TECNICHE DI PROGRAMMAZIONE

03/06/19

Tema d’esame “UfoSightings” (parte 1).

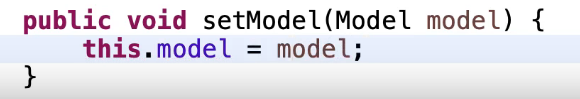
In questa lezione svolgeremo il punto 1:

Immagine che contiene screenshot

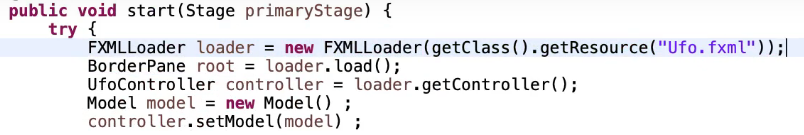
Descrizione generata automaticamente

Come prima cosa andiamo, come sempre, a creare la classe Model nel package “Model”. Successivamente lo andiamo a collegare al Controller. Andiamo quindi nella classe Controller, aggiungiamo il riferimento al Modello e il metodo “setModel” in cui dall’esterno andiamo a passare il Modello.





Fatto questo andiamo nel Main e, recuperandolo da un Main vecchio, andiamo a modificarlo nel seguente modo:



Questo passaggio all’esame non dovremo farlo in quanto ci verrà già dato un Main con la struttura corretta.

Come prima cosa dobbiamo andare a gestire la tendina relativa all’anno. Il problema ci richiede di visualizzare, oltre all’anno stesso, il numero di avvistamenti avvenuti in quel determinato anno. Dobbiamo quindi fare una query che ci permetta ci ottenere una lista di coppie (anno-n°avvistamenti).

Se andiamo su Heidi notiamo che il progetto contiene solamente una tabella e che per la data abbiamo una colonna “datetime”. Grazie alla funzione “Year” è possibile estrarre l’anno dal “datetime”. Per contare il numero di avvistamenti in ogni anno basta raggruppare per anno e contare le righe [count(chiave primaria)]. La query finale sarà quindi la seguente:

Immagine che contiene screenshot

Descrizione generata automaticamente

Nel package “Model” andiamo a creare una nuova JavaClass “AnnoCount” che useremo per recuperare le righe che ci interessano dalla tabella. Avremo un campo “year” per l’anno e un campo “integer” per il numero di avvistamenti. Ci facciamo generare i Getter/Setter e il costruttore.

**Per adesso non ci servono l’hashCode e l’equals in quanto non useremo questa classe né in un HashMap/Set né in un grafo.**

Nel nostro caso usiamo questa classe solo per riempire una tendina (useremo una lista) e quindi non ci servono. Quello che ci serve potrebbe essere il toString per stampare quello che vogliamo nella tendina.

La classe “AnnoCount” sarà:

Immagine che contiene screenshot

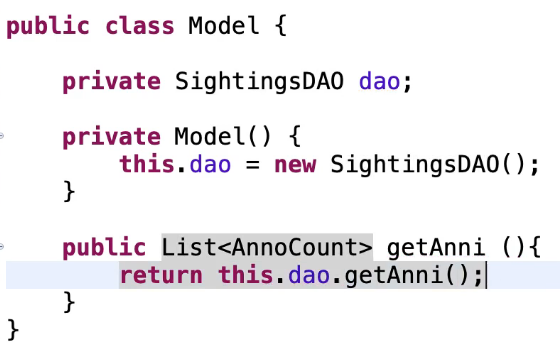
Descrizione generata automaticamente

Adesso possiamo fare il metodo del DAO per recuperare tutte le righe della tabella che ci servono. Apriamo quindi il “SightingsDAO” e creiamo un metodo che ha come obiettivo quello di restituirci una lista di “AnnoCount”. Come prima cosa andiamo ad aggiungere la query tenendo conto che il problema ci chiedeva di considerare solo gli “us” come country (clausola di where). Aggiungiamo poi il try-catch (possiamo copiarlo dai metodi sopra e modificarlo) dove ci facciamo restituire la Connection, costruiamo il PrepareStatement, creiamo una lista che restituiremo e infine scorriamo i risultati eseguendo la query. Nel while recuperiamo i valori della query e li aggiungiamo alla nostra lista. Alla fine, bisogna ricordarsi sempre di chiudere la connessione e di ritornare la lista.

Immagine che contiene screenshot

Descrizione generata automaticamente

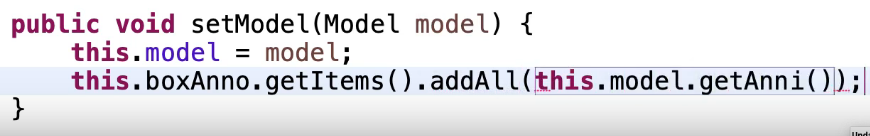
Possiamo ora definire un metodo nel Modello che richiami questo metodo appena creato nel DAO. Andiamo a creare un attributo DAO che andiamo a richiamare nel costruttore e lo usiamo per ritornare gli anni con il numero di avvistamenti. **Ricordarsi sempre il tripli collegamento tra Controller, Model e DAO.**

 (costruttore Public).

Possiamo tornare nel Controller e, appena settato il Modello, possiamo andare a riempire la lista degli anni. Ricordarsi di andare a definire cosa conterrà la ComboBox.

Immagine che contiene cielo

Descrizione generata automaticamente



Ora possiamo andare avanti con il punto (B) del primo punto in cui dobbiamo creare il grafo. In questo caso, il numero di vertici del nostro grafo (gli stati degli Stati Uniti) sono un numero relativamente basso (52) posso usare la strategia 1 (vedere lezione sui grafi) in cui aggiungo tutti i vertici, faccio un doppio ciclo for su questi vertici e se c’è un arco lo inserisco.

Prima cosa da fare è quella di farci dare tutti gli stati e di aggiungerli come vertici. Notiamo che noi abbiamo già una classe “Sightings” con tutti gli attributi della tabella e che i vertici che vogliamo aggiungere sono degli stati che sono modellati da una stringa. Possiamo quindi usare la classe “String” come classe dei vertici. Così facendo, abbiamo già l’hashCode e l’equals implementati.

Nel grafo andremo quindi ad inserire delle stringhe come vertici. Gestiamo ora la query su Heidi.

Immagine che contiene oggetto

Descrizione generata automaticamente

Con questa semplice query, otteniamo tutti i vertici del grafo. Dopo aver trovato la query, come sempre torniamo nel DAO e aggiungiamo il metodo che conterrà la query, il try-catch ecc…

Questo metodo che aggiungeremo ci restituirà una lista di “String” come abbiamo detto in precedenza. Dobbiamo stare attenti in quanto non dobbiamo prendere tutti gli stati ma solo quelli per cui ci sono degli avvistamenti nell’anno selezionato. Supponiamo quindi che il nostro metodo riceva un parametro “Year”. La nostra query avrà un parametro che andiamo a definire dopo il PreparedStatement. Il metodo sarà:

Immagine che contiene testo

Descrizione generata automaticamente

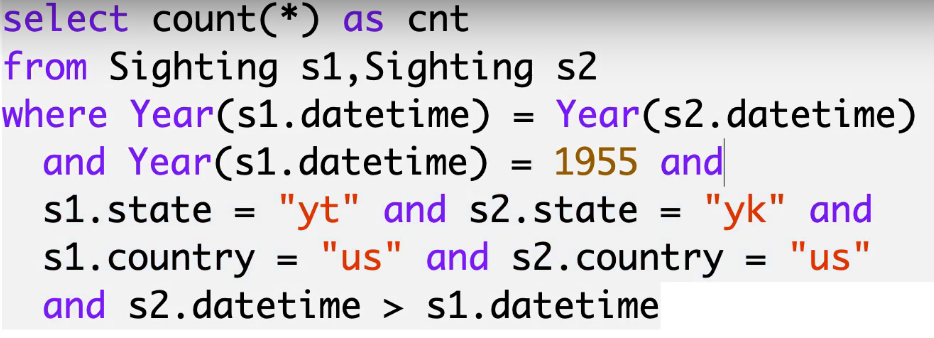
Torniamo nel Model e creiamo il metodo “creaGrafo” in cui possiamo farci dare tutti gli stati attraverso il metodo creato prima. Prima però aggiungiamo nel Model una lista di “string” per contenere gli stati (sostituisce la IdentityMap). In questo caso non usiamo l’IdentityMap perché i vertici sono delle Stringhe e non abbiamo nessuna chiave. Abbiamo quindi i nostri vertici presi dal metodo; ora dobbiamo aggiungerli al grafo (dobbiamo definirlo) aiutandoli con la classe “Graphs”.

Immagine che contiene screenshot

Descrizione generata automaticamente

Ora impostiamo la nostra soluzione 1 (due cicli for e controllo esistenza arco). Per controllare l’esistenza di un arco tra due vertici, usiamo il metodo “esisteArco” (che dobbiamo ancora implementare). In caso positivo, aggiungiamo l’arco tra i due vertici. Il problema ci dice che esiste un arco tra A e B se almeno un avvistamento in B è temporalmente successivo ad un avvistamento in A. Il nostro obiettivo è quindi quello di contare quanti avvistamenti sono avvenuti dopo gli avvistamenti in A.

Andiamo quindi a testare la query su Heidi usando due vertici a caso. Dovremo sicuramente fare un join della tabella con sé stessa. La query sarà la seguente:



Su ecllipse al posto di “1955”, “yt”e “yk” dovremo mettere i punti di domanda perché saranno valori dinamici.

Questa query la andremo ad inserire nel metodo “esisteArco” che implementiamo nel DAO.

Ricapitolando, questa è la classe “creaGrafo” del Model:

Immagine che contiene screenshot

Descrizione generata automaticamente  
In questa classe abbiamo aggiunto un metodo “esisteArco” del DAO che però dobbiamo ancora andare a creare. Ce lo facciamo creare da Eclipse e in esso andiamo ad aggiungere la nostra query.

Immagine che contiene screenshot, testo

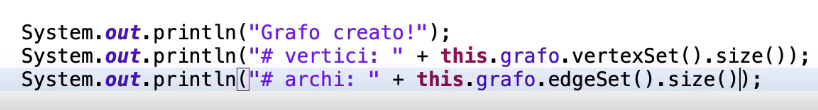
Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Per essere sicuri chiudiamo la connessione nei due rami dell’if, altrimenti si rischia di chiuderla e di non poter più accedere al ResultSet. Con il primo “if” controlliamo se il numero di avvistamenti in B dopo gli avvistamenti in A sia maggiore di 0 (in caso positivo il metodo ritorna “true” mentre in caso negativo ritorna “false”). L’ultimo else (necessario!) è collegato al primo “if” e ritorna false nel caso in cui non ci siano avvistamenti in B dopo gli avvistamenti in A.

Alla fine del metodo “creaGrafo” aggiungiamo una stampa per controllare se il metodo funziona.



Aggiungiamo nel package Model una classe TestModel per controllare il programma (ha il main!!). In questa classe abbiamo un metodo “run” in cui andiamo a creare il nostro modello e andiamo a richiamare “creaGrafo” al quale passiamo come anno “2010” (esempio).

Nel main di questa classe creiamo invece un istanza di “TestModel” e richiamiamo il metodo “run”.

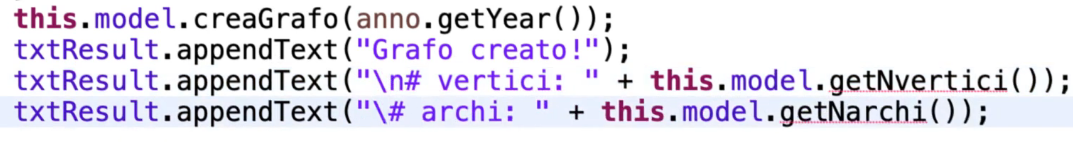
Immagine che contiene testo

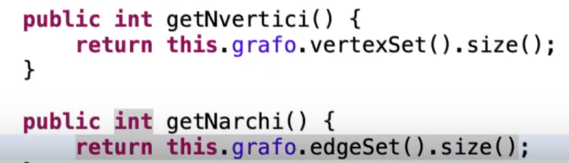
Descrizione generata automaticamente

Se runniamo il programma notiamo che ci mette molto tempo, quindi nel frattempo iniziamo a collegare il programma al controller. Il bottone “Avvistamenti” deve creare il grafo.

Il metodo “handleAvvistamenti” del Controller deve andare a recuperare l’ anno dal ComboBox. Nel frattempo, il programma ha terminato e ha restituito il risultato giusto (funziona). **Se il numero di vertici nel problema è tra 0 e 100 possiamo usare il metodo numero 1 per creare il grafo!!!**

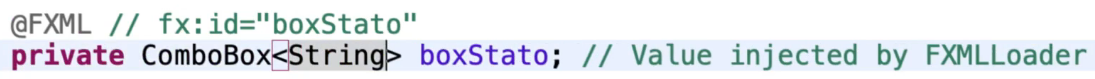
Una volta che abbiamo preso l’anno dalla ComboBox (con controllo che non sia null), andiamo a richiamare il metodo CreaGrafo passando come parametro l’anno della ComboBox. Una volta creato, lo visualizziamo a video usando 2 metodi del Model “getNvertici” e “getNarchi” (ancora da definire) che ritornano il numero di vertici e archi del grafo.



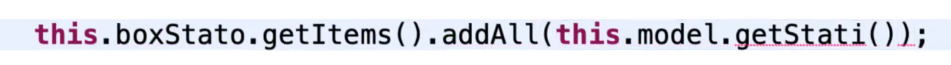


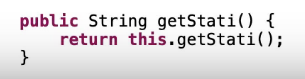
Possiamo ora passare al punto “c” del punto 1. Bisogna permettere all’utente di selezionare uno stato tra i vertici presenti sul grafo e di fare qualcosa con essi.

Andiamo nel Controller e gestiamo il ComboBox “boxStato” che conterrà delle “String”.



Nel metodo “handleAnalizza”, dopo aver creato il grafo, riempiamo il boxStato con tutti i vertici. Per far questo, dobbiamo farci dare gli Stati dal Model mediante il metodo “getStati” (ancora da definire).

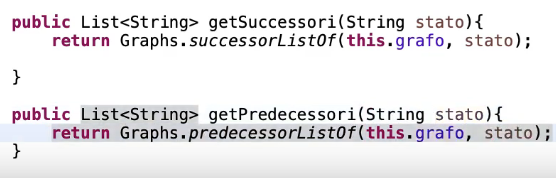


 (Nel model)

Il primo punto del punto “C” ci chiede di determinare gli stati immediatamente precedenti e successivi dello stato selezionato. Questo è abbastanza semplice in quanto abbiamo già definito queste relazioni con gli Archi. Quindi, dato un nodo, i successivi saranno tutti quelli a cui arriva un arco che parte dal mio stato considerato. Viceversa, gli stati precedenti saranno tutti gli stati collegati agli archi che arrivano al mio nodo.

**Se per risolvere questo ultimo punto, ci accorgiamo che stiamo scrivendo tante linee di codice, vuol dire che stiamo sbagliando qualcosa! (è qualcosa di più semplice, risolvibile in 4/5 linee di codice).**

Nel modello avremo quindi due metodi (uno per i successori e uno per i predecessori) che restituiranno due liste di stati (stringhe) e ai quali passeremo lo stato selezionato. In questo caso ce la caviamo con due linee di codice in quanto la classe Graphs offre proprio due metodi che ci restituiscono i successori e i predecessori.



**Controllare sempre se la classe Graphs non abbia un metodo che faccia al caso nostro!**

Il secondo punto ci chiede di determinare gli stati raggiungibili dallo stato scelto. Creiamo anche per questo punto un nuovo metodo del Model “getRaggiungibili”. Per risolvere questo punto bisogna fare una **visita in profondità.** Vedremo come nella prossima lezione.

FINE.