# Enterprise JavaBeans™ Programming

SL-351

# About This Course

# Course Goal

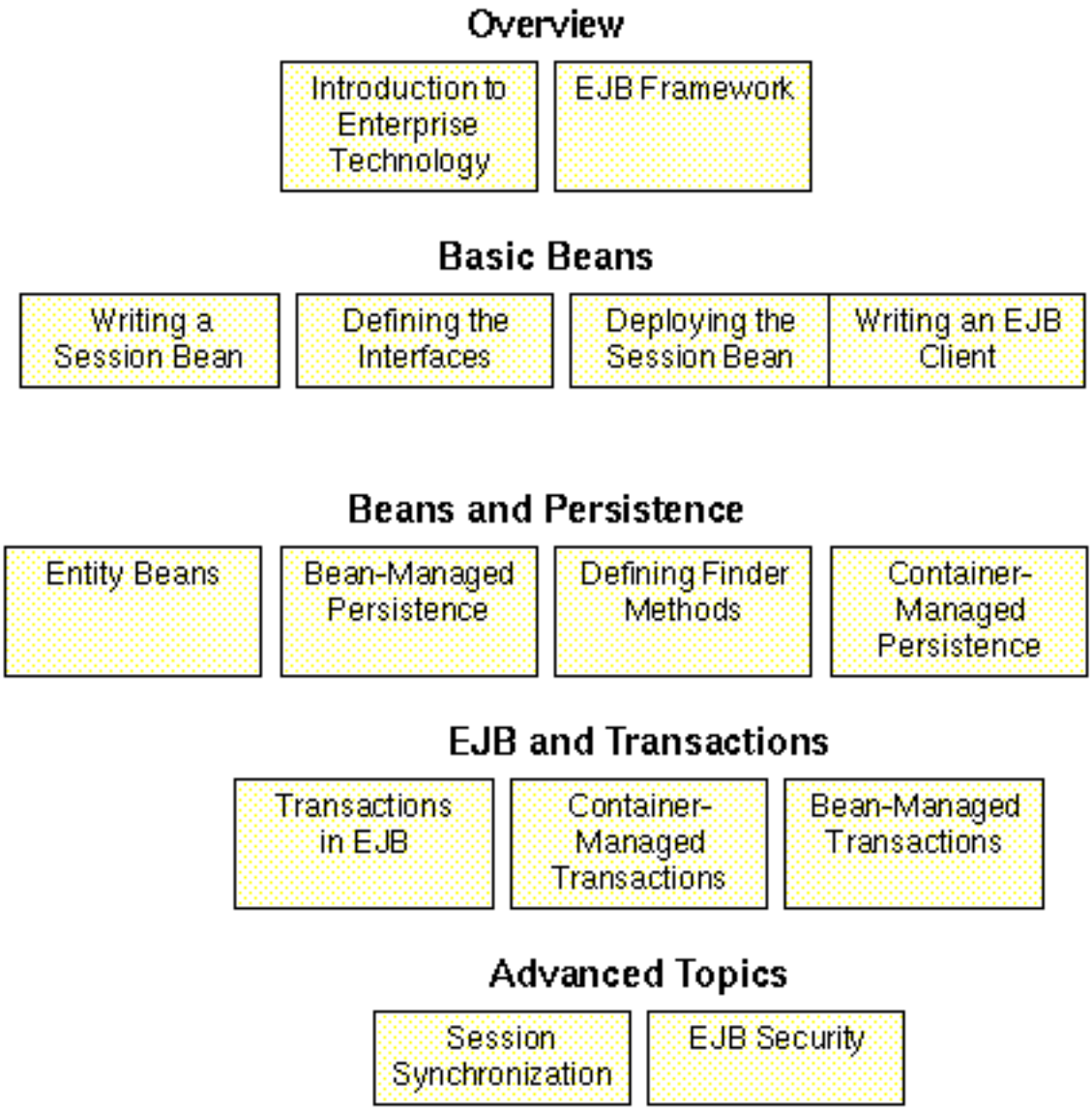
This course provides you with knowledge and skills to:

- Understand the scope of the Java™ 2 Platform, Enterprise Edition (J2EE™)
- Develop an enterprise application using Enterprise JavaBeans™ (EJB™) technology
- Create interoperable components that are used with scalable and extensible application servers

# Course Overview

- J2EE
- Writing beans: classes, methods, interfaces
- Security
- Transactions
- Object-relational mapping
- Persistence

# Course Map



# Module-by-Module Overview

- Module 1 – "Introduction to Enterprise Technology"
- Module 2 – "EJB Framework"
- Module 3 – "Writing a Session Bean"
- Module 4 – "Defining the Interfaces"
- Module 5 – "Deploying a Session Bean"
- Module 6 – "Writing an EJB Client"
- Module 7 – "Entity Beans"

# Module-by-Module Overview

- Module 8 – "Bean-Managed Persistence"
- Module 9 – "Defining Finder Methods"
- Module 10 – "Container-Managed Persistence"
- Module 11 – "Transactions in EJB"
- Module 12 – "Container-Managed Transactions"
- Module 13 – "Bean-Managed Transactions"
- Module 14– "Session Synchronization"
- Module 15 – "EJB Security"

# Course Objectives

- Describe the architecture for developing an enterprise application that adheres to the EJB 1.1 specifications
- Describe EJB architecture
- Describe how to access an EJB component
- Describe how to manage security
- Design a session bean and an entity bean
- Create a sound architecture for an EJB application

# Guidelines for Module Pacing

| Module | Day 1 | Day 2 | Day 3 | Day 4 | Day 5 |
|---|---|---|---|---|---|
| "About This Course" | X | | | | |
| Module 1 – "Introduction to Enterprise Technology" | X | | | | |
| Module 2 – "EJB Framework" | X | | | | |
| Module 3 – "Writing a Session Bean" | X | | | | |
| Module 4 – "Defining the Interfaces" | X | | | | |
| Module 5 – "Deploying a Session Bean" | X | X | | | |
| Module 6 – "Writing an EJB Client" | | X | | | |
| Module 7 – "Entity Beans" | | | X | | |
| Module 8 – "Bean-Managed Persistence" | | | | X | |
| Module 9 – "Defining Finder Methods" | | | | X | |
| Module 10 – "Container-Managed Persistence" | | | | X | |
| Module 11 – "Transactions in EJB" | | | | X | |
| Module 12 – "Container-Managed Transactions" | | | | | X |
| Module 13 – "Bean-Managed Transactions" | | | | | X |
| Module 14 – "Session Synchronization" | | | | | X |
| Module 15 – "EJB Security" | | | | | X |

# Topics Not Covered

- Programming using Java technology – Covered in SL-275: *Java Programming Language*
- Object-oriented theory – Covered in OO-226: *Object-Oriented Application Analysis and Design Using UML*

# How Prepared Are You?

- Experienced with the Java programming language
- Experienced with object-oriented design and analysis
- Familiar with distributed programming (multi-tier architecture)
- Familiar with relational or object database programming
- Familiar with transactions
- Able to create multi-tier Java application solutions
- Familiar with component technology

# Introductions

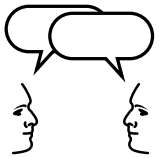
- Name
- Company affiliation
- Title, function, and job responsibility
- Java programming experience
- Enterprise computing experience
- Reasons for enrolling in this course
- Expectations for this course

# How to Use Course Materials

- Objectives
- Relevance
- Overhead image
- Lecture
- Exercise
- Check Your Progress
- Think Beyond

# Icons

Additional resources



Discussion

Exercise objective

# Typographical Conventions

- `Courier` is used for the names of command, files, and directories, as well as on-screen computer output.
- **`Courier bold`** is used for characters and numbers that you type.
- *`Courier italic`* is used for variables and command-line placeholders that are replaced with a real name or value.
- *Palatino italics* is used for book titles, new words or terms, or words that are emphasized.

# Typographical Conventions

- `Courier` is used for the class names, methods, and keywords.
- Methods are not followed by parentheses unless a formal or actual parameter list is shown.
- Line breaks occur where there are separations, conjunctions, or white space in the code.

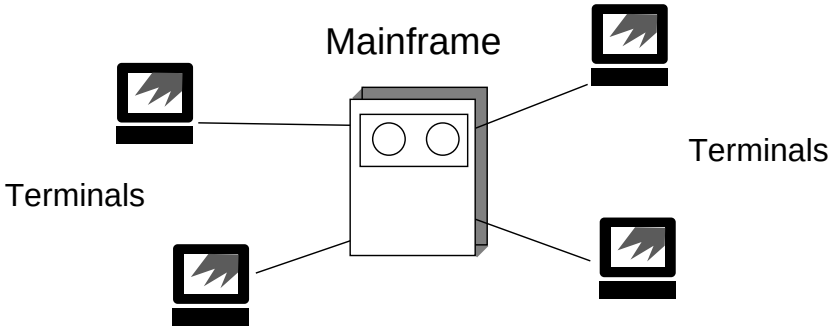
# Module 1

# Introduction to Enterprise JavaBeans™ Technology

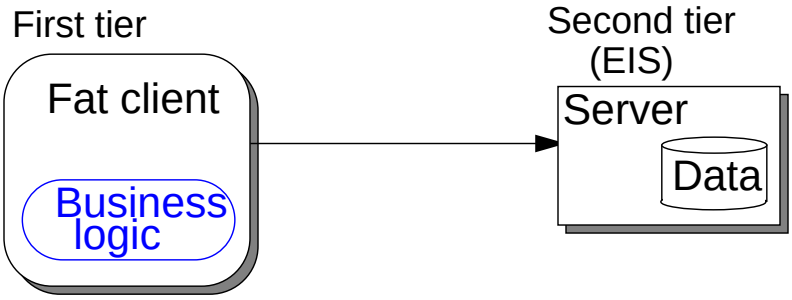
# Overview

- Objectives
- Relevance

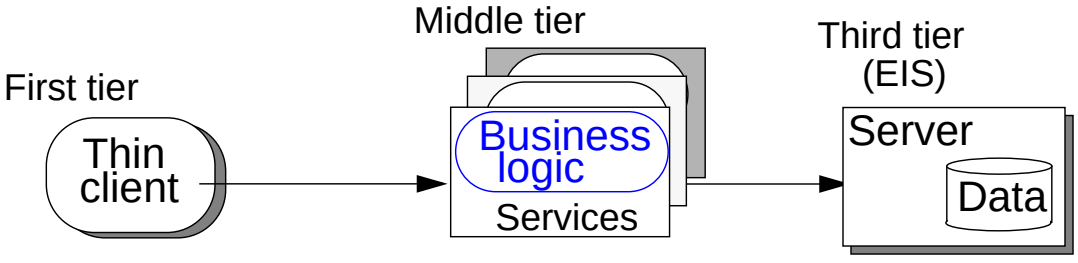
# The Tiered Model




Centralized, non-distributed




Client-server
Distributed Two-tier

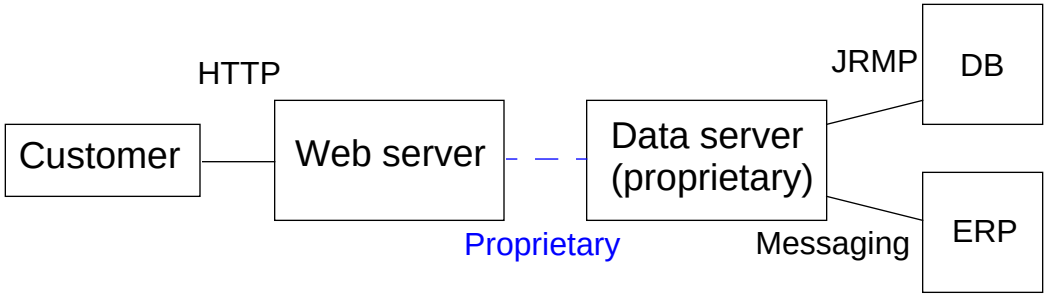



Distributed Three-tier

# Issues With the Distributed Architecture

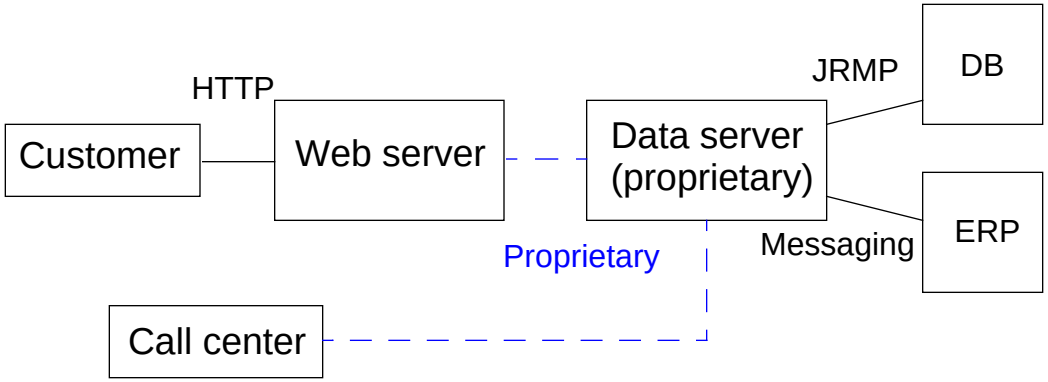
- Additional complexity
- Multiple points of failure
- Network Bandwidth

# Case Study




- Business logic resides on Web server.
- Data server manages transactions and requests to the database and the ERP system.
- Architecture affects scalability and availability.

# Case Study




- Business logic must move to the data server.
- Business process changes should not cause the enterprise to rework its entire application.

# Application Servers

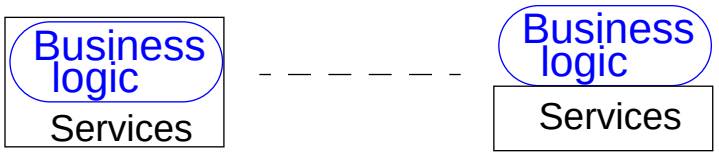
- Provide infrastructure to support management of business logic and access to services
- Can provide standard APIs instead of proprietary APIs, which leads to vendor lock
- Can provide ability to reuse business logic on other vendors' application servers

# Enterprise Goals

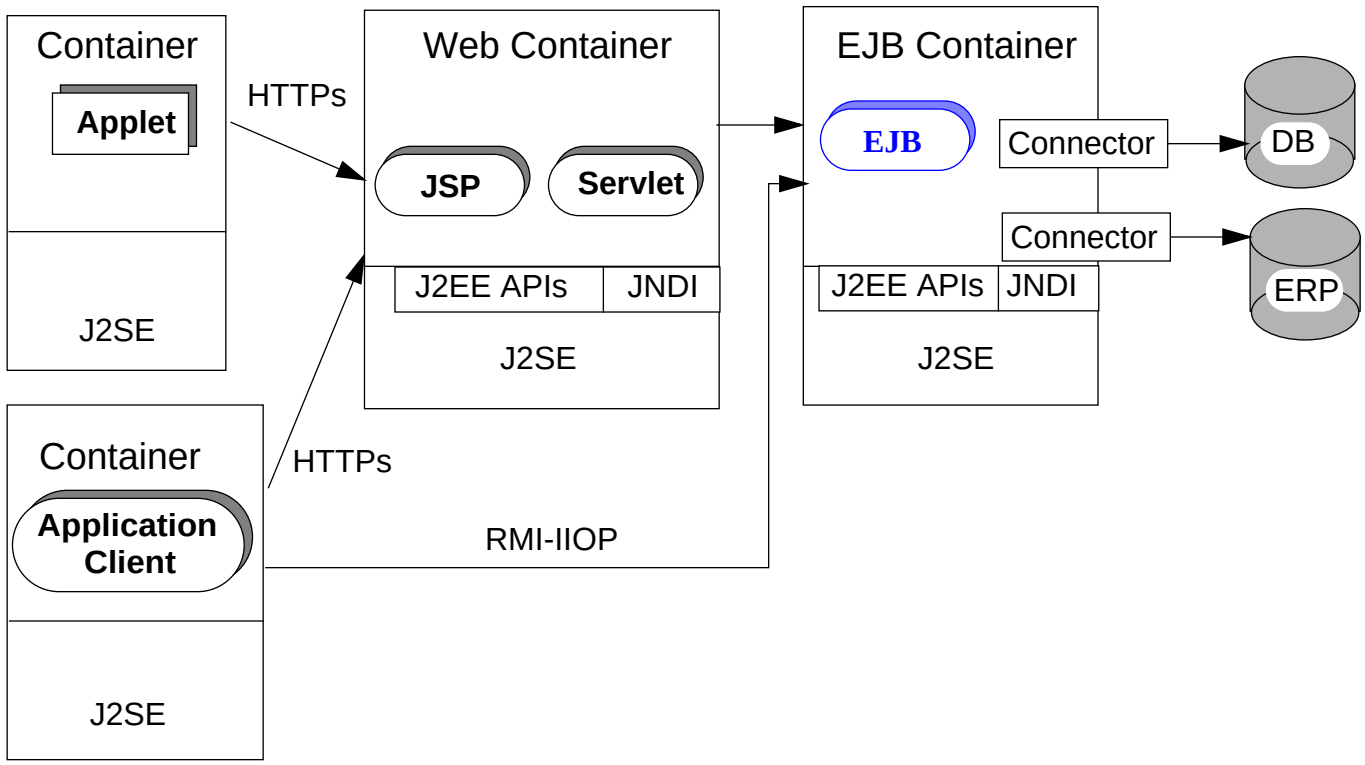
- Reuse
- Interoperability
- Portability
- Time-to-market

# What Is J2EE?

- J2EE provides a service-oriented infrastructure to automatically support and manage components.
- The enterprise developer can concentrate on application components, not the underlying services.
- Separation of business logic and services provide for better reuse of business logic.



# What Is J2EE?



# What Is J2EE?

| Services | Technology |
|---|---|
| Web | Servlets/JSPs |
| Database | JDBC™ |
| Naming And Directory | Java Naming and Directory Interface (JNDI) |
| Messaging | Java Message Service (JMS) * |
| Email | JavaMail™<br>Java Activation Framework (JAF) |
| Distributed Object Frameworks | JavaIDL<br>Remote Method Invocation (RMI)<br>RMI-IIOP |
| Transactions | Java Transaction API (JTA) |
| Data Formats | XML, HTML |
| Protocols | TCP/IP, HTTP(S), IIOP, SSL |

# What Is J2EE?

- Platform specification:
  - Defines J2EE requirements
- Reference implementation:
  - Operational J2EE platform
- Compatibility test suite:
  - Validates J2EE platform compatibility
- J2EE Blueprints:
  - Describes how to build J2EE applications

# Enterprise JavaBeans™

- *Enterprise JavaBeans™* is an architecture for component-based distributed computing:
  - Customizable at deployment time
  - Deployed on a compatible application server
  - Portable to other application servers
- *Enterprise Beans* are components of distributed transaction-oriented enterprise applications.

# Defining EJB Technology

| JavaBeans™ Component Model | Enterprise JavaBeans™ Component Model |
|---|---|
| Design-time reuse and portability in an independent development environment (IDE) | Runtime environment for deployment in a server environment |
| Design-time customization is done using introspection and modification in an IDE | Runtime environment is changed using declarative programming |
| Set of interfaces and behaviors to be visually manipulated in a builder tool | Set of interfaces and semantics for a server-side component for execution within a server framework |
| Publish-and-subscribe event model typically used for communication between JavaBeans™ components on the same JVM | Events are not typically used for EJB components because they inherently have a remote interface to their clients |
| Components are independent objects that are executed within a JVM | Components are managed objects that are deployed within an application server |

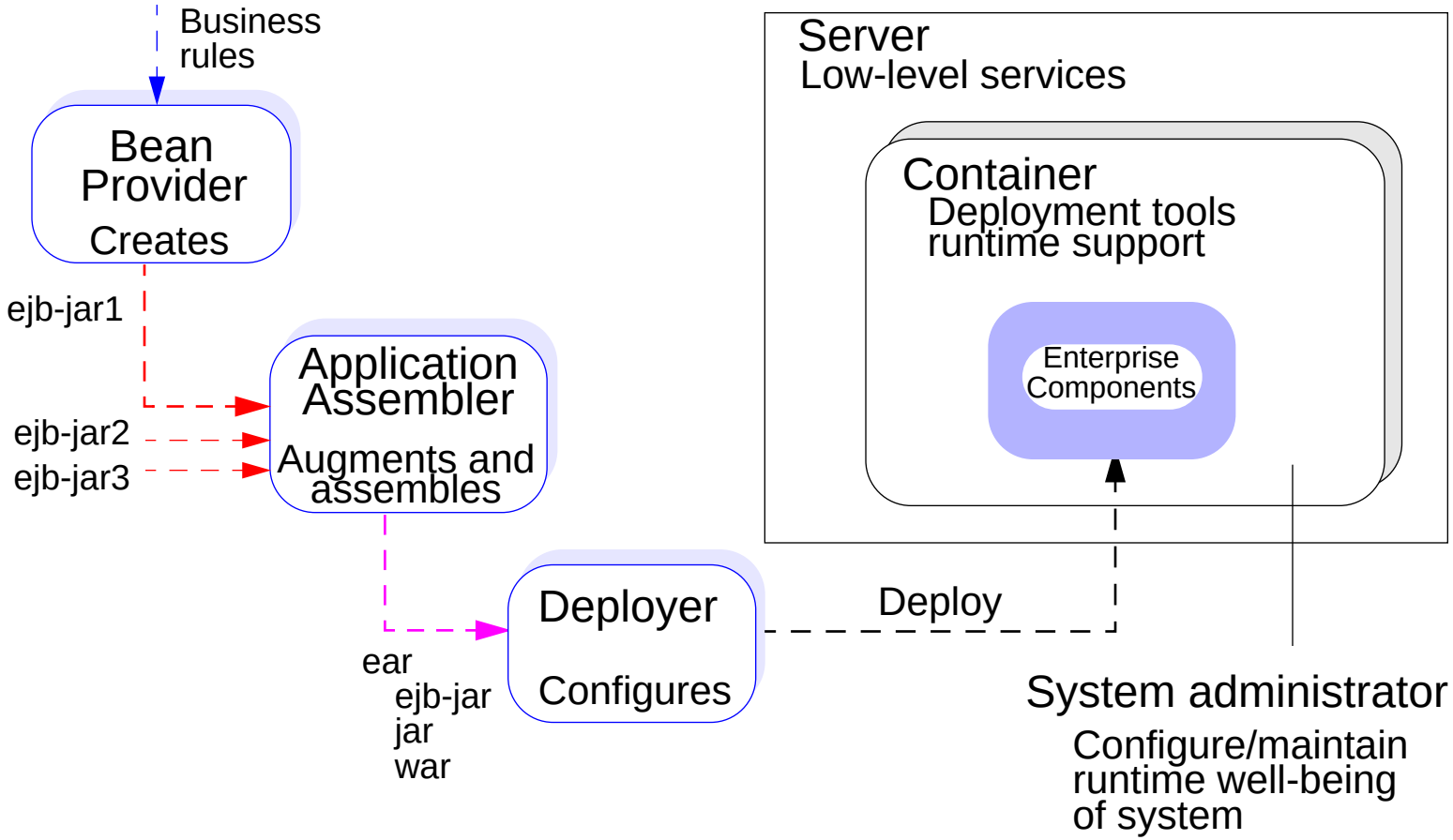
# Defining EJB Technology

- EJB servers provide core services to components:
  - Transaction
  - Security
  - Naming
  - Persistence
  - Life cycle
  - Concurrency
- EJB technology enhances:
  - Simplified access to services

# Defining EJB Technology

- A server component specification (for vendors)
- Separates and defines development roles:
  - Component creation
  - Application assembly
  - Application deployment

# EJB Developer Roles



# Check Your Progress

- Differentiate between client-server and multi-tier architecture
- List the advantages of and issues with distributed architecture
- List the advantages of J2EE
- Discuss the goals and scope of J2EE and EJB
- List two primary features of the EJB specification
- List two benefits of an EJB solution
- Differentiate between EJB and JavaBeans™

# Think Beyond

- Why is the container's level of abstraction necessary?
- How do the business components and services communicate?

# Module 2

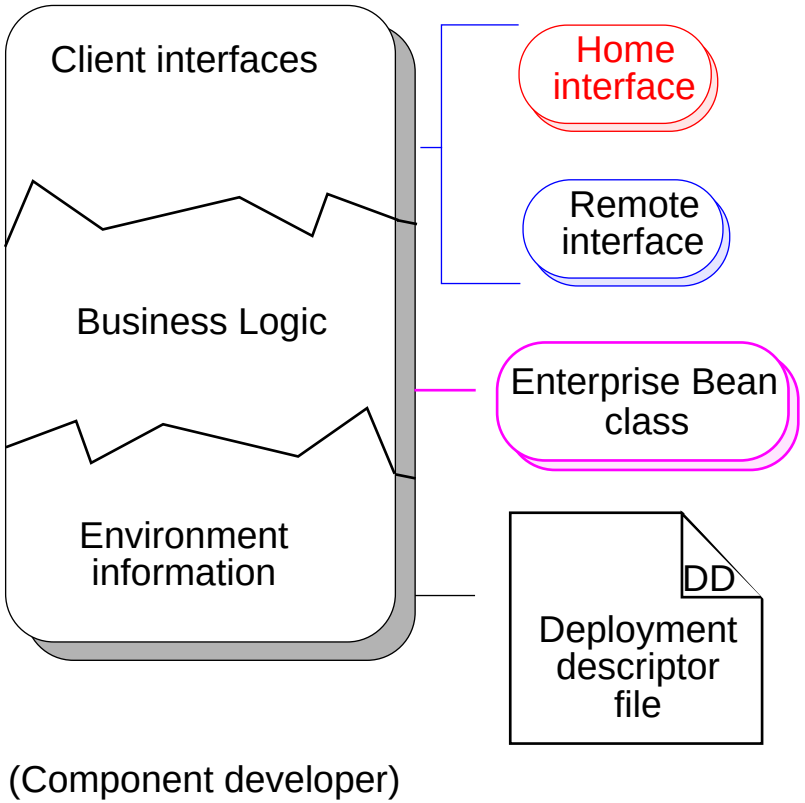
## EJB Framework

# Overview

- Objectives
- Relevance

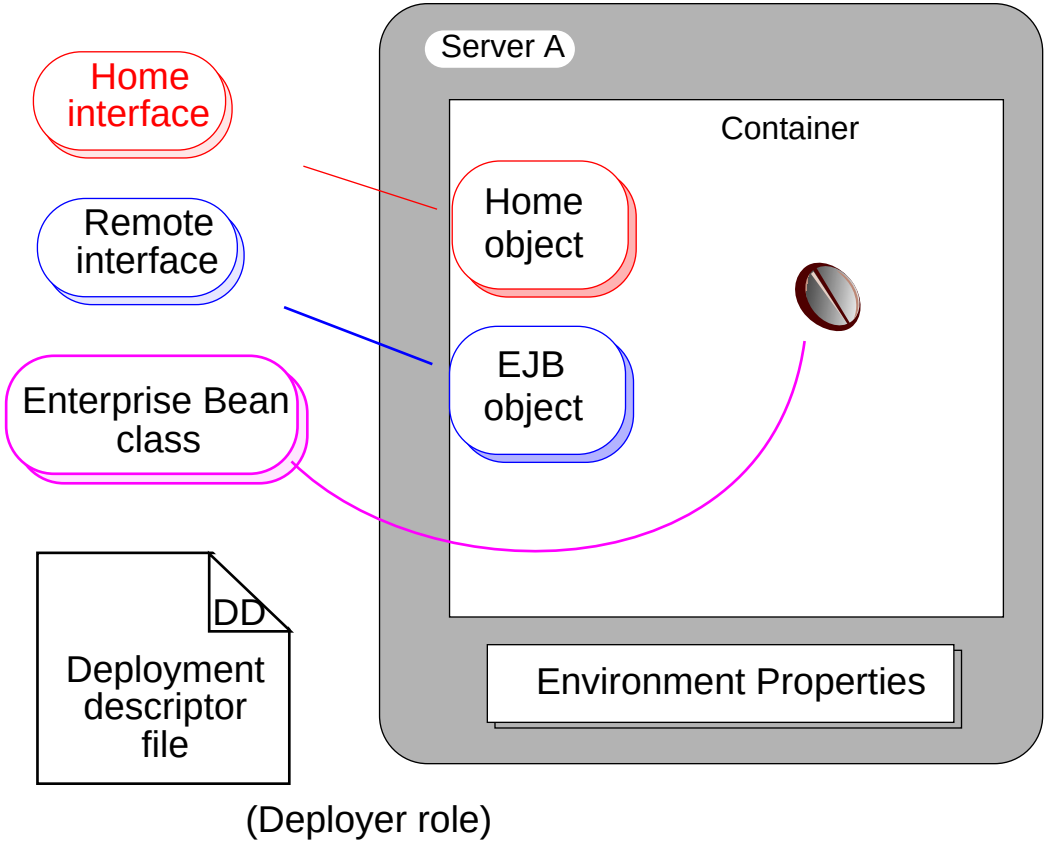
# EJB Programming Paradigm

- Declarative programming and customizing

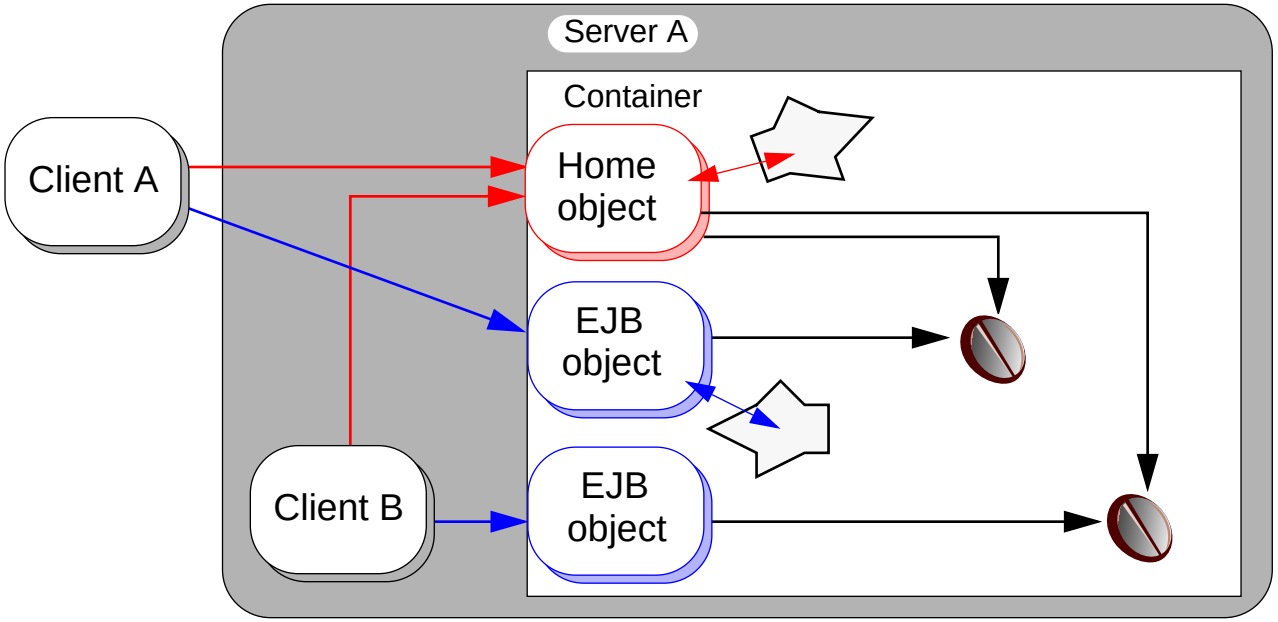


# EJB Programming Paradigm

- Deploying

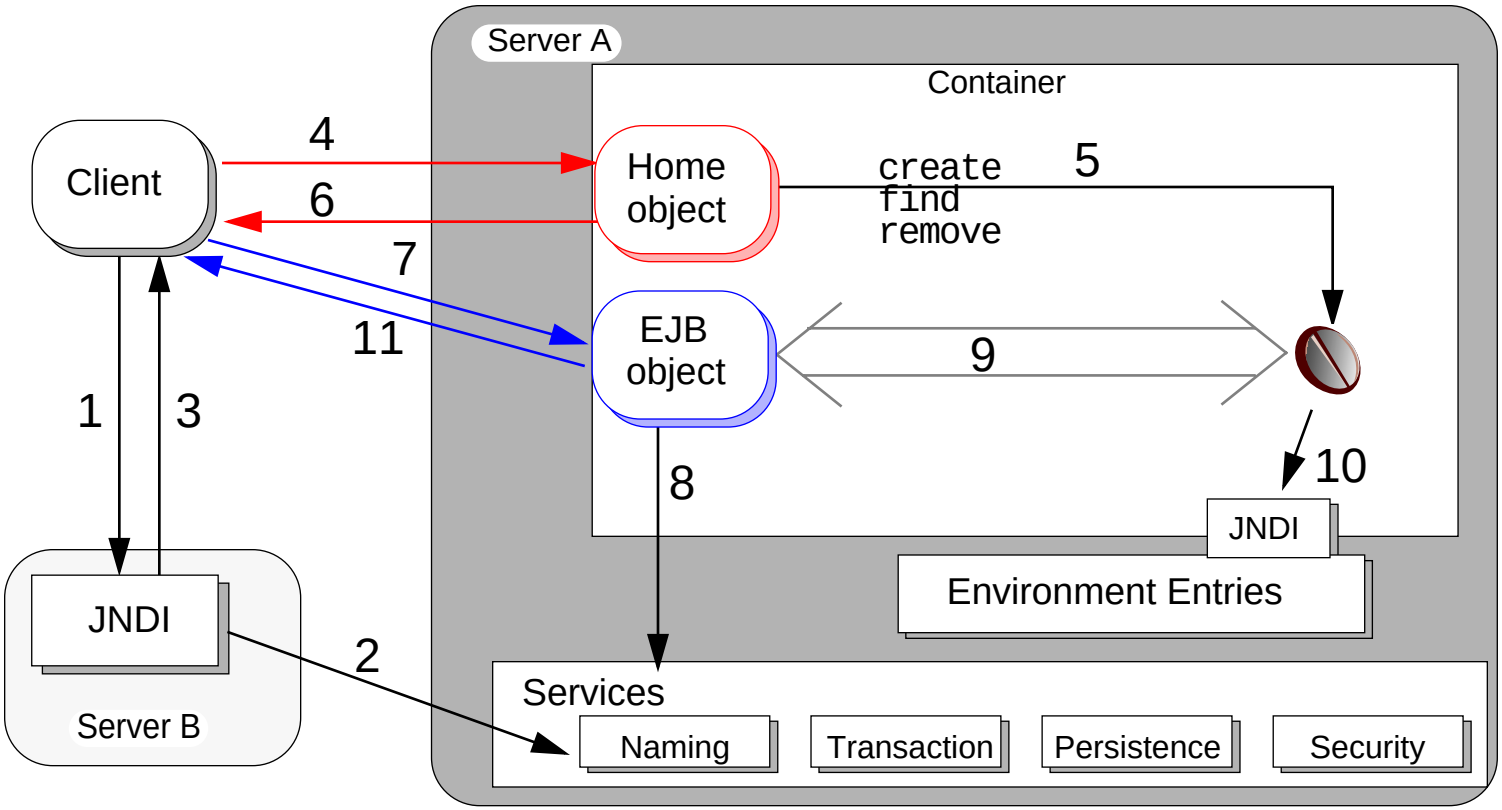


# EJB Architecture Overview




- Uniform client access whether local or remote
- Home object is a factory for EJBs
- EJB object is the remote object for accessing EJBs

# EJB Architecture Overview



# EJB Server

- Services:
  - Transactions
  - Security
  - Naming
  - Persistence
  - Lifecycle
  - Concurrency

# EJB Container

- Can contain multiple beans
- Is normally provided by EJB server vendor
- Is not visible to the client
- Is implemented in different ways by different vendors

# EJB Container

- Provides some standard services:
  - Persistence
  - Transaction control
  - Security
  - EJB instance lifecycle management
  - EJB instance identification

# Home Interface

- Factory interface for the bean:
  - Interface provided by EJB developer
  - Exposes methods for creating or locating beans
- EJB server provides:
  - Implementation class
  - Stubs and (optionally) skeletons
  - One factory per bean class
  - Registration of factory with naming service

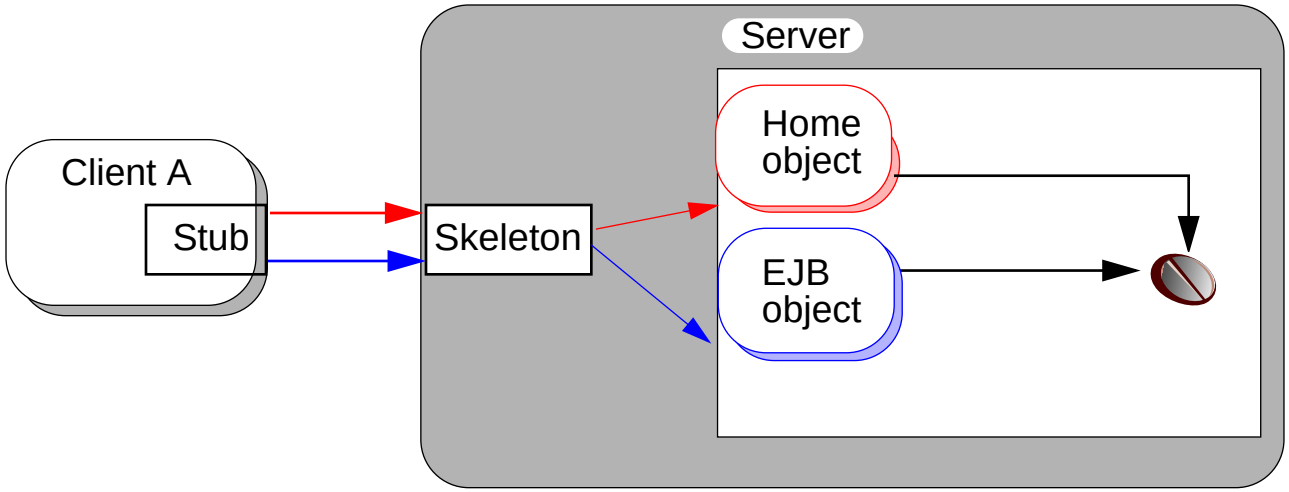
# Remote Interface

- Proxy interface for the bean:
  - Interface provided by EJB developer
  - Exposes business methods
- EJB server provides:
  - Implementation class
  - Stubs and (optionally) skeletons
  - Instances of as needed

# EJB Object

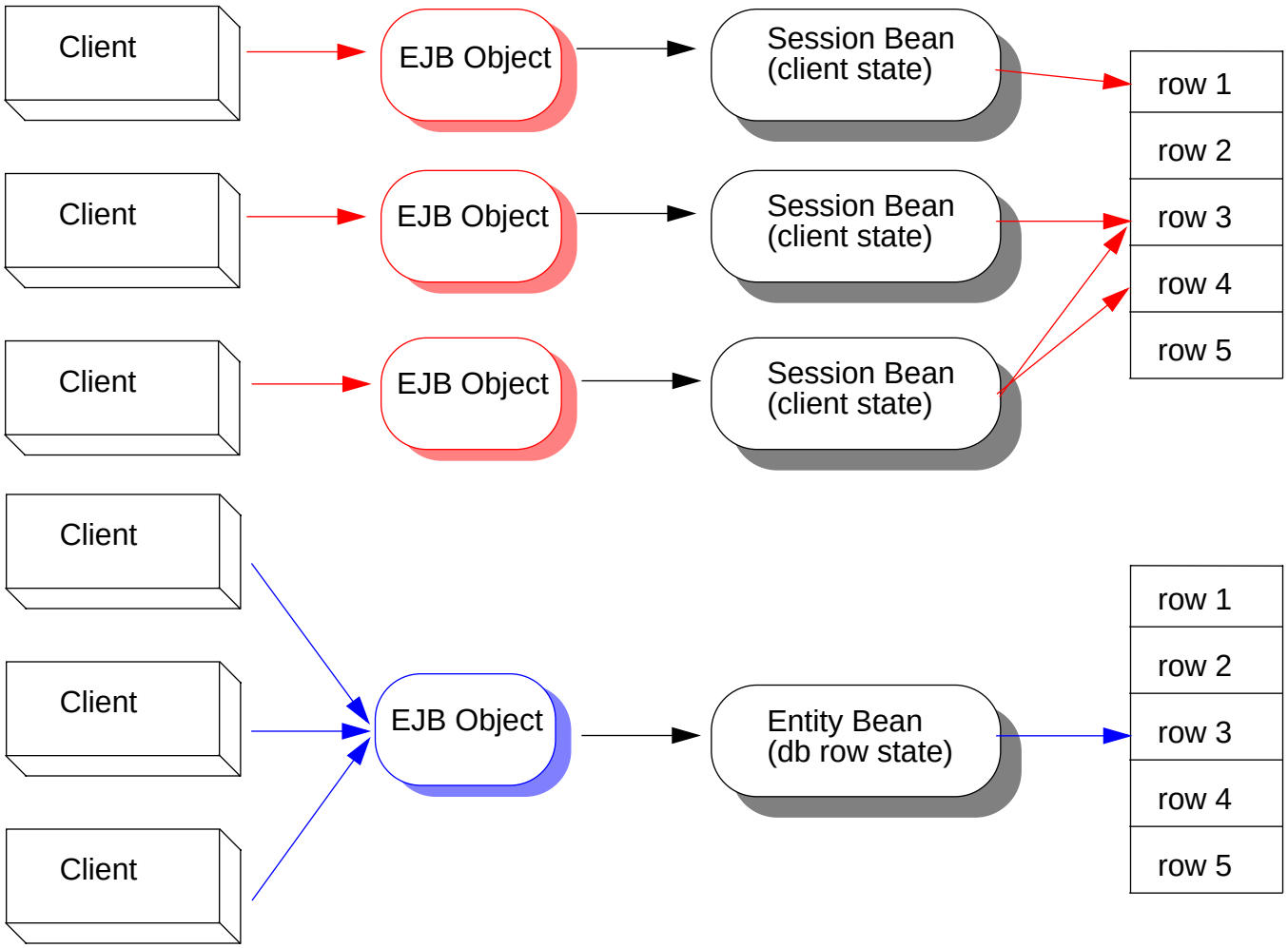
- The EJB object is a wrapper object that acts as a proxy to the bean instance.
  - Interface provided by bean developer.
  - Implementation class generated by vendor tool.
- Each business method has a corresponding wrapper method in the EJB object.
  - Wrapper method implements container-provided services.

# Stubs and Skeletons




- Stub is downloaded from server
- Communication between stub and skeleton can be proprietary, yet EJB code is still portable

# Types of EJBs



# Types of EJBs

| **Session Bean** | **Entity Bean** |
|---|---|
| Private client resource | Accessed by multiple clients concurrently |
| Client affects Bean's state | Bean represents state of data |
| Bean is not persistent; state can be stored in secondary memory | Bean is persistent; state is synchronized with corresponding state of data |
| Lifetime is normally controlled by client | Lifetime corresponds to its persistent store |
| Normally created for use | Normally located for use |
| Transaction participant (optional) | Transactional by nature |
| Single-threaded | Single-threaded by default |
| Mandatory since 1.0 specification | Mandatory since 1.1 specification |

# Session Beans

- *Stateful* beans maintain client-specific state across method calls.
  - `create` methods can pass arguments.
- *Stateless* beans do not maintain client state across method calls for the same client.
  - `create` methods are no-args.
  - Container can pool stateless Beans since they do not maintain state.

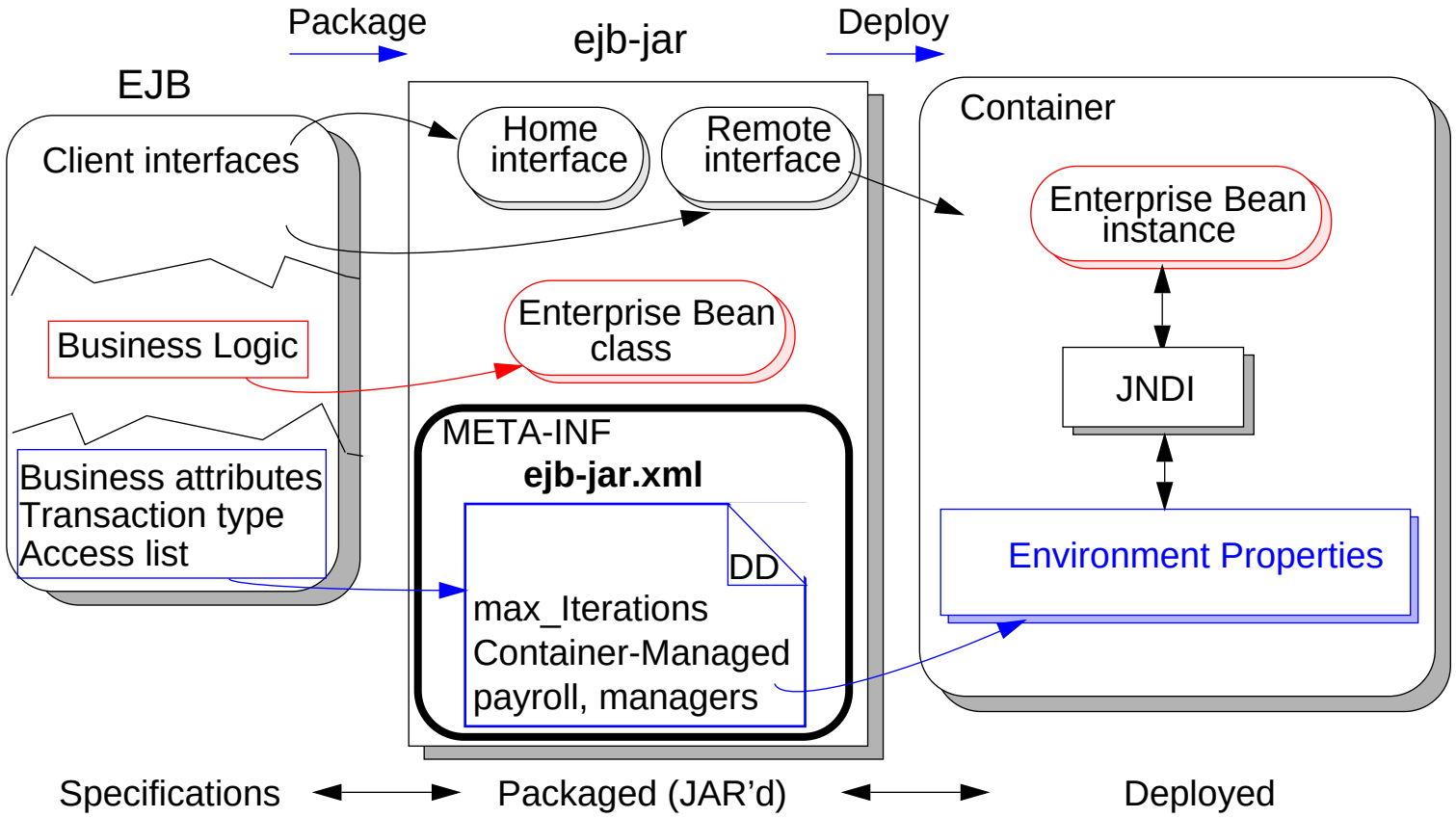
Note: Declare the bean's type in the deployment descriptor.

# Entity Beans

- Are persistent objects
- Are transactional by nature
- Use two management techniques
  - Container-managed persistence (CMP)
  - Bean-managed persistence (BMP)

Note: Declare the bean's persistence management type in the deployment descriptor.

# EJB Properties



# Deployment Descriptor (DD)

- Portable property sheet for the EJB
- XML document
- Describes various attributes of one or more EJBs:
  - Structural information
  - Application assembly information
  - Declaration of required resources
- One deployment descriptor per jar (multiple EJBs)
  - META-INF/ejb-jar.xml
- Allows for post-compile customization of EJB

# EJB-JAR File

- Contains classes and DD required by deployer:
  - Enterprise bean class
  - Home interface class
  - Remote interface class
  - Primary key class (for entity beans)
  - Helper classes
- Does not contain:
  - Home interface implementation (home object)
  - Remote interface implementation (EJB object)

# Check Your Progress

- List the three primary components in an EJB server runtime
- List four services the container can provide to the bean
- List three differences between session and entity beans
- List the four required objects that are placed in the EJB jar file

# Check Your Progress

- Discuss the shift in the programming paradigm from traditional development to a role-based, declarative development model
- List the types of clients that can access an EJB component
- List the steps involved for a client to communicate with an enterprise bean

# Think Beyond

- How would you decide which type of EJB component to use in your application?

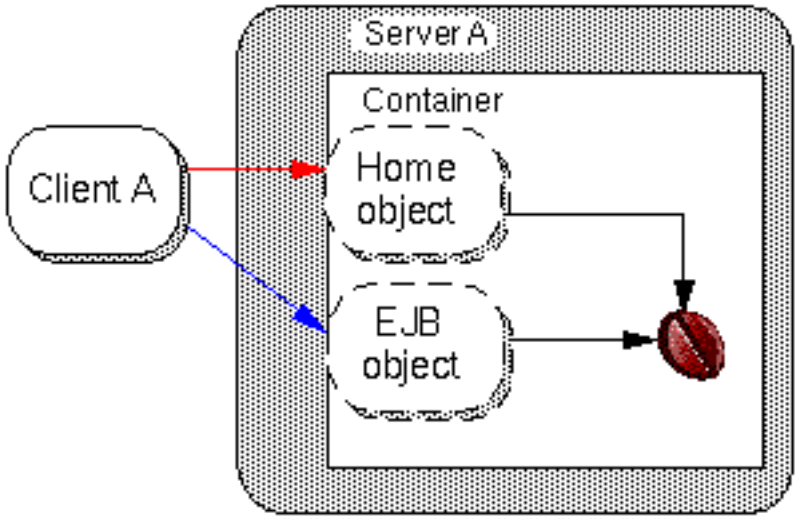
# Module 3

## Writing a Session Bean

# Overview

- Objectives
- Relevance

# Writing the Enterprise Bean Class



# Writing the Enterprise Bean Class

- Write a Java programming language class using any superclass.
  - Define one or more `ejbCreate` methods
  - Define business methods
- For a session bean, implement:
  - `javax.ejb.SessionBean`
  - `javax.ejb.SessionSynchronization` (optional)
- For an entity bean, implement:
  - `javax.ejb.EntityBean`

# Writing the Enterprise Bean Class

```
1  package com.suned.cart;
2
3  import javax.ejb.*;
4
5  public class CartEJB implements SessionBean {
6
7     public void ejbCreate(String person) throws CreateException {};
8
9     public void ejbCreate(String person, String id)
10          throws CreateException {};
11
12    public void ejbRemove() {}
13
14    // Plus, you must implement all the SessionBean interface methods.
15
16 }
```

# Defining `ejbCreate`

- Must be `public` (not `static` or `final`)
- Must have `void` return type (session bean only)
- Acts as a bean constructor
- Is defined with overloaded definitions
- Receives arguments from `create` methods
- Sets client session state per instance
- Can define and throw any exceptions
  - Note: Declare exceptions in the home interface

# Defining `ejbCreate`

```
17   public void ejbCreate(String name) throws CreateException {
18        ejbCreate(name, generateID());
19   }
20
21
22   public void ejbCreate(String name, String ID)
23     throws CreateException {
24
25     if (name == null) {
26       throw new CreateException("null name not allowed");
27     }
28
29    if (!validate(ID)) {
30      throw new CreateException("invalid ID: " + ID);
31    }
32
33    customerName = name;
34    customerID = ID;
35
36    contents = new Vector();
37  }
```

# SessionBean Interface

```
1  public interface SessionBean extends EnterpriseBean {
2
3      public void ejbRemove()        ...;
4
5      public void setSessionContext(SessionContext sc)   ...;
6
7        // These next two are used when swapping beans.
8
9      public void ejbActivate()      ...;
10
11     public void ejbPassivate()     ...;
12
13 }
```

Note: These methods are callbacks.

# `EJBContext` Interface

- Superclass for `SessionContext` and `EntityContext`

```
1  public interface EJBContext {
2
3     public EJBHome           getEJBHome();
4
5     // Access to security context.
6     public Principal         getCallerPrincipal();
7     public boolean           isCallerInRole(String role);
8
9     // Access to transaction demarcation control.
10    public UserTransaction   getUserTransaction();
11
12    public boolean           getRollbackOnly();
13    public void              setRollbackOnly();
14 }
```

# `SessionContext` Interface

```
1  public interface SessionContext extends EJBContext {
2
3      public EJBObject getEJBObject()      ...;
4
5  }
```

- Is a subclass of `EJBContext`
- Declares one additional method
- Is passed to bean during creation

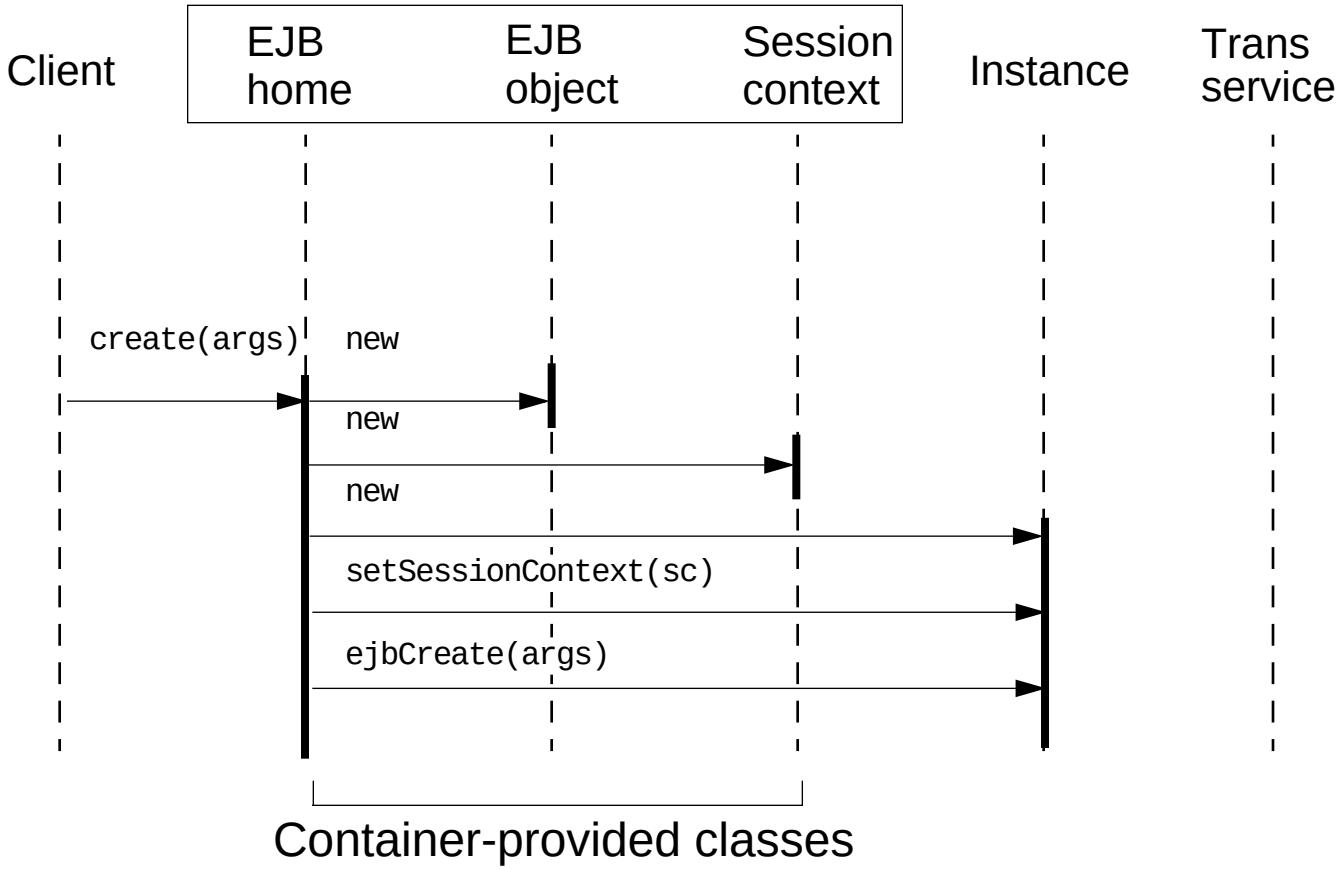
# setSessionContext

- `setSessionContext` is called exactly once for every bean instance.
- Optionally, saves the reference to the session context.

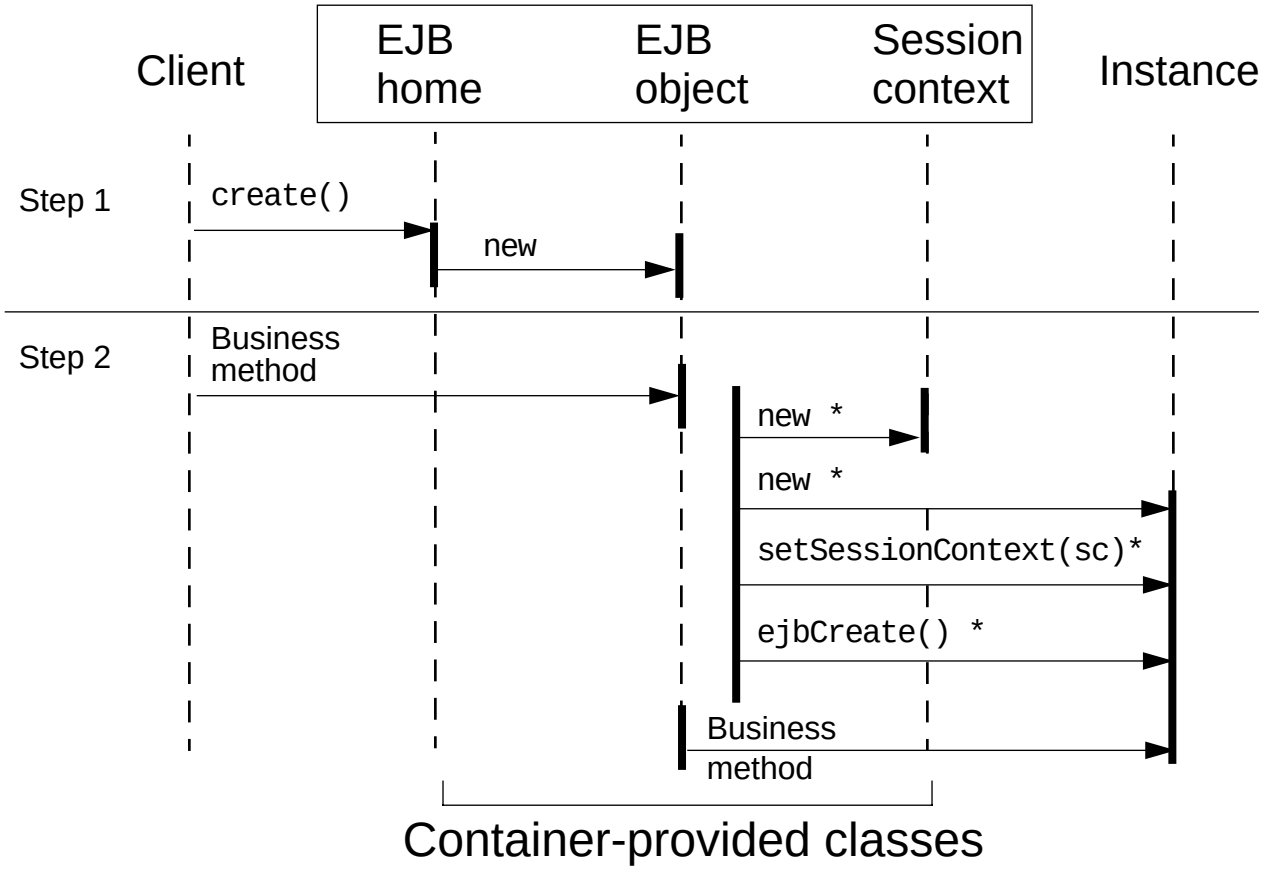
```
65   public void setSessionContext(SessionContext sc) {
66        this.sc = sc;
67    }
```

- Does not store any object references contained within the context, such as the Principal.

# Container: Creating a Stateful Session Bean



# Container: Creating a Stateless Session Bean




* If the stateless session bean is taken from the pool, these steps will not occur.

# Writing a Business Method

- Define a method in the bean class.
- Declare it in your remote interface.
- Pass and return any types that meet the requirements for RMI distributed argument passing.

# Writing a Business Method

```
43   public void addBook(String title) {
44
45      contents.addElement(title);
46   }
47
48   public void removeBook(String title) throws BookException {
49
50      boolean result = contents.removeElement(title);
51      if (result == false) {
52         throw new BookException(title + " not in cart.");
53      }
54   }
55
56   public Vector getContents() {
57      return contents;
58   }
```

# Defining `ejbRemove`

- Is called on the bean as a result of:
  - Client calling remove on a stateful bean
  - Server reducing the stateless bean pool size
- Should undo any work done in `ejbCreate`
- Is often an empty method
- Might not be called due to server crash or system exception

# Swapping Stateful Session Beans

- Container:
  - Can use swapping technique to release resources to minimize memory utilization
  - Notifies the bean via `ejbPassivate` and `ejbActivate`
  - Cannot swap out session bean while in a transaction
  - Uses timeout setting in server's tool
- Empty implementations are usually sufficient.
- Complex beans may prefer to release resources.

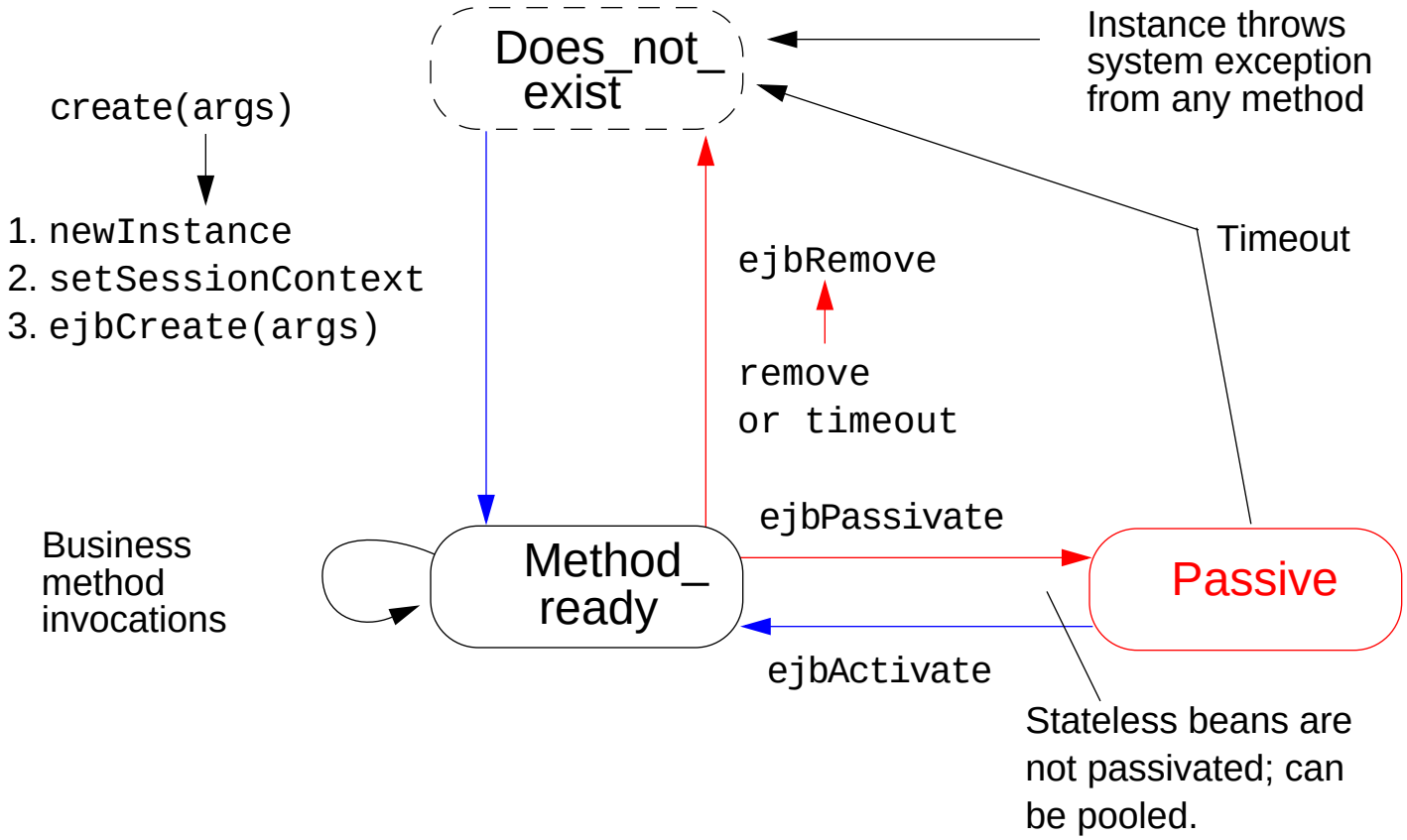
# Swapping Session Beans

```
1  public class CustomerBean implements SessionBean {
2
3      public void ejbPassivate() {
4          try {
5              conn.close();
6          } catch (SQLException sqle) {
7              throw new EJBException(sqle);
8          }
9      }
10
11     public void ejbActivate() {
12         try {
13             conn = dataSource.getConnection();
14         } catch (SQLException sqle) {
15             throw new EJBException(sqle);
16         }
17     }
18     ...
19 }
```

# Avoid the Use of Transient Fields

- Java technology serialization is not required for persistence mechanism.
- `ejbPassivate` should:
  - Not rely on Transient flag
  - Set references to null explicitly
- `ejbActivate` should offset the `ejbPassivate` work.

# Stateful Instance Life Cycle



# Complete Session Bean Example

```
1  import javax.ejb.*;
2  import java.util.*;
3
4  public class SimpleCartBean implements SessionBean {
5
6   private String customerName;
7   private String customerID;
8   private Vector contents;
9   private SessionContext ctx;
10
11
12    public void ejbCreate(String name) throws CreateException {
13         ejbCreate(name, generateID());
14   }
15
16   public void ejbCreate(String name, String ID) throws CreateException {
17     if (name == null) {
18       throw new CreateException("null name not allowed");
19     }
20
21    if (!validate(ID)) {
22      throw new CreateException("invalid ID: " + ID);
23    }
24
25    customerName = name;
26    customerID = ID;
```

```
27
28    contents = new Vector();
29  }
30
31  public boolean validate(String id) {
32
33     return IDValidator.validate(id);
34   }
35
36  public String generateID() {
37    return IDValidator.generateID();
38  }
39
40  public void addBook(String title) {
41     contents.addElement(title);
42   }
43
44  public void removeBook(String title) throws BookException {
45     boolean result = contents.removeElement(title);
46     if (result == false) {
47       throw new BookException(title + " not in cart");
48     }
49  }
50
51   public Vector getContents() {
52     return contents;
53    }
54
55   public SimpleCartBean() {}
56
57   public void ejbRemove() {}
```

```
58   public void ejbActivate() {}
59   public void ejbPassivate() {}
60   public void setSessionContext(SessionContext ctx) {
61     this.ctx = ctx;
62   }
63
64 }
```

# Recap

| Interfaces | Methods |
|---|---|
| SessionBean | ejbRemove() |
| | ejbActivate() |
| | ejbPassivate() |
| | setSessionContext(sc) |
| EJBContext | getEJBHome() |
| | getCallerPrincipal() |
| | isCallerInRole (String role) |
| | getUserTransaction |
| | getRollbackOnly() |
| | setRollbackOnly() |
| SessionContext | getEJBObject() |

# Exercise: Writing a Session Bean

- Objective
- Task

# Check Your Progress

- List the four methods in the `SessionBean` interface
- Describe the swapping mechanism for session beans
- Describe the difference between a stateful and a stateless session bean
- Explain how the enterprise bean's life cycle is managed

# Think Beyond

Given that the home object manages the life cycle of the enterprise bean, and the remote object acts as a proxy to intercede all business calls made on the enterprise bean, list which EJB methods should reside in the home interface. Which methods should reside in the remote interface?

# Module 4

## Defining the Interfaces

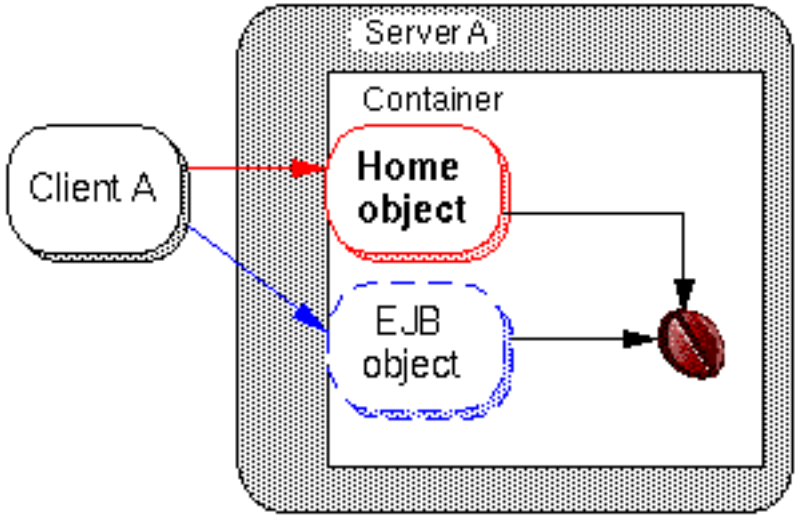
# Overview

- Objectives
- Relevance

# Writing the Home Interface

- Extend from `javax.ejb.EJBHome`.
- Declare `create` methods for each `ejbCreate` in your bean.
  - Signatures must match.
  - Declare `javax.ejb.CreateException`, `java.rmi.RemoteException`
  - `create` methods must return remote interface type.
  - Stateless session beans can have only a no-arg `create` method.

# Writing the Home Interface




- Home interface is implemented by the container's tools.
- Home class is instantiated at deployment time.

# Writing the Home Interface

```
1
2  import java.rmi.RemoteException;
3  import javax.ejb.CreateException;
4  import javax.ejb.EJBHome;
5
6  public interface CartHome extends EJBHome {
7
8     Cart create(String person) throws RemoteException, CreateException;
9
10    Cart create(String person, String id) throws RemoteException,
11                                                 CreateException;
12 }
```

# The Factory Interface

```
1  public interface EJBHome extends Remote {
2
3     public EJBMetaData getEJBMetaData() throws
              java.rmi.RemoteException;
4
5     public HomeHandle getHomeHandle() throws
              java.rmi.RemoteException;
6
7     public void remove(Handle handle) throws
              javax.ejb.RemoveException, java.rmi.RemoteException;
8
9     //The following method is used only for entity Beans.
10
11    public void remove(Object primaryKey) throws
              javax.ejb.RemoveException, java.rmi.RemoteException;
12
13 }
```

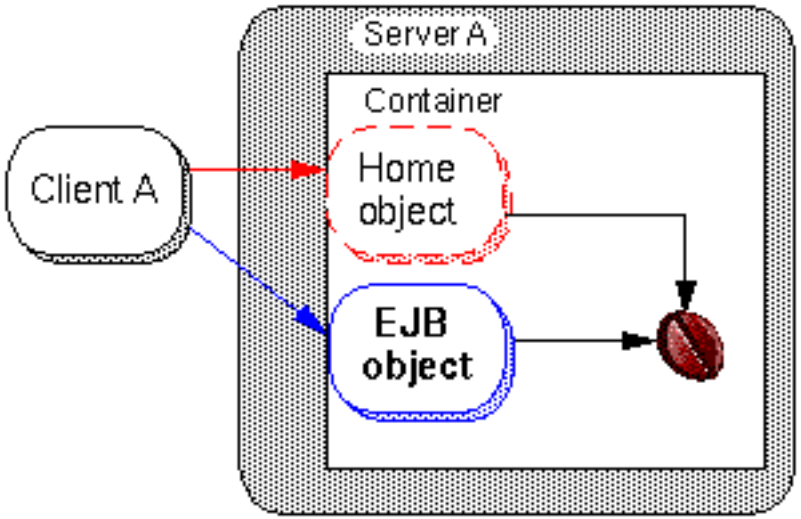
- Container vendor tool generates the factory class.

# `EJBMetaData` Interface

```
1  public interface EJBMetaData {
2
3     public EJBHome getEJBHome();
4
5     public Class getHomeInterfaceClass();
6
7     public Class getRemoteInterfaceClass();
8
9     public boolean isSession();
10
11    // The following method is used only for entity Beans.
12
13    public Class getPrimaryKeyClass();
14 }
```

- Provides basic metadata about the EJB class
- Required by development tools, not bean provider

# Writing the Remote Interface




- Remote interface is implemented by the container's tools.
- Remote class is instantiated when create method is called.

# Writing the Remote Interface

- The remote interface extends from `javax.ejb.EJBObject`.
- Declare methods that correspond to the bean's business methods that clients should access.
- All application exceptions thrown in bean methods must be declared in this interface.

# Writing the Remote Interface

```
1  package com.suned.cart;
2
3  import java.util.*;
4  import javax.ejb.EJBObject;
5  import java.rmi.RemoteException;
6
7  public interface Cart extends EJBObject {
8
9     public void addBook(String title) throws RemoteException;
10
11    public void removeBook(String title)
12          throws BookException, RemoteException;
13
14    public Vector getContents() throws RemoteException;
15
16 }
```

# `EJBObject` Interface

```
1  public interface EJBObject extends Remote {
2
3     public EJBHome getEJBHome() throws java.rmi.RemoteException;
4
5     public void remove() throws java.rmi.RemoteException,
                javax.ejb.RemoveException;
6
7     public Handle getHandle() throws java.rmi.RemoteException;
8
9     public boolean isIdentical(EJBObject eo)
10         throws java.rmi.RemoteException;
11
12    // The following is used for entity Beans only.
13    public Object getPrimaryKey() throws java.rmi.RemoteException;
14 }
```

- `javax.ejb.EJBObject` for all remote interfaces
- Client gets access to these methods implicitly

# Handles

- `HomeHandle` allows a client to re-establish a connection to the home object without using JNDI (*EJB 1.1*).
- `Handle` allows a client to re-establish a connection to the EJB Object.
- Implementations are vendor-specific and provided by container.
- Handles are serializable.

# Method Types

- Business methods are invoked because of a client call.
  - Methods defined by Bean Provider:
  - `ejbCreate` and `ejbRemove` (except for stateless session beans)
  - Container callback methods are invoked at the discretion of the container:
  - `setSessionContext`, `setEntityContext`, `unsetEntityContext`
  - `ejbActivate`, `ejbPassivate`
  - `ejbLoad`, `ejbStore` (entity bean)

# Exceptions in EJB

System exceptions:

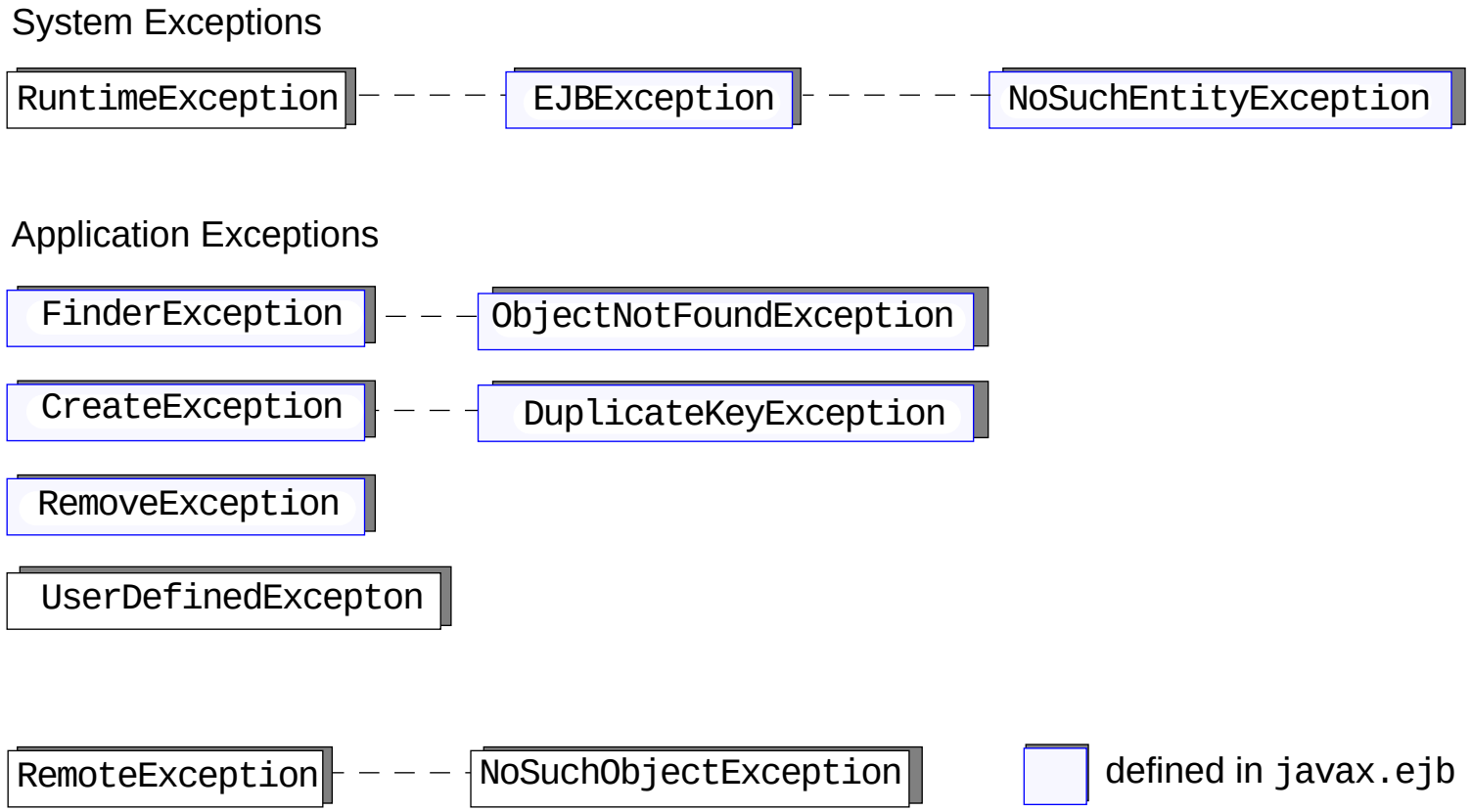
- Indicate non-application exceptions, such as a database access problem
- Cause the container to remove the bean instance
- Include `javax.ejb.EJBException` or a subclass
- Can use `EJBException` to wrap original exception
- Thrown to client as `RemoteException`

# Exceptions in EJB

Application exceptions:

- Indicate problems in business processing
- Do *not* cause the container to remove the bean instance
- Include `CreateException`, `RemoveException`, `FinderException`, and user-defined exceptions
- Can subclass EJB application exceptions, *not* `RuntimeException` or `RemoteException`
- Are thrown to client as is

# Exception Inheritance



# Issues With Session Beans

- These methods throw a `RemoteException`:
  - `EJBObject.getPrimaryKey()`
  - `EJBHome.remove(Object primaryKey)`
- This method throws a `RuntimeException`:
  - `EJBMetaData.getPrimaryKeyClass()`

# Issues With Session Beans

- The container throws a `RemoteException` if:
  - A second thread attempts to enter a session bean
  - A loopback call occurs on the session bean (similar issue)
- Container destroys a session bean if a `RuntimeException` occurs in the bean.

# Notes to Remember

- Home object handles can be passed over the network.
- Containers maintain internal session identifiers.
  `EJBObject.isIdentical(EJBObject otherSessionBean)`
- `EJBObject.isIdentical(Object otherSessionBean)` always returns true for stateless session beans from the same home.
- Session beans survive client shutdown, and can be reconnected using a handle.

# Notes to Remember

- Session beans can access databases, and take part in transactions.
- Session bean class must make a no-arg constructor available.
- Exceptions declared in interfaces do not have to be declared in the bean methods unless the method actually throws those exceptions.
- Bean methods should throw `EJBException`, not `RemoteException`, in the case of a system exception (as of EJB 1.1).

# Session Bean Example

## Home Interface

```
1 package com.suned.cart;
2
3 import java.io.Serializable;
4 import java.rmi.RemoteException;
5 import javax.ejb.CreateException;
6 import javax.ejb.EJBHome;
7
8 public interface CartHome extends EJBHome {
9
10   Cart create(String person)
11        throws RemoteException, CreateException;
12
13  Cart create(String person, String id)
14        throws RemoteException, CreateException;
15}
```

# Session Bean Example

## Remote Interface

```
1 package com.suned.cart;
2
3 import java.util.*;
4 import javax.ejb.EJBObject;
5 import java.rmi.RemoteException;
6
7 public interface Cart extends EJBObject {
8
9   public void addBook(String title) throws RemoteException;
10
11  public void removeBook(String title)
12       throws BookException, RemoteException;
13
14  public Vector getContents() throws RemoteException;
15}
```

# Recap

| Interfaces | Methods | Description |
|---|---|---|
| `EJBHome` | `getEJBMetaData()` | Obtain metadata for EJB component. |
| | `getHomeHandle()` | Obtain a handle to the home object. |
| | `remove(Handle handle)` | Remove the bean with the handle. |
| | `remove(Object primaryKey)` | Not allowed in a session bean. |
| `EJBMetaData` | `getEJBHome()` | Obtain the reference to the home object. |
| | `getHomeInterfaceClass()` | Obtain the home interface class. |
| | `getRemoteInterfaceClass()` | Obtain the remote interface class. |
| | `isSession()` | Check to see if bean is a session bean |
| | `getPrimaryKeyClass()` | Obtain the primary key class information. |
| `EJBObject` | `getEJBHome()` | Obtain the reference to the home object. |
| | `remove()` | Remove the bean. |
| | `getHandle()` | Obtain a handle to the bean. |
| | `isIdentical(EJBObject eo)` | Compare identity to another bean's identity. |
| | `getPrimaryKey()` | Not allowed in a session bean. |

# Exercise: Defining the Interfaces

- Objective
- Tasks

# Check Your Progress

- List the four methods required in your home interface, and explain their function
- List the five methods required in your remote interface, and explain their function
- List the superinterfaces for your home and remote interfaces
- Explain which exceptions should be thrown to indicate system errors
- Explain which exceptions should be thrown to indicate application errors

# Think Beyond

- Runtime information about the enterprise bean is configured and customized in an XML document.
- List the benefits of separating environment information this way.

# Module 5

## Deploying the Session Bean

# Overview

- Objectives
- Relevance

# Overview of the Deployment Descriptor

- XML document describing one or more enterprise beans
- Structural information:
  - Bean class
  - Home interface
  - Remote interface

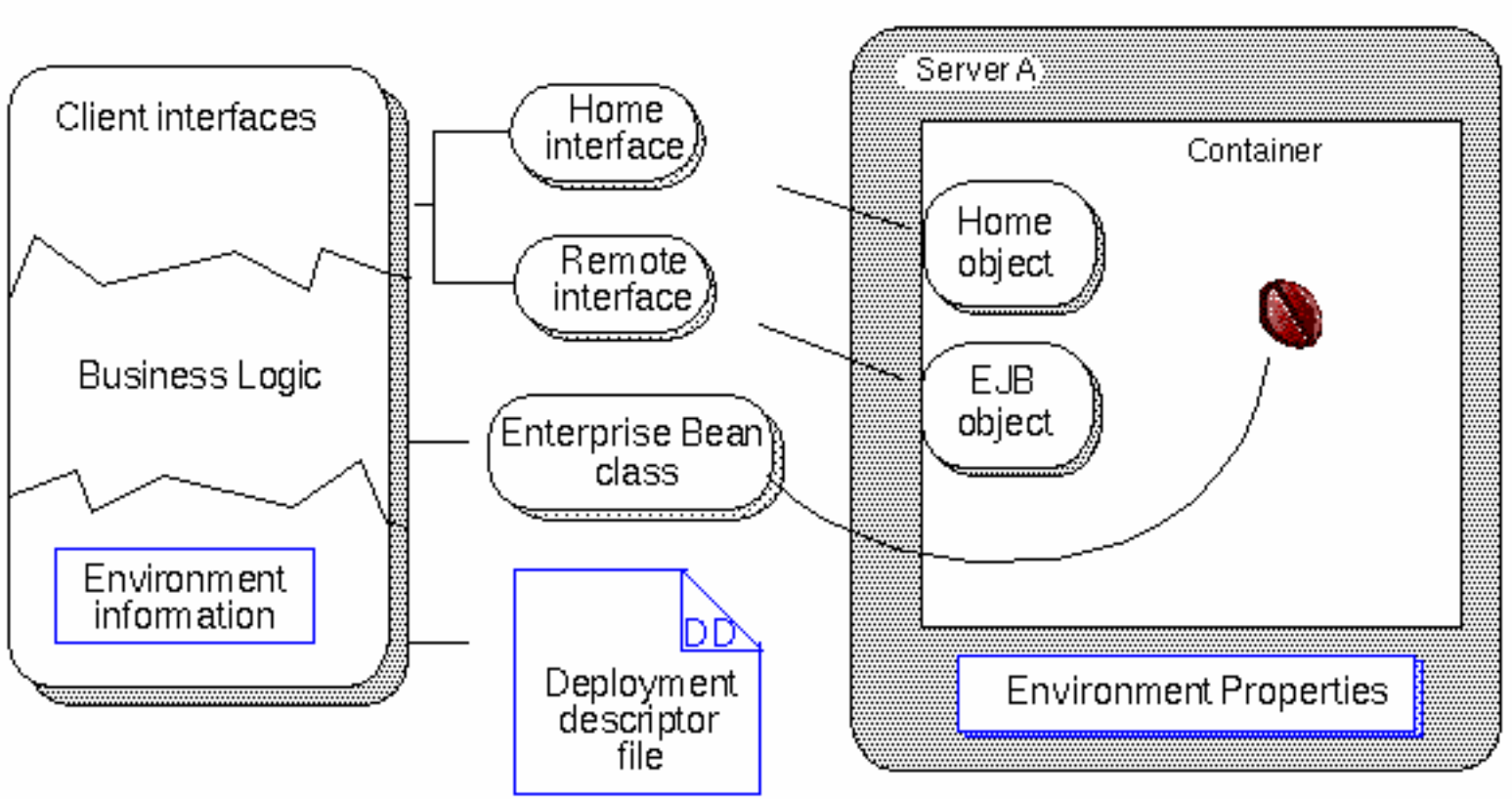
# Overview of the Deployment Descriptor

- Runtime information:
  - Environment name/value pairs
  - References to other beans
  - References to resource factories
  - Transaction controls

# Overview of the Deployment Descriptor

- Has security information:
  - Roles
  - Method invocation permissions
  - Resource access information
- Provided by bean provider, and modified by assembler and deployer
- Placed in the ejb-jar file along with the beans' classes and interfaces

# Overview of the Deployment Descriptor



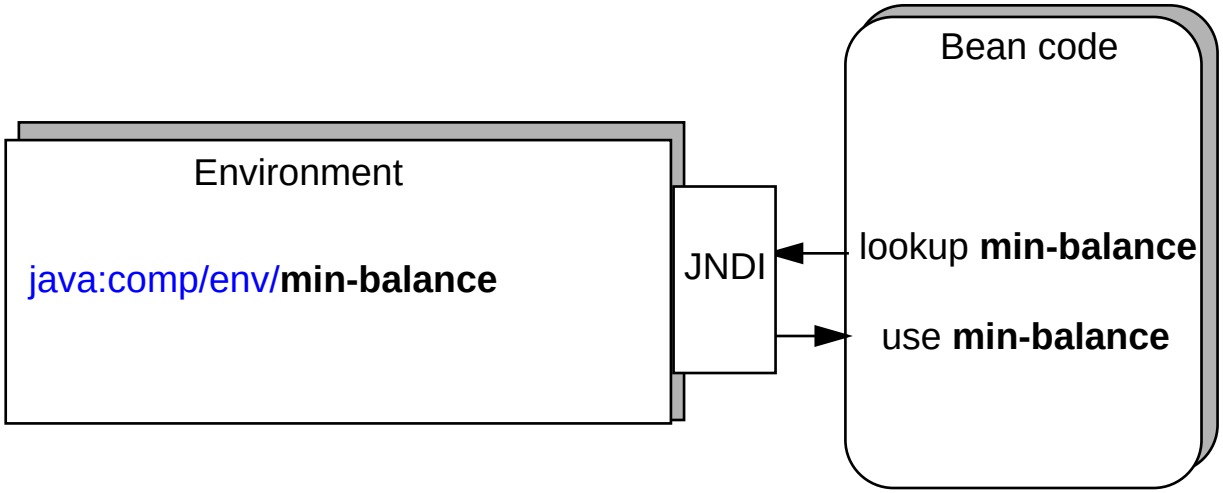
# Declaring Basic Bean Structure

| Tags | Definition |
|---|---|
| `ejb-jar` | Indicates this is an entry for one or more enterprise beans |
| `description` | Describes an enterprise bean or an assembled application |
| `enterprise-beans` | Contains a list of one or more enterprise beans |
| `session` | Describes each session bean's structural information |
| `ejb-name` | Identifies a symbolic name for a bean used as a reference by other parts of the DD |
| `ejb-class` | Identifies the name of the bean class |
| `home` | Identifies the name of the bean's home interface class |
| `remote` | Identifies the name of the bean's remote interface class |
| `session-type` | Identifies its state management type, either stateful or stateless |

# Declaring Basic Bean Structure

```
1  <ejb-jar>
2     <description>
3        Describes the entire deployment descriptor.
4     </description>
5
6     <enterprise-beans>
7        <session>
8           <description>
9              Describes this session Bean.
10          </description>
11          <ejb-name>     CustBugReport            </ejb-name>
12          <ejb-class>    com.xyz.CustBugBean      </ejb-class>
13
14          <home>         com.xyz.CustBugHome      </home>
15          <remote>       com.xyz.CustBug          </remote>
16
17          <session-type>           Stateful       </session-type>
18          <transaction-type>       Bean           </transaction-type>
19       </session>
20    </enterprise-beans>
21 </ejb-jar>
```

# Defining an Environment Entry




- JNDI is a lookup interface into a namespace.

# Defining an Environment Entry

- Name/value pairs are listed in DD.
- Entry names are provided by bean provider.
- Entry values can be added/changed by:
  - Bean provider
  - Application assembler
  - Deployer

# Defining an Environment Entry

- The following are valid environment value types:
  - String
  - Boolean
  - Integer, Long, Short, Byte
  - Double, Float
- Entries are placed in server namespace.
- Environment can be hierarchical.

# Defining an Entry in the DD

| Tag | Definition |
|---|---|
| `env-entry` | An entry for a single name/value pair |
| `description` | The purpose of this environment entry |
| `env-entry-name` | The name that will be referenced by the bean at runtime |
| `env-entry-type` | The data type of the value |
| `env-entry-value` | The value corresponding to this name. This entry is probably not provided by the bean provider, but by the assembler or deployer |

# Defining an Entry in the DD

```
1  <ejb-jar>
2       <session>
3          <ejb-name> NewAccountBean </ejb-name>
4          ...
5          <env-entry>
6             <description>
7                Minimum bank balance that is required before a loan
8                can be approved.
9             </description>
10            <env-entry-name> min-balance </env-entry-name>
11            <env-entry-type> java.lang.Integer </env-entry-type>
12            <env-entry-value> 500 </env-entry-value>
13         </env-entry>
14
```

# Defining an Entry in the DD

```
15          <env-entry>
16          <description>
17             Interest rate for a three year loan. It is
18             represented as a true multiplier, that is, 0.045 is 4.5%.
19          </description>
20          <env-entry-name>   three-year-rate    </env-entry-name>
21          <env-entry-type>   java.lang.Double   </env-entry-type>
22          <env-entry-value>  0.075              </env-entry-value>
23        </env-entry>
24      </session>
25      ...
26 </ejb-jar>
```

# Using the Environment Entry

- Use `InitialContext`.
- Access the environment with a JNDI name:

  `java:comp/env/{entry-name}`
- Can use hierarchical names such as:

```
java:comp/env/min-balance
java:comp/env/loanInfo/min-balance
java:comp/env/loanInfo/accepted/interestRate
```

- Values are read-only
- Do not narrow after lookup

# Using the Environment Entry

```
1  public class NewAccountBean implements SessionBean {
2     ...
3
4     public void setLoanInfo (int balance...)
5           throws InvalidBalanceException {
6        ...
7
8        //Obtain enterprise bean's environment naming context.
9
10       Context initCtx = new InitialContext();
11       Context myEnv = (Context)initCtx.lookup("java:comp/env");
12
13       //Obtain the minimum balance configured by the Deployer.
14       Integer min = (Integer)myEnv.lookup("min-balance");
15
16       //Use the environment entries to customize business logic.
17       If (balance < min.intValue() )
18            throw new LowBalanceException();
19
```

# Using the Environment Entry

```
20      //Get some more environment entries. These environment
21      //entries are stored in subcontexts (relative paths).
22      String val1 = (String)myEnv.lookup("foo/name1");
23      Boolean val2 = (Boolean)myEnv.lookup("foo/bar/name2");
24
25      //The enterprise Bean can also use full pathnames.
26      Integer val3 =
27         (Integer)initCtx.lookup("java:comp/env/name3");
28      Integer val4 =
29         (Integer)initCtx.lookup("java:comp/env/foo/name4");
30      ...
31   }
32 }
```

# Declaring Bean References

- Bean provider declares requirements for referencing other beans in the DD.
- Assembler can declare `ejb-names` of referenced beans.
- Deployer defines the JNDI names of the referenced beans.
- This allows for loose compile-time binding.

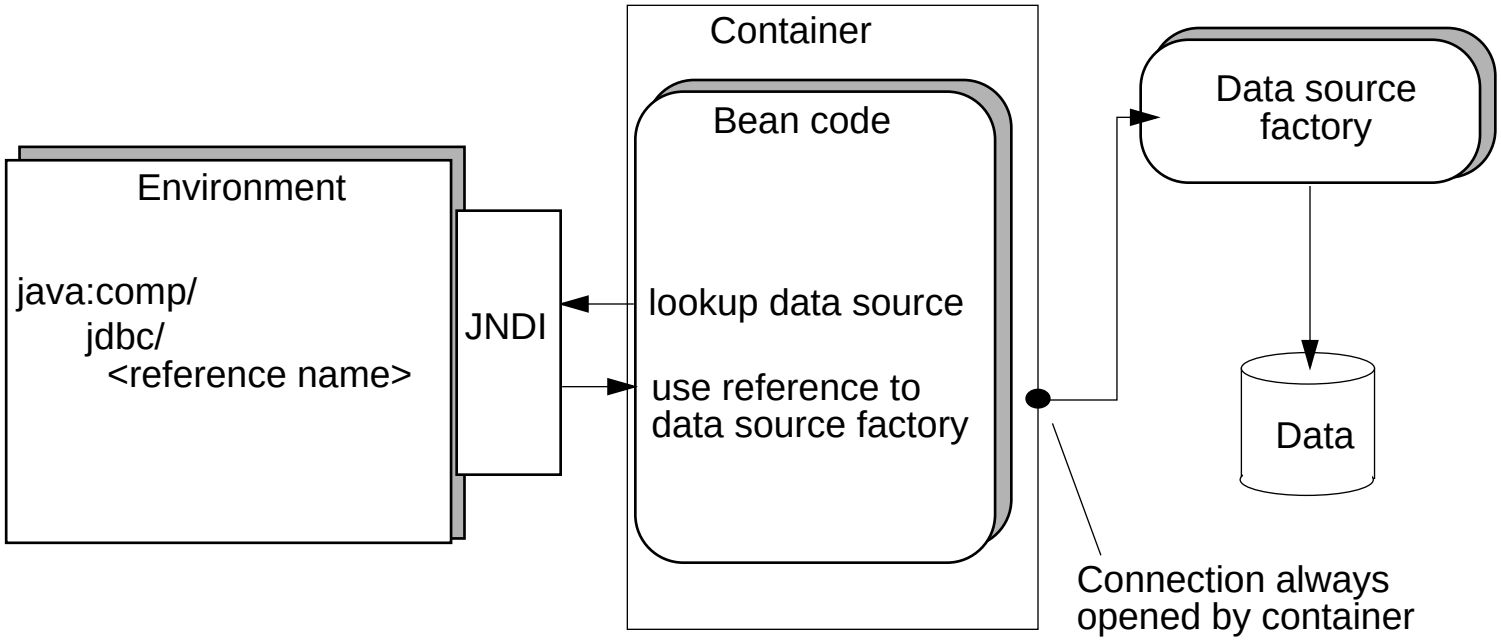
# Declaring Bean References in the DD

```
1  <ejb-jar>
2     <enterprise-beans>
3        <session>
4           <ejb-name> Shopping Cart Bean </ejb-name>
5           ...
6           <ejb-ref>
7              <ejb-ref-name>ejb/Customer</ejb-ref-name>
8              <ejb-ref-type>Entity</ejb-ref-type>
9              <home>labs.db.CustomerHome</home>
10             <remote>labs.db.Customer</remote>
11          </ejb-ref>
12       </session>
13
14          ...
15
16       </enterprise-beans>
17 </ejb-jar>
```

# Declaring Resource Factory References

- Resource factory allocates resources for the bean:
  - JDBC connections
  - JMS connections
- This is similar to the bean reference mechanism.
- Security issues are covered later.
- Connection to database is always connected through container.

# Declaring Resource Factory References



# Declaring Resource Factory References

```
1  <ejb-jar>
2       <enterprise-beans>
3       <session>
4           <ejb-name> Shopping Cart Bean </ejb-name>
5           ...
6           <resource-ref>
7             <res-ref-name>jdbc/ShoppingDB</res-ref-name>
8             <res-type>javax.sql.DataSource</res-type>
9             <res-auth>Container</res-auth>
10           </resource-ref>
11      </session>
12      </enterprise-beans>
13  </ejb-jar>
```

# Using Resource Factory References

```
14   private Connection con;
15   ...
16
17   try {
18      // Obtain a connection to the database.
19      InitialContext ic = new InitialContext();
20      DataSource ds = (DataSource) ic.lookup("java:comp/env/jdbc/ShoppingDB");
21      con =  ds.getConnection();
22
23   } catch (Exception ex) {
24         throw new EJBException
25            ("Unable to connect to database."
26               + ex.getMessage());
27
28   ...
```

# DD Responsibilities

- This code is in the `ShoppingCart` session bean:

```
1  // Obtain a reference to the Customer entity bean.
2  Context initCtx = new InitialContext();
3  Object result = iniCtx.lookup("java:comp/env/ejb/Customer");
4  CustHome custHome = (CustHome)
       javax.rmi.PortableRemoteObject.narrow(result, CustHome.class);
```

# Complete Deployment Descriptor Example

```
1
2  <?xml version="1.0" encoding="Cp1252"?>
3
4  <!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN' 'http://
java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>
5
6  <ejb-jar>
7    <description>Shopping Cart Application</description>
8    <display-name>Cart</display-name>
9
10   <enterprise-beans>
11     <session>
12
13       <ejb-name>Shopping Cart Bean</ejb-name>
14       <home>labs.CartHome</home>
15       <remote>labs.Cart</remote>
16       <ejb-class>labs.CartBean</ejb-class>
17       <session-type>Stateful</session-type>
18       <transaction-type>Bean</transaction-type>
19
20       <env-entry>
21    <description>The maximum amount allowed for purchases</description>
22    <env-entry-name>MaxAmount</env-entry-name>
```

```
23    <env-entry-type>java.lang.String</env-entry-type>
24    <env-entry-value>5000</env-entry-value>
25       </env-entry>
26
27       <ejb-ref>
28    <ejb-ref-name>ejb/Customer</ejb-ref-name>
29    <ejb-ref-type>Entity</ejb-ref-type>
30    <home>labs.db.CustomerHome</home>
31    <remote>labs.db.Customer</remote>
32       </ejb-ref>
33
34       <resource-ref>
35    <res-ref-name>jdbc/ShoppingDB</res-ref-name>
36    <res-type>javax.sql.DataSource</res-type>
37    <res-auth>Container</res-auth>
38       </resource-ref>
39
40     </session>
41   </enterprise-beans>
42 </ejb-jar>
```

# Deployer Modifies the DD

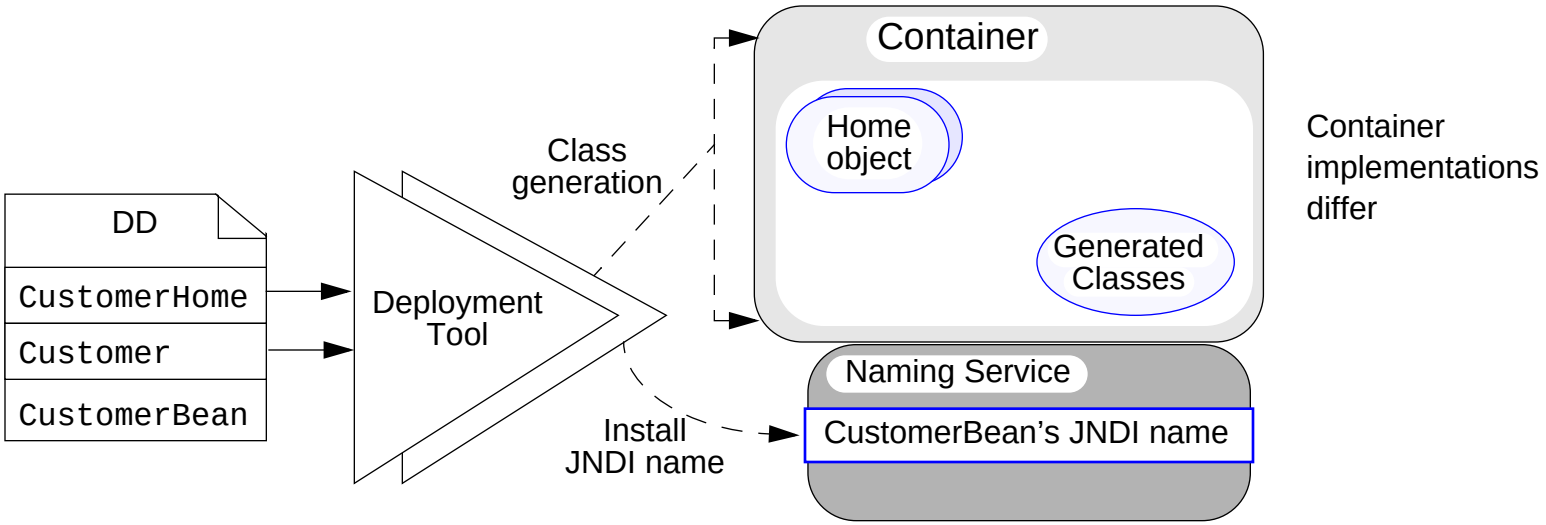
- Deployer must:
  - Map resource references to resource factories in the server
  - Ensure that all environment entries have values
  - Ensure that all `ejb-link` values are provided
- Deployer has additional tasks in the assembly descriptor (security and transaction)

# Deployer Modifies the DD

- Deployer cannot change bean's structural information
  - Bean class
  - Home interface
  - Remote interface
  - State management type
  - Transaction management type (covered later)

# Generating the Home and Remote Classes

- The DD references only home and remote interfaces.
- Each container vendor provides a tool to generate the home and remote classes.



# Generating Stubs and Skeletons

- Generating is probably done at same time as container and EJB object generation.
- Generating creates transport stubs and skeletons (optional) to allow remote access from the client.
- EJB container vendor can generate stubs that use:
  - RMI
  - RMI over IIOP (J2EE)
  - Some other protocol

# Installing an EJB Into the Server

- Every vendor can have a different procedure:
  - Entries in a properties file
  - Graphical administration tool
- Keep track of all classes created during generation to ensure all are installed correctly.
- Install a reference to the home object in the namespace supported by the server.

# Exercises: Deploying the Session Bean

- Objective
- Tasks

# Check Your Progress

- Write a deployment descriptor to describe the structural information for a session bean
- Add environment information to the DD
- Add bean and resource factory references to the DD
- Modify DD to provide assembly information resolving references to external beans and resource factories
- List which aspects of the DD the deployer can and cannot change
- Explain how JNDI is used to access the bean's environment

# Think Beyond

- What are the benefits of defining environment entries, bean references, and resource references in a deployment descriptor?

Hint — You want to make the code reusable (environment entries), or use a different backend data store (resource references).

# Module 6

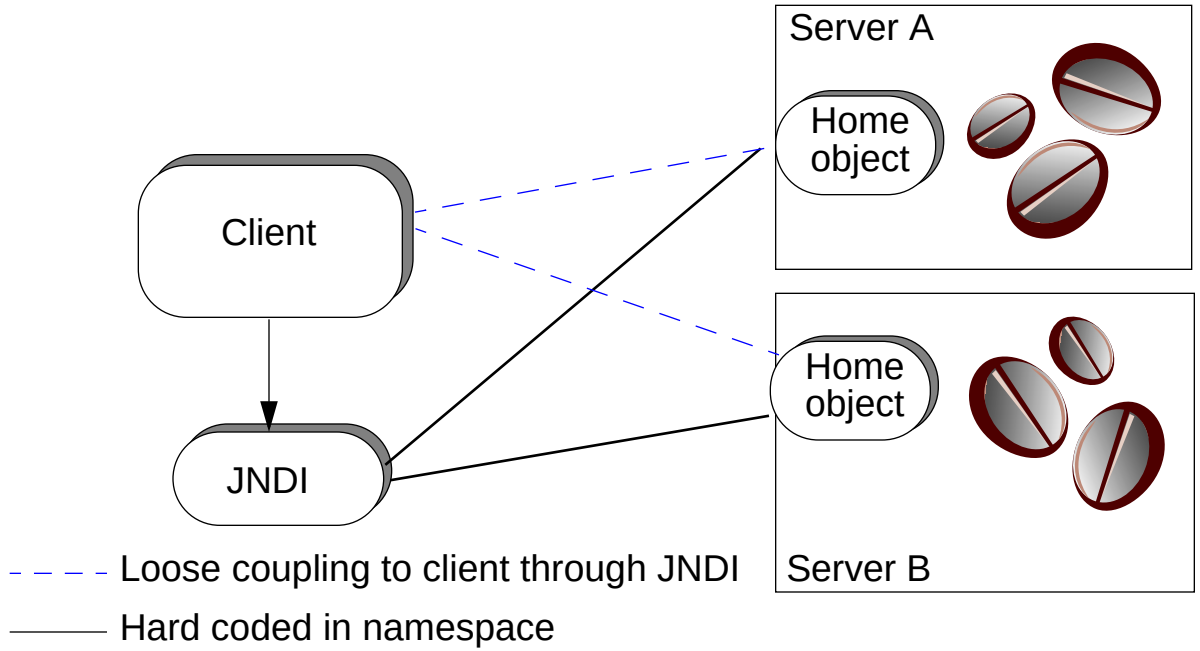
## Writing the EJB Client

# Overview

- Objectives
- Relevance

# Locating Objects With JNDI

- Allows for federation of namespaces
- Can access namespaces on remote machines



# Locating Objects With JNDI

- API to access naming/directory services
- SPI to *any* underlying service provider:
  - CORBA Naming Service
  - LDAP
  - RMI registry
  - Proprietary
- Directory services (attributes) are not used with EJB.

# The Server Namespace

- Server provides a namespace that stores instances of home objects.
- The namespace implementation can vary.
- Location of home in namespace tree must be provided to client through JNDI.
- Objects not accessible using the namespace:
  - EJB object instances
  - Session and entity bean instances
  - Stubs and skeletons

# Using JNDI `InitialContext`

- Place a namespace location and context factory class into a `Hashtable`

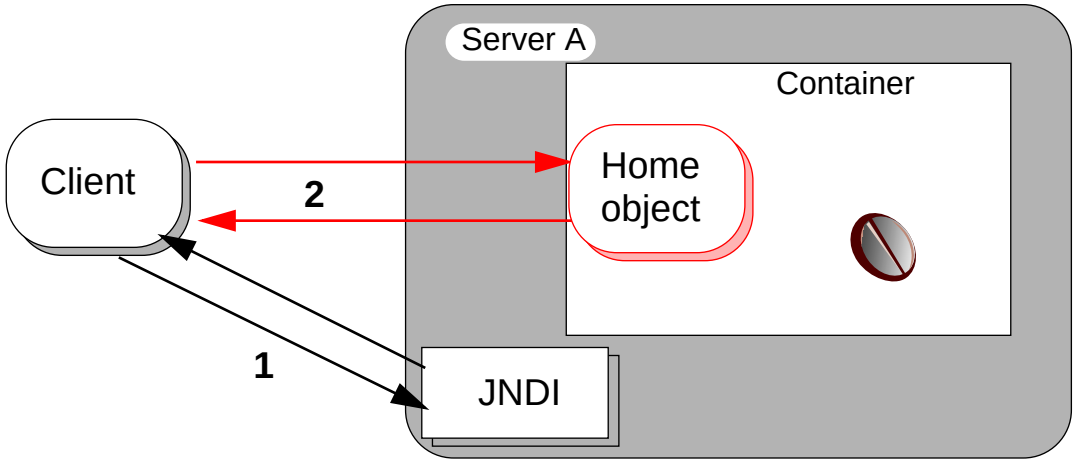
```
1  Hashtable props = new Hashtable();
2
3  props.put(“java.naming.factory.initial”,
4       “vendor.jndi.ContextFactory”);
5
6  props.put(“java.naming.provider.url”,
7       “the.server.com:81”);
```

- Establish a network connection to the root of the namespace

```
9  javax.naming.InitialContext jndiCtx = null;
10
11 try {
12    jndiCtx = new InitialContext(props);
13 } catch (javax.naming.NamingException ne) {}
```

# Getting the Home Object

- Look up the home object instance by name.
- The lookup name was decided at installation time.



# Getting the Home Object

```
1
2 Object objectef = null;
3
4 try {
5
6   Context initial = new InitialContext();
7   objref = initial.lookup("MyCart");
8
9 } catch (javax.naming.NamingException ne) { ...}
10
11try {
12  CartHome home =
13    (CartHome)PortableRemoteObject.narrow(objref,
14      CartHome.class);
15...
16} catch (ClassCastException cce) {
17
```

# PortableRemoteObject

- Package is `javax.rmi.PortableRemoteObject`.
- It is part of the RMI-IIOP set of packages.
- Java technology type-casting of references obtained using JNDI is not supported to be compatible with CORBA objects.

```
1 public class PortableRemoteObject {
2
3   public static Object narrow( Object narrowFrom,
4     Class narrowTo)
5
6     throws
7       ClassCastException;
8   ...
9 }
```

# Creating a Bean Instance

- Create the bean instance using a `create` method defined in the home interface.
- Do not narrow or cast.

```
1 try {
2
3   Cart shoppingCart = home.create("Duke DeEarl","123");
4
5 } catch (CreateException ce) {
6     ... // thrown by ejbCreate
7 }
```

# Invoking Business Methods

```
1  import javax.ejb.*;
2  import javax.rmi.*;
3  import javax.naming.*;
4
5  public class SimpleCartClient {
6
7     SimpleCartHome home;
8     SimpleCart shoppingCart;
9
10    public void startClient(String lookupName) {
11
12     // first the initial lookup
13   try {
14     InitialContext ic = new InitialContext();
15     Object obj = ic.lookup(lookupName);
16     home = (SimpleCartHome) PortableRemoteObject.narrow(obj, SimpleCartHome.class);
17     shoppingCart = home.create("Fran Healy");
18
19   }catch (Exception e) {}
20
21   // now start shopping
22
23   try {
```

```
24    shoppingCart.addBook("Girl in Landscape");
25    shoppingCart.addBook("Selected Poetry of MFE");
26    shoppingCart.addBook("Java for Cats");
27
28    shoppingCart.removeBook("Java for Cats");
29
30   } catch(BookException bex) {
31     System.out.println("caught a BookException: " + bex.getMessage());
32
33   } catch (Exception ex) {
34     System.out.println("caught an unexpected exception!");
35     ex.printStackTrace();
36   }
37
38  finally {
39    try {
40       shoppingCart.remove();
41      } catch (Exception ex) {
42        System.out.println("caught exception trying to remove");
43      }
44    }
45
46   }
47
48   public static void main (String args[]) {
49     new SimpleCartClient().startClient(args[0]);
50   }
```

```
51
52 }// close class
```

# Removing the Bean Instance

- Can remove instance directly:

```
1 try {
2      shoppingCart.remove();
3
4    } catch (RemoveException re1) {
5      ...
6    } catch (RemoteException re2) {
7      ...
8 }
```

- Or through the home interface:

```
9 try {
10   home.remove(handle);
11} catch (RemoveException re1) {
12...
13} catch (RemoteException re2) {
14...
15}
```

# CartClient.java

```
1  import javax.ejb.*;
2  import javax.rmi.*;
3  import javax.naming.*;
4
5  public class SimpleCartClient {
6
7     SimpleCartHome home;
8     SimpleCart shoppingCart;
9
10    public void startClient(String lookupName) {
11
12     // first the initial lookup
13   try {
14     InitialContext ic = new InitialContext();
15     Object obj = ic.lookup(lookupName);
16     home = (SimpleCartHome) PortableRemoteObject.narrow(obj, SimpleCartHome.class);
17     shoppingCart = home.create("Fran Healy");
18
19   }catch (Exception e) {}
20
21   // now start shopping
22
23   try {
24    shoppingCart.addBook("Girl in Landscape");
25    shoppingCart.addBook("Selected Poetry of MFE");
26    shoppingCart.addBook("Java for Cats");
```

```
27
28    shoppingCart.removeBook("Java for Cats");
29
30   } catch(BookException bex) {
31     System.out.println("caught a BookException: " + bex.getMessage());
32
33   } catch (Exception ex) {
34     System.out.println("caught an unexpected exception!");
35     ex.printStackTrace();
36   }
37
38  finally {
39    try {
40       shoppingCart.remove();
41      } catch (Exception ex) {
42        System.out.println("caught exception trying to remove");
43      }
44    }
45
46   }
47
48   public static void main (String args[]) {
49     new SimpleCartClient().startClient(args[0]);
50   }
51
52 }// close class
```

# Exercise: Writing the EJB Client

- Objective
- Task

# Check Your Progress

- Use JNDI to locate the home object
- Create a session bean instance using the home object
- Invoke the bean's business methods
- Pass and return values
- Correctly handle bean exceptions in the client

# Think Beyond

- What types of applications would be best suited to using EJB?

# Module 7

## Introduction to Entity Beans

# Overview

- Objectives
- Relevance

# Purpose of Entity Beans

- Represent underlying data object or context:
  - Record in a database
  - Set of related records in a database
  - Connection to an existing application
- Achieve persistence in different ways
- Have more complex semantics of state management
- Mandatory since EJB 1.1

# Persistence Implementations

- Bean-managed persistence:
  - Code synchronizes state to the underlying object (database row, and so on)
  - Better control for the developer
  - Simpler container/server requirements
- Container-managed persistence:
  - Implicitly synchronizes the state and the object
  - Less code to write, but also less control
  - More complex container/server requirements
  - Standard "flat" mapping (entity class to DB table)

# `EntityBean` Interface

- Implemented by entity beans

```
1  public interface EntityBean extends EnterpriseBean {
2
3     public void ejbActivate() ...;
4     public void ejbPassivate() ...;
5
6     public void setEntityContext(EntityContext ec) ...;
7     public void unsetEntityContext() ...;
8     public void ejbLoad() ...;
9     public void ejbStore() ...;
10
11    public void ejbRemove() ...;
12
13 }
```

- No equivalent interface for `SessionSynchronization`

# `EntityContext` Interface

- The interface extends `javax.ejb.EJBContext`.
- Like `SessionContext`, references to objects within `EntityContext` should not be maintained.
- `unsetEntityContext` called when deleting an EJB instance from the free pool.

```
1  public interface EntityContext extends EJBContext {
2
3     public EJBObject getEJBObject() throws
4        IllegalStateException;
5
6     public Object getPrimaryKey() throws
7        IllegalStateException;
8  }
```

# Loading and Storing

- `void ejbLoad()`

  Initializes the EJB's state from data in the database:

  - After assigning the EJB instance to its EJB object
  - At any point by the container

- `void ejbStore()`

  Saves the EJB's data to the database:

  - Prior to passivating the EJB instance
  - At any point by the container

- Implementations vary depending on BMP or CMP.

# Primary Key

- Any type that is `Serializable` or `Remote`
  - Can not be a primitive.
  - Must implement `hashcode` and `equals`.
  - Example — `java.lang.String`
- The primary key uniquely identifies the entity instance within its home.
- Clients can obtain the primary key of an entity instance and save it for later use.

# Primary Key

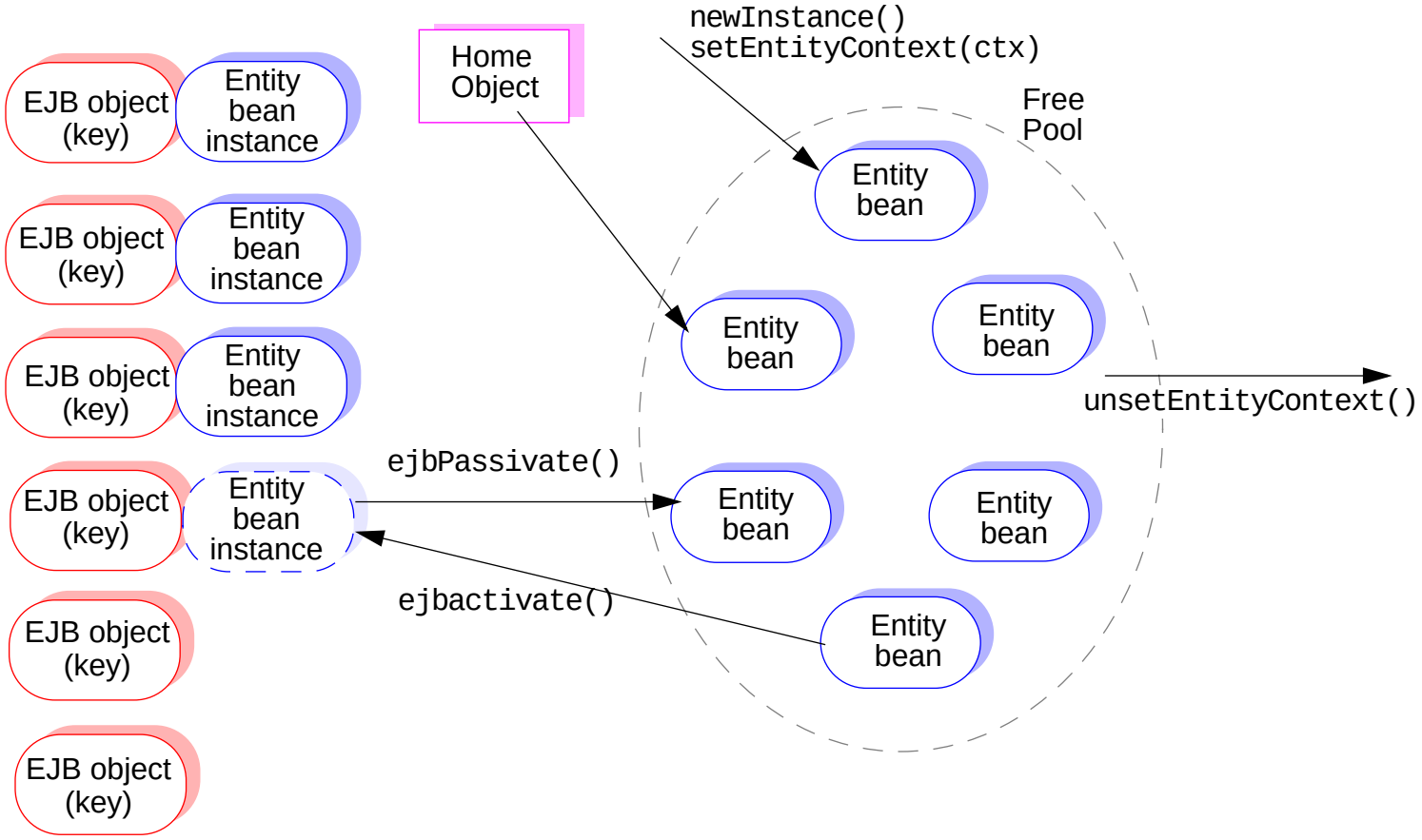
- Primary purpose is for synchronizing with database.
- Primary key is stored in EJB object, obtained via context.
- Home interface must declare `findByPrimaryKey` method.

```
1  // In the client
2  ...
3  MyBeanKey key = beanRef.getPrimaryKey();
```

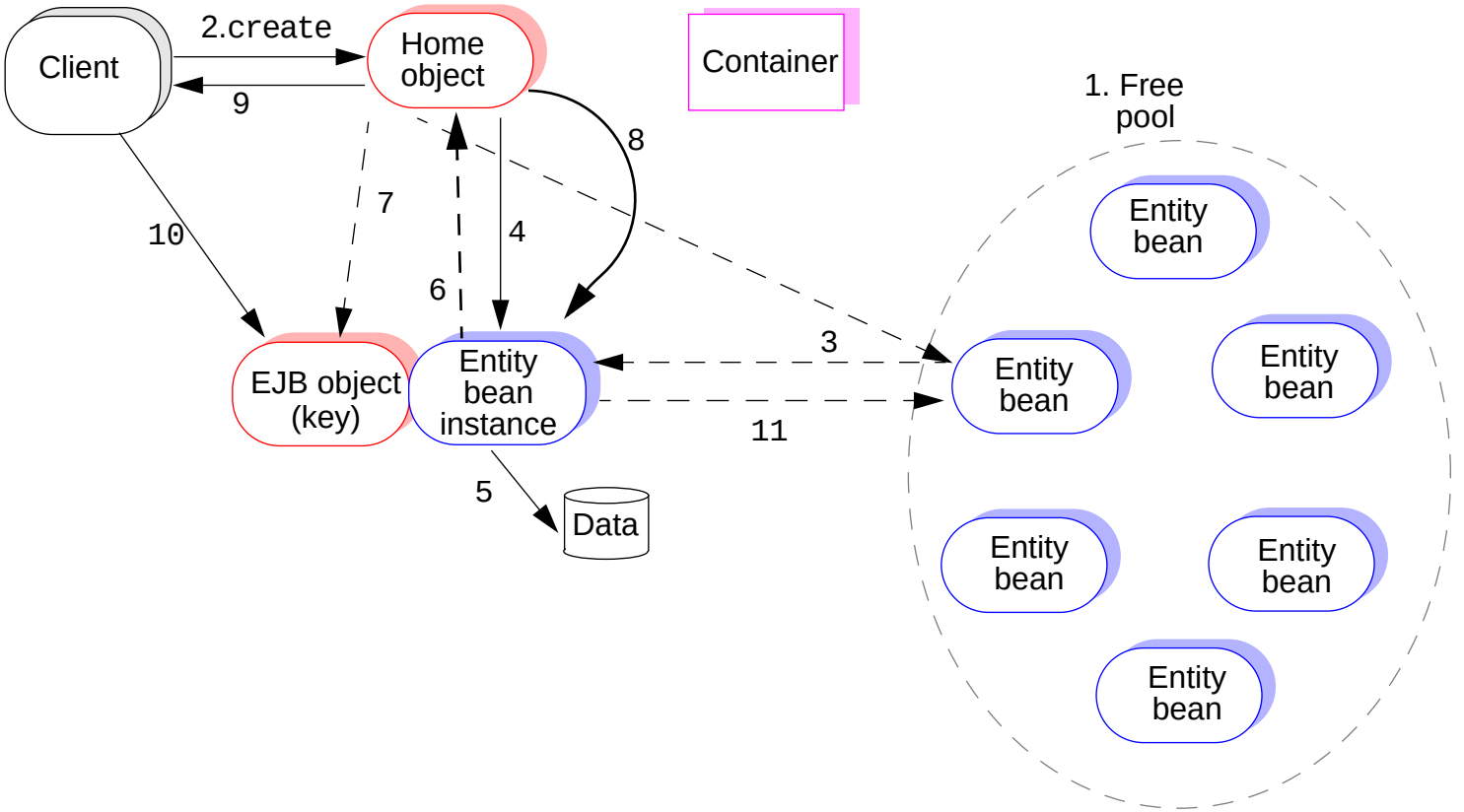
```
1  // Elsewhere, in a client
2  ...
3  beanRef = home.findByPrimaryKey(key);
```

```
1  // In the home interface
2  ...
3  public <remoteRef> findByPrimaryKey(Object key) ...;
```

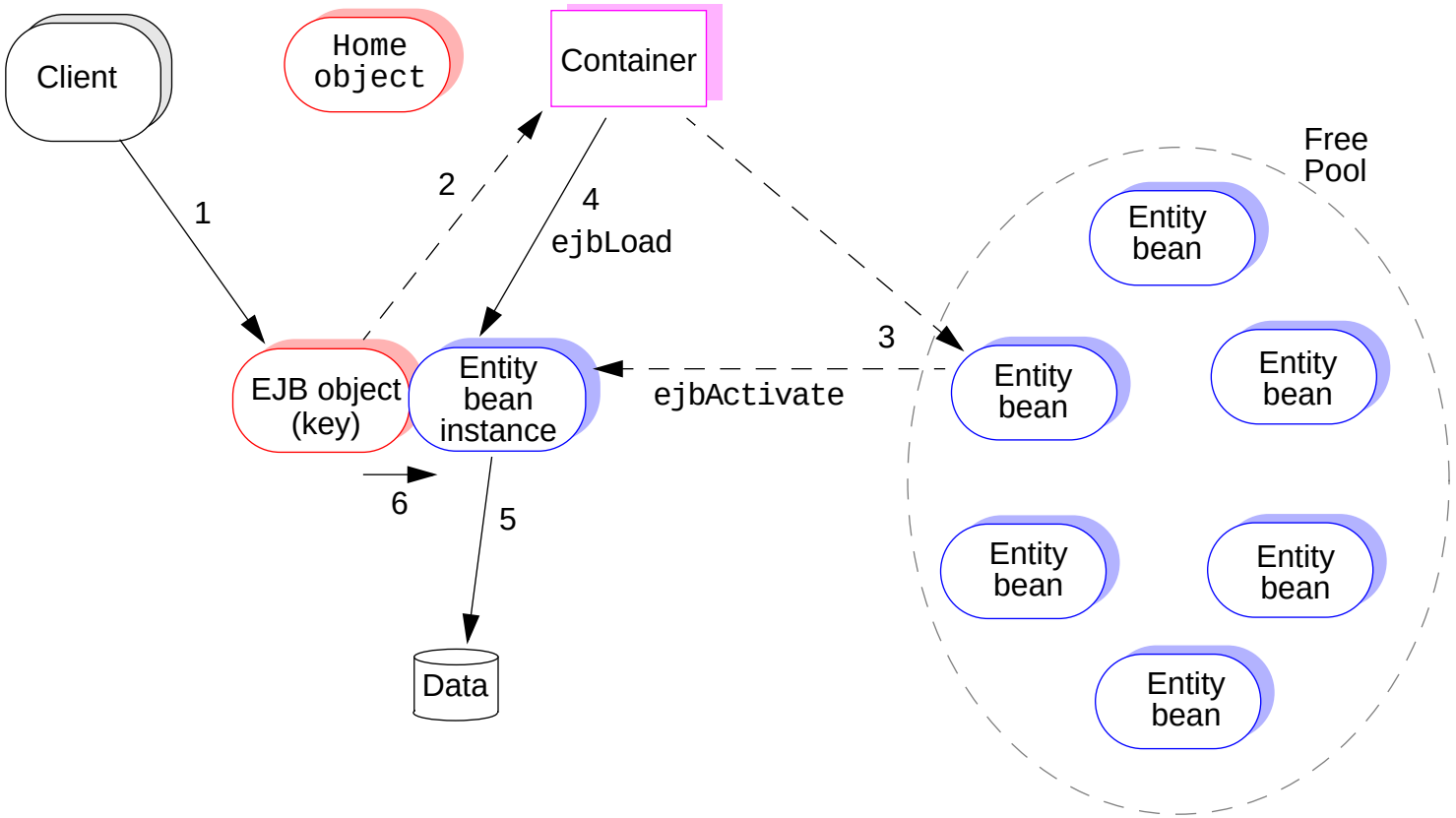
# Overview of the Entity Bean Runtime Architecture



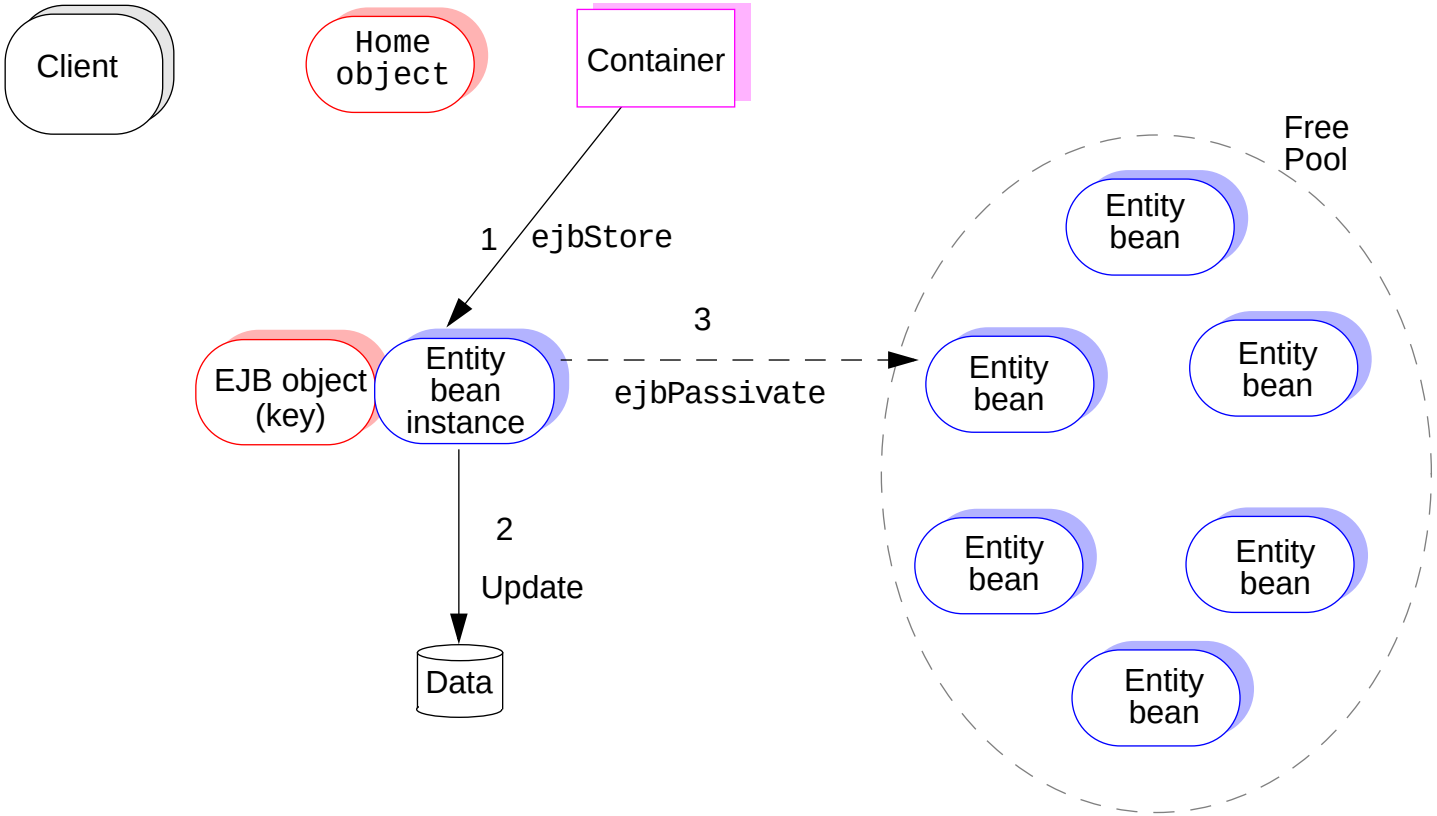
# Creating Entity Beans



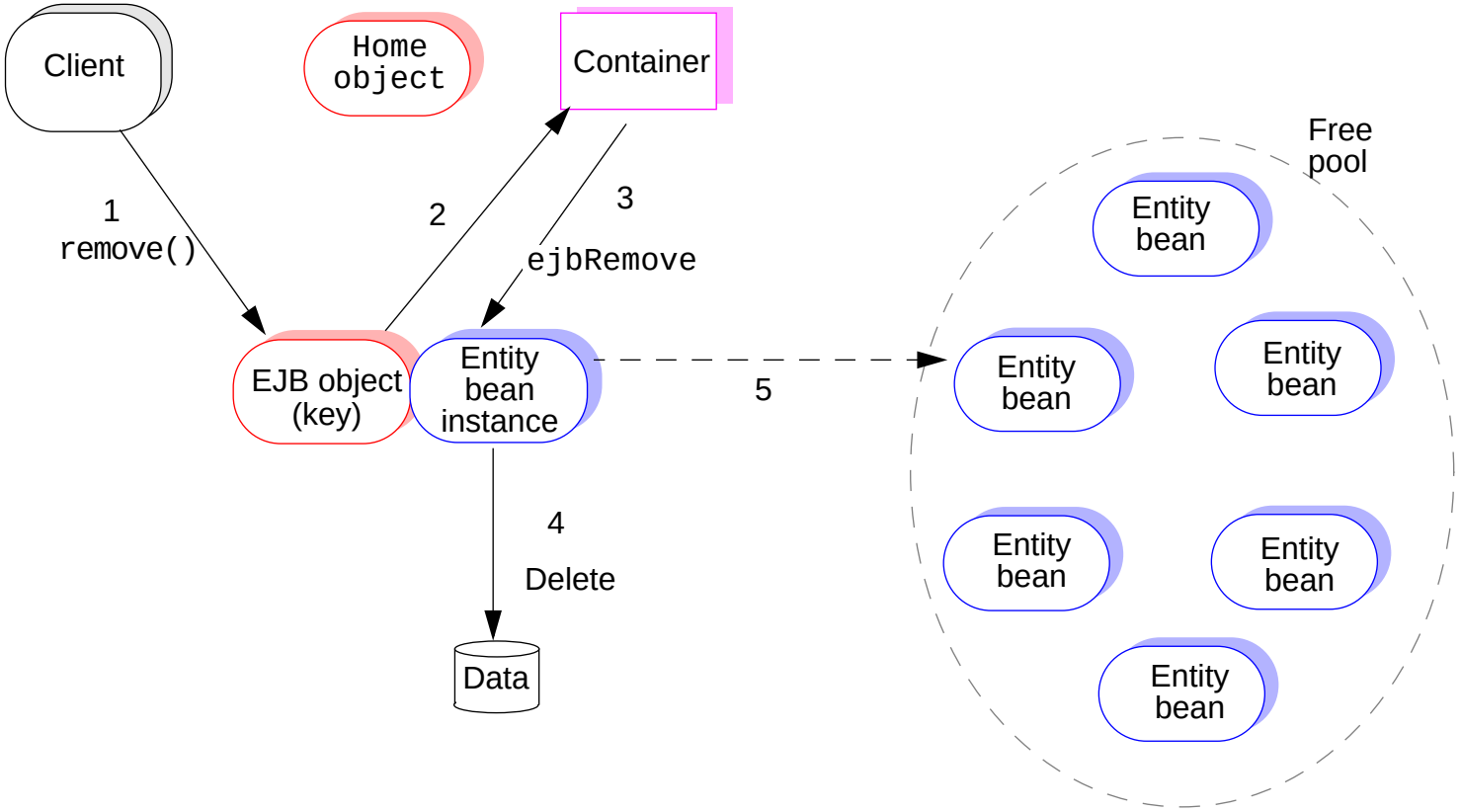
# Invoking Entity Beans



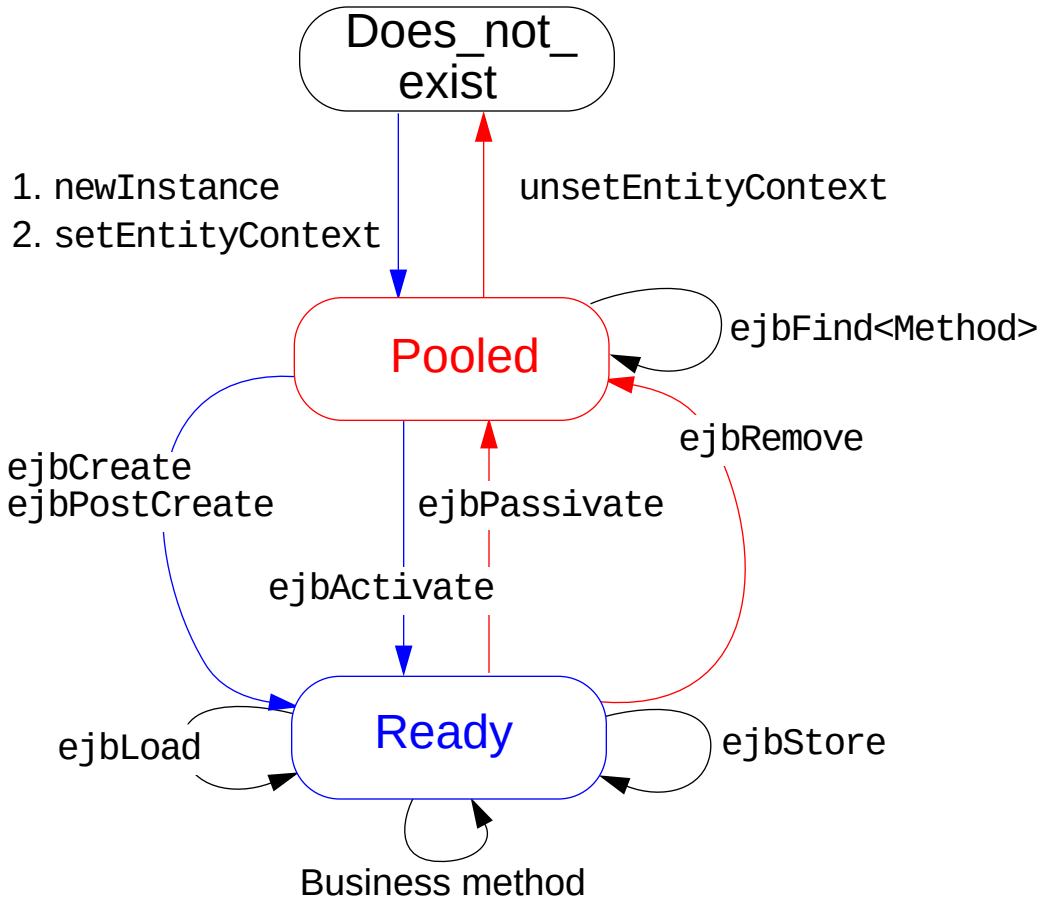
# Passivating Entity Beans



# Removing Entity Beans



# Life Cycle of an Entity Instance



# Recap

| Interfaces | Methods |
|---|---|
| EntityBean | ejbRemove() |
| | ejbActivate() |
| | ejbPassivate() |
| | setEntityContext(sc) |
| | unsetEntityContext() |
| | ejbLoad() |
| | ejbStore() |
| EJBContext | getEJBHome() |
| | getCallerPrincipal() |
| | isCallerInRole (String role) |
| | getUserTransaction |
| | getRollbackOnly() |
| | setRollbackOnly() |
| EntityContext | getEJBObject() |
| | getPrimaryKey() |

# Check Your Progress

- List and explain the two persistence management techniques
- List the additional methods in the `EntityBean` interface, and explain their purpose
- Describe what it means to load and store a bean
- Explain the requirements for defining a primary key

# Think Beyond

- Discuss what stages there must be in the life cycle of an entity bean.

# Module 8

## Bean-Managed Persistence

# Overview

- Objectives
- Relevance

# SQL Query Overview

- INSERT

```
INSERT INTO emp_details ( first_name, last_name, dept, salary)
    VALUES ('Larry', 'Sanderson', 'I.L.T.','12345')
```

- SELECT

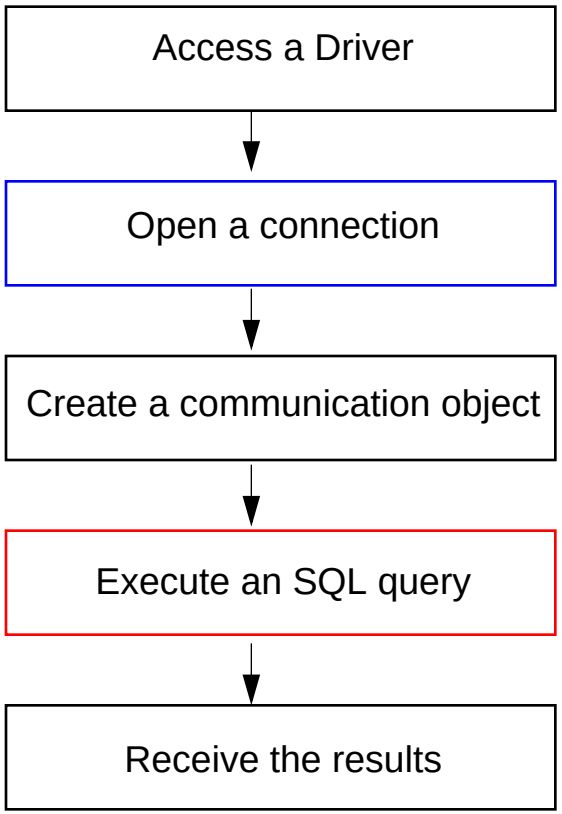
```
SELECT last_name FROM emp_details WHERE dept = 'finance'
```

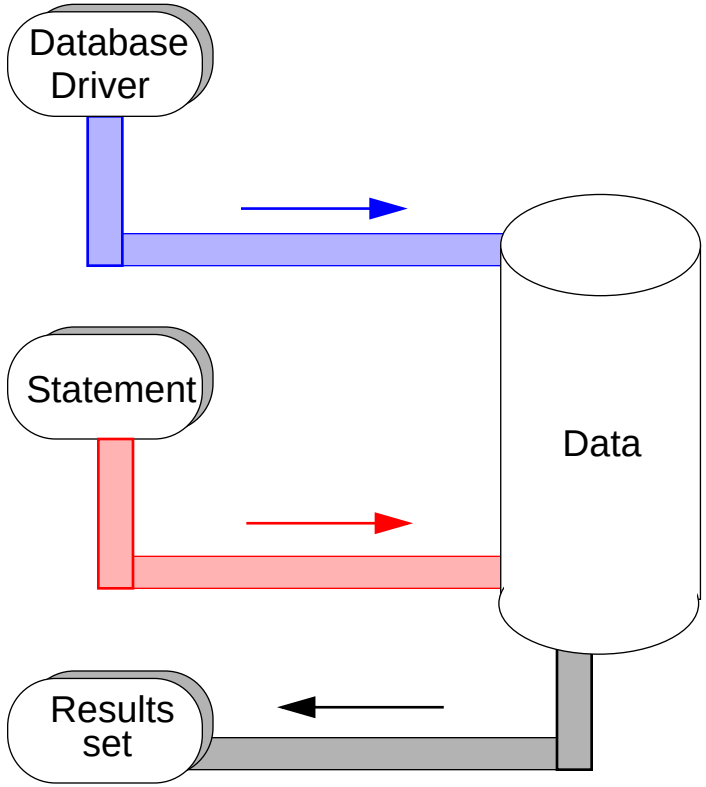
- DELETE

```
DELETE FROM emp_details WHERE emp_id = ’12345’
```

# JDBC Overview

- `DriverManager`



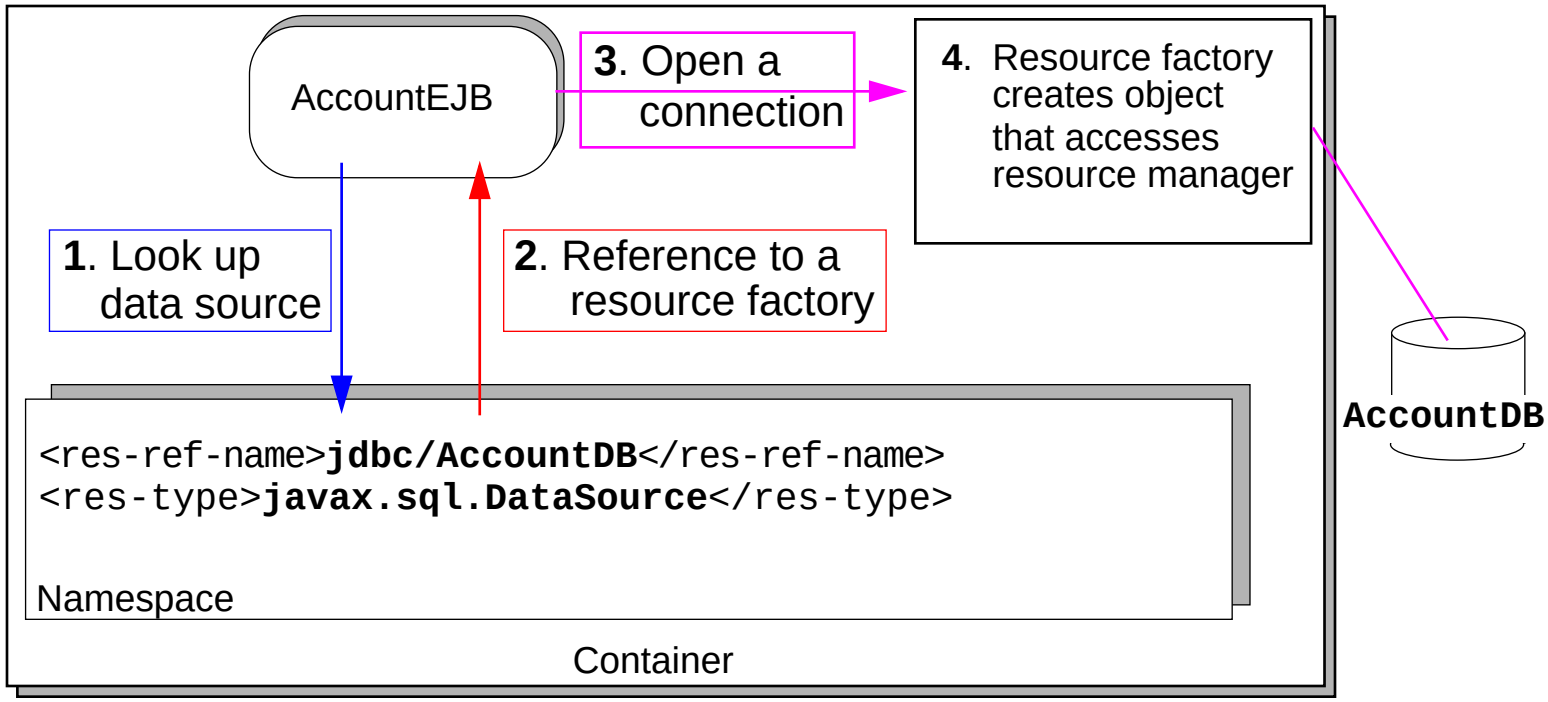
# JDBC Overview

- `DataSource`
- Preferred way to open a connection
  - Can offer automatic connection pooling and distributed transactions
  - Provides consistent access through JNDI for better portability
- Required for J2EE 1.2

# Access to a Data Source

- JDBC connections obtained through `javax.sql.DataSource` (resource factory)
- Container provides resource factory classes

# Access to a Data Source



# Benefits of BMP

- Typically uses simple JDBC calls
- Allows you flexibility in accessing database
- Does not require complex support from container
- Simpler to debug, because you are familiar with your code

# Defining `ejbCreate`

- `ejbCreate` inserts a row into the database.
- You must write the JDBC code directly.
- Multiple methods can be defined.
- Every method should provide enough arguments to satisfy the primary key.
- Method must return the primary key identifying the row to the container.
  - You can use any representation that is unique.
  - Container needs primary key to initialize EJB object.

# Defining `ejbCreate`

```
1  public String ejbCreate(String key, String name, String address) throws CreateException {
2
3   if (key == null) {
4        throw new CreateException ("can't create with null ID");
5    }
6
7    this.name = name;
8    this.address = address;
9    this.key = key;
10
11   Connection conn = null;
12   PreparedStatement stmt = null;
13   try {
14      conn = ds.getConnection();
15      stmt = conn.prepareStatement(
16      "INSERT INTO STUDENTS (STUDENTID, NAME, ADDRESS) VALUES (?, ?, ?)");
17
18      stmt.setString(1, key);
19      stmt.setString(2, name);
20      stmt.setString(3, address);
21      stmt.executeUpdate();
22
23     } catch (SQLException sqle) {
24       throw new EJBException("SQLException while creating record: " + sqle.getMessage());
25     } catch (Exception e) {
26       throw new CreateException("unable to create");
```

```
27    } finally {
28      if (conn != null) {
29       try {
30         conn.close();
31
32     } catch (SQLException e) {
33       throw new EJBException("exception while closing connection");
34      }
35     }
36   }
37
38 // Return the primary key of the new row to the container
39   return key;
40 }
```

# Defining `ejbPostCreate`

- Called after:
  - `ejbCreate(...)` called and returned to home, and
  - Home has created EJB object
- Bean performs creation work with a valid EJB object
- Same arguments passed to `ejbPostCreate` as `ejbCreate`.
- Matching `ejbPostCreate` method for each `ejbCreate` method.

```
1 public void ejbPostCreate(String ssn) {
2 ...
3 }
```

# Defining `ejbRemove`

- Deletes a row from the database
- Must obtain the primary key from the EJB object (through the context)

# Defining ejbRemove

```
1  public void ejbRemove() {
2    Connection conn = null;
3    PreparedStatement stmt = null;
4
5     try {
6       String key = (String) ctx.getPrimaryKey();
7       conn = ds.getConnection();
8       stmt = conn.prepareStatement(
9         "DELETE FROM STUDENTS WHERE STUDENTID = ?");
10       stmt.setString(1, key);
11       stmt.executeUpdate();
12        } catch (SQLException e) {
13          throw new EJBException("unable to delete");
14
15        } finally {
16        if (conn != null) {
17          try {
18            conn.close();
19        } catch (SQLException e) {
20       throw new EJBException("exception while closing  connection");
21         }
22        }
23     }
24
25     name = null;
26     address = null;
27     key = null;
28   }
```

# Defining `ejbLoad`

- Loads data from row in database

```
1  public void ejbLoad() {
2    key = (String) ctx.getPrimaryKey();
3    Connection conn = null;
4    PreparedStatement stmt = null;
5
6      try {
7       conn = ds.getConnection();
8       stmt = conn.prepareStatement(
9           "SELECT NAME, ADDRESS FROM STUDENTS WHERE STUDENTID = ?");
10      stmt.setString(1, key);
11      ResultSet rs = stmt.executeQuery();
12      if (rs.next()) {
13        name     = rs.getString(1).trim();
14        address  = rs.getString(2).trim();
15        }
16      else {
17           throw new NoSuchEntityException("Row not found: " + key);
18      }
19      } catch (SQLException e) {
20         throw new EJBException("exception during ejbLoad");
21      } finally {
22         if (conn != null) {
23           try {
24             conn.close();
25           } catch (SQLException e) {
26          throw new EJBException("exception while closing
connection");
27           }
28         }
29    }
30 }
```

# Defining `ejbStore`

- Writes field values into database

```
1  public void ejbStore() {
2   Connection conn = null;
3   PreparedStatement stmt = null;
4
5   try {
6     conn = ds.getConnection();
7     stmt = conn.prepareStatement(
8      "UPDATE STUDENTS SET NAME = ?, ADDRESS = ? WHERE STUDENTID =
?");
9     stmt.setString(1, name);
10    stmt.setString(2, address);
11    stmt.setString(3, key);
12    stmt.executeUpdate();
13
14    } catch (SQLException e) {
15       throw new EJBException("unable to update database");
16
17    } finally {
18       if (conn != null) {
19         try {
20           conn.close();
21         } catch (SQLException e) {
22        throw new EJBException("exception while closing connection");
23         }
24       }
25    }
26 }
27
```

# StudentBean.java

```
1 package bmplabs;
2
3 import javax.naming.*;
4 import javax.ejb.*;
5 import javax.sql.*;
6 import java.sql.*;
7 import java.io.*;
8 import java.util.*;
9
10
11public class StudentBean implements EntityBean {
12
13    private String name;
14    private String address;
15    private String key;
16
17    private EntityContext ctx;
18    private DataSource ds;
19    private DataAccessHelper helper = new
DataAccessHelper();
20
21//=====================================================
22      public void ejbActivate() {}
23//=====================================================
24      public String ejbCreate(String key, String name,
String address) throws CreateException {
25
26           if (key == null) {
27             throw new CreateException ("can't create with
null ID");
```

```
28          }
29          this.name = name;
30          this.address = address;
31          this.key = key;
32
33          Connection conn = null;
34          PreparedStatement stmt = null;
35
36           try {
37             conn = ds.getConnection();
38             stmt = conn.prepareStatement(
39           "INSERT INTO STUDENTS (STUDENTID, NAME,
ADDRESS) VALUES (?, ?, ?)");
40           stmt.setString(1, key);
41           stmt.setString(2, name);
42           stmt.setString(3, address);
43
44             stmt.executeUpdate();
45
46       } catch (SQLException sqle) {
47            throw new EJBException("SQLException while
creating record: " + sqle.getMessage());
48       } catch (Exception e) {
49               throw new CreateException("unable to
create");
50        } finally {
51             if (conn != null) {
52              try {
53                conn.close();
54              } catch (SQLException e) {
55                 throw new EJBException("exception while
closing connection");
56              }
57             }
58           }
```

```
59
60
61        // Return the primary key of the new row to the
container
62        return key;
63      }
64
65 //
==========================================================
66        public String ejbFindByPrimaryKey(String key)
throws FinderException {
67
68           Connection conn = null;
69           PreparedStatement stmt = null;
70
71           try {
72              conn = ds.getConnection();
73             stmt = conn.prepareStatement(
74             "SELECT STUDENTID FROM STUDENTS WHERE
STUDENTID = ?");
75
76            stmt.setString(1, key);
77            ResultSet rslt = stmt.executeQuery();
78
79            if (rslt.next()) {
80              return key;
81            } else {
82                throw new ObjectNotFoundException();
83            }
84            } catch (SQLException e) {
85                throw new EJBException("unable to do the
find");
86            } finally {
87                if (conn != null) {
88                 try {
```

```
89                  conn.close();
90                  } catch (SQLException e) {
91                    throw new EJBException("exception
while closing connection");
92                  }
93               }
94            }
95
96      }
97
98//=====================================================
99
100        public Collection ejbFindAllStudents() throws
FinderException {
101
102           Connection conn = null;
103           PreparedStatement stmt = null;
104
105           try {
106              conn = ds.getConnection();
107             stmt = conn.prepareStatement(
108            "SELECT STUDENTID FROM STUDENTS");
109            ResultSet rslt = stmt.executeQuery();
110
111            ArrayList al = new ArrayList();
112            while (rslt.next()) {
113                al.add(rslt.getString(1).trim());
114            }
115
116            return al;
117
118            } catch (SQLException e) {
119                throw new EJBException("unable to do the
find");
120            } finally {
```

```
121                if (conn != null) {
122                 try {
123                  conn.close();
124                  } catch (SQLException e) {
125                    throw new EJBException("exception
while closing connection");
126                  }
127                }
128            }
129
130      }
131
132//
==========================================================
133
134   public void ejbLoad() {
135
136         key = (String) ctx.getPrimaryKey();
137
138          Connection conn = null;
139          PreparedStatement stmt = null;
140
141         try {
142              conn = ds.getConnection();
143              stmt = conn.prepareStatement(
144             "SELECT NAME, ADDRESS FROM STUDENTS WHERE
STUDENTID = ?");
145
146              stmt.setString(1, key);
147              ResultSet rslt = stmt.executeQuery();
148
149             if (rslt.next()) {
150                name     = rslt.getString(1).trim();
151                address  = rslt.getString(2).trim();
152            }
```

```
153            else {
154
155                throw new NoSuchEntityException("Row not
found: " + key);
156            }
157
158        } catch (SQLException e) {
159                throw new EJBException("exception during
ejbLoad");
160            } finally {
161                if (conn != null) {
162                 try {
163                  conn.close();
164                 } catch (SQLException e) {
165                    throw new EJBException("exception
while closing connection");
166                 }
167               }
168            }
169    }
170//
============================================================
171
172      public void ejbPassivate() {
173        address = null;
174        name = null;
175        key = null;
176      }
177//
============================================================
178     public void ejbPostCreate(String key, String
address, String name) { }
179//
============================================================
180   public void ejbRemove() {
```

```
181
182       Connection conn = null;
183       PreparedStatement stmt = null;
184
185       try {
186              String key = (String) ctx.getPrimaryKey();
187              conn = ds.getConnection();
188              stmt = conn.prepareStatement(
189             "DELETE FROM STUDENTS WHERE STUDENTID = ?");
190              stmt.setString(1, key);
191
192              stmt.executeUpdate();
193
194             } catch (SQLException e) {
195                 throw new EJBException("unable to
delete");
196             } finally {
197                 if (conn != null) {
198                  try {
199                   conn.close();
200                  } catch (SQLException e) {
201                     throw new EJBException("exception
while closing connection");
202                  }
203                }
204             }
205
206           name = null;
207           address = null;
208           key = null;
209       }
210//
=====================================================
211    public void ejbStore() {
212
```

```
213          Connection conn = null;
214          PreparedStatement stmt = null;
215
216          try {
217              conn = ds.getConnection();
218              stmt = conn.prepareStatement(
219            "UPDATE STUDENTS SET NAME = ?, ADDRESS = ?
WHERE STUDENTID = ?");
220
221            stmt.setString(1, name);
222            stmt.setString(2, address);
223            stmt.setString(3, key);
224
225           stmt.executeUpdate();
226
227            } catch (SQLException e) {
228                throw new EJBException("unable to update
database");
229            } finally {
230                if (conn != null) {
231                 try {
232                  conn.close();
233                 } catch (SQLException e) {
234                    throw new EJBException("exception
while closing connection");
235                 }
236                }
237            }
238
239      }
240//
=====================================================
241      public String getID() {
242        return key;
243    }
```

```
244//
==========================================================
245       public String getName() {
246        return name;
247    }
248//
==========================================================
249      public String getAddress() {
250        return address;
251    }
252//
==========================================================
253    public void setAddress(String address) {
254        this.address = address;
255    }
256//
==========================================================
257      public void setName(String name) {
258        this.name = name;
259    }
260
261//
==========================================================
262    public void setEntityContext(EntityContext ctx) {
263       this.ctx = ctx;
264       try {
265          InitialContext ic = new InitialContext();
266          ds = (DataSource) ic.lookup("java:comp/env/
jdbc/StudentDB");
267       } catch (Exception e) {
268           throw new EJBException("unable to get
DataSource");
269       }
270
271    }
```

```
272//=======================================================
273    public void unsetEntityContext() { }
274  }
275}
276
```

# StudentHome.java

```
1 package bmplabs;
2
3 import javax.ejb.*;
4 import java.rmi.*;
5
6 public interface StudentHome extends EJBHome {
7
8     public Student create(String key, String name, String address) throws RemoteException,
CreateException;
9
10    public Student findByPrimaryKey(String key) throws RemoteException, FinderException;
11
12    public Collection findAllStudents() throws RemoteException, FinderException;
13
14  }
15
```

# Student.java

```
1  package bmplabs;
2
3  import javax.ejb.*;
4  import java.rmi.*;
5
6  public interface Student extends EJBObject {
7
8     public String getID() throws RemoteException;
9
10    public String getName() throws RemoteException;
11
12    public String getAddress() throws RemoteException;
13
14    public void setName(String name) throws RemoteException;
15
16    public void setAddress(String address) throws RemoteException;
17}
18
```

# Exercise: Bean-Managed Persistence

- Objective
- Tasks

# Module 9

## Defining Finder Methods

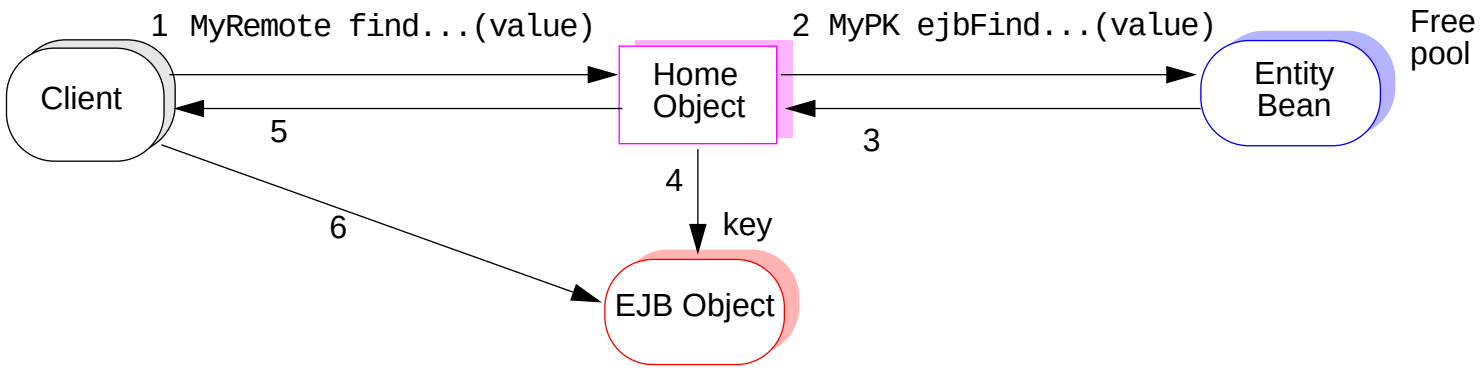
# Overview

- Objectives
- Relevance

# Understanding Finder Methods

- A finder represents a specific SELECT statement.
- Multiple finders can be defined for a bean class.
- Finders:
  - Are declared in home interface as `find<name>`
  - Are implemented in Bean as `ejbFind<name>`
  - Allow a client to look up one or more rows based on search criteria

# Single-Row Finders



# Implementing a Single-Row Finder

- Defined in the home to return a single remote reference

```
1 public Student findByPrimaryKey(String key) throws RemoteException,
2    FinderException;
```

- Defined in the bean to return a single primary key

```
1 public String ejbFindByPrimaryKey(String key) throws FinderException {
2 // code to verify that the entity with this PK exists in DB
3 ...
4 return key;
5 }
```

# Implementing a Single-Row Finder

- Defined in the bean to return a single primary key

```
1  public String ejbFindByPrimaryKey(String key) throws FinderException {
2    Connection conn = null;
3    PreparedStatement stmt = null;
4    try {
5       conn = ds.getConnection();
6       stmt = conn.prepareStatement(
7         "SELECT STUDENTID FROM STUDENTS WHERE STUDENTID = ?");
8       stmt.setString(1, key);
9       ResultSet rslt = stmt.executeQuery();
10
11       if (rslt.next()) {
12         return key;
13        } else {
14         throw new ObjectNotFoundException();
15        }
16     } catch (SQLException e) {
17         throw new EJBException("unable to do the find");
18     } finally {
19        if (conn != null) {
20           try {
21             conn.close();
22              } catch (SQLException e) {
23                throw new EJBException("exception while closing connection");
```

```
24                  }
25              }
26          }
27 }
```

# Primary Key Finder Method

- Define it in home interface (required).
- All BMP beans must implement this method.
- It verifies that this row exists in the table.

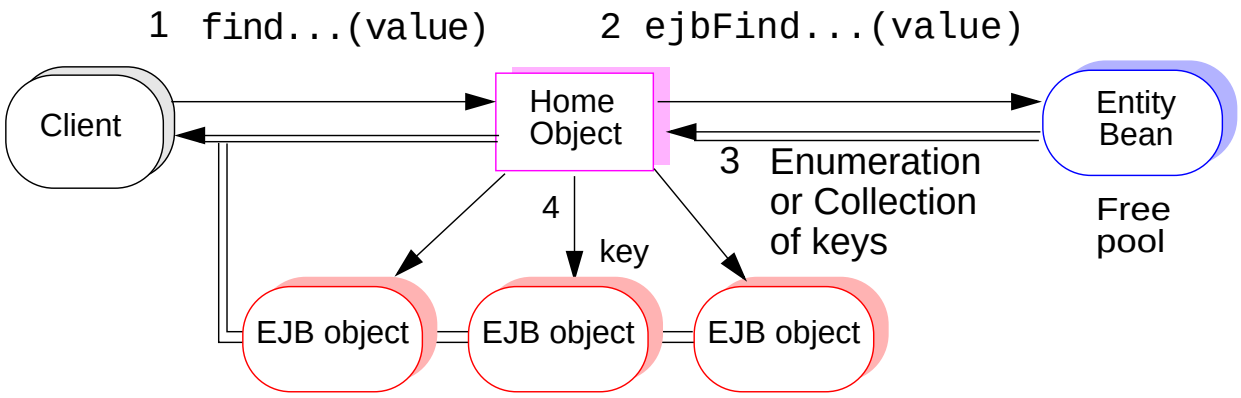
# Primary Key Finder Method

```
1 public Student findByPrimaryKey(String key) throws RemoteException,
2    FinderException;
```

```
1 public String ejbFindByPrimaryKey(String key) throws FinderException {
2   Connection conn = null;
3   PreparedStatement stmt = null;
4   try {
5      conn = ds.getConnection();
6      stmt = conn.prepareStatement(
7        "SELECT STUDENTID FROM STUDENTS WHERE STUDENTID = ?");
8      stmt.setString(1, key);
9      ResultSet rslt = stmt.executeQuery();
10
11     if (rslt.next()) {
12       return key;
13      } else {
14       throw new ObjectNotFoundException();
15      }
```

```
16    } catch (SQLException e) {
17        throw new EJBException("unable to do the find");
18    } finally {
19       if (conn != null) {
20          try {
21            conn.close();
22             } catch (SQLException e) {
23               throw new EJBException("exception while closing connection");
24              }
25          }
26       }
27 }
28 }
```

# Multiple-Row Finders



# Implementing a Multiple-Row Finder

- Bean's home returns an enumeration or collection of remote references.
- Bean returns an enumeration or collection of keys.

```
1    public Collection ejbFindAllStudents() throws FinderException {
2   Connection conn = null;
3   PreparedStatement stmt = null;
4
5   try {
6      conn = ds.getConnection();
7      stmt = conn.prepareStatement(
8        "SELECT STUDENTID FROM STUDENTS");
```

```
9       ResultSet rslt = stmt.executeQuery();
10      ArrayList al = new ArrayList();
11      while (rslt.next()) {
12         al.add(rslt.getString(1).trim());
13       }
14      return al;
15     } catch (SQLException e) {
16         throw new EJBException("unable to do the find");
17     } finally {
18       if (conn != null) {
19         try {
20           conn.close();
21         } catch (SQLException e) {
22          throw new EJBException("exception while closing connection");
23         }
24       }
25      }
26    }
```

# Exercise: Defining Finder Methods

- Objective
- Tasks

# Check Your Progress

- Define finder methods in your bean
- Declare finder methods in your home interface
- Define finder methods that return single rows and those that return multiple rows
- Explain the reason for different return types for the finder methods in the bean and in the home interface

# Think Beyond

- How are finder methods implemented in a CMP bean?

# Module 10

## Container-Managed Persistence

# Overview

- Objectives
- Relevance

# Benefits of Container-Managed Persistence

- Very little work is done by bean provider.
- Deployer maps bean fields to table columns.
- CMP might provide better portability.
- CMP might provide better database optimizations:
  - Implicit use of cursors
  - Optimized queries
  - Customizable load/store policies
  - Customizable pre-fetch policies

# Primary Key With CMP

- Primary key type must be a legal value type.
- Primary key can map to a single field or multiple fields (compound keys).
- Special situations may require that the deployer (not the bean provider) specifies a primary key.

# Defining `ejbCreate`

- Place parameters into public fields.
- Returning null still allows a subclass to override the method to provide BMP.

# Defining `ejbCreate`

```
1  public class ProductEJB implements EntityBean {
2
3     public String productId;
4     public String description;
5     public double price;
6
7     private EntityContext context;
8
9     public String ejbCreate(String productId, String description,
10       double price) throws CreateException {
11
12       if (productId == null) {
13          throw new CreateException("The productId is required.");
14       }
15
16       this.productId = productId;
17       this.description = description;
18       this.price = price;
19
20        return null;
21    }
22    ...
23 }
```

# Defining `ejbRemove`

- Nothing needs to be done.
- This method can be used when implementing object-relational mapping.

```
1  public class ProductEJB implements EntityBean {
2
3     public String productId;
4     public String description;
5     public double price;
6
7     public void ejbRemove() {
8
9        // Should set each string to null to speed up garbage
10       // collection. Setting them to null is not required.
11    }
12    ...
13 }
```

# Defining `ejbLoad`

- Perform initial adjustment to the data.
- When the client reads the product ID field, it will be formatted with dashes.

```
1  public class ProductEJB implements EntityBean {
2
3     public String productId;
4     public String description;
5     public double price;
6
7     public void ejbLoad() {
8        this.productID = insertDashes(this.productID);
9     }
10
11 ...
12 }
```

# Defining `ejbStore`

- Get the data ready for the database.
- Product ID will go into the database without dashes.

```
1  public class ProductEJB implements EntityBean {
2
3     public String productId;
4     public String description;
5     public double price;
6
7     public void ejbStore() {
8        this.productID = removeDashes(this.productID);
9     }
10
11 ...
12 }
```

# Finder Methods

- Finder methods are declared in the home interface.
- There are *no* corresponding `ejbFind` methods in the bean.
- Container vendor should provide a tool that allows developer/deployer to describe how to create finder methods.
  - Argument-to-column mapping table
  - Finder method name
  - Other criteria if necessary, such as ranges

# Declaring CMP in the DD

```
1  <ejb-jar>
2     <enterprise-beans>
3        <entity>
4        ...
5        <persistence-type> Container </persistence-type>
6        <prim-key-class> java.lang.String </prim-key-class>
7
8        <cmp-field>
9           <description> the product ID </description>
10          <field-name> productID </field-name>
11       </cmp-field>
12
13       <cmp-field>
14          <description> the product description </description>
15          <field-name> description </field-name>
16       </cmp-field>
17
18       <cmp-field>
19          <description> the product price </description>
20          <field-name> price </field-name>
21       </cmp-field>
22
23       <primkey-field> productID </primkey-field>
24
25    </entity>
26    ...
27    </enterprise-beans>
28 </ejb-jar>
```

# ProductEJB.java

```
1  import java.util.*;
2  import javax.ejb.*;
3
4  public class ProductEJB implements EntityBean {
5
6     public String productId;
7     public String description;
8     public double price;
9
10    private EntityContext context;
11
12    public void setPrice(double price) {
13
14       this.price = price;
15    }
16
17    public double getPrice() {
18
19       return price;
20    }
21
22    public String getDescription() {
23
24       return description;
25    }
26
27    public String ejbCreate(String productId, String description,
28       double price) throws CreateException {
29
30       if (productId == null) {
31          throw new CreateException("The productId is required.");
32       }
33
34       this.productId = productId;
35       this.description = description;
36       this.price = price;
37
38       return null;
39    }
```

# ProductEJB.java

```
40   public void setEntityContext(EntityContext context) {
41
42      this.context = context;
43   }
44
45   public void ejbActivate() { }
46
47
48   public void ejbPassivate() {
49
50      productId = null;
51      description = null;
52      price = null;
53   }
54
55   public void ejbLoad() {
56      this.productID = insertDashes(this.productID);
57   }
58
59   public void ejbStore() {
60      this.productID = removeDashes(this.productID);
61   }
62
63   public void ejbRemove() { }
64
65   public void unsetEntityContext() { }
66
67   public void ejbPostCreate(String productId, String description,
68      double balance) { }
69
70   /******* Helper Methods *******/
71   // These method implementations are not shown.
72   private void insertDashes(String id) { ... }
73
74   private void removeDashes(String id) { ... }
75
76 } // ProductEJB
```

# ProductHome.java

```
1  import java.util.Collection;
2  import java.rmi.RemoteException;
3  import javax.ejb.*;
4
5  public interface ProductHome extends EJBHome {
6
7      public Product create(String productId, String description,
8          double balance) throws RemoteException, CreateException;
9
10     public Product findByPrimaryKey(String productId)
11         throws FinderException, RemoteException;
12
13     public Collection findByDescription(String description)
14         throws FinderException, RemoteException;
15
16     public Collection findInRange(double low, double high)
17         throws FinderException, RemoteException;
18 }
19
```

# Product.java

```
1  import javax.ejb.EJBObject;
2  import java.rmi.RemoteException;
3
4  public interface Product extends EJBObject {
5
6     public void setPrice(double price) throws RemoteException;
7
8     public double getPrice() throws RemoteException;
9
10    public String getDescription() throws RemoteException;
11 }
```

# ProductClient.java

```
1  import java.util.*;
2  import javax.naming.Context;
3  import javax.naming.InitialContext;
4  import javax.rmi.PortableRemoteObject;
5
6  public class ProductClient {
7     public static void main(String[] args) {
8
9         try {
10            Context initial = new InitialContext();
11            Object objref = initial.lookup("MyProduct");
12
13            ProductHome home =
14                (ProductHome)PortableRemoteObject.narrow(objref,
15                                                ProductHome.class);
16
17            Product duke = home.create("123", "Ceramic Dog", 10.00);
18            duke.setPrice(14.00);
19            duke = home.create("456", "Wooden Duck", 13.00);
20            duke = home.create("999", "Ivory Cat", 19.00);
21            duke = home.create("789", "Ivory Cat", 33.00);
22            duke = home.create("876", "Chrome Fish", 22.00);
23
24            Product earl = home.findByPrimaryKey("876");
25
26            Collection c = home.findByDescription("Ivory Cat");
27            Iterator i = c.iterator();
28
29            while (i.hasNext()) {
30               Product product = (Product)i.next();
31               String productId = (String)product.getPrimaryKey();
32               String description = product.getDescription();
33               double price = product.getPrice();
34            }
35
```

# ProductClient.java

```
36            c = home.findInRange(10.00, 20.00);
37            i = c.iterator();
38
39            while (i.hasNext()) {
40               Product product = (Product)i.next();
41               String productId = (String)product.getPrimaryKey();
42               double price = product.getPrice();
43               System.out.println(productId + ": " + price);
44            }
45
46        } catch (Exception ex) {
47            System.err.println("Caught an exception." );
48            ex.printStackTrace();
49        }
50    }
51 }
```

# Exercise: Container-Managed Persistence

- Objective
- Tasks

# Check Your Progress

- Explain how the container accesses the internal data stored in your bean
- Implement the `ejbCreate` methods to allow the container to perform persistence
- Redefine the remaining methods to work correctly with container-managed persistence

# Think Beyond

- Discuss what a transaction is, and why it is useful to define actions in an application as transactions.

# Module 11

## Transactions in EJB

# Overview

- Objectives
- Relevance

# Transactions

- An atomic unit of work
- Can consist of multiple operations from multiple objects
- Example: withdrawing money from an account using an automatic teller machine
- Supports the following features:
  - Distribution across a network
  - Two-phase commits
  - Flat transactions

# Transaction Commands

- `BEGIN` – Starts a new transaction
- `COMMIT` – Applies requested operations/changes
- `ROLLBACK` – Undoes requested operations/changes

# Transaction Participants

- Resource has transactional state.
  - Example: Database connection
- Resource manager can commit and rollback.
  - Example: JDBC driver
- Application server assists in managing usage of transactional resources by beans.
  - Example: EJB server

# Transaction Participants

- Transaction manager:
  - Controls state of transaction, two-phase commit
  - Coordinates/controls all resource managers within transaction
- Transactional application obtains a limited access to the transaction manager:
  - Client application
  - Enterprise bean

# Distributed Transactions

- Transaction is associated with a thread as the thread issues requests to objects on different servers.
- Transaction context must be sent with each request.
- EJB requires transaction propagation to be transparent to the bean (implicit propagation).
- Distributed transactions are supported by two-phase commits.

# Isolation Levels

- Locking level on objects within a transaction
- Determine degree of access to a transaction-locked object by a thread in another transaction
- Simple example:
  - Read-lock (multiple readers, no writers)
  - Write-lock (one writer)
- Each resource manager may use different levels.

# Using Isolation Levels With EJBs

- Different isolation levels can be used when accessing different resource managers.
- All accesses within a transaction are done with the same isolation levels for most resource managers.
- The bean provider can specify the isolation level for bean-managed transactions in code.
- Conflicts in isolation levels should be avoided when multiple enterprise beans access the same resource manager.

# Overview of Transactions in EJB

- Transaction demarcation refers to:
  - Beginning a transaction
  - Committing a transaction (approval)
  - Rolling back a transaction (disapproval)
- Two styles of transaction demarcation:
  - Container-managed transaction (CMT)
  - Bean-managed transaction (BMT)

# Bean-Managed Transaction Demarcation

- Session bean is the only enterprise bean that can use BMT.
- Session bean uses `javax.transaction.UserTransaction.`
- The declaration is made in deployment descriptor.

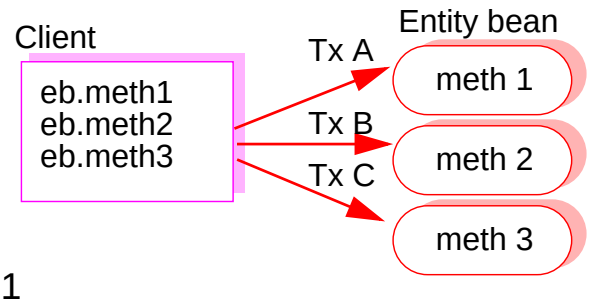
# Scope of a Transaction

```
1  ds1 = (javax.sql.DataSource)
2          initCtx.lookup(“java:comp/env/jdbcDatabase1”);
3  con1 = ds1.getConnection();
4  stmt1 = con1.createStatement();
5
6  ds2 = (javax.sql.DataSource)
7          initCtx.lookup(“java:comp/env/jdbcDatabase2”);
8  con2 = ds2.getConnection();
9  stmt2 = con2.createStatement();
10
11 javax.transaction.UserTransaction ut;
12 ut = ejbContext.getUserTransaction();
13 ut.begin();
14
15 // Use jdbc to update both databases.
16
17 ut.commit();
18
19 //Warning! Do not do a con1.commit().
20 //Close connections.
```

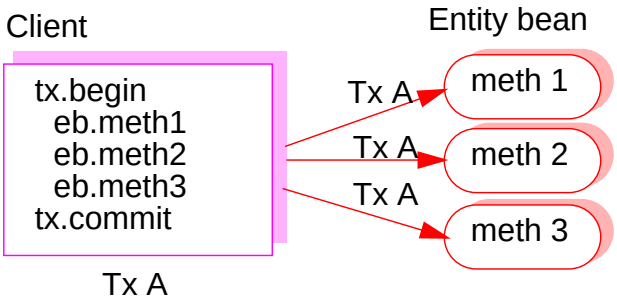
# Container-Managed Transaction Demarcation

- The declaration is made in the deployment descriptor.
- The bean *cannot* use other transaction demarcation APIs including
  - `java.sql.Connection`
  - `javax.transaction.UserTransaction`.

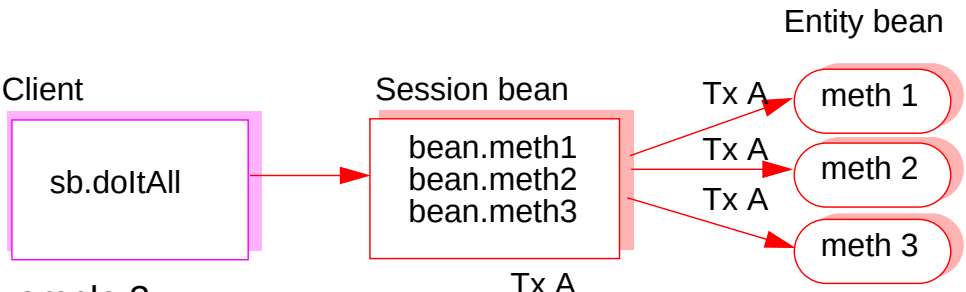
# Client-Managed Transactions




Example 1




Example 2




Example 3

# Accessing Transactional Resources

- Accessing transactions from the client:
  - Server places `UserTransaction` resource in namespace.
  - JNDI name of resource must be communicated to client as property, applet parameter, and so on.

# Check Your Progress

- List and describe the five transactional participants
- List the four transaction isolation levels
- Explain how to send transactions between two beans in a distributed environment

# Think Beyond

- What elements within the EJB application could have the responsibility of managing transactions?
- What advantages and disadvantages would there be to automatic transaction demarcation? How would you set that up?

# Module 12

## Container-Managed Transactions

# Overview

- Objectives
- Relevance

# Container-Managed Transactions

- Valid for both session and entity beans
- Container transparently manages transactions
- Bean cannot attempt to control transaction in any method
- Transaction management type flag in DD must indicate CMT
- Transaction attributes specified in DD for each method

# Container-Managed Transactions

- Who specifies transaction attributes:
  - Developer
  - Assembler can, but then must set controls for all methods
  - Deployer can change them, and must ensure that controls are set on all methods before deploying Bean

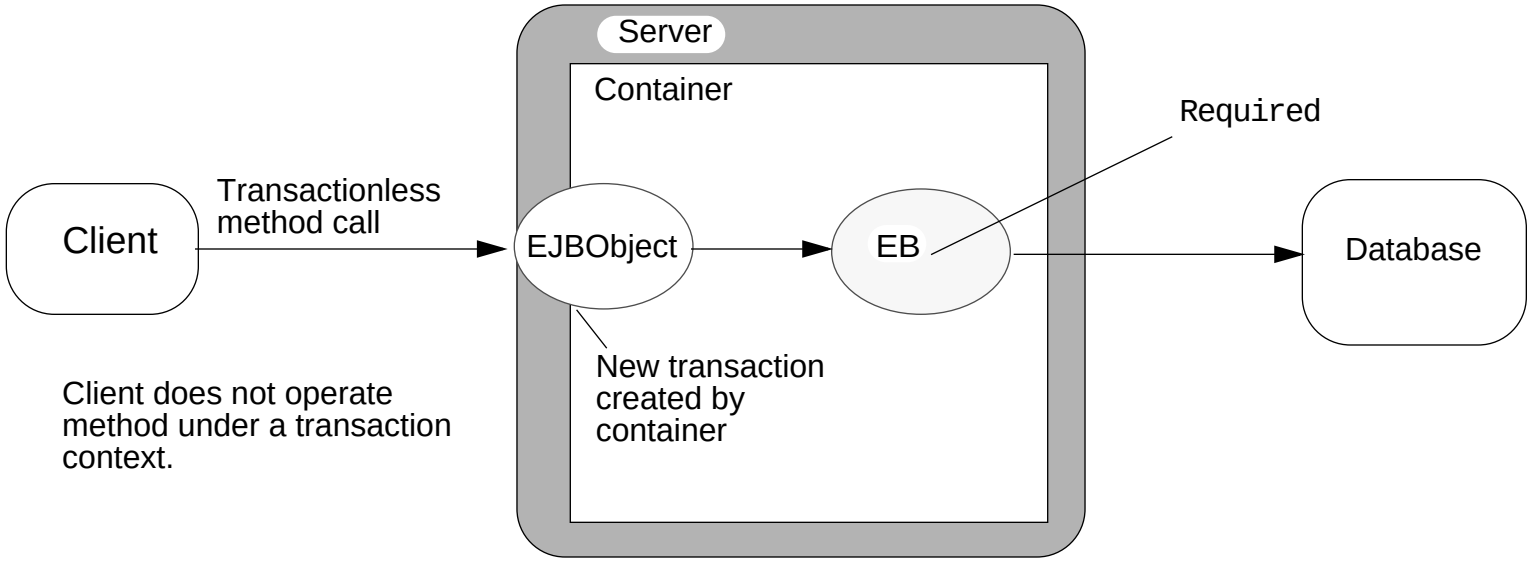
# Transaction Attributes for CMT

- `NotSupported`:
  - Does not start a transaction
  - Existing transaction association is suspended
- `Supports`:
  - Does not start a transaction
  - Existing transaction association is not suspended
- `Required`:
  - Uses existing transaction
  - Starts new transaction; terminate when done

# Transaction Attributes for CMT

- `RequiresNew`:
  - Always starts new transaction; terminate when done
  - Suspends existing transaction association
- `Mandatory`:
  - Must be called within a transaction
- `Never`:
  - Calling this method while the thread is running in a transaction causes an exception to be thrown

# Container-Managed Demarcation



# Methods Requiring Transaction Attributes

- Session beans:
  - All methods defined in the remote interface, only
- Entity beans:
  - All business methods defined in the remote interface.
  - The remove method from `EJBObject` interface.
  - The `create` and finder methods from the home interface.
  - The two `remove` methods from `EJBHome` interface.

# Declaring CMT in the DD

```
1  <ejb-jar>
2     <enterprise-beans>
3        <entity>
4           <ejb-name>         myEJB           </ejb-name>
5           <description>      My Bean         </description>
6           <ejb-class>        my.bean.Class  </ejb-class>
7           <ejb-home>         ...             </ejb-home>
8           <ejb-remote>       ...             </ejb-remote>
9           <transaction-type> Container      </transaction-type>
10       </entity>
11    </enterprise-beans>
12    <assembly-descriptor>
13       <container-transaction>
14          <trans-attribute>  Required        </trans-attribute>
15          <method>
16             <ejb-name>     myEJB           </ejb-name>
17             <method-name>   calcFractals    </method-name>
18          </method>
19          <method>
20             <ejb-name>      myEJB      </ejb-name>
21             <method-name>   fibonacci  </method-name>
22          </method>
23       </container-transaction>
24    </assembly-descriptor>
25 </ejb-jar>
```

# Declaring Method Controls in the DD

- Style 1 – All methods

```
1 <method>
2    <ejb-name>      myBean  </ejb-name>
3    <method-name>   *       </method-name>
4 </method>
```

- Style 2 – All methods of a given name (overloaded)

```
1 <method>
2    <ejb-name>     myBean          </ejb-name>
3    <method-name>  calculateTotal  </method-name>
4 </method>
```

- Style 3– Distinguishing overloaded methods

```
1 <method>
2    <ejb-name>       myBean             </ejb-name>
3    <method-name>    calculateTotal     </method-name>
4    <method-params>
5       <method-param>   int                </method-param>
6    </method-params>
7 </method>
```

# Rollback Only

- Session bean can set a flag to indicate the transaction should not be allowed to commit.
- Methods on `EJBContext`:
  - `void setRollbackOnly()`
  - `boolean getRollbackOnly()`
- Does not initiate the rollback.

# Rollback Only

- These methods cannot be used in beans declaring BMT.

```
1  // In Bean 1
2  if (thingsLookBad) {
3     ejbContext.setRollbackOnly();
4     return;
5  }
6
7  // In Bean 2
8  if (ejbContext.getRollbackOnly() == false) {
9     // do compute-intensive work
10 } else {
11    return;
12 }
```

- Bean uses these methods to mark a transaction or gather information, not to control one.

# Unspecified Transaction Context

- The EJB architecture does not specify the transaction context of session beans for:
  - A method execution in CMT with transaction attribute specified as `NotSupported`, `Never`, or `Supports`.
  - The execution of the `ejbCreate`, `ejbRemove`, `ejbPassivate`, and `ejbActivate` methods
- To ensure portability, the enterprise bean should not rely on a particular technique a container may have for handling this case.

# Exercise: Container-Managed Transactions

- Objective
- Tasks

# Check Your Progress

- List the six transaction attributes
- Name the tag/value pair used in DD to specify CMT
- List which methods of session/entity beans that require transaction attributes should be specified in DD
- Explain how a bean would roll back a transaction

# Think Beyond

- In what respects would bean-managed transactions be different from container-managed transactions?
- What additional or different tasks would need to be performed?

# Module 13

## Bean-Managed Transactions

# Overview

- Objectives
- Relevance

# Bean-Managed Transactions

- The bean controls the state of the transaction programmatically.
  - Creation
  - Commit
  - Rollback
  - Check status

# Bean-Managed Transactions

- Entity beans cannot declare BMT.
- Advantage – Transaction scoping rules do not apply.
- Disadvantage – More difficult for application assembler to build larger application using this bean.

# Accessing Transactional Environment

- Only beans that have a transaction type of BMT can invoke:
  - `getUserTransaction()`
- Only beans that have a transaction type of CMT can invoke:
  - `getRollbackOnly()`
  - `setRollbackOnly()`
- Otherwise, the `IllegalStateException` is thrown.

# UserTransaction

| Method | Description |
|---|---|
| `begin()` | Starts a new transaction. Throws the `NotSupportedException` if a transaction is currently active (nested transactions not supported). |
| `commit()` | Attempts to commit the current transaction. Throws the `IllegalStateException` if a transaction does not currently exist. Will throw `RollbackException` if the transaction could not be committed. |
| `rollback()` | Rolls back the current transaction, or throws the `IllegalStateException` if a transaction does not currently exist. |
| `setRollbackOnly()` | Allows the bean to mark the transaction for rollback later. |
| `getStatus()` | Returns the status of the current transaction. |
| `setTransactionTimeout()` | Specifies that a transaction service implementation can fail a transaction after a timeout period. |

# Accessing Transactional Environment

```
1 package javax.ejb;
2 import javax.transaction;
3 import java.lang.IllegalStateException;
4
5 public interface EJBContext {
6   ...
7   UserTransaction getUserTransaction() throws
8     IllegalStateException;
9
10  boolean getRollbackOnly() throws
11    IllegalStateException;
12
13  void setRollbackOnly() throws
14    IllegalStateException;
15}
```

# Transaction Status

- Allows bean to monitor the status of the transaction by calling `UserTransaction.getStatus()`

```
1  package javax.transaction;
2
3     public interface Status {
4     public static final int STATUS_ACTIVE = ...;
5     public static final int STATUS_COMMITTED = ...;
6     public static final int STATUS_COMMITTING = ...;
7     public static final int STATUS_MARKED_ROLLBACK = ...;
8     public static final int STATUS_NO_TRANSACTION = ...;
9     public static final int STATUS_PREPARED = ...;
10    public static final int STATUS_PREPARING = ...;
11    public static final int STATUS_ROLLEDBACK = ...;
12    public static final int STATUS_ROLLING_BACK = ...;
13    public static final int STATUS_UNKNOWN = ...;
14 }
```

# Declaring BMT in the DD

- Can be used by stateful or stateless session beans.
- Stateless (session) Beans must terminate before returning from business method.

```
1    <enterprise-beans>
2      <session>
3        <description>
4          Describes this session Bean.
5        </description>
6        <ejb-name> CustBugReport </ejb-name>
7        <ejb-class> com.xyz.CustBugBean </ejb-class>
8
9        <home> com.xyz.CustBugHome </home>
10       <remote> com.xyz.CustBug </remote>
11
12       <session-type> Stateful </session-type>
13       <transaction-type> Bean </transaction-type>
14     </session>
15   </enterprise-beans>
```

# Stateful Session Beans

- Container suspends existing transaction if not created by this bean
- Container preserves bean-created transactions across instance method calls
- Only one transaction can exist for the bean at a time

# Stateless Session Beans

- Bean must commit or roll back transaction before returning.
- Transaction cannot remain open across method calls.
- If stateless bean violates this rule:
  - Log as an application error
  - Roll back the transaction
  - Discard the bean
  - Throw `RemoteException` to the client

# TellerEJB.java

```
1  import java.util.*;
2  import javax.ejb.*;
3  import java.sql.*;
4  import javax.sql.*;
5  import javax.naming.*;
6  import javax.transaction.*;
7
8  public class TellerEJB implements SessionBean {
9
10   private String customerId;
11   private double machineBalance;
12   private SessionContext context;
13   private Connection con;
14   private String dbName = "java:comp/env/jdbc/TellerDB";
15
16   public double getCheckingBalance() {
17
18      try {
19         return selectChecking();
20      } catch (SQLException ex) {
21          throw new EJBException
22             ("Unable to get balance: "
23             + ex.getMessage());
24      }
25   }
26
```

# TellerEJB.java

```
27   public void withdrawCash(double amount) {
28      UserTransaction ut =
29           context.getUserTransaction();
30
31      try {
32        ut.begin();
33        updateChecking(amount);
34        machineBalance -= amount;
35        insertMachine(machineBalance);
36        ut.commit();
37      } catch (Exception ex) {
38        try {
39          ut.rollback();
40        } catch (SystemException syex) {
41             throw new EJBException
42             ("Rollback failed: " + syex.getMessage());
43        }
44        throw new EJBException
45          ("Transaction failed: " + ex.getMessage());
46     }
47   }
48
49   public void ejbCreate(String id)
50        throws CreateException {
51
52      customerId = id;
53
54      try {
55         makeConnection();
56         machineBalance = selectMachine();
57      } catch (Exception ex) {
58          throw new CreateException(ex.getMessage());
59      }
60
61   }
```

# TellerEJB.java

```
62    public void ejbRemove()  {
63
64       try {
65          con.close();
66       } catch (SQLException ex) {
67           throw new EJBException(ex.getMessage());
68       }
69    }
70
71    public void ejbActivate()  {
72
73       try {
74          makeConnection();
75       } catch (Exception ex) {
76           throw new EJBException(ex.getMessage());
77       }
78    }
79
80    public void ejbPassivate()  {
81
82       try {
83          con.close();
84       } catch (SQLException ex) {
85           throw new EJBException(ex.getMessage());
86       }
87    }
88
89    public void setSessionContext(SessionContext context) {
90       this.context = context;
91 }
92
93    public TellerEJB() {}
```

# TellerEJB.java

```
94
95 /*************** Database Routines ***************/
96
97    private void makeConnection()
98       throws NamingException, SQLException {
99
100      InitialContext ic = new InitialContext();
101      DataSource ds = (DataSource) ic.lookup(dbName);
102      con =  ds.getConnection();
103   }
104
105   private void updateChecking(double amount)
106      throws SQLException {
107        // Update checking account
108     }
109
110   private void insertMachine(double amount)
111      throws SQLException {
112        // Update balance in machine.
113     }
114
115   private double selectMachine() throws SQLException  {
116        // Select the machine
117     }
118
119   private double selectChecking() throws SQLException  {
120
121        // Read balance from checking account.
122     }
123
124} // TellerEJB
```

# TellerHome.java

```
1  import java.rmi.RemoteException;
2  import javax.ejb.*;
3
4  public interface TellerHome extends EJBHome {
5
6      public Teller create(String id)
7          throws RemoteException, CreateException;
8  }
9
```

# Teller.java

```
1  import javax.ejb.EJBObject;
2  import java.rmi.RemoteException;
3
4  public interface Teller extends EJBObject {
5
6     public void withdrawCash(double amount)
7         throws RemoteException;
8
9     public double getCheckingBalance()
10        throws RemoteException;
11 }
12
```

# TellerClient.java

```
1  import java.util.*;
2  import javax.naming.Context;
3  import javax.naming.InitialContext;
4  import javax.rmi.PortableRemoteObject;
5
6  public class TellerClient {
7
8     public static void main(String[] args) {
9
10         try {
11            Context initial = new InitialContext();
12            Object objref = initial.lookup("MyTeller");
13
14         TellerHome home =
15            (TellerHome)PortableRemoteObject.narrow(objref,
16               TellerHome.class);
17
18         Teller duke = home.create("123");
19         duke.withdrawCash(60.00);
20         duke.remove();
21
22        } catch (Exception ex) {
23            System.err.println("Caught an exception." );
24            ex.printStackTrace();
25        }
26    }
27 }
```

# Check Your Progress

- Indicate in the DD that the bean will be managing its own transactional state
- Use the appropriate APIs in the bean to create and terminate transactions
- Explain the issues of BMT with stateful/stateless session beans

# Think Beyond

- Does the bean need to know the current transaction status?
- How might that information be conveyed?

# Module 14

## Session Synchronization

# Overview

- Objectives
- Relevance

# `SessionSynchronization` Interface

- `javax.ejb.SessionSynchronization`
- Interface *only* for stateful session beans (optional)
- Container-managed transactions only (CMT)
- Invoked by container's transaction service to inform bean of transaction status

```
1 public interface SessionSynchronization {
2   public void afterBegin();
3   public void beforeCompletion();
4   public void afterCompletion(boolean status);
5 }
```

- Transaction is always within scope of method for stateless session beans

# Using Synchronization Callbacks

- Cannot be implemented by stateless session bean
- Assumes only one transaction associated with this bean at a time.
- Provides bean with knowledge of transactional state, while letting the container do all the work.
- `afterBegin()`
  - Tells session bean it is now in a transaction context
  - All business methods now run in this transaction context

# Using Synchronization Callbacks

- `afterCompletion(boolean status)`
  - Transaction has completed
    - If status is true – commit
    - If status is false - rollback
  - Can be used to reset conversational state in bean
- `beforeCompletion()`
  - Commit is about to be attempted by transaction service
  - No one requested a rollback
  - Last chance to write cached data to the database

# Declaring Transaction Attributes on Methods

- `Required`, `RequiresNew`, `Mandatory`
- `SessionSynchronization` interface can be detected by container tools

# BankEJB.java

```
1  import java.util.*;
2  import javax.ejb.*;
3  import java.sql.*;
4  import javax.sql.*;
5  import javax.naming.*;
6
7  public class BankEJB implements
8       SessionBean, SessionSynchronization {
9     private String customerId;
10    private double checkingBalance;
11    private double savingBalance;
12    private SessionContext context;
13    private Connection con;
14    private String dbName = "java:comp/env/jdbc/BankDB";
15
16    public void transferToSaving(double amount) throws
17          InsufficientBalanceException  {
18       checkingBalance -= amount;
19       savingBalance += amount;
20
21       try {
22          updateChecking(checkingBalance);
23          if (checkingBalance < 0.00) {
24             context.setRollbackOnly();
25             throw new InsufficientBalanceException();
26          }
27          updateSaving(savingBalance);
28       } catch (SQLException ex) {
29           throw new EJBException
30              ("SQLException in TX: " + ex.getMessage());
31       }
32    }
```

# BankEJB.java

```
33   public double getCheckingBalance() {
34
35      return checkingBalance;
36   }
37
38   public double getSavingBalance() {
39
40      return savingBalance;
41   }
42
43   public void ejbCreate(String id)
44           throws CreateException {
45      customerId = id;
46
47      try {
48
49        // This method obtains an initial context and
50        // opens a connection.
51        makeConnection();
52
53        checkingBalance = selectChecking();
54        savingBalance = selectSaving();
55      } catch (Exception ex) {
56          throw new CreateException(ex.getMessage());
57      }
58   }
59
60   public void ejbRemove() {
61      try {
62          con.close();
63      } catch (SQLException ex) {
64          throw new EJBException
             ("ejbRemove SQLException: " + ex.getMessage());
65      }
66   }
67
```

# BankEJB.java

```
68   public void ejbActivate() {
69
70      try {
71         makeConnection(); // re-establish connection
72      } catch (Exception ex) {
73          throw new EJBException
74           ("ejbActivate Exception: " + ex.getMessage());
75      }
76   }
77
78   public void ejbPassivate() {
79
80      try {
81           con.close(); // close connection
82      } catch (SQLException ex) {
83          throw new EJBException
84           ("ejbPassivate Exception: " + ex.getMessage());
85      }
86   }
87
88   public void setSessionContext(SessionContext context) {
89      this.context = context;
90   }
91
92    public void afterBegin() {
93
94      // Transaction has begun. Refresh variables.
95
96      System.out.println("afterBegin()");
97      try {
98          checkingBalance = selectChecking();
99          savingBalance = selectSaving();
100     } catch (SQLException ex) {
101         throw new EJBException
102          ("afterBegin Exception: " + ex.getMessage());
103     }
104
105  }
106
```

# BankEJB.java

```
107  public void beforeCompletion() {
108
109     // If cached data, write to database here.
110
111      System.out.println("beforeCompletion()");
112   }
113
114  public void afterCompletion(boolean committed) {
115
116      System.out.println
             ("afterCompletion: " + committed);
117
118     if (committed == false) {
119
120       // Transaction rolled back - must reset variables.
121        try {
122            checkingBalance = selectChecking();
123            savingBalance = selectSaving();
124        } catch (SQLException ex) {
125            throw new EJBException
126                   ("afterCompletion SQLException: " +
127                   ex.getMessage());
128        }
129     }
130   }
131
132   public BankEJB() {}
133
```

# BankEJB.java

```
134/************** Database Routines **************/
135
136   private void updateChecking(double amount) throws SQLException {
137
138      // Update information in checking account.
139   }
140
141   private void updateSaving(double amount) throws SQLException {
142
143      // Update information in savings account.
144   }
145
146   private double selectChecking() throws SQLException {
147
148      // Read information from checking account.
149   }
150
151   private double selectSaving() throws SQLException {
152
153      // Read information in savings account.
154   }
155
156   private void makeConnection()
157       throws NamingException, SQLException {
158
159       InitialContext ic = new InitialContext();
160       DataSource ds = (DataSource) ic.lookup(dbName);
161       con =  ds.getConnection();
162   }
163} // BankEJB
```

# BankHome.java

```
1  import java.rmi.RemoteException;
2  import javax.ejb.*;
3
4  public interface BankHome extends EJBHome {
5
6      public Bank create(String id)
7             throws RemoteException, CreateException;
8  }
```

# Bank.java

```
1  import javax.ejb.EJBObject;
2  import java.rmi.RemoteException;
3
4  public interface Bank extends EJBObject {
5
6      public void transferToSaving(double amount)
7             throws RemoteException;
8
9      public double getCheckingBalance()
10            throws RemoteException;
11
12     public double getSavingBalance()
13            throws RemoteException;
14 }
```

# BankClient.java

```
1  import java.util.*;
2  import javax.naming.Context;
3  import javax.naming.InitialContext;
4  import javax.rmi.PortableRemoteObject;
5
6  public class BankClient {
7
8     public static void main(String[] args) {
9
10       try {
11           Context initial = new InitialContext();
12           Object objref = initial.lookup("MyBank");
13
14           BankHome home = (BankHome)
        PortableRemoteObject.narrow(objref,BankHome.class);
15
16            Bank duke = home.create("123");
17            duke.transferToSaving(40.00);
18           System.out.println
19              ("checking:" + duke.getCheckingBalance());
20         System.out.println("saving: " +
              duke.getSavingBalance());
21
22            duke.remove();
23
24       } catch (Exception ex) {
25           System.err.println("Caught an exception." );
26           ex.printStackTrace();
27       }
28    }
29 }
30
```

# Check Your Progress

- Explain the purpose of the `SessionSynchronization` interface
- Describe how each of the three methods in this interface provide transaction control to your bean
- List which transaction controls can be used on a bean that is implementing this interface
- Explain why beans that implement this interface cannot use BMT

# Think Beyond

- What considerations might there be for dealing with individual users of an EJB application?
- For groups?

# Module 15

## EJB Security

# Overview

- Objectives
- Relevance

# Defining EJB Security Terms

- User:
  - Client performing invocation
  - Not represented in DD
- Principal:
  - Represents a user in the target runtime environment
  - Possibly authenticated using some security protocol
  - Multiple users can be mapped to one principal

# Defining EJB Security Terms

- Role:
  - Group of principals sharing a common permission or set of permissions
  - Role name and method permissions declared in DD
  - Role membership (principals) defined in target environment

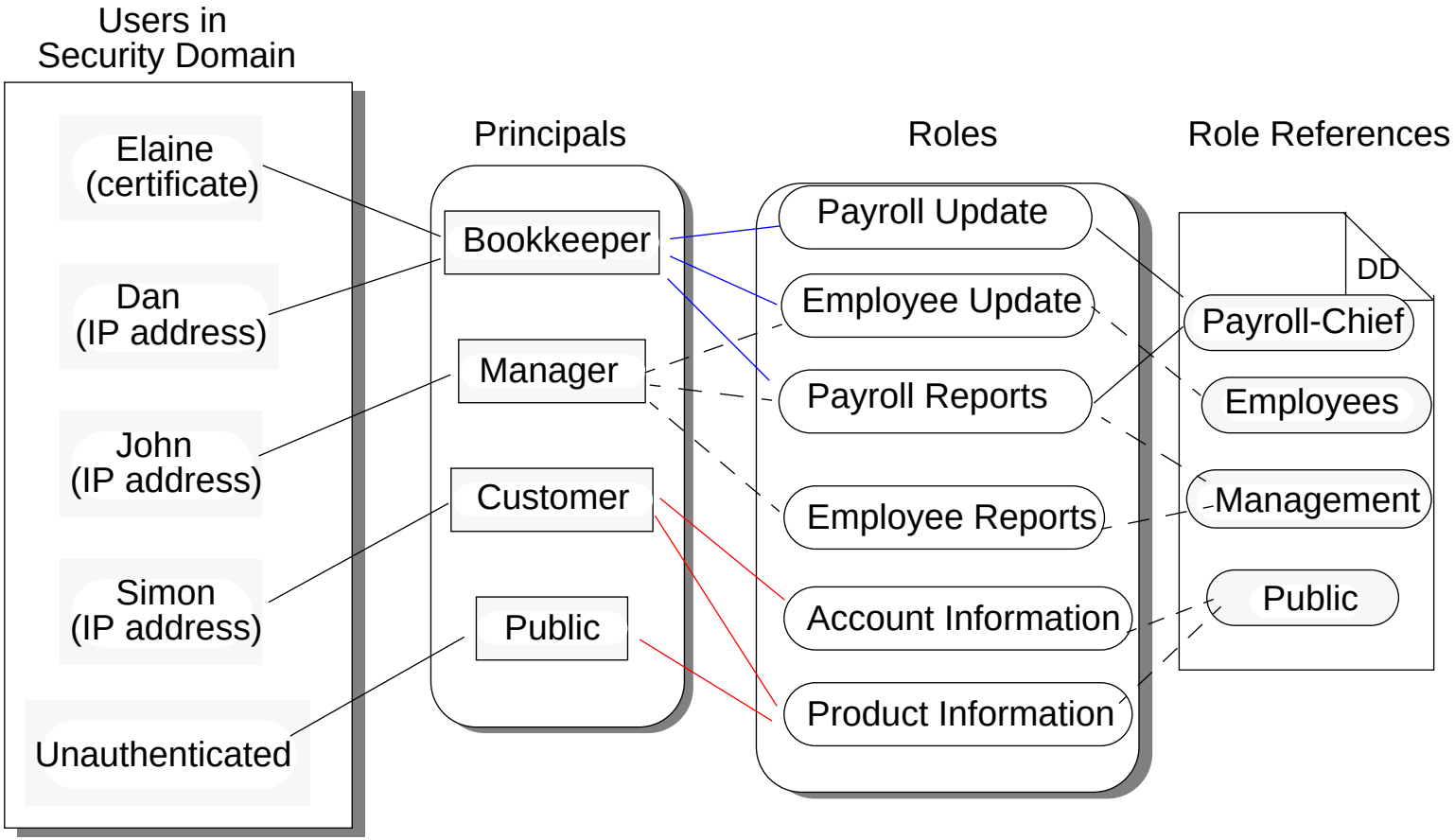
# Defining EJB Security Terms

- Security domain:
  - Set of target environment roles
  - Principal definitions (SSL 3, Kerberos, and so on)
  - Target environment role membership list (of principals)
  - Mappings from target environment roles to DD roles
- Security view:
  - Set of security roles in DD

# Concepts of EJB Security

- Users to principals
- User authentication requirements
- Principals as members of one or more container roles
- Container roles to DD roles
- All are vendor independent and not defined in specification

# Concepts of EJB Security



# Using `getCallerPrincipal()`

- Allows bean to verify principal
- Not intended for security enforcement
- Does not utilize roles
- "Fred Smith" is principal name, possibly not user name

```
1    Principal p = ctx.getCallerPrincipal();
2
3    if (p.getName().equalsIgnoreCase("Fred Smith"))
4       // tailor the method for Fred Smith
5
6    else // unrecognized name
7       throw new MyApplicationException(p.getName() + " invalid");
```

# Using `isCallerInRole()`

- Similar issues as `getCallerPrincipal()`
- Allows recognition of roles without bean trying to identify principal
- "Payroll-Admin" would be declared in DD as reference

# Responsibilities of the Bean Developer

```
1  <ejb-jar>
2     <enterprise-beans>
3        <entity>
4           <ejb-name>Employees</ejb-name>
5           ...
6           <security-role-ref>
7              <description>
8              Needs access to update employee payroll info.
9              </description>
10
11             <role-name>payroll-admin</role-name>
12
13
14          </security-role-ref>
15       </entity>
16    ...
52    </enterprise-beans>
53 </ejb-jar>
```

# Responsibilities of the Application Assembler

- Can define security information, or defer to deployer
- Assembler defines:
  - Roles
  - Method permissions
  - Role reference resolution

# Definition of Roles

```
18   <assembly-descriptor>
19      <security-role>
20         <description>Allows access to employee payroll
21            information (reads and updates)
22         </description>
23         <role-name>Payroll-Chief</role-name>
24      </security-role>
25
26      <security-role>
27         <description>...</description>
28         <role-name>...</role-name>
29      </security-role>
```

# Definition of Method Permissions

```
31      <method-permission>
32         <role-name>Payroll-Chief</role-name>
33         <method>
34            <ejb-name>Employees</ejb-name>
35            <method-name>getSalary</method-name>
36         </method>
37         <method>
38            <ejb-name>Employees</ejb-name>
39            <method-name>setSalary</method-name>
40         </method>
41      </method-permission>
42
43      <method-permission>
44         <role-name>Payroll-Issuer</role-name>
45         <method>
46            <ejb-name>Employees</ejb-name>
47            <method-name>getSalary</method-name>
48         </method>
49      </method-permission>
50
51   </assembly-descriptor>
52
53 </ejb-jar>
```

# Definition of Role Reference Resolution

```
1  <ejb-jar>
2     <enterprise-beans>
3        <entity>
4           <ejb-name>Employees</ejb-name>
5           ...
6           <security-role-ref>
7              <description>
8              Needs access to update employee payroll info.
9              </description>
10
11             <role-name>payroll-admin</role-name>
12             <role-link>Payroll-Chief</role-link>
13
14          </security-role-ref>
15       </entity>
16    </enterprise-beans>
17    ...
```

# Complete Example of the DD Security Section

```
1  <ejb-jar>
2     <enterprise-beans>
3        <entity>
4           <ejb-name>Employees</ejb-name>
5           ...
6           <security-role-ref>
7              <description>
8              Needs access to update employee payroll info.
9              </description>
10
11             <role-name>payroll-admin</role-name>
12             <role-link>Payroll-Chief</role-link>
13
14          </security-role-ref>
15       </entity>
16    </enterprise-beans>
17
18    <assembly-descriptor>
19       <security-role>
20          <description>Allows access to employee payroll
21             information (reads and updates)
22          </description>
23          <role-name>Payroll-Chief</role-name>
24       </security-role>
25
26       <security-role>
27          <description>...</description>
28          <role-name>...</role-name>
29       </security-role>
30
```

# Complete Example of the DD Security Section

```
31      <method-permission>
32         <role-name>Payroll-Chief</role-name>
33         <method>
34            <ejb-name>Employees</ejb-name>
35            <method-name>getSalary</method-name>
36         </method>
37         <method>
38            <ejb-name>Employees</ejb-name>
39            <method-name>setSalary</method-name>
40         </method>
41      </method-permission>
42
43      <method-permission>
44         <role-name>Payroll-Issuer</role-name>
45         <method>
46            <ejb-name>Employees</ejb-name>
47            <method-name>getSalary</method-name>
48         </method>
49      </method-permission>
50
51   </assembly-descriptor>
52
53 </ejb-jar>
```

# Responsibilities of the Deployer

- Declare principals and roles in the security domain
- Map DD roles to roles defined in the security domain
- Configure principal delegation for inter-component calls
- Create resource access policies and mappings

# Configuring Resource Managers

- Deployer should define access permissions for resources.
- Vendor can implement this feature to:
  - Allow list of roles that can access each resource
  - Allow list of principals that can access each resource
  - Allow for specification of unique user-name/ password for each role/resource and principal/ resource association
- Specification does not define requirements for these features.

# Check Your Progress

- Define users, principals, and roles
- Describe what security information is placed in DD
- Explain how a container can provide security implementation

# Think Beyond

- How would you make sure that your entity beans are in sync with your database?

# Appendix A

## Scope of Methods

# Method Scope for CMT and BMT Session Beans

| Methods | CMT | BMT |
|---|---|---|
| setSessionContext | getEJBHome<br><br>JNDI access to java:comp/env | getEJBHome<br><br>JNDI access to java:comp/env |
| ejbCreate<br>ejbRemove<br>ejbActivate*<br>ejbPassivate* | getEJBHome, getCallerPrincipal, isCallerInRole, getEJBObject<br><br>JNDI access to java:comp/env<br><br>Resource Manager access*<br>Enterprise bean access* | getEJBHome, getCallerPrincipal, isCallerInRole, getEJBObject getUserTransaction<br><br>UserTransaction methods<br><br>JNDI access to java:comp/env<br><br>Resource Manager access*<br>Enterprise bean access* |
| Business method from remote interface | getEJBHome, getCallerPrincipal, isCallerInRole, getEJBObject, getRollbackOnly, setRollbackOnly<br><br>JNDI access to java:comp/env<br>Resource Manager access<br>Enterprise bean access | getEJBHome, getCallerPrincipal, isCallerInRole, getEJBObject getUserTransaction, getRollbackOnly, setRollbackOnly<br><br>UserTransaction methods<br><br>JNDI access to java:comp/env<br><br>Resource Manager access<br>Enterprise bean access |

# Method Scope for CMT Entity Beans

| Methods | CMT |
|---|---|
| setEntityContext<br>unsetEntityContext | getEJBHome<br><br>JNDI access to java:comp/env |
| ejbCreate | getEJBHome,<br>getCallerPrincipal, isCallerInRole,<br>getRollbackOnly, setRollbackOnly<br><br>JNDI access to java:comp/env<br>Resource manager access<br>Enterprise bean access |
| ejbPostCreate | getEJBHome,<br>getCallerPrincipal, isCallerInRole,<br>getRollbackOnly, setRollbackOnly,<br>getEJBObject,<br>getPrimaryKey<br><br>JNDI access to java:comp/env<br>Resource manager access<br>Enterprise bean access |
| ejbRemove | getEJBHome, getCallerPrincipal,<br>isCallerInRole,<br>getRollbackOnly, setRollbackOnly,<br>getEJBObject,<br>getPrimaryKey<br><br>JNDI access to java:comp/env<br>Resource manager access<br>Enterprise bean access |

| Methods | CMT |
|---|---|
| ejbFind | getEJBHome, getCallerPrincipal,<br>isCallerInRole,<br>getRollbackOnly, setRollbackOnly,<br><br>JNDI access to java:comp/env<br>Resource manager access<br>Enterprise bean access |
| ejbActivate<br>ejbPassivate | getEJBHome, getEJBObject, getPrimaryKey<br><br>JNDI access to java:comp/env |
| ejbLoad<br>ejbStore | getEJBHome, getCallerPrincipal,<br>isCallerInRole,<br>getEJBObject,<br>getRollbackOnly, setRollbackOnly,<br>getPrimaryKey<br><br>JNDI access to java:comp/env<br>Resource manager access<br>Enterprise bean access |
| Business method from remote interface | getEJBHome, getCallerPrincipal,<br>isCallerInRole,<br>getEJBObject,<br>getRollbackOnly, setRollbackOnly,<br>getPrimaryKey<br><br>JNDI access to java:comp/env<br>Resource manager access<br>Enterprise bean access |

# Appendix B

## Entity Beans with Object-Relational Mapping

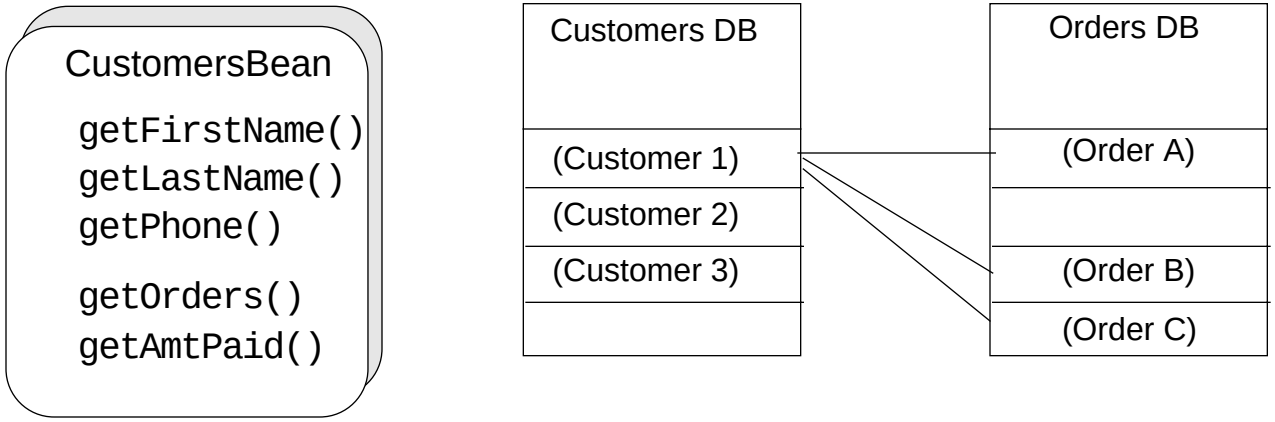
# Overview

- Objectives
- Relevance

# Principles of O-R Mapping

- Databases have relationships.
- Detail `Orders` rows are accessible from master `CustomersBean` instance.
- What kind of accessibility?
  - As a detail set only (`getOrders()`)
  - Customers to maintain cursor (`getNextOrder()`)
  - Returns set of `Orders` as data or remote references?

# Principles of O-R Mapping



# Traversing Without Entity Mapping

- Session Bean is handling relationship issues
  - Additional primary key data for detail table (`orderDate`)
  - Name of the master finder method
- Detail Bean (Orders) must provide master finder
- Session Bean must use the master finder to establish relationships

```
1 CustomersPK custPK = new CustomersPK(first, last, phone);
2
3 custHome.create(custPK);
4 ordHome.create(custPK, orderDate);
5 ...
6 ordHome.findByCustomer(custPK);
```

# Getting a Set of Data

```
1  OrdersRecord[] getOrdersAsData() { // We're in Customers Bean
2
3     CustomersPK custPK = (CustomersPK) ctx.getPrimaryKey();
4
5     String stmt = new String("SELECT * FROM ORDERS WHERE " +
6           " FirstName = '" + custPK.firstName + "' " +
7           " LastName = '" + custPK.lastName + "' " +
8           " Phone = '" + custPK.phone + "';");
9
10    ResultSet rs = conn.executeQuery(stmt);
11
12    while (rs.next()) {
13       ordRec[a].firstName = rs.getString("FirstName");
14       ...
15       ordRec[a].orderDate = rs.getTimestamp("OrderDate");
16    }
17    return ordRec;
18 }
```

# Getting a Set of Data

- More efficient SQL operation (one operation)
- Data is not reusable outside the Customers Bean
- Traversal done in calling Bean

# Getting a Set of Remote References

```
1  Enumeration getOrders() { // We're in Customers Bean
2
3     CustomersPK custPK = (CustomersPK) ctx.getPrimaryKey();
4     Enumeration Beans;
5
6     beans = ordersHome.findByCustomer(custPK);
7
8     return Beans;
9  }
```

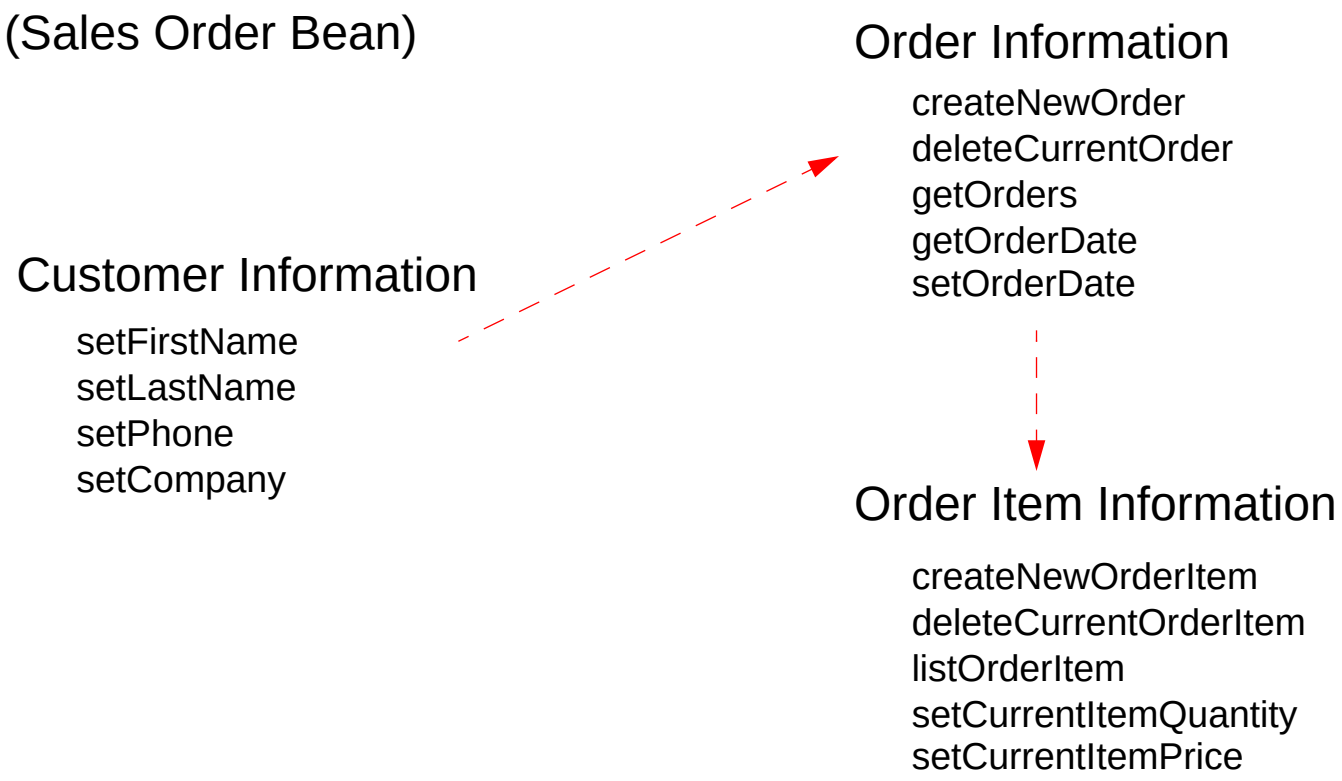
# Getting a Set of Remote References

- Less efficient SQL operation (multiple operations)
- Data is not immediately loaded
- Data is reusable (in separate `OrdersBeans`)
- Traversal supported in called Bean
- In both cases the "cursor" is maintained by caller

# Relation-based Business Methods

```
1  double getTotal() { // We're in Orders Bean
2
3     OrdersPK ordPK = (OrdersPK) ctx.getPrimaryKey();
4     Enumeration enum;
5
6     enum = ordersItemsHome.findByOrder(ordPK);
7
8     while (enum.hasMoreElements()) {
9        ordItemBean = (OrderItems) enum.nextElement();
10       total += ordItemBean.getQuantity() * ordItemBean.getPrice();
11    }
12    return total;
13 }
```

# Applying Techniques to a Business Object



# Session or Entity Bean?

- Arguments for entity beans:
  - Finder methods valuable
  - Can be completely mapped using container CMP mapping
  - Requires sharing by multiple clients
- Arguments for session beans:
  - Business logic in relationship methods
  - Requires state maintenance across transactions

# Check Your Progress

- Describe the advantages and disadvantages of two implementations of detail row data retrieval
- Give one example of a business method that uses object-relational mapping
- Give two reasons why session beans can be superior to entity beans for complex application objects
- Give two reasons why entity beans can be superior to session beans for complex application objects

# Think Beyond

- Could a client directly access entity beans?
- What might be the advantages and disadvantages?

# Appendix C

## Advanced Issues

# Overview

- Objectives
- Relevance

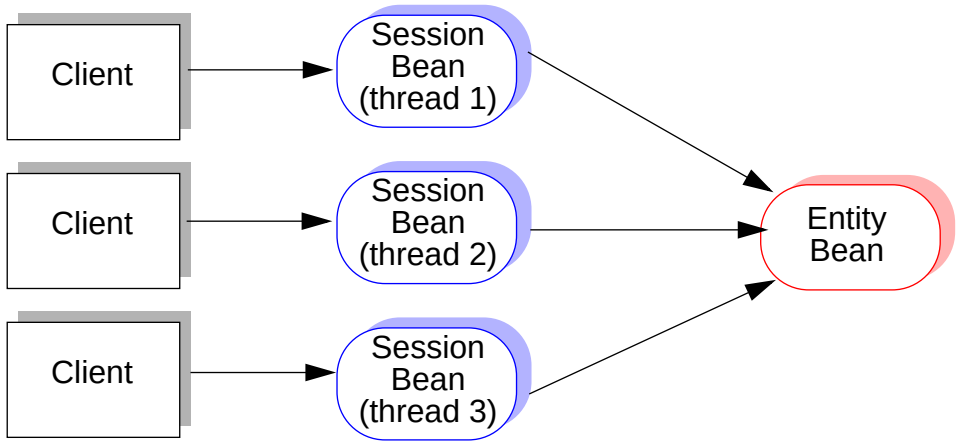
# Direct Client Access to Entity Beans

- Non-transactional access to entity beans
  - Each method runs in a separate transaction
  - Entity beans cannot use BMT
- Additional complexities in client
  - Business logic
  - Knowledge of schema relationships
  - Possible need to demarcate transactions
- Entity beans represent business data objects
- Session beans represent application business rules

# Increasing Execution Concurrency

- Compute-intensive business methods in session beans:
  - Based primarily on client state
  - DB state becomes stale unless locked in transaction
  - DB state is not reusable
- Minimal concurrency
- Data is never stale
- Data is reusable
- Container cannot generate application logic! (CMP)

# Increasing Execution Concurrency



# Data Caching in Session Beans

- Similar to optimistic concurrency
- Allows for use of CMT (only one method in transaction)
- Commit work method
- Methods more complex to write
- Bean might be *invalid* after commit

# Passing Data Directly to Entity Beans

- Similar to pessimistic concurrency
- No data cached in bean
- Database is used as data structure
- Can enable entity bean to be more directly associated with data

# Complex Entity Beans

- Models real-world object:
  - Invoice bean
  - Similar to O-R mapping methods, though possibly more abstract and less relational
  - May use O-R mapping methods of lower-level simple entities
- Advantages (over session beans):
  - Can find existing objects
  - Can push persistence relations out of session beans

# Complex Entity Beans

- Disadvantages:
  - Unlikely a container can implement using CMP
  - No state between transactions
  - Holding transaction across method calls is difficult (no BMT)
- Helper classes compared to other entities:
  - Child object visible from another object? Use entities
  - Helper classes may lead to stale data
  - Helpers can provide better database performance

# Check Your Progress

- Explain two disadvantages to direct entity access from the client
- Explain two basic design principles regarding session and entity beans

# Think Beyond

- On what type of project do you think you might incorporate EJB?

# Course Contents