# Object Relational Mapping and JPA Intro

JPA Slides #1

# Content

- Persistence
- Object relational mapping (ORM)
- Object relational mismatch
- Java Persistence API
- JPA Config files
- JavaDB (Derby)

# 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 (binary)
- XML/JSON (text)
- Different types of databases …
- … we will only use a relational database

# Object Relational Mismatch

Relational databases and object orientation doesn't fit!

- Object orientation: Objects
- Relational databases: Sets of tuples

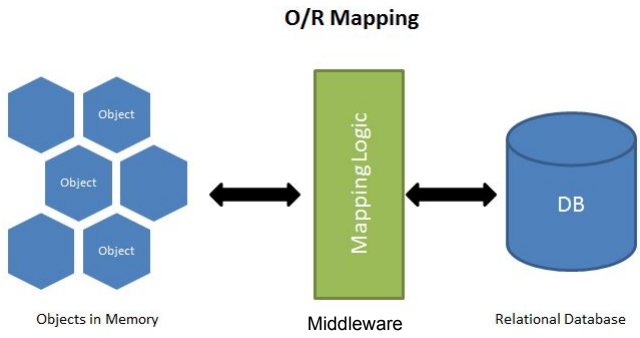
Major clash, the OO-relational mismatch

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

Some points

- Identity? Equality?
- Associations? Relationships? Multiplicity!
- Inheritance?
- Generics?
- Object graphs! Lazy fetching? Lazy object creation?
- Caching? Concurrency? Transactions?...
- Ad hoc searching
  - Possibly don't need objects (ex. statistics)
- Should database or application do the work?
  - Databases very efficient at searching/sorting, etc … we prefer!

# Handling the Mismatch



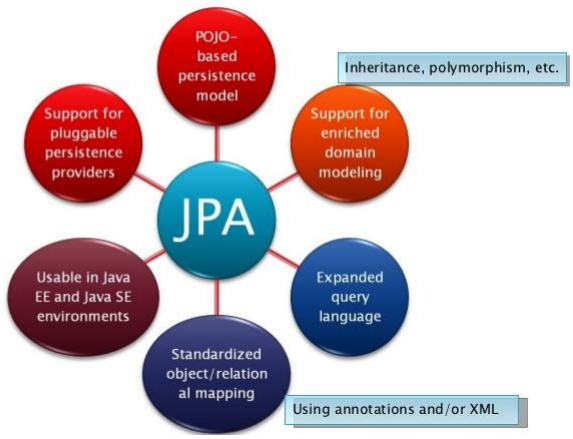

At least three different options

Option 1
- Surrender : I.e. don't use OO
- Just use primitive types, String, int, …
- Good for massive reads
  - Example: Product Catalog to web
- Fastest possible solution
- Not a solution for complex cases
  - Many cases seems simple at start, then complexity creeps up …!

Option 2
- Try to bridge 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?
  - Different strategies
- Very complex task to implement (we don't)
  - We use some **middleware** (glue layer)

Option 3
- Your data in not "relational" use some other persistence approach
- NoSQL …

# Java Persistence API

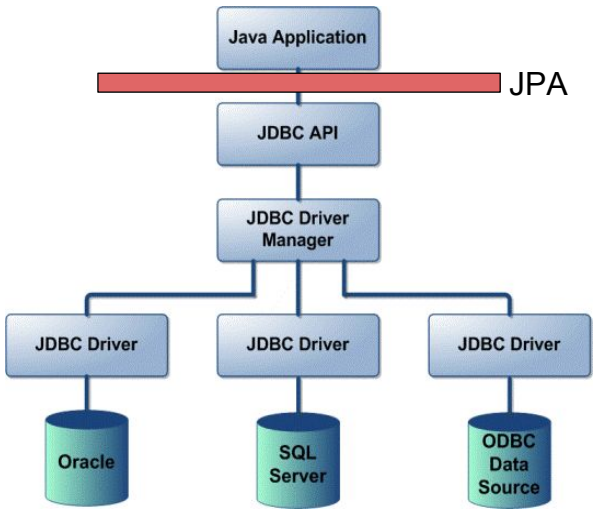



Java APIs for persistence

- Java DataBase Connectivity, JDBC
  - Low level API, no ORM (not used by us)
  - Using embedded SQL strings as parameters
  - JEE spec. makes JDBC mandatory
- Java Data Objects, JDO
  - Very (too?) general, relational database, object database , …
  - Not used in course, possible fading away…?
- Java Persistence API, JPA 2.x
  - Supports relational databases and NoSQL
  - This will be our middleware (glue application and database)
  - "The Java Persistence API (JPA) provides Java developers with an object/relational mapping facility for managing relational data in Java applications."
  - Possible to use JPA in JEE and JSE environments (= JSE, Tomcat or JUnit)
    - JSE Environment
      - Have to supply many dependencies
      - Have to handle a lot in application (more to code)
    - JEE environment, GlassFish, …
      - Fewer dependencies
      - Container will handle a lot. We use!
  - Some Java Persistence API areas:
    - Object/relational mapping
    - The Java Persistence API, to handle persistent objects

- The Query language, to query database in an OO fashion
- The Java Persistence Criteria API, same as above but typesafe

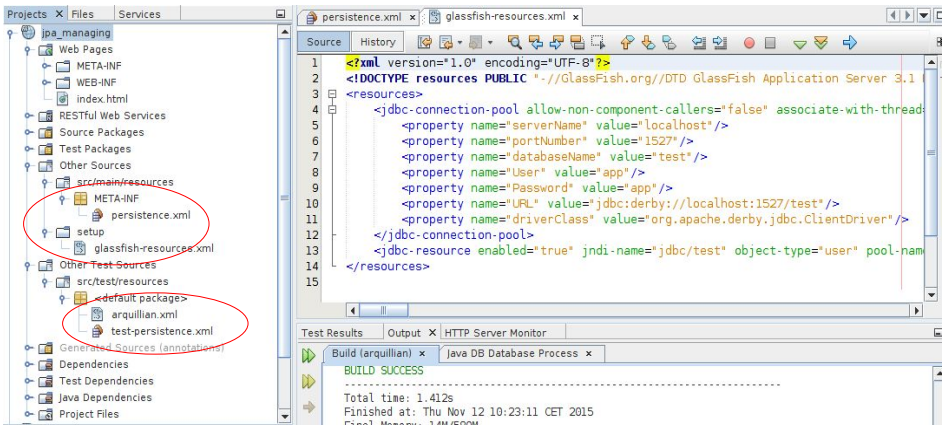
# JPA and JDBC




JPA built on top of JDBC

- We'll need a JDBC Driver (database specific middleware)
- Possibly dependencies in pom (Eclipselink)
- Handled by NetBeans

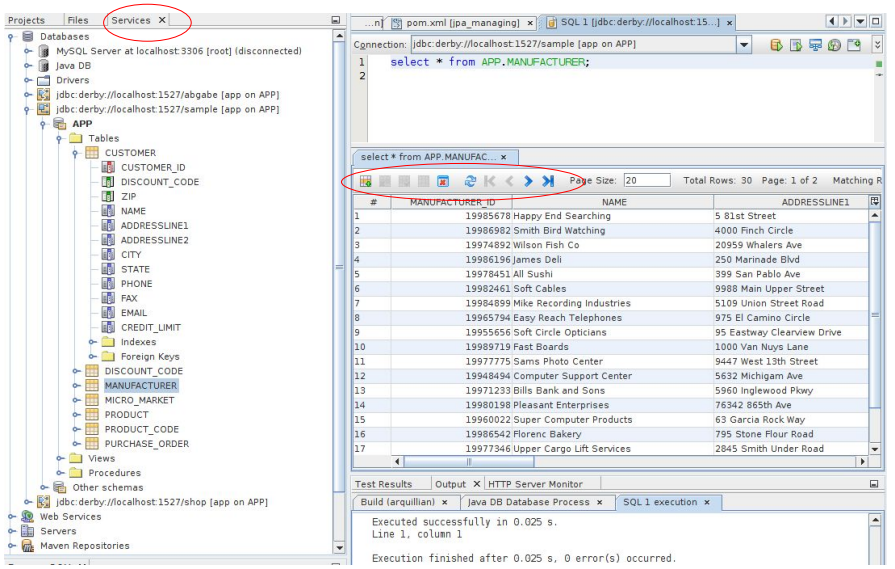
# JPA Config Files




There will be about 4 config files involved

- Mostly generated by NetBeans, possibly some tweaking
- src/main/setup/glassfish-resources.xml,
  - Technical data for the database, location, JDBC driver and more (server specific).
- src/main/resources/META-INF/persistence.xml
  - Containing persistence units (PU)
    - What classes to persist and more
    - Server independent, application specific
- src/test/resources/test-persistence.xml
  - Same as persistence.xml but for testing
- src/test/resources/arquillian.xml
  - Config for embedded EJB container for testing, more later …

# JavaDB (Derby)




For simplicity we'll use JavaDB (aka Derby) as our RDBMS

- Bundled with NetBeans!
  - Will run locally
- Databases stored as files in ~/.netbeans-derby directory
  - Possible to delete database by erasing files
- May use any database in project

Using JavaDB

- Create/drop databases from inside Netbeans
  - For code samples create a database: test
- Create/drop tables (all tables should belong to a "schema" APP)
- Connect to database: Mark database > Right Click > Connect
- Inspect table data: Mark table > Right click > View Data
- CRUD operations on table data from inside NetBeans
  - NOTE: Must commit to make persistent, click small button in table heading
- Run queries from inside Netbeans
- Sample database supplied, good for testing queries