

Práctica 1 B

Germán Esquinazi Bachoer
Patricia Losana Ferrer

15 de marzo de 2018

Configuración del sistema

Inicialmente debemos configurar algunas cosas. Primero configuramos e iniciamos las 2 máquinas virtuales (una para cliente y la otra para el servidor). Tras esto, nos debemos conectar por ssh a ellas. Para ello primero debemos ejecutar el siguiente comando en terminal:

```
sudo /opt/si2/virtualip.sh eth0
```

Luego nos conectamos con: *ssh si2@10.5.7.1* Una vez conectados, iniciamos el dominio del servidor con el comando: *asadmin start-domain domain1*. Para el cliente haremos lo mismo pero con el valor *ssh si2@10.5.7.2*. Una vez conectados, lanzamos el dominio del cliente con el mismo comando que el servidor: *asadmin start-domain domain1*.

En la máquina host, cambiamos el fichero *build.properties* y, dentro de este, *as.host* por el valor del IP de la máquina virtual (10.5.7.1). A continuación, sustituimos en el fichero *postgresql.properties* los valores de *db.host* y *db.client* por la misma dirección IP. Después creamos la base de datos con el comando *createdb -U alumnodb visa*.

Seguidamente ejecutamos el comando *export J2EE_HOME=/usr/local/glassfish-4.1.1/glassfish/*. Tras compilar con el comando *ant* todo, nos conectamos a la base de datos desde TORA configurándolo de la siguiente manera:

- Como dirección, 10.5.7.1
- Como nombre, visa
- Como puerto, 5432
- Como usuario, alumnodb
- Como contraseña, campo vacío

Para poder ver el contenido de las tablas desde TORA, hemos de seleccionar la opción *schema browser* en *tools* y seleccionar la opción *public*.

Cuestión 1

Abrir el archivo VisaDAOLocal.java y comprobar la definición de dicha interfaz. Anote en la memoria comentarios sobre las librerías Java EE importadas y las anotaciones utilizadas

Como se puede ver en el código, la anotación de la interfaz VisaDAOLocal es @Local. Las librerías importadas a la misma tienen, por un lado, procesamiento de consultas a bases de datos (java.sql), que servirá para poder interactuar con postgres. Por el otro lado, ejb (Enterprise JavaBeans) es la interfaz que define la del cliente.

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

```
@Local
public interface VisaDAOLocal {
```

Ejercicio 1

Hemos eliminado todas las referencias a @webmethod, @param y service dentro de Visadaows.java (a partir de ahora renombrada como VisaDAOBean.java), modificando así su declaración y constructor. Además hemos convertido la clase en un EJB stateless mediante la anotación @Stateless (para hacer esto necesitamos hacer el correspondiente import).

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

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

También hemos eliminado el contenido del constructor por defecto como se pide en el ejercicio, y ajustado los métodos getPagos() a la interfaz definida en VisaDAOLocal.

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

Como la interfaz de VisaDAOLocal trabaja con un PagoBean[] en lugar de con un ArrayList<PagoBean>, por lo que hemos eliminado toda la conversión que hicimos para VisaDAOWS en la práctica 1A.

Ejercicio 2

Modificamos los includes de la clase y añadimos una referencia remota (proxy) a al objeto DAO a través de la anotación @EJB. Además eliminamos todas las antiguas referencias a VisaDAOWS y a BindingProvider. Estos pasos los hemos realizado en los servlets ProcesaPago.java, DelPagos.java y GetPagos.java.

Particularmente en este último, hemos eliminado el ArrayList de PagoBean para que trabaje con el array PagoBean[] por el motivo que explicamos en el apartado anterior.

```
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.*;

import javax.xml.ws.*;
import javax.ejb.EJB;
import ssii2.visa.VisaDAOLocal;

public class ProcesaPago extends ServletRaiz {
    @EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)
    private VisaDAOLocal dao;
```

Cuestión 2

Abrir el archivo application.xml y comprobar su contenido. Verifique el contenido de todos los archivos .jar / .war / .ear que se han construido hasta el momento (empleando el comando jar -tvf). Anote sus comentarios en la memoria

En application.xml se puede ver que se han generado 2 módulos:

P1-ejb.jar, que hace referencia a la aplicación desde el punto de vista del servidor (el que va a tener las funcionalidades de gestión de pagos).

P1-ejb-cliente.war. Como se puede ver en el .xml, este .war está contenido dentro de un tag denominado <web-uri>, lo cual nos da una idea de que estará relacionado con el identificador de recurso universal (es decir, la dirección a la que tendrá que conectarse el cliente para poder llevar a cabo los pagos). Esto es coherente con los URL de las figuras del ejercicio 4.

```
<?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>
```

Ejercicio 3

Hemos modificado los valores de as.host.client y as.host.server del build.properties para que contengan la dirección IP del servidor de aplicaciones (10.5.7.2) y los valores de db.client.host y db.host de psql.properties para que contengan la IP del servidor que contiene postgresql (10.5.7.1).

Ejercicio 4

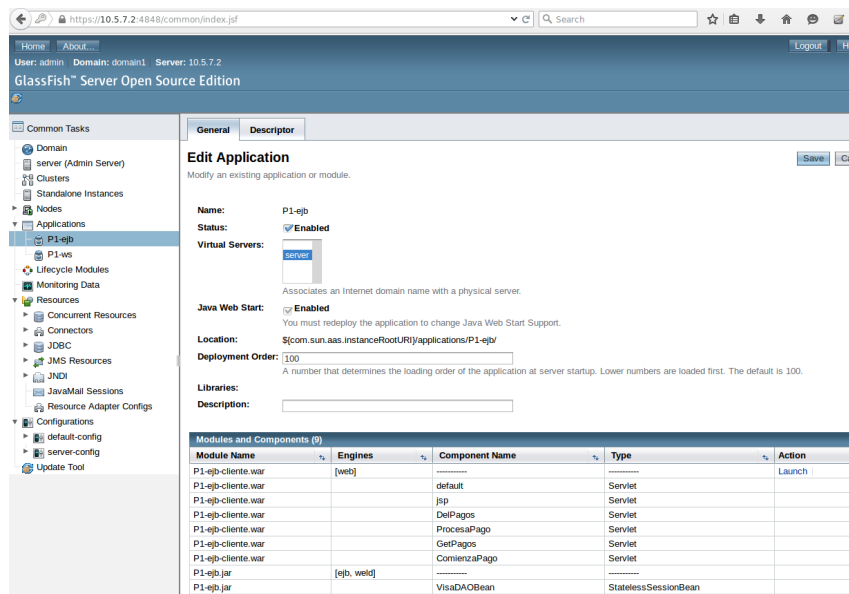


Figura 1: P1-ejb Application

Para probar el correcto funcionamiento de lo modificado anteriormente, hemos realizado 2 pagos con la ruta P1-ejb-cliente: uno a través de *pago.html* y otro a través *testbd.jsp*. Además, dentro del segundo también hemos probado a listar y eliminar uno de los pagos.

The screenshot shows a web form titled 'Id Transacción: 1', 'Id Comercio: 2', and 'Importe: 10'. There is a button labeled 'Envia Datos Pago'.

(a) HTML-1

The screenshot shows a web form titled 'Pago con tarjeta'. It contains input fields for 'Numero de visa: 1111 2222 3333 4444', 'Titular: Jose Garcia', 'Fecha Emisión: 11/09', 'Fecha Caducidad: 11/20', and 'CVV2: 123'. There is a button labeled 'Pagar'. Below the form, it displays 'Id Transacción: 1', 'Id Comercion: 2', and 'Importe: 10.0'. At the bottom, it says 'Prácticas de Sistemas Informáticos II'.

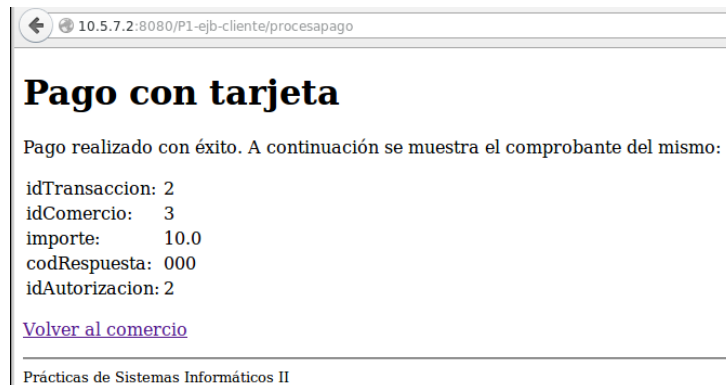
(b) HTML-2



Figura 2: HTML-3

A screenshot of a web browser window. The address bar shows the URL "10.5.7.2:8080/P1-ebj-cliente/testbd.jsp". The page title is "Pago con tarjeta". The main content area is divided into three sections: "Proceso de un pago", "Consulta de pagos", and "Borrado de pagos". The "Proceso de un pago" section contains a form with fields for "Id Transacción:" (value 2), "Id Comercio:" (value 3), "Importe:" (value 10), "Numero de visa:" (value 1111 2222 3333 4444), "Titular:" (value Jose Garcia), "Fecha Emisión:" (value 11/09), "Fecha Caducidad:" (value 11/20), "CVV2:" (value 123), and three radio button options: "Modo debug:" (True/False), "Direct Connection:" (True/False), and "Use Prepared:" (True/False). A "Pagar" button is at the bottom of this section. The "Consulta de pagos" section has an "Id Comercio:" field and a "GetPagos" button. The "Borrado de pagos" section has an "Id Comercio:" field and a "DelPagos" button. The footer of the browser window shows "Prácticas de Sistemas Informáticos II".

(a) JSP-1



(b) JSP-2



(c) Consulta



(d) Eliminación

Ejercicio 5

Antes de empezar este ejercicio hemos replegado la aplicación anterior (EJB local) por si pudiera producir conflictos con las siguientes. A partir de este punto, separamos en diferentes carpetas el cliente (a partir de ahora, cliente-remoto) y el servidor (igual que el cliente, ahora será servidor-remoto).

En la clase VisaDAOBean.java del servidor hemos añadido la interfaz VisaDAORemote además de la VisaDAOLocal.

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

Por último hemos modificado las clases PagoBean y TarjetaBean para que sean serializables.

```
public class PagoBean implements Serializable{
```

```
public class TarjetaBean implements Serializable{
```

Ahora que ya hemos modificado el servidor tenemos que cambiar el cliente.

Ejercicio 6

Para hacer el cliente remoto, partimos de la carpeta P1-base. En esta modificamos los servlets que invocan la lógica de negocios (ProcesaPago.java, GetPagos.java y DelPagos.java) sustituyendo las referencias a VisaDAOWS por VisaDAORemote de manera análoga a como lo hemos hecho para el servidor.

Además, hemos creado un archivo glassfish-web.xml que resolverá las referencias de los EJBs remotos.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GlassFish Application
  Server 3.1 Servlet 3.0//EN"
  "http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app>
<ejb-ref>
  <ejb-ref-name>VisaDAOBean</ejb-ref-name>
  <jndi-name>corbaname:iiop:10.5.7.2:3700#java:global/P1-ejb/P1-ejb/VisaDAOBean!ssii2.visa.VisaDAOBean
</ejb-ref>
</glassfish-web-app>

```

Tras esta separación de cliente y servidor en cliente-remoto y servidor-remoto respectivamente, realizamos las mismas pruebas que mostramos en el ejercicio 4. A continuación incluimos evidencias gráficas de que el pago se realiza correctamente.

(e) Pago-1

(f) Pago-2

Pago con tarjeta

Proceso de un pago

| | |
|--------------------------------------|--|
| Id Transacción: | <input type="text" value="6"/> |
| Id Comercio: | <input type="text" value="7"/> |
| Importe: | <input type="text" value="10"/> |
| Numero de visa: | <input type="text" value="1111 2222 3333 4444"/> |
| Titular: | <input type="text" value="Jose Garcia"/> |
| Fecha Emisión: | <input type="text" value="11/09"/> |
| Fecha Caducidad: | <input type="text" value="11/20"/> |
| CVV2: | <input type="text" value="123"/> |
| Modo debug: | <input type="radio"/> True <input type="radio"/> False |
| Direct Connection: | <input type="radio"/> True <input type="radio"/> False |
| Use Prepared: | <input type="radio"/> True <input type="radio"/> False |
| <input type="button" value="Pagar"/> | |

Consulta de pagos

| | |
|---|----------------------|
| Id Comercio: | <input type="text"/> |
| <input type="button" value="GetPagos"/> | |

Borrado de pagos

| | |
|---|----------------------|
| Id Comercio: | <input type="text"/> |
| <input type="button" value="DelPagos"/> | |

Pago con tarjeta

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

idTransaccion: 4
idComercio: 5
importe: 10.0
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

(g) Pago-3

Prácticas de Sistemas Informáticos II

(h) Pago-4

Pago con tarjeta

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

idTransaccion: 6
idComercio: 7
importe: 10.0
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

(i) Pago-5

y su correspondiente consulta y eliminación

10.5.7.1:8080/P1-ejb-cliente-remoto/getpagos

Pago con tarjeta

Lista de pagos del comercio 7

| idTransaccion | Importe | codRespuesta | idAutorizacion |
|---------------|---------|--------------|----------------|
| 6 | 10.0 | 000 | 2 |

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

(j) Consulta

10.5.7.1:8080/P1-ejb-cliente-remoto/delpagos

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 7

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

(k) Eliminación

Ejercicio 7

En esta nueva parte de la práctica ya no se trabaja con cliente y servidor remotos sino que se da un nuevo material de partida para analizar el funcionamiento de las transacciones. Por ello, para garantizar el correcto funcionamiento del posterior despliegue, replegamos las aplicaciones P1-ejb-cliente y servidor.

Hemos modificado varios archivos:

TarjetaBean.java. Hemos añadido el atributo double saldo con sus métodos set y get.

```
public class TarjetaBean {

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

    public double getSaldo() {
        return saldo;
    }

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

VisaDAOBean.java Hemos añadido consultas para obtener (get) y actualizar (set) el saldo de la tarjeta en VisaDAOBean.

```
private static final String SELECT_SALDO_TARJETA_QRY =
    "select saldo from tarjeta " +
    "where numeroTarjeta=? " +
    " and titular=? " +
    " and validaDesde=? " +
    " and validaHasta=? " +
    " and codigoVerificacion=? ";

private static final String UPDATE_SALDO_TARJETA_QRY =
    "update tarjeta " +
    "set saldo=? " +
    "where numeroTarjeta=? " +
    " and titular=? " +
    " and validaDesde=? " +
    " and validaHasta=? " +
    " and codigoVerificacion=? ";
```

```
String insert = SELECT_SALDO_TARJETA_QRY;
errorLog(insert);
pstmt = con.prepareStatement(insert);
pstmt.setString(1, pago.getTarjeta().getNumero());
pstmt.setString(2, pago.getTarjeta().getTitular());
pstmt.setString(3, pago.getTarjeta().getFechaEmision());
pstmt.setString(4, pago.getTarjeta().getFechaCaducidad());
pstmt.setString(5, pago.getTarjeta().getCodigoVerificacion());

rs = pstmt.executeQuery();
if(rs.next()) {
    saldo = rs.getDouble("saldo");
    if(pago.getImporte() > saldo){
        pago.setIdAutorizacion(null);
        pago = null;
        throw new EJBException("Saldo insuficiente. Pago denegado.");
    }
}

saldo -= pago.getImporte();

insert = UPDATE_SALDO_TARJETA_QRY;
errorLog(insert);
pstmt = con.prepareStatement(insert);
pstmt.setDouble(1, saldo);
pstmt.setString(2, pago.getTarjeta().getNumero());
pstmt.setString(3, pago.getTarjeta().getTitular());
pstmt.setString(4, pago.getTarjeta().getFechaEmision());
pstmt.setString(5, pago.getTarjeta().getFechaCaducidad());
pstmt.setString(6, pago.getTarjeta().getCodigoVerificacion());
pstmt.executeUpdate();
```

Ejercicio 8

Tras los pasos anteriores, probamos a realizar algún pago desde el usuario Blas Avila Sparrow como podemos ver en las siguientes imágenes.

Ejercicio 9

(p) Factoría de conexión

Ejercicio 10

(q) Cola de mensajes

Ejercicio 11

Hemos modificado el fichero sun-ejb-jar.xml para que se conecte con la factoría.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc./DTD Application Server 9.0
    EJB 3.0//EN" "http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-0.dtd">
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>VisaCancelacionJMSBean</ejb-name>
      <mdb-connection-factory>
        <jndi-name>jms/VisaConnectionFactory</jndi-name>
      </mdb-connection-factory>
    </ejb>
```

```
</enterprise-beans>
</sun-ejb-jar>
```

Por otra parte, hemos implementado el proceso de cancelación de un pago desarrollando funcionalidad que actualice la base de datos. Para ello hemos usado las consultas que vemos a continuación.

La primera de las consultas modifica el código de respuesta del pago para indicar que ha sido cancelado. Tras esto debemos obtener el número de la tarjeta a través del ID de autorización del pago. A continuación obtenemos el saldo actual de la tarjeta y el importe que pagó dicho usuario y los sumamos. Por último actualizamos el saldo del cliente con el nuevo valor calculado.

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

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

private static final String SELECT_TARJETA_IDAUTORIZACION_QRY =
    "select numeroTarjeta" +
    " from pago" +
    " where idAutorizacion = ?";

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

private static final String SELECT_IMPORTE_QRY =
    "select importe" +
    " from pago" +
    " where idAutorizacion = ?";
```

Ejercicio12

Hemos añadido anotaciones estáticas para que el cliente localice la cola de mensajes.

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

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

Por otro lado, esta sería la forma dinámica de obtención de los recursos.

```
/*Metodo de conexion dinamico*/
/*InitialContext jndi = new InitialContext();
connectionFactory =
    (ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");
queue = (Queue)jndi.lookup("jms/VisaPagosQueue");*/
```

Si bien el método estático es más directo y cuesta menos procesarlo, tiene el inconveniente de que debes conocer el nombre de los recursos en tiempo de compilación.

Ejercicio13

Hemos modificado el fichero `jms.properties` cambiando los últimos 3 atributos por el nombre de la factoría, `jms/VisaConnectionFactory`, el nombre de la cola, `jms/VisaPagosQueue` y el nombre del destino físico, `VisaPagosQueue`.

Además hemos cambiado los valores de `as.host.mdb` a `10.5.7.2` en `build.properties` y `as.host.client` a `10.5.7.2` junto con `as.host.server` a `10.5.7.2` en el fichero `jms.properties`.

En el fichero `jms.xml`, el comando equivalente para crear una cola JMS es el target: `create-jms-resource`.

Ejercicio14

Primero, hemos modificado el main del cliente para que, al recibir un mensaje con argumento distinto de `browse`, envíe un mensaje con el `IdAutorizacion` del supuesto pago a cancelar.

```
messageProducer = session.createProducer(queue);
message = session.createTextMessage();
message.setText(args[0]);

messageProducer.send(message);

messageProducer.close();
session.close();
```

Tras esto, en la web de configuración de Glassfish, en `configuration -> server config -> java message service -> jms host` hemos cambiado la variable `default_JMS_host` por `10.5.7.2`. Para que el cambio se efectúe debíamos parar el servidor con el comando `stop-domain` y restaurarlo.

En este punto nos dispusimos a ejecutar un pago desde Ejb-transaccional e intentar cancelarlo.



The screenshot shows a web browser window with the address bar displaying `10.5.7.2:8080/P1-ejb-cliente/testbd.jsp`. The page title is **Pago con tarjeta**. Below the title is the section **Proceso de un pago**. The form contains the following fields and controls:

- Id Transacción:** Text input with value `10`.
- Id Comercio:** Text input with value `11`.
- Importe:** Text input with value `30`.
- Numero de visa:** Text input with value `1111 2222 3333 4444`.
- Titular:** Text input with value `Jose Garcia`.
- Fecha Emisión:** Text input with value `11/09`.
- Fecha Caducidad:** Text input with value `11/20`.
- CVV2:** Text input with value `123`.
- Modo debug:** Radio buttons for `True` and `False`.
- Direct Connection:** Radio buttons for `True` and `False`.
- Use Prepared:** Radio buttons for `True` and `False`.
- Pagar** button.

(r) Pago-1



The screenshot shows a web browser window with the address bar displaying `10.5.7.2:8080/P1-ejb-cliente/procesapago`. The page title is **Pago con tarjeta**. The content of the page is:

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

- `idTransaccion: 10`
- `idComercio: 11`
- `importe: 30.0`
- `codRespuesta: 000`
- `idAutorizacion: 1`

[Volver al comercio](#)

(s) Pago-2

Para poder cancelarlo, debíamos ejecutar el cliente de Ejb-jms y enviar un mensaje de cancelación. Desde los ordenadores de los laboratorios entraba en conflicto el comando, puesto que requería modificar

un archivo del cual no tenemos permisos de escritura por lo que hemos tenido que ejecutarlo desde la máquina virtual 1.

Para ello debemos, en primer lugar, transferirle un jar a la máquina virtual ejecutando este comando:

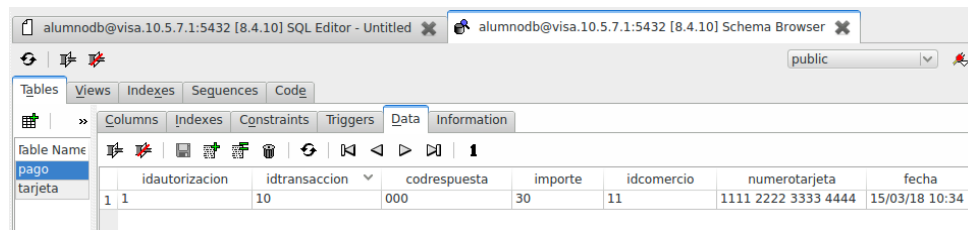
```
scp dist/clientjms/P1-jms-clientjms.jar si2@10.5.7.1:/tmp
```

En la máquina virtual debemos exportarle la variable Java y después podremos ejecutar el cliente con el segundo comando indicando, como último parámetro, el ID de autorización del pago a cancelar.

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

```
/opt/glassfish4/glassfish/bin/appclient -targetserver 10.5.7.2  
-client /tmp/P1-jms-clientjms.jar 1
```

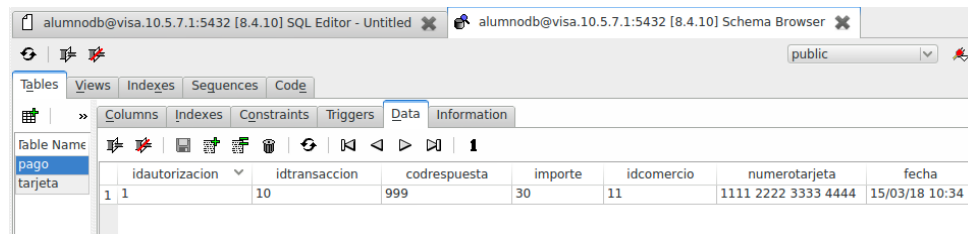
Para comprobar la correcta cancelación del pago mostramos a continuación el pago desde la aplicación Tora, antes y después de la cancelación.



The screenshot shows the Tora SQL Editor interface. The 'Data' tab is selected for the 'pago' table. The table has columns: idautorizacion, idtransaccion, codrespuesta, importe, idcomercio, numerotarjeta, and fecha. The data row shows: 1, 1, 10, 000, 30, 11, 1111 2222 3333 4444, 15/03/18 10:34.

| idautorizacion | idtransaccion | codrespuesta | importe | idcomercio | numerotarjeta | fecha |
|----------------|---------------|--------------|---------|------------|---------------|------------------------------------|
| 1 | 1 | 10 | 000 | 30 | 11 | 1111 2222 3333 4444 15/03/18 10:34 |

(t) Pago antes



The screenshot shows the Tora SQL Editor interface. The 'Data' tab is selected for the 'pago' table. The table has columns: idautorizacion, idtransaccion, codrespuesta, importe, idcomercio, numerotarjeta, and fecha. The data row shows: 1, 1, 10, 999, 30, 11, 1111 2222 3333 4444, 15/03/18 10:34.

| idautorizacion | idtransaccion | codrespuesta | importe | idcomercio | numerotarjeta | fecha |
|----------------|---------------|--------------|---------|------------|---------------|------------------------------------|
| 1 | 1 | 10 | 999 | 30 | 11 | 1111 2222 3333 4444 15/03/18 10:34 |

(u) Pago después

Durante las pruebas de cancelación de pagos nos hemos dado cuenta de que, al no tener una previa comprobación de que el pago ya estuviese cancelado, si un cliente cancela varias veces un mismo pago, el autor del pago recibiría el importe del pago todas las veces pudiendo así superar su saldo previo al pago.