TECNICHE DI PROGRAMMAZIONE

26/03/19

Continuazione programma database.

Il codice dopo il while ci permette di estrarre il risultato che vogliamo (quello che ci viene chiesto all’interno della query). Dopo che ho estratto il risultato posso fare diverse cose :

* Stamparlo a video
* Aggiungerlo ad una Collection
* Visualizzarlo all’interno di una interfaccia utente

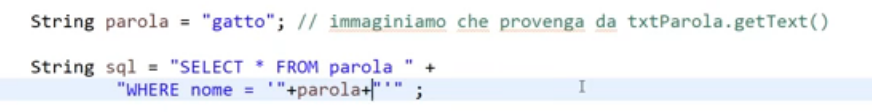
Mai dimenticarsi di chiudere la connessione. In questa lezione analizzeremo due cose.

1. La costruzione della stringa sql.
2. Dove mettere le varie parti di istruzioni.

Per esempio noi non vogliamo che si sappia il nome utente o la password della stringa di connessione o che sia accessibile a tutti. Nello Statement noi dobbiamo sempre passare la stringa SQL. Nel caso semplice la SQL contiene un select ma può essere che contenga un delete, insert ecc.

Per la select avremo una ExecuteQuery mentre per delete insert ecc avremo ExecuteUpdate. Quest’ultima restituisce un int,cioè un numero che ci indica quante righe sono state modificate.

Nel caso in cui la stringa SQL dipende da qualcosa che inserisce l’utente, il codice cambia:



Creiamo una nuova variabile che contiene la parola inserita dall’utente e cambiamo il codice di sql (attenzione alle virgolette!!).Questo metodo però a noi non piace e quindi troviamo un’alternativa migliore.

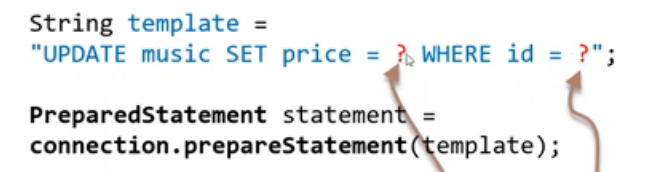
Immaginiamo infatti di avere una tabella con tutti i nomi utenti. Se un utente decide di chiamarsi nel seguente modo, rischiamo di perdere tutta la tabella utenti!!



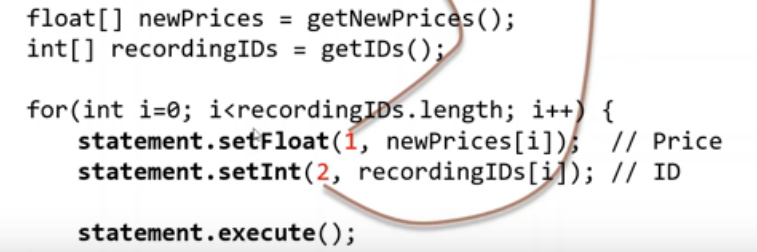
Una soluzione a questo problema potrebbe essere quello di evitare che vengano inseriti caratteri speciali ma questo lavoro risulterebbe molto lungo e faticoso. Sempre stare attenti a ciò che viene inserito dall’utente.

Per risolvere questo problema useremo il metodo “PrepareStatement”. JDBC ci permette di separare la sintassi dell’istruzione rispetto ai dati che voglio inserire nell’istruzione stessa. Evitiamo così l’interpolazione sotto forma di stringa. JDBC ci pensa lui di mandare separatamente la stringa e poi separatamente i parametri. Non li collega mai insieme!!

Ciò è possibile se noi al posto di usare la classe Statement, usiamo la classe PrepareStatement. In questo modo la sintassi della stringa rimane uguale, ma nel posto in cui dovremmo mettere i parametri, mettiamo dei punti interrogativi (senza apici!!).

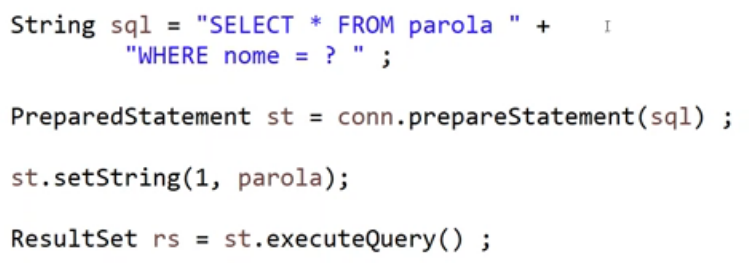


Quindi ora in fase di esecuzione non avremo più nulla perché passiamo la stringa quando definiamo il PrepareStatement. Avremo poi dei metodi setQualcosa ( della classe PrepareStatement )che ci permettono di gestire i punti interrogativi.



I parametri quindi vengono impostati a parte e, così facendo, i caratteri pericolosi non vengono mescolati con la query vera e propria). NOI NON USEREMO MAI L’OGGETTO STATEMENT MA SEMPRE L’OGGETTO PREPARESTATEMENT.

Il nuovo risultato sarà quindi il seguente:



Nuovo Argomento: Design Patterns (DAO)

Dove metto tutte queste cose di cui stiamo parlando?

Noi non vogliamo mescolare le informazioni tecniche di connessione con l’algoritmo di lavorazione dei dati e la sintassi dell’SQL.

Grazie a DAO noi possiamo distinguere nella nostra applicazione varie tipologie di classi a cui vogliamo attribuire delle responsabilità o dei compiti specifici. Cosi come abbiamo fatto con il modello separato dal controller, la stessa cosa la facciamo con il database.

La parte dell’applicazione, le classi che usano i dati, che interagiscono con il database, le chiamo Client.

Nelle classi Client, cioè nel nostro caso nel model (è lui che ha bisogno di accedere al database), noi non sappiamo niente di ciò che riguarda il database.

La classe Model del Pattern MVC diventa il Client del Data Access Object.

Io devo costruire il mio Model senza nemmeno sapere come è costruito il mio database. La classe JDBC non deve comparire nel Model. In quest’ultimo non voglio vedere Connection, non voglio vedere Statement ecc.

Tutte le volte che il Model ha bisogno di accedere al database, lo chiede ad un’altra classe che è la DAO. Questo oggetto serve proprio per fornire l’accesso ai dati. Tutto il codice JDBC lo incapsuliamo in delle classi che sono del tipo DAO. Esse contattano il database senza sapere il motivo per cui gli è stato richiesto (ignorante rispetto al Client).

Come comunicano tra di lor il DAO e il Client? Medianti gli Oggetti!!

Io ti do un oggetto voto tu salvamelo sul database. Questi oggetti vengono chiamati DTO (Data Transfer Object) o TO (Transfer Object). Essi la maggior parte delle volte non hanno metodi ma hanno solo delle proprietà private e getter e setter. È solamente un contenitore di dati. Per esempio, lui sa come è fatto il voto ma non sa come è strutturato il database o come vengono gestiti questi voti. E il più ignorante di tutti ed è per questo che viene usato da tutti. Il model userà questo oggetto per costruire le strutture dati interne, il DAO utilizzerà questi oggetti per costruire le query. Anello di congiunzione. Questi contenitori di dati viaggiano avanti e indietro tra Model e DAO (ricordarsi che il DAO fa le query!!!).

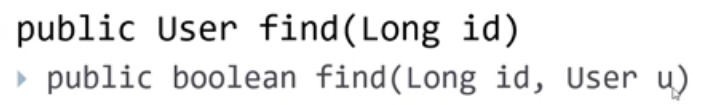
L’interfaccia tra Client e DAO è sempre sotto forma di oggetti.

**Ricordiamo che il DAO non ha stato, non ha variabili di istanza, cioè è una classe che contiene solo metodi e non proprietà. È come il Model nel MVC!!!**

Il DAO offre di metodi che chiamiamo CRUD (Create, Read, Update, Delete). Esso però può anche offrire metodi di ricerca.

Noi non dobbiamo mai usare metodi di cancellazione, non ne vale la pena. Conviene marchiare, identificare in modo diverso ciò che si vuole cancellare cosicché in un futuro se dovesse servire ce l’abbiamo pronto.

Immaginare classe UserDAO che gestisce gli utenti.



Utilizzando la prima formula per trovare un oggetto, la NEW dell’oggetto lo fa il metodo find, quindi lo fa il DAO. Utilizzando la seconda formula invece l’oggetto lo crea il Model. Se User fosse una classe, non cambia molto, ma se invece fosse un’interfaccia che è implementata da classi diverse, allora devo capire chi ha l’informazione per creare la classe giusta.

Nel nostro esercizio noi dividiamo il nostro package in due package distinti, uno chiamato database e l’altro chiamato Model. In quest’ultimo andò a mettere gli oggetti di lavoro e eventualmente la classe Model, mentre nel primo andrò a mettere la classe DAO e ciò che la supporta.

Nel Package del database creiamo una classe DizionarioDAO che gestisce il dizionario e ha un metodo che mi dice se la parola esiste. All’interno di questa classe avremo una serie di metodi che per esempio potrebbero cercarci una parola all’interno del database, oppure ci permetterebbero di visualizzare la lista di tutte le parole oppure creare una nuova parola ecc…

Per poter fornire un’interfaccia sensata a questa classe DAO, mi devo preparare degli oggetti di lavoro!!! (mi sposto nel package Model).

Noi creiamo per esempio un oggetto Parola. Questo oggetto fa un po' da specchio rispetto alla struttura del nostro database.

I campi di questo oggetto devono essere quindi paralleli alla struttura del Database. Quindi la nostra classe Parola conterrà un intero ID e una String nome e basta, non fa nient’altro!

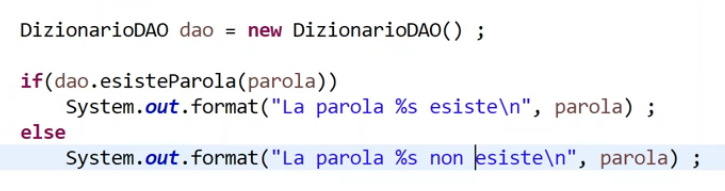
Ovviamente dovrà contenere il costruttore delle variabili per creare l’oggetto Parola, tutti i Getter e Setter e gli HashCode e gli Equals che possono lavorare per esempio sugli ID per evitare che ci siano oggetti che il Database considera uguali mentre Java no!!

In DizionarioDAO, per ogni metodo creiamo una connessione con i passi visti precedentemente.

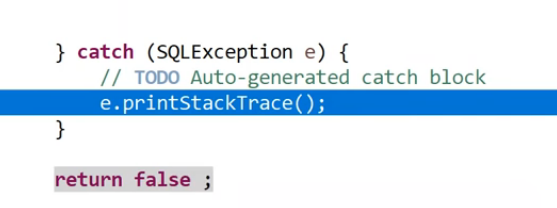


Abbiamo diverse cose da puntualizzare su questo codice. Per prima cosa che la String sql deve sempre essere scritta prima su Heidi e testata la sopra per poi essere copiata, MAI scrivere direttamente su Java. Seconda cosa, abbiamo eseguito un semplice if e non un while in quanto sappiamo che otterremo solamente una riga e quindi ci chiediamo se la riga esiste oppure no. Terza cosa, nell’if non abbiamo fatta return true ma abbiamo creato una nuova variabile booleana in quanto cosi possiamo chiudere la connessione senza problemi. Inoltre, così facendo posso chiudere la connessione e dopo avere la restituzione del risultato.

Quindi adesso noi possiamo cancellare tutte le righe di codice nel main e se voglio sapere se c’è una parola nel dizionario, lo chiedo direttamente al DAO.



Istanzio nel Main una classe “DizionarioDAO” (stesso nome della classe del package del database) e poi chiederò al Dao se esiste la parola.



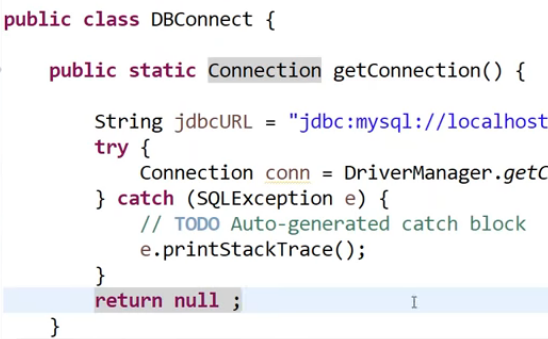
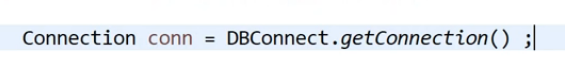
Ricordarsi dopo il catch di scrivere return false così in caso di eccezione il programma non va avanti ma si ferma.

Abbiamo un ultimo problema ; ogni volta che creo un metodo DAO dovrei ripetere le 2 istruzioni seguenti:



Inoltre, il mio database sul mio computer potrebbe avere una password diversa e quindi dovrei modificare in ogni metodo la stringa di connessione. Conviene quindi **centralizzare** queste stringhe di connessione.

Creiamo quindi una classe che ha come unico scopo quello di chiamare getConnection() , anche perché il DAO non la chiama mai direttamente. Questa classe la chiamiamo **DBConnect**. Questa classe va inserita nel package.db non in quello del Model logicamente!! Deve return conn prima del catch! Nell immagine no!

Classe DBConnect creata nel Package.db Richiamo della classe nel metodo di DizionarioDAO.

**Quindi ricordarsi:**

* **Ogni volta che abbiamo un metodo DAO, apriamo la connessione richiamando la classe DBConnect.**
* **Sempre chiudere la connessione alla fine.**
* **In mezzo fare tutte le interrogazioni che ci servono e restituire qualcosa che ha senso in Java (booleano,null,oggetto,lista).**
* **Abbiamo una classe che ha bisogno dei dati (nel nostro caso il Main ma successivamente sarà il Model.**
* **La classe DAO nel nostro caso DizionarioDAO, che sa come interrogare questi dati ed è l’unica classe a cui diamo il permesso di importare java.sql.**
* **BDConnect che per noi è solamente il metodo getConnection() nascosto, messo da parte.**
* **Uno o più oggetti, nel nostro caso la classe Parola che servono per contenere i dati che vengono letti o scritti nel Database.**