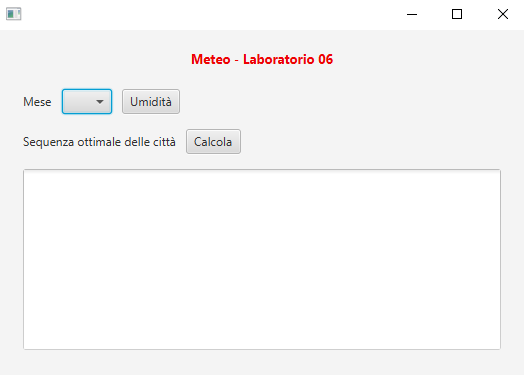
TECNICHE DI PROGRAMMAZIONE

16/04/19

Soluzione esercitazione 6 laboratorio (Ricorsione).

Come prima cosa facciamo partire il Main per vedere cosa fa il programma. Quest’ultimo apre la seguente finestra, nella quale però non funziona niente.



Come prima cosa dobbiamo gestire il menù a tendina dei mesi.

Prima però inseriamo il codice che ci permetterà di far comunicare tra di loro main, controller e model.



Il programma non saprà cosa è “setModel”. Andiamo quindi nel controller (MeteoController) e andiamo a inserire il riferimento al Model.



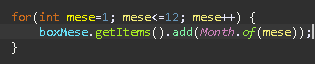
Dopodiché premiamo col destro su “model” e creiamo il Setter (Source, Generate Getters and Setters, spuntiamo model e specifichiamo solo il setModel).

Terminato questo formalismo iniziale, possiamo finalmente popolare la tendina del mese. Sempre nel controller andiamo quindi a cercare il Box che si occupa del mese e, al posto del punto interrogativo, scriviamo “Month”.

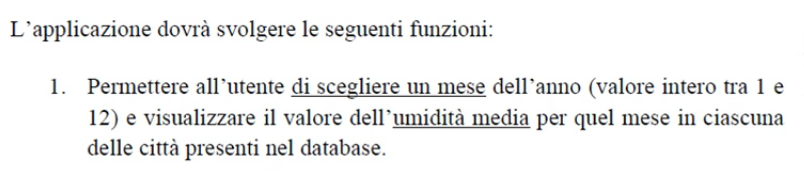


Ci verrà segnalato un errore in quanto Month è una classe che richiede l’import di java.time.Month.

Fatto ciò, dobbiamo ancora popolarlo. Per far questo andiamo nell’inizializzazione del controller. Dobbiamo quindi ciclare su 12 valori che saranno appunto i nostri mesi. Il comando sarà il seguente:



L’obiettivo dell’esercizio era il seguente:



Noi dobbiamo quindi come prima cosa scoprire quali sono le città presenti nel Database e poi dobbiamo calcolarci la media dell’umidità per ogni città. Noi in realtà sappiamo già quali sono le città ma vogliamo farcele restituire dal Database per sicurezza. Dobbiamo accedere al Database per popolare il Model e il Dao. Apriamo HEIDI e apriamo il file .sql in modo da “importare” il database.

Fatto ciò andiamo a vedere su Eclipse se il database si connette con li programma. Entriamo sul Package.db e apriamo la classe DB.Connect. Cambiamo come prima cosa il jbdcURL inserendo la nostra password con cui abbiamo aperto il database su HEIDI.

Successivamente andiamo ad aprire “TestMeteoDAO” e lo runniamo per vedere cosa fa. Notiamo che ci restituisce il contenuto del Database (in particolare sono state richieste le città, le date e l’umidità).

Quindi dobbiamo farci restituire le umidità delle varie città e quindi dobbiamo agire sul bottone “Umidità”.

Per fare questo andiamo nel controller e gestiamo il metodo “doCalcolaUmidita”. Per calcolare l’umidità dobbiamo conoscere la città su cui calcolarla e il mese scelto dal menu a tendina. Per ottenere il mese basta scrivere il seguente codice molto semplice:



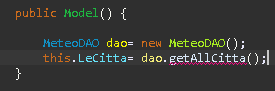
Per prendere le città, dobbiamo scorrere sulle città stesse e prenderle dal Database e quindi ci servirà un qualcosa che dica di andare ad accedere al Database. Inseriamo quindi un “for” con il quale accediamo alla lista delle città e scorriamo tutte le città per farci dare l’umidità media in un determinato mese.



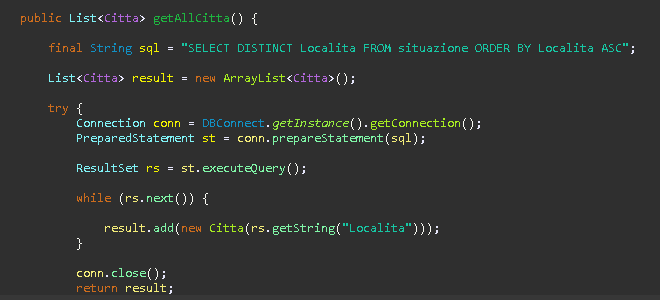
Abbiamo bisogno del metodo “getLeCitta” del model, quindi andiamo ad aprire il nostro Model e iniziamo a popolare la lista di città. Creiamo una lista privata formata da oggetti di tipo “città” (notiamo che noi abbiamo l’oggetto città come classe nel package bean).



Fatto ciò, dobbiamo anche poter accedere al DAO e quindi dentro al model avremo il seguente codice:



Logicamente, ora dobbiamo creare il metodo “getAllCitta” all’interno di “MeteoDAO”. Copiamo il metodo “getAllRilevamenti” in quanto il nostro metodo sarà molto simile a questo. Cambiamo la stringa sql in quanto a noi ci interessa che vengano selezionate sole le località (Città). Aggiungiamo dopo SELECT “DISTINCT” in quanto non vogliamo duplicati nelle città. Cambiamo il nome della lista (che non sarà più “rilevamenti” ma “Citta”) e la chiamiamo result. Il codice per la connessione al database rimane uguale. All’interno del while inseriamo il codice “result.add(new Citta(rs.getString(“Localita”)))”, in quanto noi vogliamo che il metodo restituisca una Località. Infine, noi vogliamo che il programma restituisca “result”.  
Il codice finale sarà quindi il seguente:

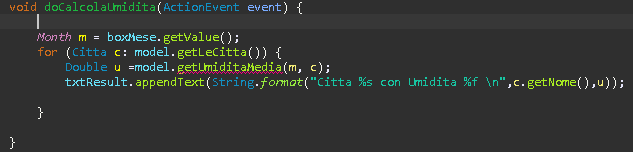


Quindi, quello che otteniamo da questo metodo è la lista di città che ci servivano. Notiamo che ora il model non ci dà più errori in quanto il metodo è stato ultimato.

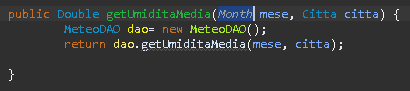
Ritorniamo al “for” che abbiamo iniziato nel Controller. Dobbiamo creare ancora il metodo “getLeCitta” nel model. È un metodo molto semplice infatti il suo unico scopo è quello di ritornare le città.



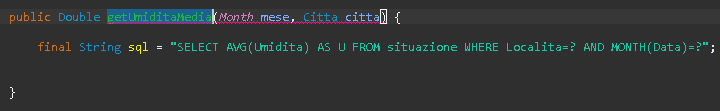
Proseguiamo nel “for” in quanto abbiamo ancora bisogno dell’umidità media. Il programma aveva già in sé il metodo “getUmiditaMedia” che però ci restituiva l’umidità media solo in base al mese e non anche in base alla città. Dovremo quindi andare a modificare questo metodo. Salviamo il risultato del metodo “getUmiditaMedia” all’interno di una variabile Double chiamata “u” e come ultima cosa, visualizziamo il risultato nella casella di testo. Il codice finale sarà il seguente:



Ora dobbiamo andare a modificare il metodo “getUmiditaMedia” in quanto all’inizio restituiva solo il mese (come già detto in precedenza). Esso prima passava un mese che era di tipo “int” mentre ora è di tipo “Month”. Inoltre, ora deve passare anche come oggetto la Citta. Anche questo metodo, così come quello del Model, deve poter accedere al DAO. Aggiungiamo un altro metodo “getUmiditaMedia” che però questa volta sarà all’interno del DAO e che ci calcolerà appunto l’umidità media. Infine, il metodo ci restituirà un Double quindi cambiamo il nome del metodo da “string” a “double”.



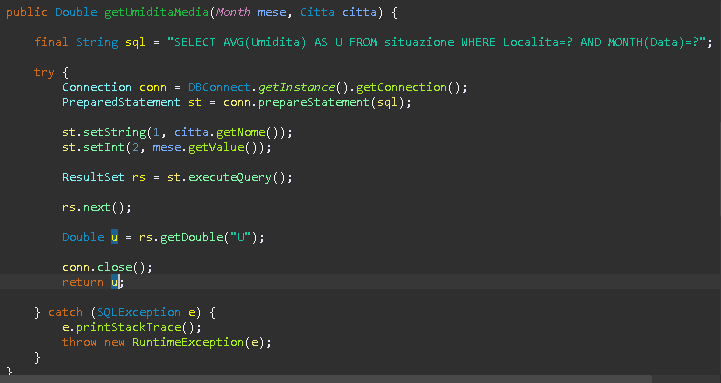
Aggiungiamo questo nuovo metodo a MeteoDAO.



Nella query questa volta avremo dei parametri da gestire (i punti interrogativi). Copiamo il try-catch dai metodi del DAO, per esempio da getAllCitta, e andiamo a eseguire tutte le modifiche necessarie.

Per prima cosa, dopo il PrepareStatement, gestiamo i due parametri, indicando cosa essi rappresentano.

Non avremo un “while(rs.next())” in quanto questo metodo ci restituirà un solo valore per ogni coppia mese-citta. Infine, il metodo dovrà restituirci l’umidità media.



Lo svolgimento del primo esercizio dovrebbe essere finito. Runniamo il programma per vedere se funziona.

Darà un errore in “TestModel”. Trasformiamo le righe che danno errore in commento in quanto sono inutili per il nostro programma.

Il programma funziona, ma vogliamo aggiungere la funziona che, una volta premuto il tasto “Umidità”, ci permette di cancellare le righe precedenti per mantenere la casella di testo pulita. Aggiungiamo in “doCalcolaUmidita” “txtResult.clear()”.

Passiamo ora allo svolgimento del secondo punto.

L’esercizio chiede di svolgere questo punto mediante un metodo ricorsivo. La ricorsione va scritta nel Model. All’interno di esso quindi aggiungiamo due metodi:

* Un metodo pubblico che rappresenta uno schermo pubblico alla ricorsione
* Un metodo privato che è la ricorsione vera e propria.

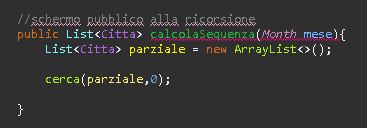
Nel metodo pubblico definiamo la lista parziale che useremo come oggetto da passare nel metodo privato (ricordarsi che è un ArrayList), e chiamiamo il metodo privato passandogli la lista parziale e il livello 0.

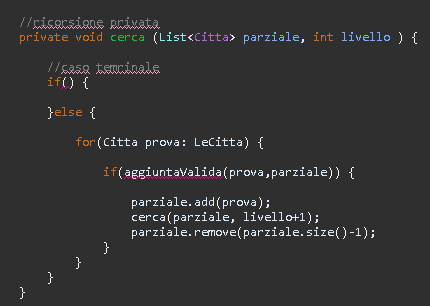
Nel metodo privato della ricorsione, adottiamo sempre la stessa forma:

* Caso terminale con un “if”.
* Un for che cicla le varie città e nel quale analizziamo se la città aggiunta è accettabile oppure no.

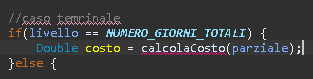
Il vincolo da rispettare (if) è molto complicato e per questo motivo lo deleghiamo ad una funzione apposita. Se il metodo restituisce esito positivo, aggiungiamo il valore nella lista parziale (inizialmente vuota) e richiamiamo il metodo ricorsivo con “livello+1”.

Come ultima cosa facciamo il Backtrack togliendo l’ultimo elemento aggiunto. La situazione sarà quindi la seguente:



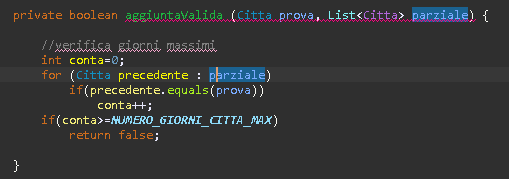


Popoliamo ora il caso terminale. La ricorsione terminerà quando il livello della ricorsione è uguale al numero di giorni totali (15). Nel caso in cui arrivassimo al caso terminale, abbiamo bisogno di una funzione che ci calcoli il coto totale. Quindi:

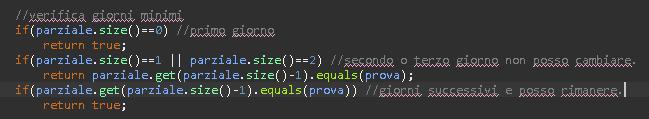


Ora andiamo a scrivere la funzione aggiuntaValida che verifica il filtro (l’if) della ricorsione. La andiamo a creare dopo la ricorsione privata. È una funzione che ci restituirà un booleano. Questa funzione passerà un valore di prova di “città” e una lista di “citta” che conterrà la soluzione parziale.  
In questa funzione creiamo un contatore che ci servirà successivamente.

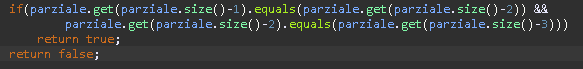
Creiamo un for che scorrerà sulla lista parziale e che chiederà se il valore precedente della lista è uguale al valore che stiamo provando ad aggiungere. In caso positivo incrementiamo il contatore. In pratica questa parte ci controlla se stiamo superando i 6 giorni che possiamo passare in una città consecutivamente.



Ora verifichiamo la condizione sui giorni minimi. Il codice è il seguente (niente di difficile):



Infine, controllo se posso cambiare città; devo controllare la città sia nella posizione precedente che nella posizione due volte precedente:

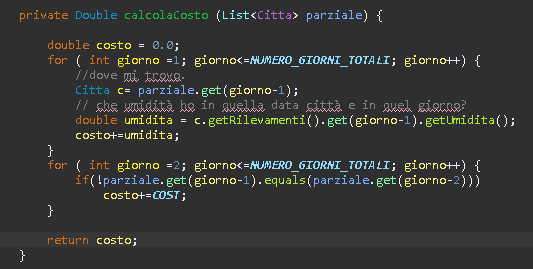


Abbiamo finito di gestire questo metodo per il controllo, adesso passiamo a quello che calcola il costo totale.

La funzione per il calcolo del costo è una funzione double che ci restituirà il costo finale e a cui passeremo la lista parziale. Ea si baserà su un for che ogni volta andrà ad aggiungere parte del costo in base al giorno.

Per ogni giorno (partiamo dal giorno 0 quindi “giorno-1”) calcoliamo grazie alle funzioni “getRilevamenti” e “getUmidita”, l’umidità della città in quel giorno. Andiamo a sommare questa umidità al costo totale.

Poi considero il caso in cui cambio città e quindi devo aggiungere al costo il valore 100. Gestiamo questa situazione con un altro for che ci controlla se la città in due giorni consecutivi è diversa. In tal caso aggiungiamo la costante (100). Concludiamo la funziona facendoci ritornare il costo.



Ora nel Model avremo ancora un errore dato dalla funzione calcolaSequenza (schermo pubblico della ricorsione). Questo perché la funzione non ci ritorna ancora niente. Creiamo quindi una nuova lista di città chiamata “best”. Questa lista dovrà ritornare dal metodo calcolaSequenza. Questa funzione inoltre dovrà ancora contenere il metodo che permette di calcolare il costo ma non c’è piu tempo. FINE

