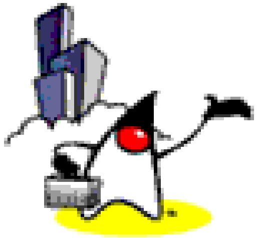
J2EE Programming with Passion!

Session Bean



Sang Shin
sang.shin@sun.com

Jeff Cutler
jpcutler@yahoo.com

www.javapassion.com/j2ee

Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgements
  - Some slides are made from the contents of the book "Applied Enterprise JavaBeans" Technology written by Kevin Boone (Sun Microsystems)
  - Some slides are made from the contents of J2EE tutorial authored by Dale Green (Sun Microsystems)
  - Some slides are made from the contents of EJB codecamp material authored by Carol McDonald (Sun Microsystems)

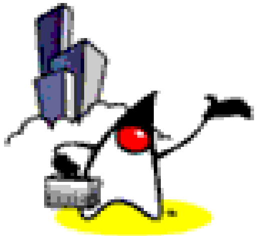
Revision History

- 02/18/2003: version 1, partially created (Sang Shin)
- 03/01/2003: version 2, updated (Jeff Cutler)
- 03/02/2003: version 3, updated (Sang Shin)
- 07/23/2003: version 4, speaker notes are added
- To be done:
  - speaker notes and slides still need some polishing
  - clean up method relationship slides

Agenda

- What is a Session bean?
- When to use Session beans?
- Types of Session beans
  - stateless
  - stateful
- Programming APIs
- Steps for creating stateless session bean
- Steps for creating stateful session bean

What is Session Bean?



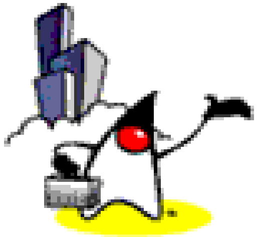
Types of Beans

- Session Beans
  - Stateful session beans
  - Stateless session beans
- Entity Beans
  - Bean Managed Persistence (BMP)
  - Container Managed Persistence (CMP)
- Message Driven Beans
  - JMS (Java Message Service)
  - JAXM (Java API for XML Messaging), SMTP

Session Beans

- Does work on behalf of a single client
  - life typically is that of its client
- Is not persistent and hence relatively short lived
  - Is gone when the EJB™ server crashes
- Does not represent data in data store, although can access/update such data
- Can be transaction aware
  - Can perform transaction demarcation

When to use Session Bean?

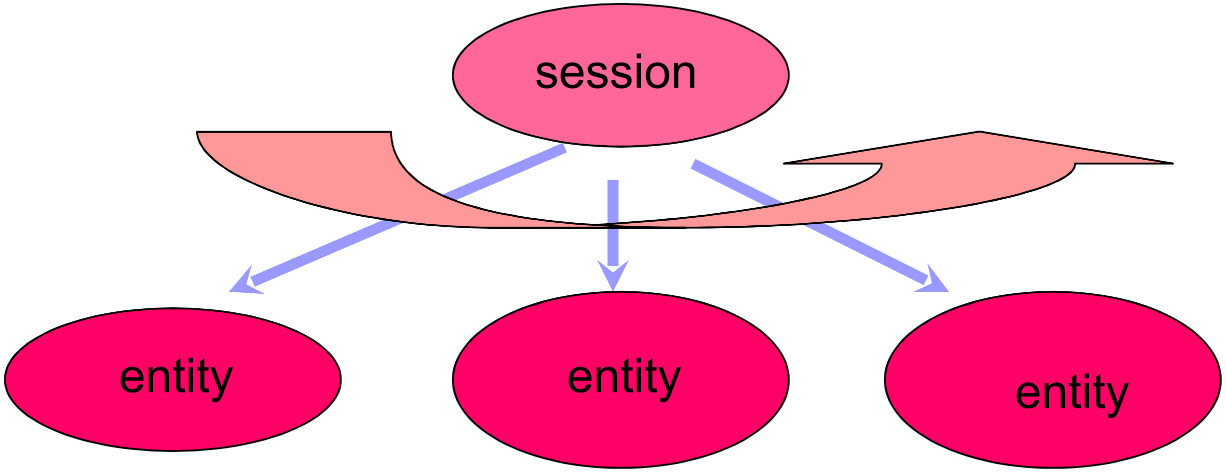


When to Use Session Beans?

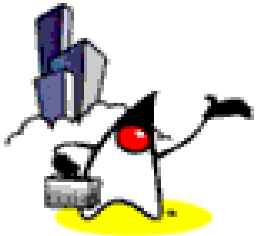
- To model workflow, processes or tasks, manage activities (make reservation, purchase...).
  - To move business application logic from client to the server Side
- Specific to a particular client
- To coordinate processes among entity beans

When to use Session Beans?

- Session beans are typically used for business process or control logic that spans multiple entity beans

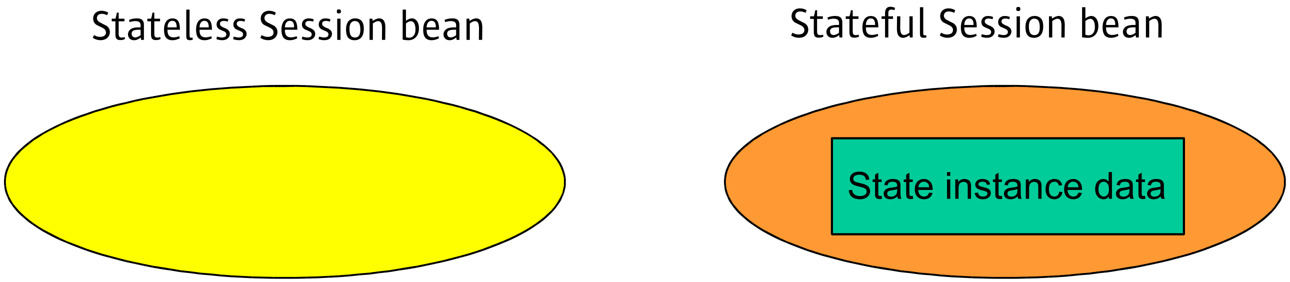


Types of Session Bean: Stateful & Stateless



2 Types of Session Beans

- Stateless: execute a request and return a result without saving any client specific state information
  - transient
  - temporary piece of business logic needed by a specific client for a limited time span
- Stateful: maintains client specific state

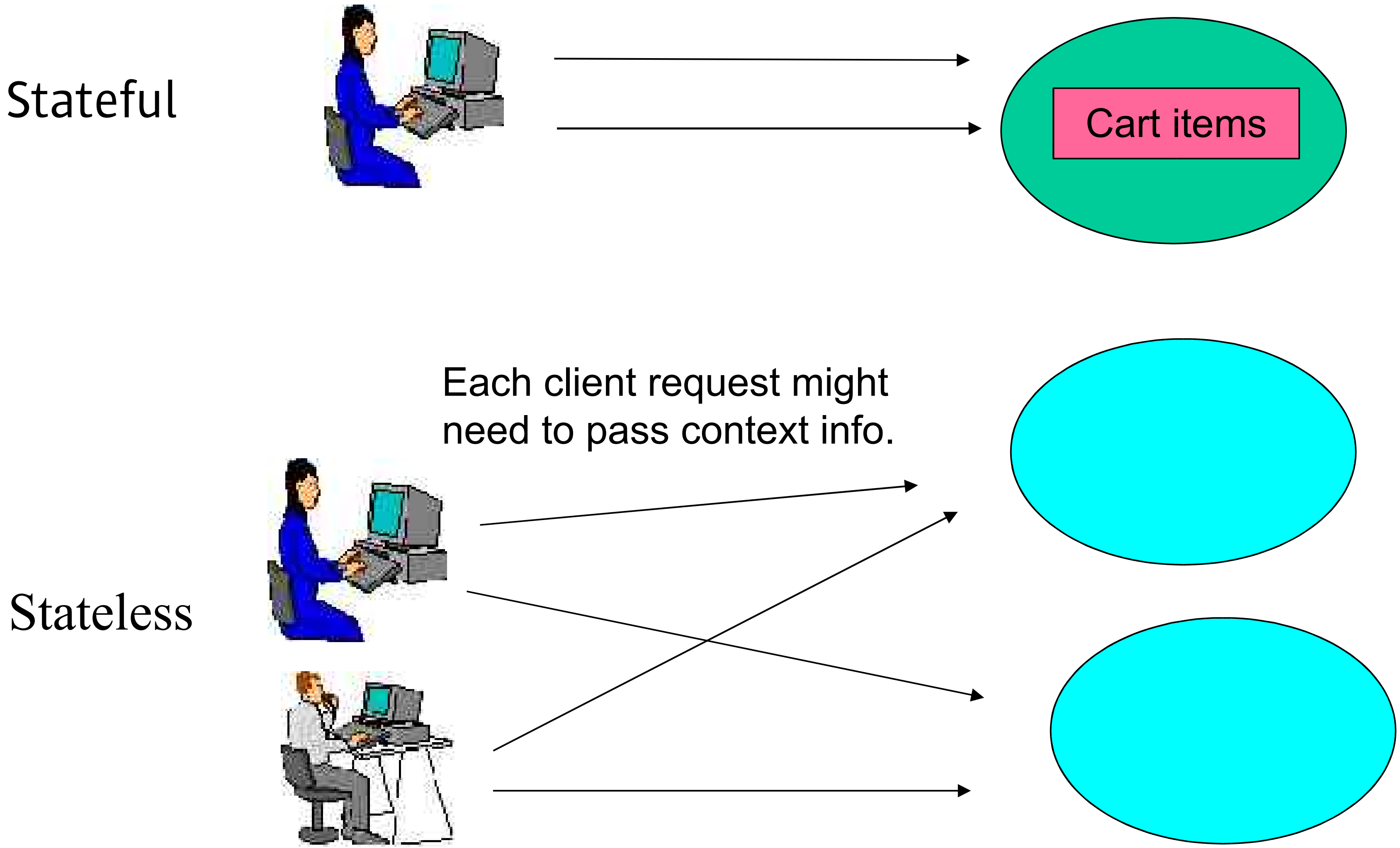


Stateless Session bean

Stateful Session bean

State instance data

Stateful versus Stateless



Stateful

Cart items

Stateless

Each client request might need to pass context info.

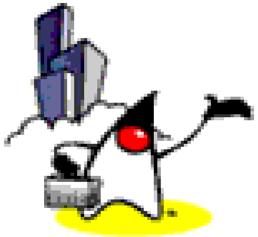
Examples of Stateless Session Bean

- Catalog
  - No client specific state needs to be preserved
  - Common catalog data for all clients
    - The data can be retrieved from database the first time it is accessed
- Interest calculator
  - No client specific state needs to be preserved
  - Common business logic for all clients

Examples of Stateful Session Bean

- Shopping cart
  - Client specific state needs to be preserved for each client
    - Items that a user wants to buy
  - State will be lost when the server crashes
- Travel ticket purchasing
  - Client specific state needs to be preserved for each client
    - Tickets to purchase and then confirm/cancel

Stateless Session Bean



What is Stateless Session Bean?

- Does not retain client specific state from one method invocation to the next
  - Bean instance can be reassigned to serve a method invocation from another client once current method invocation is done
  - Value of instance variables of a bean instance is not preserved between calls

Reusability of Stateless Session Bean Instances

- Container transparently reuses bean instances to serve different clients
  - Pool of bean instances are created by container at appropriate time (ex: at the time of system boot or when the size of pool becomes too small)
  - Bean instances are then recycled
  - Smaller number of bean instances (pool of bean ins tances) can serve larger number of clients at a single time – Improves scalability of the system
    - clients can be idle between calls

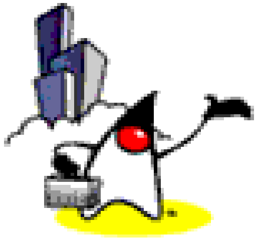
Resource usage of Stateless Session Beans

- Load-balancing & Failover (between EJB servers) is easier since no state needs to be preserved
  - Any bean instance in any EJB server can serve any client call
- High scalability since a client call can be served by any EJB server in a clustered architecture
  - In order to handle increased number of clients, just add more memory or more EJB servers

Usage Model of Stateless Session Bean

- Use it when no client specific state needs to be preserved between calls
- If stateless session bean has to deal with client specific request
  - Client then has to pass any needed information as parameters to the business methods
  - But may require the client to maintain state information on the client side which can mean more complex client code

Stateful Session Bean



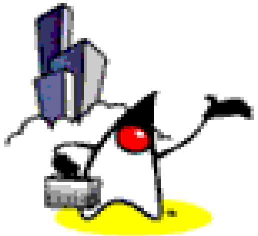
What is Stateful Session Bean?

- Does retain client specific state (session state) from one method invocation to the next
  - Value of instance variables of a bean instance is preserved between calls from same client
- Bean instances are created on demand for client, and removed afterwards
- Bean instances are to be maintained for each client
  - A client can be in active or inactive state
  - Not as scalable as stateless session bean

Failover of Stateful Session Bean

- State is not preserved when a server crashes
- High-end commercial servers can maintain session state even at the time of server failure by
  - maintaining server state in persistent storage
  - maintaining the same state in multiple servers

Quick Comparison between Stateless and Stateful Session Beans



Stateless and Stateful

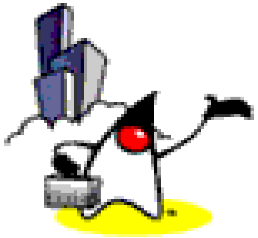
Stateless

- a pool of bean instances serve all clients
- client calls can be served by different bean instances
- a client cannot set instance variables on a per-client basis since as the client call can be handled by different bean instance possibly on a different server

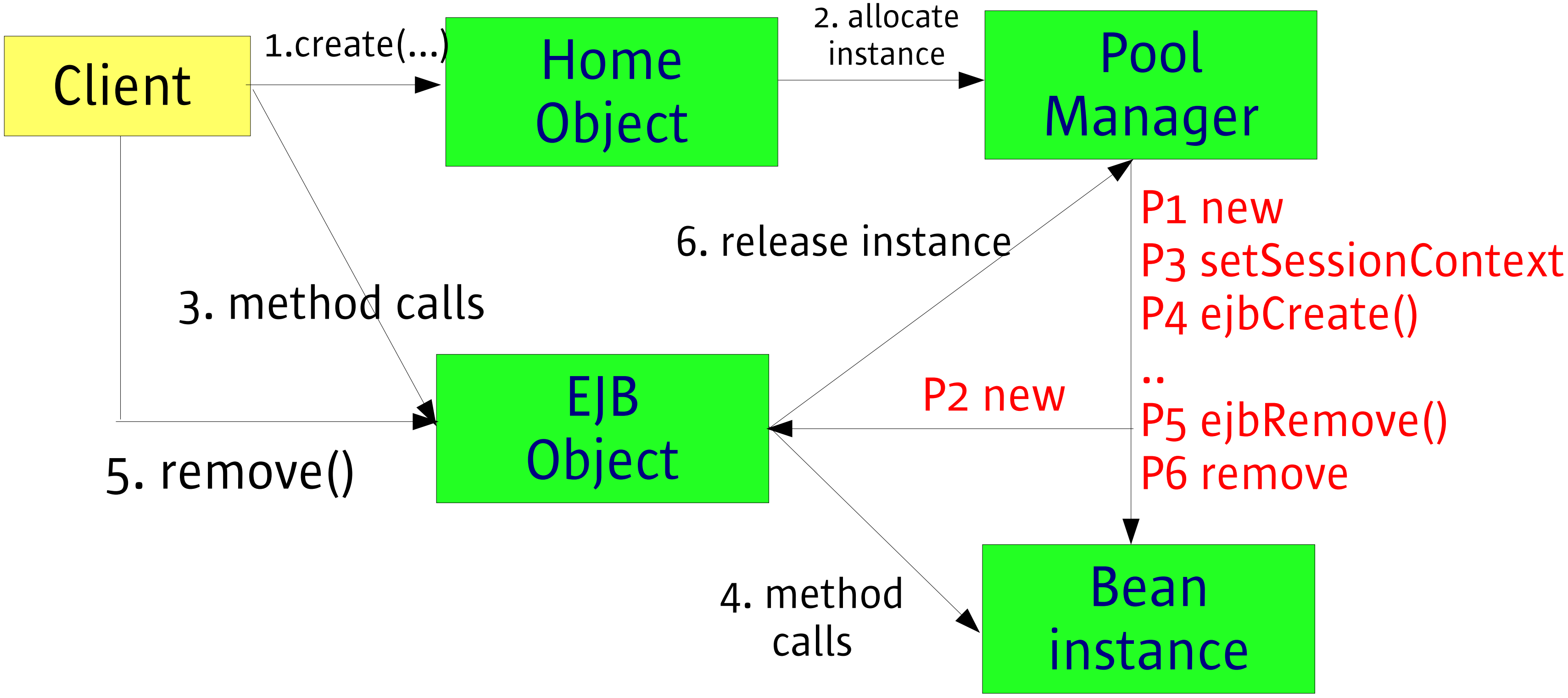
Stateful

- a bean instance is allocated to a particilar client
- method calls from a same client is always served by the same bean instance
- methods can set instance variables and they remain valid for the client

Sequence of Operations in the usage of Stateless Session Bean



Interaction between Client, Bean instance, Container for Stateless Session Bean



source: Applied Enterprise JavaBeans

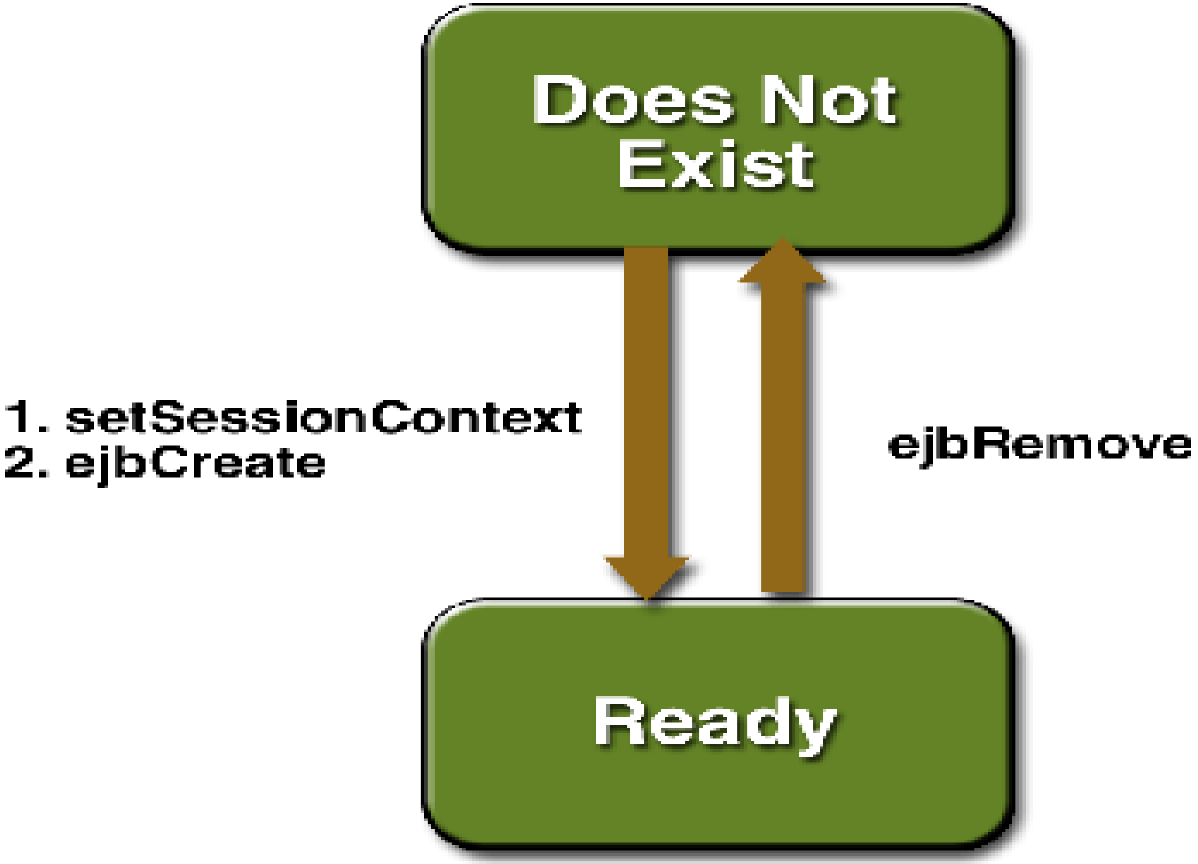
Sequence of Operations initiated by Client

1. Client calls a create() method on the home obje ct
2. Home object requests a bean instance from a pool, home object returns EJB object to client
3. Client calls business methods on EJB object
4. EJb object passes the call to Bean instance
5. Client calls remove() on the EJB object
6. EJB object releases bean instance

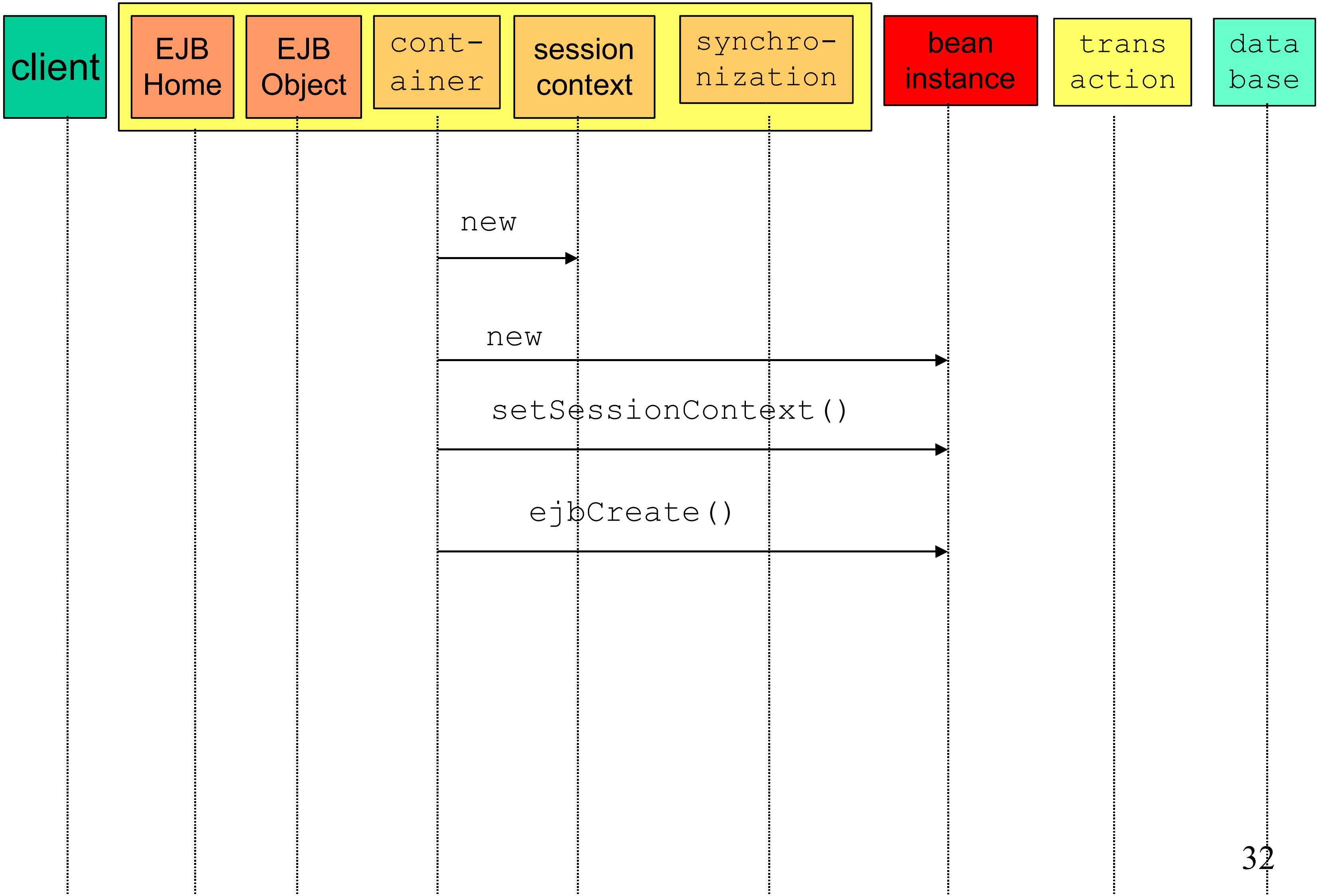
Sequence of Operations initiated by Container (independent from Client action)

1. Create Bean instance
2. Container calls setSessionContext() of the bean
3. Container calls ejbCreate() of the bean
4. Container calls ejbRemove() of a bean
5. remove

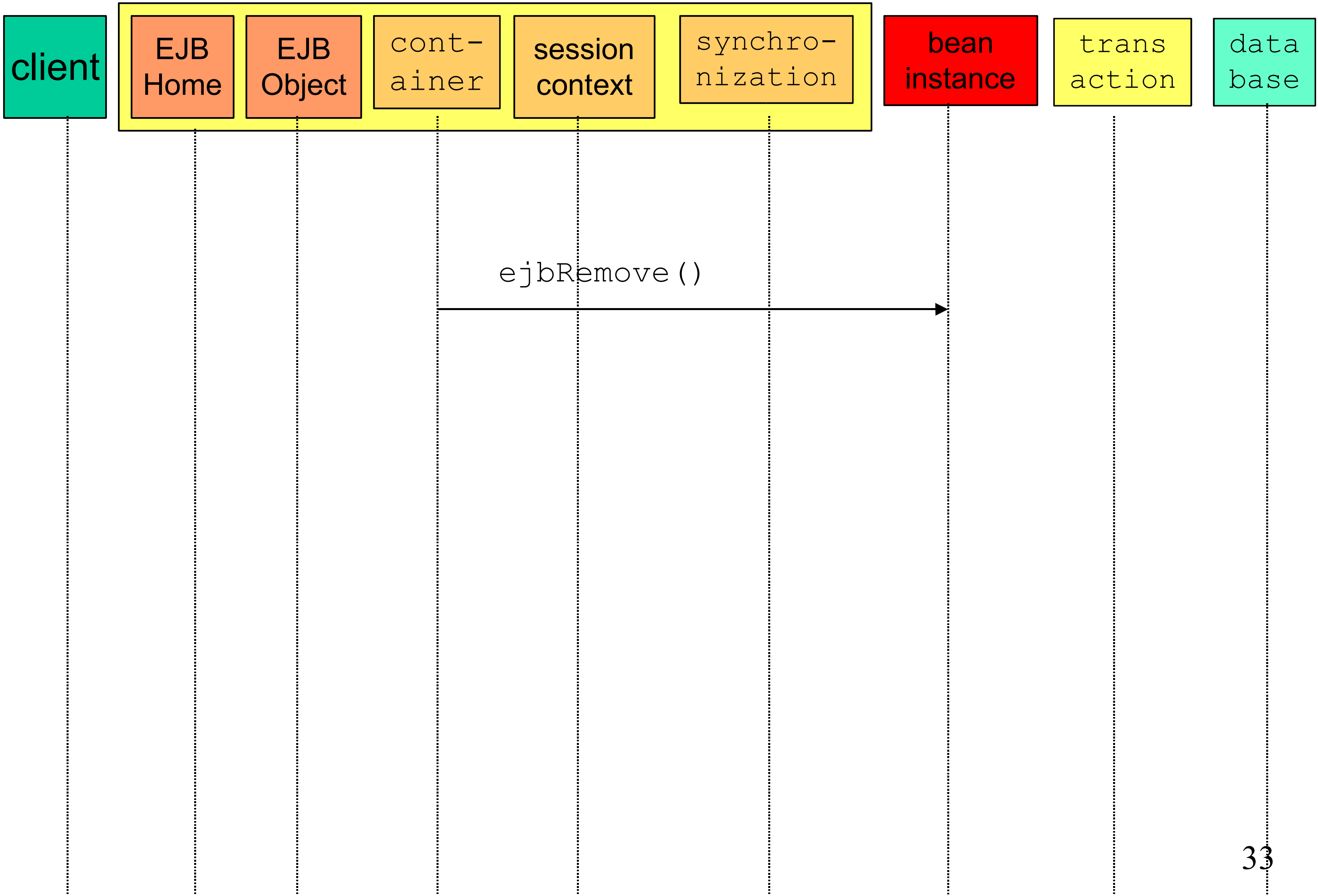
Life Cycle of a Stateless Session Bean Instance



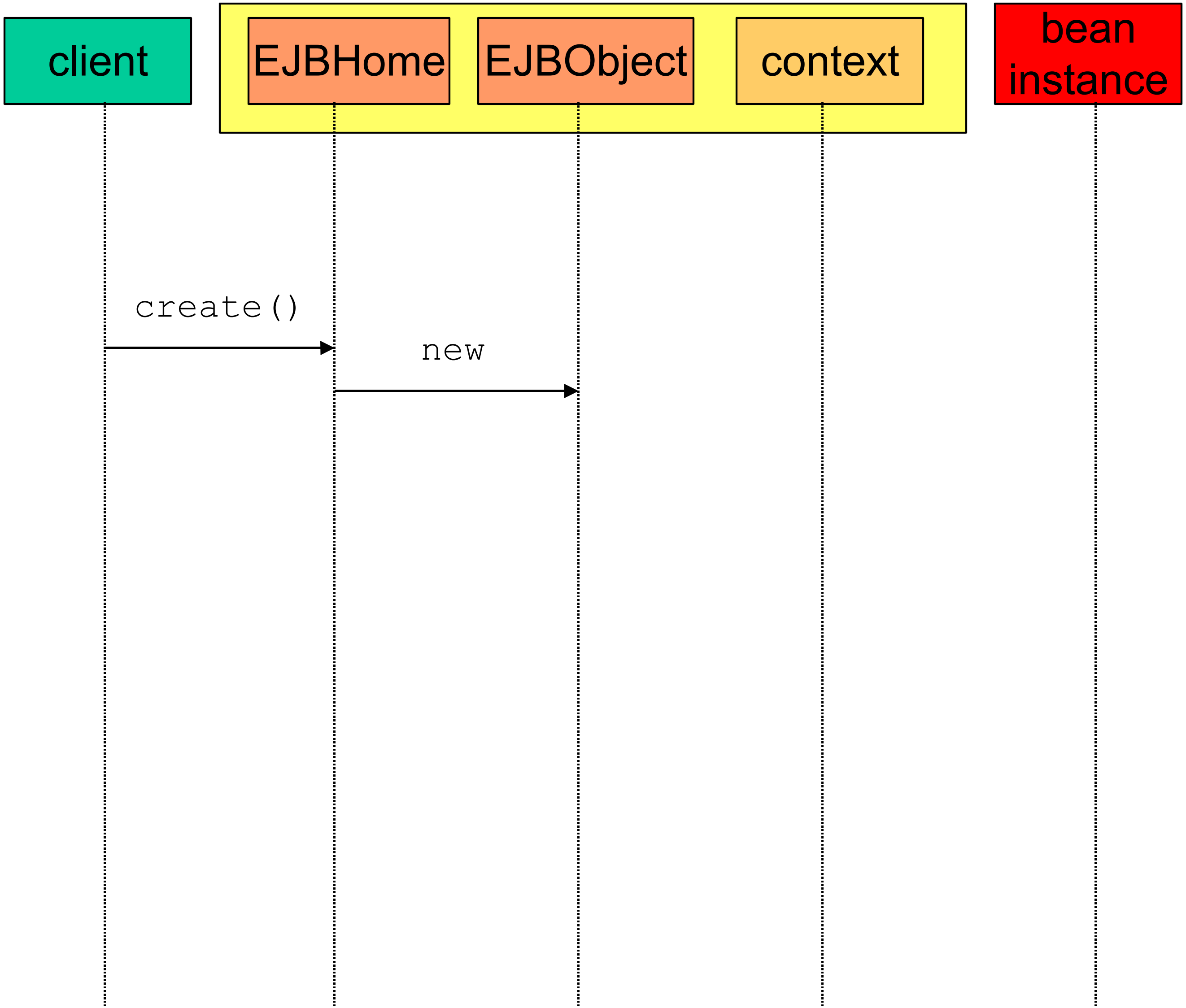
Adding Instance of Stateless Session Bean to a Method-ready Pool



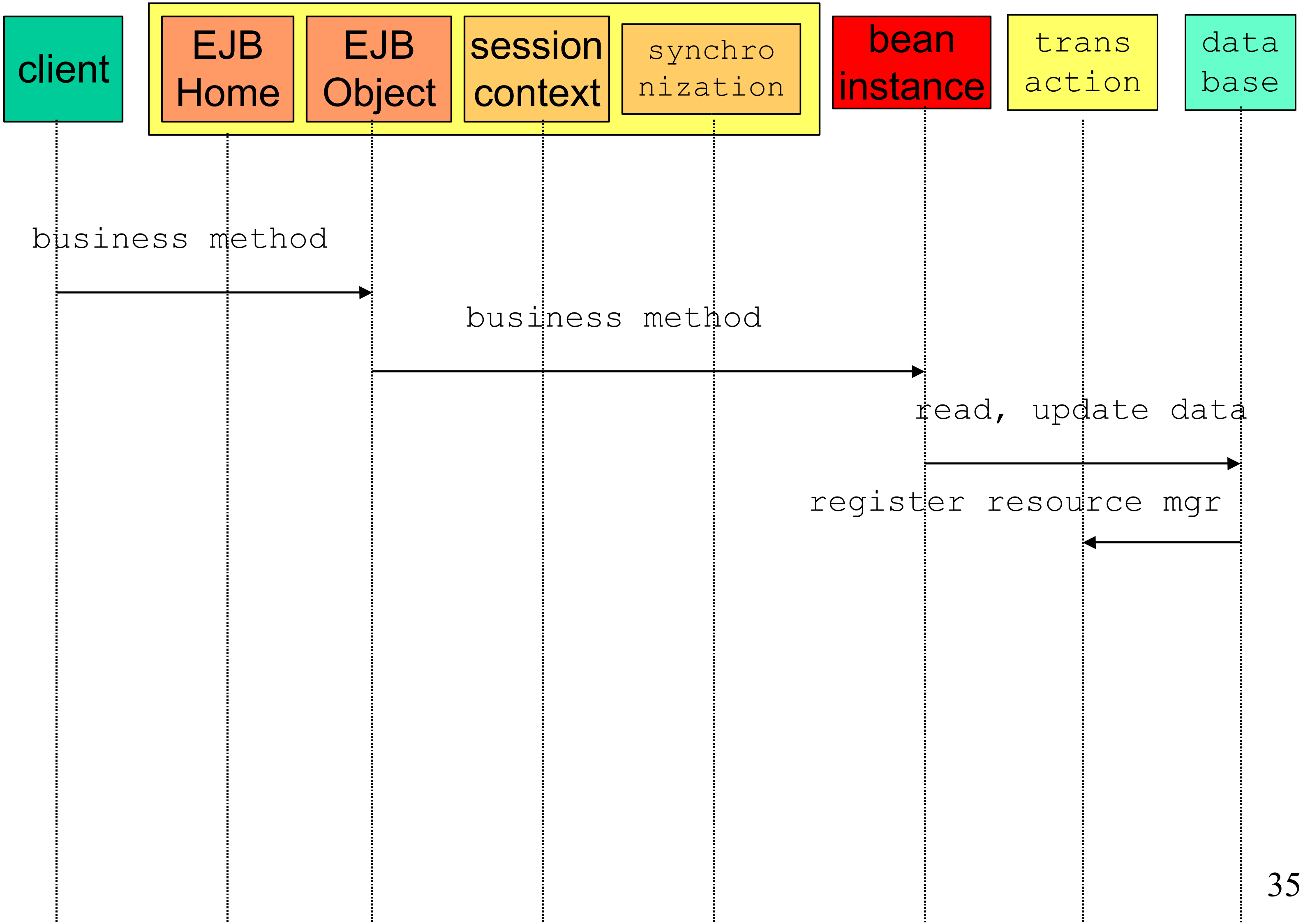
Removing Instance of Stateless Session Bean From Ready Pool



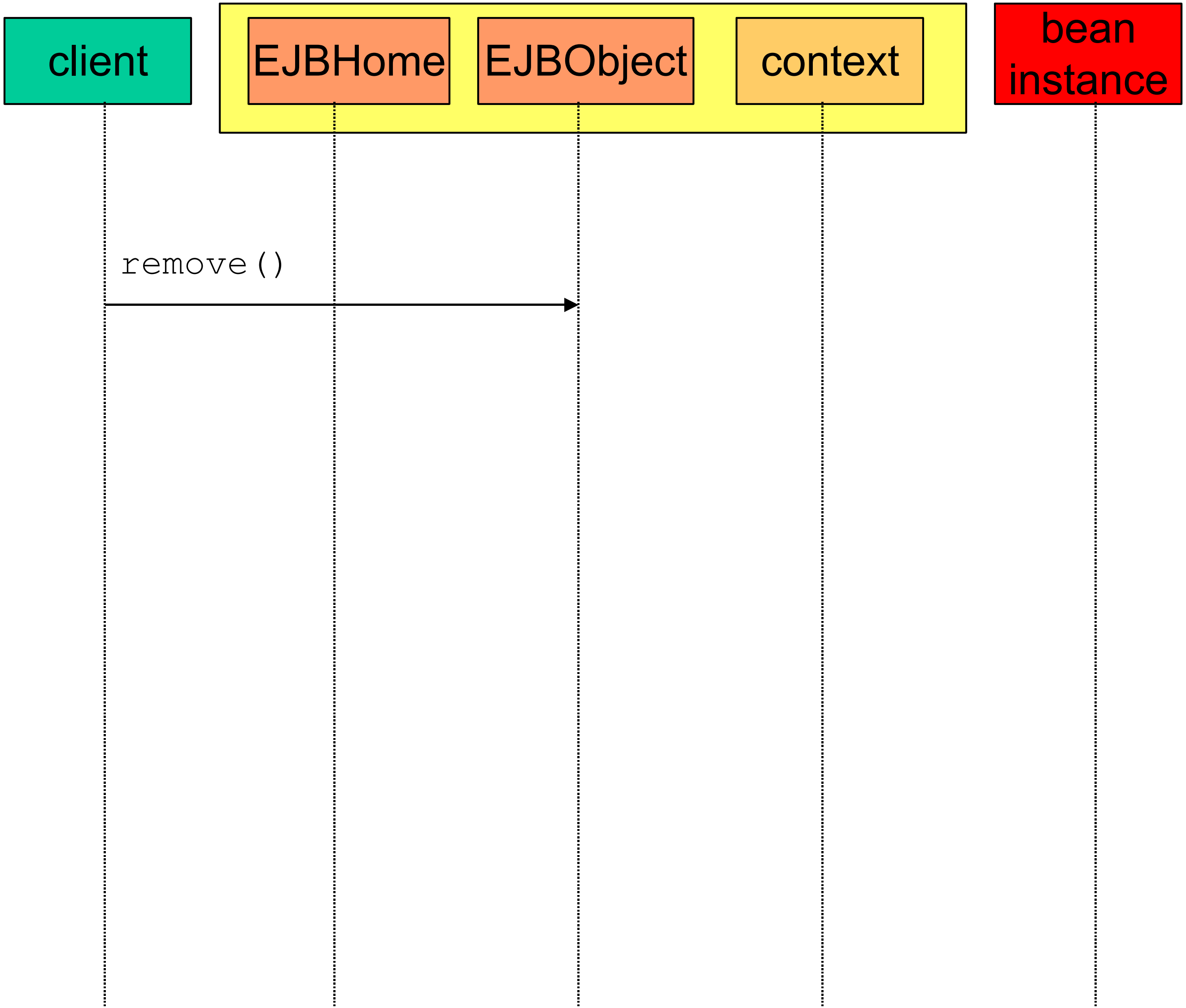
Creating a Stateless Session Bean Instance



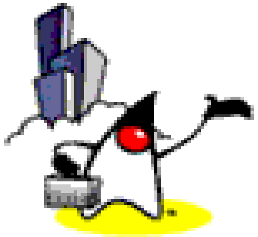
Invocation of a Business Method



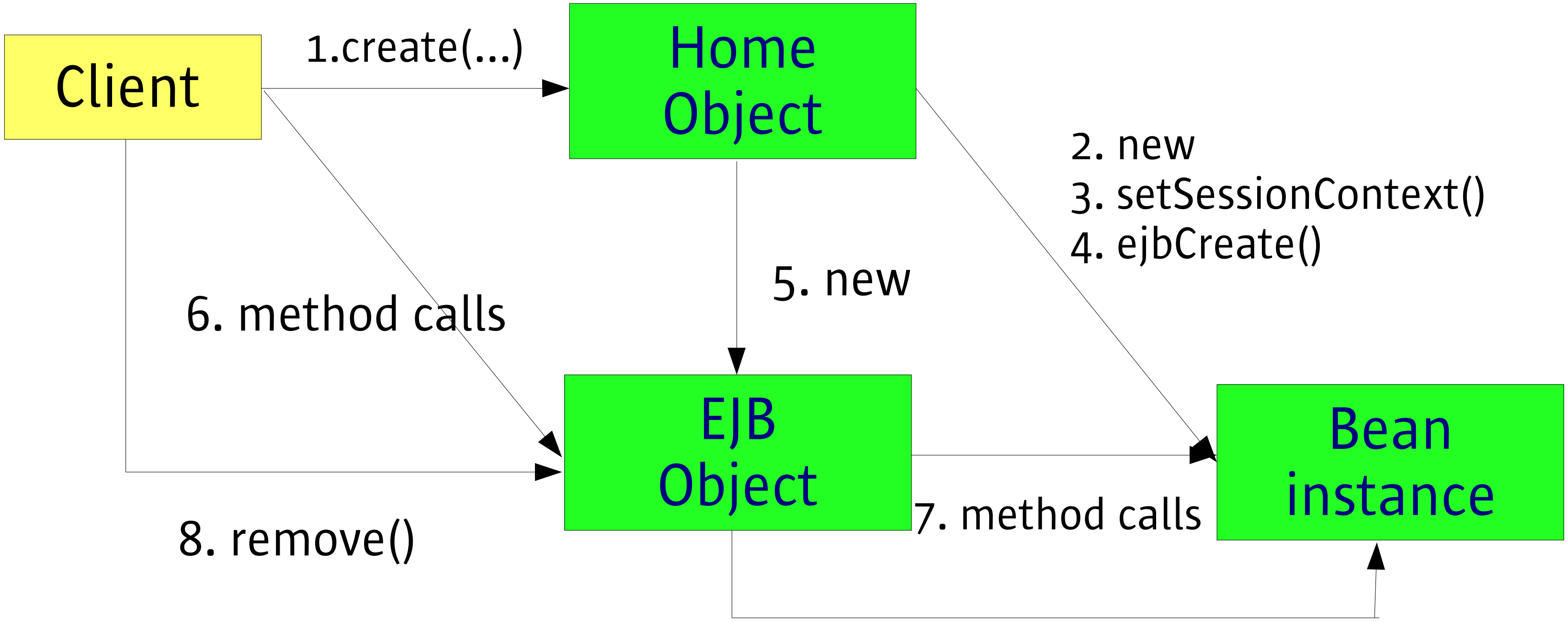
Removal of a Stateless Session Bean I nstance



Sequence of Operations in the usage of Stateful Session Bean



Interaction between Client, Bean instance, Container for Stateful Session Bean



source: Applied Enterprise JavaBeans

Sequence of Operations (Page 1)

1. Client calls a create(...) method on the hom e object
2. The home object or container instantiates bean instance
3. The home object or container creates a Session Context object, and passes it to the instance in a setSessionContext() call
   - SessionContext object contains information on runt ime context

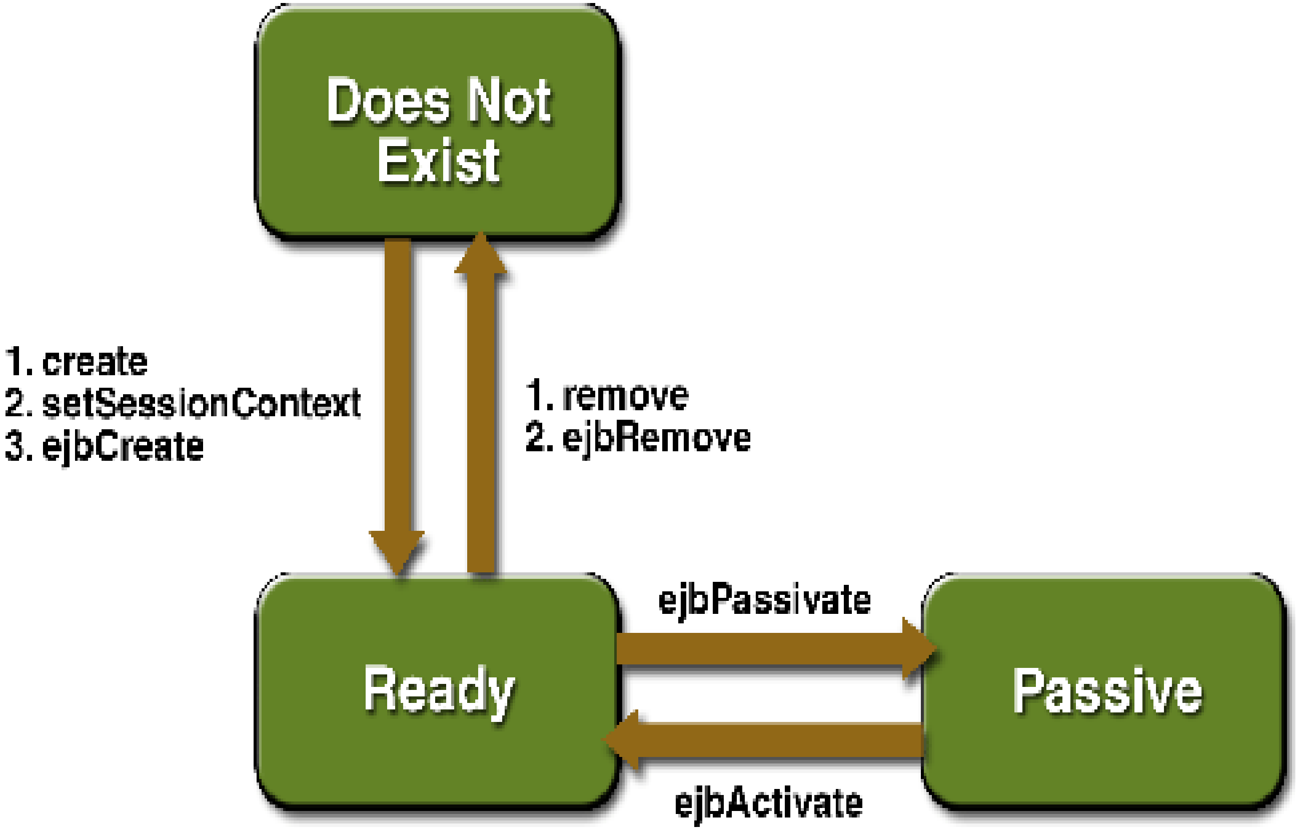
Sequence of Operations (Page 2)

4. The home object or container calls ejbCreate(...) method whose arguments match the create(...) call on the home interface
   - Bean instance initializes itself and is ready for use
5. The home object or container instantiates an EJB object for the bean instance
   - There is typically a one-to-one relationship between clients, EJB objects, and bean instances for stateful session beans

Sequence of Operations

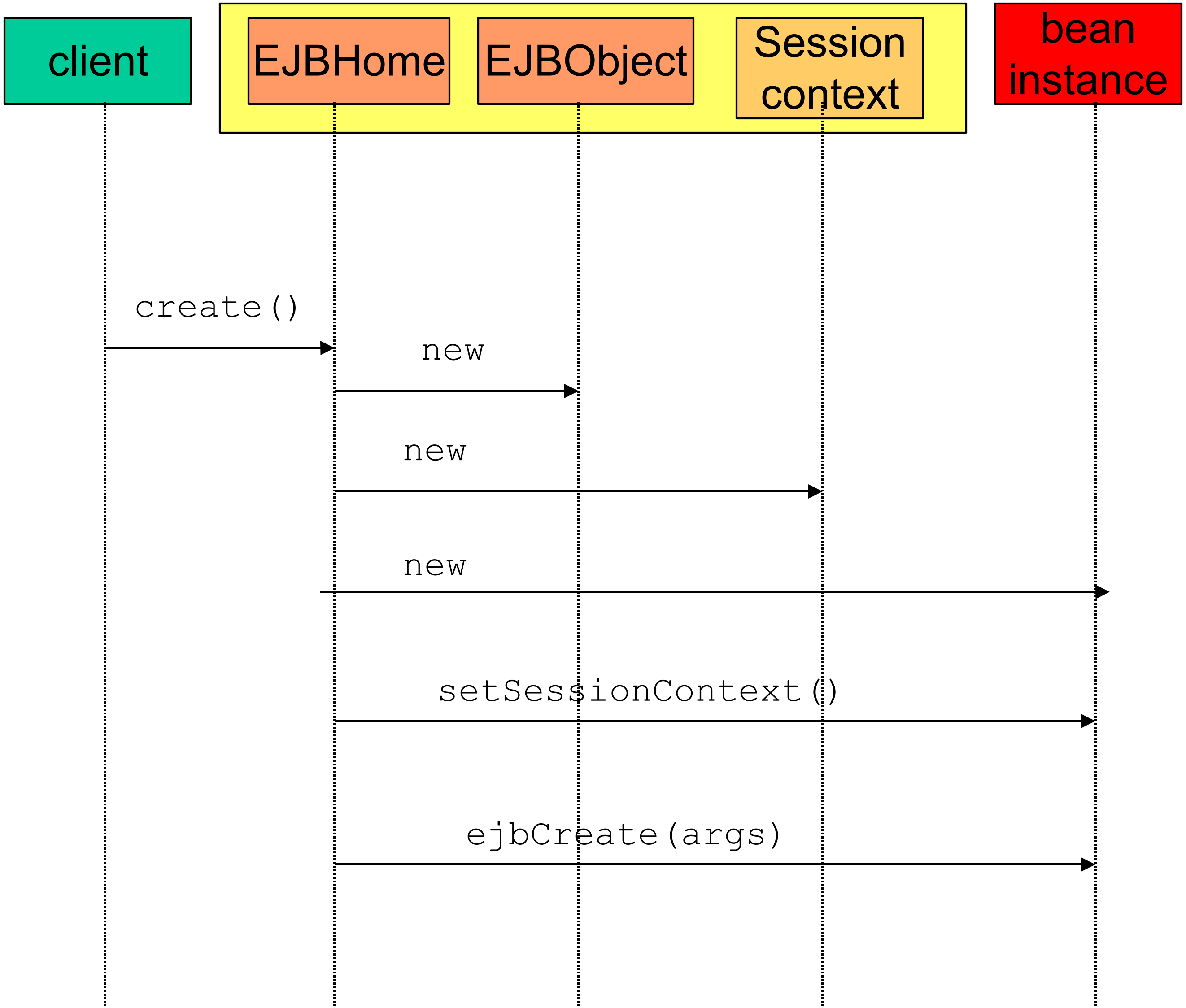
6. The client calls whatever business methods it needs on the EJB object
7. EJB object handles system operations
8. EJB object passes these calls on to the bean instance
9. Client calls remove() on the EJB object
10. EJB object or container calls ejbRemove() on the bean instance

Life Cycle of a Stateful Session Bean Instance

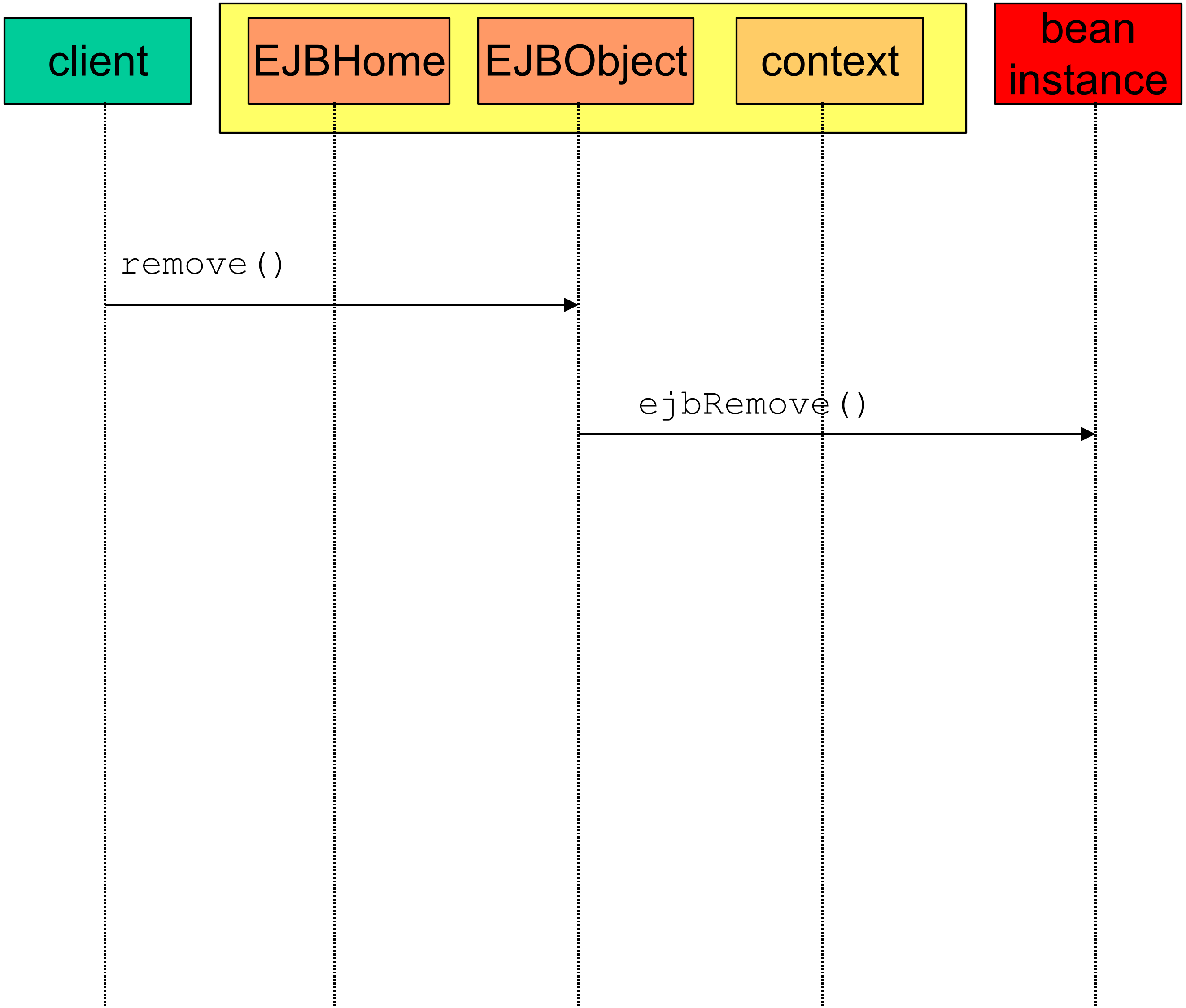


source: J2EE tutorial

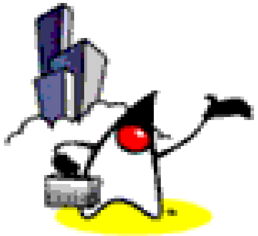
Creating a Stateful Session



Removal of a Stateful Session Object



Codes Bean Developer writes in implementing Stateful Session Bean



Things Bean developer is responsible

- Home interface
  - create(...) methods
- Logic interface (Remote interface)
  - business methods
- Bean class
  - ejbCreate(...) methods and others
  - implementation of business methods
- Deployment descriptor

Bean Class Methods

- ejbCreate(...) methods
- setSessionContext(SessionContext sc)
- ejbRemove()
- ejbActivate()
- ejbPassivate()
- implementation of business methods

ejbCreate(...) Methods

- Implement as many overloaded variants as required (at least one) with appropriate arguments to initialize the bean instance
- In a failure, throw CreateException

setSessionContext(SessionContext sc)

- As a minimum, store SessionContext object in an instance variable for later use
- Look up data resource and other EJB's

ejbRemove()

- Implement, if desired, to clean up
- You can’t rely on this method ever being called by EJB container
- In a failure, throw RemoveException

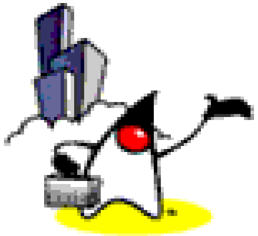
ejbActivate()

- Implement, if necessary, to recover from passivation

ejbPassivate()

- Implement, if necessary, to prepare for passivation

Passivation & Activation in Stateful Session Bean



Need for Passivation & Timeout

- Container has to maintain session state for each client, which takes server resource
- Some clients hang on to their EJB references longer than they need them
  - Servlet and JSP web-tier EJB clients which in turn serve their own browser clients
- Clients may fail (e.g., crash) without calling remove()
- Increasing server resources is not always possible/desirable

Activation & Passivation

- Stateful Session Bean Clients aren't always active
- A server can only handle a limited number of clients
- Therefore a method is needed to store and retrieve Stateful Session Beans
- Stateful Session Beans are automatically stored and retrieved by the container
- Housekeeping needs to be handled by the programmer before a Stateful Session been is stored (passivated) or restored (activated) by the containter

Passivation

- Before a Stateful Session Bean is stored non-serializable data/resources must be reconciled
  - Database, URL, and Java TCP/IP connections must be closed
  - Transient fields must be dealt with
- The EJB specification provides the ejbpassivate() method which is called before the EJB is stored

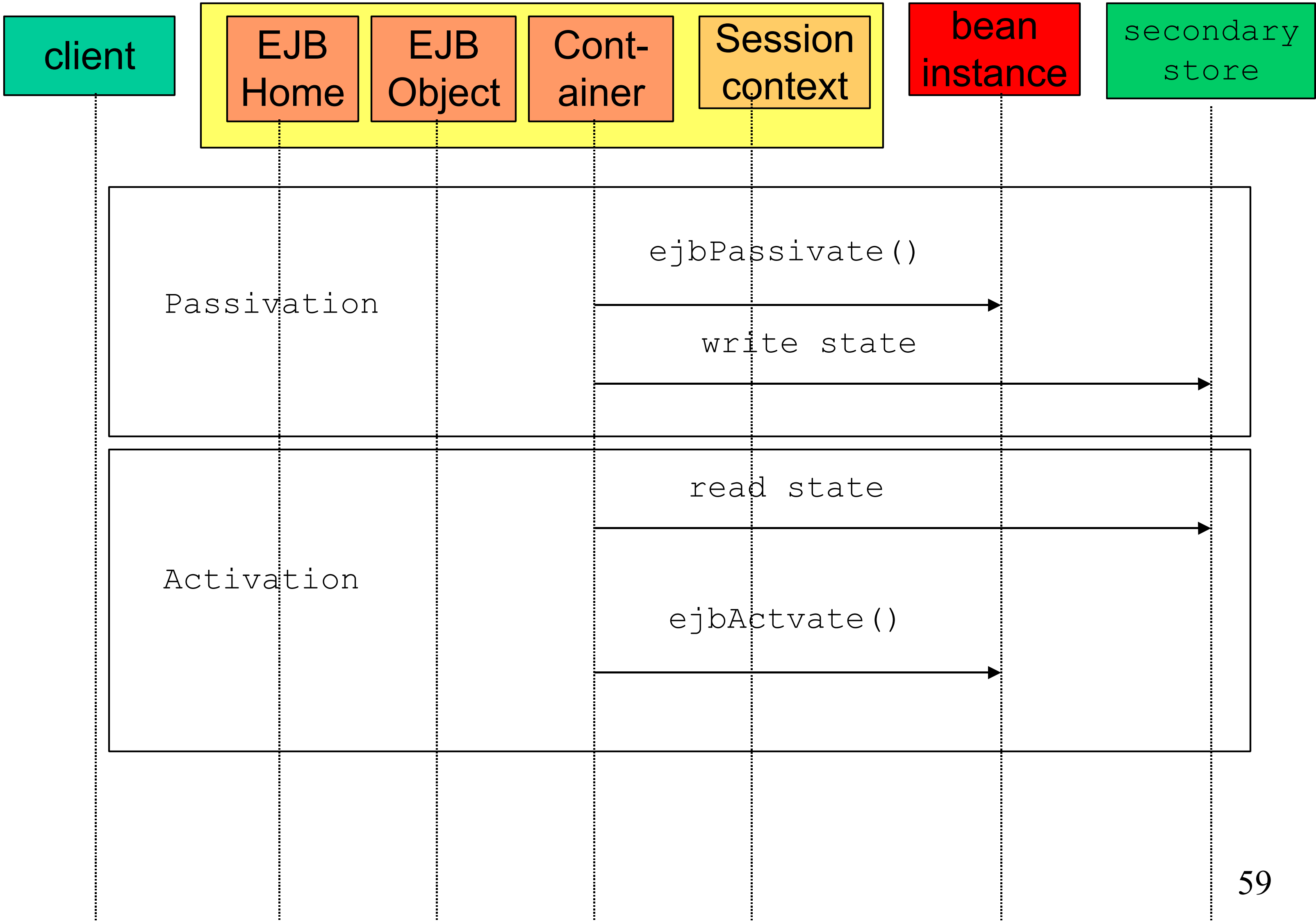
Activation

- After a Stateful Session Bean has been restored non-serializable data/resources must be reconciled
  - Database, URL, and Java TCP/IP connections must be opened
  - Transient fields must be dealt with
- The EJB specification provides the ejbactivate() method which is called after the EJB is restored

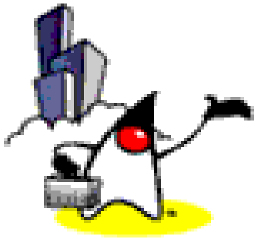
Activation/Passivation Example

```
Public class EJBBean {
        Connection conn = null;
        DataSource dataSource;

        public void ejbCreate() {
                ds =
                conn = ds.getConnection();
        }

        public void ejbActivate() {
                connection = ds.getConnection();
        }

        public void ejbPassivate() {
                conn.close();
                conn = null;
        }

        …
}
```

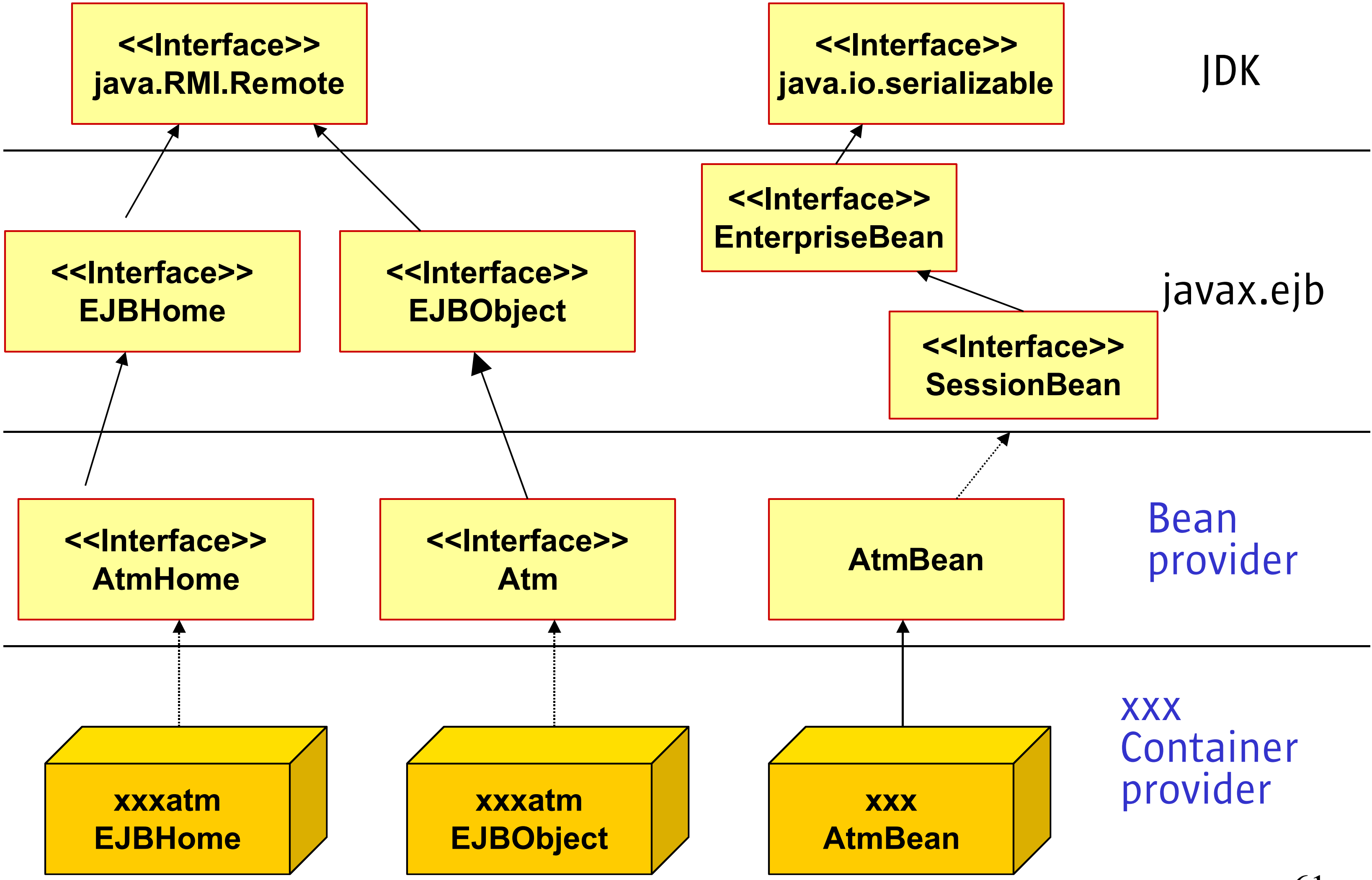
Passivation and Activation of a Stateful Session Object



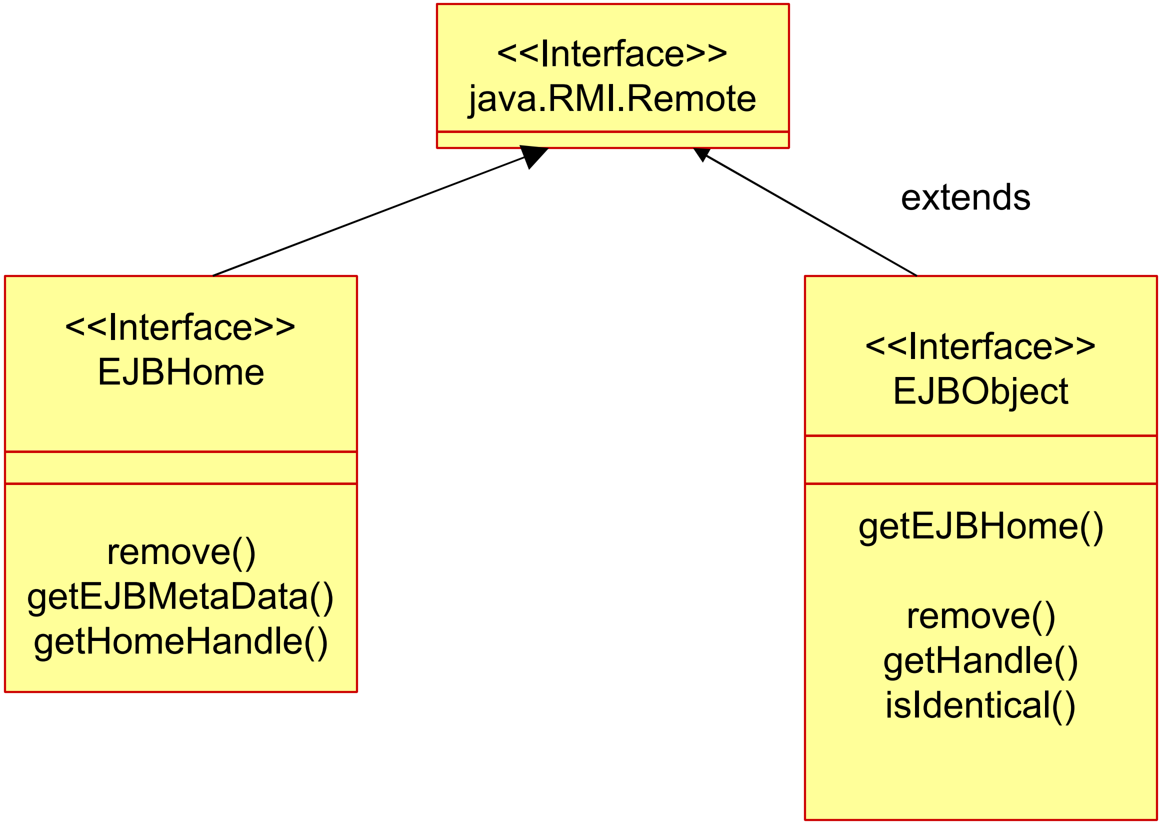
Programming API



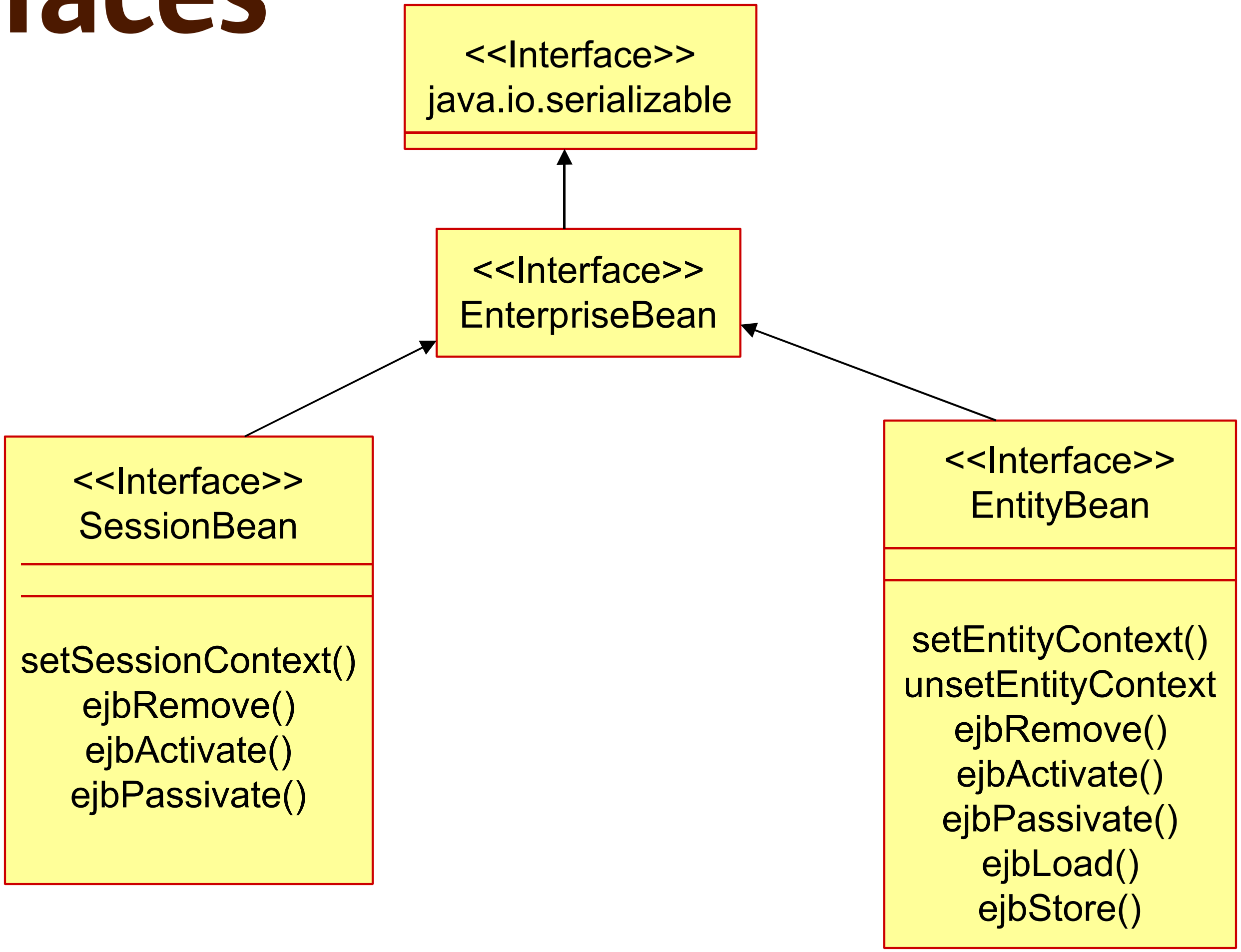
EJB™ Programming API



javax.EJB™ Client Interfaces



javax.EJB™ Server Interfaces



<<Interface>>
java.io.serializable

<<Interface>>
EnterpriseBean

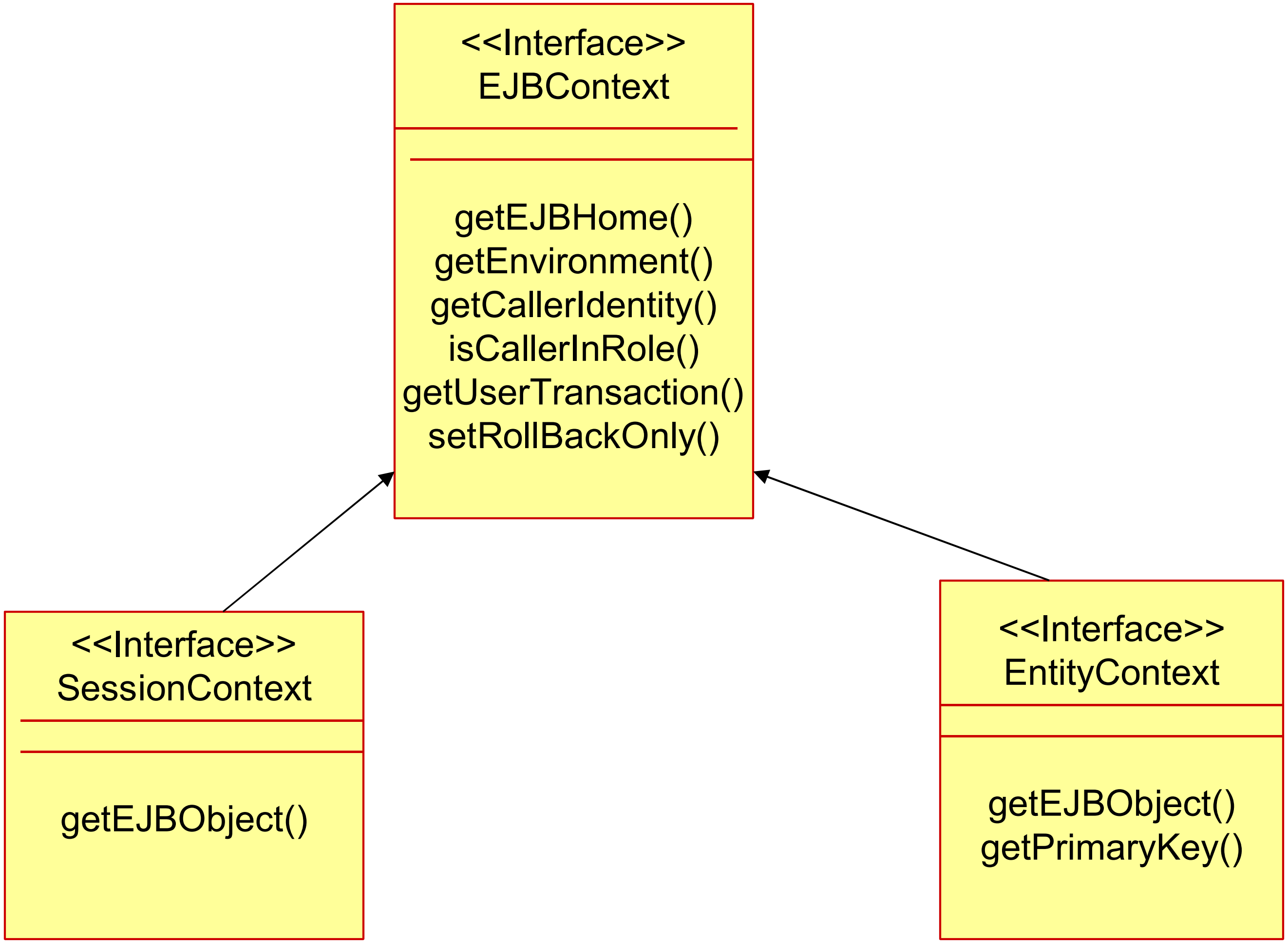
<<Interface>>
SessionBean

setSessionContext()
ejbRemove()
ejbActivate()
ejbPassivate()

<<Interface>>
EntityBean

setEntityContext()
unsetEntityContext
ejbRemove()
ejbActivate()
ejbPassivate()
ejbLoad()
ejbStore()

javax.EJB™ Server Interfaces Cont.



<<Interface>>
EJBContext

getEJBHome()
getEnvironment()
getCallerIdentity()
isCallerInRole()
getUserTransaction()
setRollBackOnly()

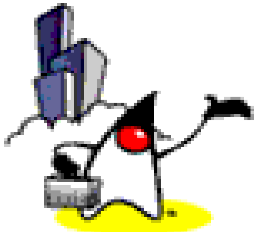
<<Interface>>
SessionContext

getEJBObject()

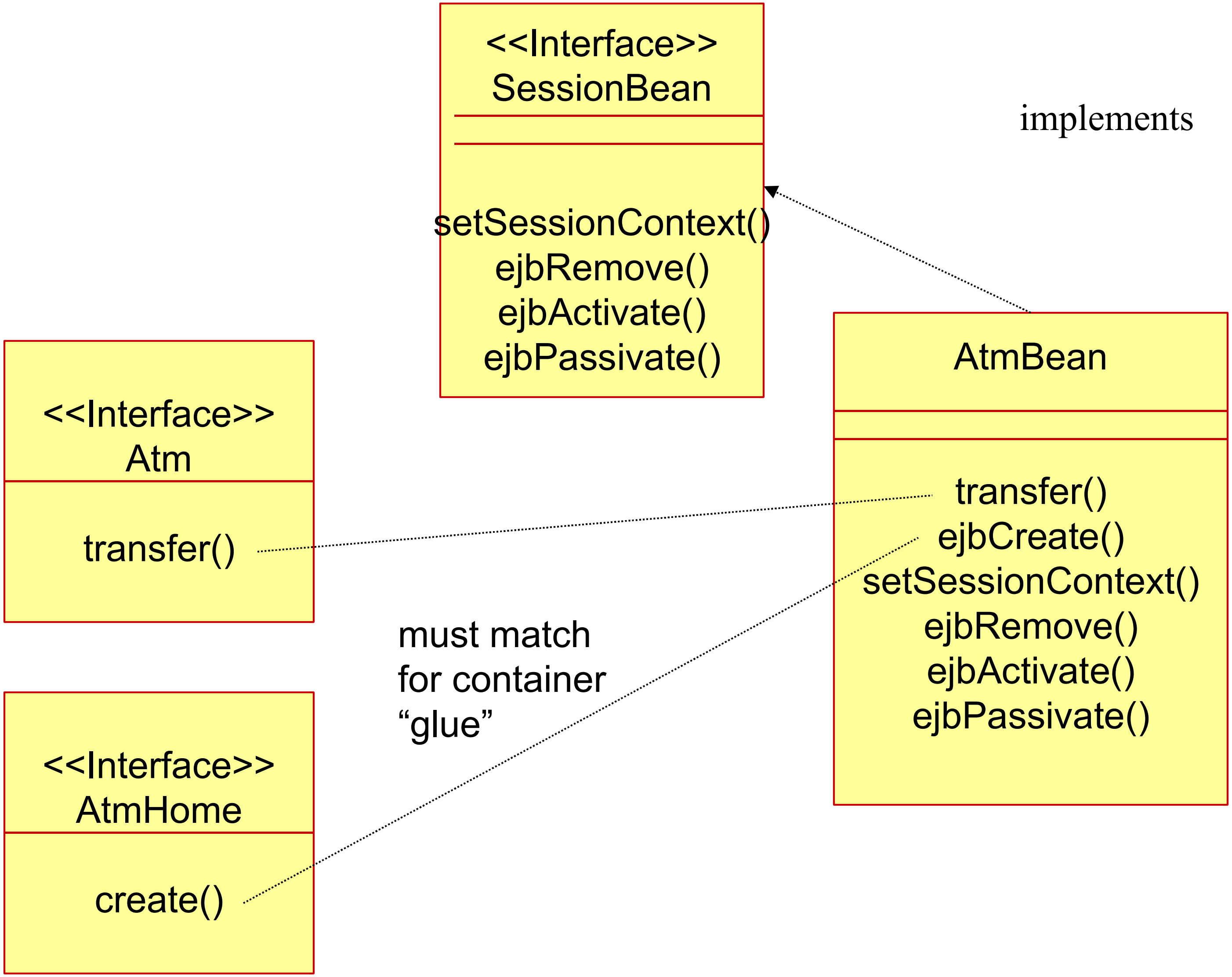
<<Interface>>
EntityContext

getEJBObject()
getPrimaryKey()

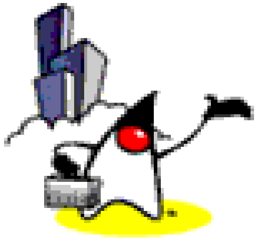
Method Relationship Among Home interface, Remote interface, Bean class



3) AtmBean Implementation



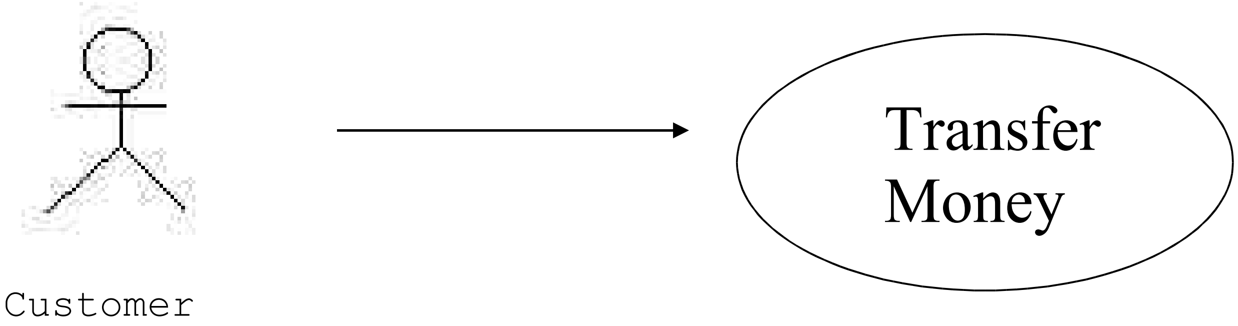
Example Stateless Session Bean



Implementing a Session Bean

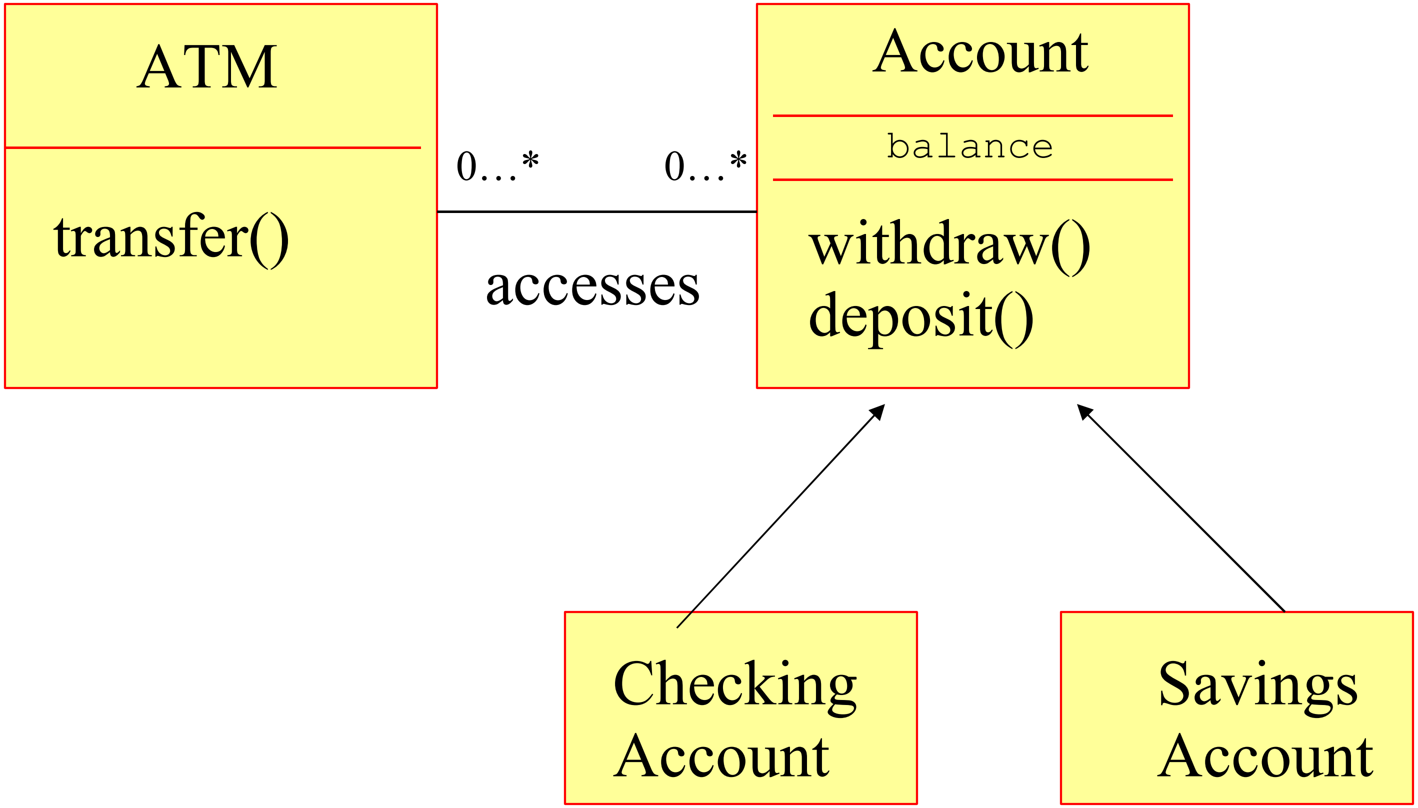
- In this session we will discuss an example stateless session bean, we will discuss stateful in a later session.
- As an example we will use the ATM session bean from the bank account transfer scenario.

Example Scenario: Use Case

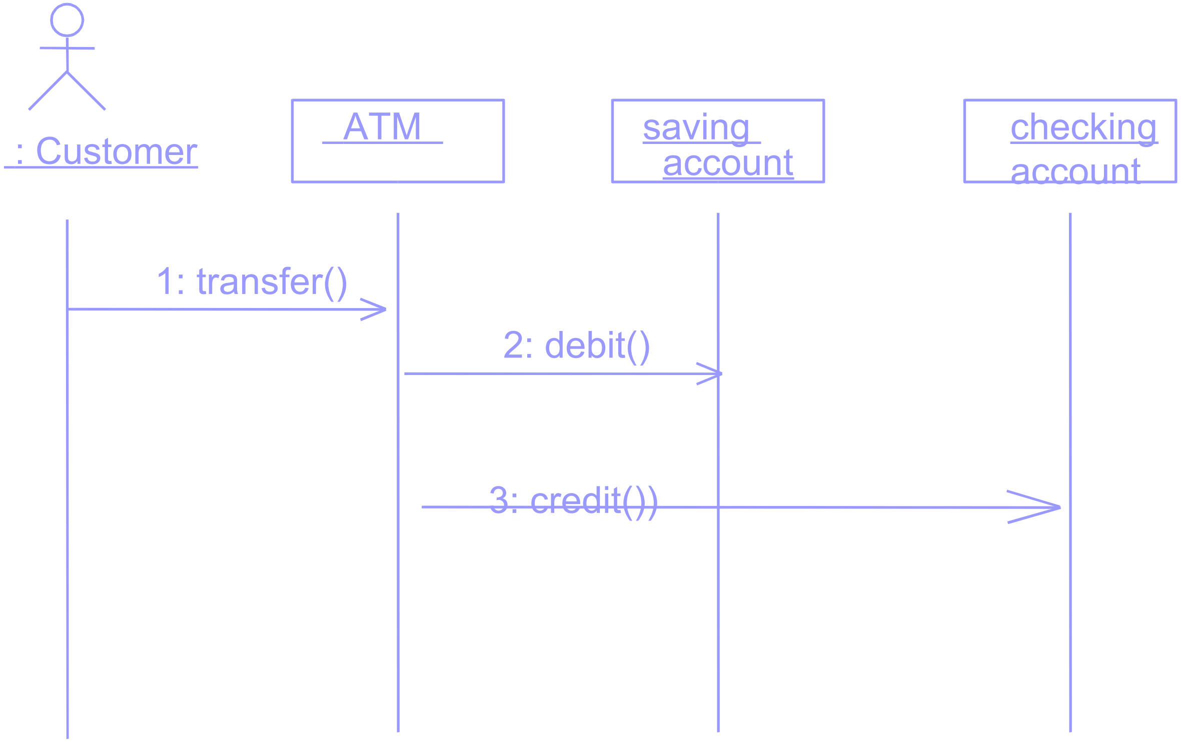


Use Case: ATM customer transfers money from checking to savings account

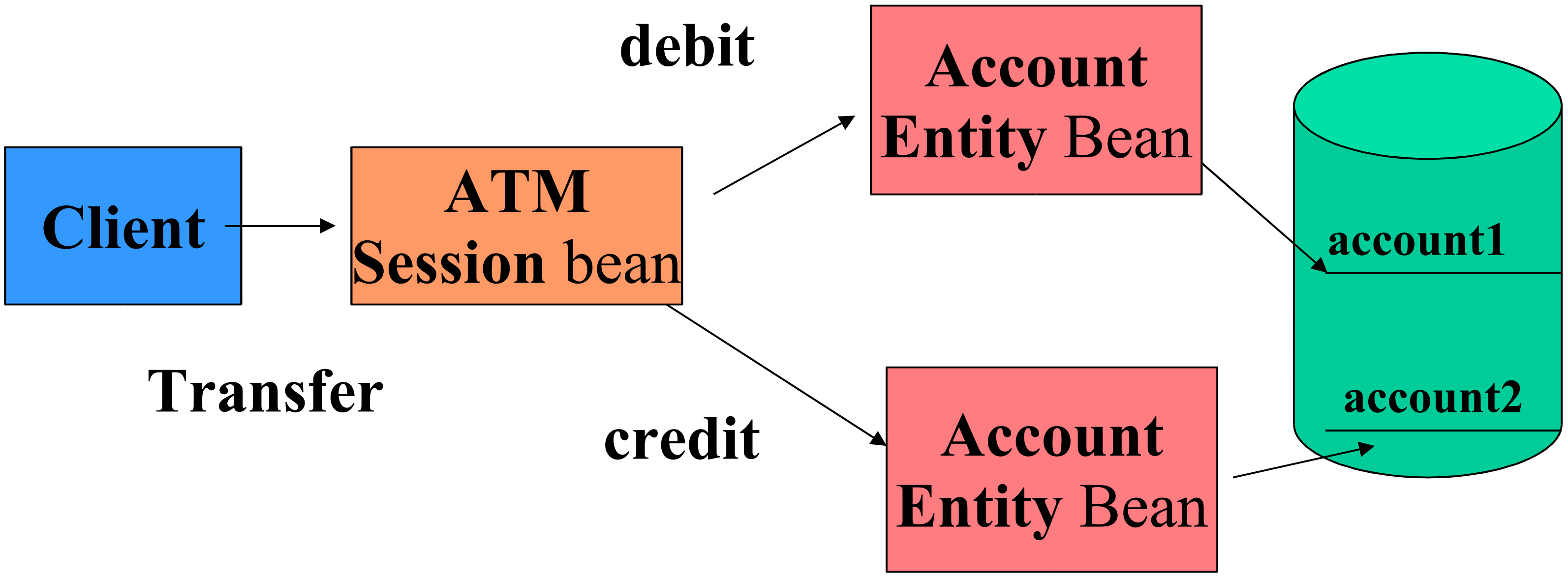
Example Scenario: Classes



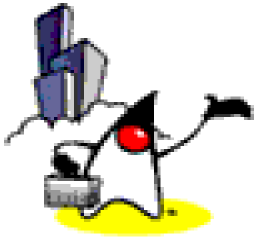
Example Scenario: Sequence Diagram



Example Scenario: EJB



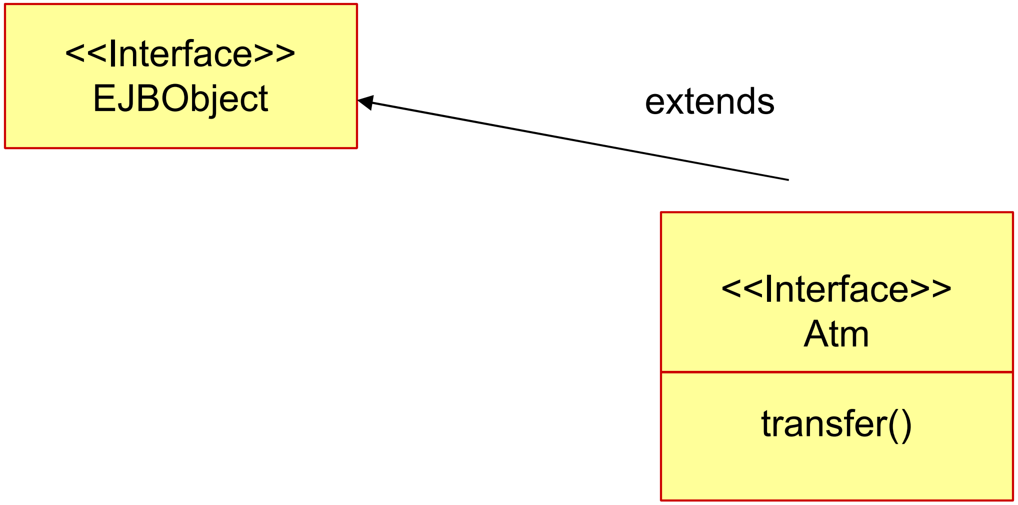
Steps for Creating Stateless Session Bean



Session Bean Implementation

1. Create the logic interface for the bean.
2. Create the bean’s home interface.
3. Create the bean’s implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

1) Create the Remote Interface



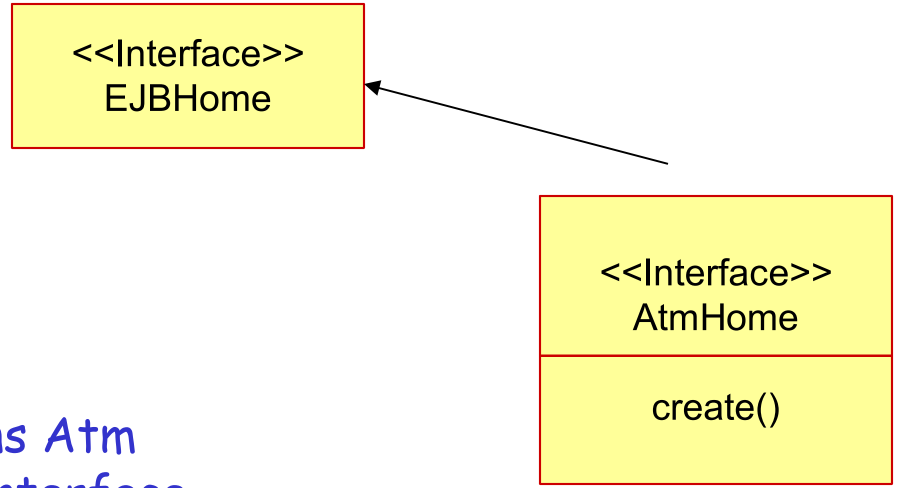
Define Business Methods:

```
public interface  Atm extends javax.ejb. EJBObject {

    public void  transfer (int fromAcctId, int toAcctId,
                               double amount)
        throws java.rmi.RemoteException,InsufficientFundsException;

}
```

Session Bean Implementation

1. Create the remote interface for the bean.
2. Create the bean’s home interface.
3. Create the bean’s implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

2) Create the Home Interface



Returns Atm remote interface

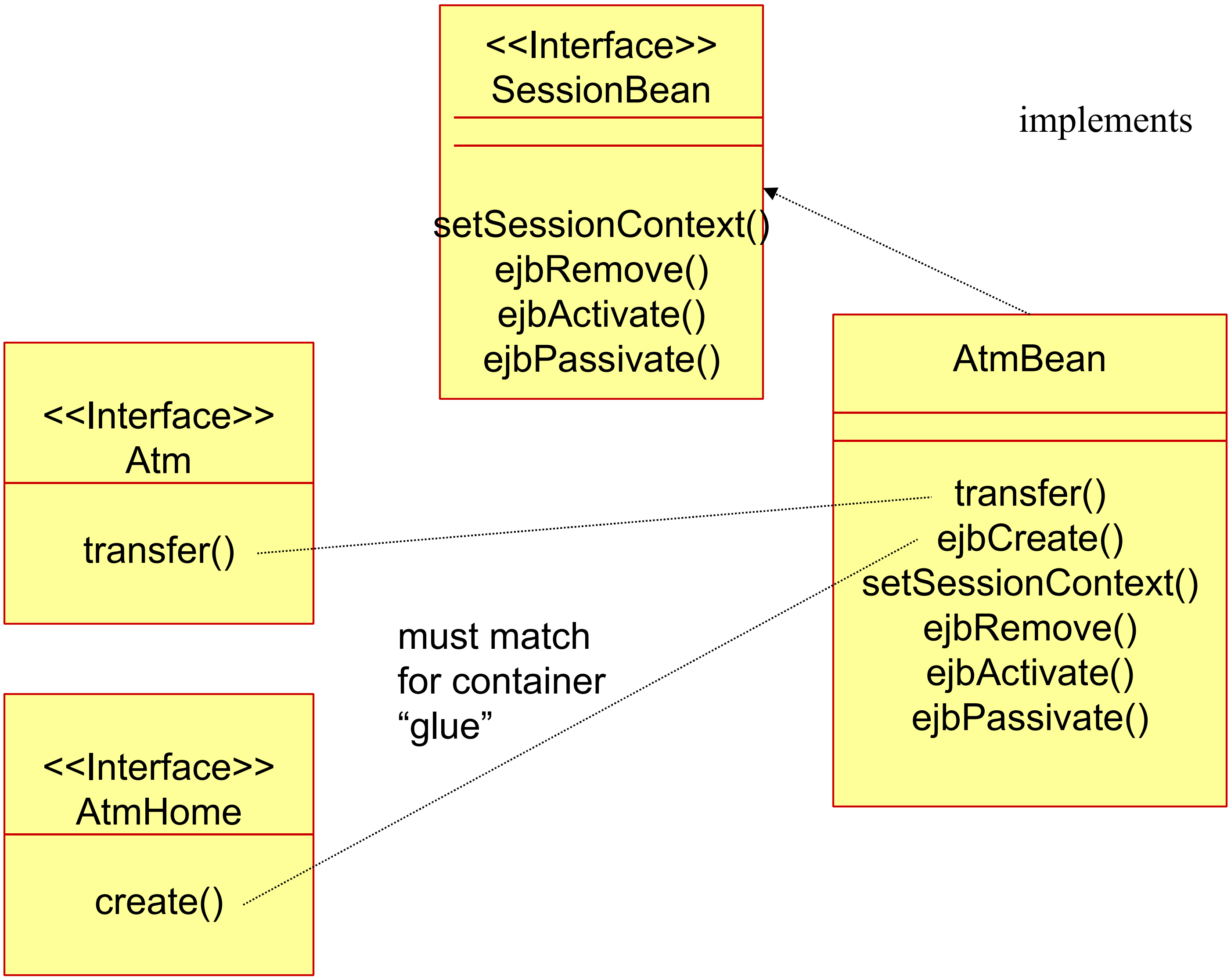
Define Create Methods:

```
public  interface  AtmHome  extends javax.ejb.  EJBHome  {

    Atm  create () throws java.rmi.RemoteException,
         javax.ejb.CreateException;

}
```

Session Bean Implementation

1. Create the remote interface for the bean.
2. Create the bean's home interface.
3. Create the bean's implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

3) AtmBean Implementation



3) AtmBean: Implement Atm Interface Business Methods

```
public class  AtmBean  implements  SessionBean  {

 // implement atm interface business methods
 public void  transfer (int fromAcctId,int toAcctId,double amount)
    throws InsufficientFundsException, FinderException {
    try {
       fromAccount = accountHome.  findByPrimaryKey(  new Integer(fromAcctId));
    } catch(FinderException ex) {
         throw new FinderException("Couldnt find account"+fromAcctId );
    }
    try {
         toAccount = accountHome.  findByPrimaryKe  y(new Integer(toAcctId));
    } catch(FinderException ex) {
      throw new FinderException("Couldnt find account");
    }
    try {
        fromAccount.withdraw(amount);
        toAccount.deposit(amount);
    } catch(InsufficientFundsException ex) {
         throw new InsufficientFundsException("Insufficient funds " + fromAcctId);
    }
 }
```

3) AtmBean: Implement Home Interface Create Method

```
//        implement atmHome interface create
 private static AccountHome accountHome = null;
 public void ejbCreate (){
  try {
    Context ic = new InitialContext();

   java.lang.Object objref = ic.lookup("java:comp/env/ejb/Account");
    accountHome=(AccountHome)PortableRemoteObject.narrow(objref,
                            AccountHome.class);
   } catch (NamingException ne) {
              System.err.println("ejbCreate: Caught unexpected
NamingException:");
   }
}
```

3) AtmBean: Implement Session Interface Container Callback Methods

```
// save the session context in an instance variable
public void setSessionContext(SessionContext sc) {
        this.context= sc;
}

// release resources allocated in ejbCreate
public void ejbRemove() throws RemoveException {
        accountHome = null;
}

// Stateless Session Beans are not activated/passivated
// so these methods are always empty
public void ejbActivate() {
}
public void ejbPassivate() {
}
```

Session Bean Implementation

1. Create the remote interface for the bean.
2. Create the bean's home interface.
3. Create the bean's implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

4) Compile the Remote & Home Interfaces and Implementation Class.

```
javac –classpath $J2EE_HOME/lib/j2ee.jar
          Atm.java AtmHome.java
          AtmEJB.java
```

Session Bean Implementation

1. Create the remote interface for the bean.
2. Create the bean’s home interface.
3. Create the bean’s implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

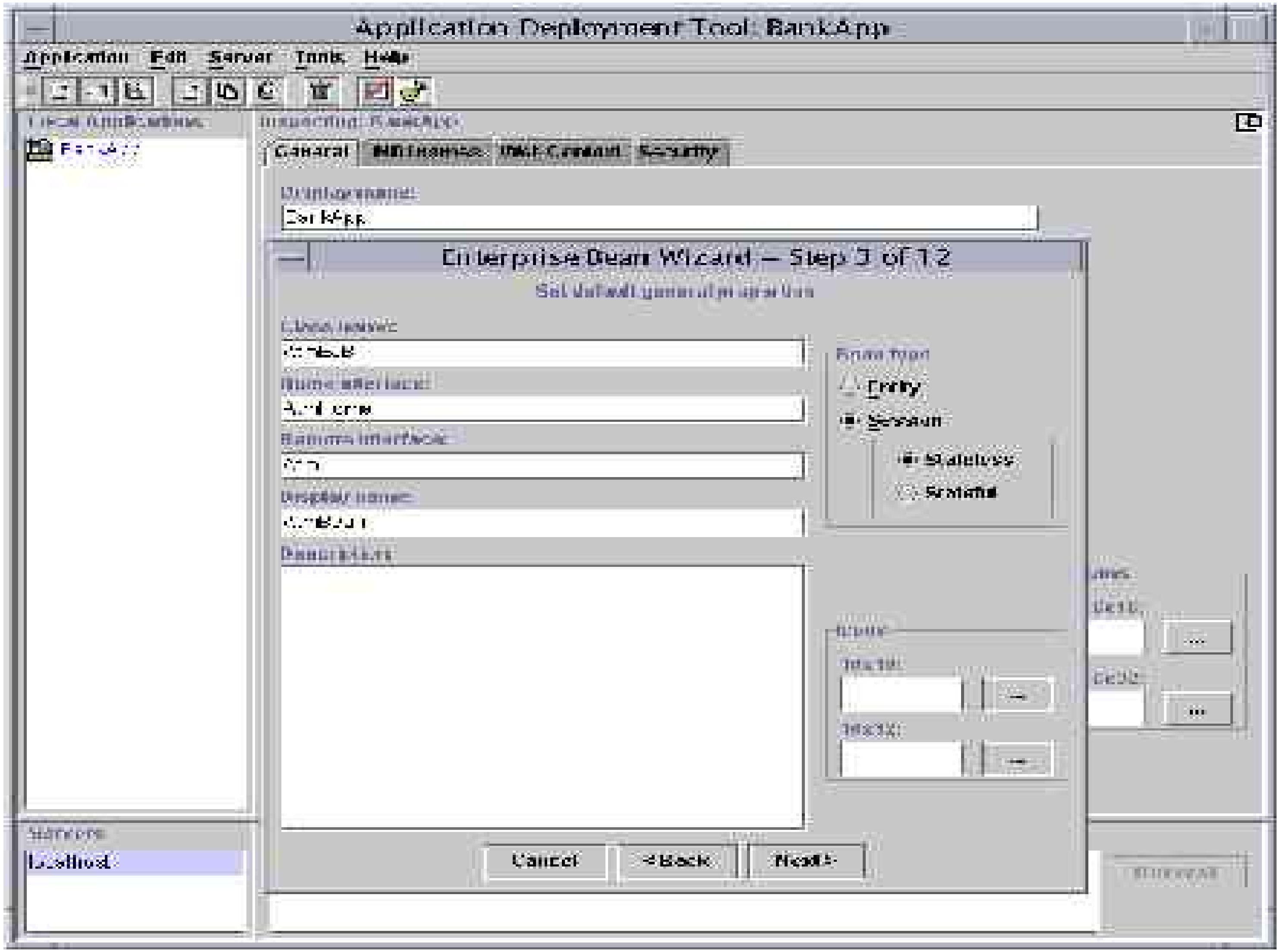
EJB descriptor composition

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//SunMicrosystems Inc.//DTD>
<ejb-jar>
    <enterprise-beans>
        <session>
        . . .
        </session>
    </enterprise-beans>

    <assembly-descriptor>
     . . .
    </assembly-descriptor>
</ejb-jar>
```

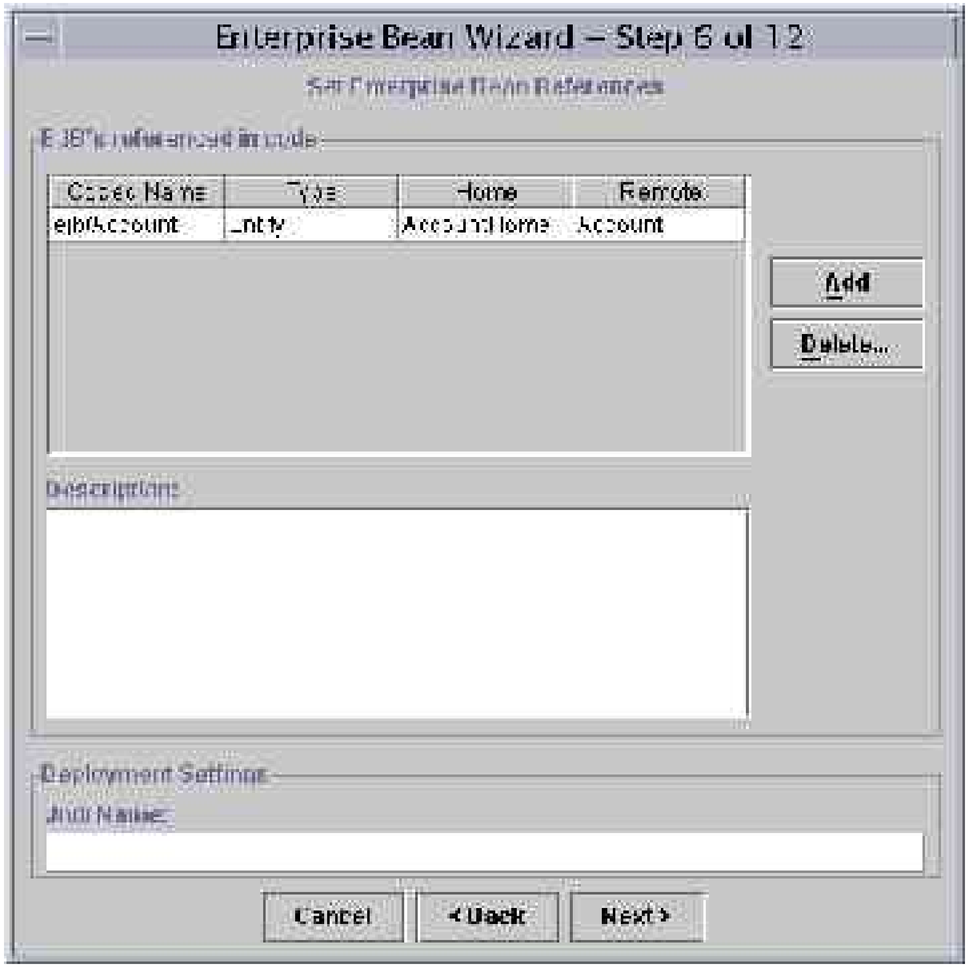
Description of all EJBs in the jar archive

Assembly data

5) Create Deployment Descriptor



Create Deployment Descriptor



Enterprise Bean Wizard – Step 6 of 12

Set Enterprise Bean References

EJB's referenced in code

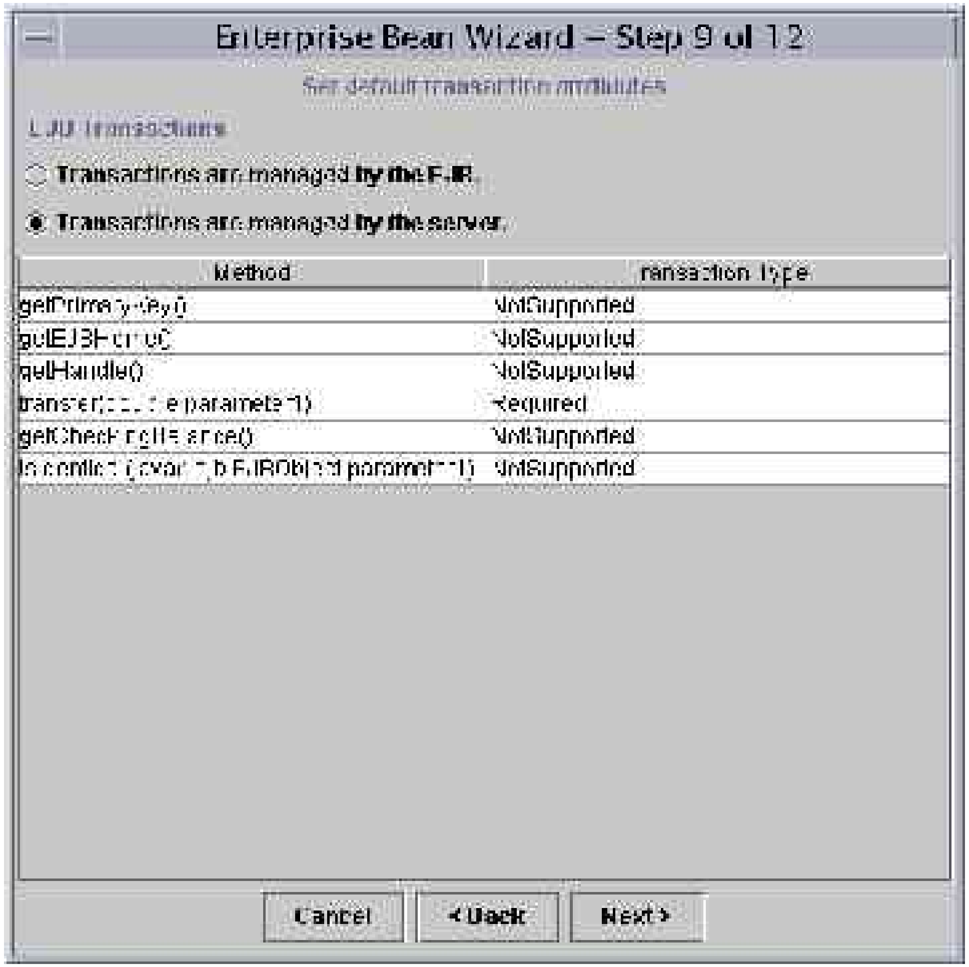
Coded Name	Type	Home	Remote
ejb/Account	Entity	AccountHome	Account

Description:

Deployment Settings

JNDI Name:

Create Deployment Descriptor



Enterprise Bean Wizard – Step 9 of 12

Set default transaction attributes

EJB transactions

○ Transactions are managed by the EJB.

● Transactions are managed by the server.

Method	Transaction Type
getPrimaryKey()	NotSupported
getEJBHome()	NotSupported
getHandle()	NotSupported
transfer(double parameter1)	Required
getCheckingBalance()	NotSupported
isIdentical(javax.ejb.EJBObject parameter1)	NotSupported

Create Deployment Descriptor

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems Inc.//DTD.. . >
<ejb-jar>
  <description>no description</description>
  <display-name>Atm</display-name>
  <enterprise-beans>
    <session>
      <description>no description</description>
      <display-name>AtmBean</display-name>
      <ejb-name>AtmBean</ejb-name>
      <home>AtmHome</home>
      <remote>Atm</remote>
      <ejb-class>AtmEJB</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <ejb-ref>
        <description>no description</description>
        <ejb-ref-name>ejb/Account</ejb-ref-name>
        <ejb-ref-type>Entity</ejb-ref-type>
        <home>AccountHome</home>
        <remote>Account</remote>
      </ejb-ref>
    </session>
```

5) Create DD Cont.

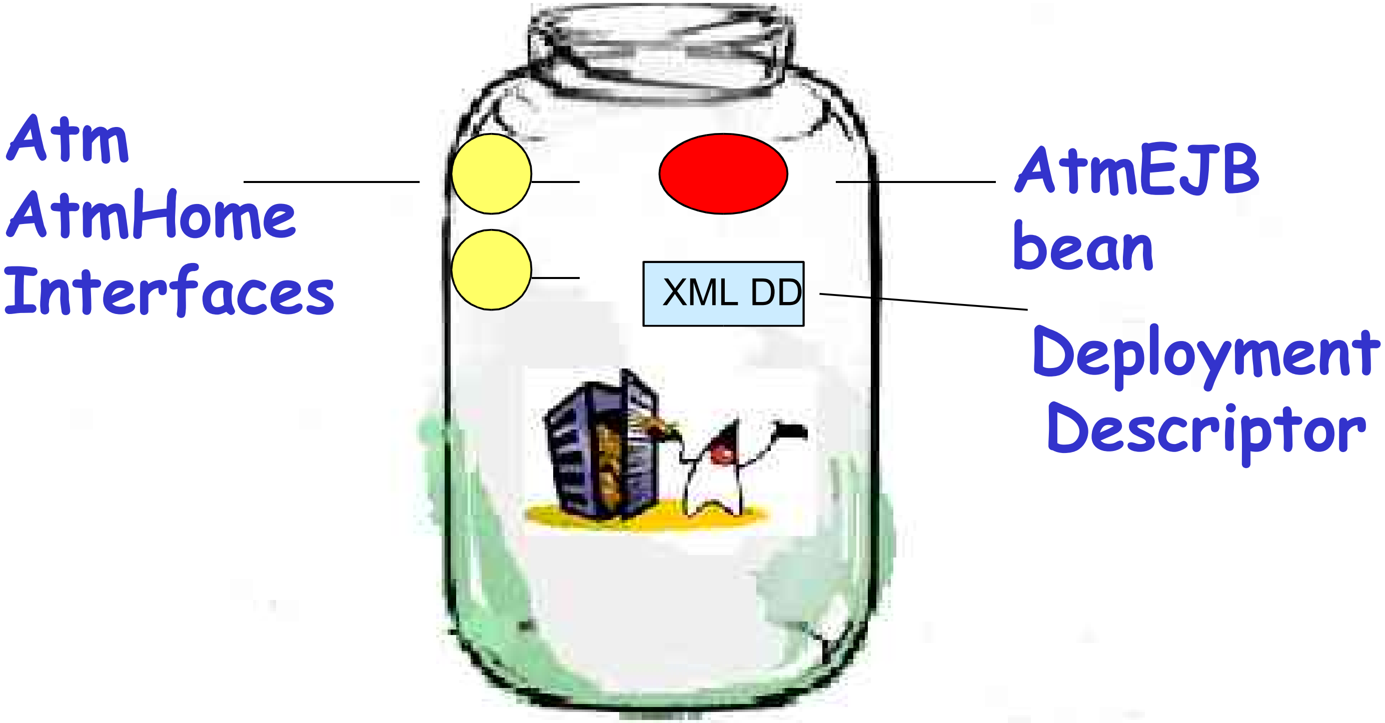
```
</enterprise-beans>
  <assembly-descriptor>
    <container-transaction>
      <method>
       <ejb-name>AtmBean</ejb-name>
       <method-intf>Remote</method-intf>
       <method-name>transfer</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

Session Bean Implementation

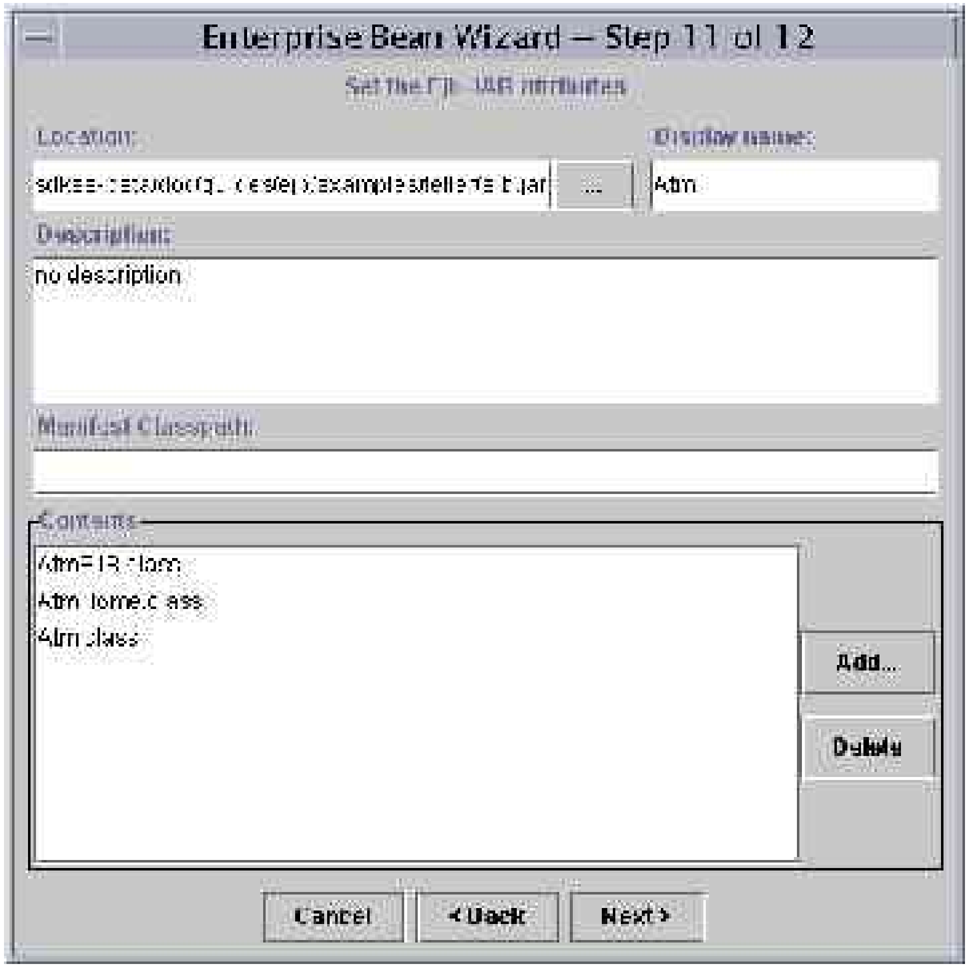
1. Create the remote interface for the bean.
2. Create the bean's home interface.
3. Create the bean's implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

6) Package in an ejb-jar File.

packager –ejbJar
Atm.class:AtmEJB.class:AtmHome.class
Atm-ejb-jar.xml Atm.jar



6) Package in an ejb-jar File.


Enterprise Bean Wizard — Step 11 of 12
Set the EJB JAR attributes
Add...
Delete
Cancel
< Back
Next >

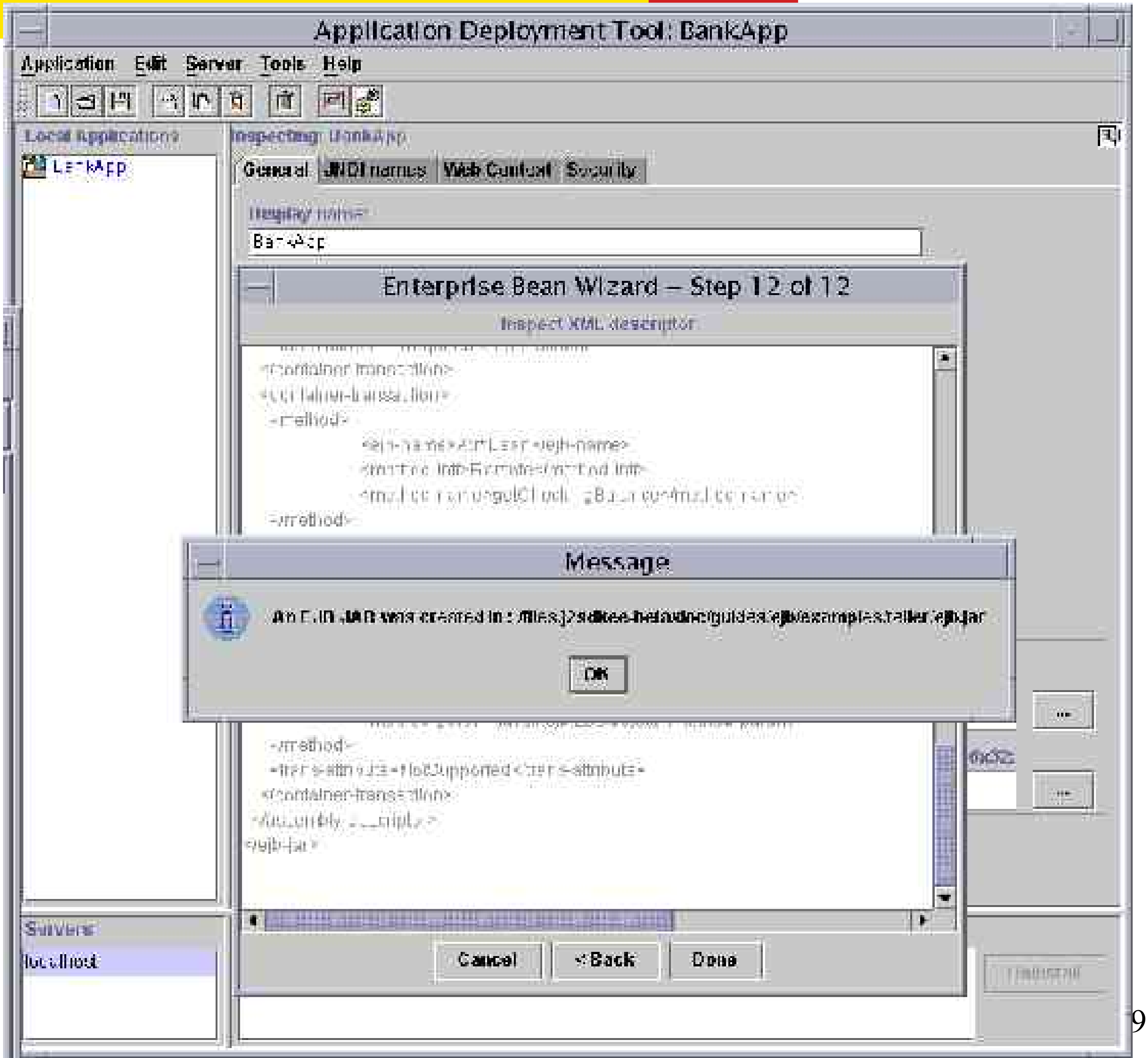
Location: sdkee-1_3/doc/guides/ejb/examples/[illegible] ejb.jar

Display name: Atm

Description: no description

Manifest Classpath:

Contents:
AtmEJB.class
AtmHome.class
Atm.class



Application Deployment Tool: BankApp
Application Edit Server Tools Help
Local Applications
BankApp
Inspecting: BankApp
General JNDI names Web Context Security
Display name:
Servers
localhost
Enterprise Bean Wizard – Step 12 of 12
Inspect XML descriptor
Message
OK
Cancel
< Back
Done

BankApp

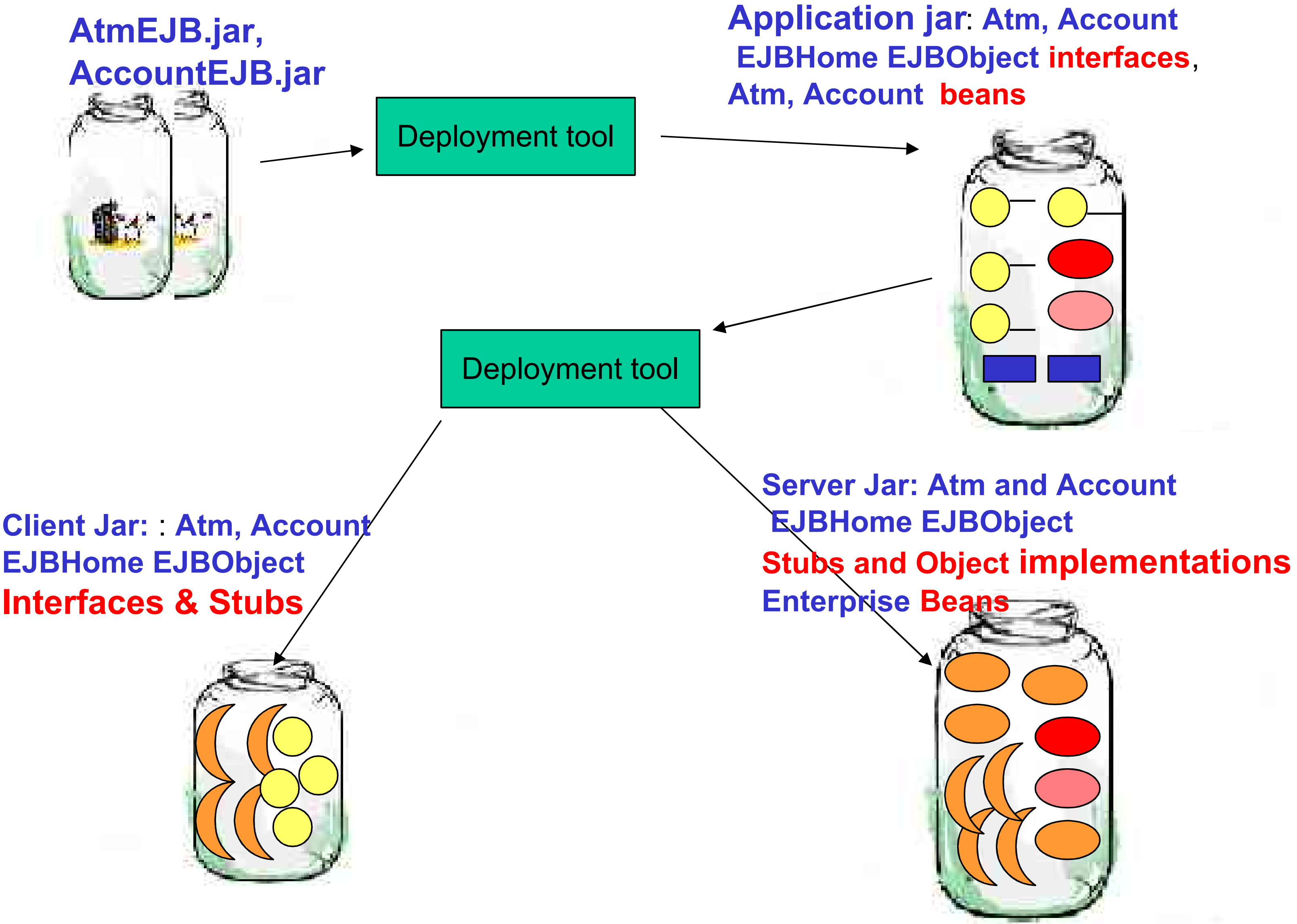
```
<container-transaction>
<container-transaction>
  <method>
        <ejb-name>[illegible]</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>[illegible]</method-name>
  </method>
  </method>
  <trans-attribute>NotSupported</trans-attribute>
 </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

An EJB JAR was created in: /files/j2sdkee-beta/doc/guides/ejb/examples/teller/ejb.jar

Session Bean Implementation

1. Create the remote interface for the bean.
2. Create the bean's home interface.
3. Create the bean's implementation class.
4. Compile the remote interface, home interface, and implementation class.
5. Create a deployment descriptor.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

7) Deploy the Enterprise Application



AtmEJB.jar,
AccountEJB.jar

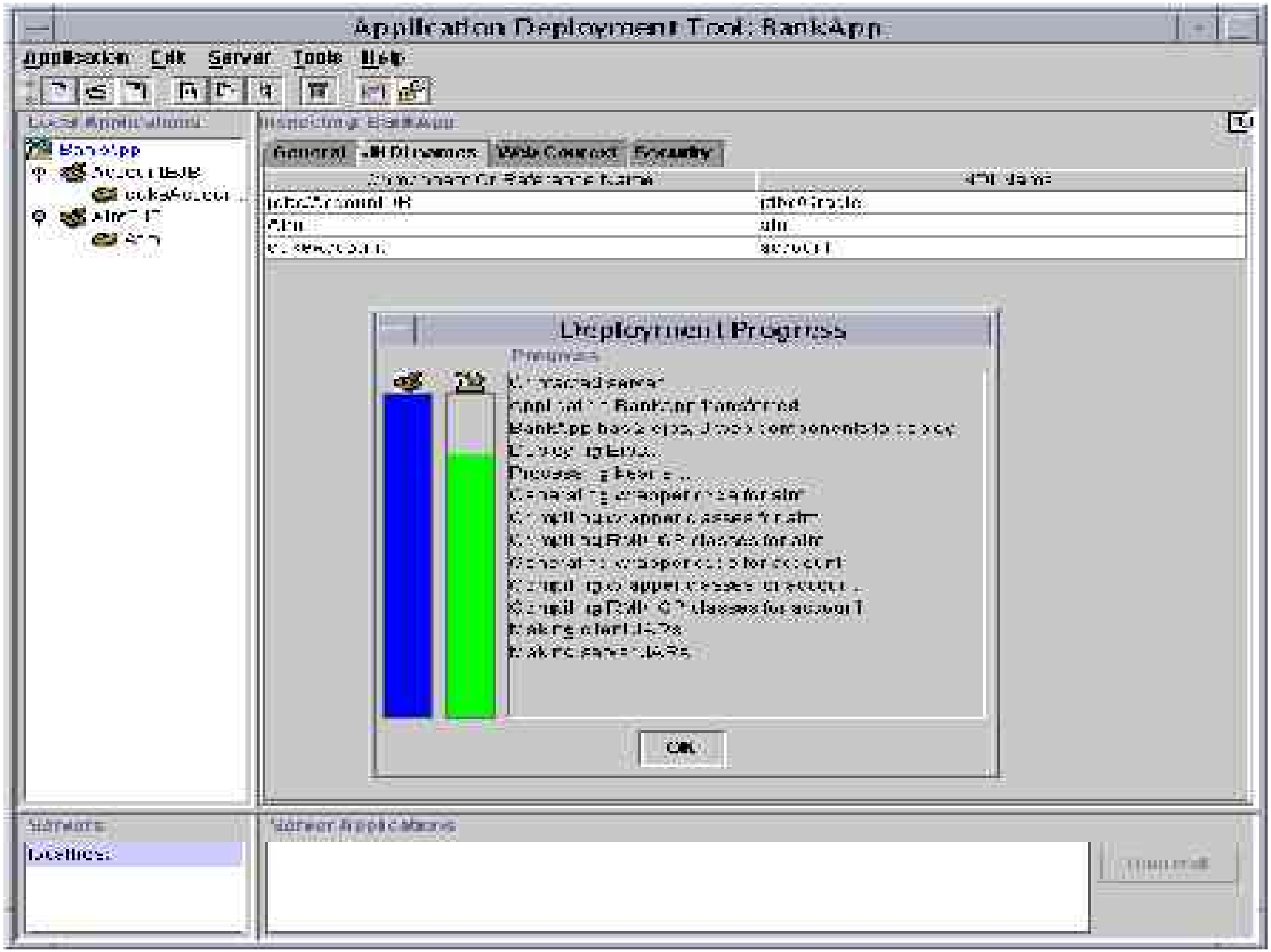
Deployment tool

Application jar: Atm, Account
EJBHome EJBObject interfaces,
Atm, Account beans

Deployment tool

Client Jar: : Atm, Account
EJBHome EJBObject
Interfaces & Stubs

Server Jar: Atm and Account
EJBHome EJBObject
Stubs and Object implementations
Enterprise Beans



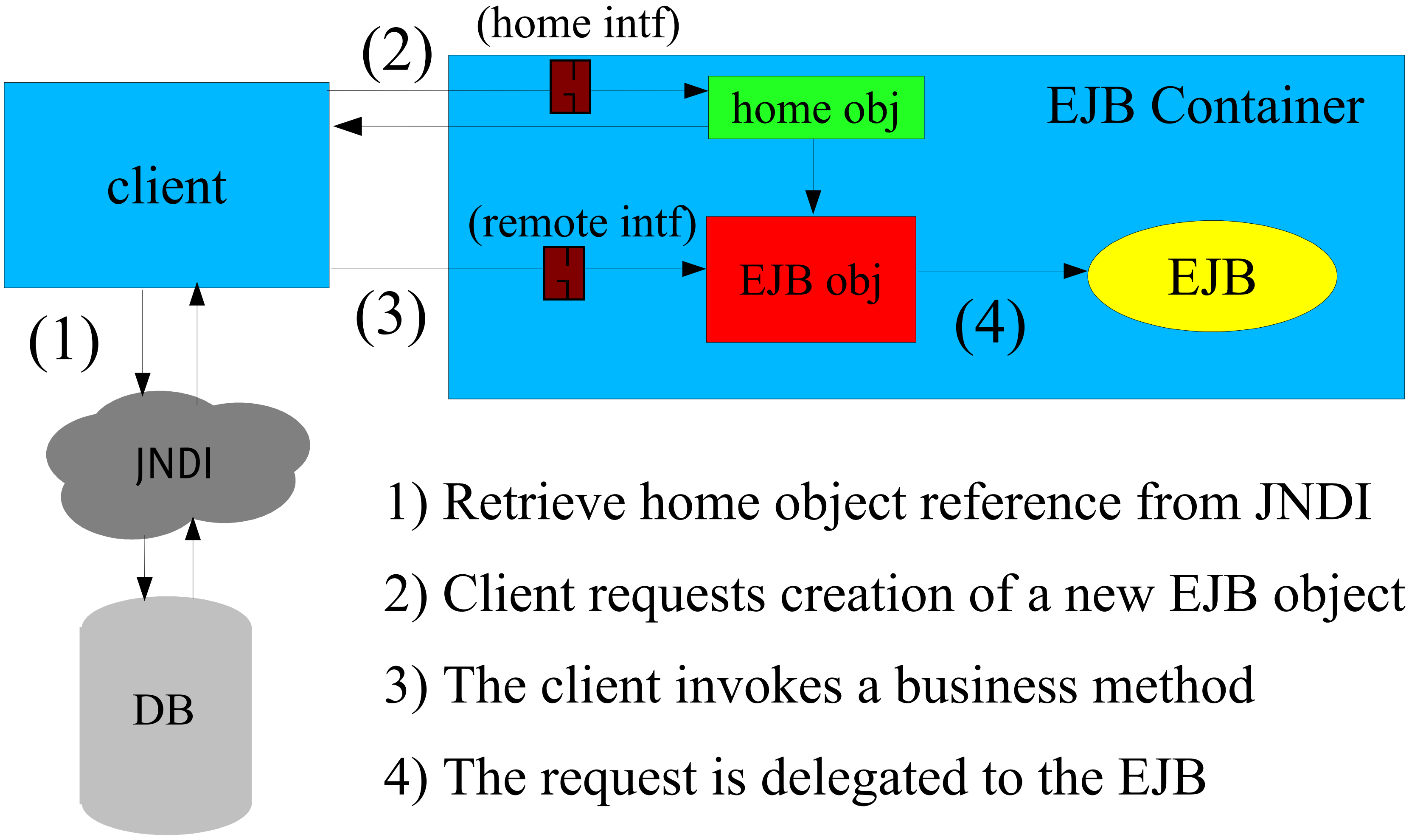
Component/Reference Name	JNDI Name
jdbc/AccountDB	jdbc/Oracle
ATM	atm
[illegible]	account

```
Contacted server...
Application BankApp transferred.
BankApp has 2 ejbs, 0 web components to deploy.
Deploying Ejbs...
Processing beans ...
Generating wrapper code for atm
Compiling wrapper classes for atm
Compiling RMI-IIOP classes for atm
Generating wrapper code for account
Compiling wrapper classes for account
Compiling RMI-IIOP classes for account
Making client JARs
Making server JARs
```

Create a Client

1. Use JNDI to lookup EJB's home interface.
2. Call home's create method to get the EJB™ remote object interface.
3. Call bean's business methods thru remote interface.

Accessing a bean



1) Retrieve home object reference from JNDI

2) Client requests creation of a new EJB object

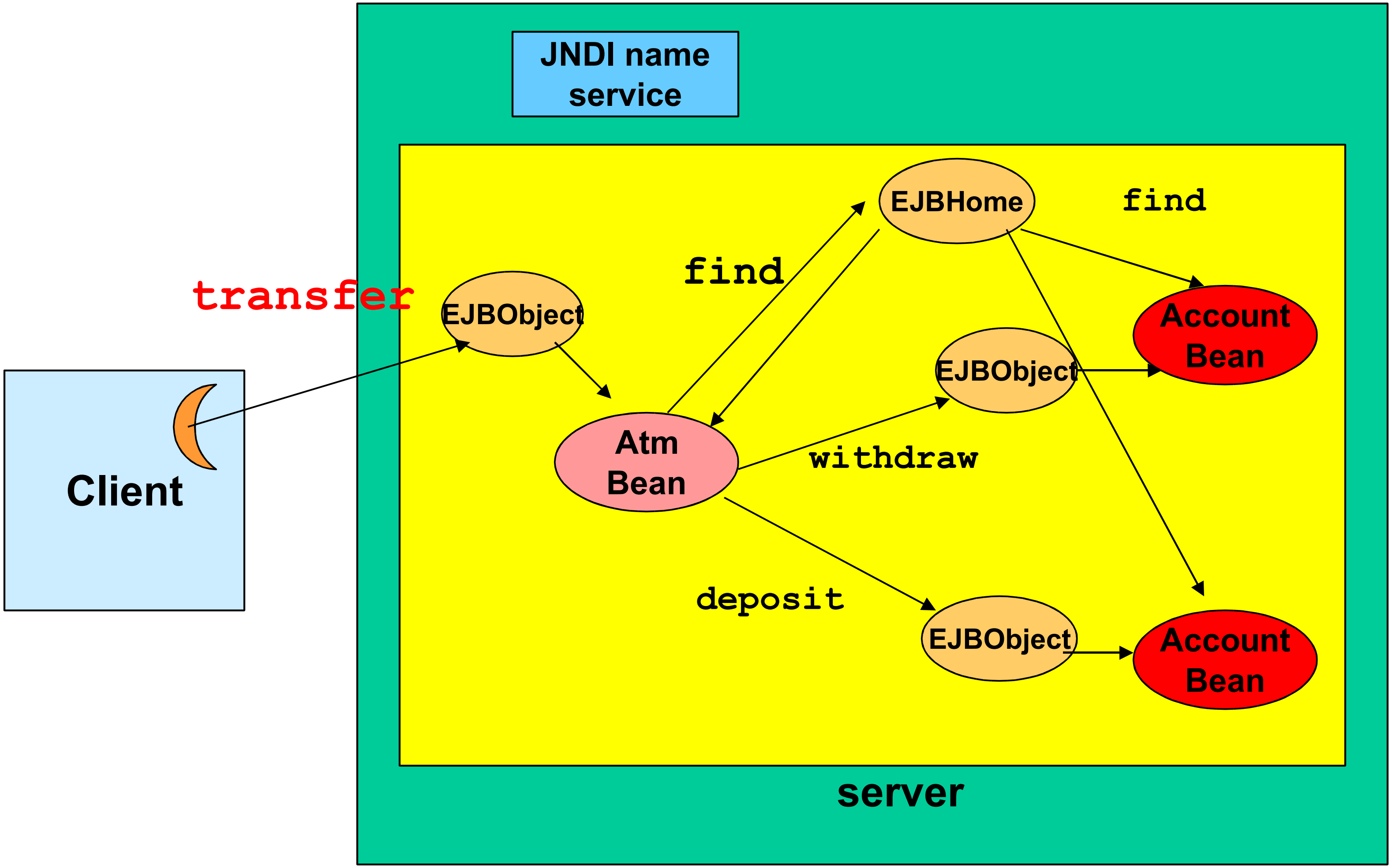
3) The client invokes a business method

4) The request is delegated to the EJB

ATM Client Code

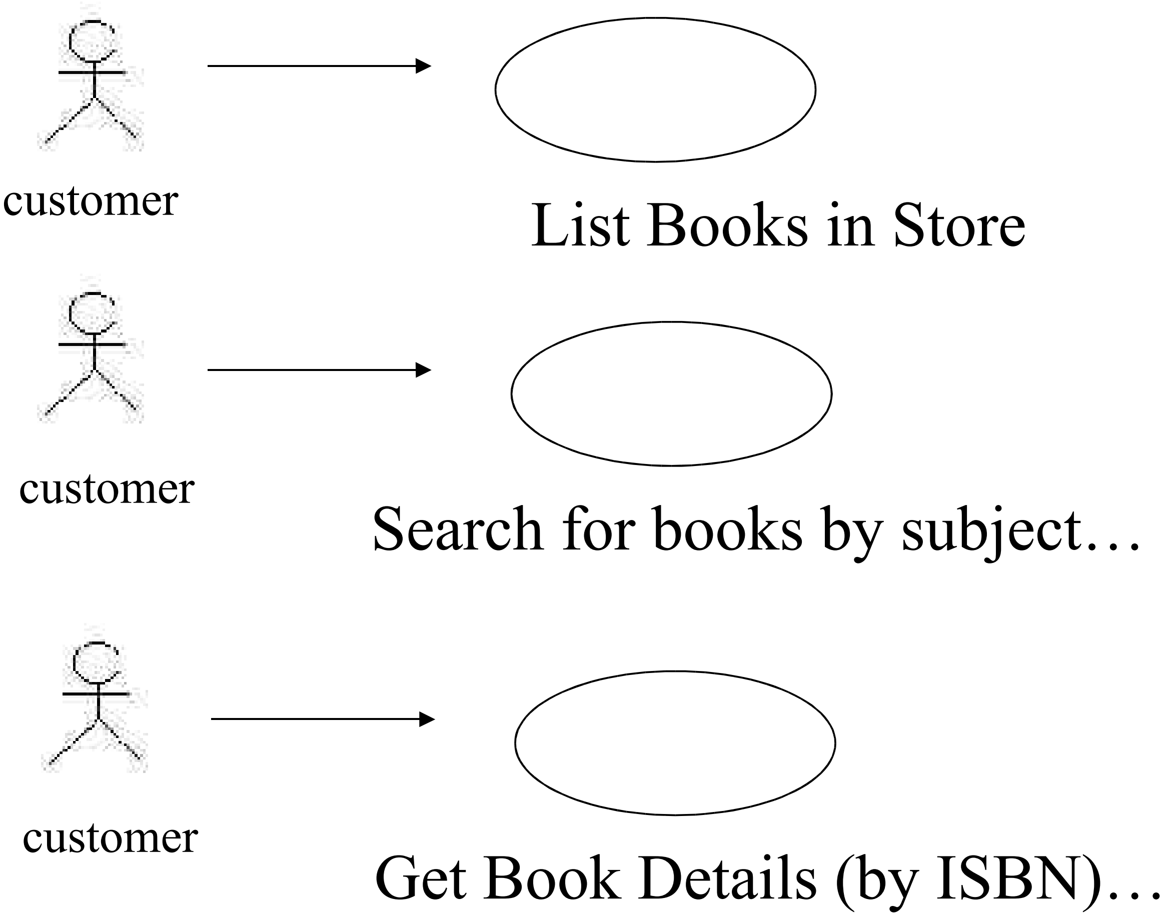
```
// create an initial context (starting point in name tree)
javax.naming.Context ic =new javax.naming.InitialContext();

// lookup jndi name (set by deployer in deployment
descriptor)
java.lang.Object objref = ic.lookup("Atm");

AtmHome home = (AtmHome)PortableRemoteObject.narrow(
        objref, AtmHome.class);

//call AtmHome Create method to get Atm interface
Atm atm = home.create();

// call Atm business methods
atm.transfer(41476633, 4443332121, 100000);
```

Session Bean Accessing Entity Bean



Exercise: Design and Implement Catalog Stateless Session Bean

Use Case Scenarios



customer

List Books in Store

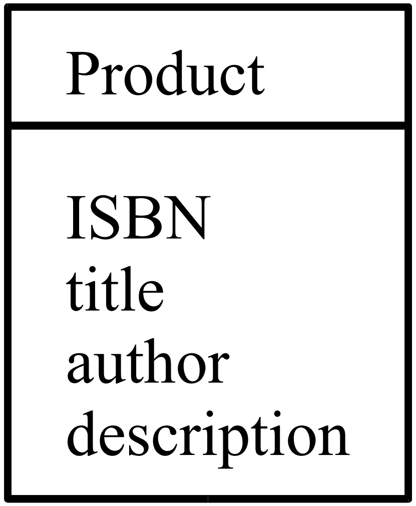
customer

Search for books by subject…

customer

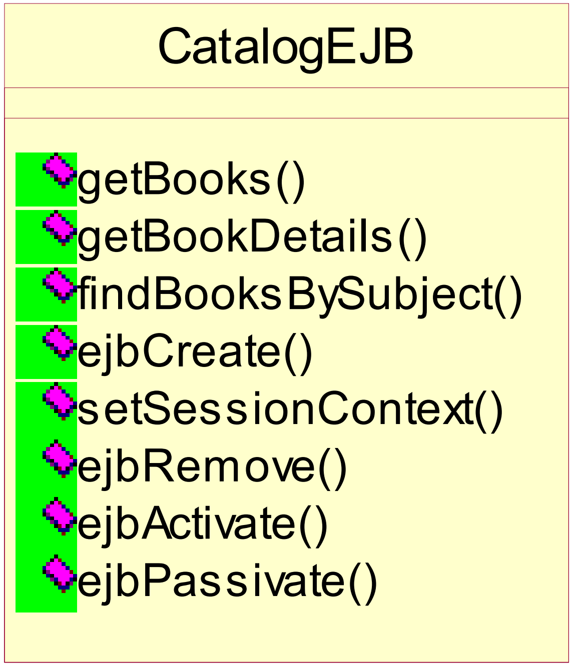
Get Book Details (by ISBN)…

Stateless Services: Catalog Session EJB



Product

ISBN
title
author
description

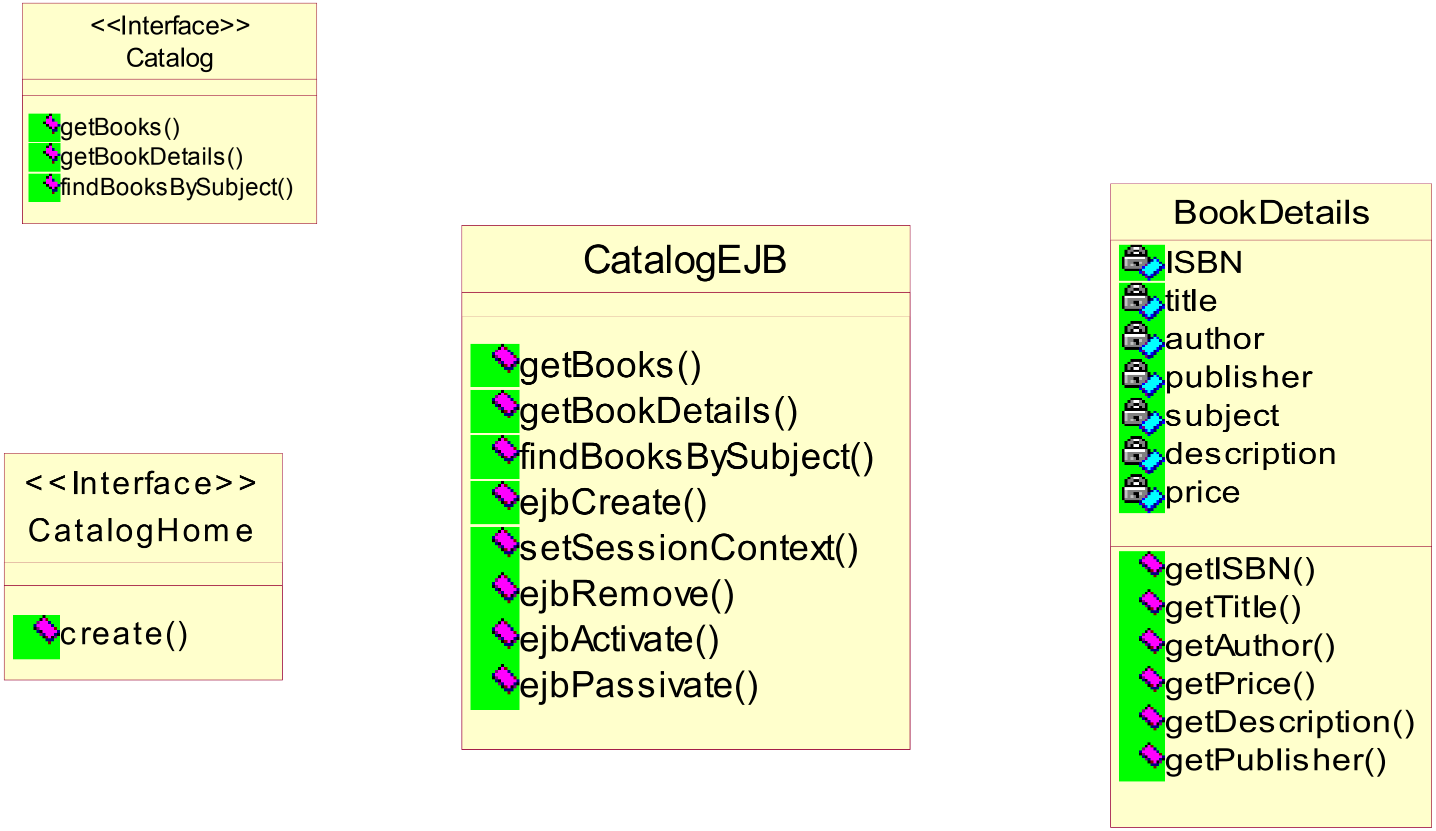


CatalogEJB

getBooks()
getBookDetails()
findBooksBySubject()
ejbCreate()
setSessionContext()
ejbRemove()
ejbActivate()
ejbPassivate()

A **Catalog** object represents different products and provides **browsing and searching services** to its clients. Both of the primary functions of the catalog, browsing and searching, are **generic services** which are **not tied to any particular client**. Also, the catalog object **reads multiple rows in the database at the same time** and **provides a shared view** of the data.

CatalogEJB



<<Interface>>
Catalog

getBooks()
getBookDetails()
findBooksBySubject()

<<Interface>>
CatalogHome

create()

CatalogEJB

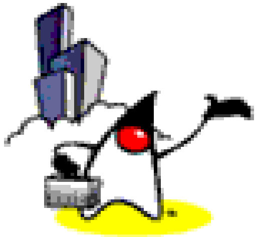
getBooks()
getBookDetails()
findBooksBySubject()
ejbCreate()
setSessionContext()
ejbRemove()
ejbActivate()
ejbPassivate()

BookDetails

ISBN
title
author
publisher
subject
description
price

getISBN()
getTitle()
getAuthor()
getPrice()
getDescription()
getPublisher()

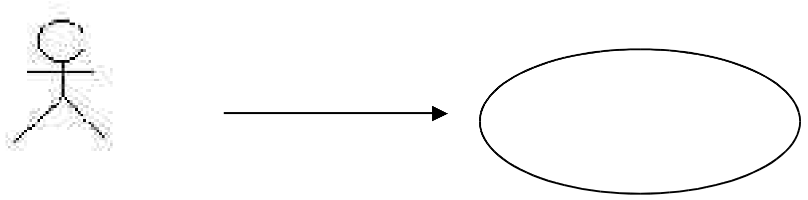
Example Stateful Session Bean



Implementing a Stateful Session Bean

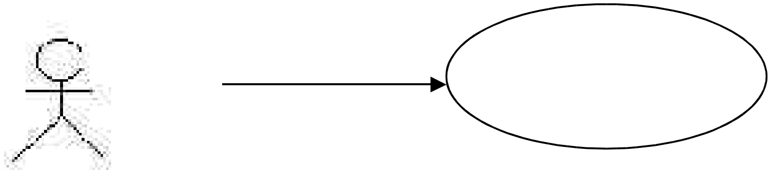
- A typical example of a Stateful session bean is an online shopping cart

Example Scenario: Use Cases



customer

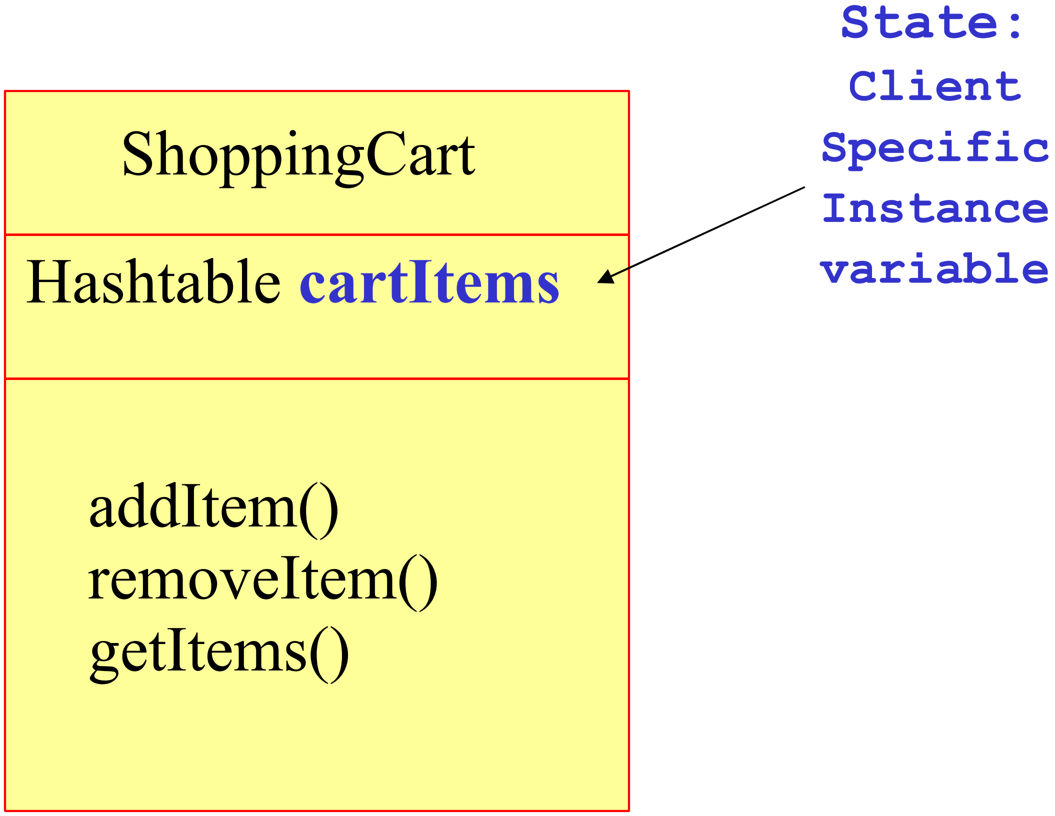
add/remove item to/from shopping cart



customer

get items in shopping cart

Example: Shopping Cart Class



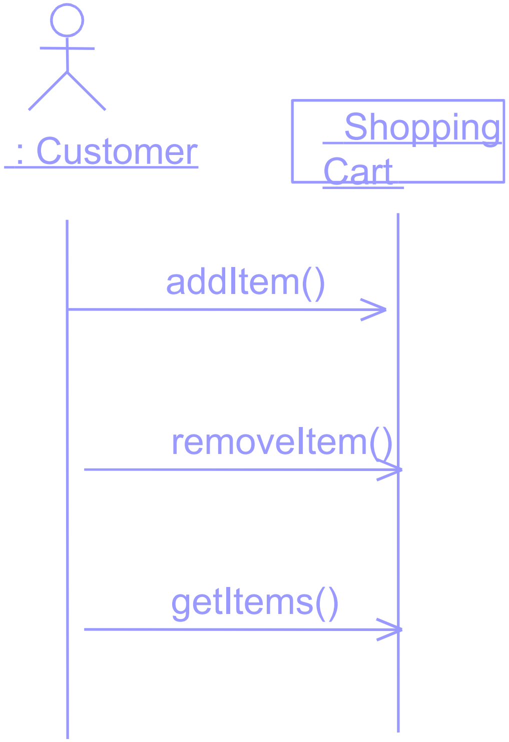
ShoppingCart

Hashtable **cartItems**

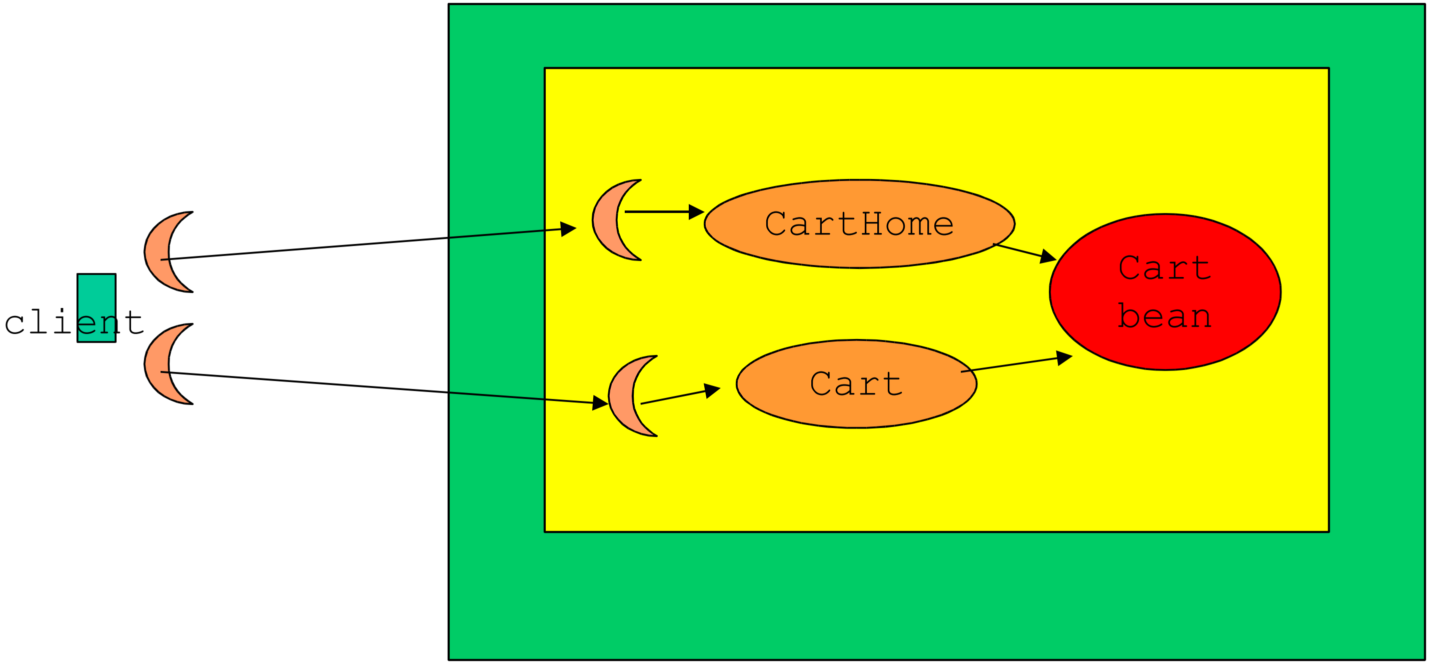
addItem()
removeItem()
getItems()

State: Client Specific Instance variable

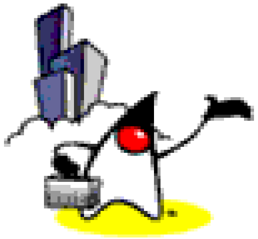
Example Scenario: Sequence Diagram



Example Continued



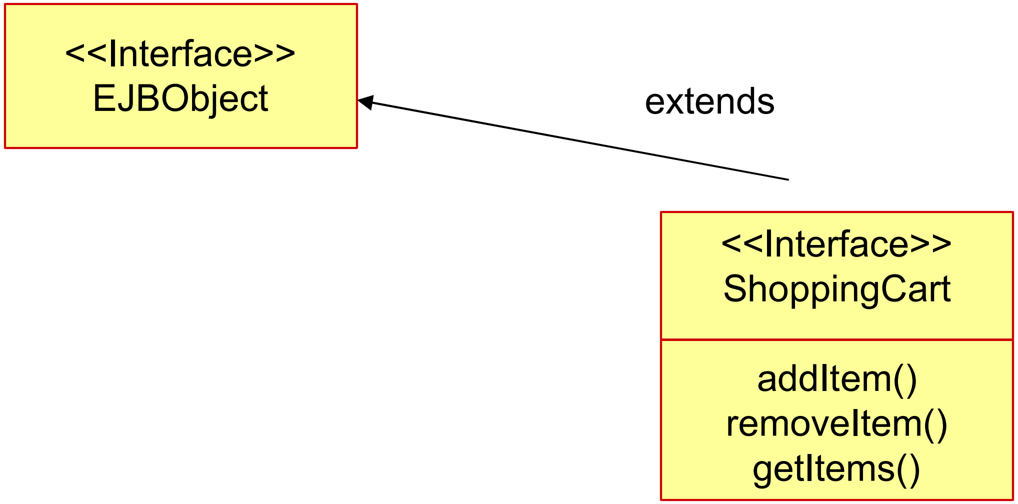
Steps for Creating Stateful Session Bean



Stateful Session Bean Implementation

1. Define the session bean’s remote interface (Cart)
2. Define a home interface (CartHome) for the session bean
3. Write the business logic in the session bean class (CartBean)
4. Compile the remote interface, home interface, and implementation class
5. Define a deployment descriptor specifying any declarative metadata
6. Package in an ejb-jar file
7. Deploy the enterprise application

1) Create the Remote Interface

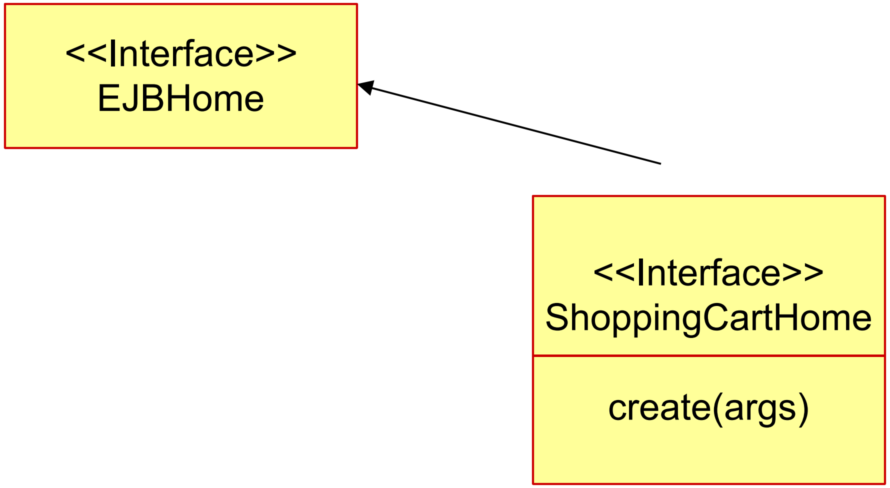


```
public interface  ShoppingCart  extends javax.ejb.  EJBObject  {

  public void  addItem (String itemId, int qty)   throws  java.rmi.RemoteException;
  public void  removeItem (String itemId, int qty)
               throws java.rmi.RemoteException;
  public Hashtable  getItems () throws         java.rmi.RemoteException;

}
```

Stateful Session Bean Implementation

1. Define the session bean’s remote interface (Cart)
2. Define a home interface (CartHome) for the session bean
3. Write the business logic in the session bean class (CartBean)
4. Compile the remote interface, home interface, and implementation class
5. Define a deployment descriptor specifying any declarative metadata
6. Package in an ejb-jar file
7. Deploy the enterprise application

2) Create the Home Interface

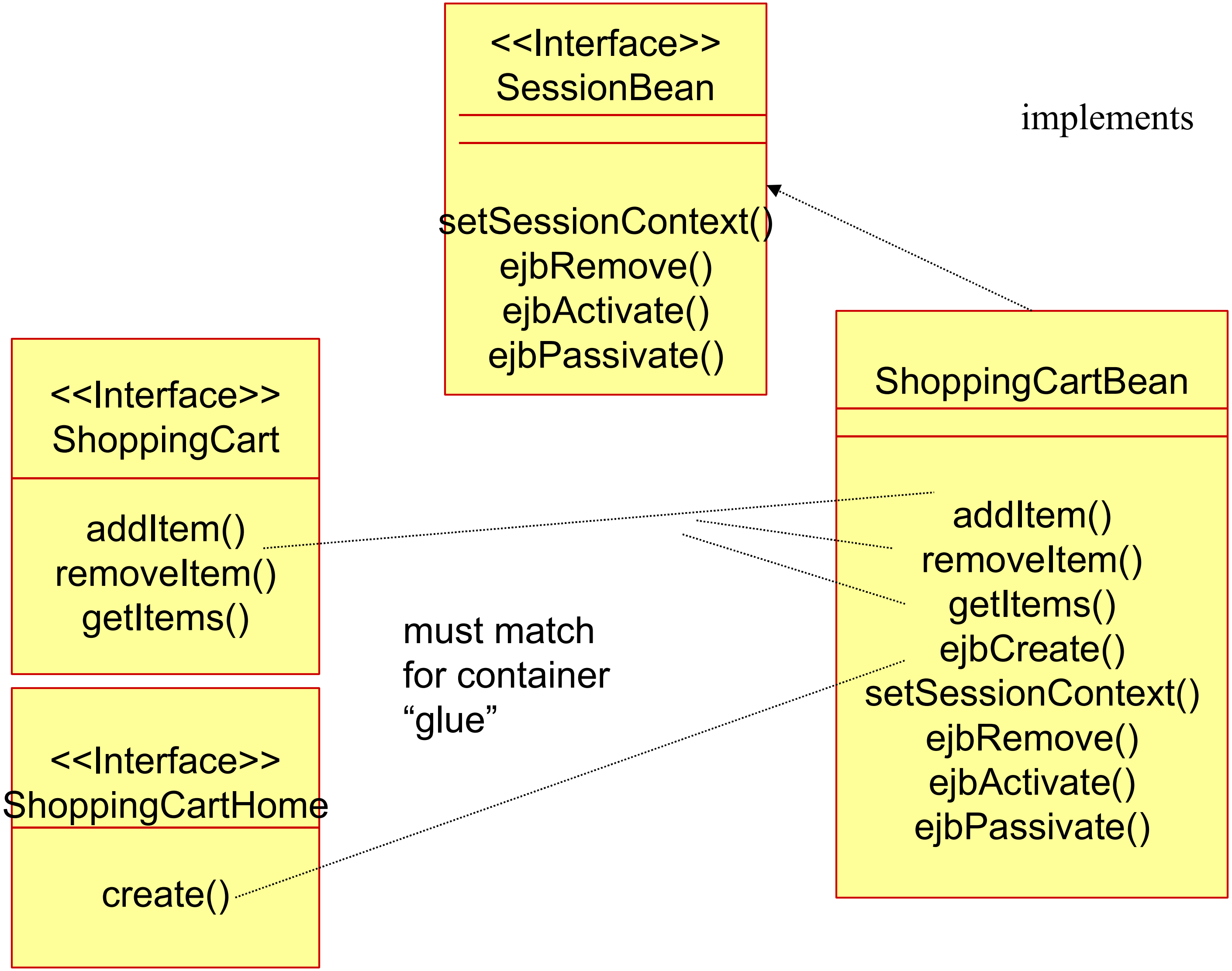


```
public interface  ShoppingCartHome  extends javax.ejb.  EJBHome  {

  Cart  create (String custId)   throws java.rmi.RemoteException,
             javax.ejb.CreateException;


}
```

Stateful Session Bean Implementation

1. Define the session bean’s remote interface (Cart)
2. Define a home interface (CartHome) for the session bean
3. Write the business logic in the session bean class (CartBean)
4. Compile the remote interface, home interface, and implementation class
5. Define a deployment descriptor specifying any declarative metadata
6. Package in an ejb-jar file
7. Deploy the enterprise application

3) ShoppingCartBean Implementation



3) ShoppingCartBean: Implement Shopping Cart Interface Business Methods

```
public class  ShoppingCartBean  implements  SessionBean  {

 // instance variables
 private Hashtable cart=null;
 int  customerId;
 private SessionContext sessionContext=null;

 // implement ShoppingCart interface business methods
 public void addItem  (String itemId,int qty) {
   cart.put(itemId,new Integer(qty));
 }
 public void removeItem (String itemId, int qty){
     cart.remove(itemId);
 }
 public Hashtable  getItems () {
    return cart;
 }
```

3) ShoppingCartBean: Implement Home Interface Create Method

```
//create/store Client specific state (instance variables

public void ejbCreate(int customerId) {
        this.cart = new Hashtable();
        this.customerId= customerId;
}
```

3) ShoppingCartBean: Implement Session Interface Container Callback Methods

```
//associate a session bean instance with its context maintained by
the container.
public void setSessionContext  (SessionContext sc){
   this.sessionContext=sc;
}



// signals the instance it has just been reactivated.
// open any needed resources
public void ejbActivate () {}

// signals the intent of the container to passivate the instance
// close any open resources
public void ejbPassivate () {}
```

3) ShoppingCartBean: Implement Session Interface Container Callback Methods

```
// signals that the instance is in the process of being removed
by // the container, release resources
public void ejbRemove () {
            this.ca rt = null;
      this.customerId= null;
}
```

Stateful Session Bean Implementation

1. Define the session bean's remote interface (Cart)
2. Define a home interface (CartHome) for the session bean
3. Write the business logic in the session bean class (CartBean)
4. Compile the remote interface, home interface, and implementation class
5. Define a deployment descriptor specifying any declarative metadata
6. Package in an ejb-jar file
7. Deploy the enterprise application

4) Compile the Remote & Home Interfaces and Implementation Class.

```
javac –classpath $J2EE_HOME/lib/j2ee.jar
        ShoppingCart.java
        ShoppingCartHome.java
        ShoppingCartEJB.java
```

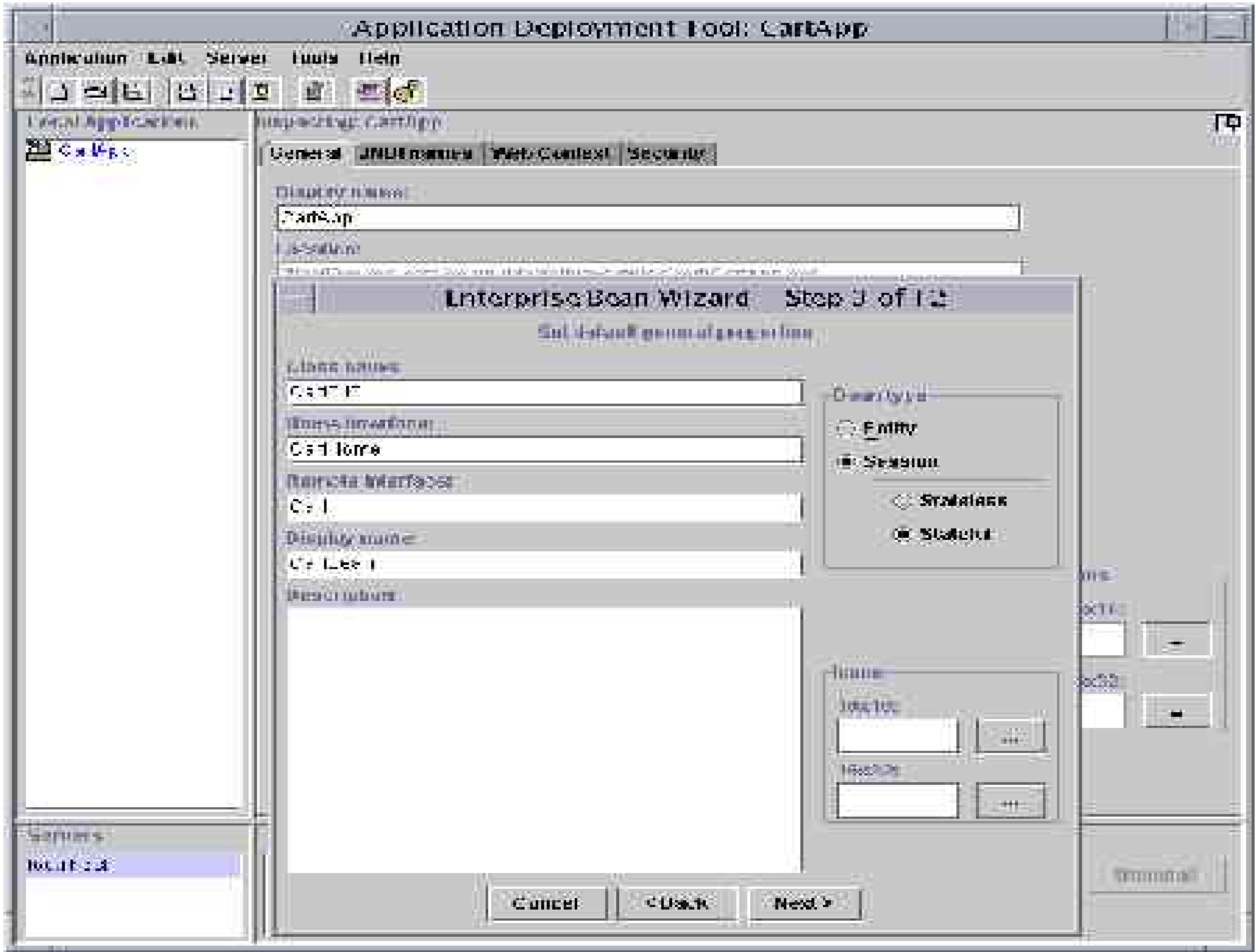
Stateful Session Bean Implementation

1. Define the session bean’s remote interface (Cart)
2. Define a home interface (CartHome) for the session bean
3. Write the business logic in the session bean class (CartBean)
4. Compile the remote interface, home interface, and implementation class
5. Define a deployment descriptor specifying any declarative metadata
6. Package in an ejb-jar file
7. Deploy the enterprise application

5) Create Deployment Descriptor

deploytool

```
Set the values for the class names, transaction
attributes,
Environment values, resource references…
```



Application Deployment Tool: CartApp

Enterprise Bean Wizard Step 3 of 12

Set default general properties

Class name: CartEJB

Home interface: CartHome

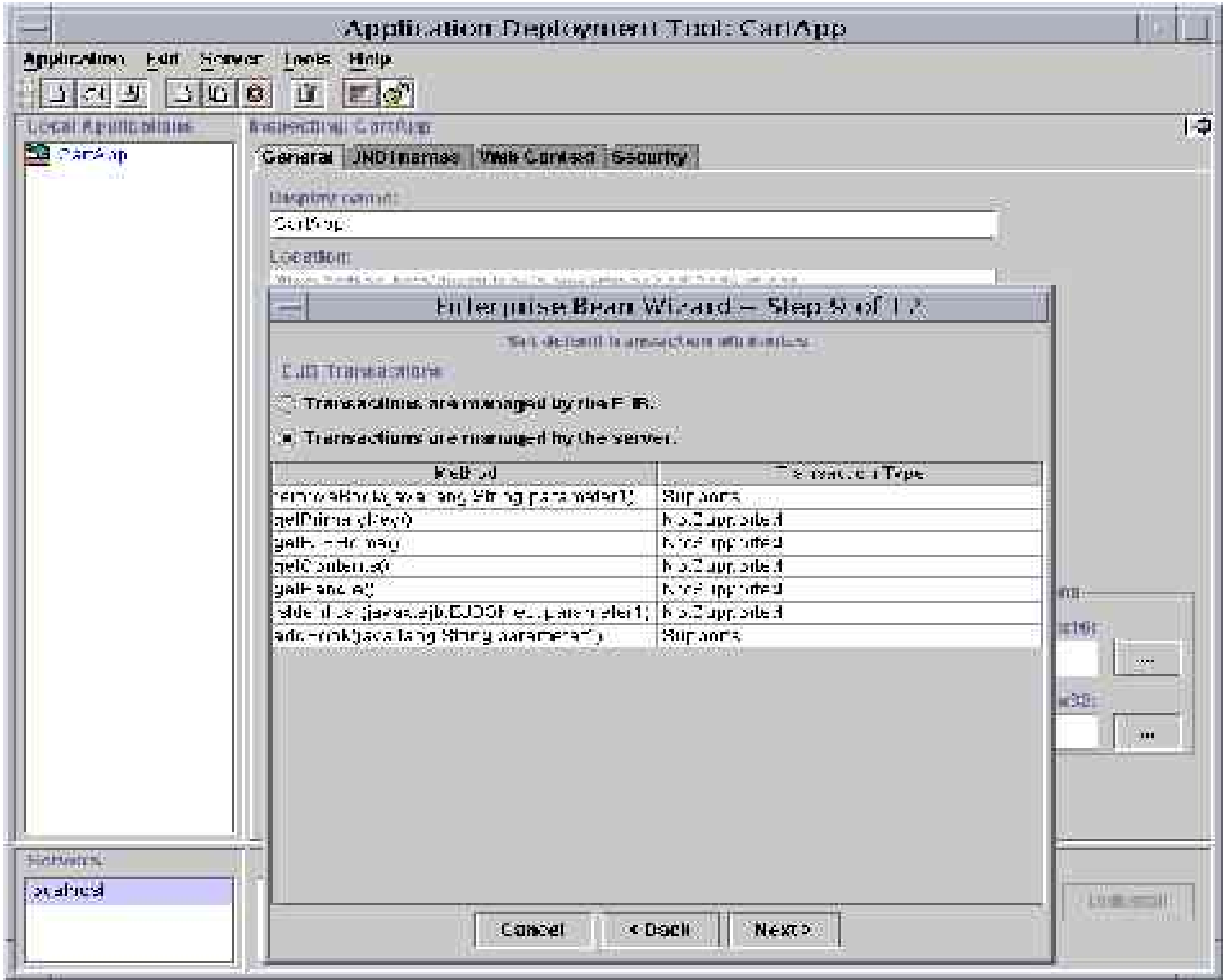
Remote interface: Cart

Display name: CartEJB

Description:

Bean type: ○ Entity ● Session

○ Stateless ● Stateful



Display name:

CartApp

Location:

Enterprise Bean Wizard — Step 9 of 12

Set default transaction attributes

EJB Transactions

○ Transactions are managed by the EJB.

● Transactions are managed by the server.

Method	Transaction Type
removeBook(java.lang.String parameter1)	Supports
getPrimaryKey()	NotSupported
getEJBHome()	NotSupported
getContents()	NotSupported
getHandle()	NotSupported
isIdentical(javax.ejb.EJBObject parameter1)	NotSupported
addBook(java.lang.String parameter1)	Supports

5) Create Deployment Descriptor

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems Inc.//DTD
Enterprise JavaBeans 1.2//EN' 'http://java.sun.com/j2ee/dtds/ejb-
jar_1_2.dtd'>
<ejb-jar>
  <description>no description</description>
  <display-name>CartEjb</display-name>
  <enterprise-beans>
    <session>
      <description>no description</description>
      <display-name>CartBean</display-name>
      <ejb-name>CartBean</ejb-name>
      <home>CartHome</home>
      <remote>Cart</remote>
      <ejb-class>CartEJB</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
```

5) Create DD Cont.

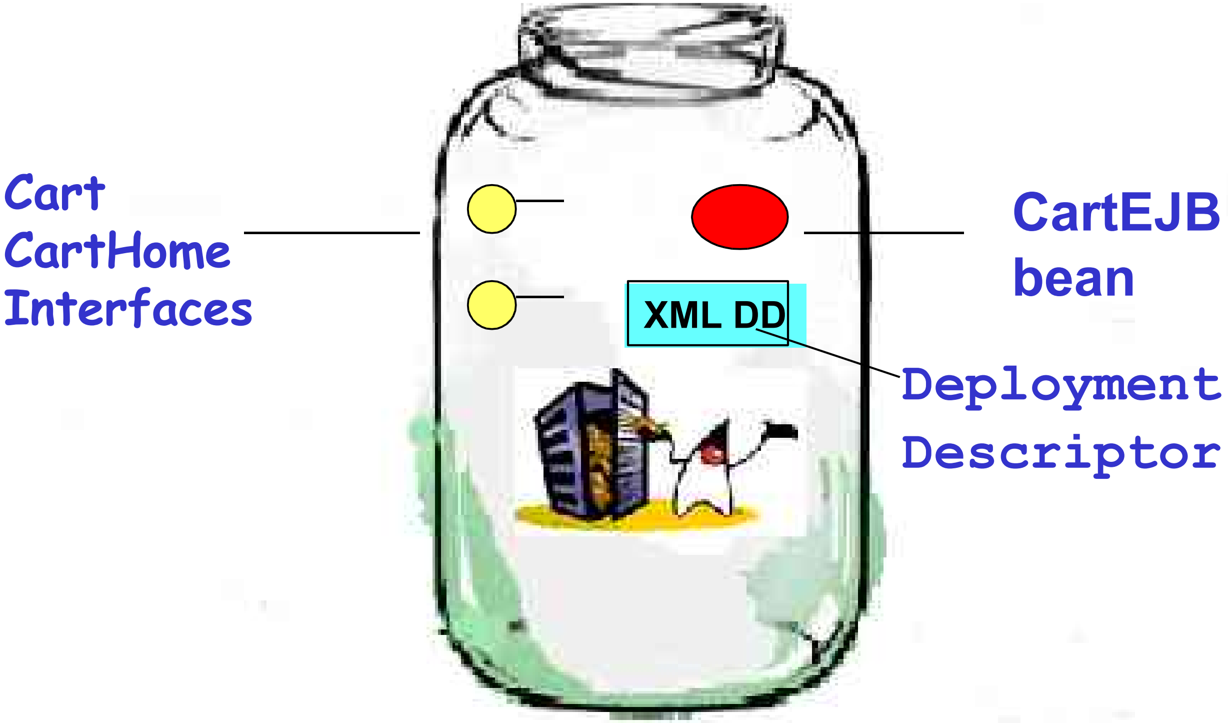
```
 <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>CartBean</ejb-name>
        <method-intf>Remote</method-intf>
        <method-name>removeBook</method-name>
        <method-param>java.lang.String</method-param>
      </method>
      <trans-attribute>Supports</trans-attribute>
    </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

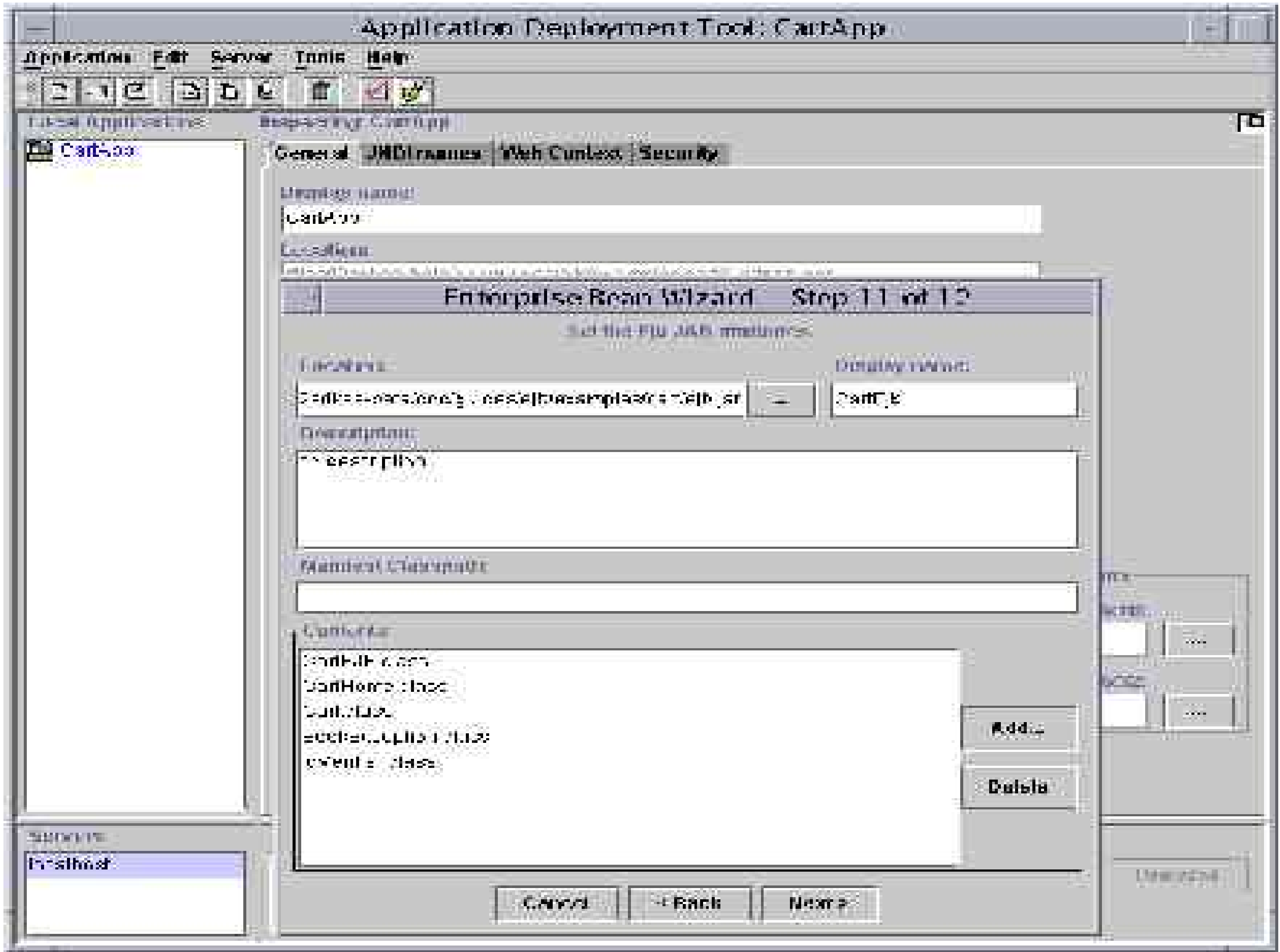
Stateful Session Bean Implementation

1. Define the session bean’s remote interface (Cart)
2. Define a home interface (CartHome) for the session bean
3. Write the business logic in the session bean class (CartBean)
4. Compile the remote interface, home interface, and implementation class
5. Define a deployment descriptor specifying any declarative metadata
6. Package in an ejb-jar file
7. Deploy the enterprise application

6) Package in an ejb-jar File.

packager –ejbJar Cart.class CartEJB.class CartHome.class ejb-jar.xml





Application Deployment Tool: CartApp

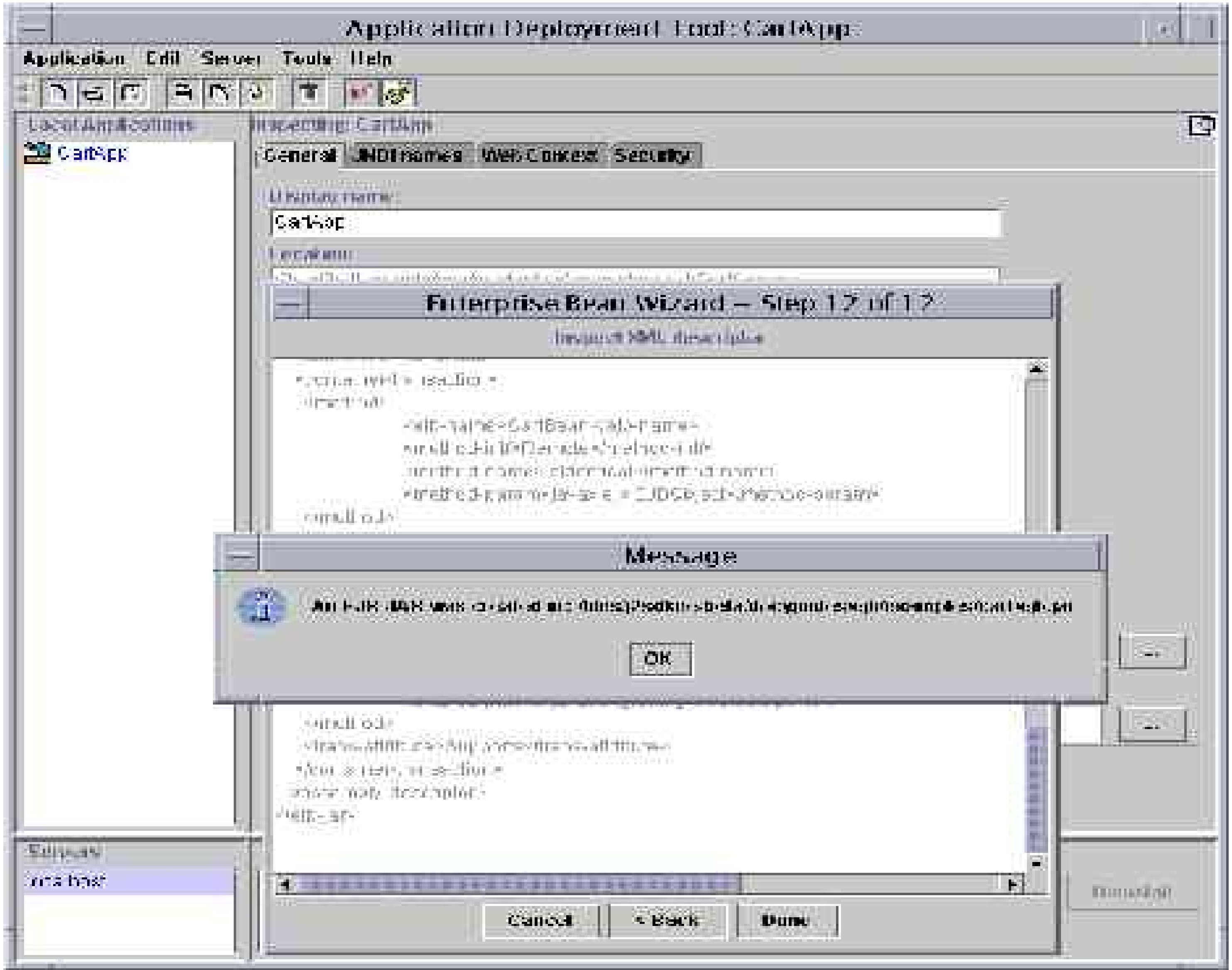
Enterprise Bean Wizard - Step 11 of 12

Set the EJB JAR Properties

Display name: CartEJB

Description: no description

Contents:
CartEJB.class
CartHome.class
Cart.class
BookException.class
IdVerifier.class



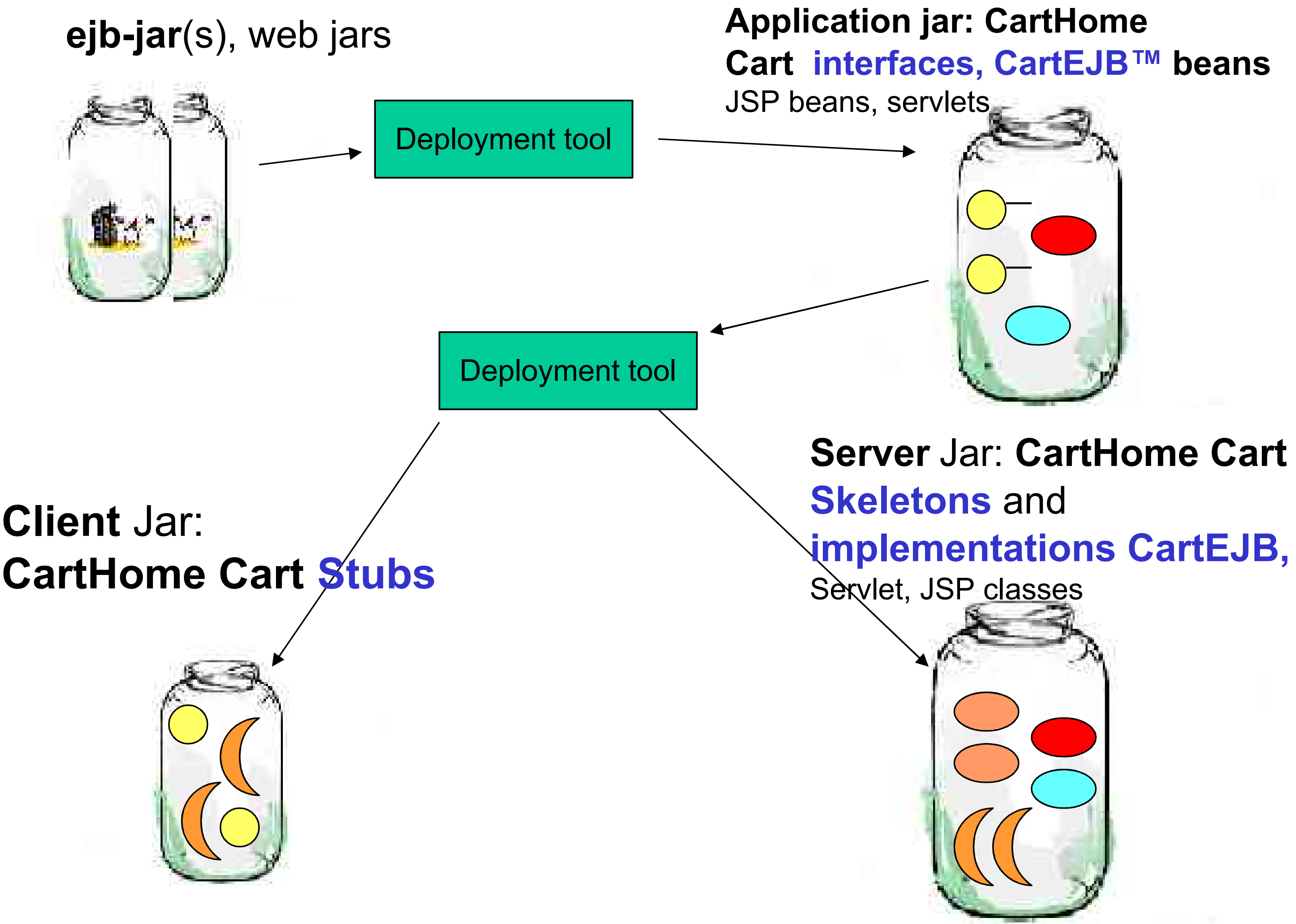
```
<container-transaction>
  <method>
      <ejb-name>CartBean</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>[illegible]</method-name>
      <method-param>[illegible]</method-param>
  </method>
...
  </method>
  <trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

An EJB JAR was created at: [illegible]

Session Bean Implementation

1. Define the session bean's remote interface (cart).
2. Define a home interface (CartHome) for the session bean.
3. Write the business logic in the session bean class (CartBean).
4. Compile the remote interface, home interface, and implementation class.
5. Define a deployment descriptor specifying any declarative metadata.
6. Package in an ejb-jar file.
7. Deploy the enterprise application.

7) Deploy the Enterprise Application



ejb-jar(s), web jars

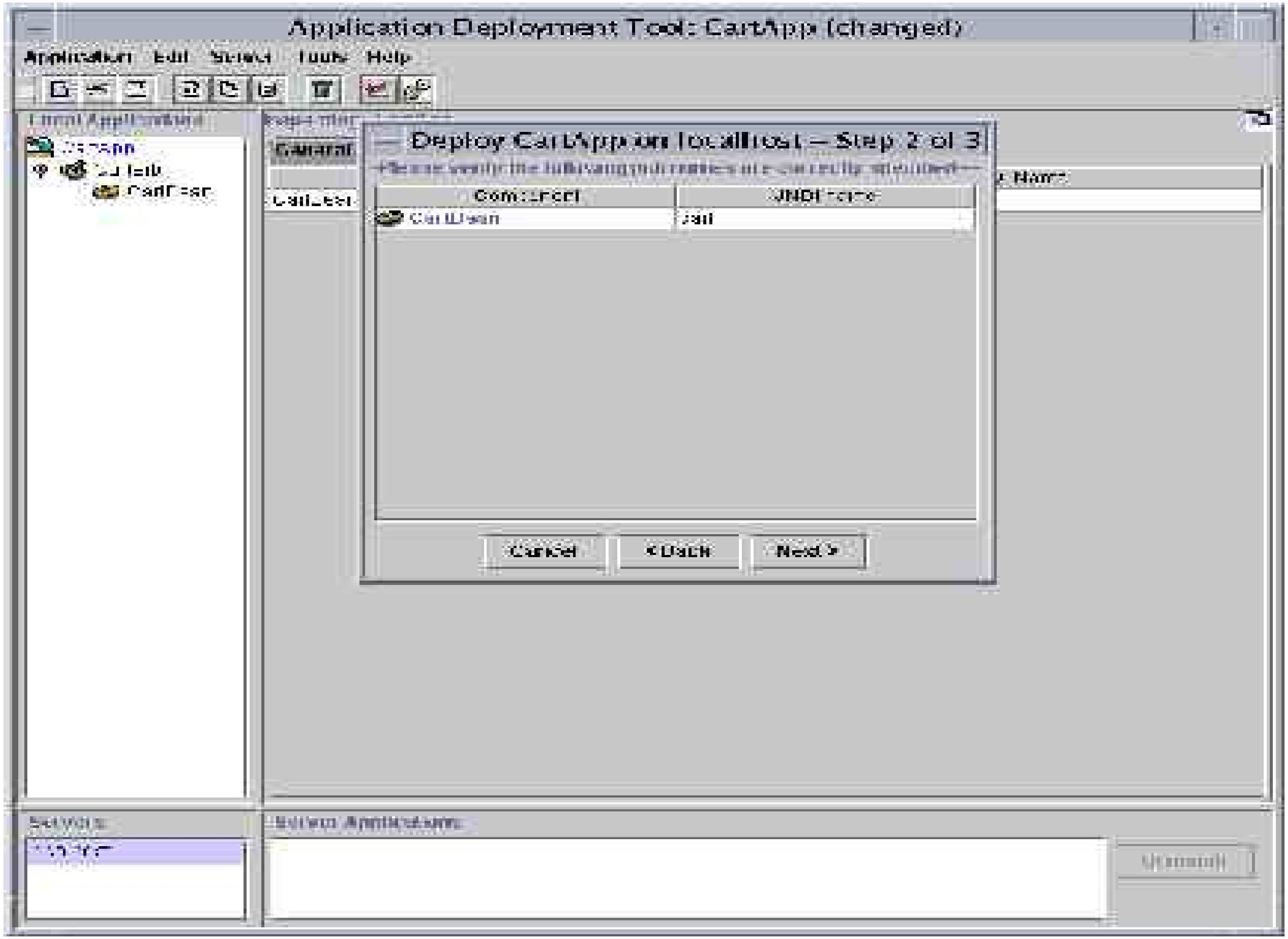
Deployment tool

Application jar: CartHome Cart interfaces, CartEJB™ beans
JSP beans, servlets

Deployment tool

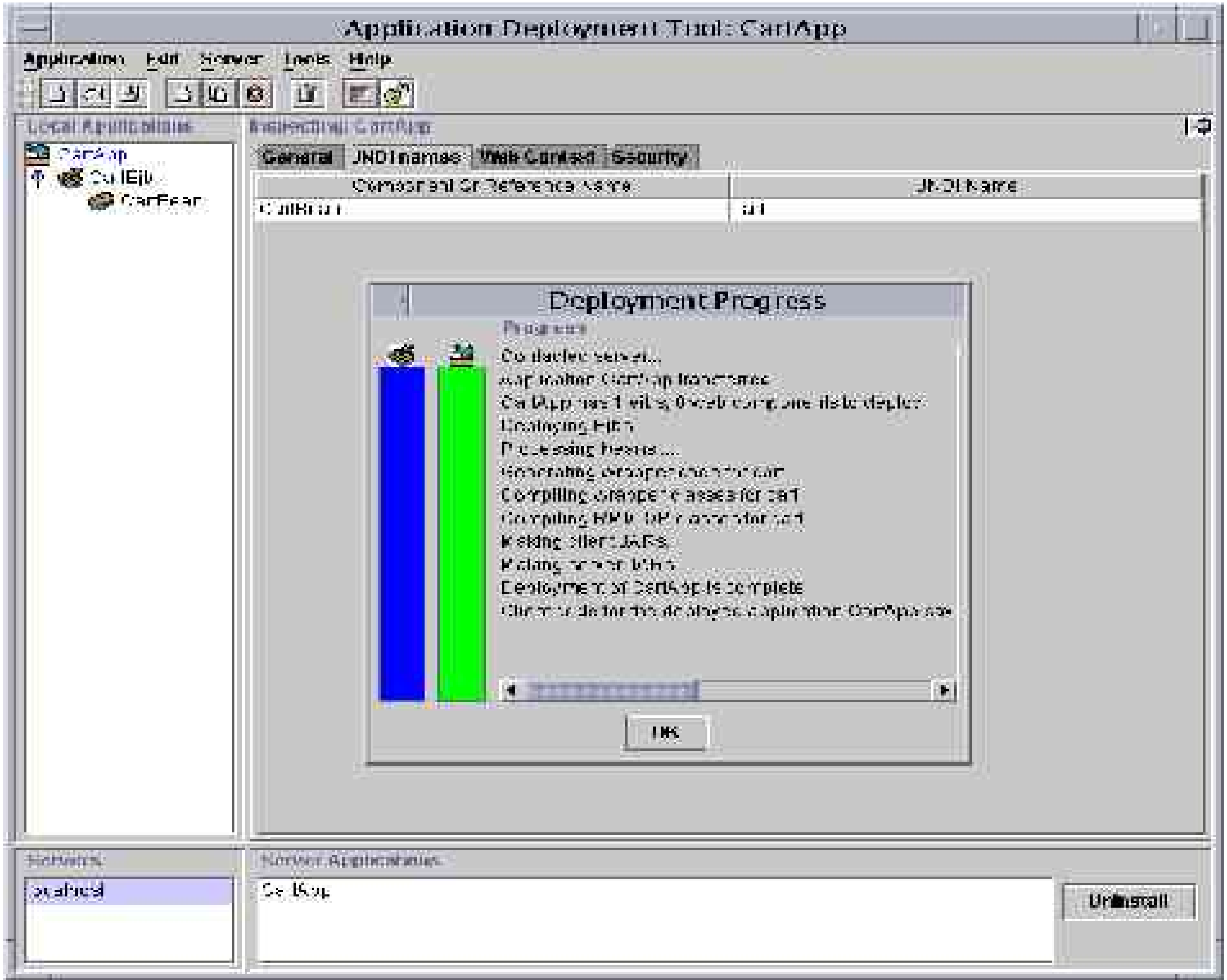
Client Jar:
CartHome Cart Stubs

Server Jar: **CartHome Cart Skeletons** and **implementations CartEJB,**
Servlet, JSP classes



Please verify the following JNDI names are correctly specified

Component	JNDI Name
CartBean	Cart



Component or Reference Name	JNDI Name
CartBean	[illegible]

```
Contacted server...
Application CartApp transferred.
CartApp has 1 ejbs, 0 web components to deploy.
Deploying Ejbs...
Processing beans ...
Generating wrapper code for cart
Compiling wrapper code for cart
Compiling RMI-IIOP code for cart
Making client JARS
Making server JARS
Deployment of CartApp is complete
Client code for the deployed application CartApp save
```

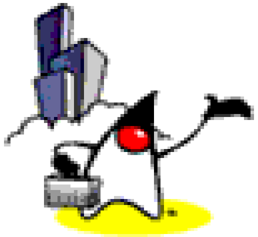
Create a Client

1. Use JNDI to lookup CartHome.
2. Call CartHome's create methods to get the Cart interface.
3. Call business methods thru Cart remote interface.

Cart Client Code

```
//Create an InitialContext (starting point)
javax.naming.Context ic = new javax.naming.InitialContext();

// lookup the EJBHome interface using the JNDI name
// given in deployment descriptor
String jndiName = “ShoppingCart”;
java.lang.Object objref = ic.lookup(jndiName);

CartHome cartHome = (CartHome)
PortableRemoteObject.narrow(objref,CartHome.class);

Cart cart = cartHome.create(41476633);

// call business methods
cart.addItem( 111222, 1);
```

Resources



Resources

- Applied Enterprise JavaBeans Technology written by Kevin Boone (Sun Microsystems, Inc.), published by Prentice Hall [1]
- J2EE Tutorial in java.sun.com [2]

Live your life with Passion!