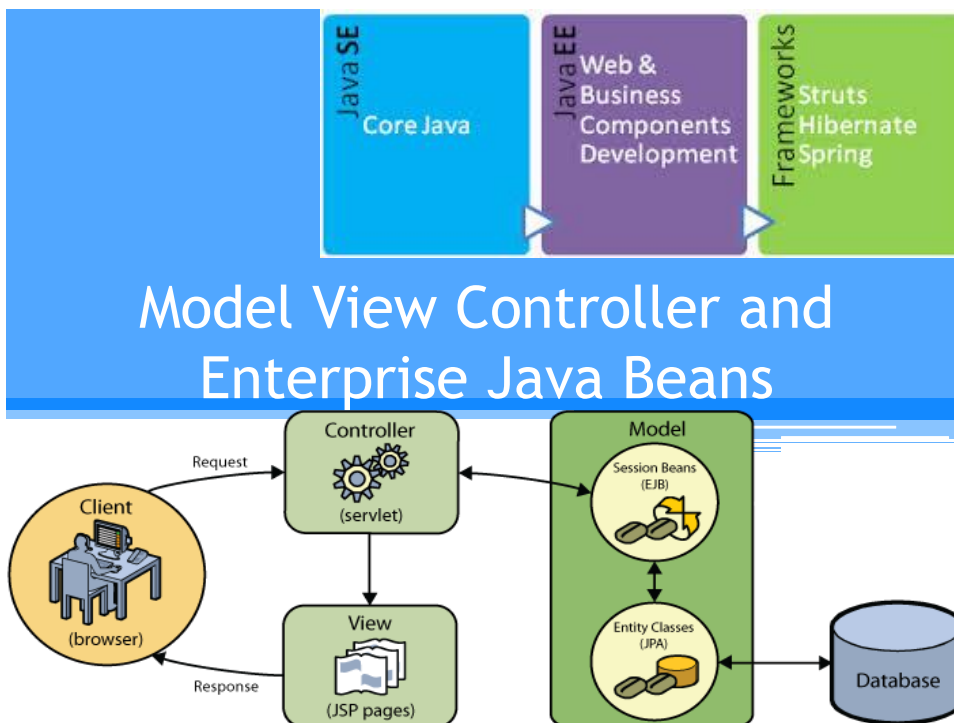


**CS-25 ADVANCE JAVA PROGRAMMING (J2EE)**  
**HANDOUT MATERIAL**  
BCA 5TH SEM.  
2018-19

## Unit 4 Part1

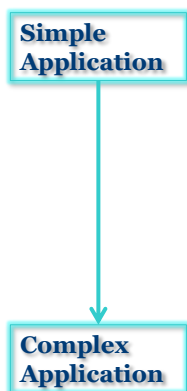
Dr. Sharon V. Mohtra  
Asst. Professor  
Dept. of Computer  
Applications  
Christ College  
Rajkot



# MVC (Model View Controller)

- Introduction to MVC
- Implementation of MVC Architecture

## Introduction to MVC



- **Some of the MVC Misconceptions**
- **An elaborate framework is necessary**
  - Frameworks are sometimes useful
    - Struts
    - JavaServer Faces (JSF)
  - They are *not required!*
  - Implementing MVC with the builtin RequestDispatcher - works very well for most simple and moderately complex applications
- **MVC totally changes overall system design**
  - You can use MVC for individual requests
  - Think of it as the MVC *approach*, *not the MVC architecture* - Also called the *Model 2 approach*

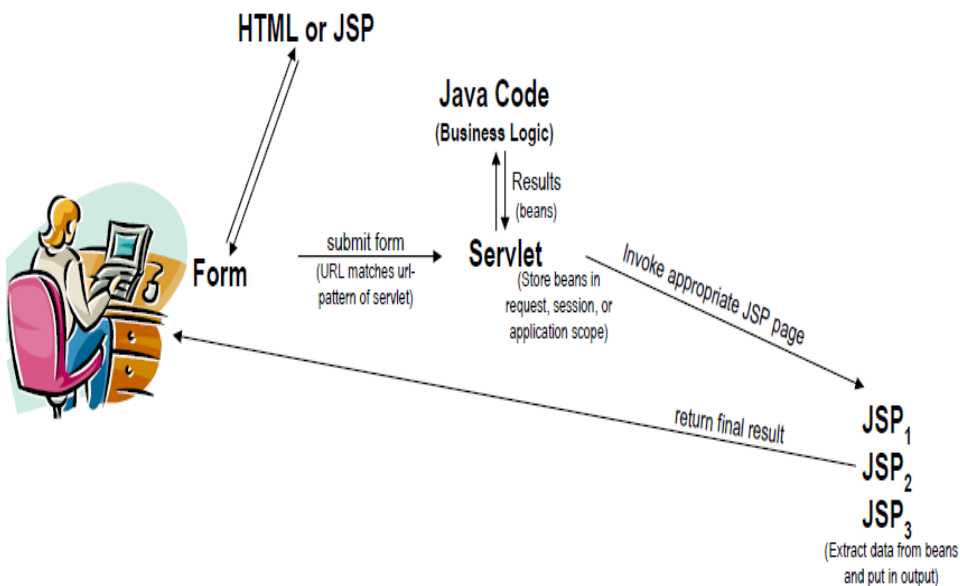
## Why Combine Servlets & JSP?

- **Typical picture: use JSP to make it easier to develop and maintain the HTML content**
  - For simple dynamic code, call servlet code from scripting elements
  - For slightly more complex applications, use custom classes called from scripting elements
  - For moderately complex applications, use beans and custom tags
- **But, that's not enough**
  - For complex processing, starting with JSP is awkward
  - Despite the ease of separating the real code into separate classes, beans, and custom tags, the assumption behind JSP is that a *single page gives a single basic look*

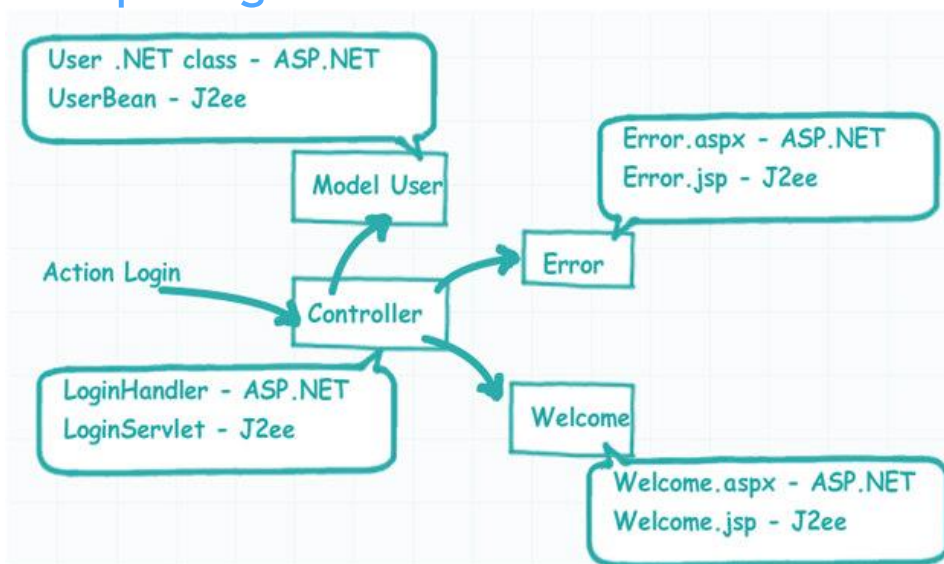
## Possibilities for Handling a Single Request

- **Servlet only. Works well when:**
  - Output is a binary type. E.g.: an image
  - There is *no output*. E.g.: *you are doing forwarding or redirection* as in Search Engine example.
  - Format/layout of page is highly variable. E.g.: portal.
- **JSP only. Works well when:**
  - Output is mostly character data. E.g.: HTML
  - Format/layout mostly fixed.
- **Combination (MVC architecture). Needed when:**
  - A single request will result in multiple substantially different looking results.
  - You have a large development team with different team members doing the Web development and the business logic.
  - You perform complicated data processing, but have a relatively fixed layout.

## MVC Flow of Control



## Comparing MVC : J2EE vs. .NET



## Determining the Architecture

- When working with JSP technologies, one can code all of the business logic into JSP pages using scriptlets. Scriptlets are snippets of Java code enclosed in `<% %>` tags.
- As JSP pages are compiled into servlets before they run, so Java code is perfectly valid in JSP pages.
- However, there are several reasons why this practice should be avoided, especially when working in large projects.

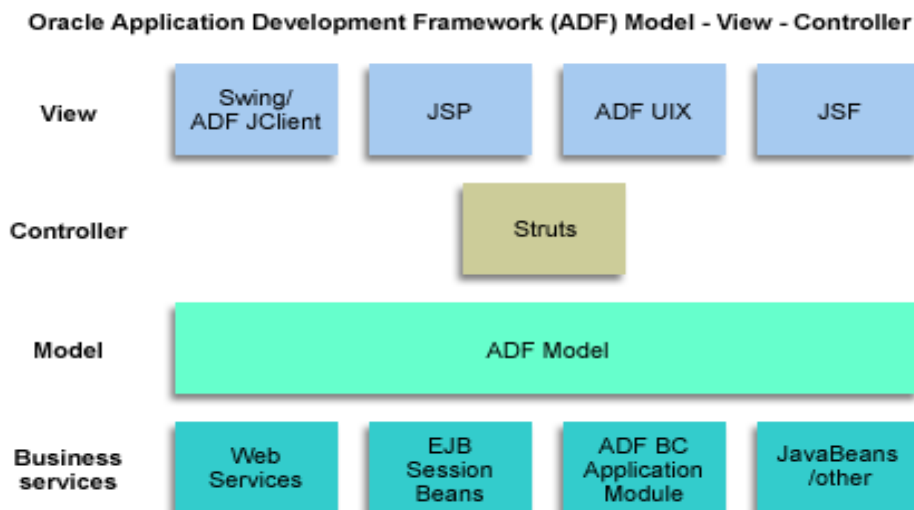
## Reasons for avoiding Scriptlets in Enterprise Applications - I

- **Scriptlet code is not reusable:** Scriptlet code appears in exactly one place: the JSP page that defines it. If the same logic is needed elsewhere, it must be either included (decreasing readability) or copied and pasted into the new context.
- **Scriptlets mix logic with presentation:** Scriptlets are islands of program code in a sea of presentation code. Changing either requires some understanding of what the other is doing to avoid breaking the relationship between the two. Scriptlets can easily confuse the intent of a JSP page by expressing program logic within the presentation.

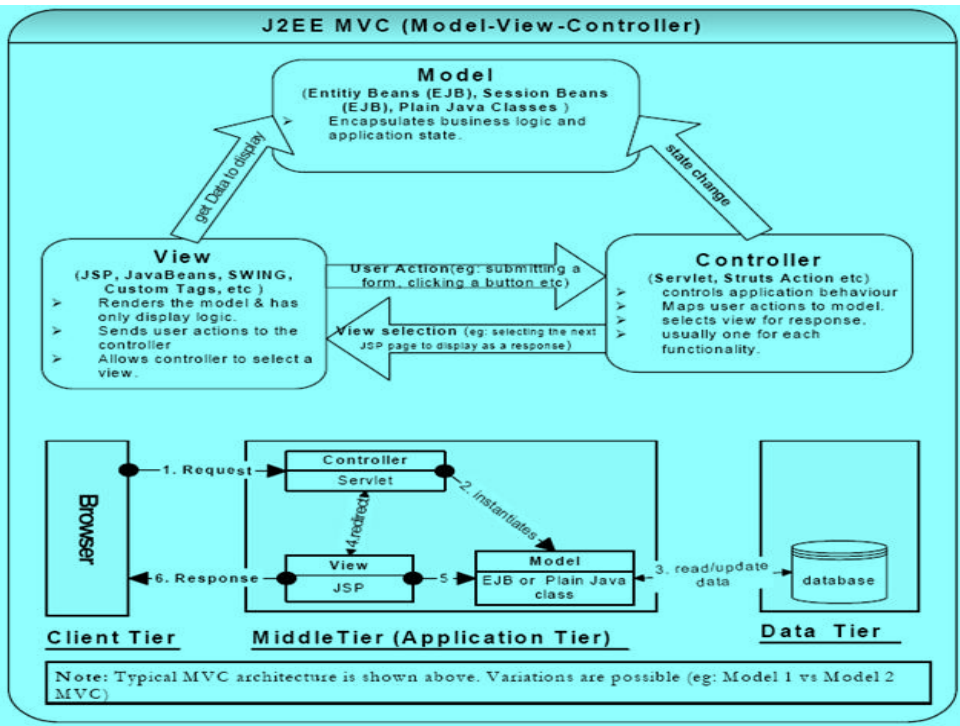
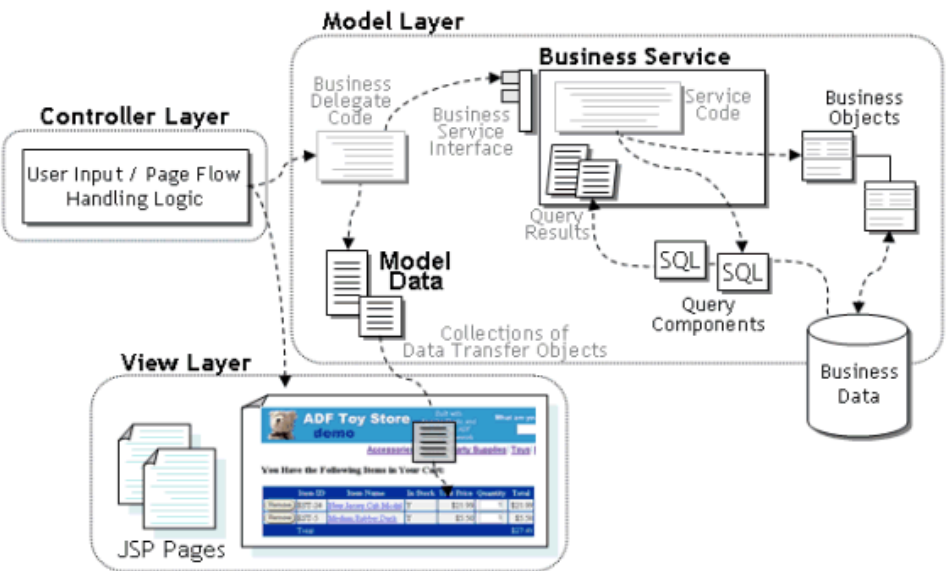
## Reasons - II

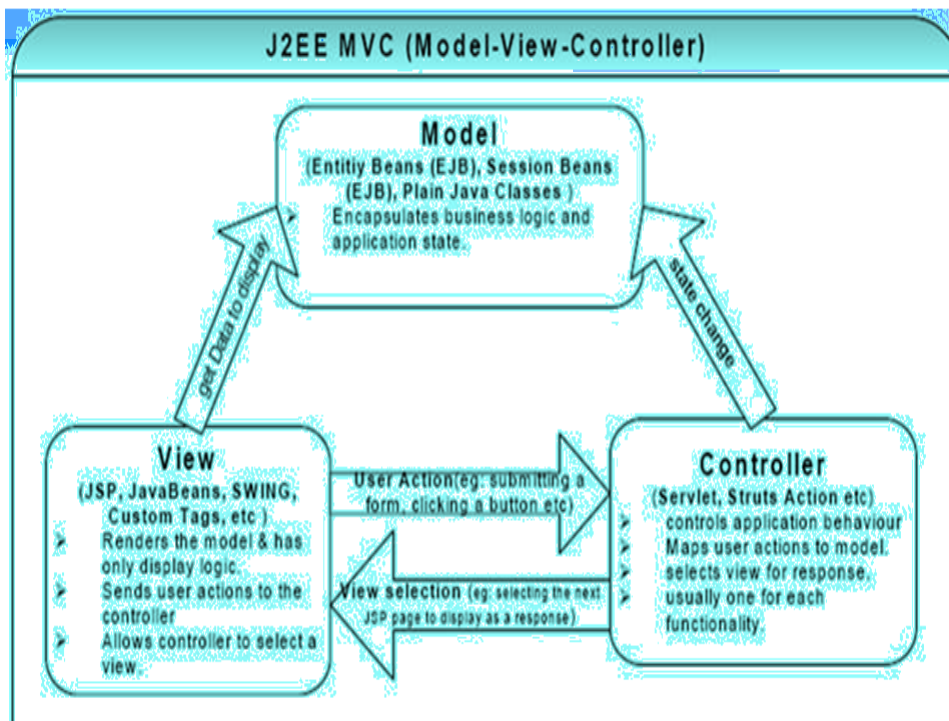
- **Scriptlets break developer role separation:** Because scriptlets mingle programming and Web content, Web page designers need to know either how to program or which parts of their pages to avoid modifying.
- **Scriptlets make JSP pages difficult to read and to maintain:** JSP pages with scriptlets mix structured tags with JSP page delimiters and Java language code.
- **Scriptlet code is difficult to test:** Unit testing of scriptlet code is virtually impossible. Because scriptlets are embedded in JSP pages, the only way to execute them is to execute the page and test the results.

## Implementation of MVC Architecture

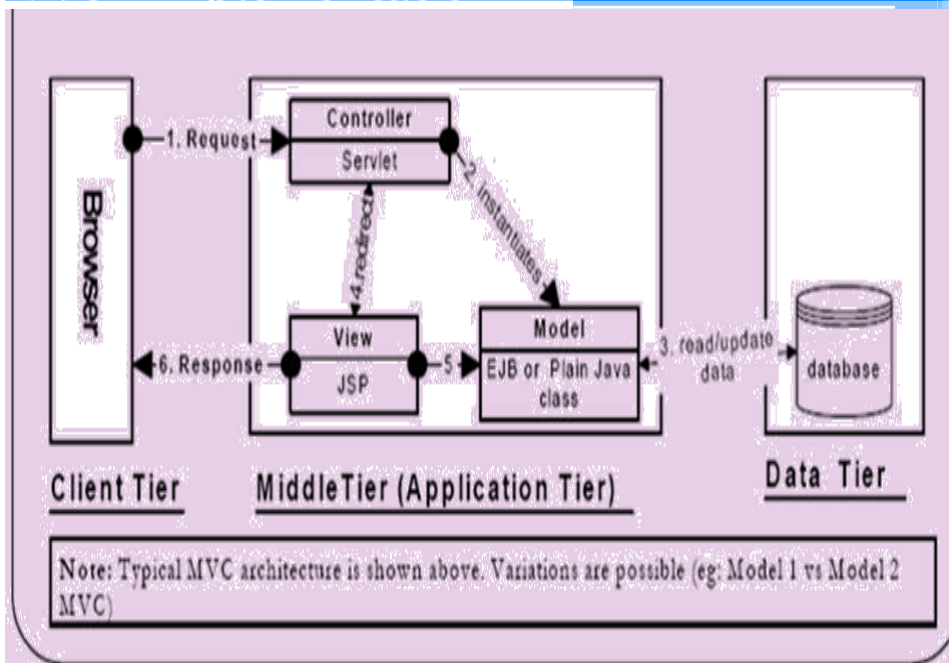


# MVC By Oracle ADF™





## MVC - Implementation





# MVC part I - without framework

## Java Beans – JSP – Servlet

```
public class UserBean
```

The model - Javabean in J2EE

```
{  
    public UserBean() { this.username="user";  
        this.password="pass"; }  
    public String getPassword() { return  
        password; }  
    public void setPassword(String password)  
        { this.password = password; }  
  
    public String getUsername() { return  
        username; }  
    public void setUsername(String  
        username) { this.username = username; }  
  
    public boolean IsValid (String  
        username,String password) { return  
        this.username.equals(username) &&  
        this.password.equals(password); }  
}
```

•For example, create a simple class called as “User”, this class will have two properties “Username” and “Password”.

•The client, which for now the controller will set these two properties can call the “IsValid” function to check if the user is valid or not.

## J2EE Controller using Servlet

```
public class LoginServlet extends
    HttpServlet
{
    protected void
    processRequest(HttpServletRequest
    request, HttpServletResponse response)
    throws ServletException, IOException
    {
        // Write the code for

        // connecting the model to the view.
    }
}
```

- To create a controller in J2EE we create a class which inherits from 'HttpServlet' class. The logic of the controller is written in the "processrequest" method.
- You can access the request and response object using the 'HttpServletRequest' class and 'HttpServletResponse' class.

## J2EE Controller using Servlet

```
public class LoginServlet extends
    HttpServlet
{
    protected void
    processRequest(HttpServletRequest
    request, HttpServletResponse response)
    throws ServletException, IOException
    { // Write the code for connecting the model to the
    //view.
        String username = (String)
        request.getParameter("username";
        String password = (String)
        request.getParameter("password");
        UserBean model = new UserBean();
    }
}
```

- To create a controller in J2EE we create a class which inherits from 'HttpServlet' class. The logic of the controller is written in the "processrequest" method.
- You can access the request and response object using the 'HttpServletRequest' class and 'HttpServletResponse' class.

```

public class LoginServlet extends HttpServlet
{
    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException
    { // Write the code for connecting the model to the //view.
        boolean isValidUser = model.isValid();
        if(isValidUser) { request.setAttribute
            ("welcome","Welcome"+username); }
        else
            nextJsp = "Error.jsp"; RequestDispatcher

        dispatch =
        request.getRequestDispatcher(nextJsp);
        dispatch.forward(request,response);
    }
}

```

Checking  
Logic



```

<servlet-name>LoginServlet</servlet-name>
<servlet-class>com.controller.LoginServlet
</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>LoginServlet</servlet-name>
<url-pattern>/Login</url-pattern>
</servlet-mapping>

```

```
<form action="Login"
  method="post">
Username<input type="text"
  name="username" value="" />
<br/>
Password<input
  type="password"
  name="password" value="" />
<br/>

<input type="submit"
  value="Submit" /> </form>
```

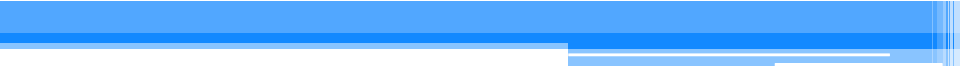
J2EE  
[index.jsp](#)

The view is nothing but a simple page with the form tag and action having the URL pattern.



## Enterprise Java Beans (EJB)

- Introduction
- Benefits of EJB
- Restriction on EJB
- Types of EJB
  - Session Beans
  - Entity Beans
  - Message-driven beans (MDBs)
- Timer service



“An EJB is reusable software component that make a collection of Java classes and XML file, bundled into a single unit.”



## Introduction

- Enterprise Java Beans ( EJB ) is
  - a *middleware component model* for Java and CORBA
  - a **specification** for creating server-side, scalable, transactional, multi-user and secure enterprise-level applications
  - one of several Java APIs in the Java
- Presented by Sun in the 1999, they are easier than other technologies as RMI or Corba

## The EJB specification

- It is one of several Java APIs in the Java EE specification. EJB is a server-side model that encapsulates the business logic of an application.
  - Originally developed in 1997 by **IBM** and later adopted by **Sun Microsystems** (EJB 1.0 and 1.1) in 1999 and enhanced under the Java Community Process as JSR 19 (EJB 2.0), JSR 153 (EJB 2.1), JSR 220 (EJB 3.0), JSR 318 (EJB 3.1) and JSR 345 (EJB 3.2).
  - Intends to provide a standard way to implement the back-end 'business' code.
  - Addresses the same types of problems, and solutions to these problems are often repeatedly re-implemented by programmers.
  - Intended to handle such common concerns as
    - persistence,
    - transactional integrity,
    - and security
- in a standard way, leaving programmers free to concentrate on the particular problem at hand.

## Version History

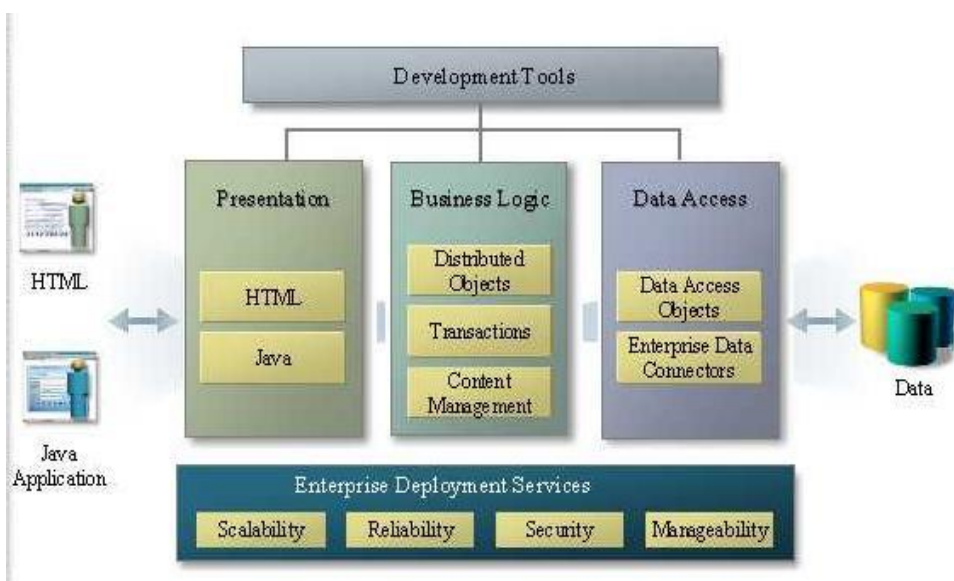
- EJB 3.2, final release (2013-05-28)
- EJB 3.1, final release (2009-12-10)
- EJB 3.0, final release (2006-05-11)
- EJB 2.1, final release (2003-11-24)
- EJB 2.0, final release (2001-08-22)
- EJB 1.1, final release (1999-12-17)
- EJB 1.0 (1998-03-24)

## EJB specification details how an application server provides the following responsibilities:

1. **Transaction processing**
2. **Integration** with the persistence services offered by the Java Persistence API (JPA)
3. **Concurrency control**
4. **Event-driven programming** using Java Message Service and Java EE Connector Architecture (AS – EIS)
5. **Asynchronous method invocation**
6. **Job scheduling**
7. **Naming and directory services (JNDI)**
8. **Interprocess Communication** using RMI-IIOP(Internet Inter-Orb Protocol) and Web services
9. **Security** (JCE(Java Cryptography Extension) and JAAS(Java Authentication and Authorization Service))
10. **Deployment** of software components in an application server

## Introduction

- This is the three level structure for Application Server



## Application Server

- Presentation
  - HTML Application
  - Java Application
- Business Logic
- Data Access

## Presentation

- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>• HTML<ul style="list-style-type: none"><li>▫ Generated server-side HTML</li><li>▫ Runs on any Web browser</li><li>▫ Less client-side power</li></ul></li></ul> | <ul style="list-style-type: none"><li>• Java<ul style="list-style-type: none"><li>▫ Required Java virtual Machine</li><li>▫ More client side power</li><li>▫ Runned on a page</li><li>▫ Security (Applet)</li><li>▫ Launched from a browser or a standalone application</li></ul></li></ul> |
|---|---|



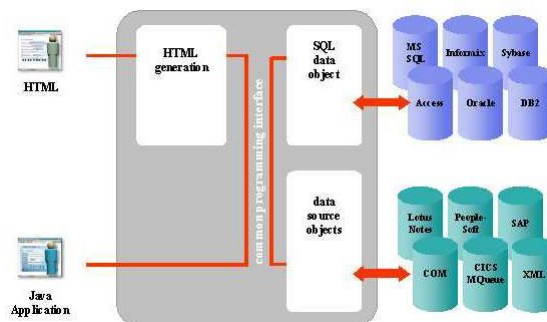
## Business Logic

- Implements the logic of the application defining all the function that may be used from a client
  - Change Business Rules Easily
  - Re-use components
  - Make complex applications manageable
  - Secure Data hiding



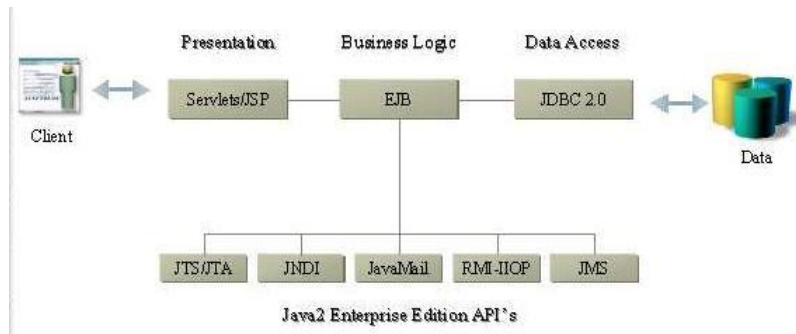
## Data Access

- Utility to access external datas such as Database or other Web component
- Access other SOA



## J2EE Application Server

- Java 2 Enterprise Edition standardizes interfaces for Application Server components



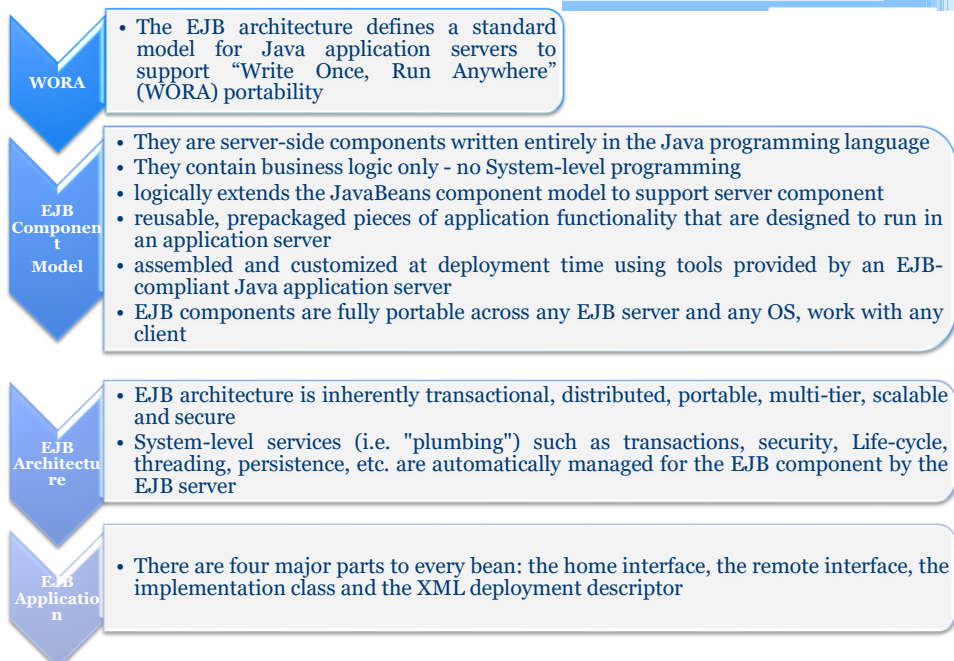
## What is EJB?

- An EJB is just a collection of Java classes and XML file, bundled into a single unit. The Java classes must follow certain rules and provide certain callback methods.
- EJB is just a specification. It is not a product.
- EJBs are reusable components.
- It is a server side component written in Java Language.
- Industry standard distributed component model.
- Incorporates the business logic of an application ( the code that implements the purpose of the application).
- Replicates the table model as objects.
- EJB is a widely-adopted server-side component architecture for J2EE.
- EJB components are designed to encapsulate business logic, and to protect the application developer from having to worry about system level issues.

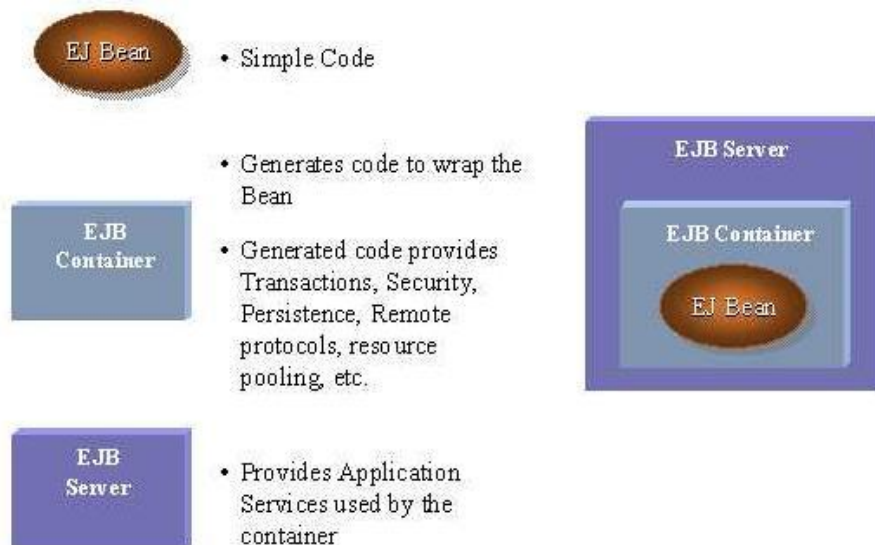
## EJB Properties

- Bean writers need not write
  - Remote access Protocols
  - Transactional Behaviour
  - Threads
  - Security
  - State Management
  - Object life cycle
  - Resource pooling
  - Persistence
  - Native queries execution

## Features



## EJB Overview



EJB	JavaBeans
The EJB architecture provides a format for encapsulation and management of business logic.	The JavaBeans architecture is meant to provide a format for general-purpose components .
EJB is at Server-side (specifically business logic tier).	JavaBeans has tier of execution at Client .
The EJB runtime environment provides services of Persistence, declarative transactions and security, connection pooling and lifecycle services.	The runtime execution environment provides services like Java libraries, Java application etc.
An EJB is a nonvisual, remote object.	JavaBeans may be visible or nonvisible at runtime.
EJB's are remotely executable components or business objects that can be deployed only on the server.	JavaBeans are intended to be local to a single process and are primarily intended to run on the client side. Although one can develop server-side JavaBeans, it is far easier to develop them using the EJB specification instead.
Even though EJB is a component technology, it neither builds upon nor extends the original JavaBean specification.	JavaBeans is a component technology to create generic Java components that can be composed together into applets and applications.

EJB	JavaBeans
EJBs have a deployment descriptor that describes its functionality to an external builder tool or IDE.	JavaBeans have an external interface called the properties interface, which allows a builder tool to interpret the functionality of the bean.
EJB's have no concept of BeanInfo classes, property editors or customizers and provide no additional information other than that described in the deployment descriptor.	JavaBeans may have BeanInfo classes, property editors or Customizers.
EJBs are of three types - session beans, entity beans and message-driven beans.	JavaBeans have no types.
EJB's may be transactional and the EJB servers provide transactional support.	No explicit support exists for transactions in JavaBeans.
An EJB cannot be deployed as an ActiveX control because ActiveX controls are intended to run at the desktop and EJB's are server side components. However CORBA-IIOP compatibility via the EJB-to-CORBA mapping is defined by the OMG	Component bridges are available for JavaBeans.

## Benefits of EJB

- **Component portability**
  - The EJB architecture has a simple, elegant component container model such that the Java server components can be developed once and deployed in any EJB-compliant server.
- **Architecture independence**
  - The platform, proprietary protocol, or middleware infrastructure independency is observed in the EJB architecture. Applications developed for one platform can be redeployed on any other platforms.
- **Developer productivity**
  - The EJB architecture standard makes the automatic the use of complex infrastructure services such as transaction management and security checking. This helps developers to create complex applications by focusing on business logic rather than environmental and transactional issues.

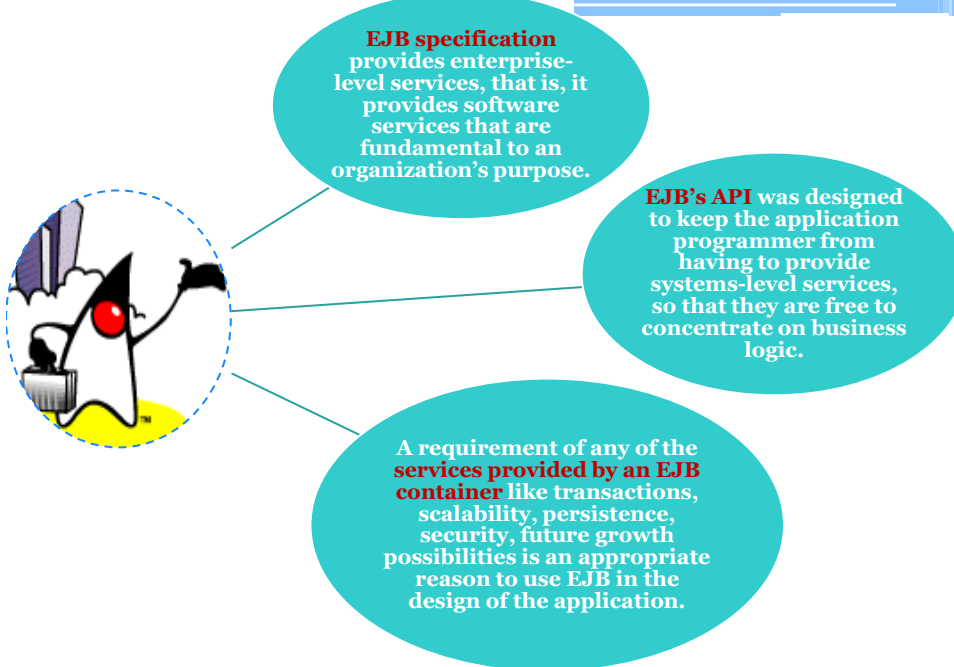
## Benefits of EJB

- Customization
  - Enterprise bean applications can be easily customized without access to the source code. The behavior and runtime settings are defined through attributes that can be modified when the enterprise bean application is deployed.
- Multitier technology
  - The EJB architecture overlays existing infrastructure services at multiple levels.
- Versatility and scalability
  - The EJB architecture can be used for small-scale or large-scale business transactions. The enterprise beans can be migrated to more powerful operating environments as and when processing requirements grow.

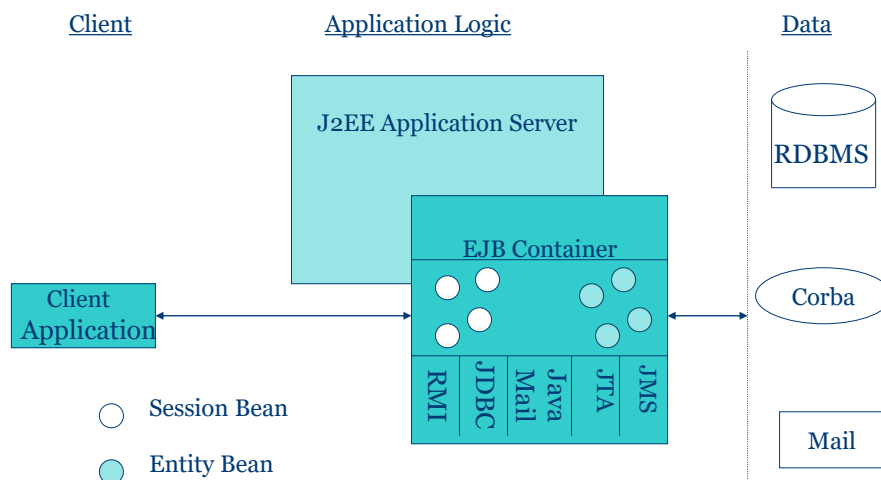
## EJB Restrictions

<b><i>Run-time distribution</i></b>	<ul style="list-style-type: none"><li>• Enterprise bean instances may be distributed; that is, running in separate Java Virtual Machines (JVMs) or on separate machines. Some restrictions exist due to logical results of distributed programming semantics.</li></ul>
<b><i>Security</i></b>	<ul style="list-style-type: none"><li>• Many restrictions imposed maintain the integrity of the enterprise bean security model.</li></ul>
<b><i>Container control</i></b>	<ul style="list-style-type: none"><li>• The container is responsible for coordinating system services, controlling threads, managing security, bean lifecycle, and so on. Some restrictions prevent enterprise bean classes from interfering with proper container operation.</li></ul>
<b><i>Server model</i></b>	<ul style="list-style-type: none"><li>• The EJB server, EJB container, and EJB component have clearly-specified roles in the enterprise bean programming model. These may take over the enterprise bean classes' responsibilities of the server or container.</li></ul>

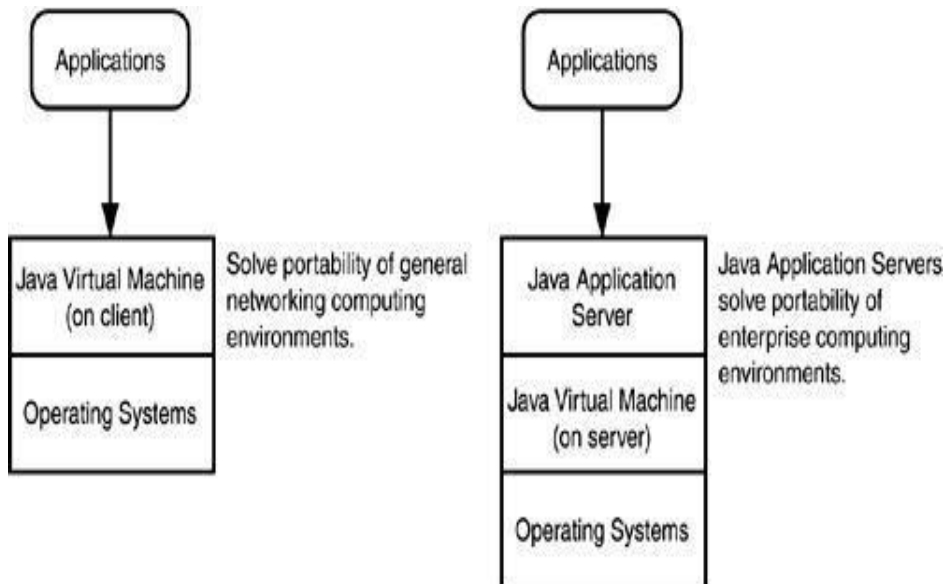
## Why use EJBs in your design?



## EJB Architecture - I



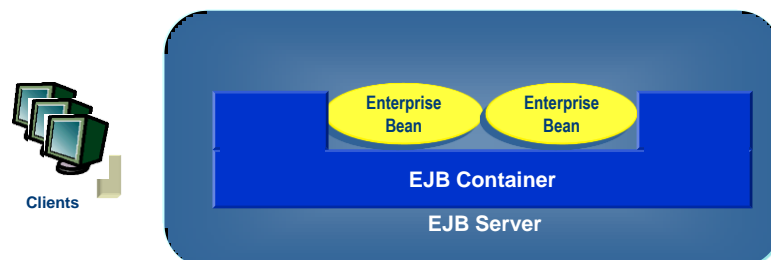
## Java on the client and on the server



## EJB Architecture - II

**The EJB architecture specifies the responsibilities and interactions among EJB entities**

- ◆ **EJB Servers**
- ◆ **EJB Containers**
- ◆ **Enterprise Beans**
- ◆ **EJB Clients**





## EJB Server

Provides a Runtime Environment

The EJB Server provides system services and manages resources

- Process and thread management
- System resources management
- Database connection pooling and caching
- Management API



## EJB Container

Provides a Run-time Environment  
for an Enterprise Bean

Hosts the Enterprise JavaBeans

Provides services to Enterprise JavaBeans

- Naming
- Life cycle management
- Persistence (state management)
- Transaction Management
- Security

Likely provided by server vendor

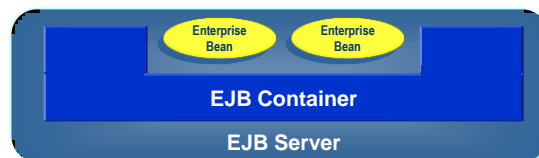


## Enterprise JavaBeans

A specialized Java class where the real business logic lives

- May be developer-written or tool-generated
- Distributed over a network
- Transactional
- Secure

Server vendors provide tools that automatically generate distribution, transaction and security behavior



## EJB Clients

Client access is **controlled** by the container in which the enterprise Bean is deployed

Clients **locates** an Enterprise JavaBean through Java Naming and Directory Interface (JNDI)

RMI is the standard method for accessing a bean over a network



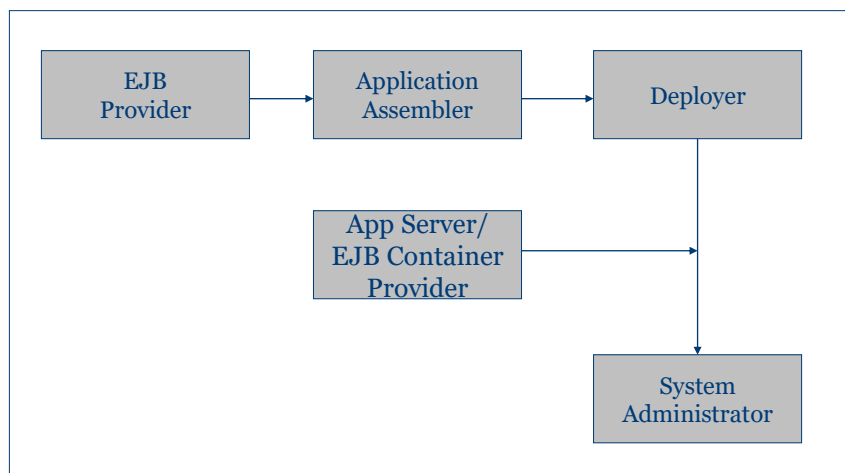
## What's Unique About EJB

**Declarative Programming Model**

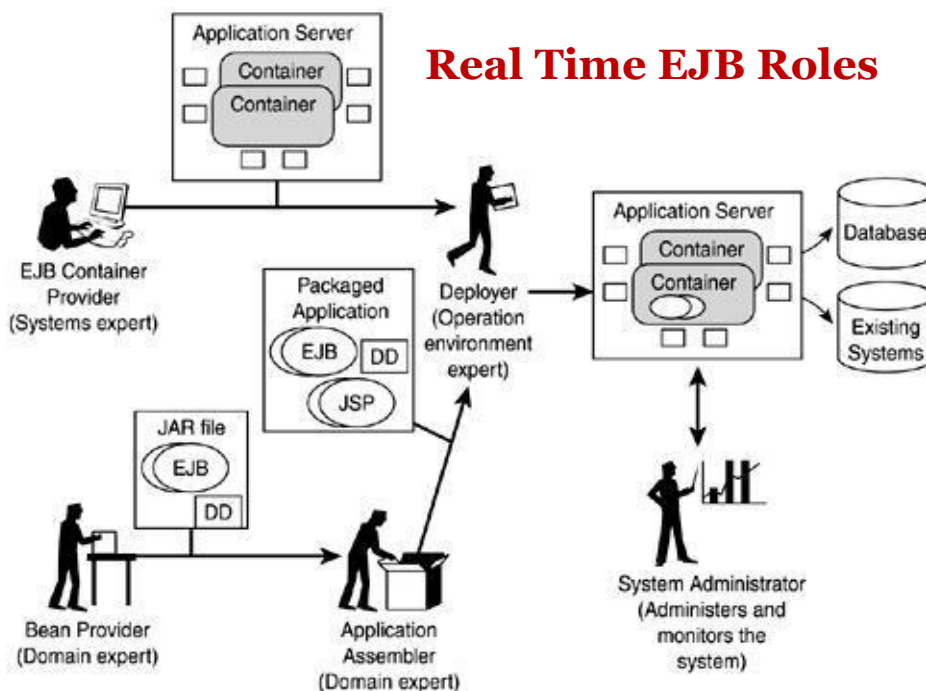
Mandates a container model where common services are declared, not programmed

- At **development and/or deployment time**, attributes defining the bean's transaction and security characteristics are specified
- At **deployment time**, the container introspects the Enterprise JavaBean attributes for the runtime services it requires and wraps the bean with the required functionality
- At **runtime**, the container intercepts all calls to the object
  - Provides transactional, threading and security behavior required before the method invocation
  - Invokes the method on the object
  - Cleans up after the call

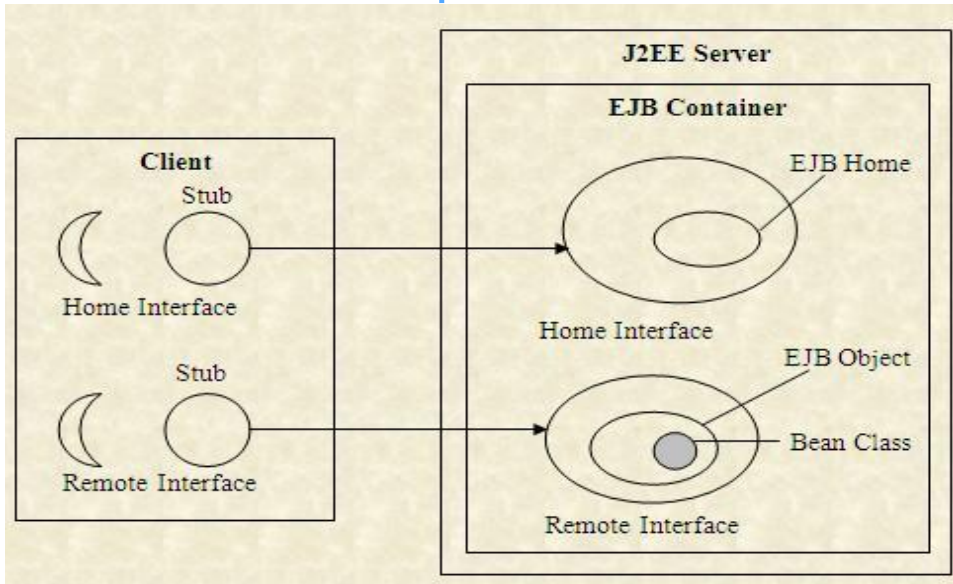
## Roles in EJB Development



Role	Description
EJB provider	a person who develops EJB Components
EJB Deployer	a person responsible for deploying EJB's in EJB server
Application Server/ EJB Container Vendor	one who provides application server on which the application is deployed
Application assembler	one who combine the EJB components with other software to make a complete application
System administrator	one who manages the application after it has been deployed into a target environment



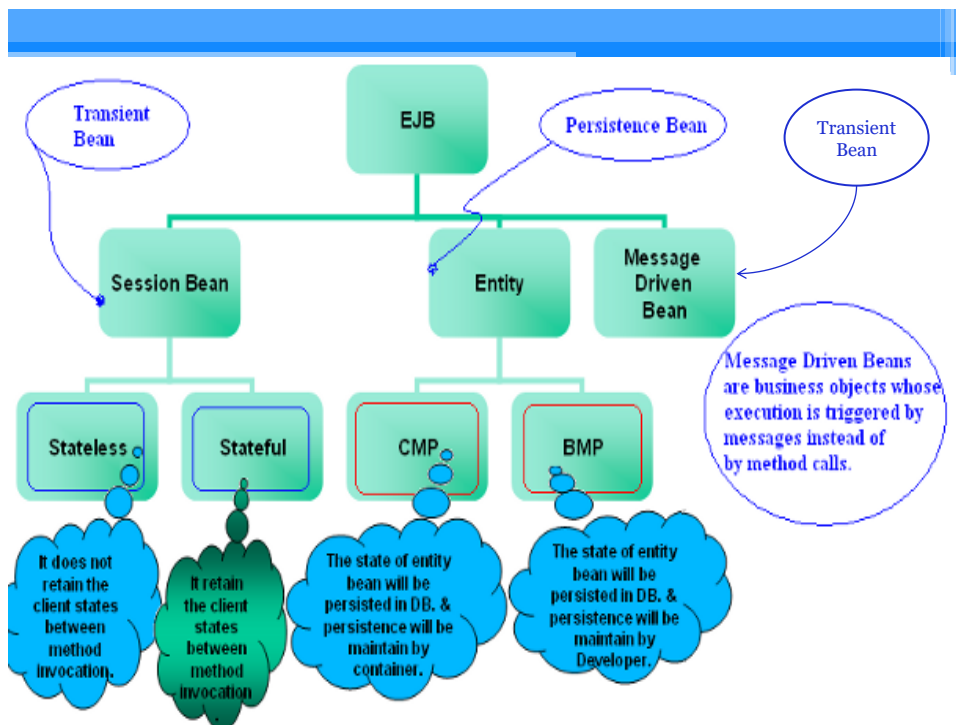
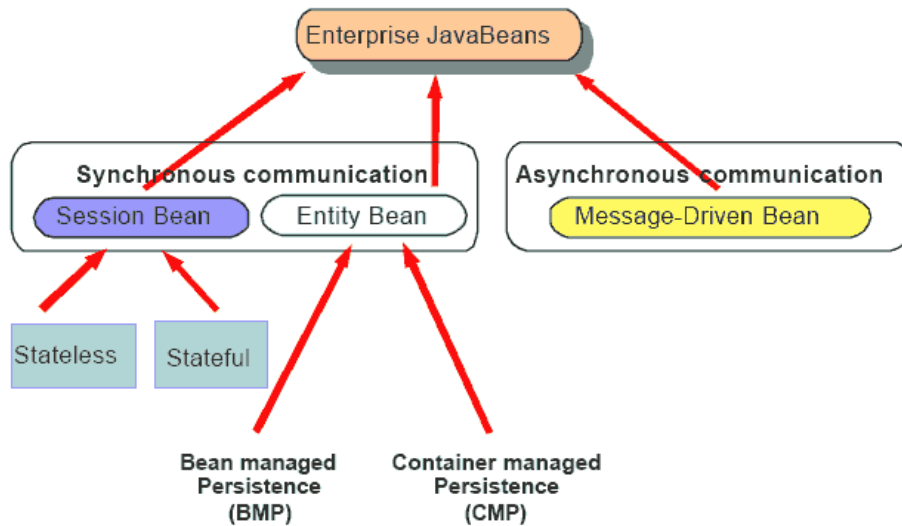
## Architecture Components



## Varieties of Beans

- Session Beans
  - ⤴ Stateful session bean
  - ⤴ Stateless session bean
- Entity Beans
  - ⤴ With container-managed persistence
  - ⤴ With bean-managed persistence
- Message-Driven Beans

# Enterprise JavaBeans



## 1. Session Bean

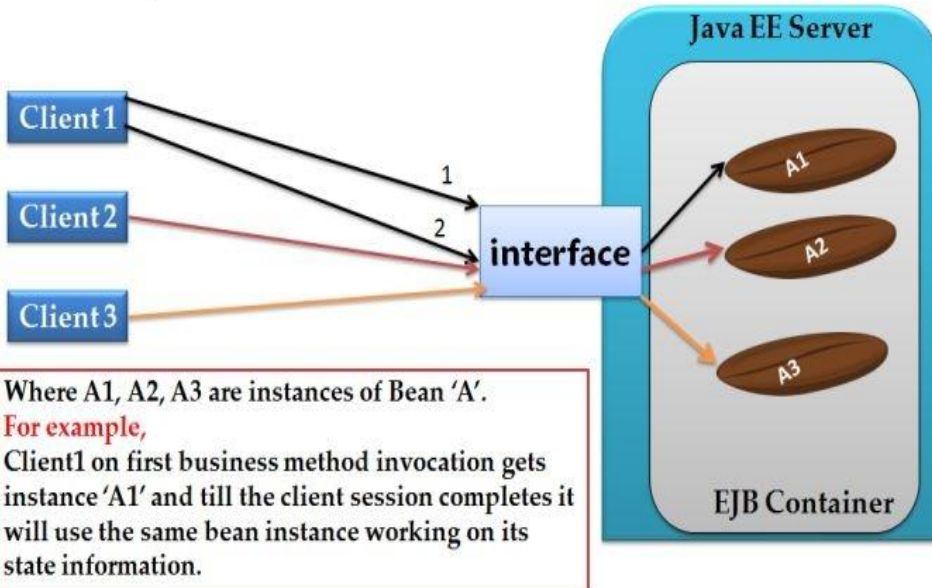
- Represents a single client inside the application server
- The client calls the session bean to invoke methods of an application on the server
- Perform works for its client, hiding the complexity of interaction with other objects in the server
- As its name suggests, a session bean is similar to an interactive session
- Is not shared
- Is not persistent but transient
- When the client stops the session, the bean can be assigned to another client from the server
- Unique to each client
- Types
  - **Stateful session bean**
  - **Stateless session bean**

### (i) Stateful Session Bean

- The state of an object consists of the values of its instance variables.
- Contains the state of a single client session:
  - Information on the client
  - On method called
  - Return values
- This state is called *conversational state* and is not retained when the session ends, also if the client not removes the bean
- Remembers previous request, and response of session
- The state is retained for the duration of the client-bean session (If the client removes the bean or terminates, the session ends and the state disappears)
- This **transient** nature of the state is not a problem, however, because when the conversation between the client and the bean ends there is no need to retain the state.

( Stateful session bean's life cycle )

## Stateful Session Bean State Management



## Stateful session bean's life cycle

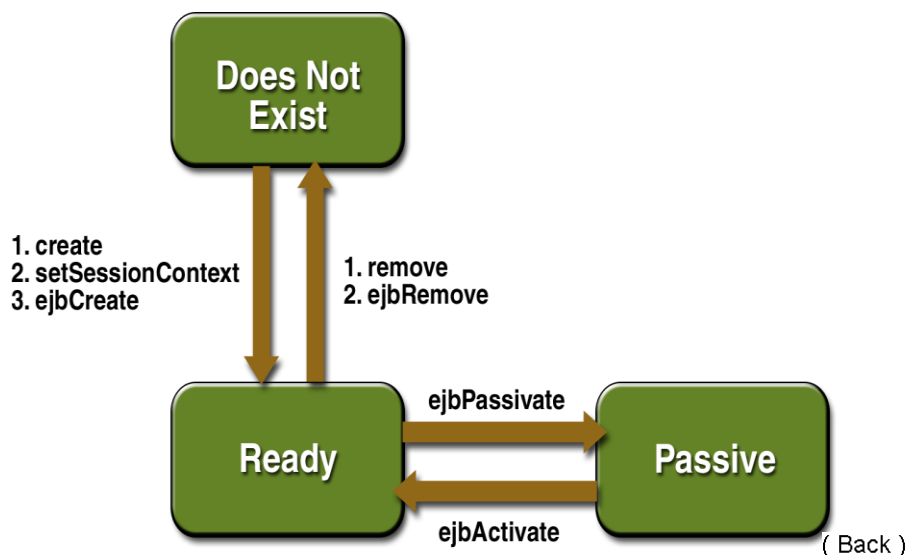
- The client invoke the create method
- The EJB container :
  - Instantiates the bean
  - Invokes the `setSessionContext`
  - Invokes `ejbCreate`
- The bean is ready
- Business methods ready to be called



## Stateful session bean's life cycle

- While in the ready state
  - EJB container may *passivate* the bean moving it from memory to secondary storage
  - A client may invoke a business method
  - EJB container may activate a bean, moving it back to the ready stage, and then calls the bean's `ejbActivate` method
  - A client may invoke the remove method and the container calls the bean's `ejbRemove` method
  - Client cannot invoke *passivate*

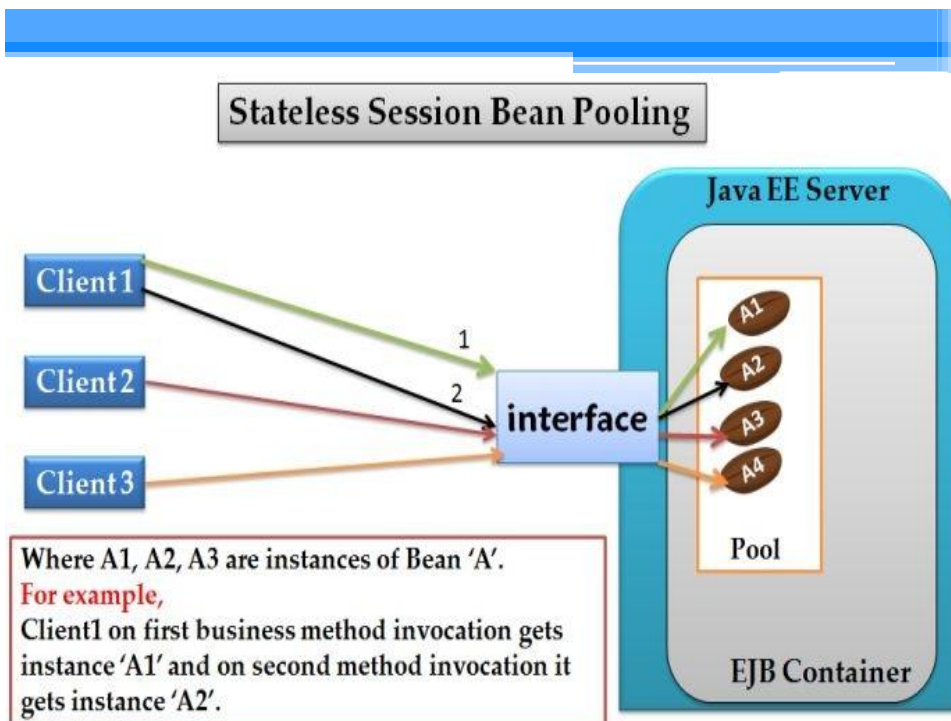
## Stateful session bean's life cycle



## (ii) Stateless Session Bean

- Not maintain a **conversational** state for a particular client
- Contains values only for the duration of the **single** invocation
- When the method is finished, the client-specific state should **not be retained**.
- Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean.
- Except during method invocation, all instances of stateless session bean are **equivalent** allowing the EJB container to assign an instance to any client. (That is, the state of a stateless session bean should apply across all clients.)
- Because stateless session beans can support multiple clients, they can offer better scalability for applications that require large numbers of clients. Typically, an application requires **fewer** stateless session beans than stateful session beans to support the same number of clients.
- A stateless session bean **can implement a web service**, but other types of enterprise beans cannot.

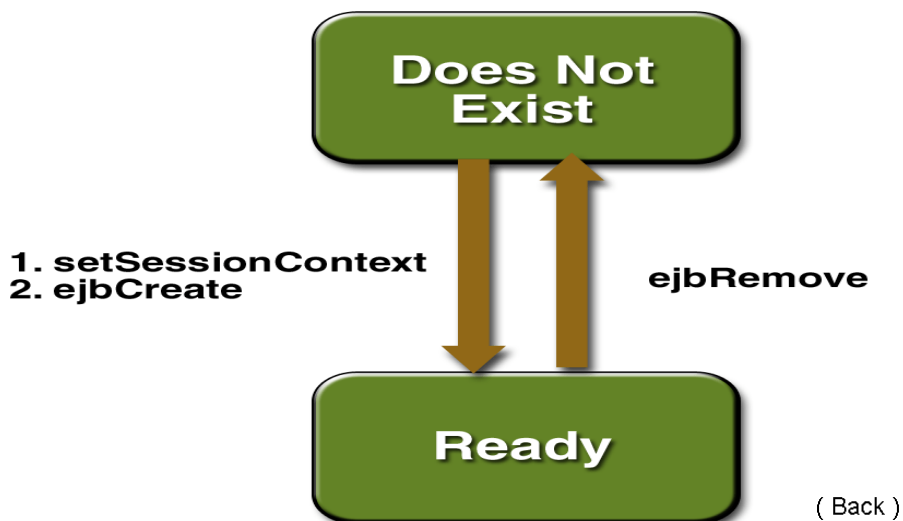
( Stateless session bean's life cycle )



## Stateless session bean's life cycle

- The client invoke the create method
- The EJB container :
  - Instantiates the bean
  - Invokes the `setSessionContext`
  - Invokes `ejbCreate`
- The bean is ready
- While in the ready state
  - A client may invoke a business method
  - A client may invoke the remove method and the container calls the bean's `ejbRemove` method
  - It's never passivate
  - It's can be pooled

## Stateless session bean's life cycle



### (iii) Singleton Session Bean

- Available from **EJB 3.1**, Singleton Session Beans are business objects having a **global shared state** within a JVM.
- The **@Singleton** annotation is used to mark the class as Singleton Session Bean.
- They are instantiated once per application and exist for the lifecycle of the application.
- In cases where the container is distributed over many virtual machines, each application will have one bean instance of the Singleton for each JVM.
- It **maintains its state between client invocations** but that state is not required to survive container shutdown or crash.
- A Singleton session bean is intended to be **shared and supports concurrent access by clients**.
- Concurrent access to the one and only bean instance can be controlled by the container (Container-managed concurrency, CMC) or by the bean itself (Bean-managed concurrency, BMC).

## 2. Entity Bean

- Represents a business object in a persistent storage mechanism such as a relational database
- Usually is a table in the database and each instance of that entity bean is a row in that table

Properties:

- Persistent
- Allow shared access
- Have primary key
- Have relationship with other entity beans
- Auto commit

### Types

- **Bean managed persistence**
- **Container managed persistence**

( Entity bean's life cycle )

### (i) Bean managed persistence

- Who write the bean's code must access the database and save his own data
- You will have more control over how the entity bean accesses a database

### (ii) Container managed persistence

- The container save the data
- There is no code in the bean for access the database
- The container handles all database access required for the bean
- Links between beans are created using a structure called **abstract schema**
- The EJB container transparently and implicitly manages the persistent state

## Entity bean's shared access

- Entity beans can be used by different clients
- It's important that they work within transactions
- The EJB container provides transaction management
- The transaction's attributes are specified in the bean's deployment description
- Concurrency management

## Entity bean's primary key

- Each entity bean has a unique object identifier like a key in a database table
- Each instance represents as Row in table

## Entity bean's relationship

- Container managed persistent
  - The container performs all the operation to create relationship
- Bean managed persistent
  - The code to perform relations must be written in the bean

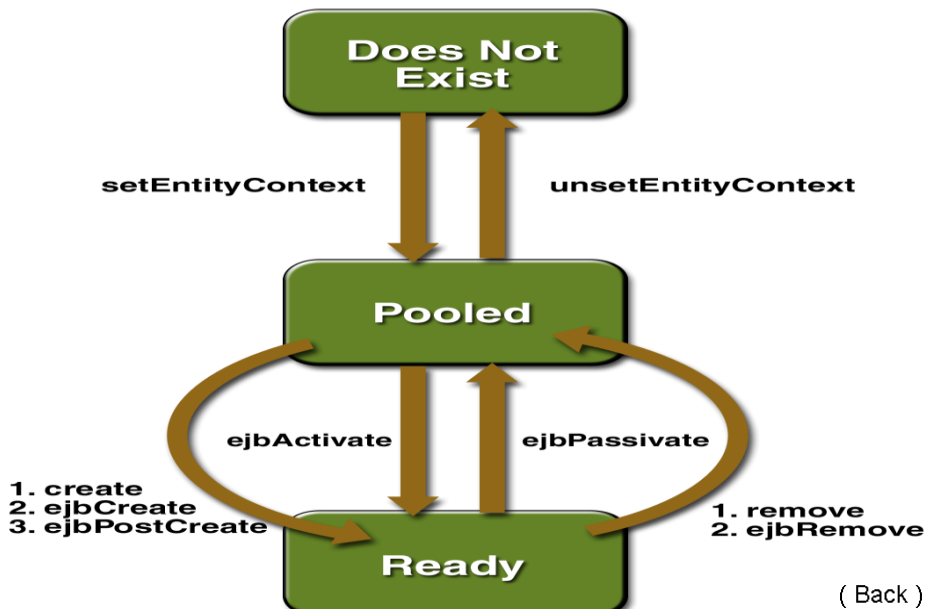
## Entity bean's life cycle

- The EJB container :
  - Creates the instance
  - Calls the `setEntityContext`
- The entity bean moves to a pool of available instances

## Entity bean's life cycle

- While in the pool :
  - Instance is not associated with any particular object identity
  - All instances in the pool are identical
  - EJB container may assign an identity to an instance when moving it to the ready stage invoking the `ejbActivate` method
  - A client may invoke the `create` method
    - EJB container calls `ejbCreate` and `ejbPostCreate`
  - EJB container may remove the instance invoking `unsetEntityContext`
  - Same bean instance (row) shared by all client
- While in the ready state :
  - A client may invoke entity bean's business methods
  - A client may invoke the `remove` method
    - EJB container calls the `ejbRemove` method
  - EJB container may invoke the `ejbPassivate` method

## Entity bean's life cycle



## Message Driven bean

- Allows applications to process messages asynchronously
- The messages may be sent by :
  - An application client
  - Another enterprise bean
  - A Web component
  - A JMS Client
- Retain no data or conversational state for a specific client
- All instances are equivalent, allowing the EJB container to assign a message to any message-driven bean instance. The container can pool these instances.
- The instance variables of the message-driven bean can contain some state across the handling of client messages- for example, a JMS API connection, an open database connection, or an object reference to an ejb.

( Message driven bean's life cycle )

## Message Driven bean

- A client can't access directly to a message driven bean
- When a message arrive, the container gives it to a message driven bean
- The bean process the message
- The onMessage method may call helper methods, or it may invoke a session or entity bean to process the information in the message or to store it in a database

## Client access with interfaces

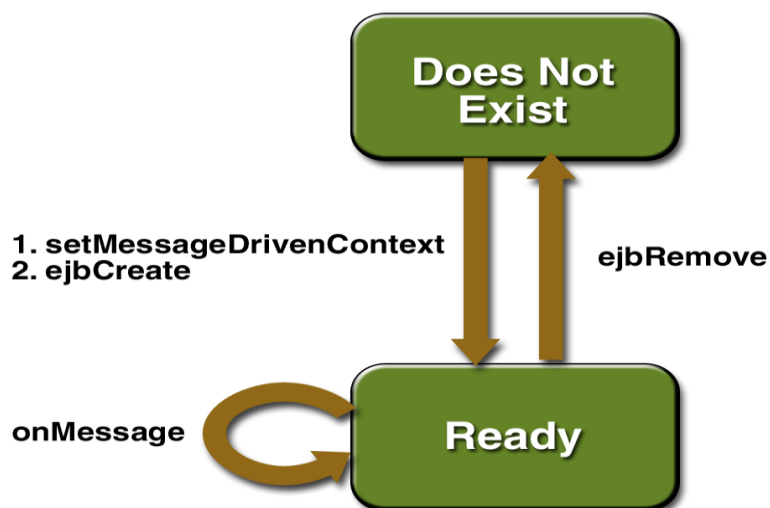
- A client may access a session or an entity bean only through the **methods** defined in the bean's interfaces
- They define the client's view of a bean
- Public business methods declared in Bean interface's can be visible to client, to invoke
- Types of access: Remote access & Local access



## Message driven bean's life cycle

- EJB container creates a pool of message-driven bean instances
- For each instance, the EJB container instantiates the bean :
  - It calls the `setMessageDrivenContext`
  - It calls the instance's `ejbCreate`
- Like a stateless session bean, it's never passivated, It has only two states:
  - Nonexistent
  - Ready to receive messages.
  - is only a bean class – no interfaces
- While in the ready state :
  - EJB container may call `onMessage`
  - EJB container may call the `ejbRemove`

## Message driven bean's life cycle

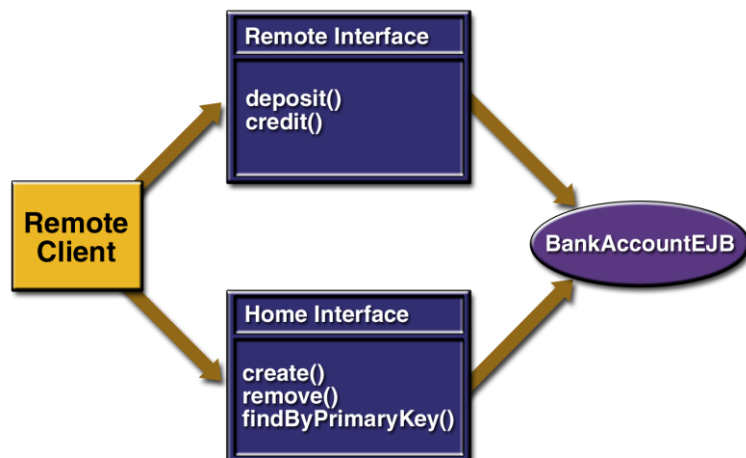


( Back )

## Remote access

- A remote client of an enterprise bean has the following traits:
  - It may run on a different machine and a different Java virtual machine than the enterprise bean it accesses (It is not required to run on a different JVM )
  - It can be a Web component
  - It can be another enterprise bean
  - It can be RMI object
- To create an enterprise bean with remote access, you must :
  - Code a remote interface
    - Business methods
  - Code a home interface
    - Finder methods
    - Home methods
    - Utility methods (to get home)

## Remote access example



## Local access

- A local client has these characteristics
  - It must run in the same JVM as the enterprise bean it accesses
  - It may be a Web component or another enterprise bean
  - To the local client, the location of the enterprise bean it accesses is not transparent
  - It is often an entity bean that has a container-managed relationship with another entity bean
- To create an enterprise bean with local access, you must :
  - Code the local interface
    - Bean's business methods
  - Code the local home interface
    - Life cycle
    - Finder methods
    - Utility methods

## Local interfaces

- If an entity bean is the target of a container managed relationship it **MUST** have local interfaces
- An EJB can use local client view only if it is really guaranteed that other enterprise beans or clients will only address the bean within a single JVM

## Using the Timer Service

- Applications that model business work flows often rely on timed notifications.
- The timer service of the enterprise bean container enables to schedule timed notifications for all types of enterprise beans **except for stateful session beans**.
- For eg, schedule a timed notification to occur according to a calendar schedule, at a specific time, after a duration of time, or at timed intervals.
- Enterprise bean timers are either programmatic timers or automatic timers.
  - **Programmatic timers** => set by explicitly calling one of the timer creation methods of the TimerService interface.
  - **Automatic timers** => created upon the successful deployment of an enterprise bean that contains a method annotated with the java.ejb.Schedule or java.ejb.Schedules annotations.

## Timer Service Using EJB

- The following timeout method uses **@Schedule** to set a timer that will expire every Sunday at 12:00 a.m.

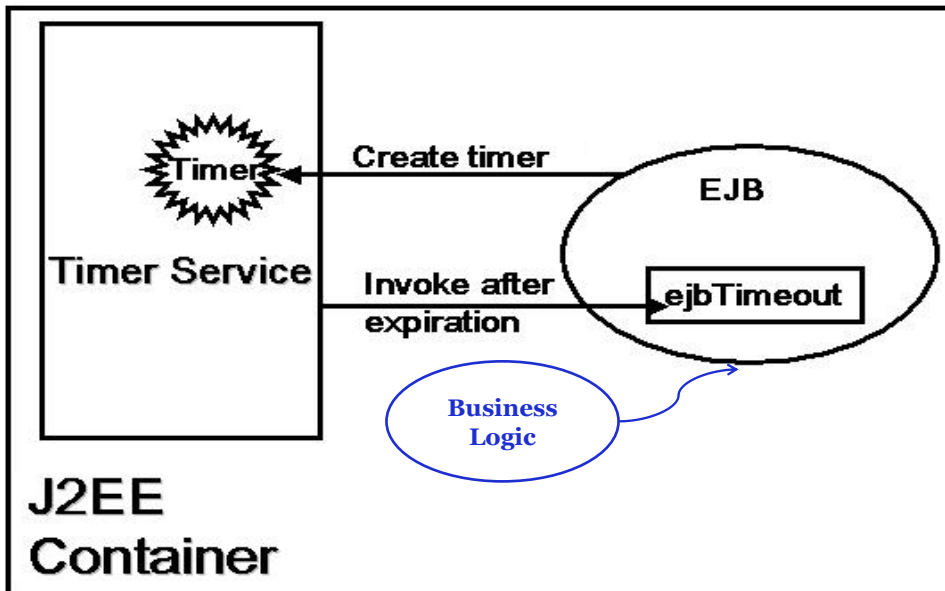
**@Stateless**

```
public class TimerBean
{
```

```
@Schedule(dayOfWeek="Sun", hour="0")
```

```
public void emitNotification()
{
    .....
}
}
```

## EJB and the timer service



CONCLUSION