1

Dati persistenti in Java (1)

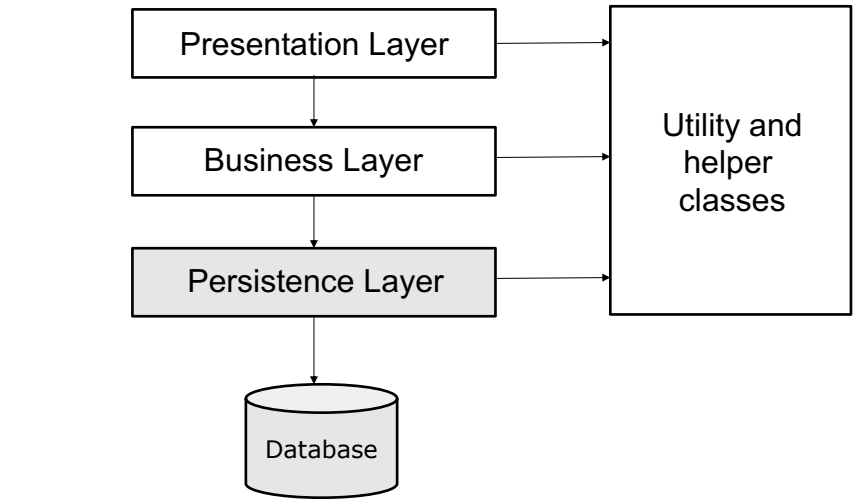
Obiettivi:

- Permettere ad *applicazioni Java* di accedere a dati contenuti in un DB relazionale
  - Query di dati esistenti
  - Modifica di dati esistenti
  - Inserimento di nuovi dati
- ...**in modo uniforme**, indipendentemente dal DBMS

2

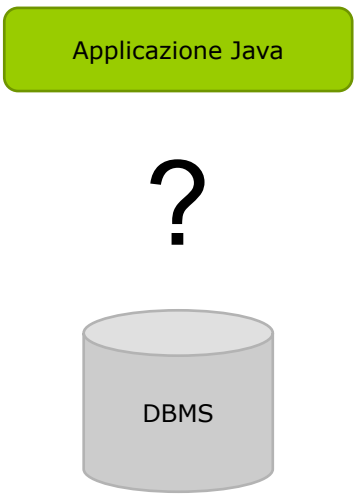
Dati persistenti in Java (2)

Architettura a strati



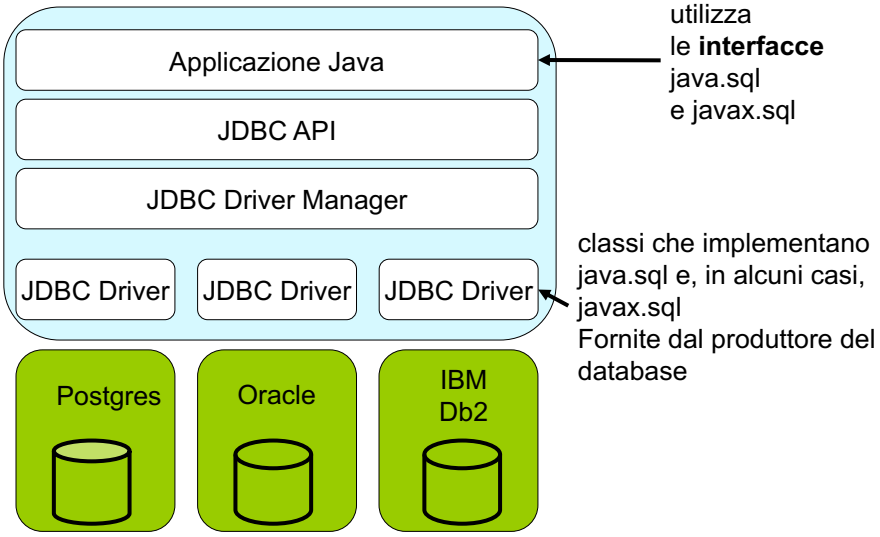
3

Architettura JDBC (1)



4

Architettura JDBC (2)



5

Java Persistence using JDBC (1)

JDBC: Java Database Connectivity

- API standard di Java (pacchetti `java.sql` e `javax.sql`) per accesso a DB relazionali

Dal sito Oracle:

- *The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database*
- *JDBC helps you to write Java applications that manage these three programming activities:*
  - *Connect to a data source, like a database*
  - *Send queries and update statements to the database*
  - *Retrieve and process the results received from the database in answer to your query*

6

JDBC: classi e interfacce fondamentali (1)

Package `java.sql` **(va importato)**

Classe `DriverManager`
Interfaccia `Driver`
Interfaccia `Connection`
Interfaccia `PreparedStatement`
Interfaccia `ResultSet`

7

Driver JDBC: implementazione del vendor

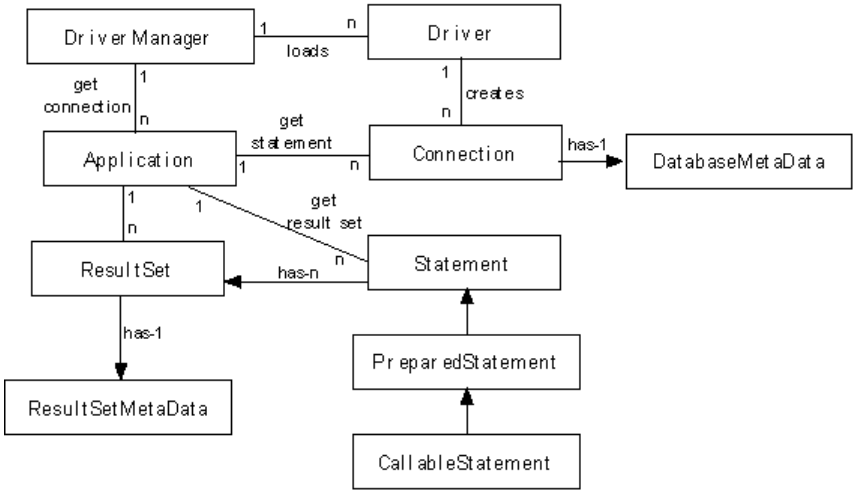
Il vendor del DBMS deve fornire l'implementazione di alcune interfacce definite nel package java.sql:

- Driver;
- Connection;
- Statement;
- PreparedStatement;
- CallableStatement;
- ResultSet;
- DatabaseMetaData;
- ResultSetMetaData.

Normalmente ciò avviene fornendo un file .jar che contiene le classi che implementano queste interfacce.

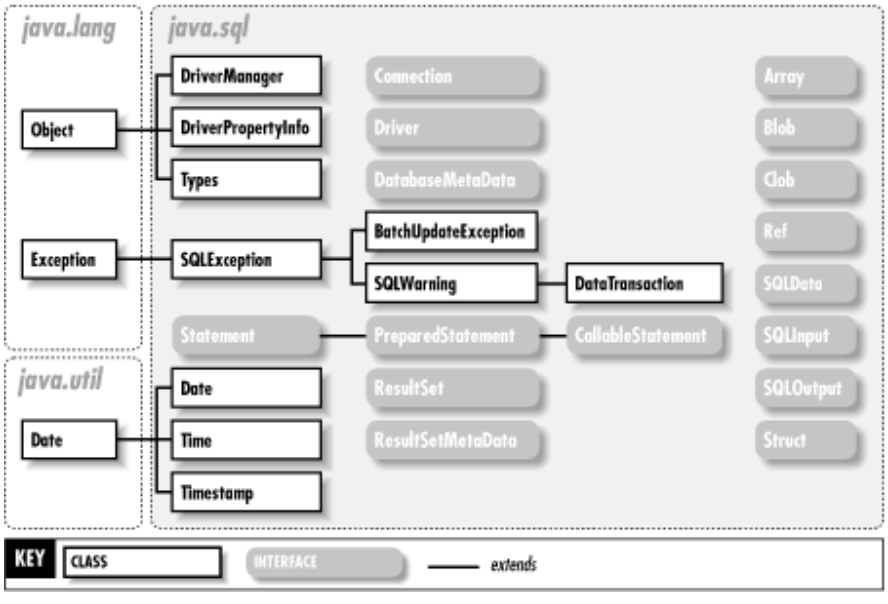
8

JDBC: classi e interfacce fondamentali (2)



9

The java.sql Package



10

Come utilizzare JDBC in short

Passi principali da seguire:

1. Caricare il driver
2. Importare i pacchetti necessari
3. Specificare l'URL per la connessione
4. Aprire una connessione con il database
5. Creare un oggetto Statement
6. Eseguire una query SQL mediante l'oggetto Statement
7. Processare il ResultSet e gestire i risultati
8. Chiudere la risorsa (anche gli oggetti ResultSet e Statement)

Ovvio che:

- Vanno gestite le eccezioni!

11

Step 1

Caricare il driver:

- Il driver in genere è un **file .jar** messo a disposizione dal produttore del DBMS
  - Dovrebbe venire inserito nelle librerie del progetto
  - Dovrebbe essere accessibile dal Class Path del progetto
- Viene utilizzato a run-time (in fase di compilazione non è necessario)

12

Step 1

Caricare il driver:

- Per MySQL: MySQLConnector/J
  - http://dev.mysql.com/downloads/connector/j/
- Per MariaDB: MariaDB Connector/J
  - https://mariadb.com/kb/en/mariadb-connector-j/
  - URL possibili: jdbc:mariadb:// oppure jdbc:mysql://

13

Step 2

Importare i pacchetti necessari:

```
• import java.sql.*;

• import java.sql.ResultSet;
• import java.sql.Statement;
• ....
```

14

Step 3

Specificare l'URL per la connessione

- Per poter attivare la connessione con il DB, il **Driver Manager** deve conoscere:
  - Il *tipo* del DB (per richiamare il Driver corretto)
  - *Dove* si trova il DB (indirizzo del server)
  - *Credenziali* di accesso (username e password)
  - Il *nome del database* al quale collegarsi
- Tutte le informazioni contenute in una **stringa**, detta **URL**

15

Step 3 - Esempio

Specificare l'URL per la connessione

Formato:

```
jdbc:mysql://[host:port],[host:port].../
[database][?propertyName1][=propertyValue1
][&propertyName2][=propertyValue2]...
```

- host: in genere `localhost`
- porta: indicare a meno che non sia la porta standard (3306)
- database: nome del database
- esempi di proprietà: `user=username&password=pippo`

16

Step 4

Aprire una connessione con il database

- Usare **`Connection connection= DriverManager.getConnection(URLString)`**
  - Utilizza il driver appropriato specificato nell'URL
  - Ritorna un oggetto Connection
  - Contatta il DBMS, autentica l'utente con le credenziali dell'URL e seleziona il DB
  - Tutte le operazioni successive andranno fatte sull'oggetto connection che funge da "messaggero" lungo il canale aperto

17

Step 4 - Esempio

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

        try {
            Connection conn = DriverManager.getConnection(
"jdbc:mysql://localhost/test?user=monty&password=secret");

            // Do something with the Connection
           ....

        } catch (SQLException ex) {
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
        }
```

18

Step 5

Creare un oggetto Statement (all'inizio, poi no….)

- Usare **`Statement statement= connection.createStatement();`**
  - crea un *oggetto Statement* che potrà contenere le istruzioni SQL da inviare al DB
  - per query senza parametri va bene, altrimenti usare `PreparedStatement`

19

Step 6

Eseguire una query SQL mediante l'oggetto Statement

- Usare il metodo `executeQuery` della classe `Statement`:
  - **`ResultSet executeQuery(String sql)`**
  - `sql`: stringa che contiene la query (NB: deve essere una `SELECT`)
  - ritorna un oggetto di tipo `ResultSet` da utilizzare per recuperare il risultato della query

20

Step 6 - Esempio

```
String query = "SELECT id, name FROM user" ;
ResultSet resultSet =
statement.executeQuery(query) ;
```

21

Classe `Statement`: altri metodi

- `int executeUpdate(String sql)`
  - Per istruzioni di INSERT, UPDATE, o DELETE o istruzioni che non ritornano una tabella come risultato della query
  - Ritorna o il numero delle righe inserite o eliminate, oppure 0
- `boolean execute(String sql)`
  - Per altri tipi di query

22

Step 7

Processare il ResultSet e gestire i risultati

- L'oggetto `ResultSet` è una sorta di puntatore ai risultati della query
  - I dati sono disponibili una riga per volta (metodo `ResultSet.next()`)
  - I valori delle colonne (della riga selezionata – a cui punta il puntatore) sono disponibili usando metodi della forma `getXXX`, dove XXX è il tipo di dato
    - `getInt, getString, getBoolean, getDate, getDouble, ...`
  - I tipi dei dati sono convertiti da tipi SQL a tipi Java

23

Step 7 - Esempio

Processare il ResultSet e gestire i risultati

`getXXX(int columnIndex)`

- number of column to retrieve (starting from 1 – *beware*!)

`getXXX(String columnName)`

- name of column to retrieve
- Always preferred

```
while( resultSet.next() )
{
    out.println(
       resultSet.getInt("ID") + " - " +
       resultSet.getString("name") ) ;
}
```

24

SQL Types/Java Types Mapping

SQL Type	*Java Type*
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.Math.BigDecimal
DECIMAL	java.Math.BigDecimal
BIT	boolean
TINYINT	int
SMALLINT	int
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

25

Step 8

Chiudere la risorsa (anche gli oggetti ResultSet e Statement)

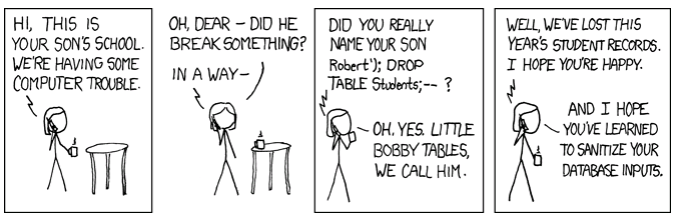
- `connection.close();`
- `resultSet.close()`

26

ALCUNE CONSIDERAZIONI DI SICUREZZA...

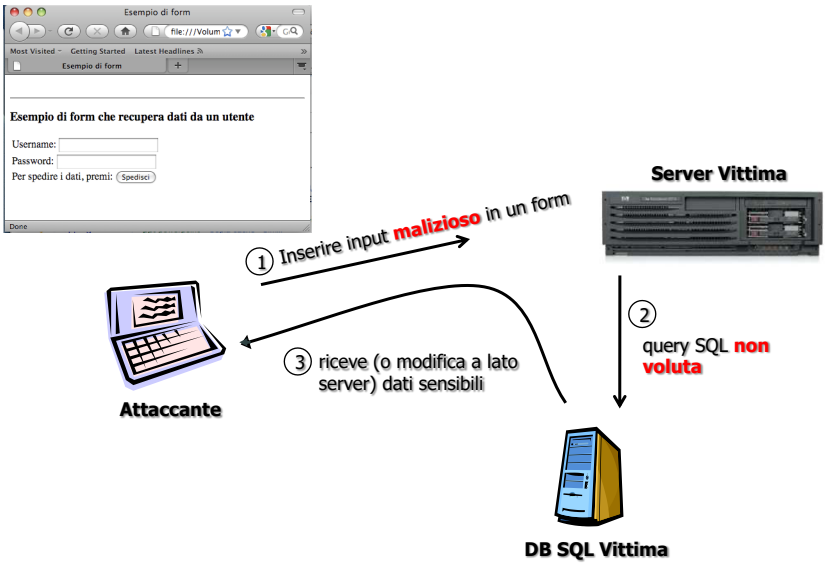
27

Back to Statements



28

SQL Injection - Funzionamento di base



29

Prepared Statement e Bind Variable

Idea di base: fare in modo che dati e elementi di controllo rimangano distinti

***Prepared Statement*:**

- template statico di una query SQL con parametri (bind variable) che vengono sostituiti con i valori reali in fase di esecuzione

Bind Variable

- ? segnaposto che garantisce si tratti di dati (non di controllo)

30

Esempio di Java Prepared Statements (1)

```
PreparedStatement stmt =
connection.prepareStatement("SELECT * FROM users
                             WHERE userid=? AND password=?");
stmt.setString(1, userid);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

- Il posto assegnato a ciascun valore in input (immesso dall'utente) è indicato dal **?**
- Le bind variable sono tipate: i vari metodi **`set*()`** sono utilizzati per verificare che il tipo del dato di input sia quello richiesto

31

Esempio di Java Prepared Statements (2)

```
String query = "SELECT * FROM users WHERE userid ='"+ userid +
   "'" + " AND password='" + password + "'";
PreparedStatement stmt = connection.prepareStatement(query);
ResultSet rs = stmt.executeQuery();
```

- **Codice vulnerabile!**
- Anche se usa i prepared statement crea le query dinamicamente tramite concatenazione di stringhe.

32

Reminder

I dati salvati in un database vengono salvati (e spediti al database) in chiaro…ha sempre senso?

33

QUALE RELAZIONE TRA OGGETTI E DB?

34

Java Persistence using JDBC (1)

- Richiede la creazione manuale delle *query SQL* che sono **stringhe** che vengono passate come argomenti a delle funzioni
  - una sorta di ***traduzione da classi Java a istruzioni SQL***
  - da fare per ogni classe del modello che necessita persistenza e per ogni operazione sul DB
  - PROBLEMA: ***paradigm mismatch*** (detto anche Object/Relational mismatch)

35

Paradigm mismatch (1)

- Associazioni
  - In OO sono direzionali (mediante riferimenti ad oggetti), i RDBMS usano le chiavi esterne, che non sono direzionali
  - In OO sono possibili associazioni *molti-a-molti*, in RDBMS sono possibili solo associazioni *uno-a-molti (o uno-a-uno)*

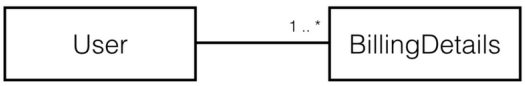
36

Paradigm mismatch (2)

In Java:

```java
public class User {
  private String username;
  private String name;
  private String address;
  private Set billingDetails;
  // metodi accessori getter/setter, metodi business, etc.
  ...
}
```

```java
public class BillingDetails {
  private String accountNumber;
  private String accountName;
  private String accountType;
  private User user;
  // metodi accessori getter/setter, metodi business, etc.
  ...
}
```



37

Paradigm mismatch (2)

In RDBMS:

```sql
create table USERS (
  USERNAME varchar(50) not null primary key,
  NAME varchar(100) not null,
  ADDRESS varchar(100)
)
```

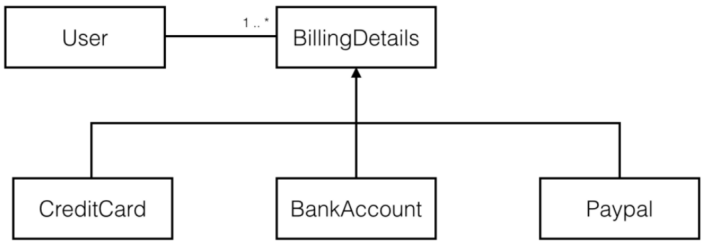
```sql
create table BILLING_DETAILS (
  ACCOUNT_NUMBER varchar(15) not null primary key,
  ACCOUNT_NAME varchar(100) not null,
  ACCOUNT_TYPE varchar(2) not null,
  USERNAME varchar(50) foreign key references user
)
```

La relazione tra le due entità è rappresentata con la chiave esterna USERNAME in BILLING_DETAILS

38

Paradigm mismatch (3)

- Ereditarietà e Polimorfismo
  - In RDBMS non hanno un modo standard e nativo di venire rappresentate



- Anche se SQL supporta le supertable e le subtable, l'ereditarietà applicata alle classi è un'ereditarietà di tipo e una tabella non è un tipo
- Un vincolo di tipo foreign key può puntare ad una tabella, non a più tabelle

39

Quindi?

- Progetto il database **DOPO** avere stabilito il diagramma delle classi
- Non ci deve essere un mapping preciso classe-tabella

40

Altri problemi?

Codice molto confuso e poco organizzato:

- Informazione di basso livello (connessione al DB, creazione delle query, ecc.) insieme al resto dell'applicazione
- Poca portabilità e manutenibilità
- Codice complesso da leggere (e da capire!)

41

Soluzioni?

Design Pattern DAO (Data Access Object)

Obiettivo:

- Separare in classi ad hoc tutto il codice che gestisce gli accessi al DB
- Classi dell'applicazione indipendenti dalle classi di interazione con DB
  - Possibile riutilizzare parti?

42

DAO (1)

Classi separate:

- **Classi "Client"**
  - Classi che devono accedere al database
  - Classi che devono ignorare i dettagli legati all'interazione con il DB: connessione, query, schema del DB, ecc.
- **Classi "DAO"**
  - Contengono (e incapsulano) tutti gli accessi al DB
  - Sono gli unici che interagiscono direttamente con il DB
  - Ignorano di cosa si occupano le classi Client (e.g., come usano i dati)

43

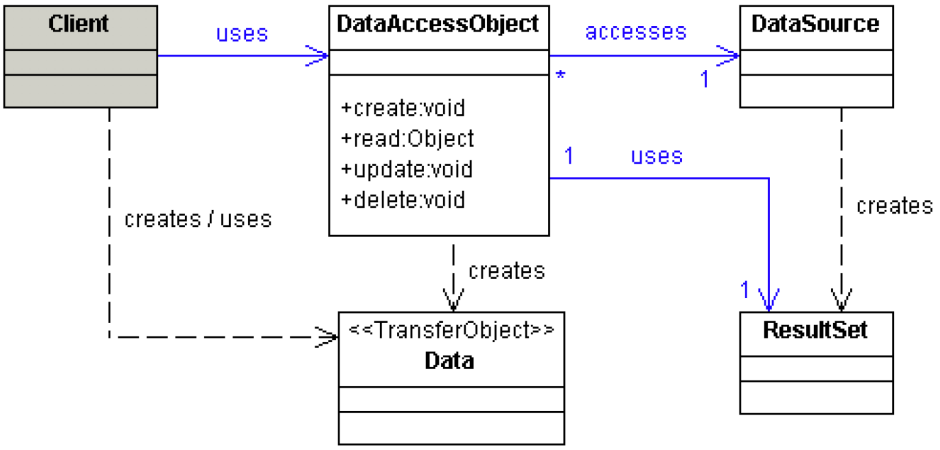
DAO (2)

Classi separate:

- **Classi "DTO" (Data Transfer Object)**
  - Contengono dati spediti dal Client al DAO e/o ritornati dal DAO al Client
  - In genere rappresentano il modello dei dati, così come visto dall'applicazione
  - Non conoscono DAO, Client e DB (sono POJO, Plain Old Java Object)

44

DAO (3) – Diagramma delle classi



45

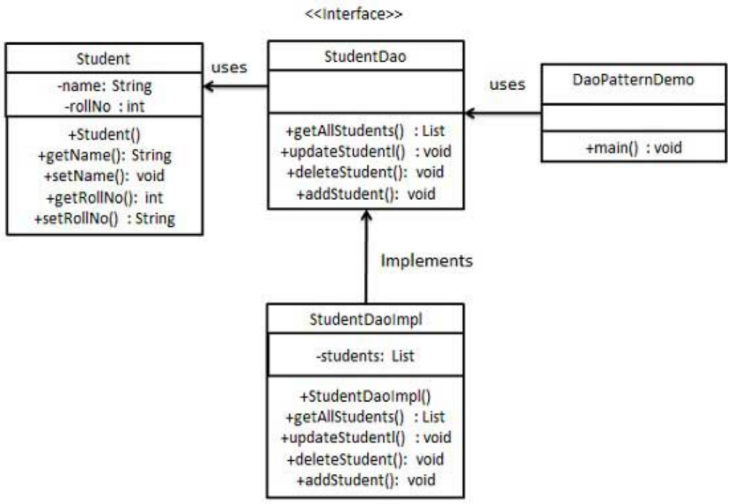
DAO (4)

Data Access Object Pattern or DAO pattern is used to separate low level data accessing API or operations from high level business services. Following are the participants in Data Access Object Pattern.

- **Data Access Object Interface** - This interface defines the standard operations to be performed on a model object(s).
- **Data Access Object concrete class** - This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- **Model Object or Value Object** - This object is simple POJO containing get/set methods to store data retrieved using DAO class.

46

DAO (5)



Fonte:
https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

47

JDBC - Documentazione

- http://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/index.html
- JDBC FAQ: https://www.oracle.com/database/technologies/faq-jdbc.html
- Tutorial: http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

48