Software Development using MacroMedia's JRun

B.Ramamurthy

Objectives

- To study the components and working of an enterprise java bean (EJB).
- Understand the features offered by Jrun4 environment.
- To be able to deploy and execute application using JMC of Jrun4.
- Analyzing a problem and arriving at a component-based solution.

Topics for Discussion

- General introduction to Enterprise EJB
- JRUN 4 application server from Macromedia
- Demos on JRUN 4
- From problem statement to J2EE "components" via use case analysis

What are EJBs?

- **Enterprise JavaBeans™** is the server-side component architecture for the J2EE™ platform. EJB™ enables rapid and simplified development of distributed, transactional, secure and portable Java applications.
- An EJB is a collection of Java classes, and a XML file (deployment descriptor) bundled into a single unit.
- Java classes in this bundle follow certain rules and provide specific callbacks for the containers.

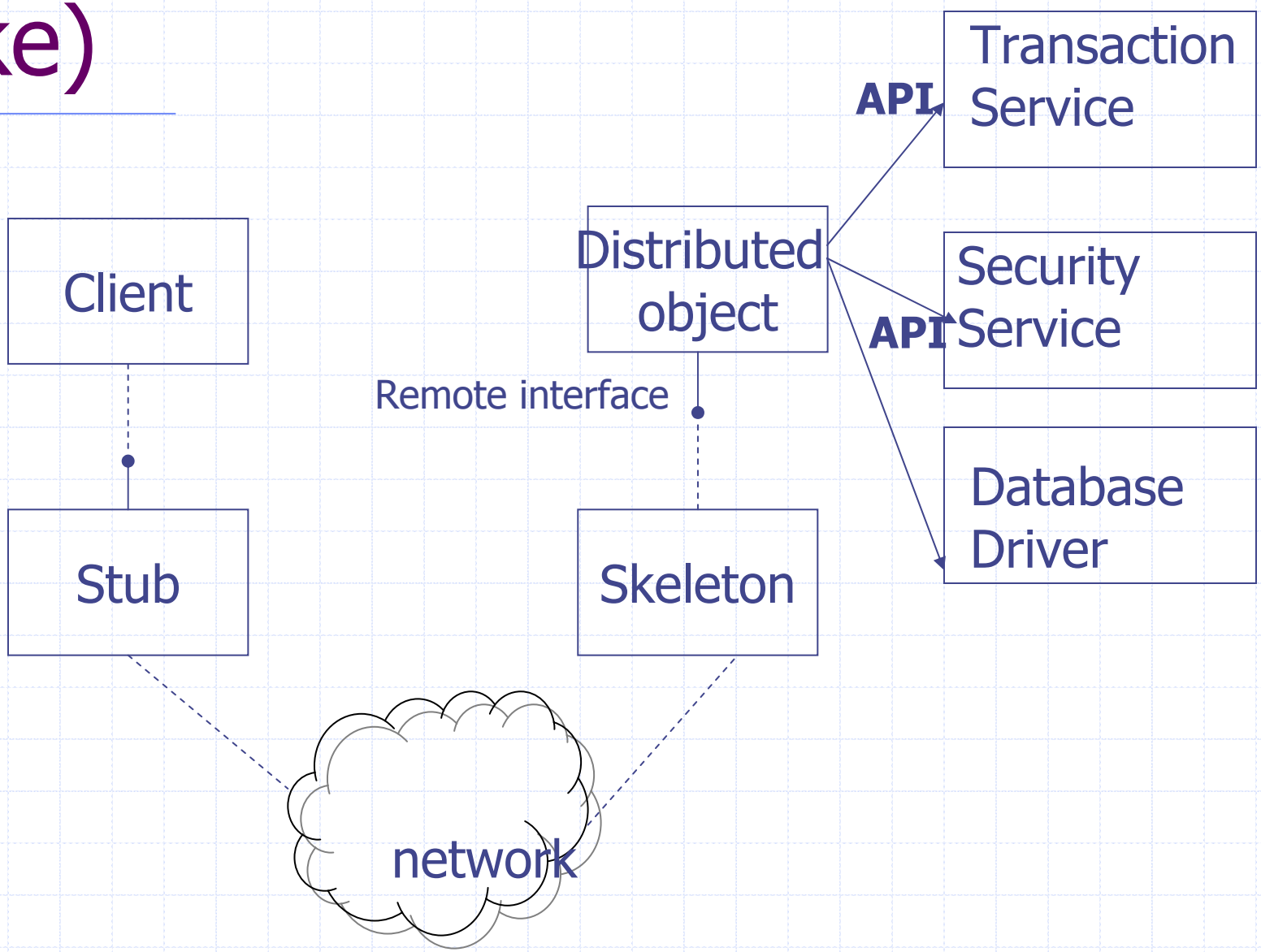
EJB Types

- There are three major types of EJBs:
  - Session: Represents **conversational**/transient state; stateless and stateful
  - Entity bean: Represents a **persistent** relation in the relational DB. Bean-managed persistence (BMP), container-managed persistence (CMP)
  - Message-driven: Alternative to remote method call: asynchronous and used for realizing loose coupling among systems. Uses **messaging** middleware.
- Lets look at Ed Roman's view of the EJB technology.

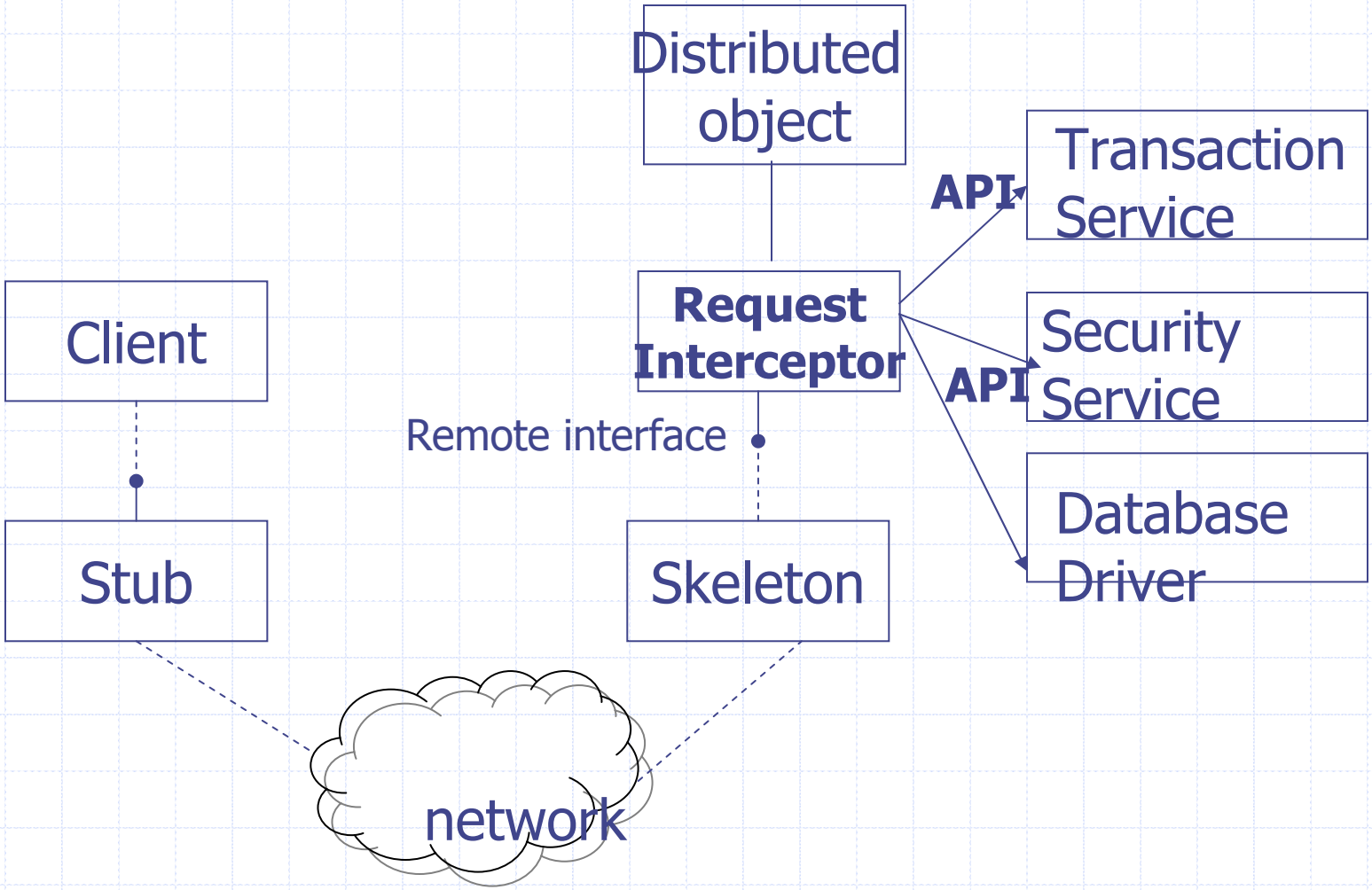
Examples of Session beans calling entity beans

Session bean	Entity bean
Bank teller	Bank account
Credit card authorizer	Credit card
Order entry form	Order, line item
Catalog engine	Product
Auction broker	Bid, item
Purchase order router	Purchase order

Explicit Middleware (CORBA-like)



Implicit Middleware (Through declarations as in J2EE)



Implicit VS Explicit services

- We used to include the services such as transaction, security, data base drivers, etc. programmatically making every programmer learn the inner details all the possible services needed in an application.
- Now we can declare what we want and let the container take care of carrying it out.
- Container is the silent partner: container's glue code tools are responsible for transforming an enterprise into a fully managed, distributed server-side component.
- Declaration is done through a XML deployment descriptor.

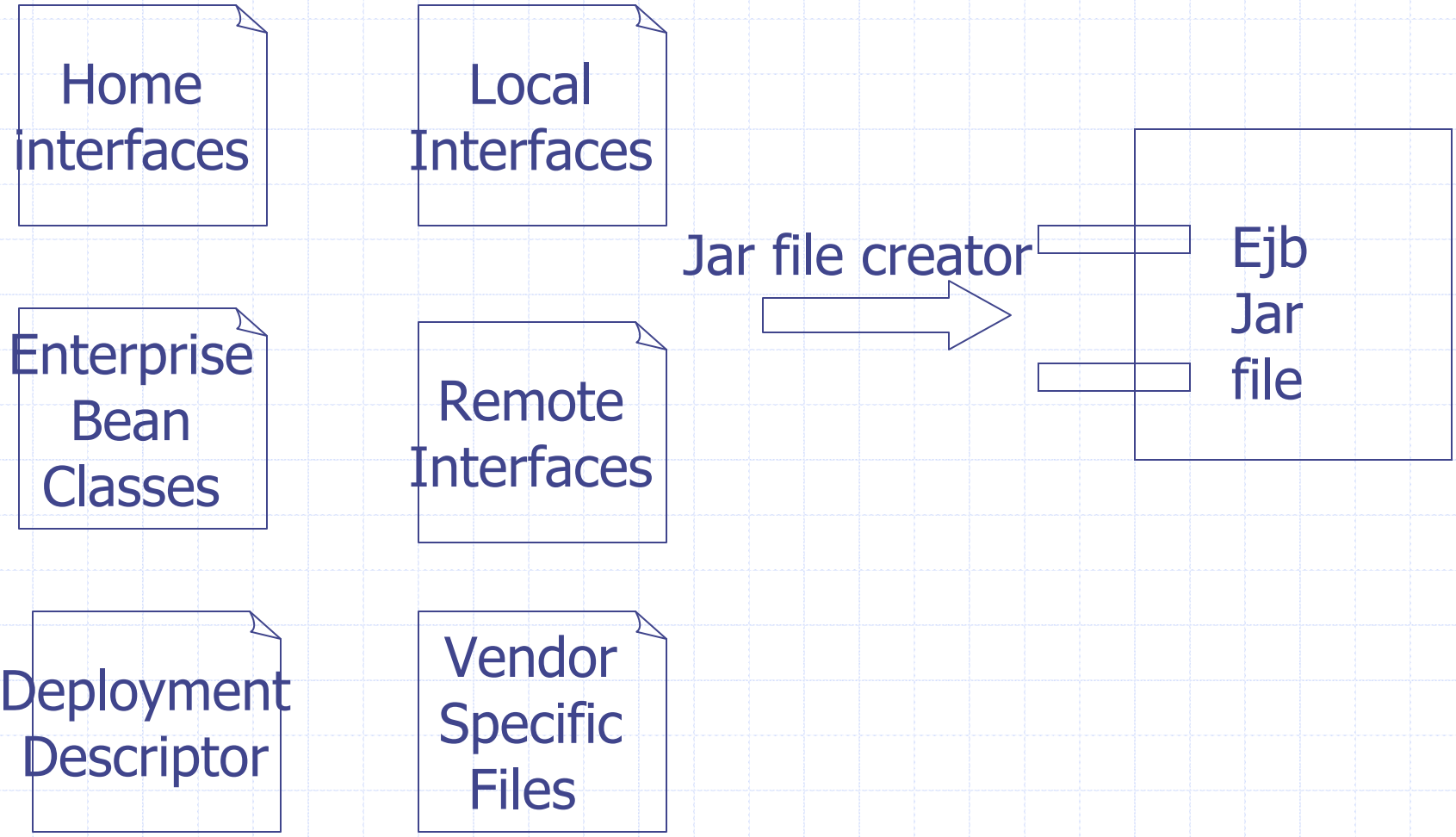
Parts of EJB

- **EJB class** that implements the business methods and life cycle methods; uses other helper classes and libraries to implement.
- **Client-view API**: consists of EJB home interface and remote interface.
  - Home interface: controls life cycle : invokes Home Object methods: create, remove, find methods
  - Remote interface: to invoke the EJB object methods

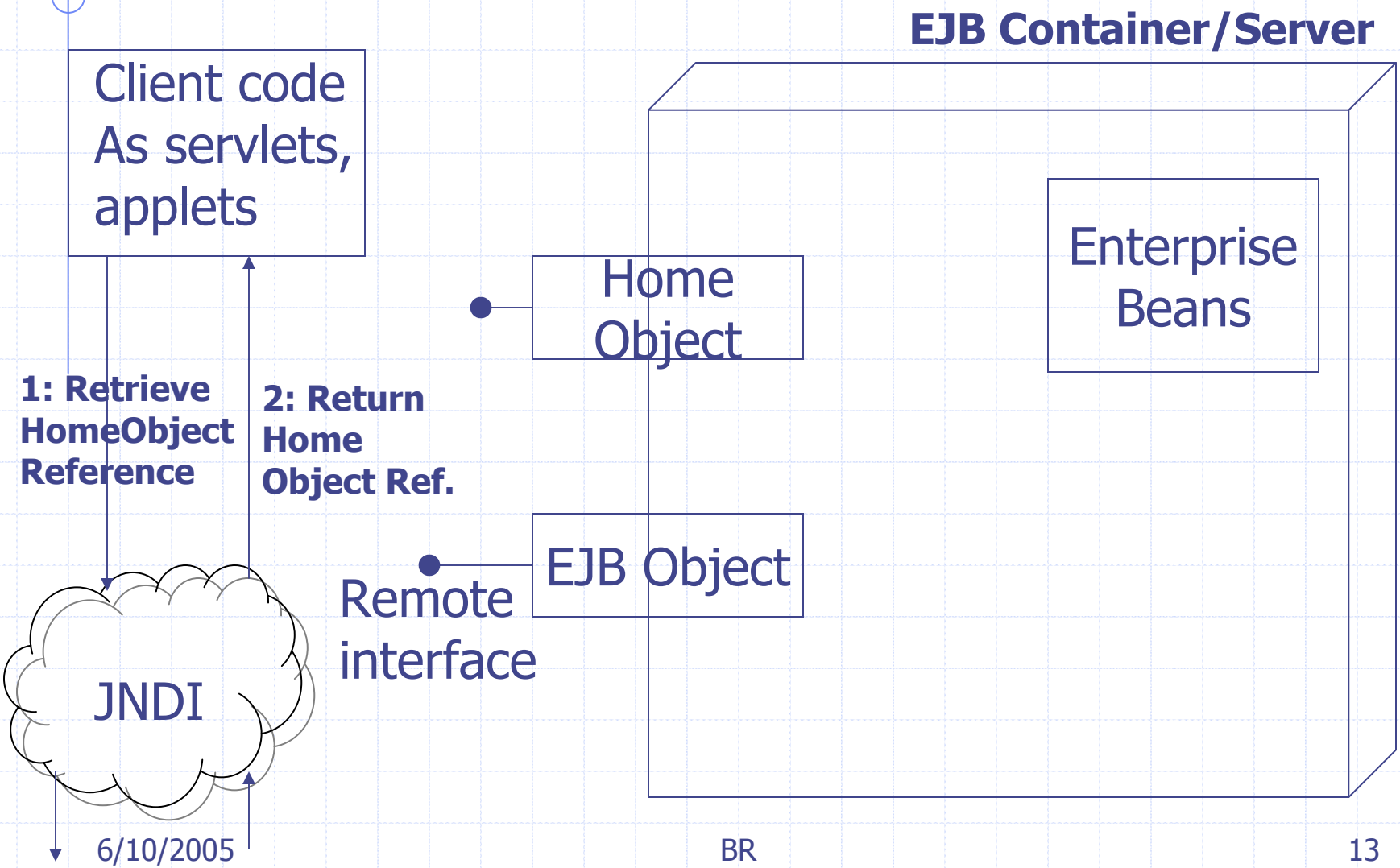
Parts of EJB (contd.)

- **Deployment Descriptor**: XML document for bean assembler and deploy tool;
  - A declaration about EJB environment needed for customizing the bean to the operating environment.
  - Container Runtime services that can be **declared** include: transactions, security,distribution,load balancing, multithreading, persistence, failure recovery, resource pooling, state management, clustering..

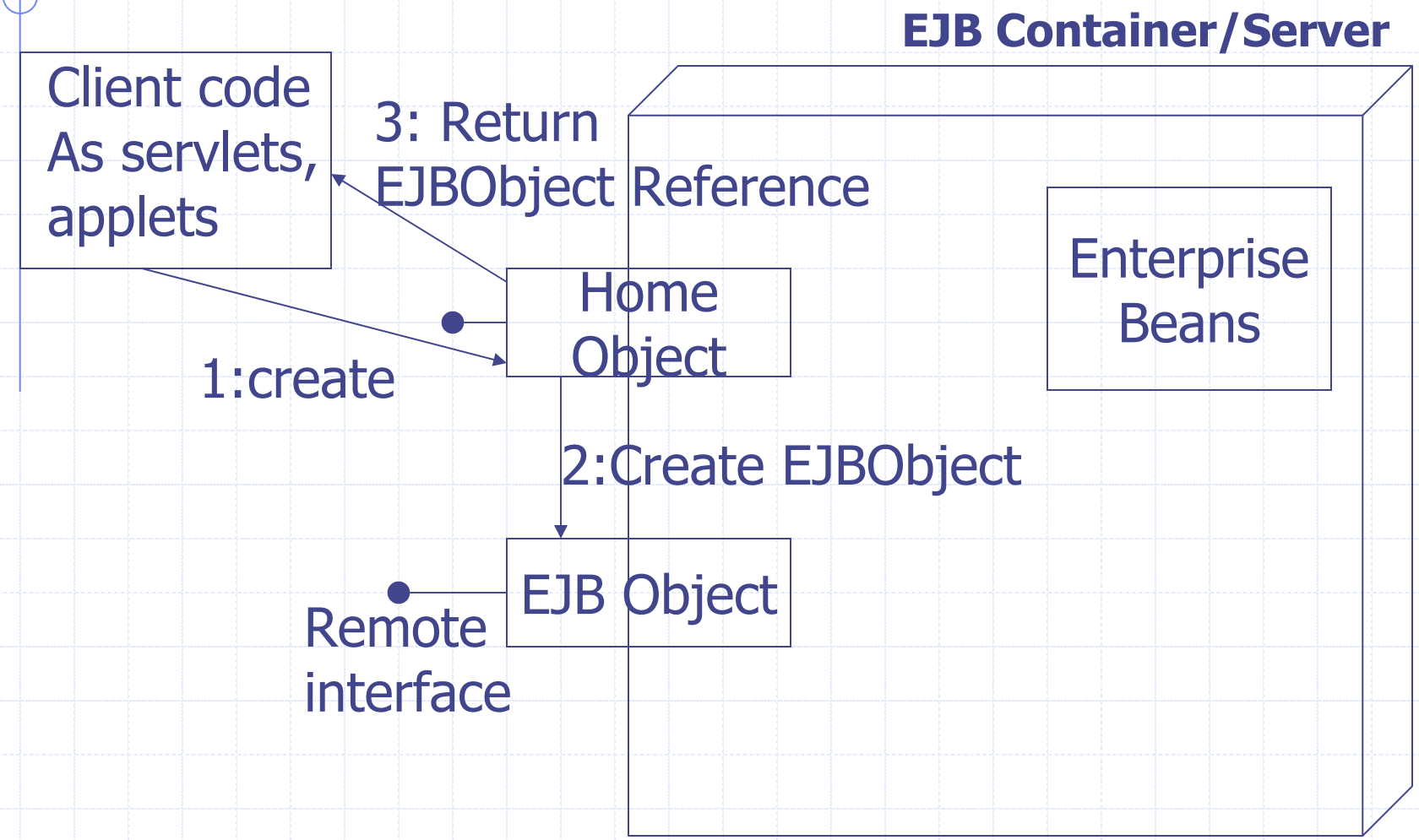
Creating a EJB-jar file



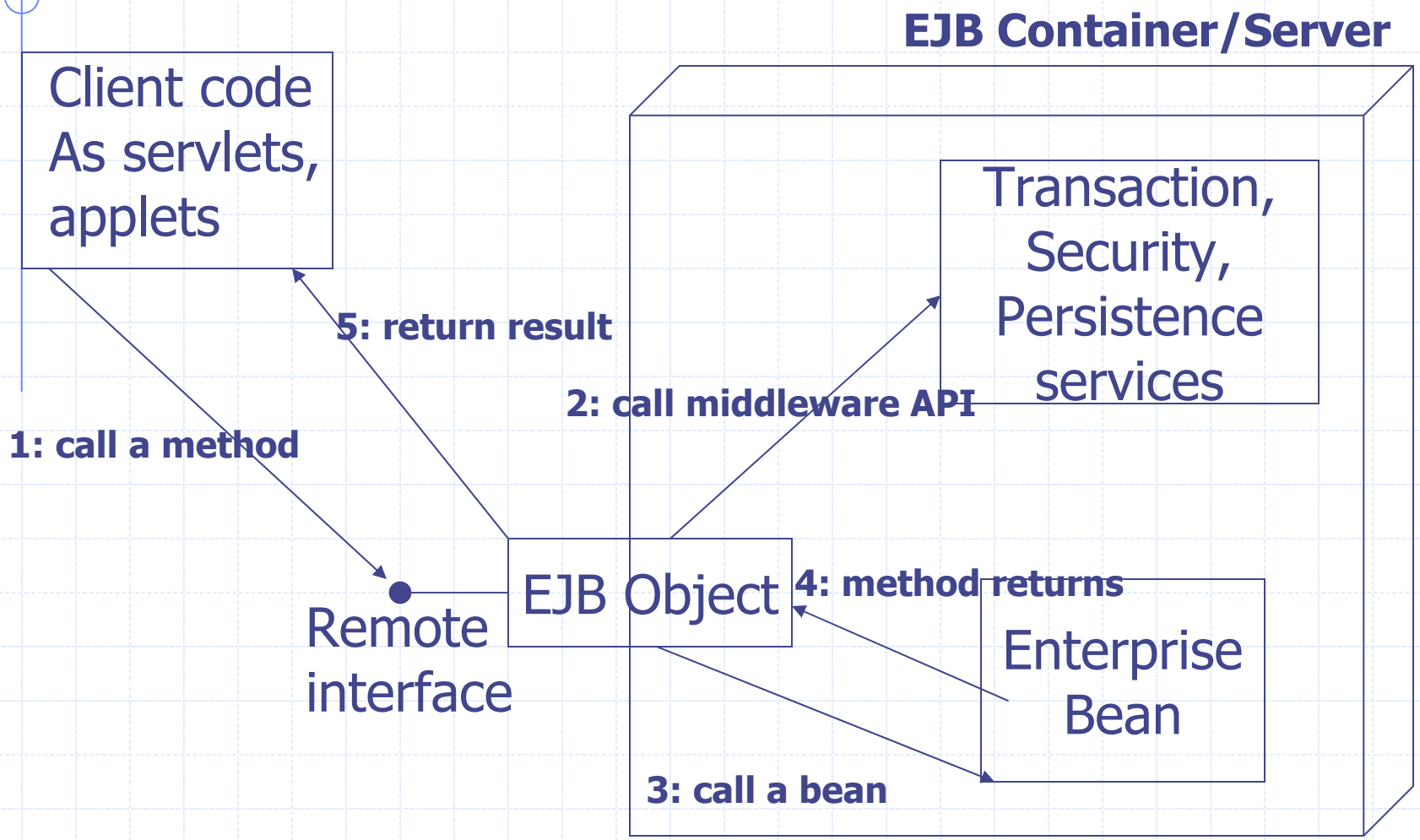
Step 1: Retrieve Home Object reference using JNDI



Step 2: Retrieve EJBObject using Home Interface and Objects



Step 3: Invoke Business Methods Using Remote Interface and EJB Objects



JRUN4

- JRun (J2EE) Server can be started, stopped, refreshed, and status checked three different ways,
  - From command line
  - Using a JLauncher
  - Using a web-based JRun Management Console (JMC)
- Demo1: Jrun4 Environment

JRun4 Development Version

- Comes with three servers: admin, default and sample
- Admin: is reserved for running administrative tools such as JMC. So you are advised not to do any application development on this. At port 8000.
- Samples: has many applications already deployed for you to study the working code for various J2EE technologies. At port 8200.
- Default: is where we will do most of our development and deployment. At port 8100.
- Demo2: Lets study the application “compass” served by the “samples” server.

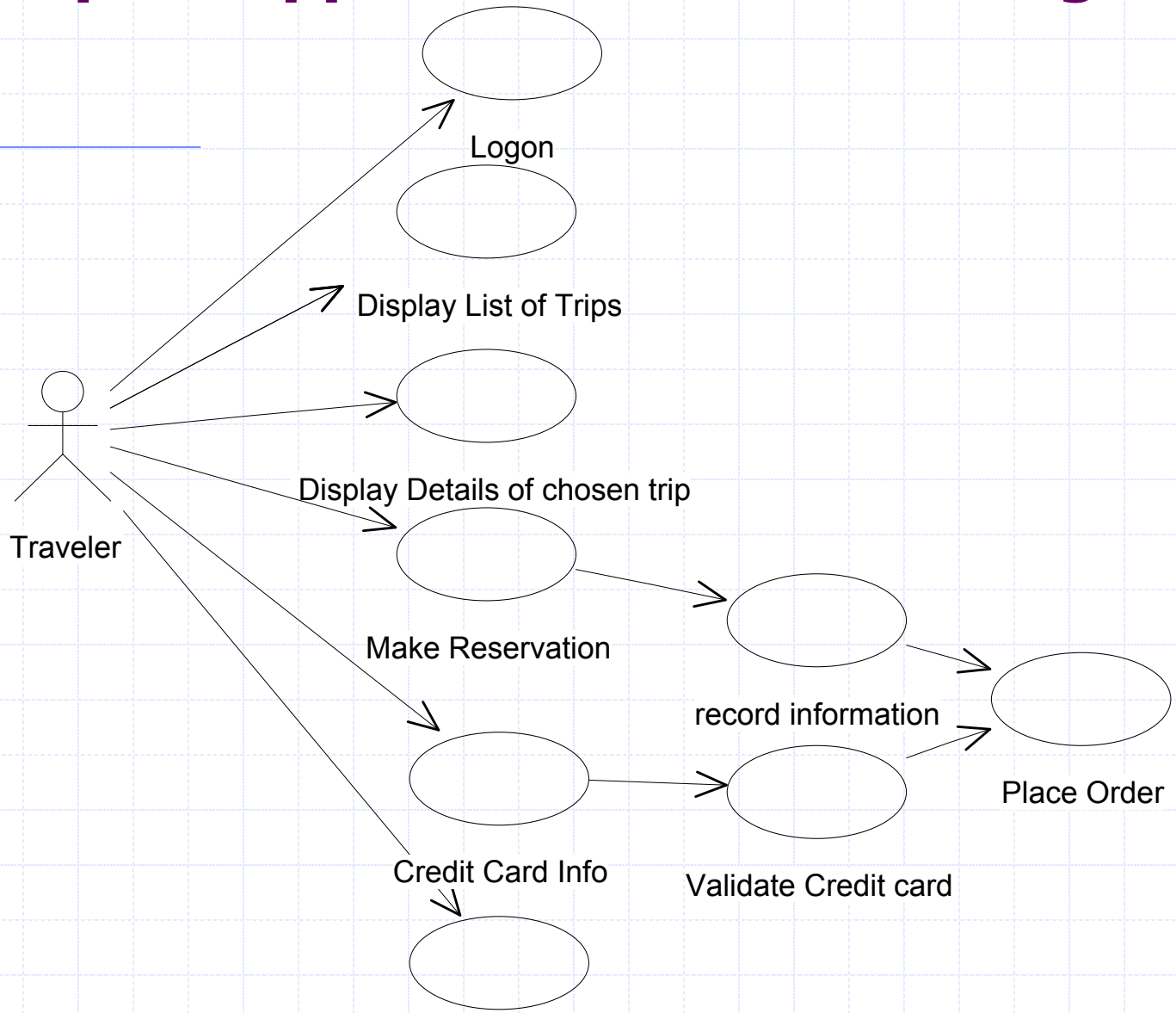
Demo 3: Add a server "tutorial" at port 8101

- Add a server tutorial. We can do hot deployment by copying over the compass application.
- Also see how the data access to the pointbase data base is declaratively added to the server using the JMC.
- Look around the other features offered by JMC.
- Observe how easy it is to delete, refresh, and stop a server using the various iconized buttons.
- Study the explorer window on the left pane of JMC to see the various declarative customization possible for your applications.

Compass Online Vacation Reservation System

1. User **logon** for authentication using a registered user id and password.
2. Application **home** provides a list of trips you can choose from. Click the name of a trip to get details about that trip
3. **Trip details** provides details about the selected trip. Click the book button to book the trip.
4. **Reservation** allows you to enter payment information. Select a credit card type, specify a credit card number and the expiration date.
5. **Confirmation** displays your confirmation number if the reservation was successful, and an error message if a problem occurred.

Compass Application: Use Case Diagram



Compass Application: From Use Cases to Component List

- Use JSP for all web user interface: Logon.jsp, home.jsp, Triplist.jsp, Atrip.jsp, Reservation.jsp (includes confirmation/denial use case), creditcard.jsp
- Data access for non-conversational JSP can be direct.
- For reservation and credit card we have a conversation with the user, so will have a stateless session bean ReservationBean and CreditCardBean. These two are **remote**ly accessible beans.
- Finally all the information gets conveyed to a entity bean OrderBean for storing the order information. This could be a "local" bean not (remotely) accessible to the client.

Business Entities, Processes and Rules

- EJB Applications organize business entities, processes and rules into components.
- Entity: is an object representing some information maintained in the enterprise. Has a "state" which may be persistent.
  - Example: Customer, Order, Employee
- Process: Is an object that typically encapsulates an interaction of a user with business entities. A process typically updated and changes the state of the entities.
  - Example: Sales, pricing, place order, reservation
- Rules: constraints on the state of the entities.
  - Example: overDraft, creditLow, validity

Choosing the type of Bean

- Entity (business entity) is typically implemented as entity bean or a dependent object of an entity bean.
- Conversational (business) process as a session bean.
- Collaborative bean as an entity bean.
- Any process that requires persistence is implemented as an entity bean.
- When exposure to other applications are not needed for an entity or process (local/private process) then they are implemented as bean dependent objects. You may use local EJBs for this purpose if container services are needed.

Review

- We studied the basics of Enterprise Java Beans. We will develop on these concepts further in the next lectures.
- We also looked JRUn4 environment: its JLauncher, JRun Management Console (JMC), and servers and deployment of applications.
- We looked at how to analyze a problem to arrive at a set of components (web components and different types of ejb components).

On To EJBs

- Understand the parts of the EJBs
- Package the EJBs and deploy them
- Design web application to access the EJBs
- Understand the various descriptors and directory structure
- Understand local naming conventions and JNDI naming conventions

Designing Components

- Designing components: esp. enterprise java beans: session beans: stateless and stateful.
- Connecting web component to an EJB.
- Enterprise application (ear) directory structure and naming conventions; hot deploy.
- XYZ-INF : META-INF, WEB-INF, SERVER-INF, web.xml, ejb-jar.xml, jrun.xml.
- Analyzing compass application of the samples server; JNDI and java naming.

Contents of an Enterprise Bean

- Interfaces: The remote and home interface for remote access. Local and local home accesses for local access.
- Enterprise bean class: Implements the methods defined in the above interfaces.
- Deployment descriptor: An XML file that specifies information about the bean such as its type, transaction attributes, etc.
- Helper classes: non-bean classes needed by the enterprise bean class such as utility and exception classes.

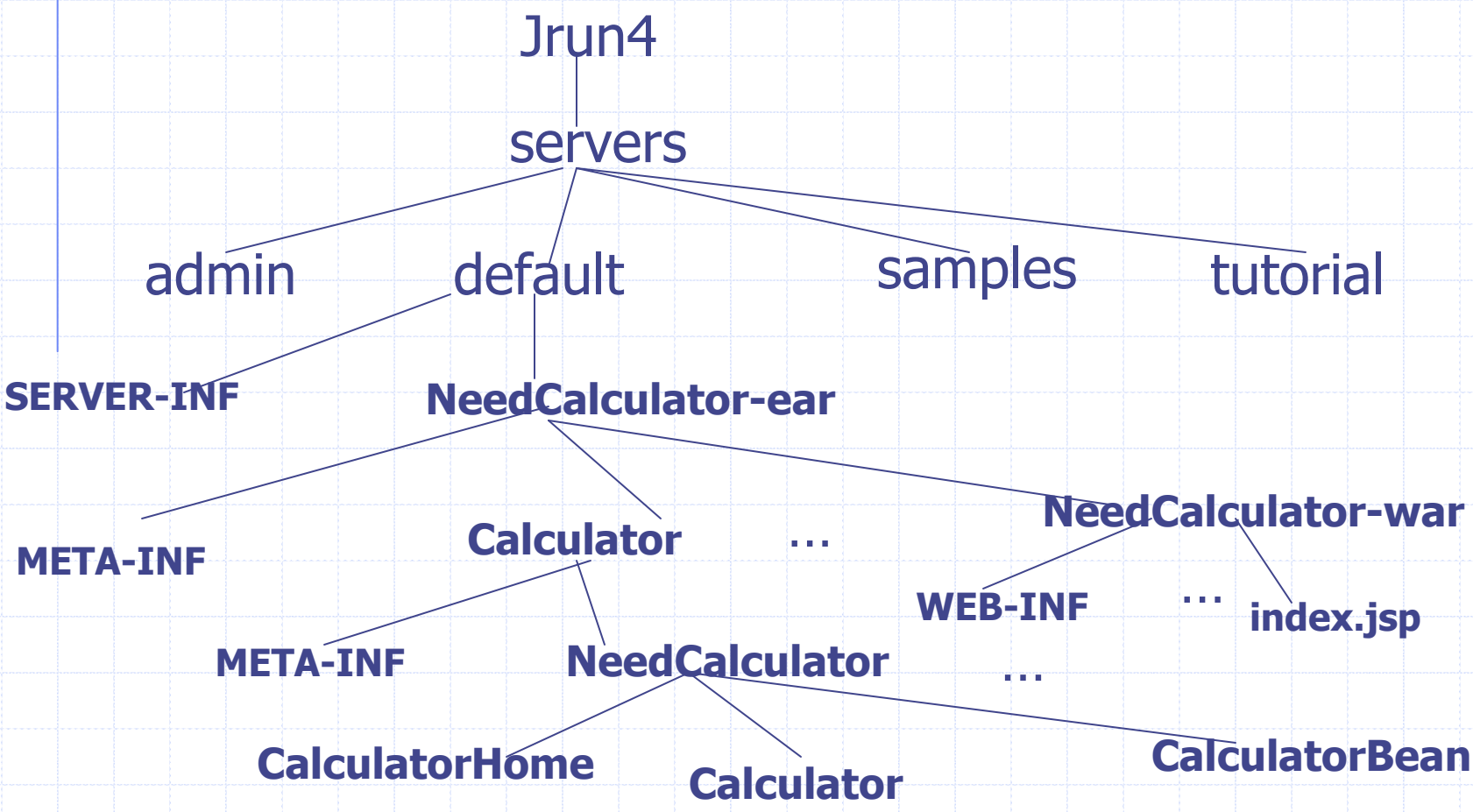
Naming Conventions

Item	Syntax	Example
Directory Name	**<name>-ear**	**Account-ear**
EJB JAR display name (DD)	**<name>**	**Account**
Enterprise bean class	**<name>Bean**	**AccountBean**
Home interface	**<name>Home**	**AccountHome**
Remote interface	**<name>**	**Account**
Local home interface	**Local<name>Home**	**LocalAccountHome**
Local interface	**Local<name>**	**LocalAccount**

Session Beans

- Tuition Need Calculator application.
  - It takes in many numbers and uses handful of formulae to come up a dollar amount for financial need for attending a given college.
  - We will implement this using a session bean.

Directory Structure for Need Calculator examples



INF Directories

- Contain the descriptor files
- Descriptor files are in XML
  - Can be auto-generated by tools.
- SERVER-INF has configuration of the server such as users, security.
- META-INF directory for ejb has ejb-jar.xml (ejb specific details) and jrun-ejb-jar.xml (ejb services specific details)
- WEB-INF directory for web applications has web.xml and jrun-web.xml.
- In general, xyz.xml and jrun-xyz.xml separate the application-server dependent and independent descriptors respectively.

Session Beans

- Session beans implement the "façade" design pattern, typically facilitating the data transfer between the user interface and the business logic beans (possible entity beans).
- These are **conversational** as opposed to entity beans being **transactional**.
- Stateless session beans don't remember anything about the user data so can be freely shared.
- Lets say we have 5000 users accessing your system, instead of 5000 sessions running, 50 stateless sessions can be shared among the users.

Home Interface: CalculatorHome.java

```
package NeedCalculator;

import javax.ejb.EJBHome;

import java.rmi.RemoteException;

import javax.ejb.CreateException;

import java.util.Collection;

public interface CalculatorHome extends EJBHome

{

        public Calculator create() throws
RemoteException,  CreateException;

}
```

Remote Interface: Calculator.java

```
package NeedCalculator;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Calculator extends EJBObject
{
   public double calc (double cost, double avail) throws
   java.rmi.RemoteException;
}
```

Session Bean: CalculatorBean.java

```
package NeedCalculator;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;
public class CalculatorBean implements SessionBean
{  private SessionContext context;
   public double calc (double cost, double avail) {
      return (cost – avail); }
   public CalculatorBean() {}
```

CalculatorBean (contd.)

```
public void ejbCreate()  throws CreateException { }
public void setSessionContext(SessionContext context) {
                    this.context = context; }
public void ejbRemove() { }
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbPostCreate() {}
}
```

Descriptor (ejb-jar.xml)

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>Calculator</display-name>
      <ejb-name>Calculator</ejb-name>
      <home>NeedCalculator.CalculatorHome</home>
      <remote>NeedCalculator.Calculator</remote>
      <ejb-class>NeedCalculator.CalculatorBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <security-identity>
        <use-caller-identity />
      </security-identity>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Creating the files

- Can do it manually using templates of previous applications.
- Can use JWizard that will automatically generate the XML descriptors.
- Can use XDocLet which will automatically generate files and regenerate to reflect any changes.
- Other methods from a integrated development environment such as Sun Studio, and IntelliJ.

Deployment

- Hot deploy: This is most convenient way to deploy the components. Lets try this with compass example.
- Create the standard directory structure either manually or using tools. Place the files in the appropriate directories.
- Start the server or restart the server.
- If there are errors, correct them, recompile and restart/redeploy.

Web Application to test the NeedCalculator

- We will write a very simple JSP file called index.jsp that:
  1. Resolves the JNDI name from the initial context to create the home directory.
  2. Narrows and casts the object reference obtained in the above steps to the home object of the NeedCalculator.
  3. Creates the EJbObject representing the remote interface of the Calculator.
  4. Invokes the calc method on the reference obtained in step3.

NeedCalculator-war/index.jsp

```
<%@ page import="NeedCalculator.*" %>
<html>
<head>
<title>Need Calculator</title>
</head>
<body>
<%
        try {
        javax.naming.InitialContext ctx = new javax.naming.InitialContext();
                Object obj = ctx.lookup("java:comp/env/ejb/Calculator");
```

Web Application (contd.)

```
CalculatorHome home =
(CalculatorHome)javax.rmi.PortableRemoteObject.narrow(ob
j, CalculatorHome.class);
                    Calculator needCal = home.create();
                    double d= needCal.calc(10000, 5000);
                    out.println("Your Need is = $" + d);
%>
<br>Thank you.Your need has been calculated.<br><br>
<%
          } catch (Exception e) {
%>
                    <br>Sorry, unable to calculate need.
```

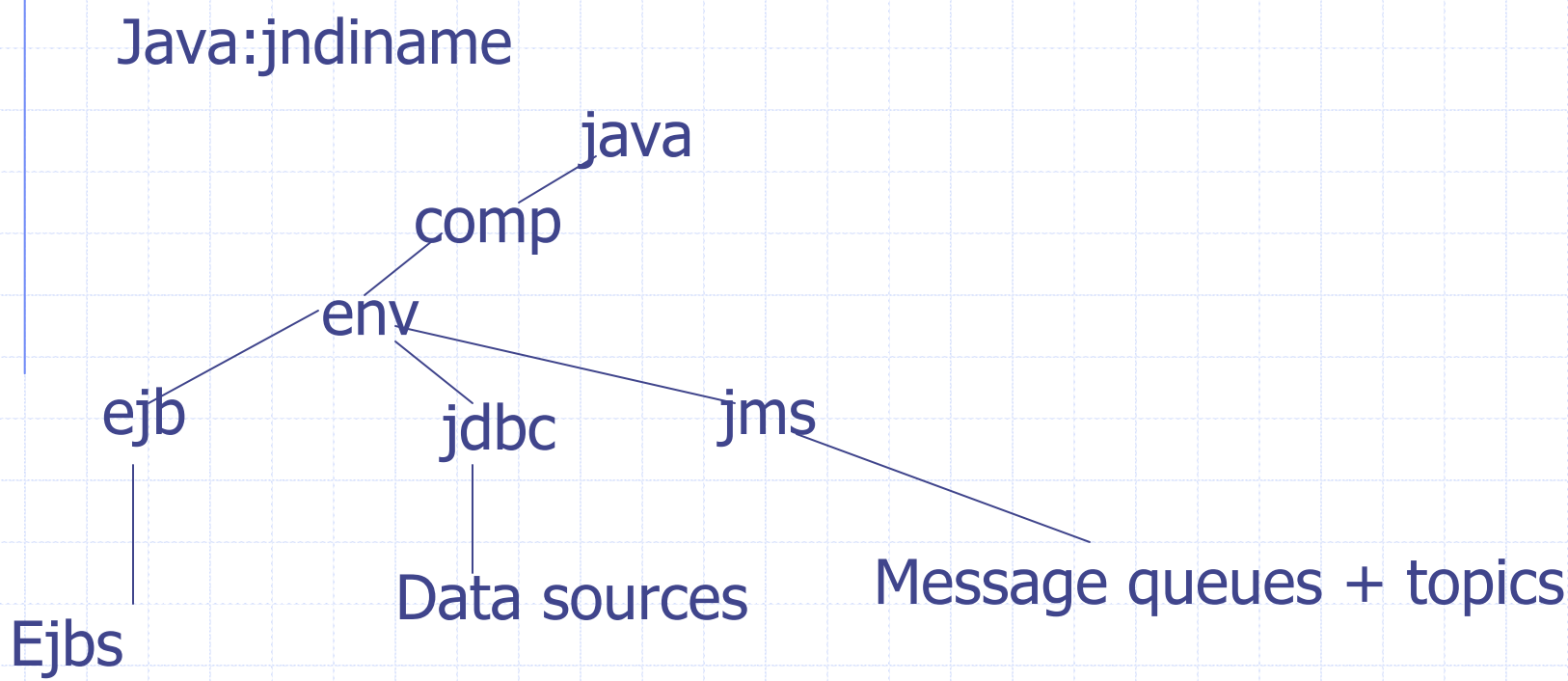
WEB-INF/web.xml

```
<welcome-file-list>
 <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<ejb-ref>
 <description>Calculator session bean</description>
 <ejb-ref-name>ejb/Calculator</ejb-ref-name>
 <ejb-ref-type>Session</ejb-ref-type>
 <home>NeedCalculator.CalculatorHome</home>
 <remote>NeedCalculator.Calculator</remote>
</ejb-ref>
```

JNDI Names

- **This application uses ejb-refs so that clients can always locat the ejb under the java:comp/env environment naming context (ENC).**
- **The jrun-web.xml file maps the ejb-ref-name to the actual JNDI location.**
- **Clients can then lookup the EJB using either the actual JNDI location or java:comp/env/*ejb-ref-name***
- **If there is no tags corresponding to ejb-ref then lookup will be to the actual name “Calculator” of the java naming service.**

JNDI Names (contd.)



java:comp/env/ejb/Calculator
java:comp/env/jdbc/compass
java:comp/env/jms/newsQueue