		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2401	Práctica	1B	Fecha	13/03/2022
Alumno/a	Villa, Rodríguez, Juan Carlos				
Alumno/a	Sánchez, Signorini, Martín				

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 interfaz VisaDAOLocal especifica los métodos que VisaDAOBean implementará, en ellos se encuentran reflejados las funciones que en la práctica 1A se ofrecían como servicio web, como puede verse en VisaDAOWS.. Así pues, en esta interfaz aparecen los mismos métodos que previamente en VisaDAOWS fueron especificados mediante anotaciones como WebServices.

Las librerías Java EE importadas son solamente:

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

Esta se usa para poder utilizar la anotación:

```
@Local
```

La anotación @Local utilizada se pone para especificar que la interfaz se usará para un cliente local de manera que, para el cliente local, la localización no será transparente pero se mejorará el rendimiento frente a llamadas remotas.

Ejercicio 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. Modificar el servlet ProcesaPago para que acceda al EJB local

```
import javax.ejb.Stateless;

/**
 * @author jaime
 */
@Stateless(mappedName="VisaDAOBean")
public class VisaDAOBean extends DBTester implements VisaDAOLocal {
    private boolean debug = false;
```

Fig. 1: importamos y convertimos en un EJB stateless mediante la anotación.

```

/**
 * Buscar los pagos asociados a un comercio
 * @param idComercio
 * @return
 */
@WebMethod(operationName = "getPagos")
public PagoBean[] getPagos(@WebParam(name = "idComercio") String idComercio) {

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

    try {

        // Crear una conexion u obtenerla del pool
        pcon = getConnection();
        qry = SELECT_PAGOS_QRY;
        errorLog(qry + "[idComercio=" + idComercio + "]");

        // La preparacion del statement
        // es automaticamente tomada de un pool en caso
        // de que ya haya sido preparada con anterioridad
        pstmt = pcon.prepareStatement(qry);

        pstmt.setString(1, idComercio);
        rs = pstmt.executeQuery();

        pagos = new ArrayList<PagoBean>();

        while (rs.next()) {
            TarjetaBean t = new TarjetaBean();
            PagoBean p = new PagoBean();
            p.setIdTransaccion(rs.getString("idTransaccion"));
            p.setIdComercio(rs.getString("idComercio"));
            p.setImporte(rs.getFloat("importe"));
            t.setNumero(rs.getString("numeroTarjeta"));
            p.setTarjeta(t);
        }
    } catch (SQLException e) {
        errorLog(e.getMessage());
    }

    return pagos.toArray(new PagoBean[pagos.size()]);
}

```

Fig. 2: ajustamos el método, cambiando el tipo del retorno para que no haya problemas con la implementación de la interfaz.

```

        p.setTarjeta(t);
        p.setCodRespuesta(rs.getString("codRespuesta"));
        p.setIdAutorizacion(String.valueOf(rs.getInt("idAutorizacion")));

        pagos.add(p);
    }

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

    // Cerramos / devolvemos la conexion al pool
    pcon.close();

} catch (Exception e) {
    errorLog(e.toString());
} finally {
    try {
        if (rs != null) {
            rs.close(); rs = null;
        }
        if (pstmt != null) {
            pstmt.close(); pstmt = null;
        }
        if (pcon != null) {
            closeConnection(pcon); pcon = null;
        }
    } catch (SQLException e) {
    }
}

return ret;
}

```

Fig. 3: cambiamos el objeto a retornar.

Ejercicio número 2:

Modificar el servlet ProcesaPago para que acceda al EJB local.

Cabe destacar, que todos los cambios que se adjuntan debajo, no solo serán realizados en ProcesaPago, también se harán estos mismos en las clases DelPagos y GetPagos.

```

/* import ssii2.visa.VisaDAOWebService; // Stub generado automáticamente
import ssii2.visa.VisaDAO; // Stub generado automáticamente */
import javax.xml.ws.WebServiceRef;
import javax.xml.ws.BindingProvider;

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

```

Fig. 4: añadimos los imports de clases que se van a utilizar y eliminamos aquellos de las clases que dejan de existir en el proyecto.

```

/**
 * 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;

```

Fig. 5: nuevo atributo de clase que permite acceder al EJB local a través de un objeto proxy.

```

// MODIFIED, Toda esta parte se va fuera en el apartado 1-B
/*
VisaDAOWSService service = new VisaDAOWSService();
VisaDAOWS dao = service.getVisaDAOWSPort();
BindingProvider bp = (BindingProvider) dao;
String remote_server_url = getServletContext().getInitParameter("visadaows");
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, remote_server_url);
*/

```

Fig. 6: eliminamos toda la parte del código debida a la declaración del webservice VisaDAOWS, el código necesario para obtener la referencia remota y todas las referencias a BindingProvider.

```

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

// Use null instead of boolean
pago = dao.realizaPago(pago);
if (pago == null) {
    enviaError(new Exception("Pago incorrecto"), request, response);
    return;
}
*****/

```

Fig. 6.B: quitamos los controles de errores mediante try y catch.

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.

El fichero application.xml es el descriptor de despliegue. En él se especifican los módulos y contenido de la aplicación. En nuestro caso, el fichero comienza detallando las versiones, codificación y demás metadatos del fichero. A continuación, se especifican 2 módulos, el primero contiene el EJB P1-ejb.jar y el segundo es un módulo web (P1-ejb-cliente.war) donde se especifica la dirección raíz (/P1-ejb-cliente).

Viendo los ficheros en el directorio dist, encontramos que el fichero P1-ejb.ear contiene los datos de la aplicación incluyendo los otros dos ficheros .war y .jar. El fichero client/P1-ejb-cliente.war contiene las clases y datos necesarios para el cliente de la aplicación, en particular los ficheros empleados para la parte de la vista y controlador, como los .jsp. Finalmente, el fichero server/P1-ejb.jar contiene las clases de la parte del servidor, incluyendo todo lo respectivo a Visa, es decir el modelo. A su vez, cada uno contiene su metainformación correspondiente.

```

martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-ejb/dist$ jar -tvf P1-ejb.ear
0 Wed Mar 09 15:32:42 CET 2022 META-INF/
125 Wed Mar 09 15:32:40 CET 2022 META-INF/MANIFEST.MF
508 Tue Mar 08 17:28:16 CET 2022 META-INF/application.xml
20976 Wed Mar 09 15:32:42 CET 2022 P1-ejb-cliente.war
6891 Wed Mar 09 15:32:40 CET 2022 P1-ejb.jar
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-ejb/dist$ jar -tvf client/P1-ejb-cliente.war
0 Wed Mar 09 15:32:42 CET 2022 META-INF/
125 Wed Mar 09 15:32:40 CET 2022 META-INF/MANIFEST.MF
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/controlador/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/filtros/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/visa/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/visa/error/
0 Wed Mar 09 15:32:26 CET 2022 WEB-INF/lib/
0 Wed Mar 09 15:32:26 CET 2022 error/
2844 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/controlador/ComienzaPago.class
1513 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/controlador/DelPagos.class
1365 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/controlador/GetPagos.class
4919 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/controlador/ProcesaPago.class
1894 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/controlador/ServletRaiz.class
2608 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/filtros/CompruebaSesion.class
3170 Wed Mar 09 15:32:26 CET 2022 WEB-INF/classes/ssii2/visa/ValidadorTarjeta.class
616 Wed Mar 09 15:32:42 CET 2022 WEB-INF/classes/ssii2/visa/error/ErrorVisa.class
198 Wed Mar 09 15:32:42 CET 2022 WEB-INF/classes/ssii2/visa/error/ErrorVisaCVV.class
209 Wed Mar 09 15:32:42 CET 2022 WEB-INF/classes/ssii2/visa/error/ErrorVisaFechaCaducidad.class
207 Wed Mar 09 15:32:42 CET 2022 WEB-INF/classes/ssii2/visa/error/ErrorVisaFechaEmision.class
201 Wed Mar 09 15:32:42 CET 2022 WEB-INF/classes/ssii2/visa/error/ErrorVisaNumero.class
202 Wed Mar 09 15:32:42 CET 2022 WEB-INF/classes/ssii2/visa/error/ErrorVisaTitular.class
5609 Wed Mar 09 15:32:26 CET 2022 WEB-INF/web.xml
455 Wed Mar 09 15:32:26 CET 2022 borradoerror.jsp
501 Wed Mar 09 15:32:26 CET 2022 borradook.jsp
509 Wed Mar 09 15:32:26 CET 2022 cabecera.jsp
283 Wed Mar 09 15:32:26 CET 2022 error/muestraerror.jsp
2729 Wed Mar 09 15:32:26 CET 2022 formdatosvisa.jsp
1257 Wed Mar 09 15:32:26 CET 2022 listapagos.jsp
1178 Wed Mar 09 15:32:26 CET 2022 pago.html
1142 Wed Mar 09 15:32:26 CET 2022 pagoexito.jsp
104 Wed Mar 09 15:32:26 CET 2022 pie.html
5011 Wed Mar 09 15:32:26 CET 2022 testbd.jsp
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-ejb/dist$ jar -tvf server/P1-ejb.jar
0 Wed Mar 09 15:32:40 CET 2022 META-INF/
125 Wed Mar 09 15:32:38 CET 2022 META-INF/MANIFEST.MF
0 Wed Mar 09 15:32:26 CET 2022 ssii2/
0 Wed Mar 09 15:32:26 CET 2022 ssii2/visa/
255 Wed Mar 09 15:32:26 CET 2022 META-INF/sun-ejb-jar.xml
1729 Wed Mar 09 15:32:26 CET 2022 ssii2/visa/DBTester.class
1464 Wed Mar 09 15:32:26 CET 2022 ssii2/visa/PagoBean.class
856 Wed Mar 09 15:32:26 CET 2022 ssii2/visa/TarjetaBean.class
7034 Wed Mar 09 15:32:26 CET 2022 ssii2/visa/VisaDAOBean.class
593 Wed Mar 09 15:32:26 CET 2022 ssii2/visa/VisaDAOLocal.class

```

Fig. 7: Contenido de los ficheros .ear, .war y .jar generados.

Ejercicio número 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 por qué 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 por qué son esos valores.

Antes de realizar el despliegue se realizaron las siguientes modificaciones.

En el caso de build.properties: as.host = 10.1.7.2

En el caso de postgresql.properties: db.host = 10.1.7.1 y db.client.host = 10.1.7.2

Una vez realizados estos cambios, procedemos a realizar el despliegue de la aplicación mediante ant desplegar.

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.

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.

Pago con tarjeta

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

idTransaccion: 2
idComercio: 2
importe: 2.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Fig. 8: pago realizado con éxito desde pago.html.

Pago con tarjeta

Lista de pagos del comercio 2

idTransaccion	Importe	codRespuesta	idAutorizacion
2	2.0	000	1

[Volver al comercio](#)

Fig. 9: listado de pagos una vez realizada la transacción.

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 2

[Volver al comercio](#)

Fig. 10: eliminación del pago.

Pago con tarjeta

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

idTransaccion: 3
idComercio: 3
importe: 10.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Fig. 11: pago realizado desde testbd.jsp.

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.

```
import java.io.Serializable;

public class TarjetaBean implements Serializable{
```

Fig. 12: cambios realizados en TarjetaBean.

```
import java.io.Serializable;

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

Fig. 13: cambios realizados en PagoBean.

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.

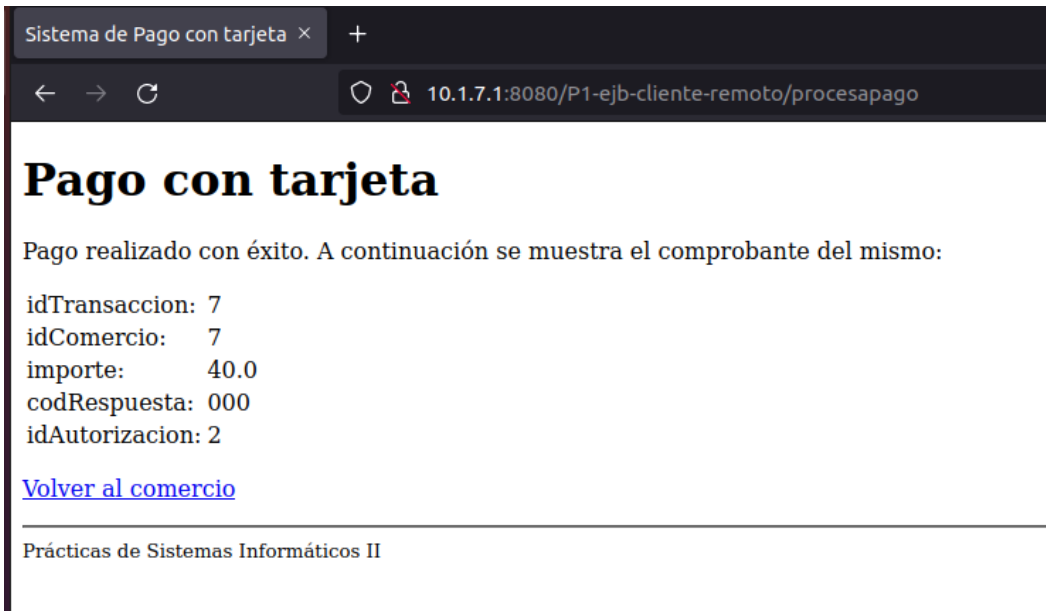


Fig. 14: pago realizado con éxito.

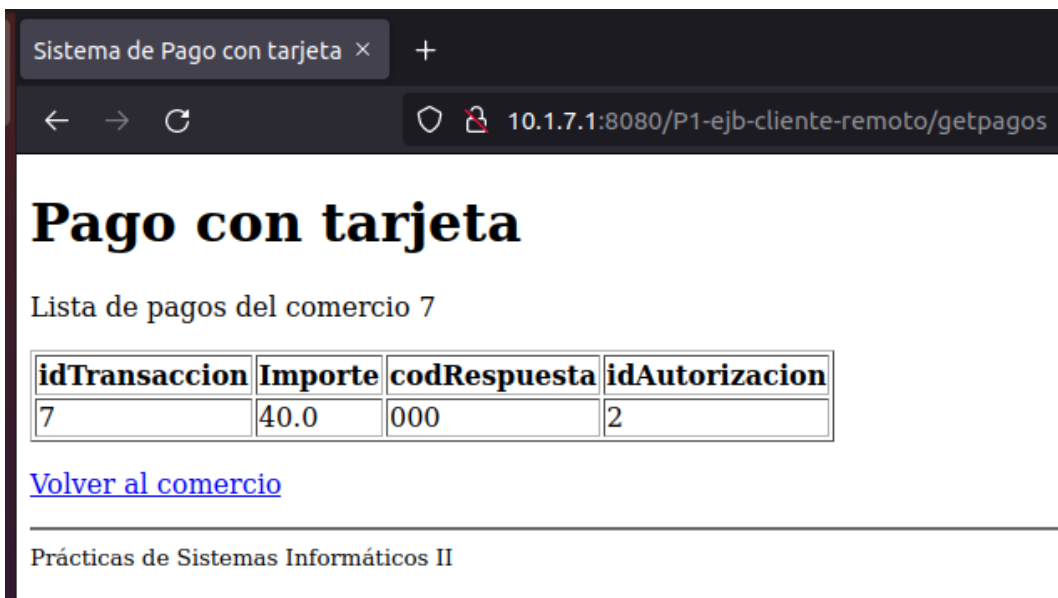


Fig. 14.B: lista actualizada con el pago en cuestión.

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;

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.
- Declarar un prepared statement para actualizar el nuevo saldo calculado en la base de datos.
- 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.
- 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).
- Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas.

```

/*NEW*/
/*****/
private double saldo;

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

/**
 * Establece el saldo de la cuenta
 * @param saldo
 */
public void setSaldo(double saldo) {
    this.saldo = saldo;
}

/*****/

```

Fig. 15: modificaciones realizadas en TarjetaBean.

```

/**
 * Prepared statements para recuperar y actualizar saldo
 * Ejercicio 7
 */
/*****/
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 = ?";
/*****/

```

Fig. 16: prepared statements para actualizar y obtener el saldo de la cuenta asociada a una tarjeta.

```

// Obtener conexion
con = getConnection();

/**
 * Comprobar saldo con statements nuevos y decrementarlo
 * Ejercicio 7
 */
/*****
// Comprobacion del saldo
String saldoqry = SELECT_SALDO_QRY;
double saldo = 0;
errorLog(saldoqry);
pstmt = con.prepareStatement(saldoqry);
pstmt.setString(1, pago.getTarjeta().getNumero());
rs = pstmt.executeQuery();

if(rs.next()) {
    // Saldo encontrado
    saldo = rs.getDouble("saldo");
}
else {
    // Saldo no encontrado
    throw new EJBException("Tarjeta no encontrada.");
}

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

// Decrementar saldo
String actualizasaldoqry = UPDATE_SALDO_QRY;
errorLog(actualizasaldoqry);
pstmt = con.prepareStatement(actualizasaldoqry);
pstmt.setDouble(1, saldo - pago.getImporte());
pstmt.setString(2, pago.getTarjeta().getNumero());
pstmt.executeUpdate();

*****/

```

Figura 17: En VisaDAOBean, en realizaPago realizamos previa a la inserción del pago, la comprobación del saldo y su reducción si es posible.

```

        pago.setIdAutorizacion(String.valueOf(rs.getInt("idAutorizacion")));
        pago.setCodRespuesta(rs.getString("codRespuesta"));
    } else {
        ret = false;
        throw new EJBException("Pago no realizado");
    }
}
else {
    throw new EJBException("Error en la insercion del pago");
}
} catch (EJBException ejb) {
    errorLog(ejb.toString());
    throw ejb;
} catch (Exception e) {
    errorLog(e.toString());
    ret = false;
    throw new EJBException("Pago incorrecto: error en su insercion");
} finally {
    try {

```

Figura 18: En caso de error lanzamos una excepción tipo EJBException.

```

// MODIFIED
// Added new exception managers
/*****/
if (! dao.compruebaTarjeta(tarjeta)) {
    enviaError(new Exception("Tarjeta no autorizada:"), request, response);
    return;
}

// Use null instead of boolean
try{
    pago = dao.realizaPago(pago);
    if (pago == null) {
        if (sesion != null) sesion.invalidate();
        enviaError(new Exception("Pago incorrecto"), request, response);
        return;
    }
} catch (EJBException e){
    errorLog(e.toString());
    if (sesion != null) sesion.invalidate();
    enviaError(e, request, response);
    return;
}

/*****/

request.setAttribute(ComienzaPago.ATTR_PAGO, pago);
if (sesion != null) sesion.invalidate();
reenvia("/pagoexito.jsp", request, response);
return;
}

```

Figura 19: En ProcesaPago añadimos control para las excepciones de tipo EJBException invalidando la sesión en caso de que ocurra un error en realizaPago.

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.
- 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.
- 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.

Pago con tarjeta

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

idTransaccion: 10
idComercio: 10
importe: 100.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Fig. 20: pago realizado con éxito.

0028 1652 2262 7263	Clodoveo Moss Cozar	05/08	08/23	080	900
---------------------	---------------------	-------	-------	-----	-----

Fig. 21: saldo (última columna) actualizado con éxito, el valor inicial de dicho campo era 1000.

Pago con tarjeta

Pago incorrecto: error en su insercion

Fig. 22: pago fallido por ids repetidos.

0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/23	126	1000
---------------------	---------------------	-------	-------	-----	------

Fig. 23: no se ha actualizado la cuenta del usuario.

Ejercicio número 9:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la factoría de conexiones empleando la consola de administración, tal y como se adjunta en la Figura 4.

Incluye una captura de pantalla donde se muestre dicha consola de administración con los cambios solicitados.

New JMS Connection Factory

The creation of a new Java Message Service (JMS) connection factory also creates a connector connection pool for t

General Settings

JNDI Name: *

Resource Type:

Description:

Status: ☒ Enabled

Pool Settings

Initial and Minimum Pool Size: Connections
Minimum and initial number of connections maintained in the pool

Maximum Pool Size: Connections
Maximum number of connections that can be created to satisfy client requests

Pool Resize Quantity: Connections
Number of connections to be removed when pool idle timeout expires

Idle Timeout: Seconds
Maximum time that connection can remain idle in the pool

Max Wait Time: 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 us

Transaction Support:
Level of transaction support. Overwrite the transaction support attribute in the Re

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

Additional Properties (0)

Select	Name	Value
No items found.		

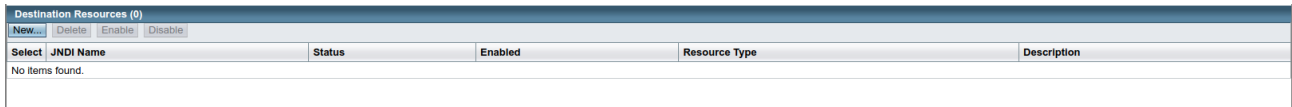
Fig. 24: creación de la factoría.

Connection Factories (2)				
<input type="button" value="New..."/> <input type="button" value="Delete"/> <input type="button" value="Enable"/> <input type="button" value="Disable"/>				
Select	JNDI Name	Logical JNDI Name	Enabled	Resource Type
<input type="checkbox"/>	jms/_defaultConnectionFactory	java:comp/DefaultJMSConnectionFactory	<input checked="" type="checkbox"/>	javax.jms.ConnectionFactory
<input type="checkbox"/>	jms/VisaConnectionFactory		<input checked="" type="checkbox"/>	javax.jms.QueueConnectionFactory

Fig. 25: la nueva factoría aparece creada, es la segunda, la de nombre jms/VisaConnectionFactory.

Ejercicio número 10:

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



Select	JNDI Name	Status	Enabled	Resource Type	Description
No items found.					

Fig. 26: ninguna cola de mensajes está creada.

New JMS Destination Resource

The creation of a new Java Message Service (JMS) destination resource also creates an admin object resource

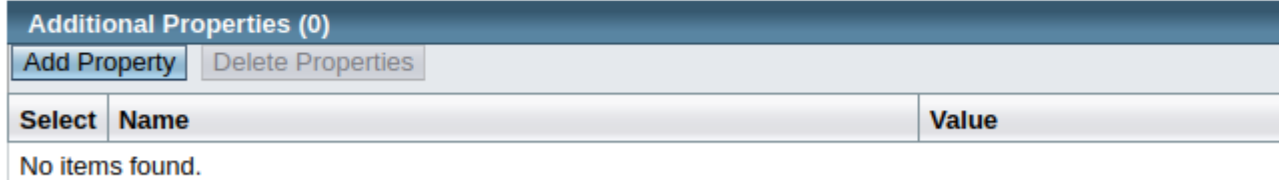
JNDI Name: *

Physical Destination Name: *
Destination name in the Message Queue broker. If the destination does not e

Resource Type: *

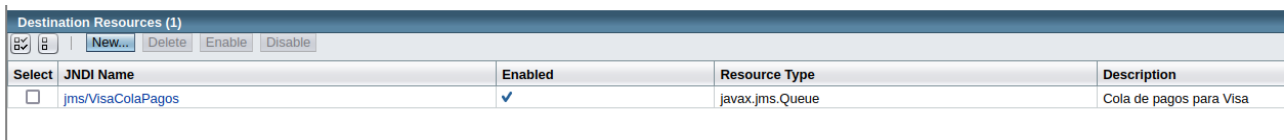
Description:

Status: ☒ Enabled



Select	Name	Value
No items found.		

Fig. 27: información para crear la cola.



Select	JNDI Name	Enabled	Resource Type	Description
<input type="checkbox"/>	jms/VisaColaPagos	<input checked="" type="checkbox"/>	javax.jms.Queue	Cola de pagos para Visa

Fig. 27: cola creada.

Ejercicio número 11:

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 coincida 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 coincida 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.

```
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//  
<sun-ejb-jar>  
  <enterprise-beans>  
    <ejb>  
      <ejb-name>VisaCancelacionJMSBean</ejb-name>  
      <!-- Definir el nombre de la connection factory -->  
      <mdb-connection-factory>  
        <jndi-name>jms/VisaConnectionFactory</jndi-name>  
      </mdb-connection-factory>  
    </ejb>  
  </enterprise-beans>  
</sun-ejb-jar>
```

Fig. 28: Nombre de la connection factory previamente creada.

```
@MessageDriven(mappedName = "jms/VisaColaPagos")  
public class VisaCancelacionJMSBean extends DBTester implements MessageListener {  
    static final Logger logger = Logger.getLogger("VisaCancelacionJMSBean");  
    @Resource  
    private MessageDrivenContext mdc;  
  
    // Definir UPDATE sobre la tabla pagos para poner  
    // codRespuesta a 999 dado un código de autorización  
    private static final String UPDATE_CANCELA_QRY =  
        "update pago " +  
        "set codrespuesta = '999' " +  
        "where idautorizacion = ?";  
  
    // Obtener el código de respuesta del pago cuyo idAutorizacion  
    // coincida con lo recibido por el mensaje  
    private static final String SELECT_CODERES_QRY =  
        "select codrespuesta, importe, numerotarjeta " +  
        "from pago " +  
        "where idautorizacion = ?";  
  
    // Rectificar el saldo de la tarjeta que realizó el pago  
    private static final String UPDATE_TARJETA_QRY = |  
        "update tarjeta " +  
        "set saldo = saldo + ? " +  
        "where numerotarjeta = ?";
```

Fig. 29: SQL para las consultas necesarias y cambio del nombre de la anotación MessageDriven.


```

/*****/
ResultSet rs = null;
Connection con = null;
int idReceived = -1;
String codRespuesta = null;
String numTarjeta = null;
double importe = 0;
PreparedStatement pstmt = null;
/*****/

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

        /*****/
        con = getConnection();

        try {
            idReceived = Integer.parseInt(msg.getText());
        }
        catch (NumberFormatException ne) {
            idReceived = -1;
        }

        // Comprobar codigo de respuesta es 000
        pstmt = con.prepareStatement(SELECT_CODERES_QRY);
        pstmt.setInt(1, idReceived);
        rs = pstmt.executeQuery();
        if(!rs.next()) {
            throw new Exception("ERROR: No existe pago con idAutorizacion = " + idReceived);
        }
        importe = rs.getDouble("importe");
        numTarjeta = rs.getString("numerotarjeta");
        codRespuesta = rs.getString("codrespuesta");
        if(!codRespuesta.equals("000")) {
            throw new Exception("ERROR:Codigo de respuesta del pago distinto de 000");
        }

        // Actualizar codigo de respuesta a 999
        pstmt = con.prepareStatement(UPDATE_CANCELA_QRY);
        pstmt.setInt(1, idReceived);
        if(pstmt.executeUpdate() < 1) {
            throw new Exception("ERROR: No se actualizo ningun pago con idAutorizacion = " + idReceived);
        }

        // Actualizar tarjeta reintegrando el importe
        pstmt = con.prepareStatement(UPDATE_TARJETA_QRY);
        pstmt.setDouble(1, importe);
        pstmt.setString(2, numTarjeta);
        if(pstmt.executeUpdate() < 1) {
            throw new Exception("ERROR: No se actualizo ninguna tarjeta con numero = " + numTarjeta);
        }
        rs.close(); rs = null;
        pstmt.close(); pstmt = null;
        closeConnection(con); con = null;
    }
}
/*****/

```

Fig. 30: Código añadido dentro del método onMessage.

```

    } finally {
        /*****/
        try {
            if (rs != null) {
                rs.close(); rs = null;
            }
            if (pstmt != null) {
                pstmt.close(); pstmt = null;
            }
            if (con != null) {
                closeConnection(con); con = null;
            }
        } catch (SQLException e) {

        }
        /*****/
    }
}

```

Fig. 31: Control de errores dentro del método onMessage.

Ejercicio número 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.

Las ventajas que ofrece obtener dinámicamente la localización de la cola es que no hace falta conocer el nombre en tiempo de compilación y durante la ejecución se puede especificar el nombre de manera dinámica, por ejemplo, usando una variable, lo cual permite mayor flexibilidad. Por otro lado, las anotaciones estáticas requieren conocer de antemano la localización de la cola, pero a cambio son una alternativa más robusta ya que se evitan posibles excepciones en tiempo de ejecución y, a su vez, sobrecarga menos el sistema en la ejecución, ya que no se realiza una búsqueda explícita.

Nótese que se ha cambiado la anotación de la clase VisaCancelacionJMSBean, para utilizar el nombre `jms/VisaColaPagos` en vez de `jms/VisaPagosQueue`. Esto está reflejado en Fig. 29.

```
public class VisaQueueMessageProducer {  
  
    // Anotar los siguientes objetos para  
    // conectar con la connection factory y con la cola  
    // definidas en el enunciado  
    @Resource(mappedName = "jms/VisaConnectionFactory")  
    private static ConnectionFactory connectionFactory;  
  
    @Resource(mappedName = "jms/VisaColaPagos")  
    private static Queue queue;  
}
```

Fig. 32: Modificaciones mediante anotaciones realizadas.

```
try {  
    // Inicializar connectionFactory  
    // y queue mediante JNDI  
    /***/  
    InitialContext jndi = new InitialContext();  
    connectionFactory = (ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");  
    queue = (Queue)jndi.lookup("jms/VisaColaPagos");  
    /***/  
}
```

Fig. 33: Cambios que se realizarían para una conexión dinámica (comentado en el código).

Ejercicio número 13:

Automatice la creación de los recursos JMS (cola y factoría de conexiones) en el `build.xml` y `jms.xml`. Para ello, indique en `jms.properties` los nombres de ambos y el Physical Destination Name de la cola de acuerdo a los valores asignados en los ejercicios 9 y 10. Recuerde también asignar las direcciones IP adecuadas a las variables `as.host.mdb` (`build.properties`) y `as.host.server` (`jms.properties`). ¿Por qué ha añadido esas IPs?

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`.

Las IPs empleadas tanto para el `as.host.mdb` y `a.host.server` son la del servidor de aplicación donde se encuentra desplegado el MDB, en nuestro caso 10.1.7.2. Se emplea esta IP porque para las operaciones que emplean estas variables se debe conocer la dirección en la que reside la cola y factoría de conexiones. En el fichero `jms.xml` se pueden ver los comandos `asadmin` usados para crear la factoría de conexiones y la

cola. En particular, para crear la cola JMS se usa el comando **asadmin create-jms-resource --restype javax.jms.QueueConnectionFactory --enabled=true --property VisaColaPagos jms/VisaColaPagos** junto al resto de opciones de host, contraseña, puerto y usuario.

```
<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>
```

Fig. 34: Extracto del fichero jms.xml donde se especifica el comando para crear colas

Connection Factories (2)				
<div><input checked="" type="checkbox"/> <input type="checkbox"/> <input type="button" value="New..."/> <input type="button" value="Delete"/> <input type="button" value="Enable"/> <input type="button" value="Disable"/></div>				
Select	JNDI Name	Logical JNDI Name	Enabled	Resource Type
<input type="checkbox"/>	jms/_defaultConnectionFactory	java:comp/DefaultJMSConnectionFactory	<input checked="" type="checkbox"/>	javax.jms.ConnectionFactory
<input type="checkbox"/>	jms/VisaConnectionFactory		<input checked="" type="checkbox"/>	javax.jms.QueueConnectionFactory

Fig. 35: Factoría creada automáticamente con ant.

Destination Resources (1)			
<div><input checked="" type="checkbox"/> <input type="checkbox"/> <input type="button" value="New..."/> <input type="button" value="Delete"/> <input type="button" value="Enable"/> <input type="button" value="Disable"/></div>			
Select	JNDI Name	Enabled	Resource Type
<input type="checkbox"/>	jms/VisaColaPagos	<input checked="" type="checkbox"/>	javax.jms.Queue

Fig. 36: Cola creada automáticamente con ant.

Ejercicio número 14:

Modifique el cliente, VisaQueueMessageProducer.java, implementando el envío de args[0] como mensaje de texto.

Fije la variable de la IP del host, y envíe un mensaje a la cola.

Detenga la ejecución del MDB, envíe un mensaje y compruebe el contenido de la cola.

Habilite la ejecución del MDB, realice un pago, compruebe su realización, envíe un mensaje de cancelación del mismo y compruebe su correcta cancelación (reintegro en la tarjeta y código 999).

```
} else {  
    // Enviar argv[0] como mensaje de texto  
    /***/  
    messageProducer = session.createProducer(queue);  
    message = session.createTextMessage();  
  
    message.setText(argv[0]);  
    System.out.println("Enviando el siguiente mensaje: " + message.getText());  
    messageProducer.send(message);  
  
    messageProducer.close();  
    messageProducer = null;  
    session.close();  
    session = null;  
    connection.close();  
    connection = null;  
    /***/  
}
```

Fig. 37: Código para enviar args[0] a la cola de mensajes.

Edit JMS Host

The Java Message Service (JMS) host specifies the system where the JMS service is running.

[Load Defaults](#)

Configuration Name: server-config

Name:	default_JMS_host
Host:	<input type="text" value="10.1.7.2"/> <small>Name or IP address; if name, must contain only alphanumeric, underscore, dash, or dot char</small>
Port:	<input type="text" value="\${JMS_PROVIDER_PORT}"/> <small>Listener port for servicing JMS requests</small>
Admin Username: *	<input type="text" value="admin"/> <small>User name for maintaining the JMS service; can be up to 255 characters, must contain only a</small>
Admin Password: *	<input type="password" value="....."/> <small>Password for JMS administrator</small>
Confirm New Password: *	<input type="password" value="....."/>

Fig. 38: Modificación del atributo host por la IP del servidor de aplicaciones.

Hosts (1)		
<input checked="" type="checkbox"/>	Name	Host
<input type="checkbox"/>	default_JMS_host	10.1.7.2
		Port
		\$(JMS_PROVIDER_PORT)

Fig. 39: Resultado de la modificación.

Comprobación del envío y recepción de mensajes:

```

martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$ /opt/glassfish4/g
lassfish/bin/appclient -targetserver 10.1.7.2 -client dist/clientjms/P1-jms-clie
ntjms.jar 2
Mar 11, 2022 12:18:02 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 11, 2022 12:18:02 AM 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, 2022 12:18:02 AM 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, 2022 12:18:02 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: 2
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$

```

Fig. 40: Envío de un mensaje tras haber modificado default_JMS_host y reiniciado Glassfish.

Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
1405	SEVERE	javax.jms.JMSException: ERROR: No existe pago con idAutorizacion = 2 at ssl2 visa VisaCancelacionJ... (details)		Mar 10, 2022 14:58:24.456	{levelValue=1000, timeMillis=1646953104456}
1404	INFO	MESSAGE BEAN: Message received: 2(details)	VisaCancelacionJMSBean	Mar 10, 2022 14:58:24.455	{levelValue=800, timeMillis=1646953104455}

Fig. 41: Como puede observarse en server.log, se ha recibido el mensaje y procesado correctamente.

Comprobación del listado de mensajes en la cola:

General

Descriptor

Edit Application

Modify an existing application or module.

Name: P1-jms-mdb

Status: ☐ Enabled

Implicit CDI: ☒ Enabled
Implicit discovery of CDI beans

Location: \${com.sun.aas.instanceRootURI}/application:

Deployment Order: 100
A number that determines the loading order c

Libraries:

Description:

Modules and Components (2)	
Module Name	Engines
P1-jms-mdb	[ejb, weld]
P1-jms-mdb	

Fig. 42: Desactivamos el consumo de mensajes

```
glassfish/domains/domain1/logs/
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$ /opt/glassfish4/gla
ssfish/bin/appclient -targetserver 10.1.7.2 -client dist/clientjms/P1-jms-clientjm
s.jar idAutorizacionTest
Mar 11, 2022 12:35:53 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 11, 2022 12:35:53 AM 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, 2022 12:35:53 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE
, connection mode is TCP
Mar 11, 2022 12:35:53 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: idAutorizacionTest
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$ /opt/glassfish4/gla
ssfish/bin/appclient -targetserver 10.1.7.2 -client dist/clientjms/P1-jms-clientjm
s.jar -browse
Mar 11, 2022 12:36:06 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 11, 2022 12:36:07 AM 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, 2022 12:36:07 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE
, connection mode is TCP
Mar 11, 2022 12:36:07 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensajes en cola:
idAutorizacionTest
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$
```

Fig. 43: Enviamos un mensaje y comprobamos que efectivamente está en la cola sin consumir.

Realización de un pago y consecuente cancelación:

Edit Application x Sistema de Pago con tarjeta x General Inform

← → ↻ 10.1.7.2:8080/P1-ejb-cliente/testbd.jsp

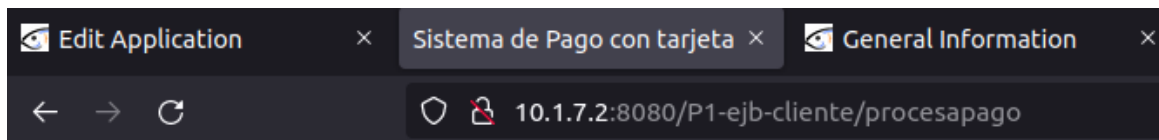
Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="113"/>
Id Comercio:	<input type="text" value="114"/>
Importe:	<input type="text" value="115"/>
Numero de visa:	<input type="text" value="0028 1652 2262 7263"/>
Titular:	<input type="text" value="Clodoveo Moss Cozar"/>
Fecha Emisión:	<input type="text" value="05/08"/>
Fecha Caducidad:	<input type="text" value="08/23"/>
CVV2:	<input type="text" value="080"/>
Modo debug:	<input checked="" type="radio"/> True <input type="radio"/> False
Direct Connection:	<input type="radio"/> True <input checked="" type="radio"/> False
Use Prepared:	<input checked="" type="radio"/> True <input type="radio"/> False

Pagar

Fig. 44: Realizamos el pago desde el cliente.



Pago con tarjeta

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

idTransaccion: 113
 idComercio: 114
 importe: 115.0
 codRespuesta: 000
 idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Fig. 45: Vemos que se realiza correctamente.





Data Output								Explain	Messages	Notifications
	 Idautorizacion [PK] integer	 Idtransaccion character (16)	 codrespuesta character (3)	 Importe double precision	 Idcomercio character (16)	 numerotarjeta character (19)	 fecha timestamp without time zone			
1	1	113	000	115	114	0028 1652 2262 7263	2022-03-10 15:49:57.015287			

Fig. 46: Comprobamos que en la base de datos existe el pago nuevo con código de respuesta 000.

Data Output		Explain	Messages	Notifications			
	numerotarjeta [PK] character (19)	titular character varying (128)	validadesde character (5)	validahasta character (5)	codigoverificacion character (3)	saldo double precision	
1	0004 9839 0829 3274	Blas Avila Sparrow	10/10	04/23	227	1000	
2	0028 1652 2262 7263	Clodoveo Moss Cozar	05/08	08/23	080	885	
3	0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/23	126	1000	
4	0039 2818 1198 8592	Hugo Linus Sparrow	01/10	09/23	971	1000	

Fig. 47: Comprobamos que en la base de datos se ha actualizado el saldo de la tarjeta (885=1000-115)

```
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.7.2 -client dist/clientjms/P1-jms-clientjms.jar 1
Mar 11, 2022 12:56:59 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 11, 2022 12:56:59 AM 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, 2022 12:56:59 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
Mar 11, 2022 12:56:59 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: 1
martin@ss-VivoBook:~/Documents/gits-eps/SI2/practica_1/P1-jms$
```

Fig. 48: Procedemos con el envío del mensaje de cancelación usando la idautorización del pago (1).

	numerotarjeta [PK] character (19)	titular character varying (128)	validadesde character (5)	validahasta character (5)	codigoverificacion character (3)	saldo double precision
1	0004 9839 0829 3274	Blas Avila Sparrow	10/10	04/23	227	1000
2	0028 1652 2262 7263	Clodoveo Moss Cozar	05/08	08/23	080	1000
3	0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/23	126	1000

Fig. 49: Vemos que se ha actualizado la tarjeta reintegrando el importe.

	Idautorizacion [PK] integer	Idtransaccion character (16)	codrespuesta character (3)	Importe double precision	Idcomercio character (16)	numerotarjeta character (19)	fecha timestamp without time zone
1	1	113	999	115	114	0028 1652 2262 7263	2022-03-10 15:49:57.015287

Fig. 50: Vemos que ahora el pago tiene código de respuesta 999.