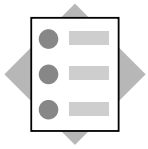
Using EJBs as Web Dynpro Model

Contents:

- **Overview Enterprise Java Beans**
- **Using EJBs as Web Dynpro Model**

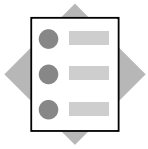
Using EJBs as Web Dynpro Model: Objectives



After completing this lesson, you will be able to:

- **Understand what EJBs are.**
- **Use EJBs as Model for Web Dynpro applications.**

Enterprise Java Beans: Topic Objectives

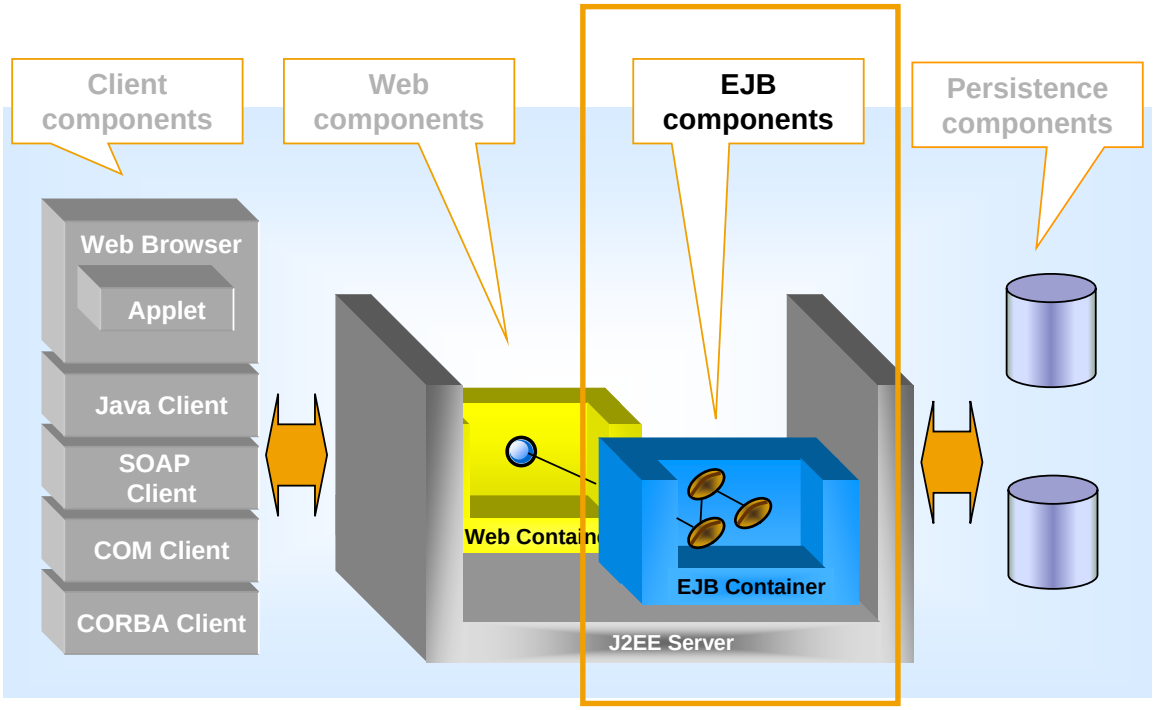


After completing this topic, you will be able to:

- **Understand what EJBs are.**

Components

A J2EE application can consist of the following components:

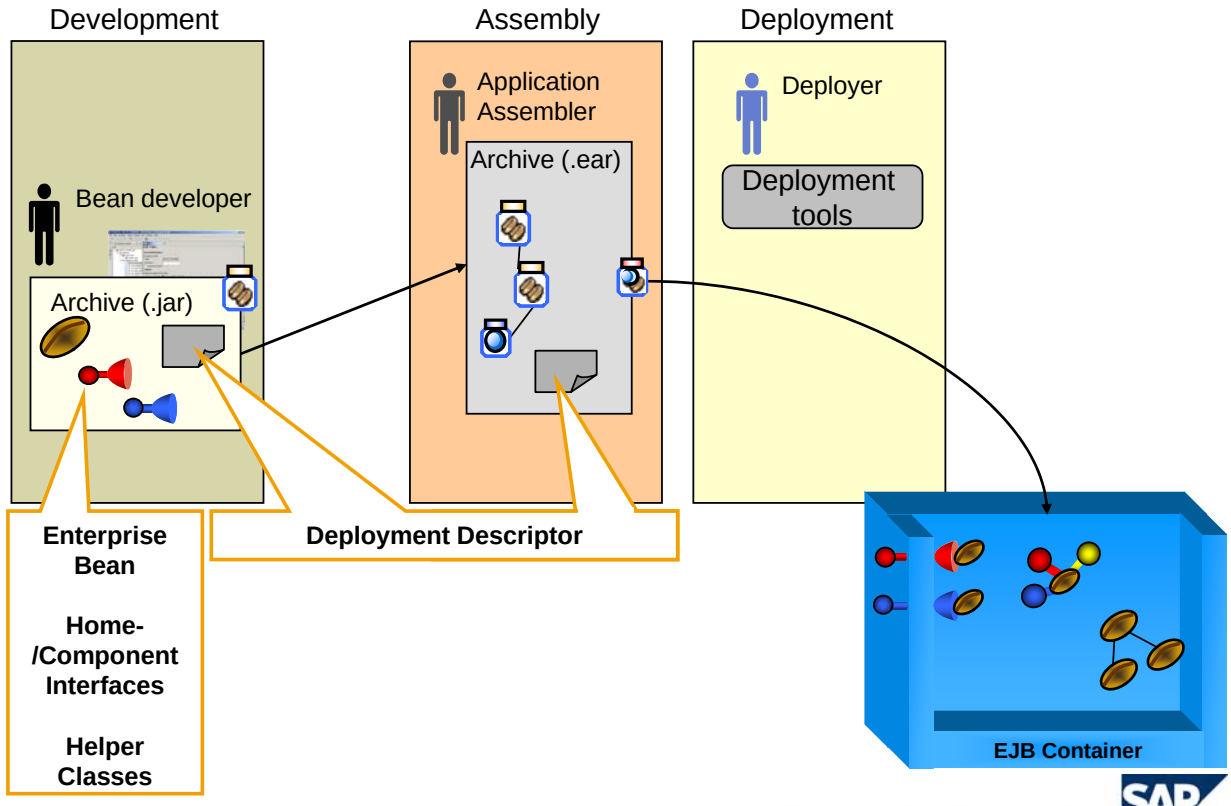


■ **Components**
J2EE applications provide their functions in various components. These are always dependent on a specific runtime environment (container) when they are executed.

J2EE specifies the following main types of components:

- Client components
  This is usually an application that is outside of the J2EE server, which accesses components that are also external to the J2EE server. Client applications are usually combined in jar-archives.
- Web components
  This includes HTML pages, servlets, JSPs and pictures (.gif or.jpeg files), which are combined in a war-archive.
- EJB components
  These are one or more Enterprise Beans, which are combined in a jar-archive.

What Makes Up An EJB Component ?



■ **The Parts of an Enterprise Bean**

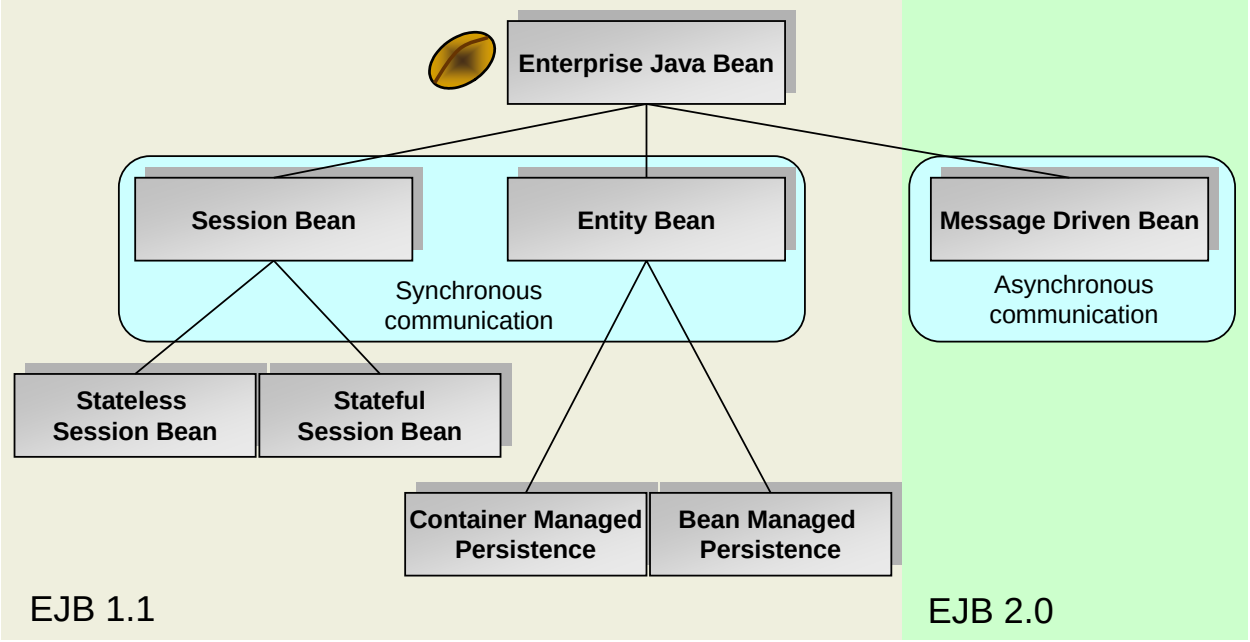
To develop an enterprise bean, you must provide the following files:

- **Deployment descriptor**: An XML file that specifies information about the bean such as its persistence type and transaction attributes. The deploytool utility creates the deployment descriptor when you step through the New Enterprise Bean wizard.
- **Enterprise bean class**: Implements the methods defined in the following interfaces.
- **Interfaces**: The remote and home interfaces are required for remote access. For local access, the local and local home interfaces are required. (Please note that these interfaces are not used by message-driven beans.)
- **Helper classes**: Other classes needed by the enterprise bean class, such as exception and utility classes.

You package the files in the preceding list into an EJB JAR file, the module that stores the enterprise bean. An EJB JAR file is portable and may be used for different applications. To assemble a J2EE application, you package one or more modules—such as EJB JAR files—into an EAR file, the archive file that holds the application. When you deploy the EAR file that contains the bean's EJB JAR file, you also deploy the enterprise bean onto the J2EE server.

Enterprise Java Beans Types

Enterprise Bean types are distinguished by their area of implementation, communication type and lifetime.



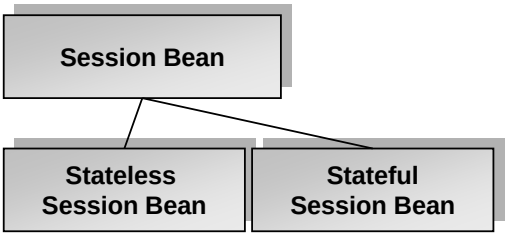
- **EJB types**

  3 different versions of the Enterprise Bean are available:

Session Beans	Usually implement transactions or process flows, which are executed as services that are performed for clients.
Entity Beans	Represent business objects. These are associated persistent objects that contain data from a database.
Message Driven Beans	Also implement transactions or process flows, but are triggered by receiving a message.

Session Beans

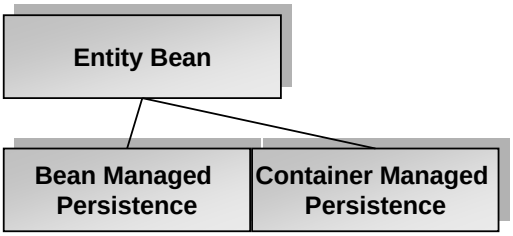
- Purpose: Session Beans usually represent business processes.
- Characteristic: Stateless Session Bean (SLB). Stateful Session Bean (SFB).
- Collective usage: SFB: Assigned to a client. SLB: No dedicated client assignment.
- Persistence: Transient. Lifetime defined by the Client Process (Session duration).



- **Session Beans**
  Session Beans implement specific functions, which you make available to clients, such as purchase order entry, bank transfers etc. A session bean can in turn access other Enterprise Beans or services for the EJB container.
  Session objects are always short-lived. A Session Bean usually only exists for as long as the session lasts. This means that there is a connection between the client and the EJB container.
- **Stateful Session Bean**
  The respective session object must be able to trace the client state across the various steps involved for business processes that are divided between several dialog steps (as a conversation). These types of business process, which can cover several method calls or transactions, have been implemented using Stateful Session Beans.
- **Stateless Session Beans**
  This type of bean is used for business processes, which are initiated from the client via a method call. The state of the client is only valid during the method call.

Entity Beans

Purpose:	Entity Beans represent data (usually from DBMS).
Characteristic	Bean Managed Persistence (BMP). Container Managed Persistence (CMP).
Collective usage:	Typically collective usage by several clients
Persistence:	Persistent State remains in the persistent memory (such as the DB) even after the container is terminated.



■ **Entity Beans**

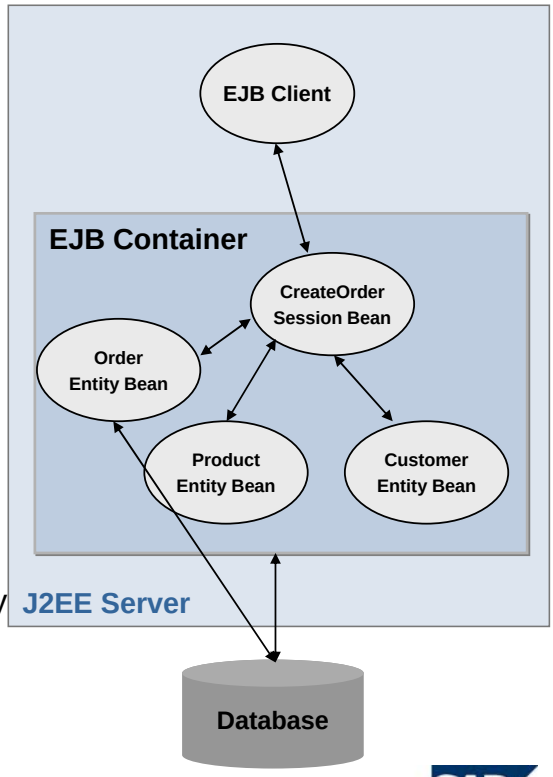
Entity Beans in their most simple form, correspond conceptionally to a table line in a relational database. They represent persistent objects, meaning that they encapsulate data from a persistent memory, usually a database. In doing so, they represent an interface between the business logic and the database and give clients transaction-secure access to data.

Entity objects are persistent (long-lived), since their state is saved to the database.

An additional feature of entity objects is that several clients can access them in parallel. The EJB container is responsible for capturing problems that can occur during competitive access, for instance by using synchronization.

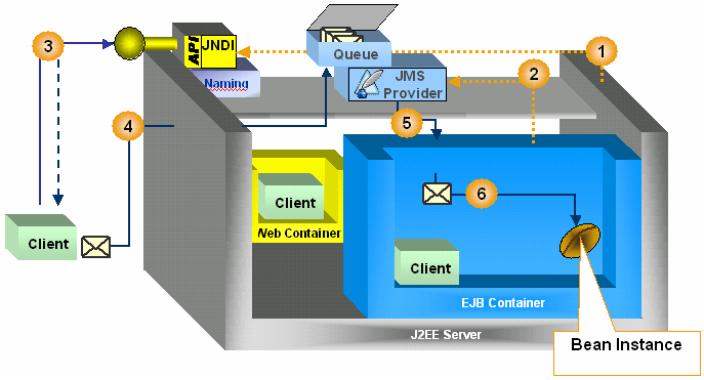
EJB, Big Picture

- Session Bean
  - Is thought to represent business process
  - Provides state management
- Entity Bean
  - Is thought to represent business object
  - Memory representation of persistent object
- Bean Managed Persistence
  - Persistence management to be coded by EJB developer, thus allowing higher flexibility
- Container Managed Persistence
  - Persistence management automatically done by EJB container
  - Declarative mapping of container-managed fields to table columns outside Java code (XML)



Message Driven Beans

- Purpose: Represents message recipient. A Message Driven Bean is a stateless component, which is accessed by messages.
- Characteristic: There is only one bean type.
- Collective: There is no direct client assignment. Bean usage: “listens” to a JMS channel.
- Persistence: Transient. Lost during container termination.



- **Message Driven Bean**
  The client uses asynchronous communication for MDBs to execute the business logic for the bean. Communication uses the JMS log. The client acts as a message producer, whilst the MDB is a message consumer here.
  The basis for use of Message Driven Beans is to transfer the JMS communication model to the EJB Specification 2.0 that contains the two central communication models (Point-to-Point and Publish/Subscribe). MDBs are not directly connected with queues here or entered as a subscriber. Connection to the Messaging System takes place via the EJB container, which receives the incoming messages and forwards these to an MDB instance.
- **Lifecycle of an MDB**
  A Message Driven Bean has a relatively short lifecycle. When a message is received from the client, the onMessage() method is accessed by the container and the message is transferred as a parameter. The Message Driven Bean only exists for as long as the onMessage() method is being processed.

The Deployment Descriptor (DD)

- Deployment descriptors contain information on how components are installed on the respective application server (declarative specification).

ejb-jar.xml

```
 1  <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE ejb-jar PUBLIC "-//Sun
    Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
    "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
 2 <ejb-jar>
 3  <description>My first EJB</description>
 4  <display-name>HelloWorld</display-name>
 5  <enterprise-beans>
 6   <session>
 7    <ejb-name>HelloWorldBean</ejb-name>
 8    <home>com.sap.training.HelloWorldHome</home>
 9    <remote>com.sap.training.HelloWorld</remote>
10    <local-home>com.sap.training.HelloWorldLocalHome</local-home>
11    <local>com.sap.training.HelloWorldLocal</local>
12    <ejb-class>com.sap.training.HelloWorldBean</ejb-class>
13    <session-type>Stateless</session-type>
14    <transaction-type>Container</transaction-type>
15   </session>
16  </enterprise-beans>
17 </ejb-jar>
```


■ **The Deployment Descriptor**

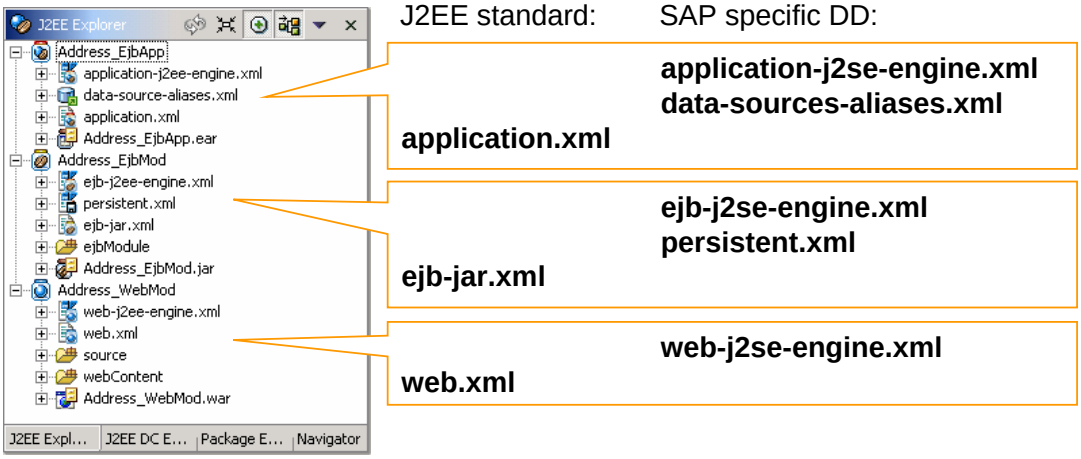
The deployment descriptor is a well-formed XML file called *ejb-jar.xml* contained in *ejb-jar* archive's subdirectory META-INF. It describes the structure and runtime behaviour of a bean and allows *customizing* and *reusing* the bean without changing the source code

Covered meta data:

- bean name
- names of bean's components class files
- security issues
- persistence mechanism
- transactional behaviour
- references to other beans, data sources and other resources used

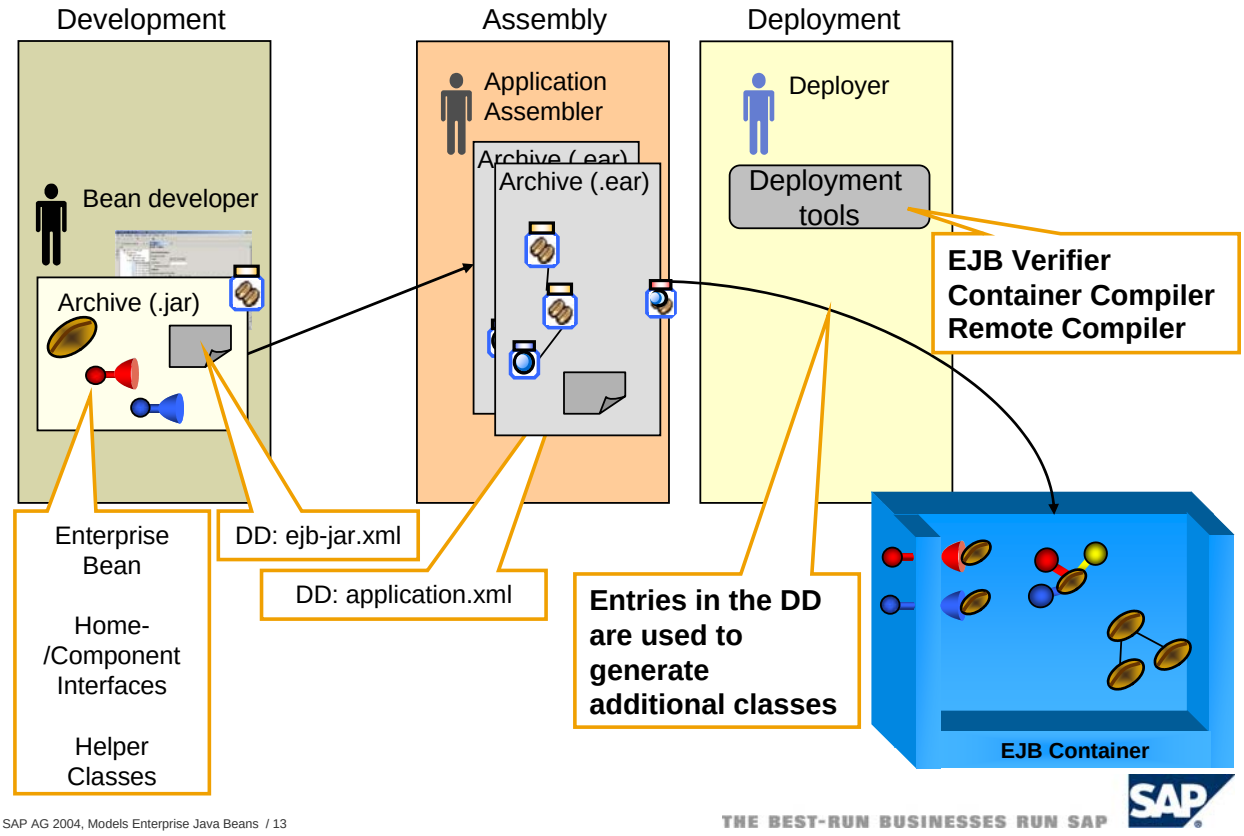
Types Of Deployment Descriptors

- Enterprise Application, Web Application, and EJB Assembly projects (and their respective archive files: EAR, WAR, and JAR) contain deployment descriptors.
- These are automatically generated when the projects are created and when they are modified.



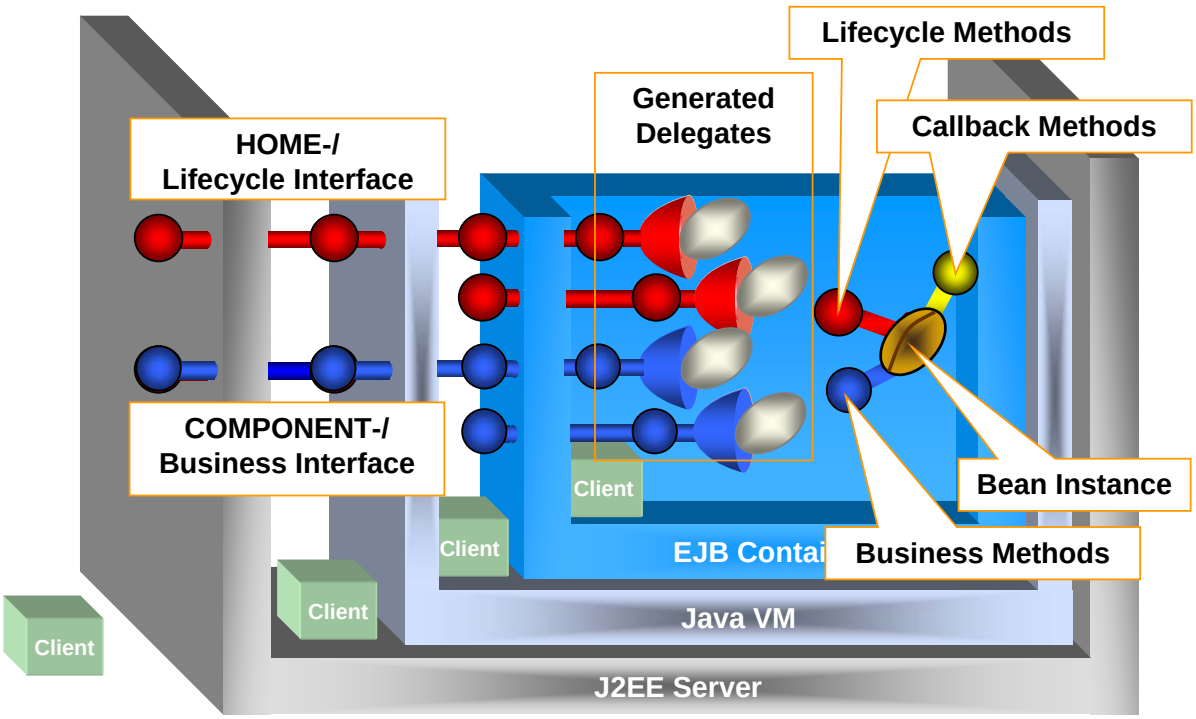
- **application.xml**
  Describes the standard J2EE properties of the complete application and the referenced modules. In particular, you must specify the URL where the J2EE application is stored on the J2EE Engine.
- **application-j2ee-engine.xml (SAP specific)**
  Contains additional entries specific to the SAP J2EE Engine. You do not need to make any entries for the car rental application in this deployment descriptor.
- **ejb-j2ee-engine.xml (SAP specific)**
  Contains entries specific to the J2EE Engine.
- **persistent.xml (SAP specific)**
  Describes the mapping of entity beans and their CMP fields to the corresponding database tables and table fields.
- **ejb-jar.xml**
  Describes the standard J2EE properties of the Enterprise JavaBeans.
- **web-j2ee-engine.xml (SAP specific)**
  Contains entries concerning Web resources specific to the SAP J2EE Engine.
- **web.xml**
  Describes the standard J2EE properties of the Web resources (JSPs, servlets, and so on). These properties include mapping information, security entries (access restrictions and security roles), and entries concerning EJB reference names.

The Deployment Process



- **How does EJB work?**
  Now that we have our EJB-Jar file containing our Bean, Home and Component interfaces and Deployment Descriptor, Let's take a look at how all of these pieces fit together and why Home and Component interfaces are needed and how the EJB Container uses them.
- **Deployment**
  This work step has not been standardized in the J2EE Specification. Each container product has its own solution here.
  During deployment of an .ear file, the following steps are usually executed by the container:
  - The system checks whether components in the EJB Jar file adhere to the rules in the EJB specification.
  - The container tool generates the EJB and home classes for the Enterprise Beans.
    Methods in the remote object correspond to methods in the Enterprise Bean.
    However, methods in the remote object contain additional code that is added using entries in the deployment descriptor. The remote object then acts as a proxy object.
  - The container tool generates all stub and skeleton classes that are required to support RMI-IIOP.

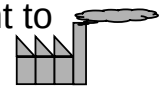
How Does EJB Work ?



- **How does EJB work?**
  The container generated classes for EJBHome, EJBLocalHome, EJBObject and EJBLocalObject will include the code for managing the bean's security, concurrency, persistence, remote access, transaction handling, … issues transparently to the application

The Home Interface

- A bean's home interface specifies methods that allow the client to create, remove, and find objects of the same type (^Factory).



- Remote Home Interface

HelloWorldHome

```
1 import javax.ejb.EJBHome;
2 import java.rmi.RemoteException;
3 import javax.ejb.CreateException;
4 public interface HelloWorldHome extends EJBHome {
5   public HelloWorld create() throws CreateException,
6                                     RemoteException;
7 }
```

- Local Home Interface

HelloWorldLocalHome

```
1 import javax.ejb.EJBLocalHome;
2 import javax.ejb.CreateException;
3 public interface HelloWorldLocalHome extends EJBLocalHome {
4   public HelloWorldLocal create() throws CreateException;
5 }
```


■ **The Home Interface**

The home interface provides life-cycle methods for creating, destroying, and locating beans. These life-cycle behaviors are separated out of the remote interface because they represent behaviors that are not specific to a single bean instance.

The home interface may also provide definitions for *home business methods* for entity beans. Home business methods are methods that are not specific to a particular bean instance. While the developer writes the home interface, the container creates the implementation for client interaction.

In essence, the home interface provides bean management and life cycle methods.

The Component Interface

- Enterprise JavaBean functionality is obtained through the bean's component interface, which defines the business methods visible to, and callable by, the client.
- Remote Component Interface

HelloWorld

```
1 import javax.ejb.EJBObject;
2 import java.rmi.RemoteException;
3 public interface HelloWorld extends EJBObject {
4   public String sayHello(String name) throws RemoteException;
5   public String sayHelloWorld() throws RemoteException;
6 }
```

- Local Component Interface

HelloWorldLocal

```
1 import javax.ejb.EJBLocalObject;
2 public interface HelloWorldLocal extends EJBLocalObject {
3   public String sayHello(String name);
4   public String sayHelloWorld();
5 }
```


■ **The Component Interface**

Enterprise JavaBean functionality is obtained through the bean's component interface, which defines the business methods visible to, and callable by, the client. Again, the developer writes the component interface, and the container provides the communication glue that is created at deploy time.

The client uses a home interface's create() method to create a logical instance to a bean's component. In the entity bean section, we will see that a component interface may also be returned by findByPrimaryKey() and other *finder methods.*

The Enterprise JavaBean, Example “HelloWorld”

Enterprise JavaBean

HelloWorldBean

```
 1 import javax.ejb.SessionBean;
 2 import javax.ejb.SessionContext;
 3 import javax.ejb.CreateException;

 4 public class HelloWorldBean implements SessionBean {
 5   private SessionContext myContext;
 6   public void ejbCreate() throws CreateException {
 7     // TODO : Implement
 8   }
 9   public void ejbRemove() {}
10   public void ejbActivate() {}
11   public void ejbPassivate() {}
12   public void setSessionContext(SessionContext context) {
13     myContext = context;
14   }
15   public String sayHello(String name) {
16     return "Hello "+name;
17   }
18   public String sayHelloWorld() {
19     return "Hello World!";
20   }
21 }
```

Lifecycle Methods

Callback Methods

Business Methods

■ **The Enterprise JavaBean Class**

This class provides the implementation of the business logic and methods for the EJB container (callbacks and lifecycle events). It has to implements one of the javax.ejb.EnterpriseBean derived interfaces SessionBean, EntityBean or MessageDrivenBean.

At Runtime, Instances of the Enterprise JavaBean class is fully managed by the container and it can only indirectly used by its clients

The diagram above shows a Stateless Session Bean.

Line 4: Session Beans implement the javax.ejb.SessionBean interface

Line 6-8: Lifecycle method for the container, which corresponds to the method with the same name in the Home Interface.

Lines 9-14: Callback methods in the container that are proposed by javax.ejb.SessionBean and implemented in the Session Bean.

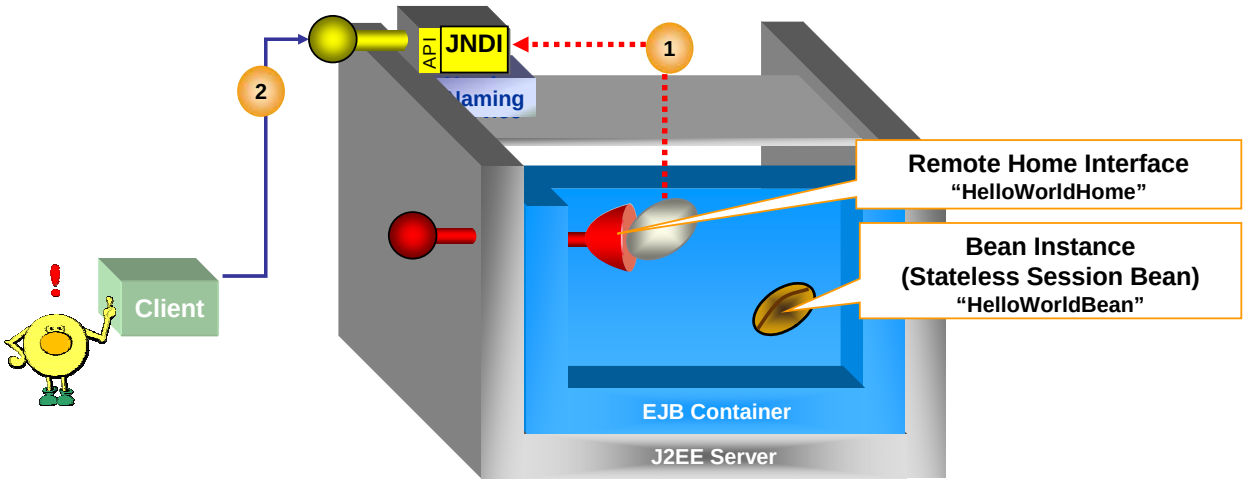
Line 15-20: Business methods that correspond to the methods of the remote interface with the same name.

Naming Conventions

Item	Syntax	Example
Remote Home interface	<*name*>Home	HelloWorldHome
Local home interface	<*name*>Local Home	HelloWorldLocalHome
Remote (Component) interface	<*name*>	HelloWorld
Local (Component) interface	<*name*>Local	HelloWorldLocal
Enterprise bean class	<*name*>Bean	HelloWorldBean
Web Service	<*name*>WS	HelloWorldWS
SAP Web Dynpro	<*name*>WD	HelloWorldWD

- **Naming Conventions**
  Because enterprise beans are composed of multiple parts, it's useful to follow a naming convention for your applications.
- **Local Home vs. Remote Home Interface**
  There are some issues to keep in mind when using local interfaces:
  The beans must run in the same VM -- they are, after all, *local.*
  Parameters running under a local interface are sent by *reference* rather than being copied, as is the case for remote objects.
  Unexpected side effects can result if you ignore this distinction and do not code accordingly.
  Typically, you'll decide whether to use local or remote access based on:
  Unless the **client is always expected to run in another VM** choose remote access.
  Whether the beans are tightly or loosely coupled. If beans depend on each other and interact frequently, you should consider local access.
  Scalability. Remote access is inherently scalable and should be used if scalability is an important factor.
  With the advent of local interfaces in the EJB 2.0 specification, it is recommended that *entity beans* should almost always be based on local access. When using local interfaces, most performance issues regarding very fine-grained data access go away. If the client is remote, the standard design pattern has the client use a remote interface to access a session bean, which then acts as a liaison to the entity bean. The session bean communicates with the entity bean through a local interface (*Session Façade Pattern*).

The (Remote) Client's View, Create JNDI Lookup



HelloWorldClient

```
1 Properties jndiCtxProp = new java.util.Properties();
2 jndiCtxProp.put(Context.INITIAL_CONTEXT_FACTORY, jndiInitCtxFactory);
4 . . .
5 try {
6   Context jndiCtx = new InitialContext(jndiCtxProp);
7   Object obj      =
8       (Object)jndiCtx.lookup("com.sap.training.HelloWorldBean");
```

2

■ **Performing Lookup from Application Client**

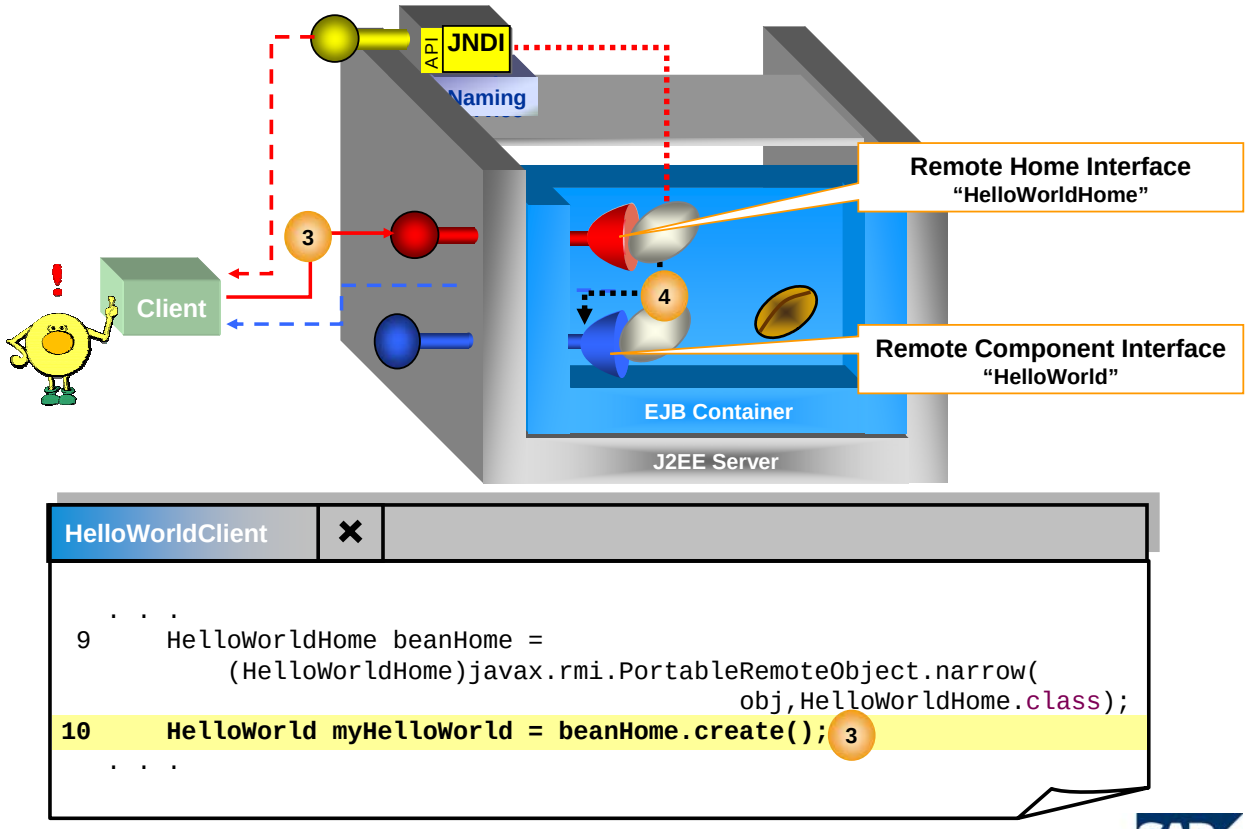
The application client model enables you to access enterprise beans and other resources (for example, JMS or database) from an application client. To use these resources (that is, to be able to invoke their methods), you must first obtain a reference to them. The resource and enterprise bean references are bound in the JNDI namespace and are obtained by performing a lookup operation in a relevant location in the naming.

■ **Procedure**

Line 6: Create InitialContext, which provides client access to the JNDI Registry Service through the SAP J2EE Engine as a name service provider. For more information, refer to the SAP NetWeaver Developer Manual.

Line 7: (Step2) The clients asks the naming service via JNDI for the reference to the home object of the Session Bean.

The (Remote) Client's View, Call Home Interface

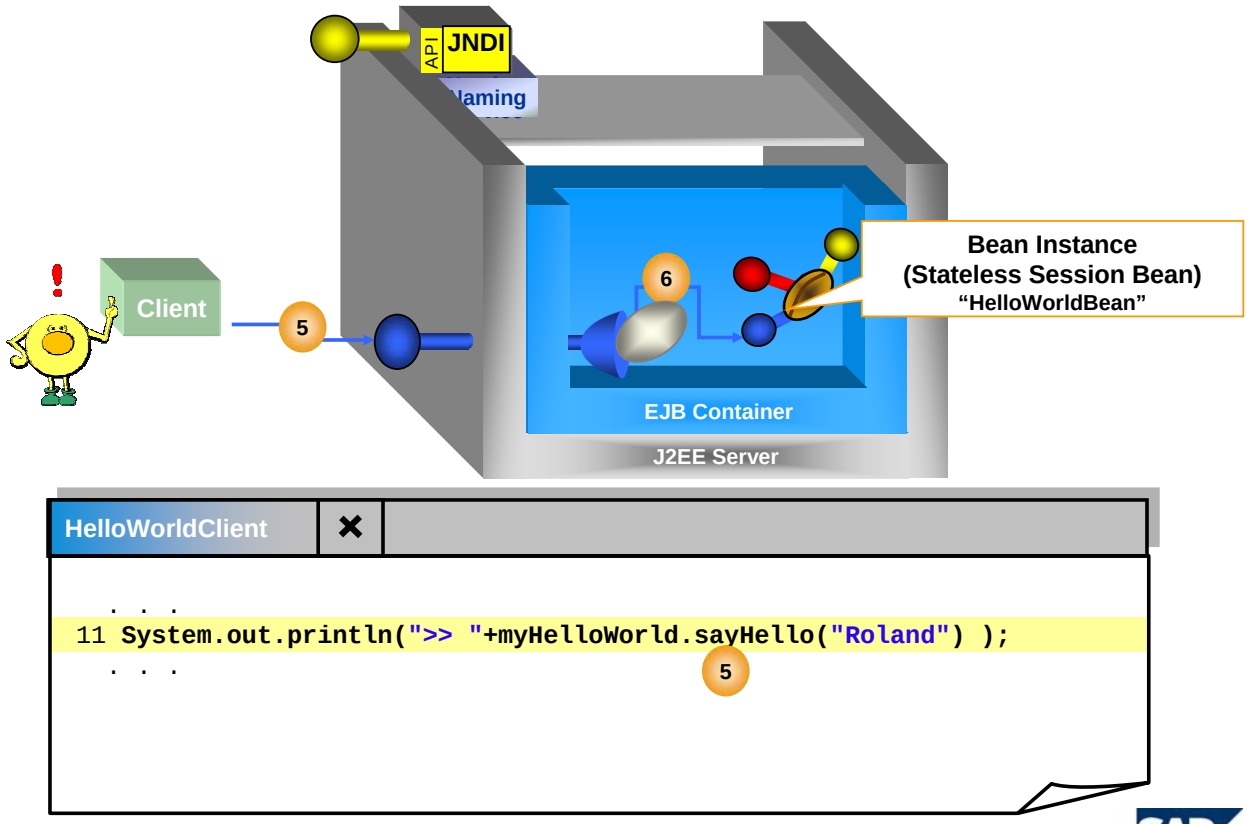


- **Call the Beans create-Method**

  Line 9: The client has to cast the object reference to the corresponding datatype of the Remote Home Interface.

  Line 10: (Step 3) The client now calls the create –Method of the Remote Home Object. As a result, the containers creates a Remote Object and returns its reference to the client, using the Remote Interface datatype.

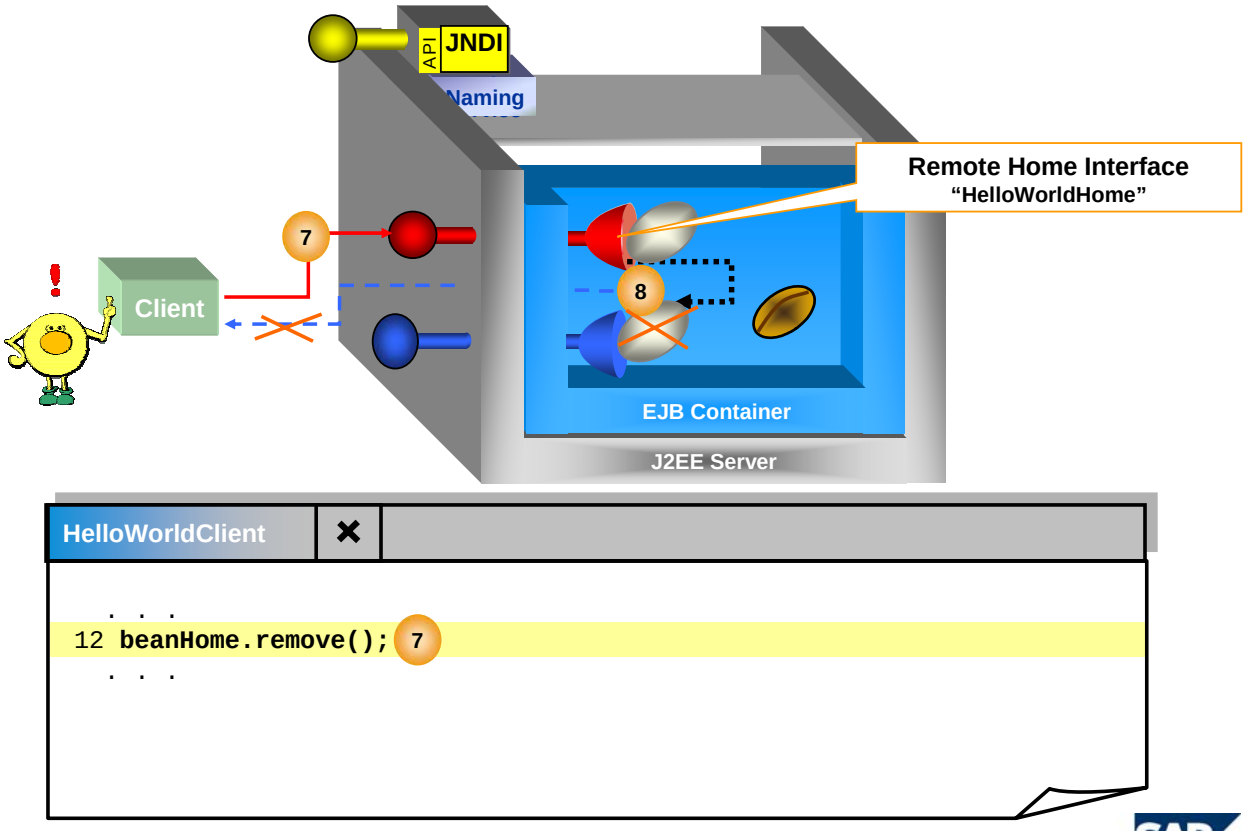
The (Remote) Client's View, Call Remote Interface



- **Call the Business Method**

  Line 11: (Step 5) The client calls the business method sayHello.
  (Step 6) The EJB-Object passes the parameters to the corresponding methods of the stateless session bean, which returns the complete „Hello"-String to the client.

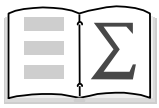
The (Remote) Client's View, Remove EJB



- **Call the remove() Method**

  Line 12: (Step 7) The client calls the remove() method.
  (Step 8) The EJB-Object is deleted by the EJBHomeObject. The client loses associated reference.

Enterprise Java Beans: Topic Summary

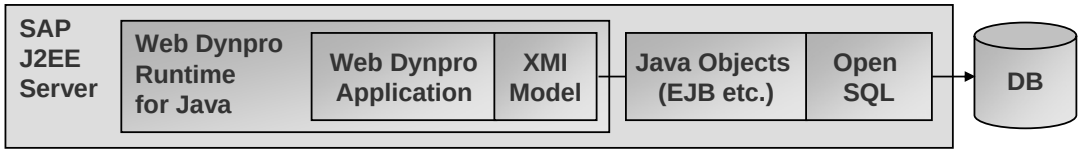


You should now be able to:

- **Understand what EJBs are.**

Application Scenario: Native Java Backend

- Web Dynpro Model Type: XMI Model
  - Developer models backend interface layer (EJB or others) in UML
  - UML is exported in XMI format and imported into Web Dynpro tools
  - Web Dynpro tools generate model layer and bind the frontend to it
- Development according to J2EE 1.3 standards
  - JSP, Servlets, etc. possible - but not recommended internally!
  - All types of Enterprise Java Beans (CMP or BMP) available
- Java Persistence Layer
  - Open SQL, including performance features (Tracing, Caching, ...)
  - Note: separate schemas / repositories for ABAP and Java!



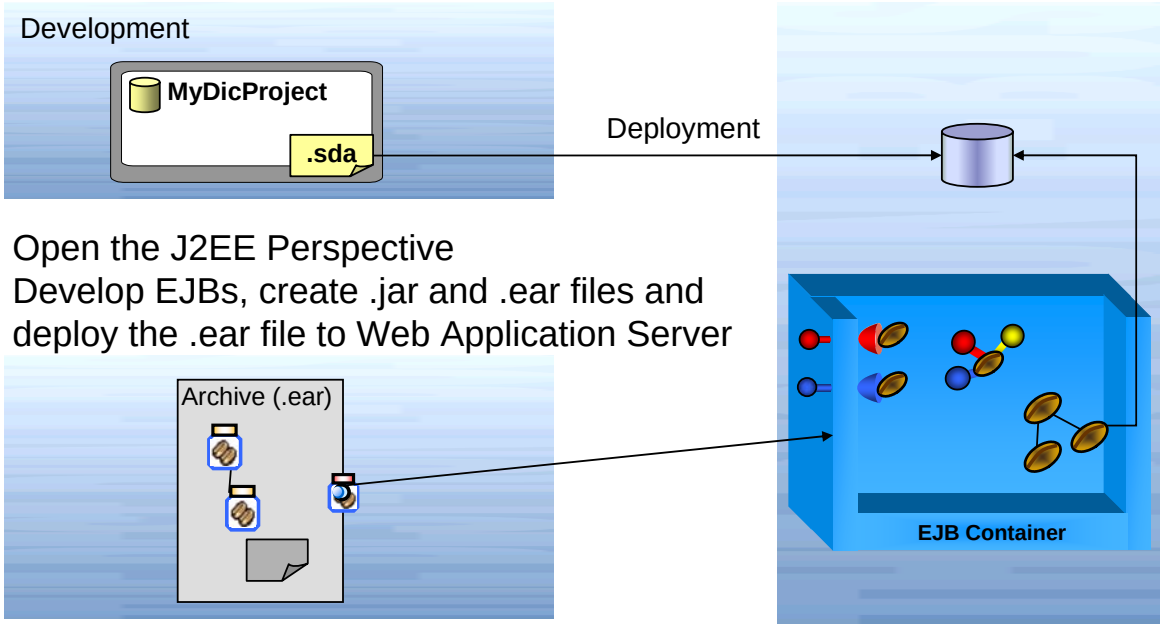
- **Web Dynpro Model Type: XMI Model**
  Web Dynpro technology allows you to use external data imported from an (XMI) model from external modeling tools; the source file must have the extension .xmi or .xml. The model tools also provide comprehensive functions and a wizard for importing these files. The model tools also provide support when displaying and changing imported model classes.

  Note: This feature not shown in the course. If you want to learn more about it, please refer to the documentation.

  In the course material an alternative is shown: Binding an existing EJB application to Web Dynpro using JNDI.

Step 1: Deploy Dictionary Tables and EJBs

- Open the Dictionary Perspective
  Define tables and data types, create a Dictionary Archive and deploy it to the Web Application Server

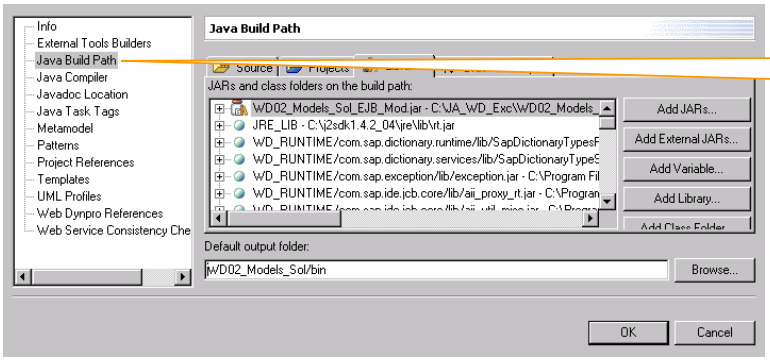


- Open the J2EE Perspective
  Develop EJBs, create .jar and .ear files and deploy the .ear file to Web Application Server

■ **Step 1: Prerequisits**
If you want to use Enterprise Java Beans as model for Web Dynpro, you can use the tools of SAP NetWeaver Developer Studio to define Enterprise Java Beans and platform independent database objects. After development you have to put the objects in archives and deploy them to the Web Application Server.

Step 2: Customize Web Dynpro Project

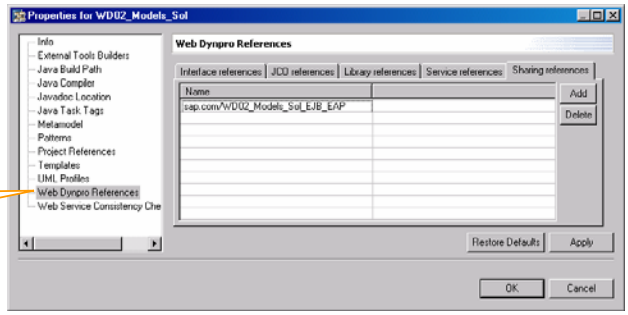
- Add the EJB .jar-file to the Web Dynpro project.



Java Build Path
Tab: *Libraries*

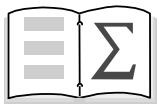
- Define the Sharing Reference of your Web Dynpro Project

Web Dynpro References
Tab: *Sharing References*



- **Java Build Path**
  In the Properties wizard of the Web Dynpro project, choose *Java Build Path* and *Libraries* tab.
  Make sure, that the classpath contains the necessary .jar-files.
- **Sharing References**
  In the *Properties* wizard, choose the *Web Dynpro References* and *Sharing references* tab. Add a new Sharing reference with the following syntax<vendor name>/<name of the ear file without extension>In our case this would be *sap.com/WD02_Models_Sol_EJB_EAP*
  Note: This is the name under which the EJB application is stored on the Web Application server. You find an entry in the Visual Administrator tool under*Services/Deploy/EJBContainer*.

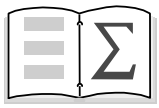
Use Enterprise Java Beans: Topic Summary



You should now be able to:

- **Customize a Web Dynpro project to use EJBs.**

Using EJBs as Web Dynpro Model: Summary



You should now be able to:

- **Explain Enterprise Java Beans.**
- **Use EJBs as Web Dynpro Model.**