|  | | Escuela Politécnica Superior  Ingeniería Informática  Prácticas de Sistemas Informáticos 2 | | | |
| --- | --- | --- | --- | --- | --- |
| **Grupo** | **2311** | **Práctica** | 1B | **Fecha** | 05/03/2022 |
| **Alumno/a** | | Cerrato, Sánchez, Daniel | | | |
| **Alumno/a** | | Garitagoitia, Romero, David T. | | | |

# Práctica 1B: Arquitectura de Java EE

## Ejercicio número 1:

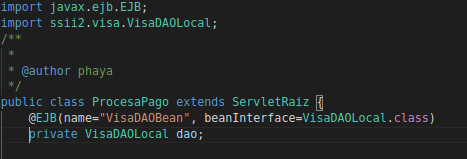
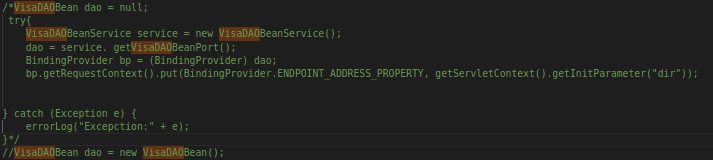
**Introduzca las siguientes modificaciones en el bean VisaDAOBean para convertirlo en un EJB de sesión stateless con interfaz local: Hacer que la clase implemente la interfaz local y convertirla en un EJB stateless mediante la anotación Stateless ... import javax.ejb.Stateless; ... @Stateless(mappedName="VisaDAOBean") public class VisaDAOBean extends DBTester implements VisaDAOLocal { ... Eliminar el constructor por defecto de la clase. Ajustar el método getPagos() a la interfaz definida en VisaDAOLocal Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas.**

Como se nos indica, el primer paso será modificar los import para incluir el javax.ejb.Stateless, tras hacer esto se procede a eliminar el constructor de la clase y toca modificar el método getPagos() para que se adapte a la interfaz (el retorno pasa de ser un ArrayList a un PagoBean[]).

Para ello simplemente cambiamos el retorno y añadimos estas dos líneas de código:



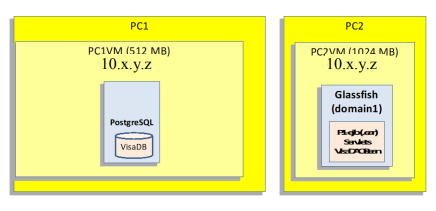
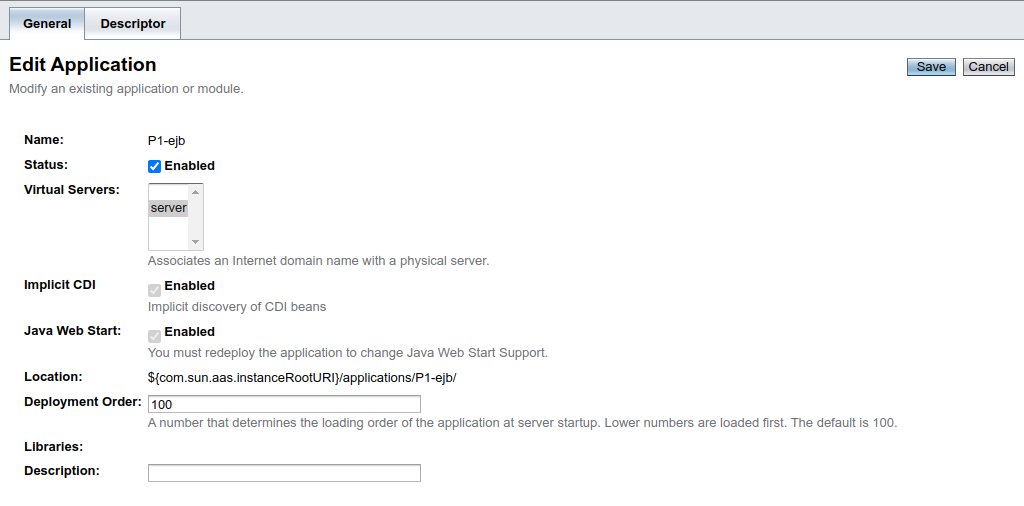
## Ejercicio número 2:

* 1. **Modificar el servlet ProcesaPago para que acceda al EJB local. Para ello, modificar el archivo ProcesaPago.java de la siguiente manera: En la sección de preproceso, añadir las siguientes importaciones de clases que se van a utilizar: ... import javax.ejb.EJB; import ssii2.visa.VisaDAOLocal; ... Se deberán eliminar estas otras importaciones que dejan de existir en el proyecto, como por ejemplo: import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente import ssii2.visa.VisaDAOWS; // Stub generado automáticamente Añadir como atributo de la clase el objeto proxy que permite acceder al EJB local, con su correspondiente anotación que lo declara como tal: ... @EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class) private VisaDAOLocal dao; ... En el cuerpo del servlet, eliminar la declaración de la instancia del antiguo webservice VisaDAOWS, así como el código necesario para obtener la referencia remota ... VisaDAOWS dao = null; VisaDAOWSService service = new VisaDAOWSService(); dao = service.getVisaDAOWSPort() Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas. Eliminar también las referencias a BindingProvider.**
  2. Para esta parte simplemente ponemos el atributo que nos indican y comentamos todo lo referente a la parte anterior de VisaDOS
  3. ****
  4. ****

## También tenemos que modificar el GetPagos.java para que al llamar al getPagos() tenga en cuenta el nuevo retorno.

* 1. Tras realizar estas pequeñas modificaciones ya somos capaces de compilar el cliente.

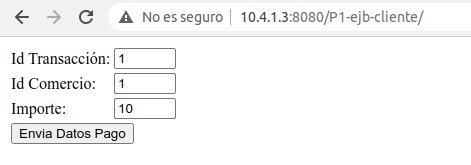
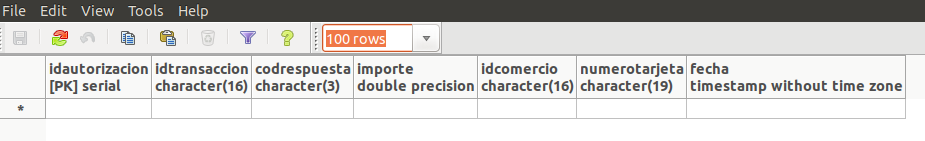
## Ejercicio número 3:

* 1. **Preparar los PCs con el esquema descrito y realizar el despliegue de la aplicación: Editar el archivo build.properties para que la propiedad as.host contenga la dirección IP del servidor de aplicaciones. Indica el valor y porqué es ese valor. Editar el archivo postgresql.properties para la propiedad db.client.host y db.host contengan las direcciones IP adecuadas para que el servidor de aplicaciones se conecte al postgresql, ambos estando en servidores diferentes. Indica qué valores y porqué son esos valores. Desplegar la aplicación de empresa ant desplegar**
  2. ****
  3. Sobre el servidor de aplicaciones residente en PC2VM se desplegará la aplicación, accediendo a la base de datos que residirá en PC1VM.
  4. Por como se puede ver en el dibujo lo referente a la base de datos debe estar en el PC1 mientras que lo referente al dominio va en el PC2
  5. ****

## 

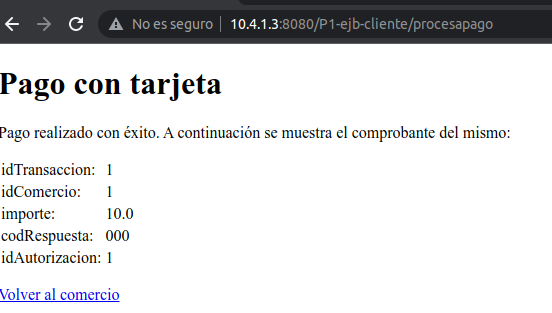
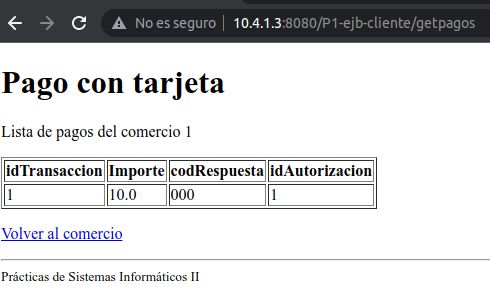
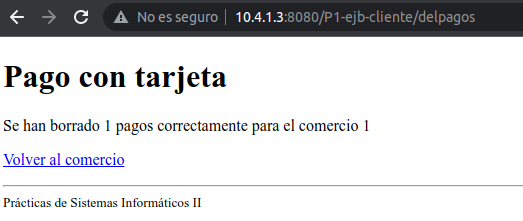
## 

## Ejercicio número 4:

* 1. **Comprobar el correcto funcionamiento de la aplicación mediante llamadas directas a través de las páginas pago.html y testbd.jsp (sin directconnection). Realice un pago. Lístelo. Elimínelo. Téngase en cuenta que la aplicación se habrá desplegado bajo la ruta /P1-ejb-cliente. Si la base de datos no se ha generado previamente, será necesario crearla usando ant regenerar-bd Incluya en la memoria de prácticas todos los pasos necesarios para resolver este ejercicio, así como las evidencias obtenidas. Se pueden incluir por ejemplo capturas de pantalla.**
  2. ****
  3. **Vemos en la base de datos como no hay de momento ningún pago.**
  4. ****

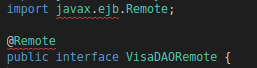
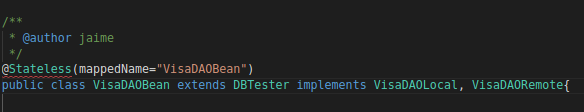
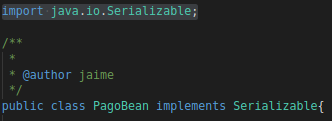
## 

## 

* 1. 
  2. 
  3. 

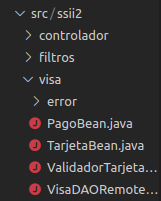
## 

## Ejercicio número 5:

* 1. **Realizar los cambios indicados en P1-ejb-servidor-remoto y preparar los PCs con el esquema de máquinas virtuales indicado. Compilar, empaquetar y desplegar de nuevo la aplicación P1-ejb como servidor de EJB remotos de forma similar a la realizada en el Ejercicio 3 con la Figura 2 como entorno de despliegue. Esta aplicación tendrá que desplegarse en la máquina virtual del PC2. Se recomienda replegar la aplicación anterior (EJB local) antes de desplegar ésta. Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas así como detallando los pasos realizados.**
  2. ****
  3. ****
  4. ****
  5. ****

## Ejercicio número 6:

* 1. **Realizar los cambios comentados en la aplicación P1-base para convertirla en P1-ejb-clienteremoto. Compilar, empaquetar y desplegar de nuevo la aplicación en otra máquina virtual distinta a la de la aplicación servidor, es decir, esta aplicación cliente estará desplegada en la MV del PC1 tal y como se muestra en el diagrama de despliegue de la Figura 2. Conectarse a la aplicación cliente y probar a realizar un pago. Comprobar los resultados e incluir en la memoria evidencias de que el pago ha sido realizado de forma correcta.**
  2. ****
  3. ****



Como se aprecia en esta imagen, se ha eliminado la ruta a ssii2/visa/dao y se ha copiado la interfaz VisaDAORemote.java desde P1-ejb-servidor-remoto.

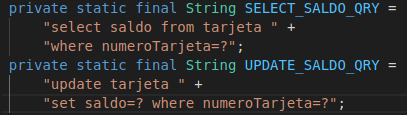
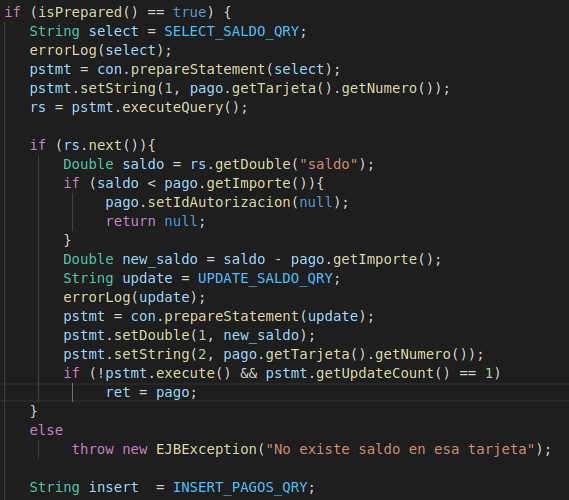
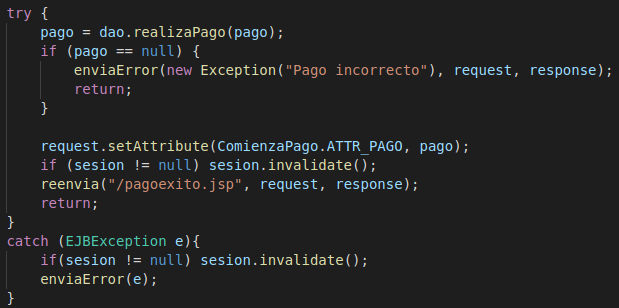
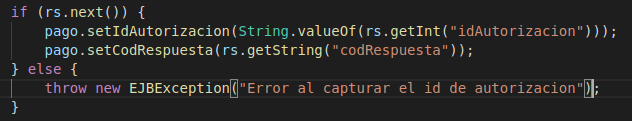
## 

## 

## Ejercicio número 7:

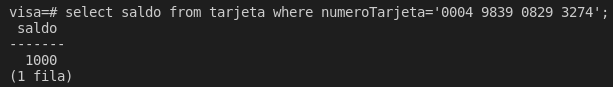
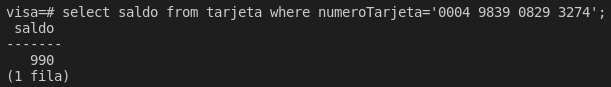
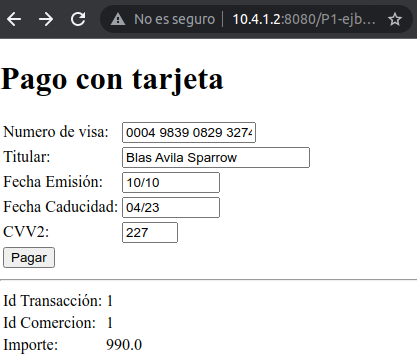
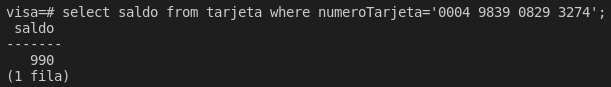
1. **Modificar la aplicación VISA para soportar el campo saldo**

## 

* 1. 
  2. 
  3. 

## 

## Ejercicio número 8:

* 1. **Desplegar y probar la nueva aplicación creada.**
  2. ****
  3. ****
  4. ****
  5. ****
  6. ****

## 

## Ejercicio número 9:

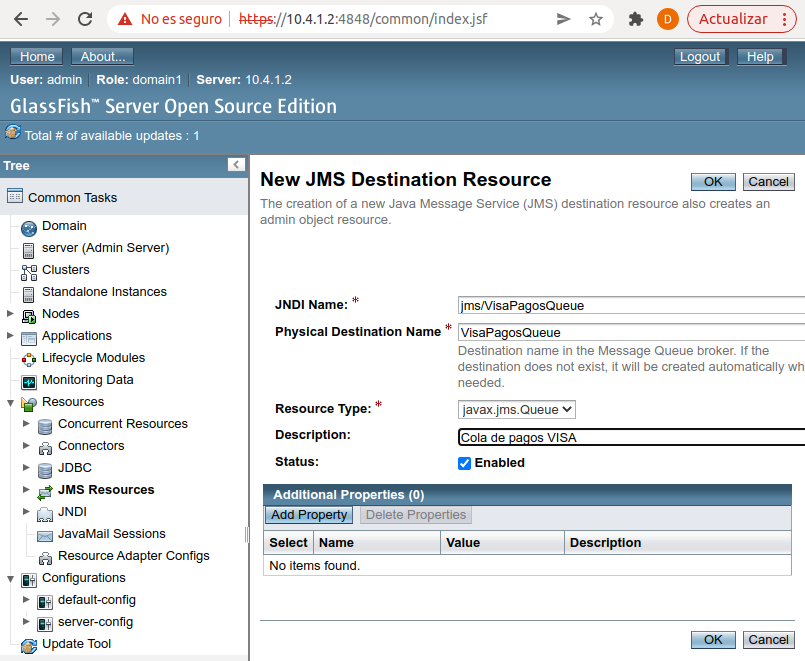
* 1. **En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la factoría de conexiones empleando la consola de administración, tal y como se adjunta en la Figura 4. Incluye una captura de pantalla donde se muestre dicha consola de administración con los cambios solicitados.**

## 

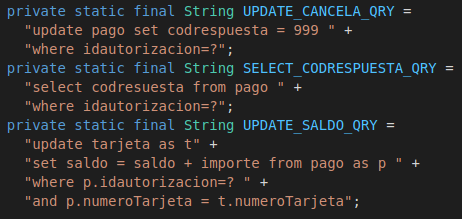
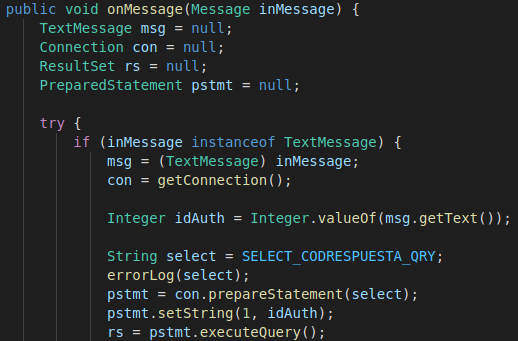
## 

## Ejercicio número 10:

* 1. **En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la conexión empleando la consola de administración, tal y como se adjunta en la Figura 5 Incluye una captura de pantalla donde se muestre dicha consola de administración con los cambios solicitados**

1. ****

## Ejercicio número 11:

* 1. ****
  2. ****
  3. 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

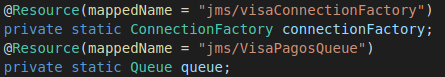
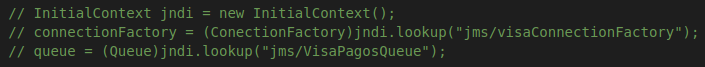
## 

## 

## 

## 

## Ejercicio número 12:

* 1. **Implemente ambos métodos en el cliente proporcionado. Deje comentado el método de acceso por la clase InitialContext de la API de JNDI. Indique en la memoria de prácticas qué ventajas podrían tener uno u otro método. Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas**
  2. ****
  3. ****

Una de las ventajas que tiene el segundo método frente al primero es que proporciona transparencia de ubicación, ya que no hace falta conocer el nombre de los recursos en tiempo de compilación, permitiendo así compilar sin problemas.

## Ejercicio número 13:

* 1. **Automatice la creación de los recursos JMS (cola y factoría de conexiones) en el build.xml y jms.xml. Para ello, indique en jms.properties los nombres de ambos y el Physical Destination Name de la cola de acuerdo a los valores asignados en los ejercicios 9 y 10. Recuerde también asignar las direcciones IP adecuadas a las variables as.host.mdb (build.properties) y as.host.server (jms.properties). ¿Por qué ha añadido esas IPs? Borre desde la consola de administración de Glassfish la connectionFactory y la cola creadas manualmente y ejecute: cd P1-jms ant todo Compruebe en la consola de administración del Glassfish que, efectivamente, los recursos se han creado automáticamente. Incluye una captura de pantalla, donde se muestre la consola de administración con los recursos creados. Revise el fichero jms.xml y anote en la memoria de prácticas cuál es el comando equivalente para crear una cola JMS usando la herramienta asadmin.**
  2. 

## 

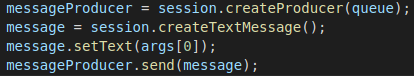
* 1. 
  2. Como se puede observar, se han creado tanto la factory como la queue perfectamente.

1. El comando resultante es el siguiente (con los campos completados):

asadmin --user admin --passwordfile passwordfile --host 10.4.1.2 --port 4848 create-jms-resources --restype javax.jms.QueueConnectionFactory --enabled=true --property Name=VisaPagosQueue jms/VisaPagosQueue

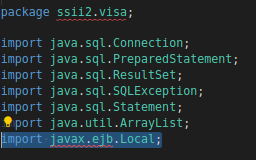
## 

## Ejercicio número 14:

* 1. **Modifique el cliente, VisaQueueMessageProducer.java, implementando el envío de args[0] como mensaje de texto (consultar los apéndices). Incluye en la memoria el fragmento de código que ha tenido que modificar.**
  2. ****

## Cuestiones:

**Cuestión 1. Abrir el archivo VisaDAOLocal.java y comprobar la definición de dicha interfaz. Anote en la memoria comentarios sobre las librerías Java EE importadas y las anotaciones utilizadas. ¿Para qué se utilizan? Comparar esta interfaz con el fichero de configuración del web service implementado en la práctica P1A.**



Como se puede ver se importa javax.ejb.Local desde javadoc podemos leer lo siguiente:

When used on the bean class, declares the local business interface(s) for a session bean. ... Use of the Local annotation is only required when the bean class does not implement only a single interface other than any of the following: java.io.Serializable; java.io.Externalizable; any of the interfaces defined in javax.ejb.

Es decir, nos sirve para emplear la etiqueta @Local que sirve para indicar que los métodos serán ejecutados por un cliente local.

**Cuestión 2. Abrir el archivo application.xml y explicar su contenido. Verifique el contenido de todos los archivos .jar / .war / .ear que se han construido hasta el momento (empleando el comando jar –tvf). Explique brevemente el contenido y ponga evidencias en la memoria.**

**El archivo se encuentra en conf/application/META-INF al abrirlo podemos ver la siguiente información.**

<?xml version="1.0" encoding="UTF-8"?>

<application version="5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/application\_5.xsd">

<display-name>P1-ejb</display-name>

<module>

<ejb>P1-ejb.jar</ejb>

</module>

<module>

<web>

<web-uri>P1-ejb-cliente.war</web-uri>

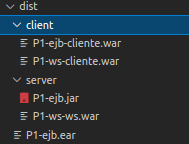
<context-root>/P1-ejb-cliente</context-root>

</web>

</module>

</application>

Estos son los archivos generados hasta el momento:



Y esta es la información mostrada con el jar

