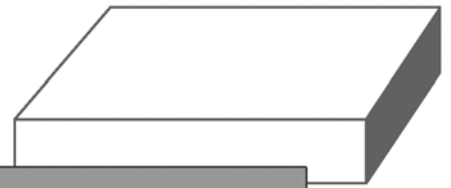
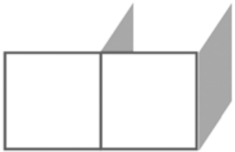
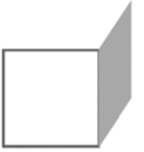
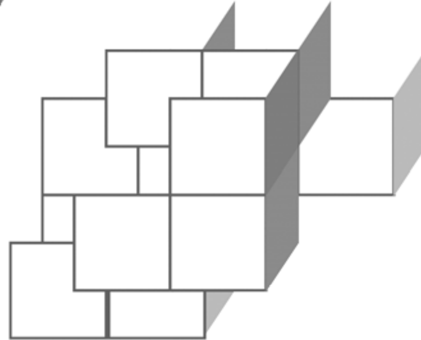


Chapter 4

Transactions



OBJECTIVES

After completing "Transactions," you will be able to:

- **Describe the importance of transactions in enterprise-class software.**
- **Describe the EJB transaction model.**
- **Protect your bean code from impingements on its atomicity, consistency, isolation and durability by declaring transaction attributes for bean methods.**
- **Define transaction boundaries at other than method scope programmatically.**

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Transactions – Who Needs 'Em?

- **The concept of a transaction provides a model for changes to an application's data.**
- **Various components, by participating in transactions, guarantee that their changes will be:**
  - **Atomic –** work cannot be interrupted meaningfully.
  - **Consistent** – work cannot fail in such a way as to leave the persistent data on which they're working in an inconsistent state. That is, work either proceeds to completion and is **committed**, or it fails **completely**, thanks to a transaction **rollback**.
  - **Isolated** – other readers and writers of persistent data will share information and changes according to application-defined standards, known as **isolation levels**.
  - **Durable** – changes made by the code are guaranteed to either fail completely or get into the persistent store, even in the case of server failure, network failure, etc., after the transaction completes.
- **Without these critical transaction features – known by their first letters as the ACID properties – bean providers would have a terrible time writing truly robust code.**

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Declarative *vs.* Programmatic

- **EJB requires the container to manage transaction contexts, and to interact with the resource managers which can effect commit or rollback, ensure isolation, etc. These are never exposed directly to bean code.**
- **Only a flat transaction model is supported in the EJB specification.**
  - Other models exist in the world, but for portable EJB code only flat transactions can be assumed.
- **An EJB can assume some control over the demarcation of transaction boundaries.**
  - Declarative control is established, as usual, in the deployment descriptor, in this case by specifying **transaction attributes**.
  - Programmatic control can be assumed through the entity context and the **Java Transactions API**, or **JTA**.
  - Only session beans can assume programmatic control of transactions.
  - There are a few other restrictions on what actions a stateless session bean can take in starting and completing transactions – we will not delve into these too deeply.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Transaction Attributes

- **Each bean must declare a transaction attribute which dictates how the container will enlist the bean's methods in transactions as they are invoked.**
  - Based on these attributes, the container may also create transactions in order to enlist a bean, or may require that the caller provide a transaction context.
  - The bean may define transaction attributes per method as well, but at least an attribute at bean scope is required.
- **Possible attributes:**
  - **TX_NOT_SUPPORTED** – any existing transaction will be suspended for the method call and resumed afterwards.
  - **TX_SUPPORTS** – will be enlisted in an existing transaction, but will run without a transaction as well.
  - **TX_REQUIRED** – requires a transaction: the container will start one if it is not already present.
  - **TX_REQUIRES_NEW** – like the required attribute, but the container will suspend any existing transaction before starting a new one.
  - **TX_MANDATORY** – requires a transaction to be provided by the caller. Fails otherwise.
  - **TX_NEVER** – forbids a transaction to be provided by the caller. Fails otherwise.
  - **TX_BEAN_MANAGED** – this bean handles transactions programmatically.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Possible Attributes for EJBs

- **The full range of transaction attributes is available for declaration on a session bean.**
- **Entity beans cannot manage their own transactions.**
  - **TX_BEAN_MANAGED** is not an option.
  - Also, entities supporting the 2.x CMP contract must declare either **TX_REQUIRED**, **TX_REQUIRES_NEW**, or **TX_MANDATORY** as the transaction attribute for any method.
- **Message-driven beans can manage their own transactions, but are never called directly by a client.**
  - Thus several transaction attributes are meaningless for message-driven beans.
  - In addition to **TX_BEAN_MANAGED**, message-driven beans can declare either **TX_REQUIRED** or **TX_NOT_SUPPORTED** for their **onMessage** methods.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Isolation Levels

- **As of the EJB 1.1 specification, there can be no portable, declarative control of isolation levels.**
- **Bean-managed transactions can specify isolation levels via JDBC.**
- **An EJB container may provide a proprietary declarative means of isolation level control.**
- **JDBC-defined isolation levels are mirrored in the EJB specification, with one exception. Each of the following assumes the guarantees of the previous ones:**
  - **TRANSACTION_NONE** – this JDBC level is irrelevant to EJB, since a method can simply say that it doesn't support transactions at all.
  - **TRANSACTION_READ_UNCOMMITTED** – if one transaction writes a value but has yet to commit it, another transaction might read the uncommitted (cached) value.
  - **TRANSACTION_READ_COMMITTED** – only committed writes are available to other transactions.
  - **TRANSACTION_REPEATABLE_READ** – concurrent reads are allowed, but once a datum is read in any transaction, it is guaranteed to hold its value until the end of the transaction.
  - **TRANSACTION_SERIALIZABLE** – total isolation of transactions: multiple transactions are not allowed to run concurrently.

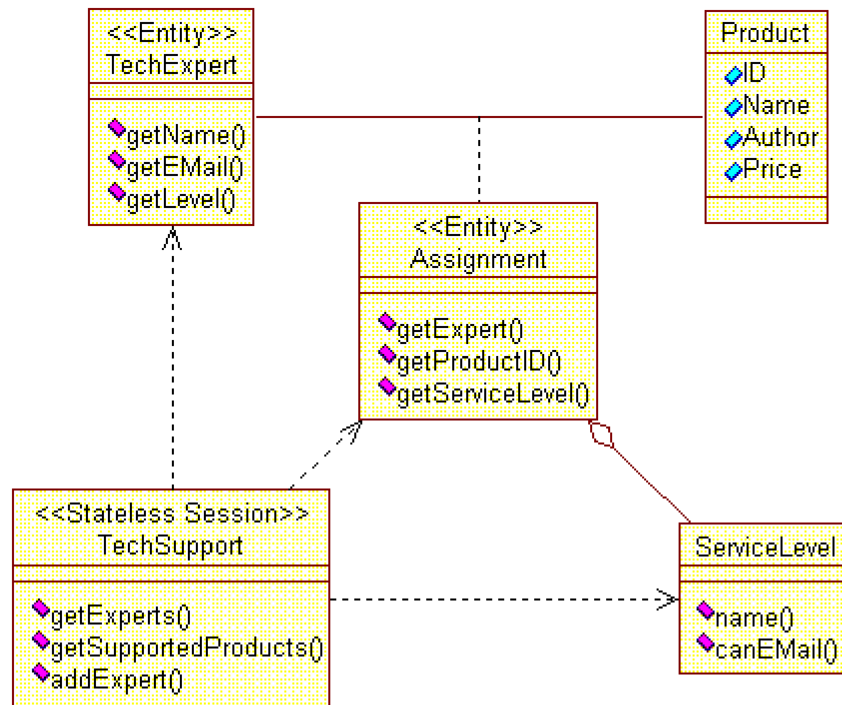
Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Shareware Support

DEMO

- We will work through a simple demonstration of declarative transaction control using an example EJB/web application **Shareware.**
- This is found in **Demos\Transactions.**



- On the session bean, the method **addExpert** assists in building the database.
  - It creates a **TechExpert** bean.
  - It creates an **Assignment** bean to link the expert to a single product by its ID.
- The **AssignmentHome.create** method can also be called to create additional assignments.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Shareware Support

DEMO

- The **Assignment** bean manages a many-to-many link between **TechExpert** objects and **Product**s.
  - For our purposes, we declare that the **PRODUCT** table existed prior to the use of EJB in this system.
  - This will force us to adjust, using transaction attributes, for problems that ordinarily would be handled by container-managed persistence.
- The **PRODUCT** table will be primed via a SQL script to simulate this situation:

```
INSERT INTO PRODUCT (ID, NAME, AUTHOR, PRICE)
  VALUES (1, 'TextPad', 'Helios Software', 28.99);

INSERT INTO PRODUCT (ID, NAME, AUTHOR, PRICE)
  VALUES (2, 'FTP Voyager', 'deerfield.com', 40);

INSERT INTO PRODUCT (ID, NAME, AUTHOR, PRICE)
  VALUES (3, 'FullShot', 'Inbit', 29.99);

INSERT INTO PRODUCT (ID, NAME, AUTHOR,  PRICE)
  VALUES (4, 'The Greatest Game',
    'Bob Nielden', 50);

INSERT INTO PRODUCT (ID, NAME, AUTHOR, PRICE)
  VALUES (5, 'Typing Teacher',
    'Matiltha Harwitch', 25);
```

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Shareware Support

DEMO

1. From **Demos\Transactions\Step1**, start by creating and configuring the new **Shareware** database:

```
asant init-DB
Failed to execute: DROP TABLE PRODUCT ...
7 of 8 SQL statements executed successfully
  (Again, the failure to DROP TABLE is normal.)
```

2. Review the code for methods **clean** and **prime** in the **TechSupport** bean. **prime** builds a new set of experts with assignments to products; **clean** empties the database out. The **SharewareClient** application creates a bean and calls these methods in sequence to populate the database.

```
public String prime ()
  throws RemoteException
{
  ...
  TechSupport myEJB = (TechSupport)
    context.getEJBObject ();
  try
  {
    ...
    myEJB.addExpert (1, "Brian Noonan",
      "noonan@goalsoft.com", "", 1,
      ServiceLevel.get (ServiceLevel.EMAIL_DAY));
    myEJB.addExpert (2, "Sergei Gonchar",
      "sergei@softstick.com", "412-456-4564", 2,
      ServiceLevel.get (ServiceLevel.PHONE));
    assignmentHome.create (3, "Sergei Gonchar", 3,
      ServiceLevel.get (ServiceLevel.PHONE));
    ...
  }
  ...
}
```

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Shareware Support

DEMO

3. Note that the **addExpert** method is a convenient way to perform two closely related steps: create a new expert, and create the first assignment for that expert.

```
public void addExpert (int assignmentID,
    String name, String email, String phone,
    int productID, ServiceLevel serviceLevel)
  throws CreateException, RemoteException
{
  try
  {
    expertHome.create (name, email, phone);
    assignmentHome.create (assignmentID, name,
      productID, serviceLevel);
  }
  catch (NullPointerException ex) {}
}
```

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

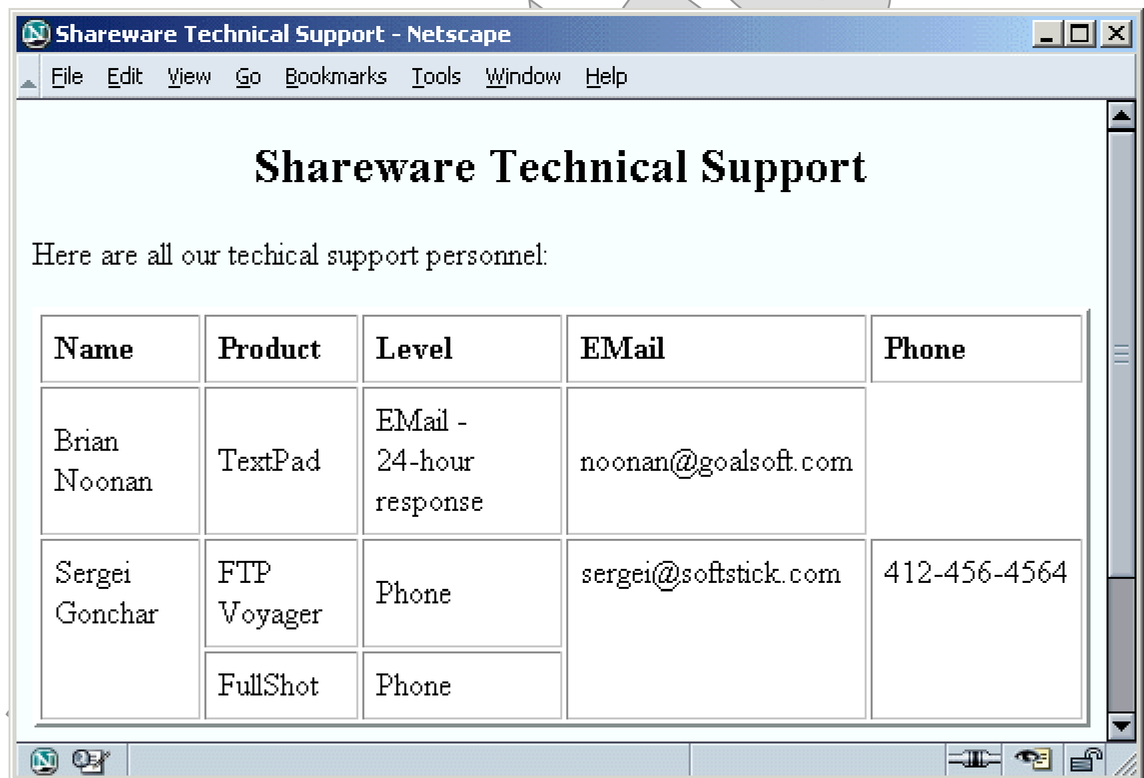
Shareware Support

DEMO

4. Build and deploy with **asant**, which concludes with invocation of the **populate** target, which in turn runs **clean** and **prime**.

5. Navigate to the following URL to review the support database:

```
http://LOCALHOST:8000/Shareware/index.jsp
```



© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Shareware Support

DEMO

6. Now switch to **Demos\Transactions\Step2**, in which the **prime** method has the following additional code:

```
myEJB.addExpert (1, "Brian Noonan",
  "noonan@goalsoft.com", "", 1,
    ServiceLevel.get (ServiceLevel.EMAIL_DAY));
myEJB.addExpert (2, "Sergei Gonchar",
  "sergei@softstick.com", "412-456-4564", 2,
    ServiceLevel.get (ServiceLevel.PHONE));
assignmentHome.create (3, "Sergei Gonchar", 3,
    ServiceLevel.get (ServiceLevel.PHONE));

myEJB.addExpert (4, "Shayne Corson",
  "bighit@canadasoft.com", "", 100,
    ServiceLevel.get (ServiceLevel.EMAIL_WEEK));
```

7. Note that there is no product ID of 100 in the database. What do you suppose will happen?

8. Run **asant** on this version to build, deploy, and populate. You should see output along the following lines:

```
Removing Assignments ...
Removing TechExperts ...

Building database ...
Exception - aborting.
```

… and a look at the server log file should show the root of the problem. This is entirely correct, and as far as it goes it is even desirable. The new code is wrong, so an exception is an appropriate response.

```
javax.ejbCreateException: FK constraint violated -
  product ID not found in PRODUCT table.
```

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Evaluation Only

Shareware Support

DEMO

9. Reload the JSP and see what resulted: the foreign-key constraint was violated when **addExpert** tried to create the assignment to a non-existent product. Unfortunately, the expert had already been created, and so we have poor Shayne Corson volunteering to do nothing:

Shareware Technical Support - Netscape

Shareware Technical Support

Here are all our techical support personnel:

Name	Product	Level	EMail	Phone
Brian Noonan	TextPad	EMail - 24-hour response	noonan@goalsoft.com	
Sergei Gonchar	FTP Voyager	Phone	sergei@softstick.com	412-456-4564
	FullShot	Phone		
Shayne Corson			bighit@canadasoft.com	

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

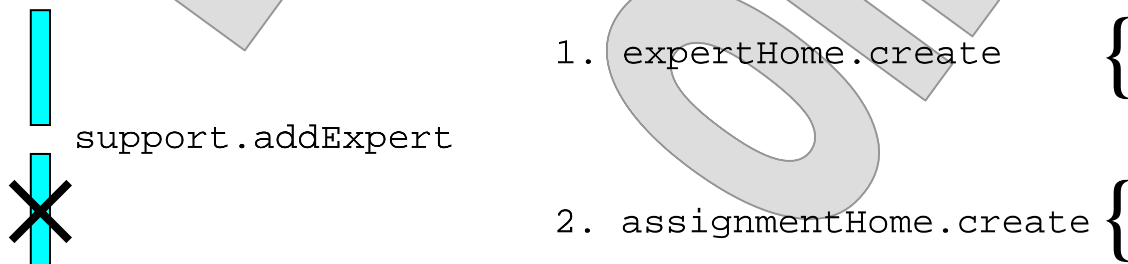
Shareware Support

DEMO

10. Again, the **prime** code is wrong, so we can't make everything perfect. What we want is a more robust response that protects data integrity. How to fix this? Look at the **ejb-jar.xml** for the clue: the transaction attribute for **addExpert** is set to **Supports**.

```
<container-transaction>
  <method>
    <ejb-name>TechSupport</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>addExpert</method-name>
    <method-params>...</method-params>
  </method>
  <trans-attribute>Supports</trans-attribute>
</container-transaction>
```

11. This means that since no transaction was in force at the time of the method call, none was created for the method. A transaction was created for each of the entity-bean calls, which require them (see elsewhere in the same file), but the two transactions were separated, not governed by the session bean, and hence **isolated** from each other. The rollback that kicked in on the violated constraint did not touch the creation of the expert, which had already been committed.



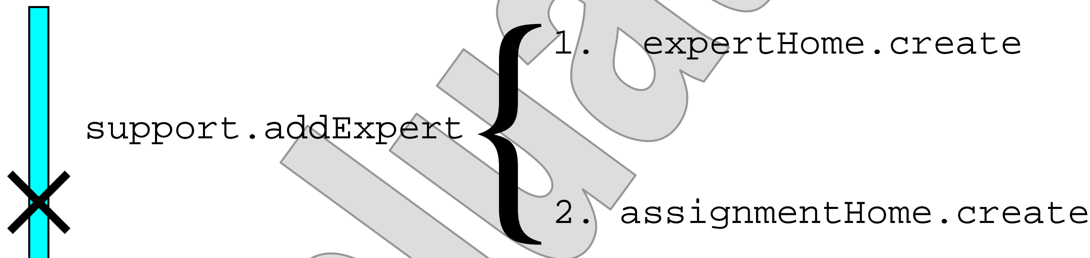
© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Shareware Support

DEMO

12. Change the attribute to **Required** and redeploy the server.

13. Run the misbehaving client again, and you will still see the FK violation propagated back to the client. However, if you reload the JSP, you should see that the hapless Mr. Corson has indeed been pulled back out of the system until he has something to do. (The **Step3** version has this change already made and is the final answer step for Shareware.)



14. Note that a tempting alternative would be to require a transaction on the **prime** method. This is a common enough pattern, but in this case a closer examination of the method's behavior reveals that a finer-grained transaction policy will yield better results. If **prime** were running in a transaction context and this same error were to occur, the result would be rollback of the entire prime process, leaving an empty database. This approach also preserves data integrity, but at a much higher cost.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Programmatic Control

- A bean can query its EJB context for the transaction context, using **getUserTransaction**.
- Using this interface the bean can **begin**, **commit** or **rollback** transactions.

```
public interface UserTransaction
{
    public void begin ()
        throws NotSupportedException,
            SystemException;

    public void commit ()
        throws RollbackException,
            HeuristicMixedException,
            HeuristicRollbackException,
            SecurityException,
            IllegalStateException, SystemException;

    public int getStatus ()
        throws SystemException;

    public void rollback ()
        throws IllegalStateException,
            SecurityException, SystemException;

    public void setRollbackOnly ()
        throws IllegalStateException,
            SystemException;

    public void setTransactionTimeout (int)
        throws SystemException;
}
```

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Managing Transactions

- **With the UserTransaction and related interfaces it is possible to:**
  - Define **multiple transactions** within a single method
  - Define **transaction boundaries** to span method invocations (stateful beans only)
  - Generally, take **complete control** over transaction behavior
- **Transactions can be unwieldy beasts, especially when they must span method invocations.**
  - As you can see, the **UserTransaction** method signatures are laden with exceptions.
  - These must be caught and handled in your EJB methods.
  - A few can be passed along in some cases, such as the security exception.
  - Once an EJB begins a transaction, it cannot be enlisted in any other transactions. This increases the burden of state management on a stateful session bean: it must be aggressive about throwing **IllegalStateException**s if methods are called out of their natural order.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Dealing with Failure

LAB 4

In this lab you will add a bean-managed transaction to the Electronic DJ system. The new concept in the design is that playback of an album might fail over the network or on the client's machine. So where it used to be considered an automatic incrementing of the **spins** count, playback now must be monitored to some extent. You will implement a long-running transaction (that is, spanning multiple user-triggered calls) to correctly govern the playback count and other durable data.

Detailed instructions are contained in the Lab 4 write-up at the end of the chapter.

Suggested time: 60 minutes.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

SessionSynchronization Interface

- **Short of full programmatic control of transactions, a session bean might want to allow for declarative control, but to be advised of transaction progress.**
- **This is important especially for stateful beans that need to assure that their conversational state is in sync with the durable data.**
  - Stateless beans don't have this issue, obviously.
  - Entity beans have a similar issue but there is already a load/store mechanism in place to assure complete synchronization.

```
public interface javax.ejb.SessionSynchronization
{
    void afterBegin ();
    void afterCompletion (boolean);
    void beforeCompletion ();
}
```

- **Implement the SessionSynchronization interface, and specifically the afterCompletion method, to catch up on any failures in commission of the desired changes.**
- **Session state can thus be adjusted to allow for this failure.**

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Session Synchronization

EXAMPLE

- Consider the code in **Examples\ElectronicDJ\Step8**, specifically the **MyDJ** bean. (This is the answer code set for the lab just completed.)
- The method **audition** must set the conversational state element **nowPlaying** and then play the requested album.
- What if there is a problem in playing the record?
  - In the previous lab, you implemented code to assure that the **spins** counter was reset in the case of playback failure.
  - This is the correct approach for durable data.
  - For conversational state, which would not be enlisted in any transaction, it's not possible to rely on rollback by a resource manager.
  - Instead, **MyDJEJB** could implement the **SessionSynchronization** interface, and handle rollback notification by resetting **nowPlaying** to **null.**

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

DTC, 2PC ... EJB!

- **A distributed transaction is one that spans multiple resource managers, which are unused to coordinating their activities.**
  - Any resource manager must coordinate multiple changes to some durable data – say, two tables in a single database.
  - It does this by its own, usually proprietary, code.
  - When multiple managers are involved, they each act on their own managed resources, but must work with each other to assure consistency across those resources.
  - This requires a protocol by which the resource managers effect a **two-phase commit (2PC)**: first all managers are polled to assure that the changes can be committed, and then they are all told to proceed. Each manager must lock the relevant information for the time spanning these phases.
  - The master resource manager, if you will, is the **distributed transaction coordinator (DTC)**.
- **EJB 2.0 <u>specifies</u>, <u>but</u> <u>does</u> <u>not</u> <u>require</u>, support for distributed transaction control.**
  - EJB containers may choose to participate in a standardized means of coordinating transactions, and if they do there are certain protocols they must implement.
  - They may opt out, however, in which case they are required only to say so (via CORBA transaction context information) and to throw certain CORBA exceptions when a distributed transaction attempts to use their resources.

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Evaluation Only

SUMMARY

- **Transactions have long played a critical role in large-scale software.**
- **EJB's declarative model for transaction control is still somewhat immature, owing to the great variations that still exist across the industry in standards and practices.**
  - The absence of any portable isolation-level attribute on EJBs exemplifies this: there simply was not enough industry consensus to support a single portable standard.
- **Still, the transaction attribute, all by itself, offers a good deal of control over transactional behavior, with minimal development effort.**
- **Programmatic control over transactions is of course even more powerful, but the jump in effort level is great.**
- **In the next chapter we will consider some guidelines for the best choices in transaction design for EJB systems.**

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

Dealing with Failure

LAB 4

Introduction

In this lab you will add a bean-managed transaction to the Electronic DJ system. The new concept in the design is that playback of an album might fail over the network or on the client's machine. So where it used to be considered an automatic incrementing of the spins count, playback now must be monitored to some extent. You will implement a long-running transaction (that is, spanning multiple user-triggered calls) to correctly govern the playback count and other durable data.

Suggested Time: 60 minutes

Root Directory:	**Capstone\EJBEffective**	
Directories:	**Labs\Lab4**	(do your work here)
	Examples\ElectronicDJ\Step7	(backup copy of starter files)
	Examples\ElectronicDJ\Step8	(answer)
Files:	**cc\music\MyDJEJB.java**	
	Website\MyDJ\MyDJ.jsp	
	Website\MyDJ\AuditionOne.jsp	
	Website\MyDJ\PlaybackFailed.jsp	
Packages:	**cc.music**	

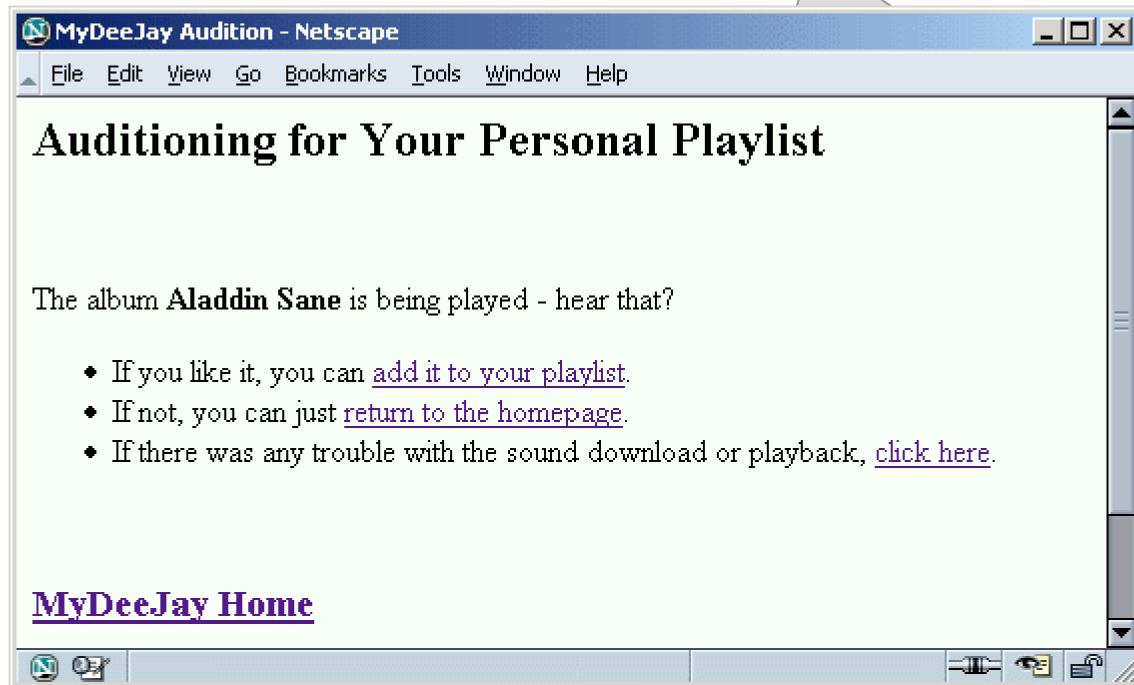
Instructions

1. Compile and deploy the server. Rebuild the database as usual by browsing to **BuildDatabase.jsp**, logging in as the AS8 administrator.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

2. Restart your browser and visit the **MyDJ.jsp** as your adult user from the previous lab. Try auditioning a song and you will see the major change to the website: when you've chosen a song, you'll see a new version of the **AuditionOne.jsp** page that gives you three options:



3. Don't bother to proceed from here, since some of the EJB calls that the next pages make are not yet implemented – that's going to be your job! Close the browser.

Evaluation Only

4. Review the calls made from **AuditionOne.jsp**, **MyDJ.jsp**, and **PlaybackFailed.jsp**:

```
From AuditionOne.jsp:
<%
    MyDJ sessionBean = (MyDJ) session.getValue ("MyDJ");
    String title = request.getParameter ("Title");
    try
    {
        sessionBean.audition (title);
%>

From MyDJ.jsp:
        if (command.equals ("add"))
        {
            sessionBean.addToPlaylist ();
            out.println ("Added " +
                sessionBean.getNowPlaying ().getTitle () +
                " to your playlist.<br><br>");
        }

From PlaybackFailed.jsp:
<%
    MyDJ sessionBean = (MyDJ) session.getValue ("MyDJ");
    sessionBean.playbackFailed ();
%>
```

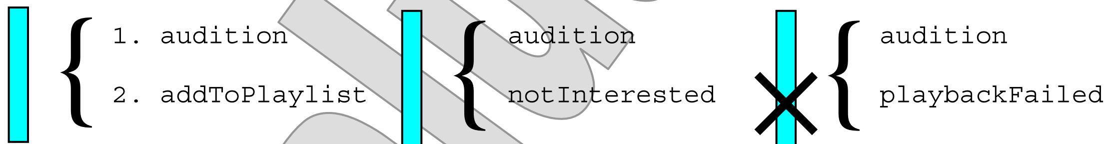
5. Add the following to the **MyDJ.jsp** code, right after the processing of the **add** command:

```
        if (command.equals ("dontadd"))
         {
            sessionBean.notInterested ();
            out.println ("Okay, well, never mind then.<br><br>");
        }
```

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

6. Now you have the following flow control from the **AuditionOne** page, after the **audition** method has been called. For each, consider the desired outcome on the durable data, where step one was an increment of **spins** on the record and step two is a possible addition to the listener's playlist:
   - If the user likes the record, the **add** command will result in a call to **addToPlaylist**. In this case, steps one and two should both be committed.
   - If the user doesn't like the record, the **dontadd** command will result in a call to **notInterested**. (This was not necessary before, but because of the possibility of playback failure, we now want some confirmation from the user that the record was at least heard.) In this case, step one can be committed and step two will not be taken.
   - If there was a problem with the sound playback – failure to download, failure to play, whatever – the user can click the link to **PlaybackFailed.jsp**, resulting in a call to **playbackFailed**. Here, step one (which has already been taken) cannot be committed – the count should be **rolled back** to its previous value.
7. Thus there are three possible outcomes after the call to **audition**, and three methods on the stateful session bean by which to effect these outcomes.

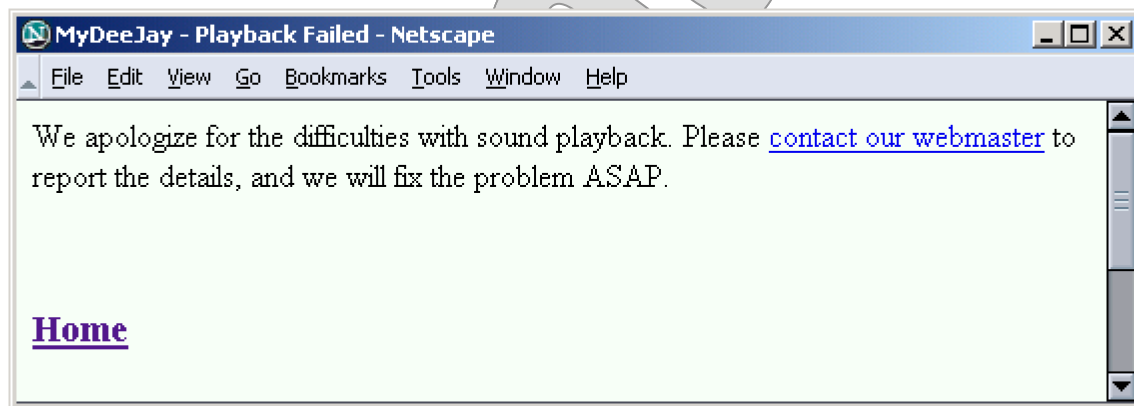


8. Open the **ejb-jar.xml** file. Find the declaration of **transaction-type** for the **MyDJ** bean and change it from **Container** to **Bean**. At the bottom of the file, you can remove the **container-transaction** declaration for the **MyDJ** bean – be sure to leave the declarations for all the other beans, however. Save the file.
9. Open the **MyDJ** source file. Add the two methods **notInterested** and **playbackFailed**. Like **addToPlaylist**, each of these takes no parameters and throws the **RemoteException** and the **NothingPlaying** state exception. Save the file.
10. Open the **MyDJEJB** source file. In the **audition** method, start a new transaction using **context.getUserTransaction ().begin ()**. Note that you will have to catch exceptions here – just return from the method if there is any failure at this point.
11. In **addToPlaylist**, after making the call on the **Listener** bean, **commit** the transaction. Here, catch exceptions with an empty code block, effectively throwing the exception away and proceeding. (No, this is not good practice! but you have enough on your plate at the moment.)
12. Now create the implementation of **notInterested**. This method is almost identical to **addToPlaylist** – you can start by copying the implementation of that method – but there is no call to add to the listener's playlist. Still, the transaction must be completed, and **commit** is correct here because work has already been done in the **audition** method that must be made durable.

Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.

13. Create the implementation of **playbackFailed**. In turn, this method is nearly identical to **notInterested**. The only difference is that here you will **rollback** the transaction. This allows you to undo what was done in **audition**, since the user didn't actually hear the record: the **spins** count will not be incremented in the durable data. The increment has already been performed, of course, but thanks to the long-running transaction it only exists in a cache, which will not be flushed to the database thanks to this rollback.
14. Compile the server and fix any code errors.
15. Deploy the new application and rebuild the database.
16. Now visit **MyDJ.jsp** again, and audition a new song. Try each of the possible paths in sequence, and each time you return to **MyDJ.jsp**, and then **AuditionOne.jsp**, you should be able to confirm that the correct outcome was obtained: added to the personal playlist only on **addToPlaylist**, and incremented the play count unless playback failed.



Evaluation Only

© 2000-2006 Will Provost.
All Rights Reserved by Capstone Courseware, LLC.