



The Details of Writing Enterprise Java Beans

Your Guide to the Fundamentals
of Writing EJB Components



P. O. Box 80049
Austin, TX 78708
Fax: +1 (801) 383-6152

Presentation Outline

In this presentation we will discuss:

- EJB Development Overview
- The Enterprise Bean Class
- The Remote Interface
- The Home Interface
- Deployment Descriptors and EJB-JAR files

Where are we?

- *EJB Development Overview*

- Explore the steps needed to write a simple Enterprise Java Bean
- Identify all the different parts of an Enterprise Bean

- The Enterprise Bean Class

- The Remote Interface

- The Home Interface

- Deployment Descriptors and EJB-JAR files

The "EJB Development Process"

- ▼ Develop your beans using your favorite IDE
- ▼ Bundle beans in one or more *EJB-JAR files*
 - An EJB-JAR file is a .ZIP file containing beans
 - Use the *jar* command to create EJB-JAR file
- ▼ Start up your application server
- ▼ Application server then:
 - Loads EJB-JAR files from disk
 - Unzips the EJB-JAR files
 - Retrieves beans from EJB-JAR files
 - Makes beans available to be called by clients
- ▼ Start your client application, which calls the beans

What is an Enterprise Bean?

- ▼ **EJB-JAR file:** A deployable .ZIP file containing each of these pieces:
 - **Home Interface:** Interface clients use to create a bean
 - **Remote Interface:** Interface clients use to call a bean's business methods
 - **Enterprise Bean class:** Class where your bean's implementation logic goes
 - **Deployment Descriptor:** XML file that describes your bean's "middleware needs" to the application server

EJB.JAR

Home
Interface

Remote
Interface

Enterprise
Bean
Class

Deployment
Descriptor

Simple Bean

Our first example:

A simple stateless session bean which implements the classic "Hello, World!"

Home Interface

- ▼ Clients use the home interface to create beans

```
// HelloHome.java
public interface HelloHome extends javax.ejb.EJBHome
{
    Hello create() throws java.rmi.RemoteException,
                    javax.ejb.CreateException;
}
```

Remote Interface

- ▼ Clients use the remote interface to call beans

```
// Hello.java
public interface Hello extends javax.ejb.EJBObject
{
    public String hello()
        throws java.rmi.RemoteException;
}
```

Enterprise Bean Class

- ▼ Enterprise bean class is the guts of the bean – the biz logic

```
// HelloBean.java
public class HelloBean implements javax.ejb.SessionBean
{
    // EJB-required methods
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void
    setSessionContext(javax.ejb.SessionContext ctx) {}

    // Business methods
    public String hello() { return "Hello, World!"; }
}
```

XML Deployment Descriptor

ejb-jar.xml

▼ Deployment descriptor gives app server info about your bean

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD  
Enterprise JavaBeans 2.0//EN"  
"http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
```

```
<ejb-jar>  
  <enterprise-beans>  
    <session>  
      <ejb-name>Hello</ejb-name>  
      <home>examples.HelloHome</home>  
      <remote>examples.Hello</remote>  
      <ejb-class>examples.HelloBean</ejb-class>  
      <session-type>Stateless</session-type>  
      <transaction-type>Container</transaction-type>  
    </session>  
  </enterprise-beans>  
</ejb-jar>
```

Application Server-Specific Descriptor

- ▼ All information is not specified in the standard deployment descriptor. For example:
 - CMP settings
 - Security Realm integration
- ▼ To compensate, some application servers require additional proprietary files bundled with beans
- ▼ These supply application server-specific info

Example Client

▼ Client that looks up and invokes a bean

```
// HelloClient.java
public class HelloClient {
    public static void main(String[] args) throws Exception {
        java.util.Properties props = System.getProperties();

        javax.naming.Context ctx =
            new javax.naming.InitialContext(props);

        HelloHome home = (HelloHome)
            javax.rmi.PortableRemoteObject.narrow(
                ctx.lookup("HelloHome"), HelloHome.class);

        Hello hello = home.create();
        System.out.println(hello.hello());
        hello.remove();
    }
}
```

Where are we?

- EJB Development Overview
- *The Enterprise Bean Class*
 - Learn how to write an Enterprise Bean Class
- The Remote Interface
- The Home Interface
- Deployment Descriptors and EJB-JAR files

What Is the Enterprise Bean Class?



- ▼ The implementation of your bean
- ▼ Where you write your logic
- ▼ Physically, a `.class` file
- ▼ Could be either session bean, entity bean, or message driven bean

Choosing Your Bean Type

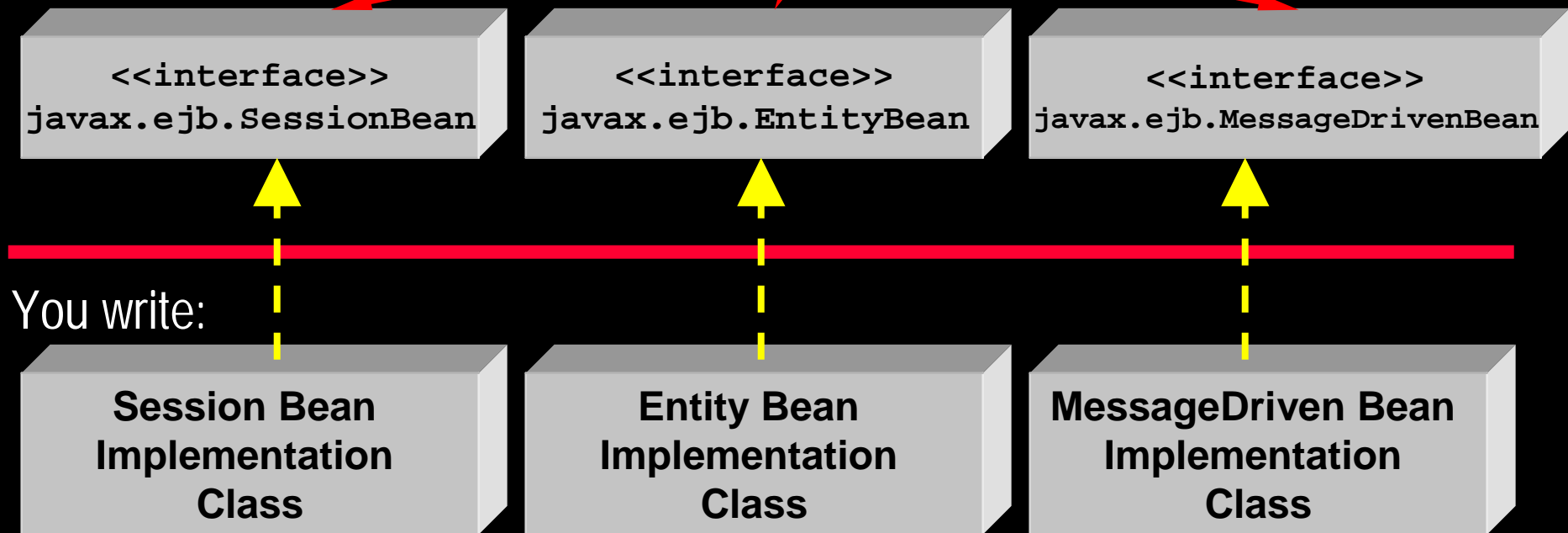
- ▼ Our example bean included this code:

```
public class HelloBean implements javax.ejb.SessionBean
```

- ▼ You choose the bean type by implementing the proper interface:

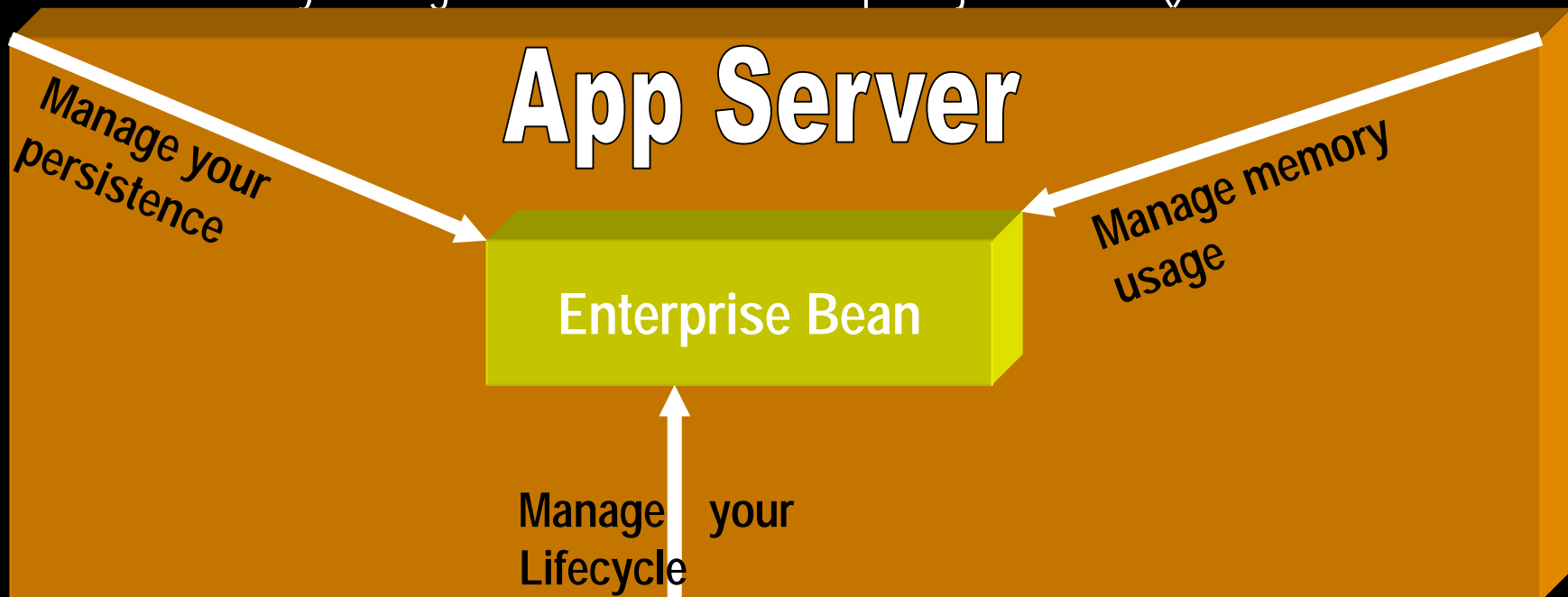
- SessionBean, EntityBean, or MessageDrivenBean

Comes with EJB Distribution



Parent Interfaces

- ▼ What is in the `SessionBean`, `EntityBean`, and `MessageDrivenBean` interfaces?
 - These interfaces define methods you **must** implement
 - They are methods the app server calls to 'manage' your bean:
 - Bean lifecycle methods – example: `ejbRemove()`
 - Persistence methods – example: `ejbStore()`
 - Memory management methods – example: `ejbPassivate()`



Let's Look Inside These Parent Interfaces

- ▼ Here's the complete set of methods you must implement:

<<Interface>

SessionBean

```
ejbActivate()  
ejbPassivate()  
ejbRemove()  
setSession  
Context()
```

<<Interface>

EntityBean

```
ejbActivate()  
ejbPassivate()  
ejbRemove()  
setEntity  
Context()  
unsetEntity  
Context()  
ejbLoad()  
ejbStore()
```

<<Interface>

MessageDrivenBean

```
ejbRemove()  
onMessage()  
setMessageDriven  
Context()
```

- ▼ Mini-Lab:

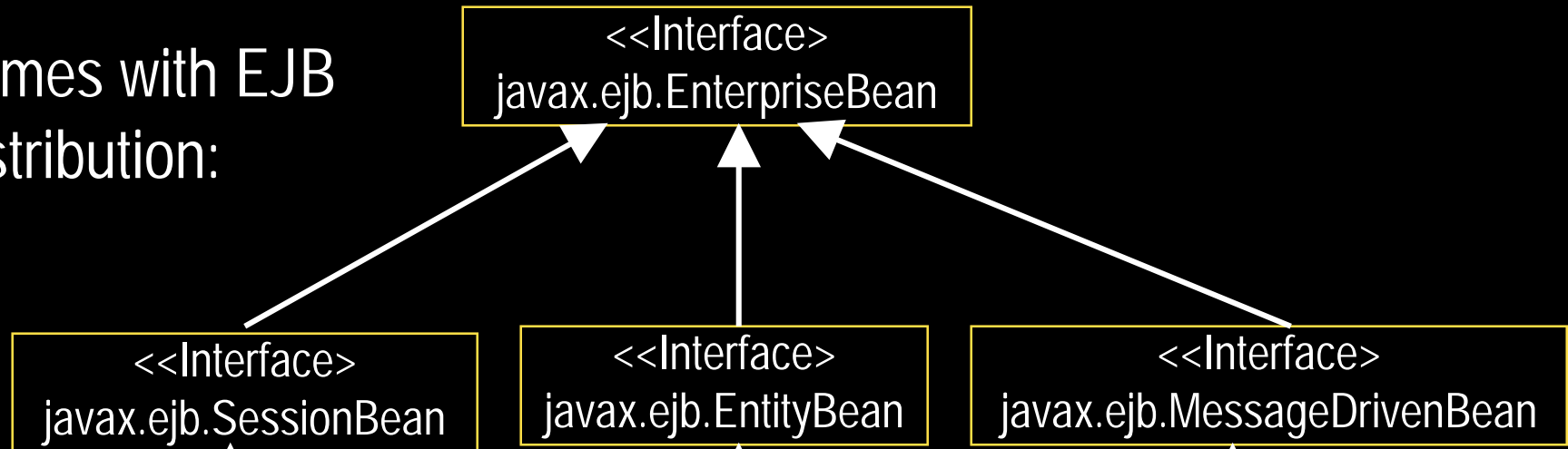
- Try typing `javap javax.ejb.SessionBean`
- Notice this line: **public interface SessionBean** extends EnterpriseBean
- Try typing `javap javax.ejb.EnterpriseBean`

- ▼ Discussion point: *Why did Sun create an empty parent interface called EnterpriseBean?*

The EnterpriseBean Interface

- ▼ EnterpriseBean indicates that your class is indeed a bean
- ▼ Allows the app server to treat all bean types the same

Comes with EJB
Distribution:



You Write:

Session Bean
Implementation
Class

Entity Bean
Implementation
Class

MessageDriven
Bean
Implementation
Class

The EnterpriseBean Interface

...continued

▼ Let's take another look at EnterpriseBean:

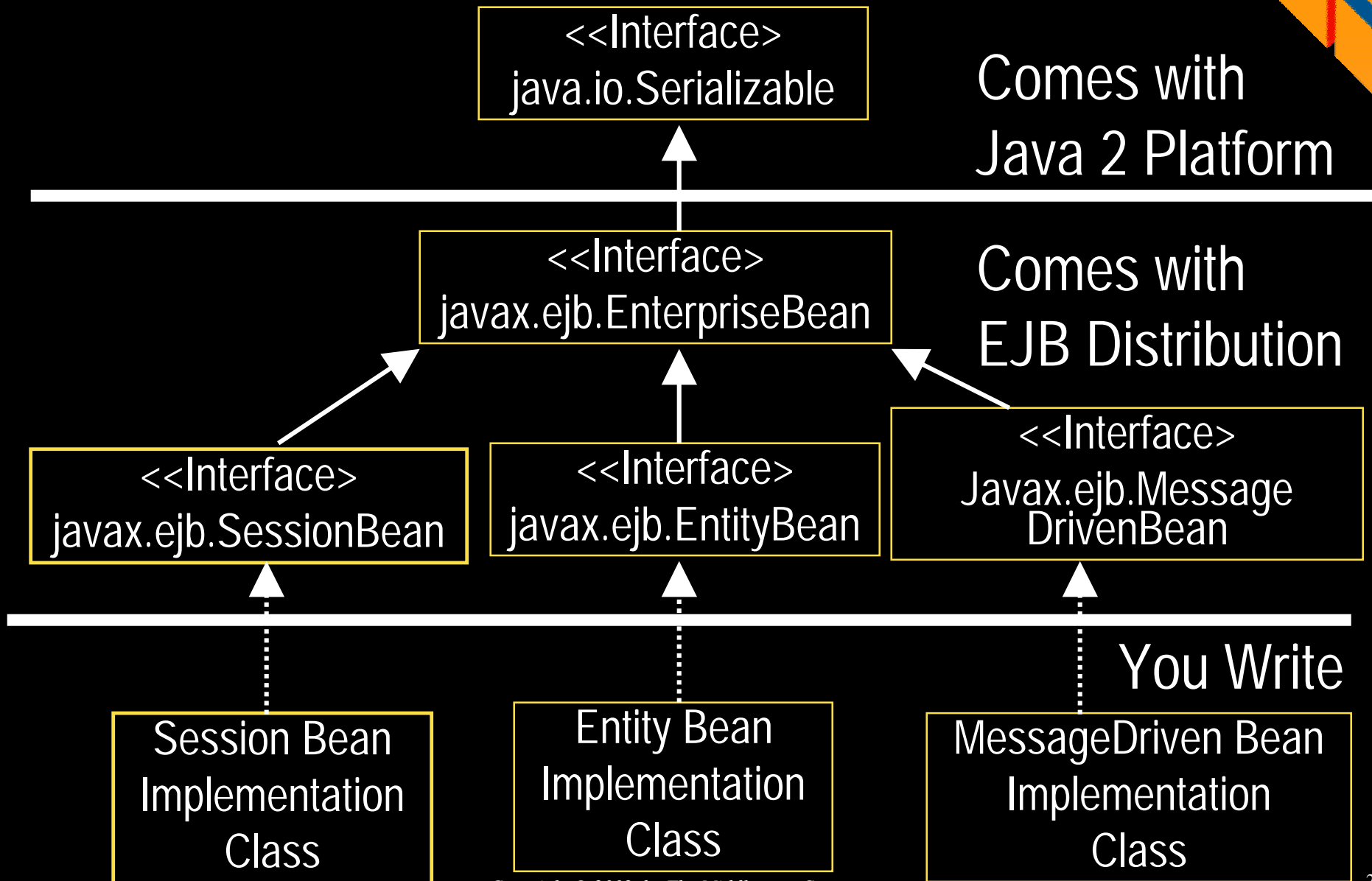
```
public interface javax.ejb.EnterpriseBean
    extends java.io.Serializable
{ }
```

▼ Notice that EnterpriseBean extends java.io.Serializable

▼ *Discussion points:*

- *What is java.io.Serializable used for?*
- *Why would the EJB spec authors want beans to be serializable?*

Enterprise Bean Class Summary



Presentation Roadmap

Where are we?

- EJB Development Overview
- The Enterprise Bean Class
- *The Remote Interface*
- The Home Interface
- Deployment Descriptors and EJB-JAR files

The Remote Interface

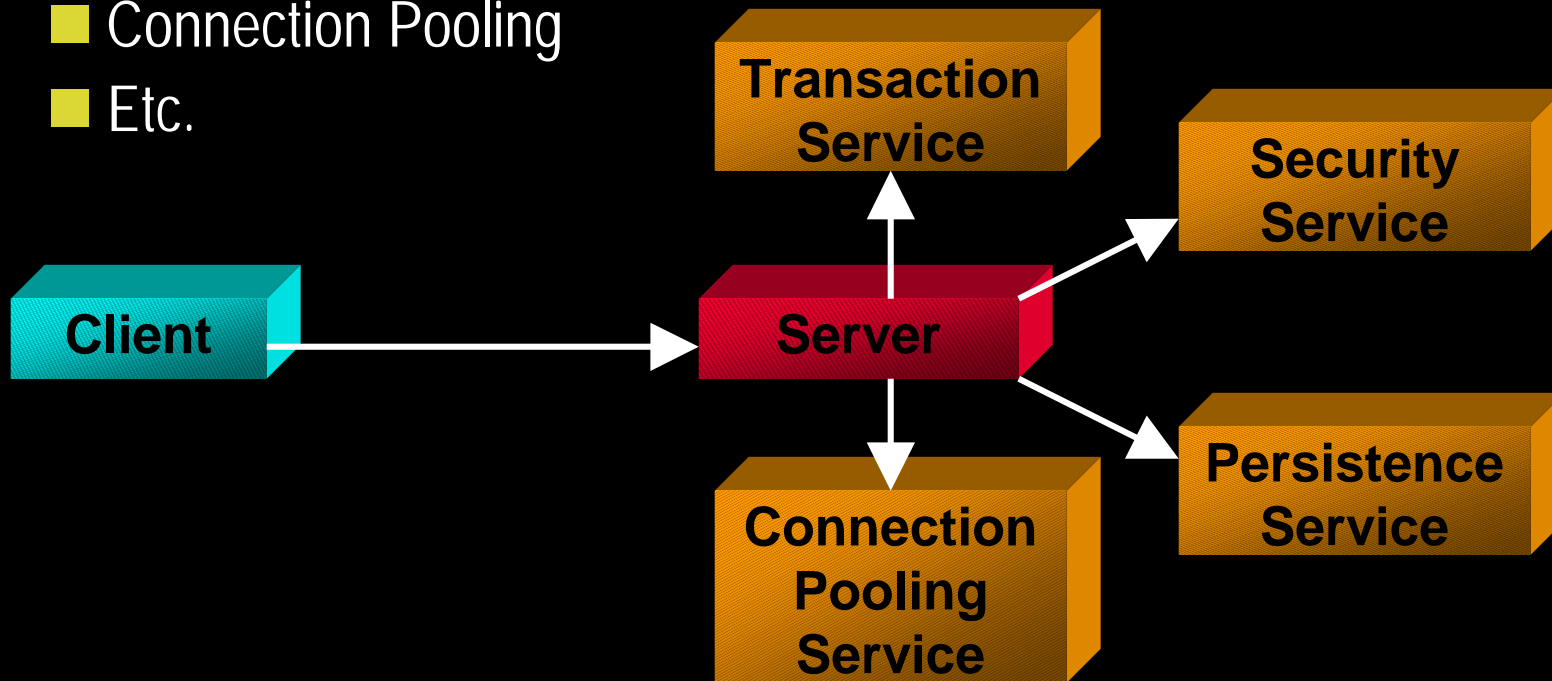
- ▼ Recall: Clients use the remote interface to call beans
- ▼ To refresh your memory, here is that same remote interface:

```
public interface Hello extends javax.ejb.EJBObject
{
    public String hello() throws java.rmi.RemoteException;
}
```

Motivation for Remote Interface

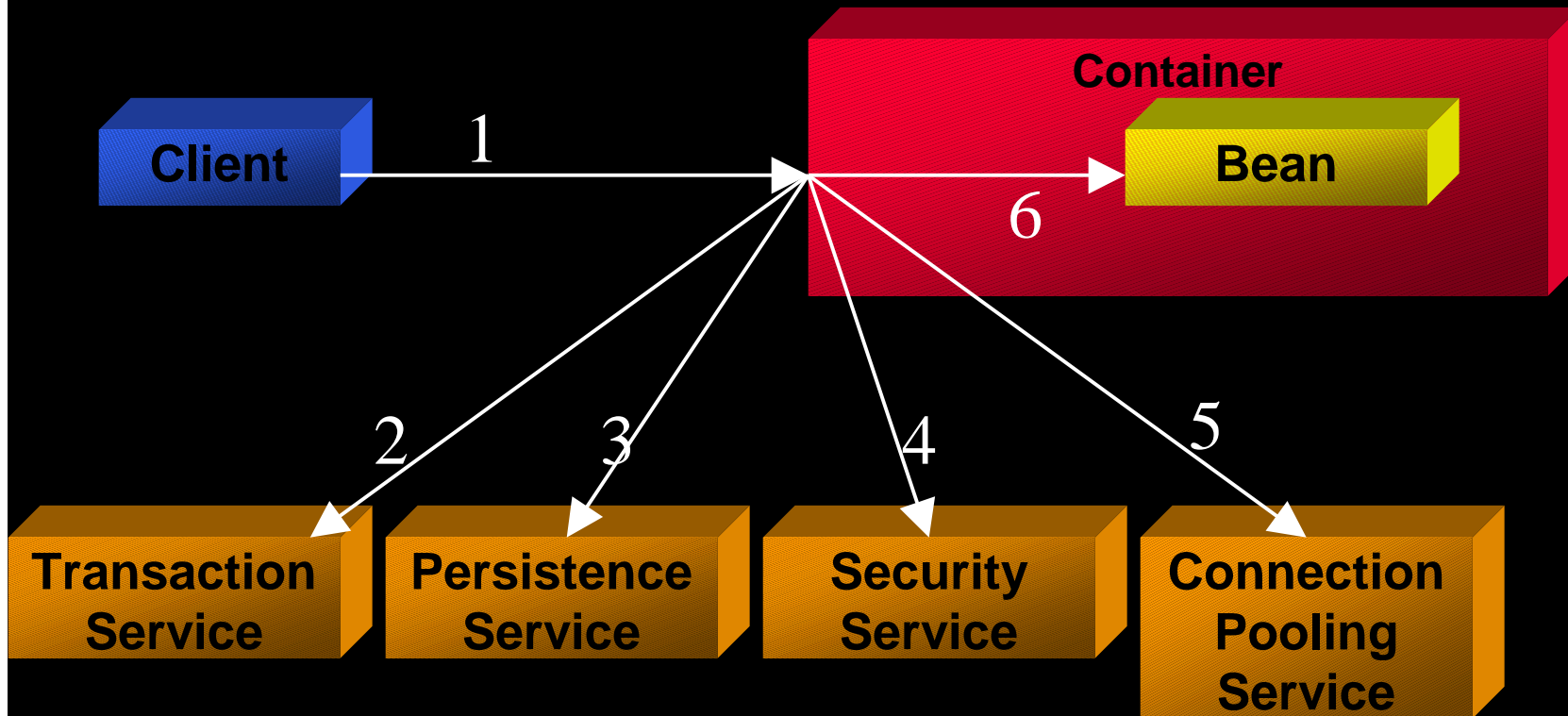
▼ In traditional client/server, developers would write to middleware APIs

- Persistence
- Security
- Transactions
- Connection Pooling
- Etc.



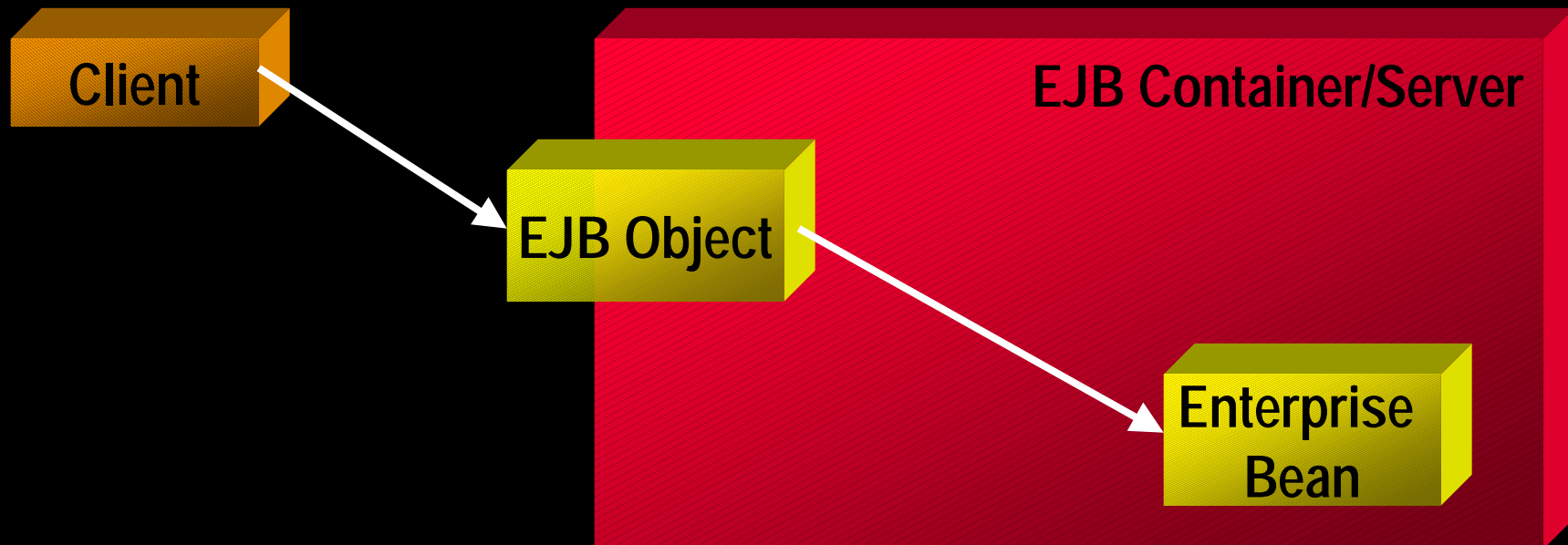
Idea of Request Interception

- ▼ EJB app servers relieve you of the burden of writing to APIs
- ▼ Rather, the container *intercepts* all requests and *adds-on middleware*
- ▼ Takeaway point: **Every call to a bean is intercepted by the container**



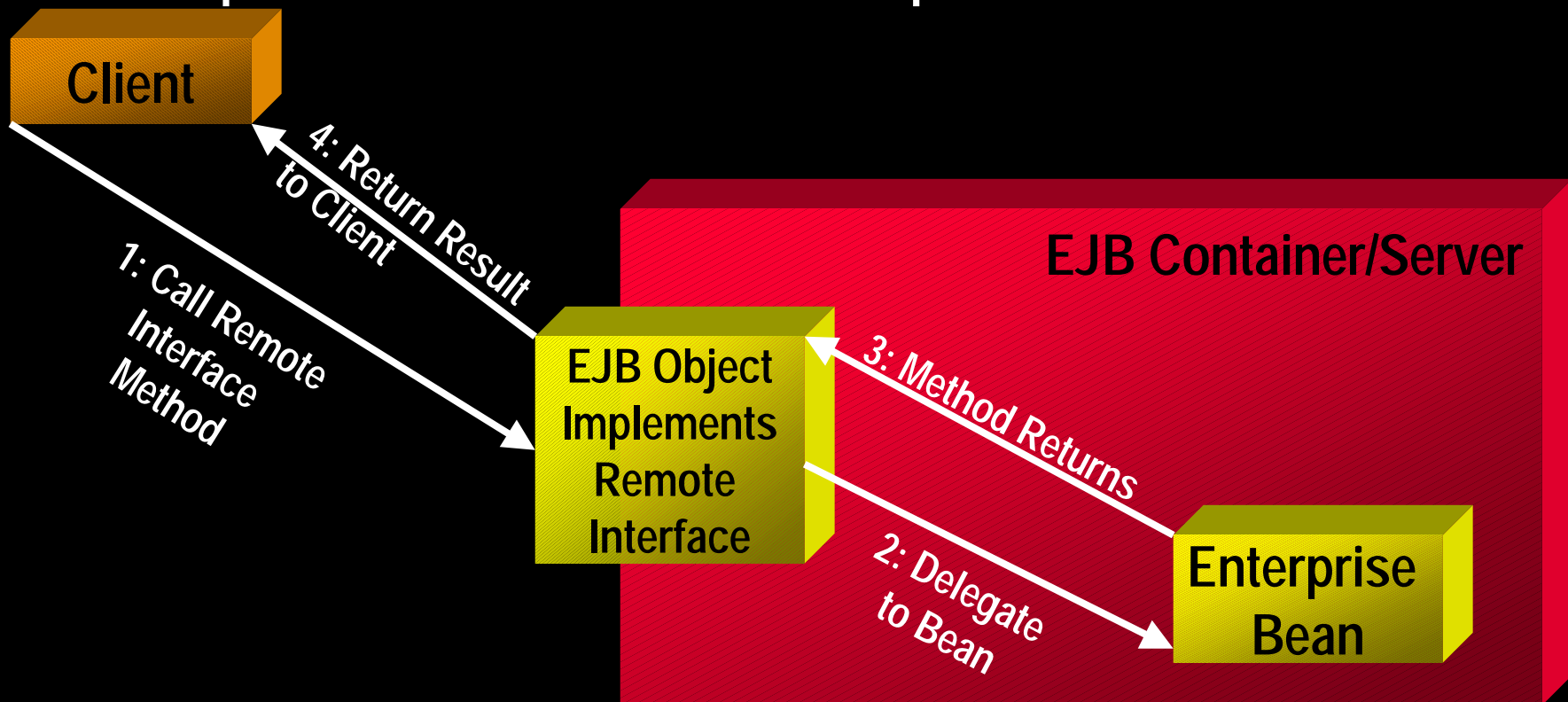
EJB Objects Defined

- ▼ The request interceptor is called the EJB Object
- ▼ An EJB Object:
 - Receives the client's request
 - Adds middleware services to the request
 - Delegates the call to the bean
 - This is the huge value that EJB provides – implicit middleware!



EJB Objects and the Remote Interface

- ▼ Clients call EJB objects through the *Remote Interface*
 - Remote interface clones every method that a bean exposes
 - The EJB object, provided by app server, implements remote interface
 - Important: Your bean does *not* implement the remote interface!!!



EJB Objects are Generated Files

- ▼ We need a different EJB object for each bean
 - Every bean has different business methods
 - Therefore every EJB object has different business methods
- ▼ Solution: EJB server provides a tool to *generate* EJB objects
- ▼ How code generation works:
 1. Write home and remote interfaces and xml deployment descriptors
 2. Bundle all files in an EJB-JAR file
 3. Run EJB server's tool on EJB-JAR file
 4. Tool inspects remote interface
 5. Tool generates EJB object and supporting .class files based on interfaces and xml



Introducing javax.ejb.EJBObject

- ▼ The javax.ejb.EJBObject interface defines methods common to all remote interfaces
 - Example: A method that clients call to obtain a persistable Handle
- ▼ EJB server tool generates EJB objects that implement these methods

<<Interface>
javax.ejb.EJBObject

Comes with EJB
distribution

<<Interface>
Remote Interface

Supplied by Bean
provider (we write this)

EJB Object

Generated for us
by container &
vendor tools

Introducing javax.ejb.EJBObject ...continued



▼ To make things concrete, lets javap javax.ejb.EJBObject:

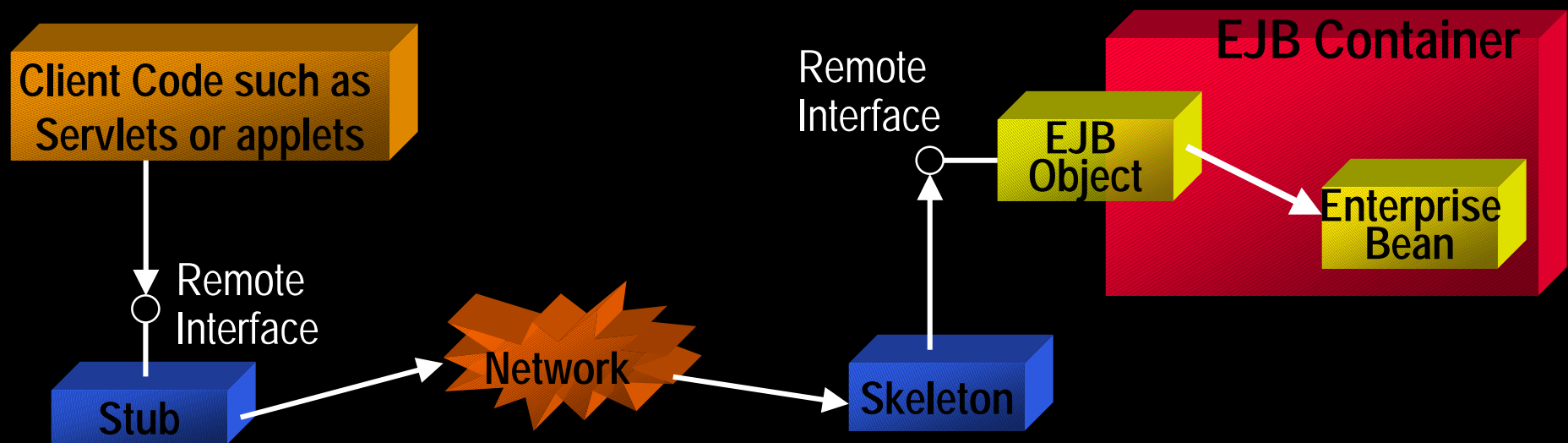
```
public interface javax.ejb.EJBObject extends
    java.rmi.Remote
{
    javax.ejb.EJBHome getEJBHome();
    javax.ejb.Handle getHandle();
    java.lang.Object getPrimaryKey();
    boolean isIdentical(javax.ejb.EJBObject);
    void remove();
}
```

Distributing Objects in EJB

- ▼ Recall: EJB is a *distributed* architecture
 - We need a mechanism to call beans remotely
- ▼ EJB achieves this w/Java RMI
 - (technically: RMI-IIOP for CORBA compatibility)

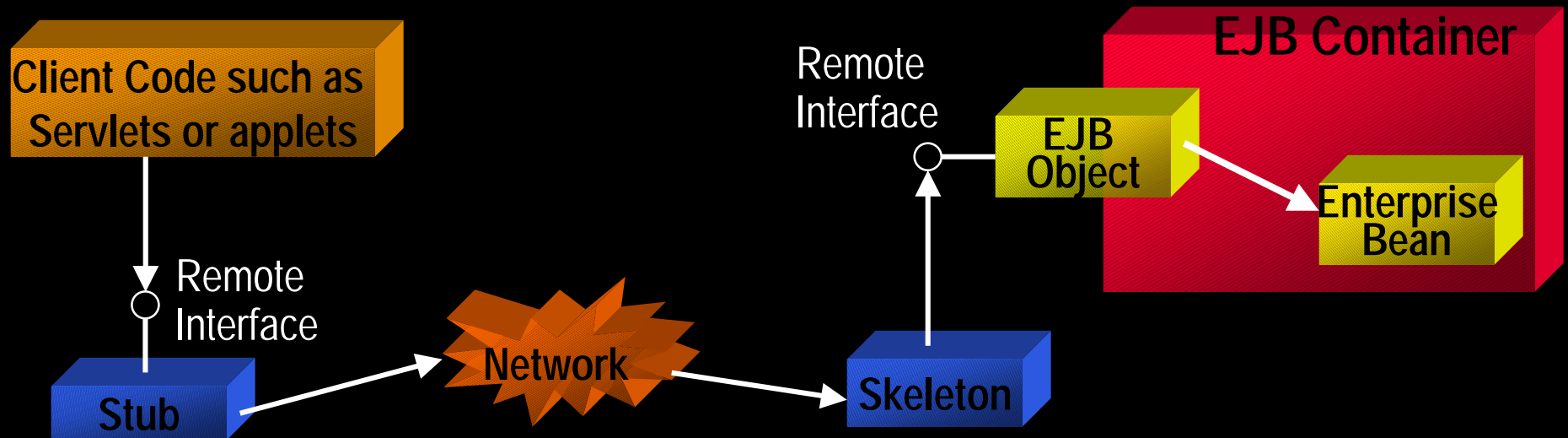
EJB Objects and RMI

- ▼ The good news – you don't need to worry about distributing beans
 - Beans are *not* RMI objects!
 - They are local! They cannot be called remotely
 - You don't need to obey the rules of RMI when writing your beans
- ▼ EJB Objects are really the RMI objects!
 - They can be called remotely
 - They have a stub and skeleton
 - EJB objects delegate to the beans



Remote Interfaces and RMI

- ▼ Corollary: EJB remote interfaces are actually RMI remote interfaces!
 - They extend `java.rmi.Remote`
 - The methods throw `RemoteException` types
 - Stub implements the remote interface



Remote Interfaces and RMI

...continued

- ▼ To make things concrete, here is `javax.ejb.EJBObject` again:

```
import java.rmi.RemoteException;

public interface javax.ejb.EJBObject extends
    java.rmi.Remote
{
    javax.ejb.EJBHome getEJBHome() throws RemoteException;
    javax.ejb.Handle getHandle() throws RemoteException;
    java.lang.Object getPrimaryKey() throws
        RemoteException;
    boolean isIdentical(javax.ejb.EJBObject) throws
        RemoteException;
    void remove() throws RemoteException,
        javax.ejb.RemoveException;
}
```

Remote Interface Summary

<<Interface>
Java.rmi.Remote

Comes with the Java 2 Platform

<<Interface>
javax.ejb.EJBObject

Comes with EJB distribution

<<Interface>
Remote Interface

Supplied by Bean provider (we write this)

EJB Object

Generated for us by container & vendor tools

Module Roadmap

Where are we?

- EJB Development Overview
- The Enterprise Bean Class
- The Remote Interface
- *The Home Interface*
- Deployment Descriptors and EJB-JAR files

The Home Interface

- ▼ Recall: Clients use the home interface to create beans
- ▼ To refresh your memory, here is that same home interface again:

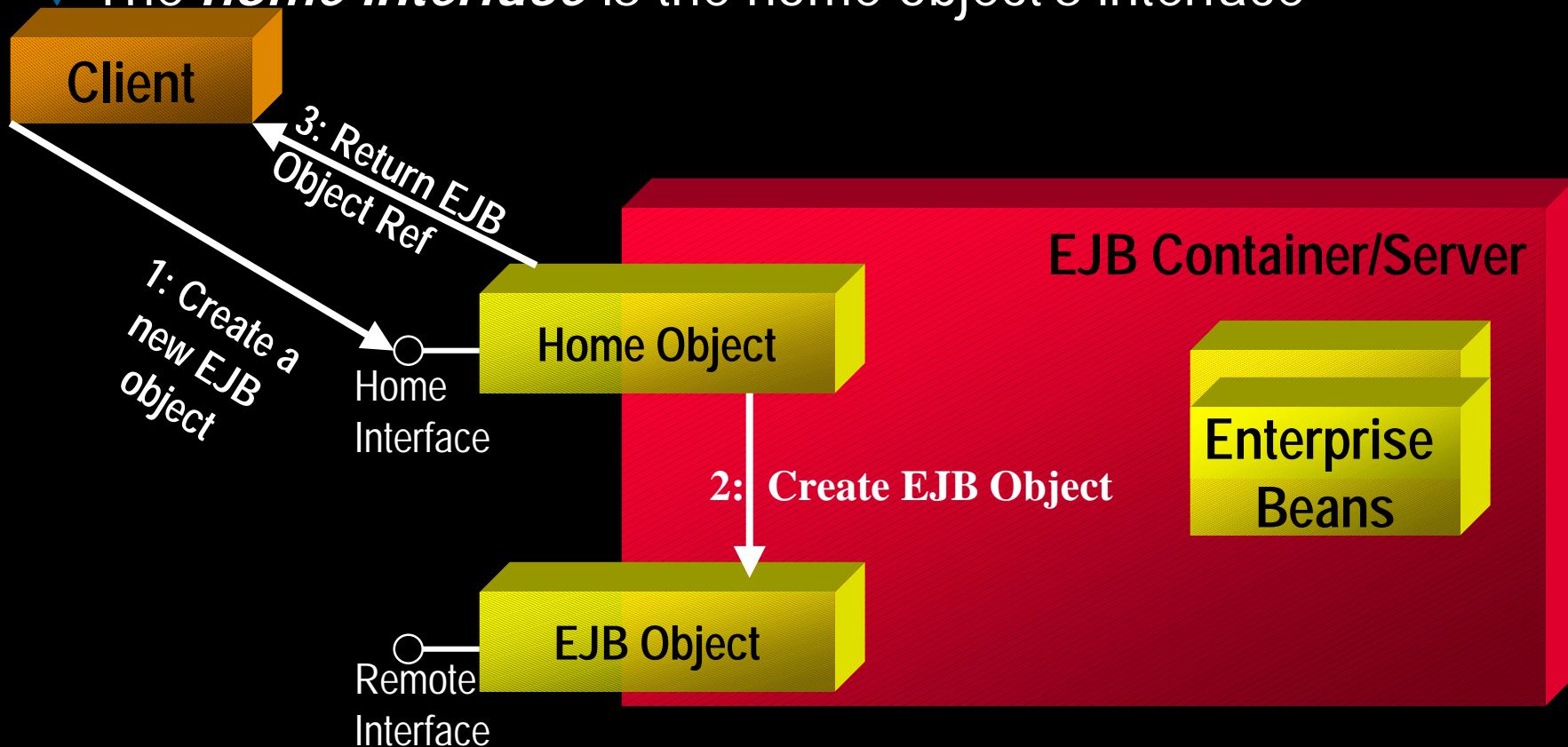
```
public interface HelloHome
    extends javax.ejb.EJBHome
{
    Hello create() throws java.rmi.RemoteException,
                    javax.ejb.CreateException;
}
```

Motivation for Home Interfaces

- ▼ To understand home interfaces, we first must understand how beans are created
- ▼ Discussion points:
 - *How does one instantiate objects in Java?*
 - *Can that same instantiation technique be used to create an object remotely?*
 - *What are techniques to instantiate objects remotely?*

Introducing Home Objects

- ▼ In EJB, *home objects* are your EJB object factories
 - They always exist on the server
 - They are responsible for instantiating EJB objects
- ▼ The *home interface* is the home object's interface



Home Objects are Also Generated

- ▼ Every bean can be created differently
 - Different beans take different initialization parameters when created
 - Every home object has different creation methods
 - We need a different home object for each bean
- ▼ **Solution:** Your container vendor provides a tool to *generate* home object classes
 - You supply home interface, and the vendor generates home object



Motivation for javax.ejb.EJBHome

- ▼ Just like with EJB objects...
 - There are some methods that all home objects should have
 - Those methods are defined in javax.ejb.EJBHome

<<Interface>
javax.ejb.EJBHome

Comes with EJB
distribution

<<Interface>
Home Interface

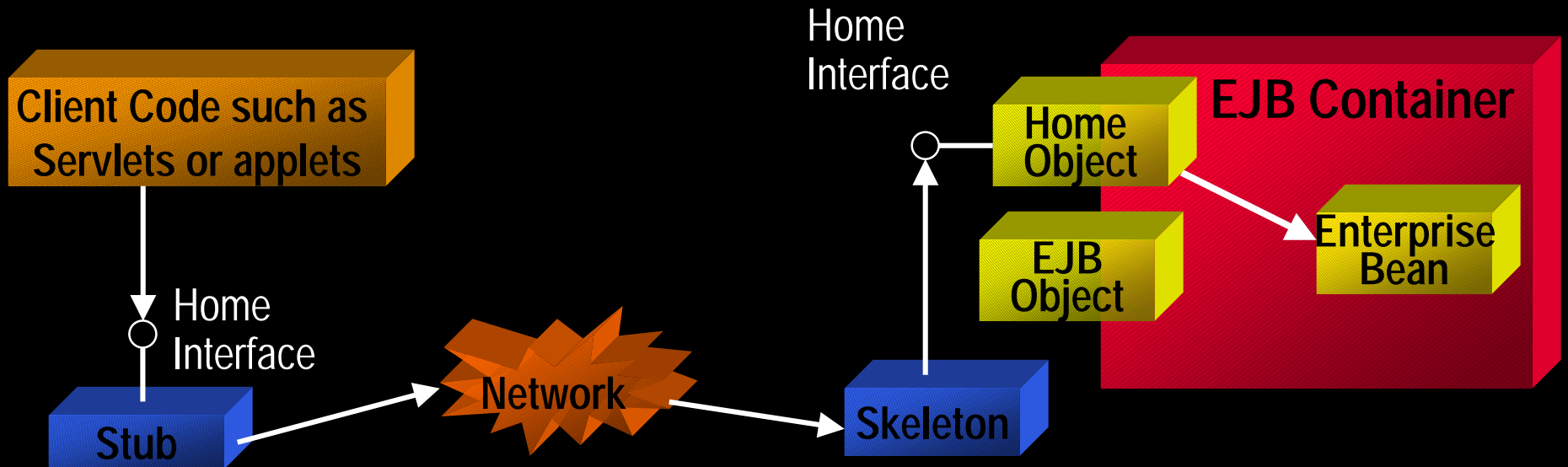
Supplied by Bean
provider (we write
this)

Home Object

Generated for us
by container &
vendor tools

Home Objects and RMI

- ▼ Just like with EJB objects..
 - Home Objects are also RMI objects
 - Home Interfaces are also RMI remote interfaces



▼ To make things concrete, here is javax.ejb.EJBHome:

```
public interface javax.ejb.EJBHome
    extends java.rmi.Remote {
    javax.ejb.EJBMetaData getEJBMetaData() throws
        RemoteException;
    javax.ejb.HomeHandle getHomeHandle() throws
        RemoteException;
    void remove(Object) throws RemoteException;
    void remove(javax.ejb.Handle) throws RemoteException;
}
```

▼ *Question:*

- Why is there no create() defined here?

Home Interface Summary

<<Interface>
Java.rmi.Remote

Comes with the
Java 2 Platform

<<Interface>
javax.ejb.EJBHome

Comes with EJB
distribution

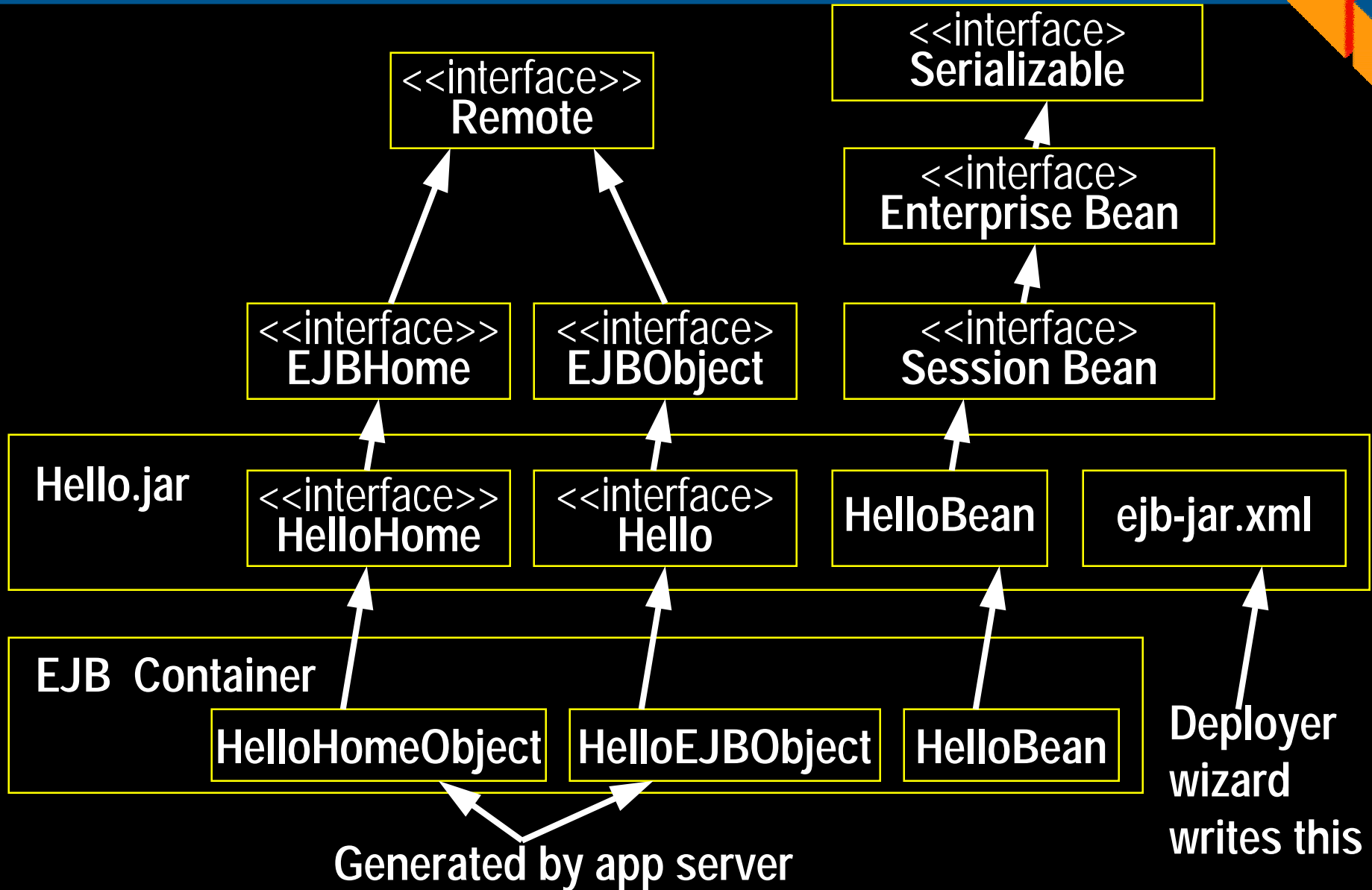
<<Interface>
Home Interface

Supplied by Bean
provider (we write
this)

Home Object

Generated for us
by container &
vendor tools

The Big Picture



Presentation Roadmap

Where are we?

- EJB Development Overview
- The Enterprise Bean Class
- The Remote Interface
- The Home Interface
- *Deployment Descriptors and EJB-JAR files*

What is a Deployment Descriptor?



- ▼ A deployment descriptor (DD) is a contract between you and the application server
 - A DD dictates the middleware generated by app server
 - Enable you to ‘set middleware properties’ on your components
 - Idea of *declarative middleware* – I declare what I need, rather than write code
 - What goes into a DD? Plenty of things that you can tune:
 - Transactions
 - Security
 - Relationships between data objects

Why Deployment Descriptors?

- ▼ Deployment descriptors (DDs) are the key to EJB
 - It's how *implicit middleware* is achieved – middleware without APIs
 - Less code, faster development
- ▼ DDs are essential for bean providers (Independent Software Vendors)
 - ISVs don't like to give away source code
 - Intellectual property issues
 - Makes upgrades hard
 - With DDs, customers can tune middleware without writing java code

Limitations of Deployment Descriptors

- ▼ Unfortunately, not everything can be specified in a DD
- ▼ Example: *how does an entity bean map to a datastore?*
 - Sun devised entity beans with an expert group of datastore vendors
 - RDBMS vendors: Oracle, Sybase, etc.
 - OODBMS vendors: GemStone, Secant, etc.
 - Flat file vendors: IBM (IMS files running on MVS), etc.
 - Obviously it would be really nice if Sun could devise a standard for object-relational mappings however:
 - Every RDBMS is slightly different
 - There are other types of datastores which may be used
 - Conclusion: Almost impossible to specify this in a portable way

Solution to Non-portable Middleware



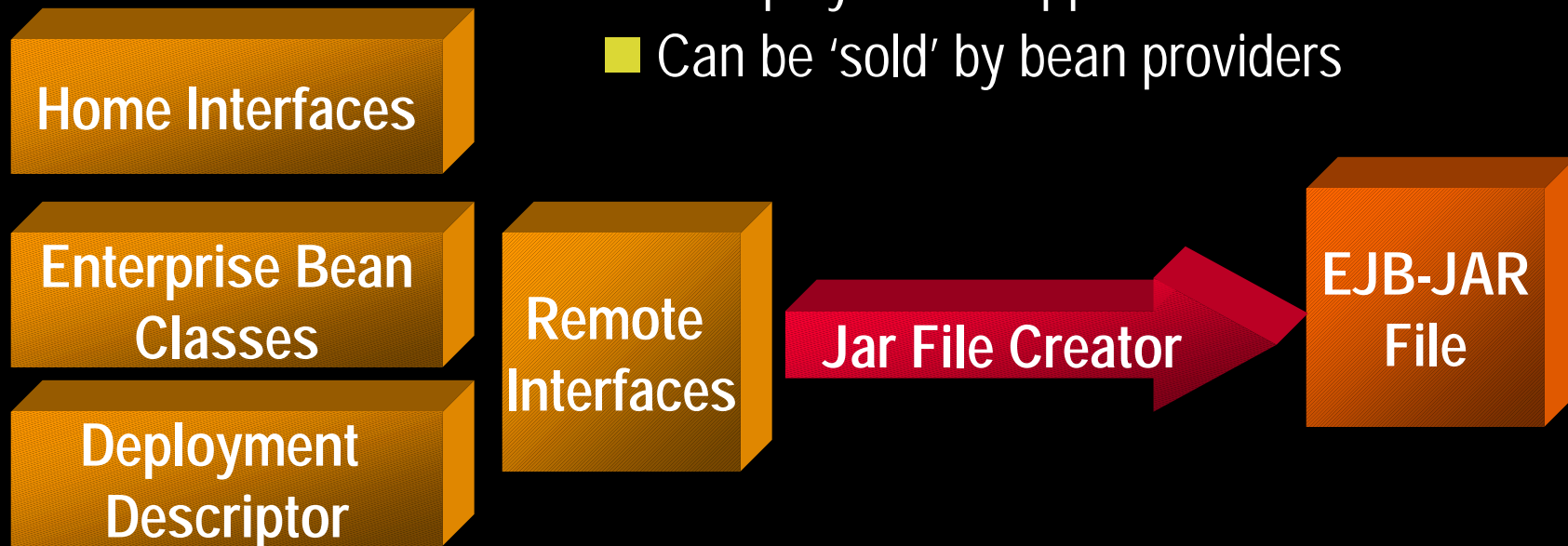
▼ Solution is to allow for 2 deployment descriptors:

- Portable DD file
- ejb-jar.xml
- Mandated by Sun
- Works in any app server
- Specifies transaction, security, settings, etc.

▼ Vendor-specific DD file

- Use to set vendor-specific settings, such as object-relational mapping
- NB: Most vendors use XML

- ▼ Recall that an EJB-JAR file:
- Uses .ZIP file format
 - May be compressed
 - Is created with the *jar* utility
 - Is deployed into app servers
 - Can be 'sold' by bean providers

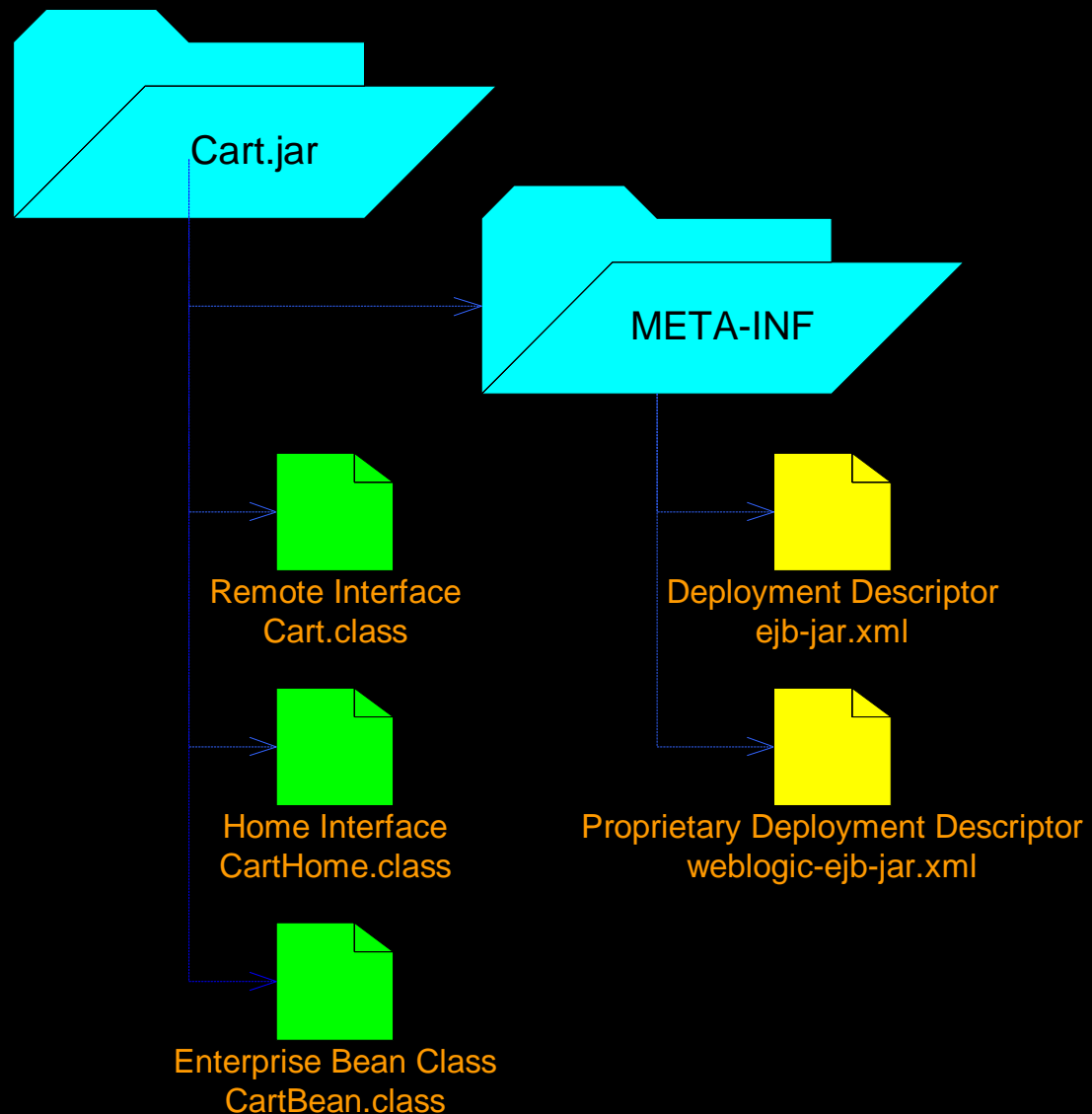


EJB-JAR Files

- ▼ The container must understand how to parse an EJB-JAR file's contents
- ▼ The DD informs the container of the filenames of
 - home interface
 - remote interface
 - bean class

EJB-JAR file layout

- ▼ Typical layout of an EJB-JAR file:



▼ How do I decide what to put in an EJB-JAR file?

- Answer: Put related beans together
- Rule of thumb: Bundle beans that you would never want to separate from one another
- Examples:
 - Purchase orders, employees, purchase order processors
 - Product, catalog engine
 - DNA strand, DNA sequencer, DNA result

EJB Build Process Summary

- ▼ Build process is different for various app servers
- ▼ Typically follows these steps:
 1. Write code
 2. Create deployment descriptors
 3. Compile code
 4. JAR files up using the prescribed directory structure
 5. Invoke tool to generate EJB Object, Home Object, stubs, & skeletons

What is deployment?

- ▼ **Deployment** is making an EJB available on an App Server
- ▼ There are several steps to deployment:
 - Create the code for an EJB
 - Create XML deployment descriptors (DD's)
 - Create a Java Archive (JAR) file that includes the code and DD's
 - **Varies:** compile the JAR file with the **vendor's** compiler
 - **Varies:** Configure the EJB further using the **vendor's** tools
 - **Varies:** Target a server instance to host the deploy-ready EJB jar

An Example `ejb-jar.xml` File

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD  
Enterprise JavaBeans 2.0//EN"  
"http://java.sun.com/j2ee/dtds/ejb-jar_2_0.dtd">
```

```
<ejb-jar>  
  <enterprise-beans>  
    <session>  
      <ejb-name>Hello</ejb-name>  
      <home>examples.HelloHome</home>  
      <remote>examples.Hello</remote>  
      <ejb-class>examples.HelloBean</ejb-class>  
      <session-type>Stateless</session-type>  
      <transaction-type>Container</transaction-type>  
    </session>  
  </enterprise-beans>  
</ejb-jar>
```

Further information

- ▼ EJB Spec: <http://java.sun.com/products/ejb>
- ▼ Ed Roman's "Mastering EJB" book
 - Available for free download on TheServerSide.com
- ▼ Web sites
 - <http://www.TheServerSide.com>
 - <http://www.EJBInfo.com>
 - <http://java.sun.com/products/ejb>
 - EJB-INTEREST list: <http://archives.java.sun.com>