

UAM UNIVERSIDAD AUTÓNOMA DE MADRID	
Grupo	M2323
Práctica	1b
Fecha	10/03/2023
Alumno/a	Sanchez, Fernandez del Pozo, Pablo
Alumno/a	Pérez de Lema, Diez, Javier



Práctica 1B

○ **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? Comparar esta interfaz con el fichero de configuración del web service implementado en la práctica P1A.

La librería javax.ejb.Local es una librería que incluye las Enterprise Java Bean. Esta sirve para designar como Local una interfaz y definir tanto los métodos como el Bean.

@Local se utiliza para especificar que dicha interfaz es local, aunque esto no es necesario que se especifique si la interfaz por defecto ya es local.

El fichero de configuración del WS de la P1A se implementa en XML.

○ **Ejercicio 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.

Copiamos la clase visaDAO

Realizamos los imports necesarios:

```
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.jws.WebMethod;  
import javax.jws.WebParam;  
import javax.jws.WebService;
```

```
import javax.ejb.Stateless;
```

Creamos la clase VisaDAOBean:

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

Eliminamos el constructor

```
/**  
 * Constructor de la clase  
 */  
public VisaDAO() {  
    return;  
}
```

Cambiamos el método getPagos() a como se encuentra en VisaDAOLocal para tener la misma interfaz:

```
/**  
 * Buscar los pagos asociados a un comercio  
 * @param idComercio  
 * @return  
 */  
public PagoBean[] getPagos(String idComercio) {
```

- **Ejercicio 2:**

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 preprocesso, 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.

Importante: Esta operación deberá ser realizada para todos los servlets del proyecto que hagan uso del

antiguo VisaDAOWS. Verifique también posibles errores de compilación y ajustes necesarios en el código

al cambiar la interfaz del antiguo VisaDAOWS (en particular, el método getPagos()).

En el servlet ProcesaPago (ProcesaPago.java) se han añadido los imports especificados y se ha añadido el objeto proxy con atributo de la clase, anotando como se nos pide.

Finalmente, hemos eliminado la declaración antigua del servicio web antiguo como se nos especifica y hemos eliminado también todos los usos de BindingProvider (se usaba en la función processRequest), así como los imports del mismo.

```

...
package ssii2.controlador;

import java.io.IOException;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.Collections;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
//import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente
//import ssii2.visa.VisaDAO; // Stub generado automáticamente
import ssii2.visa.*;

import javax.xml.ws.WebServiceRef;

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

```

```

117
118     @EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)
119     private VisaDAOLocal dao;
120
121

```

```

149     /
150     @Override
151     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
152     throws ServletException, IOException {
153
154         TarjetaBean tarjeta = creaTarjeta(request);
155         ValidadorTarjeta val = new ValidadorTarjeta();
156         PagoBean pago = null;
157         //VisaDAOWS dao = null;
158
159         // printAddresses(request,response);
160         if (! val.esValida(tarjeta)) {
161             request.setAttribute(val.getErrorName(), val.getErrorVisa());
162             reenvia(ruta: "/formdatosvisa.jsp", request, response);
163             return;
164         }
165
166         //try {
167         //    VisaDAOWSService service = new VisaDAOWSService();
168         //    dao = service.getVisaDAOWebService();
169 +     //    BindingProvider bp = (BindingProvider) dao;      You, 19 seconds ago * Uncommitted changes
170         //    bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
171         //        getServletContext().getInitParameter("urlServidorRemoto"));
172         //
173         //} catch (Exception e) {
174         //    enviaError(e, request, response);
175         //    return;
176         //}
177
178         HttpSession sesion = request.getSession(false);
179         if (sesion != null) {
180             pago = (PagoBean) sesion.getAttribute(ComienzaPago.ATTR_PAGO);
181         }
182         if (pago == null) {
183             pago = creaPago(request);
184             boolean isdebug = Boolean.valueOf(request.getParameter("debug"));

```

En el servlet GetPagos (GetPagos.java) se han añadido los imports especificados y se ha añadido el objeto proxy con atributo de la clase, anotando como se nos pide. Finalmente, hemos modificado el método processRequest para que, cuando llame a getPagos, obtenga un array de PagoBean, en vez del ArrayList de la práctica anterior, para que concuerde con la interfaz de la función. También se ha eliminado referencias a BindingProvider.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    /* Se recoge de la petición el parámetro idComercio*/
    String idComercio = request.getParameter(PARAM_ID_COMERCIO);

    /* Petición de los pagos para el comercio */
    //List<PagoBean> pagosList = dao.getPagos(idComercio);
    PagoBean[] pagos = dao.getPagos(idComercio);

    request.setAttribute(ATTR_PAGOS, pagos);
    reenvia(ruta: "/listapagos.jsp", request, response);
    return;
}
```

En el servlet DelPagos (DelPagos.java) se han añadido los imports especificados y se ha añadido el objeto proxy con atributo de la clase, anotando como se nos pide. Finalmente, hemos modificado el método processRequest para eliminar las referencias al servicio antiguo y a BindingProvider.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    You, now + Uncommitted changes
    /* Se recoge de la petición el parámetro idComercio*/
    String idComercio = request.getParameter(PARAM_ID_COMERCIO);

    /* Petición de los pagos para el comercio */
    int ret = dao.delPagos(idComercio);

    if (ret != 0) {
        request.setAttribute(ATTR_BORRADOS, ret);
        reenvia(ruta: "/borradook.jsp", request, response);
    }
    else {
        reenvia(ruta: "/borradoerror.jsp", request, response);
    }
    return;
}
```

○ 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). Explique brevemente el contenido y ponga evidencias en la memoria.

```
<?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>

```

En este archivo se configuran las características de la aplicación más esenciales. Primeramente la versión y codificación del archivo xml. A continuación se especifica la versión de javaee y la localización del espacio de nombres (xmlns). En <display-name> se especifica el título del de la aplicación para las herramientas con interfaz de usuario. En <ejb> se especifica la localización de un módulo de la aplicación (en este caso un ejecutable de java). En <web> especificamos tanto la localización del módulo web de la aplicación <web-uri> (archivo war) como la ruta base de la aplicación web en <context-root>.

El comando jar -tvf listará (-t) los contenidos del archivo (-f) en formato legible (-v).

Al ejecutar jar -tvf ruta/P1-ejb.jar obtenemos esta salida:

```

X 1 ➤ ~/Desktop/Asignaturas/SI2/PRACTICAS/P1-ejb ➤ P1B ±
> jar -tvf dist/server/P1-ejb.jar
  0 Mon Feb 27 12:47:30 CET 2023 META-INF/
  118 Mon Feb 27 12:47:28 CET 2023 META-INF/MANIFEST.MF
    0 Mon Feb 27 12:03:26 CET 2023 ssii2/
    0 Mon Feb 27 12:47:18 CET 2023 ssii2/visa/
  255 Mon Feb 27 12:04:02 CET 2023 META-INF/sun-ejb-jar.xml
  1729 Mon Feb 27 12:03:26 CET 2023 ssii2/visa/DBTester.class
  1464 Mon Feb 27 12:03:26 CET 2023 ssii2/visa/PagoBean.class
   856 Mon Feb 27 12:03:26 CET 2023 ssii2/visa/TarjetaBean.class
  6929 Mon Feb 27 12:47:18 CET 2023 ssii2/visa/VisaDAOBean.class
   593 Mon Feb 27 12:47:18 CET 2023 ssii2/visa/VisaDAOLocal.class
  7451 Mon Feb 27 12:47:18 CET 2023 ssii2/visa/VisaDAOWS.class
%
```

Aquí se pueden ver los distintos directorios que intervienen en la aplicación así como las diferentes clases compiladas y archivos .xml.



○ Ejercicio 3:

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

Sendos archivos se encuentran configurados con las IPs del servidor.

```
nombre=P1-ejb
build=${basedir}/build
build.client=${build}/client
build.server=${build}/server
dist=${basedir}/dist
dist.client=${dist}/client
dist.server=${dist}/server
src=${basedir}/src
src.client=${src}/client
src.server=${src}/server
web=${basedir}/web
conf=${basedir}/conf
conf.server=${conf}/server
conf.application=${conf}/application
paquete=ssii2
war=${nombre}-cliente.war
jar=${nombre}.jar
ear=${nombre}.ear
asadmin=${as.home}/bin/asadmin
as.home=${env.J2EE_HOME}
as.lib=${as.home}/lib
as.user=admin
as.host=10.6.3.1
as.port=4848
as.passwordfile=${basedir}/passwordfile
as.target=server
```

```
# Parametros propios de postgresql
db.name=visa
db.user=alumnodb
db.password=*****
db.port=5432
db.host=10.6.3.1
# 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=10.6.3.1
db.client.port=4848

db.delimiter=;
db.driver=org.postgresql.Driver
db.datasource=org.postgresql.ds.PGConnectionPoolDataSource
db.vendorname=SQL92

# Herramientas
db.createdb=/usr/bin/createdb
db.dropdb=/usr/bin/dropdb

# Scripts de creacion / borrado
db.create.src=./sql/create.sql
db.insert.src=./sql/insert.sql
db.delete.src=./sql/drop.sql
```

Realizamos el despliegue del servidor con este comando: *ant desplegar*

```
~/Desktop/Asignaturas/SI2/PRACTICAS/P1-ejb ▶ P1B ±
> ant desplegar
Buildfile: /home/pablos/Desktop/Asignaturas/SI2/PRACTICAS/P1-ejb/build.xml

desplegar:
    [exec] Application deployed with name P1-ejb.
    [exec] Command deploy executed successfully.

BUILD SUCCESSFUL
Total time: 7 seconds
```

Tras ingresar en el panel de administrador podemos observar que el servicio se ha creado correctamente, y podemos ingresar en la aplicación:

Select	Name	Deployment Order
<input type="checkbox"/>	P1-ejb	100

Ejercicio 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. 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.

En el ejercicio anterior podemos ver cómo se ha desplegado bajo la ruta /P1-ejb-cliente anteriormente teníamos la base de datos generada por tanto hemos usado la que teníamos generada.

Realizamos el pago con tarjeta:

Pago con tarjeta

Proceso de un pago

Id Transacción: 12
Id Comercio: 123
Importe: 1234
Número de visa: 1111 2222 3333 4444
Titular: Jose Garcia
Fecha Emisión: 11/09
Fecha Caducidad: 11/24
CVV2: 123
Modo debug: True False
Direct Connection: True False
Use Prepared: True False

Pagar

Pago con tarjeta

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

idTransaccion: 12
idComercio: 123
importe: 1234.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Listamos el pago con tarjeta:

Pago con tarjeta

Lista de pagos del comercio 123

idTransaccion	Importe	codRespuesta	idAutorizacion
12	1234.0	000	2

[Volver al comercio](#)

Podemos comprobar que se ha realizado el pago mediante una consulta a la base de datos:

	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
○	2	12	000	1,234	123	1111 2222 3333 4444	2023-03-06 03:07:50.811

Por último borramos el pago :

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 123

[Volver al comercio](#)

- **Ejercicio 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.

Hemos copiado VisaDAOLocal.java bajo el nombre VisaDAORemote.java

Sustituyendo @Local por @Remote

```
@Remote  
public interface VisaDAORemote {
```

Además hemos añadido el siguiente import:

```
import javax.ejb.Remote;
```

También hemos hecho que visaDAOBeans implemente ambas interfaces:

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

Además añadiendo los imports correspondientes:

```
import javax.ejb.Local;  
  
import javax.ejb.Remote;
```

En TarjetaBean y PagoBean hemos implementado la serialización:

```
import java.io.Serializable;  
  
  
public class TarjetaBean implements Serializable {  
  
  
import java.io.Serializable;  
  
  
public class PagoBean implements Serializable {
```

Compilamos y realizamos el despliegue con: *ant desplegar*

```
~/Desktop/Asignaturas/SI2/PRACTICAS/P1-ejb-servidor-remoto ➔ P1B ±  
• > ant desplegar  
Buildfile: /home/pablos/Desktop/Asignaturas/SI2/PRACTICAS/P1-ejb-servidor-remoto/build.xml  
  
desplegar:  
    [exec] Application deployed with name P1-ejb.  
    [exec] Command deploy executed successfully.  
  
BUILD SUCCESSFUL  
Total time: 7 seconds
```

En la consola del administrador podemos ver que se ha creado el servicio:

The screenshot shows the GlassFish administration interface. On the left, there's a navigation tree with sections like Common Tasks, Virtual Servers, Implicit CDI, Java Web Start, and Modules and Components. Under Modules and Components, there's a table titled "Modules and Components (8)". The table has columns for Module Name, Engines, Component Name, Type, and Action. The rows show various components of the P1-ejb application, such as P1-ejb-cliente.war, P1-ejb-cliente.war, P1-ejb-cliente.war, P1-ejb-cliente.war, P1-ejb-cliente.war, P1-ejb-cliente.war, P1-ejb-cliente.war, and P1-ejb.jar. The last row is VisaDAOBean, which is a StatelessSessionBean.

○ Ejercicio 6:

Realizar los cambios comentados en la aplicación P1-base para convertirla en P1-ejb-cliente-remoto. 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.

Hemos importado java.io.Serializable tanto en PagoBean.java como en TarjetaBean.java, y hemos hecho que ambas implementen la interfaz Serializable, de la misma forma que hicimos en el ejercicio anterior.

The image shows two code snippets. The left snippet is for PagoBean.java, which implements Serializable and includes annotations @author jaime. The right snippet is for TarjetaBean.java, which also implements Serializable.

```

import java.io.Serializable;

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

package ssii2.vista;
import java.io.Serializable;

public class TarjetaBean implements Serializable {
}

```

Después, eliminamos las declaraciones de VisaDAO dao en los 3 servlets del cliente.

Tras importar EJB y VisaDAORemote, hemos declarado de nuevo la variable dao pero utilizando la interfaz que hemos declarado (VisaDAORemote).

The image shows a code snippet where the DAO interface is declared using the @EJB annotation. It specifies the name of the bean as "VisaDAOBean" and the interface as VisaDAORemote.class.

```

24
25     import javax.ejb.EJB;
26     import ssii2.vista.VisaDAORemote;
27
28     /**

```

```

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

```

Finalmente, hemos creado el fichero glassfish-web.xml y hemos puesto la dirección 10.6.3.1 para indicarle que esta es la dirección al servidor, y hemos modificado los ficheros build.properties y postgresql.properties para indicar que el cliente se alojará en la dirección 10.6.3.2.

10.6.3.1 -> IP servidor remoto

10.6.3.2 -> IP cliente remoto

En glassfish-web.xml:

`corbaname:iiop:10.6.3.1:3700#java:global/P1-ejb/P1-ejb/VisaDAOBean!ssii2.vista.VisaDAORemote</jndi-name>`

```

postgresql.properties
1 # Propiedades de la BD postgresql
2
3 # Parametros propios de postgresql
4 db.name=visa
5 db.user=alumnodeb
6 db.password=*****
7 db.port=5432
8 db.host=10.6.3.2
9 # Recursos y pools asociados
10 db.pool.name=VisaPool
11 db.jdbc.resource.name=jdbc/VisaDB
12 db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
13 db.client.host=10.6.3.2
14 db.client.port=4848
15

```

```

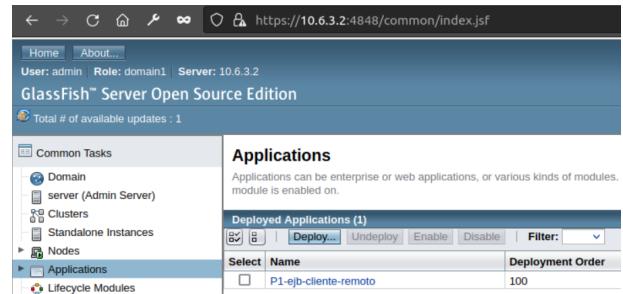
13
14
15 paquete=ssii2
16 war=${nombre}.war
17
18
19
20 asadmin=${as.home}/bin/asadmin
21 as.home=${env.J2EE_HOME}
22 as.lib=${as.home}/lib
23 as.user=admin
24 as.host=10.6.3.2
25
26 as.port=4848
27 as.passwordfile=${basedir}/passwordfile
28 as.target=server

```

Compilamos el cliente y lo lanzamos, podemos ver en el a consola del administrador que la aplicación está desplegada:

A continuación realizamos una transacción conectándonos al cliente remoto:

(Nótese que la transacción se ejecuta desde la ip del cliente remoto).



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: 12
 idComercio: 123
 importe: 1234.0
 codRespuesta: 000
 idAutorizacion: 2

[Volver al comercio](#)

Pago con tarjeta

Lista de pagos del comercio 123

idTransaccion	Importe	codRespuesta	idAutorizacion
12	1234.0	000	2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

SQL Enter a SQL expression to fetch results (use Ctrl+Space)

	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta
1	12		000	1,234	123	1111 2222 3333 4444

Ejercicio 7:

Modificar la aplicación VISA para soportar el campo saldo:

Archivo TarjetaBean.java:

Añadir el atributo saldo y sus métodos de acceso:

Hemos creado dicho atributo y sus métodos setter y getter:

```
private String numero;
private String titular;
private String fechaEmision;
private String fechaCaducidad;
private String codigoVerificacion; /* CVV2
private double saldo;

/**
 * Devuelve el saldo de la tarjeta
 * @return saldo de la tarjeta
 */
public double getSaldo() {
    return saldo;
}

/**
 * Establece el saldo de la tarjeta
 * @param titular
 */
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.

Hemos importado la excepción EJBException y hemos creado una consulta que utilizan prepared statement para obtener el saldo asociado a una tarjeta:

```
62
63     private static final String SELECT_SALDO_QRY =
64             "select saldo from tarjeta " +
65             "where numeroTarjeta=?";
66
```

- Declarar un prepared statement para actualizar el nuevo saldo calculado en la base de datos.

Hemos creado una consulta que utilizan prepared statement para actualizar el saldo asociado a una tarjeta:

```
66
67     private static final String UPDATE_SALDO_QRY =
68             "update tarjeta set saldo=?"
69             "where tarjeta.numeroTarjeta=?";|
70     /*****
```

- Modificar el método realizaPago con las siguientes acciones:

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

- ❖ 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`)
 - ❖ 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.
 - ❖ 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).
 - ❖ 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.

Hemos cambiado el método realizaPago. Al abrir una nueva conexión, preparamos el prepared statement pasándole el número de tarjeta de la que comprobar el saldo como parámetro. Dado que esta consulta retorna un resultado, lo guardaremos en rs y comprobaremos su valor. Si su valor es mayor que el importe del pago entonces procedemos a actualizar el saldo de la tarjeta utilizando la otra consulta creada anteriormente.

En caso de que el pago sea mayor que el saldo, se devuelve nulo y se establece el id de autorización también a nulo.

Se tratan las excepciones en caso de que haya ocurrido alguna. Al final, si ha ocurrido algún problema se lanzará EJBException, y sino se continuará registrando el pago en la base de datos.

Si durante ese proceso ocurriese otro problema también lanzaremos esta excepción.

```
try {
    // Obtener conexion
    con = getConnection();

    String getSaldo = SELECT_SALDO_QRY;
    errorLog(getSaldo);
    pstmt = con.prepareStatement(getSaldo);
    pstmt.setString(1, pago.getTarjeta().getNumero());

    rs = pstmt.executeQuery();
    if (rs.next()) {
        double saldo = rs.getDouble("saldo");

        if (saldo >= pago.getImporte()){
            String insert = UPDATE_SALDO_QRY;
            errorLog(insert);
            pstmt = con.prepareStatement(insert);
            pstmt.setDouble(1, saldo-pago.getImporte());
            pstmt.setString(2, pago.getTarjeta().getNumero());

            ret = false;
            if (!pstmt.execute()
                && pstmt.getUpdateCount() == 1) {
                ret = true;
            }
        } else {
    }
```

```
47                                     pago.setIdAutorizacion(null);
48                                     return null;
49                                 }
50
51             } else {
52                 ret = false;
53             }
54
55         } catch (Exception e) {
56             errorLog(e.toString());
57             ret = false;
58         } finally {
59             try {
60                 if (rs != null) {
61                     rs.close(); rs = null;
62                 }
63                 if (pstmt != null) {
64                     pstmt.close(); pstmt = null;
65                 }
66                 if (pstmt != null) {
67                     pstmt.close(); pstmt = null;
68                 }
69                 if (con != null) {
70                     closeConnection(con); con = null;
71                 }
72             } catch (SQLException e) {
73             }
74         }
75     }
```

```
71         }
72     } catch [SQLException e] {
73     }
74 }
75
76 if(!ret){
77     throw new EJBException();
78 }
79
80 // Registrar el pago en la base de datos
81 try {
82
83     // Obtener conexion
84     con = getConnection();
85
86     // Insertar en la base de datos el pago
87
88     /* TODO Usar prepared statement si
89      | isPrepared() == true */
90     /*****************************************************************/
91     if (isPrepared() == true) {
92         String insert = INSERT_PAGOS_QRY;
93         errorLog(insert);
94         pstmt = con.prepareStatement(insert);
95         pstmt.setString(1, pago.getIdTransaccion());
96         pstmt.setDouble(2, pago.getImporte());
97         pstmt.setString(3, pago.getIdComercio());
98         pstmt.setString(4, pago.getTarjeta().getNumero());
```

- 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).

Hemos importado la excepción EJBException y hemos añadido un try-catch en el if donde se llama a dao.realizaPago. Si se entra en el catch, invalidamos la sesión si existiese y devolvemos la página de error.

```

import javax.ejb.EJB;
import ssii2.visa.VisaDAOLocal;
import javax.ejb.EJBException;
|
/**

    // Almacenamos la tarjeta en el pago
    pago.setTarjeta(tarjeta);

    if (! dao.compruebaTarjeta(tarjeta)) {
        enviaError(new Exception("Tarjeta no autorizada:"), request, response);
        return;
    }

    try {
        pago = dao.realizaPago(pago);

    } catch (EJBException e){
        if (sesion != null) sesion.invalidate();
        enviaError(new EJBException("Pago incorrecto"), request, response);
    }

    if (pago == null) {
        if (sesion != null) sesion.invalidate();
        enviaError(new Exception("Pago incorrecto"), request, response);
        return;
    }
}

```



○ Ejercicio 8:

- 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.

Para comprobar que la transacción funciona, se realiza un pago de 90€ y comprobamos en la base de datos si efectivamente se ha modificado el saldo de la tarjeta:



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: 2222
idComercio: 2222
importe: 90.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

tarjeta					
Properties Data ER Diagram					
tarjeta saldo < 1000					
Grid	Añadir numerotarjeta	Añadir titular	Añadir validadesde	Añadir validahasta	Añadir codigoverificacion
1	1111 2222 3333 4444	Jose Garcia	11/09	11/24	123
					910

- 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.

Repetimos la transacción con el mismo identificador de transacción y de comercio, y obtenemos el mensaje de error:



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 incorrecto

Prácticas de Sistemas Informáticos II

Nótese que no se ha cambiado el valor de saldo en la tarjeta:

tarjeta						
Properties		Data	ER Diagram			
tarjeta saldo <1000						
Id	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1	1111 2222 3333 4444	Jose Garcia	11/09	11/24	123	910

- 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.

Todas las modificaciones se han hecho en el ejercicio 7

○ Ejercicio 9:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.6.3.1), 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.

The screenshot shows the GlassFish Administration Console interface. On the left, there's a navigation tree with options like Home, About..., Domain, Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources (Concurrent Resources, Connectors, JDBC, JMS Resources), and Configurations. The 'JMS Resources' section is expanded, and 'Connection Factories' is selected. A sub-tree shows 'jsm_defaultConnectionFactory' and 'Destination Resources'. The main content area is titled 'New JMS Connection Factory' and contains two tabs: 'General Settings' and 'Pool Settings'. In 'General Settings', the 'JNDI Name' is set to 'jsmVisaConnectionFactory', 'Resource Type' is 'javax.jms.QueueConnectionFactory', 'Description' is 'Factoría de conexiones a la cola de pagos', and 'Status' is checked for 'Enabled'. In 'Pool Settings', various connection pool parameters are configured: Initial and Minimum Pool Size (8), Maximum Pool Size (32), Pool Resize Quantity (2), Idle Timeout (300 seconds), Max Wait Time (60000 milliseconds), and Transaction Support (None). There's also an 'On Any Failure' section with a checkbox for 'Close All Connections'. At the bottom, there's an 'Additional Properties (0)' section with an 'Add Property' button and a table for properties. The table has columns for Select, Name, Value, and Description, with a note stating 'No items found.'

○ Ejercicio 10:

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

Incluya en la clase VisaCancelacionJMSBean:

- Consulta SQL necesaria para obtener el código de respuesta del pago cuyo idAutorizacion coincide con lo recibido por el mensaje
- Consulta SQL necesaria para actualizar el código de respuesta a valor 999, de aquella autorización existente en la tabla de pagos cuyo idAutorizacion coincide con lo recibido por el mensaje
- Consulta SQL necesaria para rectificar el saldo de la tarjeta que realizó el pago
- Método onMessage() que obtenga el idAutorización de la cola de mensajes, compruebe si el pago con dicho idAutorizacon tiene código de respuesta 000 y en ese caso actualice el código de respuesta y rectifique el saldo de la tarjeta. 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.
- Control de errores en el método onMessage y cierre de conexiones.

Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas

The screenshot shows the GlassFish Administration Console interface. The left sidebar navigation tree is expanded to show the 'Resources' section, specifically the 'JMS Resources' and 'Destination Resources' sub-sections. The main content area is titled 'New JMS Destination Resource'. It contains fields for 'JNDI Name' (set to 'jms/VisaPagosQueue'), 'Physical Destination Name' (set to 'VisaPagosQueue'), 'Resource Type' (set to 'javax.jms.Queue'), 'Description' (set to 'Cola Pagos VISA'), and 'Status' (checkbox checked). Below these fields is a table titled 'Additional Properties (0)' with three columns: 'Select', 'Name', and 'Value'. A note at the top of the form states: 'The creation of a new Java Message Service (JMS) destination resource also creates an admin object resource.'

○

- **Ejercicio 11:**

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

Añadimos el connection factory, que creamos antes en el ejercicio 9, de nombre jms/VisaConnectionFactory.

```

2   <!DOCTYPE sun-ejb-jar PUBLIC -//Sun Microsystems, Inc.//DTD Applet
3   <sun-ejb-jar>
4     <enterprise-beans>
5       <ejb>
6         <ejb-name>VisaCancelacionJMSBean</ejb-name>
7         <mdb-connection-factory>
8           <jndi-name>jms/VisaConnectionFactory</jndi-name>
9           </mdb-connection-factory>
10      </ejb>
11    </enterprise-beans>
12  </sun-ejb-jar>
```

- **Incluya en la clase VisaCancelacionJMSBean:**

- **Consulta SQL necesaria para obtener el código de respuesta del pago cuyo idAutorizacion coincide con lo recibido por el mensaje.**

La añadimos a VisaCancelacionJMSBean:

```

private static final String SELECT_COD_REAPUESTA =
    "select codrespuesta, importem numerotarjeta  " +
    "from pago " +
    "where idautorizacion=? ";
```

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

La añadimos a VisaCancelacionJMSBean:

```

private static final String UPDATE_CANCELAR_QRY =
    "update pago " +
    "set codrespuesta=999 " +
    "where idautorizacion=? ";
```

- **Consulta SQL necesaria para rectificar el saldo de la tarjeta que realizó el pago.**

La añadimos a VisaCancelacionJMSBean:

```

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

- **Método onMessage() que obtenga el idAutorización de la cola de mensajes, compruebe si el pago con dicho idAutorizaon tiene código de respuesta 000 y en ese caso actualice el código de respuesta y rectifique el saldo de la tarjeta. 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.**

De forma semejante a como lanzábamos los prepared statements en VisaDAOBean, aquí realizamos algo semejante:

```

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

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

            pstmt = con.prepareStatement(SELECT_COD_REAPUESTA);
            pstmt.setInt(1, idAutorizacion);
            rs = pstmt.executeQuery();
            rs.next();

            if (rs.getString("codRespuesta").equals("000")) {
                pstmt = con.prepareStatement(UPDATE_CANCELAR_QRY);
                pstmt.setInt(1, idAutorizacion);
                pstmt.execute();

                pstmt = con.prepareStatement(FIX_SALDO_QRY);
                pstmt.setInt(1, idAutorizacion);
                pstmt.execute();
            }
        } else {
            logger.warning(
                "Message of wrong type: "
                + inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        e.printStackTrace();
        mdc.setRollbackOnly();
    } catch (Throwable te) {
        te.printStackTrace();
    }
    if (con != null) {
        try {
            closeConnection(con);
        } catch (SQLException sqle) {
            sqle.printStackTrace();
        }
        con = null;
    }
}

```

- **Control de errores en el método onMessage y cierre de conexiones. Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas.**

El método no retorna nada, sin embargo si es necesario hacer uso de catch de todas las excepciones para poder gestionarlas correctamente:

```
    }
} catch (JMSException e) {
    e.printStackTrace();
    mdc.setRollbackOnly();
} catch (Throwable te) {
    te.printStackTrace();
}
if (con != null) {
    try {
        closeConnection(con);
    } catch (SQLException sqle) {
        sqle.printStackTrace();
    }
    con = null;
}
```

- **Ejercicio 12:**

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.

Primero implementamos en VisaQueueMessageProducer los recursos JMS estaticos mediante estas anotaciones:

```
public class VisaQueueMessageProducer {  
  
    @Resource(mappedName = "jms/VisaConnectionFactory")  
    private static ConnectionFactory connectionFactory;  
    @Resource(mappedName = "jms/VisaPagosQueue")  
    private static Queue queue;  
  
    // Método de prueba
```

Dejamos comentado el metodo de acceso por la clase InitialContext de la API JNDI.

```
65  
66     try {  
67         // TODO: Inicializar connectionFactory  
68         // y queue mediante JNDI  
69         /*  
70             * InitialContext jndi = new InitialContext();  
71             * connectionFactory =  
72             * (ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");  
73             * queue = (Queue)jndi.lookup("jms/VisaPagosQueue");  
74         */  
75     }
```

- **Ejercicio 13:**

Automaticé 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.

Después de borrar manualmente en la consola de administración la cola y la factoría de conexiones creadas en los ejercicios 9 y 10, modificamos los campos de configuración en build.properties y jsm.properties y ejecutamos ant todo, compila y se despliega correctamente:

```
jms.properties
properties
    as.home=${env.J2EE_HOME}
    as.lib=${as.home}/lib
    as.user=admin
    as.host.server=10.6.3.1
    as.port=4848
    as.passwordfile=${basedir}/passwordfile
    as.target=server
    jms.factoryname=jms/VisaConnectionFactory
    jms.name=jms/VisaPagosQueue
    jms.physname=Visa
```

Se puede comprobar a continuación los nuevos recursos creados y el nuevo servicio:

```
1 # Propiedades de despliegue
2 nombre=P1-jms
3 build=${basedir}/build
4 build.clientjms=${build}
5 build.mdb=${build}/mdb
6 dist=${basedir}/dist
7 dist.clientjms=${dist}/dist
8 dist.mdb=${dist}/mdb
9 src=${basedir}/src
10 src.clientjms=${src}/clientjms
11 src.mdb=${src}/mdb
12 web=${basedir}/web
13 conf=${basedir}/conf
14 conf.mdb=${conf}/mdb
15 paquete=ssii2
16 clientjms-jar=${nombre}.jar
17 mdb-jar=${nombre}-mdb.jar
18 asadmin=${as.home}/bin/asadmin
19 as.home=${env.J2EE_HOME}
20 as.lib=${as.home}/lib
21 as.user=admin
22 as.host.mdb=10.6.3.1
23 as.port=4848
24 as.passwordfile=${basedir}/passwordfile
25 as.target=server
26
```

Deployment		
Select	Name	Deployment
<input type="checkbox"/>	P1-ejb	100
<input type="checkbox"/>	P1-jms-mdb	100

Destination Resources (1)		
Select	JNDI Name	
<input type="checkbox"/>	jms/VisaPagosQueue	

Connection Factories (2)		
Select	JNDI Name	
<input type="checkbox"/>	jms/_defaultConnectionFactory	
<input type="checkbox"/>	jms/VisaConnectionFactory	

Revisando el jms.xml el comando equivalente a la creación de una cola JMS es el siguiente:

```
<!-- ... -->
<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>
```

En los argumentos se especifica el nombre (definido en jms.properties) y tipo del recurso, que en este caso es una cola:

```
<antcall target="create-jms-resource">
    <param name="jms.restype" value="javax.jms.Queue" />
    <param name="jms.resource.property" value="Name=${jms.physname}" />
    <param name="jms.resource.name" value="${jms.name}" />
</antcall>
```

- **Ejercicio 14:**

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);

    message = session.createTextMessage();
    message.setText(args[0]);
    messageProducer.send(message);
}
```

Para enviar mensajes a la cola de mensajes usando el cliente se debe usar el comando:

```
<dir-glassfish>/glassfish/bin/appclient -targetserver 10.6.3.1 -client
dist/clientjms/P1-jms-clientjms.jar idAutorizacion
```

Donde 10.6.3.1 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.

Para verificar el contenido de la cola se debe ejecutar:

```
<dir-glassfish>/glassfish/bin/appclient -targetserver 10.6.3.1 -client dist/clientjms/P1-jms-clientjms.jar -browse
```

Lo primero que debemos hacer es cambiar el host por default de la JMS en la consola de administración, poniendo la dirección IP de la máquina con el servidor de aplicaciones, y después la reiniciamos:

The screenshot shows the 'Edit JMS Host' configuration page in the GlassFish Administration Console. The configuration name is 'server-config'. The host is set to '10.6.3.1'. The port is '\$JMS_PROVIDER_PORT'. The admin username is 'admin' and the admin password is '*****'. The configuration name is 'server-config'. The left sidebar shows the navigation tree with 'JMS Hosts' selected under 'Resources'.

Desactivamos el servicio para que no procese peticiones:

The screenshot shows the 'Applications' section in the GlassFish Administration Console. It displays two deployed applications: 'P1-ejb' and 'P1-jms-mdb'. Both are marked as 'Selected application(s)'. The left sidebar shows the navigation tree with 'Applications' selected under 'Resources'.

Ejecutamos el siguiente comando:

```
scp dist/clientjms/P1-jms-clientjms.jar si2@10.3.1.1:/tmp
```

Ejecutamos el siguiente comando:

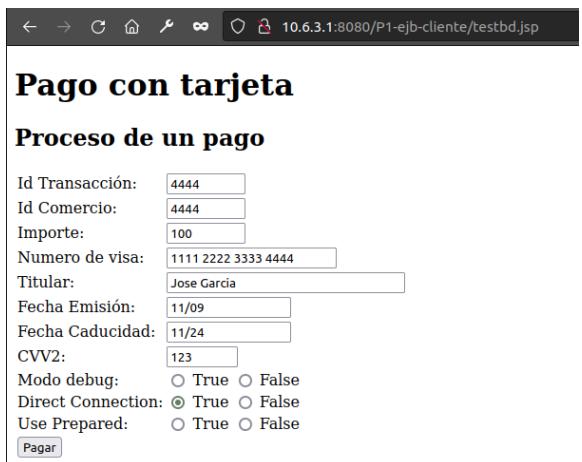
```
> si2@si2srv01:~$ /opt/glassfish-4.1.2/glassfish/bin/appclient -targetserver 10.3.1.2 -client /tmp/P1-jms-clientjms.jar 1
Mar 11, 2023 2:15:31 PM org.hibernate.validator.internal.util.Version <clinit> INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 11, 2023 2:15:31 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
Mar 11, 2023 2:15:31 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO TE, connection mode is TCP
Mar 11, 2023 2:15:31 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started: REMOTE
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle/
```

Como desactivamos la aplicación P1-jms-mdb, no se procesan los mensajes de la cola. Por lo tanto, con el siguiente comando podemos ver que el 1 enviado anteriormente está en la cola de mensajes:

```
> si2@si2srv01:~$ /opt/glassfish-4.1.2/glassfish/bin/appclient -targetserver 10.6.3.1 -client /tmp/P1-jms-client jms.jar -browse
Mar 11, 2023 2:15:45 PM org.hibernate.validator.internal.util.Version <clinit> INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 11, 2023 2:15:46 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 11, 2023 2:15:46 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMO TE, connection mode is TCP
Mar 11, 2023 2:15:46 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started: REMOTE Mensajes en cola:
1
```

Volvemos a habilitar la ejecución del MDB y realizamos un pago de 100 euros:

	
--	---

Operación con codrespuesta == 000

	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	1	4444	000	100	4444	1111 2222 3333 4444	2023-03-11

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1	1111 2222 3333 4444	Jose Garcia	11/09	11/24	123	900

A continuación lo cancelamos con el cliente:

```
> si2@si2srv01:~$ /Escritorio/SI2/practicalb/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.6.3.1 -client dist/client jms/P1-jms-client jms.jar 1
mar 11, 2023 12:54:55 PM org.hibernate.validator.internal.util.Version <clinit> INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 11, 2023 12:54:56 PM 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 11, 2023 12:54:56 PM com.sun.messaging.jms.ra.ResourceAdapter start INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
mar 11, 2023 12:54:56 PM com.sun.messaging.jms.ra.ResourceAdapter start INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started: REMOTE Envío el siguiente mensaje: 1
```

Y como observamos en la base de datos tanto el codRespuesta como el saldo de la tarjeta se han modificado satisfactoriamente:

	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	1	4444	999	100	4444	1111 2222 3333 4444	2023-03-11

	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1	1111 2222 3333 4444	Jose Garcia	11/09	11/24	123	1,000