Mapping Java Objects to XML and Relational Databases
O-R and O-X Mapping

Donald Smith
Technology Director, Oracle Corporation
blog:  jroller.com/page/dsmith
Speaker’s Qualifications

- Decade of experience in OO Persistence
- Presented at Java One, Oracle World, OOPSLA, JAOO, Sun Tech Days, TheServerSide Symposium, etc.
- Author of numerous articles on persistence challenges
About the Audience...

- Who considers themselves first and foremost to be a DBA or “Database expert”?
- Who considers themselves first and foremost to be a Java and/or Web Services developer?
- Who considers themselves first and foremost to be an Architect?
- Who considers themselves first and foremost to be a manager, and will you admit it?
Goal

Discuss and demonstrate the challenges of mapping Java Objects to Relational Databases and XML documents
Agenda

- Quick Review of Java Mapping
  - O-R and O-X Overview
  - Terminology

- O-R Mapping
  - Direct, 1-1, Aggregate, 1-M, M-M, Inheritance

- O-X Mapping
  - Direct, Composite, Transformation, Inheritance
Agenda

- Quick Review of Java Mapping
  - O-R and O-X Overview
  - Terminology
- O-R Mapping
- O-X Mapping
Impedance Mismatch

- Difference in relational, XML and object technology known as “impedance mismatch”
- Challenging problem
  - Requires relational, XML and object expertise
# O-R Impedance Mismatch

<table>
<thead>
<tr>
<th>Factor</th>
<th>J2EE</th>
<th>Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technical</strong></td>
<td>Logical Data Format</td>
<td>Objects, methods, inheritance</td>
</tr>
<tr>
<td>Scale</td>
<td>Hundreds of megs</td>
<td>Gigabytes, terabytes</td>
</tr>
<tr>
<td>Relationship</td>
<td>Memory references</td>
<td>Foreign keys</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>Internal object identity</td>
<td>Primary keys</td>
</tr>
<tr>
<td><strong>Business</strong></td>
<td>Key Skills</td>
<td>Java development, object modeling</td>
</tr>
<tr>
<td>Tools</td>
<td>IDE, Source code management, Object Modeler</td>
<td></td>
</tr>
<tr>
<td><strong>Political</strong></td>
<td>Corporate Org.</td>
<td>“Newer technology” often with weak organizational ties to database mgmt</td>
</tr>
<tr>
<td>Structure</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## O-X Impedance Mismatch

<table>
<thead>
<tr>
<th>Factor</th>
<th>J2EE</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technical</strong></td>
<td>Logical Data Format: Objects, methods,</td>
<td>XSD, XPATH</td>
</tr>
<tr>
<td></td>
<td>methods, inheritance</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>Hundreds of megs</td>
<td>Depends</td>
</tr>
<tr>
<td>Relationship</td>
<td>Memory references</td>
<td>Generally use Aggregation</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>Internal object identity</td>
<td>Unique Identifier</td>
</tr>
<tr>
<td><strong>Business</strong></td>
<td><strong>Key Skills</strong></td>
<td><strong>XML, XSD, XPATH, XQUERY</strong></td>
</tr>
<tr>
<td></td>
<td>Java development, object modeling</td>
<td></td>
</tr>
<tr>
<td><strong>Tools</strong></td>
<td>**IDE, Source code management, Object</td>
<td><strong>XML Design Tools and Viewers</strong></td>
</tr>
<tr>
<td></td>
<td>Modeler**</td>
<td></td>
</tr>
<tr>
<td><strong>Political</strong></td>
<td>Corporate Org. Structure: Often integrating</td>
<td>High expectations of data portability</td>
</tr>
<tr>
<td></td>
<td>or using legacy application code</td>
<td></td>
</tr>
</tbody>
</table>
Agenda

- Quick Review of Java Mapping
  - O-R and O-X Overview
  - Terminology
- O-R Mapping
- O-X Mapping
JDBC

- Java standard for accessing data sources
- Simply – data source connection
- OK with:
  - “Window on data” applications
  - Business logic entrenched on database
  - Java nothing more than GUI tool
Persistence Layer

- Abstract persistence details from application layer
- Requires Object to Data source mapping...

Java & Web Services

- Objects
  - Persistence Layer
  - Object creation and updates through object-level API
  - API uses SQL or database specific calls

- SQL
- rows
- results are returned as raw data
- object-level querying and creation results are objects
O-R Mapping Tools

O-R Mapping Tools help you manage the mapping’s meta data.
Agenda

- Quick Review of Java Mapping
  - O-R and O-X Overview
  - Terminology
- O-R Mapping
- O-X Mapping
3 Levels of XML Representation

DOM

Unmanaged Object

Managed Object/EJB

O-X Persistence Manager

O-X Data Converter

XML Parser

XML Document

File

Web Service

BPM

JMS

JDBC

J2C

XDB

RDB

EIS
XML Parser

- JAXP – Java API for XML Parsing
  - DOM
  - SAX
- Very low level
- Akin to straight using straight JDBC for database interactions
- Useful for simple and raw GUI based apps where a business model is overkill
O-X Data Converter

- Converts XML data to Java objects and vice-versa
- Accessed by applications through Marshal/Unmarshal interface
- Usually for non-transactional data sources
- JAXB implementations
O-X Data Converter Limitations

- Beware of your JAXB implementation...
  - Most generate Java classes from XML Schema – static, inflexible
  - Most offer no control over the mappings
  - Can’t use your own Java classes
  - Application code is tightly coupled to a specific XSD
- Usually no GUI tools to do mappings
- Conversion only, no run-time manager available for transactional data sources
- Homogeneous data support
  - Specific interfaces and generation for XML
O-X Persistence Manager

- Flexible mapping, developers control how objects are mapped to XML – “meet in the middle”
  - Can use developer-defined Java classes
  - Independence between object model and XML schema
  - Business logic can be safely added into Java model
  - Classes can be mapped to multiple schemas – *vice versa*
  - Can be JAXB compliant
O-X Persistence Manager

- Support complex XML mappings
  - Positional, path information
  - Examples coming...
- May provide visual mapping interface
- May support other data sources – relational and EIS
O-X Persistence Manager

- Persistence manager functionality may be required for transactional XML data sources such as EIS systems, XML databases.
- Provides additional capabilities on top of data conversion such as:
  - Caching
  - Querying
  - Transactions
  - Concurrency
O-X Mapping

- Map Object Model to XSD
- Either code gen XSD from Object Model, *vice versa*, or “meet in the middle”
O-X Mapping Tools

- O-X Mapping Tools help you manage the mapping’s meta data
O-R versus O-X

O-X
- Much more aggregation (aka “denormalized”)
- Relationships represented in structure of document
- Related data usually contained in single “bite”
  - “Less Chatty”

O-R
- Relationships represented by data in table (FK)
- Frequent round-tripping
- Data storage and practices more standardized
Agenda

- Quick Review of Java Mapping
  - O-R and O-X Overview
  - Terminology
- O-R Mapping
- O-X Mapping
Mapping Terminology

- **Direct Mappings**
  - Field in Object model maps to field in datasource
    - May require conversion, aggregation or serialization

- **Relationship Mappings**
  - Thought of always as uni-directional
    - “Bi-directional” relationships are two distinct mappings
  - Source is the originator of the relationship
  - Target is referenced object or set of objects from the source
Mapping Terminology

public class Source {
    private String directAttribute;
    private Target target;
    private Collection targetCollection;
}

Source

<table>
<thead>
<tr>
<th>String:directAttribute</th>
<th>target</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>targetCollection</td>
</tr>
</tbody>
</table>
Agenda

- Quick Review of Java Mapping
- O-R Mapping
  - Direct, 1-1, Aggregate, 1-M, M-M, Inheritance
- O-X Mapping
Simple Direct Mapping

Customer

id: int
name: String
creditRating: int

CUST

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>C_RATING</th>
</tr>
</thead>
</table>
Direct Conversion Mappings

**Customer**

<table>
<thead>
<tr>
<th></th>
<th>O</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Female</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

**Employee**

<table>
<thead>
<tr>
<th></th>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>START</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
</tbody>
</table>
1-1 Object Model

Customer
- id: int
- name: String
- creditRating: int

Address
- id: int
- city: String
- zip: String

1:1 Relationship
Typical 1-1 Relationship

<table>
<thead>
<tr>
<th>CUST</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>NAME</td>
</tr>
<tr>
<td>ID</td>
<td>NAME</td>
</tr>
<tr>
<td>7</td>
<td>Don</td>
</tr>
<tr>
<td>13</td>
<td>Trisha</td>
</tr>
</tbody>
</table>
1-1 FK in Target Table

<table>
<thead>
<tr>
<th>CUST</th>
<th>ADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NAME</td>
</tr>
<tr>
<td>7</td>
<td>Don</td>
</tr>
<tr>
<td>13</td>
<td>Trisha</td>
</tr>
</tbody>
</table>
Association Table

**CUST**

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Don</td>
</tr>
<tr>
<td>13</td>
<td>Trisha</td>
</tr>
</tbody>
</table>

**ADDR**

<table>
<thead>
<tr>
<th>ID</th>
<th>CITY</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Ottawa</td>
<td>K2J4X7</td>
</tr>
<tr>
<td>17</td>
<td>Toronto</td>
<td>K1P2T2</td>
</tr>
</tbody>
</table>

**CUST_ADDR**

<table>
<thead>
<tr>
<th>C_ID</th>
<th>A_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
</tr>
</tbody>
</table>
# Aggregation

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>C_RATING</th>
<th>CITY</th>
<th>ZIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Don</td>
<td></td>
<td>Ottawa</td>
<td>K2J4X7</td>
</tr>
<tr>
<td>13</td>
<td>Trisha</td>
<td></td>
<td>Toronto</td>
<td>K1P2T2</td>
</tr>
</tbody>
</table>
Multiple Table Mappings

Diagram showing the relationship between CUST, CUST_CREDIT, and ADDR tables with their respective columns:
- CUST: ID, NAME, A_ID
- CUST_CREDIT: ID, C_RATING, A_ID
- ADDR: ID, CITY, ZIP
Multiple Table Mappings

```
<table>
<thead>
<tr>
<th>CUST</th>
<th>CUST_CREDIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>NAME</td>
<td>C_RATING</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ADDR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CITY</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>CUST</th>
<th>CUST_CREDIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>NAME</td>
<td>C_RATING</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ADDR</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CITY</td>
</tr>
</tbody>
</table>
```
Multiple Table Mappings

```
CUST
ID  NAME  CC_ID

CUST_CREDIT
ID  C_RATING  A_ID

ADDR
ID  CITY  ZIP
```
1-M Object Model

Customer

- id: int
- name: String
- creditRating: int

Phone

- id: int
- type: String
- areaCode: String
- number: String

1:M Relationship
1-M Relationship

<table>
<thead>
<tr>
<th>CUST</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NAME</td>
<td>C_RATING</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PHONE_NUMBER</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>TYPE</td>
<td>A_CODE</td>
<td>PNUMB</td>
<td>C_ID</td>
</tr>
</tbody>
</table>
1-M Relationship Trick

- Mapping a 1-1 “back reference” from the target of the 1-M to the source
- Allows persistence layer to more easily handle the foreign key
1-M Relationship Trap!

- What if order is important?
- Specify field to order by

```
Customer
| id: int |
| name: String |
| creditRating: int |

Phone
| id: int |
| String: type |
| areaCode: String |
| number: String |
```
1-M Relationship Trap!

- What if *collection index* is the order criteria?

```java
* :Customer
  name = "Don"

  :Collection
    1

    :Phone1
    type = 'home'
    numb = 45367

    :PhoneN
    type = 'cell'
    numb = 6573

  :Phone
    id: int
    String: type
    String: areaCode
    String: number

* :Customer
  name: String
  creditRating: int
```
1-M Relationship Trap!

- Challenging problem because “state” from Java utility class needs to be persisted
- Best solution is to find or make a persistent attribute of the target the order by field
- Regardless, database requires SOMETHING to order by
1-M Relationship Trap!

Employee

name = "Don"

Collection

1

Phone1

type = 'home'
numb = 45367

Phone1

type = 'cell'
numb = 6573

EMP

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Don</td>
</tr>
</tbody>
</table>

PHONE

<table>
<thead>
<tr>
<th>ID</th>
<th>TYPE</th>
<th>NUMBER</th>
<th>ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>home</td>
<td>45367</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>cell</td>
<td>6573</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>work</td>
<td>99882</td>
<td>2</td>
</tr>
</tbody>
</table>
M-M Object Model

**Employee**

- id: int
- name: String

**Project**

- id: int
- name: String
- budget: double

M:M Relationship
M-M Relationship

**EMP**

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Don</td>
</tr>
<tr>
<td>13</td>
<td>Mike</td>
</tr>
</tbody>
</table>

**PROJECT**

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>BUDGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Tiger</td>
<td>5,000,000</td>
</tr>
<tr>
<td>17</td>
<td>Dragon</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

**EMP_PROJ**

<table>
<thead>
<tr>
<th>E_ID</th>
<th>P_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
</tr>
</tbody>
</table>
M-M Relationship Trap!

Common challenge – RDBs often intertwine data and relationships. Where would this go in object model?
M-M Relationship Trap!

- In relational world, putting “data” on a relationship is not uncommon
- Not possible in Java
- Must store data in an object – requires some re-thinking of Object model
M-M Trap Solution

Responsibility
roleName: String
emp: Employee
proj: Project

Employee
id: int
name: String

Project
id: int
name: String
budget: double
Direct Collection Mapping

:Employee

name = “Don”
tasks = {“schmooze”, “gossip”, “coffee”}

EMP

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Don</td>
</tr>
</tbody>
</table>

EMP_TASK

<table>
<thead>
<tr>
<th>ID</th>
<th>TASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>schmooze</td>
</tr>
<tr>
<td>7</td>
<td>gossip</td>
</tr>
<tr>
<td>7</td>
<td>coffee</td>
</tr>
</tbody>
</table>
Direct Collection Issues

- Inefficient for large direct collections
  - Changes require lots of “change detection” analysis, or massive purge and insert on database
- Only useful for PRIMITIVES, more than 1 piece of date requires 1-M mapping
- Consider Maps of Primitives...
Direct Map Mapping

Employee

name = “Don”
contact = {
    “email” -> “d@m.com”,
    “home” -> “555-1212”,
    “cell” -> “555-2323”
}

EMP

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Don</td>
</tr>
</tbody>
</table>

CONTACT

<table>
<thead>
<tr>
<th>ID</th>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>email</td>
<td><a href="mailto:d@m.com">d@m.com</a></td>
</tr>
<tr>
<td>7</td>
<td>home</td>
<td>555-1212</td>
</tr>
<tr>
<td>7</td>
<td>cell</td>
<td>555-2323</td>
</tr>
</tbody>
</table>
Inheritance

- Very straight forward at first, but can become challenging because of the number of possible solutions
- Compounded with relationships
- Can be source of contention with DBA – generally leads to lots of joining on queries!
“Leaf Table Mapping”

Map only concrete classes

<table>
<thead>
<tr>
<th>ID</th>
<th>VAR</th>
<th>SUB1VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
<td>Pish</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>VAR</th>
<th>SUB2VAR</th>
<th>SUB22VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Tim</td>
<td>677</td>
<td>Flim</td>
</tr>
</tbody>
</table>

Root {a}

id: int
var: String

Sub1

sub1var: String

Sub2 {a}

sub2var: int

Sub22

sub22var: String
“Leaf Table” Issues

- How do you query for the Root class?
- Data is not normalized
"All Table Mapping"

Root \{a\}
- id: int
- var: String

Sub1
- sub1var: String

Sub2 \{a\}
- sub2var: int

Sub22
- sub22var: String

Map all classes

ROOT
<table>
<thead>
<tr>
<th>ID</th>
<th>VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
</tr>
<tr>
<td>13</td>
<td>Bar</td>
</tr>
</tbody>
</table>

Problem!
How to tell type?!

SUB1
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB1VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Pish</td>
</tr>
</tbody>
</table>

SUB2
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB2VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>677</td>
</tr>
</tbody>
</table>

SUB22
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB22VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Flim</td>
</tr>
</tbody>
</table>
Identifying Type

\[ \text{Root} \{a\} \]
- \( \text{id: int} \)
- \( \text{var: String} \)

Map all classes

\[ \text{Sub1} \]
- \( \text{sub1var: String} \)

\[ \text{Sub2} \{a\} \]
- \( \text{sub2var: int} \)

\[ \text{Sub22} \]
- \( \text{sub22var: String} \)

\[ \text{ROOT} \]
<table>
<thead>
<tr>
<th>ID</th>
<th>VAR</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
<td>S1</td>
</tr>
<tr>
<td>13</td>
<td>Bar</td>
<td>S22</td>
</tr>
</tbody>
</table>

Need Type Identifier!

\[ \text{SUB1} \]
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB1VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Pish</td>
</tr>
</tbody>
</table>

\[ \text{SUB2} \]
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB2VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>677</td>
</tr>
</tbody>
</table>

\[ \text{SUB22} \]
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB22VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Flim</td>
</tr>
</tbody>
</table>
Identifying Type

Lots of strategies exist...

if id < 10
  type = Sub1
else
  type = Sub22
“All Table” Issues

- Lots of joining
- Type identification can be inefficient
"Single Table Mapping"

**Map to one table**

<table>
<thead>
<tr>
<th></th>
<th>ID</th>
<th>VAR</th>
<th>SUB1VAR</th>
<th>SUB2VAR</th>
<th>SUB22VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>Foo</td>
<td>Pish</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>Bar</td>
<td>null</td>
<td>677</td>
<td>Flim</td>
</tr>
</tbody>
</table>

- **Root**
  - `id`: int
  - `var`: String

- **Sub1**
  - `sub1var`: String

- **Sub2**
  - `sub2var`: int
  - **Sub22**
    - `sub22var`: String
Identifying Type

<table>
<thead>
<tr>
<th>ROOT</th>
<th>ID</th>
<th>VAR</th>
<th>SUB1VAR</th>
<th>SUB2VAR</th>
<th>SUB22VAR</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
<td>Pish</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>S1</td>
</tr>
<tr>
<td>13</td>
<td>Bar</td>
<td>null</td>
<td>677</td>
<td>Flim</td>
<td></td>
<td>SUB22</td>
</tr>
</tbody>
</table>

Two options for type detection:

```java
if SUB1VAR != null
    type = Sub1
else
    type = Sub22
```
“Single Table” Issues

- DBA will freak out – de-normalized database
- Type identification can be inefficient
"Combination Table Mapping"

**Root {a}**
- Sub1
  - sub1var: String
- Sub2 {a}
  - sub2var: int

**Map appropriate classes**

**ROOT**
<table>
<thead>
<tr>
<th>ID</th>
<th>VAR</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
<td>S1</td>
</tr>
<tr>
<td>13</td>
<td>Bar</td>
<td>S22</td>
</tr>
</tbody>
</table>

**SUB1**
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB1VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Pish</td>
</tr>
</tbody>
</table>

**SUB22**
<table>
<thead>
<tr>
<th>ID</th>
<th>SUB22VAR</th>
<th>SUB2VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Flim</td>
<td>677</td>
</tr>
</tbody>
</table>
Which Strategy to Use?

- **Leaf Table**
  - When never querying at abstract class level

- **All Table**
  - When not doing much querying
    - Too many joins

- **Single Table**
  - When one subclass is much more prevalent, and your DBA doesn’t care about normalization

- **Combination**
  - Use to optimize above situations
Wouldn’t It Be Cool?

Map to view for reads…

ROOT (View)

<table>
<thead>
<tr>
<th></th>
<th>VAR</th>
<th>SUB1VAR</th>
<th>SUB2VAR</th>
<th>SUB22VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
<td>Pish</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>13</td>
<td>Bar</td>
<td>null</td>
<td>677</td>
<td>Flim</td>
</tr>
</tbody>
</table>

Map to tables for writes…

ROOT

<table>
<thead>
<tr>
<th></th>
<th>VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Foo</td>
</tr>
<tr>
<td>13</td>
<td>Bar</td>
</tr>
</tbody>
</table>

SUB2

<table>
<thead>
<tr>
<th>ID</th>
<th>SUB2VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>677</td>
</tr>
</tbody>
</table>

SUB22

<table>
<thead>
<tr>
<th>ID</th>
<th>SUB22VAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Flim</td>
</tr>
</tbody>
</table>
**Agenda**

- Quick Review of Java Mapping
- O-R Mapping
- O-X Mapping
  - Direct, Composite, Transformation, Inheritance
Example Object Model

Order
- id: long
- orderedBy: String

LinItem
- lineNumber: long
- itemName: String
- quantity: long
- itemPrice: BigDecimal

* lineltems

Address
- addressee: String
- city: String
- street1: String
- street2: String
- state: String
- country: String
- zipCode: String

1 address
Direct Mapping

- : Address
  - : Order
    - id = 1234
    - orderedBy = “Jane Doe”
  - : Lineltem
  - : Lineltem

```xml
<Order ORDER_ID="1234">
  <ORDERED_BY>Jane Doe</ORDERED_BY>
  <ADDRESS>
    ...
  </ADDRESS>
  <LINES>
    ...
  </LINES>
</ORDER>
```
Composite Object Mapping

```xml
<Order ORDER_ID="1234">
  <ORDERED_BY>Jane Doe</ORDERED_BY>
  <ADDRESS>
    ...
  </ADDRESS>
  <LINES>
    ...
  </LINES>
</ORDER>

: Address

: Order
  id = 1234
  orderedBy = "Jane Doe"

: LineItem
  : LineItem
```
Composite Collection Mapping

Possible <lineitems> tag here

```
<ORDER ORDER_ID="1234">
  <ORDERED_BY>Jane Doe</ORDERED_BY>
  <ADDRESS>
    ...
  </ADDRESS>
  <LINES>
    ...
  </LINES>
</ORDER>
```

Possible <lineitems> tag here

```
<ORDER ORDER_ID="1234">
  <ORDERED_BY>Jane Doe</ORDERED_BY>
  <ADDRESS>
    ...
  </ADDRESS>
  <LINES>
    ...
  </LINES>
</ORDER>
```
### Positional Information

#### Good O-X Support

```java
: Address
addressee = "Jane Doe"
street = "123 Some St."
apartment = "Apt. #123"
...
```

```xml
<Address>
  <Addressee>Jane Doe</Addressee>
  <Street>123 Some St.</Street>
  <Street>Apt. #123</Street>
  ...
</Address>
```

#### JAXB/Class Generation Today...

```java
: Address
addressee = "Jane Doe"
street = ["123 Some St.", "Apt. #123"]
...
```

```xml
<Address>
  <Addressee>Jane Doe</Addressee>
  <Street>123 Some St.</Street>
  <Street>Apt. #123</Street>
  ...
</Address>
```
Path Information

Good O-X Mapping

: LineItem

|(lineNumber = 1) |
|itemPrice = 2.50|
|quantity = 50|

: Items

|lineNumber = 1|
|quantity = 50|

JAXB/Class Generation Today…
Direct Collection Mapping

- Compare with Positional Mapping

```java
: Employee

tasks = {"this", "that", "other thing"}

<EMPLOYEE>
  <TASK>this</TASK>
  <TASK>that</TASK>
  <TASK>other thing</TASK>
</EMPLOYEE>
```
Transformation Mapping

Unmarshal (Read)

```xml
<EMPLOYEE>
  <START-TIME>9:00:00</START-TIME>
  <END-TIME>17:00:00</END-TIME>
</EMPLOYEE>
```

: Employee

normalHours = {9am,5pm}

Marshal (Write)

```xml
<EMPLOYEE>
  <START-TIME>9:00:00</START-TIME>
  <END-TIME>17:00:00</END-TIME>
</EMPLOYEE>
```

: Employee

normalHours = {9am,5pm}
Conversion Mapping

: Employee

gender = “Female”

“Female” to “F”
“Male” to “M”

<EMPLOYEE>
  <GENDER>F</GENDER>
</EMPLOYEE>
Simple Type Translator

Class Model

PhoneNumber

areaCode: Object
number: Object

<element name="AREA-CODE" type="anySimpleType"/>
<element name="NUMBER" type="anySimpleType"/>

Instance Model

: PhoneNumber
areaCode = 613
number = "KL51234"

<PHONE-NUMBER>
<AREA-CODE xsi:type="xsd:int">613</AREA-CODE>
<NUMBER xsi:type="xsd:string">KL51234</NUMBER>
</PHONE-NUMBER>
Namespace Support

Namespace Support:

```
  <def:FIRST-NAME>Jane</def:FIRST-NAME>
  <def:LAST-NAME>Doe</def:LAST-NAME>
</abc:EMPLOYEE>

<ns1:EMPLOYEE xmlns:ns1="http://X" xmlns:ns2="http://Y">
  <ns2:FIRST-NAME>Jane</ns2:FIRST-NAME>
  <ns2:LAST-NAME>Doe</ns2:LAST-NAME>
</ns1:EMPLOYEE>
```

Employee Default Root = abc:EMPLOYEE
XPath firstName = def:FIRST-NAME
XPath lastName = def:LAST-NAME

Namespace Resolver:
abc = http://X
def = http://Y

firstName = “Jane”
lastName = “Doe”
Inheritance

**Class Model**

- **Address**
  - street: String
  - city: String

- **CanadianAddress**
  - province: String
  - postalCode: String

**Instance Model**

: CanadianAddress

- street = “123 Any St.”
- city = “Ottawa”
- province = “ON”
- postalCode = “A1B 2C3”

```xml
<Address xsi:type="CANADIAN-ADDRESS-TYPE">
  <STREET>123 Any St.</STREET>
  <CITY>Ottawa</CITY>
  <PROVINCE>ON</PROVINCE>
  <POSTAL-CODE>A1B 2C3</POSTAL-CODE>
</EMPLOYEE>
```
Conclusions

- Much more to O-R and O-X Mapping than first meets the eye
- O-R and O-X Share several common challenges, but each have their own unique perspectives
Recommended Sessions...

- Donald Smith
  - Impact of Tiger on Java Persistence

- Mike Keith
  - Standards Based Persistence through Web Services
  - Evolution of the EJB Entity

- Neil Graham
  - Lower-level Data Manipulation with J2SE 5.0

- Matthew Wakeling
  - [Advanced Techniques | Performance] in Object Warehousing