to the SyllableCounter class. Even if you only call it from one place in the code, it will make the calling code look cleaner and more understandable. The six vowels are A E I O U Y .

The main program - FleschTest.java

The main program in the third assignment should allow the user to specify the following on the commandline:

```
  -f       to use the Flesch index for measuring text (default)
  -l       (lowercase L) to use the Lix index for measuring text
  -s       to use the SimpleMeter to count words and sentences

  file     the name of a file to measure
```

or

```
  -h       prints a short help on the commandline options and syntax
           and exit the program
```

Metric selection switches and filenames are allowed in any order and sequence. The arguments are read from left to right and apply as they appear. The last metric selection switch is the one that is used for the next file. More than one file can be given. For example:

```
(1)   java FleschTest Foo.txt
```

would measure the file Foo.txt using the Flesch metric, because that is the default metric in this program.

```
(2)   java FleschTest -f Foo.txt
```

is the same as (1).

```
(3)   java FleschTest -s -l Foo.txt Bar.txt
```

measures files Foo.txt and Bar.txt using the Lix metric.

```
(4)   java FleschTest -l Foo.txt -f Bar.txt Saga.txt
```

measures file Foo.txt with the Lix metric, and files Bar.txt and Saga.txt with the Flesch metric.

Start with the source file for the main program from assignment 2, LixTest.java, copy it to the assignment 3 folder and rename it to FleschTest.java. Then make the necessary changes to it.

Some example output:

```
>java FleschTest -h
usage : [{-f|-l|-s}] file ...
-f  Flesch Reading Ease
-l  LIX readability index
-s  Word and sentence counter

>java FleschTest ..\HCAndersen.txt
File: ..\HCAndersen.txt
   syllables :         2583             (corrected /FK)
       words :         1881
      Flesch : 74.76618617842261
   sentences :          124

>java FleschTest -s ..\HCAndersen.txt
File: ..\HCAndersen.txt
       words :         1881
   sentences :          124

>java FleschTest -l ..\HCAndersen.txt
File: ..\HCAndersen.txt
       words :         1881
   sentences :          124
  long words :          337
         lix : 33.08535696523812

>java FleschTest -l ..\HCAndersen.txt -f ..\OskarI.txt ..\MaryWollstonecraft.txt

File: ..\HCAndersen.txt
       words :         1881
   sentences :          124
  long words :          337
         lix : 33.08535696523812
File: ..\OskarI.txt
   syllables :          688
       words :          443
      Flesch : 34.61137143443466
   sentences :           11
File: ..\MaryWollstonecraft.txt
   syllables :          306
       words :          223
      Flesch : 79.4708576233184
   sentences :           20

>
```

Final important points

Do not use packages! We do not need them.

Your folder for assignment 3 should only need these Java source files:
FleschMeter.java, FleschTest.java, and SyllableCounter.java.

Set the CLASSPATH environment variable so that the javac and java commands can find you assignment folders. For example, if your assignment folders are named a1, a2 and a3, then in a Windows environment:

```
SET CLASSPATH=.;..\a1;..\a2;..\a3
```

The first dot includes the current directory, which probably is one of the three folders. The .. syntax means the parent directory to the current directory, and then down again into a1, a2 or a3.

If you do this you will be able to compile and run assignemnts 2 and 3 without having to copy the TextMeter, Parser, and token classes into each folder.

Grades for this assignment:

For the E, D and C grades the general grading criteria apply.

For the C grade it is important that you use the proper javadoc syntax in the source code you write. You must document the class, using the @author tag.

You must document every method, explaining what it is for. If the method takes arguments, use the @param tag. If the method returns values, use the @return tag. If you have programmed the method to throw exceptions, use the @throws tag. Remember that a javadoc comment starts with /** and ends with */, and it goes immediately before that which is commented.

For the B and A grades, your solution fulfills the requirement for a C, and is also able to accumulate meter selectors on the commandline, and apply them all to subsequently listed file(s). This means that if the commandline is

(3) `java FleschTest -s -l Foo.txt Bar.txt`

the program will apply both SimpleMeter and LixMeter to the files Foo.txt and Bar.txt. The commandline

(4) `java FleschTest -l Foo.txt -f Bar.txt Saga.txt`

will apply the Lix metric to Foo.txt, and Lix and Flesch to Bar.txt and Saga.txt.

Some examples of this behaviour:

```
>java FleschTest -l ..\HCAndersen.txt -f ..\OskarI.txt ..\MaryWollstonecraft.txt

File: ..\HCAndersen.txt
       words :         1881
   sentences :          124
  long words :          337
         lix : 33.08535696523812

File: ..\OskarI.txt
       words :          443
   sentences :           11
  long words :          113
         lix : 65.78062794992817
File: ..\OskarI.txt
   syllables :          688
       words :          443
      Flesch : 34.61137143443466
   sentences :           11

File: ..\MaryWollstonecraft.txt
       words :          223
   sentences :           20
  long words :           37
         lix : 27.741928251121074
File: ..\MaryWollstonecraft.txt
   syllables :          306
       words :          223
      Flesch : 79.4708576233184
   sentences :           20

>



B - an unambiguous algorithmic outline
A - you code it, document it, and it works.

-fk
```