Reti e Laboratorio III
Modulo Laboratorio III
A.A. 2025-2026

docente: Laura Ricci

laura.ricci@unipi.it

Correzione Assignment 4
”Conteggio Occorrenze”
31/10/2025

ASSIGNMENT 4: CONTEGGIO OCCCORRENZE

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:

```
  <carattere1>,<numero di occorrenze>
  <carattere2>,<numero di occorrenze>
  <carattere3>,<numero di occorrenze>
… <carattereN>,<numero di occorrenze>
```

ASSIGNMENT 4 : CONTEGGIO OCCORRENZE

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.text.Normalizer;
import java.util.HashMap;
import java.util.Map;
/**
 *     @author Matteo Loporchio
 */
public class Counter implements Runnable {
   // Dimensione del buffer per la lettura del file di input.
   public static final int BUFSIZE = 32768;
   // Riferimento alla hash map globale per memorizzare i risultati.
   private Map<Character, Integer> counters;
   // Hash map locale per memorizzare i risultati.
   private Map<Character, Integer> localCounters;
   // Riferimento al file di input.
   private File file;
public Counter(Map<Character, Integer> counters, File file) {
    this.counters = counters;
    this.localCounters = new HashMap<>();
    this.file = file; }
```

ASSIGNMENT 4 : CONTEGGIO OCCORRENZE

```
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 e' alfabetico, aggiorno atomicamente la hash map.
       localCounters.compute(key, (k, v) -> ((v == null) ? 1 : v + 1));
       }
     // Scrivo i risultati nella hash map globale.
    for (var 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 : CONTEGGIO OCCORRENZE

```
/**
 *    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 METODO COMPUTE

- serve per aggiungere o aggiornare un valore in una mappa (`HashMap`) in modo intelligente
- senza dover scrivere tanti `if`.
- come funziona
  - indicare quale chiave si vuole modificare (es. "`Bitcoin`")
  - `Java` controlla se quella chiave esiste già nella mappa
  - restituisce il valore associato, oppure null se non c'è, in una funzione
  - la funzione decide che nuovo valore mettere per quella chiave
  - se la funzione restituisce un valore, viene salvato nella mappa
  - se invece restituisce null, quella chiave viene cancellata
- opera in modo thread safe

ASSIGNMENT 4 : CONTEGGIO OCCORRENZE

```
import java.io.*;
import java.util.concurrent.*;
/**
 * @author Matteo Loporchio
 */
public class Main {
// Numero di thread del pool.
public static final int N_THREADS = 8;
// 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<>();
public static void main(String[] args) {
  // Ho bisogno di almeno due parametri di input.
  if (args.length < 2) {
      System.err.println("Usage: Main <inputFile1> ... <inputFileN> <outputFile>");
      System.exit(1); }
```

ASSIGNMENT 4 : CONTEGGIO OCCORRENZE

```
// 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));
    }
// 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 : CONTEGGIO OCCORRENZE

```
// Prima di terminare, 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 (var 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);
      }
    }
   }
  }
 }
}
```