



Alma Mater Studiorum - Università di Bologna  
CdS Laurea Magistrale in Ingegneria Informatica  
I Ciclo - A.A. 2017/2018

**Sistemi Distribuiti M**  
**Proposte di Esercizio- Introduzione Generale**

A cura di:

Stefano Monti, [smonti@imolinfo.it](mailto:smonti@imolinfo.it)  
Samuele Pasini, [samuele.pasini@gmail.com](mailto:samuele.pasini@gmail.com)  
Paolo Bellavista, [paolo.bellavista@unibo.it](mailto:paolo.bellavista@unibo.it)



Uno ***stesso dominio applicativo*** alla base di tutti gli esercizi proposti

- ❑ ***Gestione di una biblioteca***
- ❑ Estrapolato e ridotto da una applicazione Web distribuita ***reale***

Occasione per sperimentare diverse tecnologie di integrazione

- ❑ ***EJB3*** (integrazione e remoting)
- ❑ ***Hibernate & JPA*** (mapping object-relational)
- ❑ ***JMX*** (monitoraggio e gestione)
- ❑ ***SpringFramework*** (integrazione mediante IoC, utilizzo di AOP)
- ❑ ***Java Business Integration*** (integrazione su Enterprise Service Bus Apache ServiceMix)



Sviluppo di componenti software in grado di eseguire

- ❑ All'interno di **applicazioni Web**
  - Offerta di servizi a chi effettua delle richieste
  - Esecuzione supportata da un Web server o da un application server
- ❑ All'interno di **ambienti di test**
  - Lotti batch di operazioni e verifica di correttezza
  - Esecuzione supportata da una suite di test
- ❑ All'interno di **applicazioni stand-alone**
  - Dotate di proprio *main()*
  - Ad esempio impiegate per l'inizializzazione dei database con dati di prova



# Dominio del Problema

Modellazione delle seguenti entità e reciproche relazioni

- ❑ *Autori*  $n \leftrightarrow m$  *Libri*  $n \leftrightarrow 1$  *Editori*

Necessità di disporre di componenti in grado di fornire  
**accesso allo strato di persistenza**

- ❑ Racchiudere, isolare e nascondere le modalità di accesso a DB
- ❑ Astrarre le diverse tipologie di realizzazione (MySQL, Hsqldb, ...)
- ❑ Permettere agli altri componenti di trattare i dati in termini di *Plain Old Java Object* (POJO), senza curarsi di come questi siano mappati

***Adozione del pattern Data Access Object (DAO)***

- ❑ Interfacce che stabiliscono i metodi di lettura/scrittura
- ❑ Pattern creazionali (es: factory) per l'ottenimento delle implementazioni

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>



Utilizzare le tecnologie citate per

- ❑ **Implementare gli oggetti DAO veri e propri**, ove necessario
  - Fornita un'implementazione JDBC di esempio
  - Richieste re-implementazioni via Hibernate e/o JPA
- ❑ Permettere la loro **integrazione con le altre parti del sistema**
  - Funzionalità proprie della logica di business
  - Componenti Web
  - Suite di test
  - Ambiente runtime
  - Java Console
  - ...
- ❑ Verificare le **potenzialità e criticità di ciascuna tecnologia**
  - Innalzamento del livello di astrazione
  - Semplificazione del codice
  - Migliorata manutenibilità
  - Difficoltà di adozione
  - Aumento della complessità dell'applicazione
  - ...



# Come Procedere (1)

Tutti gli esercizi prendono spunto dallo **stesso codice sorgente iniziale**

- ❑ Modellazione del dominio del problema
  - Specifica delle interfacce DAO
  - Definizione degli oggetti POJO scambiati attraverso di esse
- ❑ Inizializzazione della base di conoscenza
  - Classi con metodi *main()*
  - Utilizzo dei pattern DAO e factory per accedere al DB
  - Scrittura, cancellazione e riletture di informazioni predefinite
- ❑ Set minimale di test
  - Basati su JUnit4
  - “Declinabili” per ciascuna tecnologia
  - Estensibili su iniziativa dello studente



# Come Procedere (2)

- Applicazione Web minimale
  - Una singola pagina JSP
  - Accesso al DB in lettura e scrittura
  - Presentazione di informazioni e risultati all'utente dotato di browser
  - Base per sviluppare (opzionalmente) altri componenti relativi alla logica di business e presentazione



Il sorgente iniziale è fornito tramite un file ZIP contenente un **progetto gestibile con ANT e/o all'interno dell'ambiente di sviluppo Eclipse**

- ❑ **Importazione in Eclipse** (grazie alla presenza di metadati “specifici”)

*File → Import → General → Existing Project into Workspace →  
Select archive file → [fill in the blank or browse] → Finish*

- ❑ File di **build di ANT** pronti all'uso

*Modificare con i dati relativi alla propria macchina **solamente**  
il file `$PROJECT_ROOT/ant/environment.properties`*

- ❑ Ovviamente è possibile/consigliato importare in Eclipse **e** lanciare i target di ANT dall'interno dell'IDE
  - sfruttare funzionalità proprie dell'IDE  
parsing, autocompilazione, refactoring, autocomplet., ...
  - ottimizzare l'esecuzione di operazioni ripetitive via ANT  
compilazione, packaging, deployment, undeployment, ...
- ❑ Aprite pure il codice del progetto: vale più di mille parole 😊...



In formato analogo al sorgente iniziale, viene fornita ***una possibile implementazione degli oggetti DAO***, basata su tecnologia ***JDBC***

- ❑ API unificata
  - Diverse librerie (“connettori”) forniscono i “driver” che implementano le API per l’accesso ai rispettivi tipi di database server  
*MySQL, Hsqldb, DB2, PostgreSQL, ...*
  - Operazioni eseguite passate stringhe SQL agli oggetti  
Il mapping tra oggetti Java e tabelle è completamente fatto “a mano”!
  - Database diversi spesso supportano “dialetti” diversi per le stesse operazioni, quindi richiedono stringhe SQL diverse  
Una implementazione del DAO, basata su JDBC, per ciascuno!
- ❑ Il progetto d’esempio supporta
  - *MySQL* (connettore fornito, server da installare)
  - *Hsqldb* (connettore fornito, server scritto in Java e contenuto nella stessa libreria del connettore: target di ANT per avviarlo!)
- ❑ Scopo è semplicemente poter disporre di un ***possibile esempio di implementazione delle specifiche!***



# Alcune Puntualizzazioni sul Codice di Partenza

Nel progetto d'esempio fornito, la superclasse `factory` astratta `it.unibo.sdls.sampleproject.dao.DAOFactory` **istanzia una nuova *factory concreta*** a seguito di ogni richiesta di tipo `getDAOFactory (String)`

- ❑ Il ***pattern Singleton*** (creazione la prima volta e restituzione della stessa istanza a tutte le successive richieste) è volutamente ***NON UTILIZZATO***
  - diversa istanza di `factory` ad ogni richiedente, ad es. per raggruppare successive operazioni sugli oggetti DAO all'interno di una sola "conversazione" di cui gestire la transazionalità (v. esercizio su *Hibernate*)
- ❑ Le operazioni "pesanti" sono collocate in parti statiche del codice delle `factory concrete`, così da venire eseguite ***solo la prima volta***
  - Verifica dell'esistenza delle tabelle
  - Eventuale creazione di un pool di connessioni (v. esercizio su *JMX*)
  - ...