UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2**			
Grupo	**2401**	**Práctica 1**	B	**Fecha**	13/03/2019
Alumna		Barroso, Honrubia, María			
Alumno		Carvajal, Moreno de Barreda, Alfonso			

Práctica 1: Arquitectura de JAVA EE (Segunda parte)

Cuestión número 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?

Las librerías importadas son las siguientes:

import java.sql.Connection;

Esta permite establecer una conexión con una base de datos desde el lenguaje de programación Java.

import java.sql.PreparedStatement;

Esta permite mandar al servidor exclusivamente los parametros para una consulta SQL 'precompilada' de manera que aumenta la velocidad de consulta.

import java.sql.ResultSet;

Un ResultSet es un objeto JAVA que representa el resultazdo de una consulta SQL. Es lo que devuelve la base de datos a través de JDBC

import java.sql.SQLException;

Esto permite traducir todos los errores de SQL a una exepción comprendida por Java.

import java.sql.Statement;

Esto permite hacer consultas SQL a la base de datos desde Java.

import java.util.ArrayList;

ArrayList es un tipo objeto en Java que representa una lista enlazada. Tiene la ventaja de que puede almacenar todo tipo de objetos. Este import permite a la aplicación trabajar con ellos.

import javax.ejb.Local;

Esto permite declarar las interfaces locales del negocio para una sesión de Bean.

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

```
//    public VisaDAOBean() {
//       return;
//    }
```

• Ajustar los métodos getPagos() a la interfaz definida en VisaDAOLocal

En el método getPagos() de VisaDAOBean, devolvemos los array de PagoBean:

```
PagoBean[] ret = null;
[...]
ret = new PagoBean[pagos.size()];
        ret = pagos.toArray(ret);
```

En getPagos.java:

```
PagoBean[] pagos = dao.getPagos(idComercio);
```

Ejercicio número 2

Modificar el servlet ProcesaPago, GetPagos y DelPagos para que accedan al EJB local.

• Importaciones de clases que se van a utilizar

```
import javax.ejb.EJB;
import ssii2.visa.VisaDAOLocal;
```

• Eliminadas importaciones que dejan de existir en el proyecto

• Añadir como atributo de la clase el objeto proxy que permite acceder al EJB local

```
@EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)
private VisaDAOLocal dao;
```

• Eliminada la declaración de la instancia del antiguo webservice VisaDAOWS, así como el código necesario para obtener la referencia remota

• También se han eliminado las referencias al BindingProvider

Para más detalle, mirar los servlets especificados.

Cuestión número 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). Anote sus comentarios y evidencias en la memoria.

El contenido del archivo application.xml es el siguiente:

<?xml version="1.0" encoding="UTF-8"?> → Indica al parseador de xml que la versión de xml que debe parsear es la 1.0. También que la codificación de los caractéres es la de 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">

→ Primero indica que la versión de la aplicación es la 5. El atributo 'xmlns' declara un espacio de nombres con el valor de la URI http://java.sun.com/xml/ns/javaee.

→ xmlns:xsi declara un prefijo standard de un espacio de nombres. Los atributos están en un espacio de nombres distinto al anterior; en este caso en http://www.w3.org/2001/XMLSchema-instance. Nos permite configurar el uso de una serie de atributos.

→ En este caso usamos el atributo xsi:schemaLocation que le indica al parseador XML cómo asociar un XSD con el documento XML. Tiene valores en pares: espacio de nombres y XSD-location-URI. En este caso la XSD-location-URI es http://java.sun.com/xml/ns/javaee/application_5.xsd y el espacio de nombres es http://java.sun.com/xml/ns/javaee.

<display-name>P1-ejb</display-name> indica el nombre de la ventana para la aplicación si se muestra por una GUI.

<module> Indica que comienza la descripción de un módulo. Cada elemento del módulo contiene una ejb, java, o elemento web, que indica el tipo de módulo y la localización del mismo dentro de la aplicación.

<ejb>P1-ejb.jar</ejb> Define un módulo de tipo EJB en el fichero de la aplicación. En este caso P1-ejb.jar.

<web> Define un módulo de tipo aplicación Web. Este elemento contiene un elemento de tipo "web-uri" y otro de tipo "context-root".

<web-uri>P1-ejb-cliente.war</web-uri> Define la localización del módulo Web (El nombre del .war)

<context-root>/P1-ejb-cliente</context-root> Define el context-root para la Web.

Ejercicio número 3:

Preparar los PCs con el esquema descrito y realizar el despliegue de la aplicación:

• Editar el archivo build.properties para que las propiedades as.host.client y as.host.server contengan la dirección IP del servidor de aplicaciones. Indica qué valores y porqué son esos valores.

Ambas ip's hacen referencia a la maquina virtual 2 porque es donde se encuentra el servidor glasflish (as.host.server) y porque a su vez actúa como cliente de la base de datos (as.host.client).

- as.host.client=10.1.10.2
- as.host.server=10.1.10.2

• 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.

La base de datos se encuentra en la maquina virtual 1, luego:

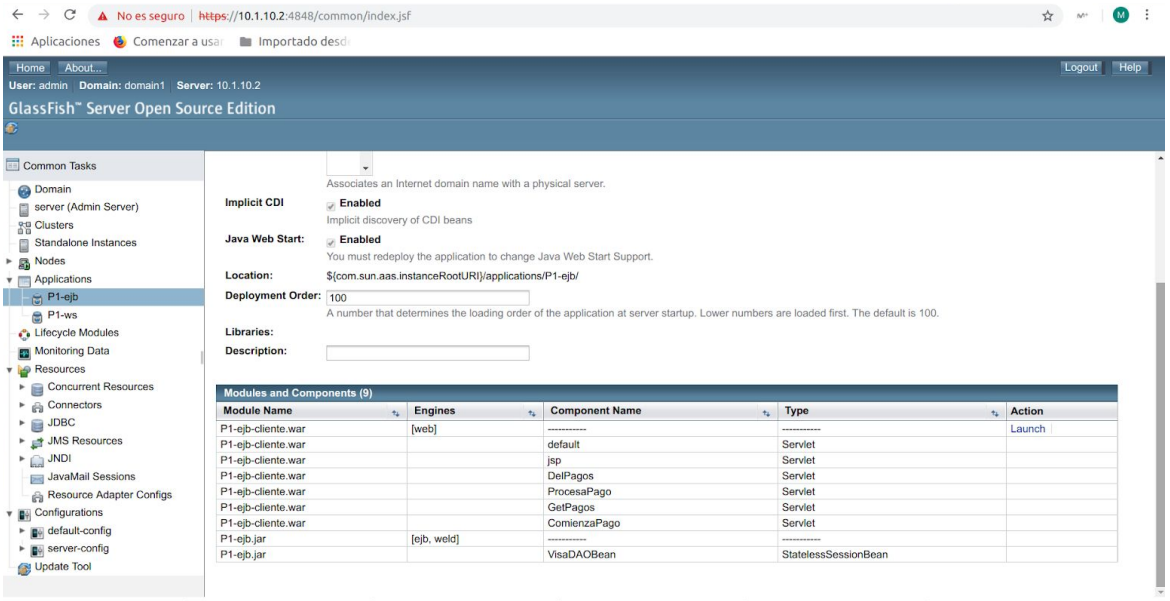
- db.host=10.1.10.1

Es el servidor, alojado en la máquina virtual 2, el que se comunica con la base de datos, luego:

- db.client.host=10.1.10.2

Desplegar la aplicación de empresa

En la siguiente imagen se muestra el despliegue de la aplicación en la dirección 10.1.10.2

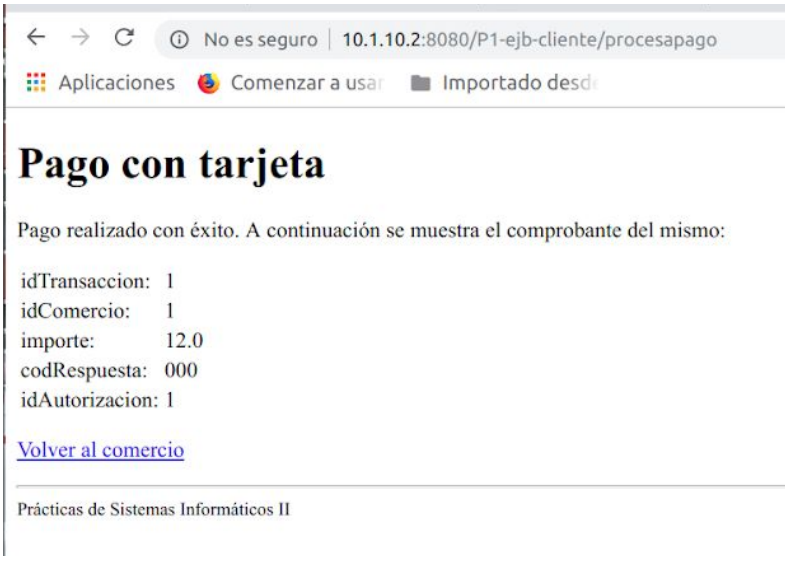


Ejercicio número 4:

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.

En https://10.1.10.2:8080/P1-ejb-cliente/testbd.jsp

- Realizamos el pago



- Listamos el pago

- Borramos el pago

Ejercicio número 5:

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.

- En VisaDAORemote:

  @Remote

  public interface VisaDAORemote

- En VisaDAOBean:

  public class VisaDAOBean extends DBTester implements VisaDAOLocal, VisaDAORemote

- En PagoBean:

  public class PagoBean implements Serializable

- En TarjetaBean:

public class TarjetaBean implements Serializable

No ha sido necesario modificar las direcciones IP's en los ficheros de configuración ya que el esquema de máquinas virtuales es el mismo.

Ejercicio número 6:

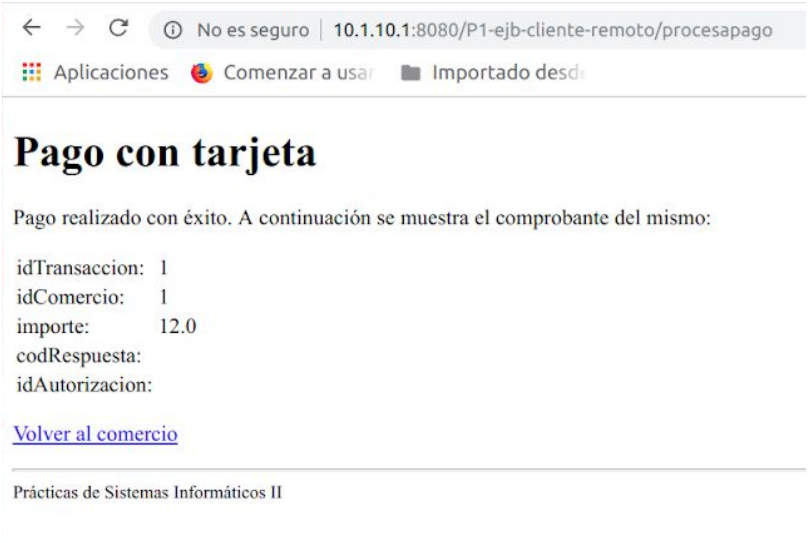
Realizar los cambios comentados en la aplicación P1-base para convertirla en P1-ejb-clienteremoto. Compilar, empaquetar y desplegar de nuevo la aplicación en otra máquina virtual distinta a la de la aplicación servidor, es decir, esta aplicación cliente estará desplegada en la MV del PC1 tal y como se 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.

Para implementar el cliente remoto hemos seguido los pasos descritos en el enunciado:

- Eliminar el directorio ssii2/visa/dao
- PagoBean y TarjetaBean se han marcado como Seralizables
- Copiar la interfaz remota VisaDAORemote.java de P1-ejb-servidor-remoto a P1-ejb-clienteremoto/src/ssii2/visa
- Eliminar la declaración de VisaDAO dao en los servlets que invocan la lógica de negocio e insertar una referencia al objeto remoto obtenida por inyección
- Crear un fichero glassfish-web.xml en la ruta web/WEB-INF que especifica cómo se resuelven las referencias a los EJBs remotos y cómo se invocan a los métodos remotos

Comprobamos su correcto funcionamiento realizando un pago:

- Realizamos un pago



- Listamos el pago

- Borramos el pago

Ejercicio número 7:

Modificar la aplicación VISA para soportar el campo saldo.

Archivo TarjetaBean.java:

• Añadir el atributo saldo y sus métodos de acceso

```
private double saldo;
public double getSaldo() {
    return saldo;
}

public void setSaldo(double saldo) {
    this.saldo = saldo;
}
```

Archivo VisaDAOBean.java:

• Importar la definición de la excepción EJBException que debe lanzar el servlet para indicar que se debe realizar un rollback:

```
import javax.ejb.EJBException;
```

• Declarar un prepared statement para recuperar el saldo de una tarjeta de la base de datos.

```
private static final String SELECT_SUELDO_TRANSACCION_QRY =
        "select saldo "+
        "from tarjeta "+
        "where numeroTarjeta=?";
```

• Declarar un prepared statement para insertar el nuevo saldo calculado en la base de datos.

```
private static final String UPDATE_SUELDO_TRANSACCION_QRY =
        "update tarjeta "+
        "set saldo=? "+
        "where numeroTarjeta=?";
```

• Modificar el método realizaPago con las siguientes acciones:

o Recuperar el saldo de la tarjeta a través del prepared statement declarado anteriormente.

```
String select1 = SELECT_SUELDO_TRANSACCION_QRY;
errorLog(select1);
pstmt = con.prepareStatement(select1);
pstmt.setString(1, pago.getTarjeta().getNumero());
ret = false;
rs = pstmt.executeQuery();
if (rs.next()){
        double saldo = rs.getDouble("saldo");
```

o Comprobar si el saldo es mayor o igual que el importe de la operación. Si no lo es, retornar denegando el pago (idAutorizacion= null y pago retornado=null)

```
 if (saldo < pago.getImporte()){
      pago.setIdTransaccion(null);
      return null;
    }
```

o Si el saldo es suficiente, decrementarlo en el valor del importe del pago y actualizar el registro de la tarjeta para reflejar el nuevo saldo mediante el prepared statement declarado anteriormente.

```
else{
      //actualizamos sueldo
      double nuevoSaldo = saldo-pago.getImporte();
      String update = UPDATE_SUELDO_TRANSACCION_QRY;
      errorLog(update);
      pstmt = con.prepareStatement(update);
      pstmt.setDouble(1, nuevoSaldo);
```

```
            pstmt.setString(2, pago.getTarjeta().getNumero());
            if (!pstmt.execute() && pstmt.getUpdateCount() == 1) {
                ret = true;
            }
        }
```

o Si lo anterior es correcto, ejecutar el proceso de inserción del pago y obtención del idAutorizacion, tal como se realizaba en la práctica anterior (este código ya debe estar programado y no es necesario modificarlo).

o En caso de producirse cualquier error a lo largo del proceso (por ejemplo, si no se obtiene el idAutorizacion porque la transacción está duplicada), lanzar una excepción EJBException para retornar al cliente

```
        else{
          throw new EJBException("Algo ha ido mal");
        }
```

• Modificar el servlet ProcesaPago para que capture la posible interrupción EJBException lanzada por realizaPago, y, en caso de que se haya lanzado, devuelva la página de error mediante el método enviaError (recordar antes de retornar que se debe invalidar la sesión, si es que existe).

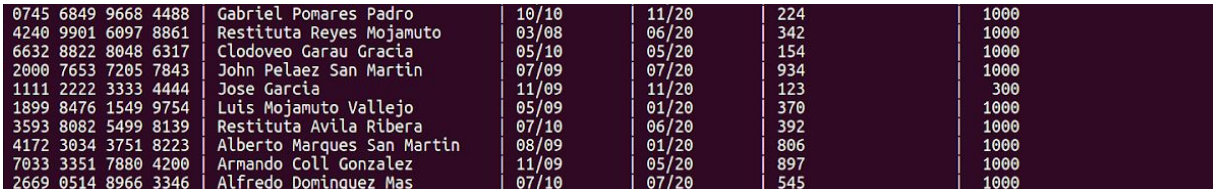
```
try{
        pago = dao.realizaPago(pago);
        if (pago == null) {
            enviaError(new Exception("Pago incorrecto"), request, response);
            sesion.invalidate(); //aqui tb hace falta???
            return;
        }
    }
    catch (EJBException e){
        enviaError(new Exception("Pago incorrecto"), request, response);
        sesion.invalidate();
        return;
    }
```

Ejercicio número 8:

Desplegar y probar la nueva aplicación creada.

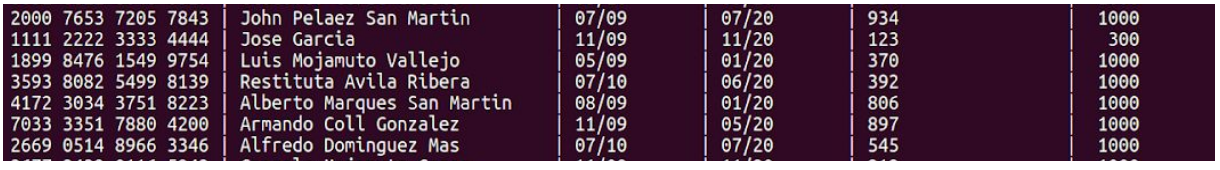
• Probar a realizar pagos correctos. Comprobar que disminuye el saldo de las tarjetas sobre las que realice operaciones. Añadir a la memoria las evidencias obtenidas.

Se puede comprobar que el pago se ha realizado con éxito observando el saldo del usuario.



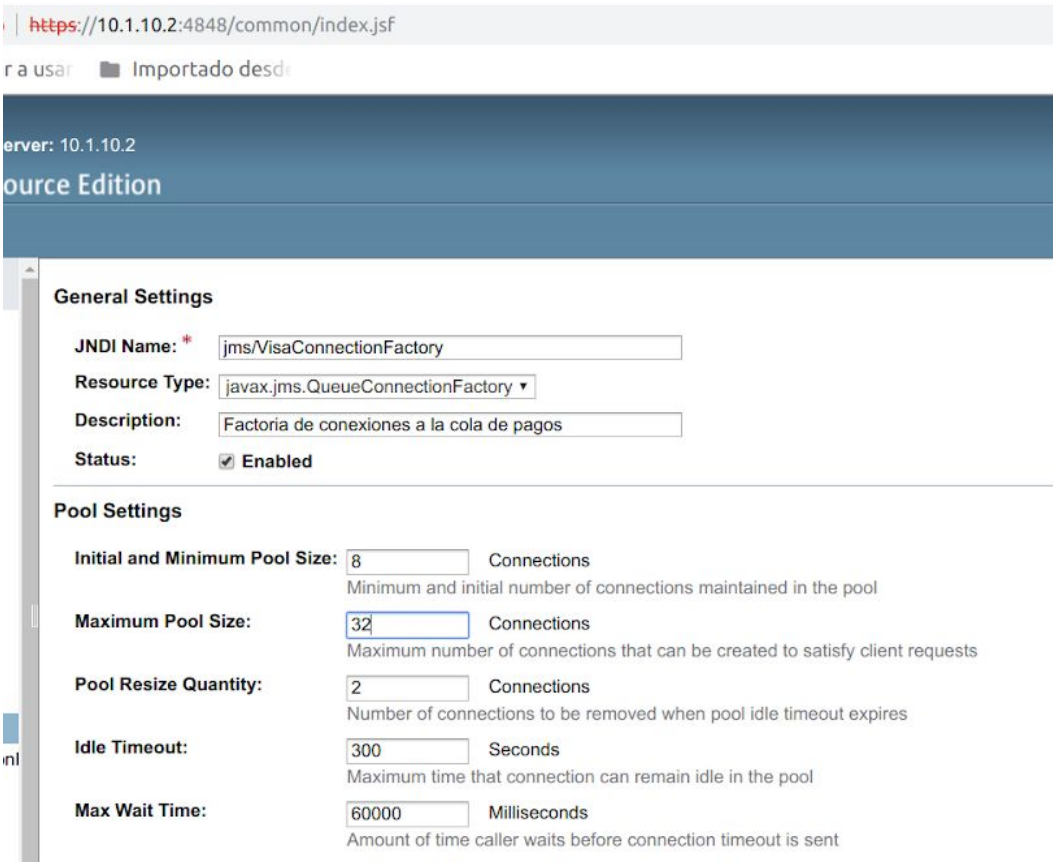
• Realice una operación con identificador de transacción y de comercio duplicados. Compruebe que el saldo de la tarjeta especificada en el pago no se ha variado.

Se puede comprobar como dicho saldo no ha variado para el mismo usuario.



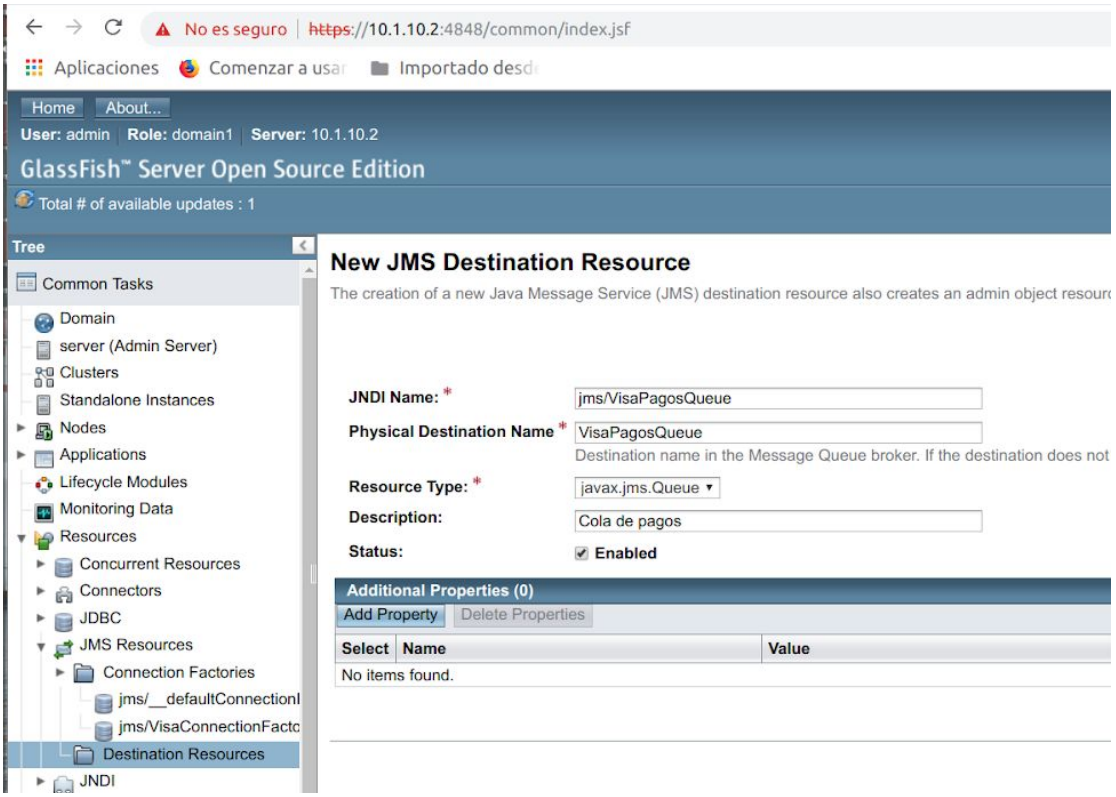
Ejercicio número 9:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.1.10.2), 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:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.10.1.2), 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.



Ejercicio número 11:

• Modifique el fichero sun-ejb-jar.xml para que el MDB conecte adecuadamente a su connection factory

```
<ejb>
    <ejb-name>VisaCancelacionJMSBean</ejb-name>
    <mdb-connection-factory>
    <jndi-name>jms/VisaConnectionFactory</jndi-name>
    </mdb-connection-factory>
</ejb>
```

Incluya en la clase VisaCancelacionJMSBean:

• Consulta SQL necesaria para actualizar el código de respuesta a valor 999, de aquella autorización existente en la tabla de pagos cuyo idAutorizacion coincida con lo recibido por el mensaje.

```
private static final String UPDATE_CANCELA_QRY = "update pago set codRespuesta=999 where idAutorizacion=?";
```

• Consulta SQL necesaria para rectificar el saldo de la tarjeta que realizó el pago

```
private static final String UPDATE_SALDO_QRY = "update tarjeta set saldo = saldo + pago.importe from pago where pago.numeroTarjeta = tarjeta.numeroTarjeta and pago.idautorizacion = ?";
```

• Método onMessage() que implemente ambas actualizaciones. Para ello tome de ejemplo el código SQL de ejercicios anteriores, de modo que se use un prepared statement que haga

bind del idAutorizacion para cada mensaje recibido.

```
if (inMessage instanceof TextMessage) {
        msg = (TextMessage) inMessage;
        logger.info("MESSAGE BEAN: Message received: " + msg.getText());
        int id = Integer.parseInt(msg.getText());

        con = getConnection();

        String cancela = UPDATE_CANCELA_QRY;
        pstmt = con.prepareStatement(cancela);
        pstmt.setInt(1, id);
        pstmt.execute();

        String saldo = UPDATE_SALDO_QRY;
        pstmt = con.prepareStatement(saldo);
        pstmt.setInt(1, id);
        pstmt.execute();
}
```

Ejercicio número 12:

• Implemente ambos métodos en el cliente proporcionado. Deje comentado el método de acceso por JNDI. Indique en la memoria de prácticas qué ventajas podrían tener uno u otro método.

```
@Resource(mappedName = "jms/VisaConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/VisaPagosQueue")
private static Queue queue;

//InitialContext jndi = new InitialContext();
//connectionFactory =
///(ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");
//queue = (Queue)jndi.lookup("jms/VisaPagosQueue");
```

El método estático tiene de inconveniente que el nombre de la cola hay que conocerlo en tiempo de compilación. En cambio, el método dinámico supone que no hay que conocer el nombre de la cola con antelación sino que se realiza una búsqueda de la misma. Esto puede ser más costoso a la hora de ejecución ya que el método estático no pierde tiempo en esta búsqueda, sin embargo permite mayor versatilidad a la hora de un realizar un proyecto distribuido.

Ejercicio número 13:

Automatice la creación de los recursos JMS (cola y connection factory) 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 7 y 8. 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?

En jms.properties:

jms.factoryname=jms/VisaConnectionFactory

jms.name=jms/VisaPagosQueue

jms.physname=VisaPagosQueue

En build.properties

as.host.mdb=10.1.10.2 (porque el host se encuentra en la maquina virtual 2)

En jms.properties:

as.host.client=10.1.10.1 (el consumidor de mensajes se encuentra en la maquina virtual 1)

as.host.server=10.1.10.2 (el productor de mensajes se encuentra en la máquina virtual 2)

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.

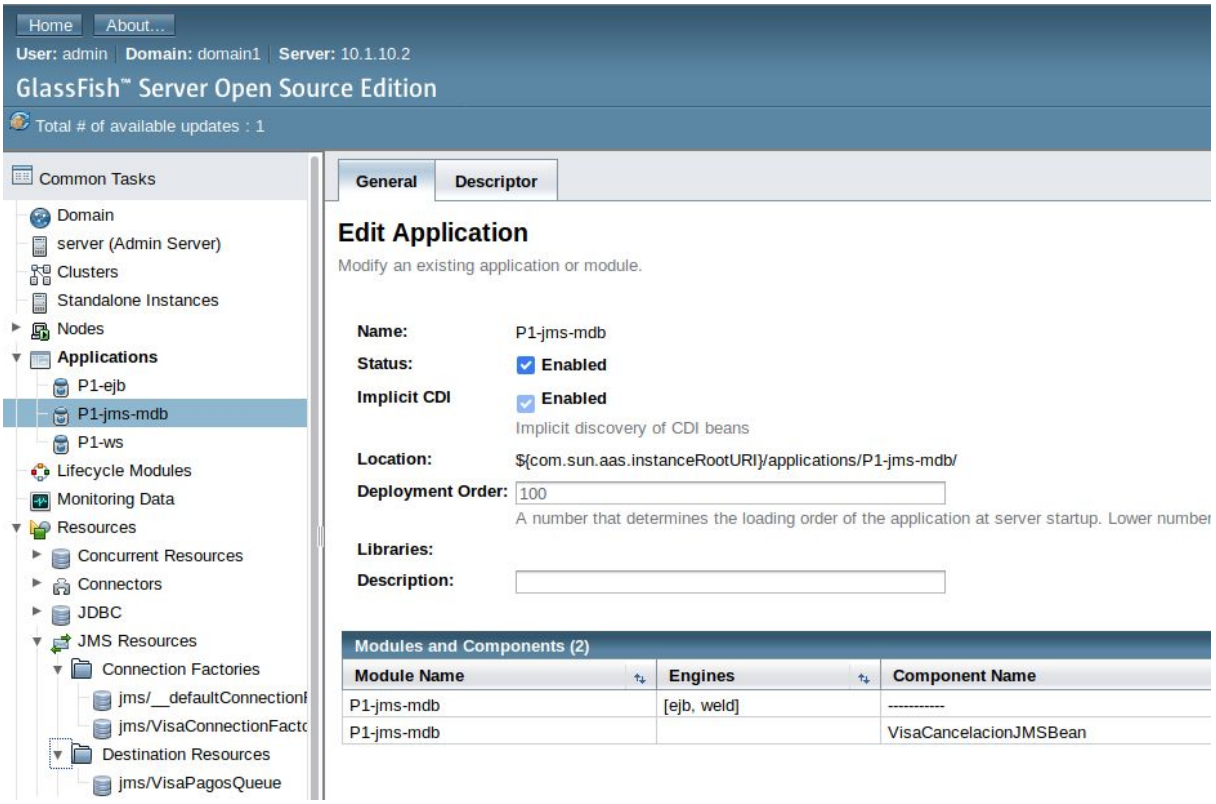
Este es el código xml para la creación de la ConnectionFactory.

```
<target name="create-jms-connection-factory"
        description="creates jms connection factory">
        <exec executable="${asadmin}">
                <arg line=" --user ${as.user}" />
                <arg line=" --passwordfile ${as.passwordfile}" />
                <arg line=" --host ${as.host.server}" />
                <arg line=" --port ${as.port}" />
                <arg line="create-jms-resource"/>
                <arg line=" --restype ${jms.restype}" />
                <arg line=" --enabled=true" />
```

```
                <arg line=" ${jms.resource.name}" />
            </exec>
        </target>
```

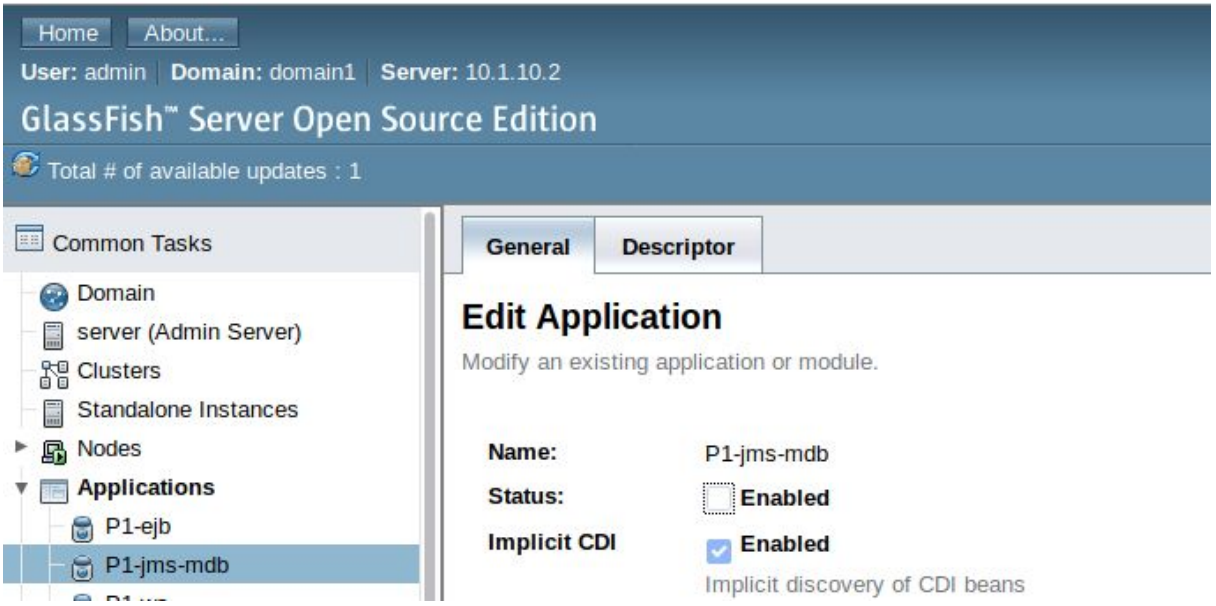
La el comando asadmin para crear la cola es “create-jms-resource”

Tras desplegar al aplicación mediante el comando “ant todo” podemos comprobar que se han creado automáticamente las colas.



Ejercicio número 14:

Importante: Detenga la ejecución del MDB con la consola de administración para poder realizar satisfactoriamente el siguiente ejercicio (check de ‘Enabled’ en Applications/P1-jms-mdb y guardar los cambios).



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.

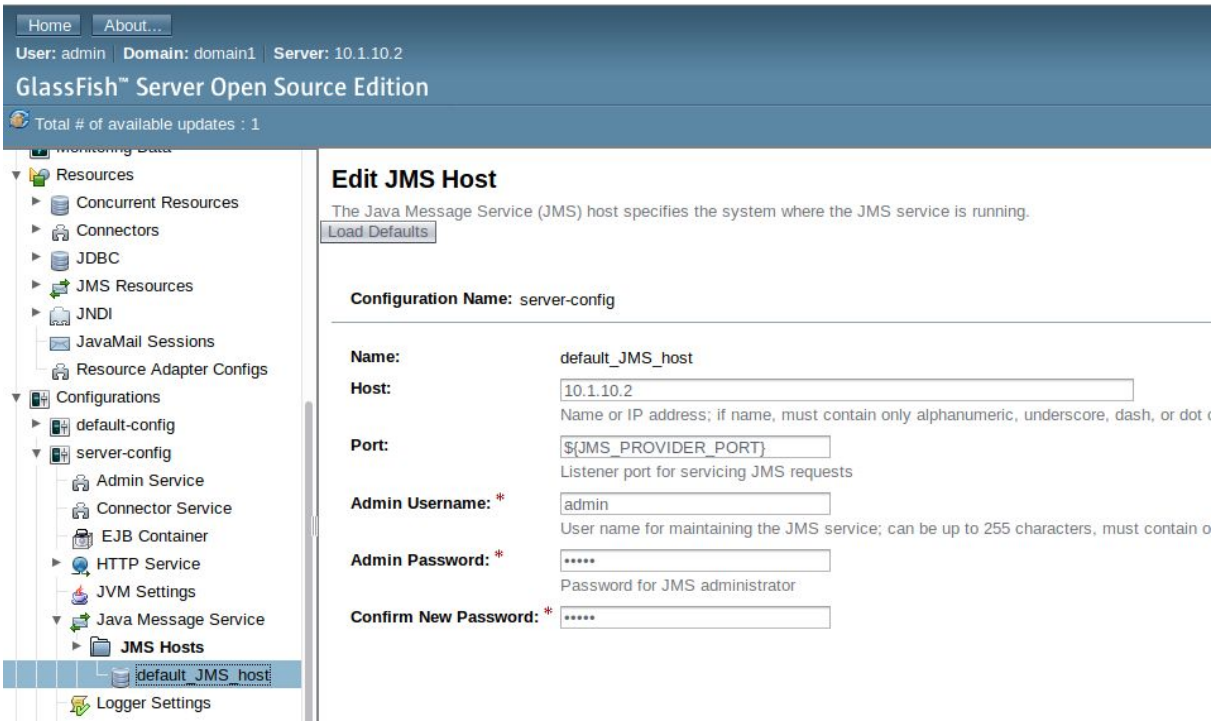
```
if (args[0].equals("-browse")) {
  browseMessages(session);
} else {
                messageProducer = session.createProducer(queue);
                textMessage = session.createTextMessage();
                textMessage.setText(args[0]);
                System.out.println("Enviando el mensaje: " + textMessage.getText());
                messageProducer.send(textMessage);

}
```

El fragmento que hemos modificado está en negrita.

Ejecute el cliente en el PC del laboratorio mediante el comando: /opt/glassfish-4.1.2/glassfish/bin/appclient –targetserver 10.X.Y.Z -client dist/clientjms/P1-jms-clientjms.jar idAutorizacion

Donde 10.X.Y.Z representa la dirección IP de la máquina virtual en cuyo servidor de aplicaciones se encuentra desplegado el MDB. Para garantizar que el comando funcione correctamente es necesario fijar la variable (web console->Configurations->server-config->Java Message Service->JMS Hosts→default_JMS_host) que toma el valor "localhost" por la dirección IP de dicha máquina virtual. El cambio se puede llevar a cabo desde la consola de administración. Será necesario reiniciar el servidor de aplicaciones para que surja efecto.



Tras ejecutar el comando **/opt/glassfish-4.1.2/glassfish/bin/appclient –targetserver 10.1.10.2**

-client dist/clientjms/P1-jms-clientjms.jar idAutorizacion y el comando **/opt/glassfish-4.1.2/glassfish/bin/appclient –targetserver 10.X.Y.Z -client dist/clientjms/P1-jms-clientjms.jar -browse** obtenemos las siguientes salidas:

```
fons@fonsmb:~/Documents/SI2/p1b/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.10.2 -client dist/clientjms/P1-jms-clientjm
s.jar idAutorizacion
Mar 13, 2019 7:23:49 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 13, 2019 7:23:50 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1.1  (Build 2-c) Compile:  March 17 2015 1045
Mar 13, 2019 7:23:50 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
Mar 13, 2019 7:23:50 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el mensaje: idAutorizacion
fons@fonsmb:~/Documents/SI2/p1b/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.10.2 -client dist/clientjms/P1-jms-clientjm
s.jar -browse
Mar 13, 2019 7:24:20 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 13, 2019 7:24:21 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1.1  (Build 2-c) Compile:  March 17 2015 1045
Mar 13, 2019 7:24:21 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
Mar 13, 2019 7:24:21 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensajes en cola:
idAutorizacion
```

Ahora Realizamos un pago:

Comprobamos que se ha realizado:

```
 idautorizacion |  idtransaccion   | codrespuesta | importe |    idcomercio    |    numerotarjeta    |          fecha
----------------+------------------+--------------+---------+------------------+---------------------+-------------------------
              1 | 1                | 000          |     100 | 1                | 1111 2222 3333 4444 | 2019-03-13 11:20:41.468191
(1 row)
```

Mandamos un mensaje con el identifcador 1:

```
si2@si2srv01:~$ export JAVA_HOME=/usr/lib/jvm/java-8-oracle/
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.10.2 -client /tmp/P1-jms-clientjms.jar 1
Mar 13, 2019 12:35:43 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 13, 2019 12:35:43 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1.1  (Build 2-c) Compile:  March 17 2015 1045
Mar 13, 2019 12:35:43 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
Mar 13, 2019 12:35:43 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el mensaje: 1
```

Comprobamos que se ha cancelado el pago:

```
visa=# select * from pago
;
 idautorizacion |  idtransaccion   | codrespuesta | importe |    idcomercio    |    numerotarjeta    |          fecha
----------------+------------------+--------------+---------+------------------+---------------------+-------------------------
              1 | 1                | 999          |     100 | 1                | 1111 2222 3333 4444 | 2019-03-13 12:35:15.359905
(1 row)
```

```
5012 6973 8189 6576 | Restituta Cozar Punset          | 01/08 | 07/20 | 623 | 1000
0745 6849 9668 4488 | Gabriel Pomares Padro           | 10/10 | 11/20 | 224 | 1000
4240 9901 6097 8861 | Restituta Reyes Mojamuto        | 03/08 | 06/20 | 342 | 1000
6632 8822 8048 6317 | Clodoveo Garau Gracia           | 05/10 | 05/20 | 154 | 1000
2000 7653 7205 7843 | John Pelaez San Martin          | 07/09 | 07/20 | 934 | 1000
1111 2222 3333 4444 | Jose Garcia                     | 11/09 | 11/20 | 123 | 1000
1899 8476 1549 9754 | Luis Mojamuto Vallejo           | 05/09 | 01/20 | 370 | 1000
```

Finalmente comprobamos que el saldo se ha restaurado

Se puede observar como el saldo de la tarjeta ha vuelto a 1000, su valor original.