

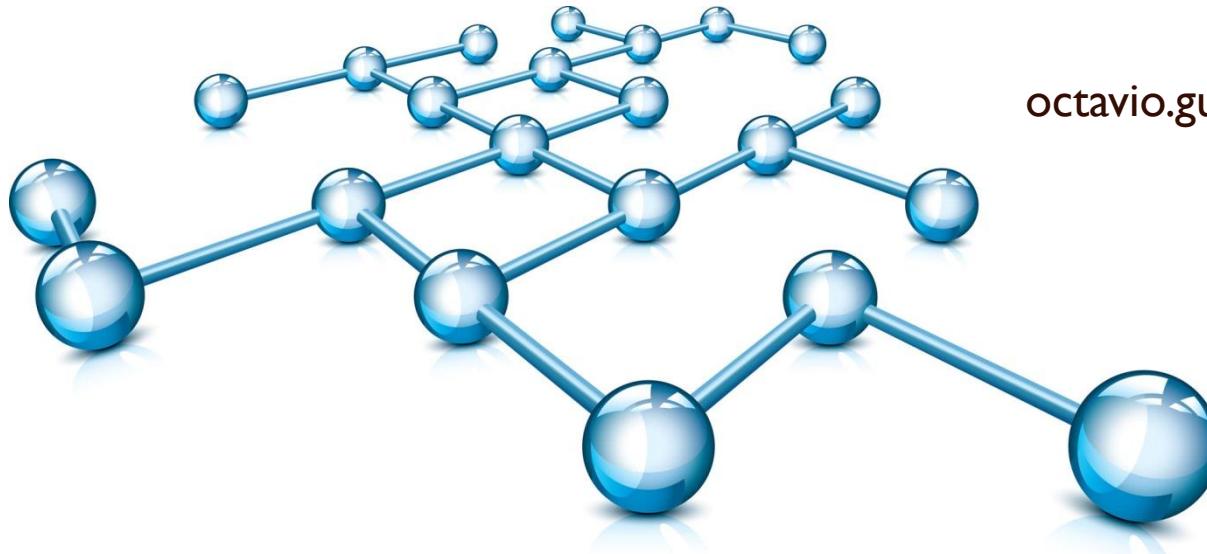


# Sistemas de objetos y componentes distribuidos



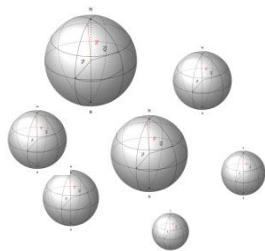
Profesor:  
Dr. J. Octavio Gutiérrez García

[octavio.gutierrez@itam.mx](mailto:octavio.gutierrez@itam.mx)



# Sistemas de objetos distribuidos y componentes

- La tarea de un **middleware** es proveer un alto nivel de abstracción en la programación de sistemas distribuidos para soportar heterogeneidad, interoperabilidad y portabilidad.



- Middleware de **objetos distribuidos**



- Middleware **basado en componentes**

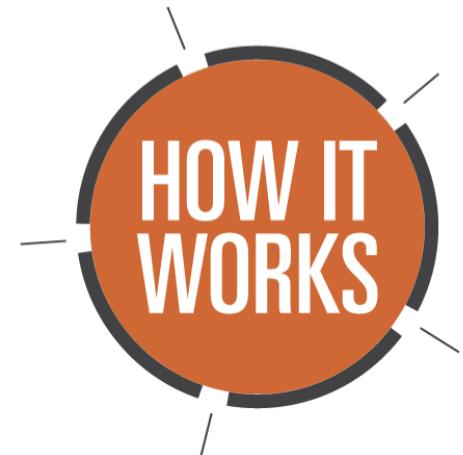
# Middleware de objetos distribuidos

- Adopción de paradigma **orientado a objetos**.
- Objetos se comunican vía
  - RMI
  - Eventos
- Ejemplos: Java RMI y CORBA



# Middleware de objetos distribuidos

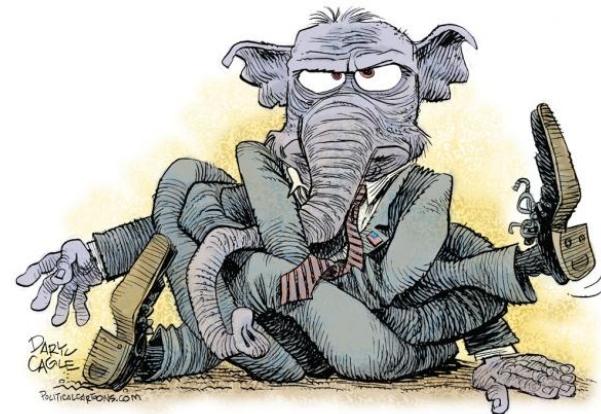
- Referencias a objetos **remotos**
- Interfaces **remotas**
- Acciones **distribuidas** (RMI)
- Excepciones **distribuidas**, ej. pérdida de mensajes.
- Recolección **distribuida** de basura



# Middleware de objetos distribuidos

## *Complejidad*

- Comunicación inter-objetos.
- Gestión distribuida del ciclo de vida de los objetos.
- Activación y desactivación de objetos.
  - Persistencia del estado de los objetos
- Más servicios:
  - Servicios de nombres, seguridad, etc.



# Middlewares basados en componentes

Desarrollados para superar **limitantes** de los middlewares de **objetos** distribuidos:



- **Dependencias implícitas**
- **Complejidad** de programación
- **Falta de separación** de aspectos de distribución: seguridad, manejo de fallas, concurrencia, etc.
- **Sin soporte para el despliegue.**

¿Con qué objetos se comunican?

Los programadores son expuestos a cuestiones del middleware

# Middlewares basados en componentes

Un **componente** de software es una unidad de composición con **interfaces** especificadas contractualmente y dependencias explícitas.

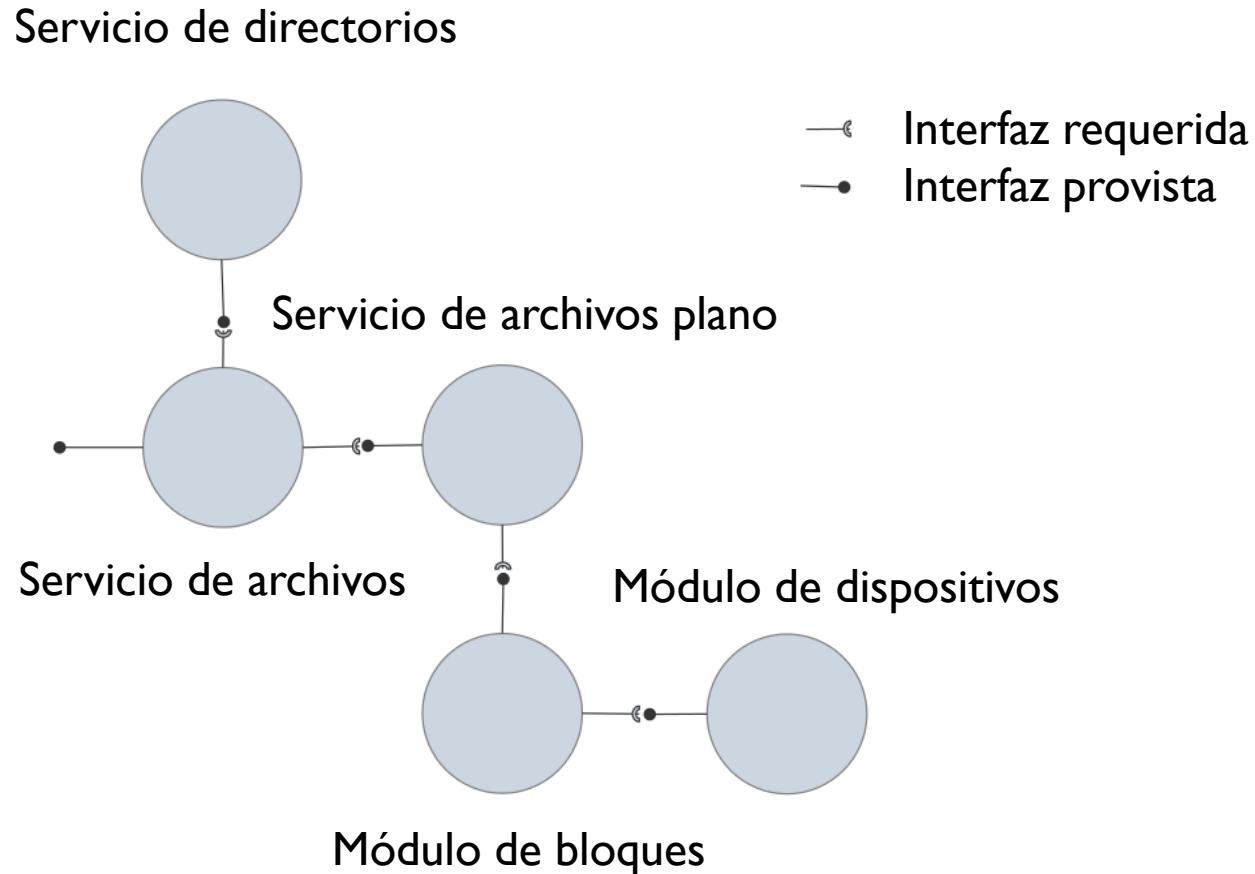


Un componente se especifica en términos de un **contrato** que incluye:

- Interfaces **provistas**
- Interfaces **requeridas**



# Middlewares basados en componentes



# Middlewares basados en componentes



- Los **contenedores** proveen un patrón común, que consiste en:
  - Un **front-end**.
  - Un **recipiente** que contiene uno o más componentes que implementan la aplicación o la lógica de negocios.
  - **Servicios** del sistema que administran los datos asociados en **almacenamiento persistente**.

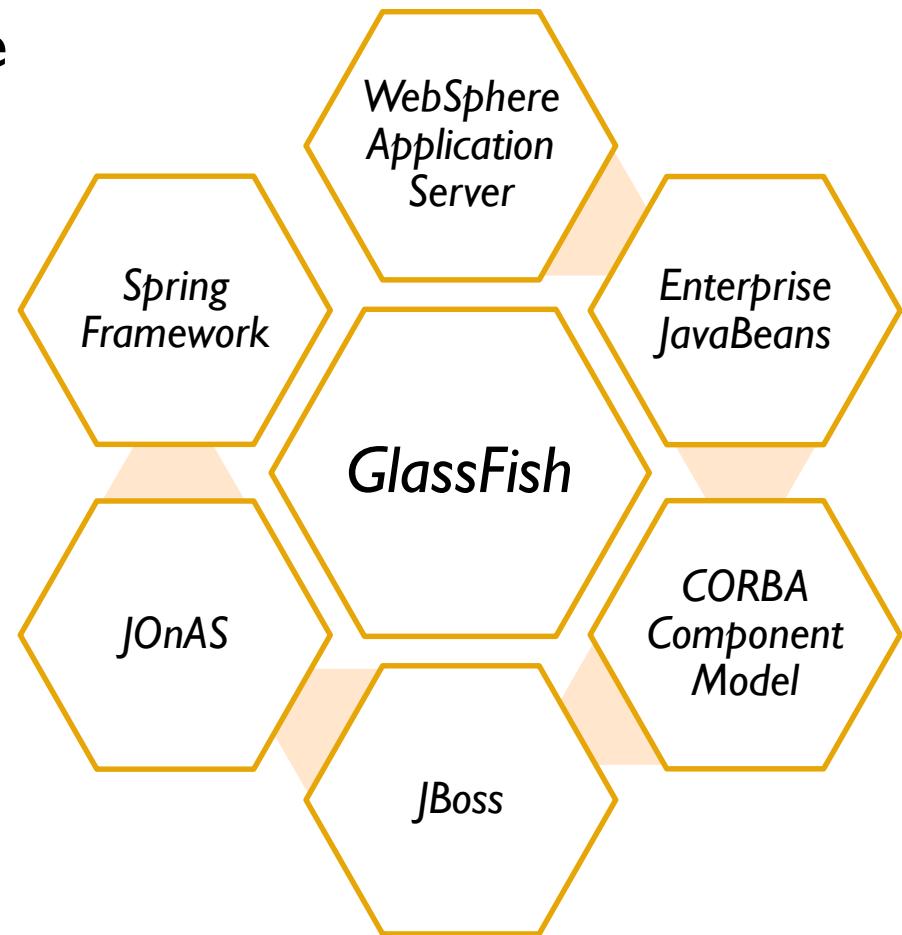
# Middlewares basados en componentes



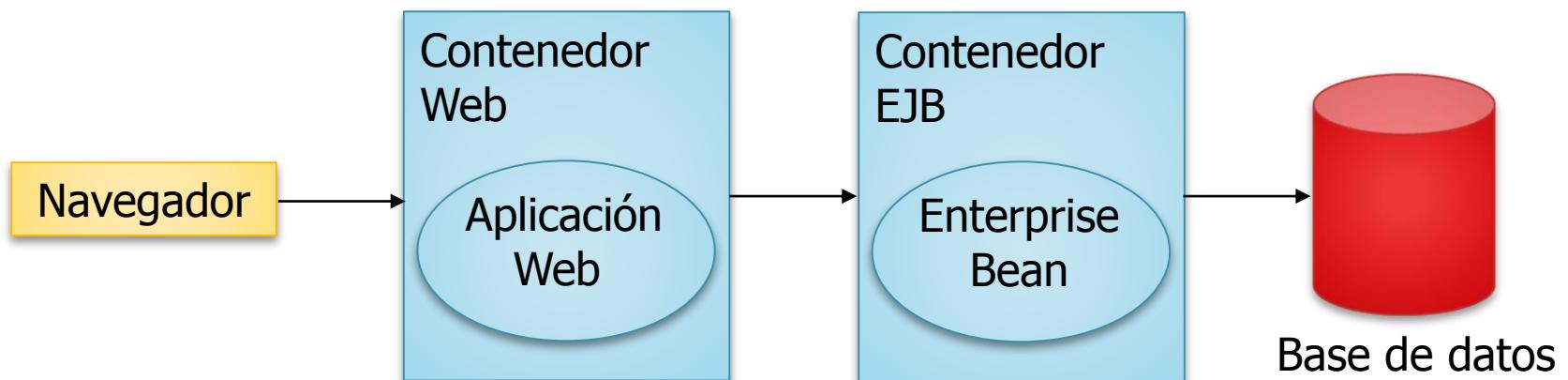
Los contenedores **separan la funcionalidad** de la aplicación (que implementan los componentes) de las cuestiones **del middleware** distribuido.

# Middlewares basado en componentes

- Los middlewares que soportan el uso de componentes se conocen como **servidores de aplicaciones**



# Enterprise Java Beans (EJB)



Arquitectura de 4 niveles

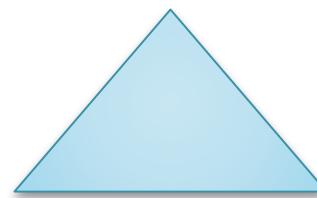
# Enterprise Java Beans

Modelo de programación, constituido por convenciones/ protocolos y un conjunto de clases e interfaces sobre las cuales se apoya el EJB API.

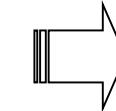
Portabilidad

Convenciones  
Protocolos  
Interfaces

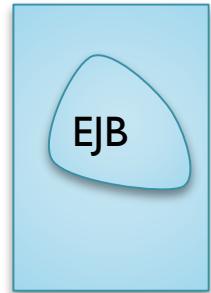
Especificaciones



Objetos distribuidos y componentes del lado del servidor



EJB Container



# Enterprise Java Beans



- Tecnología para el desarrollo de **componentes** en la parte del servidor con la plataforma J2EE (**Java EE**)
- Los **contenedores** de EJBs **soportan**:
  - Transacciones
  - Seguridad
  - Persistencia



El objetivo de EJBs es mantener una fuerte separación de las preocupaciones entre los diversos roles implicados en el desarrollo de aplicaciones distribuidas.

# ¿Cuándo usar EJBs?

IF NOT  
NOW?  
**WHEN?**



- No siempre, para simples aplicaciones web con servlets y JSPs es más que suficiente.
- Solución para alta disponibilidad y escalabilidad.
- Cuando el sistema debe manejar transacciones distribuidas
- Cuando hay múltiples clientes heterogéneos

# Categorías de EJBs



- **Session Beans:** Un bean de sesión es un componente que **implementa una tarea específica** dentro de la lógica de la aplicación de un servicio.
- Son usados para modelar **procesos de negocio** que son accedidos de manera **síncrona**.
- **No** son persistentes

# Categorías de EJBs



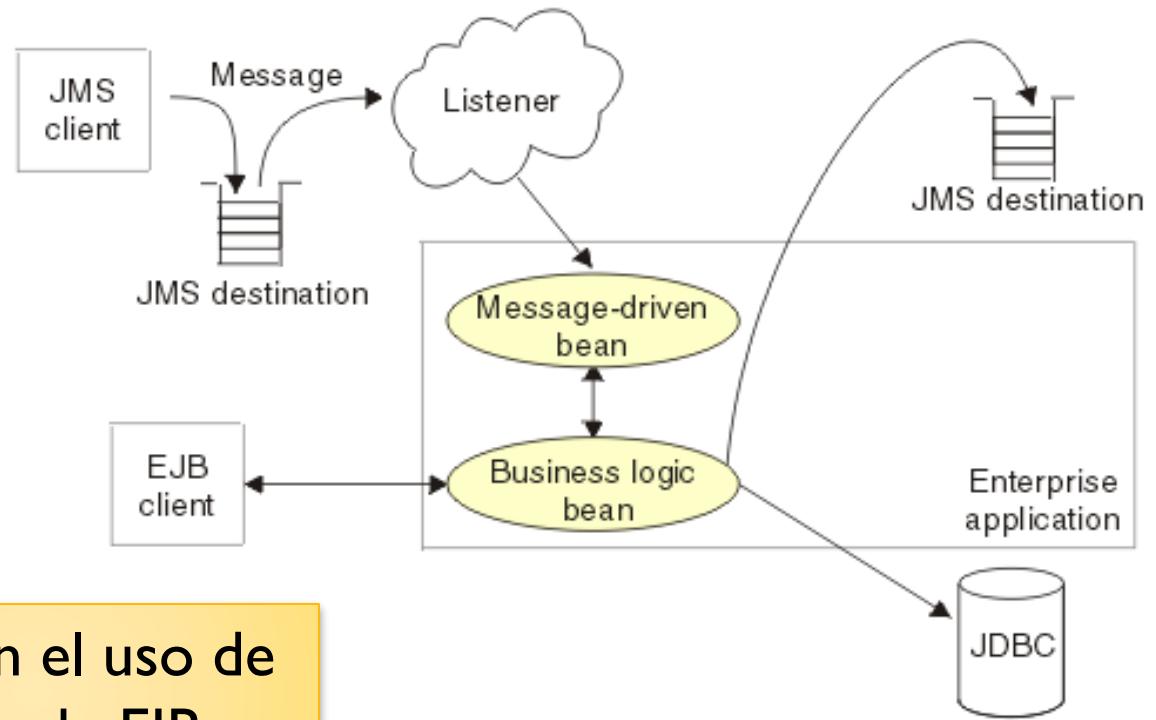
- **Session Beans**

- **Stateful:** Mantener el estado de una conversación con un solo cliente. El bean puede manejar el flujo de trabajo de varios beans.
- **Stateless:** Accedidos concurrentemente por muchos clientes pero sin estado

# Categorías de EJBs



- **Message-driven Beans:** Diseñados para modelar procesos de negocio que son accedidos de manera **asíncrona** a través de mensajes.



# Categorías de EJBs



- **Singleton Session Beans:** Un bean de sesión singleton se instancia **una vez por cada aplicación** y existe para el ciclo de vida de la aplicación.
- Los Beans de sesión Singleton se diseñaron para circunstancias en las que una **instancia de bean enterprise es compartida entre y accedida concurrentemente por múltiples clientes**

# Desventajas de EJBs



- **Tiempo** de desarrollo
- Conocimiento **avanzado** de JAVA
  - EJBs
  - RMI
  - JNDI
  - JDBC

# Session Bean



Paso I. Crear una "Enterprise Application"

New Project

Steps

1. Choose Project  
2. ...

Choose Project

Filter:

Categories:

- Java
- JavaFX
- Java Web
- Java EE

Projects:

- Enterprise Application
- Enterprise Application with Exist
- EJB Module
- Web Application

Description:

Creates a new enterprise application in a standard project. You can also create an EJB module project and Web application project in

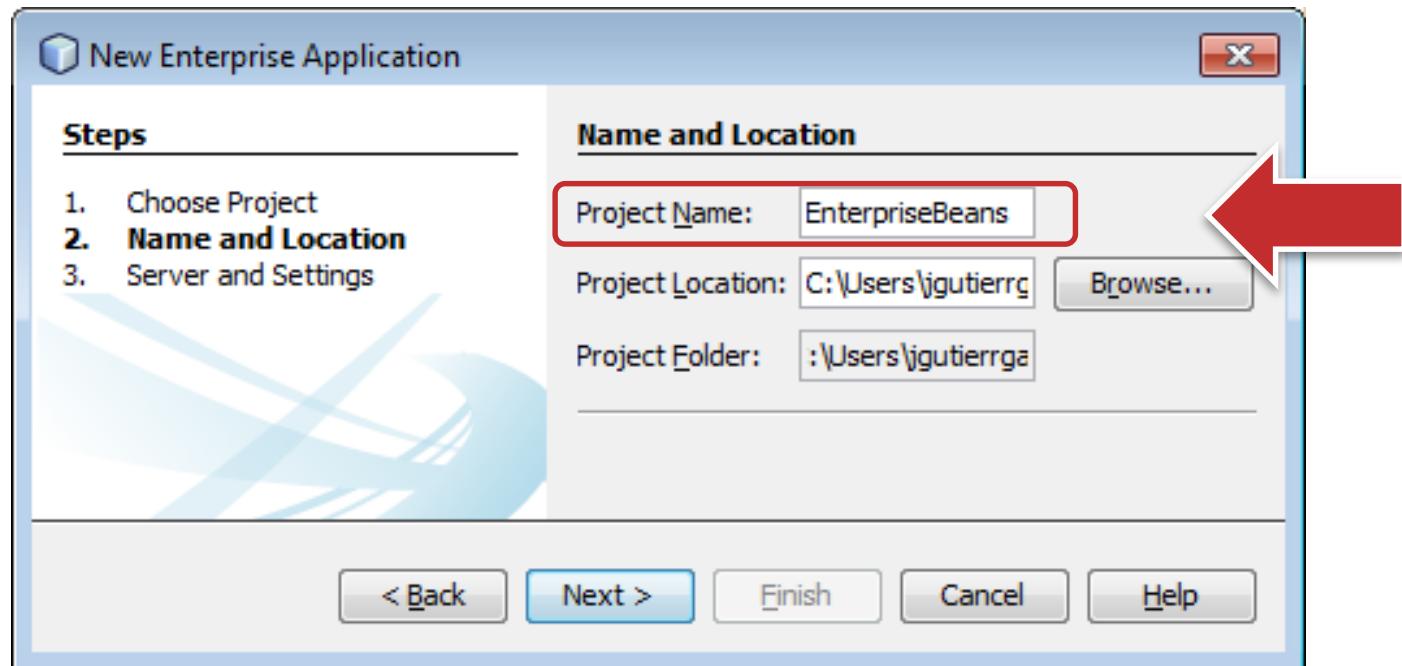
< Back Next > Finish Cancel Help

A screenshot of a 'New Project' dialog box. The title bar says 'New Project'. On the left, under 'Steps', it says '1. Choose Project' and '2. ...'. The main area is titled 'Choose Project' with a 'Filter:' input field. It has two sections: 'Categories:' and 'Projects:'. Under 'Categories:', there are four items: Java, JavaFX, Java Web, and Java EE. Under 'Projects:', there are four items: Enterprise Application (which is highlighted with a blue selection bar and a red arrow pointing to it), Enterprise Application with Exist, EJB Module, and Web Application. Below these sections is a 'Description:' box containing text about creating a new enterprise application. At the bottom are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

# Session Bean



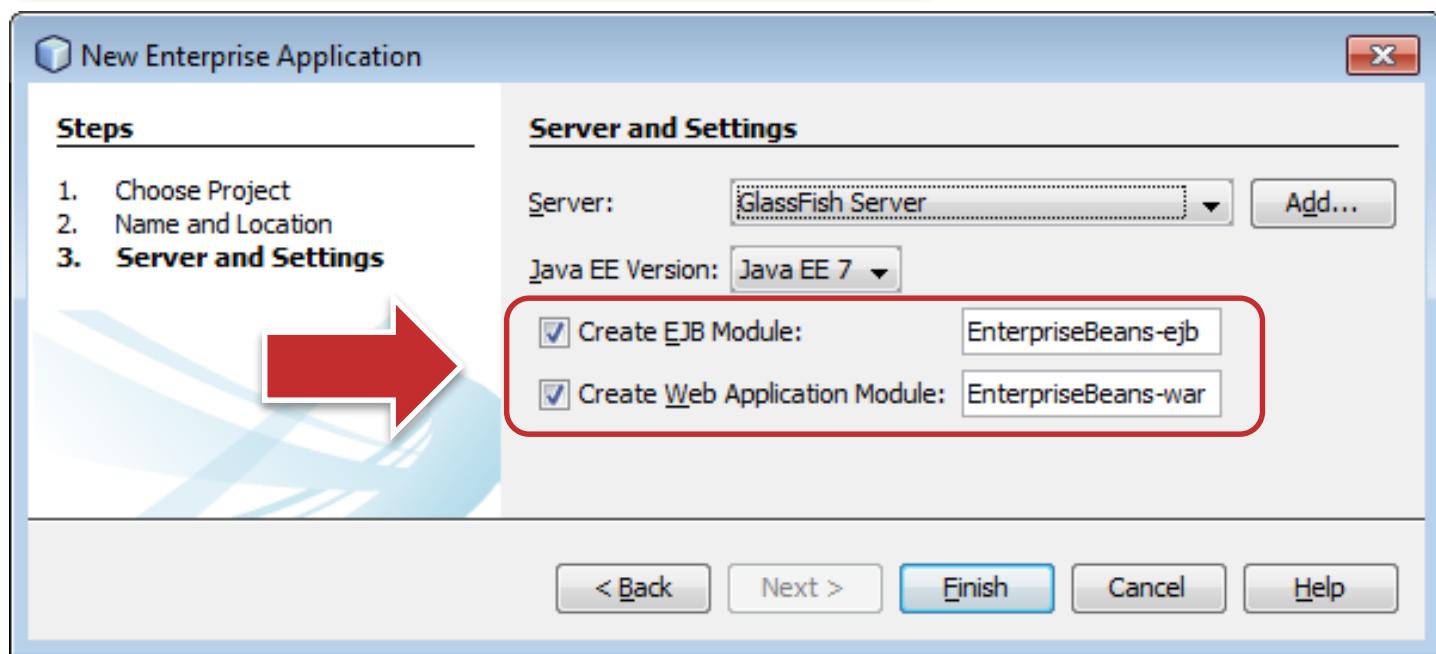
Paso I. Crear una "Enterprise Application"



# Session Bean



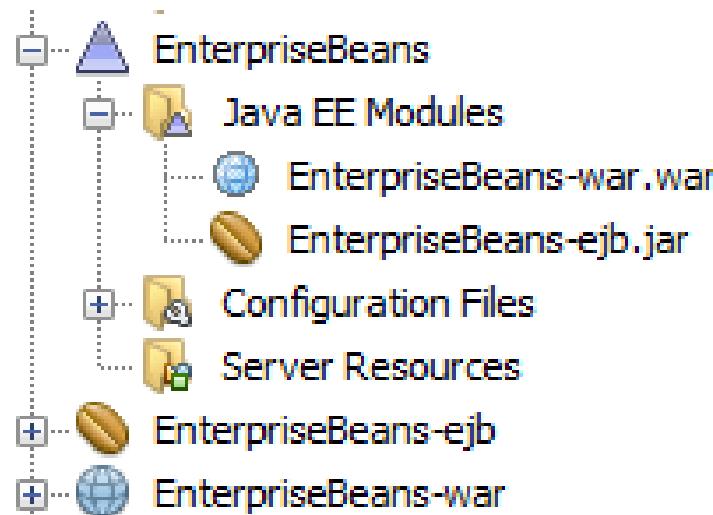
## Paso 1. Crear una "Enterprise Application"



# Session Bean



3 proyectos



# Session Bean



Paso 2. Crear el “JavaBean”

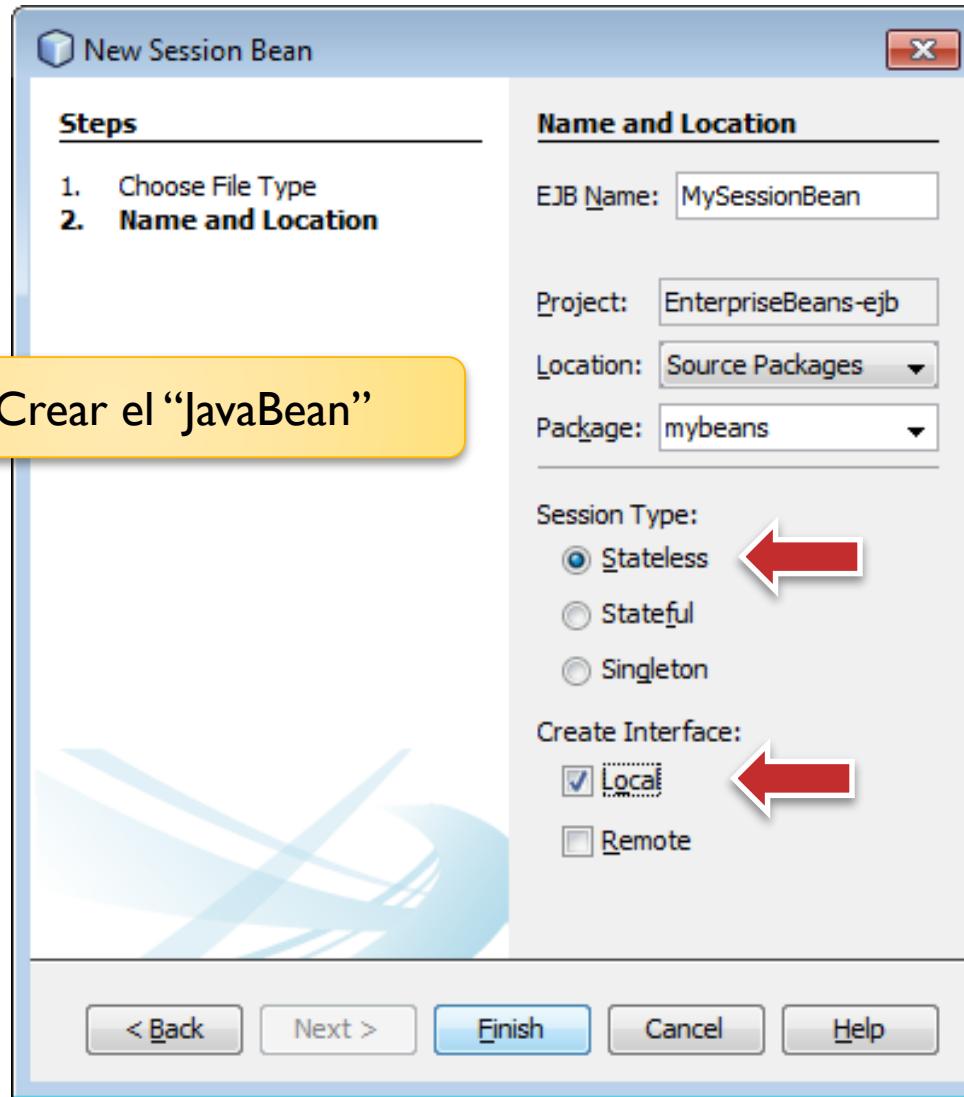
The screenshot shows a file structure for an Enterprise Beans project. The 'EnterpriseBeans-ejb' folder is selected, revealing its contents: Java EE Modules (EnterpriseBeans-war.war, EnterpriseBeans-ejb.jar), Configuration Files, and Server Resources. A context menu is open over the 'EnterpriseBeans-ejb' folder, listing options like New, Build, Clean and Build, Clean, Verify, Generate Javadoc, Run, Deploy, Debug, Profile, Test (Alt+F6), Set as Main Project, Open Required Projects, and Close. A red arrow points from the text 'Paso 2. Crear el “JavaBean”' to the 'Session Bean...' option in the 'New' submenu, which is highlighted with a blue border.

- New
  - Folder...
  - Message-Driven Bean...
  - Session Bean...**
  - Empty File...
  - Web Service Client...
  - Java Class...
  - Web Service...
  - XML Document...
  - Java Package...
  - Java Interface...
  - Entity Class...
  - Entity Classes from Database...
  - Timer Session Bean...
  - Other...

# Session Bean



Paso 2. Crear el “JavaBean”



# Session Bean



### Paso 3. Especificar la “Lógica de Negocios”

```
@Stateless  
public class MySessionBean implements MySessionBeanLocal {  
  
    }  
  
    Generate  
        Add Business Method...  
        Constructor...  
        Logger...  
        toString()...  
        Override Method...  
        Add Property...  
        Call Enterprise Bean...  
        Use Database...  
        Send JMS Message...  
        Send E-mail...  
        Call Web Service Operation...  
        Generate REST Client...  
  
    }  
  
    ic below. (Right-click in editor and choose  
    d "Add Business Method")  
}
```

# Session Bean



Paso 3. Especificar la “Lógica de Negocios”

→

→

Add Business Method...

Name:  Return Type:

Name	Type	Final
a	int	<input type="checkbox"/>
b	int	<input type="checkbox"/>

Use in Interface:  Local  Remote  Both

# Session Bean



Paso 3. Especificar la “Lógica de Negocios”

```
@Stateless  
public class MySessionBean implements MySessionBeanLocal {  
  
    @Override  
    public int add(int a, int b) {  
        return a+b;  
    }  
}
```



# Session Bean



- Paso 4. 1. Clean & Build
- Paso 4. 2. Deploy

```
@Stateless  
public class MySessionBean implements MySessionBeanLocal {  
  
    @Override  
    public int add(int a, int b) {  
        return a+b;  
    }  
}
```

# Session Bean

Paso 5. Crear Servlet para invocar al JavaBean



The screenshot shows a file structure for a web application named "EnterpriseBeans-war". The "Source Packages" folder contains a default package named "<default package>". A context menu is open over this package, with the "New" option selected. The "Servlet..." option is highlighted with a blue selection bar. Other options in the menu include Folder..., Session Bean..., Message-Driven Bean..., Empty File..., Web Service Client..., JSP..., RESTful Web Services from Patterns..., Java Class..., Web Service..., RESTful Java Client..., RESTful JavaScript Client..., XML Document..., Java Package..., Java Interface..., JavaScript File..., and Other... .

EnterpriseBeans-war

- Web Pages
- Source Packages
  - <default package>
- Libraries
- Configuration Files

New

- Find... Ctrl+F
- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V
- Delete Suprimir
- Rename...
- Compile Package F9
- Test Package Ctrl+F6
- History
- Tools

Servlet...

Folder...  
Session Bean...  
Message-Driven Bean...  
Empty File...  
Web Service Client...  
JSP...  
RESTful Web Services from Patterns...  
Java Class...  
Web Service...  
RESTful Java Client...  
RESTful JavaScript Client...  
XML Document...  
Java Package...  
Java Interface...  
JavaScript File...  
Other...

MyServlet

# Session Bean



## Paso 5. Invocar al JavaBean

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        ...
    }
}
```

A red arrow points from the code editor towards a context menu that appears over the closing brace of the try block. The menu contains the following options:

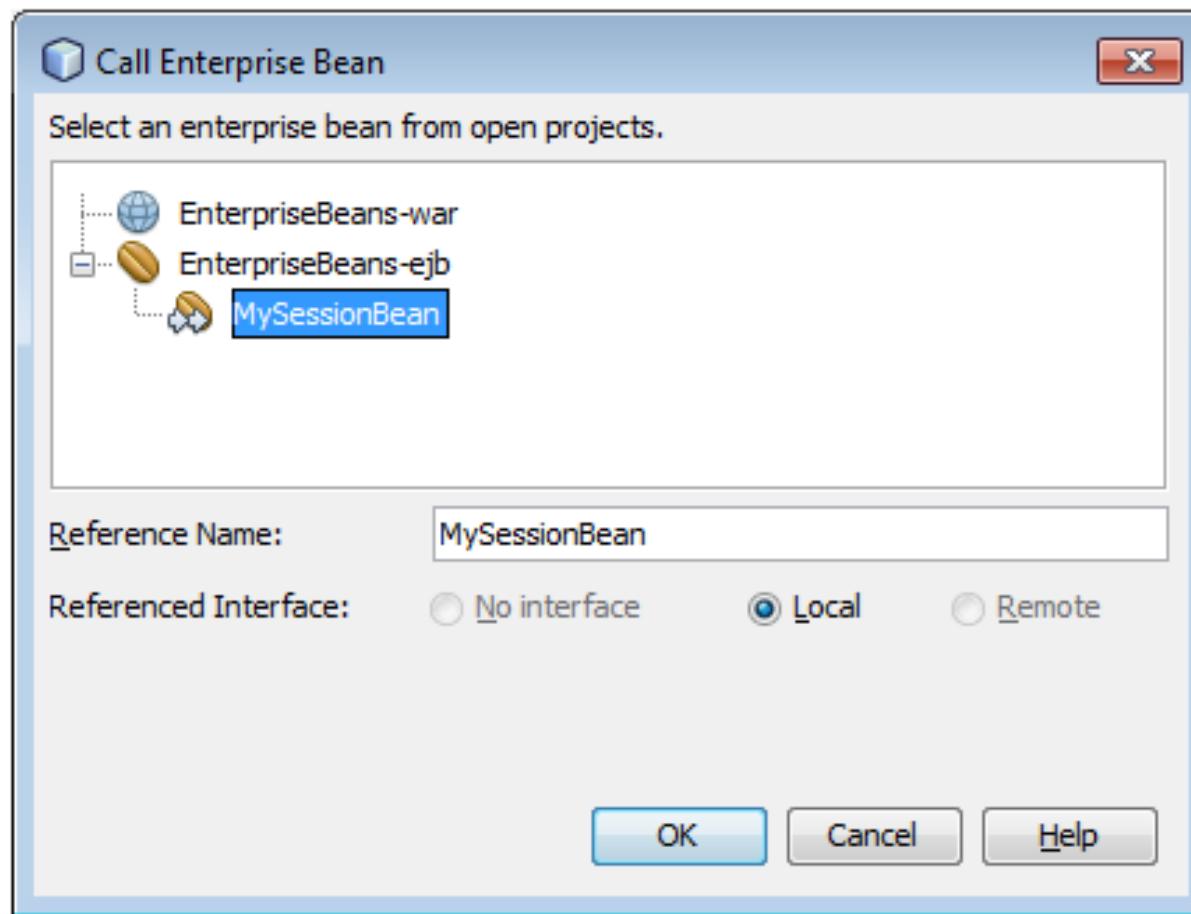
- Generate
- Constructor...
- Logger...
- toString()...
- Override Method...
- Add Property...
- Call Enterprise Bean...**
- Use Database...
- Send JMS Message...
- Send E-mail...
- Call Web Service Operation...
- Generate REST Client...

The code itself shows the generation of an HTML page with a title and a header section.

# Session Bean



## Paso 5. Invocar al JavaBean



# Session Bean



## Paso 5. Invocar al JavaBean

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet MyServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet MyServlet at " + mySessionBean.add(3, 5) + "</h1>"); mySessionBean.add(3, 5)
        out.println("</body>");
        out.println("</html>");
    }
}
```

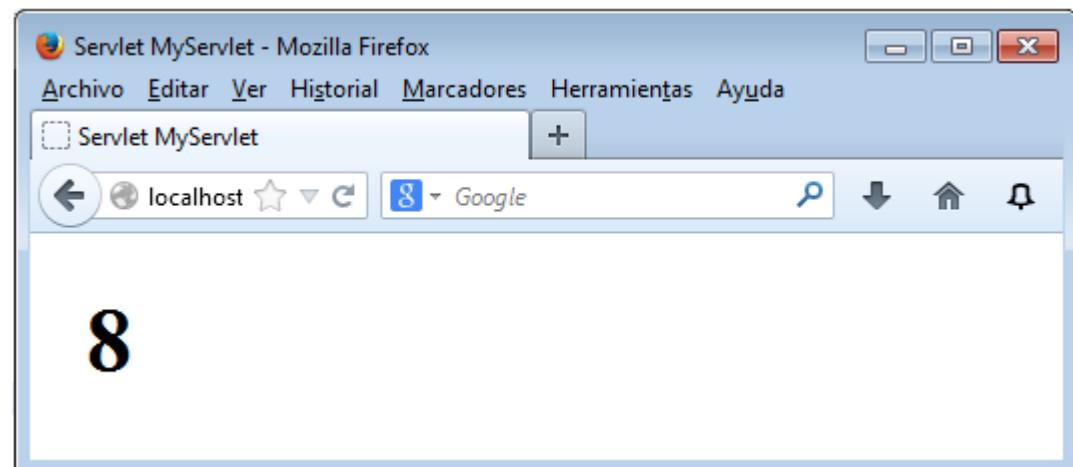
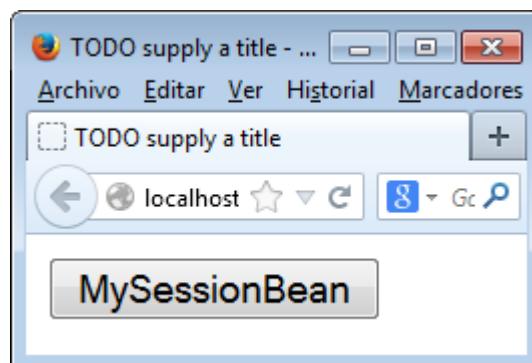


# Session Bean



Paso 6. Incluir forma en index.html para llamar al Servlet

```
<html>
  <head>
    <title>TODO supply a title</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <form action="MyServlet" method="POST">
      <input type="submit" value="MySessionBean" />
    </form>
  </body>
</html>
```

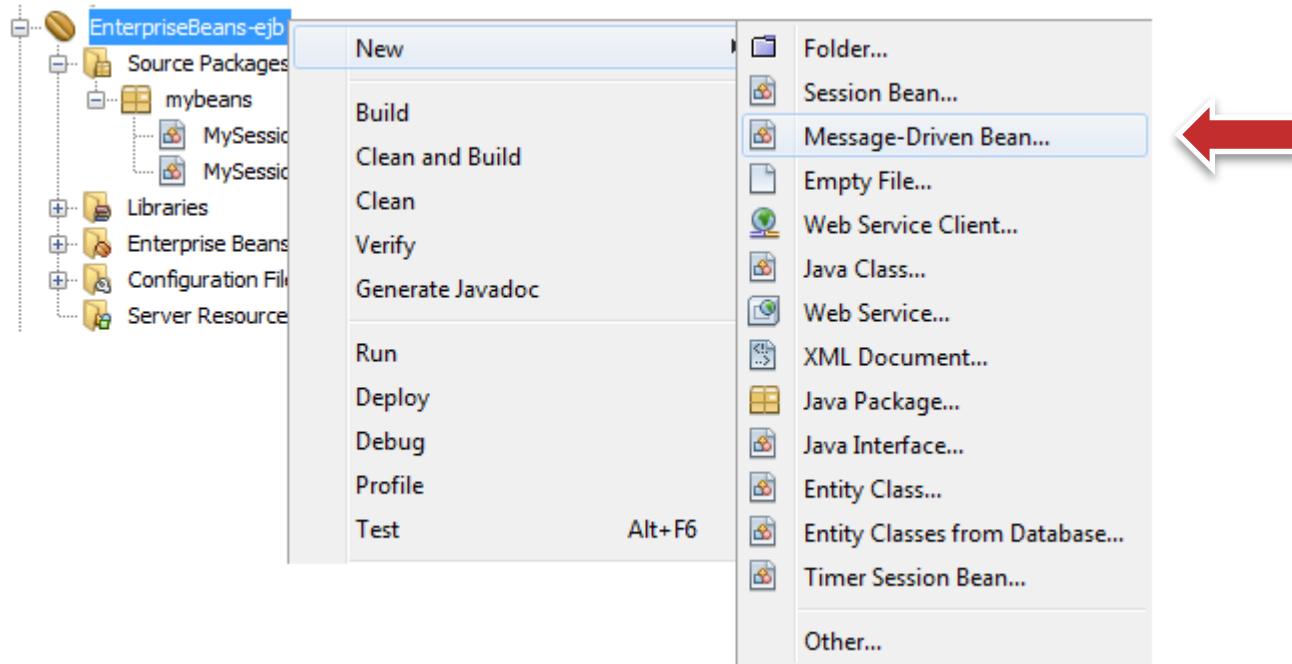




# Message-Driven Bean



## Paso 1. Crear el “JavaBean”





# Message-Driven Bean



## Paso 1. Crear el “JavaBean”

New Message-Driven Bean

**Steps**

1. Choose File Type
2. **Name and Location**
3. Activation Config Properties

**Name and Location**

EJB Name: MyMessageBean

Project: EnterpriseBeans-ejb

Location: Source Packages

Package: mybeans

Project Destinations:

Server Destinations: jms/GlassFishTestQueue

< Back  Finish Cancel Help

The screenshot shows the 'New Message-Driven Bean' wizard. The 'Name and Location' step is active. The EJB Name is 'MyMessageBean'. The Project is 'EnterpriseBeans-ejb', Location is 'Source Packages', and Package is 'mybeans'. The 'Server Destinations' dropdown is set to 'jms/GlassFishTestQueue'. Three red arrows point from the right towards the 'Server Destinations' field.



# Message-Driven Bean



## Paso 2. Definir el manejo de los mensajes entrantes

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
    @ActivationConfigProperty(propertyName = "destinationLookup", propertyValue = "jms/GlassFishTestQueue")
})
public class MyMessageBean implements MessageListener {

    public MyMessageBean() {
    }

    @Override
    public void onMessage(Message message) {
        try {
            System.out.println("A message: "+((TextMessage)message).getText());
        } catch (JMSException ex) {
            Logger.getLogger(MyMessageBean.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```



La línea de código que imprime el contenido del mensaje es la que se ha resaltado con un cuadro amarillo.



# Message-Driven Bean



Paso 3. Desplegar el JavaBean

Paso 4. Probar el JavaBean ejecutando un MessageProducer(Queue)



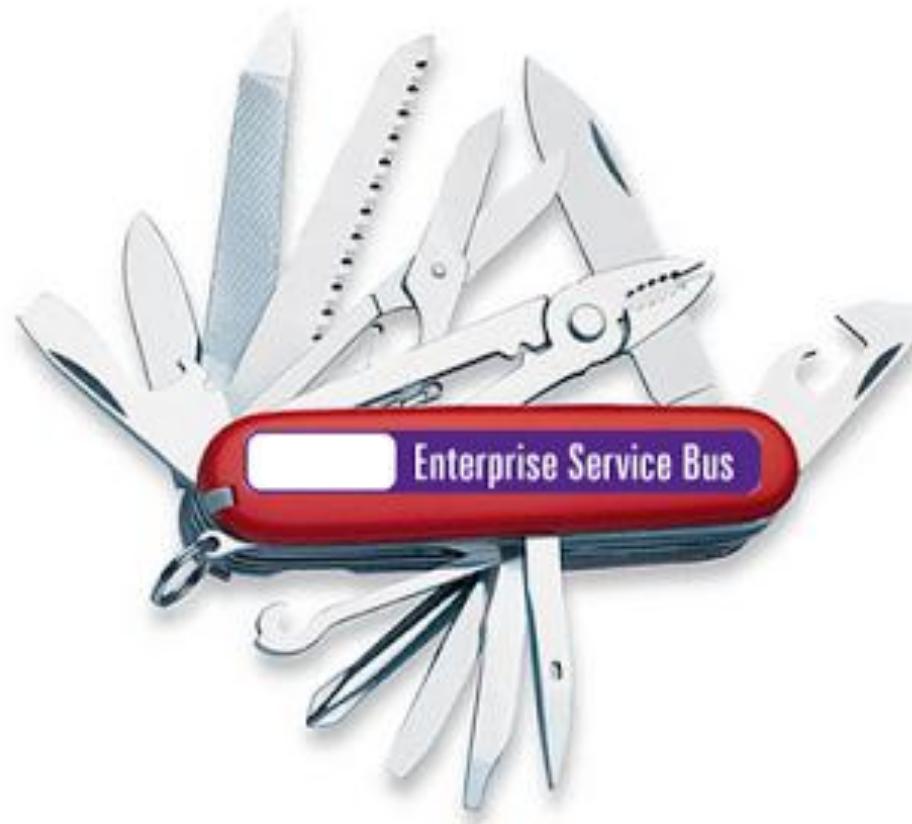
```
Output ✘
Java DB Database Process ✘ GlassFish Server ✘ JMSqueue (run-single) ✘
Información: ACDEPL104: Java Web Start services stopped for t...
Información: visiting unvisited references
Información: visiting unvisited references
Información: ACDEPL103: Java Web Start services started for th...
JMSqueue was successfully deployed in 270 millise...
A message: Good bye!
A message: Do you copy?
A message: Testing, 1, 2, 3. Can you hear me?
```



GlassFishServer 4.0  
EnterpriseBeans  
EnterpriseBeans-ejb  
EnterpriseBeans-war  
JSMQueue

# ESB

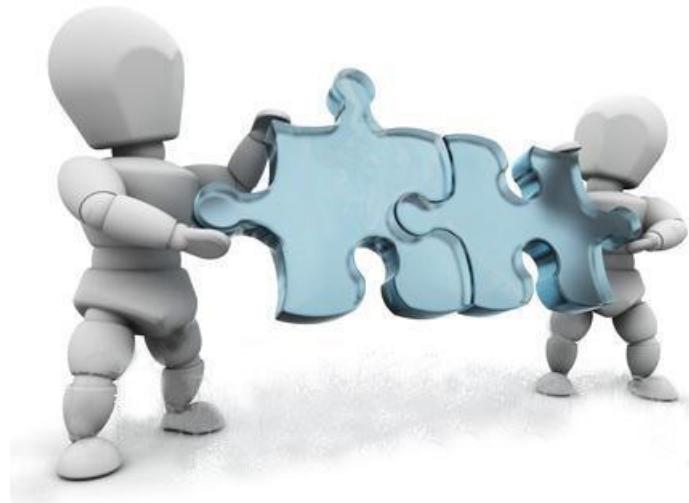
## Enterprise Service Bus



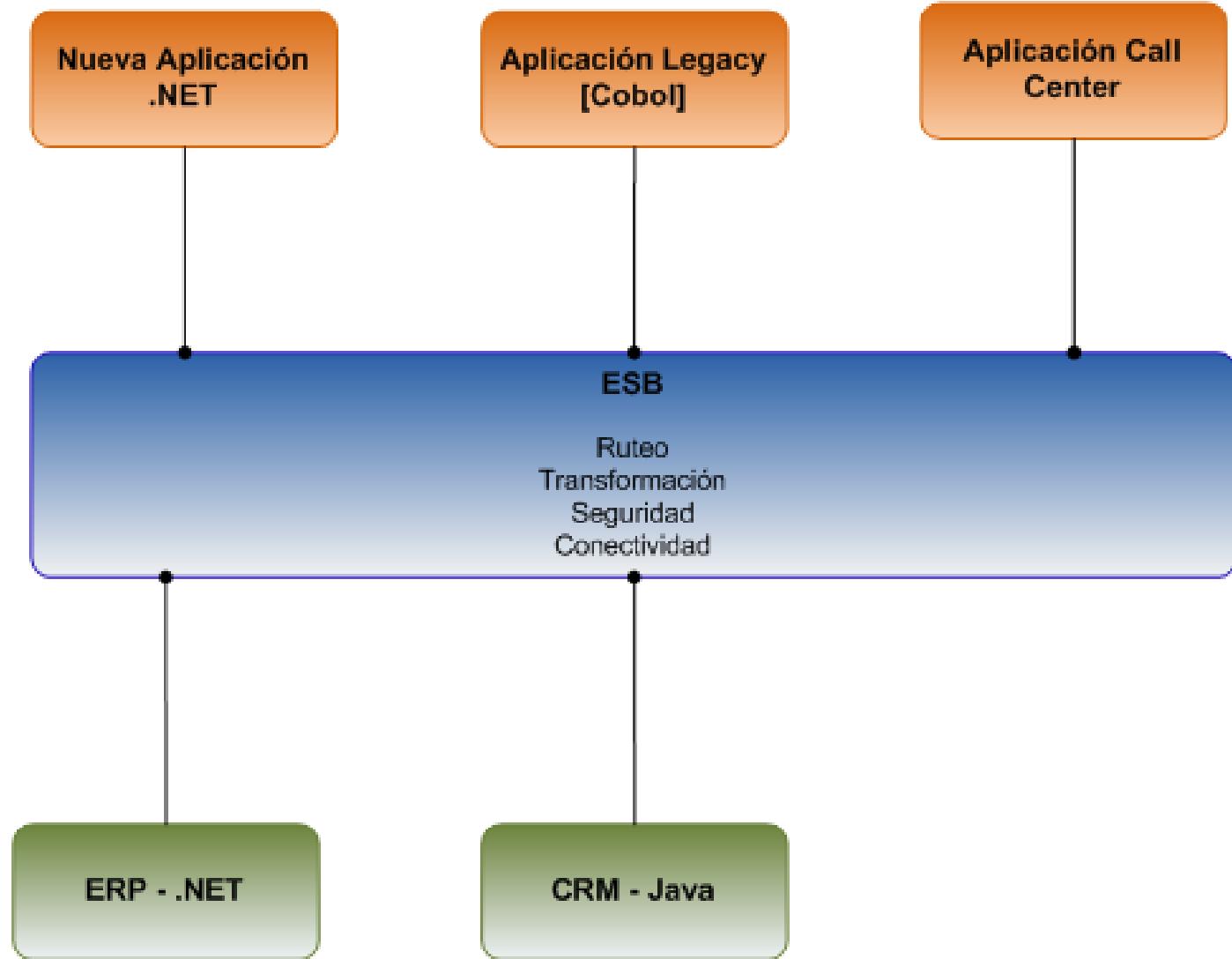
# Enterprise Service Bus

Enterprise Service Bus (ESB) es una **infraestructura** de **conectividad** flexible para **integrar** aplicaciones y servicios.

Un ESB proporciona la conectividad para implementar una arquitectura orientada a servicios (**SOA**), lo que reduce la complejidad de la integración de aplicaciones y servicios.

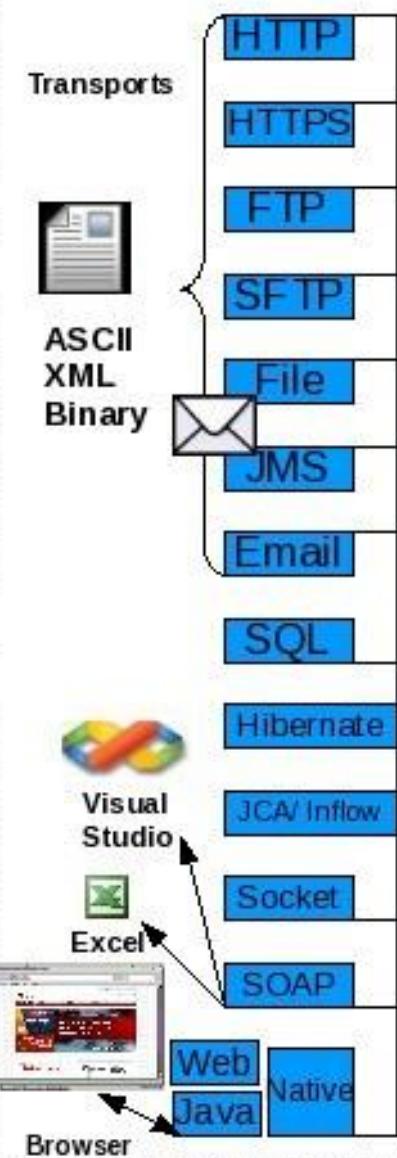


# Enterprise Service Bus



## Event Listeners and Actions

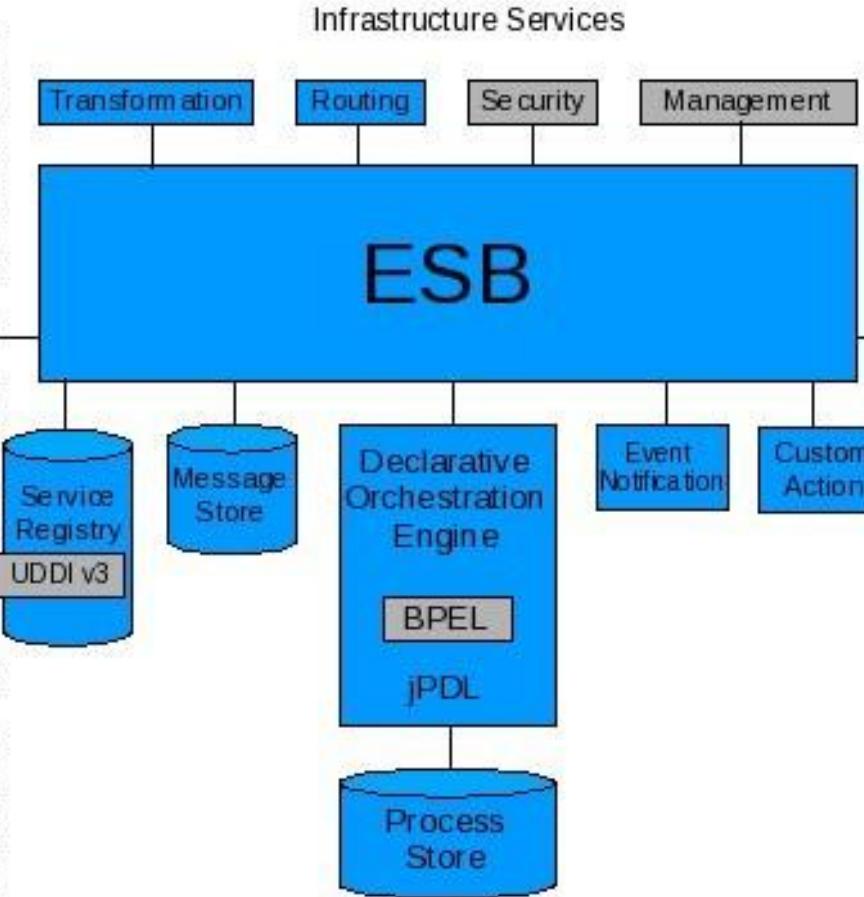
Provide Transport Mediation



Now Future Partners

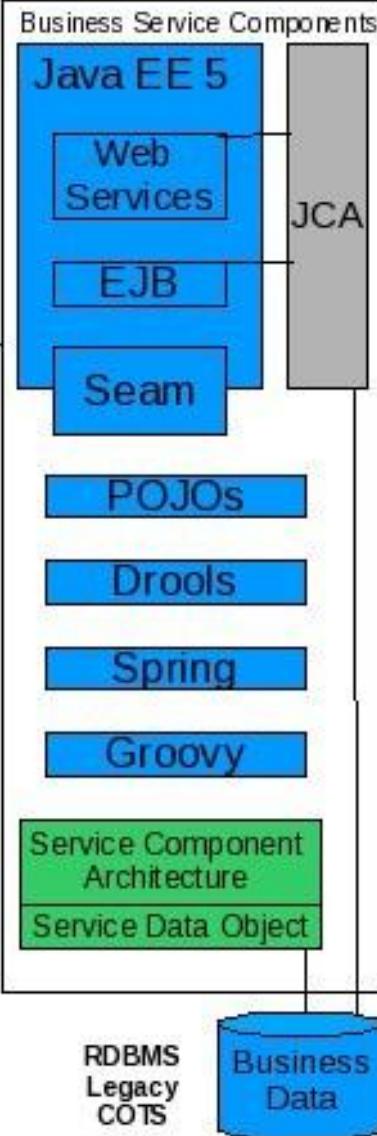
## Pluggable Architecture For Integrating Infrastructure Services

Infrastructure Services



## Business Services

Runs Within a Container or Standalone



# ¿Cuándo usar JBOSS – ESB?

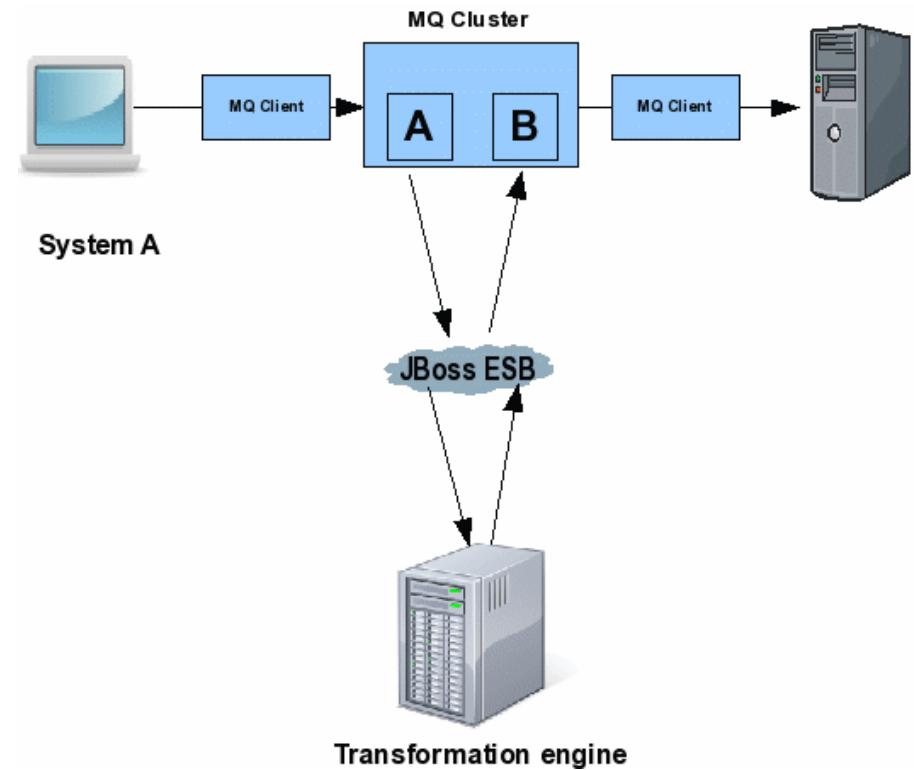
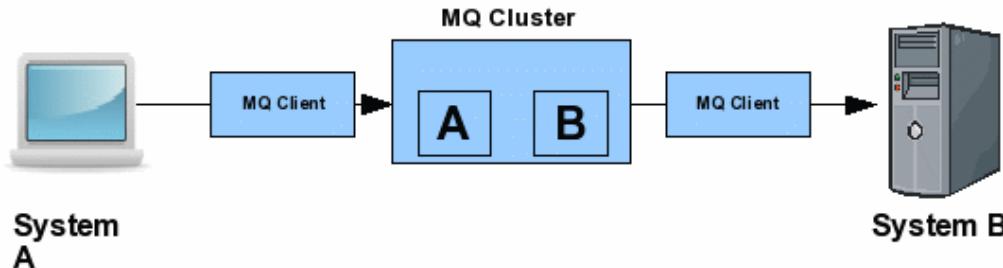
Legacy  
Software



Cutting edge  
Software



# ¿Cuándo usar JBOSS – ESB?



Transformation engine

# ¿Cuándo usar JBOSS – ESB?

