03FYZ TECNICHE DI PROGRAMMAZIONE

Istruzioni per effettuare il fork di un repository GitHub

- Effettuare il login su GitHub utilizzando il proprio username e password.
- Aprire il repository su GitHub relativo al terzo laboratorio: https://github.com/TdP-2020/Lab03
- Utilizzare il pulsante *Fork* in alto a destra per creare una propria copia del progetto. L'azione di Fork crea un nuovo repository nel proprio account GitHub con una copia dei file necessari per l'esecuzione del laboratorio.
- Aprire Eclipse, andare su *File -> Import*. Digitare *Git* e selezionare *Projects from Git -> Next -> Clone URI -> Next*.
- Utilizzare la URL del **proprio** repository che si vuole clonare (**non** quello in TdP-2020!), ad esempio:

  https://github.com/my-github-username/Lab03
- Fare click su *Next*. Selezionare il branch (*master* è quello di default) fare click su *Next*.
- Selezionare la cartella di destinazione (quella proposta va bene), fare click su *Next*.
- Selezionare *Import existing Eclipse projects*, fare click su *Next* e successivamente su *Finish*.
- Il nuovo progetto Eclipse è stato clonato ed è possibile iniziare a lavorare.
- A fine lavoro ricordarsi di effettuare Git commit e push, utilizzando il menù *Team in Eclipse*.

ATTENZIONE: solo se si effettua Git **commit** e successivamente Git **push** le modifiche locali saranno propagate sui server GitHub e saranno quindi accessibili da altri PC e dagli utenti che ne hanno visibilità.

03FYZ TECNICHE DI PROGRAMMAZIONE

Esercitazione di Laboratorio 25 marzo 2020

Obiettivi dell'esercitazione:

- Utilizzo del pattern MVC
- Creazione di una struttura dati
- Introduzione alla complessità

Esercizio 1

Dopo aver fatto il fork del progetto relativo al laboratorio 3, creare in Java una semplice applicazione dotata di interfaccia grafica che funga da correttore ortografico di parole: dato un testo in input, il programma stampa le parole errate, il numero di errori, e il tempo impiegato per effettuare il controllo ortografico. L'interfaccia grafica dell'applicazione deve rispecchiare quella riportata in Figura 1.

In particolare, l'utente inserisce un testo che vuole verificare nella casella di testo in alto. Dopo aver selezionato la lingua da utilizzare, fa click sul bottone *Spell Check* per avviare il controllo ortografico. Si suggerisce di filtrare il testo ricevuto in input trasformandolo tutto in minuscolo ed eliminando i segni di punteggiatura. Nell'area di testo sottostante, viene restituito l'elenco delle parole errate. Utilizzare il pulsante *Clear Text* per cancellare il testo inserito da entrambe le caselle di testo.

Nota1: per eliminare tutti i segni di punteggiature utilizzare il metodo

```
thisIsAString.replaceAll("[.,\\/#!$%\\^&\\*;:{}=\\-_`~()\\[\\]\"]", "")
```

Nota2: sviluppare l'applicazione secondo il pattern **Model View Controller** (MVC)

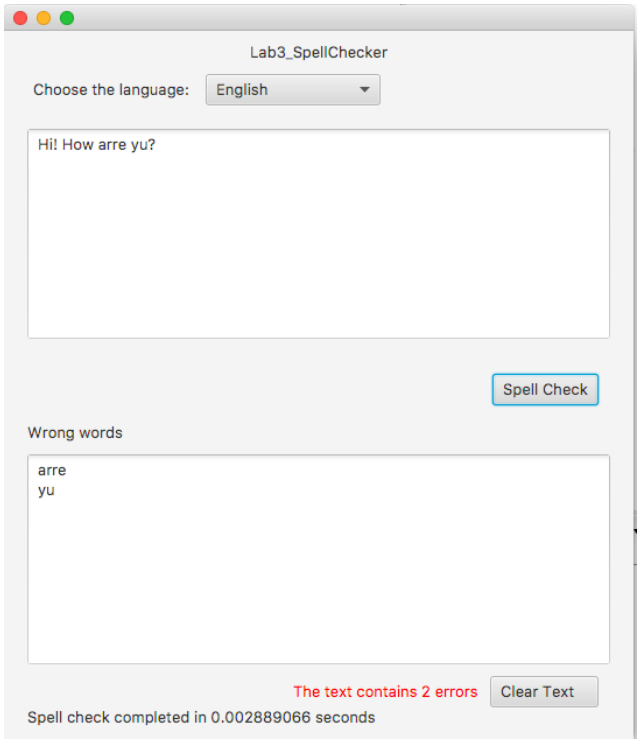

Figura 1.

Di seguito, una possibile traccia per la soluzione:

1. Creare l'interfaccia grafica con *SceneBuilder*. Associare al bottone *Spell Check* il metodo `doSpellCheck()` ed al bottone *Clear Text* il metodo `doClearText()`.

2. Creare il package `it.polito.tdp.spellchecker.model` ed una nuova classe (Java Bean) denominata `RichWord`. Ogni istanza di questa classe conterrà una parola del testo in input, e l'indicazione se tale parola è corretta o meno (utilizzare un `boolean`).

3. Definire nel package `it.polito.tdp.spellchecker.model` una nuova classe `Dictionary`, il **modello** dell'applicazione, in cui definire i seguenti metodi:

   ```
   public void loadDictionary(String language)
   ```

   Questo metodo permette di caricare in memoria il dizionario della lingua desiderata. A questo proposito, utilizzare i file `Italian.txt` e `English.txt` nella cartella `src/main/resources`. I file contengono una parola per riga. Salvare le parole del dizionario in una struttura dati appropriata.

   Di seguito viene riportato un esempio di codice per leggere le parole da file:

   ```
           try {
           FileReader fr = new FileReader("English.txt");
                  BufferedReader br = new BufferedReader(fr);
           String word;
           while ((word = br.readLine()) != null) {
                        // Aggiungere parola alla struttura dati
           }
                  br.close();

    } catch (IOException e){
          System.out.println("Errore nella lettura del file");
           }
   ```

   ```
   public List<RichWord> spellCheckText(List<String> inputTextList)
   ```

   Questo metodo esegue il controllo ortografico sul testo in input (rappresentato da una lista di parole) e restituisce una lista di `RichWord`. Per ogni elemento di `inputTextList`, `spellCheckText` controlla se la parola è presente nel dizionario. In caso affermativo, la `RichWord` corrispondente sarà corretta, altrimenti sarà errata. La lista delle `RichWord` viene restituita in output.

4. Implementare nel **model** tutta la logica dello spell checking, il **controller** si limita alla gestione dell'interfaccia e richiama i metodi definiti nel **model**.

Esercizio 2

Modificare l'algoritmo di ricerca del metodo `spellCheckText` implementando una ricerca lineare ed una dicotomica (vedere spiegazione nella pagina seguente). Si consiglia di creare nel model due nuovi metodi che sostituiscono `spellCheckText`: `spellCheckTextLinear` e `spellCheckTextDichotomic`, dove il primo utilizza una ricerca lineare, mentre il secondo quella dicotomica.

Confrontare le differenze di prestazioni tra le due implementazioni utilizzando sia un `ArrayList` ed una `LinkedList`. Riempire la tabella nella pagina seguente con i tempi di esecuzione per ciascun caso.
Quale implementazione utilizza il metodo `contains` di Java `List`?

Ricerca lineare (source Wikipedia):

Iterare su tutti gli elementi del vocabolario a partire dal primo. La ricerca termina quando viene trovato l'elemento cercato o si raggiunge l'ultimo, nel caso in cui l'elemento cercato non sia presente nella lista.

Ricerca dicotomica (source Wikipedia):

Sapendo che il vocabolario è ordinato alfabeticamente, l'idea è quella di non iniziare la ricerca dal primo elemento, ma da quello centrale, cioè a metà del dizionario. Si confronta questo elemento con quello cercato:

- Se corrisponde, la ricerca termina indicando che l'elemento è stato trovato
- se è superiore, la ricerca viene ripetuta sugli elementi precedenti (ovvero sulla prima metà del dizionario), scartando quelli successivi
- se è inferiore, la ricerca viene ripetuta sugli elementi successivi (ovvero sulla seconda metà del dizionario), scartando quelli precedenti.

Il procedimento viene ripetuto iterativamente fino a quando o si trova l'elemento cercato, o tutti gli elementi vengono scartati. In quest'ultimo caso la ricerca termina indicando che il valore non è stato trovato.

TABELLA CONFRONTO PRESTAZIONI

	ArrayList	LinkedList
list.contains()		
Linear search	0.0594625 secondi	0.1094834 secondi
Dichotomic search	0.001662 secondi	1.5526185 secondi