ICT/KTH
05-sep-2010/FK

id1006 Java Programming

Assignment 2 - Lix metric and file reading

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.

There is for example the LIX (Sv. "läsbarhetsindex", Eng. "readability index") which is computed thus:

$$lix = \frac{O}{P} + \frac{L * 100}{O}$$

where

O = the number of words
L = the number of long words (longer than six characters)
P = the number of sentences

in the text.

LIX is primarily for swedish texts, but can of course be computed on english too. As can be seen, it is the sum of two values: the average number of words per sentence, and the percentage of long words in the text. This means that long sentences and long words will give a higher value, indicating a text that is harder to read.

One interpretation of the Lix value for the Swedish language is:

```
   - 29     children's books
30 - 36     fiction, a novel
37 - 43         news article
44 - 52         average factual text
53 - 60         advanced factual text
61 -      academic thesis
```

[Source: http://www.teknolingva.fi/webbinarium/S-Asemi2000/sld025.htm]

Here is another interpretation:

```
   - 25     children's books
25 - 30         simple texts
30 - 40         ordinary texts / fiction
40 - 50         factual text
50 - 60         advanced factual
61 -        very adv. factual, research, thesis
```

[Source: http://sv.wikipedia.org/wiki/LIX]

Assignment 2 - LixMeter.java and LixTest.java

The LIX TextMeter - LixMeter.java

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

The second assignment consists of creating two new source files, a TextMeter implementation called LixMeter.java and a new main program called LixTest.java.

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

```
words     the number of words
long words  the number of long words
sentences   the number of sentences
lix         the LIX index
```

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

The class LixMeter should of course implement interface TextMeter.

The main program - LixTest.java

The main program in the second assignment should allow the user to specify a filename on the command line. The program opens the file, reads it, sends each line to the parser, and finally prints the property map of the LixMeter to the standard output stream (System.out).

Start with the source file for the main program from assignment 1, ParserTest.java, copy it to the assignment 2 folder and rename it to LixTest.java. Then make the necessary changes to it.

In order to read the lines of a text file, the following outline is useful:

```
import java.io.BufferedReader;
import java.io.FileReader;
  ...
  String fileName

  BufferedReader in = new BufferedReader (new FileReader (fileName));
  for (String t = in.readLine (); t != null; t = in.readLine ()) {
    ... // do something with the current line
  }
```

Final important points

Do not use packages! We do not need them.

Your folder for assignment 2 should only contain two Java source files, LixMeter.java and LixTest.java. After compilation you should also have LixMeter.class and LixTest.class, and editing may of course leave backup files.

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.

Example output:

```
>java LixTest MaryWollstonecraft.txt
       words :          223
   sentences :           20
  long words :           37
         lix : 27.741928251121074

>java LixTest HCAndersen.txt
       words :         1881
   sentences :          124
  long words :          337
         lix : 33.08535696523812

>java LixTest OskarI.txt
       words :          443
   sentences :           11
  long words :          113
         lix : 65.78062794992817

>
```

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 also accepts more than one filename on the commandline. Each file is processed independent of the other files, and its name and LixMeter properties are printed. Furthermore, if there is a problem when accessing the file, it is reported in the output and the program continues with the next file. For example:

```
>java LixTest MaryWollstonecraft.txt HCAndersen.txt Missing.file OskarI.txt
File: MaryWollstonecraft.txt
       words :          223
   sentences :           20
  long words :           37
         lix : 27.741928251121074
File: HCAndersen.txt
       words :         1881
   sentences :          124
  long words :          337
         lix : 33.08535696523812
File: Missing.file
java.io.FileNotFoundException: Missing.file (The system cannot find the file spe
cified)
File: OskarI.txt
       words :          443
   sentences :           11
  long words :          113
         lix : 65.78062794992817

>
```

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

-fk