RELAZIONE ASSIGNMENT 2 PCD (Regexp Matching Tool)

**Semprini Luca 0000854447**

**Riciputi Jacopo \***

\*\*\*DA SEPARARE LE DUE RELAZIONI\*\*\*

La richiesta di questo assignment consisteva nell’implementazione di tre diverse versioni di un Regexp Matching Tool, risolvendo il problema utilizzando differenti modelli implementativi, rispettivamente:

* **Esercizio 1**: Utilizzare il modello a *tasks* ed *executors*.
* **Esercizio 2**: Utilizzare la programmazione asincrona ed *event-loop*.
* **Esercizio 3**: Utilizzare i *reactive* *streams*.

Per quanto riguarda la parte comune ai tre esercizi, si è scelto di utilizzare un’interfaccia grafica costruita tramite JavaFX e di fare riuso di classi di modello come **Document** e **Folder**, e di alcune classi di utility. Il software è stato progettato per ricercare, all’interno di un percorso di File System, tutti i file testuali contenenti *parole* che iniziano con la lettera “i” o “n” (espressione regolare: “**[n, i][a-z]\***”).

La GUI è stata progettata in modo da fornire la possibilità all’utente di inserire un percorso (relativo o assoluto) di file system e specificare la profondità massima della ricerca, inoltre, è possibile specificare il numero dell’esercizio che si desidera eseguire, semplicemente selezionandolo da una combo-box. Una volta avviata la ricerca, la ListView verrà riempita in maniera reattiva mano a mano che verrà trovato un file con almeno una corrispondenza, così come i valori della percentuale dei file con almeno una occorrenza e della media delle occorrenze all’interno dei file con almeno un “match”, resi visibili tramite due Label.

**ESERCIZIO 1**

La versione task-oriented dell’assignement è stata progettata utilizzando un executor globale (ForkJoinPool) all’interno della classe **WordCounter**: l’executor invoca il task di ricerca sulle Folder a partire dal percorso indicato dall’utente. Il task di ricerca sulle cartelle (**FolderSearchTask**) è organizzato in modo da invocare dei fork ogni qualvolta vengano identificati subfolders (invoca dunque un nuovo FolderSearchTask) o documenti contenuti nella cartella stessa (invocazione di **DocumentSearchTask**), tenendo ovviamente traccia della lista di tutti i task chiamati; entrambi i task presentano una Map<String, Long> come type-argument, in modo da poter restituire, per ogni Documento trovato, una entry contenente il nome del documento (rappresentato dal suo percorso) e il numero di parole trovate che corrispondano all’espressione regolare. Infine, una volta raggiunta la profondità massima specificata dall’utente, la classe FolderSearchTask utilizza l’operazione di **join** su tutti i task tracciati nella lista ed aggiunge i valori di ritorno ad una mappa (assicurandosi di non creare entry ripetute) che associa ad ogni documento trovato la quantità di occorrenze trovate per esso.

Per rendere l’aggiornamento della GUI reattivo come richiesto dalle specifiche, si è pensato di introdurre un evento ed un relativo Listener (**FileFoundEvent** e **FileEventListener**), che si occupassero di notificare alla view l’avvenuto ritrovamento di un nuovo Document da parte dei task: la classe WordCounter, all’interno del metodo che effettua il conteggio delle parole all’interno di un documento, notifica una lista di FileEventListener aggiungendo un evento contenente tutte le informazioni necessarie per aggiornare correttamente l’interfaccia grafica (numero di file totali trovati al momento, numero di file con almeno una corrispondenza e documento corrente corrispondente all’evento).

**ESERCIZIO 2**

Basandosi sugli event-loop creati da Vert.x, l’esercizio due viene avviato grazie ad un wrapper della classe WordCounter, che ne estende le funzionalità tramite l’implementazione di un metodo per avviare la ricerca eseguendo classi che estendono **AbstractVerticle**. Il metodo in questione prende in ingresso la directory di partenza, la regular expression, la profondità e una **FutureOperation**, una classe creata appositamente per sfruttare il pattern funzionale di Java, che verrà fornita alle successive classi che si occupano della ricerca dei documenti e del conteggio delle occorrenza.La chiamata al metodo countOccurencesInEventLoop esegue il deploy della classe **VerticleFolderSearch**, un verticle che scansiona la cartella fornita in ingresso e crea un nuovo Verticle per ogni sotto-cartella e un **VertileDocumentSearch** per ogni documento presente.

Le operazioni terminano proprio su quest’ultima classe, che, al momento della creazione lancia il conteggio nel metodo Start e scatena il completamento della Future alla quale è stata associata l’operazione di aggiornamento dell’interfaccia grafica.

Per l’implementazione del secondo esercizio si è cercato di riutilizzare il più possile il codice della versione a Task, mantendo invariato lo scheletro delle chiamate per avviare la ricerca su tutti i documenti e unificando l’aggiornamento dell’interfaccia grafica.

**ESERCIZIO 3**

Similarmente a quanto avviene nell’esercizio 2 la ricerca delle occorrenze della regular expression viene avviata da un wrapper di WordCounter che implementa il metodo **countOccurencesReactively**, il quale oltre a riceve in ingresso la directory di partenza, la regexp e la profondità, richiede un **Observer<Map<String, Long>** che verrà fornito alla successive classi, così da notificare il termine del conteggio delle parole in un documento tramite il metodo onNext(). L’observer infatti, al momento della crezione, è stato istruito per eseguire l’aggiornamento della GUI ad ogni chiamata su onNext().

La sequenza della ricerca rimane identica a quella dell’esercizio due, con l’unica differenza, come anticipato, che al termine del conteggio delle occorrenze all’interno di un file viene scatenato il metodo onNext().

In termini prestazionali l’esercizio 3 risulta nettamente più lento dei precedenti data l’assenza di concorrenza.

NOTE IMPLEMENTATIVE:

* Si è scelto di utilizzare una **Thread.sleep()** di 100 millisecondi all’interno di WordCounter ad ogni documento trovato dal programma, per permettere di apprezzare più facilmente ad occhio nudo l’aggiornamento reattivo dell’interfaccia grafica.
* Sono stati aggiunti al progetto diverse cartelle e file testuali di prova.
* Come analisi delle performance dei tre esercizi si è scelto di stampare su un file “performance.txt” l’output del programma al lancio di ogni versione, monitorando il tempo impiegato alla fine della ricerca (tenendo ovviamente conto della pausa di 100ms ad ogni documento trovato).

LINK AL REPOSITORY: https://github.com/lucasemprini/Assignment2PCD