Module 7
Implementing EJB 3.0
Session Beans

Objectives

► Compare stateless and stateful behavior
► Describe the operational characteristics of a stateless session bean
► Describe the operational characteristics of a stateful session bean
► Create session beans
► Package and deploy session beans
► Create a session bean client
Implementing Session Beans

Session beans can be stateless or stateful. The statefulness of a bean depends on the type of business function it performs:

► In a stateless client-service interaction, no client-specific information is maintained beyond the duration of a single method invocation.
► Stateful services require that information obtained during one method invocation be available during subsequent method calls:
  • Shopping carts
  • Multi-page data entry
  • Online banking

Comparison of Stateless and Stateful Behavior

Stateless Session Bean Operational Characteristics

With stateless session beans:

► The bean does not retain client-specific information.
► A client might not get the same session bean instance.
► Any number of client requests can be handled by the same session bean instance. This has a profound impact on performance.
Stateless Session Bean Cardinality

![Stateless Session Bean Diagram]

Stateful Session Bean Operational Characteristics

With stateful session beans:

▸ The bean belongs to a particular client for an entire conversation.

▸ The client connection exists until the client removes the bean or the session times out.

▸ The container maintains a separate EJB object and EJB instance for each client.
Creating Session Beans

- Declare a business interface for the session bean.
- Create the session bean class that implements the business interface.
- Configure the session bean by either annotating the session bean class or providing a deployment descriptor (DD).
A session EJB component’s business interface has the following characteristics:

- Outlines the EJB component’s business methods.
- Declares the Session as a distributed component (optional)

```java
import javax.ejb.*;

@Remote
public interface HelloSession {
    public java.lang.String hello();
}
```

Creating the Session Bean Class That Implements the Business Interface

A session EJB component’s class has the following characteristics:

- Often referred to as the bean class, implementation instance, or EJB class
- Implements the component’s business interface
- Contains code that implements bean life-cycle methods (optional)
Creating the Session Bean Class That Implements the Business Interface

```java
import javax.ejb.*;

@Stateless
public class HelloSessionBean implements HelloSession {
    public java.lang.String hello() {
        return "Hello World!";
    }
}
```

Declaring a Local Business Interface for a Session Bean

How can a business interface be designated a local interface?

```java
@Local
public interface Calculator {...}
@Stateless
public class CalculatorBean implements Calculator {...}

public interface Calculator {...}
@Local @Stateless
public class CalculatorBean implements Calculator {...}

public interface Calculator {...}
@Stateless
public class CalculatorBean implements Calculator {...}
```
Declaring a Remote Business Interface for a Session Bean

How can a business interface be designated a remote interface?

```java
public interface Calculator {...}
@Remote @Stateless
public class CalculatorBean implements Calculator {...}

@Remote
public interface Calculator {...}
@Stateless
public class CalculatorBean implements Calculator {...}
```

► Are there any additional requirements for remote interfaces?
► Can a session bean have more than one business interface?

Requirements for a Session Bean Class

► The class must be a top-level class. In addition, the class must be defined as public, must not be final, and must not be abstract.
► The class must have a public constructor that takes no parameters. The container uses this constructor to create instances of the session bean class.
► The class must not define the `finalize` method.
► The class must implement the methods of the bean’s business interface(s), if any.
Annotating the Session Bean Class

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Applies to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateful</td>
<td>Session bean class</td>
</tr>
<tr>
<td>Stateless</td>
<td>Session bean class</td>
</tr>
<tr>
<td>Local</td>
<td>Session bean class or business interface</td>
</tr>
<tr>
<td>Remote</td>
<td>Stateful session bean class</td>
</tr>
<tr>
<td>Remove</td>
<td>Stateful session bean class</td>
</tr>
</tbody>
</table>

Other common annotations:
- Transaction-related annotations
- Security-related annotations
- Life-cycle callback handler method annotations
- Resource-related annotations

Life Cycle of a Stateless Session Bean

- PostConstruct callback
- PreDestroy callback
Life Cycle of a Stateful Session Bean

Implementing Session Beans

Defining Life-Cycle Event Handlers

A callback method is designated using the appropriate callback annotation.

The same method can be designated to handle multiple callback events.

Each callback event can only have one callback method designated to handle it.

A callback method can access entries in the bean’s environment.

Callback methods can throw runtime exceptions.

Callback methods must not throw application exceptions.

Callback methods can have any type of access modifier.
Example of a Callback Method in a Bean Class

```java
import javax.ejb.*;
import java.util.*;

@Stateful public class ShoppingCartBean implements ShoppingCart {
    private float total;
    private List productCodes;
    public int someShoppingMethod() {...};
    //...
    @PreDestroy private void endShoppingCart() {...};
}
```

The SessionContext Object

Use the SessionContext object to access EJB objects, obtain current transaction status, and obtain security information.

```java
import javax.ejb.*;
import javax.annotation.*;

@Stateful
public class BankingBean implements Bank {
    @Resource private javax.ejb.SessionContext context;
    ...
}
```
Implementing Session Beans

Developing Applications for Java EE Platform

Session Bean Packaging and Deployment

Packaging and deployment tasks:
► Optionally create a DD file for the session bean component
► Create a deployable session bean component archive
► Deploy the session bean component archive

Session Bean DD Example

```
<ejb-jar>
<enterprise-beans>
  <session>
    <ejb-name>ReportBean</ejb-name>
    <business-remote>services.Report</business-remote>
    <ejb-class>services.ReportBean</ejb-class>
    <session-type>Stateful</session-type>
    <transaction-type>Container</transaction-type>
  </session>
  <!-- Deployment information for additional beans goes here-->  
</enterprise-beans>
<assembly-descriptor>
  <!-- Add assembly information here-->  
</assembly-descriptor>
</ejb-jar>
```
Creating a Session Bean Component Archive

1. Create a working directory structure.
2. Copy the EJB component class files and helper classes into their corresponding subdirectories.
3. Create a META-INF subdirectory off the root directory.
4. From a command line, execute the jar utility to create the EJB JAR archive.

Example of the EJB Module JAR File

```
(\) Root
  | META-INF
  |  \ejb-jar.xml
  |  \MANIFEST.MF
  | auctions
  |  \auctions
  |  \system
  |  \ejbs
  |  \AuctionManagerBean.class
  |  \AuctionManagerRemote.class
  |  \AuctionManagerLocal.class
  |  \dtos
  |  \BidStatusMessage.class
  |  \PlaceBidMessage.class
```
Creating a Session Bean Client

Any managed component, such as a session EJB, message EJB, or servlet can perform dependency injection to locate an EJB using annotations and an instance variable.

Creating a client using container services:

```java
public class InternalClient {
    @EJB // step 1
    private static Calculator calc;
    public static void main (String args[]) {
        int sum = calc.add(1, 2); // step 2
    }
}
```

Creating a client without using container services:

```java
public class InternalClient {
    public static void main (String args[]) {
        // step 1 begin
        InitialContext ctx = new InitialContext();
        Object obj = ctx.lookup("ejb/Calculator");
        Calculator calc = (Calculator)
        PortableRemoteObject.narrow(obj, Calculator.class);
        // step 1 end
        int sum = calc.add(1, 2); // step 2
    }
}
```
Portable JNDI Session Bean Clients

The EJB specification does not require local session beans to have a default JNDI name. To perform portable JNDI lookups of a session bean requires editing the web.xml file or the ejb-jar.xml file depending on the client type.

A web tier example:

```xml
<ejb-ref>
    <ejb-ref-name>BrokerLookup</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <remote>trader.BrokerModel</remote>
</ejb-ref>

try {
    InitialContext ctx = new InitialContext();
    model = (BrokerModel)ctx.lookup("java:comp/env/BrokerLookup");
} catch (NamingException ne) {
    ...
}
```