Intro, Persistence, Object Relational Mapping and Java

JPA Slides #1

Persistence

Persistent object: Object that outlives the execution of the program

- Have to store for later retrieval (next execution)

Many persistence mechanisms

- Flat files
- Serialization
- XML
- Different types of databases, … we use (simplest possible is to use the JavaDB (aka Derby) bundled with NetBeans)!

Java API's for Persistence

Java database connectivity JDBC

- Low level API, not used
- Using embedded SQL strings as parameters

Java Data Object, JDO 2.0, Spec: JSR 243

- Very (too?) general, relational database, object database , ...
- Not used in course, possible fading away...

Java Persistence API, JPA 2.0, JSR 317

- Supports **only** relational databases
- Built on top of JDBC (JDBC pops up in between)
- **We'll use...**

JPA 2.x

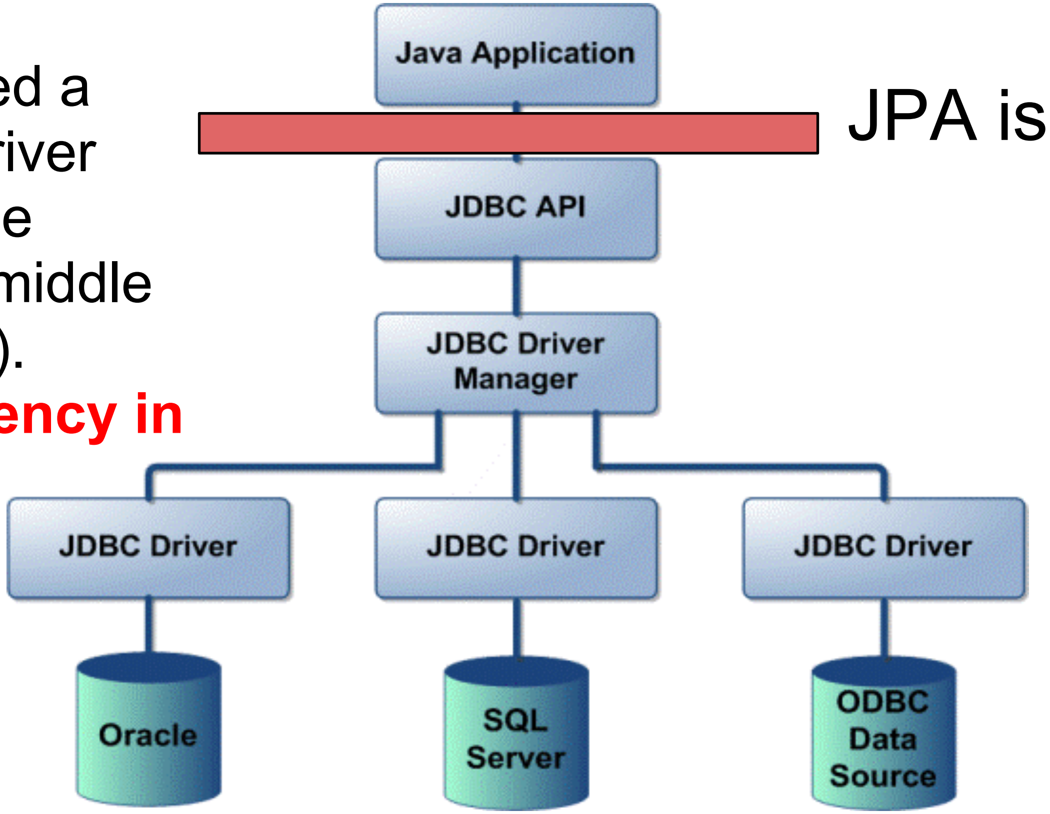
Possible to use in JEE and JSE environments

- Different configurations, service levels
- If EJB container (GlassFish): Few lib. dependencies, less code, more "automagical"
- JSE or Tomcat (not an EJB container) and JUnit (JSE) have to supply dependencies in pom.xml
    - JPA lib's, JDBC driver lib, ...

JDBC Architecture

As noted JDBC will popup in between, need some insights

We'll need a JDBC Driver (database specific middle ware, lib). **Dependency in pom**



JPA is here

The OO-Relational Mismatch

Relational databases and object orientation <u>doesn't fit!</u>

Object orientation: **Objects**

Relational databases: **Sets of tuples**

- No objects, classes
- No inheritance, polymorphism, generics...

Major clash, <u>the OO-relational mismatch</u>

- Relational databases won't change, mathematical foundations...
- Unsolved problem, ...

Handling the Mismatch, Option 1

Surrender : I.e. don't use OO

Possible solution (good for massive reads)

- Example: Product Catalog to web
- Just use primitive types, String, int, ...
- Fastest possible solution
- Not a solution for complex cases

Handling the Mismatch, Option 2

Try to fix the mismatch

- Map between objects and tuples, **object relational mapping**, ORM

No general best strategy

- Must know how database in going to be used
- Mostly reads?
- Mostly writes?
- http://www.agiledata.org/essays/mappingObjects.html

Very complex task to implement

- "Must" use a "framework" to do the mapping, i.e. we use JPA 2.0

ORM Cases to Handle

Associations? Multiplicity! Inheritance? Generics?

Object graphs! Lazy object creation? Lazy fetching?
Caching? Concurrency? Transactions?...

Ad hoc searching

- Possible don't need objects (ex. statistics)

Should database or application do the work?

- Databases <u>very</u> efficient at searching/sorting … we prefer

JavaDB (Derby)

As noted in the crash couse; We use the **Java DB (aka Derby)** bundled with Netbeans
- Create/drop databases from inside Netbeans
- Create/drop tables (all tables should belong to a "schema" APP)
- CRUD or edit tables directly from inside NetBeans (NOTE: Must commit to make persiste, click small button in table heading)
- Run SQL queries from Netbeans
- Sample database supplied (good for testing queries)

Databases stored as files in ~/.netbeans-derby directory
Possible to delete database by erasing files