DEVELOPING ENTERPRISE APPLICATIONS USING EJB

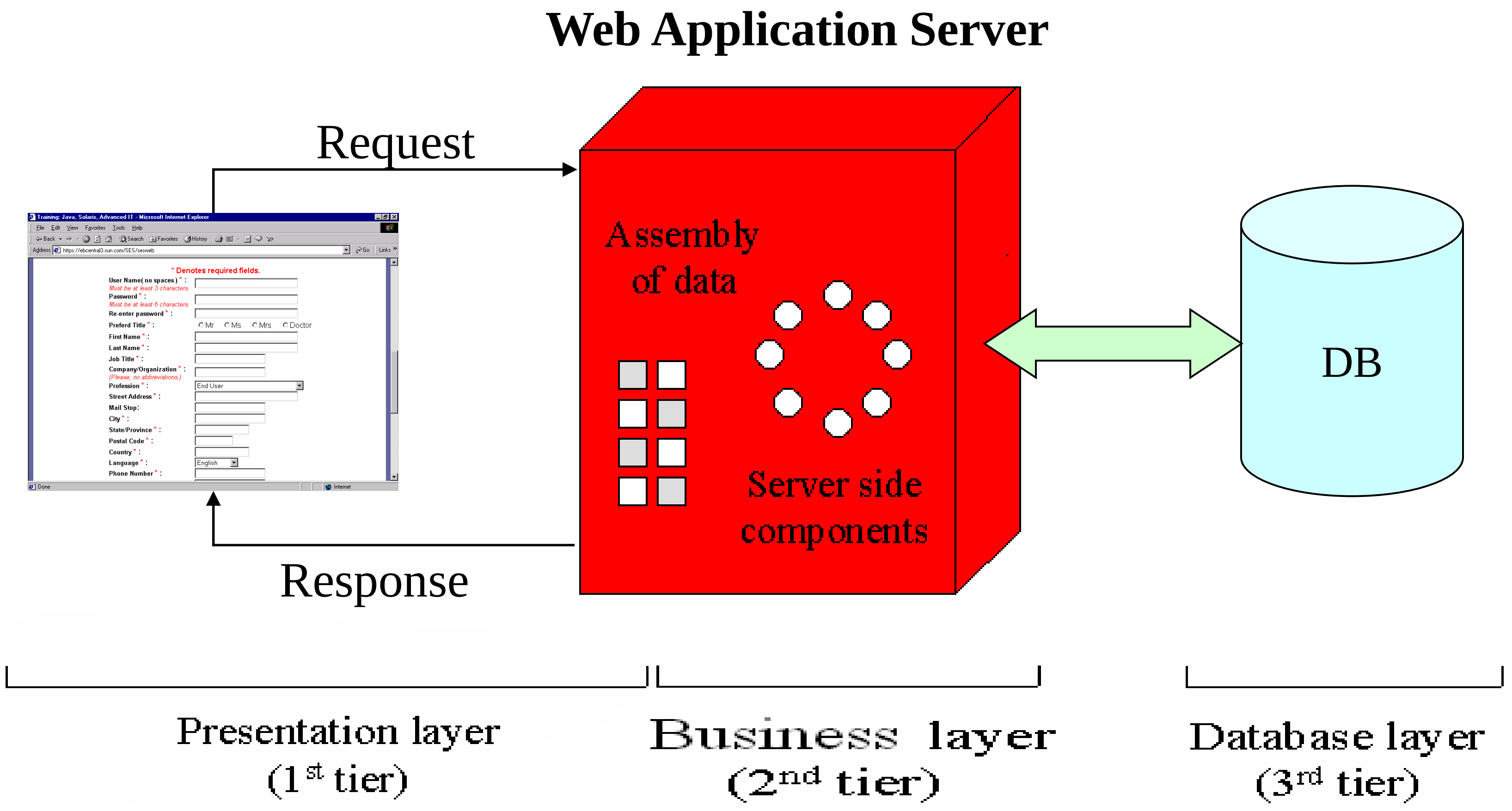
SESSION BEAN

CONTENTS

COMPONENT ARCHITECTURE

- Components are building blocks of an application
- Provides a set of services or functions, such that it can easily interact with other applications or components
- Consists mainly of Web components (JSP, Servlet, …), Business components (EJB), and Service components (Mail, JDBC, JMS …).
- Flexible, Portability and Reusable
- An enterprise application is usually composed of a three-layer architecture
  - Presentation Layer (Web Component, GUI Component, Client console)
  - Business Layer (Business Component)
    - Business logic
      - Comprises business rules or methods using which specific business functions can be managed
      - Refers to the workflow or the ordered task of passing data from one software sub-system to another
    - Business objects
      - Are the set of objects and the relationships between them
      - Encapsulate both the data & business behavior associated with the entity that it represents
      - Have the required features: reusability, access control, remote access, multi-user, highly available, state maintenance, transactional, and shared data
  - Data Layer

DISTRIBUTED OBJECT ARCHITECTURE


Web Application Server
Request
Response
Assembly of data
Server side components
DB
Training: Java, Solaris, Advanced IT - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address https://ebcentral3.sun.com/SES/sesweb
Go Links
Done
Internet
Presentation layer (1st tier)
Business layer (2nd tier)
Database layer (3rd tier)

* Denotes required fields.

User Name(no spaces) * : *Must be at least 3 characters.*	
Password * : *Must be at least 6 characters.*	
Re-enter password * :	
Preferd Title * :	○ Mr ○ Ms ○ Mrs ○ Doctor
First Name * :	
Last Name * :	
Job Title * :	
Company/Organization * : *(Please, no abbreviations.)*	
Profession * :	End User
Street Address * :	
Mail Stop:	
City * :	
State/Province * :	
Postal Code * :	
Country * :	
Language * :	English
Phone Number * :	

EVOLUTION of EJB

- Building application software is complexity
  - The software can process multi-data
  - The software is available online.
  - The developer worries about the security, transaction, scalability, concurrency, resource management, persistent, error handling, and many more system level problems.
- Software assurance and performance are affected because the developer can not concentrate fully on the developing the business logic (from implementation logic).
- EJB was developed so that it would:
  - Specialize in handling the business logic of an application
  - Be robust
  - Be secure so that it cannot be tampered.
  - EJB provides a component to create middleware which is deployed on Application Server (3 tiers architecture).
  - EJB Component has been designed to encapsulate business logic.

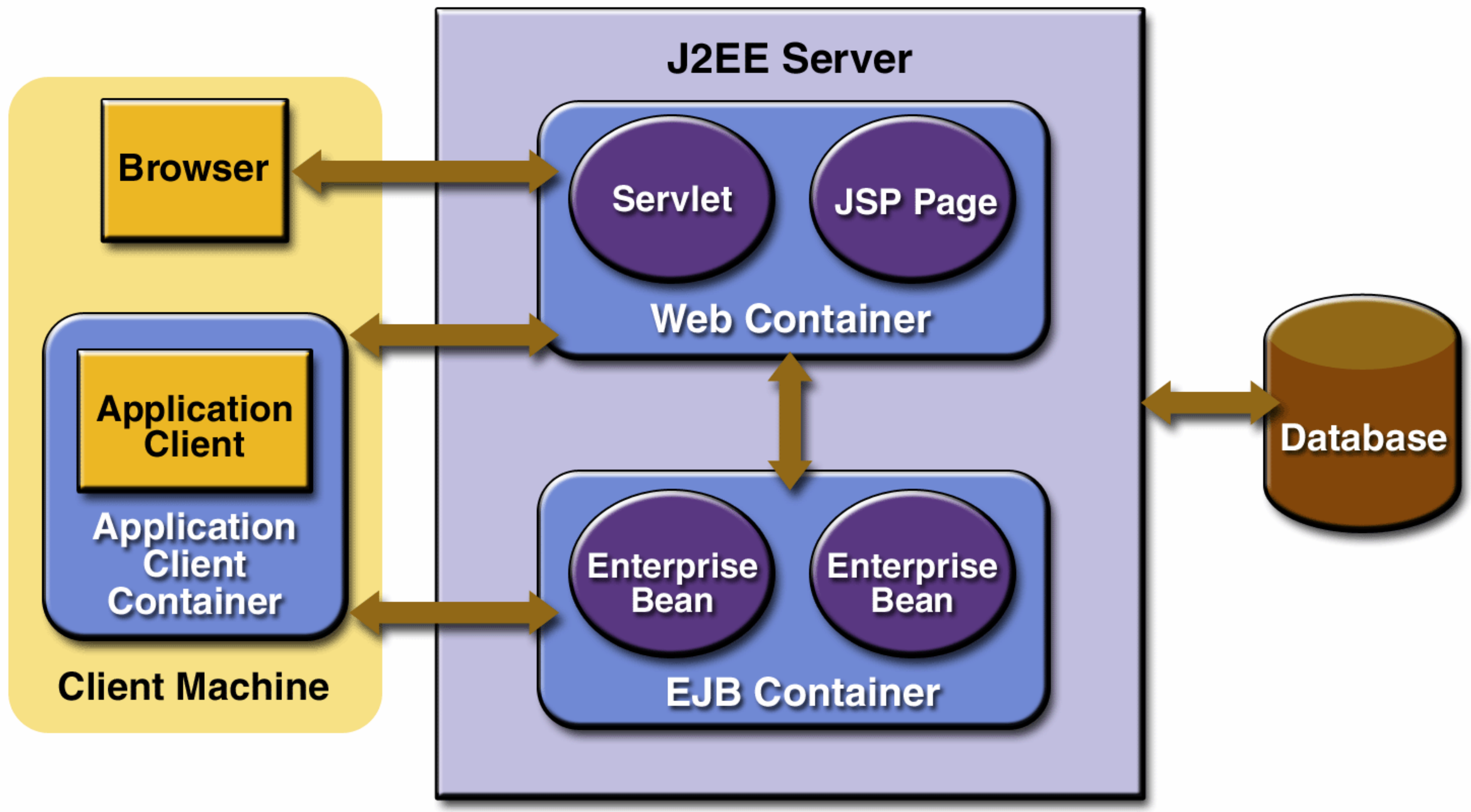
EJB

- Released by Sun Microsystems in 1998
  - EJB 1.1, final release (1999-12-17)
  - EJB 2.0, final release (2001-08-22)
  - EJB 2.1, final release (2003-11-24)
  - EJB 3.0, final release (2006-05-11)
  - EJB 3.1, final release (2009-12-10)
- EJB is a server-side component that simplifies the process of building enterprise-level & distributed applications in java
- Write-once, run-anywhere, middle-tier components which consists of methods that implements the business logic
- EJBs are interprocess components and Java Beans are intraprocess components
- There are two more ways of looking at EJBs
  - EJBs are specification: lay out rules and standards on how you should code your EJBs.
  - EJBs are Java interfaces: EJBs are code by the extending/implementing the EJB interfaces available in the J2EE package

EJB (cont)

- Characteristics
  - Hides server-side system level issues from developers.
  - Defines a standard component architecture that enables you to build distributed object-oriented business applications.
  - Facilitates creating enterprise-level applications by allowing easy integrating with other EJB components as well as with other components, such as servlets, Java Server Pages (JSP), and Java Beans.
  - Enables you to create components and applications that are reusable across various J2EE-compliant servers
- Types
  - Session Bean – Represents business process without having persistent storage mechanism
    - Stateless Session Bean
    - Stateful Session Bean
  - Entity Bean – Persists across multiple sessions and multiple clients & Having persistence storage mechanism
    - Bean-managed Persistence [BMP]
    - Container-managed Persistence [CMP]
  - Message-driven Bean – Asynchronous messaging between components of EJB.

EJB IN J2EE ARCHITECTURE

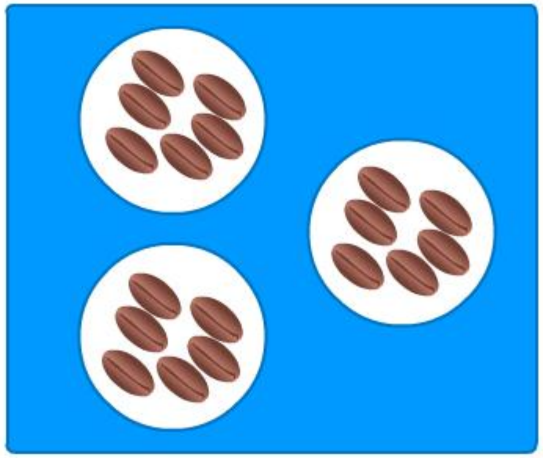


APPLICATION/EJB SERVER

- Provides many services
  - **Network connectivity** to the container
  - **Instance Passivation** – Temporarily swap out a bean from memory storage
  - **Instance Pooling** – Multiple clients share same instance
  - **Database Connection Pooling** – Contains a set of database connection
  - **Precached Instances** – Maintains cache, which contains information about the state of the EJB
- Other services
  - Runtime Environment
  - Support the containers interaction
  - Process and Thread Management
  - Receive and process requests
  - System Resource Management

EJB CONTAINER

- Acts as an interface between an enterprise bean and client
- Provides following services
  - Security
  - Transaction Management
  - Persistence
  - Life Cycle management
  - Remote Client Connectivity



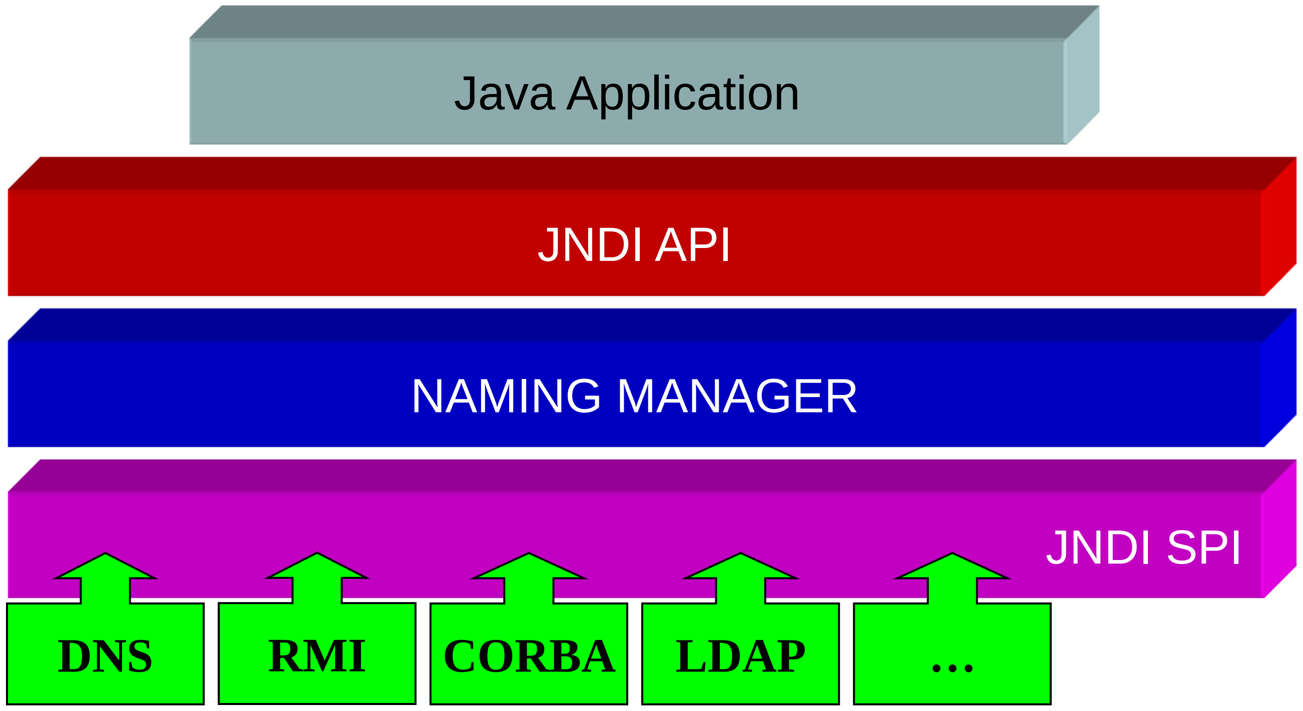
EJB Container with Bean pools

- Responsible for providing several APIs
  - J2SE API
  - EJB Standard Extension
  - JDBC Standard Extension
  - JNDI Standard Extension
  - JMS Standard Extension
  - JavaMail Standard Extension (for Sending mail only)
  - JAXP (Java API for XML Processing)
  - JTA Standard Extension (Only UserTransaction interface)

JAVA NAMING and DIRECTORY INTERFACE

- A naming service (which has its own set of rules for creating valid names) allows you finding an object in a system based on the name associated with the object which is called “binding”.
- A directory service is an extension of a naming service (an object is also associated with a name, which can be look up, and allowed to have attributes)
- Java Naming and Directory Interface (JNDI) is a specification for accessing naming and directory services for Java applications.
- Provides a standard interface to locate the components, users, networks, and services placed across the network.
- Bridges the gap between directory services and makes it possible for the developer to write portable naming and directory services
- JNDI abstracts the code from a dicretory service and allows the user to plug in a different directory services. (without changing the service code)
- JNDI provides javax.naming.* interface
- JNDI separate two parts
  - JNDI API (use access a variety of naming and directory services)
  - JNDI SPI (enables a variety of naming and directory services to be plugged in transparently, allowing the Java application using the JNDI API to access their services)
- Naming Concepts of JNDI
  - Atomic: It’s a simple and basic name. Ex: Windows
  - Compound: the collection of one or more atomic names. Ex: C:\Windows\System32
  - Composite: A name has multiple naming system. Ex: http://localhost:8080/JSP/index.html

JNDI ARCHITECTURE



Accessor Naming Directory services through APIs

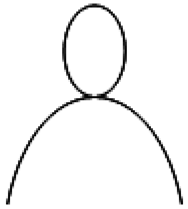
allows Naming and Directory vendors can plugin transparency

JNDI API & LIBRARIES

- The **Context** is represented by the **javax.naming.Context interface** that has the necessary methods to put objects into the naming service, and also to locate them.
- The starting point is called an **InitialContext**, represented by **javax.naming.InitialContext interface**
- The references in JNDI are represented by **javax.naming.Reference interface**
  - **The lookup() method** retrieves the object bound to the name and throws a **javax.naming.NamingException**, if a naming exception is encountered

    <Biến context>.lookup(object name)
- The remote calls in EJB make use of **RMI-IIOP** (Remote Method Invocation-Internet Inter-Orb Protocol) which does not support explicit casting of the EJB object obtained from the remote object to a local object. Instead, Java a RMI-IIOP provides a mechanism to narrow the Object you have received from your lookup to the appropriate type by using the javax.rmi.PortableRemoteObject class & its ***narrow()*** method
  - The method narrow() of which parameters narrowFrom is the object that has to be narrowed and narrowTo is the desired type. It returns the object which is cast to the desired type and throws ClassCastException, if narrowFrom cannot be cast to narrowTo
- The supported files are **jndi.jar, fscontext.jar, providerutil.jar**

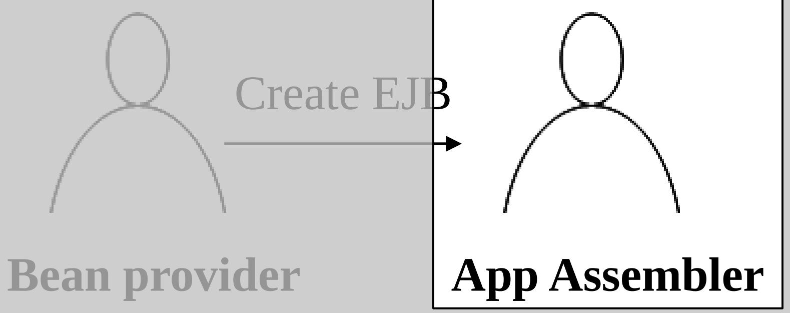
PARTIES INVOLVED in EJB DEPLOYMENT



Bean provider

- Provide the components to solve business problem (that are packaged them to the ejb-jar file
- Reusable components
- Assemble other components into application.
- Distribution

PARTIES INVOLVED in EJB DEPLOYMENT



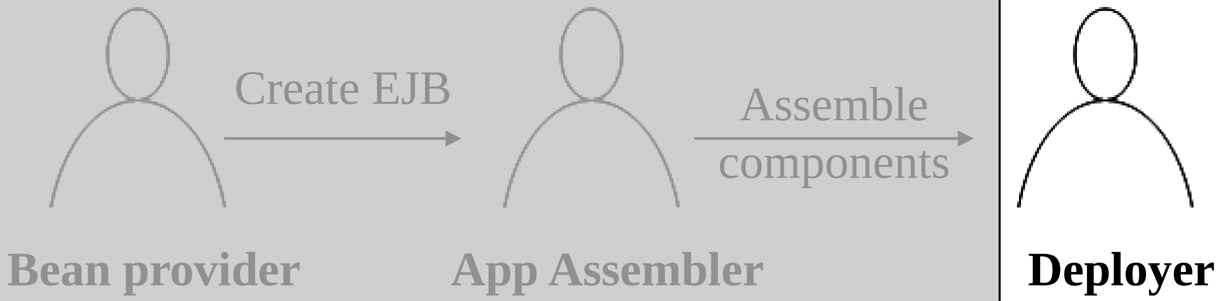
•For Assembling different EJB Components in order to build a complete application

•Analyzing a business problem and assembling EJB components accordingly to solve the problem

•Building new EJB components

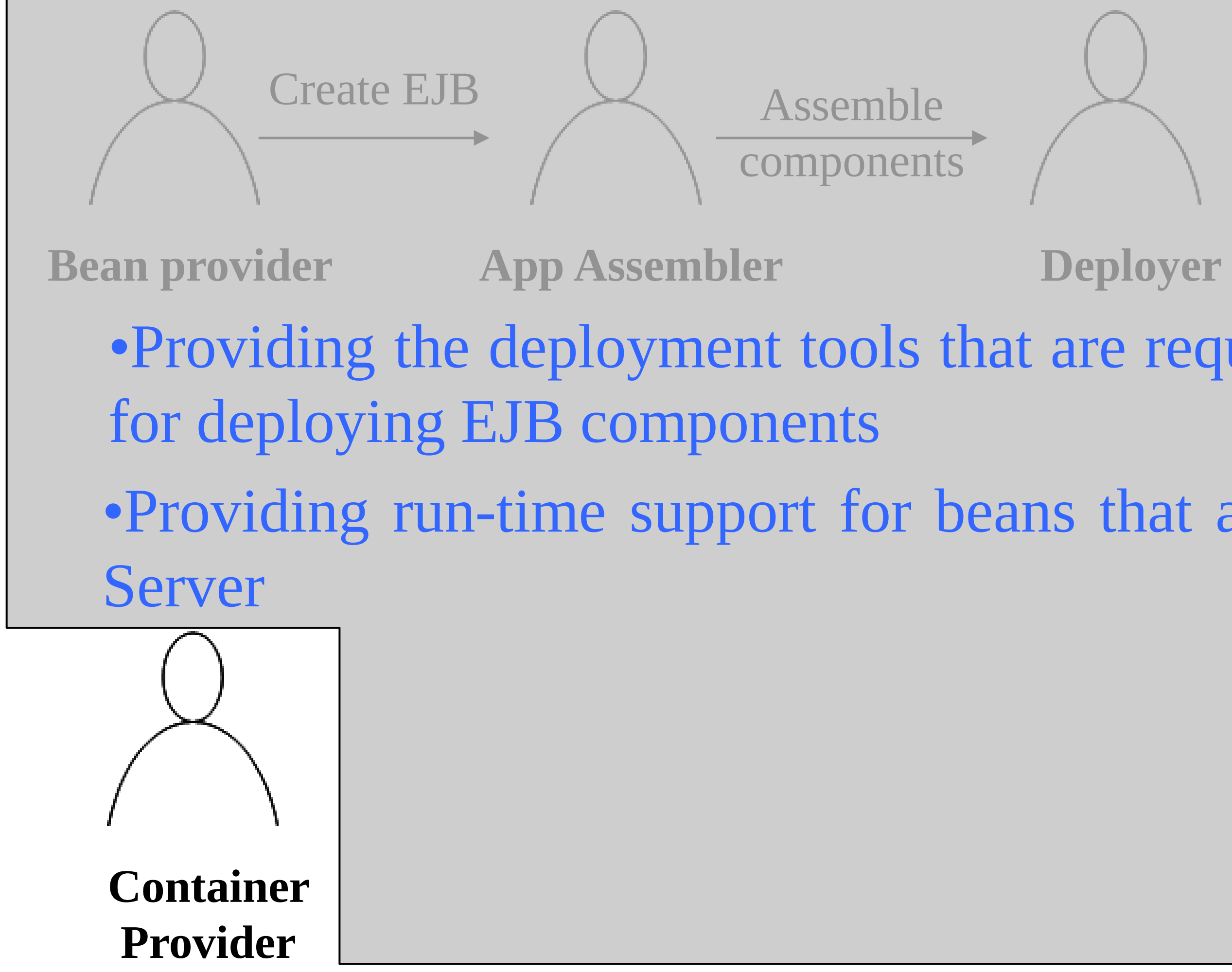
•Writing the integration code required to associate the EJB components build by different bean providers

PARTIES INVOLVED in EJB DEPLOYMENT



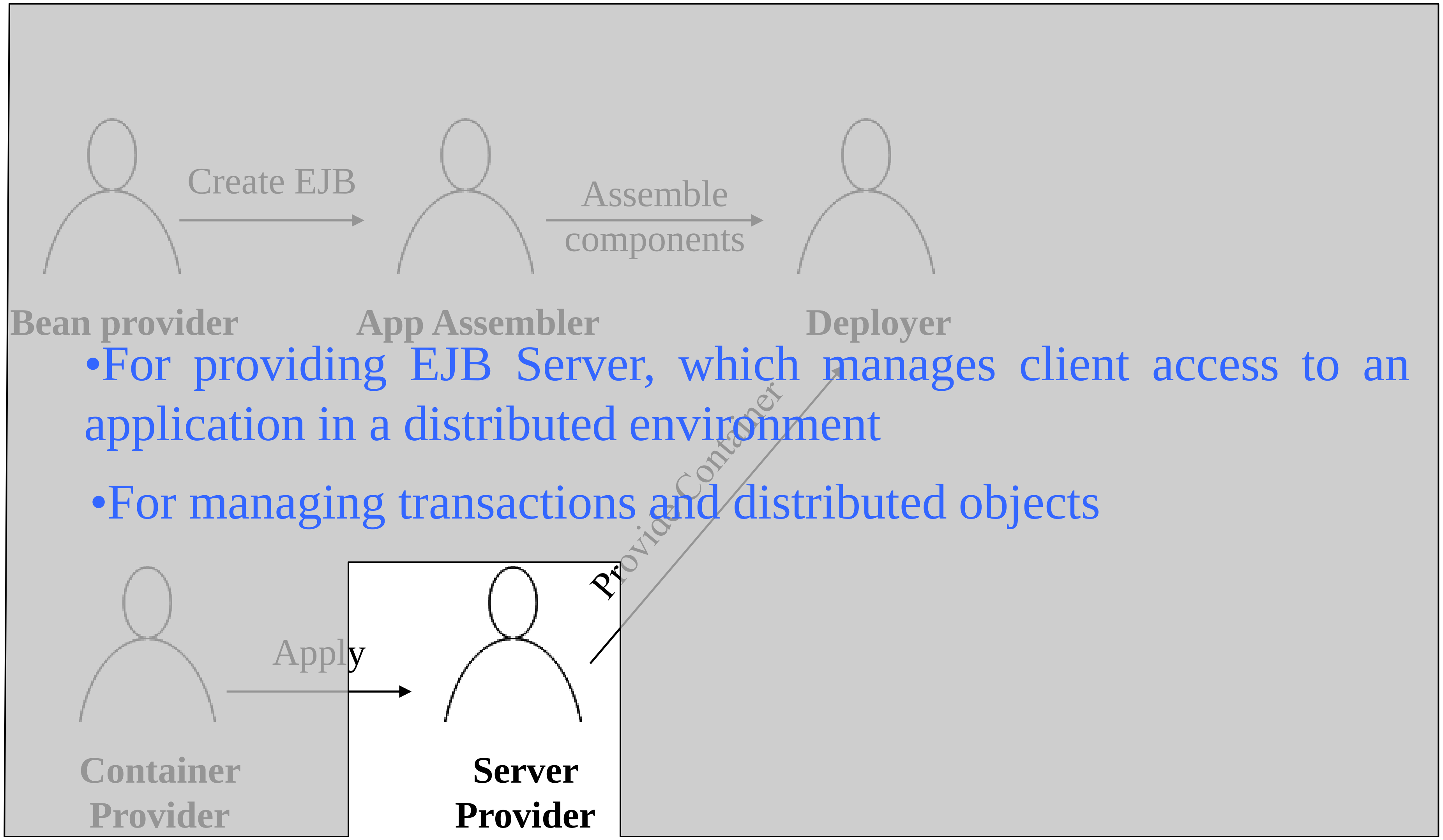
- Customizing enterprise bean
- Accumulate information about operational requirements such as security, hardware, and transaction before deploying the bean
- For deploying an assembled application in an application server

PARTIES INVOLVED in EJB DEPLOYMENT



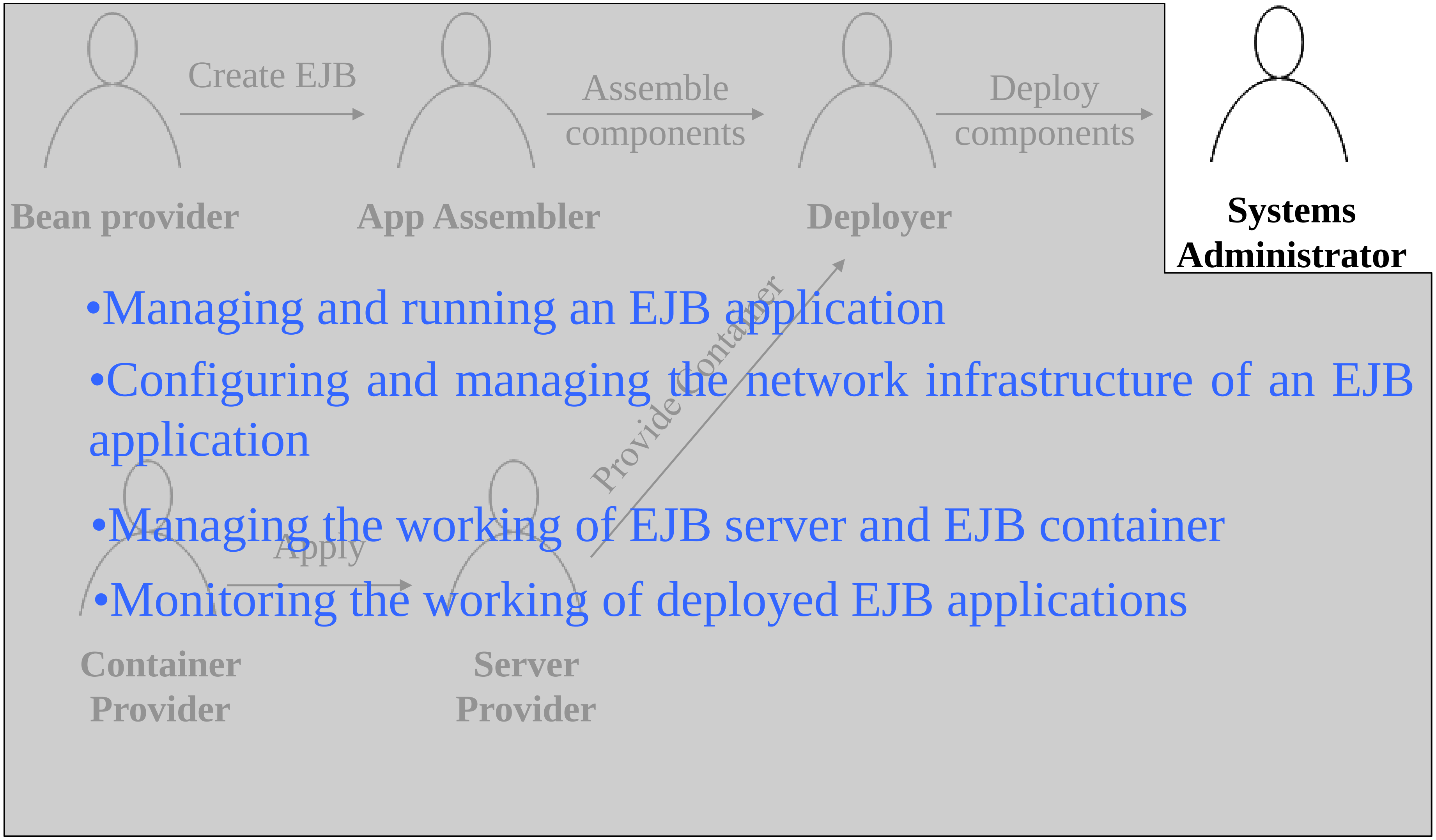
- Providing the deployment tools that are required by the deployer for deploying EJB components
- Providing run-time support for beans that are deployed on EJB Server

PARTIES INVOLVED in EJB DEPLOYMENT



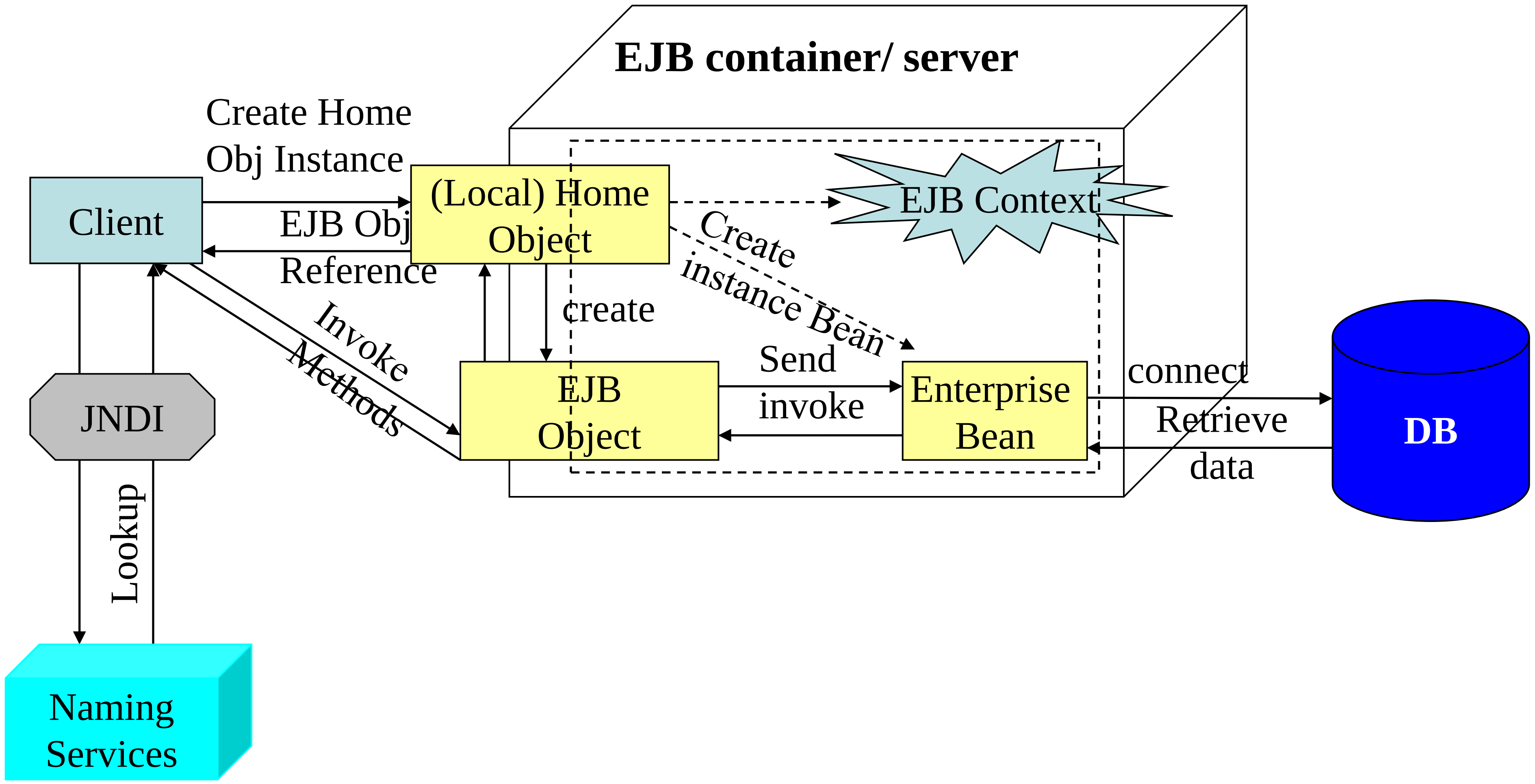
- For providing EJB Server, which manages client access to an application in a distributed environment
- For managing transactions and distributed objects

PARTIES INVOLVED in EJB DEPLOYMENT



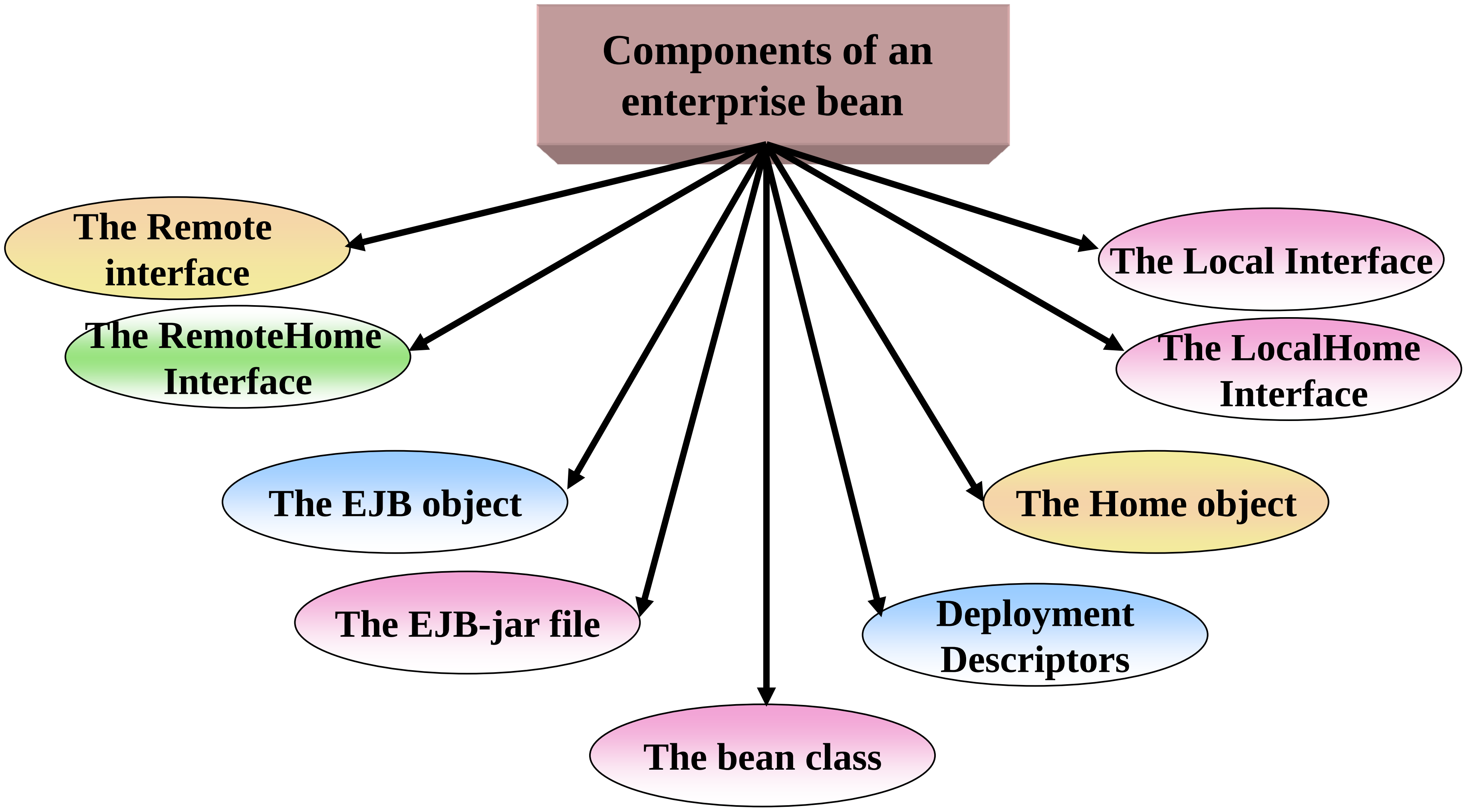
- Managing and running an EJB application
- Configuring and managing the network infrastructure of an EJB application
- Managing the working of EJB server and EJB container
- Monitoring the working of deployed EJB applications

LOGICAL ARCHITECTURE of EJB



03 tiers Architecture

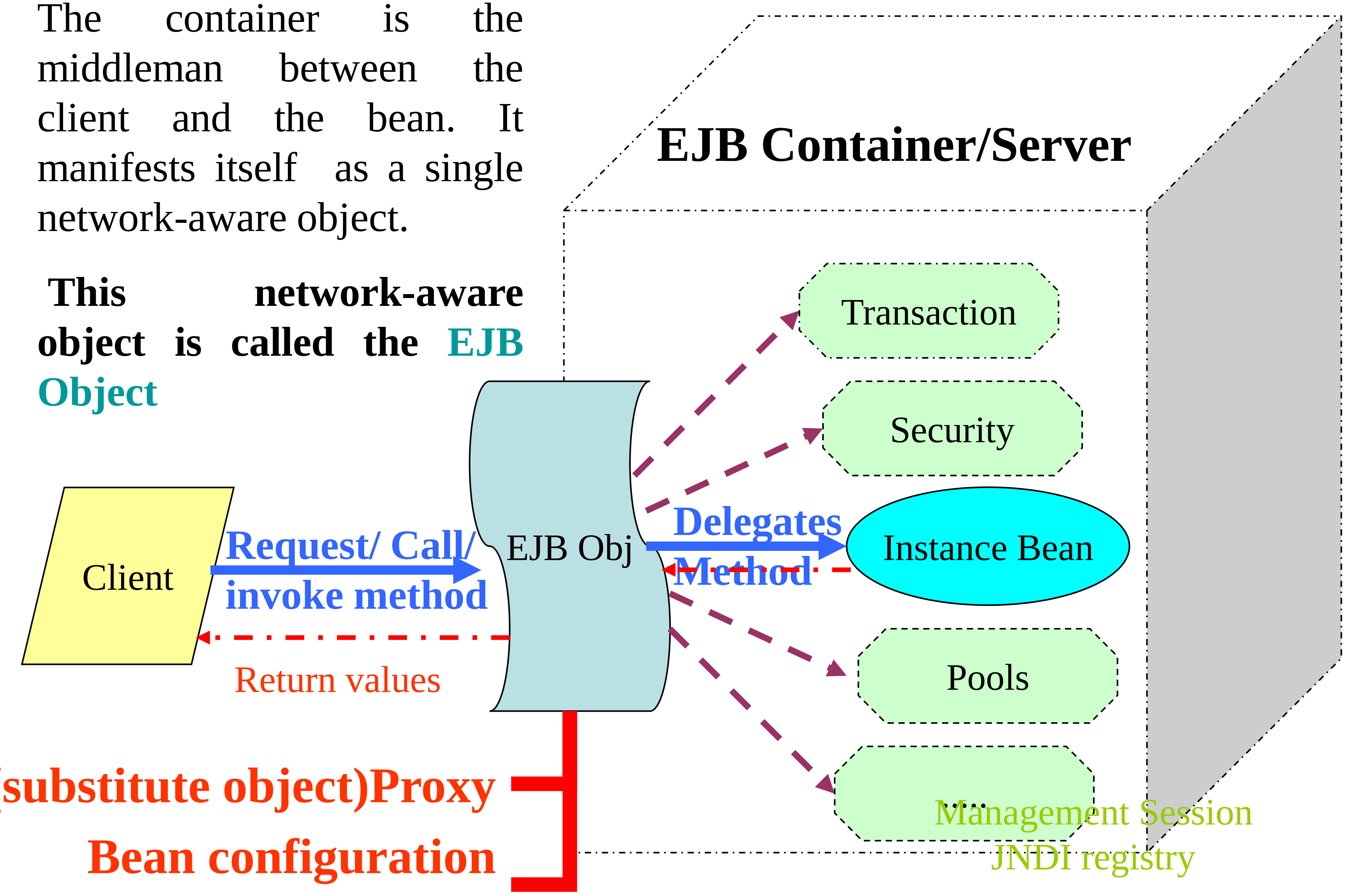
COMPONENTS OF EJB



EJB OBJECT

The container is the middleman between the client and the bean. It manifests itself as a single network-aware object.

This network-aware object is called the EJB Object



(substitute object)Proxy

Bean configuration to use

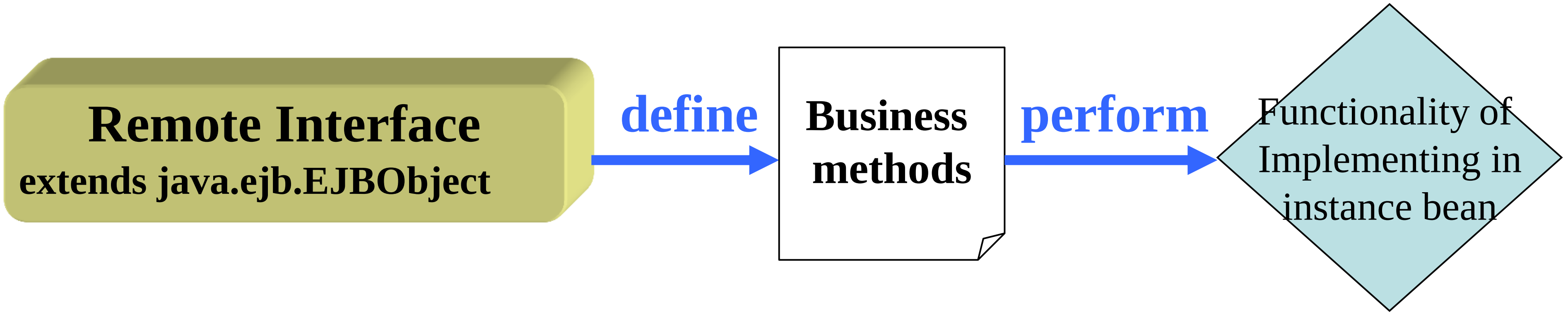
EJB OBJECT (con't)

- Interface **javax.ejb.EJBObject**

Methods	**Descriptions**
getEJBHome()	Retrieves the reference to the corresponding Home Object
getPrimaryKey()	Return the PrimaryKey for EJB Object (Entity Bean)
remove()	Destroy EJB Object (delete the bean from the underlying persistent store, means delete a record on DB – Entity Bean)
getHandle()	Obtain the handle (is a persistent reference to the EJB Object) for the EJB Object
isIdentical()	Checks whether two EJB Objects are similar

- Relationship between Java RMI and EJB Objects
  - public interface javax.ejb.EJBObject extends java.rmi.Remote (*The physical location of remote object is hidden from the Client RMI*)
  - Can be called from a different JVM
  - Offers Location Transparency (Portability of Client Code)

REMOTE INTERFACE


Remote Interface
extends java.ejb.EJBObject
define
Business methods
perform
Functionality of Implementing in instance bean

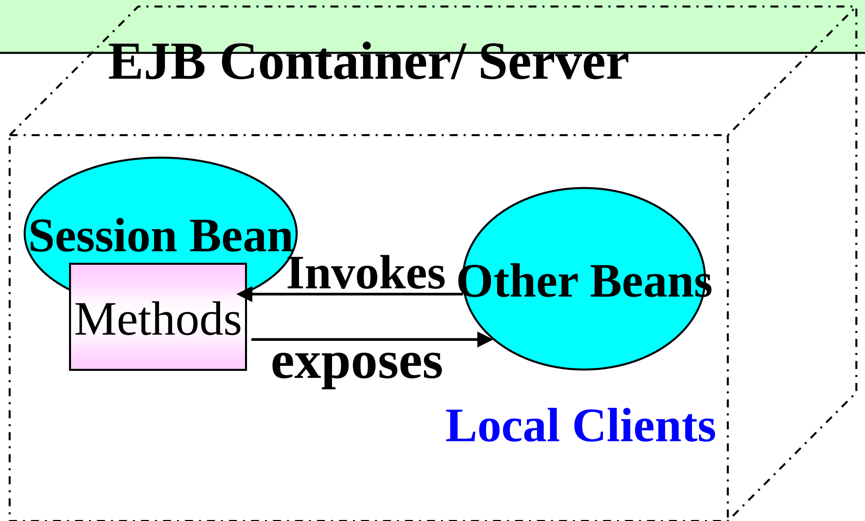
```
public interface Welcome extends javax.ejb.EJBObject

{

    public String welcome()  throws java.rmi.RemoteException;

    ...

}
```

•**Note**: System level operations such as persistence, security and concurrency are not included in remote interface.

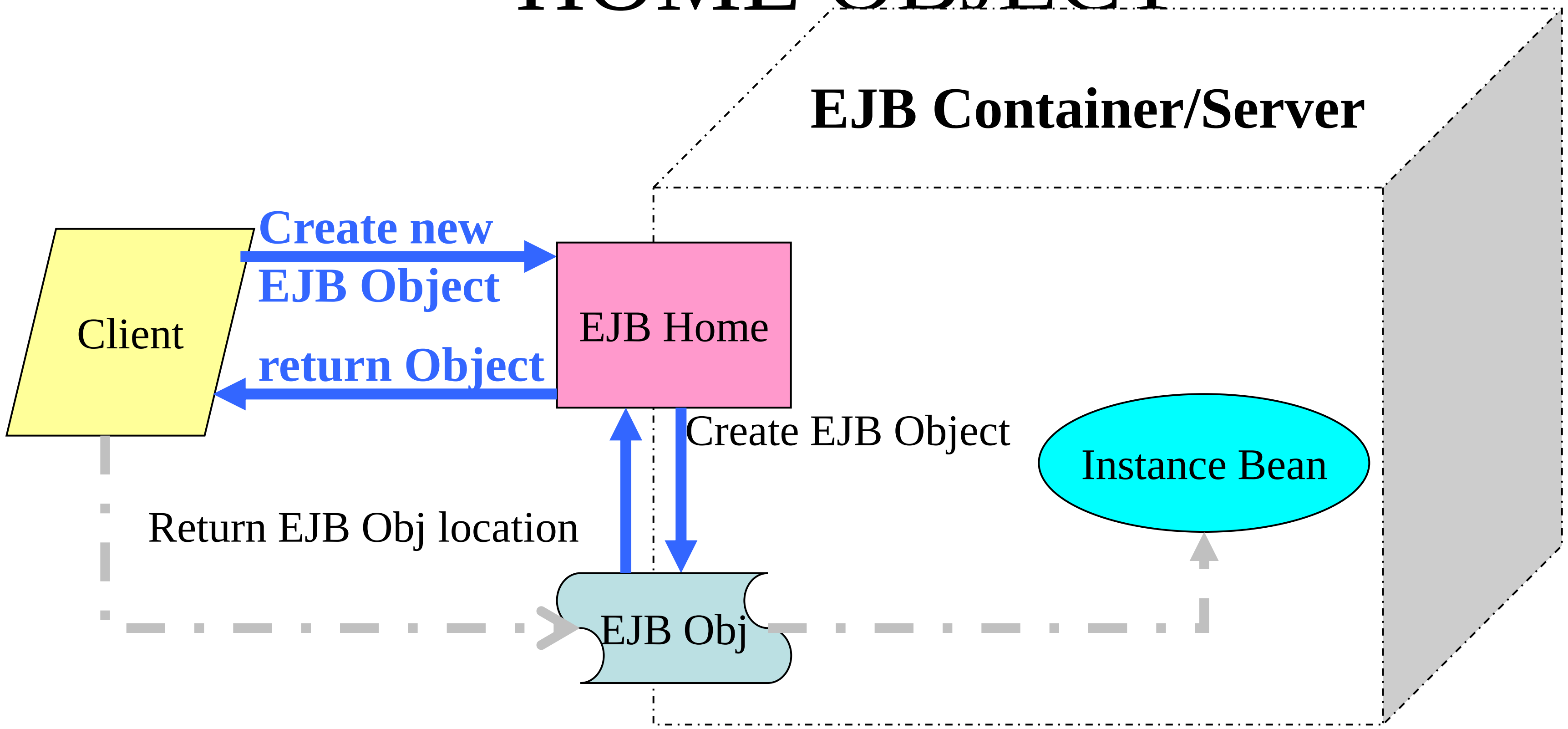
LOCAL INTERFACE

- EJB 2.0 can expose their methods to clients through new Local Interface
- Standard Java interface which allows the beans to expose its methods to other bean reside within the same container (local clients)
- Eleminate the overhead of the remote method call (java.rmi.RemoteException)
- Use pass by reference semantics (speed up in processing and efficiency)
- Not inherit from RMI (extends javax.ejb.EJBLocalObject)

```
public interface WelcomeLocal extends javax.ejb.EJBLocalObject
{
   public String welcome();
   ...
}
```



HOME OBJECT



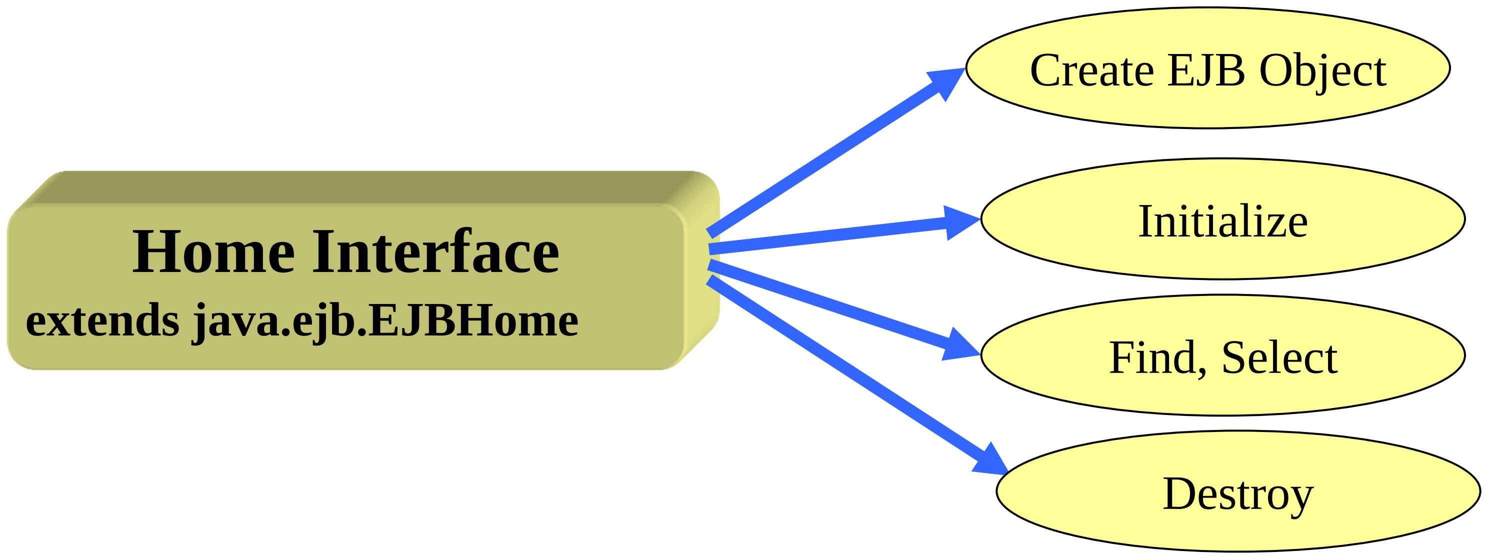
- Client code will request for an object from the EJB Object Factory, which know as the home object (instantiates EJB Object)
- Responsibilities
  - Create (Instantiate) EJB Objects
  - Initial information for EJB Object s
  - Find or search for existing EJB Objects(Entity Bean)
  - Remove EJB Objects (deletes the bean from the underlying persistent store)
  - Select EJB Objects (Entity Bean)

HOME OBJECT (con’t)

- Interface javax.ejb.EJBHome (extends java.rmi.Remote)

Methods	**Descriptions**
getEJBMetaData()	Retrieve information about EJB (Beans’ information) that are being worked on. The information received is encapsulated in the EJBMetaData object, which returns the method.
remove()	Destroy EJB Object following - Passing the javax.ejb.Handle object, which remove EJB Object that is based on the already retrieved EJB Handle - Passing a primary key to remove beans (one record) from the underlying persistent store.

HOME INTERFACE



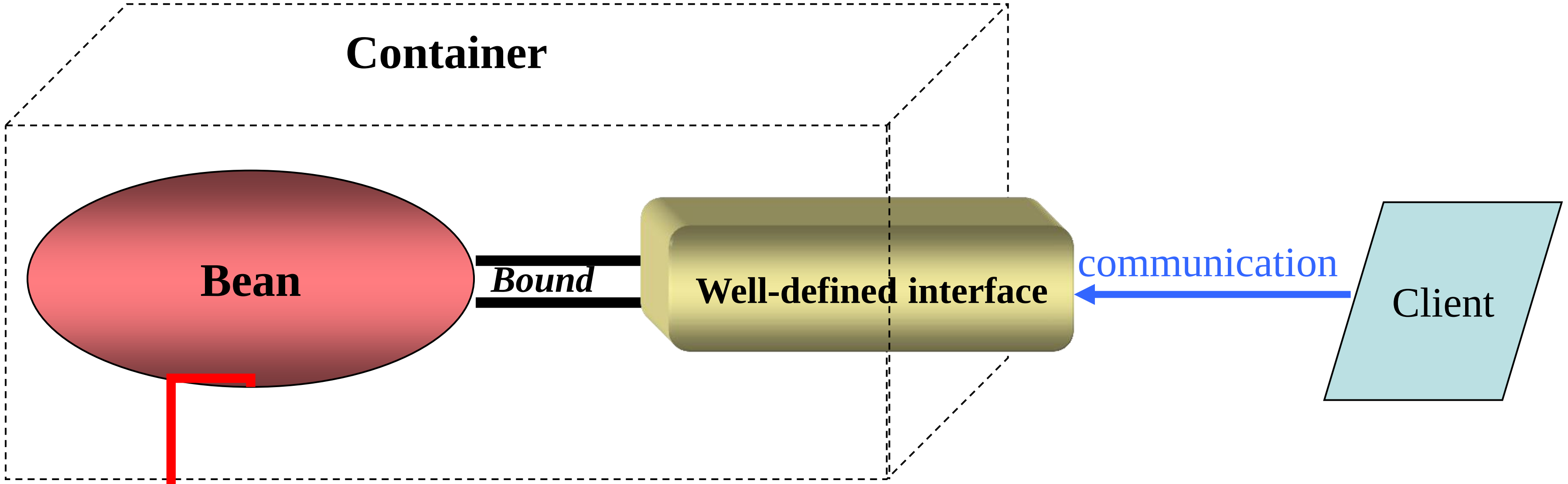
```
public interface WelcomeHome  extends java.ejb.EJBHome
{
  public Welcome   create()    throws java.rmi.RemoteException;
  public Welcome findByPrimaryKey() ...;
  ...
}
```

LOCAL HOME INTERFACE

- Standard Java interface which allows the beans to expose its methods to other bean reside within the same container (local clients)
- Eleminate the overhead of the remote method call (java.rmi.RemoteException)
- Use pass by reference semantics (speed up in processing and efficiency)
- extends javax.ejb.EJBLocalHome
- **Notes**: LocalObject is used as Return Values

```
public interface WelcomeLocalHome extends javax.ejb.EJBLocalHome
{
   public WelcomeLocal create();
   public WelcomeLocal findByPrimaryKey();
   ...
}
```

BEAN CLASS



Implements defined method from Component Interface (Remote and Local)

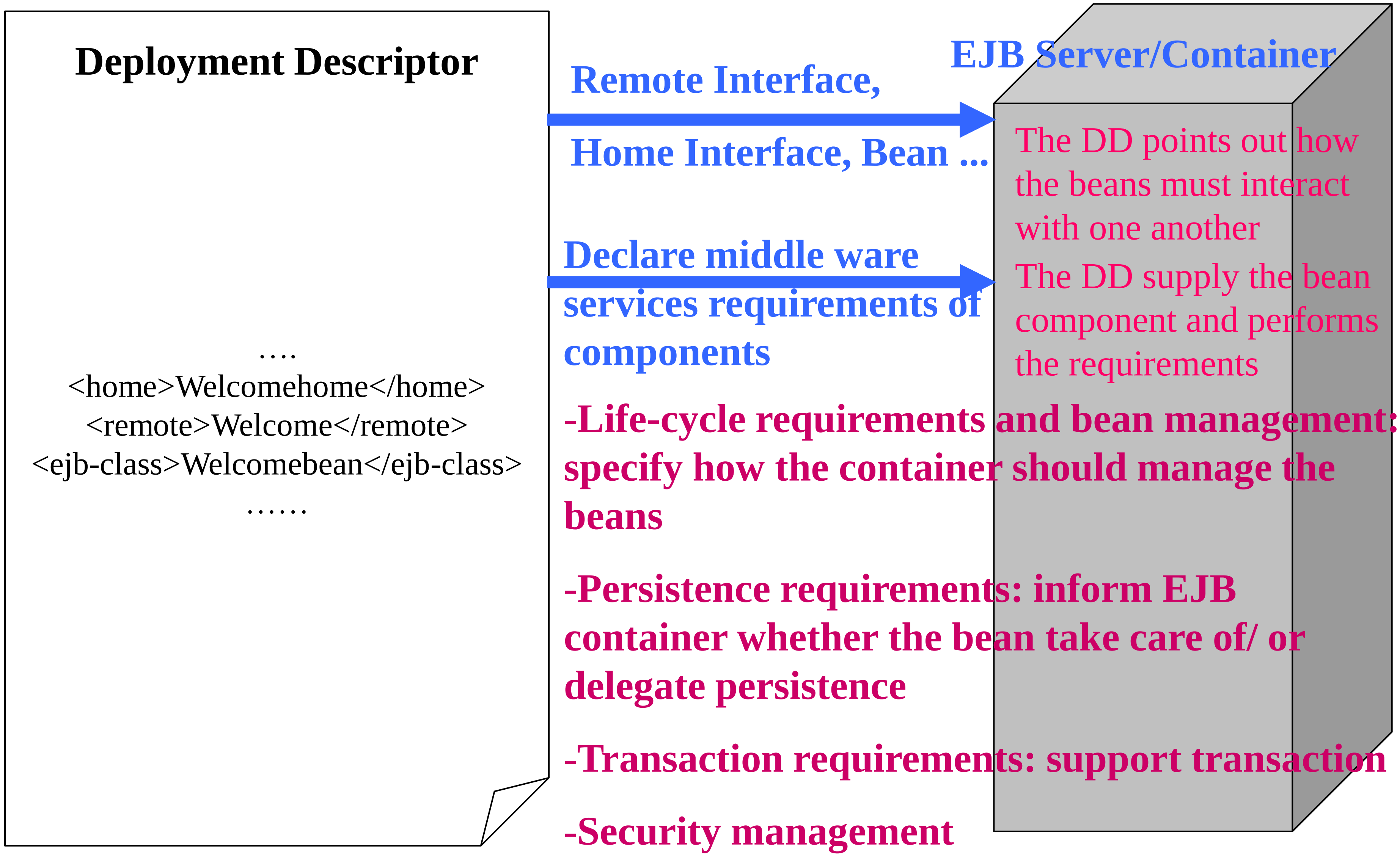
Override default Bean class

These methods are then called by container manage bean and keep the bean informed of important events

EJB can share the propertiess of the serialiable objects because the javax.ejb.EnterpriseBean extends Serializable

Once the interface javax.ejb.EnterpriseBean is implemented, the bean class is confirmed

DEPLOYMENT DESCRIPTOR



The DD points out how the beans must interact with one another

The DD supply the bean component and performs the requirements

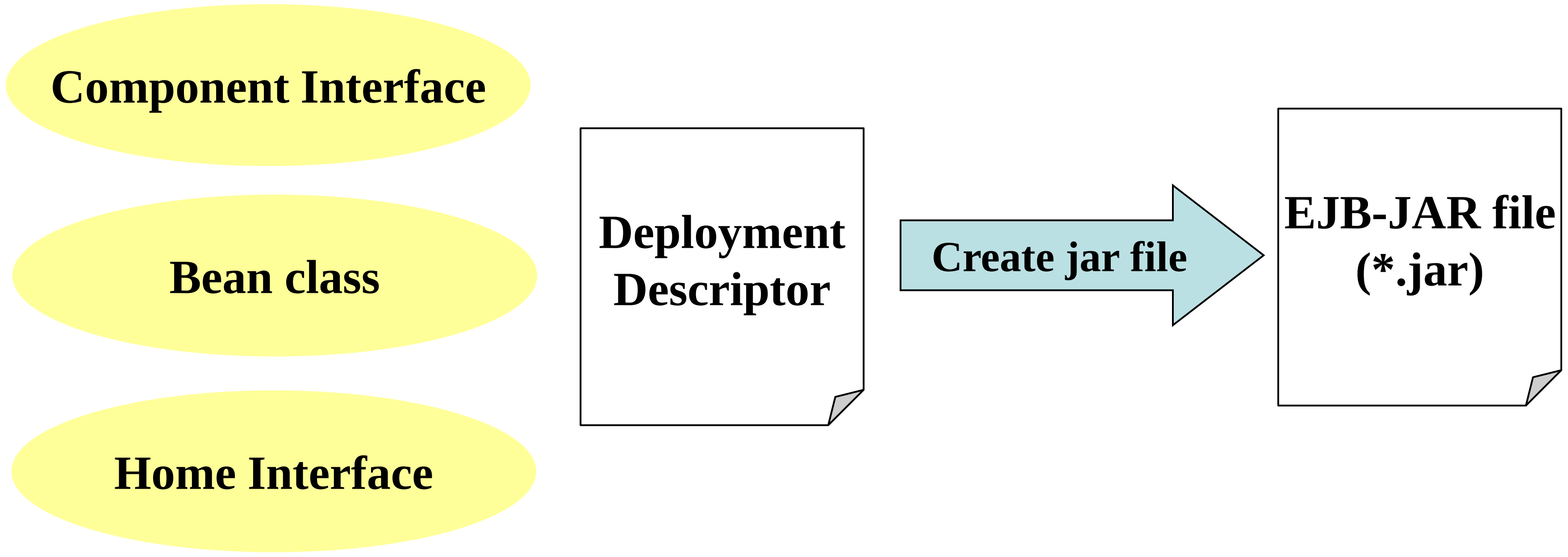
-Life-cycle requirements and bean management: specify how the container should manage the beans

-Persistence requirements: inform EJB container whether the bean take care of/ or delegate persistence

-Transaction requirements: support transaction

-Security management

EJB-JAR FILE



- EJB container decompress, read and extract information contained in the EJB-JAR file.
- Generation of the EJB object and the home objects, and the bean. (deployer)

SESSION BEANS

- A kind of enterprise beans that represent business processes (any task with logic, workflow, and algorithms).
- Perform business functions for the clients inside the application server
- Represents business process without having persistent storage mechanism
- Not permanent in Nature
- Are not shareable between clients (only one client can deal with that particular session bean)
- Enable a Conversation between a client and the J2EE Server
- Life Cycle of a Session Bean
  - A session bean may last as long as the client session.
  - Will not survive if the application server changes or crashes.
  - They are objects which are present in-memory which die along with the surrounding environment and are not persisted in a database.
- Types of Session Bean
  - Stateless Session Bean
  - Stateful Session Bean
- Session bean implement
  - From javax.ejb.Session Bean
  - callback methods (invoked on the bean by the container)
  - business methods

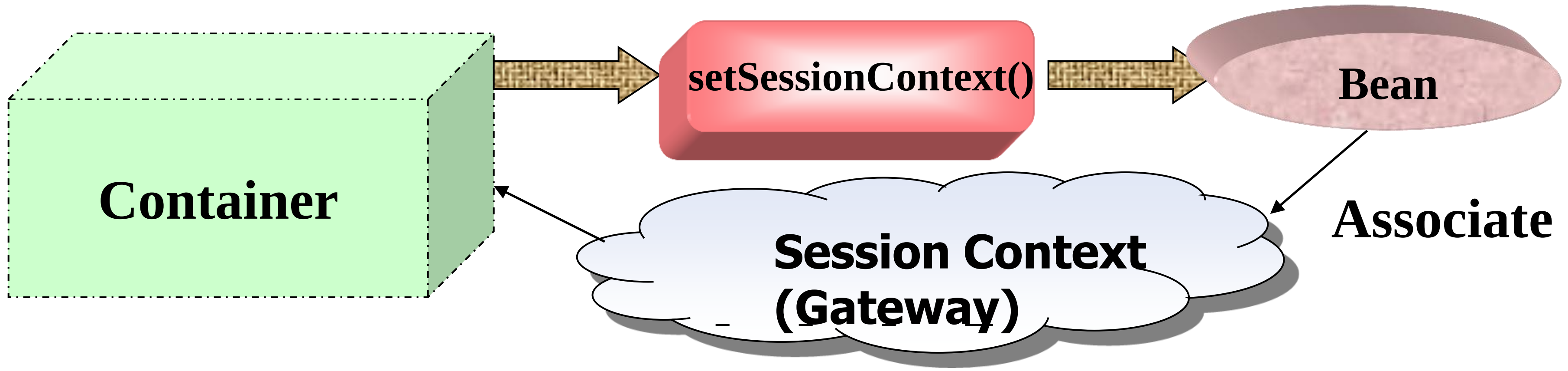
CONVERSATION & NON-CONVERSATION

- Conversation
  - Define as an interaction between a client and a bean.
  - Is composed of a number of method calls between them
  - Stretches across a business process with respect to the client.
  - Stateful session beans can retain their conversational state.
  - Ex: Shopping Cart
- Non-Conversation:
  - Clear of all previous information (same as the HTTP protocol)
  - A stateless session bean conducts a conversation that spreads over a single method call.

CALLBACK METHODS

- public void **setSessionContext**(SessionContext ctx)
- public void **ejbCreate**([args])
- public void **ejbPassivate**() (Least Recently Used)
- public void **ejbActivate**()
- public void **ejbRemove**()

EJB SESSION CONTEXT



- Contains information about bean's status such as reference of bean home interface, client's security permissions and transactions currently associated with the bean instance.
- The EJB session context object enables session beans to interact with EJB container to perform the following functions:
  - Retrieve the reference to the home objects
  - Retrieve transaction attributes
  - Set transaction attributes
- javax.ejb.SessionContext interface
  - setSessionContext()

ejb-jar.xml STRUCTURE

```
<?xml version="1.0"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">
    <enterprise-beans>
        <session>
          <ejb-name>Representation Name</ejb-name>
          <home>[package.]Home Interface class</home>
          <remote>[package.]Remote Interface class</remote>
          <ejb-class>[package.]Bean Class</ejb-class>
          [<local-home>[package.]Local Home interface class</local-home>]
          [<local>[package.]Local interface class</local>]
          <session-type>Stateless/Stateful</session-type>
          <transaction-type>Bean</transaction-type>
        </session>
    </enterprise-beans>
</ejb-jar>
```

jboss.xml STRUCTURE

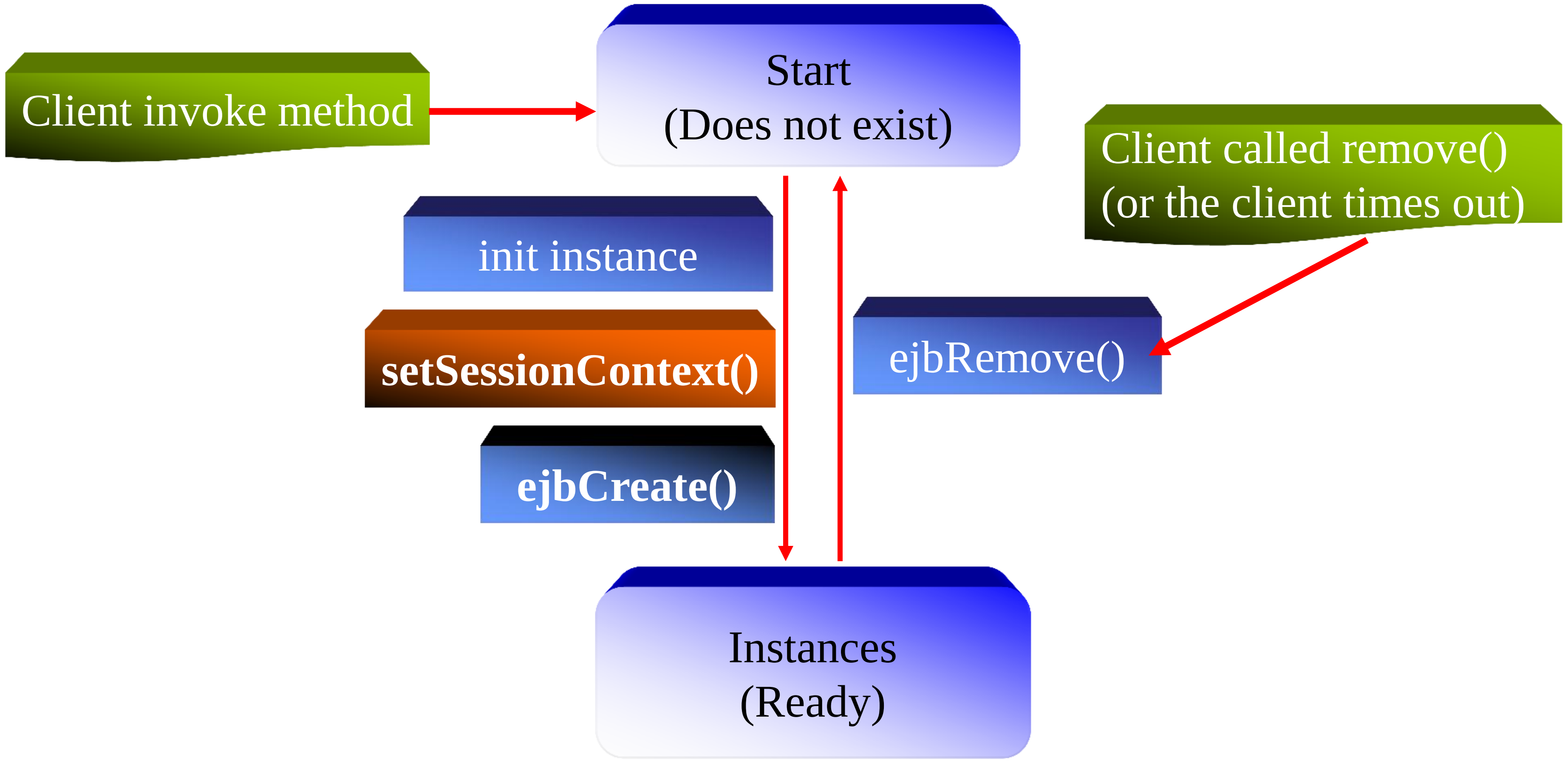
- Provides the container information about the JNDI mapping, persistence information and database mapping.

```
<?xml version="1.0"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
 "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">
<jboss>
     <enterprise-beans>
       <session>
             <ejb-name>Representation name</ejb-name>
             <jndi-name>name reference to home</jndi-name>
             <local-jndi-name>name reference to local home</local-jndi-name>
        </session>
     </enterprise-beans>
</jboss>
```

STATELESS SESSION BEAN

- Is used for a single request conversation
- Executes Business operations without maintaining the client state (non-conversation)
- Lightweight Component – Doesn’t contains complex data structure
- Same Session Bean instance can handle multiple client requests
- Improves the scalability of an application
- Not having instance variables to store information
- Can’t used when information need to be used various method calls
- The client has to pass on all the information necessary for the bean through the parameters for the business method.
- The client is not responsible for the creation or destruction of the bean
- Life cycle
  - Bean Creation: Stateless are created by the container and added to a bean pool.
  - Bean Use: Stateless are used when the clients call their business methods.
  - Bean Removal: Stateless are removed when the EJB container decides there are too many beans in the pool or the beans throw a system exception.

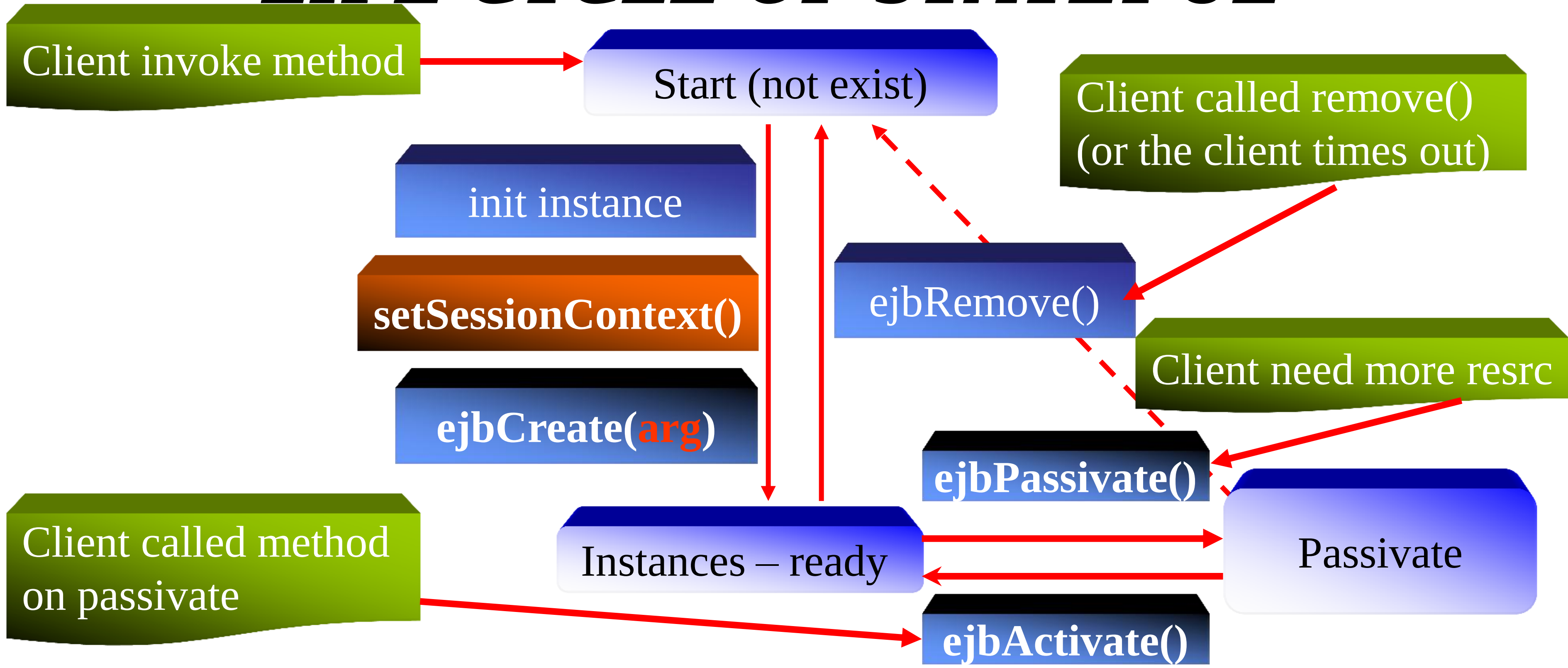
LIFE CYCLE OF STATELESS



STATEFUL SESSION BEAN

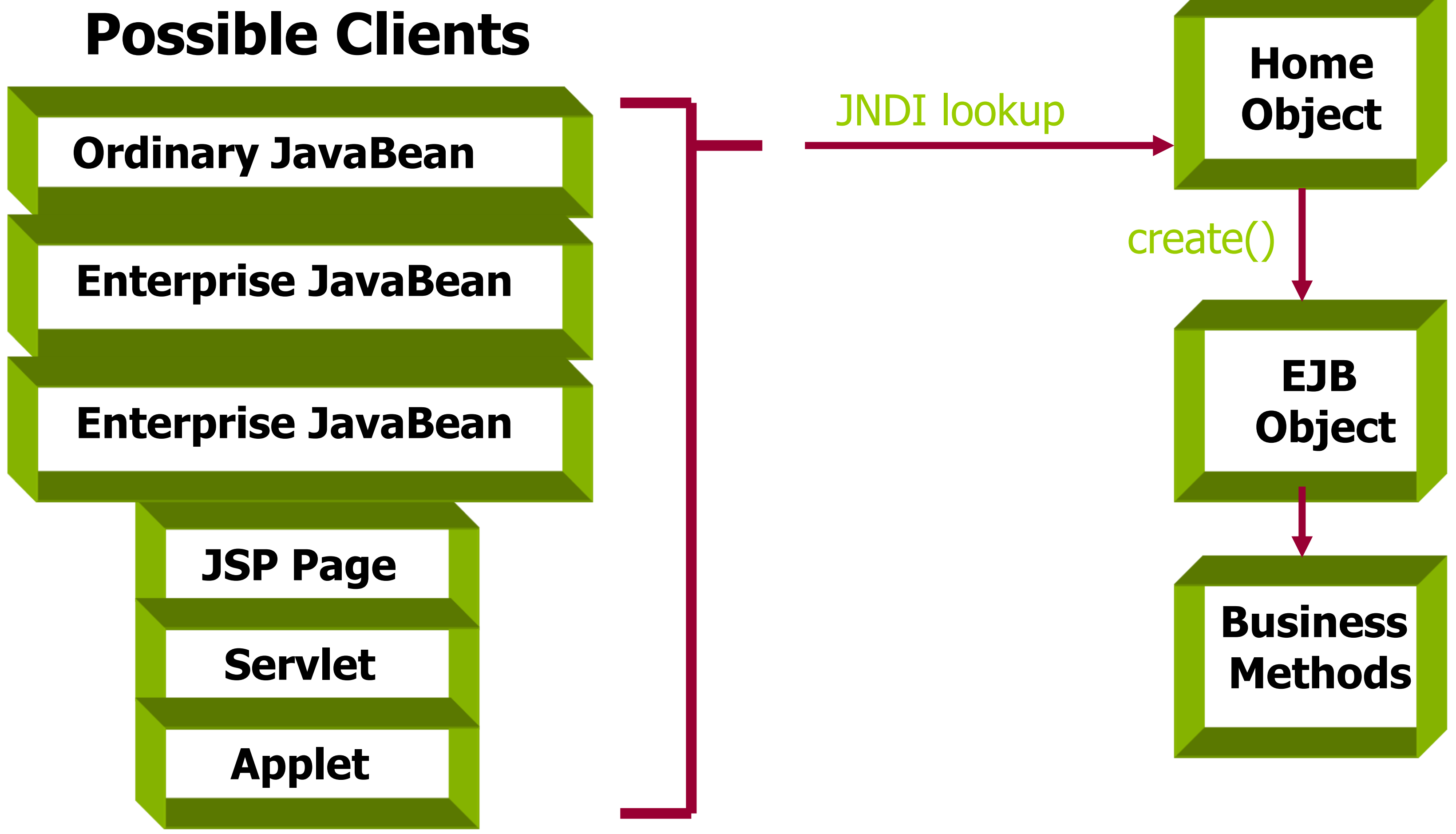
- Is used for business processes that span multiple method request or transactions have to be serviced
- Maintains the State of a client
- The conversional state must be stored in the bean (Stores client state in instance variable)
- Pooling has to be done to conserve resources and enhances scalability.
- The container swaps out a bean and saves the conversational state to the hard disk or other storage devices. This process is called passivation.
- To passivate a bean a container uses Least Recently Used (LRU) method.
- When the client requests for a method, the passivated conversational state is returned to the bean. The bean is again ready to meet the request. This process is called activation.
- To activate a bean a container uses Just-in-Time (JIT) method.
- Can used when information need to be used various methods calls
- Stateful bean instance can service the requests of a single client only
- Reduces application scalability

LIFE CYCLE OF STATEFUL

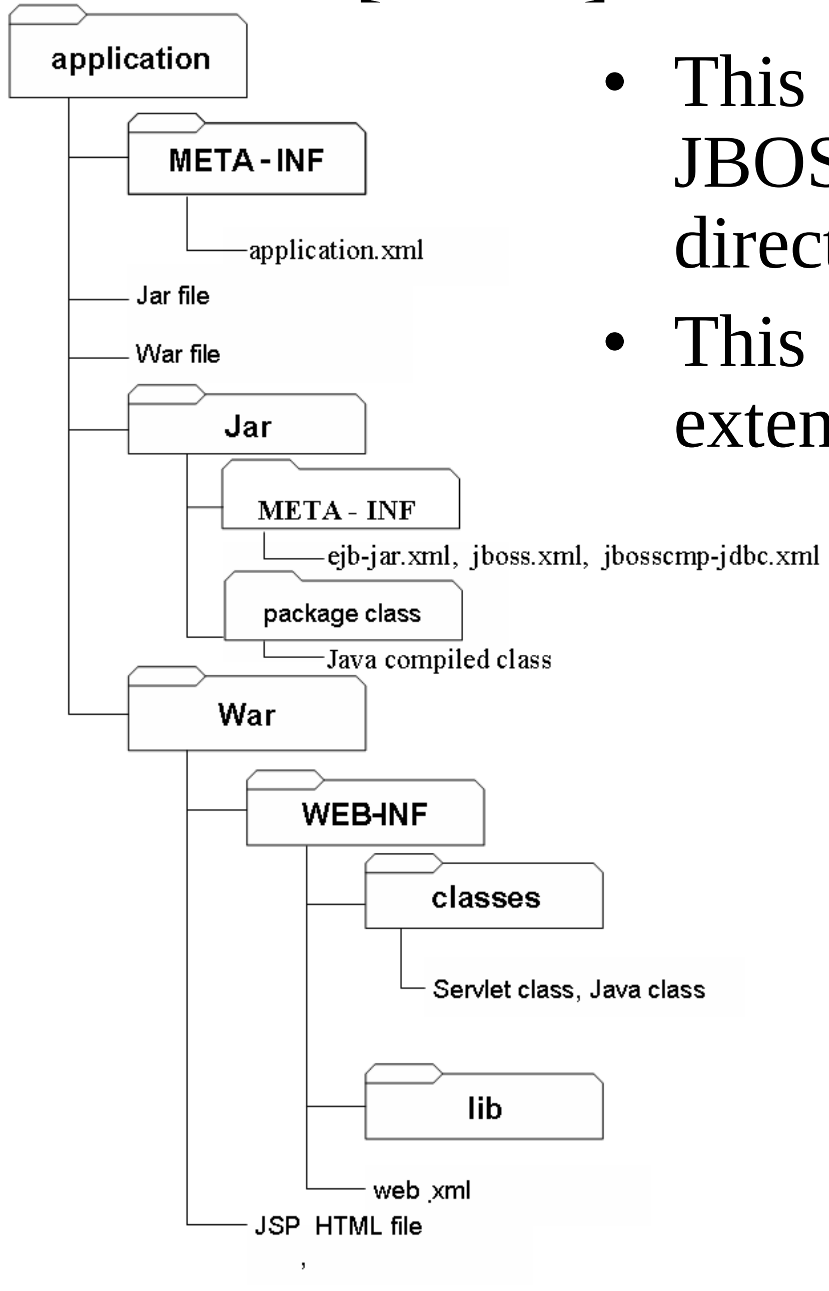


- Bean Creation: Statelful are created by the container when a client makes a request for a bean.
- Bean Use: Statelful are used when the clients call their business methods.
- Bean Passivation: Statelful are passivated or made inactive when the client does not call the session bean business methods for a specified period of time.
- Bean Activation: from passivation, the Stateful moves to activation stage when the client request for bean services after a prolonged period of inactivity.
- Bean Removal: beans are removed when the container decides there are too many beans in the pool or the beans throw a system exception.

ACCESSING from CLIENT SIDE



[WEB] APPLICATION In EJB



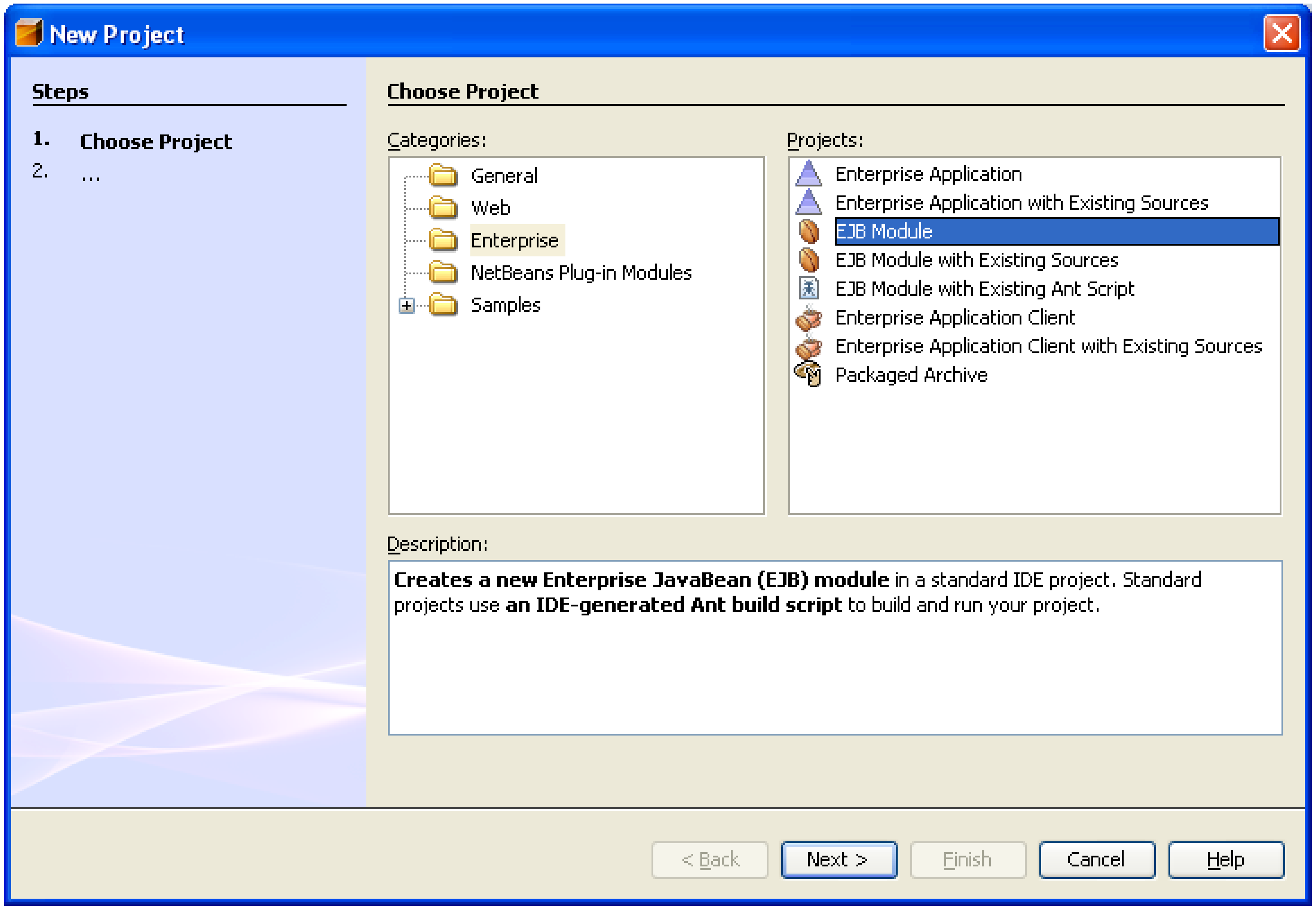
- This structure is deployed at JBOSS_HOME\server\default\deploy directory
- This structure is name with the extension **.ear (include jar and war)**

EJB DEVELOPMENT PROCESS

- **Requirement: JBoss 4.2.2 GA Application Server**
- **Step 1:** Creating a new EJB Module project
- **Step 2:** Creating the new corresponding bean depending on your purpose.
- **Step 3:** Building/ Modifying the business/callback methods on Beans
- **Step 4:** Mapping the JNDI to beans
- **Step 5:** Building the project to jar file
- **Step 6:** Deploying the project on Application server
- **Step 7:** Creating the client application to consume
- **Step 8:** Running the client to test the EJB

Step 1: Creating a new EJB Module project

- Create new Project
- Choose “Enterprise” on “Categories”
- Then, choose “EJB Module” on “Projects”. Click Next button


New Project

Steps

1. Choose Project
2. ...

Choose Project

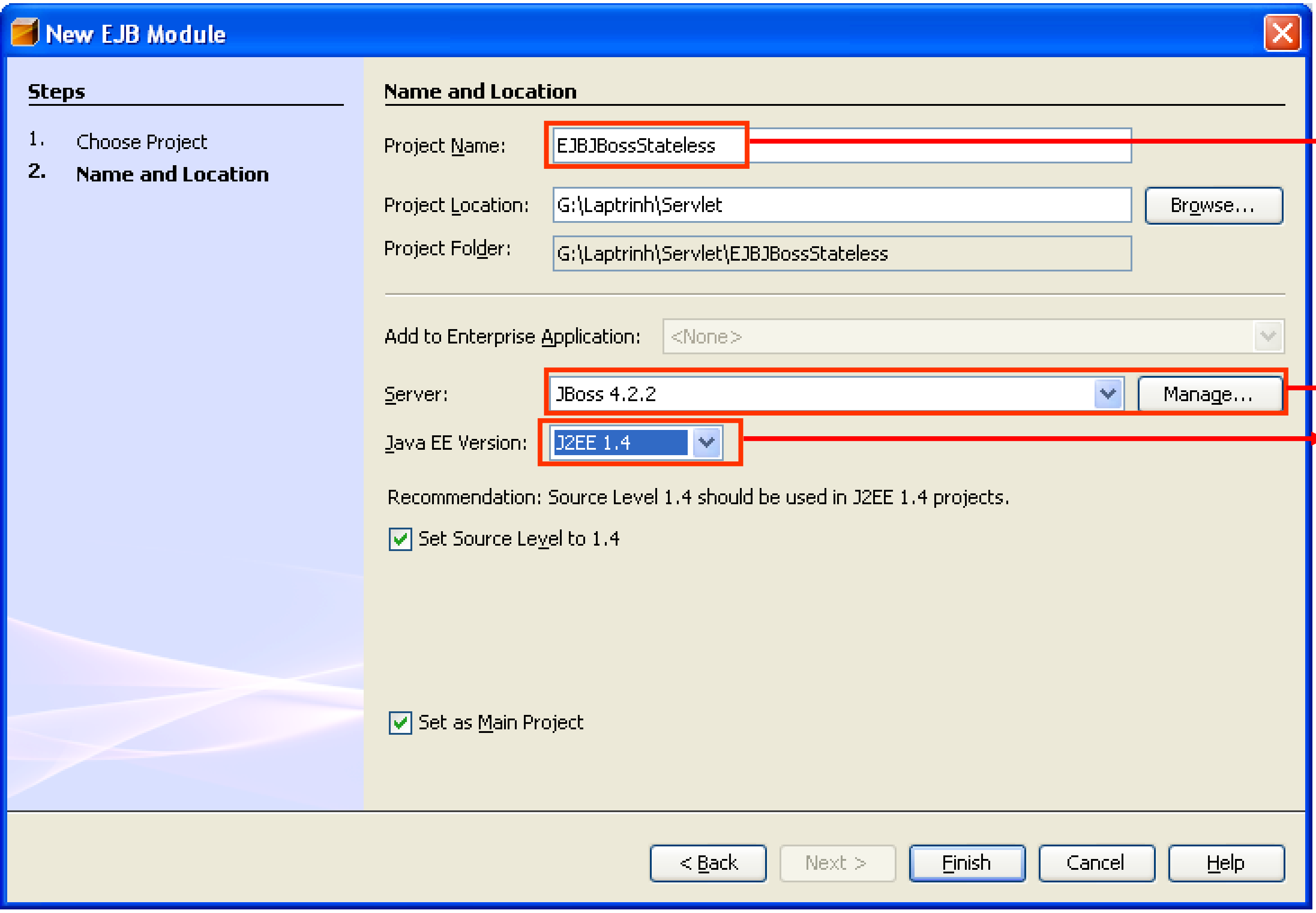
Categories:
- General
- Web
- Enterprise
- NetBeans Plug-in Modules
- Samples

Projects:
- Enterprise Application
- Enterprise Application with Existing Sources
- EJB Module
- EJB Module with Existing Sources
- EJB Module with Existing Ant Script
- Enterprise Application Client
- Enterprise Application Client with Existing Sources
- Packaged Archive

Description:

Creates a new Enterprise JavaBean (EJB) module in a standard IDE project. Standard projects use **an IDE-generated Ant build script** to build and run your project.

Step 1: Creating a new EJB Module project (cont)



New EJB Module

Steps
1. Choose Project
2. **Name and Location**

Name and Location

Project Name: EJBJBossStateless

Project Location: G:\Laptrinh\Servlet

Project Folder: G:\Laptrinh\Servlet\EJBJBossStateless

Add to Enterprise Application: <None>

Server: JBoss 4.2.2

Java EE Version: J2EE 1.4

Recommendation: Source Level 1.4 should be used in J2EE 1.4 projects.

[✓] Set Source Level to 1.4

[✓] Set as Main Project

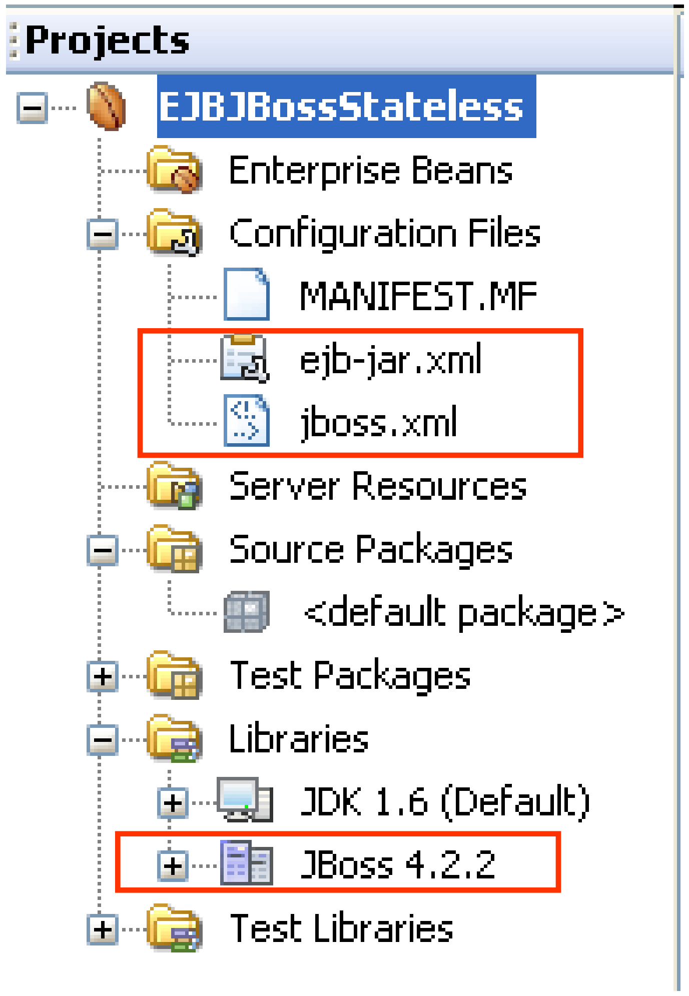
Fill your project name

Choose the Jboss Server that is added

Choose the J2EE1.4

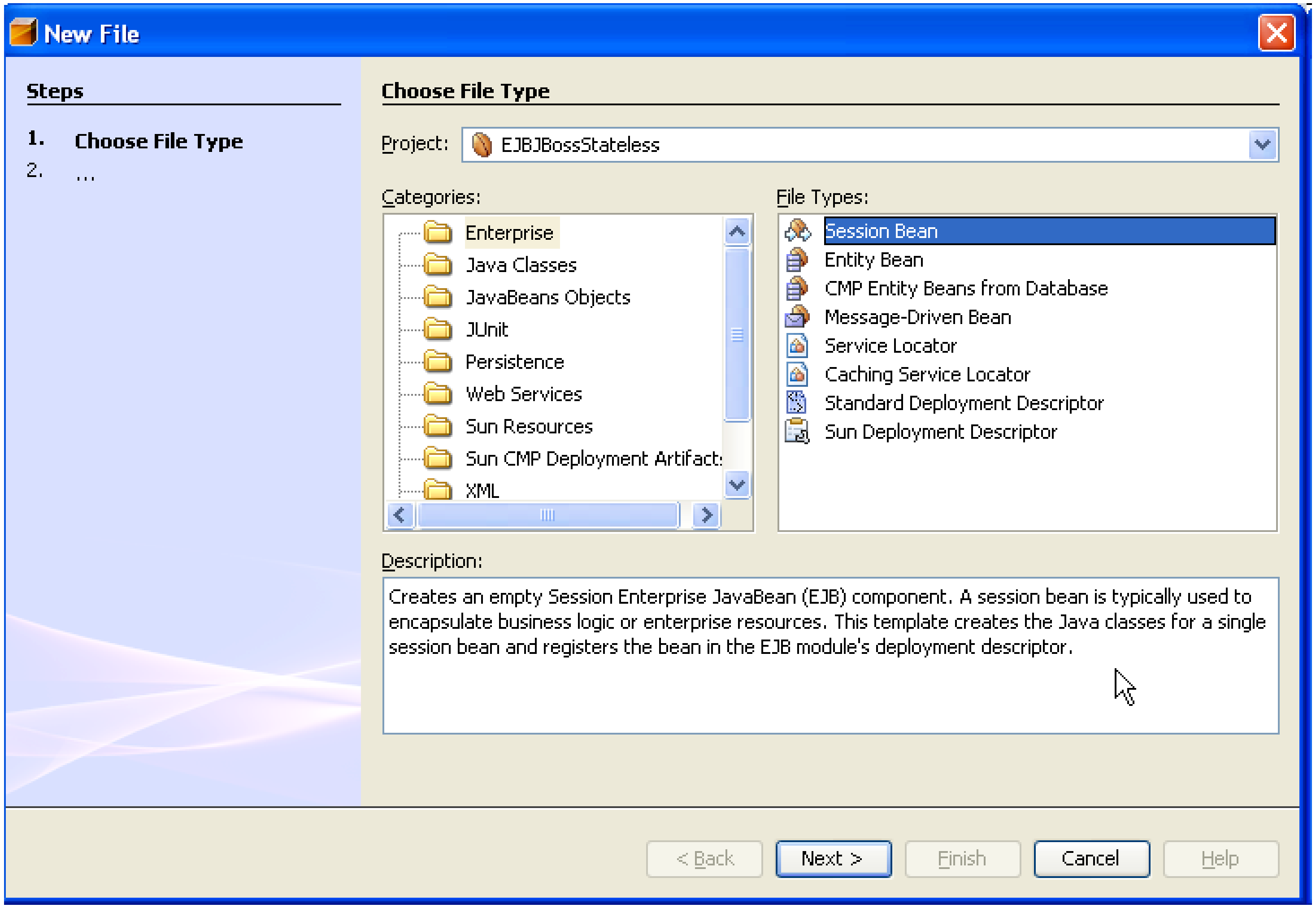
- Click Finish button

Step 1: Creating a new EJB Module project (cont)



Step 2: Creating the new corresponding bean

- Choose File menu/ click new File
- Choose "Enterprise" on "Categories"
- Then, choose "Session Bean" on "File Types". Click Next button


New File

Steps

1. Choose File Type
2. ...

Choose File Type

Project: EJBJBossStateless

Categories:
- Enterprise
- Java Classes
- JavaBeans Objects
- JUnit
- Persistence
- Web Services
- Sun Resources
- Sun CMP Deployment Artifact:
- XML

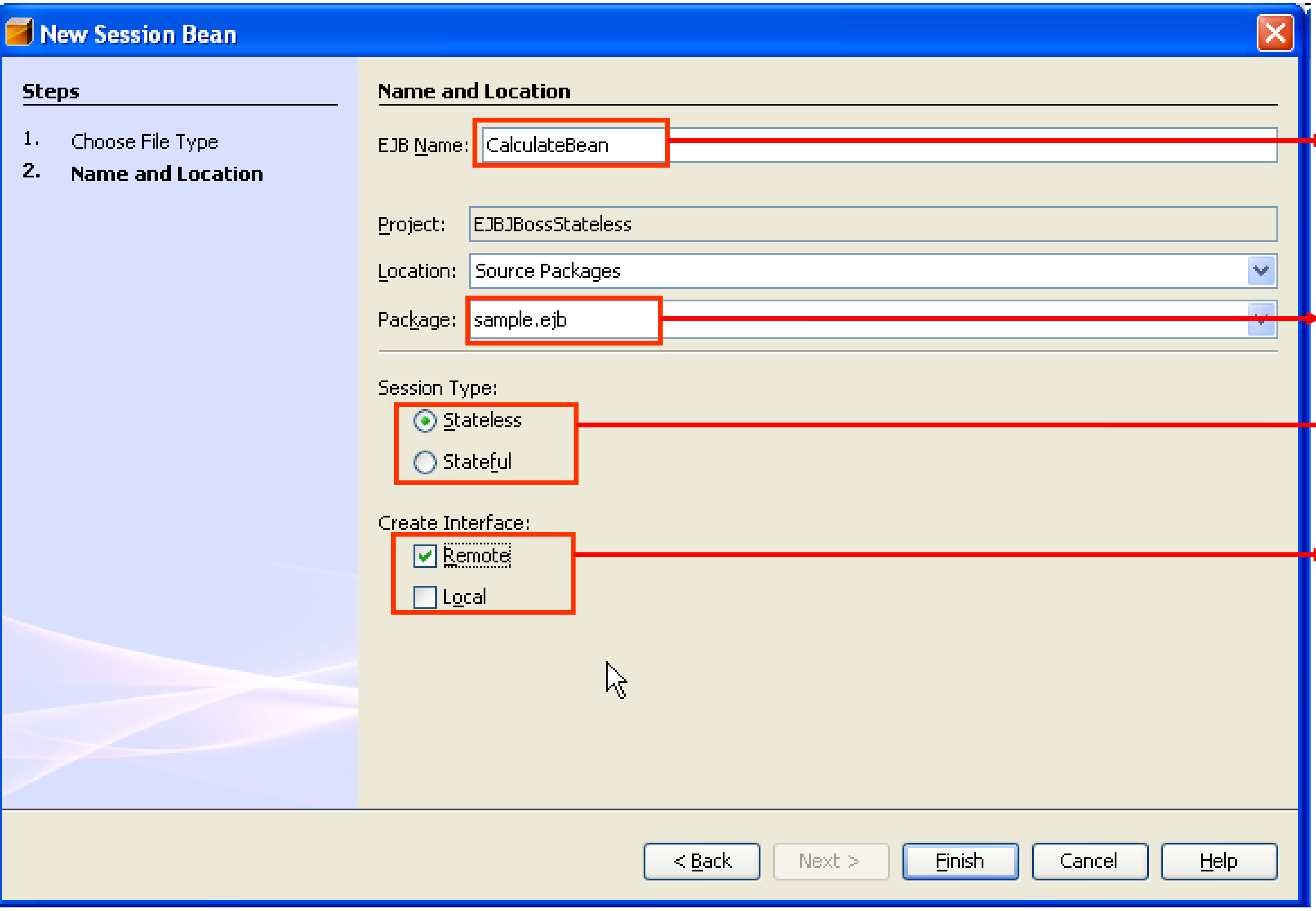
File Types:
- Session Bean
- Entity Bean
- CMP Entity Beans from Database
- Message-Driven Bean
- Service Locator
- Caching Service Locator
- Standard Deployment Descriptor
- Sun Deployment Descriptor

Description:

Creates an empty Session Enterprise JavaBean (EJB) component. A session bean is typically used to encapsulate business logic or enterprise resources. This template creates the Java classes for a single session bean and registers the bean in the EJB module's deployment descriptor.

< Back | Next > | Finish | Cancel | Help

Step 2: Creating the new corresponding bean (cont)



Name and Location

EJB Name: CalculateBean

Project: EJBJBossStateless

Location: Source Packages

Package: sample.ejb

Session Type: [✓ Stateless] Stateful

Create Interface: [✓ Remote] Local

Fill your bean name

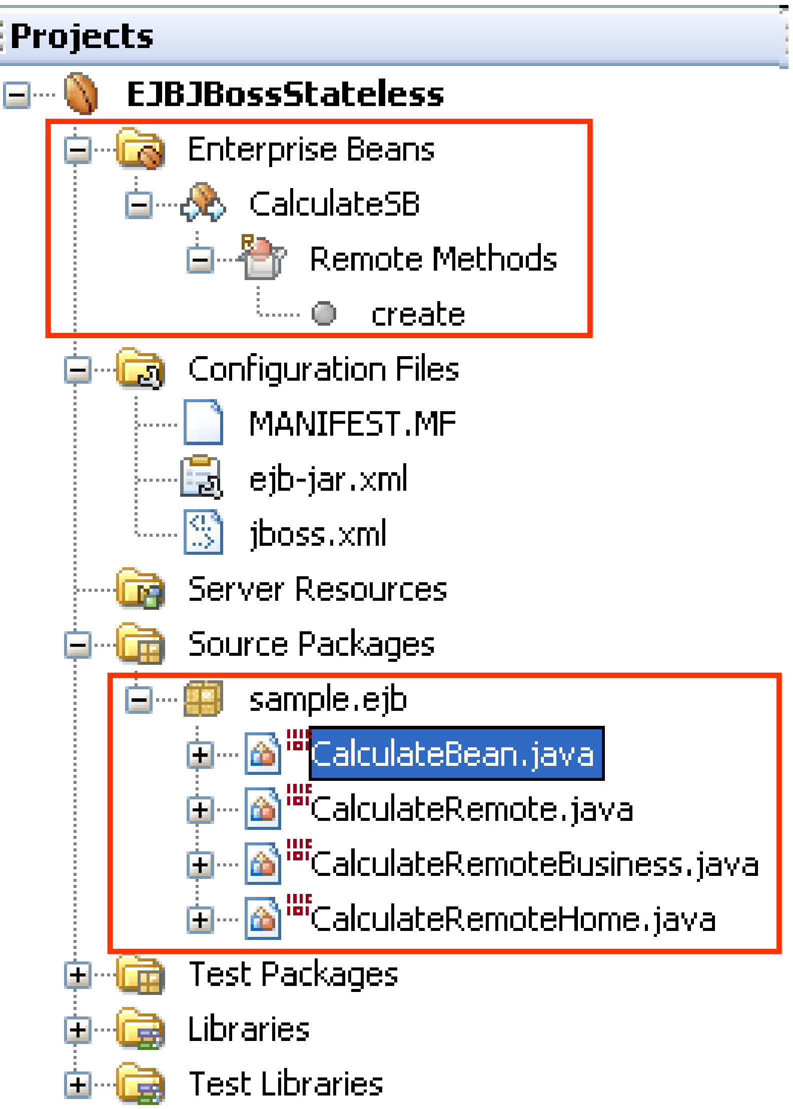
Fill or choose the package name

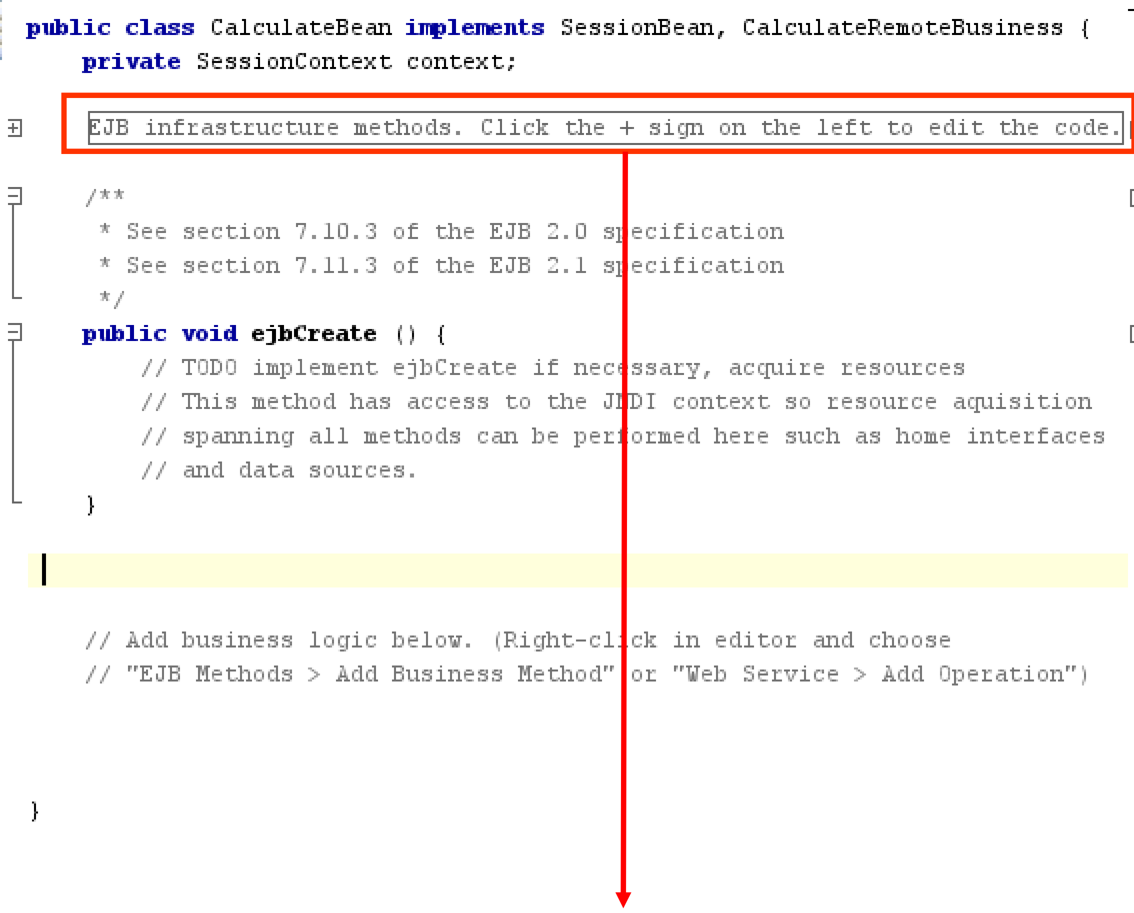
Choose stateless or stateful

Choose remote or local or both

- Click Finish button

Step 2: Creating the new corresponding bean (cont)





```
public class CalculateBean implements SessionBean, CalculateRemoteBusiness {
    private SessionContext context;

    EJB infrastructure methods. Click the + sign on the left to edit the code.

    /**
     * See section 7.10.3 of the EJB 2.0 specification
     * See section 7.11.3 of the EJB 2.1 specification
     */
    public void ejbCreate () {
        // TODO implement ejbCreate if necessary, acquire resources
        // This method has access to the JNDI context so resource aquisition
        // spanning all methods can be performed here such as home interfaces
        // and data sources.
    }



    // Add business logic below. (Right-click in editor and choose
    // "EJB Methods > Add Business Method" or "Web Service > Add Operation")


}
```

Generate automatically four callback methods:

•setSessionContext

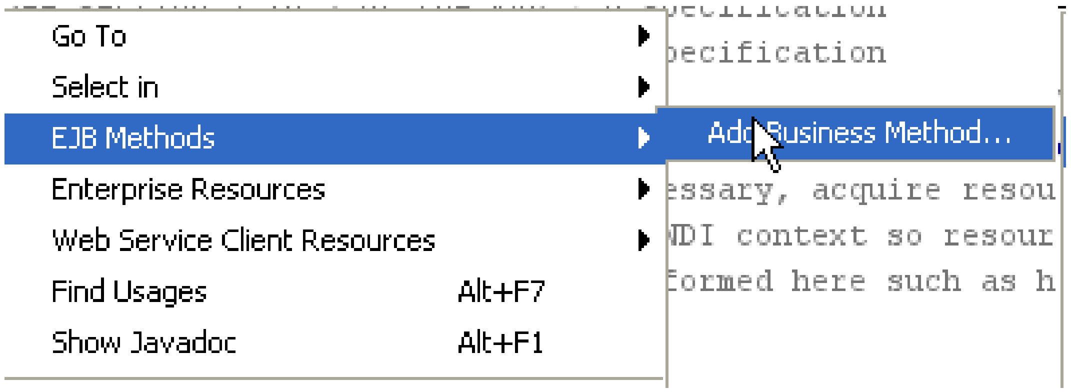
•ejbActivate

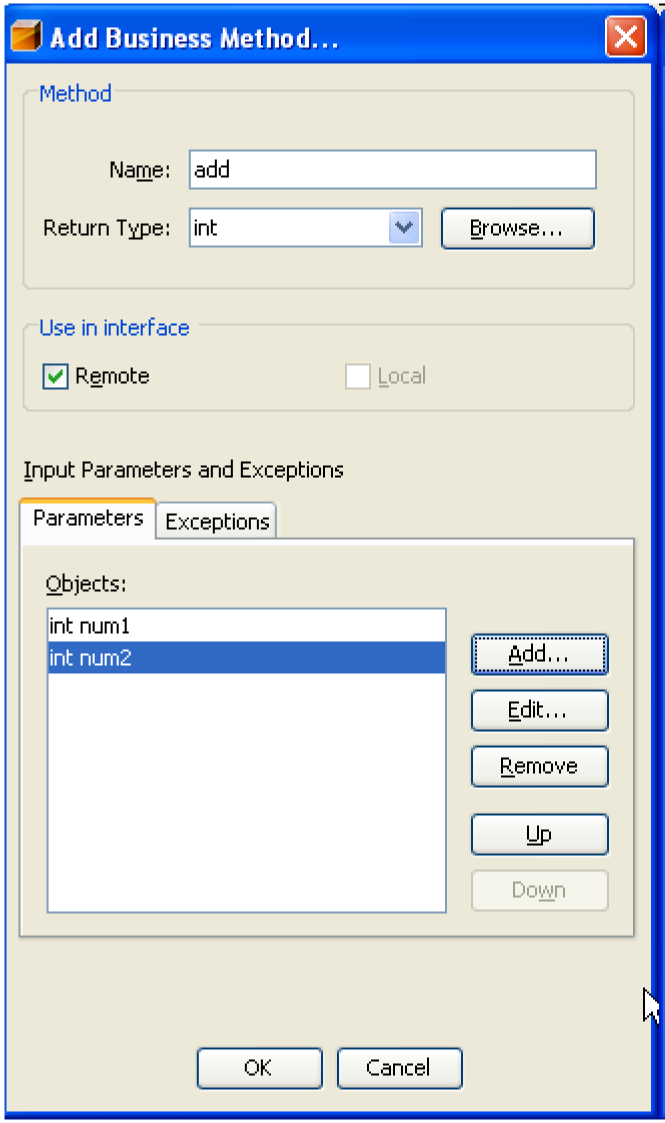
•ejbPassivate

•ejbRemove

Step 3: Building/ Modifying the business/callback methods on Beans

- Modifying the callback method if necessary
- Adding a new business method
  - Right click on source code of the Bean file (Ex: CalculateBean)
  - Then, choose EJB Methods, click Add Business Method…




Add Business Method...

Method

Name: add

Return Type: int

Use in interface

Remote ✓ / Local ☐

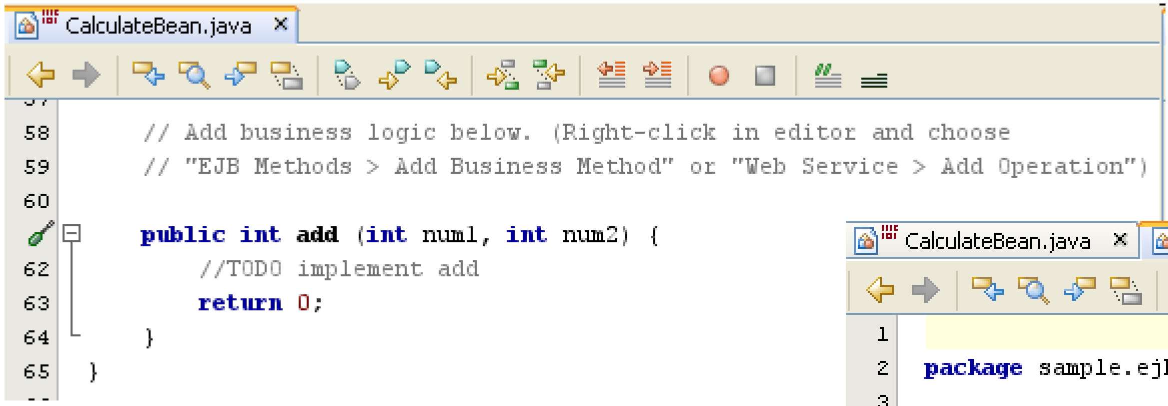
Input Parameters and Exceptions

Objects:

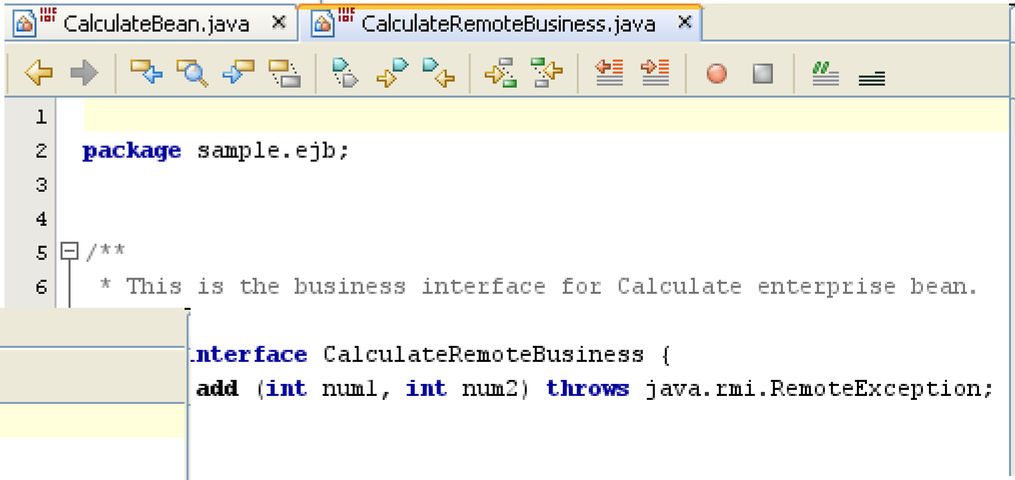
int num1
int num2

- Fill or type the method name with return type and all parameters
- Then, click OK Button

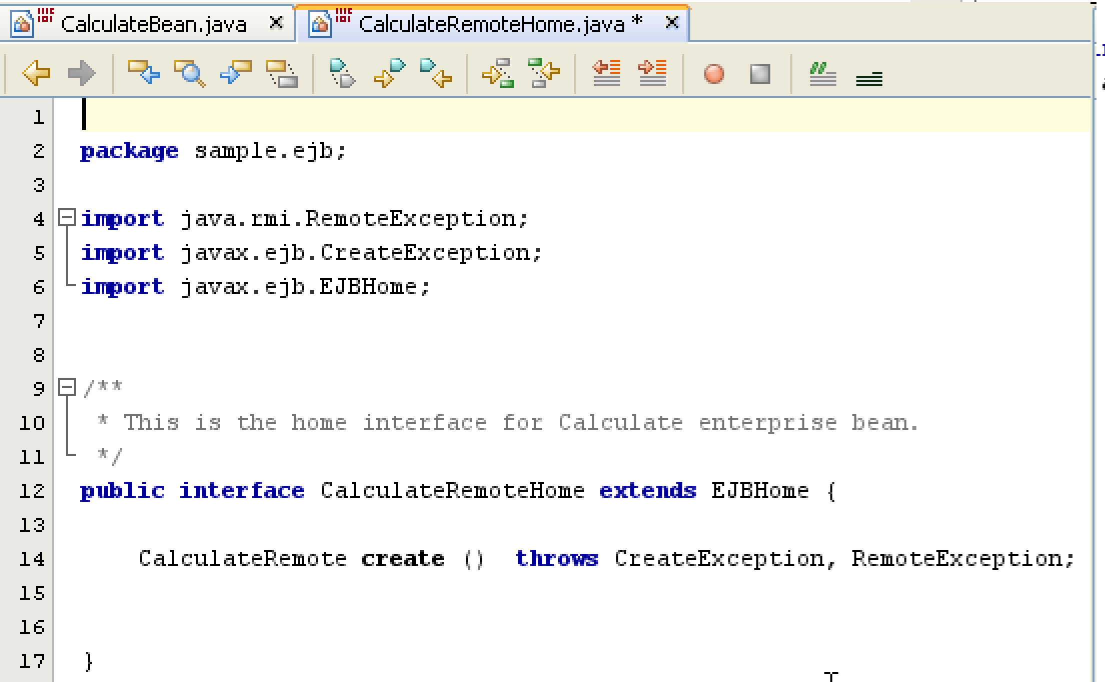
Step 3: Building/ Modifying the business/callback methods on Beans (cont)


CalculateBean.java

```java
    // Add business logic below. (Right-click in editor and choose
    // "EJB Methods > Add Business Method" or "Web Service > Add Operation")

    public int add (int num1, int num2) {
        //TODO implement add
        return 0;
    }
}
```


CalculateBean.java
CalculateRemoteBusiness.java

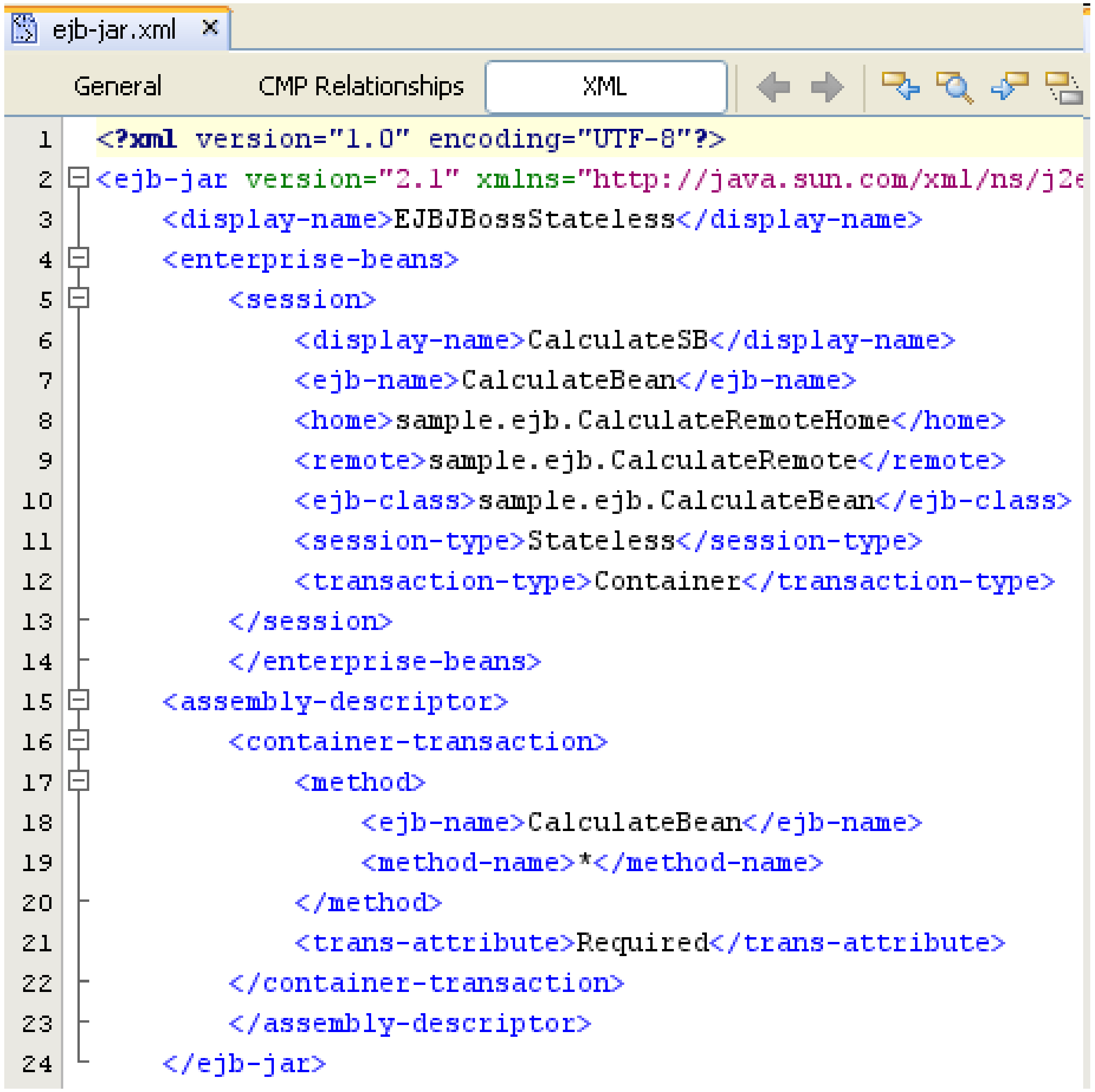
```java

package sample.ejb;


/**
 * This is the business interface for Calculate enterprise bean.
nterface CalculateRemoteBusiness {
 add (int num1, int num2) throws java.rmi.RemoteException;
```


CalculateBean.java
CalculateRemoteHome.java *

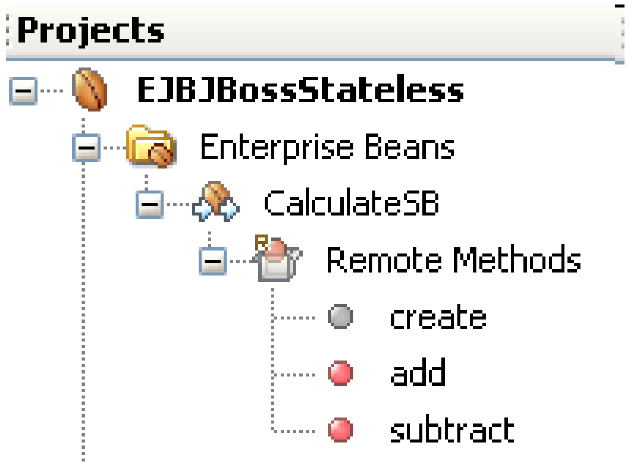
```java

package sample.ejb;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;


/**
 * This is the home interface for Calculate enterprise bean.
 */
public interface CalculateRemoteHome extends EJBHome {

    CalculateRemote create ()  throws CreateException, RemoteException;


}
```

- Implement the body of method corresponding with your purpose

Step 3: Building/ Modifying the business/callback methods on Beans (cont)

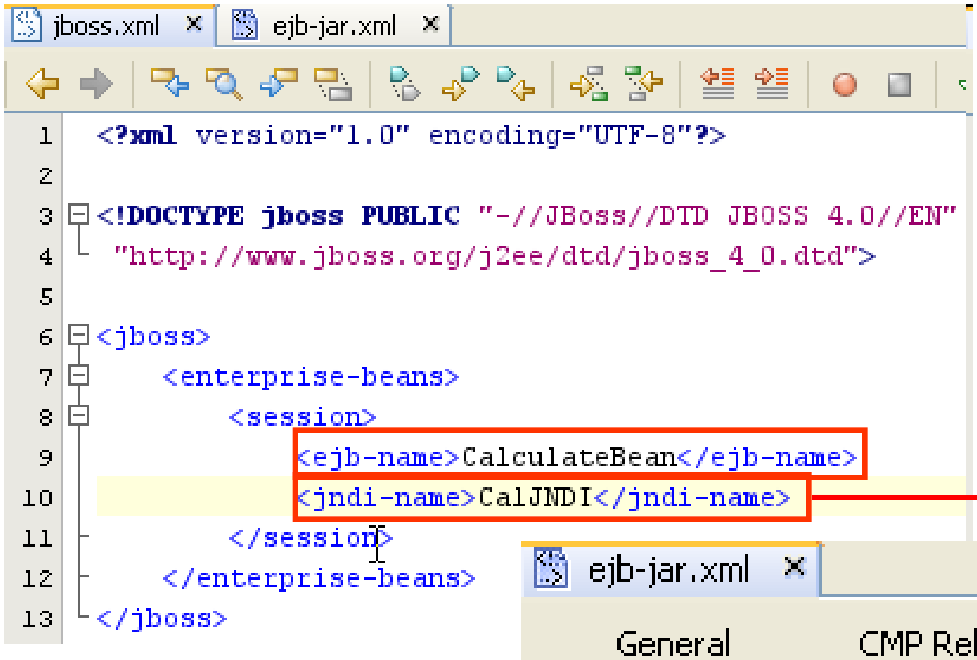


```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2e
    <display-name>EJBJBossStateless</display-name>
    <enterprise-beans>
        <session>
            <display-name>CalculateSB</display-name>
            <ejb-name>CalculateBean</ejb-name>
            <home>sample.ejb.CalculateRemoteHome</home>
            <remote>sample.ejb.CalculateRemote</remote>
            <ejb-class>sample.ejb.CalculateBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
        </enterprise-beans>
    <assembly-descriptor>
        <container-transaction>
            <method>
                <ejb-name>CalculateBean</ejb-name>
                <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
        </container-transaction>
        </assembly-descriptor>
    </ejb-jar>
```



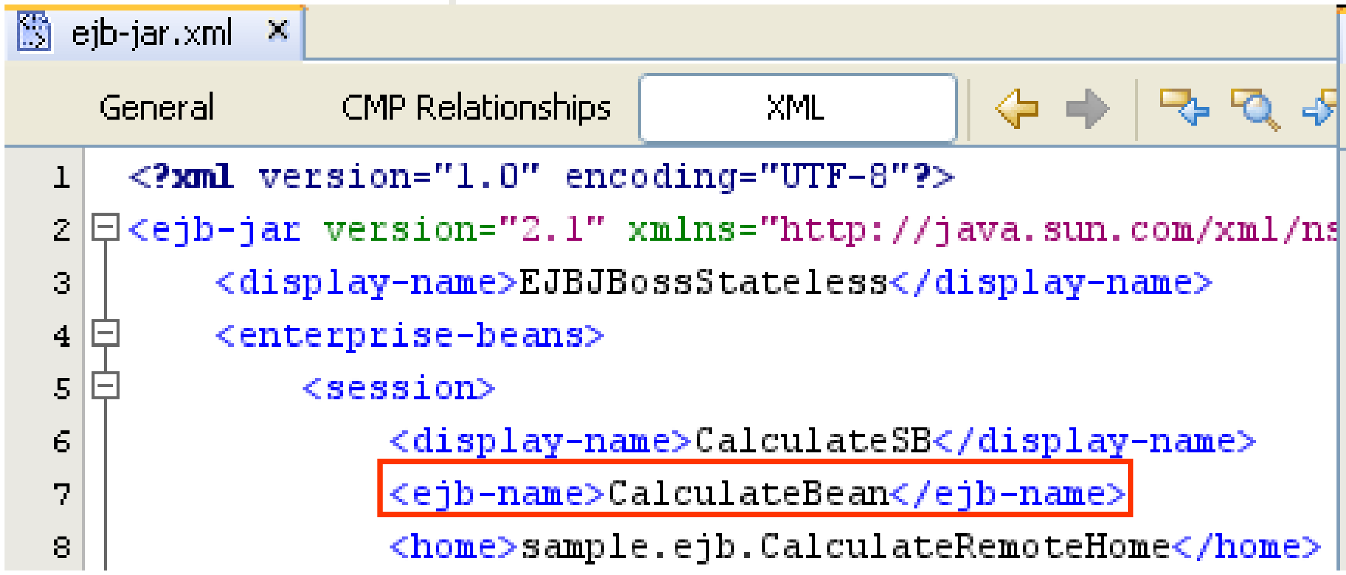
Step 4: Mapping the JNDI to beans

- Modify the jboss.xml file as following



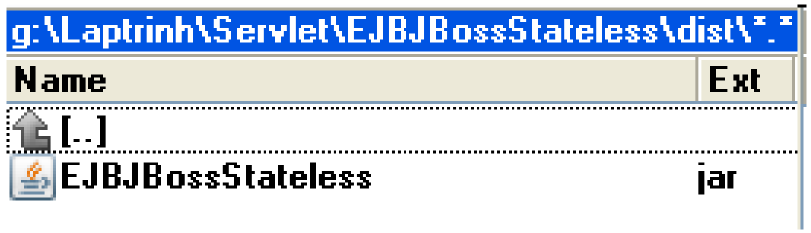
```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 4.0//EN"
 "http://www.jboss.org/j2ee/dtd/jboss_4_0.dtd">

<jboss>
    <enterprise-beans>
        <session>
            <ejb-name>CalculateBean</ejb-name>
            <jndi-name>CalJNDI</jndi-name>
        </session>
    </enterprise-beans>
</jboss>
```

Fill your wanted JNDI that you want to reference



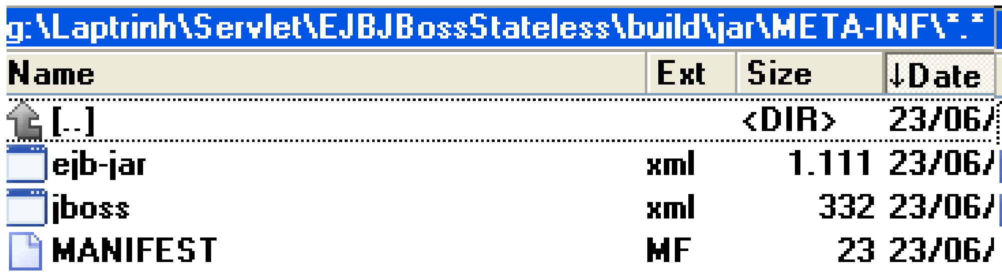
```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns
    <display-name>EJBJBossStateless</display-name>
    <enterprise-beans>
        <session>
            <display-name>CalculateSB</display-name>
            <ejb-name>CalculateBean</ejb-name>
            <home>sample.ejb.CalculateRemoteHome</home>
```

Step 5 and 6: Building & Deploying



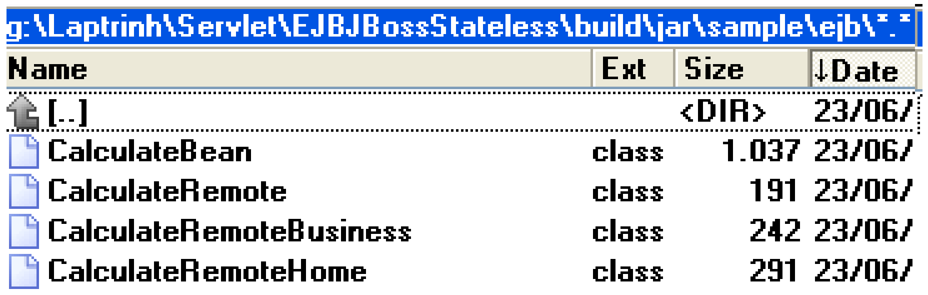
g:\Laptrinh\Servlet\EJBJBossStateless\dist\*.*

Name	Ext
[..]	
EJBJBossStateless	jar



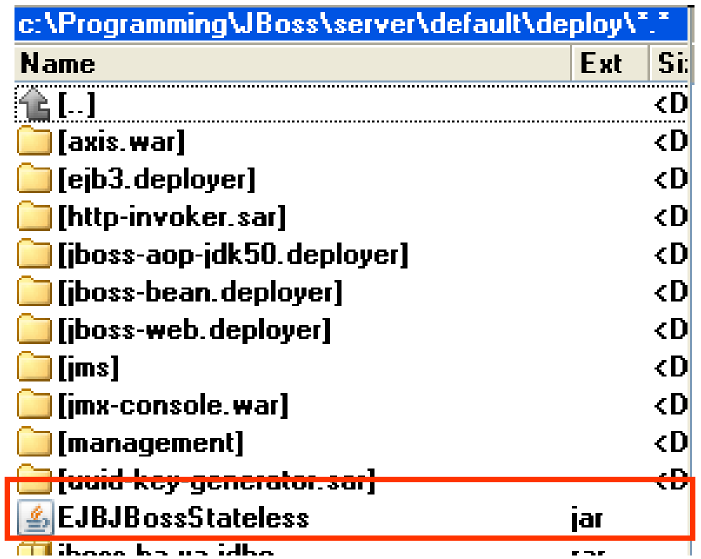
g:\Laptrinh\Servlet\EJBJBossStateless\build\jar\META-INF\*.*

Name	Ext	Size	↓Date
[..]		<DIR>	23/06/
ejb-jar	xml	1.111	23/06/
jboss	xml	332	23/06/
MANIFEST	MF	23	23/06/



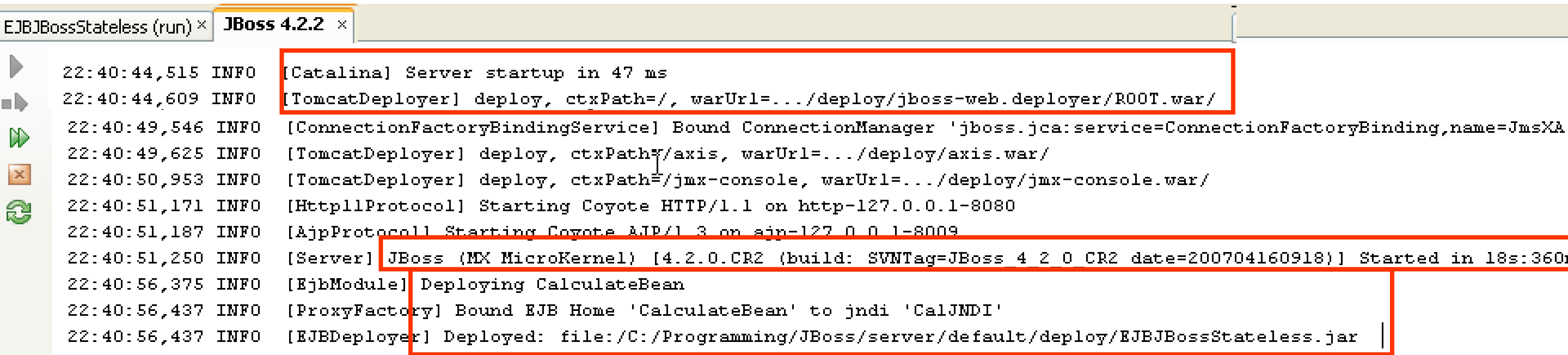
g:\Laptrinh\Servlet\EJBJBossStateless\build\jar\sample\ejb\*.*

Name	Ext	Size	↓Date
[..]		<DIR>	23/06/
CalculateBean	class	1.037	23/06/
CalculateRemote	class	191	23/06/
CalculateRemoteBusiness	class	242	23/06/
CalculateRemoteHome	class	291	23/06/



c:\Programming\JBoss\server\default\deploy\*.*

Name	Ext	Si
[..]		<D
[axis.war]		<D
[ejb3.deployer]		<D
[http-invoker.sar]		<D
[jboss-aop-jdk50.deployer]		<D
[jboss-bean.deployer]		<D
[jboss-web.deployer]		<D
[jms]		<D
[jmx-console.war]		<D
[management]		<D
[uuid-key-generator.sar]		<D
EJBJBossStateless	jar	

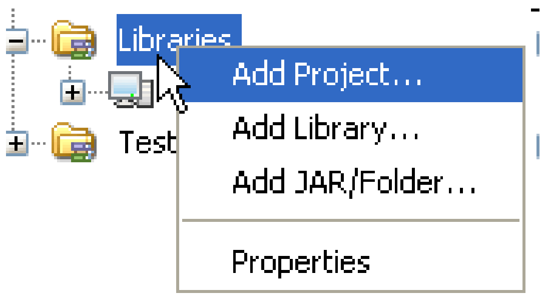


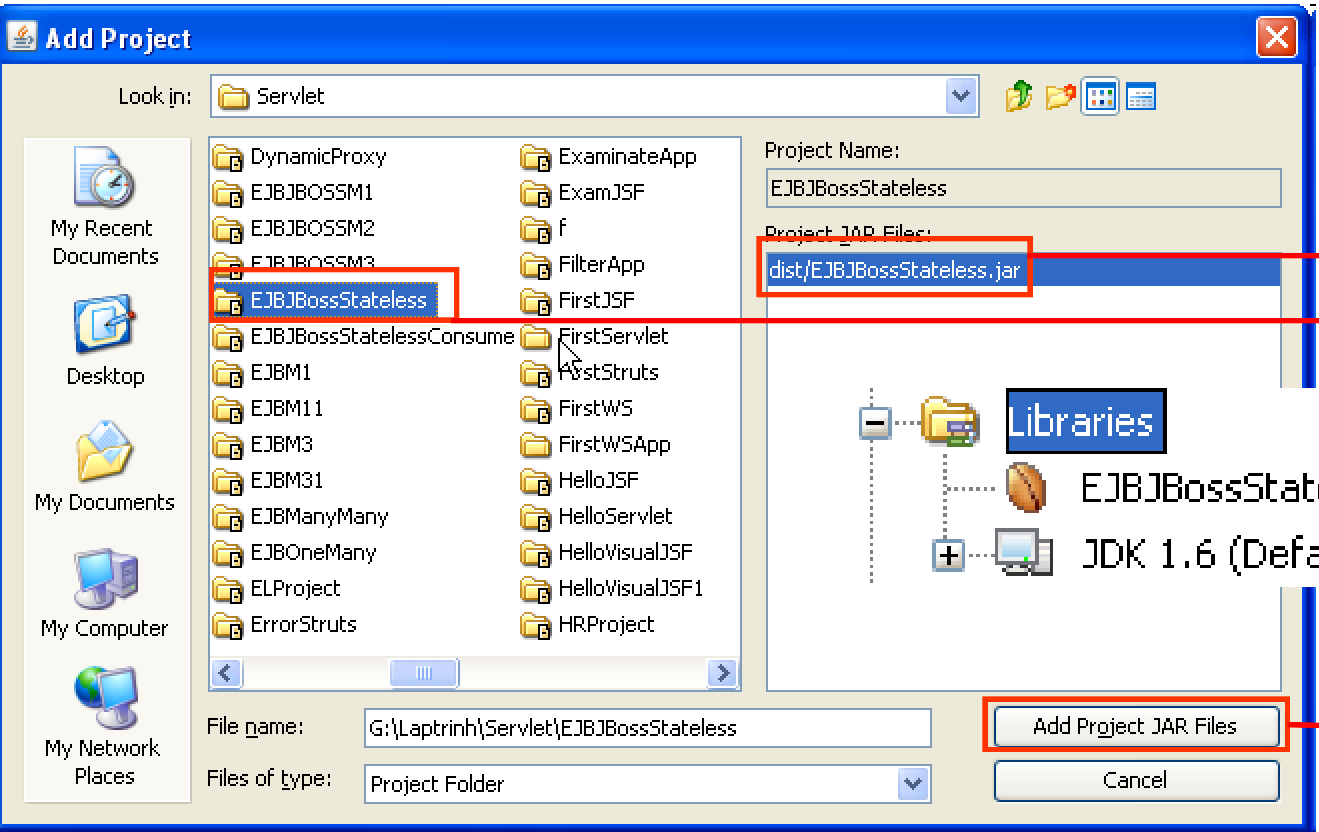
EJBJBossStateless (run) × JBoss 4.2.2 ×

```
22:40:44,515 INFO  [Catalina] Server startup in 47 ms
22:40:44,609 INFO  [TomcatDeployer] deploy, ctxPath=/, warUrl=.../deploy/jboss-web.deployer/ROOT.war/
22:40:49,546 INFO  [ConnectionFactoryBindingService] Bound ConnectionManager 'jboss.jca:service=ConnectionFactoryBinding,name=JmsXA
22:40:49,625 INFO  [TomcatDeployer] deploy, ctxPath=/axis, warUrl=.../deploy/axis.war/
22:40:50,953 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../deploy/jmx-console.war/
22:40:51,171 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-8080
22:40:51,187 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
22:40:51,250 INFO  [Server] JBoss (MX MicroKernel) [4.2.0.CR2 (build: SVNTag=JBoss_4_2_0_CR2 date=200704160918)] Started in 18s:360
22:40:56,375 INFO  [EjbModule] Deploying CalculateBean
22:40:56,437 INFO  [ProxyFactory] Bound EJB Home 'CalculateBean' to jndi 'CalJNDI'
22:40:56,437 INFO  [EJBDeployer] Deployed: file:/C:/Programming/JBoss/server/default/deploy/EJBJBossStateless.jar
```

Step 7: Creating the client application to consume

- Create Java console application
- Add reference to EJB project mapping to invoke the remote method on application Server
  - Right click on library of client project/ click "Add Project …"





Project Name: EJBJBossStateless

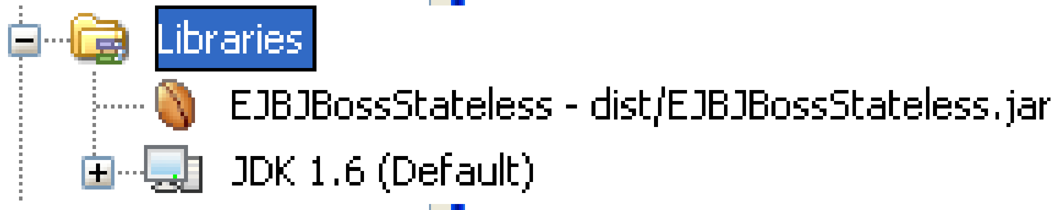
Project JAR Files: dist/EJBJBossStateless.jar

File name: G:\Laptrinh\Servlet\EJBJBossStateless

Files of type: Project Folder

Choose the jar file

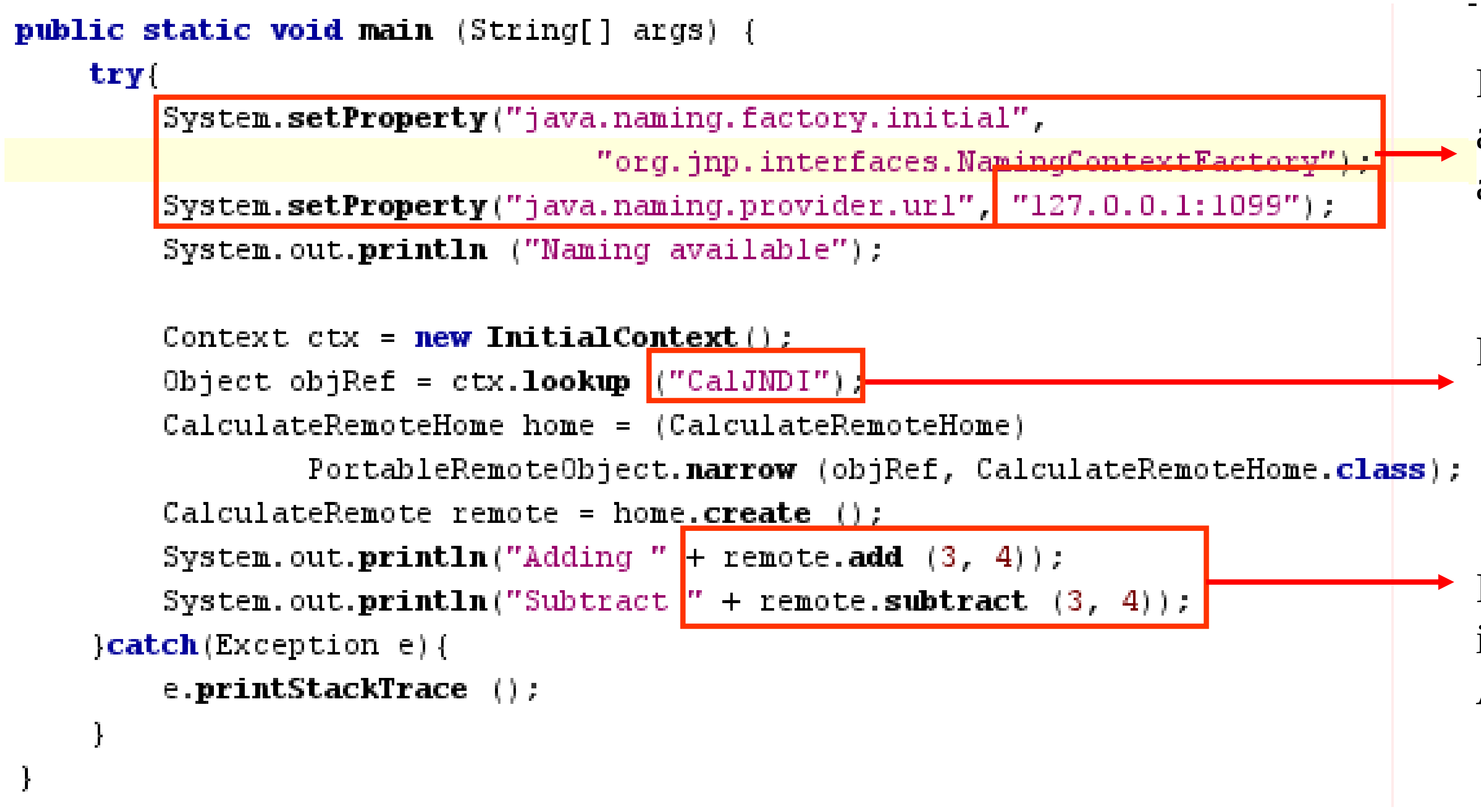
Modify servlet name



Click Add Project Jar File

Step 7: Creating the client application to consume (2)

- Adding the code as following (**notes: addition the jbossall-client.jar and jnp-client.jar from JBOSS_HOME\client to application project**)



```
public static void main (String[] args) {
    try{
        System.setProperty("java.naming.factory.initial",
                           "org.jnp.interfaces.NamingContextFactory");
        System.setProperty("java.naming.provider.url", "127.0.0.1:1099");
        System.out.println ("Naming available");

        Context ctx = new InitialContext();
        Object objRef = ctx.lookup ("CalJNDI");
        CalculateRemoteHome home = (CalculateRemoteHome)
                PortableRemoteObject.narrow (objRef, CalculateRemoteHome.class);
        CalculateRemote remote = home.create ();
        System.out.println("Adding " + remote.add (3, 4));
        System.out.println("Subtract " + remote.subtract (3, 4));
    }catch(Exception e){
        e.printStackTrace ();
    }
}
```

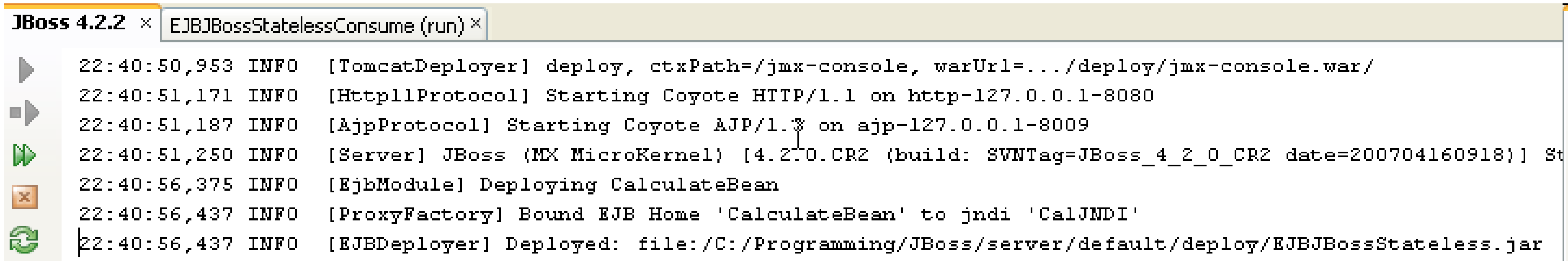
Reference to the application server address (location)

Reference to EJB

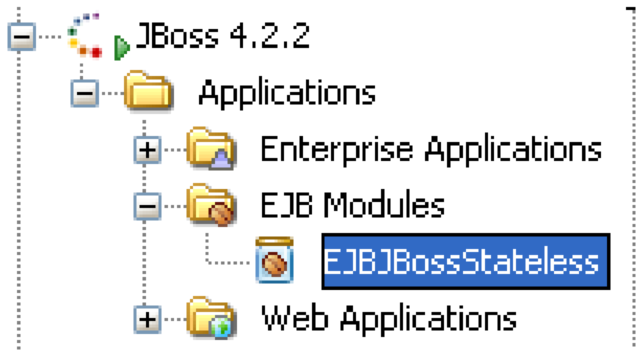
Invoke the remote interface of EJB on Application Server

Step 8: Running the client to test the EJB

```
run:
Naming available
Adding 7
Subtract -1
```



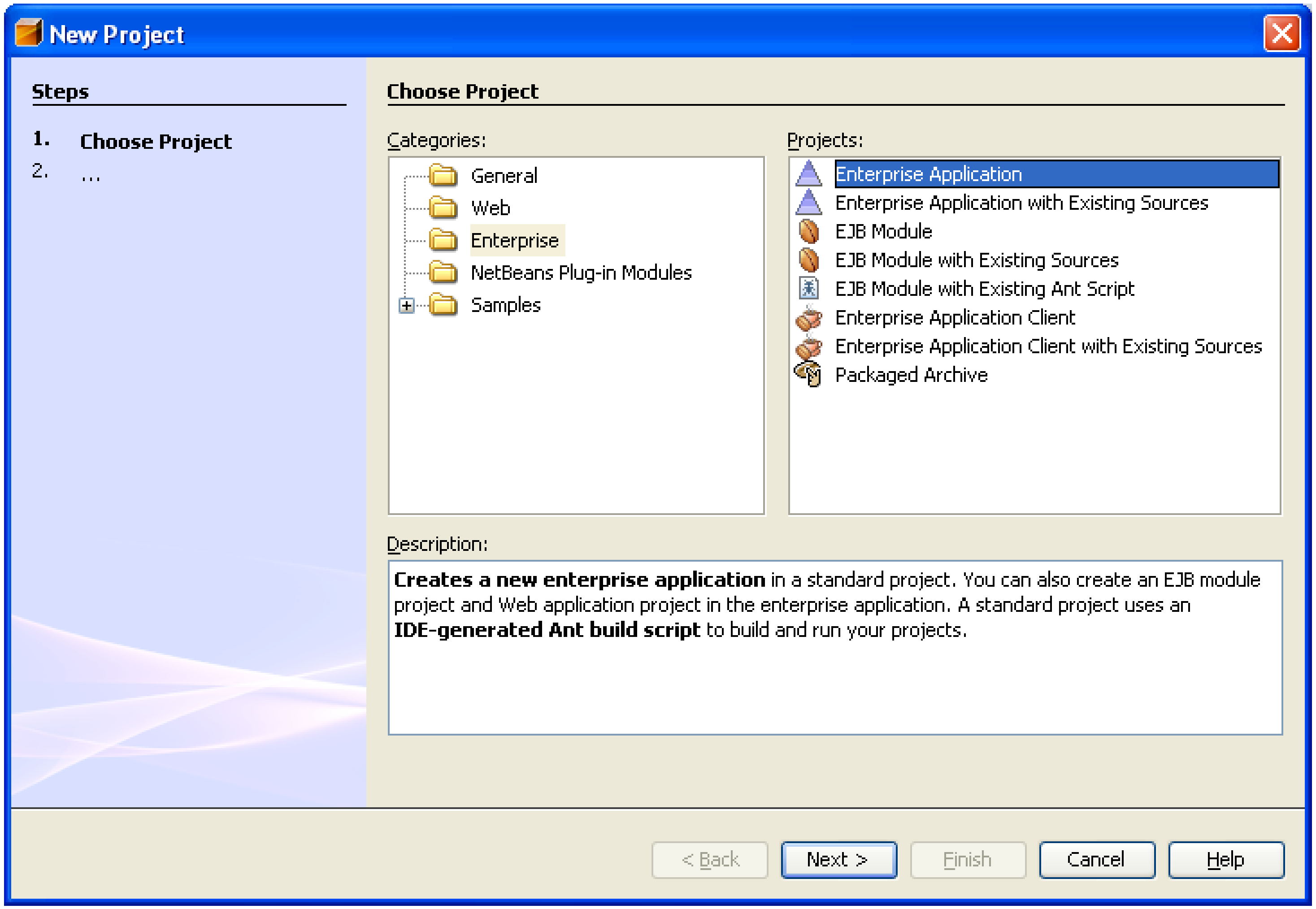
```
22:40:50,953 INFO  [TomcatDeployer] deploy, ctxPath=/jmx-console, warUrl=.../deploy/jmx-console.war/
22:40:51,171 INFO  [Http11Protocol] Starting Coyote HTTP/1.1 on http-127.0.0.1-8080
22:40:51,187 INFO  [AjpProtocol] Starting Coyote AJP/1.3 on ajp-127.0.0.1-8009
22:40:51,250 INFO  [Server] JBoss (MX MicroKernel) [4.2.0.CR2 (build: SVNTag=JBoss_4_2_0_CR2 date=200704160918)] St
22:40:56,375 INFO  [EjbModule] Deploying CalculateBean
22:40:56,437 INFO  [ProxyFactory] Bound EJB Home 'CalculateBean' to jndi 'CalJNDI'
22:40:56,437 INFO  [EJBDeployer] Deployed: file:/C:/Programming/JBoss/server/default/deploy/EJBJBossStateless.jar
```



ENTERPRISE APPLICATION DEVELOPMENT PROCESS

- **Step 1:** Creating a new Enterprise Application project (EJB and Web Client – ear file)
- **Step 2:** Creating the new corresponding bean depending on your purpose.
- **Step 3:** Building/ Modifying the business/callback methods on Beans
- **Step 4:** Mapping the JNDI to beans
- **Step 5:** Creating the GUI to consumes EJB on web modules
- **Step 6:** Building and Deploying Enterprise application on Application Server
- **Step 7:** Executing the Enterprise Application

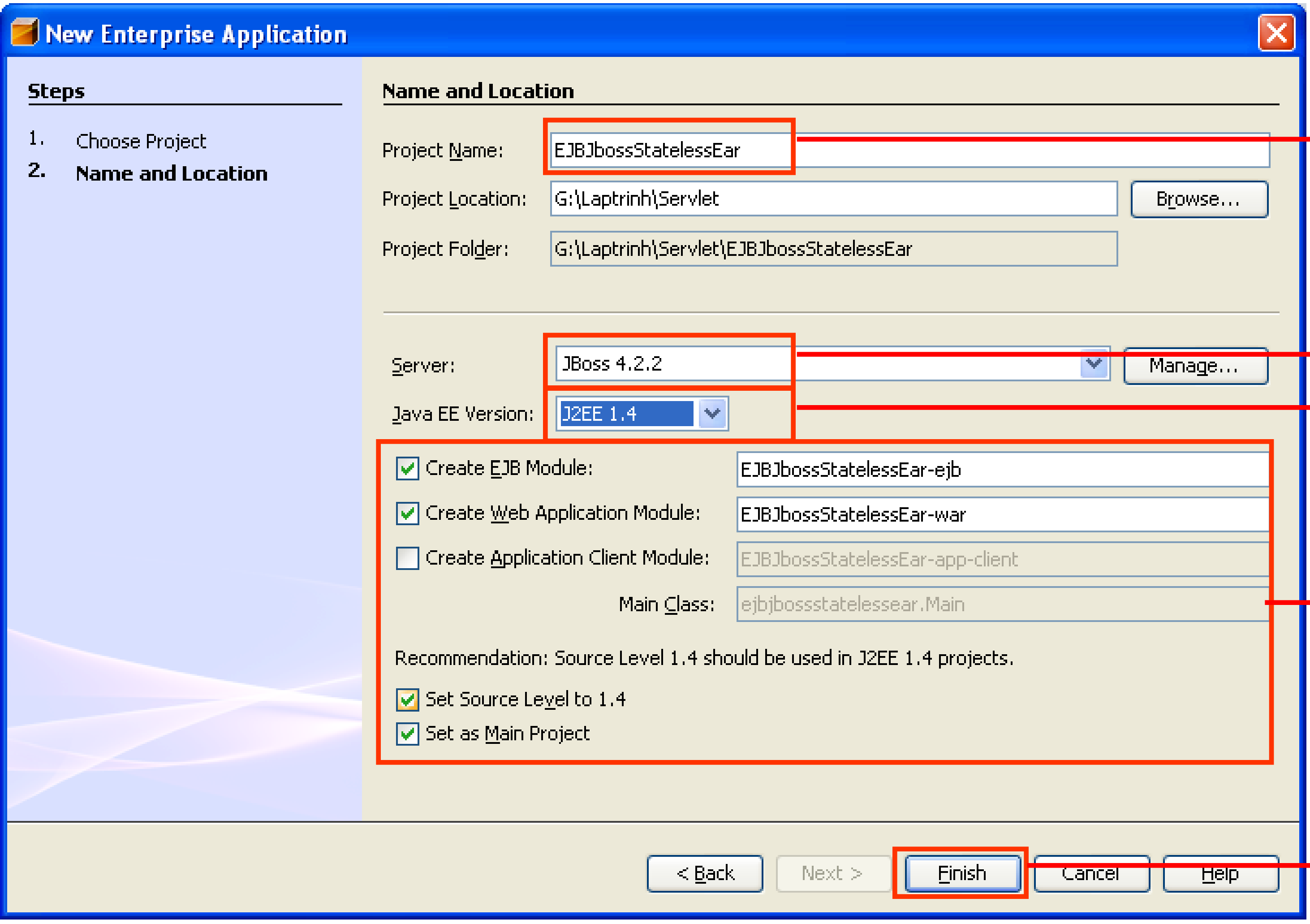
Step 1: Creating a new Enterprise Application project



Creates a new enterprise application in a standard project. You can also create an EJB module project and Web application project in the enterprise application. A standard project uses an **IDE-generated Ant build script** to build and run your projects.

- Choose the Enterprise in Categories
- Then, choose the “Enterprise Application” in Projects
- Click Next Button

Step 1: Creating a new Enterprise Application project



Name and Location

Project Name: EJBJbossStatelessEar

Project Location: G:\Laptrinh\Servlet

Project Folder: G:\Laptrinh\Servlet\EJBJbossStatelessEar

Server: JBoss 4.2.2

Java EE Version: J2EE 1.4

☑ Create EJB Module: EJBJbossStatelessEar-ejb

☑ Create Web Application Module: EJBJbossStatelessEar-war

☐ Create Application Client Module: EJBJbossStatelessEar-app-client

Main Class: ejbjbossstatelessear.Main

Recommendation: Source Level 1.4 should be used in J2EE 1.4 projects.

☑ Set Source Level to 1.4

☑ Set as Main Project

Type the Project name

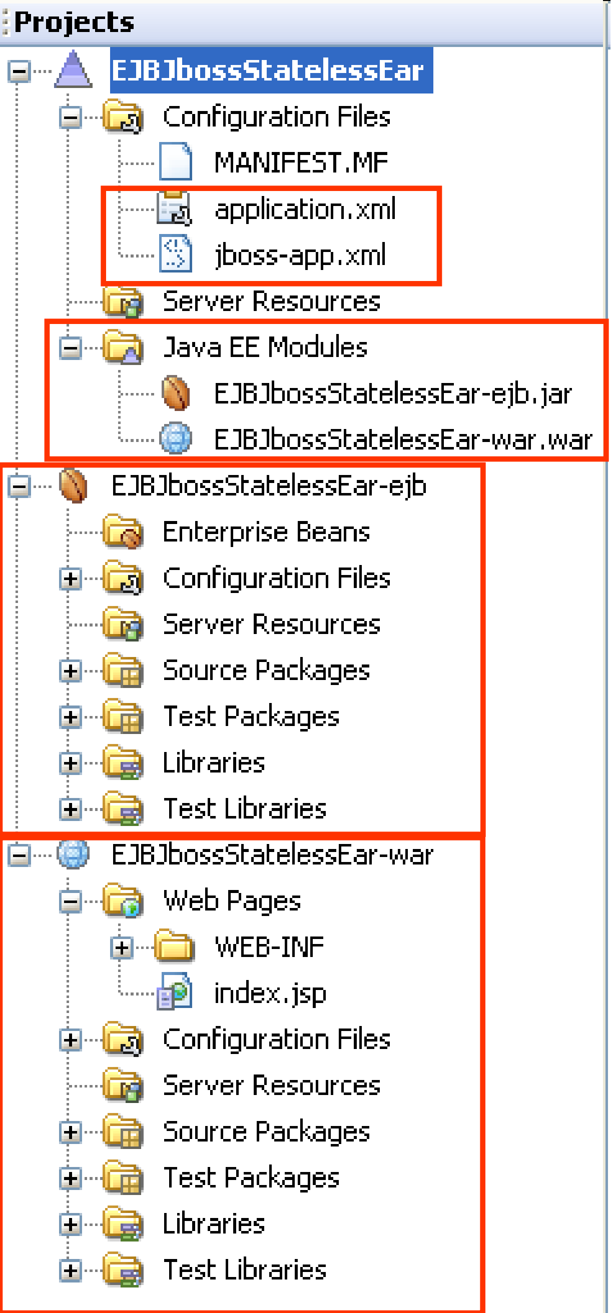
Choose the Jboss 4.2.2

Choose J2EE 1.4

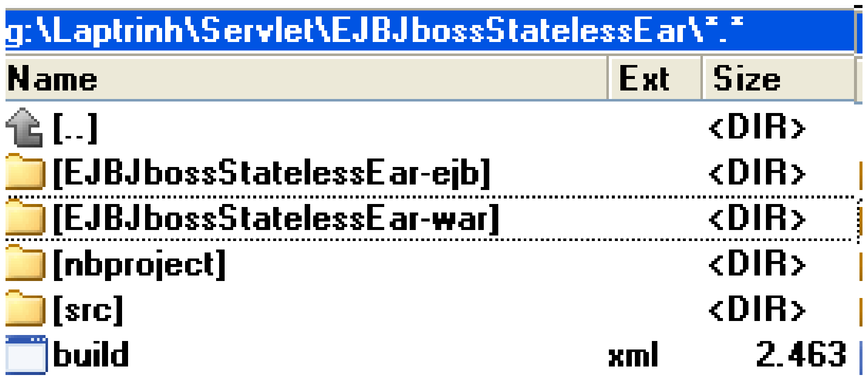
Don’t change anything that is default by tools

Click Finish Button

Step 1: Creating a new Enterprise Application project

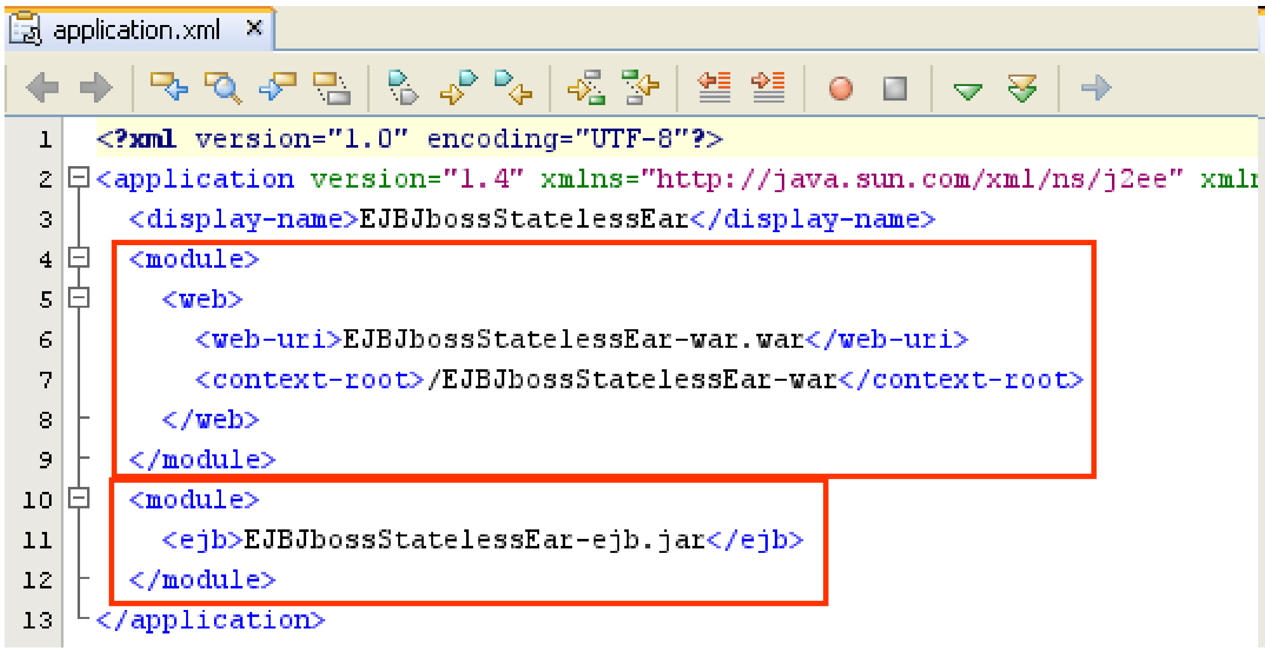


```
Projects
EJBJbossStatelessEar
    Configuration Files
        MANIFEST.MF
        application.xml
        jboss-app.xml
    Server Resources
    Java EE Modules
        EJBJbossStatelessEar-ejb.jar
        EJBJbossStatelessEar-war.war
EJBJbossStatelessEar-ejb
    Enterprise Beans
    Configuration Files
    Server Resources
    Source Packages
    Test Packages
    Libraries
    Test Libraries
EJBJbossStatelessEar-war
    Web Pages
        WEB-INF
        index.jsp
    Configuration Files
    Server Resources
    Source Packages
    Test Packages
    Libraries
    Test Libraries
```



g:\Laptrinh\Servlet\EJBJbossStatelessEar\*.*

Name	Ext	Size
[..]		<DIR>
[EJBJbossStatelessEar-ejb]		<DIR>
[EJBJbossStatelessEar-war]		<DIR>
[nbproject]		<DIR>
[src]		<DIR>
build	xml	2.463



application.xml ×

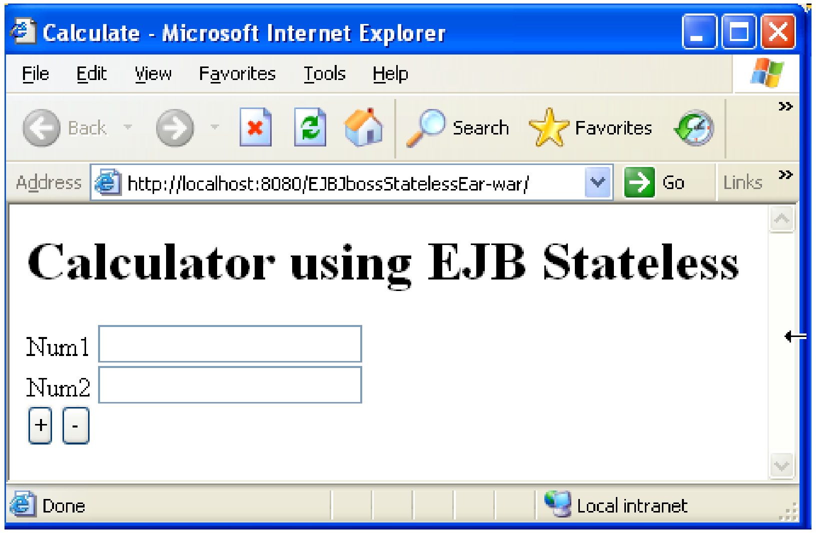
```xml
<?xml version="1.0" encoding="UTF-8"?>
<application version="1.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlr
  <display-name>EJBJbossStatelessEar</display-name>
  <module>
    <web>
      <web-uri>EJBJbossStatelessEar-war.war</web-uri>
      <context-root>/EJBJbossStatelessEar-war</context-root>
    </web>
  </module>
  <module>
    <ejb>EJBJbossStatelessEar-ejb.jar</ejb>
  </module>
</application>
```

Step 2: Creating the new corresponding bean
Step 3: Building/ Modifying the business/callback methods on Beans
Step 4: Mapping the JNDI to beans

- **Creating stateless bean as whole steps in above tutorials in EJB Development process on the xxx-ejb module**

Step 5: Creating the GUI to consumes EJB on web modules

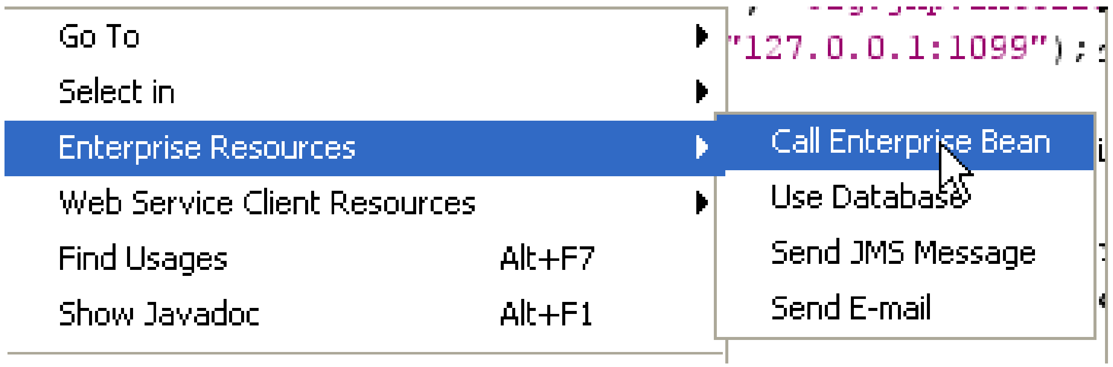
- Creating the GUI application


Calculate - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/EJBJbossStatelessEar-war/ Go Links
Done Local intranet

Calculator using EJB Stateless

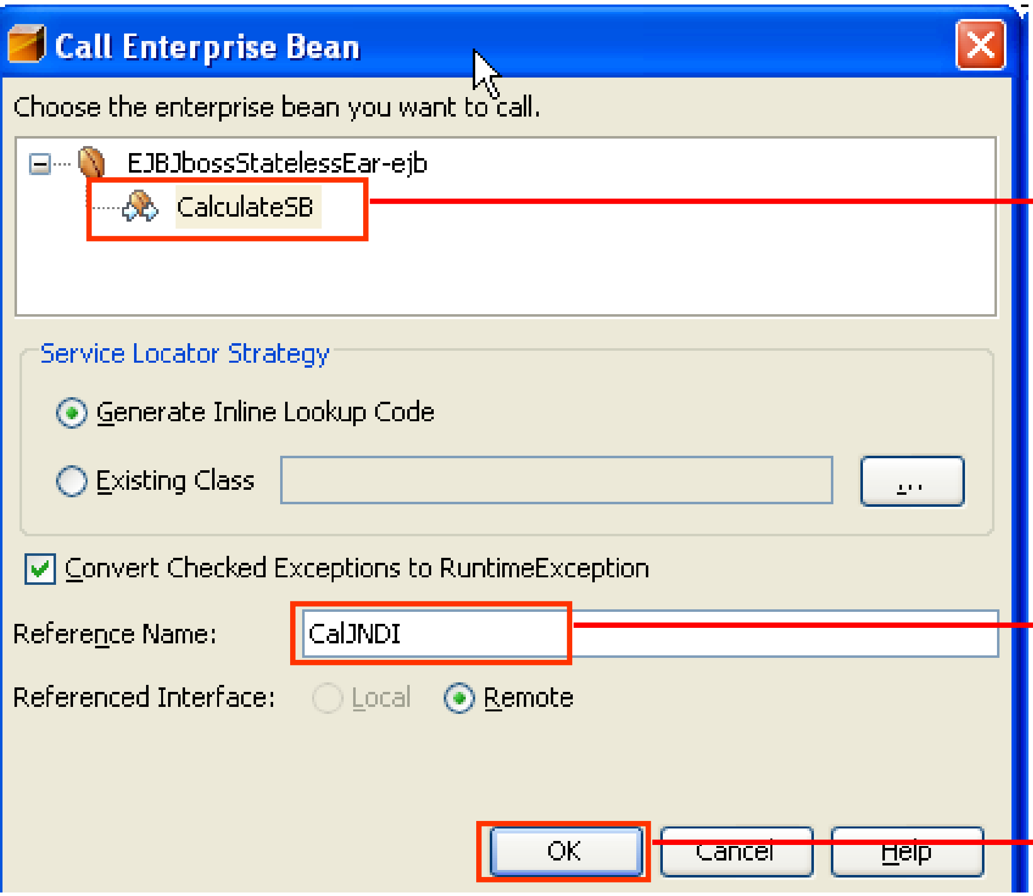
Num1
Num2
+ -

- Creating the Servlet to process and consume the EJB
  - Creating the reference to the EJB on the coding by right click on code
  - Then choose Enterprise Resources, click Call Enterprise Bean



```
"127.0.0.1:1099");
```

Step 5: Creating the GUI to consumes EJB on web modules


Call Enterprise Bean

Choose the enterprise bean you want to call.

EJBJbossStatelessEar-ejb
CalculateSB

Service Locator Strategy

[✓ Generate Inline Lookup Code]

Existing Class

[✓] Convert Checked Exceptions to RuntimeException

Reference Name: CalJNDI

Referenced Interface: Local [✓ Remote]

Choose the appropriate bean

Modify the Reference Name

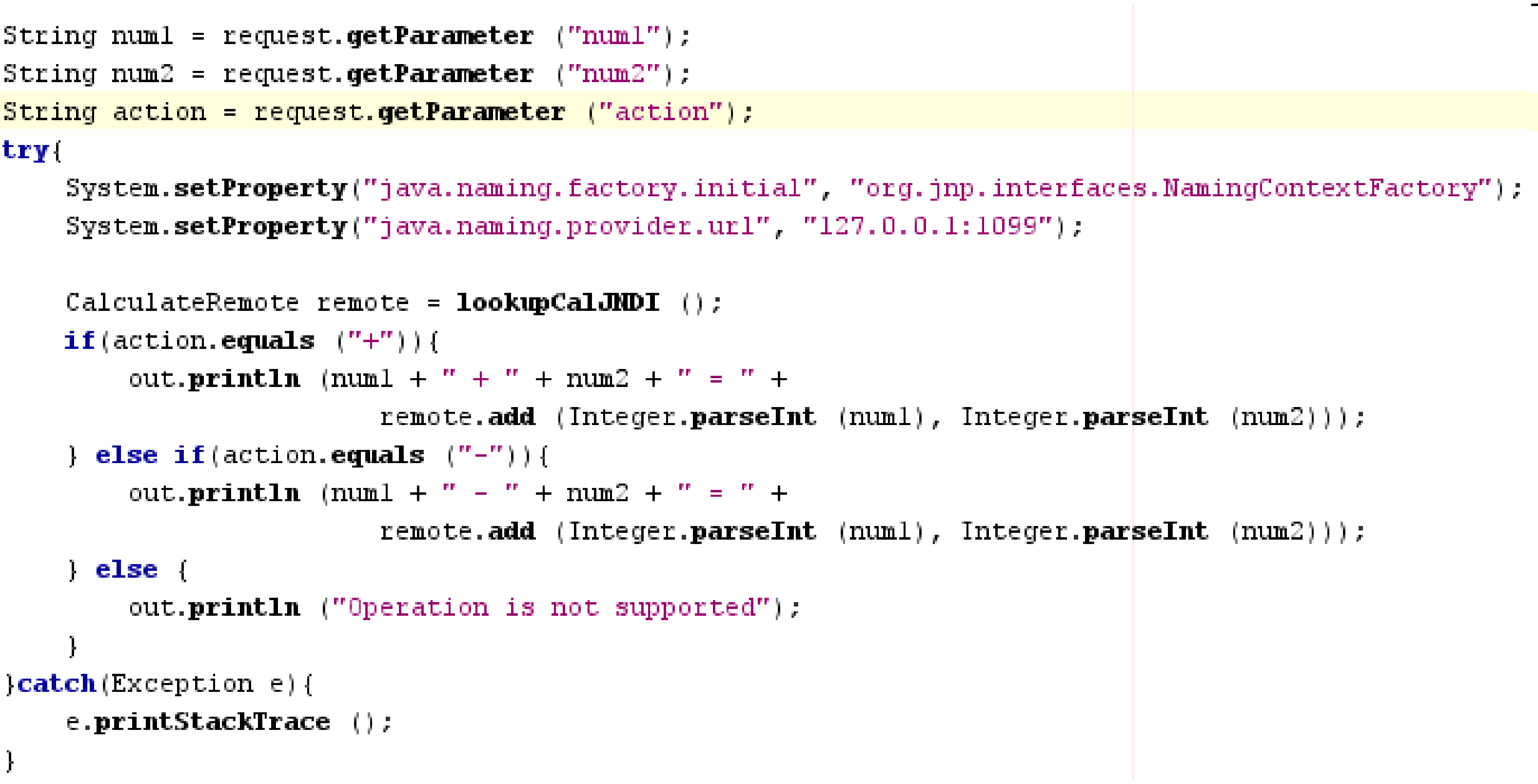
Click Ok Button

```
private CalculateRemote lookupCalJNDI () {
    try {
        Context c = new InitialContext();
        Object remote = c.lookup("CalJNDI");
        CalculateRemoteHome rv = (CalculateRemoteHome)
            PortableRemoteObject.narrow(remote, CalculateRemoteHome.class);
        return rv.create();
    }
    catch(NamingException ne) {
```

Modify the Reference Name that is named in jboss.xml at <jndi-name> tag

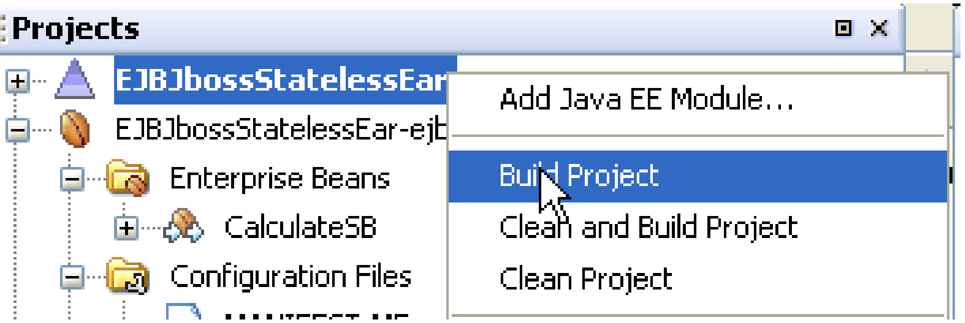
Step 5: Creating the GUI to consumes EJB on web modules

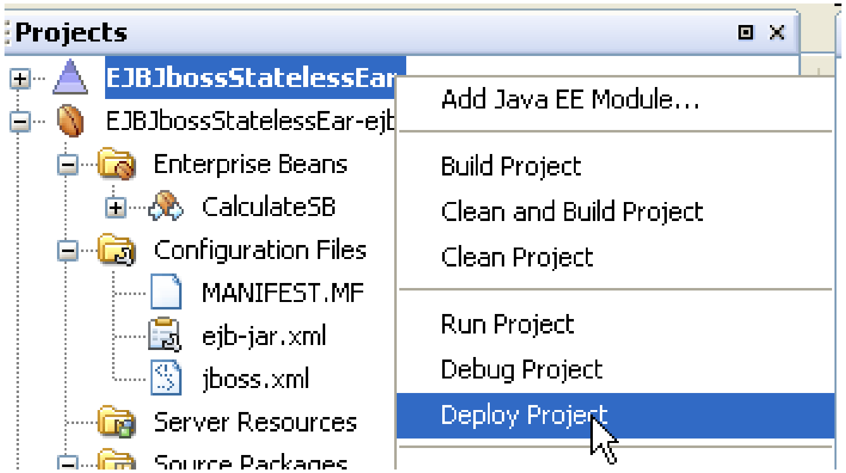
- Modifying the code in servlet as following

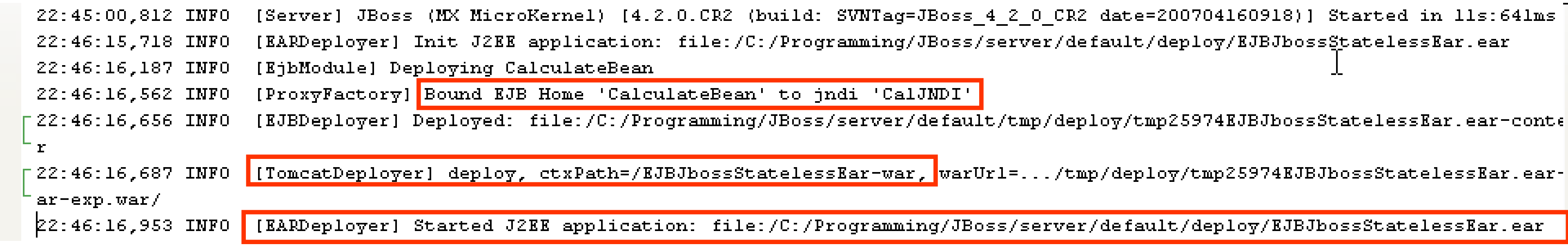


```
String num1 = request.getParameter ("num1");
String num2 = request.getParameter ("num2");
String action = request.getParameter ("action");
try{
    System.setProperty("java.naming.factory.initial", "org.jnp.interfaces.NamingContextFactory");
    System.setProperty("java.naming.provider.url", "127.0.0.1:1099");

    CalculateRemote remote = lookupCalJNDI ();
    if(action.equals ("+")){
        out.println (num1 + " + " + num2 + " = " +
                        remote.add (Integer.parseInt (num1), Integer.parseInt (num2)));
    } else if(action.equals ("-")){
        out.println (num1 + " - " + num2 + " = " +
                        remote.add (Integer.parseInt (num1), Integer.parseInt (num2)));
    } else {
        out.println ("Operation is not supported");
    }
}catch(Exception e){
    e.printStackTrace ();
}
```

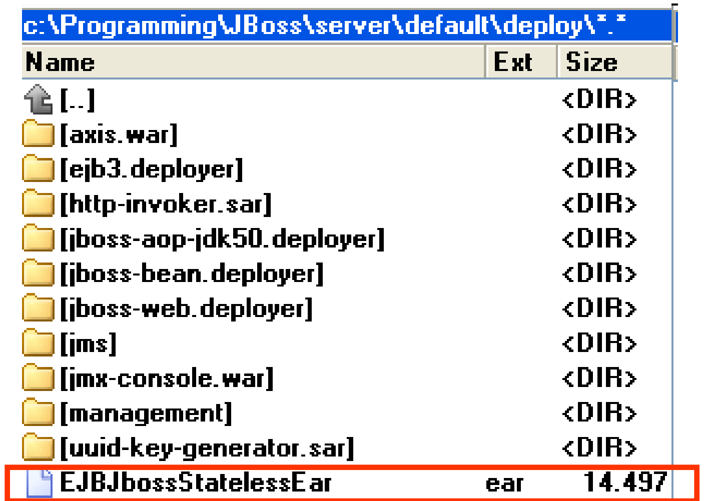
Step 6: Building and Deploying Enterprise application





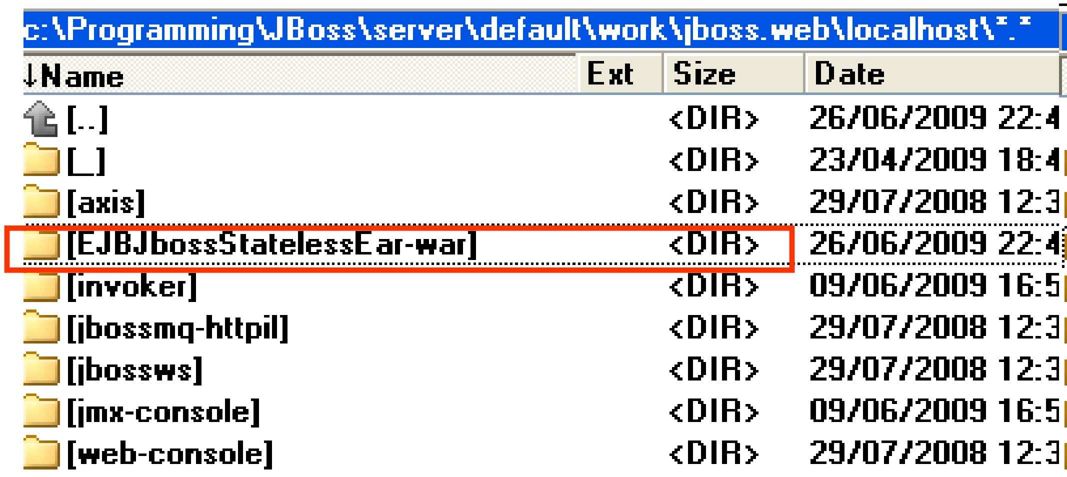


```
22:45:00,812 INFO  [Server] JBoss (MX MicroKernel) [4.2.0.CR2 (build: SVNTag=JBoss_4_2_0_CR2 date=200704160918)] Started in 11s:641ms
22:46:15,718 INFO  [EARDeployer] Init J2EE application: file:/C:/Programming/JBoss/server/default/deploy/EJBJbossStatelessEar.ear
22:46:16,187 INFO  [EjbModule] Deploying CalculateBean
22:46:16,562 INFO  [ProxyFactory] Bound EJB Home 'CalculateBean' to jndi 'CalJNDI'
22:46:16,656 INFO  [EJBDeployer] Deployed: file:/C:/Programming/JBoss/server/default/tmp/deploy/tmp25974EJBJbossStatelessEar.ear-conte
r
22:46:16,687 INFO  [TomcatDeployer] deploy, ctxPath=/EJBJbossStatelessEar-war, warUrl=.../tmp/deploy/tmp25974EJBJbossStatelessEar.ear-
ar-exp.war/
22:46:16,953 INFO  [EARDeployer] Started J2EE application: file:/C:/Programming/JBoss/server/default/deploy/EJBJbossStatelessEar.ear
```



c:\Programming\JBoss\server\default\deploy\*.*

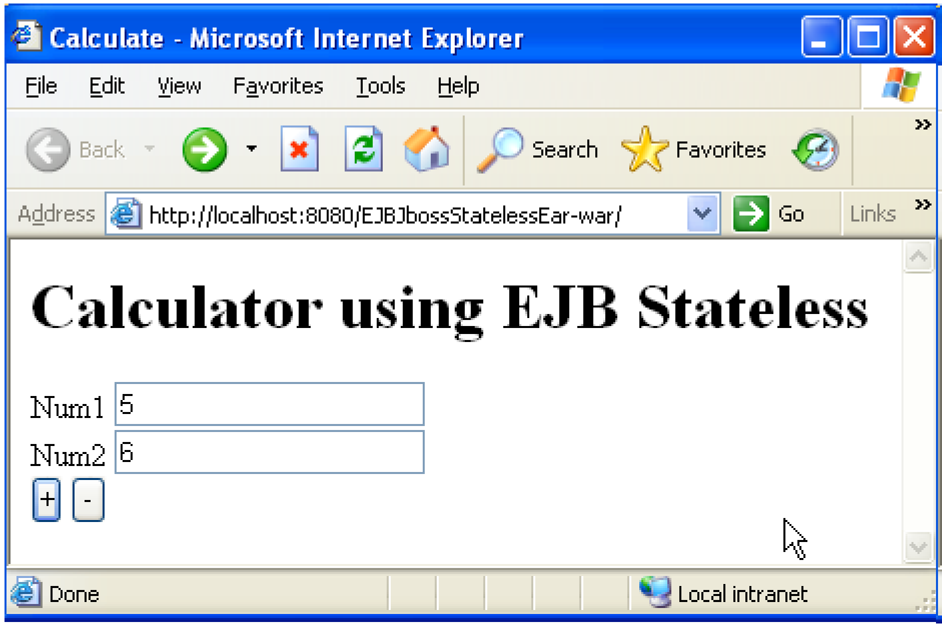
Name	Ext	Size
[..]		<DIR>
[axis.war]		<DIR>
[ejb3.deployer]		<DIR>
[http-invoker.sar]		<DIR>
[jboss-aop-jdk50.deployer]		<DIR>
[jboss-bean.deployer]		<DIR>
[jboss-web.deployer]		<DIR>
[jms]		<DIR>
[jmx-console.war]		<DIR>
[management]		<DIR>
[uuid-key-generator.sar]		<DIR>
EJBJbossStatelessEar	ear	14.497



c:\Programming\JBoss\server\default\work\jboss.web\localhost\*.*

↓Name	Ext	Size	Date
[..]		<DIR>	26/06/2009 22:4
[_]		<DIR>	23/04/2009 18:4
[axis]		<DIR>	29/07/2008 12:3
[EJBJbossStatelessEar-war]		<DIR>	26/06/2009 22:4
[invoker]		<DIR>	09/06/2009 16:5
[jbossmq-httpil]		<DIR>	29/07/2008 12:3
[jbossws]		<DIR>	29/07/2008 12:3
[jmx-console]		<DIR>	09/06/2009 16:5
[web-console]		<DIR>	29/07/2008 12:3

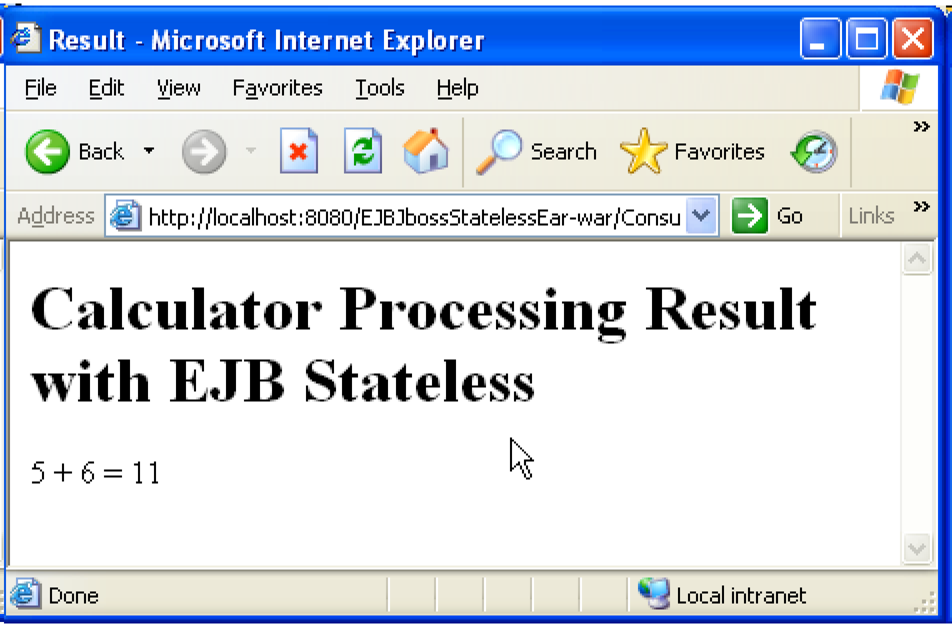
Step 7: Executing the Enterprise Application



Calculate - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/EJBJbossStatelessEar-war/ Go Links
Done Local intranet

Calculator using EJB Stateless

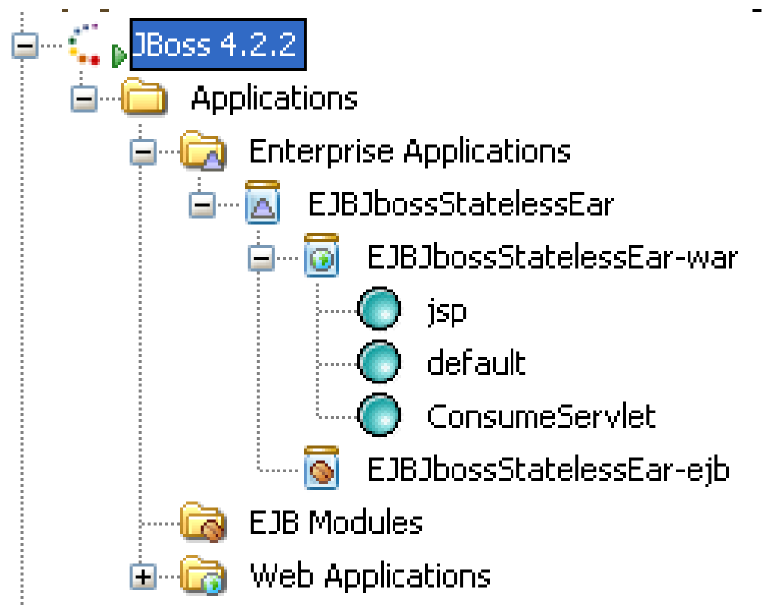
Num1 5
Num2 6
+ -

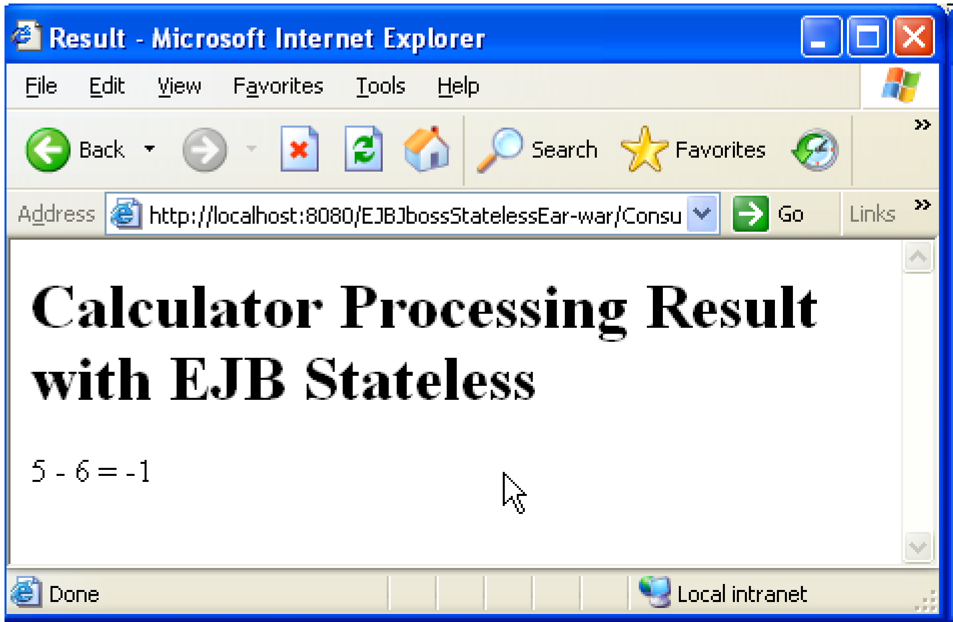


Result - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/EJBJbossStatelessEar-war/Consu Go Links
Done Local intranet

Calculator Processing Result with EJB Stateless

5 + 6 = 11



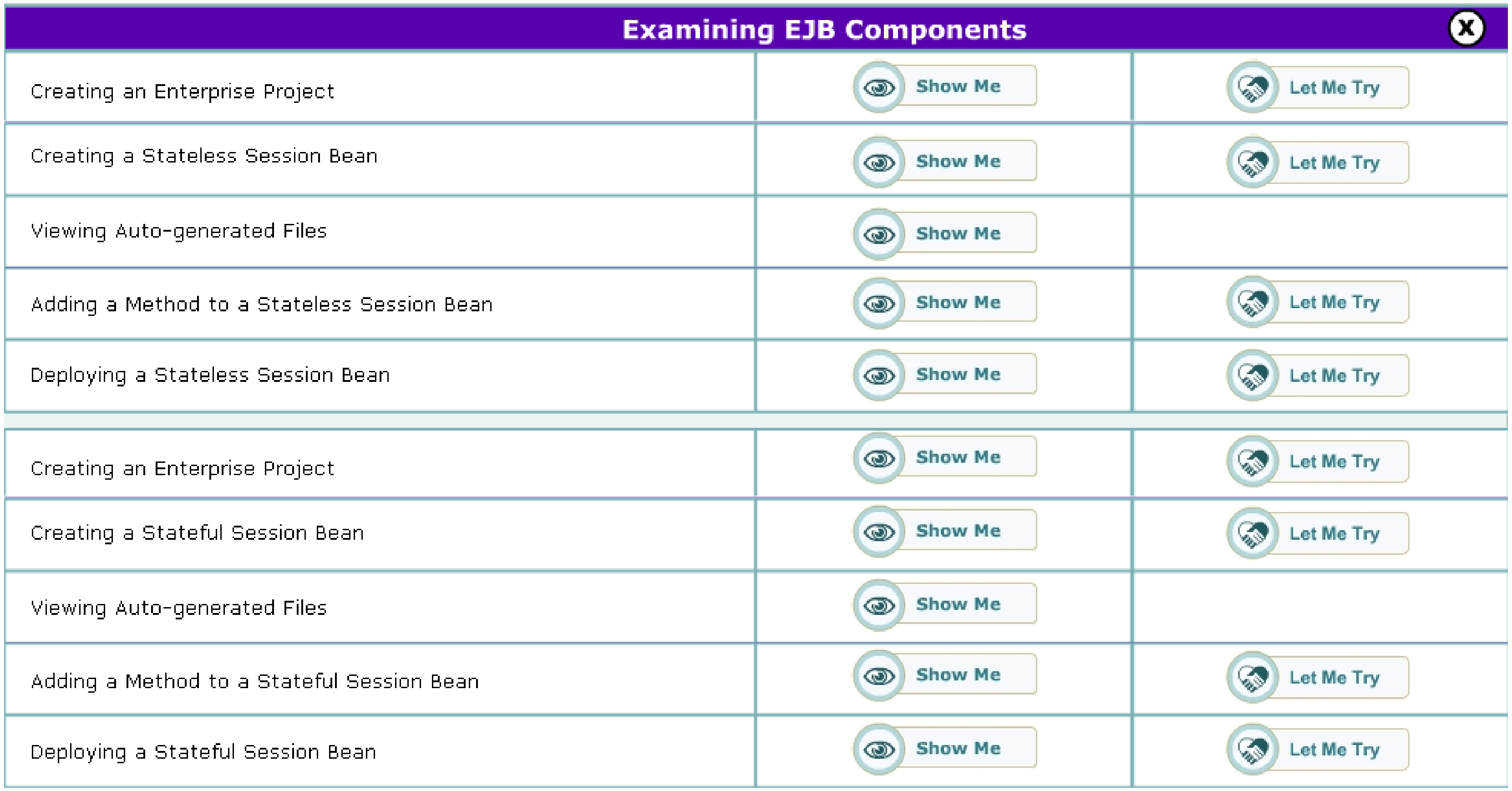


Result - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/EJBJbossStatelessEar-war/Consu Go Links
Done Local intranet

Calculator Processing Result with EJB Stateless

5 - 6 = -1

WORKSHOP ACTIVITIES

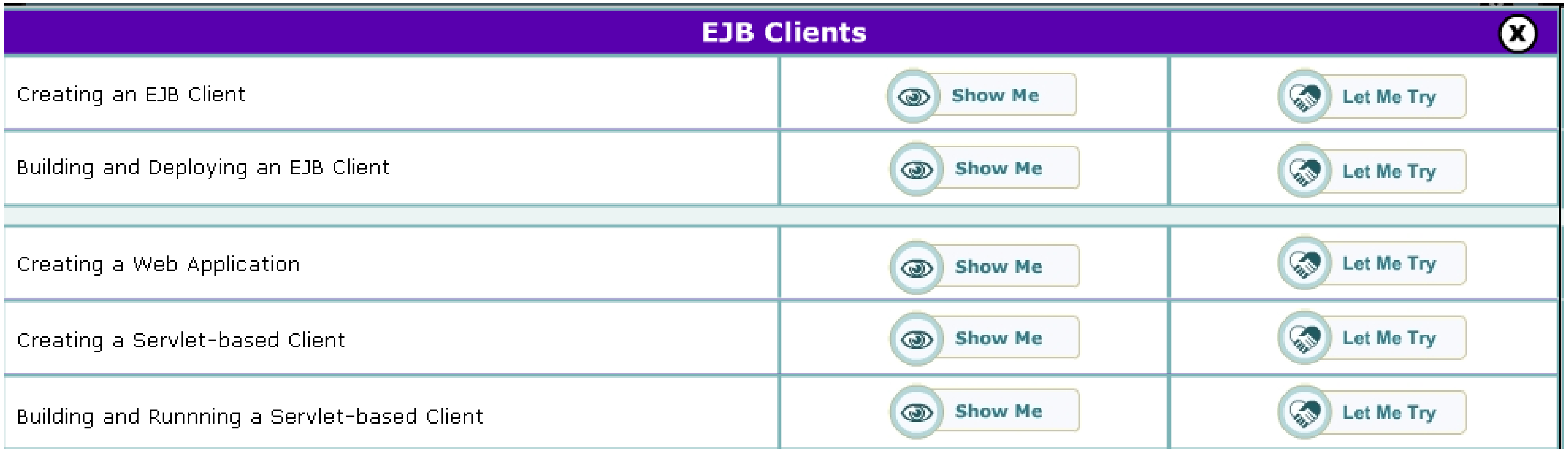


Examining EJB Components

Creating an Enterprise Project	Show Me	Let Me Try
Creating a Stateless Session Bean	Show Me	Let Me Try
Viewing Auto-generated Files	Show Me	
Adding a Method to a Stateless Session Bean	Show Me	Let Me Try
Deploying a Stateless Session Bean	Show Me	Let Me Try
Creating an Enterprise Project	Show Me	Let Me Try
Creating a Stateful Session Bean	Show Me	Let Me Try
Viewing Auto-generated Files	Show Me	
Adding a Method to a Stateful Session Bean	Show Me	Let Me Try
Deploying a Stateful Session Bean	Show Me	Let Me Try

Building the EJB with stateless and stateful on Java Sun Application Server

WORKSHOP ACTIVITIES (cont)



EJB Clients		
Creating an EJB Client	Show Me	Let Me Try
Building and Deploying an EJB Client	Show Me	Let Me Try
Creating a Web Application	Show Me	Let Me Try
Creating a Servlet-based Client	Show Me	Let Me Try
Building and Runnning a Servlet-based Client	Show Me	Let Me Try

Building the EJB client to consume EJB Application

EXERCISES

- Do it again all of demos, workshops, and the following exercises on JBoss Server
- Using stateless to write the programs that can do
  - Building checkLogin() method including 02 parameters username & password using DB
  - Building the sayHello() method printing Welcome message with the name as the parameter passing to sayHello() method
- Using stateful to write the programs that can do
  - Write a Shopping Cart Program using stateful session bean with some functions as add cart, view cart, and remove cart.
  - Write a exchange money program