LEZIONI 49-50
PERSISTENZA IN JAVA
JDBC E PERSISTENCY DATA ACCESS INTERFACES

Ingegneria del Software e Progettazione Web
Università degli Studi di Tor Vergata - Roma

Guglielmo De Angelis
guglielmo.deangelis@isti.cnr.it

JDBC – 1

- Java DB Connectivity framework
- framework per la connessione e l'interrogazione di DBMS relazionali
- consente di progettare applicazioni Java che siano agnostiche rispetto alla scelta di un DBMS
  - garantisce l'indipendenza di un'applicazione Java rispetto al deployment di differenti tecnologie (relazionali) per la persistenza
- … tuttavia resta nelle responsabilità del programmatore la realizzazione di una completa astrazione rispetto al DBMS
  - nella interazione con JDBC utilizzare solo sintassi SQL standard (e.g. ANSI SQL 2)

JDBC – 2

- l'architettura di JDBC implementa concetti tipici della progettazione O.O.
  - astrazione delle funzionalità offerte attraverso una API comune: `java.sql.*`
  - binding su implementazioni alternative sulla stessa API
    - driver JDBC specifico per ogni DBMS relazionare
      - PostgreSQL JDBC Driver
      - JDBC Driver for MySQL (Connector/J)
      - HSQLDB JDBC Driver
      - UCanAccess an open-source JDBC Driver for Microsoft Access

macro classi per i driver JDBC

- Tipo 1: JDBC-ODBC Bridge
  - ODBC: Open DB Connectivity, legacy standard per l'interconnettività di DBMS.
  - le invocazioni JDBC sono mappate in ODBC
- Tipo 2: Driver Nativi
  - driver scritti in C, la loro invocazione richiede installazioni locali all'applicazione e dipndenti dalla piattaforma di esecuzione
- Tipo 3: Network Protocol Driver
  - scritti in Java, e con possibilità di essere installati remotamente all'applicazione
  - il driver ha la responsabilità di comportarsi da middleware verso uno o più DBMS
- Tipo 4:
  - scritti in Java e pensati per risiedere nella stessa macchina dove gira l'applicazione

una applicazione JDBC

- struttura tipica:

```
1. load a driver for a specific DBMS
2. connect to a DB
3. foreach query to the DB {
4.    instantiate an interaction Statement
5.    query the DB
6.    do something with the results
7. }
8. close the connection with the DB
```

una applicazione JDBC

- struttura tipica:

```
1. load a driver for a specific DBMS
2. connect to a DB
3. {
4.    instantiate an interaction Statement
5.    query the DB
6.    do something with the results
7. }
8. close the connection with the DB
```

ATTENZIONE:

- E' UNA BUONA PRATICA MINIMIZZARE IL NUMERO DI CONNESSIONI APERTE CON IL DBMS
- SE POSSIBILE APRIRE UNA SOLA CONNESSIONE PER APPLICAZIONE PER CHIUDERLA SOLO QUANDO NON CI SARANNO ULTERIORI INTERAZIONI CON IL DBMS

entità e dati in persistenza

- entità
  - (nel contesto O.O.) sono classi che modellano le astrazioni chiave del dominio applicativo
  - le varie istanze sono relazionate tra loro attraverso riferimenti in memoria
- DAO (Data Access Object)
  - in una applicazione O.O. sono classi particolari che hanno il compito di mediare tra la rappresentazione in memoria delle istanze e la rappresentazione su layer di persistenza (e.g. DBMS relazionali)
  - ad ogni classe entità dovrebbe corrispondere un DAO che ne gestisce il salvataggio ed il recupero delle istanze su layer di persistenza

java.sql.Connection

- rappresenta una sessione di comunicazione con il DB
- è necessaria aprirne una per leggere e scrivere sul DB
- l'apertura della connessione va richiesta al DriverManager specificando:
  - url del DBMS
  - username e password sul DB

```
DriverManager.getConnection(
"jdbc:mysql://address/database_name",
"database_username", "database_password");
```

`java.sql.Statement`

- astrae il concetto di operazione sul DB
  - deve essere usato per incapsulare il concetto di query SQL
- gli statement vanno istanziati a partire da una istanza di connessione attiva su un DB:

```
connection.createStatement()
```

java.sql.ResultSet

- modellano i dati in forma tabellare come estratti dal DB
- ResultSet offre l'operazione `next()` che scorre i record estratti dal DB
- per ogni record, si può accedere ai singoli campi del DB
  - per indice della colonna (la numerazione comincia da 1)
  - per nome della colonna (come creato nel DB)
- i valori dei campi possono essere prelevati già convertiti nel tipo di interesse
  - getString(), getDouble(), getInt(),
  - getBoolean()...

```
String s = resultSetItem.getString(1); // preleva la
prima colonna come stringa

int eta = resultSetItem.getInt("eta"); // preleva la
colonna "età" come intero
```

esempi JDBC

VEDERE MATERIALE ALLEGATO ALLA LEZIONE :

it.uniroma2.dicii.ispw.jdbcExamples.*

esercizio JDBC

- Progettare versioni alternative per le classi DAO in:
  - `it.uniroma2.dicii.ispw.jdbcExamples.entities.dao.*`
- che
  - evitino di stabilire ripetute connessioni con il DB
  - aprano connessioni solo se effettivanemte non ne esistono di attive
  - condividano necessariamente lo stesso DB
- suggerimento: sfruttare altre classi e pattern GoF