

# EJB 3.0: There's Something Different About You

Dennis Leung – VP Development, Oracle

ORACLE

1

## Oracle Space Sweepstakes

### HAUPTPREIS (1):

Suborbitaler Raumflug\*,  
zur Verfügung gestellt von Space Adventures.



Be the First Java  
Developer in Space

>> REGISTER  
for your chance to win!

ORACLE

2

[www.otn.oracle.com](http://www.otn.oracle.com) / [www.oracle.de](http://www.oracle.de)

- Beim Flug in den Orbit betrachten Sie die Erde aus einer komplett anderen Perspektive und erleben die Sterne und das Weltall hautnah. Natürlich kümmert sich Oracle um alles, damit Sie bestens vorbereitet Ihre einmalige Reise vollen Zügen genießen können. Es erwarten Sie:
  - ein viertägiges Trainingsprogramm vor dem Flug
  - ein Flug 100 Kilometer weit ins All
  - ein atemberaubender Blick auf die Erde
  - 5 bis 10 Minuten der Schwerelosigkeit
  - Zusätzlich 35.000 US\$ in bar!
- Gesamtwert: 138.000 US\$

ORACLE

3

## Agenda

1. EJB 2.1 – Improvements Needed
2. EJB 3.0 – Overview of Changes
3. Simplification Examples
4. Oracle Support of JSR 220 - EJB 3.0

ORACLE

4

## EJB 2.1 – Complexity and Overhead

- EJB 2.1 was difficult to use and had a lot of overhead
  - Heavyweight component persistence model
  - “Fixed” costs regardless of how simple the app was
  - Too many artifacts that needed to be managed
  - Made the hard apps possible but made simple apps hard

ORACLE

5

## EJB 2.1 Technology View

- Contractual view of components
- Container specifies contracts
  - Interfaces
  - Callbacks
  - Deployment descriptor contract
- Beans written to contracts
  - Whether they need them or not
- This is a very static viewpoint
  - Significant negative impact on application complexity

ORACLE

6

## Ease of Development Goal

- Simplify EJB — make it easier to use
  - Simplified set of APIs
  - Eliminate deployment descriptor from developer's view
  - Facilitate test-driven development
  - Improve developer productivity
- Capture broader range of developers
  - Make it simpler for average developer
  - Increase developer base, target more corporate developers

ORACLE

7

## EJB 3.0 Expert Group Agenda

- Define “Simplified API” for EJB 3.0
- Critical examination of EJB complexity
  - “Data-mining” of common complaints, RFEs
  - Anti-patterns, etc.
- Focus on most common cases
- Ensure EJB 2.0/2.1 APIs are still supported
  - Simplifications / enhancements to improve use of these APIs as well

ORACLE

8

## EJB 3.0 Direction

- Simplify developer experience
  - EJB's now Plain Old Java Objects (POJO)
  - Can use metadata annotations
    - XML descriptors are no longer necessary
    - Use defaults when possible
  - Unnecessary artifacts are optional
  - Simplify resource access using dependency injection
- Standardize persistence API for Java platform
  - Based on success of leading ORM solutions

ORACLE

9

## EJB 3.0 – Technology View

- View of the container as a service injection facility
  - Injection points specified through metadata
  - View of the container “disappears”
- View of “component” changes:
  - Classes that require / request services
  - Pick and choose what they need
- Contractual view becomes inverted
  - Bean does not implement predefined set of APIs
  - Instead, bean specifies what it needs
- More extensible architecture => our view of “container” evolves

ORACLE

10

## Java Metadata Interface

- Annotations attach metadata to code
- Great enabling technology for EJB software
  - Elimination of mandatory deployment descriptors
- “Configuration by Exception”
- Leverage defaulting in conjunction with metadata
  - Default values for annotation members
  - Metadata itself used for non-default cases
- Gives developer very simple-to-use and yet powerful capability

ORACLE

11

## Simplification Through Defaults

- Default values are applied automatically
  - Defaulting of names
  - Defaulting use of transaction management types
  - Defaulting of transaction attributes
  - Default use of unchecked methods
  - Default use of caller identity
  - Etc.
- Any default can be overridden through metadata

ORACLE

12

## EJB 3.0 Overview: Environment

- Simplification of access to bean's environment and runtime context
  - Encapsulation of environment dependencies and JNDI access
    - Through metadata annotations
    - Dependency injection mechanisms
  - Simplification of access to other components

ORACLE

13

## EJB 3.0 Overview: POJOs

- Simplification of enterprise bean types
  - More closely resemble “POJOs”
- Elimination of requirement for EJB component interfaces
  - Use “POJIs” for session beans
  - No requirement for entity bean interfaces
- Elimination of requirement for Home interfaces
- Elimination of requirement for callback interfaces
  - Allow selective implementation of callback methods

ORACLE

14

## EJB 3.0 Overview: CMP

- Simplification of container-managed persistence
  - POJO architecture approach
  - Support use of new()
  - Allow for testing outside the container
- Support for light-weight domain modelling, including
  - Inheritance and polymorphism
  - Object-relational mapping metadata
- Elimination of need for data transfer objects and related anti-patterns

ORACLE

15

## Prototype: Session Beans

- Let's compare and analyze what it would take to build a simple Session Bean
- “Hello World” entry experience into EJB
  - Create a shopping cart session bean

ORACLE

16

## EJB 2.1 SFSB Required Steps

1. Create Cart business interface
2. Create Cart home interface
3. Create Cart Session Bean
4. Implement business methods
5. Implement lifecycle methods
6. Create deployment descriptor

ORACLE

17

## EJB 2.1 Session Bean

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Cart extends EJBObject
{
    public void add(String item) throws RemoteException;
    public Collection getItems() throws RemoteException;
    public void completeOrder() throws RemoteException;
}
```

```
import javax.ejb.*;
import java.rmi.RemoteException;
public interface CartHome extends EJBHome
{
    public Cart create() throws CreateException,
                          RemoteException;
}
```

ORACLE

18

## EJB 2.1 Session Bean Class

```
import javax.ejb.*;
public class CartEJB implements SessionBean {
    protected Collection items = new ArrayList();
    public void add(String item) {
        items.add(item);
    }
    public Collection getItems() {
        return items;
    }
    public void completeOrder(){ .. }

    public void ejbCreate(){}
    public void ejbRemove(){}
    public void ejbActivate(){}
    public void ejbPassivate(){}
    public void setSessionContext(SessionContext context){}
}
```

ORACLE

19

## EJB 2.1 Deployment Descriptor

```
<session>
  <display-name>Shopping Cart</display-name>
  <ejb-name>MyCart</ejb-name>
  <home>CartHome</home>
  <remote>Cart</remote>
  <ejb-class>CartEJB</ejb-class>
  <session-type>Stateful</session-type>
  <transaction-type>Container</transaction-type>
</session>
...
<assembly-descriptor>
  <container-transaction>
    <method> ...
  </method>
  <trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>
```

ORACLE

20

## Pain points

- Heavy-weight programmer view
- Bean class has no compile time dependency on business interfaces
- Requires methods that serve no purpose
- Extend a lot of interfaces and implementation classes
- Too many artifacts
  - Remote, RemoteHome, Local, LocalHome, WebService EndPoint, Bean & descriptors
- Complex XML deployment descriptors

ORACLE

21

## EJB 3.0 Session Bean

```
public interface Cart {  
    public void add(String item);  
    public Collection getItems();  
    public void completeOrder();  
}
```

ORACLE

22

## EJB 3.0 Session Bean

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

ORACLE

23

## EJB 3.0 Session Bean class

```
import javax.ejb.*;

public class CartBean implements Cart {
    private ArrayList items;

    public void add(String item) {
        items.add(item);
    }
    public Collection getItems() {
        return items;
    }

    public void completeOrder() {...}
}
```

ORACLE

24

## EJB 3.0 Session Bean class

```
import javax.ejb.*;

@Stateful
public class CartBean implements Cart {
    private ArrayList items;

    public void add(String item) {
        items.add(item);
    }
    public Collection getItems() {
        return items;
    }
    @Remove
    public void completeOrder() {...}
}
```

ORACLE

25

## Deployment Descriptor



ORACLE

26

## Significant Simplifications

- Eliminated requirement for Home Interface
  - Not needed for session beans
- Business interface is a POJI
  - Don't need to extend specific interface
  - Bean implements business interface
  - Bean can have more than one business interface
  - RemoteExceptions are removed from programmer and client view
- Eliminated requirement for unnecessary callback methods
  - Removed requirement to implement `javax.ejb.SessionBean`

ORACLE

27

## Simplified Access to Bean's Environment

- Get JNDI APIs out of developer's view
  - Not at all a good "hello world" experience
- Declarative expression of dependencies in metadata
- Techniques/Mechanism
  - Container injection of resource entries, etc.
  - Simple programmatic lookup mechanisms
- Different usages, both have their place
  - Very simple; facilitates testability (especially setter injection techniques)
  - More flexible; dynamic

ORACLE

28

## Example

```
@Stateless public class MySessionBean {  
  
    @Resource public DataSource customerDB;  
  
    public void myMethod (String myString){  
        try {  
            Connection conn =  
                customerDB.getConnection();  
            ...  
        } catch (Exception ex) {...}  
    }  
}
```

ORACLE

29

## With Setter Injection

```
@Stateless public class MySessionBean {  
  
    private DataSource customerDB;  
  
    @Resource  
    public void setDataSource(DataSource myDB) {  
        customerDB = myDB;  
    }  
  
    public void myMethod (String myString){  
        ...  
        Connection conn = customerDB.getConnection();  
        ...  
    }  
}
```

ORACLE

30

## Dynamic Lookup

- JNDI lookup is greatly simplified
  - Adding lookup() method to EJBContext
  - Can use injection to get EJBContext

```
@Resource
private void setSessionContext(SessionContext ctx) {
    this.ctx = ctx;
}
...
myShoppingCart =
    (ShoppingCart) ctx.lookup("shoppingCart");
```

ORACLE

31

## Entity Beans: Goals

- Simplify programming model
- Improve modeling capabilities
  - Inheritance and polymorphism;
- Add standard O/R mapping
- Make instances usable outside the container
- Facilitate test driven development
- Remove need for data transfer objects (DTOs)
- Leverage industry best practices

ORACLE

32

## POJO Entity Beans

- Concrete classes (no longer abstract)
- No required interfaces
  - No required business interfaces
  - No required callback interfaces
- Supports new()
- getter/setter methods are not abstract
  - can contain logic (e.g., for validation, etc.)
- Usable (and testable) outside the EJB container

ORACLE

33

## EntityManager

- No Entity Bean Homes
- EntityManager serves as un-typed home or “session”
  - Provides lifecycle operations
    - create(), remove(), merge() etc.
    - Factory for Query objects
- Can wrapper EntityManager to provided typed homes
  - Allows for strong typing design

ORACLE

34

## Prototype: Entity Beans

- “Hello World” example
  - Develop a Customer Entity Bean
  - Create a method to add an Account to it
  - Test the bean

ORACLE

35

## EJB 2.1 Entity Bean Steps

1. Create Business Interface (getters and setters)
2. Create Primary Key Class
3. Create Home Interface (create and finder methods)
4. Create abstract Bean class
5. Implement lifecycle methods
6. Implement business methods
7. Create descriptor file

ORACLE

36

## EJB 2.1 Entity Bean Interfaces

```
import javax.ejb.*;
public interface Customer extends EJBLocalObject
{
    public abstract String getName();
    public abstract void setName (String name);
    public abstract Account getAccount();
    public abstract void setAccount(Account acct);
}
```

```
import javax.ejb.*;
public interface CustomerHome extends EJBLocalHome
{
    public abstract Customer create(String name);
    public abstract Customer
        findByPrimaryKey(String name);
}
```

ORACLE

37

## EJB 2.1 Entity Bean class

```
import javax.ejb.*;
public abstract class CustomerBean
    implements EntityBean {

    public abstract String getName();
    public abstract void setName(String name);
    public abstract Account getAccount();
    public abstract void setAccount(Account acct);}

    // plus life cycle methods...
```

ORACLE

38

## EJB 2.1 Entity Bean class

```
// Life cycle methods

public String ejbCreate(String name)
    throws CreateException {...}
public String ejbPostCreate(String name)
    throws CreateException {}
public void ejbRemove() throws RemoveException {}
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbLoad() {}
public void ejbStore() {}
public void setEntityContext(EntityContext ec) {}
public void unsetEntityContext() {}
}
```

ORACLE

39

## Steps to Test It

1. Write servlet or client application with test case
2. Package test case (WAR or client jar) and bean (EJB jar) together into an EAR file
3. Deploy EAR to server
4. Run test
5. Repeat

ORACLE

40

## Developer Pain Points

- All the complexity of session beans and more
- Entity relationships are defined through complicated XML
- Bean class mixes business method and home method implementations
- Cannot be tested outside of the server
  - Abstract class with no testable state
  - Local interfaces can only be tested by servlets or session beans

ORACLE

41

## EJB 3.0 Entity Bean

```
public class Customer {  
  
    private String name;  
    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;  
    }  
}
```

ORACLE

42

## EJB 3.0 Entity Bean

```
@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;  
    }  
}
```

ORACLE

43

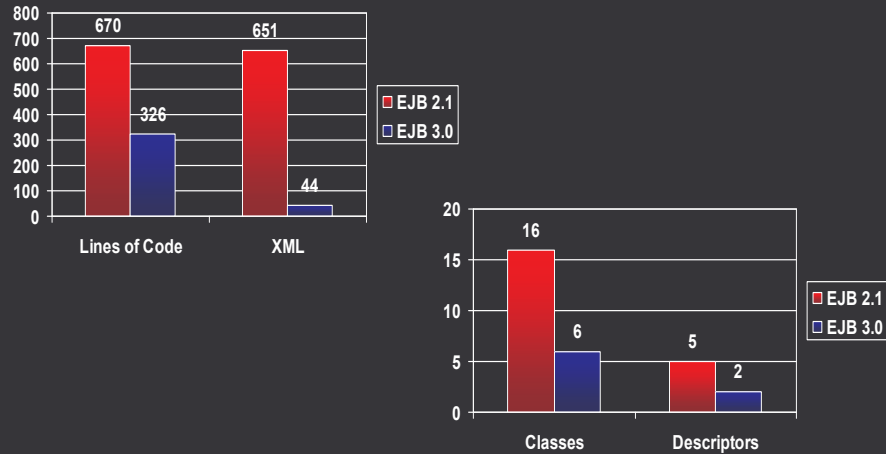
## EntityManager Injection

```
@Stateless public class CustomerAccessBean {  
    private EntityManager em;  
  
    @Inject void setEntityManager(EntityManager em) {  
        this.em = em;  
    }  
    public void updateAccountAccess(String cust) {  
        Customer c = em.find(Customer.class, cust);  
        c.getAccount().setLastAccessTime();  
    }  
}
```

ORACLE

44

## EJB 2.1 Versus 3.0: Simplifying Complexity



ORACLE

45

## Flexibility and Performance

- Flexibility and performance are not being ignored for ease of development
  - O/R Mapping
    - Custom type mappings, inheritance ...
  - EJB QL Enhancements
    - Sub queries, bulk operations, group-by ...
  - Named queries, direct SQL, dynamic queries
  - Eager and lazy relationship fetching
  - Optimistic locking – version & timestamp

ORACLE

46

## Oracle Support of EJB 3.0

- Primary contributor (Mike Keith) on JSR 220 Expert Group
- Released EJB 3.0 preview in March 2005  
[www.otn.oracle.com](http://www.otn.oracle.com)
  - Includes Entity Test Harness for out-of-container testing
- Leading EJB 3.0 Eclipse project
- Today Oracle TopLink supports both POJO and EJB 2.1 and 3.0
  - Provides seamless path to EJB 3.0
- Substantial EJB 3.0 functionality in 10.1.3 production release – 2<sup>nd</sup> half 2005
  - Usable in JDK 1.4 with XML descriptors

ORACLE

47

## Summary

- EJB 3.0 significantly improves ease of development
- It is real and can be used today
- Is the direction for Java persistence
  - Based on proven technology
- Get a preview and try it out yourself!

ORACLE

48