Laboratorio di Linguaggi e Programmazione Orientata agli Oggetti

20 marzo 2018

a.a. 2017/2018

1. Provare a implementare la classe `WordCount` che realizza una semplice applicazione eseguibile da linea di comando, in grado di contare il numero di occorrenze delle parole contenute in un testo. Con "parola" si intende una qualsiasi sequenza non vuota di caratteri composta da sole lettere; due parole sono considerate uguali se sono costituite dalla stessa sequenza di lettere, con distinzione tra minuscole e maiuscole (quindi, le parole `test` e `Test` sono considerate diverse).

   Se il file da esaminare contiene il seguene testo

   ```
   The Java programming language is strongly and statically typed. This specification
   clearly distinguishes between the compile-time errors that can and must be detected
   at compile time, and those that occur at run time. Compile time normally consists
   of translating programs into a machine-independent byte code representation.
   Run-time activities include loading and linking of the classes needed to execute
   a program, optional machine code generation and dynamic optimization of the
   program, and actual program execution.
   ```

   allora un possibile output del risultato atteso è

   ```
   {language=1, run=1, clearly=1, program=3, that=2, statically=1, compile=2, optimization=1,
   This=1, dynamic=1, programming=1, between=1, those=1, generation=1, actual=1, occur=1,
   byte=1, linking=1, is=1, optional=1, loading=1, execute=1, at=2, strongly=1, must=1,
   programs=1, errors=1, translating=1, be=1, code=2, independent=1, needed=1, classes=1,
   Compile=1, representation=1, The=1, can=1, into=1, and=6, of=3, Java=1, a=2, include=1,
   execution=1, specification=1, Run=1, the=3, typed=1, machine=2, activities=1,
   detected=1, consists=1, distinguishes=1, time=5, to=1, normally=1}
   ```

   L'applicazione gestisce le due opzioni, specificabili in qualsiasi ordine, `-i` *input* e `-o` *output* che permettono di indicare rispettivamente i path del file di testo da esaminare e del file di testo sul quale salvare i risultati dell'analisi; in assenza di indicazioni specifiche, lo standard input e lo standard output vengono utilizzati come input e output. È ammesso che una stessa opzione venga ripetuta più volte; in tal caso l'applicazione considera come valida l'ultima occorrenza dell'opzione (seguendo l'ordine convenzionale); per esempio, con il comando

   ```
   $ java lab09_03_20.WordCount  -i in1.txt -o out1.txt -o out2.txt -i in2.txt
   ```

   il file `in2.txt` verrà esaminato e il risultato verrà salvato sul file `out2.txt`.

   Suggerimento: per la lettura del testo utilizzare la classe `Scanner` del package `java.util` e i suoi metodi `useDelimiter(String pattern)`, `hasNext()` e `next()`; per memorizzare i dati usare `HashMap<String,Integer>`. Usare il costrutto **`try-with`** per gestire correttamente eccezioni e chiusura degli stream di input e output.

2. Provare a modificare l'applicazione del punto precedente in modo che l'analisi venga visualizzata rispettando l'ordine lessicografico delle parole.

   Suggerimento: usare i metodi `keySet()` di `Map`, `toArray(T[] a)` di `Set` e `sort(Object[] a)` di `Arrays`.