EclipseWorld 2007

November 6-8, 2007

Eclipse JPA

Building JPA Applications with EclipseLink and Dali

Neil Hauge, Oracle
Dali Project co-Lead
Doug Clarke, Oracle
EclipseLink Project co-Lead

Agenda

- JPA Primer
- Eclipse JPA Projects
  - WTP’s JPA Tooling Project : Dali
  - Eclipse Persistence Services Project “EclipseLink”
- Eclipse JPA in action … the demo
- Where are we headed?

About Java Persistence API (JPA)

- Separate document bundled as part of EJB 3.0 specification
- Suitable for use in different modes
  - Standalone in Java SE environment
  - Hosted within a Java EE Container
- Standardization of current persistence practices
- Merging of expertise from persistence vendors and communities including: TopLink, Hibernate, JDO, EJB vendors and individuals

JPA—in a Nutshell

- A Java standard that defines:
  - how Java objects are stored in relational databases (specified using a standard set of mappings)
  - a programmer API for reading, writing, and querying persistent Java objects (“Entities”)
  - a full featured query language
  - a container contract that supports plugging any JPA runtime in to any compliant container.

Implementations

- Persistence provider vendors include:
  - Oracle,Sun / TopLink Essentials (RI)
  - Eclipse JPA – EclipseLink Project
  - BEA Kodo / Apache OpenJPA
  - RedHat / JBoss Hibernate
  - SAP JPA
  - Oracle TopLink
- JPA containers:
  - GlassFish/SunAS, OracleAS, SAP, BEA, JBoss, Spring 2.0
- IDE: Eclipse, MyEclipse, NetBeans™, IntelliJ, Oracle JDeveloper

JPA—POJO Entities

- Concrete classes
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- new() for instance creation
- Direct access or getter/setter methods
  - Can contain logic (e.g. for validation, etc.)
- “Managed” by an EntityManager
- Can leave the Container (become “detached”)

Object-Relational Mappings

- Core JPA Mappings
  - Id
  - Basic
  - Relationships
    - OneToOne
    - OneToMany/ManyToOne
    - ManyToMany
  - And more…
- Can be specified using Annotations or XML

JPA Entity—Mappings on Fields

```
@Entity public class Customer {

   @Id
   private String name;
   @OneToOne
   private Account account;

   public String getName() { return name; }
   public void setName(String name) {
       this.name = name;
   }
   public Account getAccount() { return account; }
   public void setAccount(Account account) {
       this.account = account;
   }
}
```

JPA Entity—Mappings on Properties

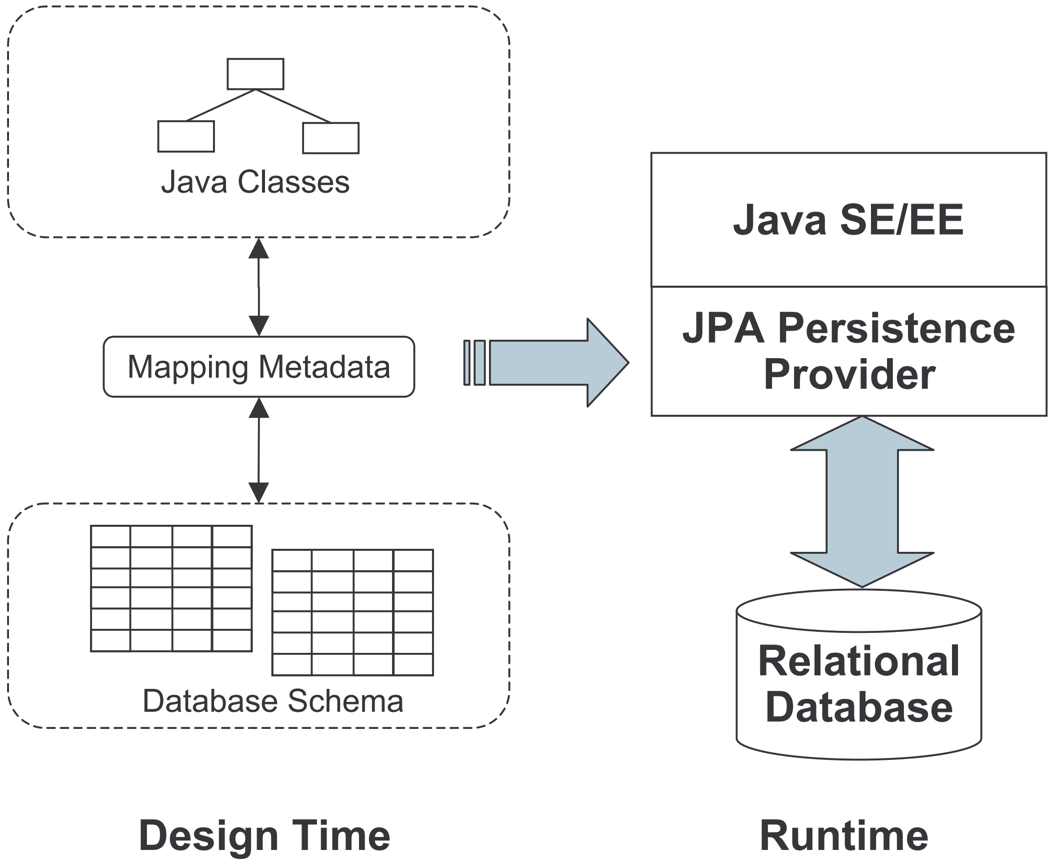
```
@Entity public class Customer {

   private String name;
   private Account account;

   @Id
   public String getName() { return name; }
   public void setName(String name) {
       this.name = name;
   }
   @OneToOne
   public Account getAccount() { return account; }
   public void setAccount(Account account) {
       this.account = account;
   }
}
```

JPA Entity—Mappings in XML

```
<entity-mappings
  xmlns="http://java.sun.com/xml/ns/persistence/orm"
…
  <entity class="Customer">
    <attributes>
      <id name="name"/>
      <one-to-one name="account"/>
    </attributes>
  </entity>
…
</entity-mappings>
```

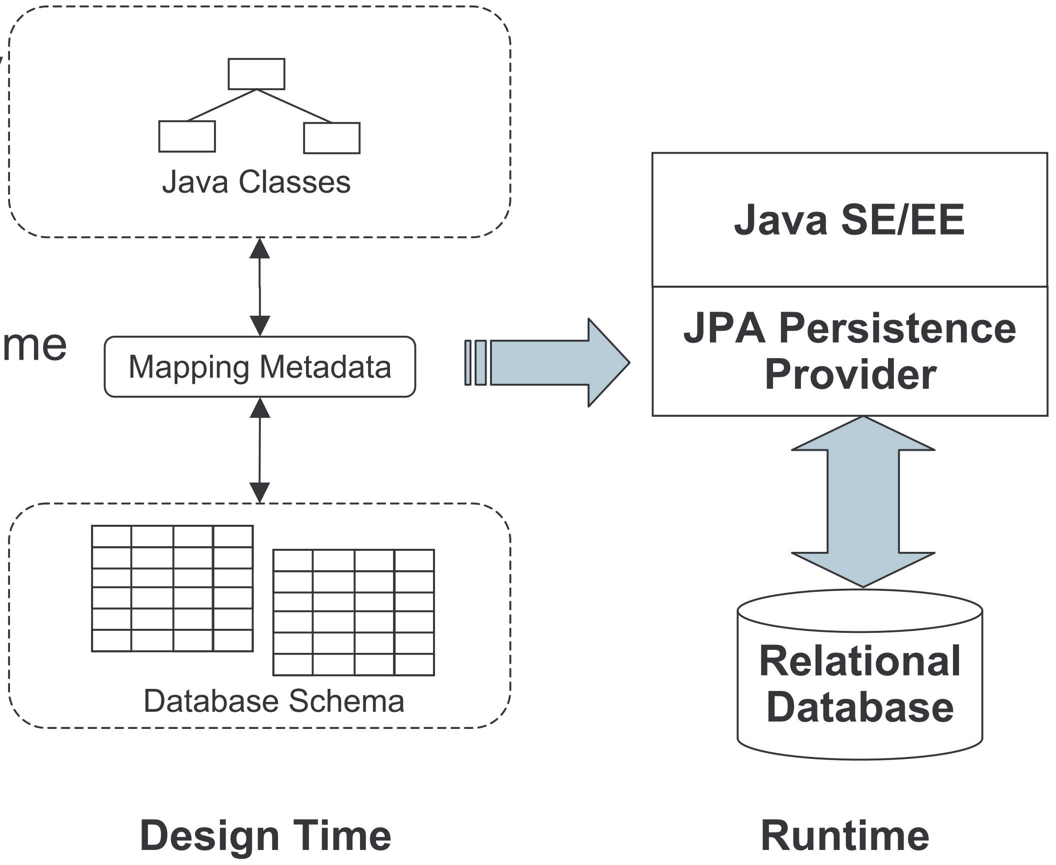
Why do you need JPA Tools?

- JPA runtime combines:
  - Java Classes
  - Mapping Metadata
  - Database schema



Why do you need JPA Tools?

- How can you tell if they all match?
  - Deploy and run tests?
    - ✗ slow
    - ✗ find one problem at a time (fix, run, fix, …)
    - ✓ definitive
  - Design time validation?
    - ✓ quick
    - ✓ finds all issues
    - ✓ validates against spec
    - ✗ runtime may not match spec



About Dali

- Support for the definition, editing, and deployment of Object-Relational (O/R) mappings for JPA Entities
- Simplify mapping definition and editing through:
  - intelligent mapping assistance
  - dynamic problem identification
  - Mapping generation
- Extensible frameworks and tools so vendors and open source projects can provide specific support for their JPA runtimes
- A subproject of the Web Tools Platform (WTP)

Dali Goals

- Simplicity
  - Intelligent mapping assistance and automated generation
- Intuitiveness
  - Use existing and consistent modeling and tooling practices in Eclipse
  - Light-weight views offer assistance but don’t get in the way of power users
- Compliance
  - Support any and all EJB 3.0 compliant runtime implementations
  - Test using EJB 3.0 Reference Implementation
- Extensibility
  - Provide the ability for vendors and open source projects to seamlessly add their own value-add features

Why ‘Dali’?

- JPA supports “The Persistence of Memory”—which is the title of a well known Salvador Dali painting.

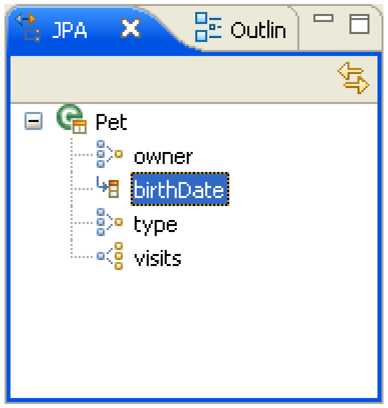
Salvador Dalí. *The Persistence of Memory*. 1931.
© 2005 Salvador Dalí, Gala-Salvador Dalí Foundation/Artists Rights Society (ARS), New York

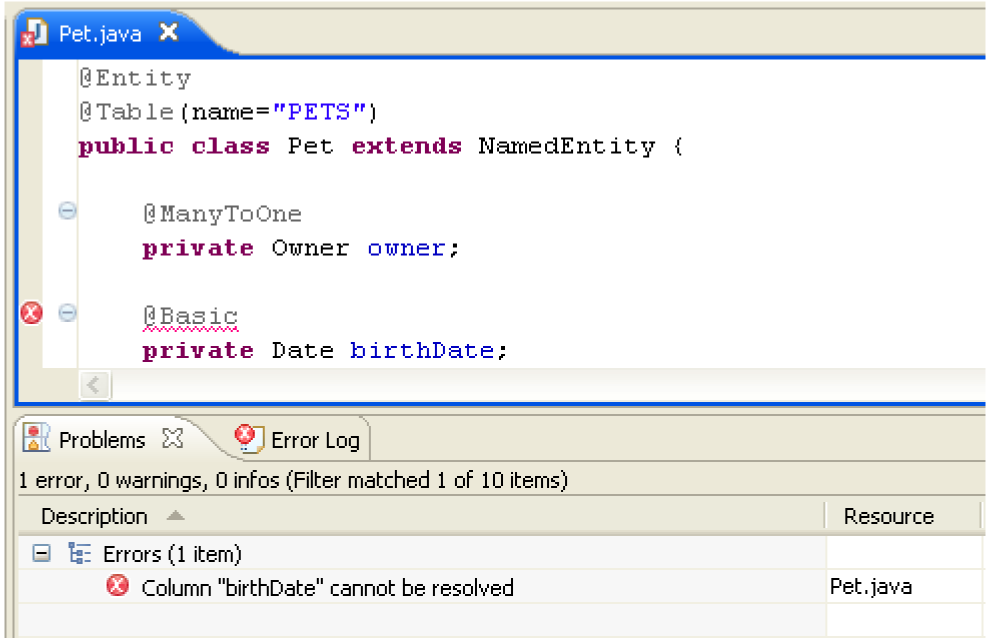
The Dali Value Proposition

You don’t need Dali to build JPA applications—but you’re going to be way more productive if you do!

Dali Contributions to Eclipse

- JPA Mapping Validation
- JPA Structure and Details Views
- Java Source Editor enhancements

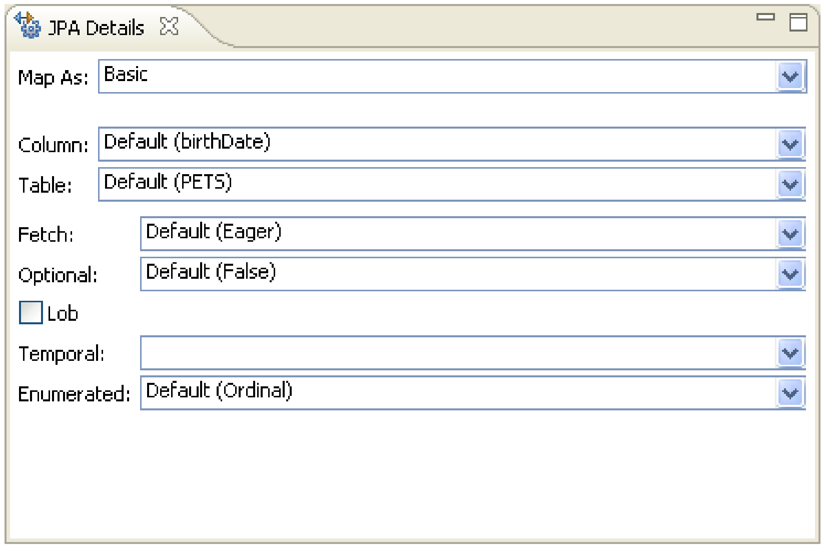




```java
@Entity
@Table(name="PETS")
public class Pet extends NamedEntity {

    @ManyToOne
    private Owner owner;

    @Basic
    private Date birthDate;
```

1 error, 0 warnings, 0 infos (Filter matched 1 of 10 items)

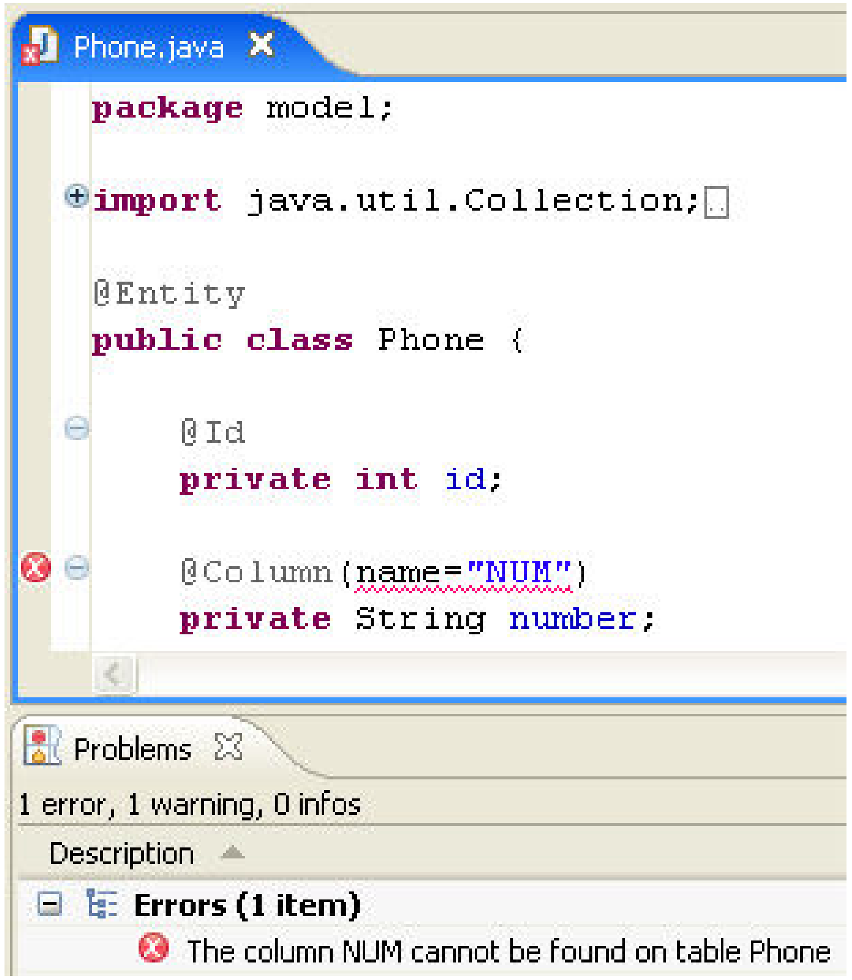
Description	Resource
Errors (1 item)	
Column "birthDate" cannot be resolved	Pet.java



Map As: Basic
Column: Default (birthDate)
Table: Default (PETS)
Fetch: Default (Eager)
Optional: Default (False)
☐ Lob
Temporal:
Enumerated: Default (Ordinal)

Mapping Validation

- Annotations and/or XML used to define JPA Entities.
- JDT validates syntax, but doesn't understand what the annotations *mean*.

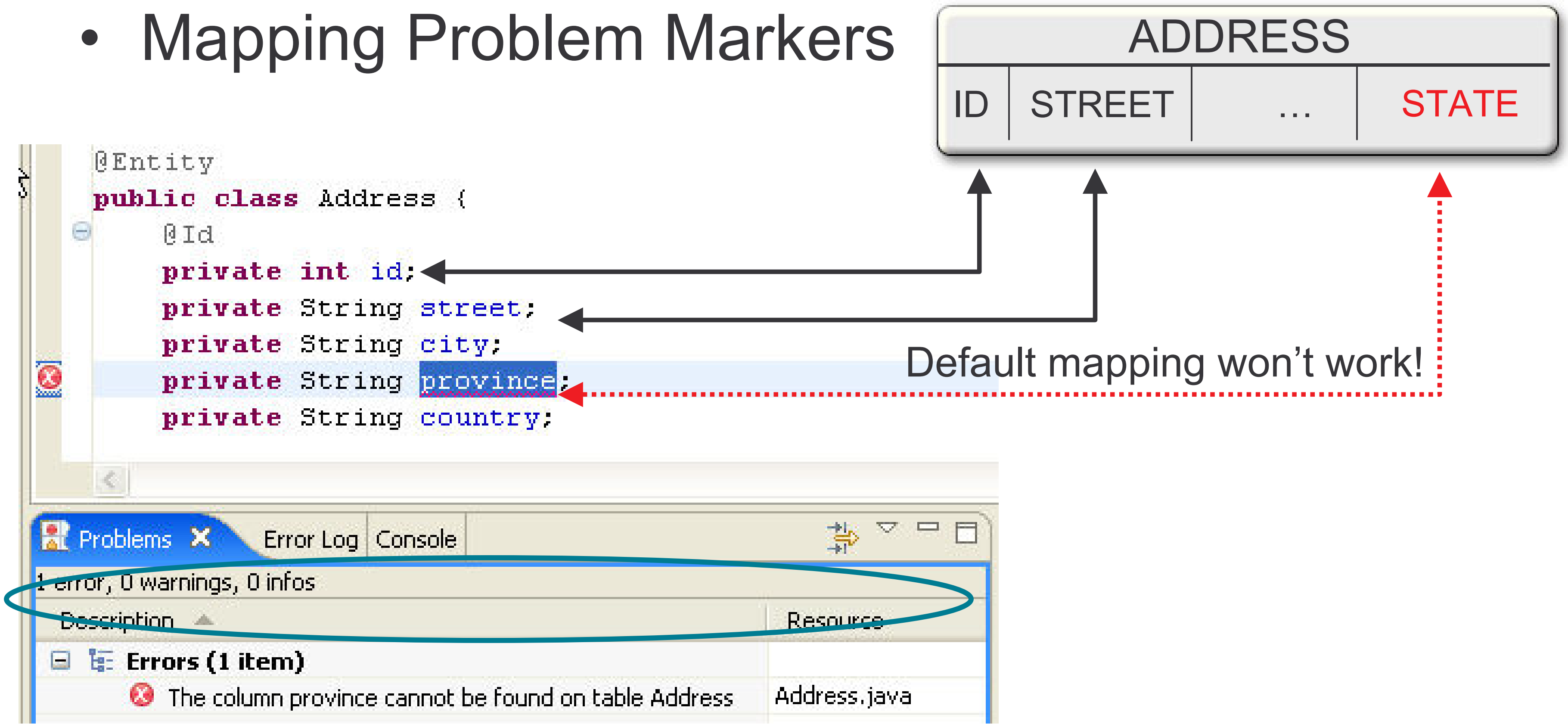


```
package model;

import java.util.Collection;

@Entity
public class Phone {

    @Id
    private int id;

    @Column(name="NUM")
    private String number;
```

```
1 error, 1 warning, 0 infos
Description
Errors (1 item)
    The column NUM cannot be found on table Phone
```

Mapping Validation

- Java Source Editor enhancements
- Mapping Problem Markers



```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    private String province;
    private String country;
```

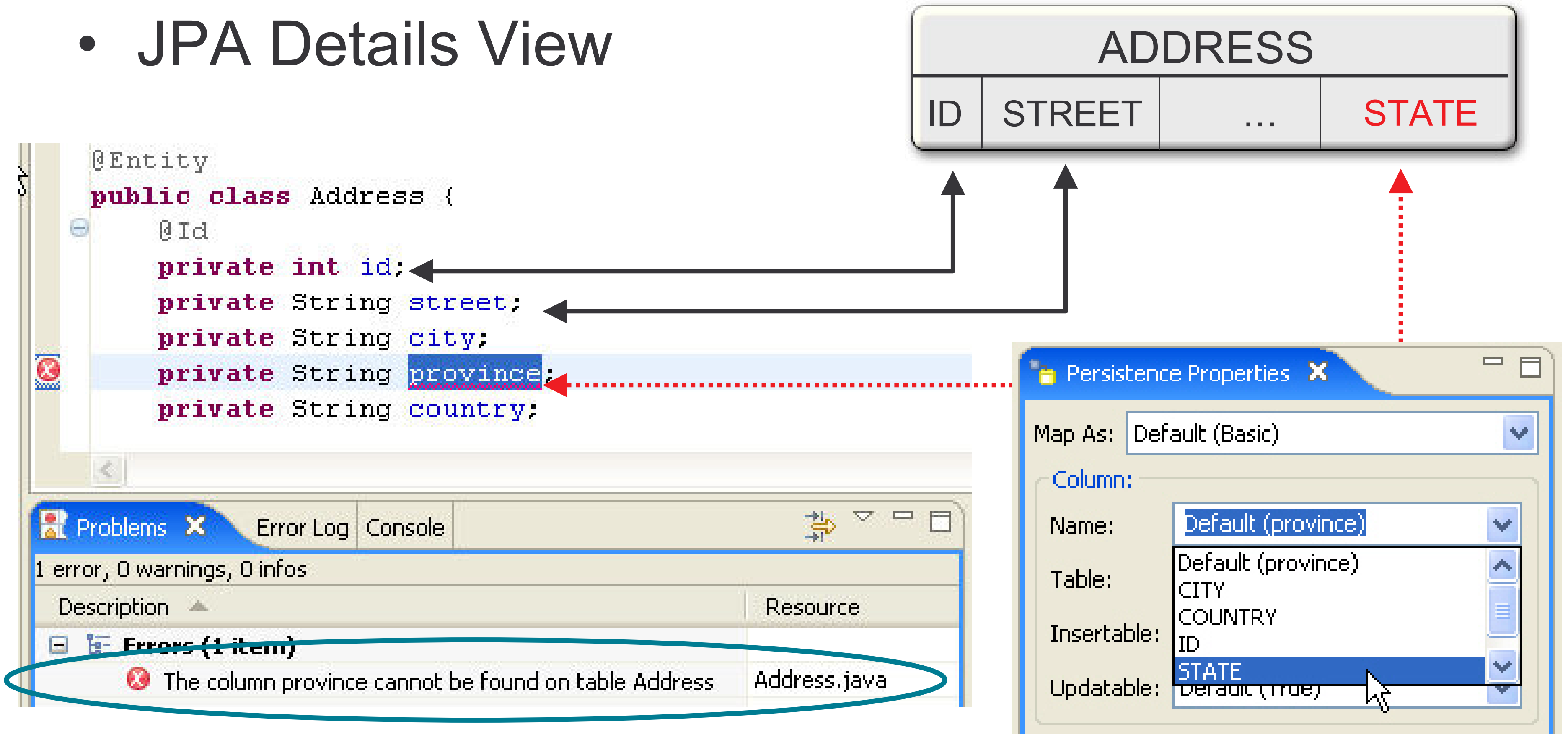
Problems	Error Log	Console

1 error, 0 warnings, 0 infos

Description	Resource
Errors (1 item)	
The column province cannot be found on table Address	Address.java

Mapping Assistance

- JPA Details View



```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    private String province;
    private String country;
```

1 error, 0 warnings, 0 infos

Description	Resource
The column province cannot be found on table Address	Address.java

Map As: Default (Basic)

Column:

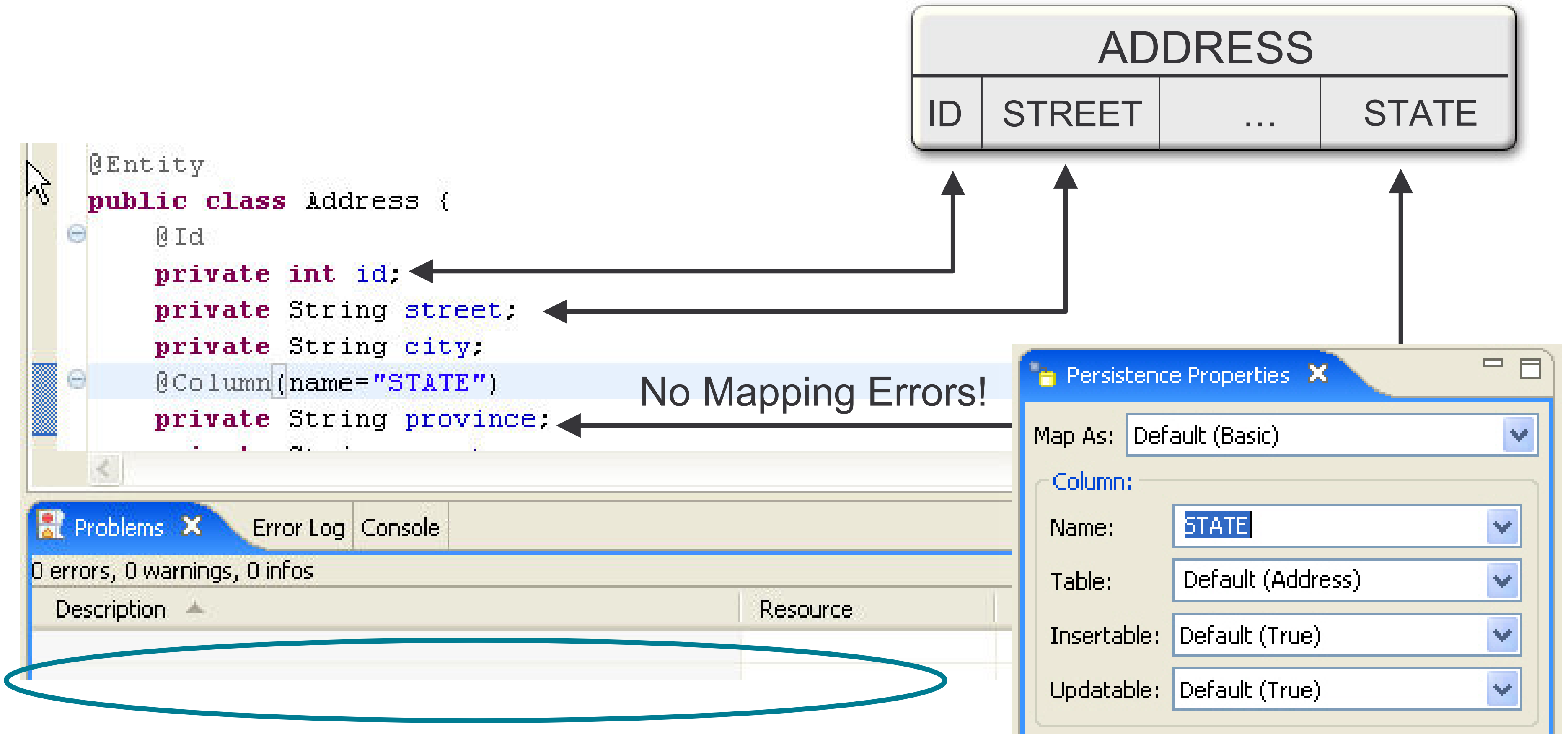
Name: Default (province)

Table: Default (province), CITY, COUNTRY, ID, STATE

Insertable:

Updatable: Default (True)

Mapping Assistance for Basic Mapping



```
@Entity
public class Address {
    @Id
    private int id;
    private String street;
    private String city;
    @Column(name="STATE")
    private String province;
```

0 errors, 0 warnings, 0 infos

Map As: Default (Basic)

Name: STATE

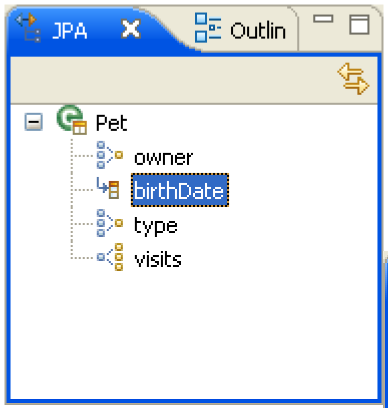
Table: Default (Address)

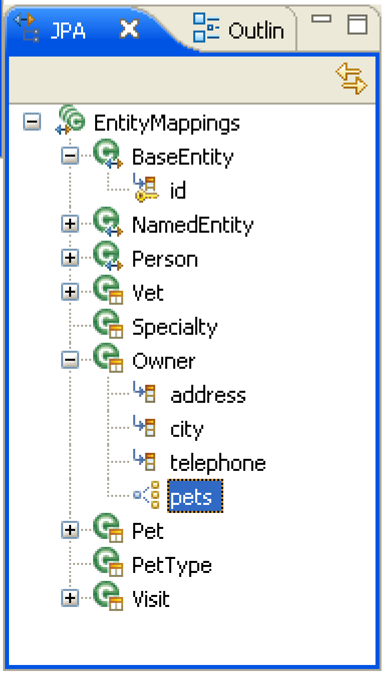
Insertable: Default (True)

Updatable: Default (True)

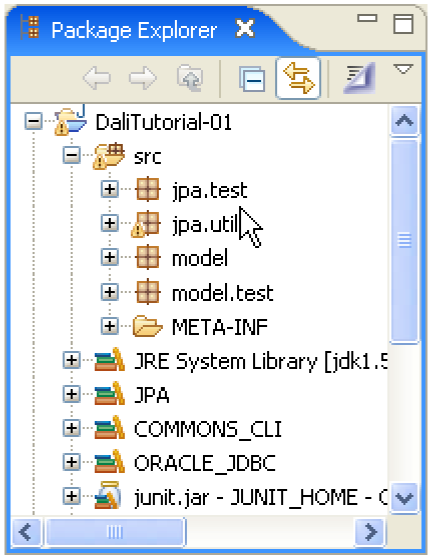
JPA Structure View

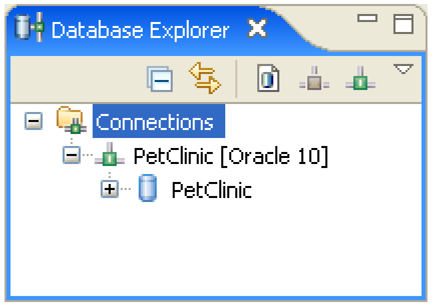
- Provides a JPA specific view of Java Class or ORM XML Mapping File
- A thumbnail sketch of how an Entity is mapped
- Supports navigation between mappings
- Automatically adjusts to either property or field mapping in Java
- Represents structure in Java and XML artifacts

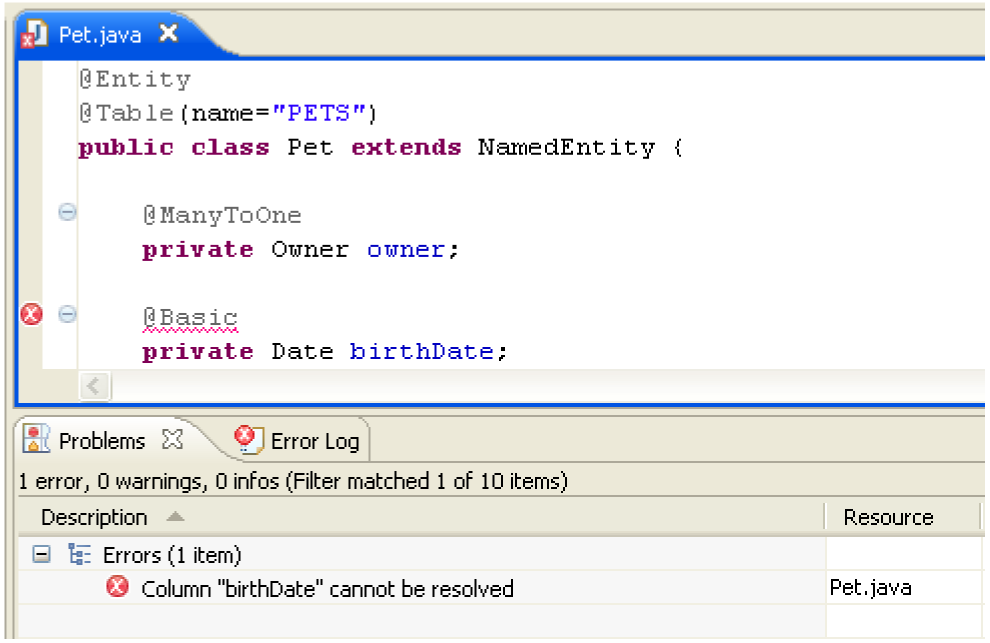




JPA Perspective—all your JPA Views



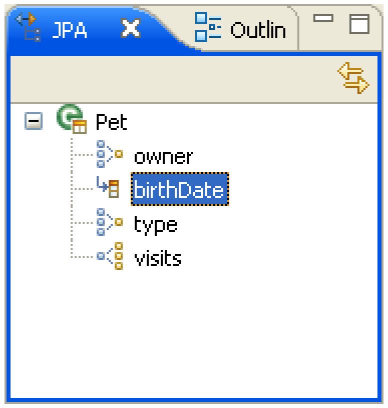


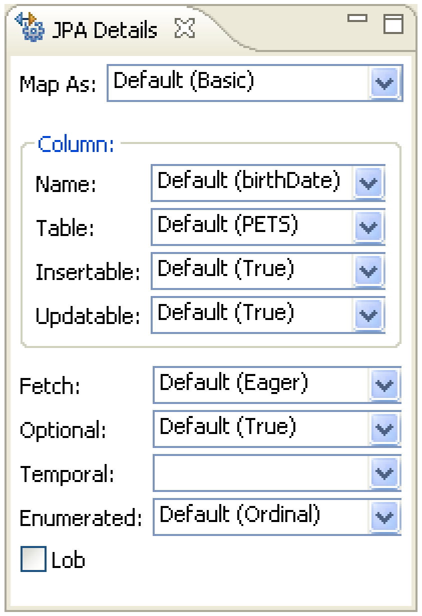


```
@Entity
@Table(name="PETS")
public class Pet extends NamedEntity {

    @ManyToOne
    private Owner owner;

    @Basic
    private Date birthDate;
```

1 error, 0 warnings, 0 infos (Filter matched 1 of 10 items)

Description	Resource
Errors (1 item)	
Column "birthDate" cannot be resolved	Pet.java





Map As: Default (Basic)

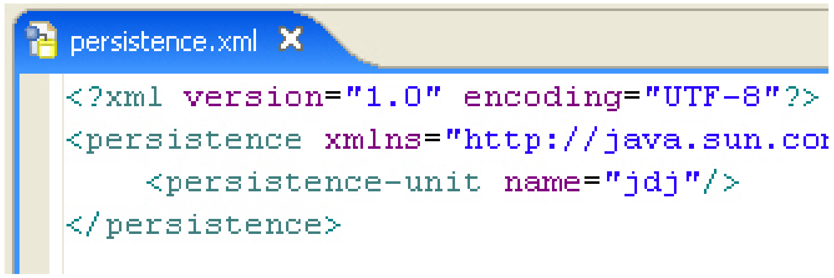
Column:

Name:	Default (birthDate)
Table:	Default (PETS)
Insertable:	Default (True)
Updatable:	Default (True)
Fetch:	Default (Eager)
Optional:	Default (True)
Temporal:	
Enumerated:	Default (Ordinal)

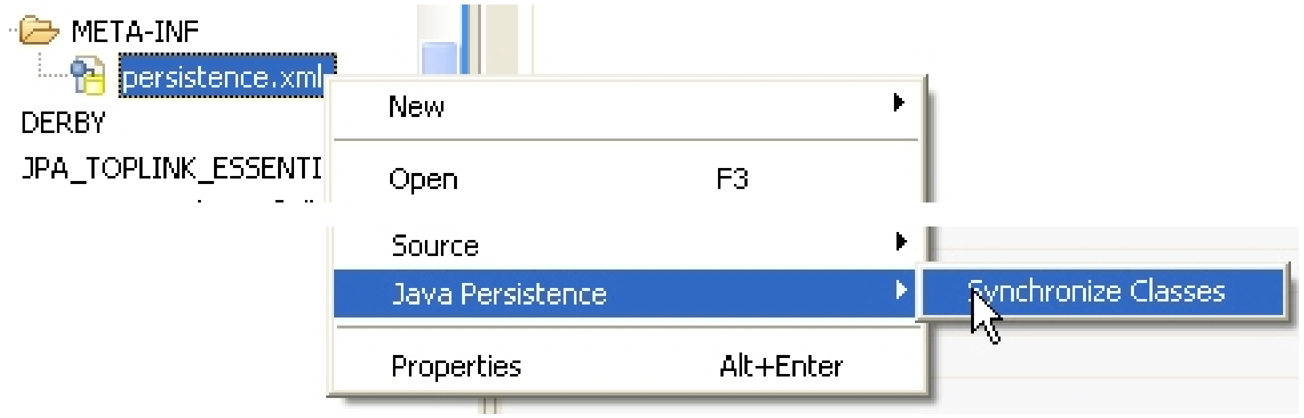
☐ Lob

Synchronizing Persistence.xml

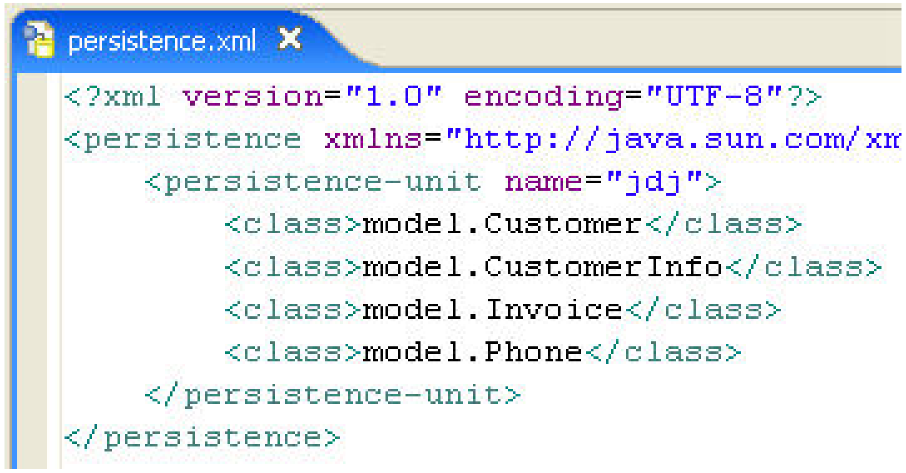
- In Java SE environment, persistence.xml must list the Entities—Dali offers synchronization



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.coı
    <persistence-unit name="jdj"/>
</persistence>
```



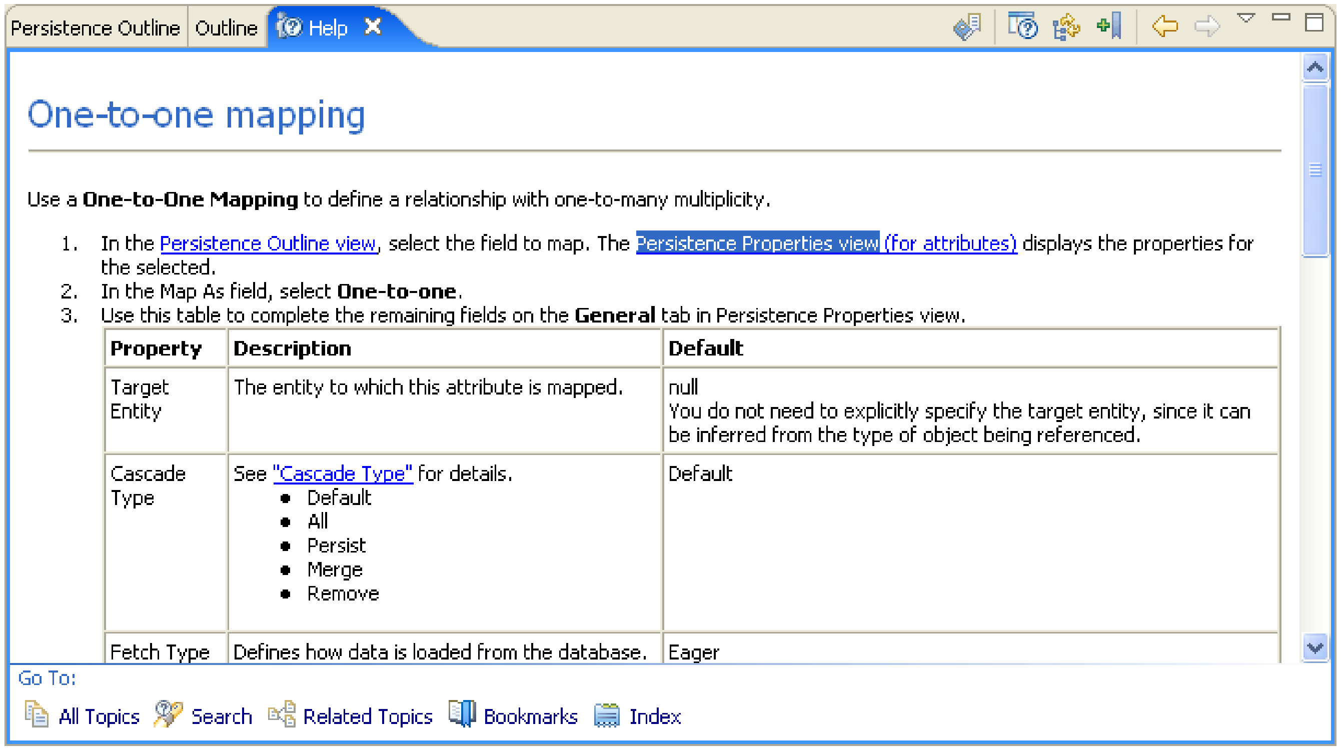
Synchronizing Persistence.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xm
    <persistence-unit name="jdj">
        <class>model.Customer</class>
        <class>model.CustomerInfo</class>
        <class>model.Invoice</class>
        <class>model.Phone</class>
    </persistence-unit>
</persistence>
```

Integrated F1 Help

- In any Dali view you can hit F1 to get context specific help.



One-to-one mapping

Use a **One-to-One Mapping** to define a relationship with one-to-many multiplicity.

1. In the Persistence Outline view, select the field to map. The Persistence Properties view (for attributes) displays the properties for the selected.
2. In the Map As field, select **One-to-one**.
3. Use this table to complete the remaining fields on the **General** tab in Persistence Properties view.

Property	**Description**	**Default**
Target Entity	The entity to which this attribute is mapped.	null You do not need to explicitly specify the target entity, since it can be inferred from the type of object being referenced.
Cascade Type	See "Cascade Type" for details. • Default • All • Persist • Merge • Remove	Default
Fetch Type	Defines how data is loaded from the database.	Eager

Dali 0.5 Features

- Support for majority of annotations including relationships
- Integration with WTP RDB components
- JPA Details view supports mapping using database schema with table and column drop downs
- Design time validation of mappings against object and data model (including unspecified default mappings)
- Problems markers (e.g., Entity missing @Id)
- Wizards for basic Entity generation from tables

Dali 1.0 Features

- Support for XML Mapping File (orm.xml) configuration
  - Uses same views as JPA Annotation editing
- Integration with Data Tools Platform (DTP) for database meta-data
- Facet based functionality for better integration with various WTP project types
- Enhanced design-time validation for XML and Java Annotations mappings as well as the combination as defined by the spec.
- Extensibility for vendor specific extensions

JPA Application Development Scenarios

Possible Approaches using Dali

- Meet in the Middle
  - existing object and data models
- Bottom Up
  - generate mapped object model from data model
- Top Down
  - generate data model from mapped object model

Meet In the Middle (MITM)

- The advantage of MITM is that you can focus on getting your object model and data models correct.
  - Table != Class
    - 1 Table could be N classes (using embedded)
    - N tables could be 1 class (with secondary table)
  - Use Java language features like inheritance not present in relational model
- Dali's validation makes MITM practical
  - Avoids map, deploy, debug cycle
  - Provides access to database schema to provide valid choices

Bottom Up

- Generate Entities from Tables
  - Great way to bootstrap a JPA application from an existing database
  - Uses an Entity == Table approach
  - Do it once and then modify the generated Entities
    - Dali mapping validation will help identify issues resulting from modifications to Entities

Top Down

- Generate DDL from Entities
  - Some support in Dali 0.5 release
  - Removed in 1.0 in favor of using full featured support implemented by JPA runtimes.
  - Extension point in Dali to allow for plugging in runtime DDL generation.

Dali Adoption

- Oracle
  - Providing resources for Dali project
  - Building extensions to Dali for EclipseLink
- RedHat/JBoss
  - Plans to bundle/integrate with Dali for Hibernate Tools
  - Dali project member
- IBM
  - Incorporating Dali into future tooling products
- SAP
  - Shipped Dali in Eclipse based SAP NetWeaver Developer Studio Java EE 5 preview

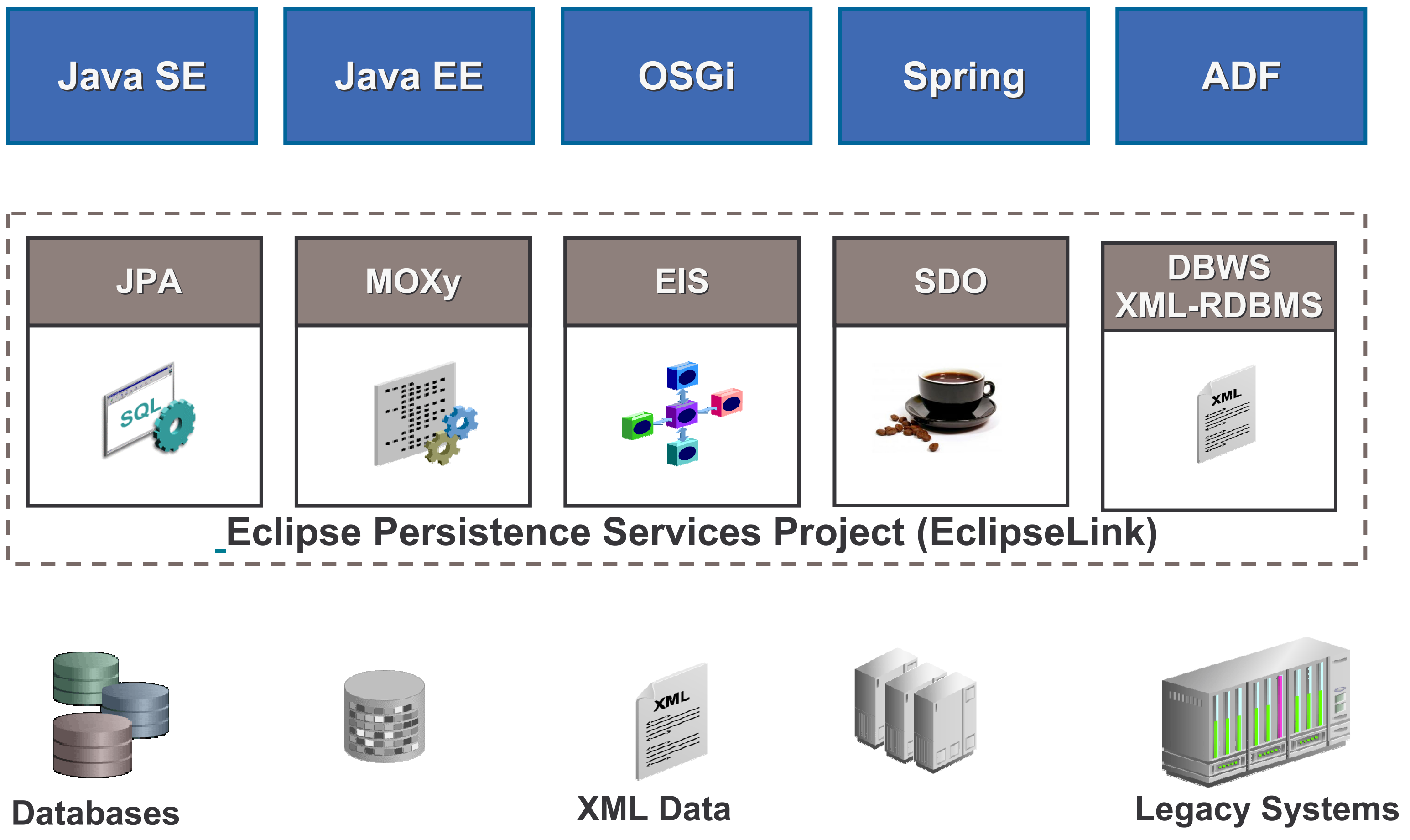
Dali 2.0 Plans

- Themes
  - Extensibility
  - Comprehensive JPA configuration
  - Enhanced Validation
  - Usability/Integration
- Related requirements
  - Public API for adopters
  - Persistence.xml UI
  - Named Query definition
  - Project Explorer

Dali Summary

- JPA—the new Java EE standard for object-relational mapping (inside and outside of a container)
- Dali—the WTP project bringing developer productivity to JPA
  - Mapping validation to avoid the map, deploy, debug cycle
  - Intelligent mapping assistance to avoid problems and speed up the process of mapping
  - Integrated with WTP to support development for Java SE and EE

Eclipse Persistence Services Project – *“EclipseLink”*



EclipseLink JPA Extensions

- Query Hints
  - Pessimistic Locking (*toplink-pessimistic-lock*)
  - Refresh (*toplink.refresh*)
- Persistence Unit Properties
  - JDBC connection pooling
  - Logging
  - DDL Generation
  - Platforms
  - Customization callbacks
  - Caching
- Weaving

JDBC Connection Settings

- Resource-level JDBC technology settings are vendors responsibility
- Need to specify the four basic JDBC technology properties to obtain driver connections
  - Driver class, URL, username, password
- The property keys will be different, but the values for a given JDBC technology data source will be the same for all vendors
- Used when not in a container, or when managed data sources are not available or not desired

JDBC Connection Settings

```
<properties>
...
<property name="eclipselink.jdbc.driver"
  value="oracle.jdbc.Driver"/>

<property name="eclipselink.jdbc.url"
  value="jdbc:oracle:thin:@localhost:1521:XE"/>

<property name="eclipselink.jdbc.user"
  value="scott"/>

<property name="eclipselink.jdbc.password"
  value="tiger"/>
```

Logging

- Users want to control over logging, but vendors use different logging APIs
- Can usually configure to use one of the well-known logging APIs
  - java.util.logging, log4J, etc.
- Common requirement is to configure the logging level to show the generated SQL

Logging

```
<properties>
  ...

  <property
        name=“eclipselink.logging.level"
        value="FINE"/>
  ...

</properties>
```

DDL Generation

- Standard enables it but does not currently dictate that providers support it
- Mapping metadata specifies how DDL should be generated
- Vendors may offer differing levels of support, including:
  - Generating DDL to a file only
  - Generating and executing DDL in DB
  - Dropping existing tables before creating new ones

DDL Generation

```
<properties>
  ...
 <property
      name=“eclipselink.ddl-generation"
      value="create-tables"/>
  ...
</properties>
```

Database Platform

XML:

```
<properties>
<property
      name="eclipselink.target-database"
      value="Derby"/>
</properties>
```

API:

```
  properties.put(
   EclipseLinkProperties.TARGET_DATABASE,
   TargetDatabase.ORACLE);
```

Target Database Options

```
public class TargetDatabase {
    public static final String  Auto = "Auto";
    public static final String  Oracle = "Oracle";
    public static final String  DB2 = "DB2";
    public static final String  DB2Mainframe = "DB2Mainframe";
    public static final String  Derby = "Derby";
    public static final String  HSQL = "HSQL";
    public static final String  Informix = "Informix";
    public static final String  JavaDB = "JavaDB";
    public static final String  MySQL4 = "MySQL4";
    public static final String  PointBase = "PointBase";
    public static final String  PostgreSQL = "PostgreSQL";
    public static final String  SQLAnyWhere = "SQLAnyWhere";
    public static final String  SQLServer = "SQLServer";
    public static final String  Sybase = "Sybase";
    public static final String  TimesTen = "TimesTen";
}
```

Server Platform

- Enables simplified configuration of the target application server
- Used to enable integration with:
  - JTA transaction manager
  - Logging
  - JDBC connection un-wrapping

```
<property name="eclipselink.target-server"
          value="SunAS9"/>
```

- Supported options: *None, SunAS9, OC4J_10_1_3*

Customization Using Properties

```
<properties>
  ...
  <property
    name="toplink.session.customizer"
    value="acme.MySessionCustomizer"/>
  <property
    name="toplink.descriptor.customizer.Employee"
    value="acme.MyDescriptorCustomizer"/>
  ...
</properties>
```

Descriptor & Session Customizers

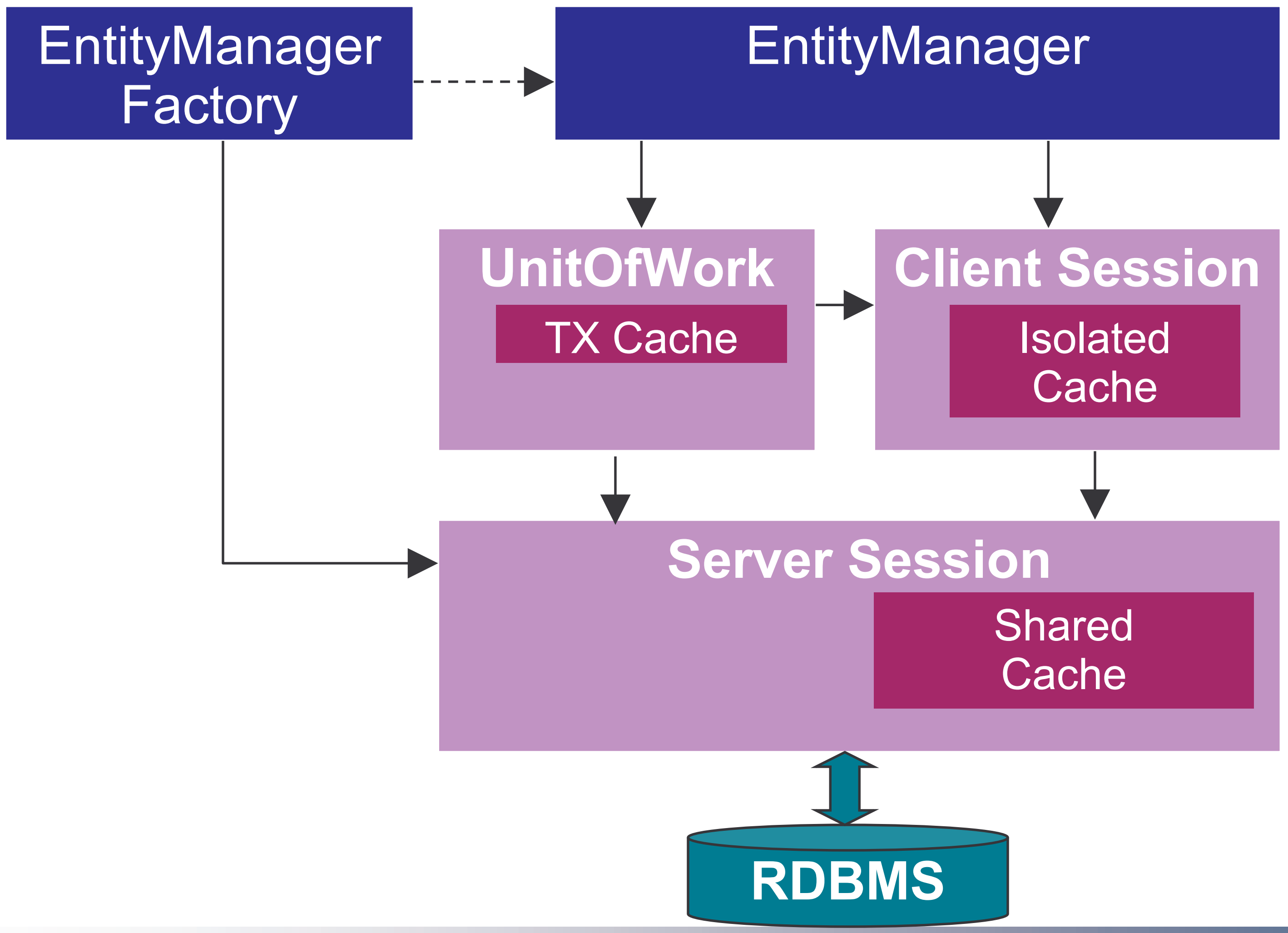
```
public class MySessionCustomizer
                implements SessionCustomizer {

  public void customize(Session session) {
   session.setProfiler(new PerformanceProfiler());
  }
}

public class MyDescriptorCustomizer
                implements DescriptorCustomizer {

  public void customize(ClassDescriptor desc) {
    desc.disableCacheHits();
  }
}
```

Concurrency Protection - Locking

- Optimistic concurrency is built into JPA, but no support for pessimistic locking is specified
- Will likely be addressed in a future JPA release
- All credible JPA implementations support pessimistic locks in some way or another
- EntityManager lock() method can be used with optimistic locking, and error handling
- EclipseLink query hint for pessimistic locking

```
query.setHint(PESSIMISTIC_LOCK, LockNoWait);
```

EclipseLink Caching



Cache Configuration

- Cache Shared/Isolated

```
<property name="eclipselink.cache.shared.default"
  value="true"/>
```

- Cache Type & Size
  - SoftWeak, HardWeak
  - Weak
  - Full
  - None

```
<property name="eclipselink.cache.type.default"
  value="Full"/>
<property name="eclipselink.cache.type.MyEntity"
  value="Weak"/>
```

Minimize stale cache

1. Configure the cache relative to application data’s usage
   - Is the data shared between users/processes?
   - Is the data volatile?
     - Only through JPA application?
     - Through direct DB modification?
2. Ensure you protect any data that can be concurrently modified with a locking strategy
   - Must handle optimistic lock failures on flush/commit
3. Use query refreshing to minimize optimistic lock failures

Weaving Support

- EclipseLink makes use of Weaving (ASM) to introduce additional functionality into the JPA entity classes
  - Needed for M:1 and 1:1 lazy fetching
  - Integrated with OC4J 10.1.3.1 and Spring 2.0
  - Available for Java SE using JDK/JRE's –javaagent:
  - Optional
  - Static weaving also supported
    - Weaving of .class files before deployment

EclipseLink Status

- Incubating Technology Project
- Initial contribution of Oracle TopLink complete
- 1.0 Milestone 1 Features
  - Automated test and build
  - JPA 1.0 compliance verified by Oracle
  - MOXy
  - SDO
  - Wiki Content to help early adopters

Getting Started with Eclipse JPA

- EJB 3.0 & JPA Specification
  - JPA 1.0: www.jcp.org/en/jsr/detail?id=220
  - JPA 2.0: www.jcp.org/en/jsr/detail?id=317
- Dali Project
  - www.eclipse.org/dali
- EclipseLink
  - www.eclipse.org/eclipselink