Enterprise JavaBeans 2.0

Dan Harkey
Client/Server Computing Program Director
San Jose State University
dharkey@email.sjsu.edu
www.corbajava.engr.sjsu.edu

Agenda

✓ Overview of new Features of EJB 2.0
✓ EJB 2.0 Details
EJB 2.0: New Features

- New Container-Managed Persistence Features
  - "Persistence Manager"
- Local Interfaces
- Container-Managed Relationships
- EJB QL
- Home Methods
- Select Methods
- Integration with JMS
  - Message Driven Beans
- Network Interoperability

EJB 2.0: New Features

- Defines Entity instance state and relationships via abstract bean class and get/set abstract methods
- Container provides implementation of abstract class
Entity Bean Class Requirements

- **EJB Class** must be declared public and abstract.
- **Public, Abstract** accessor (get/set) methods must replace the CMP fields. Relationship fields may also be defined.
  - Data fields must not be defined in the class.
  - Fields must be defined in the deployment descriptor using `<cmp-field>` tag. The name must start with a lower-case character.
  - The accessor methods of cmp-fields must not be exposed in the remote interface.
  - The field types must be Java primitive types or serializeable types.

Entity Bean Class Requirements (cont)

- **Must implement the following:**
  - The EntityBean interface
  - Zero or more ejbCreate and ejbPostCreate methods
  - The get and set access methods, defined as abstract, for the persistent and relationship fields
  - Any select methods, defining them as abstract
  - The home methods
  - The business methods

- **Must not implement:**
  - The finder methods (for CMP)
  - The finalize method
EJB 2.0 Entity Lifecycle
(Modified to include ejbSelect methods)

1. newInstance()
2. setEntityContext()
3. ejbCreate()<METHOD>=<args>
4. ejbPostCreate()<METHOD>=<args>
5. ejbActivate()<METHOD>=<args>
6. ejbPassivate()<METHOD>=<args>
7. ejbLoad()<METHOD>=<args>
8. ejbStore()<METHOD>=<args>
9. ejbRemove()<METHOD>=<args>
10. setEntityContext()<METHOD>=<args>

instance throws

system exception

from any method

does not exist

pooled

_ready_

business method

Count Entity CMP 2.0 EJB

<<Interface>>

EJBHome
(from ejb)

<<Interface>>

CountHome
(from entity20)

<<Interface>>

Count
(from entity20)

<<Interface>>

EJBObject
(from ejb)

<<Interface>>

EntityBean
(from ejb)

<<Interface>>

CountClient
(from entity20)

Copyright 2001-2002, Dan Harkey. All rights reserved.
// Count.java -- Count Entity Remote Interface
package count.entity20;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Count extends EJBObject {
    public int getCurrentSum() throws RemoteException;
    public void setCurrentSum(int val) throws RemoteException;
    public int increment() throws RemoteException;
}

// CountHome.java -- Count Entity Remote Interface
package count.entity20;
import count.entity20.Count;
import java.rmi.RemoteException;
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface CountHome extends EJBHome {
    public Count create(String name, int initialSum)
        throws RemoteException, CreateException;
    public Count findByPrimaryKey(String primaryKey)
        throws RemoteException, FinderException;
}
// CountBean.java  -- Count CMP Entity Example
package count.entity20;

import javax.ejb.*;

public abstract class CountCmpEJB implements EntityBean
{
    public abstract String getCounterName();
    public abstract void setCounterName(String name);
    public abstract int getSum();
    public abstract void setSum(int value);
    protected EntityContext entityContext;

    // callback methods
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbRemove() throws RemoveException {}
    public void setEntityContext(EntityContext ctx) {entityContext = ctx;}
    public void unsetEntityContext() {entityContext = null;}

    // Home interface methods

    public String ejbCreate(String name, int sum) throws CreateException
    {
        // Set the initial data on the instance
        setCounterName(name);
        setSum(sum);
        return getCounterName();
    }

    public void ejbPostCreate(String name, int sum) throws CreateException
    {
    }
}
// Business methods

// get sum
public int getCurrentSum()
{ return getSum();
}

// set sum
public void setCurrentSum(int val)
{ setSum(val);
}

public int increment()
{ int i = getSum();
   i++;
   setSum(i);
   return i;
}
}
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE j2ee-ri-specific-information PUBLIC '-//Sun Microsystems Inc.//DTD J2EE Reference Implementation 1.3//EN' 'http://localhost:8000/sun-j2ee-ri_1_3.dtd'>
<j2ee-ri-specific-information>
<server-name></server-name>
<rolemapping />
<enterprise-beans>
<module-name>ejb-jar-ic.jar</module-name>
<unique-id>0</unique-id>
<ejb>
<ejb-name>CountCmpEJB</ejb-name>
<jndi-name>CountCMP20</jndi-name>
<ior-security-config>
<transport-config>
<integrity>supported</integrity>
<confidentiality>supported</confidentiality>
<establish-trust-in-target>supported</establish-trust-in-target>
<establish-trust-in-client>supported</establish-trust-in-client>
</transport-config>
<as-context>
<auth-method>username_password</auth-method>
<realm>default</realm>
</as-context>
</ior-security-config>
</ejb>
</enterprise-beans>
</j2ee-ri-specific-information>
<sql-statement>
<operation>deleteRow</operation>
<sql>DELETE FROM "CountCmpEJBTable" WHERE "counterName" = ? </sql>
</sql-statement>
<sql-statement>
<operation>deleteTable</operation>
<sql>DROP TABLE "CountCmpEJBTable"</sql>
</sql-statement>
<sql-statement>
<operation>findByPrimaryKey</operation>
<sql>SELECT "counterName" FROM "CountCmpEJBTable" WHERE "counterName" = ? </sql>
</sql-statement>
<sql-statement>
<operation>createTable</operation>
<sql>CREATE TABLE "CountCmpEJBTable" ("counterName" VARCHAR(255) , "sum" INTEGER NOT NULL, CONSTRAINT "pk_CountCmpEJBTable" PRIMARY KEY ("counterName") )</sql>
</sql-statement>
<create-table-deploy>true</create-table-deploy>
<delete-table-undeploy>true</delete-table-undeploy>
</ejb20-cmp>
</cmpresource>
<ds-jndi-name>jdbc/Cloudscape</ds-jndi-name>
<default-resource-principal>
<name>guest</name>
<password>guest123</password>
</default-resource-principal>
</cmpresource>
</enterprise-beans>
</j2ee-ri-specific-information>
Local Interfaces

- Local interfaces allow optimized calls from a client to an EJB within a single VM
- Data passed by reference
  - Be careful not to assign state of one EJB to another
- References are not usable outside of current call chain

Local/Remote Design Trade-offs

- Remote interface advantages
  - Location independence
  - Loose coupling between client and bean
  - Flexibility in distribution of components
  - Isolation of components: call by value
- Remote interface costs
  - More expensive
  - Under the cover APIs
  - Stubs/Skeletons
  - Remote Exceptions
Local Interfaces

Local Home Interfaces

- Similar to Home Interfaces except only local references are returned
  - create returns local reference
  - find returns local reference or collection of local references

```java
package team;
import java.util.*;
import javax.ejb.*;
public interface LocalPlayerHome extends EJBLocalHome
{
  public LocalPlayer create (String id, String name,
      String position, double salary)
      throws CreateException;
  public LocalPlayer findByPrimaryKey (String id)
      throws FinderException;
  public Collection findByPosition(String position)
      throws FinderException;
  public Collection findByLeague(LocalLeague league)
      throws FinderException;
  ...
}
```
Local Interfaces: The LocalPlayer Interface

package team;
import java.util.*;
import javax.ejb.*;
public interface LocalPlayer extends EJBLocalObject
{
    public String getPlayerId();
    public String getName();
    public String getPosition();
    public double getSalary();
    public Collection getTeams();
    public Collection getLeagues() throws FinderException;
    public Collection getSports() throws FinderException;
}

Container-Managed Relationships

Like Foreign Key Relationships in SQL Tables, Entity Beans can have relationships
Implemented by Container for CMP
Relationships can be:
- One-to-one, Many-to-one, One-to-many, any-to-many
- Unidirectional or Bidirectional
If an entity bean is the target of a relationship, it must have local interfaces
Select Methods

A select method is similar to a finder method:
- A select method queries a database and returns objects
- The deployment descriptor specifies an EJB QL query for a select method
- The entity bean class does not implement the select method

Select Methods (continued)

A select method differs from a finder method:
- Returns persistent fields or the home interfaces of related entity beans.
  - A finder method can return only the home interface (or a collection of home interfaces) that defines it
  - Is not exposed in the local or remote interfaces
- Is not exposed in the local or remote interfaces
  - A select method cannot be invoked by a client. It may be invoked only by the methods implemented within the entity bean class. A select method is usually invoked by a business method.
- A select method is defined in the entity bean class
**EJB QL**

- Defines queries for the finder/select methods of CMP entity beans
- Subset of SQL92
  - Has extensions that allow navigation over relationships in an entity bean’s abstract schema
- The scope of an EJB QL query is the abstract schema of related entity beans that are packaged in the same EJB JAR file
- EJB QL queries are defined in the DD
  - Container translates EJB QL into query language of underlying datastore making EJBs portable

---

**Message Driven Beans**

- Combines features of a session bean and JMS
  - Allows a business component to receive an asynchronous message
  - Similar to a stateless session bean
- The onMessage method is called when a message arrives
A Message-Driven Bean Example

Implements the MessageDrivenBean and MessageListener interfaces

Class is defined as public

Class cannot be defined as abstract or final

Implements one onMessage method

Implements one ejbCreate method and one ejbRemove method

Contains a public constructor with no arguments

Must not define the finalize method

Does not have a home or remote interface
The Message-Driven Bean Life Cycle

1. setMessageDrivenContext
2. ejbCreate

Does Not Exist

onMessage

Ready

ejbRemove

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.MessageDrivenBean;
import javax.ejb.MessageDrivenContext;
import javax.naming.*;
import javax.jms.*;

public class SimpleMessageBean implements MessageDrivenBean, MessageListener {
    private transient MessageDrivenContext mdc = null;
    private Context context;

    public SimpleMessageBean() {
        System.out.println("In SimpleMessageBean.SimpleMessageBean()");
    }

    public void setMessageDrivenContext(MessageDrivenContext mdc) {
        System.out.println("In " + "SimpleMessageBean.setMessageDrivenContext()");
        this.mdc = mdc;
    }
}
Message-Driven EJB (2 of 3)

```java
public void onMessage(Message inMessage) {
    TextMessage msg = null;
    try {
        if (inMessage instanceof TextMessage) {
            msg = (TextMessage) inMessage;
            System.out.println("MESSAGE BEAN: Message received: "
                               + msg.getText());
        } else {
            System.out.println("Message of wrong type: "
                               + inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        e.printStackTrace();
        mdc.setRollbackOnly();
    } catch (Throwable te) {
        te.printStackTrace();
    }
} // onMessage
```

Message-Driven EJB (3 of 3)

```java
public void ejbCreate() {
    System.out.println("In SimpleMessageBean.ejbCreate()");
}

double price = 3.0;
```

```java
public void ejbRemove() {
    System.out.println("In SimpleMessageBean.remove()");
}
} // class
```
import javax.jms.*;
import javax.naming.*;

public class SimpleMessageClient {

    public static void main(String[] args) {

        Context jndiContext = null;
        QueueConnectionFactory queueConnectionFactory = null;
        QueueConnection queueConnection = null;
        QueueSession queueSession = null;
        Queue queue = null;
        QueueSender queueSender = null;
        TextMessage message = null;
        final int NUM_MSGS = 3;

        try {
            jndiContext = new InitialContext();
        } catch (NamingException e) {
            System.out.println("Could not create JNDI " +
                    "context: " + e.toString());
            System.exit(1);
        }

        // locate queue
        try {
            queueConnectionFactory = (QueueConnectionFactory)
                    jndiContext.lookup("java:comp/env/jms/MyQueueConnectionFactory");
            queue = (Queue)
                    jndiContext.lookup("java:comp/env/jms/QueueName");
        } catch (NamingException e) {
            System.out.println("JNDI lookup failed: " + e.toString());
            System.exit(1);
        }

        // create connection, send messages
        try {
            queueConnection = 
                    queueConnectionFactory.createQueueConnection();
            queueSession = 
                    queueConnection.createQueueSession(false,
                    Session.AUTO_ACKNOWLEDGE);
            queueSender = queueSession.createSender(queue);
            message = queueSession.createTextMessage();
        }
    }
}
for (int i = 0; i < NUM_MSGS; i++) {
    message.setText("This is message " + (i + 1));
    System.out.println("Sending message: " +
                        message.getText());
    queueSender.send(message);
}

} catch (JMSException e) {
    System.out.println("Exception occurred: " + e.toString());
} finally {
    if (queueConnection != null) {
        try {
            queueConnection.close();
        } catch (JMSException e) {}
    } // if
    System.exit(0);
} // finally
} // main
} // class