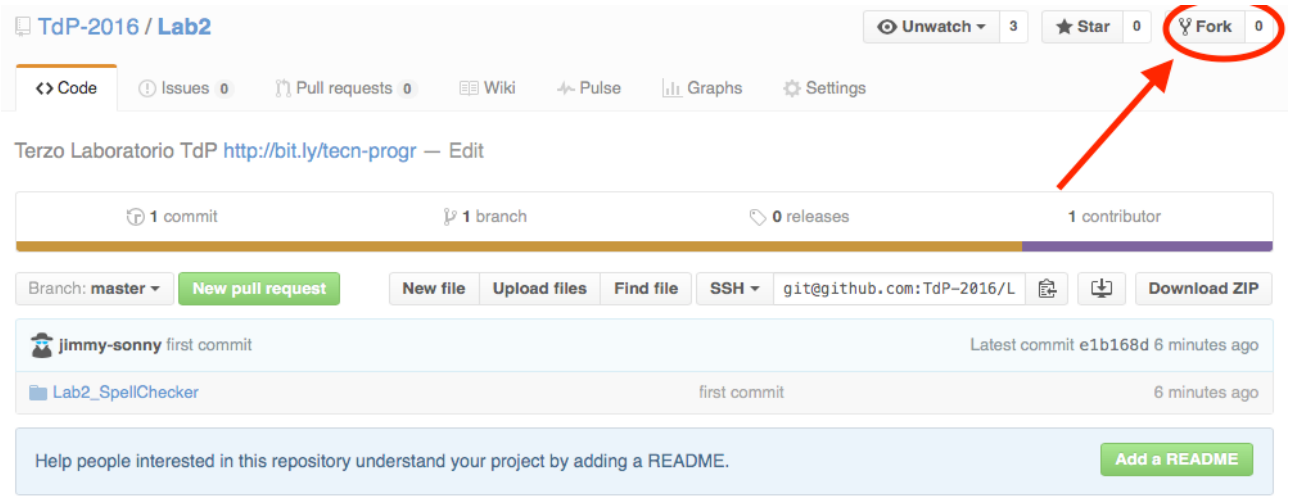
I03FYZ TECNICHE DI PROGRAMMAZIONE

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 all'undicesimo laboratorio:

  https://github.com/TdP-2016/Lab10-Emergency
- 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/Lab10-Emergency
- 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 10 – 18 maggio 2016

Obiettivi dell'esercitazione:

- Simulazioni

ESERCIZIO 1:

Scopo dell'esercitazione: Prendendo in considerazione il codice relativo al progetto "*Emergency*" visto a lezione, apportare le seguenti modifiche:

Esercizio 1.1 Utilizzando il pattern DAO, accedere ai dati presenti nel database **emergency.sql** [github/DB/emergency.sql].

Esercizio 1.2 Considerare che un dottore possa lavorare al massimo per 8 ore al giorno. Per testare l'effettiva differenza delle statistiche, provare a far iniziare il turno dei dottori ognuno con 2 ore di ritardo rispetto all'altro, in modo tale da coprire i periodi di pausa.

SUGGERIMENTI:
Inserire due nuovi eventi "*DOCTOR_INIZIA_TURNO*" e "*DOCTOR_FINE_TURNO*". Il primo dottore inizierà il turno nel momento in cui arriva il primo paziente. Schedulare all'interno dell'evento "*DOCTOR_INIZIA_TURNO*" il momento in cui il dottore dovrà terminare il proprio turno lavorativo (8 ore dopo l'evento di inizio turno) e schedulare all'interno dell'evento "*DOCTOR_FINE_TURNO*" il momento in cui lo stesso dottore dovrà riprendere servizio (quindi 16 ore dopo l'evento di fine turno).

Esercizio 1.3 Realizzare un'interfaccia grafica con *JavaFX* contenente un campo di testo per poter inserire manualmente il nome di un dottore, un campo di testo per inserire le ore di sfalsamento di inizio servizio rispetto all'inizio della simulazione, un bottone per aggiungere il dottore alla lista dei dottori (sarà quindi possibile inserire *n* dottori) e un altro bottone per lanciare la simulazione. Al completamento della simulazione, visualizzare le statistiche finale in una *TextArea*.

Esercizio 1.4 Inserire la figura degli "*Assistenti*", che saranno in grado di curare tutti i pazienti tranne quelli relativi alle emergenze in codice Rosso. Gli assisteni seguono gli stessi orari di lavoro dei dottori (8 ore di lavoro e 16 ore di pausa). Provare ad inserire 3 assistenti che lavorino su 3 turni, in modo tale che sia sempre presente a lavoro un solo assistente.

Esercizio 1.5 Modificare il codice in modo tale che, nell'eventualità in cui si verifichi la morte sotto i ferri di un paziente, il medico (o l'assistente) torni immediatamente libero in occasione di questo evento (in modo da poter curare altri pazienti) senza attendere l'arrivo dell'evento "*DOCTOR_AVAILABLE*", schedulato precedentemente per indicare la fine del tempo necessario per curare il paziente.

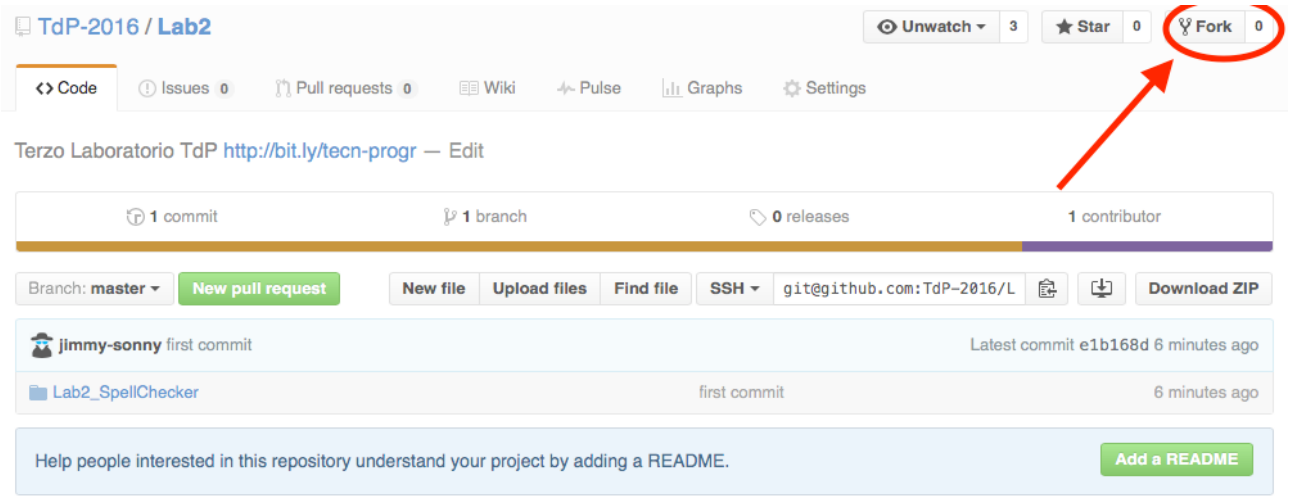
SUGGERIMENTI:
Dato che gli eventi già schedulati non possono essere cancellati, non è possibile eliminare l'evento "*DOCTOR_AVAILABLE*". In alternativa è possibile rischedularlo in modo che avvenga contemporaneamente all'eventuale decesso del paziente ed ignorare l'evento di fine cura schedulato precedentemente. Il problema è che l'evento è assegnato ad un preciso dottore, dunque nel caso in qui egli riceva un altro paziente, l'evento precedentemente schedulato non sarebbe ignorato. Per risolvere, bisogna assegnare l'evento "*DOCTOR_AVAILABLE*" al paziente attualmente in cura e inserire un controllo per liberare il dottore che sta curando un determinato paziente.

I03FYZ 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 all'undicesimo laboratorio:

  https://github.com/TdP-2016/Lab10-Bar
- 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/Lab10-Bar
- 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.

ESERCIZIO 2:

Scopo dell'esercitazione:
Realizzare un'applicazione utilizzando JavaFX in grado di eseguire la simulazione di un bar.

Descrizione dall'applicazione:
Partendo da una coda degli eventi generata all'avvio in modo random, il programma sarà in grado di simulare il comportamento dei clienti che, arrivando al bar, richiedono un tavolo per potersi accomodare. Il numero totale di tavoli è limitato. La classe delle *Statistiche* dovrà essere in grado di raccogliere informazioni durante la simulazione ed alla fine di mostrare le statistiche relative alla soddisfazione dei clienti (*numero_totale_clienti*, *numero_clienti_soddisfatti*, *numero_clienti_insoddisfatti*).

Nel bar sono presenti 15 tavoli, con la seguente configurazione:
- 2 tavoli da 10 posti
- 4 tavoli da 8 posti
- 4 tavoli da 6 posti
- 5 tavoli da 4 posti

I gruppi di clienti in arrivo potranno sedersi in un tavolo libero con un numero di posti maggiore o uguale a quello richiesto. In alternativa i clienti possono servirsi al bancone (quindi senza usufruire del tavolo) restando comunque soddisfatti del servizio. Il bancone ha capienza illimitata.

Generare in modo random 2000 eventi di tipo "ARRIVO_GRUPPO_CLIENTI", ognuno dei quali caratterizzato dei seguenti valori:
- *time:* istante temporale in cui si verificherà l'evento creato (in minuti). L'intervallo tra due eventi dovrà essere compreso tra 1 e 10 minuti;
- *num_persone*: indica il numero di persone facenti parte del gruppo che vogliono sedersi al tavolo. Valore casuale compreso tra 1 e 10;
- *durata*: tempo in minuti indicante la permanenza dei clienti al tavolo del bar (tra 60 e 120 minuti);
- *tolleranza*: indica la tolleranza di ogni gruppo di clienti a restare al bar accomodandosi al bancone, nel caso in cui il tavolo richiesto non sia disponibile. Valore float tra 0.0 (se trovano il posto al tavolo restano al bar, altrimenti vanno via immediatamente insoddisfatti) e 0.9 (90% di probabilità di accomodarsi al bancone del bar anche senza potersi sedere al tavolo, restando comunque soddisfatti).

Esercizio 1.1 Realizzare una semplice interfaccia grafica in Java FX con un pulasante *Simula* ed una *TextArea* in cui visualizzare l'output del programma.

Esercizio 1.2 Associare ad ogni gruppo di clienti in arrivo il tavolo libero più piccolo che sia in grado di accoglierli.

Esercizio 1.3 Far accomodare i clienti ai tavoli in modo tale da occupare almeno il 50% dei posti disponibili del tavolo. Altrimenti cercare di indirizzarli verso il bancone.

Esercizio 1.4 É possibile trovare qualche altra configurazione in grado di massimizzare il numero di clienti soddisfatti?