
# Università degli Studi di Bologna
## Facoltà di Ingegneria

*Tecnologie Web L-A*
*A.A. 2009 – 2010*

## Esercitazione 6
## Database

**Tutor:**
Ing. Pasini Samuele *samuele.pasini@unibo.it*

# Agenda

- Pattern DAO
  - Concetto di pattern
  - Data Access Object e Data Transfer Object
  - Funzionamento

- Esperienza pratica
  - Un primo progetto di esempio (il negozio online riveduto e ampliato)
  - Esempi da ricreare

# Pattern DAO

- Un “pattern” è un “*modo di fare le cose*” che si è dimostrato efficace e che per questo si tende ad applicare di nuovo

- Il pattern “DAO” rappresenta “**IL**” modo di separare tra loro:
  - logica di business (es: Servlet, pagine JSP, …)
  - logica di persistenza (es: scritture su DB, letture, …)

- Infatti, i componenti della logica di business non dovrebbero mai contenere codice che accede direttamente al database!
  - scarsa manutenibilità
  - sovrapposizione di responsabilità

- Solo gli oggetti previsti dal pattern DAO...
  - hanno il permesso di “**vedere**” il DB
  - espongono metodi di accesso per tutti gli altri componenti

# Principi fondamentali

- I valori scambiati tra DB e resto dell'applicazione sono racchiusi in oggetti detti Data Transfer Object (DTO):
  - campi privati per contenere i dati da leggere/scrivere su db
  - metodi getter e setter per accedere dall'esterno a tali campi
  - metodi di utilità (confronto, stampa, calcolo dell'hashcode, …)

- Le operazioni che coinvolgono tali oggetti sono raggruppati in interfacce che definiscono i Data Access Object (DAO) disponibili:
  - metodi Create, Read, Update, Delete (CRUD)
  - altri metodi (tipicamente di lettura con parametri custom)

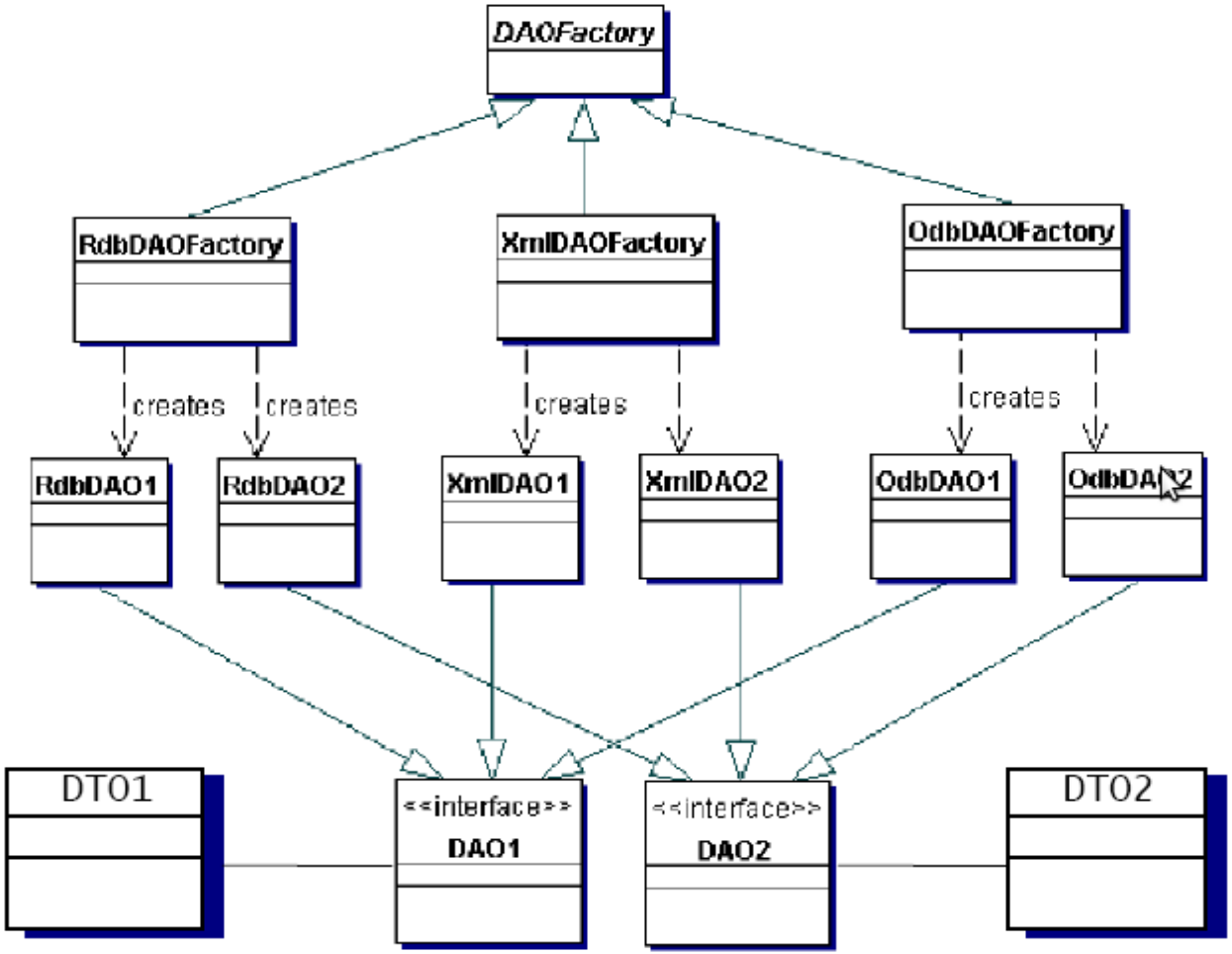
- Le implementazioni di tali interfacce (spesso indicate come oggetti DAO, per brevità) permettono l'effettivo accesso al database
  - diverse implementazione possono permettere (e rendere uniforme) l'accesso a DB di diversi vendor
  - anche se si fa uso di un solo database, tale separazione migliora comunque la divisione delle responsabilità tra le parti dell'applicazione
  - diventa facile migrare l'applicazione su DB diversi

- Gli oggetti DAO non sono istanziati direttamente dai componenti, ma:
  - possono essere ottenuti attraverso metodi factory (come faremo noi)
  - sono settate nei componenti da qualche altra entità (es: il container)

# Pattern DAO con classi Factory

- Un'unica factory astratta
  - fornisce specifiche per le factory concrete
  - espone un metodo creazionale parametrico per ottenere factory concrete
- Una factory concreta per ogni tipo di database supportato
  - permette di ottenere oggetti DAO appropriati al corrispondente tipo di database
  - può gestire aspetti quali ottenimento della connessione, autenticazione, ...
- Un oggetto DTO per ogni tipo di entità che si vuole rappresentare
- Un'interfaccia DAO per ogni oggetto DTO
- Un'implementazione dell'interfaccia DAO di un DTO per ciascun database supportato

# Pattern DAO con classi Factory - Schema UML



# JDBC

- Resta ancora da capire cosa scrivere nel codice degli oggetti DAO veri e propri
  - la risposta (per noi) è JDBC
  - esistono altri (e piu' potenti, ma complessi) strumenti per il mapping object-relational (es: JPA)

- JDBC = Java DataBase Connectivity
  - Java fornisce una API standard per formulare statement SQL verso database relazionali

- I driver di ciascun database implementano le specifiche traducendo gli statement SQL (stringhe Java composte in accordo a determinate convenzioni) in istruzioni e comandi per il database e permettono di leggerne I risultati
  - hsqldb.jar (*HSQLDB*)
  - mysql-connector-xxx.jar (*MySql*)
  - ecc…

- Gli statement SQL tuttavia...
  - non sono sempre perfettamente “portabili”: differenti database spesso parlano diversi dialetti
  - non permettono ottimizzazioni

- …ed ecco perché il pattern DAO!

# Giunti a questo punto...

- Uno sguardo al codice vale più di mille parole...

- All'interno dell'archivio ZIP dell'esercitazione, nel direttorio ***progetti***
  - il file ***06_TemplateDAO.zip*** contiene lo scheletro di un semplice progetto di esempio basato sull'uso del pattern DAO
  - creato con Eclipse, contiene già tutti i descrittori necessari a essere riconosciuto e configurato correttamente

- Importare il progetto come visto nelle precedenti esercitazioni, senza esploderne l'archivio su file system (lo farà Eclipse)
  - *File → Import → General → Existing Projects into Workspace → Next → Select archive file*

# TemplateDAO

- Il progetto riprende, sviluppa e completa il negozio online visto durante l'esercitazione sul tema JSP
  - il catalogo dei prodotto è diventato la tabella degli articoli in vendita
  - l'oggetto `it.unibo.tw.dao.ItemDTO` funge da trasporto tra l'applicazione e questa tabella
  - il carello dei prodotto scelti dall'utente è invece OVVIAMENTE rimasto un bean con scope di sessione
  - esiste poi una tabella degli ordini (e DTO corrispondenti) per memorizzare in modo persistente gli acquisti confermati dall'utente
  - sono previste due versioni degli oggetti DAO.. una per HSQLDB (completa) e una per MySQL (appena accennata)
    - ai fini di questa semplice applicazione, i due DB differiscono soltanto nelle modalità di accesso all'ID dell'ultima tupla inserita
  - la scelta dell'implementazione da utilizzare dipende da un particolare parametro DICHIARATO nel descrittore *web.xml*

- *All'interno del progetto, individuate in quali sezioni vengono richiamati ed utilizzati gli oggetti DAO*
  - nel **codice dell'applicazione Web**, all'interno di Servlet e pagine JSP
  - nelle **routine di test** (molto comode per potersi “fidare” di avere scritto oggetti DAO corretti e funzionanti prima di sviluppare il resto della logica di controllo e presentazione)
  - all'interno di **classi con metodi `main()`**, ai fini di impostare il contenuto iniziale del DB in maniera programmatica

# TemplateDAO

- **Il database è un programma autonomo e non ha niente a che fare né con l'IDE né con il Web Server** (anche se è possibile avviarlo dall'IDE o da ANT)
  - potrebbe addirittura eseguire su un host diverso da quello del Web Server
    - viene infatti acceduto attraverso socket
  - per semplicità, scegliamo di usare HSQLDB
    - interamente realizzato inJava
    - Attenzione!!! il programma database vero e proprio ed il suo connettore (driver che implementa la API JDBC) sono contenuti nello stesso archivio: *hsqldb.jar*
  - utilizzate il target di ANT per avviare il DB

- I driver del database sono un tipico esempio di libreria sulla cui ubicazione è necessario riflettere
  - servono in Eclipse per i test, ma anche in Tomcat per l'esecuzione
  - dove vi aspettate di trovarli, quindi?
  - perché le factory li richiamano per nome, anziché istanziarli direttamente?

# Primi passi...

- Se avete avviato il database...
  - provate a eseguire le routine di test (accesso CRUD e altre query)
  - provate a inizializzare lo stato del db attraverso la classe con metodo `main()`
- Lo stato del database viene salvato su file system all'interno del direttorio *db* del progetto
  - un modo veloce per “ripulire” il db è quindi quello di **arrestarne l'esecuzione** e **cancellare il contenuto di questo direttorio**!
- Provate ora a seguire l'esecuzione tenendo d'occhio il codice degli oggetti DAO che vengono chiamati in causa!
  - ad esempio facendo debug!
  - avete riavviato il database?
  - non lasciatevi impressionare... il copia e incolla è uno strumento potente...A PATTO DI AVERE CHIARO COSA SI STA COPIANDO E INCOLLANDO!

# Implementazione degli oggetti DAO

- Il consiglio è mio personale, ma questo modo di procedere nello scrivere il codice degli oggetti DAO basati su JDBC è… a prova di bomba!

- Ogni metodo…
  - *1. dichiari una variabile dove collocare il proprio risultato*
  - *2. controlli la bontà dei parametri attuali ricevuti*
  - *3. apra la connessione al DB*
  - *4. formuli gli statement SQL che lo riguardano e imposti il risultato*
  - *5. preveda di gestire le eventuali eccezioni*
  - *6. rilasci SEMPRE E IN OGNI CASO la connessione in uso*
  - *7. restituisca il risultato (eventualmente di fallimento)*

- E per quanto riguarda gli statement SQL veri e propri
  - *1. crei (se senza parametri) o prepari (se con parametri) lo statement da proporre al DB*
  - *2. pulisca e imposti i parametri (se ve ne sono, ovviamente)*
  - *3. esegua l'azione sul DB ed estragga il risultato (se atteso)*
  - *4. cicli sul risultato (se presente) per accedere a ogni sua tupla e impostare il proprio risultato con i valori in essa contenuti*
  - *5. rilasci la struttura dati del risultato stesso*
  - *6. rilasci la struttura dati dello statement*

- Potete ritrovare questo schema di operazioni nei commenti (tanti!) a corredo del codice

# Accesso al database da parte dell'applicazione Web

- Attraverso i target ANT, compilare, pacchettizzare e pubblicare sul server l'applicazione Web contenuta nel progetto...

- ...e avviare Tomcat!

- La pagina di costruzione del catalogo *(catalogue.jsp)* è cambiata rispetto al progetto visto durante l'ultima esercitazione! Ora la lista dei prodotti inseriti è letta e modificata attraverso gli oggetti DAO!
- La conferma degli ordini (*checkout.jsp*) è invece stata completata: il contenuto del carrello viene letto dalla sessione corrente e trasformato in un DTO (l'ordine confermato) da salvare su db
  - **lettura e comprensione**
  - **domande/dubbi?**
  - riuscite a modificare la pagina di gestione del carrello degli acquisti (*cart.jsp*) dell'esercitazione precedente (ma presente anche nel progetto di oggi) per mostrare i prodotto del catalogo su database, selezionarli e aggiungerli al carrello in sessione, e quindi procedere con l'ordine?