UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2**			
Grupo	**2401**	**Práctica**	1B	**Fecha**	27/02/2018
Alumno/a		Amor, Mourelle, Antonio			
Alumno/a		López, Ramos, Esther			

Práctica 1: Arquitectura de Java EE

Ejercicio 1:

Introduzca las siguientes modificaciones en el bean VisaDAOBean para convertirlo en un EJB stateless con interfaz local:

- **Hacer que la clase implemente la interfaz local y convertirla en un EJB stateless mediante la anotación Stateless**
  Borramos las librerías relativas a Web Service y las cambiamos por javx.ejb.Stateless. Añadimos la anotación @Stateless a la clase VisaDAOBean. Hemos borrado todas las anotaciones anteriores propias del Web Service.
- **Eliminar el constructor por defecto de la clase.**
- **Ajustar los métodos getPagos() a la interfaz definida en VisaDAOLocal**
  Modificamos de nuevo el método getPagos para que devuelva un array en lugar de un ArrayList.

Ejercicio 2:

Modificar el servlet ProcesaPago para que acceda al EJB local.

Aparte de hacer las modificaciones descritas en el enunciado en el servlet ProcesaPago se han realizado las mismas modificaciones en los siguientes servlets:

- DelPagos
- GetPagos

CUESTIONES:

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.

La única librería de Java EE importada es *javax.ejb.Local* que añade la anotación *@Local.* La anotación

Cuestión número 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.

Al abrir el archivo obtenemos lo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="ht
tp://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>
~
```

Los campos de la aplicación son los siguientes:

<display-name> : nombre de la aplicación (nombre que veremos en glassfish)

Consta de dos módulos:

<ejb> el jar del servidor

<web> el cliente web, contiene su correspondiente war.

El resultado de ejecutar el el comando: jar -tvf **/dist/dist/server/P1-ejb.jar** es el siguiente:

0 Fri Mar 02 10:29:44 CET 2018 META-INF/

104 Fri Mar 02 10:29:42 CET 2018 META-INF/MANIFEST.MF

0 Fri Mar 02 10:29:34 CET 2018 ssii2/

0 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/

0 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/dao/

255 Fri Mar 02 10:29:44 CET 2018 META-INF/sun-ejb-jar.xml

1464 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/PagoBean.class

856 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/TarjetaBean.class

593 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/VisaDAOLocal.class

1723 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/dao/DBTester.class

7016 Fri Mar 02 10:29:34 CET 2018 ssii2/visa/dao/VisaDAOBean.class

Como podemos observar el jar del servidor contiene todas las clases generadas tras compilar los .java. Estas clases son las que contienen la lógica de la aplicación de pagos y son las que el cliente invoca.

Repetimos el proceso sobre el archivo **/dist/client/P1-ejb-cliente.war** y el resultado es:

0 Fri Mar 02 10:29:56 CET 2018 META-INF/

104 Fri Mar 02 10:29:54 CET 2018 META-INF/MANIFEST.MF

0 Fri Mar 02 10:29:54 CET 2018 WEB-INF/

0 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/

0 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/

0 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/controlador/

0 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/filtros/

0 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/

0 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/

0 Fri Mar 02 10:29:08 CET 2018 WEB-INF/lib/

0 Fri Mar 02 10:29:54 CET 2018 error/

2856 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/controlador/ComienzaPago.class

1636 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/controlador/DelPagos.class

1600 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/controlador/GetPagos.class

5172 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/controlador/ProcesaPago.class

1894 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/controlador/ServletRaiz.class

2608 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/filtros/CompruebaSesion.class

3170 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/ValidadorTarjeta.class

616 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/ErrorVisa.class

198 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/ErrorVisaCVV.class

209 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/ErrorVisaFechaCaducidad.class

207 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/ErrorVisaFechaEmision.class

201 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/ErrorVisaNumero.class

202 Fri Mar 02 10:29:48 CET 2018 WEB-INF/classes/ssii2/visa/error/ErrorVisaTitular.class

6044 Fri Mar 02 10:29:54 CET 2018 WEB-INF/web.xml

455 Fri Mar 02 10:29:54 CET 2018 borradoerror.jsp

501 Fri Mar 02 10:29:54 CET 2018 borradook.jsp

509 Fri Mar 02 10:29:54 CET 2018 cabecera.jsp

283 Fri Mar 02 10:29:54 CET 2018 error/muestraerror.jsp

2729 Fri Mar 02 10:29:54 CET 2018 formdatosvisa.jsp

1257 Fri Mar 02 10:29:54 CET 2018 listapagos.jsp

1178 Fri Mar 02 10:29:54 CET 2018 pago.html

1142 Fri Mar 02 10:29:54 CET 2018 pagoexito.jsp

104 Fri Mar 02 10:29:54 CET 2018 pie.html

5011 Fri Mar 02 10:29:54 CET 2018 testbd.jsp

Al igual que el servidor, el cliente contiene los .class generados tras compilarlos, además de los html y jsp necesarios para hacer las pruebas desde un navegador.

Por último realizamos el comando jat -tvf sobre **/dist/P1-ejb.ear** obteniendo:

0 Fri Mar 02 10:50:42 CET 2018 META-INF/

104 Fri Mar 02 10:50:40 CET 2018 META-INF/MANIFEST.MF

508 Fri Mar 02 10:17:16 CET 2018 META-INF/application.xml

21244 Fri Mar 02 10:29:54 CET 2018 P1-ejb-cliente.war

6986 Fri Mar 02 10:29:44 CET 2018 P1-ejb.jar

El ear es el paquete generado para la aplicación, que contiene tanto los paquetes del servidor como los del cliente.

Ejercicio 3:

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

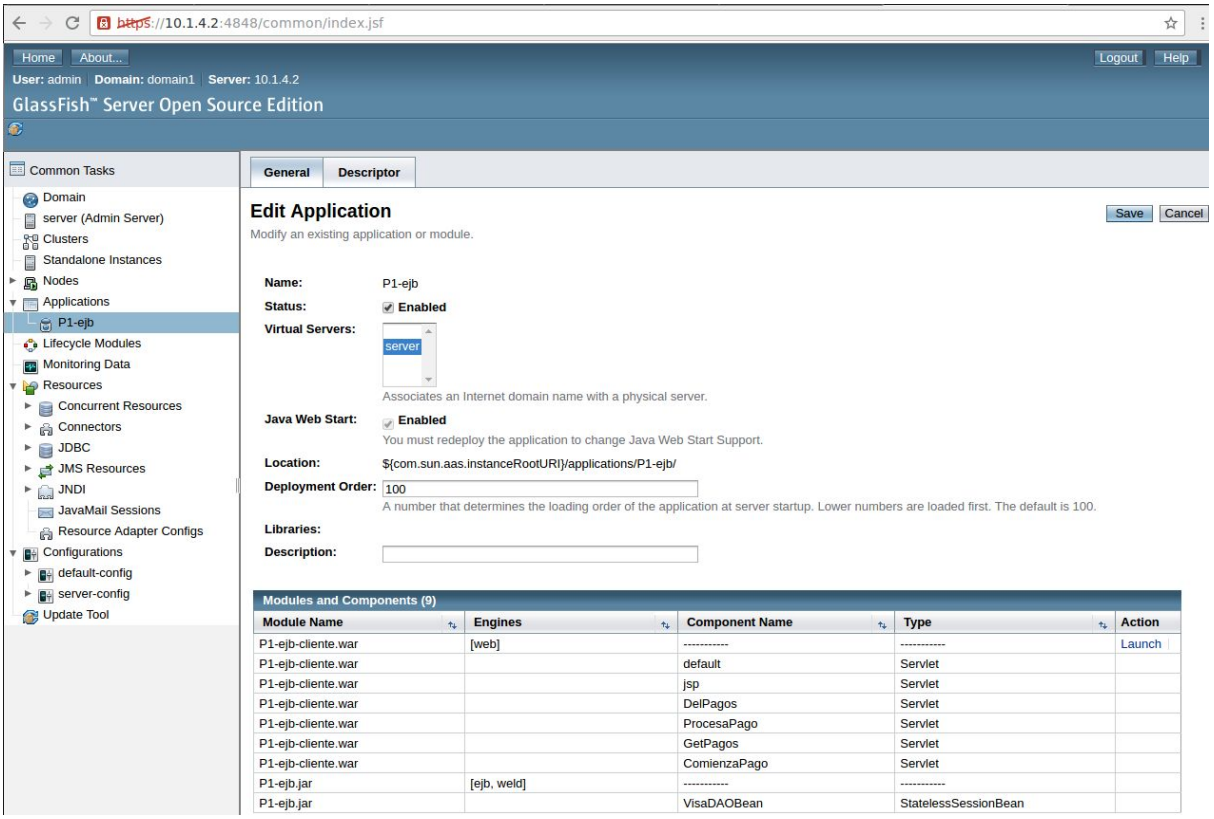
- **Editar el archivo build.properties para que las propiedades as.host.client y as.host.server contengan la dirección IP del servidor de aplicaciones.**

  La IP introducida en ambos campos es 10.1.4.2.

- **Editar el archivo postgresql.properties para la propiedad db.client.host y db.host contengan las direcciones IP adecuadas para que el servidor de aplicaciones se conecte al postgresql, ambos estando en servidores diferentes.**

  db.cliente.host es la ip 10.1.4.2 mientras que la base de datos está en el 10.1.4.1.

Desplegar la aplicación de empresa ant desplegar



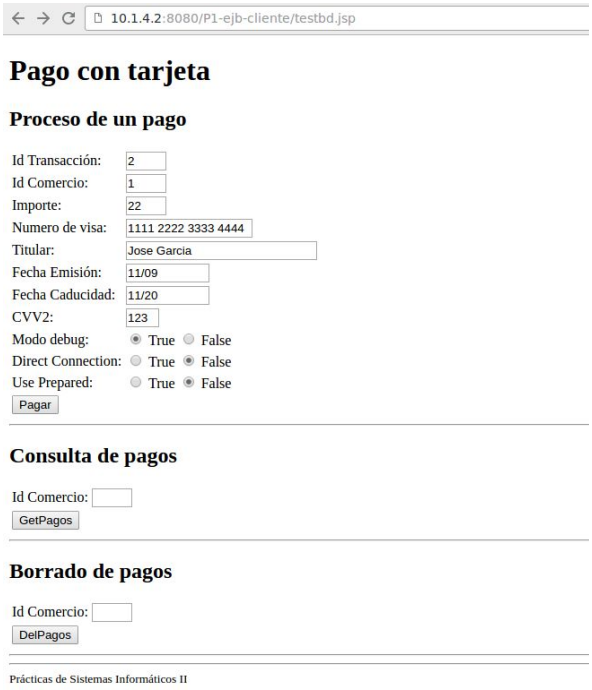
Puede observarse que es similar a la captura mostrada en el enunciado.

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.

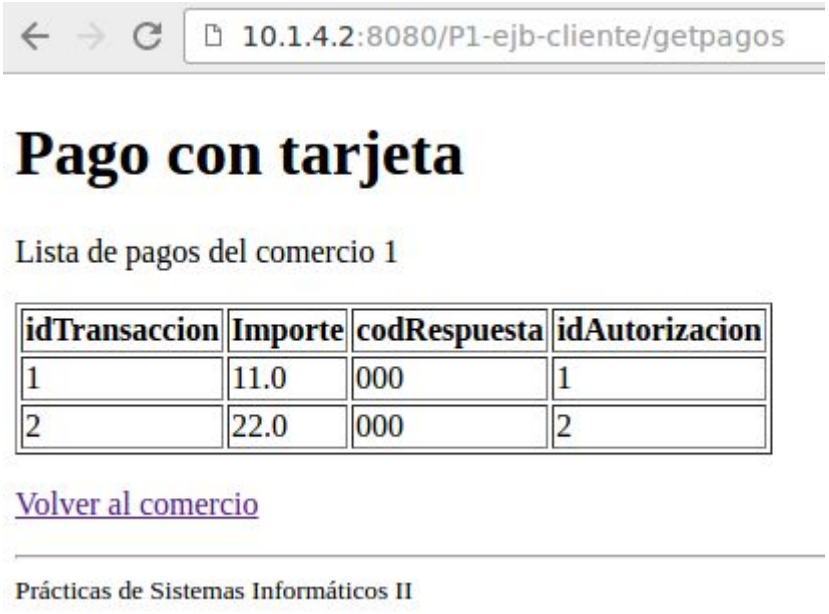
Tras realizar el pago mediante la interfaz web obtenemos el siguiente resultado:

Ahora realizamos el pago mediante testbd.jsp:



Se comprueba que este segundo pago se ha realizado con éxito en la siguiente captura:

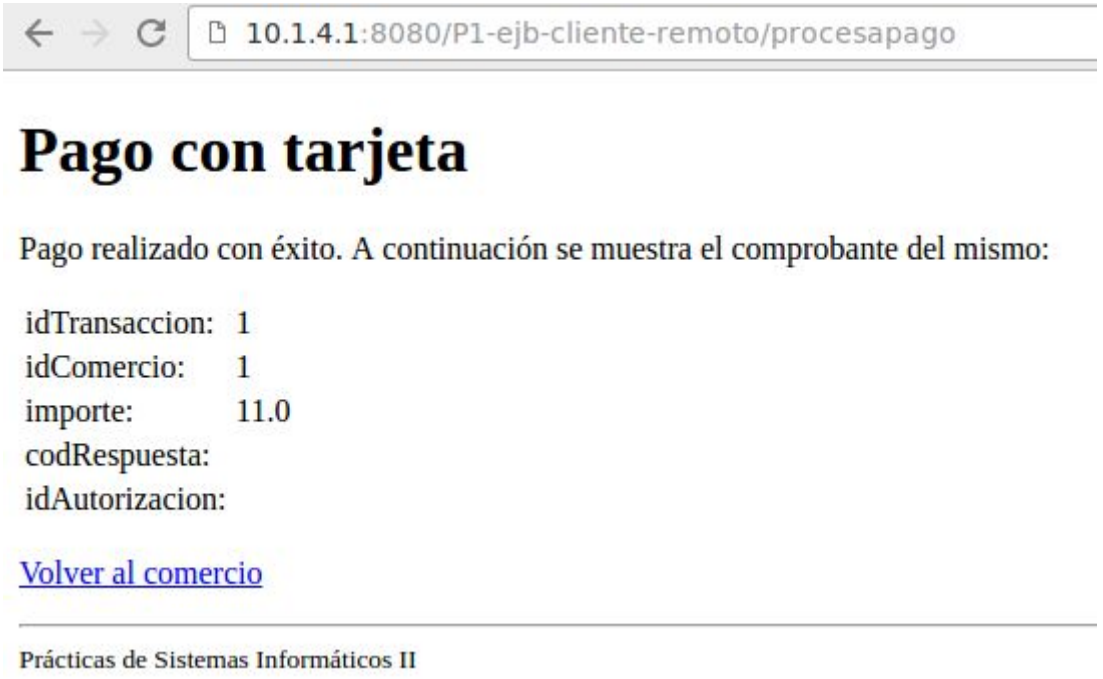
Por último, borramos los dos pagos anteriores:



Ejercicio 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.

El pago, realizado desde el cliente (ip 10.1.4.1), se realiza con exito.



Ejercicio 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 insertar 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).

Las sentencias preparadas introducidas en el código han sido las siguientes.

/*TRANSACCIONES*/

La query destinada a obtener el saldo de una tarjeta es la siguiente:

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

Los parametros se asignan con el método siguiente:

```
pstmt.setString(1, pago.getTarjeta().getNumero());
```

La sentencia preparada destinada a actualizar el saldo es la siguiente:

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

Los parametros de rellenan en las sentencias:

```
pstmt.setDouble(1, saldo);
pstmt.setString(2, pago.getTarjeta().getNumero());
```

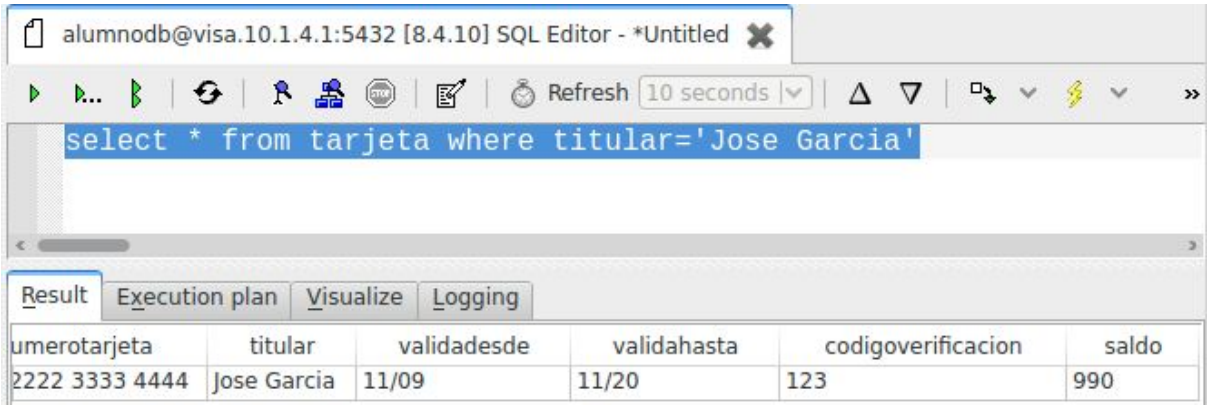
Ejercicio 8:

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.**

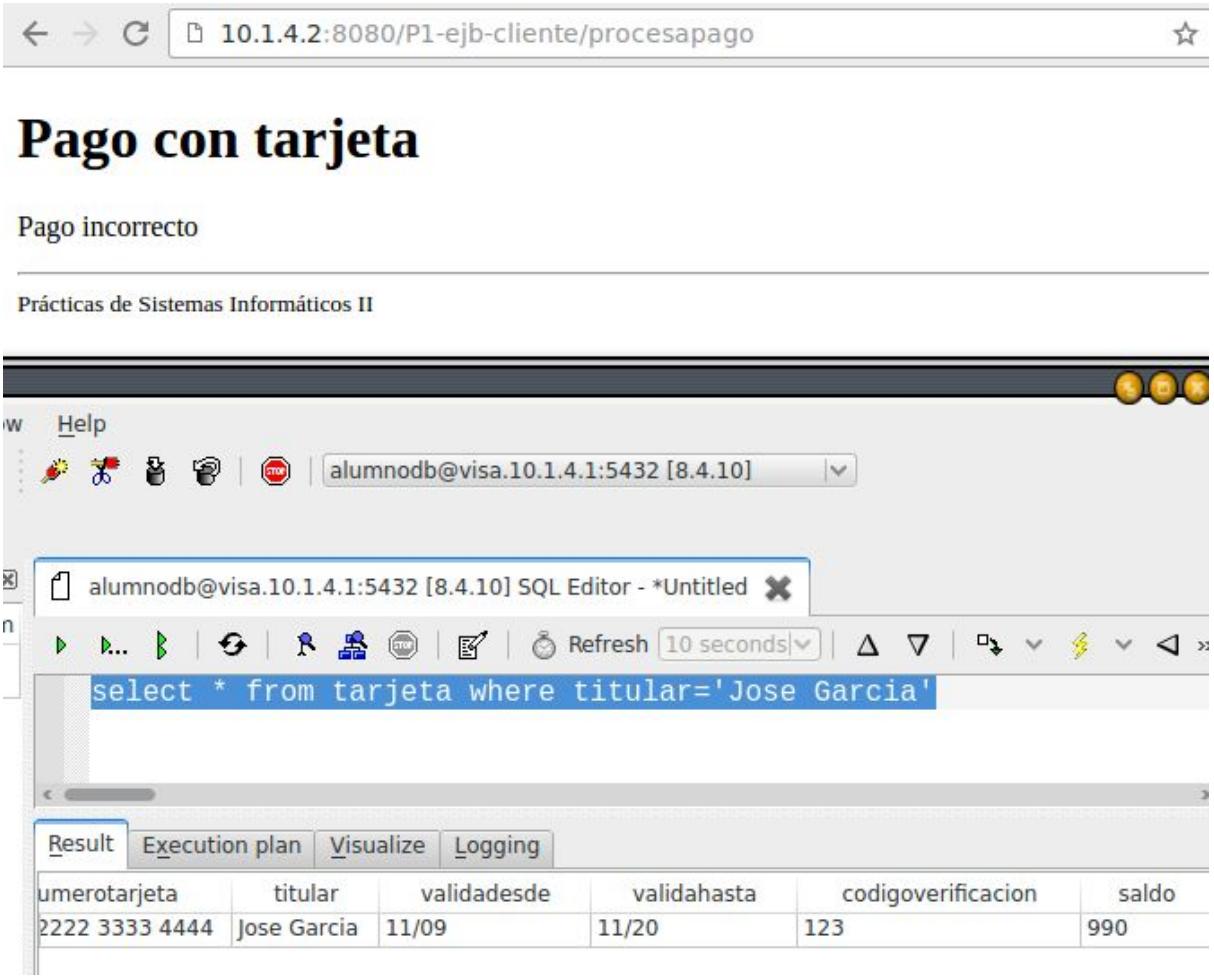
Realizamos un pago desde la página testbd.jsp y comprobamos que la salida es correcta:

Desde TOra vemos que el saldo se ha actualizado correctamente:



- **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.**

Volvemos a realizar un pago con el mismo id de comercio y de transacción. La página nos devuelve un error y en TOra no se actualiza el saldo:



Ejercicio 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 actualizar el código de respuesta a valor 999, de aquella autorización existente en la tabla de pagos cuyo idAutorizacion coincida con lo recibido por el mensaje.**

private static final String UPDATE_CANCELA_QRY ="UPDATE pago SET codRespuesta=999 "+ "where idAutorizacion=?";

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

private static final String UPDATE_SALDO_QRY ="UPDATE tarjeta SET saldo = saldo + pago.importe "+ "FROM pago WHERE idAutorizacion= ? AND tarjeta.numerotarjeta = pago.numerotarjeta";

- **Método onMessage() que implemente ambas actualizaciones. Para ello tome de ejemplo el código SQL de ejercicios anteriores, de modo que se use un prepared statement que haga bind del idAutorizacion para cada mensaje recibido.**

En el método onMessage, rellenamos los campos de las consultas anteriores mediante el método setInteger.

Ejercicio 12:

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

Las principal ventaja de escribir las anotaciones es que se ahorra código. Su desventaja es que no es modificable en tiempo de ejecución.Esta es la gran ventaja del segundo método. Los nombres pueden ser desconocidos en tiempo de compilación e introducirse al programa mediante variables.

Ejercicio 13:

Automatice la creación de los recursos JMS (cola y connection factory) en el build.xml y jms.xml. Para ello, indique en jms.properties los nombres de ambos y el Physical Destination Name de la cola de acuerdo a los valores asignados en los ejercicios 7 y 8. Recuerde también asignar las direcciones IP adecuadas a las variables as.host.mdb (build.properties) y as.host.server (jms.properties).

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. 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.

El comando que crea una cola JMS es :

```
<target name="setup-jms"
  description="Crea nuestros recursos JMS">
  <antcall target="create-jms-connection-factory">
     <param name="jms.restype" value="javax.jms.QueueConnectionFactory" />
     <param name="jms.resource.name" value="${jms.factoryname}" />
  </antcall>
  <antcall target="create-jms-resource">
     <param name="jms.restype" value="javax.jms.Queue" />
     <param name="jms.resource.property" value="Name=${jms.physname}" />
     <param name="jms.resource.name" value="${jms.name}" />
  </antcall>
 </target>
```

Ejercicio 14:

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

Modifique el cliente, VisaQueueMessageProducer.java, implementando el envío de args[0] como mensaje de texto (consultar los apéndices). Ejecute el cliente en el PC del laboratorio mediante el comando:

/usr/local/glassfish-4.1.1/glassfish/bin/appclient –targetserver 10.X.Y.Z - client dist/clientjms/P1-jms-clientjms.jar idAutorizacion

Donde 10.X.Y.Z representa la dirección IP de la máquina virtual en cuyo servidor de aplicaciones se encuentra desplegado el MDB. Para garantizar que el comando funcione correctamente es necesario fijar la variable

(web console->Configurations->server-config->Java Message Service->JMS Hosts->default_JMS_host)

que toma el valor "localhost" por la dirección IP de dicha máquina virtual. El cambio se puede llevar a cabo desde la consola de administración. Será necesario reiniciar el servidor de aplicaciones para que surja efecto.

Verifique el contenido de la cola ejecutando:

usr/local/glassfish-4.1.1/glassfish/bin/appclient –targetserver 10.X.Y.Z - client dist/clientjms/P1-jms-clientjms.jar -browse

Indique la salida del comando e inclúyala en la memoria de prácticas.

Es importante cambiar en la consola de glassfish en el apartado default_JMS_host por la dirección IP de nuestro servidor 10.1.4.2.

En primer lugar realizamos el siguiente comando:

/usr/local/glassfish-4.1.1/glassfish/bin/appclient -targetserver 10.1.4.2 -client dist/clientjms/P1-jms-clientjms.jar 1111

Que deja el mensaje "1111" en la cola.

A continuación comprobamos el contenido de la cola sustituyendo en el comando anterior el

id de autorización por “-browse”.

```
e321081@localhost:~/Desktop/SI-II/P1b/P1-jms$ /usr/local/glassfish-4.1
ms.jar -browse
mar 06, 2018 6:08:38 PM org.hibernate.validator.internal.util.Version
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 06, 2018 6:08:38 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Versio
mar 06, 2018 6:08:38 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter startin
mar 06, 2018 6:08:38 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started
Mensajes en cola:
1111
```

A continuación, volver a habilitar la ejecución del MDB y realizar los siguientes pasos:

- Realice un pago con la aplicación web

- Obtenga evidencias de que se ha realizado

- Cancélelo con el cliente

- Obtenga evidencias de que se ha cancelado y de que el saldo se ha rectificado

Al realizar este ejercicio en los laboratorios surge un error indicando que no es posible resolver el nombre del host local a una dirección IP. Esto se debe a que no hay una entrada con dicho nombre en el fichero /etc/hosts asociado a una dirección IP. Como dicho fichero no se puede editar, la solución es ejecutar el cliente de colas de mensajes desde la máquina virtual 1, para que se conecte a la máquina virtual 2. Basta con copiar el .jar del cliente a la máquina virtual, iniciar sesión de forma remota. Indicar donde se encuentra la versión 8 de java exportando la variable JAVA_HOME y ejecutar el cliente de colas con appclient desde la máquina virtual 1. Para ello, hay que ejecutar la siguiente secuencia de comandos:

Desde PC1 host:

$ scp dist/clientjms/P1-jms-clientjms.jar si2@10.X.Y.1:/tmp

Desde la máquina virtual 10.X.Y.1:

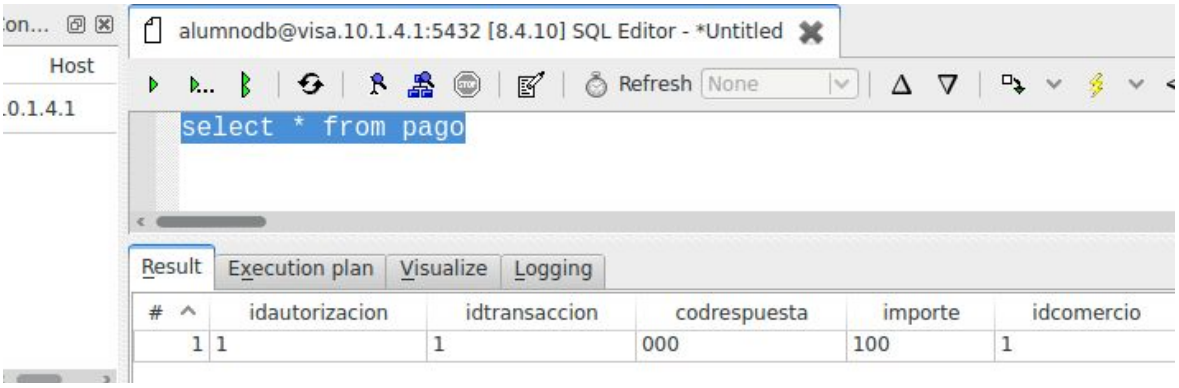
si2@si2srv01:~$ export JAVA_HOME=/usr/lib/jvm/java-8-oracle/

si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.X.Y.2 -client /tmp/P1-jms-clientjms.jar

Antes de poder realizar un pago es necesario desplegar el cliente y el servidor de transaccional, dado que son los únicos que contienen el atributo saldo.

Tras hacer esto, procedemos a realizar un pago como habitualmente y comprobamos que se realiza correctamente:

Comprobamos también que se ha introducido en la base de datos con el código de respuesta 000:

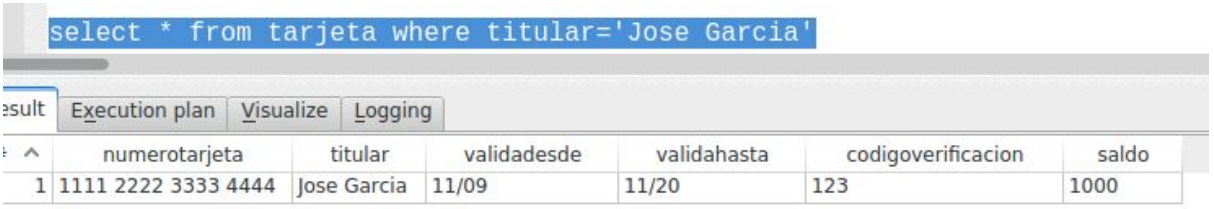


Ahora tras habilitar la cola de mensajes desde la consola de glassfish, enviamos desde la máquina cliente (IP 10.1.4.1) el id de autorización 1 para cancelar el pago con el siguiente comando:

/opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.4.2 -client /tmp/P1-jms-clientjms.jar 1

Tras ello, podemos observar como el código de respuesta se ha actualizado. Como se esperaba a cambiado a 999, lo que significa que el pago ha sido cancelado.

Ahora, solo falta comprobar que el dinero ha sido devuelto al titular de la tarjeta. Para ello, realizamos la siguiente consulta.



	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1	1111 2222 3333 4444	Jose Garcia	11/09	11/20	123	1000

Como puede verse, el dinero ha vuelto a 1000.