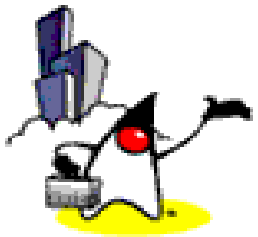
Entity Beans



Sang Shin
sang.shin@sun.com

Jeff Cutler
jpcutler@yahoo.com

www.javapassion.com/j2ee

Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus do not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any contents in this presentation.
- Acknowledgments
  - Large number of slides are made from the contents of the book "Applied Enterprise JavaBeans" Technology written by Kevin Boone (Sun Microsystems), published by Prentice Hall [1]
  - Some slides are made from the contents of J2EE tutorial authored by Dale Green (Sun Microsystems) [2]
  - Some slides (Relationship related) are made from Mastering EJBs authored by Ed Roman [3]
  - Some slides are made from the contents of EJB codecamp material authored by Carol McDonald (Sun Microsystems)

Revision History & Things to Do

- 03/01/2003: version 1, Created (Sang Shin)
- 03/03/2003: Updated (Jeff Cutler)
- 03/24/2003: Updated (Sang Shin)
- 06/20/2003: Speaker notes are added for most slides (Sang Shin)
- 07/01/2003: Some slides which were out of place at the end are cleaned up (Sang Shin)
- Things to do
  - ejBSelect section needs to be improved
  - Schema mapping issue needs to be beefed up a bit
  - Some slides still need speaker notes

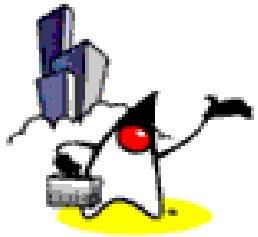
Agenda - page 1

- What is and why entity bean?
- Operational flow
  - findX(), create(), remove()
- Life cycle of entity bean instance
- Role of container for entity beans
- Primary key
- Codes bean developer write (server side)
- Codes client programmer write (client side)
- Limitations of CMP 1.1
- CMP 2.0

Agenda - page 2

- Relationship handling
  - Cardinality, Directionality, Lazy-loading, Cascade delete, Referential Integrity
- EJB-QL
- Relationship and EJB-QL
- Persistence strategies
- Commit options
- Home methods
- Passing references to EJBs within EJBs
- Exception handling

What is an Entity Bean? (Review)



Entity Beans

- Provides object view of persistent data
  - Its lifetime is not related to the duration of interaction with clients
    - Lives as long as data exists in database i.e. Long lived (reason why it is called persistent)
  - Provides database independence
    - In most cases, relational databases are used
- Shared access among many clients
- Transactional
  - Entity beans are always transactional

Entity Beans & RDBMS

- Persistent data is typically Relational database but does not have to be (even more true with CMP 2.0)
  - Could be object database
  - Could be XML-native database
  - Could be data model of legacy application
  - Could be data model of Enterprise Information Systems (SAP, PeopleSoft, etc.)
  - Could be custom persistent technology

Entity Beans Usage Pattern

- Clients normally look up (find) an existing entity EJB (existing row of database table)
  - Creation means adding a row to a database table
  - Finding means finding a row in a existing database table
  - Removal means deleting a row in a database table
- Entity bean instance has unique identifier called primary key
  - Primary key can be any class

Examples of Entity Beans

- Customer
  - Customer data has to persist, thus is maintained in the database
  - Customer data remains even after application closes
  - Customer data has to survive server crash
  - Customer data is shared by many clients
  - Each customer has unique identification such as customer number

2 Types of Entity Beans

- CMP (Container Managed Persistence)
  - Persistence logic code is provided by Container
  - Persistence requirements are specified in deployment descriptor
  - Bean developer does not have to worry about providing persistence logic in his code
- BMP (Bean Managed Persistence)
  - Persistence logic code is provided by Bean developer in code itself

When to Use CMP vs. BMP?

- **CMP entity beans**
  - **With CMP 2.0, there is no reason not to use CMP**
  - **Database independence**
  - **Higher performance (especially with CMP 2.0)**
  - **Easy to develop and deploy (no persistence logic to write in the code)**
- **BMP entity beans**
  - **Use it when more programmatic control is desired**

Session Beans and Entity Beans

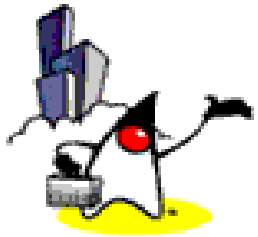
Session Beans

- Represent a business process
- One instance per client
- Short-lived (transient): Life of client matches life of bean
- Doesn't survive server crashes
- May be transactional

Entity Beans

- Represent business data
- Instances shared by multiple clients
- Long-lived (persistent): as long as data in database persists
- Survive server crashes
- Always transactional

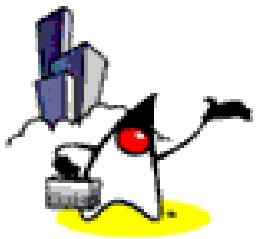
Why Entity Bean (as opposed to client-driven persistence)?



Without Entity Beans, Client has to handle Persistence themselves

- Issues when client code itself handles persistence logic
  - Client has to manage the state, thus not Object-Oriented: In OO philosophy, object should manage its own state
  - Hard to deal with concurrent access to the same data
  - Client code itself has to deal with transaction demarcation
- Entity Beans moves the chore of persistence management to the container

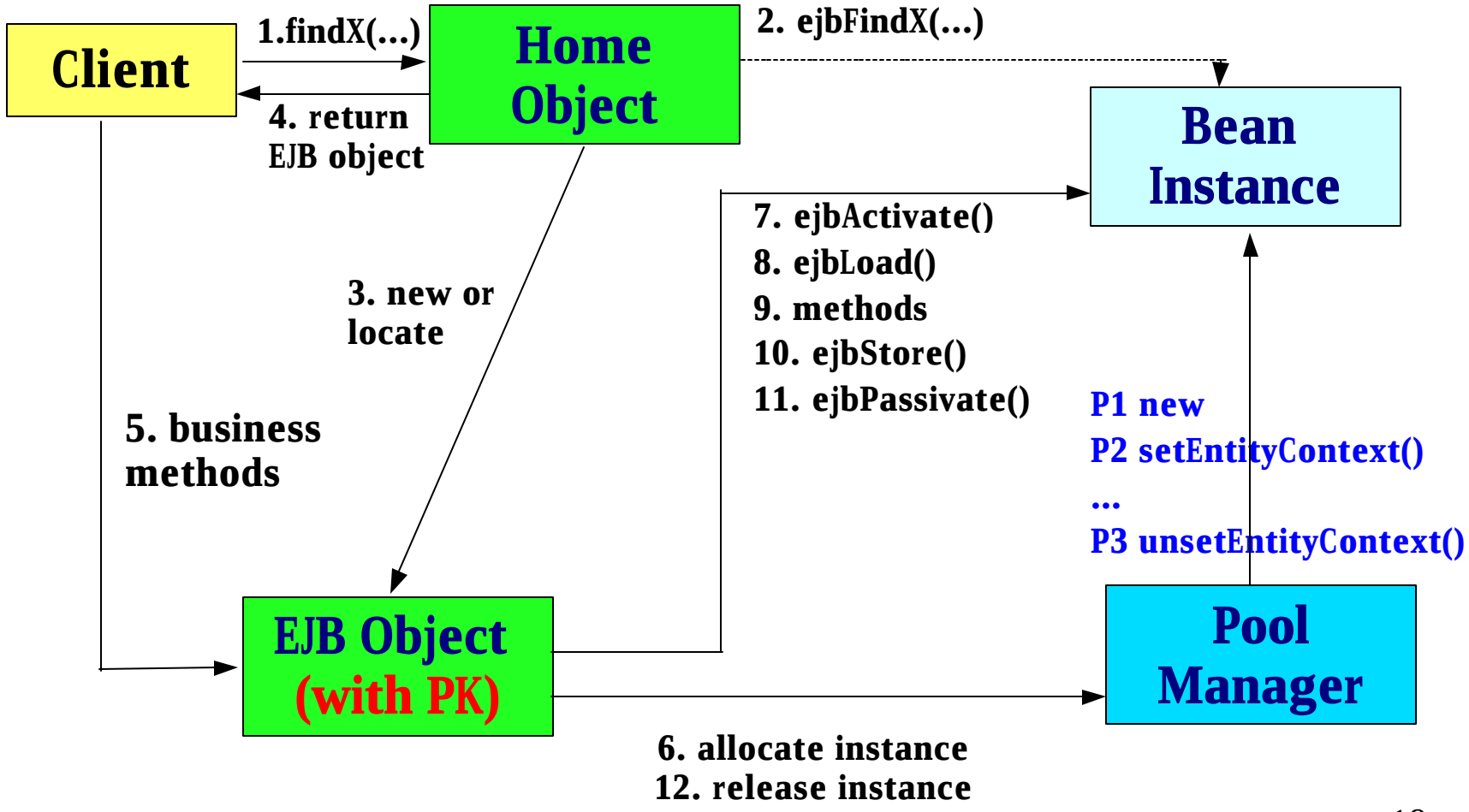
Operational Flow when a Client calls findX() method



Example: findByPrimaryKey()

```
Object ref = context.lookup("Customer");
CustomerHome home =
        (CustomerHome)PortableRemoteObject.narrow (ref,
                                                   CustomerHome.class);

// Find a single instance of EJB Object via primary key.
// (Remember EJB Object is type of Remote object.)
Customer customer = home.findByPrimitiveKey(1);

String name = customer.getName();
String address = customer.getAddress();
```

Interaction among Client, Bean instance, Container for Entity Bean (Client findX() call)



source: Applied Enterprise JavaBeans[1]

findX() Operational Flow

1. Client retrieves home object via JNDI lookup() and calls findX() method
2. Home object finds the primary key
   - BMP: Home object calls ejbFindX() method of any free bean instance
   - CMP: Home object does appropriate "SQL SELECT" operation itself using EJB QL finder logic specified in deployment descriptor
3. Home object finds/creates an EJB object corresponding to the primary key
   - Container maintains one EJB object for one primary key

findX() Operational Flow

4.Home object returns reference to EJB object (actually reference to stub of remote EJB object for the case of remote EJB object) to client

5.Client calls business methods through EJB object

6.On receipt of the first business method call, EJB object gets a bean instance from the pool

findX() Operational Flow

7. EJB object now has an associated Bean instance. Container (maybe through EJB Object) calls ejbActivate() of the bean instance

- **This is called "activation"**
- **The state of bean instance changes from "passive" to "active" state**
- **In short, you can regard an "active" bean instance has an associated EJB object which itself is associated with a particular primary key**

findX() Operational Flow

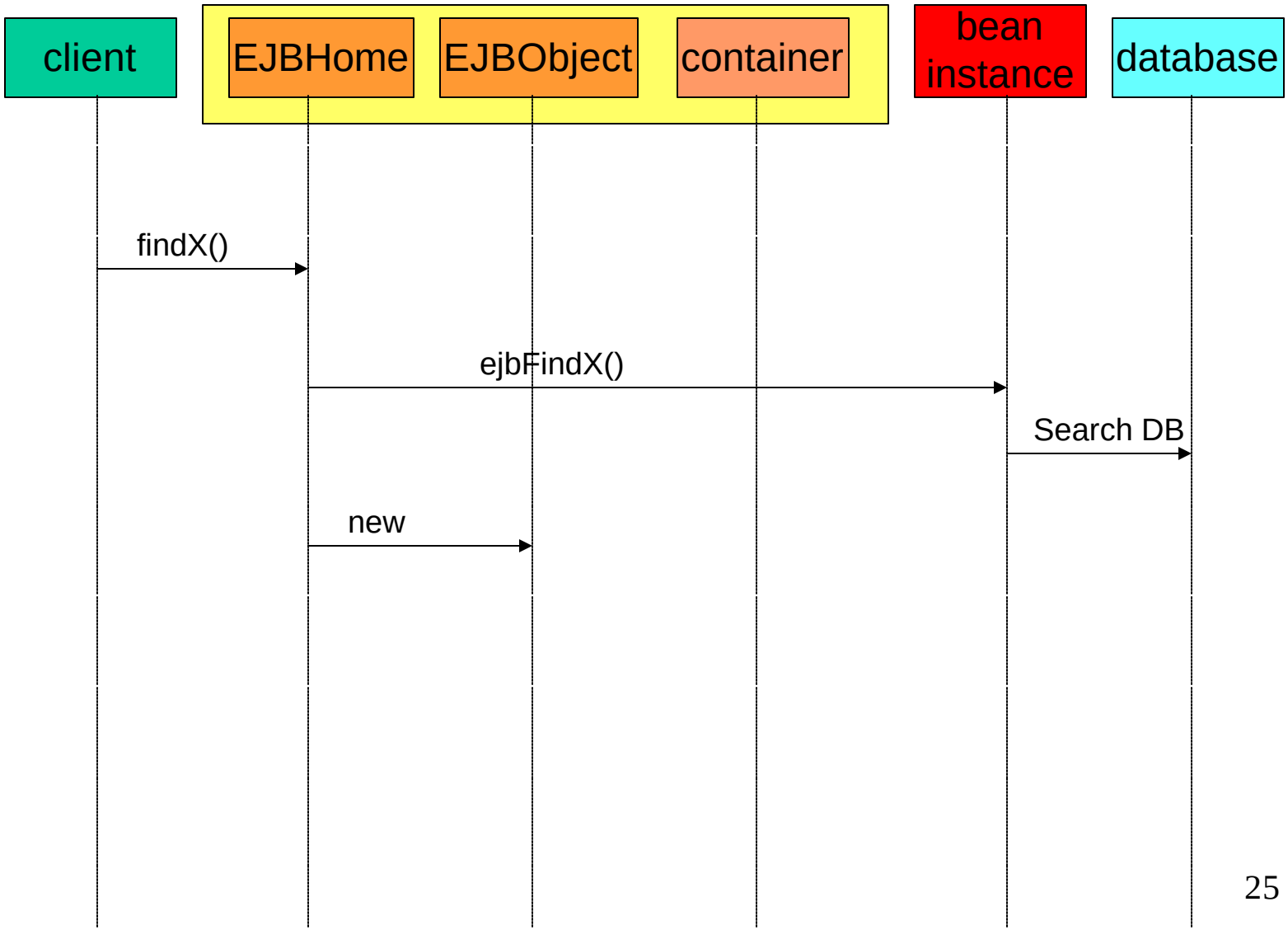
8. EJB object (or some part of Container) calls ejbLoad() of the bean instance

- To instruct the bean instance to make its internal state match the database
  - BMP: Uses the primary key to perform "SQL SELECT" statement inside the ejbLoad() code
  - CMP: Container performs the operation

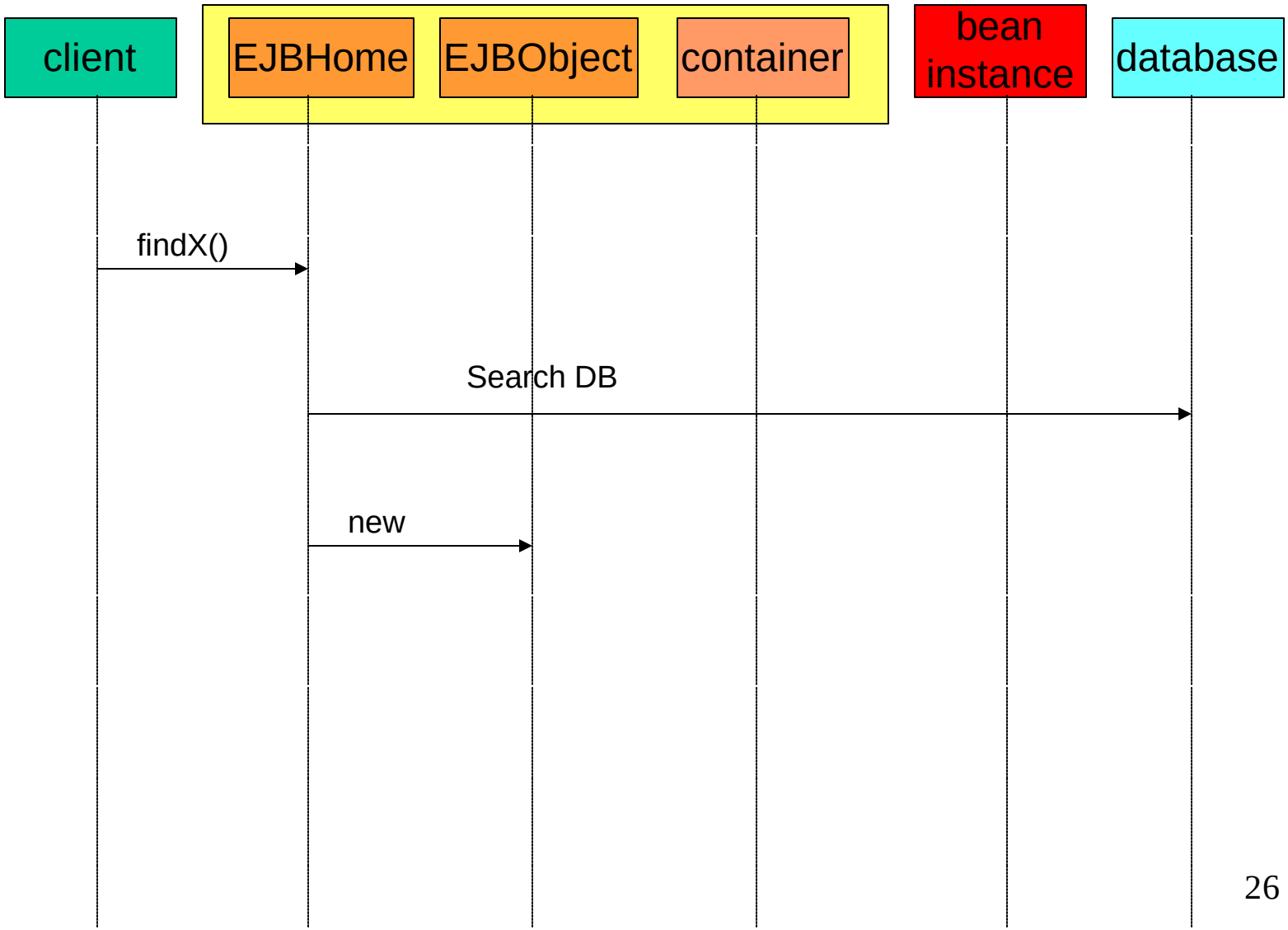
findX() Operational Flow

9. EJB Object calls business methods of bean instance
10. Container calls ejbStore() of the bean instance
    - BMP: "SQL UPDATE"
    - CMP: Container handles it
11. When the container is finished with the bean instance, container calls ejbPassivate()
    - This is called "Passivation"
12. Container puts instance back to the pool

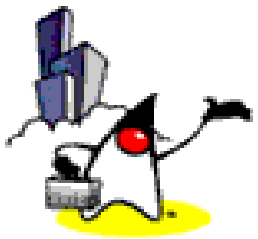
FindX() of BMP


client
EJBHome
EJBObject
container
bean instance
database
findX()
ejbFindX()
Search DB
new

FindX() of CMP


client
EJBHome
EJBObject
container
bean instance
database
findX()
Search DB
new

Multiple EJB Response (Collection) for findX() method

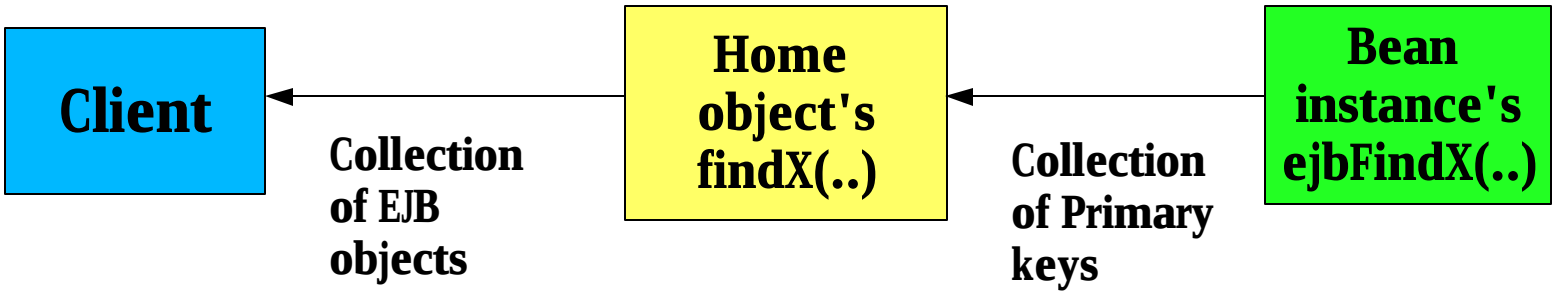


Single-EJB findX() vs Multi-EJB findX()

- Depends on findX() logic, either single EJB or multi-EJB could be returned
- Example of Single-EJB in remote interface
  - findByPrimaryKey(<primary-key>)
  - findSingleCustomer(..)
- Example of Multi-EJB in remote interface
  - findAllCustomers(100000) //with income over 100000 dollars
  - findAllBooks(1999) // which are published in 1999

Multi-EJB findX() method

- ejbFindX(..) of Bean instance can return (to calling Home object) Collection of primary keys
- findX(..) of Home object returns Collection of EJB objects



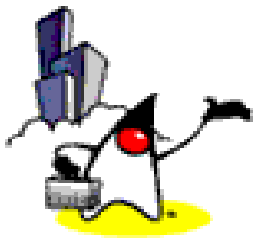
Multi-EJB findX() method

- Use Multi-EJB findX() with care
  - Container has to instantiate an EJB object for every row
  - EJB objects have to be marshaled and unmarshaled over network connection
- Use HOME method to carry out database SELECT operations and return results to clients by value
  - Increased network efficiency
  - CMP 2.0 has built-in support for passing multiple matches by value

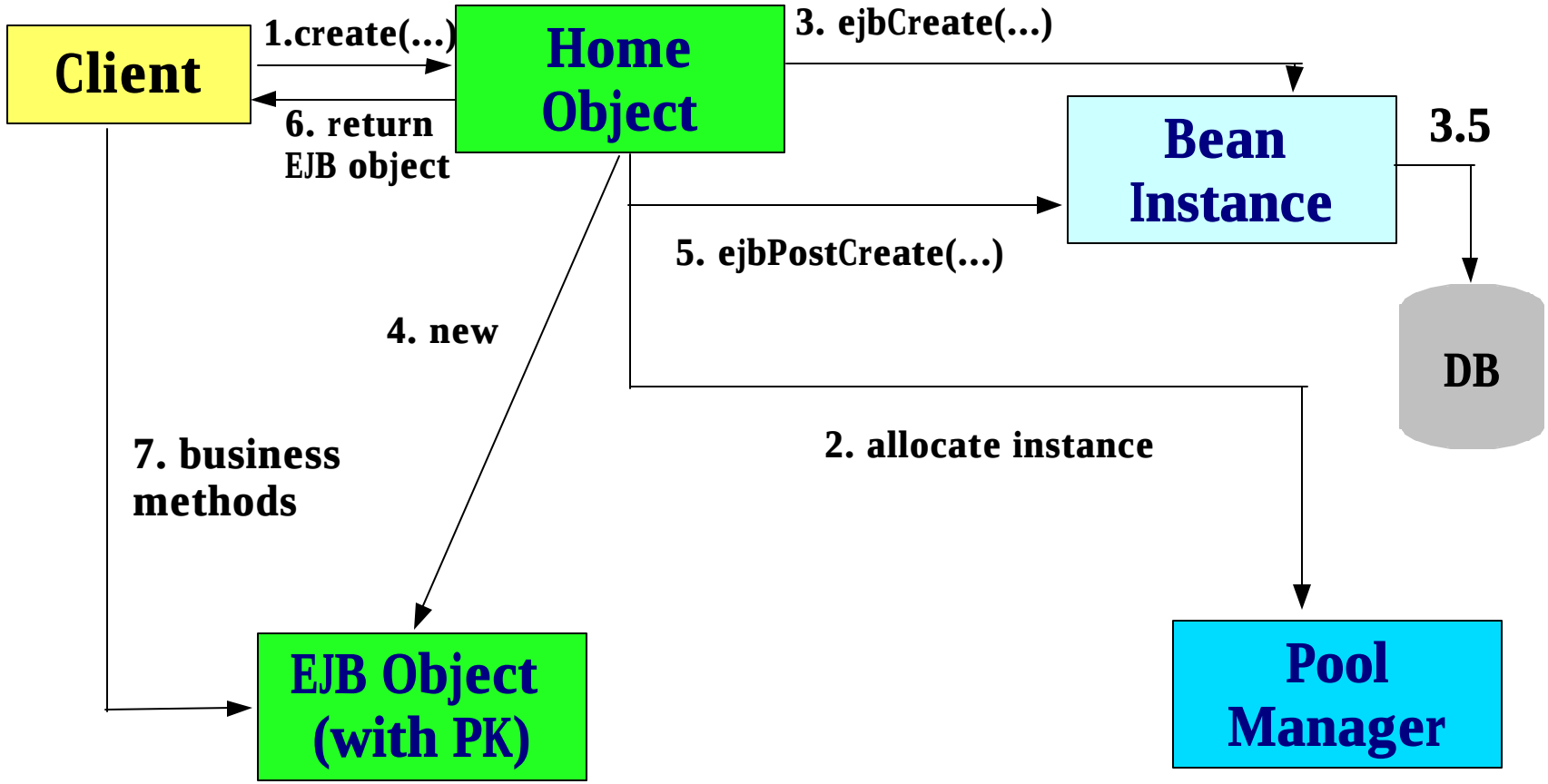
Example: findX()

```
Object ref = context.lookup("Customer");
CustomerHome home =
        (CustomerHome)PortableRemoteObject.narrow (ref,
                                                   CustomerHome.class);

// Find a single instance of EJB Object via primary key
Customer customer = home.findByPrimitiveKey(1);

String name = customer.getName();
String address = customer.getAddress();
```

Operational Flow when a Client calls create(..) method



Interaction among Client, Bean instance, Container for Entity Bean (Client create() call)



source: Applied Enterprise JavaBeans[1]

create(..) Operational Flow

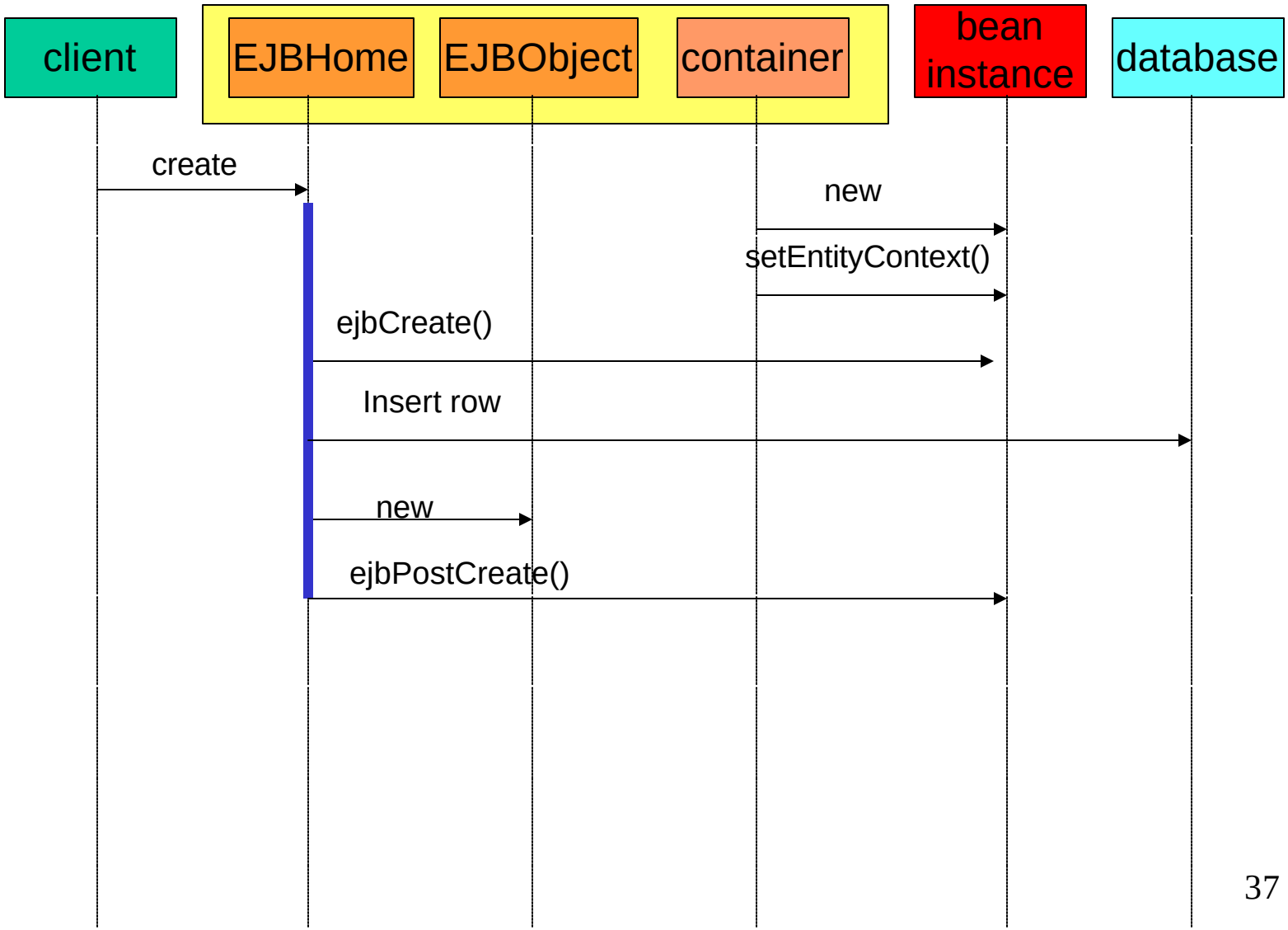
1. Client calls create(..) method on Home object
2. Home object (or some other part of Container) gets a free bean instance
3. Container calls ejbCreate(..) of the bean instance passing same arguments
   - Bean instance set the internal state with the passed arguments
     - BMP: Creates a new row via "SQL INSERT", returns a Primary Key
     - CMP: Container does this after calling ejbCreate(..), returns null

create(..) Operational Flow

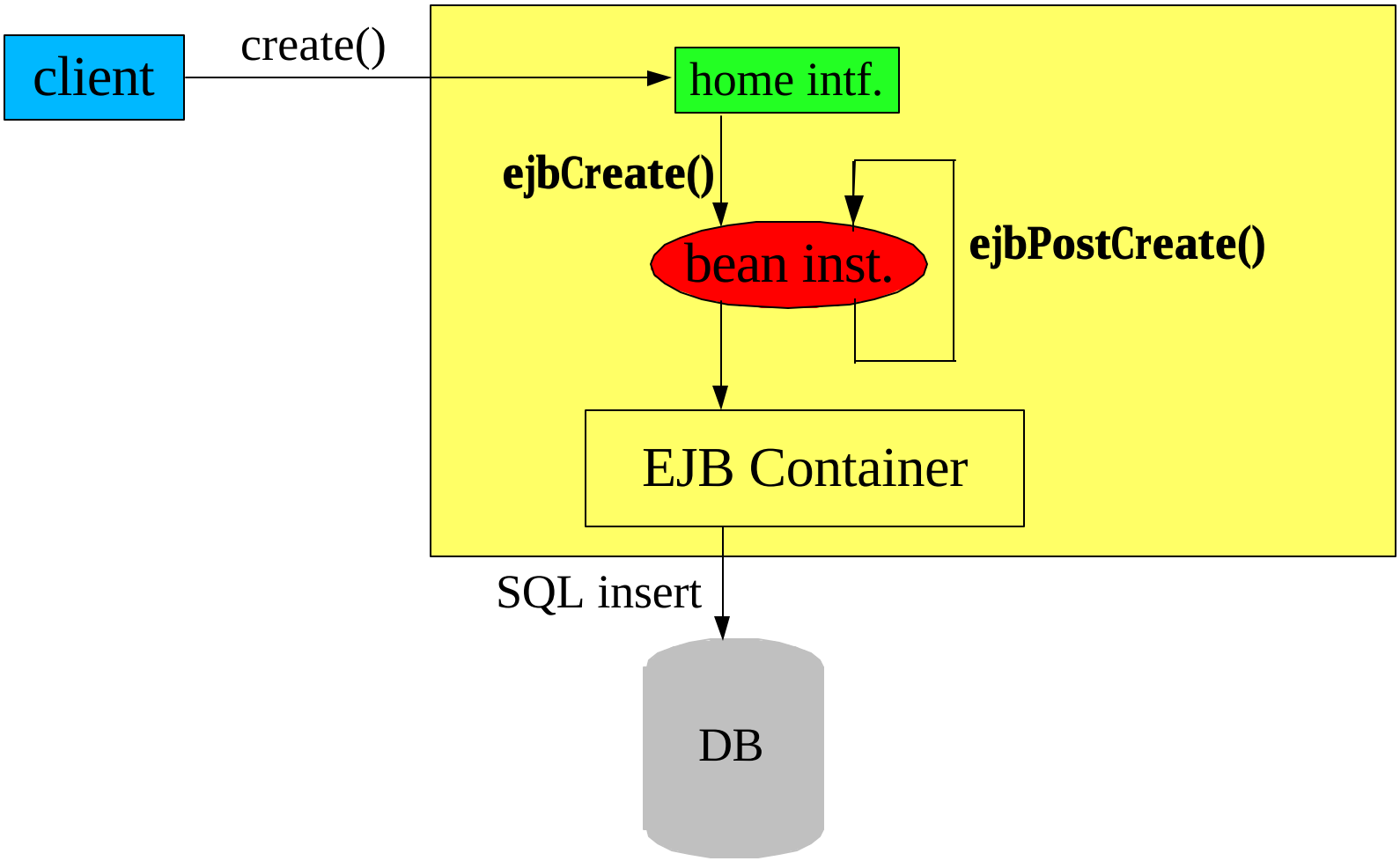
4. Container creates a new EJB object and sets the primary key
5. Container calls ejbPostCreate() on the bean instance
6. Home object returns a reference to the EJB object to the client
7. Client calls business methods

- Is used to do some initialization (by the bean instance) that require the existence of EJB object
  - Container cannot create EJB object until after ejbCreate() is returned
  - Container needs primary key in order to create EJB object
  - Primary key can be obtained only after ejbCreate() is done
- Used when Entity bean has association with other Entity beans and they have to pass references to themselves during creation period

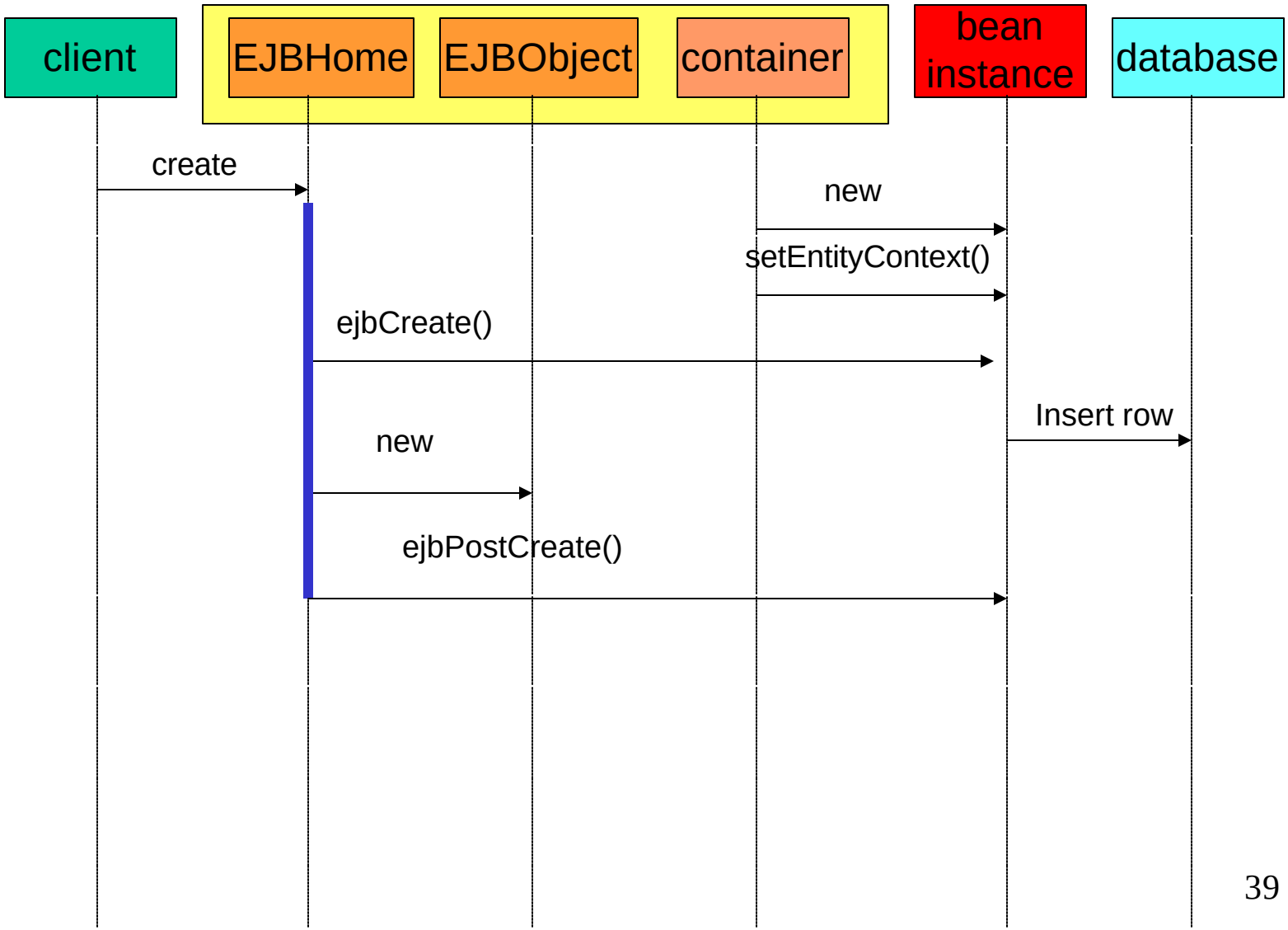
Creation of CMP Bean Instance



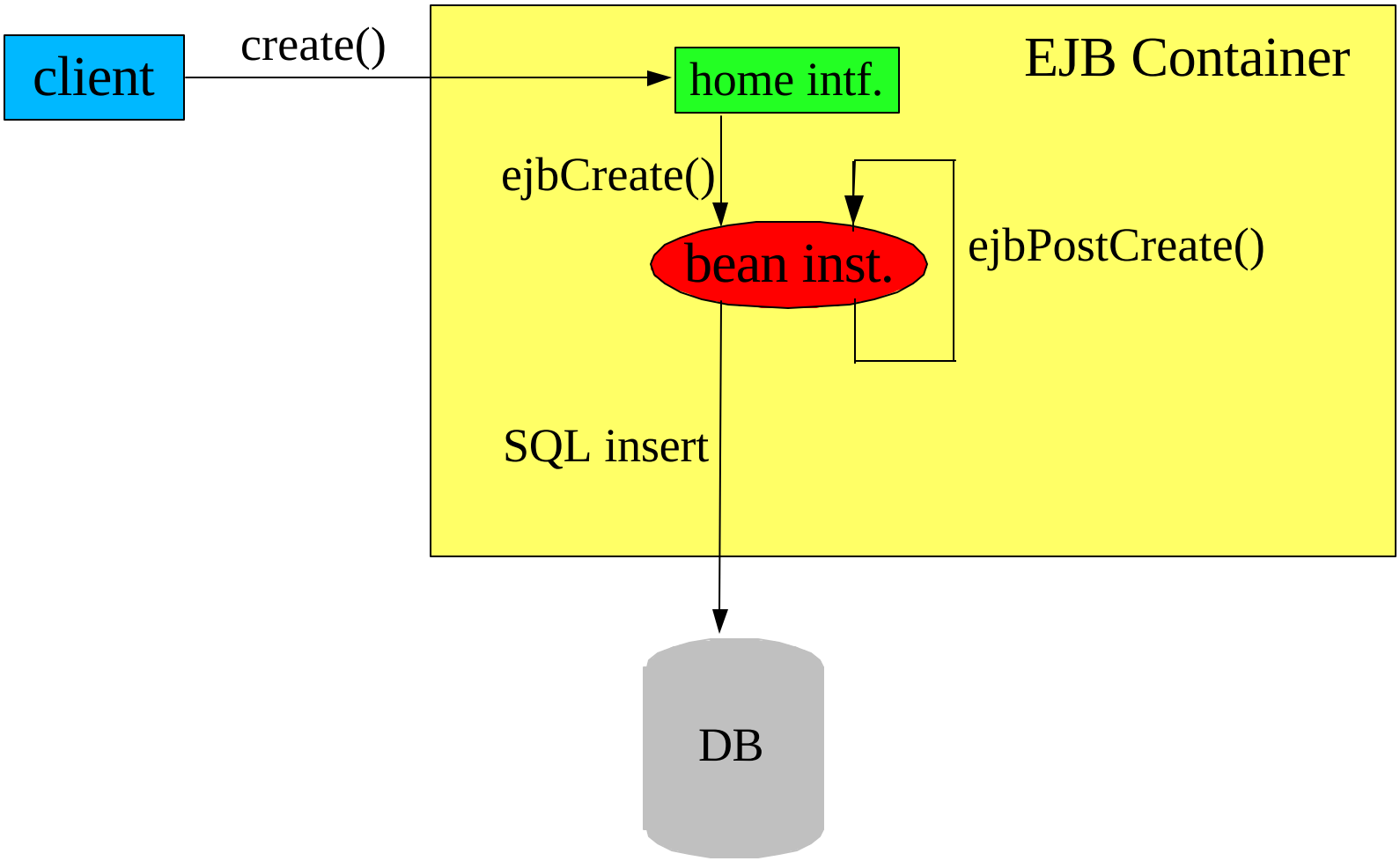
Creation of CMP Bean Instance



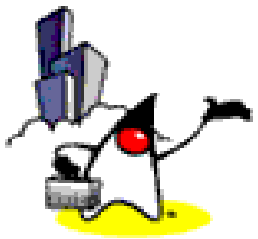
Creation of an BMP Bean Instance



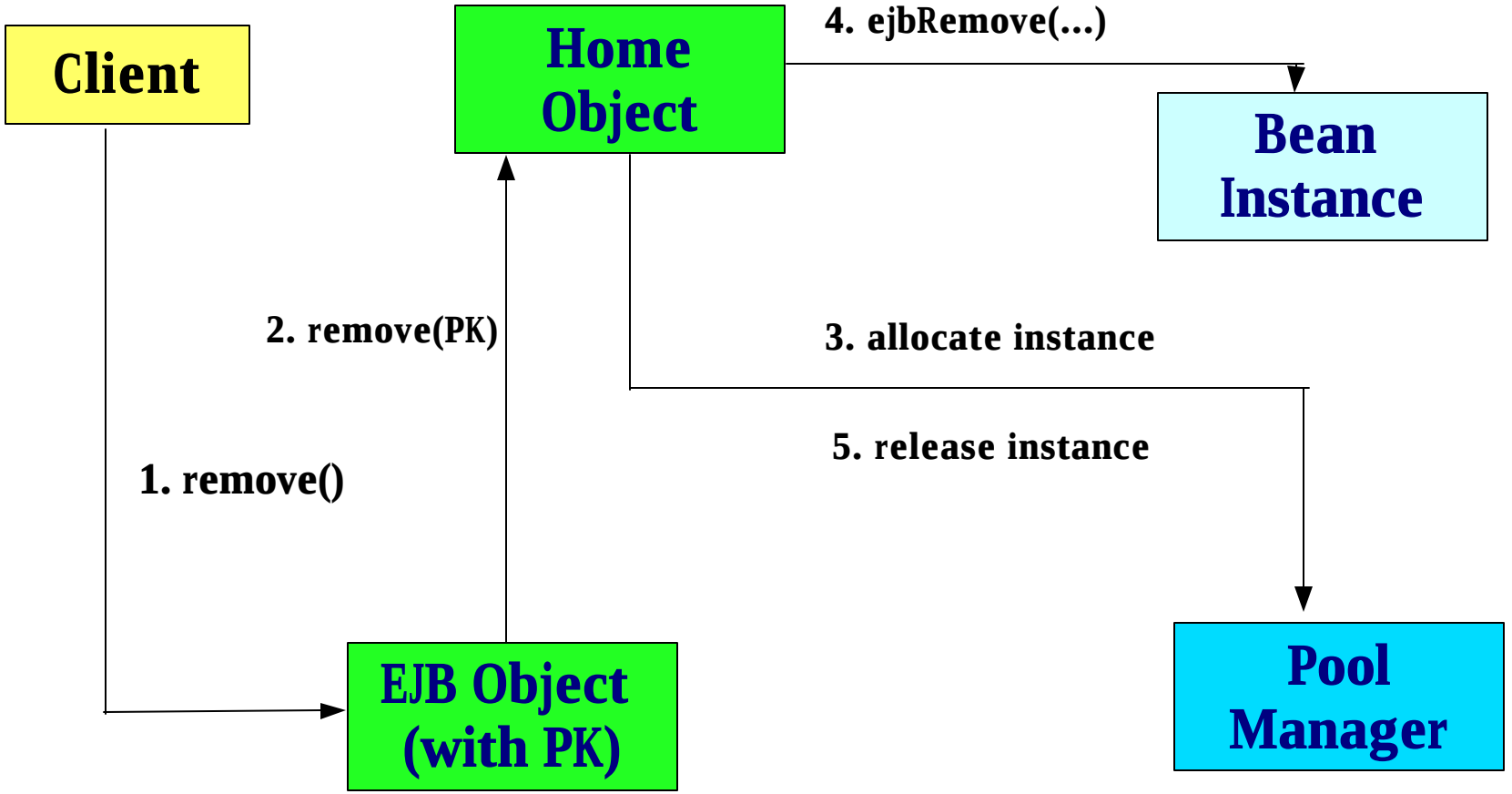
Creation of BMP Bean Instance



Operational Flow when a Client calls remove() method



Interaction between Client, Bean instance, Container for Entity Bean (Client remove() call)



source: Applied Enterprise JavaBeans[1]

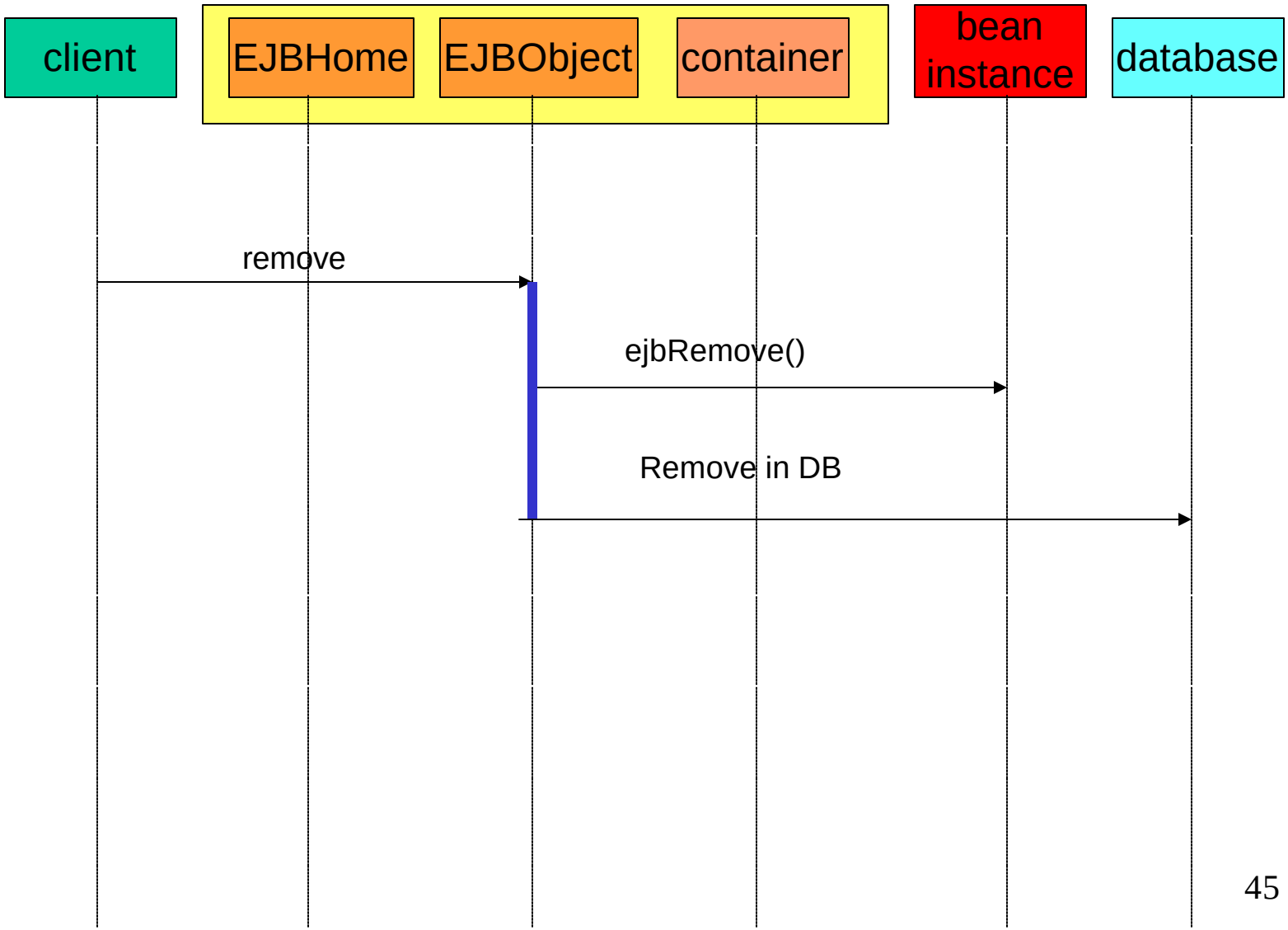
remove() Operational Flow

1. Client calls remove() on the EJB Object or remove(<primary-key>) on the Home object
2. EJB Object delegates the call to Home object
   - This step is not necessary if client calls Home object directly
3. Home object obtains an free bean instance

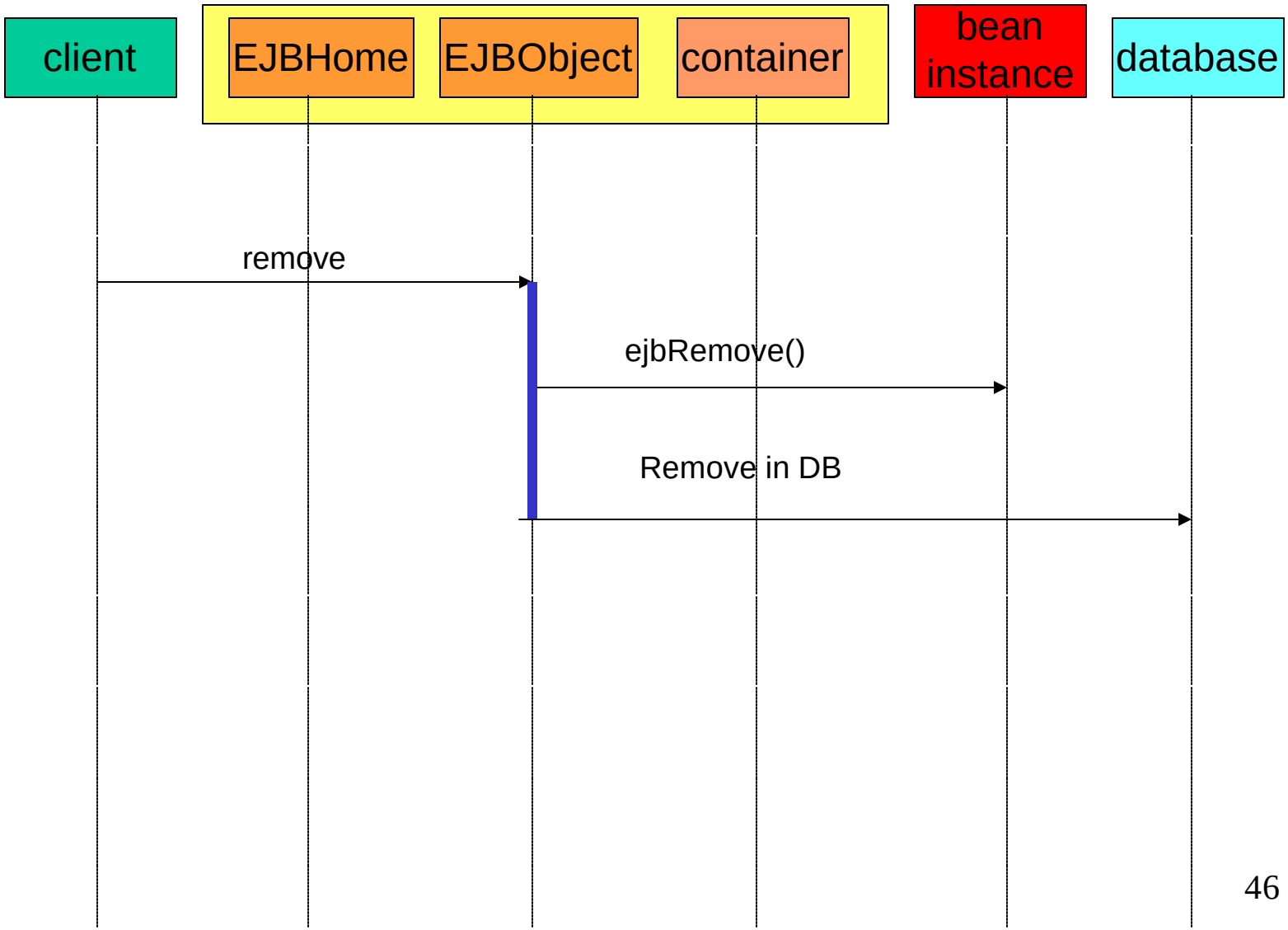
remove() Operational Flow

4. Home object calls ejbRemove() on the bean instance
   - BMP: performs "SQL DELETE" inside the code
   - CMP: Container already have deleted the data and any dependent objects
5. Bean instance is returned to the pool

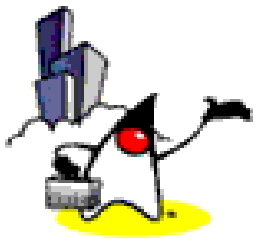
Removal of an CMP-based Entity Bean



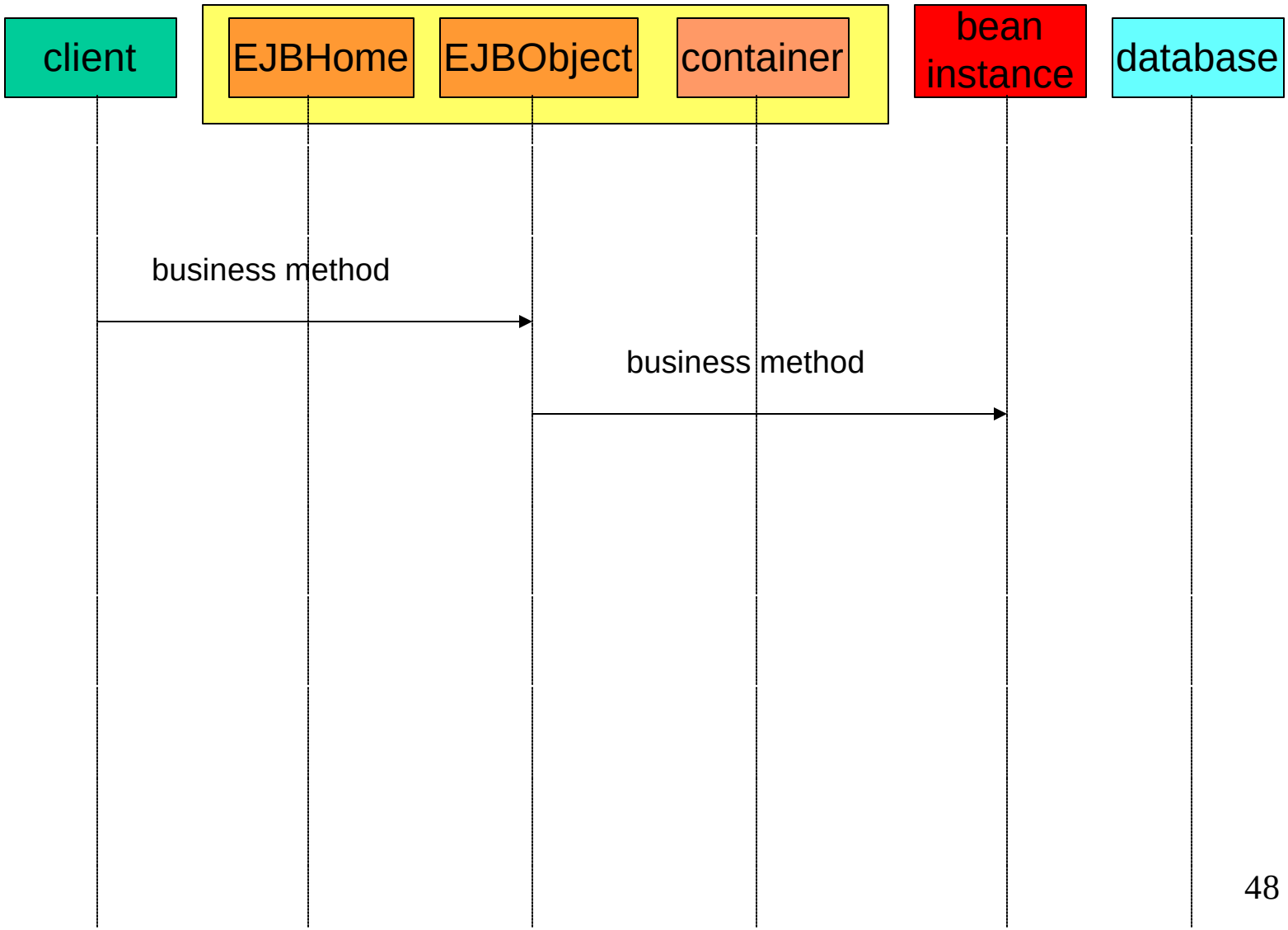
Removal of an CMP-based Entity Bean



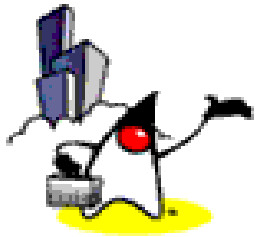
Operational Flow when a Client calls Business method



Calling Business Methods of Entity bean



Life Cycle of an Entity Bean Instance



create(..) & remove() & findX() of Entity Bean

- create(..) operation of Entity bean is not the same as instantiating a Java object
  - create(..) operation of Entity bean creates a corresponding piece of persistent data
  - remove() operation of Entity bean removes the corresponding piece of persistent data
- Clients of Entity beans typically use findX() instead of create(..) since the persistent data (i.e.) might have been typically created via different means

Home Interface of Session Beans and Entity Beans

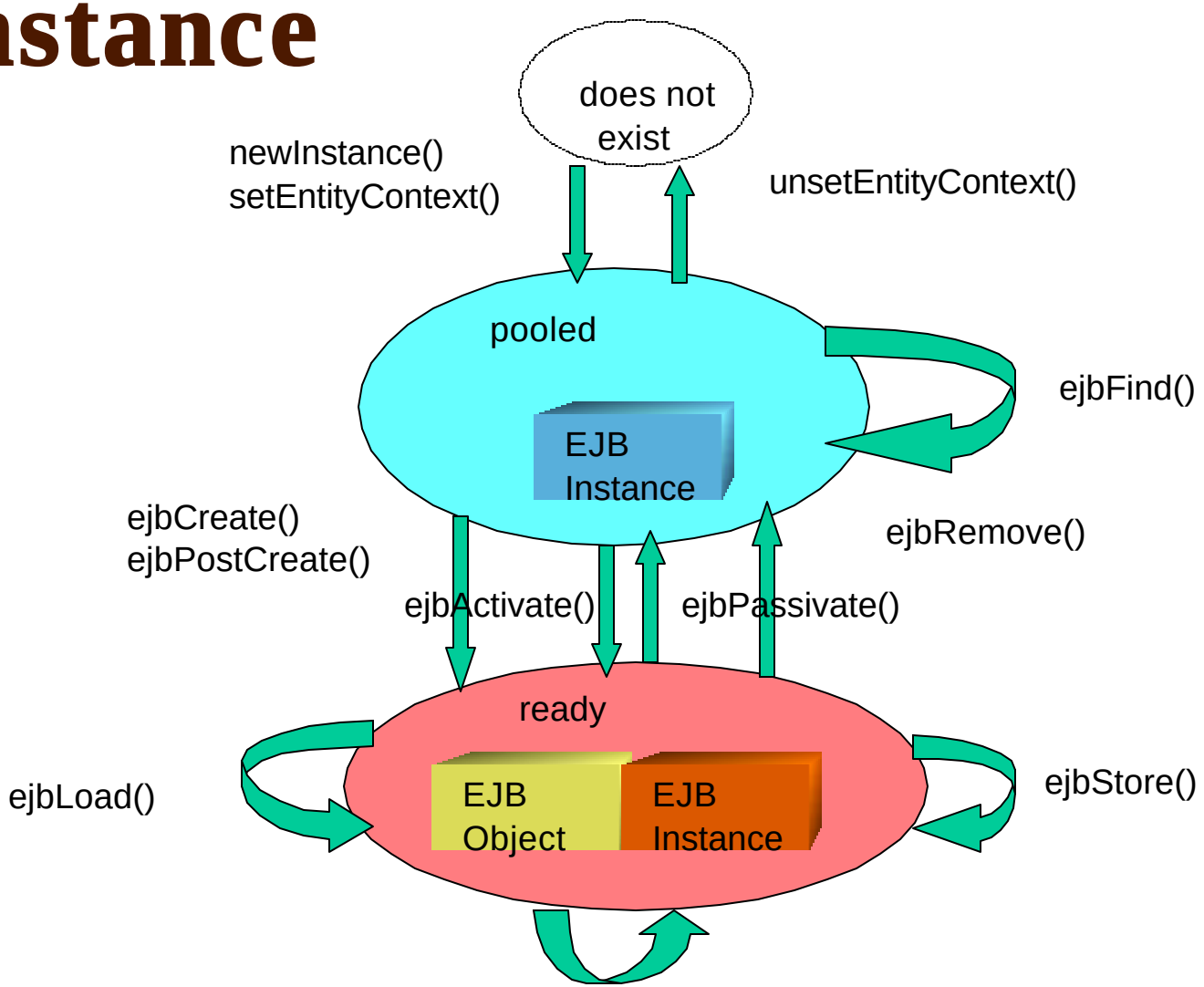
Home interface of Session Beans

- create() method does not take arguments
- Single create() method
- create() does not create a piece of persistence data
- No findX() methods
- remove() does not remove a piece of persistence data

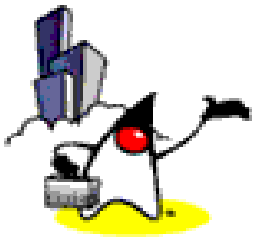
Home interface of Entity Beans

- create(...) methods take arguments
- Multiple create(...) methods
- create(...) creates a piece of persistent data (i.e. row of RDBS table)
- findByPrimaryKey() has to be provided
- multiple findX() methods
- remove() removes a piece of persistent data

Life cycle of an Entity Bean Instance



Role of the Container for Entity Beans



What Container does for Entity Beans

- Bean instance pool management
- Life cycle management of bean instances
- Transaction coordination
- Persistence management
  - Figuring out when to synchronize with database (for both BMP and CMP)
  - Providing optimized access to database (CMP)
- Security
  - Access control

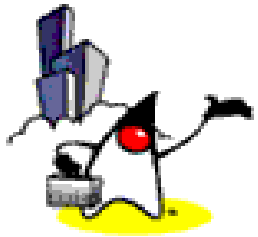
Life cycle and Pool management of Bean instances

- Container knows when is the best time to passivate and activate bean instances
  - based on the resource usage of the system
- Container knows when is the best time to create and remove bean instances

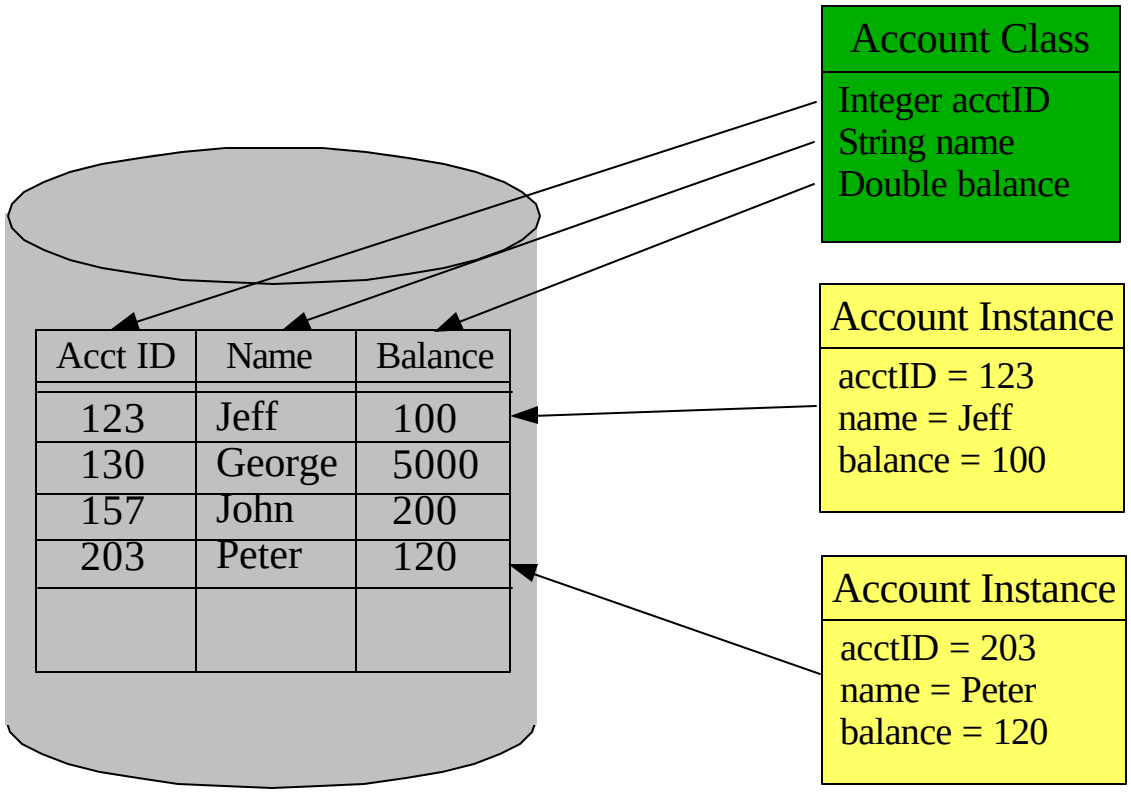
Transaction & Persistence Management

- Container knows when is the right time to synchronize state of bean instances with persistent data storage
  - Transaction semantics specified by bean developer/deployer are used
  - Container can use various information it maintains to optimize the synchronization with the persistent data storage
    - dirty writing, caching, optimistic locking, lazy loading
  - It calls ejbLoad() and ejbStore() at right time

Object View of Database



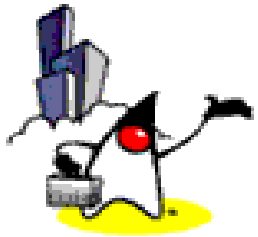
An object view of an RDBMS



Entity beans and Persistent Data type

- Entity beans can represent any persistent data
  - typically relational database
  - object database
  - data model of a legacy application
  - data model of EIS (PeopleSoft, SAP, etc.)
- For BMP, programmers have to code persistent data type specific calls to access the data
    - JDBC for relational database
- For CMP, containers handle the access logic to the data

Primary Keys in Entity Beans



What is Primary Key?

- Any piece of data that uniquely and unequivocally identify a particular entity bean
  - If we know the primary key, we can always find the rest of the data for the entity bean
  - Entity beans with same primary keys are considered identical
- Container uses primary key to manage Entity bean instances and EJB objects

How is Primary Key represented?

- Represented by Java class
  - In many cases, standard Java classes such as java.lang.String or java.lang.Integer are used
    - CustomerID is java.lang.Integer type
  - If standard Java classes are not sufficient, then custom Java class can be used
    - If there is no ready-to-usable CustomerID field, that uniquely identifies an instance, a combination of fields can constitute a custom primary key

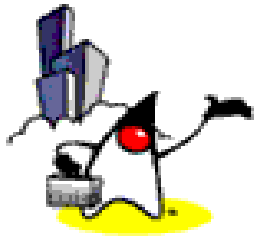
Custom Primary Key class

- Access control modifier of the class is public
- All fields are declared as public
- The fields have to be from the bean's persistent fields
- The class has a public default constructor
- The class implements the hashCode() and equals(Object other) methods
- The class is Serializable
  - they are being transported over the wire

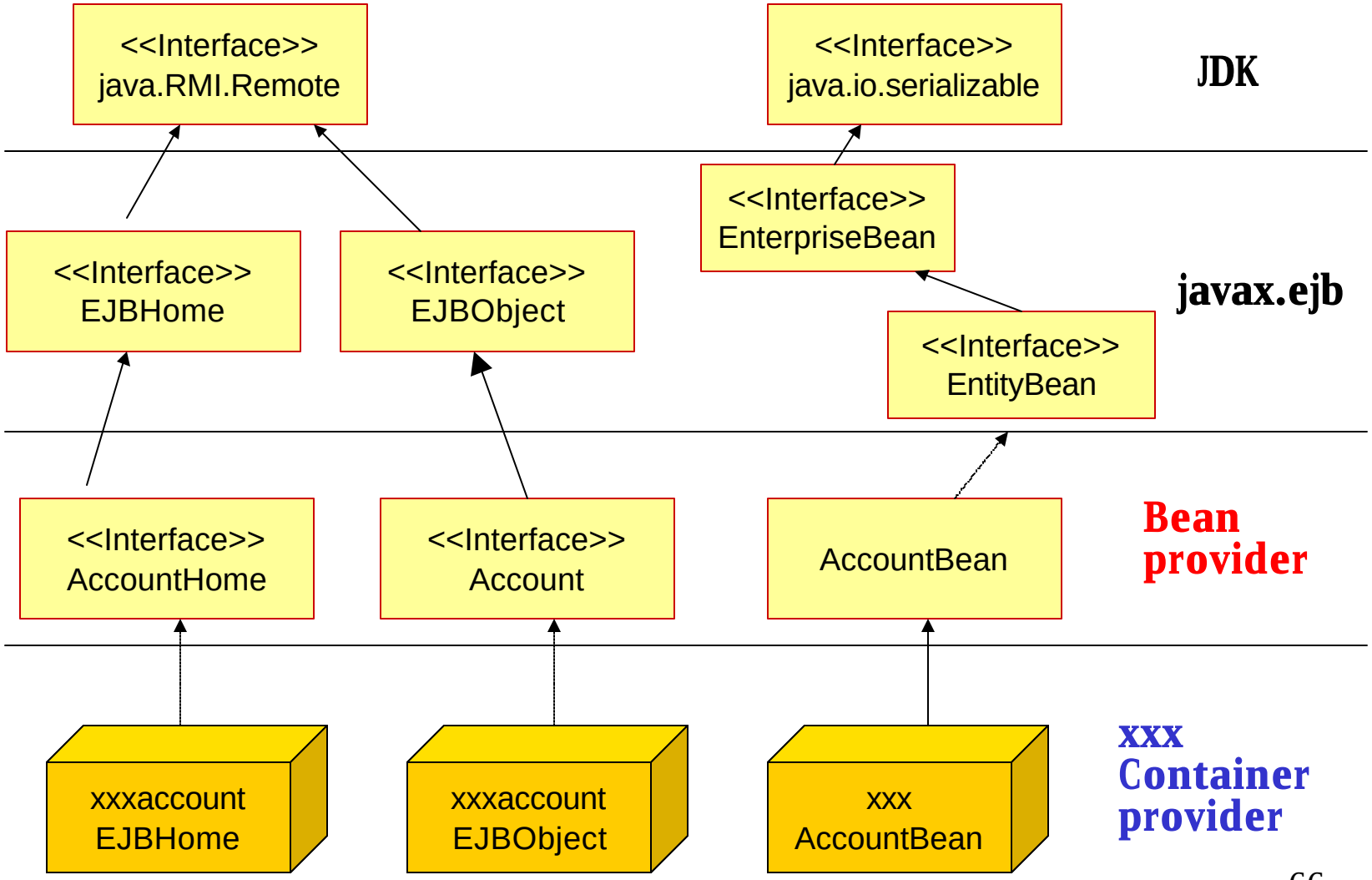
Primary Key Example

```
public class PurchaseOrderKey implements java.io.Serializable {
    public String productModel;
    public String vendorId;

    public PurchaseOrderKey() { };
    public String getProductModel() {
        return productModel;
    }
    public String getVendorId() {
        return vendorId;
    }
    public boolean equals(Object other) {
        if (other instanceof PurchaseOrderKey) { return
          (productModel.equals(((PurchaseOrderKey)other).productModel)
          && vendorId.equals( ((PurchaseOrderKey)other).vendorId));
        } return false;
    }
    public int hashCode() {
        return productModel.concat(vendorId).hashCode();
    }
}
```

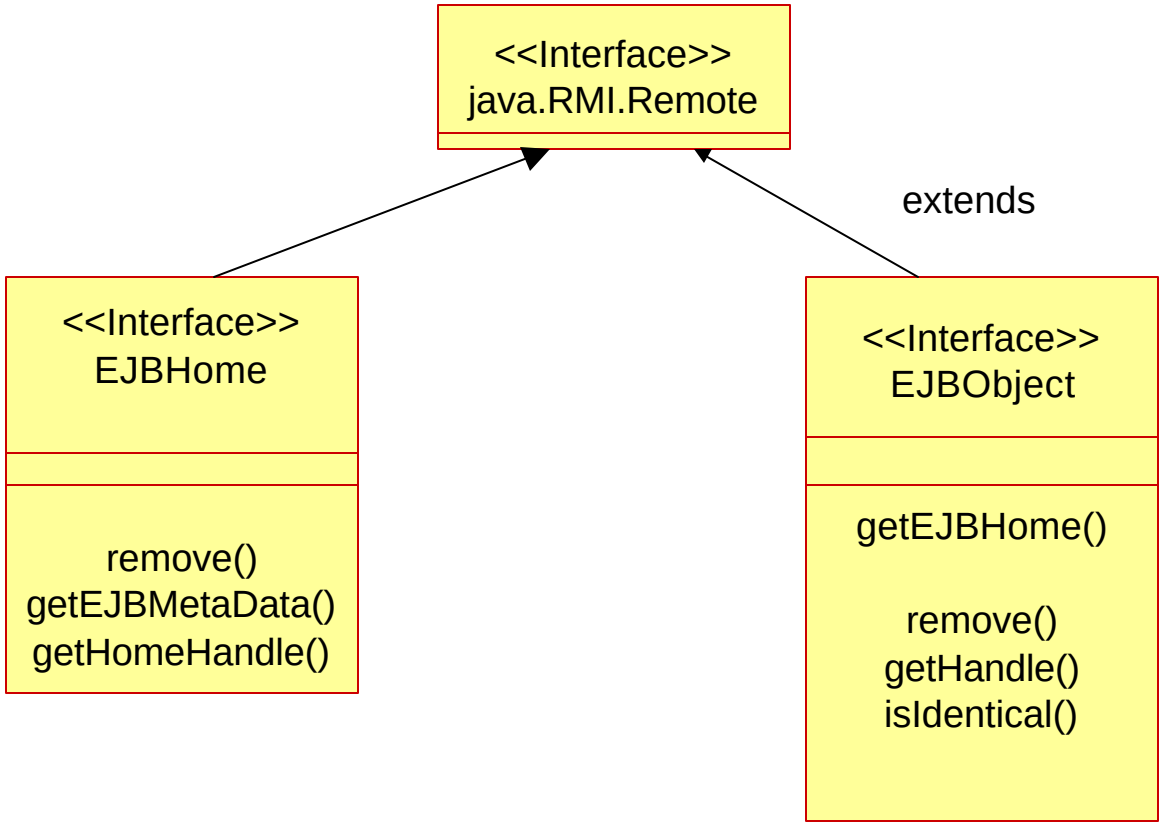
Programming Interface



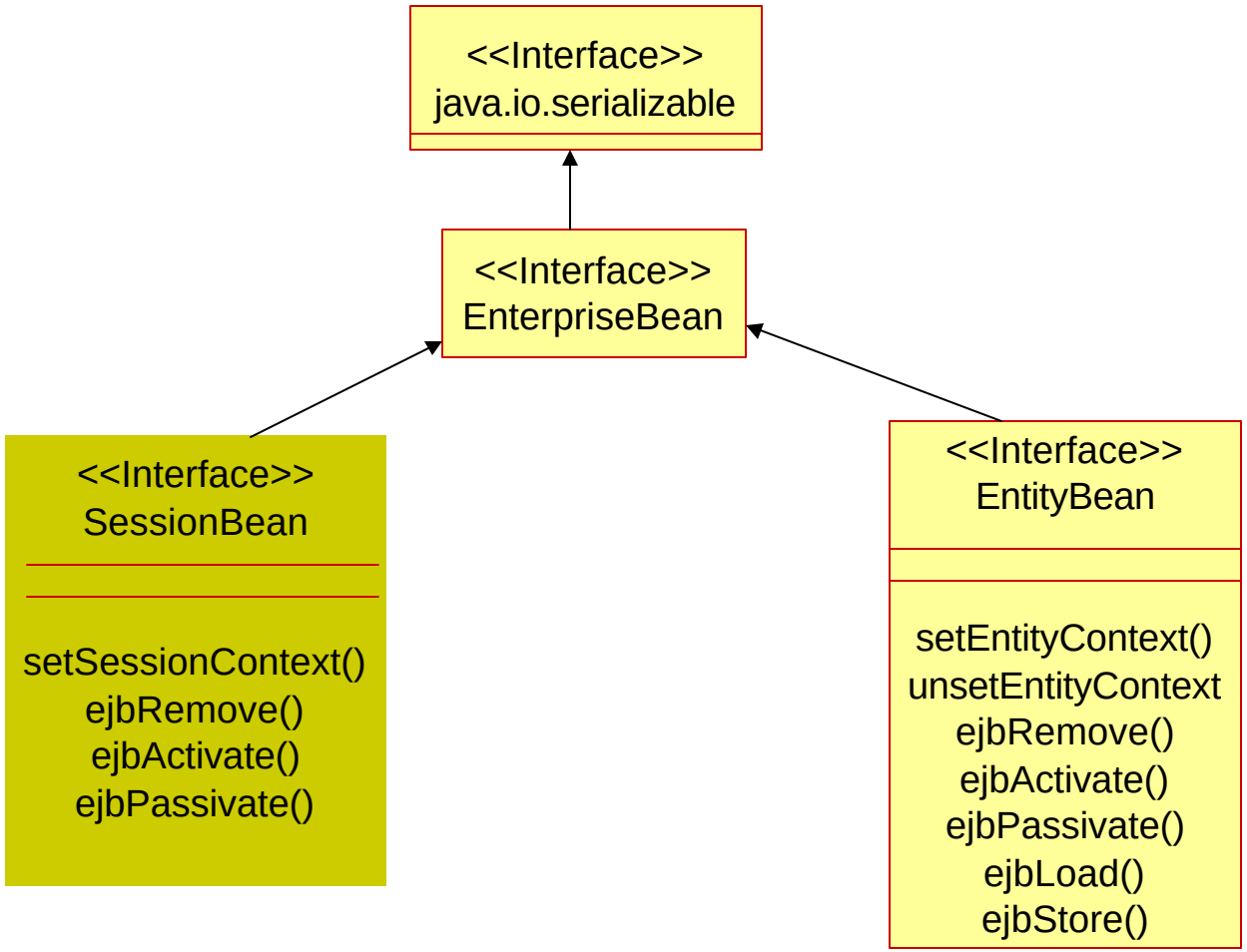
EJB™ Programming API



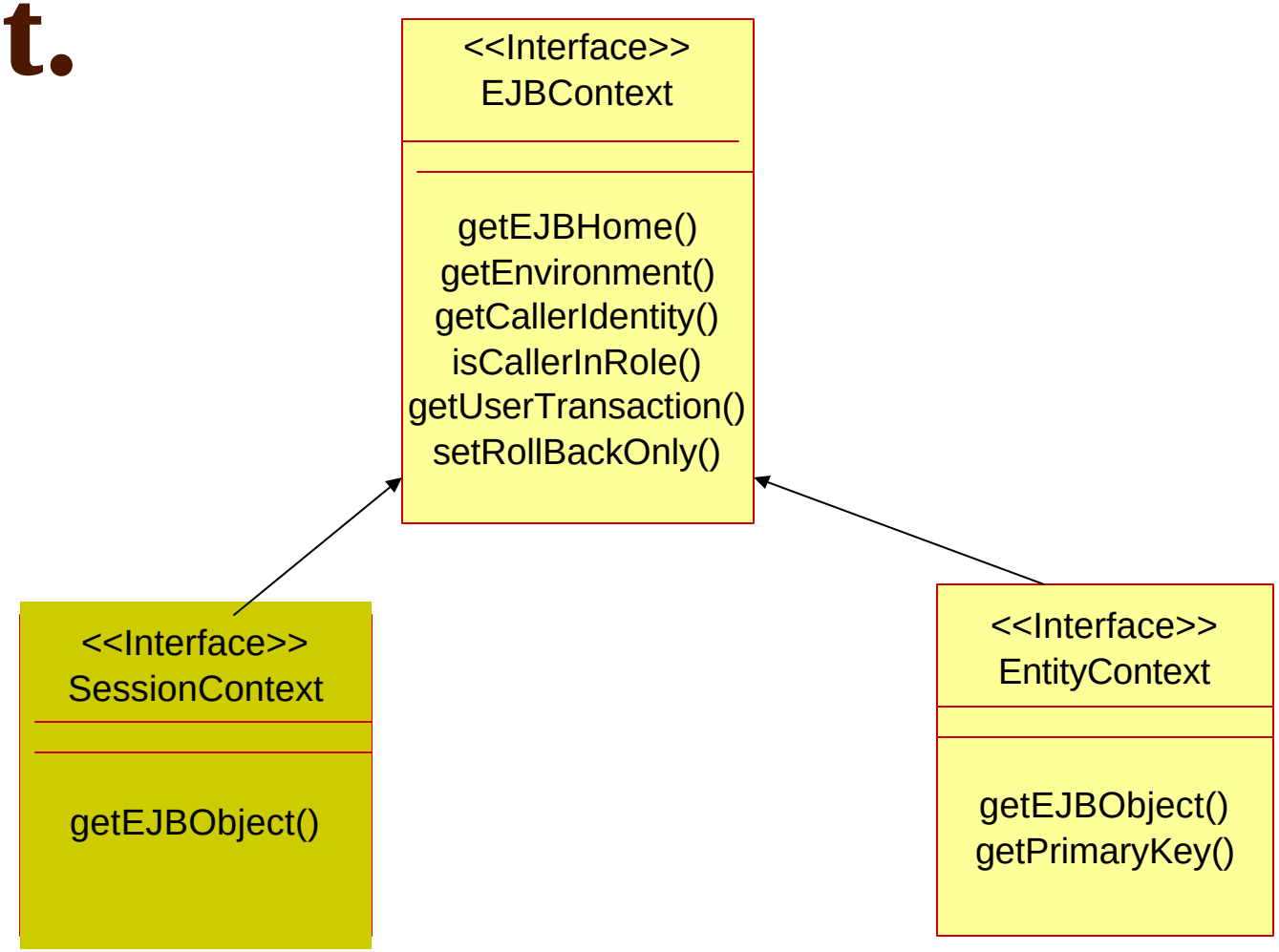
javax.EJB™ Client Interfaces



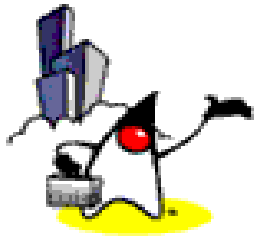
javax.EJB™ Server Interfaces



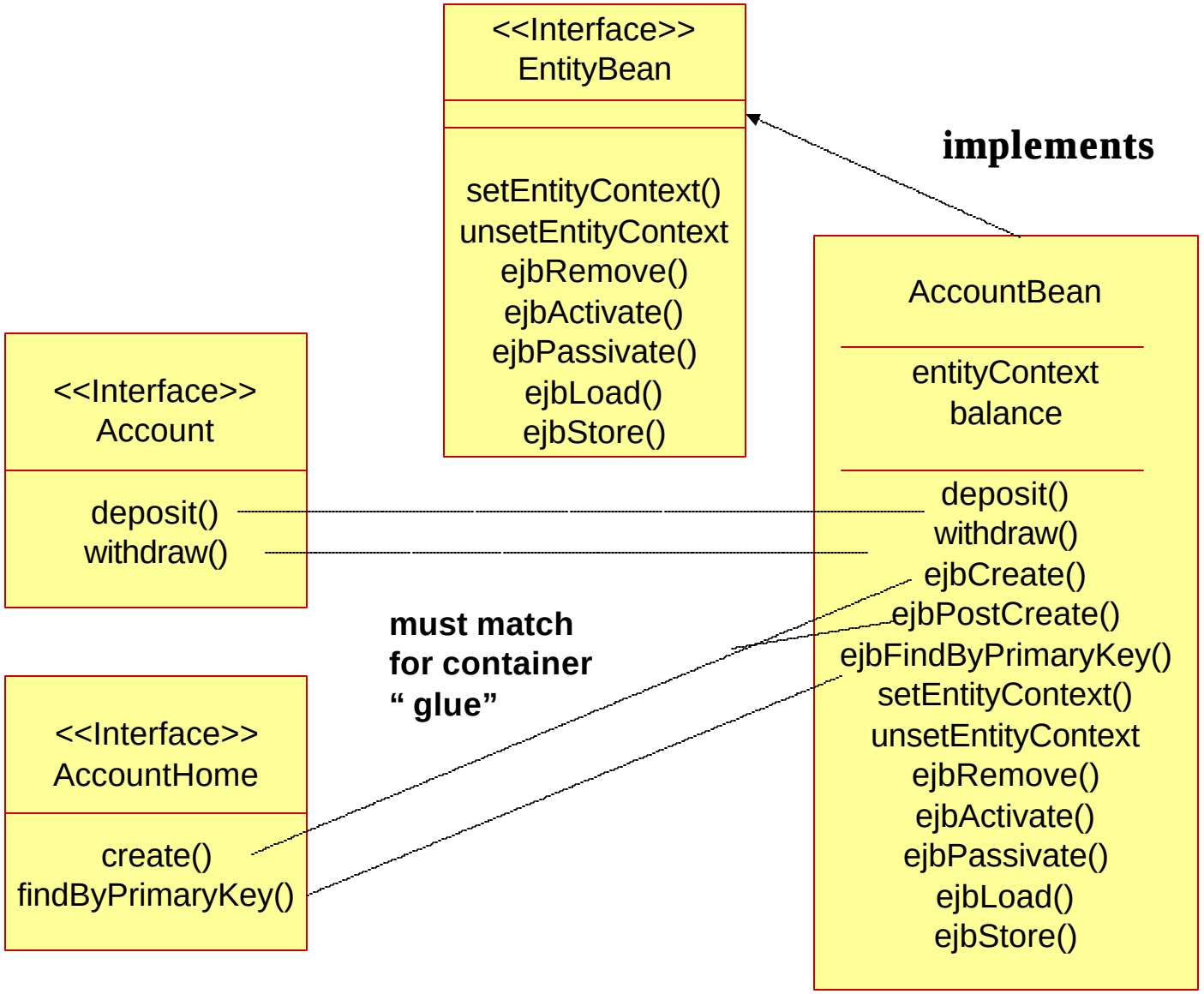
javax.EJB™ Server Interfaces Cont.



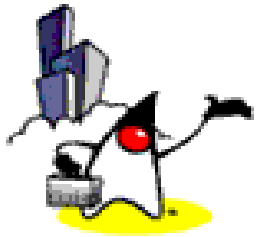
Method Relationship Among Home interface, Remote interface, Bean class



AccountBean Implementation



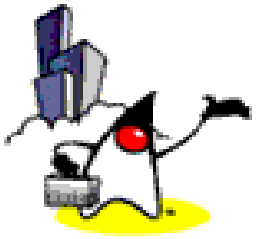
Codes Bean Developer writes in implementing Entity Bean (Server side)



Things Bean developer is responsible

- Home interface (local or remote)
  - findX() and create() methods
  - Home methods
- Remote interface (or Local interface)
  - business methods
- Bean class
  - implementation of business methods
- Deployment descriptor

Home Interface



Home interface (local or remote)

- Must extend javax.ejb.EJBHome (remote) or javax.ejb.EJBLocalHome (local) directly or indirectly
- Contains methods that allow a client to create and find an entity bean or beans
  - Multiple create(...) methods
  - Multiple findX(...) methods
- Contains Home methods

Home interface: Create(...) Methods

- Bean class must have matching ejbCreate(...) methods
- Returns the remote interface type of the enterprise bean (EJB object)
- Throws javax.ejb.CreateException

Home interface: FindX(...) Methods

- Bean class must have matching ejbFindX(...) methods
  - findByPrimaryKey(<primary-key>) is compulsory
- Returns the remote interface type of the enterprise bean (EJB object) or a collection of those types
- Throws javax.ejb.FindException

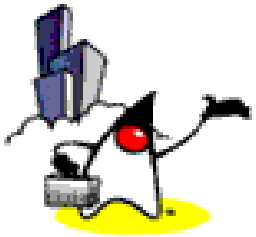
Home interface: Home Methods

- Can be arbitrary name X
- Bean class must have matching ejbHomeX(..) methods

Example: Home interface (Remote) Same for BMP and CMP

```
import java.util.Collection;import java.math.BigDecimal;
import java.rmi.RemoteException;import javax.ejb.*;

public interface SavingsAccountHome extends EJBHome {

   // Create(..) methods
   public SavingsAccount create(String id, String firstName,
      String lastName, BigDecimal balance)
      throws RemoteException, CreateException;

   // FindX(..)  methods
   public SavingsAccount findByPrimaryKey(String id)
      throws FinderException, RemoteException;
   public Collection findByLastName(String lastName)
      throws FinderException, RemoteException;
   public Collection findInRange(BigDecimal low,
      BigDecimal high)
      throws FinderException, RemoteException;

   // Home methods
   public void chargeForLowBalance(BigDecimal minimumBalance,  BigDecimal charge)
     throws InsufficientBalanceException, RemoteException;
}
```

Remote Interface (or Local Interface)



Remote interface (or Local interface)

- Must extend javax.ejb.EJBObject (remote interface) or javax.ejb.EJBLocalObject (local interface) directly or indirectly
- Contains business methods
  - For remote interface, arguments and return values must be valid RMI types
  - For local interface, arguments and return values do not have to be valid RMI types
- Bean class must have matching business methods

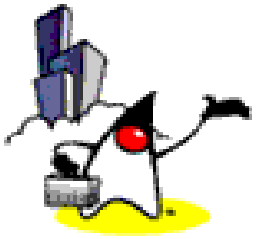
Three types of Business Methods

- Methods that manipulate properties
- Methods that manipulate relationships (associations)
- Methods that manipulate properties through value objects

Example: Remote interface

```
import javax.ejb.EJBObject;import java.rmi.RemoteException;
import java.math.BigDecimal;

public interface SavingsAccount extends EJBObject {

    // Business methods that manipulate properties
    public void debit(BigDecimal amount)
        throws InsufficientBalanceException, RemoteException;
    public void credit(BigDecimal amount)
        throws RemoteException;
    public String getFirstName()
        throws RemoteException;
    public String getLastName()
        throws RemoteException;
    public BigDecimal getBalance()
        throws RemoteException;

   // Business methods that manipulate relationships (this example has none)
   // Business methods that manipulate properties via value objects (this example
   // has none)
}
```

Bean Class



Bean Class

- Implement create(..), findX(..), and Home methods defined Home interface
  - ejbCreate(..), ejbFindX(..), ejbHomeX(..)
- Implement business methods defined in Remote interface (or Local interface)
- Implement methods of java.ejb.EntityBean interface
  - ejbLoad() & ejbStore()
  - ejbActivate() & ejbPassivate()
  - ejbRemove()
  - setEntityContext(EntityContext ctx) & unsetEntityContext()

Bean Class

- BMP and CMP implement methods of Bean class differently
  - In BMP, code contains persistence logic (database access)
  - In CMP, no persistence logic is required in the code, instead persistence logic is specified in deployment descriptor in declarative fashion
    - easier to develop beans
    - faster since container vendors can use various optimization techniques (especially in CMP 2.0)
    - can have ejbSelect(..) methods

ejbCreate(..) methods

- Performs
  - Inserts the entity state into the database in BMP
  - Initializes the instance variables
  - Returns the primary key
- There could be multiple ejbCreate(..) methods
  - must have matching create(..) methods in Home interface
- Must have matching ejbPostCreate() method

Example: BMP ejbCreate(..)

```
public String ejbCreate(String id, String firstName,
  String lastName, BigDecimal balance)
  throws CreateException {

  if (balance.signum() == -1)  {
    throw new CreateException("A negative initial balance is not allowed.");
  }

  // Perform database insert a row operation here.
  try {
    insertRow(id, firstName, lastName, balance);
  } catch (Exception ex) {
     throw new EJBException("ejbCreate: " +
       ex.getMessage());
  }
  // Initialize instance variables
  this.id = id;
  this.firstName = firstName; this.lastName = lastName;
  this.balance = balance;

  // Retunr primary key
  return id;
}
```

ejbPostCreate() methods

- Invoked after ejbCreate() is invoked
  - After Primary key and EJB object have been assigned
- ejbPostCreate() method can invoke the getPrimaryKey() and getEJBObject() methods of the EntityContext interface
  - if it needs to pass EJB Object as a parameter
- Typically empty

Example: BMP ejbPostCreate()

```
public void ejbPostCreate(String id, String firstName,
      String lastName, BigDecimal balance) {
}
```

ejbFindX(...) methods

- BMP
  - Performs SQL SELECT operations inside the code
- CMP
  - Container performs SQL SELECT operations itself based on the EJB-QL logic declared in the deployment descriptor

Example: BMP ejbFindX(..), Single EJB

```
public String ejbFindByPrimaryKey(String primaryKey)
  throws FinderException {

  boolean result;

  // Perform database operation here
  try {
    result = selectByPrimaryKey(primaryKey);
  } catch (Exception ex) {
    throw new EJBException("ejbFindByPrimaryKey: " +
      ex.getMessage());
  }
  // Return primary key
  if (result) {
    return primaryKey;
  }
  else {
    throw new ObjectNotFoundException
      ("Row for id " + primaryKey + " not found.");
  }
}
```

Example: BMP ejbFindX(..), Multi-EJB

```
public Collection ejbFindByLastName(String lastName)
  throws FinderException {

  Collection result;

  // Perform database operation here
  try {
    result = selectByLastName(lastName);
  } catch (Exception ex) {
    throw new EJBException("ejbFindByLastName " +
      ex.getMessage());
  }

  // Return Collection of primary keys
  return result;
}
```

ejbHomeX(...) methods

- Contains the business logic that is **not** tied with a particular bean instance
  - Normal business methods are always tied with a particular bean instance (with uniqie identity)
  - Cannot access persistent instances nor persistent relationships
  - Example:
    - return total value of purchase orders

Example: BMP ejbHomeX(..)

```
public void ejbHomeChargeForLowBalance(
    BigDecimal minimumBalance, BigDecimal charge)
    throws InsufficientBalanceException {

  try {
      SavingsAccountHome home =
      (SavingsAccountHome)context.getEJBHome();
      Collection c = home.findInRange(new BigDecimal("0.00"),
          minimumBalance.subtract(new BigDecimal("0.01")));

      Iterator i = c.iterator();
      while (i.hasNext()) {
        SavingsAccount account = (SavingsAccount)i.next();
        if (account.getBalance().compareTo(charge) == 1) {
          account.debit(charge);
        }
      }
  } catch (Exception ex) {
      throw new EJBException("ejbHomeChargeForLowBalance: "
          + ex.getMessage());
  }
}
```

Business methods

- Contains business logic
- Implementation of business methods defined in Remote interface (or Local interface)

Example: BMP Business Methods

```
public void debit(BigDecimal amount) throws InsufficientBalanceException {
   if (balance.compareTo(amount) == -1) {
      throw new InsufficientBalanceException();
   }
   balance = balance.subtract(amount);
}

public void credit(BigDecimal amount) {
   balance = balance.add(amount);
}

public String getFirstName() {
   return firstName;
}

public String getLastName() {
   return lastName;
}

public BigDecimal getBalance() {
   return balance;
}
```

ejbLoad() & ejbStore() methods

- For BMP, ejbLoad() contains code for retrieving data from the database while ejbStore() contains code for writing data to the database
- For CMP, they contains nothing

ejbLoad() & ejbStore() and Transaction

- ejbLoad() and ejbStore() always executes under a transaction context determined by the value of the transaction attribute in the deployment descriptor
  - If business method is running under a transaction, the container invokes ejbLoad() before the business method executes
  - Immediately after the business method executes, the container calls ejbStore()

```
public void ejbLoad() {

   // read data from the database via SQL SELECT
   try {
      loadRow();
   } catch (Exception ex) {
      throw new EJBException("ejbLoad: " +
         ex.getMessage());
   }
}

public void ejbStore() {

   // write data to the database via SQL UPDATE
   try {
      storeRow();
   } catch (Exception ex) {
      throw new EJBException("ejbStore: " +
         ex.getMessage());
   }
}
```

ejbActivate() & ejbPassivate()

- Container calls ejbActivate() when Bean instance is taken out of the pool of available instances to become associated with a specific EJB object
- Container invokes ejbPassivate() on an instance before the instance becomes disassociated with a specific EJB object
  - After this method completes, the container will place the instance into the pool of available instances

```
public void ejbActivate() {

   id = (String)context.getPrimaryKey();
}

public void ejbPassivate() {

   id = null;
}
```

ejbRemove() method

- A container invokes this method before it removes the EJB object that is currently associated with the instance
- This method is invoked when a client invokes a remove() operation on the enterprise Bean's home interface or the EJB object's remote interface
- This method transitions the Bean instance from the ready state to the pool of available instances

Example: BMP ejbRemove(..)

```
public void ejbRemove() {

    // remove a row from database
    try {
        deleteRow(id);
    catch (Exception ex) {
        throw new EJBException("ejbRemove: " +
        ex.getMessage());
    }
}
```

setEntityContext(EntityContext ctx) & unsetEntityContext()

- Container calls setEntityContext() after it creates Bean instance
- Container calls unsetEntityContext() after it removes a Bean instance
  - This is the last method Container calls to the bean instance
  - The instance will be garbage-collected

Example: BMP setEntityContext(.) & unsetEntityContext()

```
public void setEntityContext(EntityContext context) {

   this.context = context;
   try {
      makeConnection();
   } catch (Exception ex) {
       throw new EJBException("Unable to connect to database. " +
          ex.getMessage());
   }
}

public void unsetEntityContext() {

   try {
      con.close();
   } catch (SQLException ex) {
       throw new EJBException("unsetEntityContext: " + ex.getMessage());
   }
}
```

ejbSelectX() method

- Only for CMP 2.0
- Abstract methods in the same way as other abstract getter methods of regular CMP fields
- Used only internally by EJB for selection logic
  - Not exposed to clients
- Selection logic is expressed in the deployment descriptor in the same way finder logic is expressed
- Can return not only beans but also dependent value objects

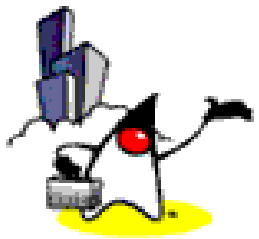
Example: CMP 2.0 ejbSelectX(..)

```
public abstract class PlayerBean implements EntityBean {

    private EntityContext context;

    // Access methods for persistent fields
    public abstract String getPlayerId();
    public abstract void setPlayerId(String id);
     ...

    // Access methods for relationship fields
    public abstract Collection getTeams();
    public abstract void setTeams(Collection teams);

    // Select methods
    public abstract Collection ejbSelectLeagues(LocalPlayer player)
        throws FinderException;
    public abstract Collection ejbSelectSports(LocalPlayer player)
        throws FinderException;
```

Example: CMP 2.0 ejbSelectX(..) Deployment Descriptor

```
<entity>
  <display-name>PlayerEJB</display-name>
  <ejb-name>PlayerEJB</ejb-name>
  ...
  <query>
    <description></description>
    <query-method>
      <method-name>ejbSelectLeagues</method-name>
      <method-params>
        <method-param>team.LocalPlayer</method-param>
      </method-params>
    </query-method>
    <result-type-mapping>Local</result-type-mapping>
    <ejb-ql>select distinct t.league from Player p, in (p.teams) as t
where p = ?1</ejb-ql>
  </query>
  ...
</entity>
```

Codes Client programmer has to write (Client side)



Client view

- Whether an entity bean is implemented as BMP or CMP 1.0 or CMP 2.0 is completely transparent to client
  - Server side architecture and implementation can change without affecting the client

Example: Client code

```
public class SavingsAccountClient {

   public static void main(String[] args) {
      try {
         // Get EJB Home object via JNDI
         Context initial = new InitialContext();
         Object objref = initial.lookup("java:comp/env/ejb/SimpleSavingsAccount");
         SavingsAccountHome home =
            (SavingsAccountHome)PortableRemoteObject.narrow(objref,
                                 SavingsAccountHome.class);

         // Create EJB object (and add a row to the database)
         BigDecimal zeroAmount = new BigDecimal("0.00");
         SavingsAccount duke = home.create("123", "Duke", "Earl", zeroAmount);

         // Invoke business methods
         duke.credit(new BigDecimal("88.50"));
         duke.debit(new BigDecimal("20.25"));
         BigDecimal balance = duke.getBalance();
         System.out.println("balance = " + balance);
         // Remove EJB object (and remove it from the database)
         duke.remove();
```

Example: Client code

```
SavingsAccount joe = home.create("836", "Joe", "Jones", zeroAmount);
joe.credit(new BigDecimal("34.55"));
SavingsAccount jones = home.findByPrimaryKey("836");
jones.debit(new BigDecimal("2.00"));
balance = jones.getBalance();
System.out.println("balance = " + balance);

SavingsAccount pat = home.create("456", "Pat", "Smith", zeroAmount);
pat.credit(new BigDecimal("44.77"));
SavingsAccount john = home.create("730", "John", "Smith", zeroAmount);
john.credit(new BigDecimal("19.54"));
SavingsAccount mary = home.create("268", "Mary", "Smith", zeroAmount);
mary.credit(new BigDecimal("100.07"));

Collection c = home.findByLastName("Smith");
Iterator i=c.iterator();

while (i.hasNext()) {
   SavingsAccount account = (SavingsAccount)i.next();
   String id = (String)account.getPrimaryKey();
   BigDecimal amount = account.getBalance();
   System.out.println(id + ": " + amount);
}
```

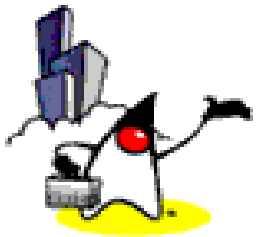
Example: Client code

```
c = home.findInRange(new BigDecimal("20.00"),
    new BigDecimal("99.00"));
i=c.iterator();

while (i.hasNext()) {
   SavingsAccount account = (SavingsAccount)i.next();
   String id = (String)account.getPrimaryKey();
   BigDecimal amount = account.getBalance();
   System.out.println(id + ": " + amount);
}

SavingsAccount pete = home.create("904", "Pete", "Carlson",
   new BigDecimal("5.00"));
SavingsAccount sally = home.create("905", "Sally", "Fortney",
   new BigDecimal("8.00"));

home.chargeForLowBalance(new BigDecimal("10.00"),
   new BigDecimal("1.00"));
```

Example: Client code

```
            BigDecimal reducedAmount = pete.getBalance();
            System.out.println(reducedAmount);
            reducedAmount = sally.getBalance();
            System.out.println(reducedAmount);

            System.exit(0);

        } catch (InsufficientBalanceException ex) {
            System.err.println("Caught an InsufficientBalanceException: "
                               + ex.getMessage());

        } catch (Exception ex) {
            System.err.println("Caught an exception." );
            ex.printStackTrace();
        }
    }
}
```

Limitations of CMP 1.1



Issues with CMP 1.1

1. Application logic and schema mapping are entangled
2. Inefficiencies in using instance variables
3. Tight coupling of client view of persistence with EJB view
4. No control over granularity of client-EJB integration

source: Applied Enterprise JavaBeans [1]

Issue 1: Application logic and Schema mapping are entangled

- Container does not know the relationship between instance variables and the fields of database tables
  - Instance variable names are not the same column names of the table
- Container does not know how to perform finder logic of findX()
  - Semantical meaning of findCustomers(income) is not known to Container
- Container specific coding results non-portable code

Solutions to Issue 1

- Mapping of instance variables to database tables (Schema mapping) can be separated from application logic
- Schema mapping can be done at deployment time
- Finder logic should be separated from schema mapping
  - Schema mapping can be done without affecting the finder logic
- Finder logic should be defined in a portable fashion

Issue 2: Inefficiencies in using Instance Variables

- In CMP 1.1, instance variables are directly declared public within a bean class
- Container has no effective means to determine which instance variables will be read and written by business methods in any one transaction
  - Container has to read/write all the instance variables and persistent associations into/from their respective variables at the start/end of transaction
- CMP 1.1 container vendors are **not given any opportunity for optimization**

Solutions to Issue 2

- Allow container to load values of instance variables only when they are needed
  - It is called "lazy-loading"
- Allow container to write only the instance variables that were actually changed during a transaction
  - It is called "selective writing"
- Allow container to load and write persistent relationship only when needed
  - Lazy-loading and selective writing for relationships

Issue 3: Tight coupling of client view of finder logic with one of EJB view

- There is a need for EJB to perform internal finder logic without exposing it to the client
- In CMP 1.1, there is no distinction between client's finder logic and internal finder logic
  - This is problematic in the management of one-to-many (1:N) relationship

Solutions to Issue 3

- Allow EJB to locate persistent associations using internal-usage only finder method
  - while not exposing the persistent associations directly to clients
  - De-couple client view of finder logic from that of internal EJB's

Issue 4: No control over granularity of client-EJB integration

- In CMP 1.1, all persistent objects have to be represented as EJB's unless the developer wants to use helper classes and manage the persistence in code
  - If one object is dependent on another (dependent object), it does not have to be EJB and it does not have to be exposed to clients
  - Requiring dependent objects to be in EJB results in unnecessary overhead

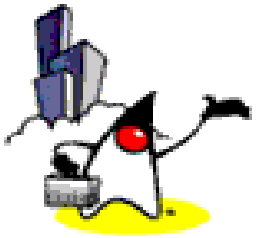
Solutions to Issue 4

- **Define a standard way of managing the relationship between entity bean and its dependent objects**

A few words on CMP 1.1

- Due to these limitations, CMP 1.1 is not considered to be useful, thus is not used widely
  - Many complaints on EJB's (un)usefulness are in fact related to the immaturity of CMP 1.1
- Yet, the concept of Container Managed Persistence (CMP) is a good one
- Interim solution products such as Toplink or Cocobase give developers “Good CMP like” features but in a proprietary way

CMP 2.0



CMP 2.0

- Addresses all the limitations of CMP 1.1
- There is no excuse not to use CMP 2.0 now
- Part of J2EE 1.3
  - Pretty much all J2EE app servers are now J2EE 1.3 compliant, thus supports CMP 2.0

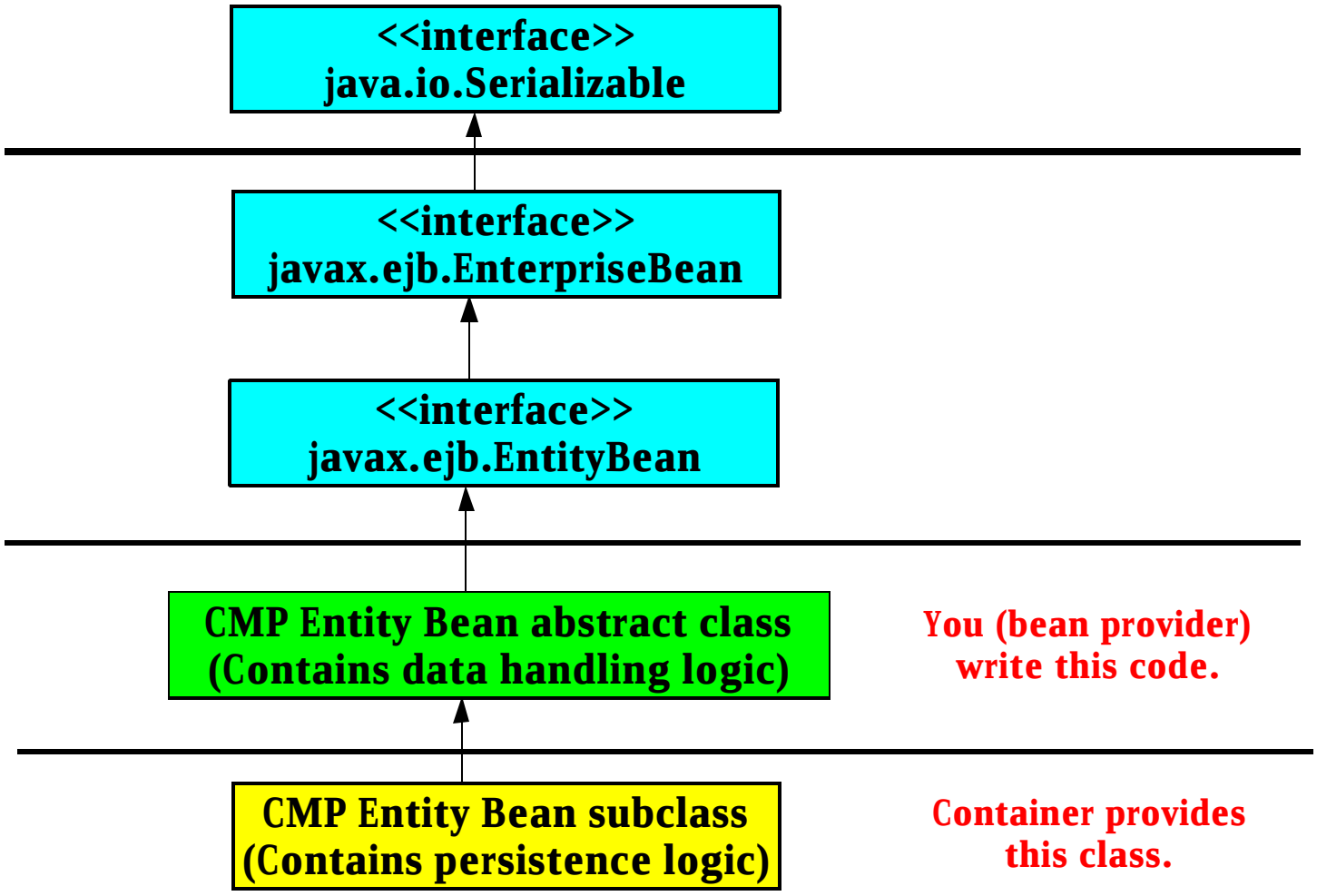
CMP 2.0 Entity Beans

- CMP Entity Bean is now an abstract class
  - CMP entity bean is subclass'ed (concretely implemented) by container-generated subclass
- CMP Entity Beans have no declared fields
  - Instead, they have abstract getter/setter methods
  - These getter/setter methods are implemented in container-generated subclass
- Contract between CMP entity bean and container is specified in Abstract Persistence Schema

Why CMP 2.0 Entity Bean?

- Enables separation of data handling logic from persistence logic
- Data handling logic is coded in the bean by bean provider
- **Persistence logic is provided by container**
  - Container handles JDBC calls
  - Container can employ various **optimization techniques**
    - lazy loading, dirty checking and update, caching, optimistic concurrency control

CMP 2.0 Entity Beans are Subclassed



Example: CMP 2.0 Entity Bean

```
public abstract class AccountBean implements EntityBean{
    // No fields

    // Abstract getter/setter methods
    public abstract String getOwnerName();
    public abstract void setOwnerName(String ownerName);

    // Data logic methods using the above getter/setter methods
    ...
    // Business logic methods
    ...
    // EJB required methods follow
    ...
}
```

CMP 2.0 Abstract Persistence Schema

- Declaratively expressed in the deployment descriptor
- Created by bean provider to instruct Container on how to implement persistence logic
- Container inspects the abstract persistence schema and generates subclasses accordingly
  - <cmp-field> elements represent container-generated and container-managed persistent fields

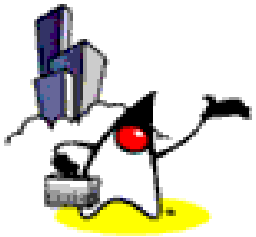
Example: Abstract Persistence Schema

```
...
<cmp-version>2.x</cmp-version>

<abstract-schema-name>AccountBean</abstract-schema-name>

<cmp-field>
    <field-name>accountID</field-name>
</cmp-field>

<cmp-field>
    <field-name>ownerName</field-name>
</cmp-field>

...
```

BMP versus CMP 2.0

	CMP 2.0	BMP
Class definition	Abstract	Not abstract
Database access	Generated by tools	Coded by developers
Persistent state & Access methods for persistent and relation fields	Represented by virtual persistent fields in deployment descriptor	Coded as instance variables
findbyPrimaryKey	Required	None
Customized finder methods	Handled by container	Coded by developers
Select methods	Handled by container but developer must defined EJB QL queries	Coded by developers
Return value of ejbCreate()	Should be null	Must be primary key

Relationships (Associations)



Relationship (Association)

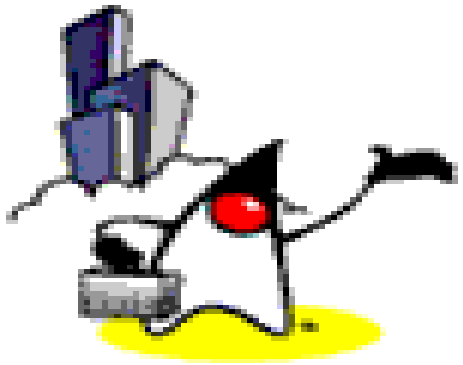
- Entity beans typically have relationship with other Entity beans
  - Reflecting relationship between database tables
- These relationships also have to be persistent
  - In addition to state of entity bean (instance variables)

Relationship Handling

- In BMP, bean developers have to handle these relationships in their code
- In CMP 1.1, the relationship management is done by container-specific schemes (or 3rd-party proprietary tool)
  - Code is not portable
- In CMP 2.0, the relationship is declared in deployment descriptor in standard fashion
  - Code is now portable

Types of Relationships

- Cardinality
- Directionality
- Aggregation vs. composition
  - Cascading delete
- Recursive, Circular, and lazy-loading relationships
- Referential integrity

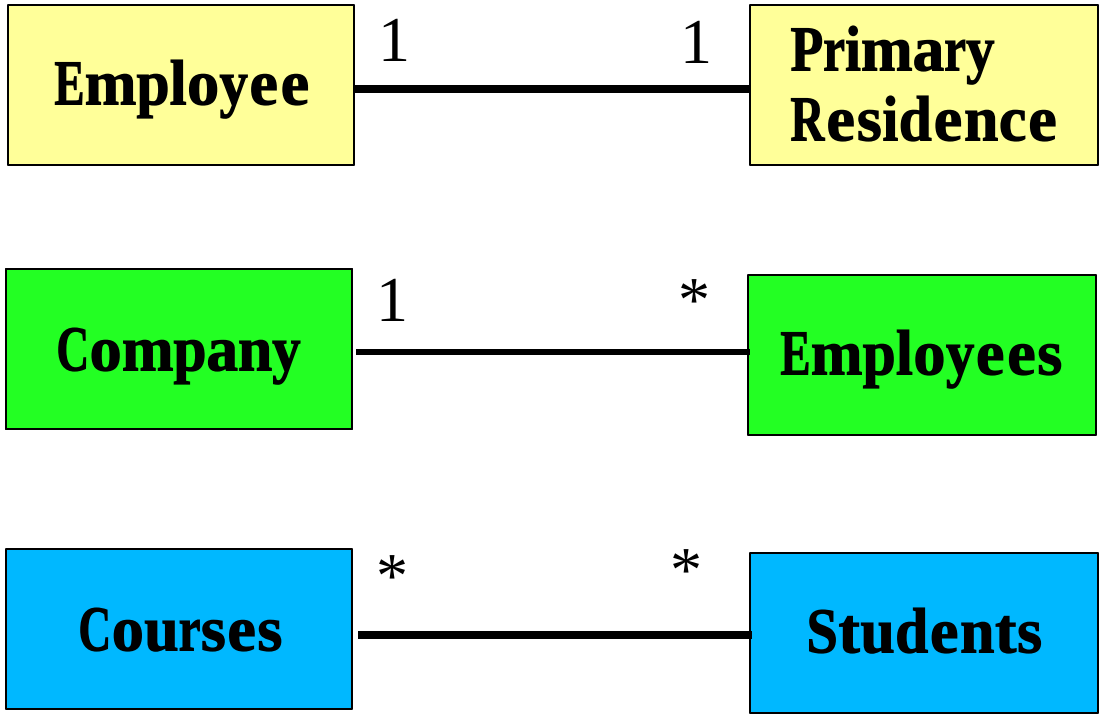


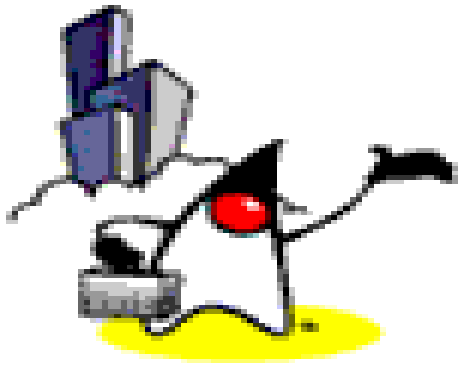
Cardinality Relationship

Cardinality Relationship

- Specifies how many instances of data can participate in a relationship
- Three types of cardinality
  - 1:1 relationship
    - employee and his primary residence
  - 1:N relationship
    - company and its employees
  - M:N relationship
    - courses and students

Cardinality Relationship





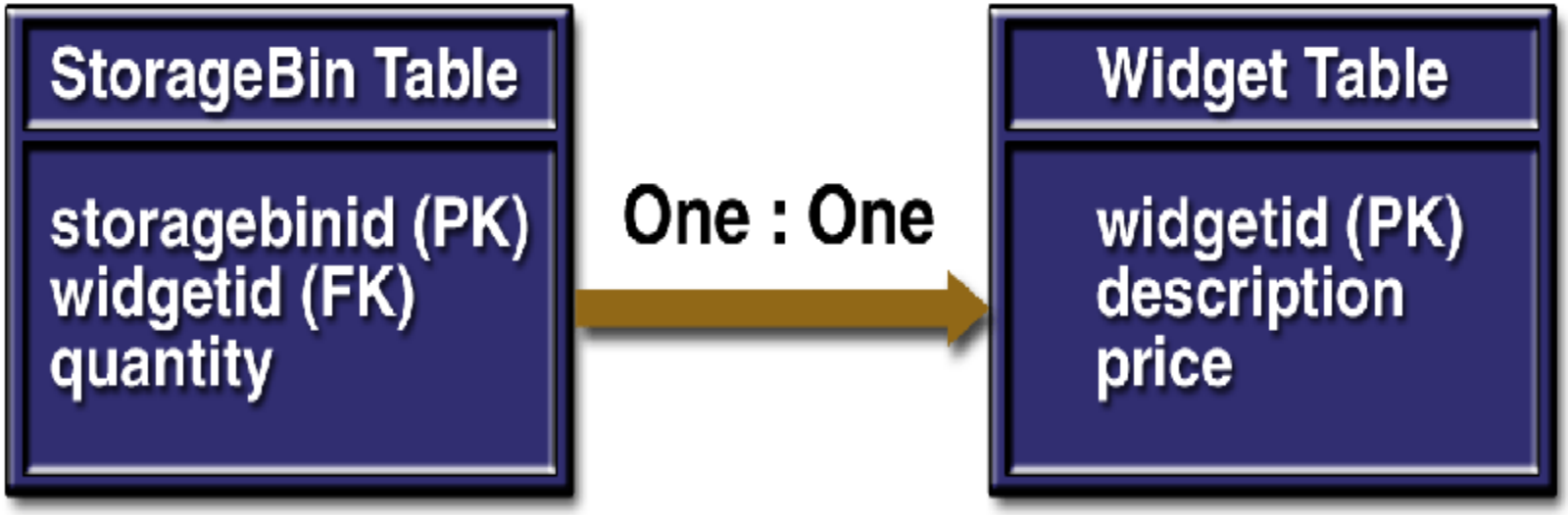
1:1 Relationship

1:1 Relationship

- Each row in a table is related to a single row in another table
- A dependent (child) table includes a foreign key that matches the primary key of the referenced (parent) table

Example: 1:1 Relationship

- Values of the foreign keys in the Storagebin (child) table depend on the primary keys in the Widget (parent) table



source: J2EE tutorial[2]

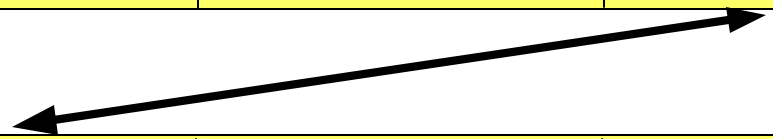
1:1 Cardinality Database Schema

StoragebinID(PK)	WidgetID (ForeignPK)	Quantity
234	456	500

WidgetID(PK)	Description	Price
456	Good widget	34

1:1 Cardinality Database Schema

OrderPK	OrderName	Shipment ForeignPK
234	Some order	5001



ShipmentPK	City	ZipCode
5001	Gotham City	02420

source: Mastering EJB [3]

1:1 Relationship Implementation in BMP

```
public class OrderBean implements EntityBean {

   private String orderPK;
   private String orderName;
   private Shipment shipment; // EJB local object stub

   public Shipment getShipment() { return shipment;}
   public void setShipment(Shipment s) { this.shipment = s;}
   ...
   public void ejbLoad(){
      // 1. SQL SELECT Order. This also retrieves the shipment foreign key.
      // 2. JNDI lookup of ShipmentHome
      // 3. Call ShipmentHome.findByPrimaryKey(), passing in the shipment foreign key
   }

   public void ejbStore(){
      // 1. Call shipment.getPrimaryKey() to retrieve the Shipment foreign key
      // 2. SQL UPDATE Order.  This also stores the shipment foreign key.
   }
}
```

source: Mastering EJB [3]

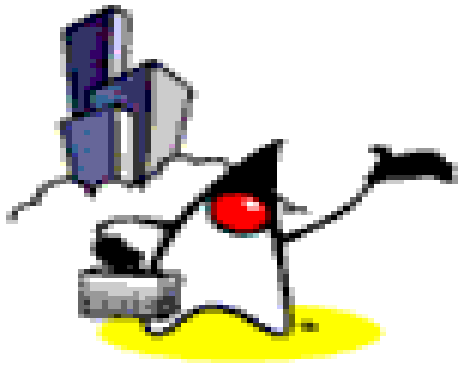
1:1 Relationship Implementation in CMP

```
public abstract class OrderBean implements EntityBean {
   // Note no fields

   // Abstract getter and setter methods
   public abstract Shipment getShipment();
   public abstract void setShipment(Shipment s);
   ...
   // Empty ejbLoad() and ejbStore()
   public void ejbLoad(){
   }
   public void ejbStore(){
   }
}
```

source: Mastering EJB [3]

1:1 Relationship Implementation in CMP

```
<ejb-jar>
  <enterprise-beans>
  ...
  </enterprise-beans>
  <relationships>
    <ejb-relation>
      ... define relationship
    </ejb-relation>
    ...
  </relationships>
```

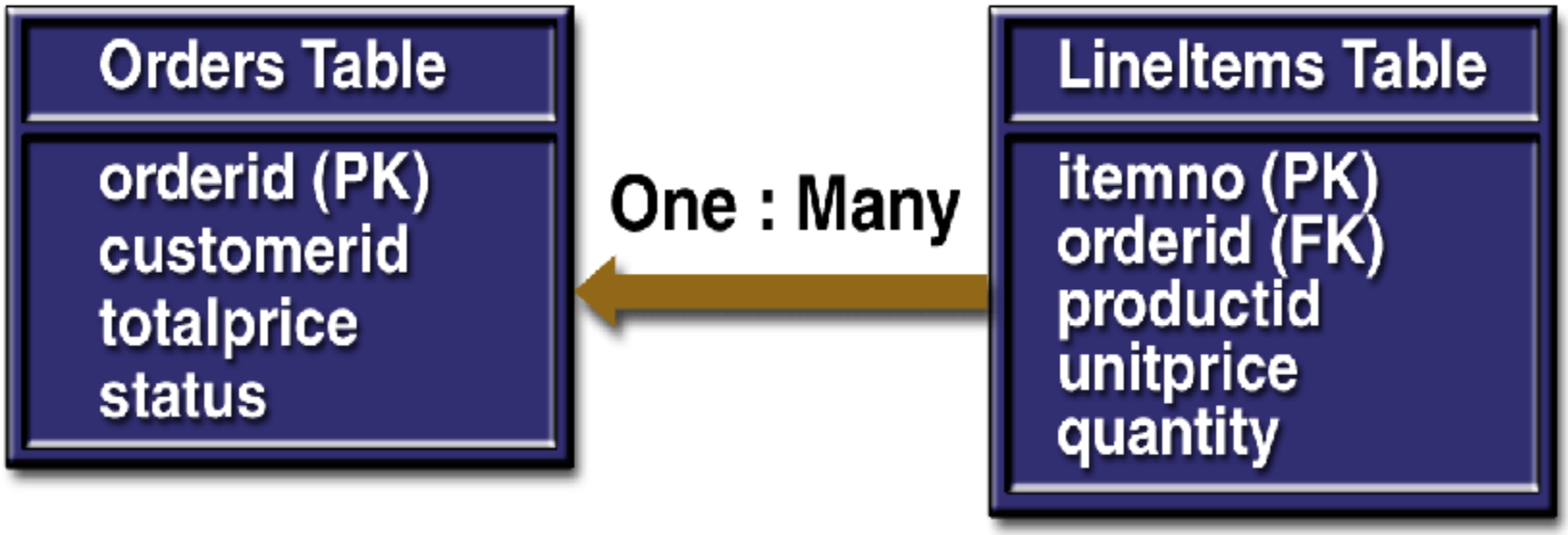


1:N Relationship

1:N Relationship

- More common relationship than 1:1
  - Because objects in 1:1 relationship can be easily combined into a single object
- Typically set up via a foreign key relationship
- Examples
  - Order:Lineitems
  - Custom:Orders
  - Company:Employees

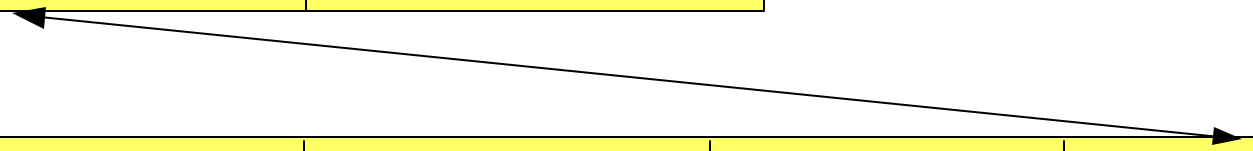
Example: 1:N Relationship



source: J2EE tutorial[2]

1:N Cardinality Database Schema

CompanyPK	Name
234	Acme



EmployeePK	Name	Sex	Company
5001	Sang	M	234
5002	Young	F	234

source: Mastering EJB [3]

1:N Relationship Handing in BMP

```
public class CompanyBean implements EntityBean{
   private String companyPK;
   private String companyName;
   private Vector employees;

   public Collection getEmployees() {return employees;}
   public void setEmployees(Collection e){
      this.emplyees = (Vector) e;
   }
   ...
   public void ejbLoad() {
      // 1. SQL SELECT Company
      // 2. JNDI lookup of EmployHome
      // 3. EmployeeHome.findByCompany(companyPK)   multi- return
   }
   public void ejbStore() {
      // SQL UPDATE Company
   }
   ...
```

source: Mastering EJB[3]

1:N Relationship Handing in CMP

```
public abstract class CompanyBean implements EntityBean{

   // No fields

   // Abstract getter/setter methods
   public abstract Collection getEmployees();
   public abstract void setEmployees(Collection employees);
   ...
   // No relationship handling code
   // Relationships are specified in the deployment descriptor
   public void ejbLoad() {}   // Empty
   public void ejbStore() {}  // Empty
   ...
}
```

source: Mastering EJB[3]

1:N Relationship Handing in CMP

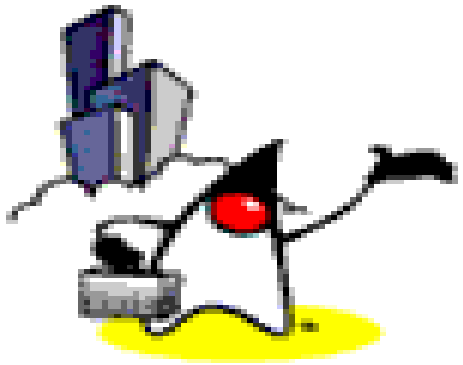
```
<ejb-jar>
  <enterprise-beans> ... </enterprise-beans>
  <relationships>
    <ejb-relation>
      <ejb-relation-name>Company-Employees</ejb-relation-name>

      <ejb-relastionship-role>
        <ejb-relationship-role-name> Company-Employs-Employees
        </ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source> <ejb-name>Company</ejb-name>
        </relationship-role-source>
        <cmr-field>
          <cmr-field-name>employees</cmr-field-name>
          <cmr-field-type>java.util.Collection</cmr-field-type>
        </cmr-field>
      </ejb-relationship-role>
```

source: Mastering EJB[3]

1:N Relationship Handing in CMP

```
        <ejb-relastionship-role>
           <ejb-relationship-role-name> Employees-WorkAt-Company
           </ejb-relationship-role-name>
           <multiplicity>Many</multiplicity>
           <relationship-role-source> <ejb-name>Employee</ejb-name>
           </relationship-role-source>
        </ejb-relationship-role>

    </ejb-relation>
  </relationships>
</ejb-jar>
```

source: Mastering EJB[3]

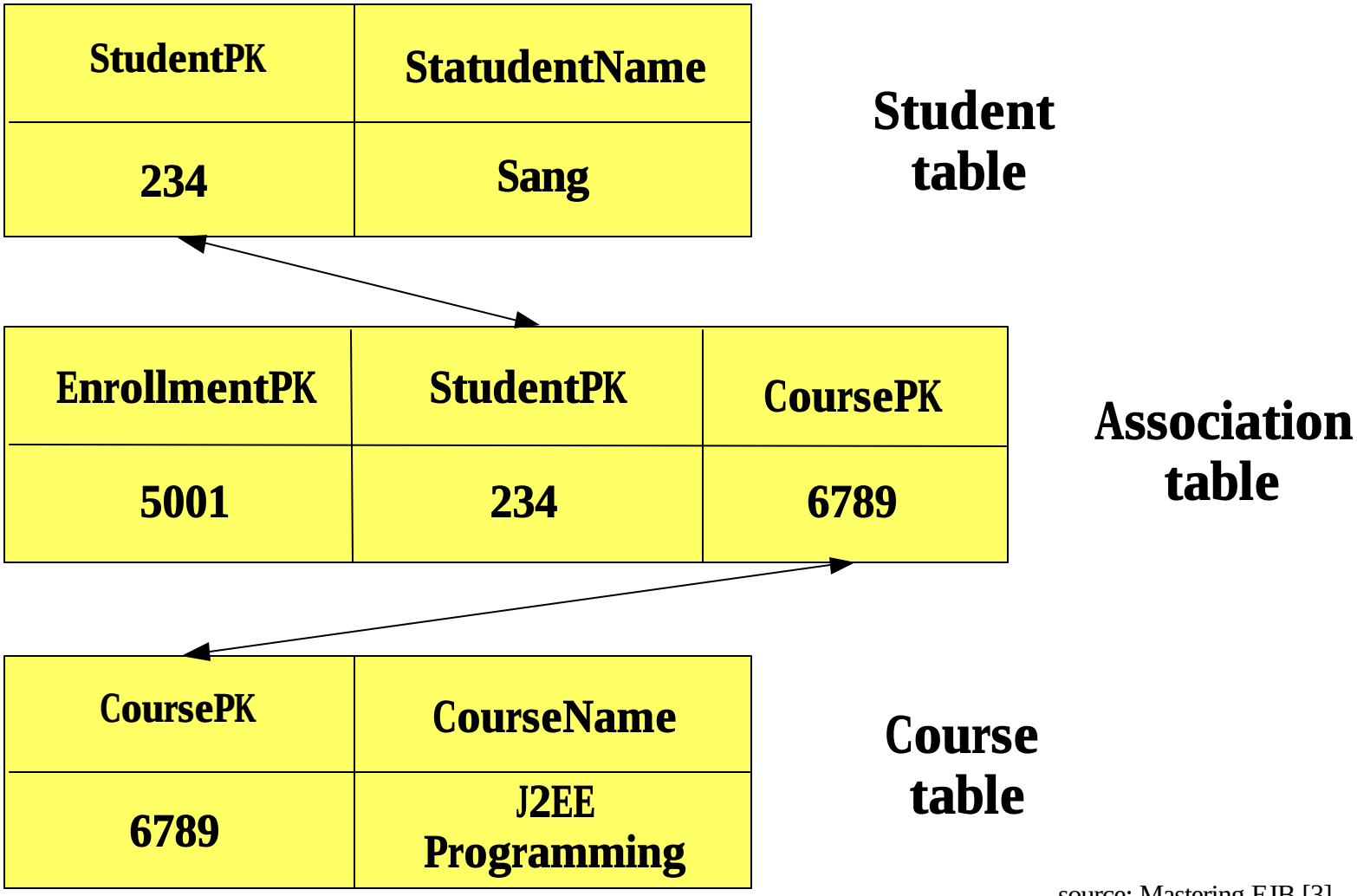


M:N Relationship

M:N Relationship

- Not as common as 1:N
- Typically set up by an association table in the database
  - Association table contains foreign keys to two other tables
- Examples
  - Students:Courses
  - People:Websites
  - Consumers:Creditcard

N:N Cardinality Database Schema



source: Mastering EJB [3]

M:N Relationship Handing in BMP: 2 choices

- **Choice 1: Make separate entity bean for association table**
  - **represent enrollment table as enrollment bean**
  - **enrollment bean handles relation handling**
  - **could be a good choice if more information can be maintained for the new table, i.e. enrollment date**
- **Choice 2: Do not make separate entity bean for association table**
  - **relationship handling is done by original beans**

M:N Relationship Handing in BMP (based on Choice 1) - page 1

```
public class StudentBean implements EntityBean {
   private String studentPK;
   private String studentName;
   ...
   public void ejbLoad() { // SQL SELECT Student }
   public void ejbStore() { // SQL UPDATE Student }
}

public class CourseBean implements EntityBean {
   private String coursePK;
   private String courseName;
   ...
   public void ejbLoad() { // SQL SELECT Course }
   public void ejbStore() { // SQL UPDATE Course }
}
```

source: Mastering EJB [3]

M:N Relationship Handing in BMP (based on Choice 1) - page 2

```
public class EnrollmentBean implements EntityBean {
  private String enrollmentPK;
  private Student student;       // EJB local object stub
  private Course course;         // EJB local object stub

  public Course getCourse() { return course; }
  public void setCourse(Course c) { this.course = c; }

  public Student getStudent() { return student;}
  public void setStudent(Student s) { this.student = s;}
  ...
  public void ejbLoad() {
     // 1: SQL SELECT Enrollment.  This loads both the enrollment plus the
     //      foreign keys to Student and Course
     //
     // 2. JNDI lookup of StudentHome, CourseHome
     //
     // 3. Call findByPrimaryKey() on both the Student and Course homes,
     //     passing the foreign keys
  }
```

source: Mastering EJB[3]

M:N Relationship Handing in BMP (based on Choice 1) - page 3

```
    ...
    public void ejbStore() {
        // 1:  Call getPrimaryKey() on Student, Course.  This gives
        //       us our foreign keys.
        //
        // 2:  SQL UPDATE Enrollment
    }

}
```

source: Mastering EJB[3]

M:N Relationship Handing in CMP

```
public abstract class StudentBean implements EntityBean {
    // Note that there are no fields
    ...
    public void ejbLoad() {}     // Empty
    public void ejbStore() {}    // Empty
}
public abstract class CourseBean implements EntityBean {
    // Note that there are no fields
    ...
    public void ejbLoad() {}     // Empty
    public void ejbStore() {}    // Empty
}
public abstract class EnrollmentBean implements EntityBean {
   // Note that there are no fields
   ...
   public abstract Course getCourse();
   public abstract void setCourse(Course c);
   public abstract Student getStudent();
   public abstract void setStudent(Student s);
   ...
   public void ejbLoad() {}     // Empty
   public void ejbStore() {}    // Empty
}
```

source: Mastering EJB[3]

M:N Relationship in CMP Deployment Descriptor - page 1

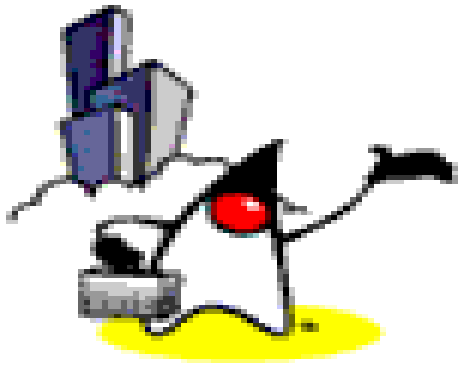
```
<ejb-jar>
  <enterprise-beans>  ...</enterprise-beans>
  <relationships>
    <ejb-relation>
      <ejb-relation-name>Enrollment-Student</ejb-relation-name>

      <ejb-relationship-role>
        <ejb-relationship-role-name> Enrollments-AreRegisteredBy-Student
        </ejb-relationship-role-name>
        <multiplicity>Many</multiplicity>
        <relationship-role-source>
          <ejb-name>Enrollment</ejb-name>
        </relationship-role-source>
        <cmr-field><cmr-field-name>student</cmr-field-name></cmr-field>
      </ejb-relationship-role>

      <ejb-relationship-role>
        <ejb-relationship-role-name> Students-Has-Enrollments
        </ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <ejb-name>Student</ejb-name>
        </relationship-role-source>
      </ejb-relationship-role>

    </ejb-relation>
```

source: Mastering EJB[3]

M:N Relationship in CMP Deployment Descriptor - page 2

```
<ejb-relation>
   <ejb-relation-name>Enrollment-Course</ejb-relation-name>

   <ejb-relationship-role>
      <ejb-relationship-role-name> Enrollments-AreRegistrationsFor-Course
      </ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <relationship-role-source>
         <ejb-name>Enrollment</ejb-name>
      </relationship-role-source>
      <cmr-field><cmr-field-name>course</cmr-field-name></cmr-field>
    </ejb-relationship-role>

   <ejb-relationship-role>
      <ejb-relationship-role-name> Course-Has-Enrollments
      </ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source>
         <ejb-name>Course</ejb-name>
      </relationship-role-source>
    </ejb-relationship-role>

  </ejb-relation>
 </relationships>
</ejb-jar>
```

source: Mastering EJB[3]



Directionality Relationship

Directionality Relationship

- Specifies the direction in which you can navigate a relationship
- Applies to all cardinalities
  - Directionality and cardinality are orthogonal concepts
- Two types of directionality
  - Bidirectional
    - You can get to entity B from entity A, and you can also get to entity A from entity B
  - Unidirectional
    - You can get to entity B from entity A, but you cannot get to entity A from entity B

Bidirectional Relationship Handing in BMP

```
public class OrderBean implements EntityBean {
    private String orderPK;
    private String orderName;

    // EJB local object stub, must be stored/loaded
    private Shipment shipment;

    public Shipment getShipment() { return shipment; }
    public void setShipment(Shipment s) { this.shipment = s;}
    ...
}

public class ShipmentBean implements EntityBean {
    private String shipmentPK;
    private String shipmentName;

    // EJB local object stub, must be stored/loaded
    private Order order;

    public Order getOrder() { return order; }
    public void setOrder(Order o) { this.order = o; }
    ...
}
```

source: Mastering EJB[3]

UniDirectional Relationship Handing in BMP

```
public class OrderBean implements EntityBean {
    private String orderPK;
    private String orderName;

    // EJB local object stub, must be stored/loaded
    private Shipment shipment;

    public Shipment getShipment() { return shipment; }
    public void setShipment(Shipment s) { this.shipment = s;}
    ...
}

public class ShipmentBean implements EntityBean {
    private String shipmentPK;
    private String shipmentName;

    // No Order stub, no Other get/set method
    ...
}
```

source: Mastering EJB[3]

Bidirectional Relationship Handing in CMP

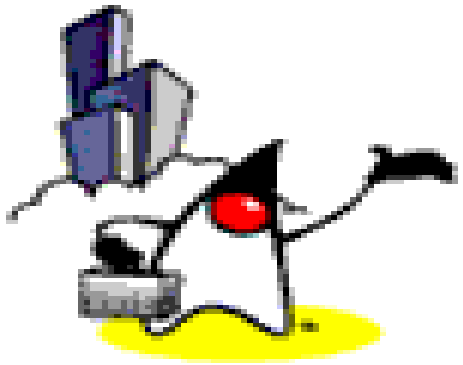
```
public abstract  class OrderBean implements EntityBean {
    // Note that there are no fields

    public abstract Shipment getShipment();
    public abstract void setShipment(Shipment s);
    ...
    public void ejbLoad() {};     // empty
    public void ejbStore() {};    // empty
}

public abstract class ShipmentBean implements EntityBean {
    // Note that there are no fields

    public abstract Order getOrder();
    public sbstract void setOrder(Order o);
    ...
    public void ejbLoad() {};     // empty
    public void ejbStore() {};    // empty
}
```

source: Mastering EJB[3]

Bidirectional Relationship Handling in CMP Deployment Descriptor

```
<ejb-jar>
  <enterprise-beans>  ...</enterprise-beans>
  <relationships>
    <ejb-relation>
      <ejb-relation-name>Order-Shipment</ejb-relation-name>
      <ejb-relationship-role>
        <ejb-relationship-role-name> order-spawns-shipment
        </ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <ejb-name>Order</ejb-name>
        </relationship-role-source>
        <cmr-field><cmr-field-name>shipment</cmr-field-name></cmr-field>
      </ejb-relationship-role>
      <ejb-relationship-role>
        <ejb-relationship-role-name> shipment-fulfills-order
        </ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <ejb-name>Shipment</ejb-name>
        </relationship-role-source>
        <cmr-field><cmr-field-name>order</cmr-field-name></cmr-field>
      </ejb-relationship-role>
    </ejb-relation>
  </relationships>
</ejb-jar>
```

source: Mastering EJB[3]



Aggressive Loading versus Lazy Loading on Relationship Handling

Aggressive vs. Lazy Loading

- **Aggressive loading**
  - **Whenever an entity bean is loaded, all the other entity beans that it has a relationship with are also loaded**
  - **Downside: Could load a very large entity bean graph which would not be used**
- **Lazy loading**
  - **Load only related beans when you need to access those beans**
  - **In BMP, you have to write your own “lazy-loading” code**
  - **In CMP 2.0, this is automatically done by Container**

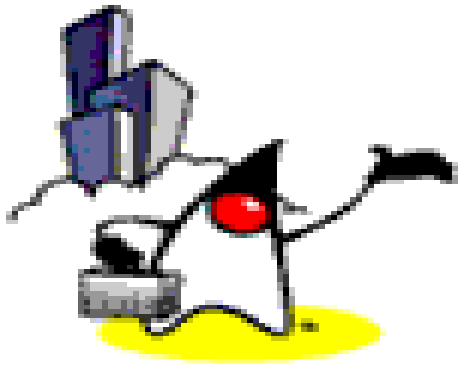
Lazy-loading Relationship Handing in BMP

```
public class OrderBean implements EntityBean {
    private String orderPK;
    private String orderName;

    private String shipmentPK;        // Foreign key to shipment
    private Shipment shipment;        // EJB local object stub

    // Do not load Shipment automatically in ejbLoad()
    public void ejbLoad() {
        // 1: SQL SELECT Order, loading the shipment foreign key
        // 2: Set ShipmentFK field to the loaded key
    }

   // Load Shipment only when a business method is called
   public Shipment getShipment() {
        // 1: JNDI lookup of ShipmentHome
        // 2: Call ShipmentHome.findByPrimaryKey(shipmentFK)
        return shipment;
    }

    ...
}
```

source: Mastering EJB[3]

Lazy Loading Relationship Handling in CMP

- In CMP 2.0, this is automatically done by Container



Aggregation vs. Composition and Cascading Delete

Aggregation vs. Composition

- Any relationship is one of two types
  - Aggregation
  - Composition
- Aggregation relationship
  - One entity bean A “uses” another entity bean B
  - Deleting A does not result in deletion of B
  - Example: Students and courses
- Composition relationship
  - One entity bean A “ is assembled of” entity bean B
  - Deleting A result in deletion of B
  - Example: Order and lineitems

Cascading Delete

- Applies only to composition relationship
  - If parent entity bean (Order) is deleted, all the child beans (Lineitems or shipment) ought to be deleted as well
- In BMP, bean developer has to implement “cascading delete” in ejbRemove() method
  - Order bean's ejbRemove() method not only perform “SQL DELETE” on the Order but also call Shipment bean's ejbRemove() method

Cascading Delete in BMP

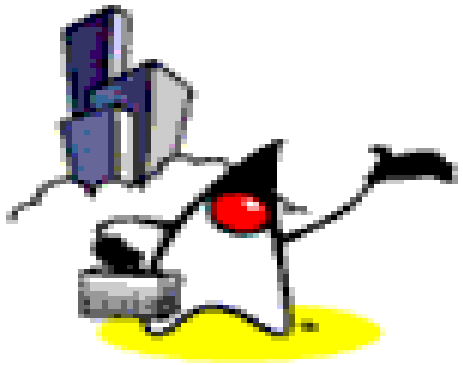
```
public class OrderBean implements EntityBean {
    private String orderPK;
    private String orderName;
    private Shipment shipment;        // EJB local object stub

    public Shipment getShipment() { return shipment;}
    public void setShipment(Shipment s) { this.shipment = s; }
    ...

    public void ejbRemove() {
        // 1: SQL DELETE Order
        // 2: shipment.remove();
    }
}
```

source: Mastering EJB[3]

Cascading Delete in CMP Deployment Descriptor

```
<ejb-jar>
  <enterprise-beans>  ...</enterprise-beans>
  <relationships>
    <ejb-relation>
      <ejb-relation-name>Order-Shipment</ejb-relation-name>
      <ejb-relationship-role>
        <ejb-relationship-role-name> order-spawns-shipment
        </ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
          <ejb-name>Order</ejb-name>
        </relationship-role-source>
        <cmr-field><cmr-field-name>shipment</cmr-field-name></cmr-field>
      </ejb-relationship-role>
      <ejb-relationship-role>
        <ejb-relationship-role-name> shipment-fulfills-order
        </ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <cascade-delete/>
        <relationship-role-source>
          <ejb-name>Shipment</ejb-name>
        </relationship-role-source>
        <cmr-field><cmr-field-name>order</cmr-field-name></cmr-field>
      </ejb-relationship-role>
    </ejb-relation>
  </relationships>
</ejb-jar>
```

source: Mastering EJB[3]



Referential Integrity

Referential Integrity

- Assurance that a reference from on entity to another entity is valid
  - Company, department, position entity bean have a relationship with employ bean: if employee is removed, all references to it must be also removed, or your system must now allow the removal
- Have to be dealt with at
  - database (keeping foreign keys correct)
  - app server (keeping stubs correct)

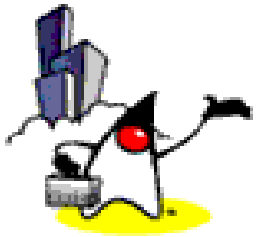
Relationship Database Design

- When designing a database application, you may choose to enforce the dependency between the parent and child tables
- Two ways for enforcing the dependency
  - via defining a referential constraint in the database
  - by performing checks in the application code
    - In BMP, developer has to deal with it
    - In CMP, Container handles referential integrity

Defining Referential Constraint in the database

```
CREATE TABLE storagebin
 (storagebinid VARCHAR(3)
  CONSTRAINT pk_storagebin PRIMARY KEY,
  widgetid VARCHAR(3),
  quantity INTEGER,
  CONSTRAINT fk_widgetid
  FOREIGN KEY (widgetid)
   REFERENCES widget(widgetid));
```

Dependent Value Classes (Dependent Objects)



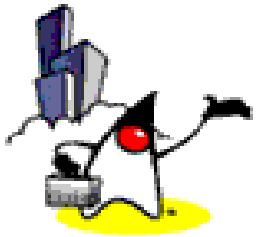
What is Dependent Value Class?

- Java class that is dependent on an entity bean
  - Order entity bean might have line items
  - Line items are dependent on Order entity bean
  - Line items can be just Java classes (as opposed to being entity bean themselves)
    - No overhead of entity beans

Dependent Value Class Handling in CMP 2.0

- Dependent value classes are defined just like CMP fields
  - Without the overhead of being entity beans
  - Use dependent value classes instead entity beans if you don't have to expose them as entity beans
- Constraints of dependent value classes
  - may not be container-managed relationship (CMR) fields - relationships only exist between entity beans
  - have to be serializable

EJB-QL
(EJB Query Language)



What is EJB QL?

- **Portable, abstract definition** of finder and selector methods for entity beans and dependent objects
  - **Declaratively expressed** in deployment descriptor
  - SQL like language
  - Compiled to native language (i.e. SQL) of data store during runtime by container
    - Container can optimize the implementation
- Enables querying through navigation
  - Difference from SQL

Why EJB-QL?

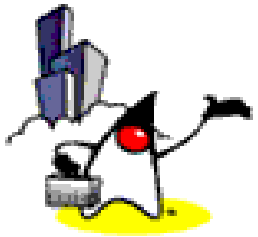
- Without EJB-QL, you have to code the finder or select logic yourself in your bean
  - a lot of coding
  - not portable
  - not optimized
- With EJB-QL, you specify the finder and select logic in declarative fashion, and container handles the rest
  - no coding required
  - portable over different database technologies
  - can be optimized

Example: EJB-QL

- Find all accounts whose balance is over X dollars
  - X is a parameter
- EJB-QL script

```
SELECT OBJECT(a)
FROM Account AS a
WHERE a.balance >?1
```

Relationship & EJB-QL



Relationship Navigation via EJB-QL

- EJB-QL allows you to traverse relationships between entity beans using a dot notation
  - difference between SQL and EJB-QL
- Example: Return all customers that have placed orders
  - Navigate from Order entity bean to Customer entity bean
  - EJB-QL script
    - SELECT o.customer
    - FROM Order o

Relationship Navigation via EJB-QL

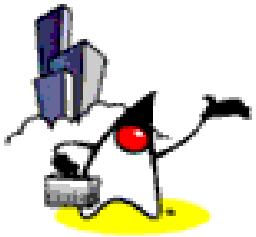
- Bean developers do not need to know about tables and columns
  - They only need to understand the relationships between entity beans
  - Navigational logic is handled by container
- Constraints
  - The entity beans have to be in the same Ejb-jar file
  - Navigation possible only for relationships that have been defined in the deployment descriptor

Navigation of Multiple Relationships

- Can traverse multiple relationships
- Relationship can be CMR (Container Managed Relation) field or CMP (Container Managed Persistent) field
- Example: Find phone numbers of all customers that have placed orders
- EJB-QL script

```
SELECT o.customer.address.homePhoneNumber
FROM Order o
(order and customer relationship: CMR)
(customer and address relationship: CMP)
(address and homePhoneNumber relationship: CMP)
```

Persistence Strategies



Persistence Strategies: Issues

- Performance
- Portability
- Database independence
- Schema independence
- Relationship modeling
- Ease of development
- Caching
- Persistence model (O/R mapping)
- Migration

Persistence Strategies: Choices

- CMP 2.0
- CMP 1.0
- BMP
- Session beans with JDBC/JDO
- Servlet/JSP with JDBC/JDO

Advantages of CMP 2.0 for Component Providers

- Rich modeling capability with relationships
  - Referential integrity
  - Cardinality
  - Cascading delete
  - Container manages the relationships not you!
- Freedom from maintaining interactions with the data store
- EJB™ Query Language (EJB QL)
- Truly portable code

Advantages of CMP 2.0 for Container Vendors

- **Optimization is possible because CMP fields are only accessible through their setters and getters**
  - **Lazy loading**
  - **Dirty checking**
  - **Optimistic locking**
- **Optimization is possible in query operation because Query is defined in deployment descriptor via EJBQL**

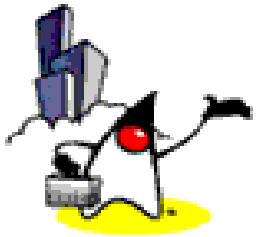
Advantages of BMP

- Dealing with Legacy
  - Database and/or other persistence store
  - Familiar mapping tool
  - Previously written complex BMP application
- Connecting to Connector driven data stores, EIS's (Enterprise Information Systems)
- Complicated operations beyond scope of the EJB 2.0 specification
  - Bulk updates
  - Multi object selects
  - Aggregates (like sorting)

Recommendations

- Use CMP 2.0 whenever possible!
  - It performs better than BMP
  - It improves portability, performance over CMP 1.0
  - It is easier to develop and deploy than BMP
  - It produces portable code over multiple databases
  - There is no reason not to use CMP 2.0 now!
- If you have to build BMP entity bean, subclass CMP 2.0 bean
  - Easy migration to CMP later on

Commit Options



Persistent Management

- Handles the synchronization of the state of bean instances with persistent data
- Key to efficient use of entity beans
- EJB 1.1
  - bean-managed (BMP)
  - primitive container-managed persistence (CMP 1.1)
- EJB 2.0 (CMP 2.0)
  - bean-managed (BMP)
  - much improved container-managed persistence (CMP 2.0)

Persistent Management

- Container calls ejbLoad(), ejbStore(), and other internal methods in whatever sequence at appropriate time to keep the state of bean instance with the persistent data
  - Developers should not have to know how the container does this
  - But understanding the principles will help developers to write optimized EJB application

DB Operations are Expensive

- Performing database operations (reading and writing) are expensive
- Minimize database operations as much as possible for both BMP and CMP using
  - caching
  - dirty write
  - proper locking

Transaction

- Client1 is modifying a database row (RowXXX) through EJB ejb1 while client2 is reading and writing that row directly
- Client1 starts transaction before calling ejbLoad(), thus have transactional isolation (lock) on RowXXX
  - client2 (or other EJB's) cannot modify RowXXX until client1 releases the lock

client1-->

ejb1.methodA() <--> RowXXX

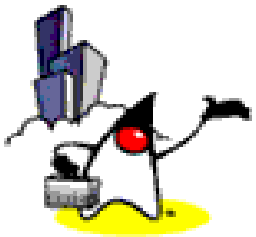
ejb1.methodB() <--> RowXXX

client2 <--> RowXXX

Commit Option B & C

- Container calls ejbLoad(), ejbStore(), and other internal methods in whatever sequence at appropriate time to keep the state of bean instance with the persistent data
  - Developers should not have to know how the container does this
  - But understanding the principles will help developers to write optimized EJB application

Home Methods



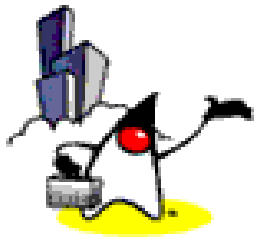
Home Methods

- Introduced in EJB 2.0
- Methods can be specified in Home interface
  - Implementation of these methods in Bean class should start "ejbHome”
- Conceptually similar with Static method of Java class
- Available only for Entity beans
  - Because session bean instances do not have to be synchronized with databases

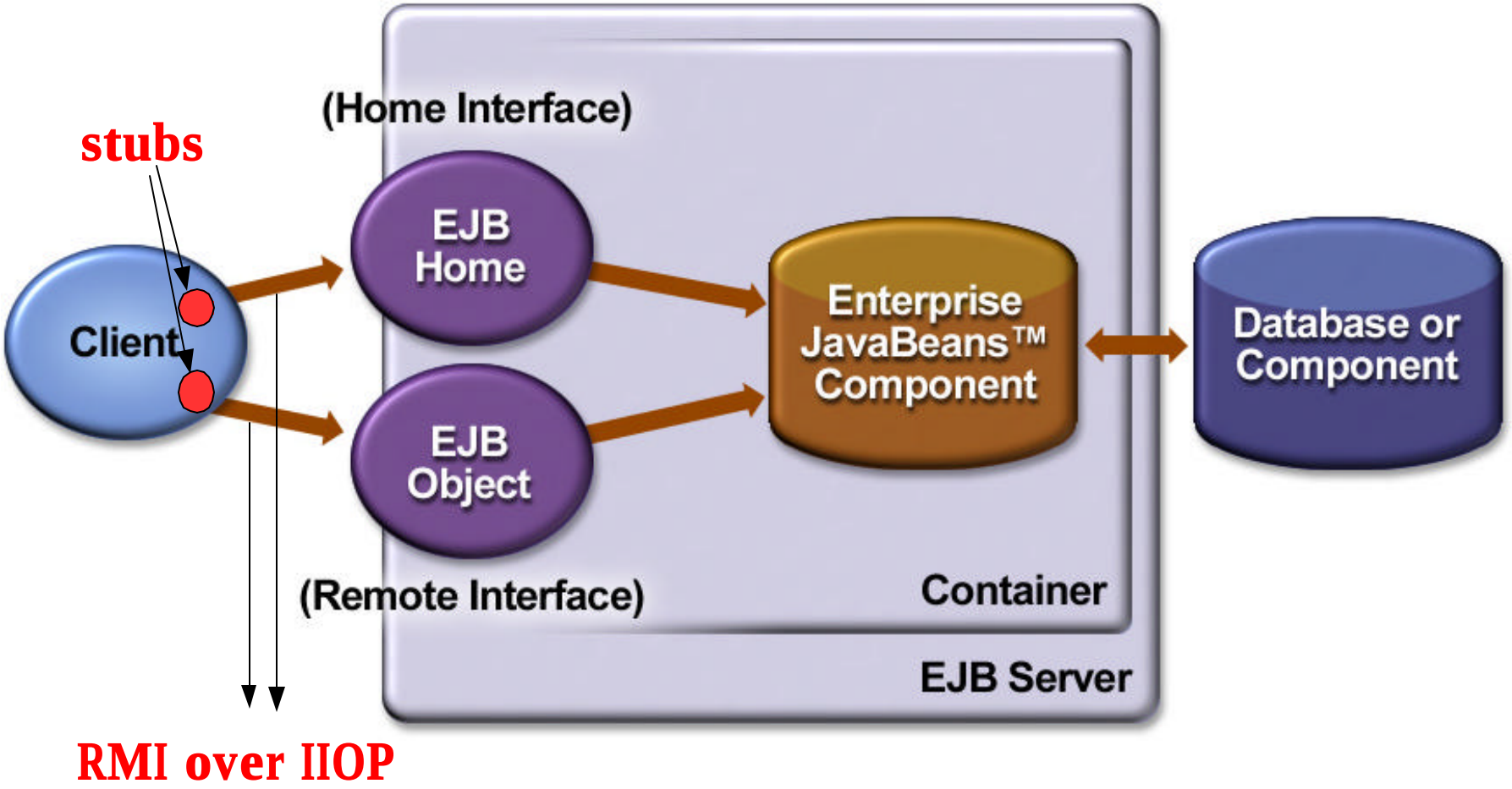
When to Use Home Methods?

- When you want to return a count of number of instances in the database
- In EJB 1.1, two ways to implement this
  - Use session bean instead of Entity bean
  - Use dummy Entity bean that does not map to database
    - findByPrimaryKey(“”)
- In EJB 2.0, use Home method
  - Any instance in the pool can be used
  - No database synchronization is needed

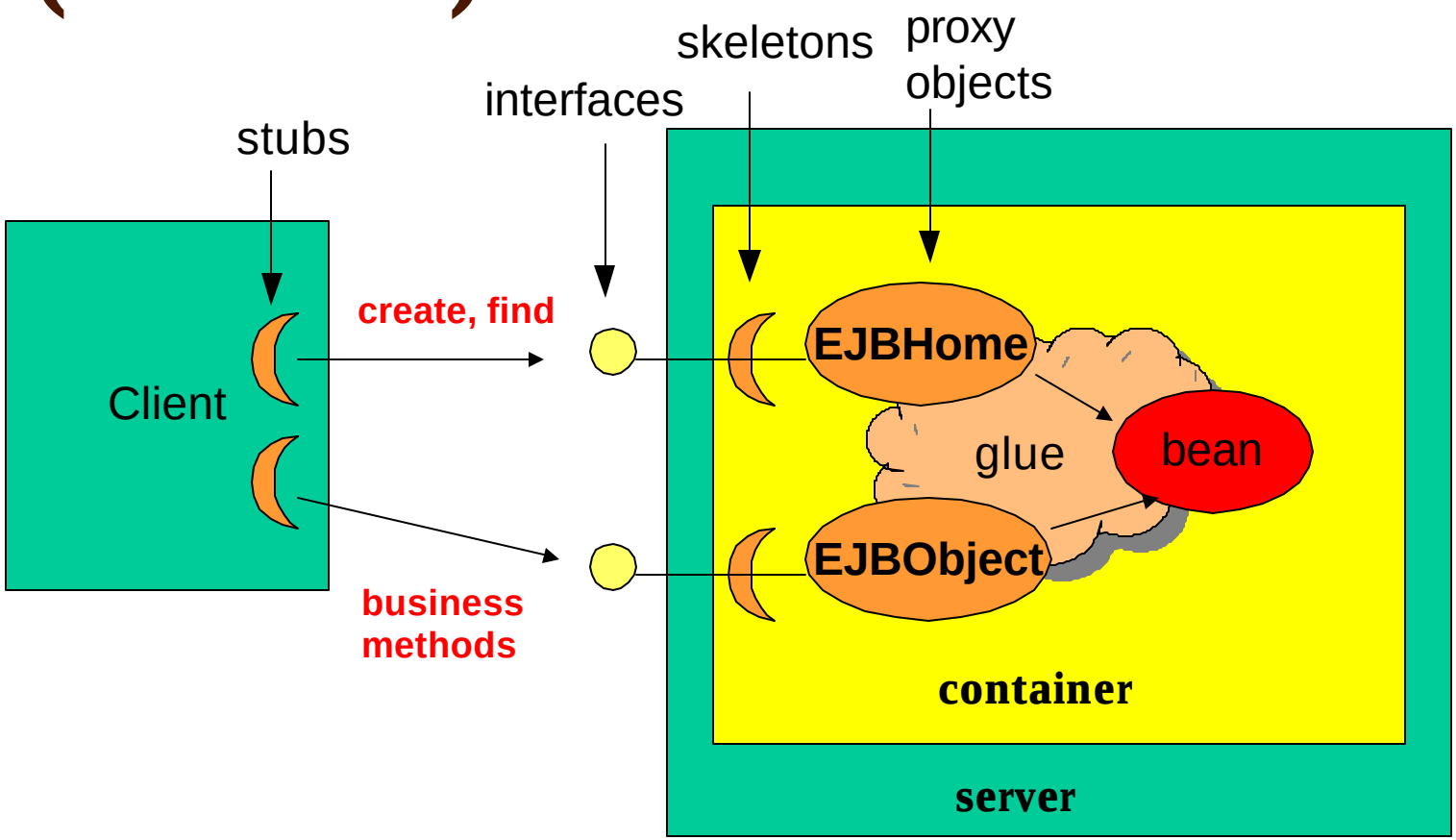
Passing References to EJBs within EJB code



Where are Stubs? (Review)



EJB™ Home & EJB™ Object (Review)

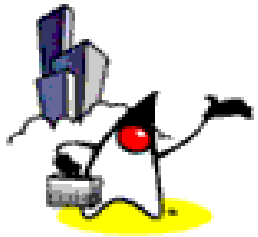


- In ordinary Java programming, if object X wants to create an instance of object Y passing a reference to itself to Y, X will do
  - Y myY = new Y(this)
- In EJB, the above will not work work (see next slide for why this is the case)

- **In EJB, passing “this” will not work due to**
  - **“ this” is a pointer that cannot be passed around in distributed environment**
    - **Only references to stub can be passed around**
    - **There are stubs for Home object and EJB object but there is no stub for Bean instance**
  - **EJBs cannot call one another on their bean instances directly, this will bypass EJB container**

- Y myY = Y.create(ctx.getEJBObject(), ...);
  - ctx is EntityContext object
  - Giving object Y a reference to X's EJB object effectively provides a reference to the EJB bean instance itself (via Container)
  - EJB must be marked as “reentrant” in deployment descriptor

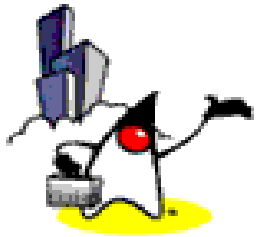
Exception Handling for Entity Beans



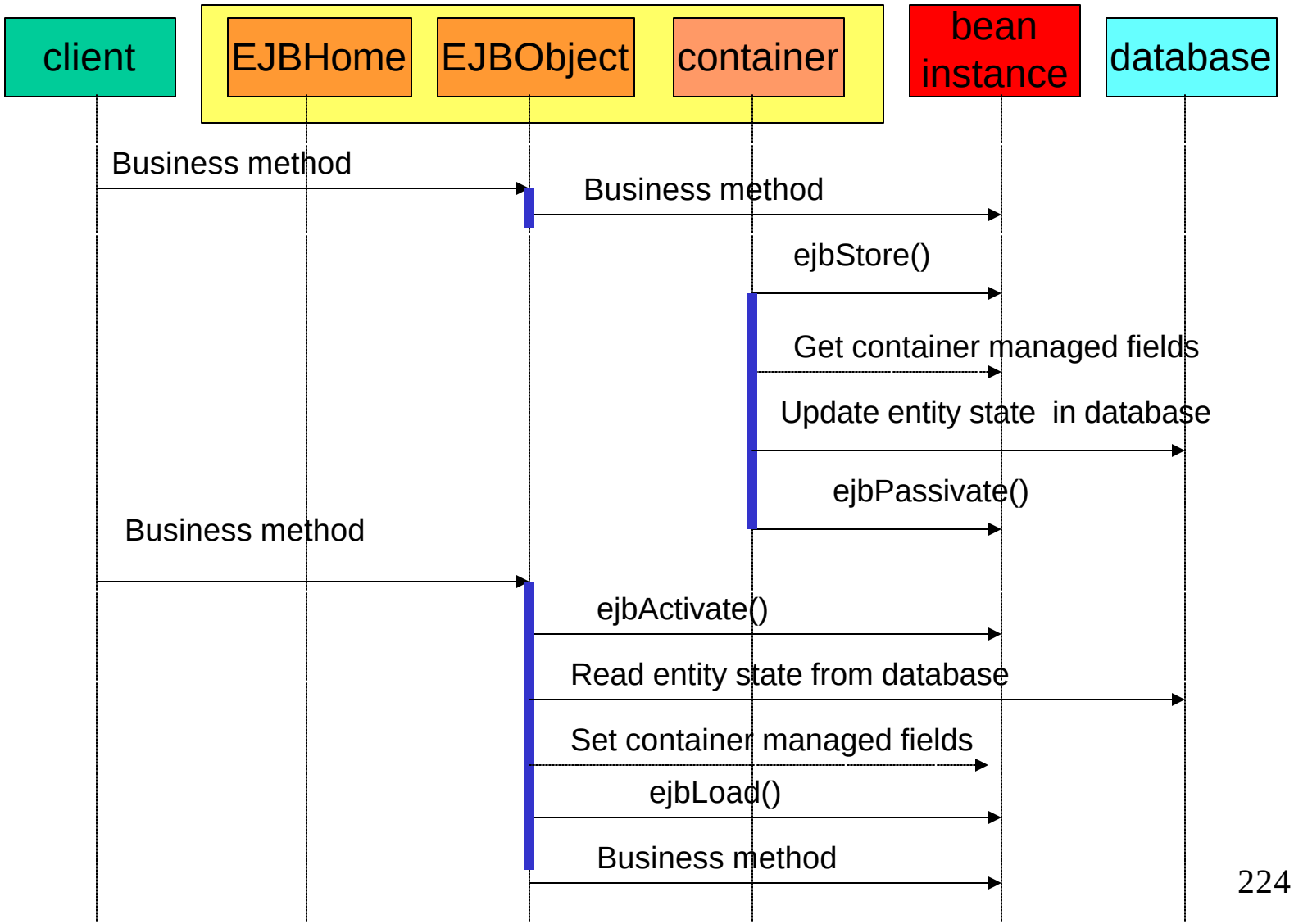
Exceptions for Entity Beans

- CreateException
- DuplicateKeyException
  - subclass of CreateException
  - when primary key is not unique
- FinderException
- ObjectNotFoundException
  - subclass of FinderException
  - for ejbFindByPrimaryKey()
- RemoveException

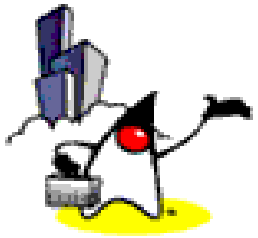
Passivation & Activation in Entity Bean



Passivation of CMP-based Entity Bean



Resources



Resources Used

- Applied Enterprise JavaBeans Technology written by Kevin Boone (Sun Microsystems, Inc.), published by Prentice Hall, 2003 [1]
- J2EE Tutorial in java.sun.com, 2002 [2]
- Mastering Enterprise JavaBeans, 2nd edition written by Ed Roman, published by Wiley, 2002 [3]
- EJB Codecamp by Carol McDonald (Sun Microsystems, Inc.), 2001 [4]

Live your life with Passion!