TECNICHE DI PROGRAMMAZIONE

02/04/19

Continuazione esercizio DAO.

Ricapitoliamo quello che abbiamo fatto la lezione precedente; dato un periodo didattico che il Model riceveva come parametro dal Controller, abbiamo provato a fare le 2 soluzioni.

Quindi il Pattern DAO ci permetter di dividere la logica per accedere al DB dalla logica applicativa del Model e usiamo la regola che per ogni tabella del DB che ci interessa, abbiamo un DAO separato. Attenzione alla tabella Iscrizioni!! Questa tabella infatti ci serve solo a mappare la relazione molti a molti tra le altre 2 tabelle e quindi non ha una relativa “IscrizioniDAO”.

Un’altra regola molto importante è che per ogni tabella, abbiamo una classe che modella le ogni riga di queste tabella. Questo Pattern è detto “PatternORM” ed ha sempre la stessa forma; ha le variabili private che rappresentano le varie colonne della tabella (usare stessi nomi!!!), ha un costruttore con i vari campi, tutti i getter e i setter e infine, cosa fondamentale, ridefinisce l’hashCode e l’equals (sulla chiave primaria).

Concludendo, abbiamo il controller (GestoreCorsiController) che ha il riferimento sempre al Model (private GestoreCorsi model) nel quale abbiamo tutta la gestione dei bottoni e quindi degli input. Nel nostro caso avevamo gestito l’input che cercava i corsi e poi, tramite il modello chiedevo i corsi per periodo.

In alcuni casi, il programma poteva dare un errore che riguardava la connessione e in particolare la timeZone. Esso si risolve cambiando il jdbcURL:



Se non dovesse andare nella sezione “avvisi” è stato pubblicato un video che spiega come risolvere il problema.

Svolgiamo ora il secondo punto della prima parte dell’esercizio.

Apriamo come sempre il Client su HeidiSQL.

Quello che dobbiamo fare è, dato un intero che è il periodo didattico, avere tutti i corsi di quel determinato periodo didattico con a fianco un numerino che mi andrà a indicare il numero di iscritti. Come faccio? Riesco a farlo tutto dalla query? Si (Query abbastanza facile).

Incominciamo quindi a provare a scrivere delle query su Heidi per estrarre il risultato che desideriamo. Analizziamo quali tabelle ci servono:

1. Sicuramente abbiamo bisogno della tabella Corso in quanto ci serve il nome del corso.
2. Mi serve sapere quanti studenti sono iscritti e quindi ho bisogno della tabella Iscrizione che mi mappa la relazione molti a molti tra Corso e Studente.
3. Non ho bisogno della tabella Studente in quanto non mi serve alcuna informazione sugli studenti.

Quindi sicuramente nel from della query avremo “Iscrzione as i” e “Corso as c”.

Nella select andiamo ad estrarre il nome dei corsi (c.nome) e il numero totale di iscritti di quel corso (COUNT (\*) as tot).

Attenzione!! Non dimentichiamoci di fare la join tra le tabelle nel WHERE mediante l’attributo che hanno in comune (c.codins=i.codins).

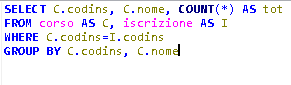
Logicamente avendo una funzione aggregata, la funzione COUNT, sotto il WHERE dobbiamo inserire una GROUP BY nella quale, nel nostro caso, gli attributi sono raggruppati secondo il nome del corso (c.nome).

Così facendo abbiamo per ogni corso, il numero totale di iscritti a quel corso.

Bisogna stare attenti, perché la chiave primaria della tabella corso è solo il codins; potrebbe quindi capire che due corsi abbiamo chiava primaria diversa ma ugual nome. Per ovviare a questo inconveniente dobbiamo aggiungere nel GROUP BY c.codins e quindi di conseguenza anche nella select.

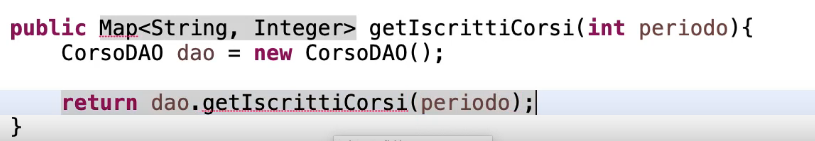
La query è una parte molto importante… il 50% della gente che non passa l’esame, sbaglia la query.

La query è dunque la seguente:



La copiamo in quanto ci servirà nel programma JAVA.

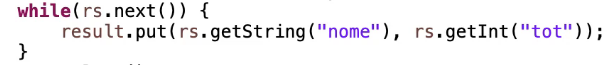
Partiamo quindi di nuovo dal nostro GestoreCorsi. Creiamo il nuovo metodo che gestisce questo problema:



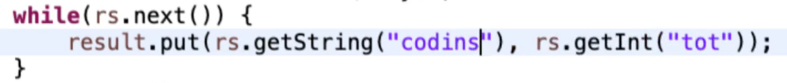
Analizziamo il codice. Abbiamo deciso di salvare i dati su una mappa, in quanto la query ci restituisce una stringa e un valore ed è quindi la scelta migliore. All’interno dei simboli di maggiore e minore inseriamo la tipologia delle chiavi che ci restituisce. Nel nostro caso abbiamo il nome del corso (string) e il numero di iscritti (integer).

In questo caso siamo riusciti a scrivere tutto nella query e quindi posso delegare tutto al DAO semplicemente inserendo il return. Ovviamente dobbiamo ricordarci di creare il DAO. Quindi vogliamo che ritorni un mappa che è data da un metodo dao.qualcosa(periodo). Questo metodo del DAO che dobbiamo ancora definire, dato un periodo didattico ci restituisce una mappa String-Integer con il risultato che vogliamo. Nel nostro caso abbiamo chiamato il nome del metodo con lo stesso usato nel modello ma solamente per un fattore di comodità. Ricordiamoci di importare la mappa. Infine, chiediamo ad Eclipse di importare il nostro nuovo metodo creato nella classe DAO. Il pattern del metodo nella nostra classe DAO è sempre lo stesso (stringa sql, connessione, ecc…). Quindi:

1. Definisco la stringa sql (copio-incollo da HeidiSQL). Attenzione agli spazi tra due righe successive.
2. Definisco la struttura dati che conterrà il risultato (nel nostro caso una Map).
3. Definisco il try-catch che gestisce la connessione, il PreparedStatement, il ResultSet ecc…
4. Ricordarsi che il st.setInt(1, periodo) consente di gestire i punti di domanda nella query



Rispetto al while del putno precedente, in questo non dobbiamo definire una nuova istanza di corso e aggiungerla ad una lista, ma ci basta inserire i risultati ottenuti nella Map “result”. Importante notare come la seconda colonna l’abbiamo chiamata come la ridenominazione della query (tot).

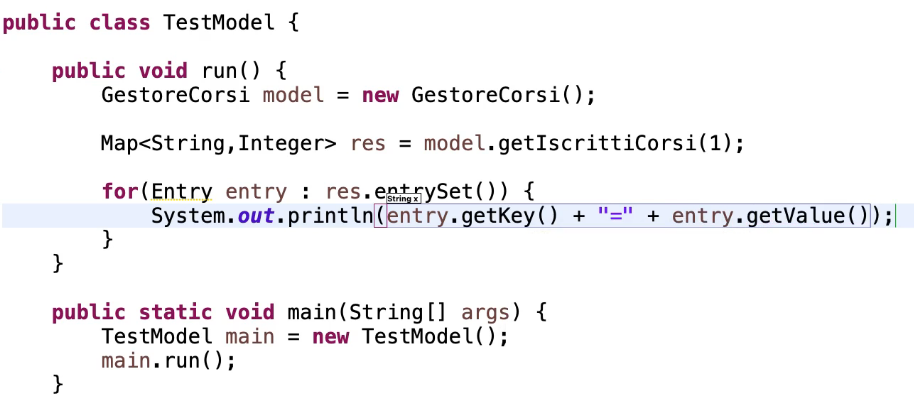


Siamo stati costretti a mettere il codins e non il nome in quanto ci sono due esami no lo stesso nome e quindi non possiamo usarlo come chiave primaria.   
Ora, prima di aggiungere la logica al controller, testiamola con una classe di Test. Creiamo dunque una nuova classe che chiamiamo TestModel, logicamente nel package Model, in cui mettiamo un Main.

Facciamo questo perché, se riusciamo a testare un modello da una classe Java, vuol dire che ho diviso bene la mia interfaccia grafica dalla logica applicativa.

In questa classe definisco un metodo run e creo un’istanza di TestModel. Nel run, creo il modello GestoreCorsi e cerco di recuperare le statistiche dei corsi in un determinato periodo (nel nostro caso 1).

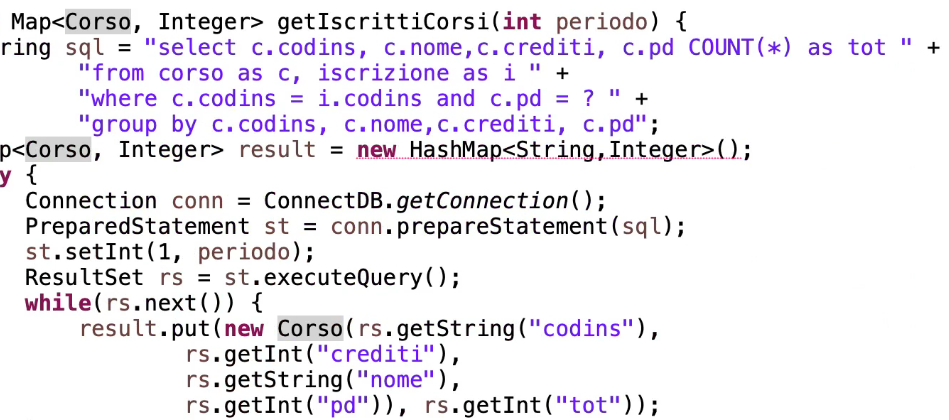
Provo a scorrere la mappa e a stampare la mappa. Noi abbiamo deciso di scorrere la mappa a coppie chiave-valore mediante un Entry.



L’unico problema è che se lo runniamo avremo come risultato non i nomi dei corsi seguiti dal numero di iscritti ma il codice del corso e quindi non è facilmente interpretabile. Soluzioni?

1. Per ogni codins vado a fare la query nel DB (problema faccio troppe query).
2. Ho da qualche parte in memoria una mappa con tutti i corsi dove la chiave è codins e il valore è il nome del corso. Chiedo a questa mappa, dato il codins, di restituire il nome del corso.
3. Fare una query con tutti i campi della classe corso e ogni volta mi vado a rigenerare un corso nuovo (come facevo nel while dell’esercizio prima).

Proviamo a fare il 3. Il nuovo codice sarà il seguente:



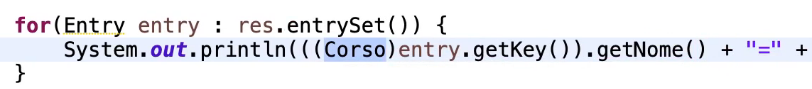
Se voglio creare un nuovo corso sulla base di quello che mi esce dalla query, ovviamente devo avere tutti i campi della tabella e quindi ho aggiunto nella sql anche i campi crediti e pd in modo tale che ‘ad ogni giro’ riesco a ricrearmi il corso. La mappa cambia e non è più una String-integer ma una Corso-integer. Nel while quindi non avrò più una getString ma una new di Corso in cui vado a prendere tutti i parametri che ho messo nella query. Questa soluzione non è molto efficiente ed è una delle soluzioni che può portare agli errori più comuni. Questo perché sto creando tantissime volte degli oggetti corso inutili.

(dopo HashMap devo mettere anche Corso)

Soluzione migliore è la seconda!!! La vedremo successivamente.

Ricordarsi di cambiare la mappa anche nel modello (GestoreCorsi) e nel testModel.

In quest’ultimo cambiamo anche la getKey perché posso recuperare come chiave il nome del corso. Dobbiamo fare un cast a Corso perché il metodo getKey mi restituisce un oggetto ma noi siamo sicuri che la chiave è un corso.



Una volta controllato nel test se funziona il codice, lo spostiamo nel Model, ricordandoci di modificare il tutto (non più system.out ma .appendtext ecc…).

Passiamo ora alla seconda parte dell’esercizio nella quale bisogna lavorare su un corso solo.

Anche per questi punti notiamo che ci servono due bottoni capaci ognuno di svolgere un punto. Inoltre, sono necessarie una label e una textField per far inserire all’utente il dato. Ricordarsi sempre di inserire l’id di ogni elemento e nel caso dei bottoni anche l’onAction.

Quando copiamo lo scheletro del controller dobbiamo poi prendere solamente le cose che ci interessano. Non possiamo prendere tutto e ricopiarlo perché nel controller abbiamo aggiunto dei nuovi elementi oltre allo scheletro vero e proprio preso dallo scene builder.

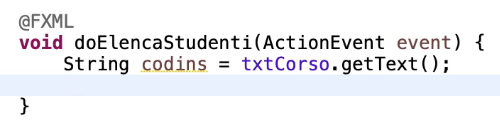
Partiamo dal bottone ElencaStudente.

Così come abbiamo fatto con la classe Corso, creiamo una classe Studente nel package Model. Questa classe avrà tutti i campi della tabella Studente ma sotto forma di variabili private. Inseriamo tutti i getter e i setter, il costruttore e l’ hashcode e l’equals (ricordarsi di selezionare la chiave primaria che è la matricola.

Inoltre, dovremmo avere il relativo DAO nel package DB.

Fatto questo ritorniamo al controller e incominciamo a gestire il bottone.

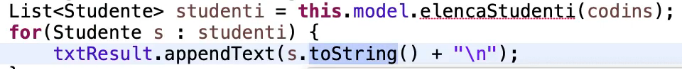
Incominciare scrivendo il codice che ci permette di prelevare ciò che è scritto nella casella di testo:



Dato questo corso, mi immagino di avere nel Model un metodo che mi permette di avere una lista di studenti. Creiamo allora questo metodo con la seguente sintassi:

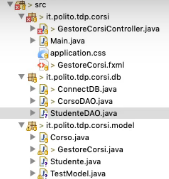
List<nomedellaclassedelModel> nomeLista = this.model.nomeMetodo(variabilecheglipasso);

Data questa lista, la scorriamo e la stampiamo. Aggiungiamo nella classe Studente del Model anche il metodo toString perché ci potrebbe essere molto utile.



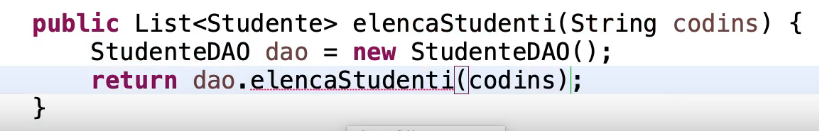
Come si può vedere all’immagine infatti, abbiamo usato il metodo toString per stampare la lista!!

Fatto ciò però dobbiamo ancora definire nel Model il metodo “elencaStudenti”. Facciamo fare da Eclipse che ci crea appunto il metodo nella classe GestoreCorsi del Model!

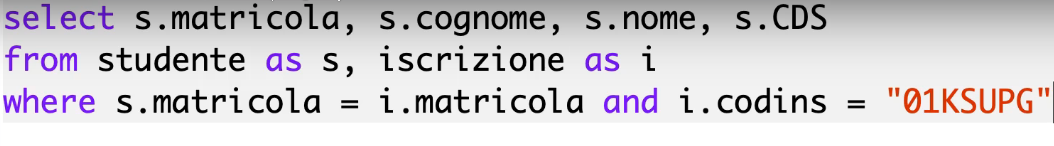


Questo è come si presenta il nostro progetto.

Come prima cosa in questo metodo andiamo ad inserire l’istanza del DAO poi diciamo che deve ritornare dao.elencaStudenti(codins ). Di nuovo questo metodo non ce l’abbiamo ancora e lo andiamo a creare.



Ci facciamo di nuovo creare da Eclipse il metodo elencaStudenti che verrà quindi creato nella classe StudenteDAO. Come al solito, dentro a questo metodo dobbiamo fare la query. Vado quindi su Heidi e provo a farla li. Come tabelle ci servono sia Iscrizione che Studente. La query finale è la seguente; non abbiamo messo select \* perché altrimenti ci saremmo portare anche codins della tabella Iscrizione.

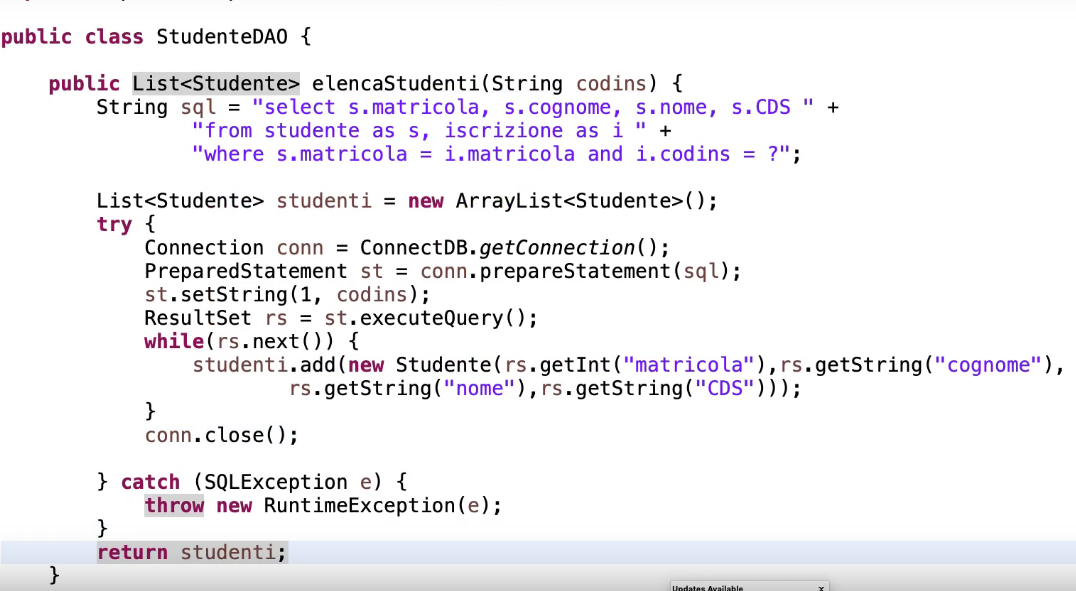


Ricordarsi gli as e gli apici prima e dopo la stringa. Nel nostro esempio abbiamo cercato tutti gli studenti che sono iscritti al corso con il codice “01KSUPG”. Ricopiamo questa query nella stringa sql del nostro metodo elencaStudenti. Attenzione sempre agli spazi e a mettere il punto di domanda al posto del codice che abbiamo messo come esempio. Per la struttura possiamo prendere spunto da CorsoDAO.

Dopo aver definito la query, andiamo subito a definire la struttura dati che poi riprenderemo che nel nostro caso è una lista di Studenti inizialmente vuota. Poi come al solito mettiamo il try-catch con dentro la connessione ecc… Ricordiamoci sempre del setString o setInt nel caso in cui ci siano punti di domanda.

Dentro al try-catch inseriamo come al solito il while che andrà a scorrere tutti i dati.

Nel nostro caso vogliamo che venga aggiunta alla lista studenti una new di Studente che vuole come sempre tutti i campi che devono essere definiti come sempre nello stesso ordine in cui sono stati definiti nel costruttore nella classe Studente del Model.



**ABBIAMO FINITO IN QUANTO SIAMO PARTITI DAL CONTROLLER E SIAMO ARRIVATI FINO AL DAO.**

FINE.