Enterprise JavaBeans

Student Workbook
Contents

Chapter 1 - Course Introduction ............................................................................................................... 9
  Course Objectives ............................................................................................................................ 10
  Course Overview .............................................................................................................................. 12
  Using the Workbook ......................................................................................................................... 13
  Suggested References ....................................................................................................................... 14

Chapter 2 - EJB and the J2EE Architecture ............................................................................................. 17
  Evolution of Distributed Computing on the Web ................................................................................ 18
  The J2EE Solution ............................................................................................................................. 20
  The Enterprise JavaBean ................................................................................................................... 22
  Roles in Enterprise JavaBeans Development ...................................................................................... 24
  EJB Container and Application Server ............................................................................................... 26
  Reference Implementation Services ................................................................................................... 28
  Labs ................................................................................................................................................. 30

Chapter 3 - Getting Started ..................................................................................................................... 33
  Defining the Bean Class ..................................................................................................................... 34
  Remote Interface ............................................................................................................................... 36
  Writing Business Methods ................................................................................................................. 38
  Home Interface ................................................................................................................................. 40
  Compiling .......................................................................................................................................... 42
  Creating the J2EE Application ............................................................................................................ 44
  Packaging the Enterprise Bean ......................................................................................................... 46
  Deploying the J2EE Application ........................................................................................................ 48
  The Client ......................................................................................................................................... 50
  Locating the Bean ............................................................................................................................. 52
  Create an Enterprise Bean Instance .................................................................................................. 54
  Invoking the Bean’s Methods ............................................................................................................ 56
  Compiling the Client .......................................................................................................................... 58
  Deploying the Client .......................................................................................................................... 60
  Deploying the Client (contd.) ............................................................................................................. 62
  Running the Client ............................................................................................................................ 64
  Labs ................................................................................................................................................. 66
<table>
<thead>
<tr>
<th>Chapter 4 - The javax.ejb Package</th>
<th>69</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Remote Interface</td>
<td>70</td>
</tr>
<tr>
<td>The Home Interface</td>
<td>72</td>
</tr>
<tr>
<td>The Local Interface</td>
<td>74</td>
</tr>
<tr>
<td>The Local Interface Usage</td>
<td>76</td>
</tr>
<tr>
<td>Container Objects</td>
<td>78</td>
</tr>
<tr>
<td>SessionBean Interface</td>
<td>80</td>
</tr>
<tr>
<td>EntityBean Interface</td>
<td>82</td>
</tr>
<tr>
<td>MessageDrivenBean Interface</td>
<td>84</td>
</tr>
<tr>
<td>EJB Exceptions</td>
<td>86</td>
</tr>
<tr>
<td>EJB Context</td>
<td>88</td>
</tr>
<tr>
<td>Three Contexts</td>
<td>90</td>
</tr>
<tr>
<td>Other Interfaces</td>
<td>92</td>
</tr>
<tr>
<td>Labs</td>
<td>94</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5 - Session Beans</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Session Bean</td>
<td>98</td>
</tr>
<tr>
<td>What About State?</td>
<td>100</td>
</tr>
<tr>
<td>Stateless Session Beans</td>
<td>102</td>
</tr>
<tr>
<td>Life Cycle of a Stateless Session Bean</td>
<td>104</td>
</tr>
<tr>
<td>Stateful Session Beans</td>
<td>106</td>
</tr>
<tr>
<td>Life Cycle of a Stateful Session Bean</td>
<td>108</td>
</tr>
<tr>
<td>HttpSession and Cookies</td>
<td>110</td>
</tr>
<tr>
<td>Labs</td>
<td>112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6 - Deployment</th>
<th>115</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJB Deployment Process</td>
<td>116</td>
</tr>
<tr>
<td>Anatomy of the Deployment Descriptor</td>
<td>118</td>
</tr>
<tr>
<td>Structural Data</td>
<td>120</td>
</tr>
<tr>
<td>Assembly</td>
<td>122</td>
</tr>
<tr>
<td>Environment Entries</td>
<td>124</td>
</tr>
<tr>
<td>Compile</td>
<td>126</td>
</tr>
<tr>
<td>Bundling</td>
<td>128</td>
</tr>
<tr>
<td>Labs</td>
<td>130</td>
</tr>
</tbody>
</table>
Chapter 10 - Container-Managed Persistence ................................................................. 187
  Why CMP? .................................................................................................................... 188
  Two Layers of Responsibility .................................................................................... 190
  Coding Conventions in the Abstract Entity Bean ..................................................... 192
  Deploying a CMP Bean .............................................................................................. 194
  ejb-jar.xml .................................................................................................................. 196
  Proprietary Deployment Descriptors ....................................................................... 198
  Container-Managed Relationships ............................................................................ 200
  EJBQL: Finders in CMP ............................................................................................. 202
  EJBQL: Finders in CMP (contd.) ................................................................................ 204
  Limitations of CMP ................................................................................................... 206
  Labs ............................................................................................................................. 208

Chapter 11 - Transactions ............................................................................................. 211
  Transactions in EJB ..................................................................................................... 212
  Container-Managed Transactions ............................................................................. 214
  Transaction Support Attributes ................................................................................ 216
  Bean-Managed Transactions ..................................................................................... 218
  Session Synchronization .......................................................................................... 220
  Labs ............................................................................................................................. 222

Chapter 12 - EJB Security ............................................................................................ 225
  Security Issues ............................................................................................................ 226
  Authentication ............................................................................................................. 228
  Deploying a Secure Bean ........................................................................................... 230
  Authorization ............................................................................................................... 232
  Container-Managed Security ..................................................................................... 234
  Security and Deployment Descriptor ....................................................................... 236
  Bean-Managed Security ............................................................................................. 238
  Bean-Managed Security and Deployment Descriptor ............................................. 240
  Labs ............................................................................................................................. 242
# Enterprise JavaBeans

## Chapter 13 - JMS
- Introduction to JMS Concepts ................................................................. 246
- What is JMS? .............................................................................................. 248
- Parent Interfaces and GMD ......................................................................... 250
- JMS Definitions .......................................................................................... 252
- Message Object ........................................................................................... 254
- Multi-Threading and JMS Exception ............................................................ 256
- PTP Domain and Interfaces .......................................................................... 258
- Pub/Sub Domain and Interfaces ................................................................. 260
- JMS Server and Destination ......................................................................... 262
- Creating the Client ....................................................................................... 264
- Handling the Message .................................................................................. 266
- Producing the Message ................................................................................ 268
- Labs .............................................................................................................. 270

## Chapter 14 - Message-Driven Beans
- Message-Driven Beans ............................................................................... 274
- Asynchronous Processing .......................................................................... 276
- Comparing MDBs ....................................................................................... 278
- MDB Interfaces ............................................................................................ 280
- Deploying MDBs ......................................................................................... 282
- MDB Deployment Descriptor Elements ....................................................... 284
- Labs .............................................................................................................. 286

## Appendix A - Supporting Technologies
- JDBC ........................................................................................................... 290
- XML ............................................................................................................. 292
- RMI .............................................................................................................. 294
- Servlets ...................................................................................................... 296
- Database Transaction Isolation Levels ......................................................... 298
- The JSP Solution ......................................................................................... 300

## Solutions - Enterprise JavaBeans

## Index

© 2002 ITCourseware, LLC

Rev 2.1.2

Page vii
CHAPTER 1 - COURSE INTRODUCTION
Course Objectives

* Describe the Enterprise JavaBean architecture.
* Write session beans.
* Write entity beans.
* Deploy EJBs within an EJB Container.
* Locate and use an Enterprise JavaBean from a client.
* Access relational database information from a bean.
* Manage EJB transactions.
* Use the EJB security mechanisms.
* Explain the difference between container-managed services and bean-managed services.
* Use Java Message Services and message-driven beans to decouple transaction providers and consumers.
**Course Overview**

**Audience:** Java programmers who wish to use Enterprise JavaBeans to develop distributed intranet and Internet applications.

**Prerequisites:** *Java Programming for the Enterprise* and some Java development experience. Familiarity with Servlets, RMI and JDBC is recommended.

**Classroom Environment:**

- Workstation per student.
Chapter 1 Course Introduction

Using the Workbook

This workbook design is based on a page-pair, consisting of a Topic page and a Support page. When you lay the workbook open flat, the Topic page is on the left and the Support page is on the right. The Topic page contains the points to be discussed in class. The Support page has code examples, diagrams, screen shots and additional information. Hands On sections provide opportunities for practical application of key concepts. Try It and Investigate sections help direct individual discovery.

In addition, there is an index for quick look-up. Printed lab solutions are in the back of the book as well as on-line if you need a little help.

Hands On:
The init() method is called when the servlet is loaded into the container.

Today.java

```java
public class Today extends GenericServlet {
    private Date bornOn;

    public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
        // Write the document
        out.println("This servlet was born on " + bornOn.toString());
        out.println("It is now " + today.toString());
    }

    public void init() {
        bornOn = new Date();
    }
}
```

The Topic page provides the main topics for classroom discussion.

The Support page has additional information, examples and suggestions.

The Servlet Life Cycle

- The servlet container controls the life cycle of the servlet.
- When the first request is received, the container loads the servlet class.
- The container uses a separate thread to call the service() method.
- The container calls the destroy() method
- As with Java’s finalize() method, don’t count on this being called.
- Override one of the init() methods for one-time initializations, instead of using a constructor.
- The simplest form takes no parameters.
  ```java
  public void init() {...}
  ```
- If you need to know container-specific configuration information, use the other version.
  ```java
  public void init(ServletConfig config) {...
  ```
- Whenever you use the ServletConfig approach, always call the superclass method, which performs additional initializations.
  ```java
  super.init(config);
  ```
- Callout boxes point out important parts of the example code.
- Code examples are in a fixed font and shaded. The on-line file name is listed above the shaded area.
- Screen shots show examples of what you should see in class.
- Topics are organized into first (★), second (✓) and third (●) level points.
- Pages are numbered sequentially throughout the book, making lookup easy.
Suggested References


java.sun.com/j2ee
www.theserverside.com
www.ejbbean.com
www.jguru.com
Chapter 2 - EJB and the J2EE Architecture

Objectives

- Explain the structure and usage of Enterprise JavaBeans.
- Describe the relationship between EJBs, EJB containers and an application server.
- Describe distributed computing techniques and the evolution of EJB.
- Distinguish container-managed services from bean-managed services.
- Describe the three main roles in building a J2EE application.
- Install the J2EE Reference Implementation.
**Evolution of Distributed Computing on the Web**

- The combination of CGI and Perl was initially the most popular way to create dynamic HTML pages.
  - A single "Internet Programmer" was responsible for all aspects of web development.
  - While relatively easy to develop, CGI/Perl servers proved to be inefficient at runtime.

- CORBA allows a developer to invoke programs or objects in other servers.
  - Developers can implement CORBA in many different languages, on many different platforms.
  - CORBA servers, or ORBs, provide location transparency between the client and server.
  - CORBA requires a significant amount of overhead for relatively lackluster performance.
  - CORBA is not implementation independent, thus one CORBA application cannot be guaranteed to run on all CORBA servers.

- Transaction Process Monitors (TPM), in combination with CORBA, provide transaction management and application server functionality.
  - TPMs are proprietary to an implementation.

- Application Servers provide a framework for system-level services.
  - This framework frees up the developer to concentrate on business logic, not on system level services.
Transaction Process Monitors (TPMs), such as CICS from IBM and Tuxedo from BEA, provide mission-critical applications with a high-speed server platform. A TPM manages the environment of the application, providing functionality such as fault tolerance, resource management and transaction support.

A major drawback to TPMs is the fact that they are not object-based. In fact, most TPM applications are written in COBOL.
**The J2EE Solution**

- J2EE is a set of related specifications for enterprise application development and deployment consisting of many complementary technologies.

- The Web container is responsible for an application's web presentation.
  - JavaServer Pages (JSP) is one of the presentation layers of J2EE.
  - Servlets provide the control layer and separate the interface from the business logic.

- The EJB container holds and manages the Enterprise JavaBeans.

- The EJB container can use several different APIs to interact with Enterprise JavaBeans.
  - JDBC (Java Database Connectivity) facilitates the interaction of the EJB container and the database source.
  - JNDI (Java Naming and Directory Interface) is used to locate resources.
  - JMS (Java Message Service) provides for asynchronous messaging between servers.
  - JavaMail is Java's email API.
  - JTA / JTS (Java Transaction API / Java Transaction Service) facilitate bean-managed transactions.

- J2EE facilitates the separation of roles in web application development.
The Enterprise JavaBean

- The Enterprise JavaBeans architecture is a distributed component architecture for creating reusable and platform-independent applications.

- Enterprise JavaBeans encapsulate business logic.

  - An EJB developer concentrates on fulfilling business requirements, not on the environment or network architecture.

- The EJB container manages both session and entity beans.

  - Session beans are the task-oriented component of EJB.
    - Contain task-oriented business logic.
    - Coordinate entity beans, system resources and other session beans.
    - Data does not survive server crashes.

  - Entity beans represent real-world business objects.
    - Usually represent data in an existing database.
    - Can be located via a primary key or other unique identifier.
    - Data is persistent and survives server crashes.

  - Message-driven beans are message consumers in MOM architectures.
Container-managed services vs. bean-managed services:

The purpose of an Enterprise JavaBean container is to provide system level services, such as persistence and transaction management. These services allow a Bean Provider to concentrate on the business logic required by the application.

The Bean Provider should use as many of these container-managed services as possible. Doing so allows faster bean development, implementation of standardized system level services, decoupling of system level services from business logic, increased code reuse and less code complexity.

The declarative nature of container-managed services may prove limiting when the bean requires services that depend on parameters that are dynamic. The Bean Provider can specify bean-managed services in the deployment descriptor, overriding some services while using the container for the remaining ones.

The disadvantage to bean-managed services lies in limitations on portability, reusability and increased code complexity. The Bean Provider should use container-managed services wherever possible and bean-managed services only when necessary.
Roles in Enterprise JavaBeans Development

The Enterprise JavaBeans Specification 2.0 defines three roles in the development of Enterprise JavaBeans.

- The Bean Provider creates the enterprise bean and is only responsible for the bean as an individual unit.
  - The Bean Provider is required to provide the enterprise bean code (including home and remote interfaces and entries in the deployment descriptor that apply to that particular enterprise bean).

- The Application Assembler puts together a set of enterprise beans (potentially provided by disparate sources) into an application with the EJBs working in conjunction.
  - The Application Assembler is required to package all beans of an application in their respective archive files (.jar, .war, or .ear files) and make edits to the deployment descriptor to enable correct communications between the beans.

- The Deployer is responsible for deploying the application (provided by the Application Assembler) on a specific application server environment.
  - The Deployer primarily makes necessary changes to environment variables and mapping of logical variables to physical variables, such as matching security roles (logically defined in the application) to users and groups (defined in the application server).
The roles listed on the facing page are the ones that are important in the development of an EJB and its incorporation into a web application. The EJB 2.0 specification defines three additional roles: EJB Server Provider, EJB Container Provider, and System Administrator. These roles are primarily concerned with the development and administration of the EJB environment, instead of bean development, so we will not be focusing on them in this class.
The EJB container provides system-level functions for its deployed EJBs.

- Accessibility through the home interface.
- Resource management.
- Security management.
- Transaction management.
- Persistence management.
- Concurrency.
- Asynchronous messaging.

The application server wraps around the EJB container, providing additional system support.

- Clustering: a set of interconnected servers.
- Load balancing: balances the workload among the clustered servers.
- Fail-Over: monitors the servers in the cluster and restarts any servers that are down.

The application server and the EJB container are combined and come from the same vendor.

- Application servers often include a web server.
EJB application servers must follow the specification provided by Sun. How the specification is implemented is up to the application server vendor. EJB servers can have differences in services provided. The specification only indicates functions of the container, not of the application server.
The J2EE Reference Implementation (RI) provides many wizards and GUI tools for deployment and management of the application server.

However, you can invoke commands in scripts to expedite tasks and edit property files without using a wizard.

The J2EE Reference Implementation is Sun's proof-of-concept application for the set of J2EE specifications.

Application Server Vendors, Bean Providers, Application Assemblers and Developers can use this implementation both as a reference and as a working example.
Here are some important RI tools and directories:

- **public_html**: This is the default location of html files.
- **lib**: This is where J2EE system .jar files are kept.
- **bin**: For the following scripts, the file extension for UNIX scripts have .sh appended to them and for Windows, .bat. Wizards and GUI tools are provided; however, you can deploy an application via the command line, or use them in other scripts that you write.
- **cleanup**: This script will remove all deployed applications from the application server.
- **cloudscape**: This is the script that you invoke to start and stop the Cloudscape database.
- **deploytool**: This script starts the GUI to build, deploy and manage EJB applications. It can also be used in command line mode.
- **j2ee**: This script starts the J2EE server.
- **j2eeadmin**: This is used to administer JDBC datasources and JMS destinations.
- **keytool**: This script generates public and private keys for encryption.
- **packager**: This script allows you to package J2EE applications into their respective component bundles.
- **realmtool**: This script allows you to add and remove J2EE users and certificate files.
- **setenv**: This script sets many of the environment variables to run the J2EE server.
- **verifier**: Validates J2EE .ear, EJB .jar, Client .jar and .war files.
Labs

Setup the J2EE Reference Implementation.

1. Install the J2EE Reference Implementation.
   Install J2EE SDK.
   Double-click on the icon of the j2sdkee-1_3_1-win.exe file. By default, the setup program installs the software in \j2sdkee1.3.1.

2. Edit the userconfig.bat file of the bin directory.

   J2EE_CLASSPATH - set this to the location of the JDBC driver you are using.
   - Don't set this for the Cloudscape database.

   JAVA_HOME - set this to the installation location of the Java Development Kit.

   Example:
   set J2EE_CLASSPATH=C:\oracle\jdbc;C:\db\driver.zip
   set JAVA_HOME=C:\jdk1.3

3. Create an environment variable named J2EE_HOME and set it to the installation directory of the J2EE SDK.

4. Edit the PATH environment variable to include the bin directory of the J2EE SDK.

5. Edit the CLASSPATH environment variable to include %J2EE_HOME%/lib/j2ee.jar.

6. Start your server by typing j2ee -verbose.
Configure the J2EE Reference Implementation.
Edit the user configuration script - location: \bin\userconfig.bat.
Edit The J2EE_CLASSPATH environment variable (if you are not using Cloudscape).
Change the JAVA_HOME environment variable.

rem J2EE_CLASSPATH is appended to the classpath referenced by the EJB server.
rem J2EE_CLASSPATH must include the location of the JDBC driver classes rem (except for the Cloudscape driver shipped with this release).
rem Each directory is delimited by a semicolon.
rem rem set J2EE_CLASSPATH=
rem rem JAVA_HOME refers to the directory where the Java(tm) 2 SDK rem Standard Edition software is installed.
rem rem set JAVA_HOME=c:\jdk1.3
rem

For more installation information go to: http://java.sun.com/j2ee/sdk_1.3/install.html.

<table>
<thead>
<tr>
<th>Service</th>
<th>Port Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJB</td>
<td>9191</td>
<td>The EJB service uses this port to download stub classes to clients.</td>
</tr>
<tr>
<td>HTTP</td>
<td>8000</td>
<td>The HTTP service uses this port to service requests.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>7000</td>
<td>The HTTPS service uses this port to service requests for https.</td>
</tr>
<tr>
<td>JNDI</td>
<td>1050</td>
<td>The Object Request Broker (ORB) underlying the JNDI name server uses this port.</td>
</tr>
</tbody>
</table>
CHAPTER 5 - SESSION BEANS

OBJECTIVES

* Write a session bean.

* Describe the purpose of a session bean.

* Explain the difference between stateless and stateful session beans.

* Maintain state outside of the EJB container.
A Session Bean

* A session bean implements business logic.
  - The bean methods conceal the complexities of business tasks from the client.
  - Business logic is decoupled from the client presentation layer, enabling a more flexible and reusable design.
  - Examples of the roles of business logic components include payroll, inventory, order entry, etc.

* A session bean presents a simple interface to the client.
  - A session bean client is generally another web application component, such as a JSP, a servlet or another EJB.

* The bean coordinates other resources to accomplish a coarse-grained task.

* Session beans can be transaction aware.
Session bean interaction with a database:

Entity beans are typically used to interact with databases. This process, however, can be resource intensive. There are times when you should use a session bean, instead of an entity bean, to retrieve data from the database:

1. When the data that you are retrieving does not need to be shared across the enterprise (among many systems).

2. When the data that you need does not represent a business object.

3. Just for the purpose of lookup: A session bean can return a string instead of the remote interface.

4. Session beans can become transaction aware by implementing the `SessionSynchronization` interface.
**What About State?**

- *State* is an object's awareness of its attributes.
  - *Conversational state* is maintenance of object attributes across method calls.

- Stateful session beans have conversational state and attribute values that can be used by other methods.

- Stateless session beans do not have conversational state.
  - They maintain state only within the duration of one method call.
  - Attributes can be placed inside the bean, but cannot be relied on.
    - Attributes are shared between many clients, resulting in inconsistent values.
  - Stateless session beans are dedicated to one client only for the duration of the method call.
    - The bean is then returned to the pool in the container.
Stateless Session Beans

- Stateless session beans do not have conversational state across method calls.

- The container pools the stateless session beans.

- They are only dedicated to a single client for the duration of one method call.
  
  Therefore, they should be used when the entire task can be accomplished via one method call.

- Stateless session beans are a group of related generic services.

- While it is possible to put object attributes in the stateless session bean, the life cycle management of the container makes them unreliable for maintaining a persistent state.

- A stateless session bean can have only one `ejbCreate()` method.
  
  The `ejbCreate()` method cannot have any parameters.
CurrencyConverterEJB.java

```java
public class CurrencyConverterEJB implements SessionBean {
    private static double yenRate=0.00749;
    private static double euroRate=0.86675;
    private static double gbpRate=1.412;
    private SessionContext context;

    public double usdToYen(double dollars) {
        return dollars / yenRate;
    }
    public double usdToEuro(double dollars) {
        return dollars / euroRate;
    }
    public double usdToGBP(double dollars) {
        return dollars / gbpRate;
    }
    public double yenToUSD(double yen) {
        return yen * yenRate;
    }
    public double euroToUSD(double euro) {
        return euro * euroRate;
    }
    public double gbpToUSD(double gbp) {
        return gbp * gbpRate;
    }
    public CurrencyConverterEJB() {}
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {
        context = sc;
    }
}
```
**Life Cycle of a Stateless Session Bean**

There are two stages in a stateless session bean’s life cycle.

- The container management of the *Method Ready Pool* involves transition of the bean from a *Does Not Exist* state to a *Method Ready* state and back.

- *Method Invocation* from the client begins with the `create()` method of the home interface.
  - The container then returns a handle to the EJB container object.
  - The client invokes a business method on the remote interface.
  - The container assigns a method ready stateless session bean from the pool and invokes the matching business method in the bean.
  - The bean is only dedicated to the client for the duration of the method call.
  - The client then invokes the `remove()` method of the remote interface and the container removes the client's handle to the container object.
  - The bean remains in the pool.

There is no need for the container to invoke the `ejbActivate()` and `ejbPassivate()` methods because the bean has no state.

Passivation occurs when the state and reference to the EJB object of a stateful session bean become serialized to secondary memory.
Life cycle of a stateless session bean:

1. `newInstance()`
2. `setSessionContext()`
3. `ejbCreate()`

Container management of the Method Ready Pool:

<table>
<thead>
<tr>
<th>Transition</th>
<th>Container Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>from Does Not Exist to Method Ready</td>
<td><code>newInstance()</code> of the class</td>
</tr>
<tr>
<td></td>
<td><code>setSessionContext()</code> of the instance</td>
</tr>
<tr>
<td></td>
<td><code>ejbCreate()</code> of the instance</td>
</tr>
<tr>
<td>from Method Ready to Does Not Exist</td>
<td><code>ejbRemove()</code> in order to reduce the number of Method Ready beans in the pool</td>
</tr>
</tbody>
</table>
Stateful Session Beans

🌟 Stateful session beans have conversational state across method calls.

🌟 They are not pooled.

🌟 They are dedicated to one client for the life of the bean.

➤ Stateful session beans become integrated with the client, providing services for that client until the client is done with it.

🌟 Stateful session beans should be used when the task cannot be accomplished via one method call.

🌟 A session bean is made stateful with the
<session-type>stateful</session-type> element in the deployment descriptor.

🌟 Stateful session beans are allowed to overload the ejbCreate() method.
ShoppingCartEJB.java

```java
public class ShoppingCartEJB implements SessionBean {
    private HashMap items;

    public void addItem(String code, String descr, int quantity, double price) {
        ArrayList item = new ArrayList();
        item.add(descr);
        item.add(new Integer(quantity));
        item.add(new Double(price));
        items.put(code, item);
    }

    public void removeItem(String code) {
        if (!items.containsKey(code)) {
            throw new EJBEException(code + " not in item list.");
        }
        items.remove(code);
    }

    public Set getItems() {
        return new HashSet(items.keySet());
    }

    public String getLine(String code) {
        if (!items.containsKey(code)) {
            throw new EJBEException(code + " not in item list.");
        }
        ArrayList item = (ArrayList) items.get(code);
        String line = "";
        for (Iterator i=item.iterator(); i.hasNext(); ) {
            line = line + i.next().toString() + " ";
        }
        return line;
    }

    public void ejbCreate(int count) {
        items = new HashMap(count);
    }

    ...
```
Life Cycle of a Stateful Session Bean

The three stages in the life cycle of a stateful session bean are:

- **Does not Exist**, when no instances are created.
- **Method Ready**, when the bean is assigned to its client and ready to be used.
- **Passive**, when the non-transient attributes and association to the container object is serialized to secondary memory.

Life cycle of a stateful session bean:

1. newInstance()
2. setSessionContext()
3. ejbCreate()
<table>
<thead>
<tr>
<th>Transition</th>
<th>Container Calls</th>
</tr>
</thead>
</table>
| from Does Not Exist to Method Ready | create() on the home interface (passing some initial parameters)  
newInstance() on the bean class  
ejbCreate() of the instance (with parameters passed via create())  
Returns a handle to the EJB container object assigned to the session bean instance to the client |
| from Method Ready to Passive | ejbPassivate() called if container determines that the bean can be moved to secondary storage  
Bean instances are serialized to secondary storage |
| from Passive to Method Ready | Client invokes a business method on the remote interface  
Instance attribute(s) restored to primary memory  
ejbActivate() is immediately called by the container |
| from Passive to Does Not Exist | Container determines that the bean needs to be removed  
EJB container object associated with the bean is also removed |
| from Method Ready to Does Not Exist | remove(...) of the home interface -or- remove() of the remote interface  
ejbRemove() of the instance removes the EJB container object |
**HttpSessions and Cookies**

- An HttpSession object can maintain conversational state for a stateless session bean.
  - The HttpSession object must be accessed from a servlet or JSP.
  - A session ID is held in a cookie in the client browser.
    - If your client doesn’t accept cookies, you can perform URL rewriting.
- The HttpSession object is held inside the web container.
- The HttpSession object is usually dependent on the implementation.
  - All of the servlets and JSP pages within that Http session share the session object.
- The HttpSession object holds more than conversational state: it holds state for the entire session.
- Depending upon the implementation, many application servers provide redundancy for the HttpSession object in an abstract environment.
- The HttpSession object may survive server crashes (but this is not stated in the J2EE Specification).
- You can push state information all the way back to the client browser with cookies or hidden HTML fields.
Labs

1. Create a stateless session bean to implement your Inventory interface. Compile and deploy the bean.
(Solution: InventoryEJB.java)

2. Write a client application that first prints out the current inventory, then adds automobiles and, finally, displays the entire inventory. When running the client application multiple times, is the inventory re-initialized? Why or why not?
(Solutions: InventoryClient.java, Inventory.txt)

3. Change the state of your bean from stateless to stateful.
(Solutions: InventoryEJB2.java, Inventory2.java, InventoryHome2.java, InventoryClient2.java)
Chapter 12 - EJB Security

Objectives

- Use role driven access control.
- Use the java.security.Principal object.
- Add security attributes to a bean's deployment descriptor.
Security Issues

There are three kinds of security in Enterprise JavaBeans.

- **Authentication** is the process that determines the validity of the user trying to access a protected resource.

- **Authorization** takes place once the user is logged into the system and the system determines which resources the user can access.

- **Secure Communications** are ensured by encrypted data transfer between client and server.
  - **Secure HTTP (HTTPS)** is the protocol that is used for data communications between the web server and the client.
  - HTTPS is not a part of the EJB Specification.
**AUTHENTICATION**

- The J2EE authentication service determines the identity and validity of a user by querying the user id and password.

- J2EE authentication service is a security realm that is separate from the operating system.
  
  - A *security realm* is an environment that shares a common set of security policies and users.

- There are two realms within the J2EE authentication service:
  
  - *Certificate realm* is used to authenticate web browser clients when using the HTTPS protocol.
    
    - Authentication of a user (the browser) in the certificate realm is via an X509 certificate.
  
  - In the *default realm*, users and groups are defined in the application server environment.
To define a user in the J2EE Reference Implementation, use the `realmtool`.

List all options for the `realmtool`:

```
realmtool
```

List all users in the default realm:

```
realmtool -list default
```

Add the group `SalesManager` to the default realm:

```
realmtool -addGroup SalesManager
```

Add a user `JohnDoe` to the group `SalesManager`:

```
realmtool -add JohnDoe password SalesManager
```

Remove a group from the default realm:

```
realmtool -removeGroup SalesManager
```

Remove a user from the default realm:

```
realmtool -remove default JohnDoe
```
Deploying a Secure Bean

When you create the bean in the deploytool you need to define its roles.

- On the Security page click the Edit Roles ... button.
- Add the defined role for your bean.

Specify the method permissions by changing the Availability and selecting the authorized roles.

After you have created the bean, select the application in the tree view and select the Security tab.

- Select the role name and add users and/or groups that you want to map it to.
- Logical roles are defined in the bean deployment.
- Physical users and groups are defined with the realm tool.
Authorization

The process that determines whether a J2EE user has permission to access methods of an enterprise bean uses both physical and logical identity.

- The physical identity of a user and group is defined in the operating environment of the application server.
- The logical identity of the user is defined in the deployment descriptor of the application by the Application Assembler.

Users and groups are mapped to roles at deployment time by the Deployer.

By keeping the physical and logical identities separate, the application is portable across different operating environments.

The authorization process begins when a user logs in to the system and is wrapped by an object that implements the `java.security.Principal` interface (contained in the `EJBContext`).

- The `Principal` must be assigned at least one role that has permission to invoke the method.
- The `Principal` is propagated to all method calls and subsequent method calls.
- The default user is `guest` and requires no user name or password (anonymous and unauthenticated), having permission to access any unprotected resource.
  - It belongs to the `ANYONE` role.
Define physical identities with `realmtool`.

Define logical identities (roles) in the `ejb-jar.xml` deployment descriptor.

A physical identity is linked to a logical identity with a `<rolemapping>` in the `sun-j2ee-ri.xml` deployment descriptor.

sun-j2ee.ri.xml

```xml
...<j2ee-ri-specific-information>
  <rolemapping>
    <role name="manager">
      <principals>
        <principal>
          <name>scott</name>
        </principal>
      </principals>
      <groups>
        <group name="SalesManager" />
      </groups>
    </role>
  </rolemapping>
  <enterprise-beans>
    ...</enterprise-beans>
</j2ee-ri-specific-information>
```
Container-Managed Security

- The purpose of *Container-Managed Security* (CMS) is to exclude security-related code in the enterprise bean.

- The responsibility of security rests on the Application Assembler and the Deployer.

- CMS uses role-driven access control.

  - The Bean Provider uses `<security-role-ref>` to identify logical roles that the Bean Developer feels should have access to the enterprise bean.
    - The role name defined is used as the argument for the method `isCallerInRole()` of the EJBContext object.

  - The Application Assembler creates the security view and defines security roles of the application.
    - It maps the `<security-role-ref>` (defined by the Bean Provider) to the security role.
    - It determines what roles have access to certain methods in the deployment descriptor.
    - It defines logical roles not specific to the operating environment that the application will be deployed on (application server).

  - The Deployer creates a security policy descriptor (proprietary) that maps users and groups specific to the operating environment (application server) to users and groups specific to the application.
Security and Deployment Descriptor

The Application Assembler defines security roles, determines the methods that the role is authorized to invoke, and maps roles to role references.

Use the `<security-role>` element to define a role.

- It can contain an optional `<description>` sub-element.
- The `<security-role>` must have a `<role-name>` sub-element defined.

Methods the security role has permission to invoke are declared within the `<method-permission>` elements of the deployment descriptor.

- The `<role-name>` sub-element maps to the role name of the corresponding security role definition.
- One or more `<method>` elements define which methods can be called by this role.

- The `<method>` element contains the following sub-elements:
  - `<ejb-name>` - name of the enterprise bean.
  - `<method-name>` - name of the method the role has permission to invoke.
  - To specify all methods, use an asterisk (*).
  - Use the method name to specify an individual method.
ejb-jar2.xml

...<ejb-jar>
  <display-name>SalesPersonJAR</display-name>
  <enterprise-beans>
    <entity>
      <display-name>SalesPersonEJB</display-name>
      <ejb-name>SalesPersonEJB</ejb-name>
      ...
      <security-role-ref>
        <role-name>SalesManager</role-name>
        <role-link>manager</role-link>
      </security-role-ref>
      ...
    </entity>
  </enterprise-beans>
  <assembly-descriptor>
    <security-role>
      <description>A manager at the dealership</description>
      <role-name>manager</role-name>
    </security-role>
    ...
  </assembly-descriptor>
</ejb-jar>
Bean-Managed Security

- By managing security directly in the code, the Bean Provider can control security without redeploying the enterprise bean.

  - Since the logic that controls bean security is in the code, access can be determined based upon a dynamic parameter.

    - Using Bean-Managed Security (BMS), parameters such as time of day, state of another object and the value of an attribute can determine whether a Principal is granted access to a resource.

  - State-based security uses the java.security.Principal object to represent a user, group or other identity in the operating environment of the application server.

    - The EJBContext object invokes security methods.

    - getCallerPrincipal() determines the physical identity (user, group, smart card, etc.) of the Principal.

    - isCallerInRole() determines if the Principal is in a role that is authorized to invoke a protected method.
Get the Principal object:

```java
EJBContext context;
Principal thePrincipal = context.getCallerPrincipal();
```

Get the user id of the Principal:

```java
String userName = context.getCallerPrincipal().getName();
```

Determine if the Principal is assigned a role authorized to invoke the method. The role name passed to isCallerInRole is the one that was defined by the Bean Provider.

```java
boolean isInRole = context.isCallerInRole("SalesManager");
```
Bean-Managed Security and Deployment Descriptor

- The Bean Provider defines the `<security-role-ref>` to aid the Application Assembler when identifying the security roles of the application.

- It contains the following elements:
  - `<description>` - an optional description of the role reference.
  - `<role-name>` - the name of the security role reference.
  - Since the Bean Provider may be a third party, the `<role-name>` of the `<security-role-ref>` may not match the `<role-name>` of the `<security-role>` tag pair.
  - `<role-link>`: the `<role-name>` of the security role that this security role reference is linking to.
    - It must be present even if the `<role-name>` of the security role reference is the same as the `<role-name>` of the `<security-role>` tag pair.
    - The `<role-link>` value is typically written by the Application Assembler.
ejb-jar2.xml

...<ejb-jar>
    <display-name>SalesPersonJAR</display-name>
    <enterprise-beans>
        <entity>
            <display-name>SalesPersonEJB</display-name>
            <ejb-name>SalesPersonEJB</ejb-name>
            ...
            <security-role-ref>
                <role-name>SalesManager</role-name>
                <role-link>manager</role-link>
            </security-role-ref>
            ...
        </entity>
    </enterprise-beans>
    <assembly-descriptor>
        <security-role>
            <description>A manager at the dealership</description>
            <role-name>manager</role-name>
        </security-role>
        ...
    </assembly-descriptor>
</ejb-jar>
Labs

1. Use realmtool to create a new user named "manager" with a password of "secret" in the "staff" group. Be sure to restart or refresh your application server so it can recognize the new account. **Hint:** for the Reference Implementation, the command-line utility for creating a new user is realmtool.

2. Modify the SalesPerson bean source in the chapter directory by adding an accessor method called getAuthName(). This method should return the account used by the container to access your bean methods. *(Solutions: SalesPerson.java, SalesPersonEJB.java)*

3. Deploy the SalesPerson bean as an entity bean with container-managed persistence. Set up the security roles to give the "manager" account access to the find and get methods in your bean. When deploying the bean, be sure to save the client jar file to the chapter directory.

4. Compile SalesPersonClient.java and deploy it as an application client in your SalesPerson bean application. Run the client using runclient. When prompted for a username/password pair, type in "manager" for the username and "secret" for the password. Verify that the client application runs successfully. Repeat the execution task with an incorrect password. What happens?

5. Modify SalesPersonClient.java to call the getAuthName() method. Display the contents for at least one of the bean objects. Redeploy the application and run it again. *(Solution: SalesPersonClient.java)*