Track # 1: Session #2
Developing Enterprise JavaBeans using the EJB 3.0 Specification
Agenda

- **Introduction to the EJB 3.0 Specification**
  - What are Enterprise JavaBeans (EJB)?
  - Goals
  - Comparison with 2.1 specification
- **Developing 3.0 Entity Beans (Entities)**
  - Persistence API
- **Developing 3.0 Session Beans**
  - Client view

Enterprise JavaBeans

- **Middle-tier Java components**
  - encapsulating business logic
- **Hosted in EJB containers**
- **Provide services for clients**
  - Business logic
  - Persistence
  - Messaging
Types of EJ Bs

<table>
<thead>
<tr>
<th>EJB Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session Beans</td>
<td>Performs a business task for a client</td>
</tr>
<tr>
<td>Entity Beans</td>
<td>Represents a business object that exists in a database</td>
</tr>
<tr>
<td>Message-Driven Beans</td>
<td>Receives asynchronous Java Message Service (JMS) messages</td>
</tr>
</tbody>
</table>

EJB 3.0 Goals

• **Simplify Development**
  – EJB = Plain Java Object (POJO)
  – Use metadata annotations
  – Reduce number of artifacts
  – Attract broad range of developers

• **Standardize persistence API**
  – O-R Mapping similar to Oracle TopLink, Hibernate
  – EntityManager API for CRUD Operations
**EJB 3.0 Simplification**

**POJO and POJI**
- Removes complexity of earlier versions using simple and familiar Java artifacts
- EJB Class will be a POJO
- EJB Interface will be a POJI (no EJB extensions)
- No need for home interface
- Annotations for type of EJB and interface

**EJB 3.0 Simplification Analysis**

The Simplicity of EJB 3.0
http://java.sys-con.com/read/117755.htm
Agenda

• Introduction to the 3.0 Specification
  – Goals
  – Comparison with 2.1 specification
• Developing 3.0 Entity Beans (Entities)
  – Persistence API
• Developing 3.0 Session Beans
  – Client view

EJB Persistence

• Simple programming model
  – Proven POJO persistence
  – O-R mapping annotations and Entity Manager API
• Improved modelling capabilities
  – Inheritance and polymorphism
• Entities usable outside the container
  – Facilitates testability
POJO Entities

• Concrete classes (no longer abstract)
• No required interfaces
• Support new()
• getter/setter methods with annotations
  – can contain logic (validation, etc.)
• Collection interfaces used for relationships
• Usable outside the EJB container

Example: EJB 3.0 Entity

```java
@Entity public class Customer {
    private Long id;
    private String name;
    private HashSet orders = new HashSet();

    @Id (generate=SEQUENCE, generator="SEQ_GEN")
    @SequenceGenerator(name="SEQ_GEN",
    @sequenceName="CUST_SEQ",allocationSize=1)
    @Column(name = "ID", primaryKey = true)

    public Long getId() {  return id;  }

    protected void setId (Long id)
    {  this.id = id;  }

    ...
```
Example: EJB 3.0 Entity

```java
...  
@OneToMany(cascade=ALL, mappedBy="customer")
public Set<Order> getOrders()
{  return orders;  }

public void setOrders(Set<Order> orders)
{  this.orders = orders;  }

// other business methods, etc.
}
```

O-R Mapping

- Use Java application metadata annotations or XML to specify mapping
- Ability to map one or more persistent object to a table
  - Embeddable
- Support for typical inheritance strategies
  - Single table per class hierarchy
  - Table per class
  - Joined subclass
- Default type mappings defined by specification
- Custom type mappings for finer control and flexibility
EntityManager

- EntityManager serves as untyped “home”
- Provides lifecycle operations
  - persist()
  - remove()
  - merge()
  - flush(), refresh(), etc.
- Factory for Query objects

EntityManager API example

```java
@Resource public EntityManager em;

  public void addLoan(String provider, long term,
                        String loan_type, double interest_rate)
  {
    Loans loan = new Loans();
    loan.setProvider(provider);
    loan.setTerm(term);
    loan.setLoanType(loan_type);
    loan.setInterestRate(interest_rate);
    em.persist(loan);
  }
```
Query API

- Utilize EJBQL, Expressions, SQL
- Defined dynamically or stored within an Entity

```java
public List findWithName(String name) {
    return em.createQuery(
        "SELECT c FROM Customer c"
        + "WHERE c.name LIKE :custName"
    ).setParameter("custName", name)
    .setMaxResults(10)
    .getResultList();
}
```

SQL

- Allow direct SQL over actual database schema
  - Very useful for some applications
  - Database portability overrated for some applications
- Allow SQL query results to be mapped into entities and/or instances of other Java classes
DEMONSTRATION

Developing Entities

Agenda

- Introduction to the 3.0 Specification
  - Goals
  - Comparison with 2.1 specification
- Developing 3.0 Entity Beans (Entities)
  - Persistence API
- Developing 3.0 Session Beans
  - Client view
EJB 3.0 Session Beans

- Java class with @Stateless/Stateful annotation
- Business interface is a POJI
  - Interface with @Local/Remote/WebService annotation(s)
  - RemoteExceptions removed from programmer and client view
- Home Interface not required
- No EJB extensions required for bean or interface(s)

EJB 3.0 Session

```java
@Remote
public interface Cart {
    public void addItem(String item);
    public void completeOrder();
    public Collection getItems();
}
```

```java
@Stateful
public class CartBean implements Cart {
    private ArrayList items;
    public void addItem(String item) { items.add(item); }
    public Collection getItems() { return items; }
    @Remove
    public void completeOrder() { ... }
}
```
Dependency Injection

- **Resources**
  - DataSource, JMS, etc

```java
@Resource(name="jms/lodging/QueueConnectionFactory")
private QueueConnectionFactory queueConnectionFactory;
```

- **Environments**
  - EJB Context, environment variables

- **Other EJBs**
  - Session Beans, entities (using EM API)

```java
@Resource public EntityManager em;
```

Enhanced Lifecycle Methods

- No need to implement unnecessary callback methods
- Mark any arbitrary methods as callback method using annotations or XML

```java
@PostConstruct
    public void initialize() {
        items = new ArrayList();
    }
```
EJB 3.0 Client View

@Stateful
public class OrderBean {
    //Dependency injection to use another EJB
    @EJB CartEJB cart;
    public void addItem() {
        cart.addItem("Item1");
    }
}

DEMONSTRATION

Developing Session Beans
Oracle and EJB 3.0

- **Oracle is co-specification lead for EJB 3.0**
  - Oracle to contribute the reference implementation for EJB 3.0 based on TopLink

- **Oracle is a leader in J2EE and first major application server vendor to support EJB 3.0**
  - Will easily facilitate migration to EJB 3.0

- **Oracle is leading an Eclipse tools project on EJB 3.0 persistence**

- **EJB3 Resource Center:**
  - [http://otn.oracle.com/ejb3](http://otn.oracle.com/ejb3)

---

*Join Over 4,500,000 Developers!*

- Free Technical Advice
- Free Software Downloads
  - [otn.oracle.com.ejb3](http://otn.oracle.com.ejb3)
  - [otn.oracle.com/tech/java](http://otn.oracle.com/tech/java)
  - [otn.oracle.com/bpel](http://otn.oracle.com/bpel)
Learn Oracle From Oracle

- Instructor led training
- Self-Study
- Online learning
- Oracle Certification
- Oracle iLearning
- Oracle Tutor

oracle.com/education

Questions & Answers
EJB QL Enhancements

• Bulk update and delete operations
• Projection list (SELECT clause)
• Group by, Having
• Sub-queries (correlated and not)
• Additional SQL functions
  – UPPER, LOWER, TRIM, CURRENT_DATE, ...
• Dynamic queries
Named Queries

```java
@NamedQuery(
    name="findCustomersByName",
    queryString="SELECT c FROM Customer c +
    "WHERE c.name LIKE :custName"
)
```

```java
@Resource public EntityManager em;
...
List customers =
    em.createNamedQuery("findCustomersByName")
    .setParameter("custName", "Smith")
    .getResultList();
```

Interceptors

- Provides fine grained control over the method invocation flow
  - may be either a method in the same bean class or an external class
  - Used with SLSB, SFSB, MDB
- Usage
  - Modify parameters before they're passed to the bean
  - Modify the value returned from the bean
  - Catch and swallow method exceptions
  - Interrupt the call completely (handy for a home-grown security framework)
  - Provide method profiling
Interceptor Example

```java
@Stateless
@Interceptor(value="oracle.ejb30.ProfilingInterceptor")
// identify external interceptors
public class HelloWorldBean implements HelloWorld {
    @AroundInvoke
    // mark this method as a bean interceptor
    public Object checkPermission(InvocationContext ctx)
        throws Exception {
        System.out.println("*** checkPermission interceptor invoked");
        return null;
    }
}
```

Agenda

- **Introduction to the 3.0 Specification**
  - Goals
  - Comparison with 2.1 specification

- **Developing 3.0 Entity Beans (Entities)**
  - Persistence API

- **Developing 3.0 Session Beans**
  - Client view