Javier Delgado del Cerro y Javier López Cano

Memoria P1B

Cuestión 1

Además de varios imports del paquete `java.sql` y el import de `java.utils.ArrayList`, destaca el `import javax.ejb.local;` junto con la anotación `@Local`.

Esta anotación especifica que para cualquier clase que implemente esta interfaz, sus métodos EJB que provengan de la interfaz *VisaDAOLocal* solo podrán ser llamados desde un cliente local.

Ejercicio 1

En primer lugar añadimos el siguiente import en la parte superior del fichero VisaDAOBean.java junto con el resto de los imports:

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

Cambiamos la declaración de la clase VisaDAOBean por:

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

De modo que emplee la anotación `@Stateless` en lugar de `@WebService` e implemente la interfaz `VisaDAOLocal`.

Eliminamos el constructor por defecto de la clase, como se nos indica y ajustamos los métodos a la interfaz, para ello eliminamos todas las anotaciones `@WebMethod` en todos los métodos, y se debe cambiar el retorno del método `getPagos()` de `ArrayList<PagoBean>` a `PagoBean[]` realizando además los cambios necesarios en el método.

De este modo las cabeceras de los métodos modificados quedarán:

```
public boolean compruebaTarjeta(TarjetaBean tarjeta);
public synchronized PagoBean realizaPago(PagoBean pago);
public PagoBean[] getPagos(String idComercio);
public int delPagos(String idComercio);
public boolean isPrepared();
public void setPrepared(boolean prepared);
public boolean isDebug();
public void setDebug(boolean debug);
public void setDebug(String debug);
public boolean isDirectConnection();
public void setDirectConnection(boolean directConnection);
```

Ejercicio 2

Tras cambiar los imports que se nos piden, el inicio de `ProcesaPago.java` queda de la siguiente forma:

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import ssii2.visa.*;
import javax.ejb.EJB;
import ssii2.visa.VisaDAOLocal;

/**
 *
 * @author phaya
 */
public class ProcesaPago extends ServletRaiz {
```

Tras esto añadimos el objeto proxy para acceder al EJB local con la anotación `@EJB` y .

```
    /**
     * Par&aacute;metro que indica el c&oacute;digo de verificaci&oacute;n
     * de la tarjeta
     */
    public final static String PARAM_CVV = "codigoVerificacion";

    /**
     * Atributo que hace referencia la bean Pago
     */
    public final static String ATTR_PAGO = "pago";


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


  private static void displayInterfaceInformation(
        NetworkInterface netint) throws SocketException {
      System.out.printf(
          "Display name: %s%n", netint.getDisplayName());
```

Por último eliminamos la declaración del webservice *VisaDAOWS* y el código para obtener el objeto remoto pues se usará el EJB local, además eliminamos también las referencias a BindingProvider. Por tanto el código quedará del siguiente modo:

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

        TarjetaBean tarjeta = creaTarjeta(request);
        ValidadorTarjeta val = new ValidadorTarjeta();
        PagoBean pago = null;

        // printAddresses(request,response);
        if (! val.esValida(tarjeta)) {
            request.setAttribute(val.getErrorName(), val.getErrorVisa());
            reenvia("/formdatosvisa.jsp", request, response);
            return;
        }

    HttpSession sesion = request.getSession(false);
    if (sesion != null) {
      pago = (PagoBean) sesion.getAttribute(ComienzaPago.ATTR_PAGO);
    }
    if (pago == null) {
      pago = creaPago(request);
      Boolean isdebug = Boolean.valueOf(request.getParameter("debug"));
      dao.setDebug(isdebug.toString());
      boolean isdirectConnection = Boolean.valueOf(request.getParameter("directConnection"));
      dao.setDirectConnection(isdirectConnection);
      boolean usePrepared = Boolean.valueOf(request.getParameter("usePrepared"));
      dao.setPrepared(usePrepared);
        }

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

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

Cuestión 2

El archivo *application.xml* contiene la siguiente información, necesaria para describir una aplicación Glassfish:

```
<?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>
```

Este archivo especifica el nombre de la aplicación que se mostrará en la consola de administración de Glassfish mediante el elemento xml *display-name*, y describe cada uno de sus dos módulos:

- El primero contiene un *ejb*, y se indica su nombre (*P1-ejb.jar*).
- El segundo módulo es un elemento web, y se especifica su URI relativa al nivel del package de la aplicación (*P1-ejb-cliente.war*) y la ruta sobre la que se despliega este módulo de la aplicación (*/P1-ejb-cliente*).

Hasta ahora se han generado entonces los ficheros `/dist/server/P1-ejb.jar` , `/dist/client/P1-ejb-cliente.war` y `/dist/P1-ejb.ear`. Tras ejecutar el comando `jar -tvf` con los tres, el primero devuelve la siguiente salida:

```
     0 Sat Mar 14 18:56:50 CET 2020 META-INF/
   125 Sat Mar 14 18:56:48 CET 2020 META-INF/MANIFEST.MF
     0 Tue Mar 03 17:55:08 CET 2020 ssii2/
     0 Tue Mar 03 17:55:08 CET 2020 ssii2/visa/
     0 Tue Mar 03 17:55:08 CET 2020 ssii2/visa/dao/
   255 Tue Feb 25 17:13:56 CET 2020 META-INF/sun-ejb-jar.xml
  1464 Tue Feb 25 17:02:36 CET 2020 ssii2/visa/PagoBean.class
   856 Tue Feb 25 17:02:36 CET 2020 ssii2/visa/TarjetaBean.class
   593 Tue Feb 25 17:02:36 CET 2020 ssii2/visa/VisaDAOLocal.class
  1723 Sat Mar 14 18:56:36 CET 2020 ssii2/visa/dao/DBTester.class
  7037 Sat Mar 14 18:56:36 CET 2020 ssii2/visa/dao/VisaDAOBean.class
```

Podemos apreciar así que se incluyen todas las clases necesarias para que el servidor funcione correctamente, junto con el fichero *sun-ejb-jar.xml* que especifica la configuración del *entreprise bean*.

El segundo archivo devuelve:

```
     0 Sat Mar 14 18:57:08 CET 2020 META-INF/
   125 Sat Mar 14 18:57:06 CET 2020 META-INF/MANIFEST.MF
```

```
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/classes/
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/classes/ssii2/
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/classes/ssii2/controlador/
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/classes/ssii2/filtros/
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/classes/ssii2/visa/
     0 Tue Mar 03 17:55:06 CET 2020 WEB-INF/classes/ssii2/visa/error/
     0 Tue Feb 25 17:01:28 CET 2020 WEB-INF/lib/
     0 Tue Mar 03 17:55:08 CET 2020 error/
  2844 Tue Feb 25 17:20:08 CET 2020 WEB-
INF/classes/ssii2/controlador/ComienzaPago.class
  1513 Tue Feb 25 17:20:08 CET 2020 WEB-
INF/classes/ssii2/controlador/DelPagos.class
  1365 Tue Feb 25 17:20:08 CET 2020 WEB-
INF/classes/ssii2/controlador/GetPagos.class
  4915 Tue Feb 25 17:20:42 CET 2020 WEB-
INF/classes/ssii2/controlador/ProcesaPago.class
  1894 Tue Feb 25 17:20:08 CET 2020 WEB-
INF/classes/ssii2/controlador/ServletRaiz.class
  2608 Tue Feb 25 17:20:42 CET 2020 WEB-
INF/classes/ssii2/filtros/CompruebaSesion.class
  3170 Tue Feb 25 17:20:42 CET 2020 WEB-
INF/classes/ssii2/visa/ValidadorTarjeta.class
   616 Sat Mar 14 18:56:54 CET 2020 WEB-
INF/classes/ssii2/visa/error/ErrorVisa.class
   198 Sat Mar 14 18:56:54 CET 2020 WEB-
INF/classes/ssii2/visa/error/ErrorVisaCVV.class
   209 Sat Mar 14 18:56:54 CET 2020 WEB-
INF/classes/ssii2/visa/error/ErrorVisaFechaCaducidad.class
   207 Sat Mar 14 18:56:54 CET 2020 WEB-
INF/classes/ssii2/visa/error/ErrorVisaFechaEmision.class
   201 Sat Mar 14 18:56:54 CET 2020 WEB-
INF/classes/ssii2/visa/error/ErrorVisaNumero.class
   202 Sat Mar 14 18:56:54 CET 2020 WEB-
INF/classes/ssii2/visa/error/ErrorVisaTitular.class
  6044 Tue Feb 25 17:20:50 CET 2020 WEB-INF/web.xml
   455 Tue Feb 25 17:20:50 CET 2020 borradoerror.jsp
   501 Tue Feb 25 17:20:50 CET 2020 borradook.jsp
   509 Tue Feb 25 17:20:50 CET 2020 cabecera.jsp
   283 Tue Feb 25 17:20:50 CET 2020 error/muestraerror.jsp
  2729 Tue Feb 25 17:20:50 CET 2020 formdatosvisa.jsp
  1257 Tue Feb 25 17:20:50 CET 2020 listapagos.jsp
  1178 Tue Feb 25 17:20:50 CET 2020 pago.html
  1142 Tue Feb 25 17:20:50 CET 2020 pagoexito.jsp
   104 Tue Feb 25 17:20:50 CET 2020 pie.html
  5011 Tue Feb 25 17:20:50 CET 2020 testbd.jsp
```

Y, de nuevo, permite ver todas las clases necesarias para que el cliente web se comunique con el servidor correctamente, las páginas *html* y *jsp* que tiene que permitir mostrar el cliente, y el archivo *web.xml* que almacena la descripción general de la aplicación web, con información sobre su nombre, mappeo entre servlets y urls, definición del tiempo de sesión por defecto y lista de archivos que el cliente envía por defecto, junto con la dirección del endpoint que usará el cliente.

Por último, el tercer archivo devuelve la siguiente información:

```
    0 Tue Feb 25 17:21:00 CET 2020 META-INF/
  105 Tue Feb 25 17:20:58 CET 2020 META-INF/MANIFEST.MF
  508 Sat Feb 11 23:33:00 CET 2012 META-INF/application.xml
20946 Tue Feb 25 17:20:52 CET 2020 P1-ejb-cliente.war
 7001 Tue Feb 25 17:14:06 CET 2020 P1-ejb.jar
```

Como era de esperar, pues es el fichero encargado de empaquetar los distintos módulos cliente y servidor de la aplicación, junto con el fichero xml *application.xml* que describe dicha aplicación, como hemos explicado anteriormente. De esta forma, podemos desplegar la aplicación de manera correcta.

Ejercicio 3

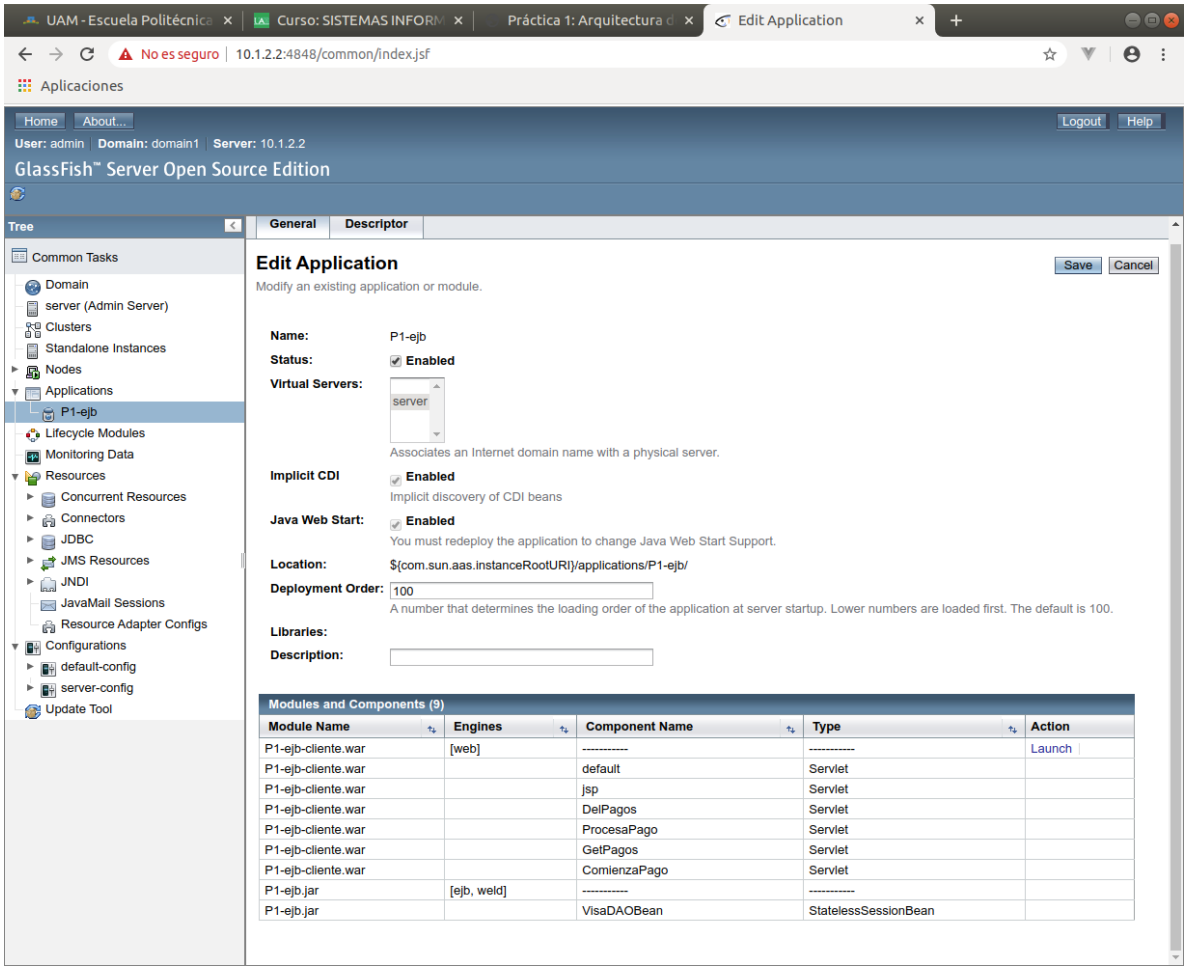
Editamos el fichero `build.properties` colocando en `as.host.client` y `as.host.server` la dirección IP `10.1.2.2`, es decir, la IP del servidor de aplicaciones, esto se debe a que tanto el cliente como el servidor van a desplegarse en la misma máquina virtual.

En el fichero `postgresql.properties` introducimos en el parámetro `db.client.host` la IP `10.1.2.2`, ya que en esta IP estará desplegado el servidor, que es quien accederá a la base de datos , y en el parámetro `db.host` la IP `10.1.2.1` ya que es la IP de la máquina virtual en que estará la base de datos.

Tras esto desplegamos el cliente, el servidor y la aplicación con los comandos:

```
$ ant compilar-servidor
$ ant empaquetar-servidor
$ ant compilar-cliente
$ ant empaquetar-cliente
$ ant empaquetar-aplicacion
$ ant desplegar
```

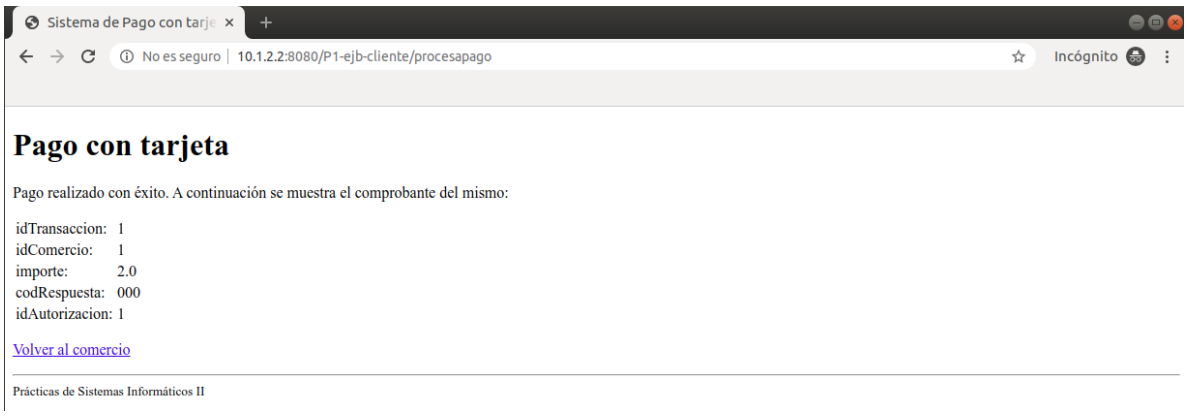
Y al abrir la pestaña de administración de Glassfish y vemos que se ha desplegado como `Enterprise Application`.



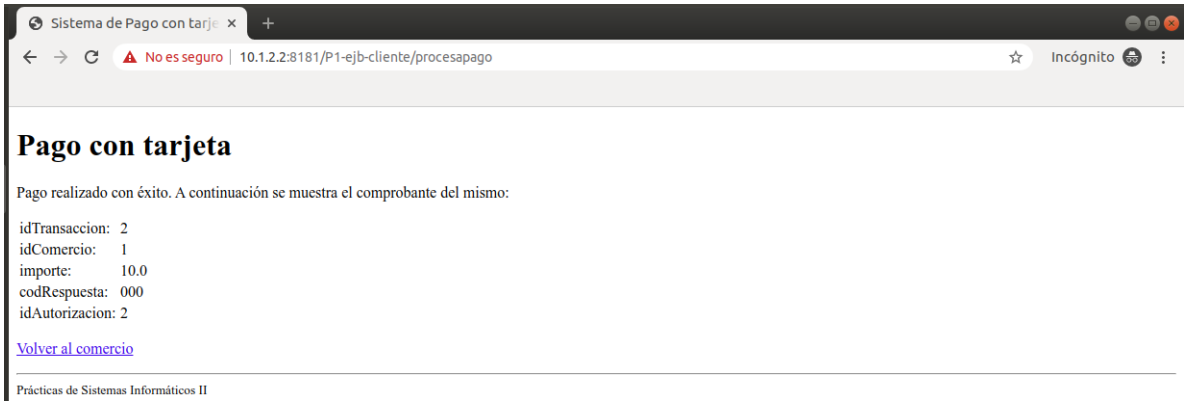
Ejercicio 4

Accedemos a la aplicación para comprobar su correcto funcionamiento.

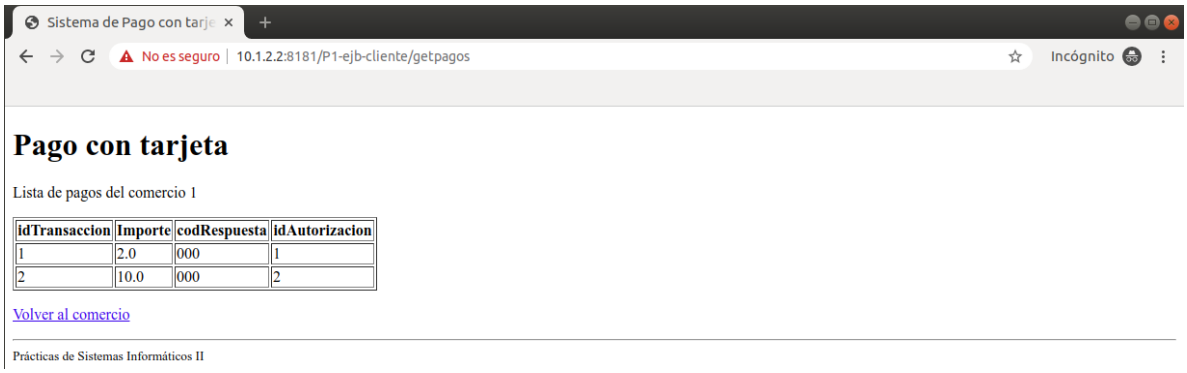
Realizamos un pago mediante `pago.html` y vemos que se realiza correctamente.



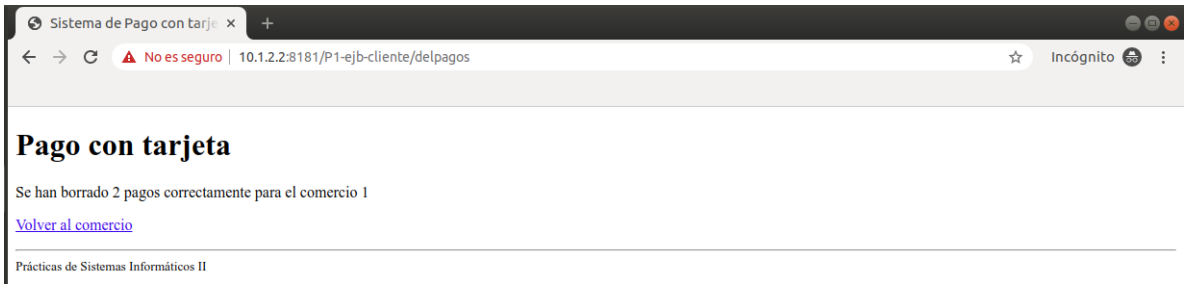
Realizamos ahora uno a través de `testbd.jsp`, sin *directconnection* y vemos de nuevo que funciona correctamente.



Listamos ahora los pagos del comercio 1 para ver que se han guardado en la base de datos correctamente:



Por último eliminamos los pagos y vemos que la acción se ejecuta sin problemas.



Como todas estas acciones han funcionado como se esperaba, deducimos que la aplicación se ha desplegado de forma correcta.

Ejercicio 5

Realizamos los cambios necesarios en el código para implementar la invocación remota de los métodos de los EJB.

Primero creamos la clase *VisaDAORemote* copiando *VisaDAOLocal* y cambiando el nombre de la interfaz a *VisaDAORemote* y la anotación `@Local` a `@Remote` para que implemente la invocación remota. Para esto es también necesario realizar el import de `Remote` de `java.ejb`. El código por tanto quedará:

```
19  import java.sql.Statement;
20  import java.util.ArrayList;
21  import javax.ejb.Remote;
22
23  @Remote
24  public interface VisaDAORemote {
25      public boolean compruebaTarjeta(TarjetaBean tarjeta);
26      public PagoBean realizaPago(PagoBean pago);
```

Tras esto modificamos *VisaDAOBean* para que implemente también la interfaz *VisaDAORemote*:

```
27   */
28  @Stateless(mappedName="VisaDAOBean")
29  public class VisaDAOBean extends DBTester implements VisaDAOLocal, VisaDAORemote {
30
31      private boolean debug = false;
32
```

Por último modificamos *TarjetaBean* y *PagoBean* para que implementen la interfaz `Serializable` realizando el import correspondiente.

De este modo *PagoBean* quedaría:

```
8   package ssii2.visa;
9
0   import java.io.Serializable;
1
2   /**
3    *
4    * @author jaime
5    */
6   public class PagoBean implements Serializable{
7
8     private String idTransaccion;
9     private String idComercio;
0     private double importe;
```

Y *TarjetaBean*:

```
9   import java.io.Serializable;
0
1   public class TarjetaBean implements Serializable{
2
3       private String numero;
4       private String titular;
5       private String fechaEmision;
6       private String fechaCaducidad;
```

Ejercicio 6

Para implementar el cliente remoto, partimos de `P1-base` y eliminamos el directorio `ssii2/visa/dao` como se nos indica.

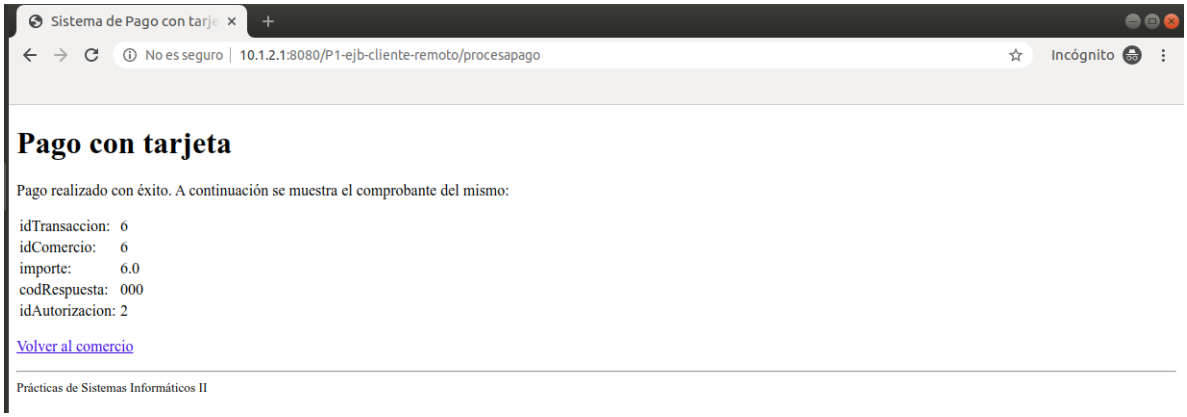
Modificamos *PagoBean* y *TarjetaBean* para que implementen la interfaz `Serializable` del mismo modo que en el ejercicio 5.

Copiamos la interfaz `VisaDAORemote` implementada en el servidor remoto en el ejercicio 5, y la pegamos en el cliente remoto en el directorio `P1-ejb-cliente-remoto/src/ssii2/visa` como se nos indica en el enunciado.

En los servlets eliminamos las declaraciones de `VisaDAO dao` y las sustituimos por referencias al objeto remoto EJB mediante la anotación `@EJB` del mismo modo que en el ejercicio 2, pero con el objeto remoto en lugar del local, realizando tambien los imports necesarios.

Creamos el fichero `glassfish-web.xml` en `web/WEB-INF` e introducimos en este las líneas que se nos indican en el enunciado que especifican las referencias a los EJBs remotos, introduciendo en el lugar adecuado la IP del servidor remoto, que en nuestro caso es `10.1.2.2`.

Desplegamos el cliente en la máquina virtual con IP `10.1.2.1` y realizamos un pago a través de `pago.html`.



Como se observa obtenemos el mensaje indicando que el pago se ha realizado de forma correcta y, por tanto, deducimos que el cliente está correctamente implementado y desplegado.

Ejercicio 7

En primer lugar, añadimos el atributo *saldo* y sus métodos de acceso a *TarjetaBean.java*:

```java
public class TarjetaBean {

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

Tras esto, modificamos el archivo *VisaDAOBean.java* para importar *EJBException*, y declaramos los dos *prepared statements* pedidos:

```java
import javax.ejb.Stateless;

import javax.ejb.EJBException;

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

```
/*************************************************/

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 = ?" ;
/**
```

Modificamos entonces el método *realizaPago*, que queda de la siguiente manera:

```
public synchronized PagoBean realizaPago(PagoBean pago) {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    boolean ret = false;
    String codRespuesta = "999"; // En principio, denegado

    // Utilizar en funcion de isPrepared()
    PreparedStatement pstmt = null;

    // Calcular pago.
    // Comprobar id.transaccion - si no existe,
    // es que la tarjeta no fue comprobada
    if (pago.getIdTransaccion() == null) {
        return null;
    }

    // Registrar el pago en la base de datos
    try {

        // Obtener conexion
        con = getConnection();

        // Obtener el saldo de la tarjeta

        String select  = SELECT_SALDO_QRY;
        errorLog(select);
        pstmt = con.prepareStatement(select);
        pstmt.setString(1, pago.getTarjeta().getNumero());
        rs = pstmt.executeQuery();
        if(rs.next()){
          double saldo = rs.getDouble("saldo");
          if (saldo < pago.getImporte()) {
              pago.setIdAutorizacion(null);
              ret = false;
          } else {
              saldo -= pago.getImporte();
              String update  = UPDATE_SALDO_QRY;
              errorLog(update);
              pstmt = con.prepareStatement(update);
              pstmt.setDouble(1, saldo);
              pstmt.setString(2, pago.getTarjeta().getNumero());
              ret = false;
              if (!pstmt.execute()
                      && pstmt.getUpdateCount() == 1) {
                  ret = true;
              }
          }
```

```
          } else {
            ret = false;
          }

          // Insertar en la base de datos el pago
          if (ret) {

          /* Usar prepared statement si
             isPrepared() == true */
          /*************************************************/
          if (isPrepared() == true) {
             String insert  = INSERT_PAGOS_QRY;
             errorLog(insert);
             pstmt = con.prepareStatement(insert);
             pstmt.setString(1, pago.getIdTransaccion());
             pstmt.setDouble(2, pago.getImporte());
             pstmt.setString(3, pago.getIdComercio());
             pstmt.setString(4, pago.getTarjeta().getNumero());
             ret = false;
             if (!pstmt.execute()
                     && pstmt.getUpdateCount() == 1) {
                ret = true;
             }

          } else {
             /*************************************************/
             stmt = con.createStatement();
             String insert = getQryInsertPago(pago);
             errorLog(insert);
             ret = false;
             if (!stmt.execute(insert)
                     && stmt.getUpdateCount() == 1) {
                ret = true;
             }
          }/***************/

          // Obtener id.autorizacion
          if (ret) {
              /* Permitir usar prepared statement si
               * isPrepared() = true */
              /*************************************************/
              if (isPrepared() == true) {
                  select = SELECT_PAGO_TRANSACCION_QRY;
                  errorLog(select);
                  pstmt = con.prepareStatement(select);
                  pstmt.setString(1, pago.getIdTransaccion());
                  pstmt.setString(2, pago.getIdComercio());
                  rs = pstmt.executeQuery();
              } else {
                  /*************************************************/
```

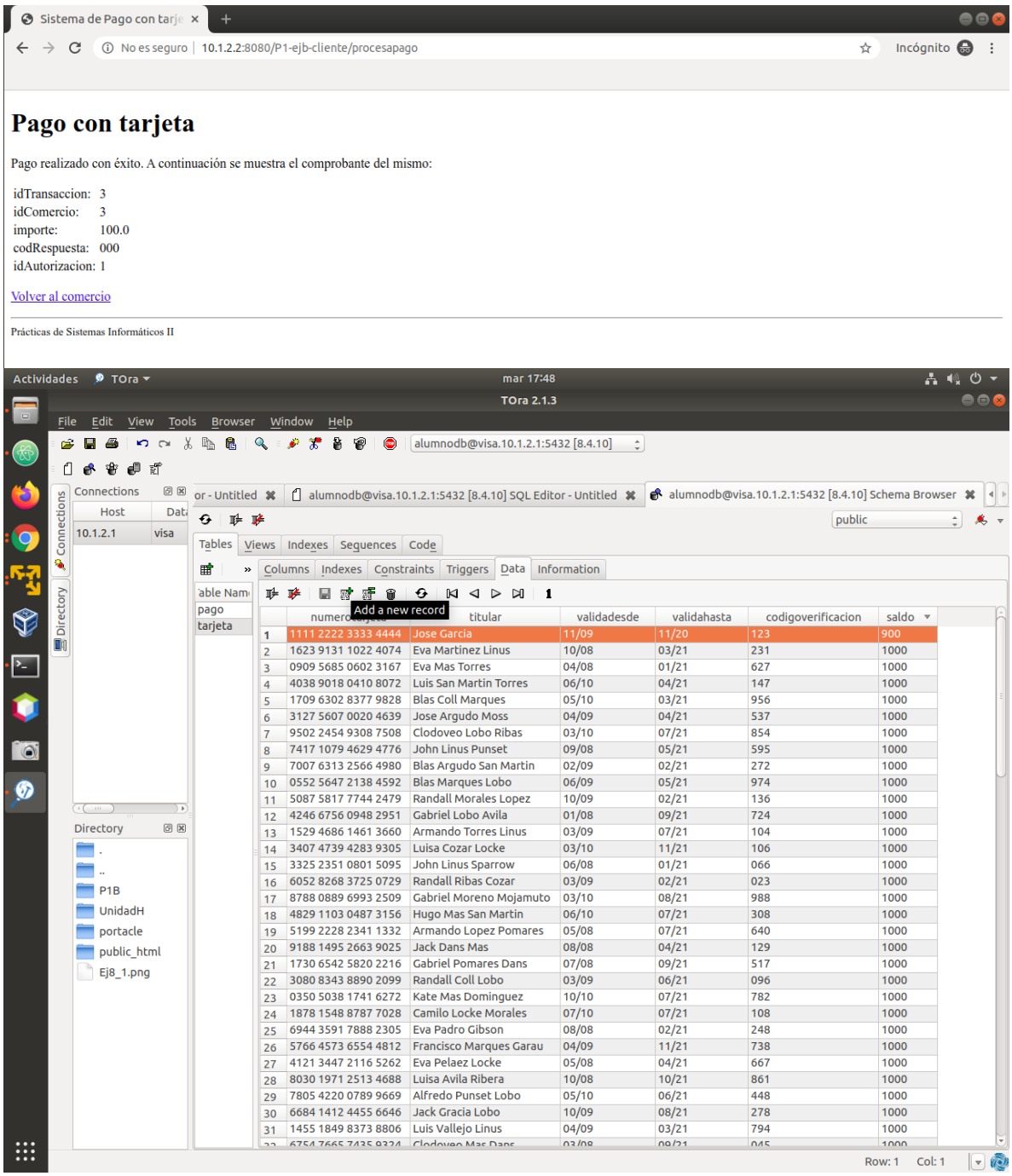
```
                    select = getQryBuscaPagoTransaccion(pago);
                    errorLog(select);
                    rs = stmt.executeQuery(select);

                }/**********************************/
                if (rs.next()) {
                    pago.setIdAutorizacion(String.valueOf(rs.getInt("idAutorizacion")));
                    pago.setCodRespuesta(rs.getString("codRespuesta"));
                } else {
                    ret = false;
                }
            }
          }

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

        if (ret == false){
          throw new EJBException("Pago incorrecto");
        }
        return pago;
    }
```

Tras esto, modificamos el servlet *ProcesaPago* para que capte la nueva posible interrupción *EJBException*:

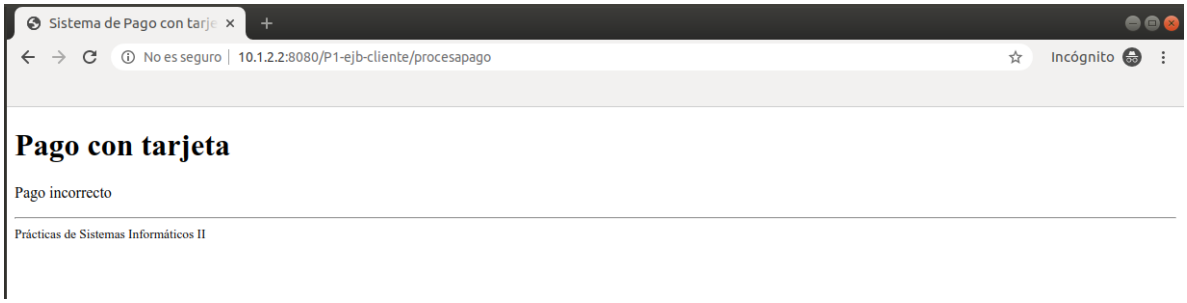
```
try{
  pago = dao.realizaPago(pago);
} catch (EJBException e) {
    errorLog(e.toString());
    if (sesion != null) sesion.invalidate();
    enviaError(new Exception("Pago incorrecto"), request, response);
    return;
}
```

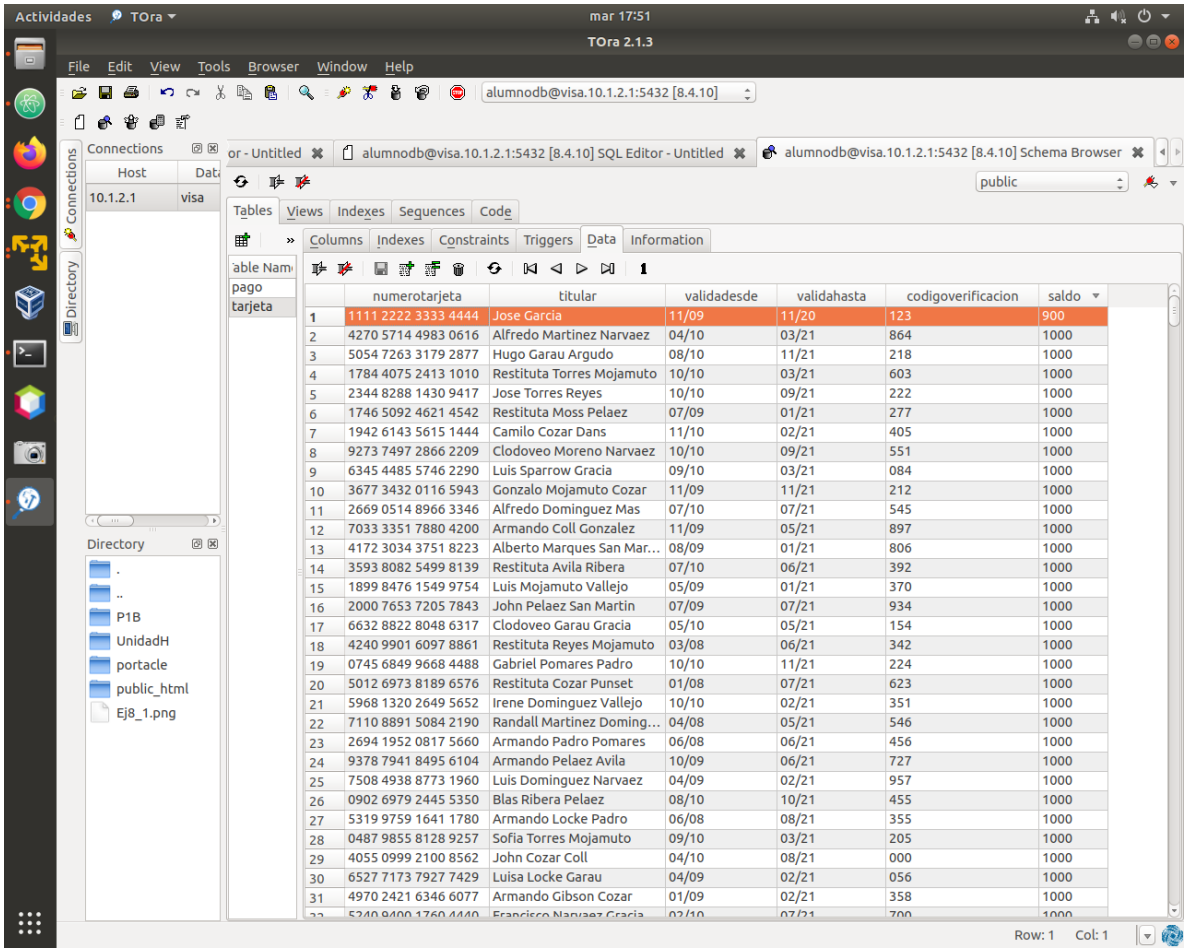
Ejercicio 8

En primer lugar, probamos a hacer un pago correcto y apreciamos en Tora como se ha actualizado el saldo:



Tras esto, probamos a hacer una operación con id de transacción y comercio duplicados y vemos que el saldo no varía, como era de esperar:

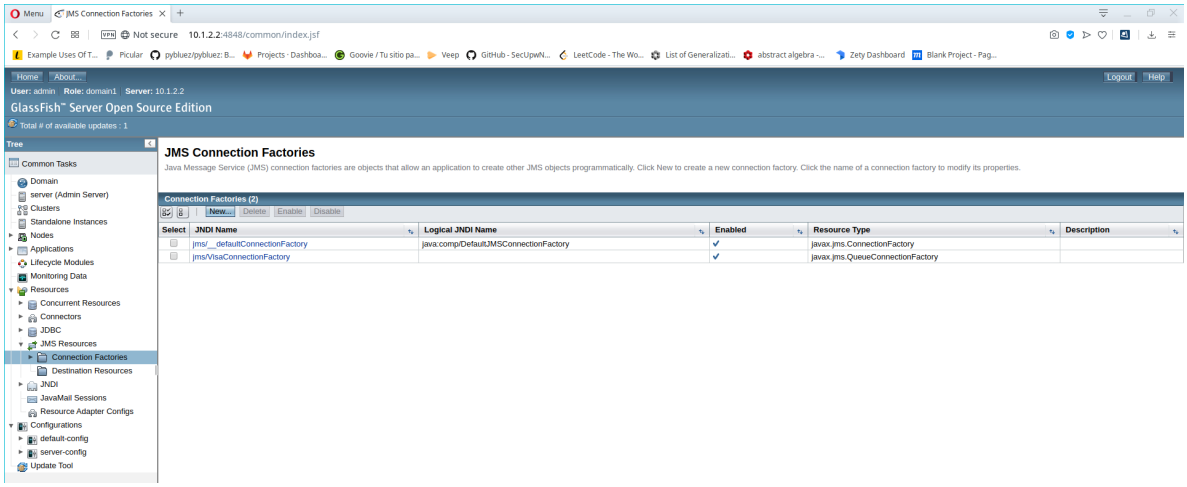




	numerotarjeta	titular	validadesde	validahasta	codigoverificacion	saldo
1	1111 2222 3333 4444	Jose Garcia	11/09	11/20	123	900
2	4270 5714 4983 0616	Alfredo Martinez Narvaez	04/10	03/21	864	1000
3	5054 7263 3179 2877	Hugo Garau Argudo	08/10	11/21	218	1000
4	1784 4075 2413 1010	Restituta Torres Mojamuto	10/10	03/21	603	1000
5	2344 8288 1430 9417	Jose Torres Reyes	10/10	09/21	222	1000
6	1746 5092 4621 4542	Restituta Moss Pelaez	07/09	01/21	277	1000
7	1942 6143 5615 1444	Camilo Cozar Dans	11/10	02/21	405	1000
8	9273 7497 2866 2209	Clodoveo Moreno Narvaez	10/10	09/21	551	1000
9	6345 4485 5746 2290	Luis Sparrow Gracia	09/10	03/21	084	1000
10	3677 3432 0116 5943	Gonzalo Mojamuto Cozar	11/09	11/21	212	1000
11	2669 0514 8966 3346	Alfredo Dominguez Mas	07/10	07/21	545	1000
12	7033 3351 7880 4200	Armando Coll Gonzalez	11/09	05/21	897	1000
13	4172 3034 3751 8223	Alberto Marques San Mar...	08/09	01/21	806	1000
14	3593 8082 5499 8139	Restituta Avila Ribera	07/10	06/21	392	1000
15	1899 8476 1549 9754	Luis Mojamuto Vallejo	05/09	01/21	370	1000
16	2000 7653 7205 7843	John Pelaez San Martin	07/09	07/21	934	1000
17	6632 8822 8048 6317	Clodoveo Garau Gracia	05/10	05/21	154	1000
18	4240 9901 6097 8861	Restituta Reyes Mojamuto	03/08	06/21	342	1000
19	0745 6849 9668 4488	Gabriel Pomares Padro	10/10	11/21	224	1000
20	5012 6973 8189 6576	Restituta Cozar Punset	01/08	07/21	623	1000
21	5968 1320 2649 5652	Irene Dominguez Vallejo	10/10	02/21	351	1000
22	7110 8891 5084 2190	Randall Martinez Doming...	04/08	05/21	546	1000
23	2694 1952 0817 5660	Armando Padro Pomares	06/08	06/21	456	1000
24	9378 7941 8495 6104	Armando Pelaez Avila	10/09	06/21	727	1000
25	7508 4938 8773 1960	Luis Dominguez Narvaez	04/09	02/21	957	1000
26	0902 6979 2445 5350	Blas Ribera Pelaez	08/10	10/21	455	1000
27	5319 9759 1641 1780	Armando Locke Padro	06/08	08/21	355	1000
28	0487 9855 8128 9257	Sofia Torres Mojamuto	09/10	03/21	205	1000
29	4055 0999 2100 8562	John Cozar Coll	04/10	08/21	000	1000
30	6527 7173 7927 7429	Luisa Locke Garau	04/09	02/21	056	1000
31	4970 2421 6346 6077	Armando Gibson Cozar	01/09	02/21	358	1000

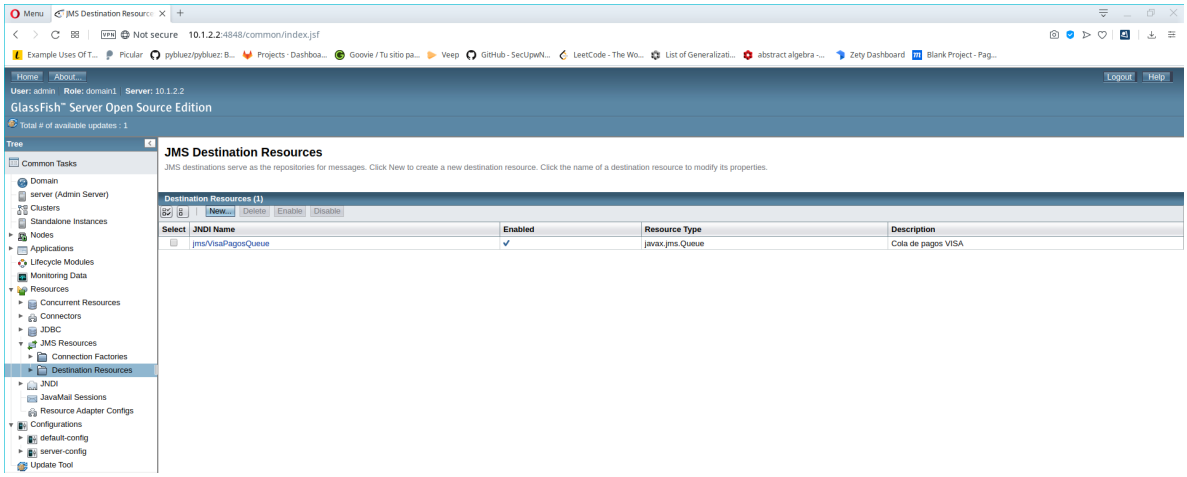
Ejercicio 9

Declaramos en la máquina *10.1.2.2* la factoría de conexiones como se indica:



Ejercicio 10

De nuevo, declaramos la cola de mensajes en la máquina *10.1.2.2* como se indica en el enunciado:



Ejercicio 11

En primer lugar modificamos el fichero sun-ejb-jar.xml como se nos indica en el enunciado, estableciendo que el nombre de la "connection factory" sea "jms/VisaConnectionFactory".

```
<?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>

```

Tras esto modificamos el fichero `VisaCancelacionJMSBean.java`. En este, implementamos las consultas SQL tanto para cambiar el código de respuesta de una transacción a 999, como para actualizar el saldo de la tarjeta tras modificar el pago. Establecemos ambas como "prepared statements" que serán invocados posteriormente en el método `onMessage()`.

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

private static final String UPDATE_SALDO_QRY =
               "update tarjeta" +
               " set saldo = saldo + pago.importe" +
               " from pago" +
               " where tarjeta.numeroTarjeta=pago.numeroTarjeta and pago.idAutorizacion = ?" ;
```

Por último modificamos el método `onMessage()` haciendo que implemente ambas consultas SQL como prepared statements, fijándonos en el código realizado para implementar prepared statements del archivo `VisaDAOBean.java` de `P1-ejb-transaccional` de ejercicios anteriores.

```
public void onMessage(Message inMessage) {
    TextMessage msg = null;
    PreparedStatement pstmt = null;
    Connection con = null;

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

            con = getConnection();

            String update = UPDATE_RESPUESTA_CODE;

            pstmt = con.prepareStatement(update);
            pstmt.setInt(1, idAutorizacion);
            if (!(!pstmt.execute() && pstmt.getUpdateCount() == 1)){
              logger.warning("Error while update response code");
            }

            update = UPDATE_SALDO_QRY;

            pstmt = con.prepareStatement(update);
            pstmt.setInt(1, idAutorizacion);
            if (!(!pstmt.execute() && pstmt.getUpdateCount() == 1)){
              logger.warning("Error while update cash");
            }

        } else {
            logger.warning(
                    "Message of wrong type: "
                    + inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        e.printStackTrace();
        mdc.setRollbackOnly();
    } catch (Throwable te) {
        te.printStackTrace();
        mdc.setRollbackOnly();
        logger.warning(te.getMessage());
    } finally {
          try {
              if (pstmt != null) {
                  pstmt.close(); pstmt = null;
              }
              if (con != null) {
                  closeConnection(con); con = null;
              }
          } catch (SQLException e) {
            logger.warning(e.getMessage());
          }
      }
}
```

Como se observa en la imagen, en este método llevamos también a cabo el control de errores mediante varios `catch` que controlan las distintas excepciones que pueden ser lanzadas, y mediante varios `if` que comprueban que las sentencias SQL se han ejecutado correctamente en la base de datos.

Ejercicio 12

La ventaja de usar el método basado en recursos JMS dinámicos en vez de en estáticos es principalmente que al poder establecer los nombres de la" connection factory" y de la cola en tiempo de ejecución, puedes usar un servidor externo para obtener dichos nombres, de forma que te permite por ejemplo añadir más colas sin modificar la aplicación, redistribuyendo los clientes.

Las modificaciones hechas en el archivo son las siguientes:

```
15  public class VisaQueueMessageProducer {
16
17      @Resource(mappedName = "jms/VisaConnectionFactory")
18      private static ConnectionFactory connectionFactory;
19      @Resource(mappedName = "jms/VisaPagosQueue")
20      private static Queue queue;
21
22      // Método de prueba
23      public static void browseMessages(Session session)
24      {
25        try
26        {
27          Enumeration messageEnumeration;
```

```
65
66        try {
67          /*
68          InitialContext jndi = new InitialContext();
69          connectionFactory = (ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");
70          queue = (Queue)jndi.lookup("jms/VisaPagosQueue");
71          */
72
73          connection = connectionFactory.createConnection();
74          session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
75          if (args[0].equals("-browse")) {
76            browseMessages(session);
77          } else {
```

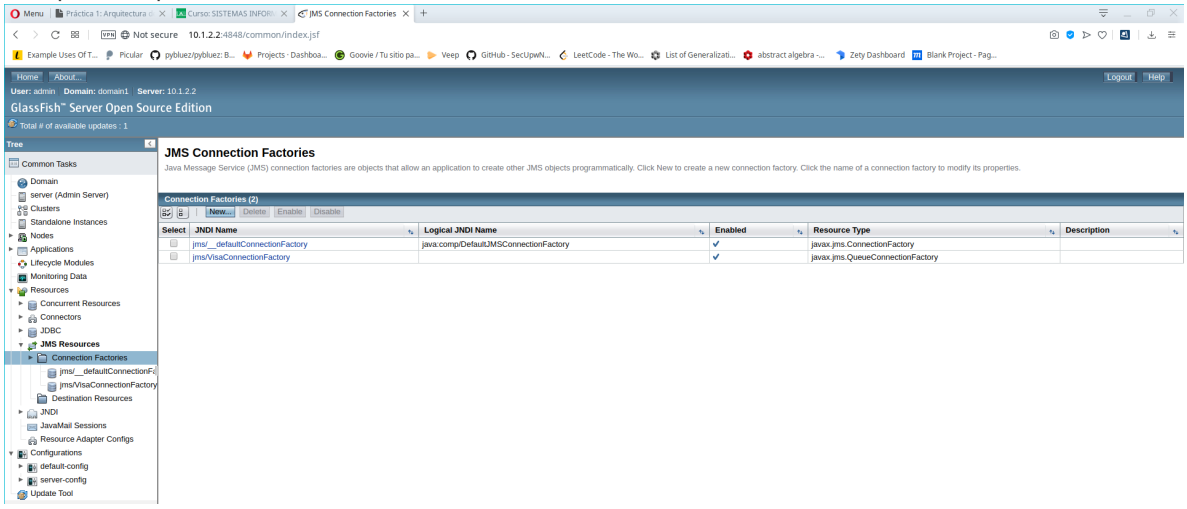
Ejercicio 13

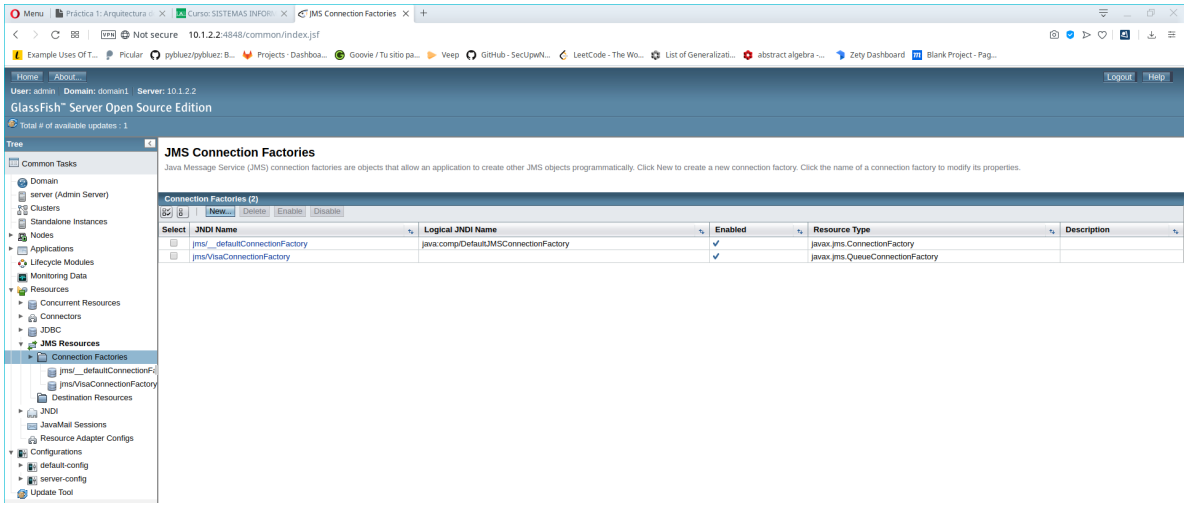
Añadimos a los campos *as.host.mdb* y *as.host.server* la IPs *10.1.2.2* porque es la máquina en la que se encuentra tanto el servidor como las colas de mensajes.

Tras esto entramos en la consola de administración de Glassfish y borramos manualmente la "connection factory".
Ejecutamos los comandos:

```
$ cd P1-jms
$ ant todo
```

Como se nos indica en el enunciado, para que la "connection factory" y la cola se generen de forma automática, tras lo cual entramos en la consola de administración de Glassfish para comprobar que la acción se ha realizado correctamente.





Como se observa en las imágenes, ambos recursos se han generado de forma correcta.

Revisando el fichero *jms.xml* podemos ver que para crear la cola JMS se utiliza:

```
<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 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>
```

Por tanto, el comando para crear la cola sería

```
asadmin --user admin --paswordfile ./passwordfile --host 10.1.2.2 --port 4848
create-jms-resource --restype javax.jms.Queue --enabled=true --property
Name=VisaPagosQueue jms/VisaPagosQueue
```

Ejercicio 14

En primer lugar modificamos *VisaQueueMessageProducer.java* para enviar *args[0]* como mensaje de texto, de forma que el código queda:

```
75          if (args[0].equals("-browse")) {
76            browseMessages(session);
77          } else {
78            messageProducer = session.createProducer(queue);
79            message = session.createTextMessage();
80
81            message.setText(args[0]);
82            System.out.println("Enviando el siguiente mensaje: " + message.getText());
83            messageProducer.send(message);
84            messageProducer.close();
85            session.close();
86          }
87        } catch (Exception e) {
88            System.out.println("Excepcion : " + e.toString());
89        } finally {
```

Tras esto, detenemos la ejecución de la aplicación *P1-jms-mdb* y una vez ejecutamos `/opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -client dist/clientjms/P1-jms-clientjms.jar 1` para enviar a la cola el id de transacción uno, la salida con `/opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -client dist/clientjms/P1-jms-clientjms.jar -browse` es la siguiente:

```
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -
client /tmp/P1-jms-clientjms.jar 1
Mar 15, 2020 9:55:25 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 15, 2020 9:55:25 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 15, 2020 9:55:25 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 15, 2020 9:55:25 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: 1
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -
client /tmp/P1-jms-clientjms.jar -browse
Mar 15, 2020 9:56:09 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 15, 2020 9:56:09 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 15, 2020 9:56:09 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 15, 2020 9:56:09 AM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensajes en cola:
1
si2@si2srv01:~$
```

Esto demuestra que el mensaje se ha enviado correctamente.

Modificamos entonces la variable *default_JMS_host* estbleciendo como dirección IP del host la *10.1.2.2*, donde está desplegada la cola de mensajes. Activamos la aplicación *P1-jms-mdb*, reiniciamos el servidor y verificamos que la cola está vacía con el último comando expuesto:

```
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -
client /tmp/P1-jms-clientjms.jar -browse
Error: Unable to access jarfile /tmp/P1-jms-clientjms.jar
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -
client /tmp/P1-jms-clientjms.jar -browse
Mar 15, 2020 12:41:00 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 15, 2020 12:41:00 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 15, 2020 12:41:00 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 15, 2020 12:41:00 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Cola de mensajes vac?a!
si2@si2srv01:~$
```

Tras esto, realizamos un pago con la web y vemos que se ha hecho correctamente:

Pago con tarjeta

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

idTransaccion: 4
idComercio: 4
importe: 100.0
codRespuesta: 000
idAutorizacion: 2

Volver al comercio

Prácticas de Sistemas Informáticos II

Usamos el cliente para cancelarlo y posteriormente comprobamos que se ha cancelado correctamente, pues la cola está vacía y la información de la base de datos está actualizada (el saldo y el estado del pago con id 2):

```
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -
client /tmp/P1-jms-clientjms.jar 2
Mar 15, 2020 12:45:41 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 15, 2020 12:45:41 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 15, 2020 12:45:41 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 15, 2020 12:45:41 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: 2
si2@si2srv01:~$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1.2.2 -
client /tmp/P1-jms-clientjms.jar -browse
Mar 15, 2020 12:46:16 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 15, 2020 12:46:17 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 15, 2020 12:46:17 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 15, 2020 12:46:17 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Cola de mensajes vac?a!
si2@si2srv01:~$
```