ENTERPRISE JAVA BEANS

WHAT IS EJB?

- Enterprise JavaBeans (EJB) is a managed, server software for modular construction of enterprise software, and one of several Java APIs. EJB is a server side software component that encapsulates the business logic of an application.

- An EJB web container provides a runtime environment for web related software components, including
- computer security,
- Java servlet lifecycle management,
- transaction processing, and other web services.

EJB GOALS

- Standard component architecture for building distributed business applications in java
- Interoperability
- Compatibility with other java API’s and CORBA protocols
- WORA philosophy
- Define the contracts for interoperability of tools from multiple vendors

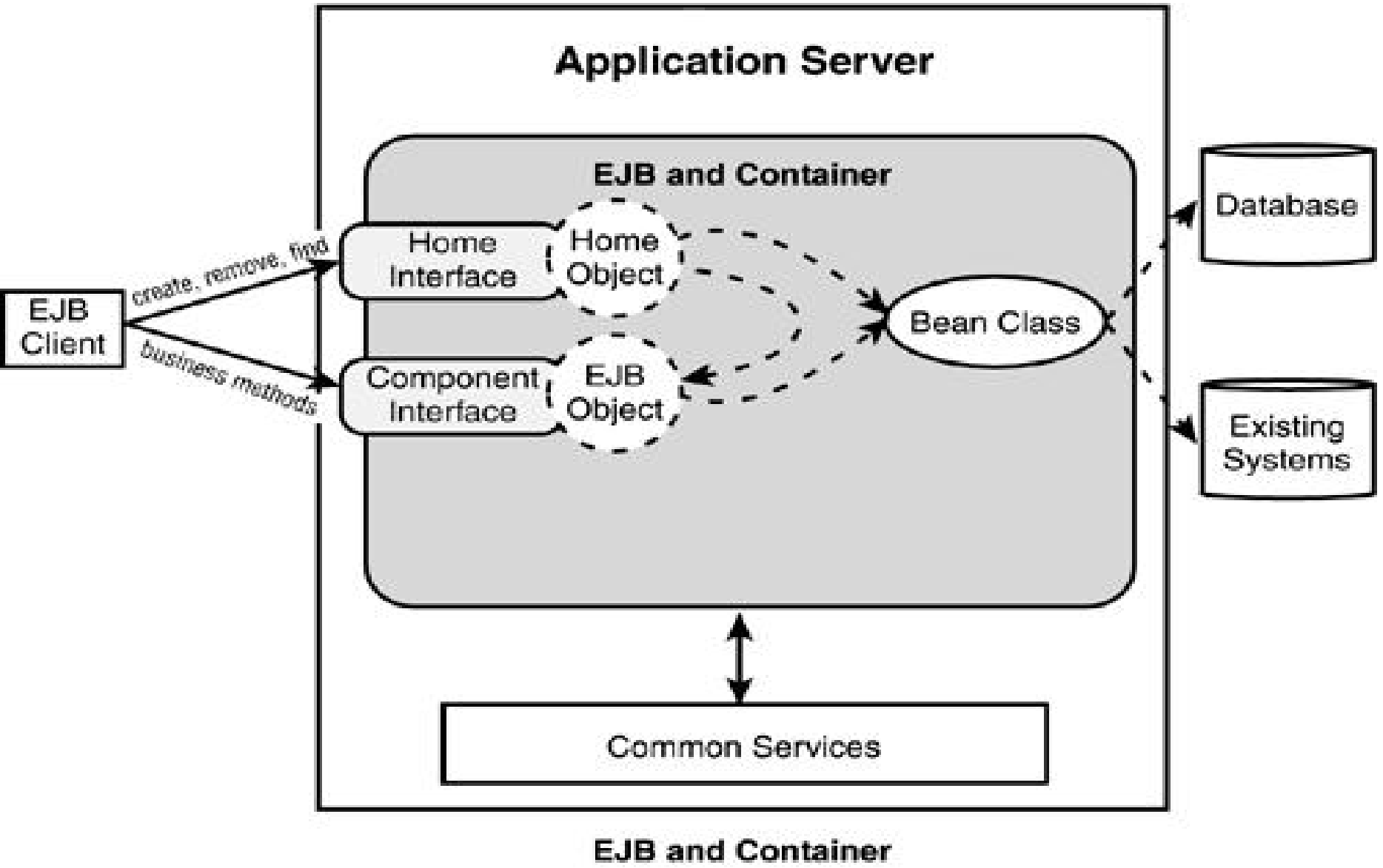
GENERAL RESPONSIBILITIES

- Transaction processing
- Integration with the persistence services offered by the Java Persistence API (JPA)
- Concurrency control
- Event-driven programming using Java Message Service and Java EE Connector Architecture
- Asynchronous method invocation
- Job scheduling
- Naming and directory services (JNDI)
- Interprocess Communication using RMI-IIOP and Web services
- Security (JCE and JAAS)
- Deployment of software components in an application server

What are the benefits???

- Simplified development of large scale enterprise level application.
- Application Server/ EJB container provides most of the system level services like transaction handling, logging, load balancing, persistence mechanism, exception handling and so on. Developer has to focus only on business logic of the application.
- EJB container manages life cycle of ejb instances thus developer needs not to worry about when to create/delete ejb objects.
- The knowledge about EJB is portable among many different products because EJB products are based on a common standard

ARCHITECTURE



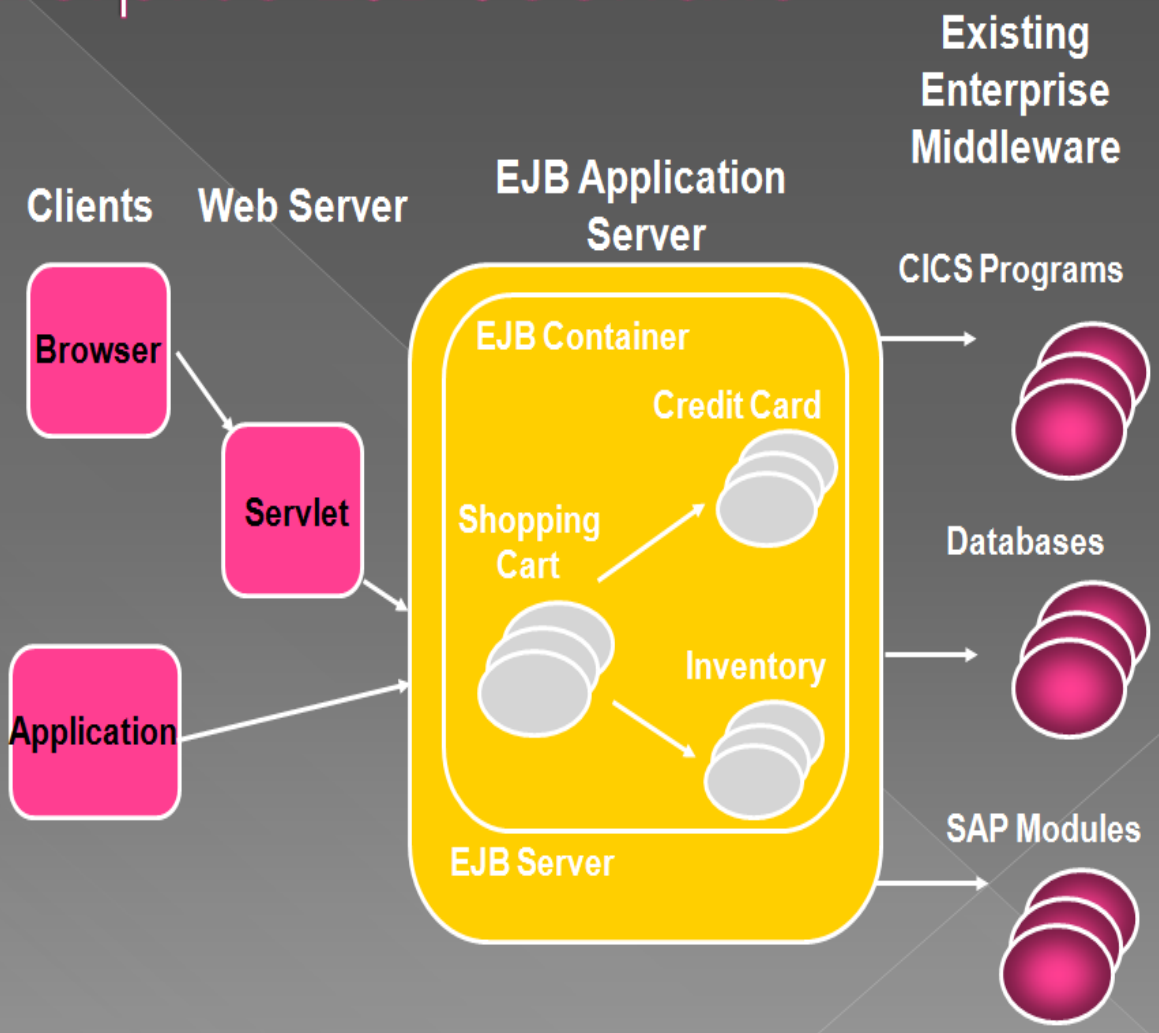
EJB CONTAINER

- Instances running within ejb container
- Runtime environment
- Controls an ejb component instance
- Provides necessary management services
- Provides services to Enterprise JavaBeans
  - Naming
  - Life cycle management
  - Persistence (state management)
  - Transaction Management
  - Security

EJB COMPONENT

- Enterprise bean: a distributed component that lives in an EJB container and accessed by remote clients over network via its network interface or by other enterprise beans on the same server over the local interface.
- Remotely executable component deployed on its server.

Enterprise EJB Scenario

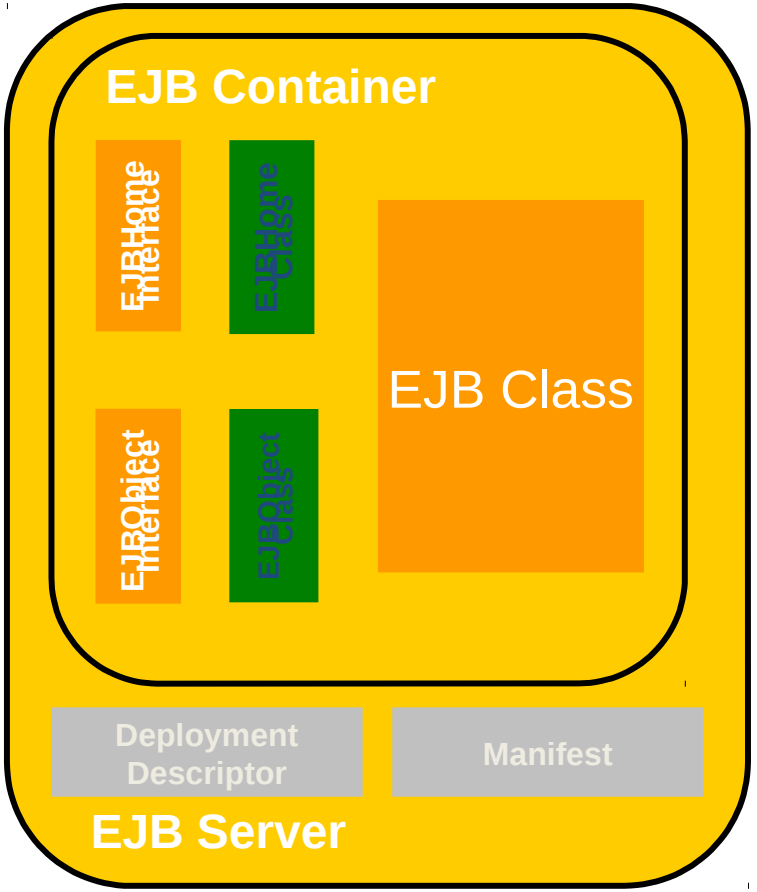


An inside look at the various components of EJB

- Bean Class is written by the developer

- EJBHome and EJBObject
- interfaces and classes
- control access to the
- Bean class

Deployment Descriptor and MANIFEST describe security and transactional characteristics of the Bean



EJB Types

- Entity Beans
- Session Beans
- Message-Driven Beans

EJB Types

- Entity Beans
- Session Beans

} RMI-based server side components
- Accessed using distributed object
- Protocols (RMI IIOP)

- Message-Driven Beans

} New in EJB 2.0
- Asynchronous server side
- component that responds to
- JMS asynchronous messages
- (Think provider like JAXM)

Session Beans

- Overview
- Session Bean Life Cycle
- Writing a Session Bean

Overview of Session beans

- Session
  - A client's conversation with a server
  - Client has server carry out business operations
- Session Bean
  - A bean that represents a client's session with server
  - Encapsulates a business process
  - Usually transient
  - Client can be end user, another bean or anything else
  - Examples: stock broker, bank teller, shopping cart

Types of session beans

- Stateless session beans
  - Stateless session beans hold no session data
  - Hence usually have no instance variables
- For example, a stock broker:
  - Acts as gateway to your portfolio
  - Does not hold your money; puts it into an account
  - A broker bean would be stateless
  - Portfolio and account beans would have state

Types of session beans

- Stateful session beans
  - Stateful session beans hold session data:
  - They create & maintain data as session proceeds
  - Data is held in instance variables
- For example, a customer's shopping cart:
  - Cart grows during a session
  - When customer "leaves the store", session ends

Session bean life cycle

- Every session bean has a home object
  - A "factory" for that bean
  - EJB client uses bean's home to create bean instances
  - All clients share one home object for given bean type.
- Creating a session:
  - Establishes client's connection to a session
  - Creates a skeleton
  - To create a session, a client:
    - Gets a reference to home object using JNDI
    - Uses home object's methods to connect to a session bean

Session bean life cycle

- Removing a session:
  - Servers client's connection to the bean
  - Destroys skeleton
- Client retains stub but can not use it
  - Client must create a new session again through home object
  - To do this, client uses a method of remote reference

Implementing Session Beans

- Define the component interfaces
  - You may choose to define all or only some of these depending on how you want your bean used
  - local interfaces do not require RMI overhead
- Define a bean class

Implementing Session Beans

- Define the component interfaces
  - The remote interface specifies how the outside world can access the bean's business methods
  - The remote home interface specifies how the outside world can access the bean's life-cycle methods (for creating, removing and finding)
  - The local interface specifies how the inside world (same EJB container) can access the bean's business methods
  - The local home interface specifies how the inside world can access the bean's life-cycle methods

Implementing Session Beans

- Implement the bean
  - Fill in the code for the business and life-cycle methods
  - It's not normal to directly implement the interfaces as we do in standard Java (though you must provide many of the methods). The calls to methods are not normal Java calls. They first go through the container.
  - Session beans implement javax.ejb.SessionBean
  - Entity beans implement javax.ejb.EntityBean
  - Both beans extend javax.ejb.EnterpriseBean

Message driven beans

- Overview
- Message driven Bean Life Cycle
- Writing a message driven bean

Message driven beans

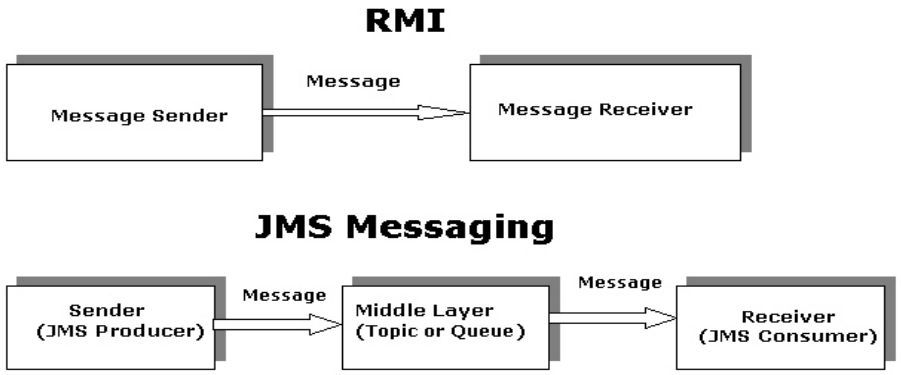
Overview

- Provide asynchronous messaging between two Java components.
- Uses Java Message Service (JMS) Application Programming Interface (API) to receive messages from the components.

Introducing JMS

- JMS API allows Java programs to send and receive messages.

Difference between JMS and RMI

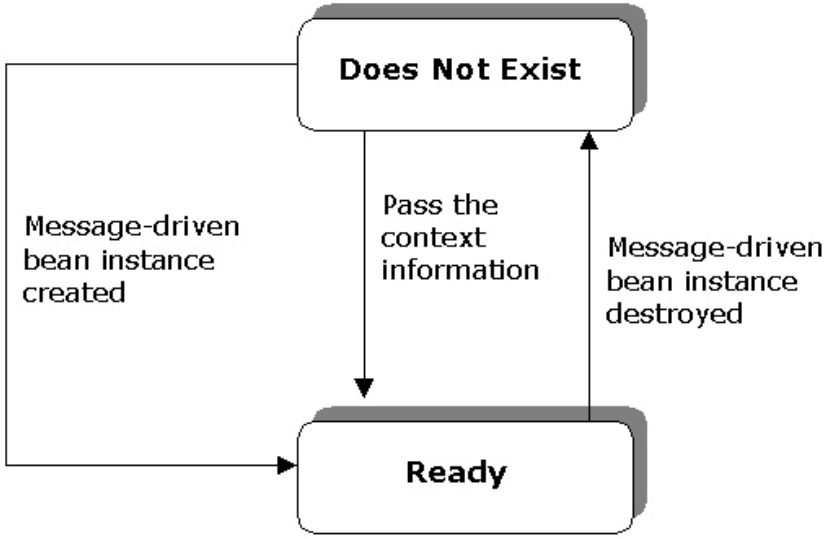


Features

- Features of Message-Driven Beans
  - They are stateless because they do not store the state of the client.
  - Instances are stored in a shared pool and EJB container can use any instance from this pool to receive and process the incoming message.
  - They cannot return values or throw exceptions to the client.
  - They can be declared as durable or non durable JMS consumers.

Life cycle of MDB

- The life cycle of an MDB is represented by the following image.



Life cycle of MDB

- Ready Stage
  - Message-driven bean instance remains in the pool to service the messages sent by the clients .
  - To add a new message-driven bean instance to the pool, EJB container performs the following steps:
  - Call the `setMessageDrivenContext()` method to pass the context object to a message-driven bean instance.
  - Call the `ejbCreate()` method of the instance to initialize the message-driven bean.

Life cycle of MDB

- Does Not Exist Stage
  - Message-driven bean is permanently removed from the message-driven bean pool.
  - The `onMessage( )` method is called whenever a message is received from the client.

Implementing a Message-Driven Bean

- Has no local, local home, remote, or remote home interfaces to define.
- The container will call the onMessage() method when an asynchronous message arrives. (Like JAXM message provider.)
- Extends the EnterpriseBean class and implements the javax.ejb.MessageDrivenBean and javax.jms.MessageListener interfaces

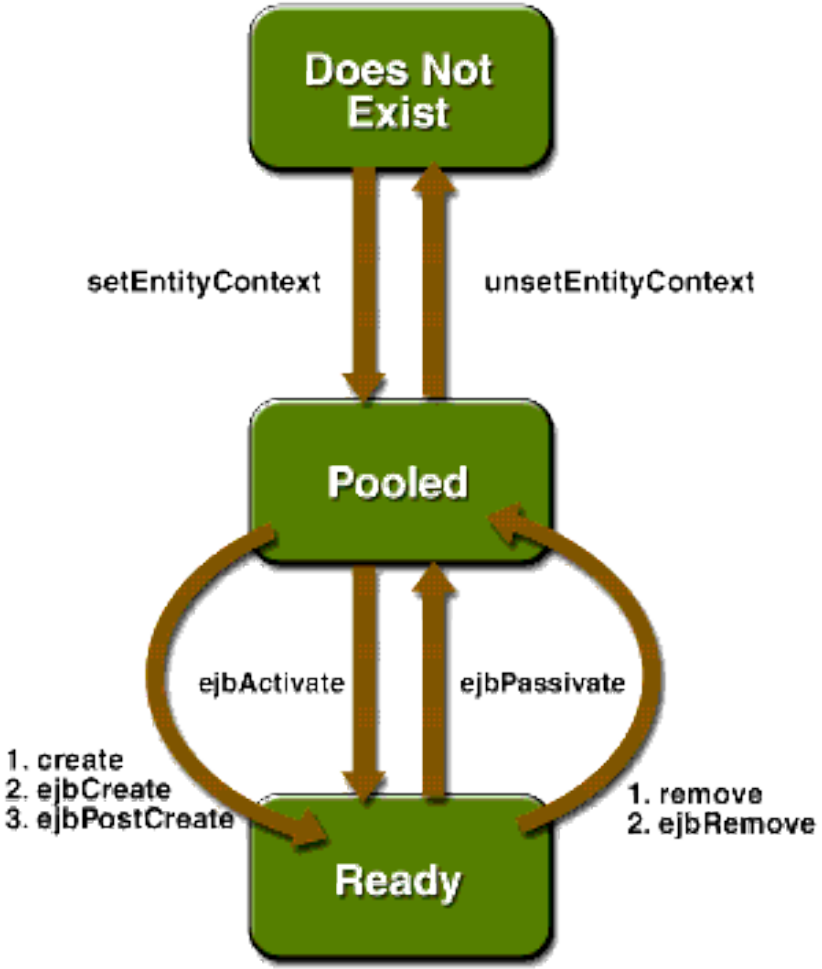
Implementing a Message-Driven Bean

- Two basic messaging-system models
  (1) point-to-point model allows messages to be sent to a message queue to be read by exactly one message consumer
  (2) publish/subscribe model allows components to publish messages on a topic to a server to be read by zero or more subscribers

PRIMARY KEY

- Each entity bean has a persistent identity associated with it. That is, the entity bean contains a unique identity that can be retrieved if you have the primary key. Given the primary key, a client can retrieve the entity bean. If the bean is not available, the container instantiates the bean and repopulates the persistent data for you.
- The type for the unique key is defined by the bean provider.

LIFE CYCLE OF ENTITY BEAN



- After instantiation, the entity bean moves to a pool of available instances.
- The EJB container assigns an identity to an instance when moving it to the ready stage.
- There are two paths from the pooled stage to the ready stage. On the first path, the client invokes the create method, causing the EJB container to call the ejbCreate and ejbPostCreate methods.
- On the second path, the EJB container invokes the ejbActivate method. While an entity bean is in the ready stage, an it's business methods can be invoked.
- There are also two paths from the ready stage to the pooled stage. First, a client can invoke the remove method, which causes the EJB container to call the ejbRemove method.
- Second, the EJB container can invoke the ejbPassivate method.

- In the pooled state, an instance is not associated with any particular EJB object identity.
- With bean-managed persistence, when the EJB container moves an instance from the pooled state to the ready state, it does not automatically set the primary key.
- Therefore, the ejbCreate and ejbActivate methods must assign a value to the primary key

- At the end of the life cycle, the EJB container removes the instance from the pool and invokes the unsetEntityContext method

Enterprise Java Bean - Working

Tasks of Bean Developer

- The bean developer must create the following interfaces and classes:
- The remote home interface for the bean. The home interface defines the methods a client uses to create, locate and destroy instances of a bean.
- The remote local interface for the bean. This defines the business methods implemented in the bean. The client accesses these methods through remote interface.

Tasks of Bean Developer

- The bean developer must create the following interfaces and classes:
- The remote home interface for the bean. The home interface defines the methods a client uses to create, locate and destroy instances of a bean.
- The remote local interface for the bean. This defines the business methods implemented in the bean. The client accesses these methods through remote interface.

Working

- Once the bean is deployed in the EJB container, the client calls the create() method defined in the home interface to instantiate the bean. The home interface isn’t implemented in the bean itself, but by the container. Other methods declared in the home interface permit the client to locate an instance of a bean and to remove a bean instance when it is no longer needed.

Working

- Once the bean is deployed in the EJB container, the client calls the create() method defined in the home interface to instantiate the bean. The home interface isn’t implemented in the bean itself, but by the container. Other methods declared in the home interface permit the client to locate an instance of a bean and to remove a bean instance when it is no longer needed.

Remote Interface

- EJB container is acting as a layer of indirection between the client code and the bean. This layer of indirection manifests itself as a single network aware object, called the EJB object.
- Bean clients invoke methods on EJB objects, rather than the beans themselves. To perform this, EJB objects must clone every business method that your bean classes expose.

Remote Interface

- EJB container is acting as a layer of indirection between the client code and the bean. This layer of indirection manifests itself as a single network aware object, called the EJB object.
- Bean clients invoke methods on EJB objects, rather than the beans themselves. To perform this, EJB objects must clone every business method that your bean classes expose.

Home Object

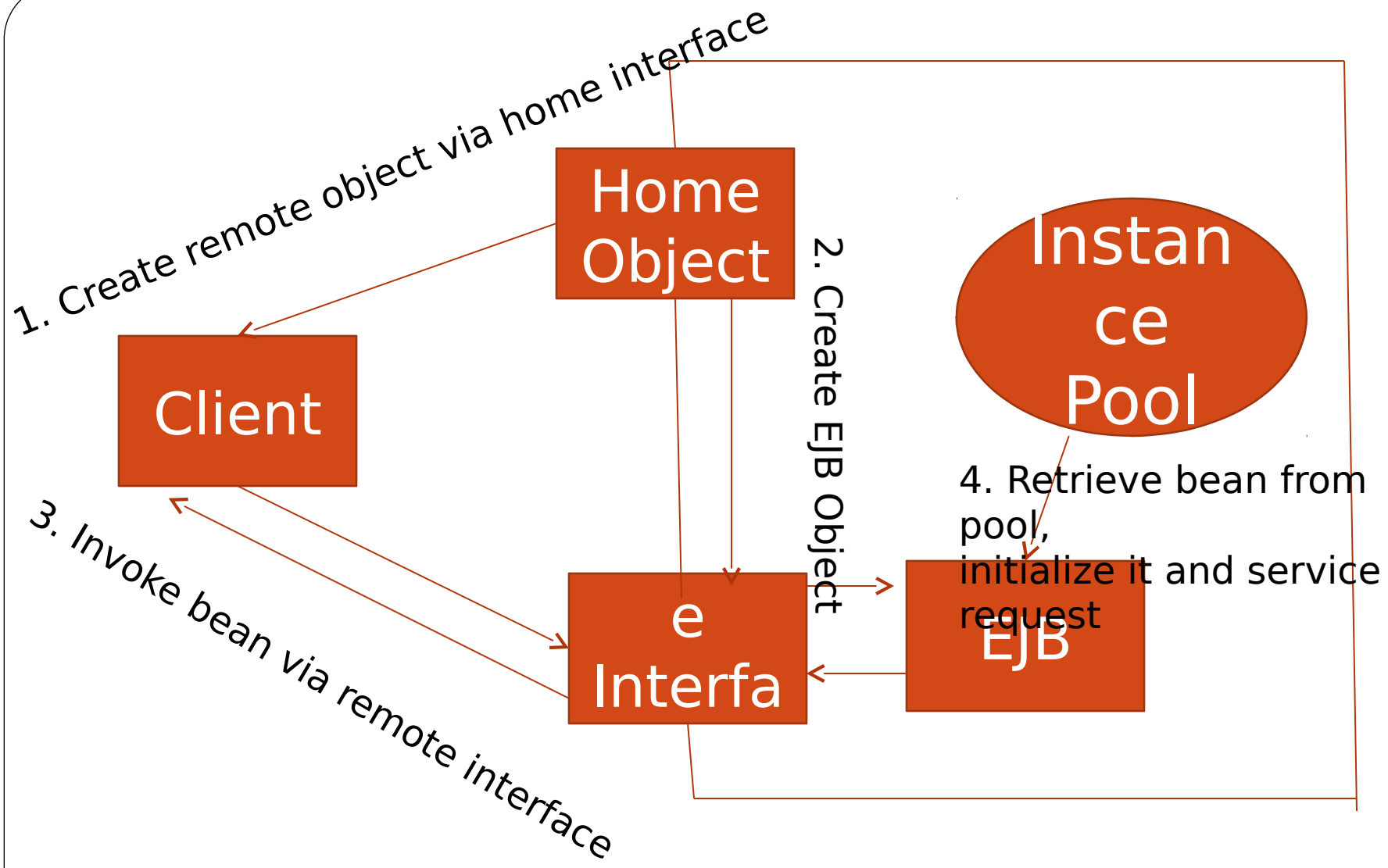
- The client cannot instantiate an EJB object directly because EJB objects could exist on a different machine than the one the client is on. Similarly, EJB promotes location transparency, so clients should never be aware of exactly where EJB objects reside.
- To acquire a reference to an EJB object, your client code asks for an EJB object from home_object.

Responsibilites of Home_Object

- Create EJB Objects
- Find existing EJB Objects
- Remove EJB Objects

Home Interface

- Home objects are factories for EJB Objects.
- Home interfaces simply define methods for creating, destroying and finding EJB objects. The container’s home object implements your home interface.


1. Create remote object via home interface
Home Object
2. Create EJB Object
Instan ce Pool
Client
4. Retrieve bean from pool, initialize it and service request
3. Invoke bean via remote interface
e Interfa
EJB

It's over!!

Thank you for attending :D

Priya. R
Vignesh. G
Vijay. V