

 <b>Escuela Politécnica Superior</b> <b>Ingeniería Informática</b> <b>Prácticas de Sistemas Informáticos 2</b>					
<b>Grupo</b>	<b>2401</b>	<b>Práctica</b>	<b>1B</b>	<b>Fecha</b>	<b>19/03/2021</b>
<b>Alumno/a</b>	Lerma, Martínez, Francisco				
<b>Alumno/a</b>	Morato, Martín, Jesús				

## **PRÁCTICA 1: ARQUITECTURA DE JAVA EE (SEGUNDA PARTE)**

### **Cuestión 1**

Vemos que en el fichero VisaDAOLocal tenemos las siguientes librerías:

java.sql.Connection: Esta librería representa una sesión con una base de datos específica. Dentro del contexto de esta conexión mediante getConnection(), se ejecutan sentencias SQL y se devuelven los resultados. Esta conexión es capaz de proporcionar información que describe sus tablas, su gramática SQL soportada, sus procedimientos almacenados, las capacidades de esta conexión, etc. Esta información se obtiene con el método getMetaData

java.sql.PreparedStatement: Permite ejecutar querys SQL utilizando el paquete JDBC mediante executeQuery(), se puede obtener un objeto PreparedStatement llamando al método prepareStatement(). Las querys SQL que pasan por este método van a la Base de Datos para precompilarse si el driver JDBC las soporta.

java.sql.ResultSet: Mediante esta librería, se organizan en tablas los resultados de las órdenes SQL devueltas por la función executeQuery(). Las filas deben leerse una detrás de otra, pero las columnas pueden leerse en cualquier orden.

java.sql.SQLException: Interfaz de excepciones que proporcionan información sobre los errores de la base de datos u otros errores.

java.sql.Statement: Mediante esta interfaz, se envían las órdenes SQL individuales a la base de datos a través del controlador JDBC y se recogen los resultados de las mismas.

java.util.ArrayList: Esta interfaz es la encargada de agrupar una colección de elementos en forma de lista, es decir, uno detrás de otro. En una lista los elementos pueden ser accedidos por un índice que indica la posición del elemento en la colección.

javax.ejb.Local: Esta interfaz local es para aquellos clientes que corren en la misma máquina virtual que el contenedor EJB. Se etiqueta con @Local antes de la definición de la interfaz.

Vemos que en este fichero comprandolo con el fichero VisaDAO.java de la práctica anterior las interfaces son prácticamente las mismas salvo la gran diferencia que en este caso se importa la interfaz local ejb de java, en lugar de las clases que nos permitían desarrollar la funcionalidad de los servicios web.

## EJERCICIO 1

Como se nos pedía en el enunciado hemos importado la interfaz local del EJB y hemos cambiado la definición de la clase para que se adapte a los nuevos requerimientos:

```
import javax.ejb.Stateless;
/*
 * @author jaime
 */

@Stateless (mappedName="VisaDAOBean")
public class VisaDAOBean extends DBTester implements VisaDAOLocal{
```

Eliminamos el constructor. Además hemos tenido que modificar también el método getPagos() ya que como tiene que seguir la interfaz definida en VisaDAOLocal, devolverá una lista de PagoBean (PagoBean[ ]) en vez de un ArrayList de PagoBeans (ArrayList<PagoBean>) como hacía antes, estos son los cambios realizados:

```
public PagoBean[] getPagos(String idComercio){

    PreparedStatement pstmt = null;
    Connection pcon = null;
    ResultSet rs = null;
    PagoBean[] ret = null;
    ArrayList<PagoBean> pagos = null;
    String qry = null;
```

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

```
return ret;
```

Ya se podría compilar el servidor con ant compile-servidor.

## EJERCICIO 2

Para este ejercicio hemos modificado el servlet procesaPago.java de la siguiente manera, hemos añadido las los import necesarios para desarrollar la funcionalidad referida al EJB local de tal forma:

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

Además hemos añadido como atributo de la clase el objeto proxy que permite acceder al EJB local:

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

Por otro lado, hemos eliminado todas las referencias al servicio de Web Services y las referencias al Binding Provider que teníamos de la práctica anterior. Además al cambiar esta interfaz hemos tenido que modificar el método getPagos(), esto es debido al cambio en la interfaz del antiguo VisaDAOWS:

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

## CUESTIÓN 2

```
<?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.s
<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>
```

Abrimos el archivo aplicacion.xml (captura arriba) este fichero describe cada uno de los módulos de la aplicación (web, xml), vemos que en este fichero se nos especifica la versión de la aplicación que en este caso es “5”, además tenemos atributo xmlns: identifica el espacio de nombres al que pertenecen los componentes incluidos en el esquema, normalmente a este espacio de nombres se le asigna el prefijo “xsi” (aunque se puede usar cualquier prefijo), por otro lado tenemos schemaLocation: asocia un documento de esquema que tiene un espacio de nombres de destino (targetNamespace) con un documento de instancia, este atributo tiene dos valores, separados por un espacio en blanco. El primer valor coincide con el del “targetNamespace” especificado en el schema. El segundo es la ubicación donde se encuentra definido el XML schema.

Además en este fichero encontramos los archivos .jar/ .war/ .ear que se han generado al empaquetar cada uno de los componentes del sistema, más concretamente: P1-ejb.jar (generado al empaquetar servidor) dentro del directorio dist/server, P1-ejb-cliente.war (generado al empaquetar el cliente) en el subdirectorio dist.

## EJERCICIO 3

Para el despliegue de la aplicación como se nos indica en la figura 2 debemos realizar las modificaciones pertinente por un lado en build.properties de tal forma que las direcciones nos quedan así:

```
as.host.client=10.1.4.2  
as.host.server=10.1.4.2
```

Los valores que hemos modificado son:

- as.host.client: 10.1.4.2 Es decir el cliente del sistema se encuentra en la dirección del PCMV2
- as.host.server: 10.1.4.2 Es decir el servidor del sistema se encuentra en la dirección del PCMV2

Ya que como se ve en la figura 2, la segunda JVM ejecutará la aplicación glassfish.

Por otro lado hemos modificado tambien el fichero postgres.properties:

```
db.name=visa  
db.user=alumnodb  
db.password=*****  
db.port=5432  
db.host=10.1.4.1  
# Recursos y pools asociados  
db.pool.name=VisaPool  
db.jdbc.resource.name=jdbc/VisaDB  
db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}  
db.client.host=10.1.4.1  
db.client.port=4848
```

Los valores que hemos modificado son:

- db.host: 10.1.4.1 Es decir el servidor de la base de datos se encuentra en la dirección del PCMV1
- db.client.host: 10.1.4.1 Es decir el cliente que conecta con la base de datos se encuentra en la dirección del PCMV1

Ya que en la figura 2 del enunciado se puede apreciar que el la JVM que contendrá PostgreSQL es la primera.

Tras el despliegue, hemos comprobado en la aplicación de administración de Glassfish en PC2VM el despliegue de la aplicación:

**Edit Application**

Modify an existing application or module.

**Name:** P1-ejb  
**Status:**  Enabled  
**Virtual Servers:** server

**Implicit CDI:**  Enabled  
**Java Web Start:**  Enabled  
**Location:** \${com.sun.aas.instanceRootURI}/applications/P1-ejb/  
**Deployment Order:** 100  
**Libraries:**  
**Description:**

Module Name	Engines	Component Name	Type	Action
P1-ejb-cliente.war	[web]	-----	-----	Launch
P1-ejb-cliente.war		default	Servlet	
P1-ejb-cliente.war		jsp	Servlet	
P1-ejb-cliente.war		DelPagos	Servlet	
P1-ejb-cliente.war		ProcesaPago	Servlet	
P1-ejb-cliente.war		GetPagos	Servlet	
P1-ejb-cliente.war		ComienzaPago	Servlet	
P1-ejb.jar	[ejb, weld]	-----	-----	
P1-ejb.jar		VisaDAOBean	StatelessSessionBean	

## EJERCICIO 4

Una vez desplegada la aplicación con el EJB local vamos a proceder a probar la funcionalidad de los pagos, primero realizamos un pago y vemos que se produce correctamente:

**Pago con tarjeta**

**Proceso de un pago**

Id Transacción:   
 Id Comercio:   
 Importe:   
 Número de visa:   
 Titular:   
 Fecha Emisión:   
 Fecha Caducidad:   
 CVV2:   
 Modo debug:  True  False  
 Direct Connection:  True  False  
 Use Prepared:  True  False

The screenshot shows a browser window with the URL <https://moodle...>. The page title is "Pago con tarjeta". The content displays a message: "Pago realizado con éxito. A continuación se muestra el comprobante del mismo:". Below this, there is a table with the following data:

idTransaccion:	45
idComercio:	12
importe:	19.0
codRespuesta:	000
idAutorizacion:	5

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Posteriormente procedemos a obtener los pagos de este comercio para comprobar que verdaderamente se han guardado en la base de datos, y eliminamos ese pago:

The screenshot shows a browser window with the URL <https://moodle...>. The page title is "Pago con tarjeta". The content displays a message: "Lista de pagos del comercio 12". Below this, there is a table with the following data:

idTransaccion	Importe	codRespuesta	idAutorizacion
45	19.0	000	6

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

The screenshot shows a browser window with the URL <https://moodle...>. The page title is "Pago con tarjeta". The content displays a message: "Se han borrado 1 pagos correctamente para el comercio 12". Below this, there is a link:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

También probamos a realizar un pago mediante pago.html y comprobamos que funciona:

10.1.4.2:8080/P1-ejb-cliente/pago.html

Id Transacción: 30  
Id Comercio: 60  
Importe: 15  
Envia Datos Pago



### Pago con tarjeta

Numero de visa: 2347 4840 5058 7931  
Titular: Gabriel Avila Locke  
Fecha Emisión: 11/09  
Fecha Caducidad: 01/22  
CVV2: 207  
Pagar

Id Transacción: 30  
Id Comercio: 60  
Importe: 15.0

Prácticas de Sistemas Informáticos II



### Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 30  
idComercio: 60  
importe: 15.0  
codRespuesta: 000  
idAutorizacion: 7

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

## EJERCICIO 5

Los pasos que hemos seguido para definir la interfaz remota del EJB son los siguientes: Copiar el fichero VisaDAOLocal.java al fichero VisaDAORemote.java.

Cambiar, en VisaDAORemote.java, el nombre de la interfaz a VisaDAORemote y cambiar la anotación @Local por @Remote y añadir import javax.ejb.Remote; y quitar import javax.ejb.Local;

```

package ssii2.visa;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import javax.ejb.Remote;

@Remote
public interface VisaDAORemote {
    public boolean compruebaTarjeta(TarjetaBean tarjeta);
    public PagoBean realizaPago(PagoBean pago);
    public PagoBean[] getPagos(String idComercio);
    public int delPagos(String idComercio);
    public boolean isDebug();
    public boolean isPrepared();
    public void setPrepared(boolean prepared);
    public void setDebug(boolean debug);
    public int getDirectConnectionCount();
    public int getDSNConnectionCount();
    public boolean isDirectConnection();
    public void setDirectConnection(boolean directConnection);
}

```

Hacer que VisaDAOBean implemente ambas interfaces, la local y la remota:

```

@Stateless (mappedName="VisaDAOBean")
public class VisaDAOBean extends DBTester implements VisaDAOLocal, VisaDAORemote{

```

Además para compatibilizar las nuevas implementaciones de clases hemos marcado como serializables las clases de los ficheros PagoBean y Tarjeta Bean, realizando los imports necesarios y haciendo que estas clases implementen la clase Serializable:

```

import java.io.Serializable;

/**
 *
 * @author jaime
 */
public class PagoBean implements Serializable{

```

```
import java.io.Serializable;

public class TarjetaBean implements Serializable{
```

Posteriormente hemos procedido a compilar, empaquetar y desplegar de nuevo la aplicación P1-ejb como servidor de EJB remotos. Esta aplicación la hemos desplegado en la máquina virtual del PC2.

## EJERCICIO 6

Tras realizar los cambios indicados en el ejercicio incluyendo la serialización de PagoBean y TarjetaBean como en el servidor remoto y en los servlets que invocan la lógica de negocio (procesapago.java, getpagos.java y delpagos.java) eliminamos declaración de VisaDAO dao, e insertamos una referencia al objeto remoto obtenida por inyección (se incluye en DelPagos.java, GetPagos.java, ProcesaPago.java).

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

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

Posteriormente, creamos un fichero con nombre glassfish-web.xml en la ruta web/WEB-INF que especificará cómo se resolverán las referencias a los EJBs remotos y cómo se invocará a los métodos remotos:

```
web > WEB-INF > glassfish-web.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  |  !DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Servlet 3.0//EN"
3  |  "http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
4  |  <glassfish-web-app>
5  |  |  <ejb-ref>
6  |  |  |  <ejb-ref-name>VisaDAOBean</ejb-ref-name>
7  |  |  |  <jndi-name>corbaname:iiop:10.1.4.2:3700#java:global/P1-ejb/P1-ejb/VisaDAOBean!ssii2.visa.VisaDAORemote</jndi-name>
8  |  |  </ejb-ref>
9  |  </glassfish-web-app>
```

Tras esto modificamos los ficheros build.properties y postgres.properties del cliente con la ip del cliente que es 10.1.4.1.

Probamos a crear un pago, buscarlo y eliminarlo con la aplicación cliente remota:

## Pago con tarjeta

### Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Número de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug:  True  False

Direct Connection:  True  False

Use Prepared:  True  False

## Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 33  
idComercio: 33  
importe: 33.0  
codRespuesta:  
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

## Pago con tarjeta

Lista de pagos del comercio 33

idTransaccion	Importe	codRespuesta	idAutorizacion
33	33.0	000	8

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

## Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 33

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

## EJERCICIO 7

Se ha formado el directorio P1-ejb-transaccional tal y como se indica en el enunciado de la práctica.

Se ha añadido a TarjetaBean el double con el saldo que es el nuevo parámetro (private double saldo;).

En VisaDAOBean.java hemos hecho lo siguiente:

- Hemos realizado el import de javax.ejb.EJBException.
- Hemos creado las dos consultas para los prepared statement para obtener y actualizar el saldo de una tarjeta:

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

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

- Para comprobar la validez de los pagos hemos añadido la siguiente funcionalidad que nos pedían en el ejercicio.

```
//Comprobar saldo
String selectSaldo = SELECT_SALDO_QRY;
errorLog(selectSaldo);
double importe = pago.getImporte();
pstmt = con.prepareStatement(selectSaldo);
pstmt.setString(1, pago.getTarjeta().getNumero());
rs = pstmt.executeQuery();
if(rs.next()){
    double saldo = rs.getDouble("saldo");
    if(saldo < importe){
        pago.setIdAutorizacion(null);
        return null;
    }
    else{
        String updateSaldo = UPDATE_SALDO_QRY;
        pstmt = con.prepareStatement(updateSaldo);
        pstmt.setDouble(1, saldo - importe);
        pstmt.setString(2, pago.getTarjeta().getNumero());
        if (!pstmt.execute()
            && pstmt.getUpdateCount() == 1){
            ret = pago;
        }
    }
}
else{
    throw new EJBException();
}
```

Por último hemos modificado en ProcesaPago para que pueda atrapar las nuevas excepciones EJB implementadas en el punto anterior, invalidando la sesión.

```
try{
    if((pago = dao.realizaPago(pago))==null){
        enviaError(new Exception("Pago incorrecto"), request, response);
        return;
    }
} catch(Exception e){
    enviaError(new Exception("Pago incorrecto"), request, response);
    if(sesion!=null) sesion.invalidate();
    return;
}
```

## EJERCICIO 8

Realizamos un pago mediante la página pago.html como se ve en las siguientes capturas de un importe de 100€.

Numero de visa: 2347 4840 5058 7931  
Titular: Gabriel Avila Locke  
Fecha Emisión: 11/09  
Fecha Caducidad: 01/22  
CVV2: 207  
Pagar

Id Transacción: 100  
Id Comercio: 100  
Importe: 100.0

Prácticas de Sistemas Informáticos II

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 100  
idComercio: 100  
importe: 100.0  
codRespuesta: 000  
idAutorizacion: 1

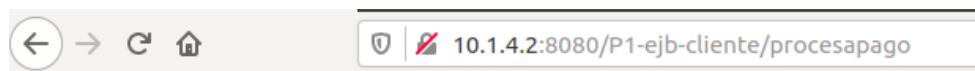
Volver al comercio

Prácticas de Sistemas Informáticos II

Si accedemos a la base de datos podemos comprobar que se ha restado una cantidad de 100€ a los 1000€ totales resultando en un total de 900€.

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo	
1	2347 4840 5058 7931	Gabriel Avila Locke	11/09	01/22	207	900	

Si volvemos a realizar el mismo pago con la misma id de comercio y transacción el pago es incorrecto y no se nos resta nada al saldo.



## Pago con tarjeta

Pago incorrecto

---

Prácticas de Sistemas Informáticos II

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo	
1	2347 4840 5058 7931	Gabriel Avila Locke	11/09	01/22	207	900	

Si realizamos un pago con más importe del total en la cuenta también resulta ser un pago incorrecto y no se nos restaría nada.



## Pago con tarjeta

### Proceso de un pago

Id Transacción:	<input type="text" value="100"/>
Id Comercio:	<input type="text" value="100"/>
Importe:	<input type="text" value="1100"/>
Numero de visa:	<input type="text" value="2347 4840 5058 7931"/>
Titular:	<input type="text" value="Gabriel Avila Locke"/>
Fecha Emisión:	<input type="text" value="11/09"/>
Fecha Caducidad:	<input type="text" value="01/22"/>
CVV2:	<input type="text" value="207"/>
Modo debug:	<input type="radio"/> True <input type="radio"/> False
Direct Connection:	<input type="radio"/> True <input type="radio"/> False
Use Prepared:	<input type="radio"/> True <input type="radio"/> False
<input type="button" value="Pagar"/>	

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo	
1	2347 4840 5058 7931	Gabriel Avila Locke	11/09	01/22	207	900	

## EJERCICIO 9

Hacemos los pasos que nos indican en el enunciado y rellenamos los siguientes campos tal y como se indica para crear la factoría de conexión:

The screenshot shows the GlassFish administration console at <https://10.1.4.2:4848/common/index.jsf>. The left sidebar shows various server components like Domain, Clusters, Applications, and Resources. Under Resources, JMS Resources is selected, showing Concurrent Resources, Connectors, JDBC, and JNDI. A 'New JMS Connection Factory' dialog is open, titled 'New JMS Connection Factory'. It contains two main sections: 'General Settings' and 'Pool Settings'. In 'General Settings', the JNDI Name is 'jms/VisaConnectionFactory', Resource Type is 'javax.jms.QueueConnectionFactory', Description is 'Factoría de conexiones a la cola de pagos', and Status is checked as 'Enabled'. In 'Pool Settings', the Initial and Minimum Pool Size is 8, Maximum Pool Size is 32, Pool Resize Quantity is 2, Idle Timeout is 300 seconds, Max Wait Time is 60000 milliseconds, and Transaction Support is set to 'None'. The 'Connection Validation' section has 'Required' checked. At the bottom, there's an 'Additional Properties (0)' section with 'Add Property' and 'Delete Properties' buttons.

Una vez le damos a “Ok” la cola de mensajes se crea y podemos ver en los recursos que se ha creado correctamente y está habilitada

### JMS Connection Factories

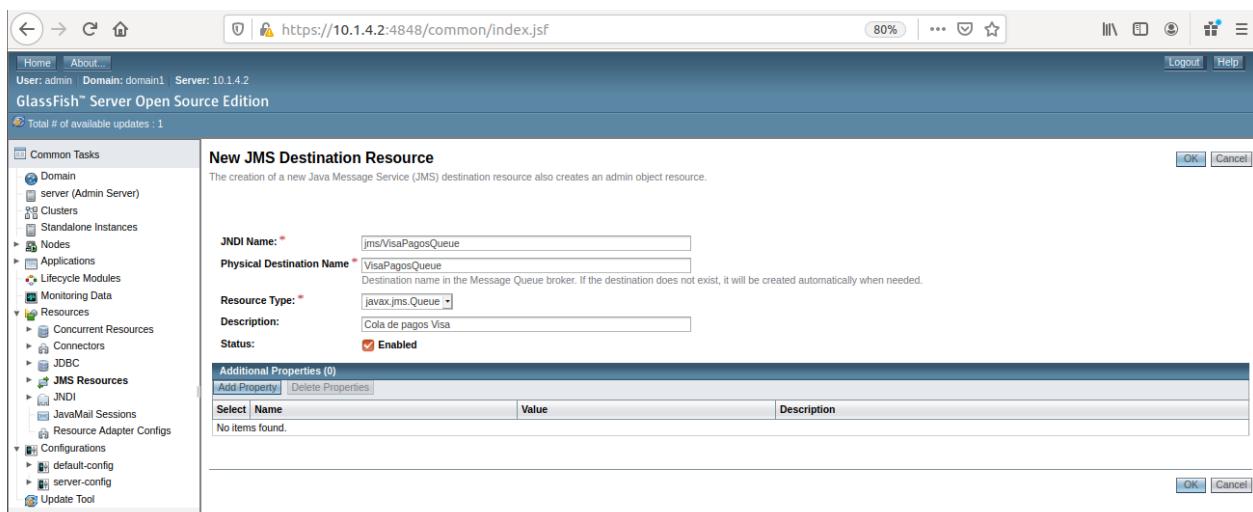
Java Message Service (JMS) connection factories are objects that allow an application to create other JMS objects programmatically. Click New to create a new connection factory. Click the name of a connection factory to modify its properties.

#### Connection Factories (2)

Select	JNDI Name	Logical JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/_defaultConnectionFactory	java:comp/DefaultJMSSConnectionFactory	<input checked="" type="checkbox"/>	javax.jms.ConnectionFactory	
<input type="checkbox"/>	jms/VisaConnectionFactory		<input checked="" type="checkbox"/>	javax.jms.QueueConnectionFactory	

## EJERCICIO 10

Hacemos los pasos que nos indican en el enunciado y rellenamos los siguientes campos tal y como se indica para crear la cola de mensajes:



Una vez le hemos dado a “Ok” la cola de mensajes se habrá creado correctamente y estará habilitada en los recursos destino tal y como se muestra en la siguiente imagen:



## EJERCICIO 11:

Tras descomprimir el archivo P1-jms-base.tgz, hemos procedido a añadir el nombre de la ConnectionFactory para que el MDB se conecte adecuadamente con esta (fichero sun-ejb-jar.xml):

```
P1-jms-base > P1-jms > conf > mdb > META-INF > sun-ejb-jar.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 9.0 EJB 3.0//EN" "http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-0.dtd">
3  <sun-ejb-jar>
4    <enterprise-beans>
5      <mdb-connection-factory>
6        <jndi-name>jms/VisaConnectionFactory</jndi-name>
7      </mdb-connection-factory>
8    </enterprise-beans>
9  </sun-ejb-jar>
10 
```

Por otro lado en la clase VisaCancelacionJMSBean hemos realizado las siguientes consultas para actualizar el código de respuesta a valor 999 y rectificar el saldo de la tarjeta que realizó el pago respectivamente:

```

private static final String UPDATE_CANCELAR_QRY =
    "update pago " +
    "set codRespuesta = '999' " +
    "where idAutorizacion = ?";

private static final String UPDATE_RECTIFY_QRY =
    "update tarjeta as t " +
    "set saldo = saldo + pago.importe " +
    "from pago as p " +
    "where p.idAutorizacion = ? " +
    "and p.numeroTarjeta = t.numeroTarjeta";

```

Además hemos modificado el método onMessage() que implementa ambas actualizaciones:

```

public void onMessage(Message inMessage) {
    TextMessage msg = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    Integer idAutorizacion = 0;
    Connection con = null;

    try {
        if (inMessage instanceof TextMessage) {
            msg = (TextMessage) inMessage;
            logger.info("MESSAGE BEAN: Message received: " + msg.getText());
            idAutorizacion = Integer.parseInt(msg.getText());
            // Obtener conexion
            con = getConnection();

            String updateSaldo = UPDATE_CANCELAR_QRY;
            String rectifySaldo = UPDATE_RECTIFY_QRY;

            pstmt = con.prepareStatement(updateSaldo);
            pstmt.setInt(1, idAutorizacion);
            if (pstmt.execute() || pstmt.getUpdateCount() != 1){
                throw new JMSException("Error en update de codRespuesta");
            }
            pstmt = con.prepareStatement(rectifySaldo);
            pstmt.setInt(1, idAutorizacion);
            if (pstmt.execute() || pstmt.getUpdateCount() != 1){
                throw new JMSException("Error rectificando el saldo");
            }

        } else {
            logger.warning(
                "Message of wrong type: "
                + inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        e.printStackTrace();
    }
}

```

## **EJERCICIO 12:**

En este ejercicio hemos desarrollado los dos principales método para que el cliente pueda localizar la cola de mensajes: anotaciones estáticas y mediante la búsqueda dinámico del recurso mediante la API JNDI (comentado en el código), además se ha procedido a implementar el código para el envío de mensaje así como el cierre de la sesión y conexiones:

```
try {  
    @Resource(mappedName = "jms/VisaConnectionFactory")  
    private static ConnectionFactory connectionFactory;  
  
    @Resource(mappedName = "jms/VisaPagosQueue")  
    private static Queue queue;  
  
    //Metodo clase InitialContext JNDI  
    //InitialContext jndi = new InitialContext();  
  
    //connection = (ConnectionFactory)jndi.lookup("jms/NombreDeLaConnectionFactory");  
    //queue = (Queue)jndi.lookup("jms/VisaPagosQueue");  
  
    connection = connectionFactory.createConnection();  
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);  
    if (args[0].equals("-browse")) {  
        browseMessages(session);  
    } else {  
        // TODO: Enviar argv[0] como mensaje de texto  
        messageProducer = session.createProducer(queue);  
        textMessage = session.createTextMessage();  
  
        textMessage.setText(argv[0]);  
        messageProducer.send(textMessage);  
    }  
} catch (Exception e) {  
    System.out.println("Excepcion : " + e.toString());  
} finally {  
    if (connection != null) {  
        try {  
            messageProducer.close();  
            session.close();  
            connection.close();  
        } catch (JMSException e) {  
        }  
    } // if  
  
    System.exit(0);
```

### Ventajas e inconvenientes (JMS estáticos vs JMS dinámicos API JNDI):

Inconveniente de los recursos JMS estáticos es que el nombre hay que conocerlo en tiempo de compilación, por lo tanto si se cambia el recurso, habría que modificar el código y no proporciona transparencia.

Una ventaja de los JMS estáticos es que no es necesario compilar el código de nuevo si se desea cambiar el recurso.

Por otro lado los JMS dinámicos necesitan mayor número de operaciones para conectarse, lo que podría producir mayor números de errores y mayor complejidad. Además esto podría afectar al rendimiento, siendo más rápida la conexión estática que la dinámica. Por lo tanto el estático sería menos complejo si se conociera el nombre y menos costosa.

## EJERCICIO 13:

Hemos modificado en jms.properties y build.properties con el nombre de los recursos de la factoría de conexión y de la cola con “jms/VisaConnectionFactory” y “jms/VisaPagosQueue”. Por último modificamos las ip del cliente con 10.1.4.1 y la del servidor con 10.1.4.2. Por último el host de la mdb será el servidor 10.1.4.2.

```
jms.properties
as.home=${env.J2EE_HOME}
as.lib=${as.home}/lib
as.user=admin
as.host.client=10.1.4.1
as.host.server=10.1.4.2
as.port=4848
as.passwordfile=${basedir}/passwordfile
as.target=server
jms.factoryname=jms/VisaConnectionFactory
jms.name=jms/VisaPagosQueue
jms.physname=VisaPagosQueue

build.properties
as.user=admin
as.host.mdb=10.1.4.2
as.port=4848
```

Tras esto desplegamos la aplicación P1-ejb-transaccional y borramos los recursos creados en los ejercicios 9 y 10. Por último hacemos “ant todo” y generamos los recursos. Se puede comprobar desde la aplicación de administración que se han creado correctamente.

A continuación se puede ver la factoría de conexión:

Home About...

User: admin Role: domain1 Server: 10.1.4.2

GlassFish™ Server Open Source Edition

Total # of available updates : 1

Tree

- Common Tasks
- Domain
- server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications
- Lifecycle Modules
- Monitoring Data
- Resources
  - Concurrent Resources
  - Connectors
  - JDBC
  - JMS Resources
    - Connection Factories
    - Destination Resources
- Configurations
  - default-config
  - server-config
  - Update Tool

Edit JMS Connection Factory

Editing a Java Message Service (JMS) connection factory also modifies the associated connector connection pool and connector resource.

Load Defaults

**General Settings**

JNDI Name: jms/VisaConnectionFactory

Logical JNDI Name: javax.jms.QueueConnectionFactory

Resource Type: javax.jms.QueueConnectionFactory

Description:

Status:  Enabled

**Pool Settings**

Initial and Minimum Pool Size: 1 Connections

Minimum and initial number of connections maintained in the pool

Maximum Pool Size: 250 Connections

Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: 2 Connections

Number of connections to be removed when pool idle timeout expires

Idle Timeout: 300 Seconds

Maximum time that connection can remain idle in the pool

Max Wait Time: 60000 Milliseconds

Amount of time caller waits before connection timeout is sent

On Any Failure:  
 Close All Connections  
Close all connections and reconnect on failure, otherwise reconnect only when used

Transaction Support:

Level of transaction support. Overwrite the transaction support attribute in the Resource Adapter in a downward compatible way.

Connection Validation:  
 Required  
Validate connections, allow server to reconnect in case of failure

A continuación se muestra la cola de mensajes:

The screenshot shows the GlassFish administration interface. On the left, there's a tree view of the server configuration. The 'Destination Resources' node under 'JMS Resources' is selected. In the main panel, a dialog titled 'Edit JMS Destination Resource' is open. It contains fields for 'JNDI Name' (set to 'jms/VisaPagosQueue'), 'Physical Destination Name' (set to 'VisaPagosQueue'), 'Resource Type' (set to 'javax.jms.Queue'), 'Deployment Order' (set to '100'), and 'Status' (with the checkbox checked). Below these fields is a table titled 'Additional Properties (0)' with one entry: 'Name' (VisaPagosQueue), 'Value' (javax.jms.Queue), and 'Description' (None). At the bottom right of the dialog are 'Save' and 'Cancel' buttons.

Si miramos el fichero jms.xml podemos ver el siguiente formato para crear colas de mensajes.

```
<target name="create-jms-resource"
      description="creates jms destination resource">
    <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="--property ${jms.resource.property}" />
        <arg line="${jms.resource.name}" />
    </exec>
</target>
```

Si sustituimos las variables nos queda el siguiente comando:

"asadmin --user admin --passwordfile directoriobase/passwordfile --host 10.1.4.2 --port 4848 create jms-resource --restype javax.jms.Queue --enabled=true --property Name=VisaPagosQueue jms/VisaPagosQueue"

## EJERCICIO 14:

Dentro del main de VisaQueueMessageProducer.java hemos cambiado el código del main para que reciba el argumento con args[0] tal y como se indica en el apéndice y el enunciado.

```

// TODO: Enviar argv[0] como mensaje de texto
messageProducer = session.createProducer(queue);
message = session.createTextMessage();
message.setText(args[0]);
messageProducer.send(message);
messageProducer.close();
}

```

Lo siguiente es deshabilitar la aplicación P1-jms-jms y modificar la variable con la ip del servidor que es 10.1.4.2.

The screenshot shows two consecutive screenshots of the GlassFish Administration Console.

**Screenshot 1: Edit Application**

- User: admin | Domain: domain1 | Server: 10.1.4.2
- GlassFish™ Server Open Source Edition
- Left sidebar: Common Tasks, Applications (selected), P1-ejb, P1-jms-mdb (selected), Lifecycle Modules, Monitoring Data, Resources, Concurrent Resources, Connectors, JDBC.
- Right panel: General tab selected. Edit Application form:
  - Name: P1-jms-mdb
  - Status:  Enabled (unchecked)
  - Implicit CDI:  Enabled (checked)
  - Location: \${com.sun.aas.instanceRootURI}/applications/P1-jms-mdb/
  - Deployment Order: 100
  - Libraries: (empty input field)
  - Description: (empty input field)

**Screenshot 2: Edit JMS Host**

- User: admin | Role: domain1 | Server: 10.1.4.2
- GlassFish™ Server Open Source Edition
- Left sidebar: Total # of available updates : 1, JNDI, JavaMail Sessions, Resource Adapter Configs, Configurations (selected), default-config, server-config (selected), Admin Service, Connector Service, EJB Container, HTTP Service, JVM Settings, Java Message Service (selected), JMS Hosts (selected), default\_JMS\_host (selected), Logger Settings, Monitoring, Network Config, ORB.
- Right panel: Edit JMS Host form:
  - Configuration Name: server-config
  - Name: default\_JMS\_host
  - Host: 10.1.4.2
  - Port: \${JMS\_PROVIDER\_PORT}
  - Admin Username: \* admin
  - Admin Password: \* (redacted)
  - Confirm New Password: \* (redacted)

Ejecutamos los comandos que nos piden para introducir el mensaje en la cola con id de autorización 1:

```

francisco@francisco:~/SI2/P1B/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.4.2 -client dist/clientjms/P1-jms-clientjms.jar 1
mar 22, 2021 12:44:00 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 22, 2021 12:44:00 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
mar 22, 2021 12:44:00 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mar 22, 2021 12:44:00 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
francisco@francisco:~/SI2/P1B/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.4.2 -client dist/clientjms/P1-jms-clientjms.jar -browse
mar 22, 2021 12:44:50 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 22, 2021 12:44:50 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
mar 22, 2021 12:44:50 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mar 22, 2021 12:44:51 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensajes en cola:
1

```

El primer comando envía el mensaje con id de autorización 1 y el segundo muestra los mensajes en cola. Al tener la aplicación JMS deshabilitada nos mostrará el mensaje anterior.

Ahora realizamos un pago con la aplicación con id de autorización 1 como se ve a continuación:

## Pago con tarjeta

### Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Número de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug:  True  False

Direct Connection:  True  False

Use Prepared:  True  False

← → C No es seguro | 10.1.4.2:8080/P1-ejb-cliente/procesapago  
 Gmail  Google  https://moodle...

## Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 1  
 idComercio: 1  
 importe: 100.0  
 codRespuesta: 000  
 idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

pago tarjeta	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
	1	1	000	100	1	2347 4840 5058 7931	21/03/21 16:55

Por último desde la aplicación de administración del admin volvemos a habilitar P1-jms-mdb y vemos que si hacemos el browse de nuevo nos aparece la cola vacía.

```

francisco@francisco:~/SI2/P1B/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -t
argetserver 10.1.4.2 -client dist/clientjms/P1-jms-clientjms.jar -browse
mar 22, 2021 1:54:16 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 22, 2021 1:54:17 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1
(Build 2-c) Compile: March 17 2015 1045
mar 22, 2021 1:54:17 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker
is REMOTE, connection mode is TCP
mar 22, 2021 1:54:17 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Cola de mensajes vacía!
Exception in thread "main" java.lang.NullPointerException
at ssii2.VisaQueueMessageProducer.main(VisaQueueMessageProducer.java:93)

```

Por último podemos ver que se ha gestionado el mensaje de la cola y si accedemos a Tora se puede ver que se le ha devuelto el saldo al usuario y el pago ha cambiado el código de respuesta.

pago	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
tarjeta							
	1 1	1	999	100	1	2347 4840 5058 7931	21/03/21 17: