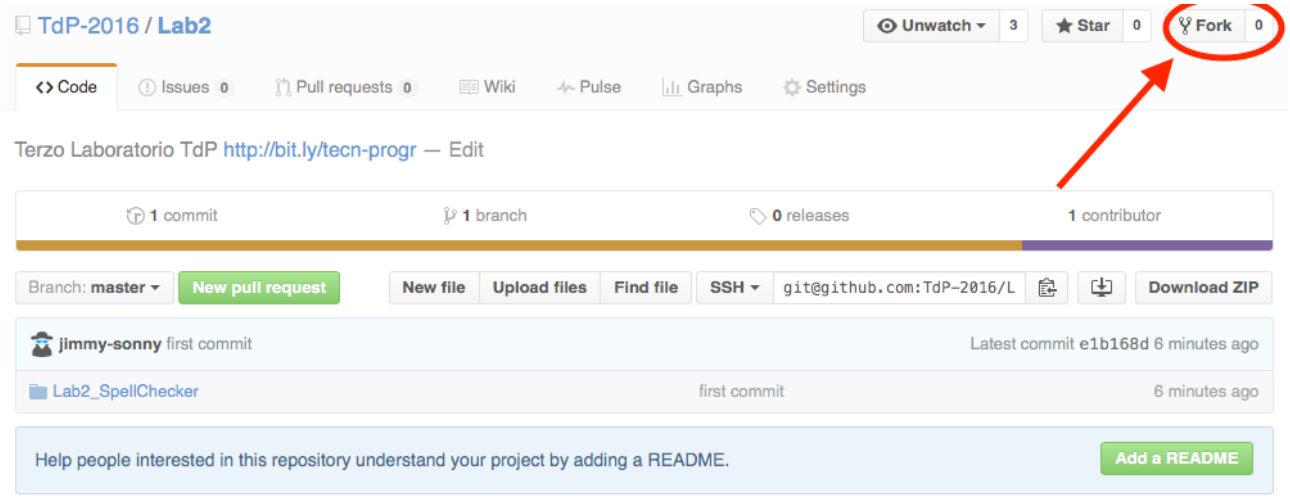
03FYZ TECNICHE DI PROGRAMMAZIONE
Istruzioni per effettuare il fork di un repository GitHub

- Effettuare il login su GitHub utilizzando il proprio username e password.
- Aprire il repository su GitHub relativo al secondo laboratorio:

  https://github.com/TdP-2016/Lab2
- 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 accout 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-2016!), ad esempio:

  https://github.com/my-github-username/Lab2
- 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 02 – 16 Marzo 2016

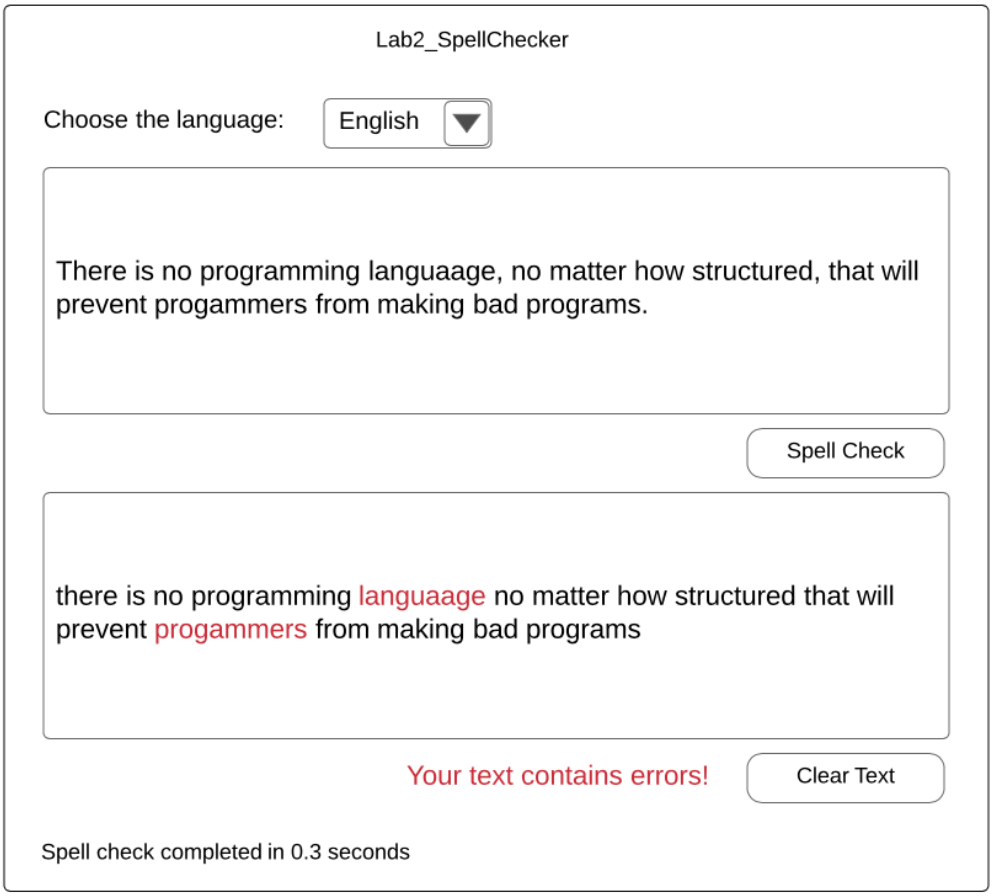
Obiettivi dell'esercitazione:

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

Scopo dell'esercitazione: Realizzare in linguaggio Java una semplice applicazione dotata di interfaccia grafica che funga da correttore ortografico di parole: dato un testo in input, il programma evidenzia le parole che presentano errori ortografici.

Funzionamento previsto per la versione finale (completa) dell'applicazione: 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. Nell'area di testo sottostante, viene restituito il testo con le parole sbagliate evidenziate in rosso.
Il testo in input ammette segni di punteggiatura, ma questi dovranno essere filtrati durante il controllo ortografico. Si suggerisce di convertire tutto il testo ricevuto in minuscolo prima di elaborarlo. Utilizzare il pulsante *Clear Text* per cancellare il testo inserito da entrambe le caselle di testo. Al fondo della finestra deve essere visualizzato il tempo necessario per effettuare il controllo ortografico.

Realizare l'interfaccia dell'applicazione secondo il **mockup** presentato di seguito:



Esercizio 1.1 Utilizzando la libreria *javafx* ed il tool *SceneBuilder*, creare un'interfaccia grafica simile a quella sopra mostrata. Associare al bottone *Spell Check* il metodo *doSpellCheck()* ed al bottone *Clear Text* il metodo *doClearText()*. In questa versione iniziale, utilizzare una *TextArea* per entramble le aree di Testo. Nella area di testo sottostante, per semplicità stampare le sole parole errate (in rosso) e non stampare nessuna delle parole corrette né della punteggiatura eventualmente presente.

Esercizio 1.3 Creare il package *it.polito.tdp.spellchecker.model* ed una nuova classe (Java Bean) denominata *RichWord*. Quest'ultima viene utilizzata per tenere traccia delle parole errate: definire una *stringa* (dove verrà memorizzata una parola) ed un *boolean* per indicare se la parola memorizzata è corretta o meno.

Esercizio 1.4 Definire nel package *it.polito.tdp.spellchecker.model* una nuova classe *Dictionary*, che assume il ruolo di *model*, in cui definire i seguenti metodi.

```
public String loadDictionary()
```
Questo metodo non va implementato in questa classe. (Vedere esercizio 1.5).

```
public List<RichWord> spellCheckText(List<String> inputTextList)
```
Implementare il metodo per poter eseguire il controllo ortografico del testo in input.

Esercizio 1.5 Definire nel package *it.polito.tdp.spellchecker.model* le due classi *ItalianDictionary* ed *EnglishDictionary* che estendono la classe *Dictionary*. Implementare in entrambe il metodo per la lettura del dizionario corrispondente.

```
@Override
public String loadDictionary()
```

I file *Italian.txt* e *English.txt*, con i dizionari delle lingue corrispondenti si trovano nella cartella *rsc*. I dizionari contentengono una parola per riga. Salvare le parole del dizionario in una struttura dati appropriata. Di seguito viene riportato un esempio della sequenza di operazioni necessarie per leggere le parole dal file:

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

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

Esercizio 1.5 Definire nel package *it.polito.tdp.spellchecker.model* una classe *TestModel* dotata di metodo *main* per effettuare alcuni test sul funzionamento del *model*. In seguito collegare il *controller* al model e verificare l'implementazione.

Esercizio 1.6 Visualizzare il tempo (in secondi) impiegato per effettuare il controllo ortografico del testo in input. Sperimentare diversi algoritmi per la ricerca delle singole parole all'interno del dizionario. Ad esempio utilizzare prima il metodo `contains` della struttura dati ed in seguito provare ad implementare una ricerca dicotomica (vedi spiegazione nella pagina seguente).

Esercizio 1.7 Sostituire la seconda *TextArea* con un nodo di tipo *TextFlow*. Visualizzare in output tutto il testo con le parole errate evidenziate in rosso.

Ricerca dicotomica (from Wikipedia):

Sapendo che il vocabolario è ordinato alfabeticamente, l'idea è quella di iniziare la ricerca non 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 invece è inferiore, la ricerca viene ripetuta sugli elementi successivi (ovvero sulla seconda metà del dizionario), scartando quelli precedenti.

Se si arriva al punto che tutti gli elementi vengono scartati, la ricerca termina indicando che il valore non è stato trovato.