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 quinto laboratorio:

  https://github.com/TdP-2019/Lab05
- 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-2019!), ad esempio:

  https://github.com/my-github-username/Lab5
- 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 3 aprile 2019

Obiettivi dell'esercitazione:

- Apprendere il meccanismo della ricorsione
- Utilizzo del pattern MVC e DAO
- Utilizzo di JDBC

ESERCIZIO 1

Analizzare su CARTA l'algoritmo ricorsivo per trovare tutti gli anagrammi di una parola. In particolare, l'algoritmo, dato una parola in input, deve generare in output tutte le permutazioni di tale parola. Una permutazione è una qualsiasi disposizione delle lettere contenute all'interno di una parola. Ad esempio, la parola "eat" ha sei permutazioni (compresa la parola stessa): "eat", "eta", "aet", "ate", "tea", "tae".

```
// Struttura di un algoritmo ricorsivo generico

void recursive (... , level) {

  // E -- sequenza di istruzioni che vengono eseguite sempre
  // Da usare solo in casi rari (es. Ruzzle)
  doAlways();

  // A
  if (condizione di terminazione) {
    doSomething;
    return;
  }

  // Potrebbe essere anche un while ()
  for () {

    // B
    generaNuovaSoluzioneParziale;


    if (filtro) { // C
      recursive (..., level + 1);
    }

    // D
    backtracking;
  }

}
```

Figura 1 Struttura base algoritmo ricorsivo

Rispondere alle seguenti domande:

- Cosa rappresenta il "livello" nel mio algoritmo ricorsivo?
- Com'è fatta una soluzione parziale?
- Come faccio a riconoscere se una soluzione parziale è anche completa?
- Data una soluzione parziale, come faccio a sapere se è valida o se non è valida?
  (nb. magari non posso)
- Data una soluzione completa, come faccio a sapere se è valida o se non è valida?
- Qual è la regola per generare tutte le soluzioni del livello+1 a partire da una soluzione parziale del livello corrente?
- Qual è la struttura dati per memorizzare una soluzione (parziale o completa)?
- Qual è la struttura dati per memorizzare lo stato della ricerca (della ricorsione)?
- Sulla base dello schema presentato in *Fig. 1*, completare i blocchi (alcuni potrebbero essere non

necessari)

- **A** – Condizione di terminazione
- **B** – Generazione di una nuova soluzione
- **C** – Filtro sulla chiamata ricorsiva
- **D** – Backtracking
- **E** – Sequenza di istruzioni da eseguire sempre

ESERCIZIO 2

Dopo aver fatto il *fork* del progetto relativo a questo laboratorio, realizzare in linguaggio Java un'applicazione dotata di interfaccia grafica (simile a quella presentata in Figura 2) che calcoli tutti gli anagrammi di una parola con l'algoritmo ricorsivo definito nell'esercizio 1. Inoltre, per ogni permutazione della parola, il programma deve controllare se l'anagramma sia valido o meno, controllando la sua esistenza nel database dizionario (file *mark.sql*, scaricabile dal sito del corso, sezione Materiale-Data-Sets-Dizionario-formato sql). Come riportato in figura, un anagramma valido va mostrato nella prima area di testo, mentre uno non valido va mostrato nella seconda area di testo.

L'applicazione va sviluppata seguendo il pattern MVC e il pattern DAO per l'accesso al dizionario.

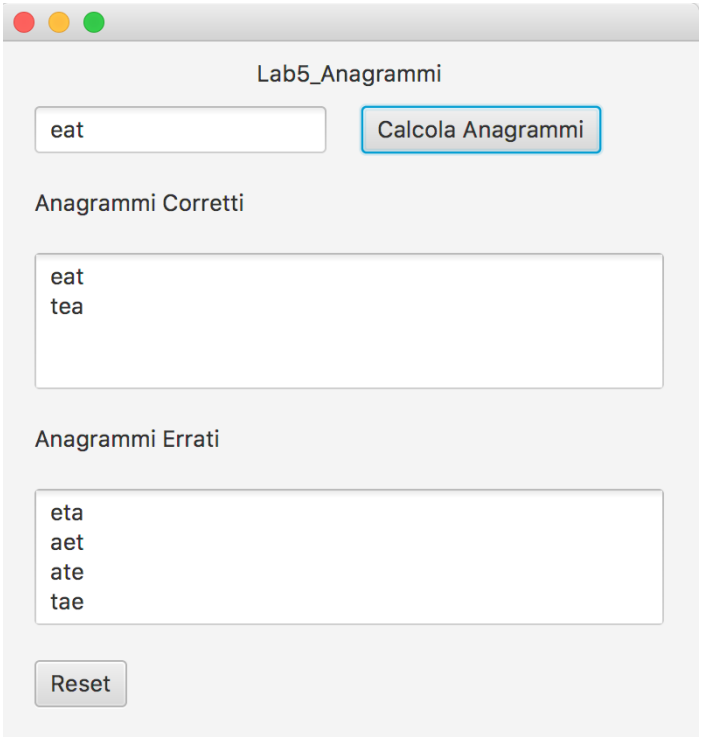

Figura 2 Interfaccia grafica per il calcolo degli anagrammi

Di seguito, una possibile traccia per la soluzione:

1. Realizzare un'interfaccia grafica con *JavaFx* simile al mockup mostrato in Fig 1. Il pulsante *Reset* permette di cancellare il contenuto di tutti i campi.
2. Nel modello dell'applicazione, implementare l'algoritmo ricorsivo per il calcolo delle permutazioni di una parola, e testarlo stampando tutti gli anagrammi in una delle due aree di testo.
3. Importare il database *mark.sql*, da scaricare dal sito del corso, sezione Materiale-Data-Sets-Dizionario-formato sql. Per fare questo, avviare il database locale *XAMPP*. Successivamente lanciare il programma *HeidiSQL* e selezionare l'opzione *File -> Carica file SQL*. Selezionare il file *mark.sql*, eseguire la query direttamente (senza caricare nell'editor i dati) per importarlo e cliccare

sul tasto *Aggiorna*, per permettere la corretta visualizzazione all'interno dell'elenco del database appena caricato.

4. Seguendo il pattern DAO, creare la classe *AnagrammaDAO*, che incapsula le operazioni sul database. In particolare, definire in *AnagrammaDAO* il metodo

```
public boolean isCorrect(String anagramma)
```

che tramite una query SQL permette di verificare se l'anagramma calcolato è presente nel dizionario.