Reti e Laboratorio III
Modulo Laboratorio III
AA. 2023-2024

docente: Laura Ricci

laura.ricci@unipi.it

Correzione Assignment 4
”Conteggio Occorrenze”
02/11/2023

ASSIGNMENT 4: CONTEGGIO OCCORRENZE

- scrivere un programma che conta le occorrenze dei caratteri alfabetici (lettere dalla “A” alla “Z”) in un insieme di file di testo. Il programma prende in input una serie di percorsi di file testuali e per ciascuno di essi conta le occorrenze dei caratteri, ignorando eventuali caratteri non alfabetici (come per esempio le cifre da 0 a 9). Per ogni file, il conteggio viene effettuato da un apposito task e tutti i task attivati vengono gestiti tramite un pool di thread. I task registrano i loro risultati parziali all’interno di una ConcurrentHashMap. Prima di terminare, il programma stampa su un apposito file di output il numero di occorrenze di ogni carattere. Il file di output contiene una riga per ciascun carattere ed è formattato come segue:

  <carattere$_1$>,<numero di occorrenze>

  <carattere$_2$>,<numero di occorrenze>

  ...

  <carattere$_n$>,<numero di occorrenze>

ASSIGNMENT 4: CONTEGGIO OCCORRENZE

- esempio di file di output:

```
a,1281
b,315
c,261
d,302
…
```

- si allega un archivio compresso contenente file testuali per effettuare i test

ASSIGNMENT 4: IL COUNTER

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.text.Normalizer;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

/**
 * Reti e laboratorio III - A.A. 2022/2023
 * Soluzione del quarto assignment
 */
public class Counter implements Runnable {

    public static final int BUFSIZE = 32768;   // Dimensione del buffer
    private Map<Character, Integer> counters;  // Riferimento alla hash map globale
    private Map<Character, Integer> localCounters;      // Hash map locale
    private File file;                          // Riferimento al file di input.
```

ASSIGNMENT 4: IL COUNTER

```
/**
 *  Costruttore della classe "Counter"
 *  che inizializza le variabili
 */

public Counter(Map<Character, Integer> counters, File file) {
        this.counters = counters;
        this.localCounters = new HashMap<>();
        this.file = file;
}
```

ASSIGNMENT 4: IL COUNTER

```
public void run() {
// Apro il file di input per la lettura.
    try (BufferedReader in = new BufferedReader(new FileReader(file), BUFSIZE)) {
        int current = -1;
        while ((current = in.read()) != -1) {
            char key = (char) current;
            // Ignoriamo tutti i caratteri che non sono alfabetici ('a'-'z').
            if (!Character.isAlphabetic(key)) continue;
            // Normalizziamo il carattere rimuovendo eventuali segni diacritici.
            key = normalize(Character.toLowerCase(key));
            // Se il carattere è alfabetico, aggiorno atomicamente la hash map.
            localCounters.compute(key, (k, v) -> ((v == null) ? 1 : v + 1));
        }
```

ASSIGNMENT 4: IL COUNTER

```
            // Scrivo i risultati nella hash map globale.
            for (Entry<Character, Integer> entry : localCounters.entrySet()) {
                char key = entry.getKey();
                int count = entry.getValue();
                counters.compute(key, (k, v) -> ((v == null)? count : v+count));
            }
        } catch (Exception e) {
            System.err.printf("Error while reading file %s\n", file.getName());
            e.printStackTrace();
        }
    }
```

ASSIGNMENT 4: NORMALIZZAZIONE CARATTERI

```java
/**
 * Metodo per trasformare un carattere con segno diacritico
 * nel suo equivalente privo di segno.
 * @param x il carattere di input
 * @return il carattere normalizzato
 */

public static char normalize(char x) {
    String s = Normalizer.normalize(
        String.valueOf(x),
        Normalizer.Form.NFKD).replaceAll("\\p{M}", "");

    return s.charAt(0);
}
```

ASSIGNMENT 4: IL MAIN

```
import java.io.*;
import java.util.Map.Entry;
import java.util.concurrent.*;

/**
 * Reti e laboratorio III - A.A. 2022/2023
 * Soluzione del quarto assignment
 */

public class Main {
    public static final int N_THREADS = 8; // Numero di thread del pool.
    // Pool di thread (di dimensione fissa).
    public static ExecutorService pool = Executors.newFixedThreadPool(N_THREADS);
    // Tempo massimo di attesa per la terminazione del pool di thread.
    public static final long poolTerminationDelay = 10000;
    // ConcurrentHashMap usata per registrare i contatori delle occorrenze.
    public static ConcurrentHashMap<Character, Integer> counters =
        new ConcurrentHashMap<>();
```

ASSIGNMENT 4: IL MAIN

```
public static void main(String[] args) {
// Ho bisogno di almeno due parametri di input: almeno un path di file di
   input e il path file di output.
    if (args.length < 2) {
        System.err.println(«Needeed:<inputFile1> ... <inputFileN> <outputFile>");
        System.exit(1);
    }
    // Per ogni file in input...
    for (int i = 0; i < args.length - 1; i++) {
        String inputFile = args[i];
        // Apro il file e controllo che esista.
        File file = new File(inputFile);
        if (!file.exists() || file.isDirectory()) {
            System.err.printf("Error: %s is not a valid file!\n", inputFile);
            continue;
        }
        // Creo un nuovo runnable e lo invio al pool.
        pool.submit(new Counter(counters, file));
    }
```

ASSIGNMENT 4: TERMINAZIONE THREADPOOL

```
// Avvio la terminazione del pool.
pool.shutdown();
try {
    if (!pool.awaitTermination(poolTerminationDelay, TimeUnit.MILLISECONDS))
        pool.shutdownNow();
} catch (Exception e) {
    System.err.printf("Error: could not close thread pool!\n");
    System.exit(1);
}
```

ASSIGNMENT 4: SCRITTURA RISULTATI

```
        // Infine, scriviamo i risultati sul file di output.
        String outputFile = args[args.length-1];
        try (PrintWriter out = new PrintWriter(outputFile)) {
            // Per farlo, iteriamo su tutte le entry della ConcurrentHashMap.
            for (Entry<Character, Integer> entry : counters.entrySet()) {
                out.printf("%c,%d\n", entry.getKey(), entry.getValue());
            }
        } catch (Exception e) {
            System.err.printf("Error: could not write output file!\n");
            System.exit(1);
        }
    }
}
```