TECNICHE DI PROGRAMMAZIONE

26/03/19

ESERCIZIO CON DAO!! Programma che gestisce un elenco di corsi universitari.

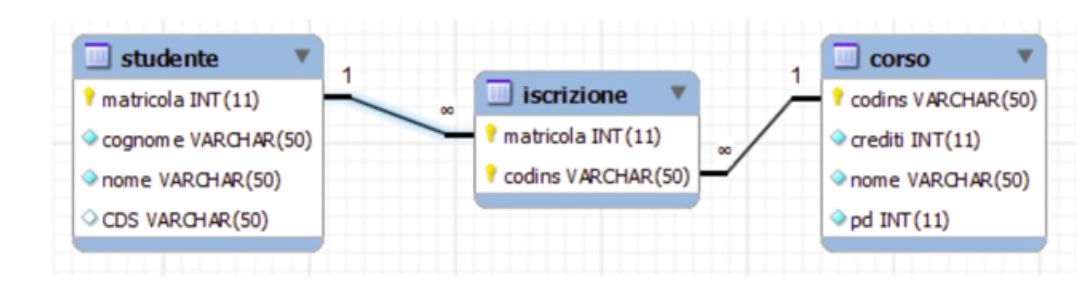


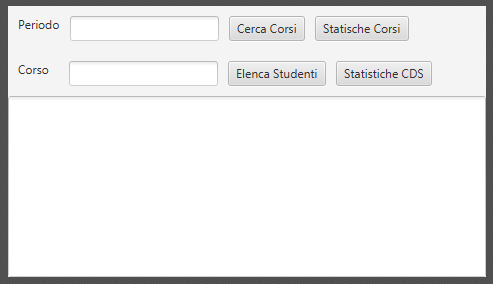
Diagramma entità-relazione del programma che dobbiamo scrivere.

Come prima cosa importiamo la base del programma da GitHub.

Dopo dovremmo fare 3 cose:

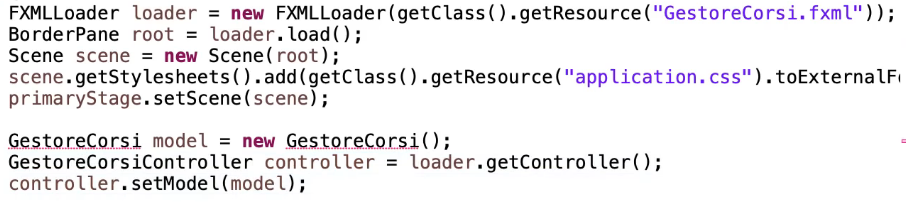
* Fare il download della cartella su GitHub in quanto ci servono il file .sql e il mysql.
* Su HeidiSQL importo il file .sql e lo runno.
* Su Eclipse, nel package creo una cartella (folder) chiamata “lib” e all’interno ci metto il mysql; successivamente premo il destro su questo file mysql e su build path faccio add to build path.

Dopo aver fatto ciò, apriamo dal nostro package principale, il file .fxml con SceneBuilder per modificare l’interfaccia grafica. Dopo di che aggiungiamo tutti gli elementi dell’interfaccia che ci servono fino a raggiungere il seguente schema:



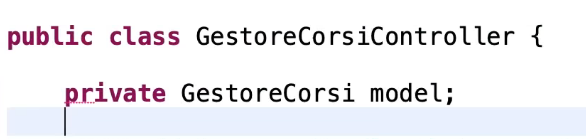
Ricordarsi di gestire bene il Padding in modo di avere una disposizione degli oggetti ottima e di inserire tutti gli fx:id delle caselle di testo, aree di testo e dei bottoni. Per quest’ultimi ricordarsi anche di inserire On Action.

Copiamo successivamente lo scheletro nel Controller di Eclipse.

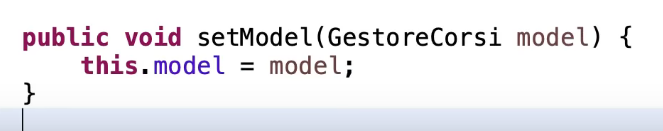


Ricordiamoci di modificare le prima due righe del Main come nell’immagine superiore e di aggiungere il model (nel nostro caso di chiama GestoreCorsi) che ci servirà successivamente.

Successivamente creiamo il package del Model e dentro creiamo la classe appunto “GestoreCorsi”.



Nel controller definisco un attributo model e creo in fondo un metodo di tipo void (non restituisce niente) che “setta” il mio model.



**Questi procedimenti sono sempre da fare all’inizio di ogni programma. All’esame questa parte ce l’avremo già fatta quindi non preoccuparsi.**

Ricordarsi di salvare il Controller perché altrimenti il codice **controller.setModel(model)** nel Main potrebbe darci un errore.

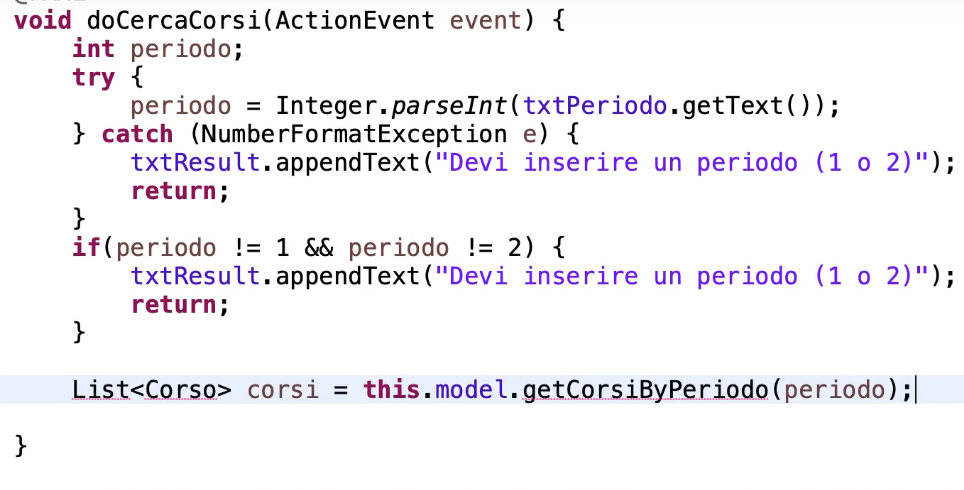
Adesso che abbiamo creato l’interfaccia grafica e abbiamo sistemato tutti i piccoli problemini nel Main, dobbiamo associare la logica ai due bottoni. Partiamo dal bottone “Cerca Corsi”.

Per testare le query come sempre ricordiamoci di usare Heidi e non Java. Nel nostro caso la query da testare su HeidiSQL è la seguente:



Essa ci permette di conoscere tutti i corsi di un determinato periodo didattico. E’ importante utilizzare il digramma che ci viene dato su GitHub per capire che tipo di query scrivere (se servono Join tra tabelle o meno). La forma delle query è molto simile a quella di basi di dati.

**DOPO AVER INDIVIDUATO LA QUERY ADOTTARE UN METODO TOP-DOWN, CIOE’ PARTO DAL PUNTO PIU’ ESTERNO (CONTROLLER) E OGNI VOLTA VADO AD IMPLEMENTARE I METODI CHE HO IMMAGINATO.**

****

Vado quindi nel controllo e nel metodo che gestisce il tasto doCercaCorsi, incomincio a riflettere su cosa potrebbe servirmi. Innanzitutto, ho bisogno di una variabile di tipo integer (periodo) che contiene il valore inserito nella textField dall’utente. Poi gestisco gli eventuali errori dell’utente che potrebbe inserire un valore diverso dal periodo didattico (il periodo didattico è un valore che può essere 1 o 2, se l’utente inserisce qualcos’altro, dobbiamo restituire un errore). La definizione della variabile “periodo” deve avvenire fuori dal try-catch in quanto ci serve anche fuori da quest’ultimo. Quindi il primo try-catch verifica che venga inserito un NUMERO, il secondo if invece serve per controllare che venga inserito 1 o 2!!

Ora, chi mi dovrà dire quali sono i corsi che rispettano le richieste inserite dall’utente??? IL MODEL!!!!

Il modello mi restituirà una lista di corsi, ma quindi una lista della **Tabella Corso.** Mi immagino un metodo getCorsiByPeriodo() in cui passo il parametro periodo. Notiamo che ci sono degli errori, perché quest’ultimo metodo non è stato definito (lo vado a definire ora). Inoltre mi sono immaginato una classe Corso.

Che dati si scambiano il DB e il nostro programma Java????

Se sono tabelle di un’unica colonna con delle stringhe , saranno delle stringhe, ma se sono più complicate, ci servono dei contenitori che mi modellino cosa c’è nel DB. Questo Pattern si chiama ORM (lo vedremo successivamente). Anticipazione:

**PER OGNI TABELLA NEL DB CI DEVE ESSERE UNA CLASSE IN JAVA.**

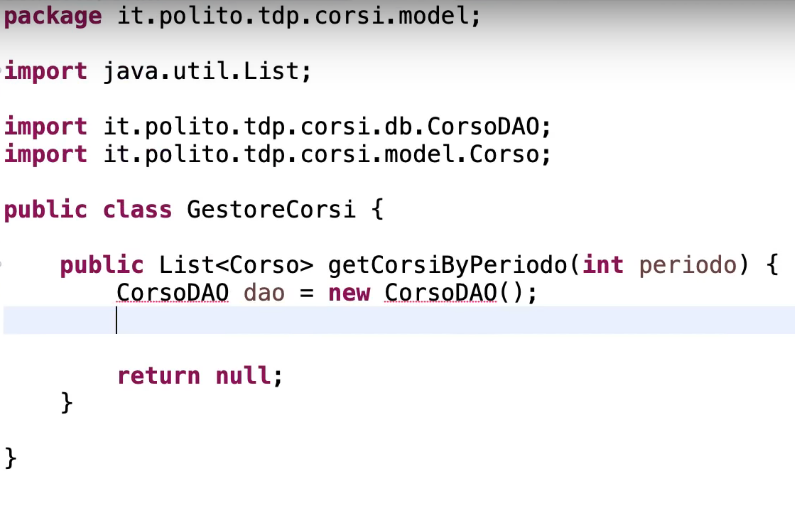
Questa classe si chiamerà nello stesso modo della tabella, avrà gli stessi campi della tabella definiti come PRIVATI e tutti i Getter e i Setter.

**ATTENZIONE: LE TABELLE CHE CI PERMETTONO DI COLLEGARE DUE TABELLE MEDIANTE LA RELAZIONE MOLTI A MOLTI, NON VANNO MODELLATE!!! (TABELLA ISCRIZIONI).**

Quindi creo la classe Corso nel Package Model. Andiamo a vedere nel diagramma tutti gli attributi della tabella e li andiamo a inserire nella classe sotto forma di variabili private e con lo stesso nome (cambierà la tipologia int,string,ecc…).Successivamente inseriamo i costruttori, i getter e i setter (fa tutto Eclipse). Nel nostro caso ci serve opzionalmente un toString in quanto noi vogliamo stampare i corsi. Inoltre è OBBLIGATORIO avere gli HashCode e gli Equals. Su cosa li definisco?? Sulla CHIAVE PRIMARIA della tabella in considerazione. Nel nostro caso la chiave primaria di CORSO è codins.

Quindi ora abbiamo creato la classe ma ci manca il modello.

Java ci suggerisce già di crearlo e noi lo facciamo. Implementiamo questo metodo nella classe GestoreCorsi del model.



Facciamo le operazioni direttamente sul DB? NO!!! Avrò un altro livello di classi che mi permetteranno di accedere al DB; queste classi sono dette “classi DAO”. Anche qui abbiamo una regola:

PER OGNI TABELLA NEL DB AVRO’ IL MIO DAO SPECIFICO.

Istanziamo quindi una classe CorsoDAO a cui posso chiedere le informazioni chi voglio estrarre dal DB.

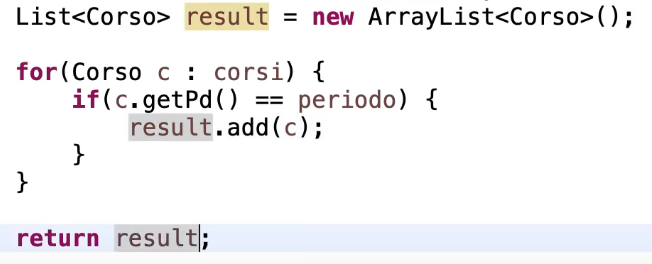
Creiamo un nuovo package .db in cui inseriremo tutte le classi che mi servono per il DB.

In questo Package andrò a inserire quindi la mia classe CorsoDAO. Da qui ho due modi di operare. Potrei farmi dare tutti i corsi dal DB e poi internamente li vado a filtrare secondo l’anno didattico, oppure potrei filtrare direttamente con la query. Se lavorassi con la query, tutto quello che faccio con a query non dovrei poi farlo in Java. All’esame dovremo capire bene quale metodo usare; noi li vedremo entrambi.

Con il primo metodo: posso chiedere al DAO la lista di tutti i corsi per esempio con un metodo ListAll();

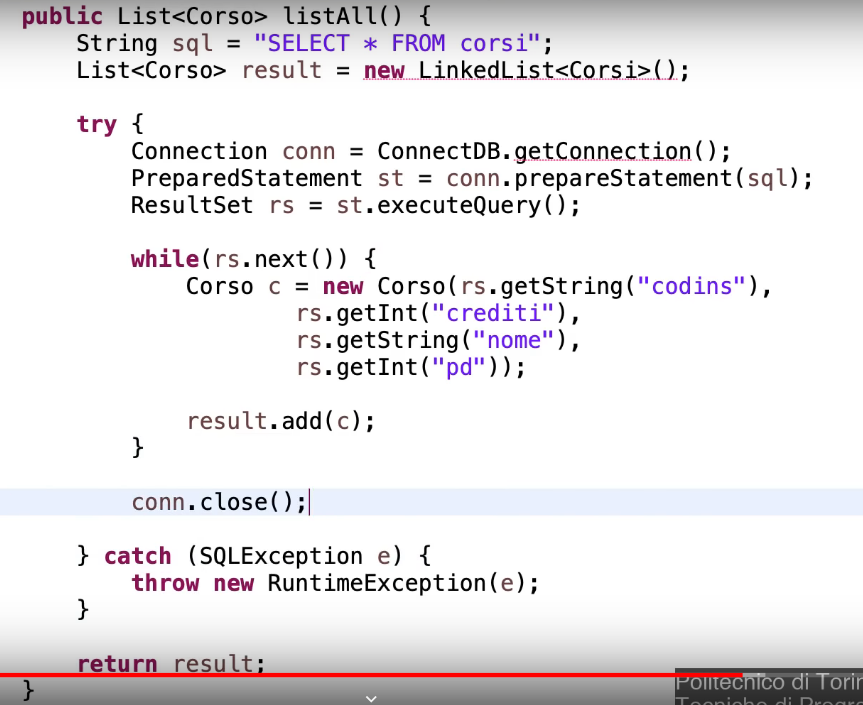


Dopodiché su questi corsi eseguo un filtraggio con un metodo for.



Quindi per ogni “c” in “corsi” mi chiedo se il periodo didattico è uguale a quello inserito dall’utente e in caso positivo lo aggiungo alla lista result. Ricordiamoci di istanziare questa nostra lista di supporto che contiene tutti i risultati (prima riga). Quindi il filtraggio non lo faccio in una WHERE della query ma lo faccio in Java con un for.

Successivamente aggiungiamo il metodo ListAll() (vedi due immagini fa) alla classe CorsoDAO del package che gestisce il database. Il templete per queste chiamate del DAO è sempre lo stesso ed è il seguente:



Analizziamo questa immagine… Come prima cosa creiamo una stringa che contiene la query da eseguire, poi creiamo una nuova lista risultati che otterrà tutti i corsi che estrarrò dal DB. Successivamente inseriamo il try-catch che gestirà la nostra connessione e il while per scorrere il ResultSet. Dentro al while creiamo un csotruttore (c) e dentro andiamo a mettere tutti i nostri campi che ci interessano.

**ATTENZIONE ALL’ORDINE!!! BISOGNA SEGUIRE LO STESSO ORDINE USATO PER DEFINIRE IL COSTRUTTORE DI CORSO DEL MODEL!!!!**

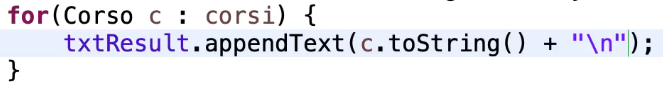
Dopo il get… nella parentesi mettiamo il nome della colonna. Infine, aggiungo il corso alla mia lista Result.

Fuori il try-catch mettiamo return result. Ricordarsi di chiudere sempre la connessione.

(la LinkedList è ‘corso’ non ‘corsi’).  
Per creare la connessione abbiamo usato il metodo getConnection della classe ConnectDB che però dobbiamo ancora definire. Quindi creiamo la nostra nuova classe ConnectDB nel package del database e all’interno di questa classe definisco il metodo statico getConnection (sempre uguale, quello con il jdbcURL, il DriverManager ecc…).



Negli esercizi che ci daranno e all’esame, la stringa dell’URL ci verrà già data. Questa classe ce l’abbiamo già non dobbiamo riscriverla. Come ultima cosa, ritorno nel Controller e implemento il codice che mi permette di visualizzare il risultato nella textResult.



Ora proviamo a runnare il programma ma ci esce l’ errore:



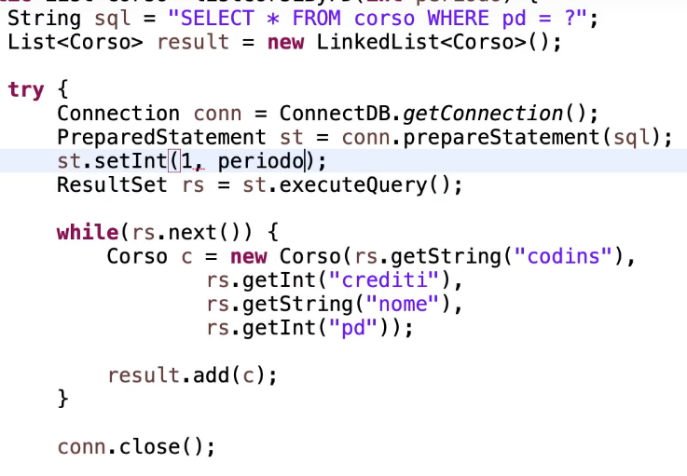
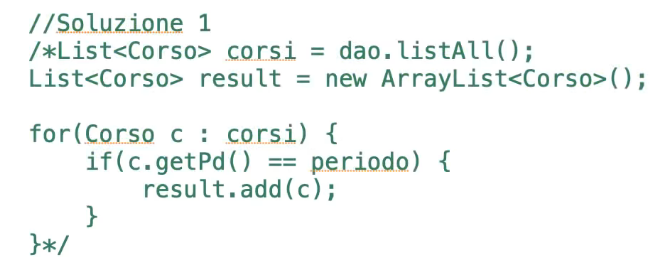
Quando l’errore riguarda l’SQLSyntax allora sappiamo che l’errore è nella Query. Nel nostro caso avevamo sbagliato a definire la tabella corso nella query in quanto l’avevamo chiamata “corsi” mentre il suo nome era “corso”.

Quindi ricapitolando, nel Controller abbiamo gestito il bottone, chiedo al model i corsi per il pattern MVC, ma il model non mi risponde direttamente ma utilizza un DAO.

Ora proviamo a vedere la soluzione 2 (facciamo fare tutto al DB mediante la query e non filtriamo niente da JAVA). Nella classe GestoreCorsi, non avremo più tutte quelle righe di codice che mi filtravano il risultato ma avremo solo più un return.



Logicamente adesso dobbiamo gestire questo nuovo metodo che abbiamo creato e quindi lo aggiungiamo alla classe DAO.

Il codice per gestire il metodo sarà molto simile a quello della prima soluzione, infatti avremo sempre l’apertura e la chiusura della connessione, ma sarà presente nella query un WHERE che mi permette di evitare tutto quel codice (immagine destra) e un st.setInt() che mi gestisce il punto di domanda.