

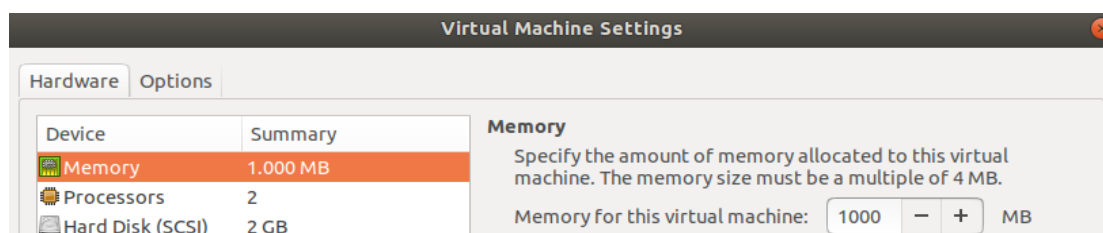
		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2361	Práctica	1A	Fecha	24/02/2020
Alumno/a		Sierra, Alonso, David			
Alumno/a		Truchado, Mazzoli, Irene			

Práctica 1A: Arquitectura de JAVA EE (Primera parte)

Ejercicio 1:

Para la realización de este ejercicio, el primer paso es descomprimir la máquina virtual entregada y abrirla utilizando *VMware player*.

Tal y cómo se pide en el enunciado, utilizaremos una máquina virtual con 1GB de RAM y 2 CPU's:



Cabe destacar que la primera vez que utilizamos dicha máquina, debemos generar direcciones MAC aleatorias tanto para el Network Adapter (NAT) como para el Network Adapter 2 (Bridged). Esto lo hacemos para evitar conflictos de duplicación de direcciones en las imágenes virtuales de los PC del laboratorio.

Así mismo, en el primer uso de esta máquina debemos configurar nuestra dirección IP única, utilizando el grupo y la pareja. En nuestro caso, hemos obtenido las siguientes direcciones IP:

- David 10.7.13.1
- Irene 10.7.13.2

Tras estas configuraciones iniciales, volvemos a la terminal del PC host. Debido al bajo rendimiento de la consola de VMware, durante el desarrollo de la práctica utilizaremos un acceso SSH contra dicha máquina virtual.

En primer lugar, declaramos la variable de entorno *J2EE_HOME* con el directorio de instalación de Glassfish. Ahora bien, ejecutamos el comando *ifconfig*:

```
e356372@11-2-11-2:~$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.250.11.2 netmask 255.255.255.0 broadcast 10.250.11.255
    inet6 fe80::42b0:34ff:fe99:df9f prefixlen 64 scopeid 0x20<link>
    ether 40:b0:34:f9:df:9f txqueuelen 1000 (Ethernet)
    RX packets 745420 bytes 1093639384 (1.0 GB)
    RX errors 0 dropped 3 overruns 0 frame 0
    TX packets 302967 bytes 21020669 (21.0 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Observamos que nuestra dirección IP inicial es la 10.250.11.2.

Utilizando el comando `sudo /opt/si2/virtualip.sh enp1s0`, asignamos a la interfaz de red del Host una dirección IP en el rango 10.X.Y.Z y comprobamos de nuevo con `ifconfig`:

```
e356372@11-2-11-2:~$ sudo /opt/si2/virtualip.sh enp1s0
[sudo] contraseña para e356372:
Dirección 10.10.11.2 asignada interfaz enp1s0:0
e356372@11-2-11-2:~$ ifconfig
enp1s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.250.11.2 netmask 255.255.255.0 broadcast 10.250.11.255
    inet6 fe80::42b0:34ff:fef9:df9f prefixlen 64 scopeid 0x20<link>
    ether 40:b0:34:f9:df:9f txqueuelen 1000 (Ethernet)
    RX packets 745498 bytes 1093649617 (1.0 GB)
    RX errors 0 dropped 3 overruns 0 frame 0
    TX packets 303040 bytes 21029674 (21.0 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp1s0:0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.11.2 netmask 255.0.0.0 broadcast 10.255.255.255
    ether 40:b0:34:f9:df:9f txqueuelen 1000 (Ethernet)
```

Ahora ya tenemos asignada a la interfaz de red del host una dirección IP en el rango deseado.

Además, podemos comprobar mediante el comando `ping` que la dirección IP establecida anteriormente es alcanzable:

```
e356372@15-8-15-8:~$ ping 10.7.13.2
PING 10.7.13.2 (10.7.13.2) 56(84) bytes of data.
64 bytes from 10.7.13.2: icmp_seq=1 ttl=64 time=0.683 ms
64 bytes from 10.7.13.2: icmp_seq=2 ttl=64 time=0.540 ms
64 bytes from 10.7.13.2: icmp_seq=3 ttl=64 time=0.544 ms
64 bytes from 10.7.13.2: icmp_seq=4 ttl=64 time=0.606 ms
```

Por último, cambiamos los valores por defecto de los ficheros `build.properties` y `postgresql.properties` del proyecto, de manera que utilicen nuestra nueva IP.

Ahora ya podemos compilar con `ant`, arrancar el servidor y acceder a él. Si entramos a la consola de administración en el puerto 4848, observamos que aparece la aplicación de nuestro proyecto:

← → ↻ No es seguro | 10.7.13.2:4848/common/index.jsf

Home About...

User: admin Domain: domain1 Server: 10.7.13.2

GlassFish™ Server Open Source Edition

Common Tasks

- Domain
 - server (Admin Server)
- Clusters
 - Standalone Instances
- Nodes
 - Applications**
 - Lifecycle Modules

Applications

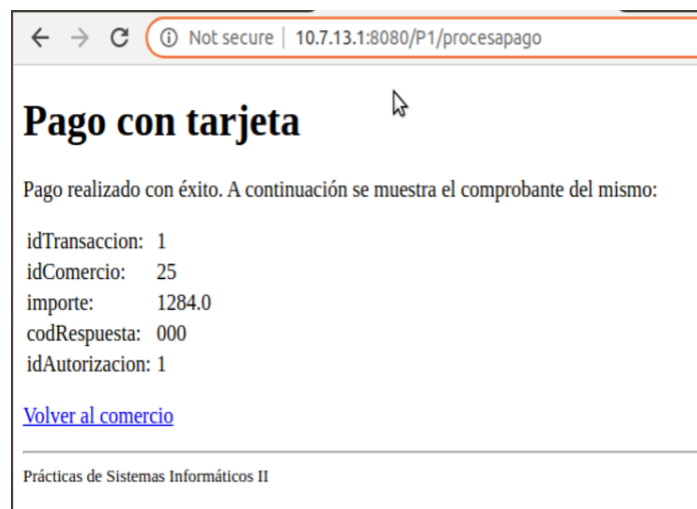
Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will application or module is enabled on.

Deployed Applications (1)

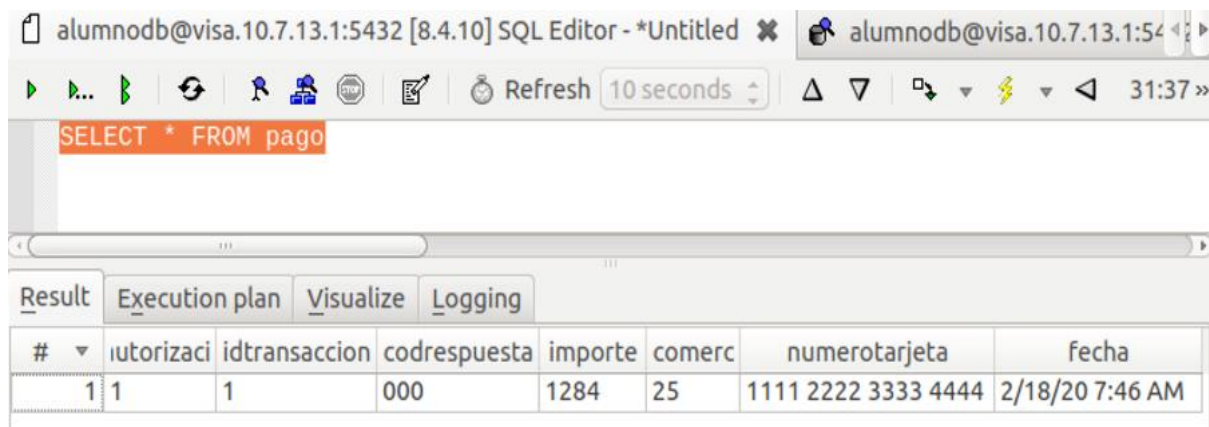
☒ ☐ Filter: ▾

Select	Name	Deployment Order	Enabled	Engines	Action
<input type="checkbox"/>	P1	100	✓	web	Launch Redeploy Reload

Al desplegar la aplicación en el puerto 8080, podemos realizar un pago.

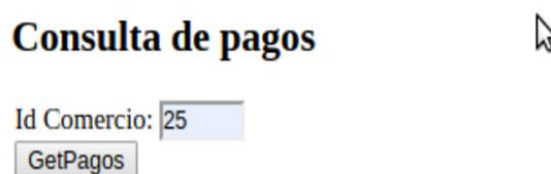


Utilizando el gestor de bases de datos *Tora*, somos capaces de comprobar en la base de datos que dicho pago efectivamente se ha realizado:



Finalmente, podemos acceder a la página de pruebas extendida *testbd.jsp* y comprobar la funcionalidad de listado y de borrado de pagos.

Consultamos el pago con el id de comercio utilizado anteriormente:



Y observamos que el pago aparece listado:



Ahora borramos dicho pago utilizando el mismo id:

Borrado de pagos

Id Comercio:

El sistema nos informa de que la operación ha sido exitosa:



Si volvemos a utilizar la funcionalidad de listado, comprobamos que el pago ya no aparece:



Ejercicio 2:

En este ejercicio, debemos completar la información necesaria para que la clase *VisaDAO* lleve a cabo la conexión directa de modo correcto. Para ello, y consultando el anexo 10 de la práctica, hemos realizado algunas modificaciones en la clase *DBTester*. Hemos redefinido las constantes `JDBC_CONNSTRING`, `JDBC_USER`, `JDBC_PASSWORD`, las cuales corresponden a la cadena de conexión JDBC, el usuario y la clave necesarios para obtener nuevas conexiones a través de la clase *DriverManager* tal y como se aprecia en la captura.

```
/**
 *
 * @author jaime
 */
public class DBTester {

    // Información de conexión
    // Para conexiones directas, requerimos: driver, cadena de conexión,
    // usuario y clave
    private static final String JDBC_DRIVER =
        "org.postgresql.Driver";

    // TODO: Definir la cadena de conexión a la base de datos
    //*****
    private static final String JDBC_CONNSTRING =
        "jdbc:postgresql://10.7.13.1:5432/visa";
    //*****
    private static final String JDBC_USER = "alumnodb";
    private static final String JDBC_PASSWORD = "";

    // Para conexión por datasource, sólo necesitamos su nombre
    // TODO: Definir el nombre del datasource
    //*****
    private static final String JDBC_DSN =
        "jdbc/VisaDB";
    //*****
}
```

Tras ello, realizamos un *ant limpiar-todo*, *ant replegar*, *ant unsetup-db* y finalmente un *ant todo*. Con esto visualizamos la página <http://10.7.13.1:8080/P1/testbd.jsp> para así realiza una prueba.

Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="2"/>
Id Comercio:	<input type="text" value="25"/>
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="11/21"/>
CVV2:	<input type="text" value="123"/>
Modo debug:	<input type="radio"/> True <input type="radio"/> False
Direct Connection:	<input checked="" type="radio"/> True <input type="radio"/> False
Use Prepared:	<input type="radio"/> True <input type="radio"/> False
<input type="button" value="Pagar"/>	

Pago con tarjeta

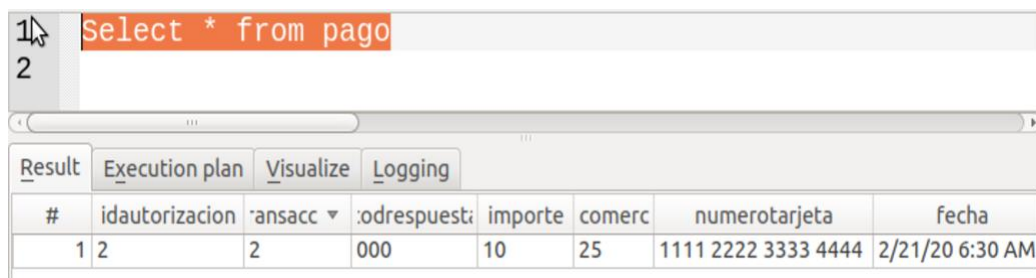
Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 2
idComercio: 25
importe: 10.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Utilizando el gestor de bases de datos *Tora*, somos capaces de comprobar en la base de datos que dicho pago efectivamente se ha realizado:



The screenshot shows the Tora SQL editor with the query `select * from pago` entered. Below the query, the results are displayed in a table with columns: #, idautorizacion, ansacc, codrespuesta, importe, comerc, numerotarjeta, and fecha. The results show one record with idautorizacion 2, importe 10, and fecha 2/21/20 6:30 AM.

#	idautorizacion	ansacc	codrespuesta	importe	comerc	numerotarjeta	fecha
1	2	2	000	10	25	1111 2222 3333 4444	2/21/20 6:30 AM

Finalmente, podemos acceder a la página de pruebas extendida *testbd.jsp* y comprobar la funcionalidad de listado y de borrado de pagos.

Consultamos el pago con el id de comercio utilizado anteriormente:

Consulta de pagos

Id Comercio:	<input type="text" value="25"/>
<input type="button" value="GetPagos"/>	

Y observamos que el pago aparece listado:

Pago con tarjeta

Lista de pagos del comercio 25

idTransaccion	Importe	codRespuesta	idAutorizacion
2	10.0	000	2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Ahora borramos dicho pago utilizando el mismo id:

Borrado de pagos

Id Comercio:

El sistema nos informa de que la operación ha sido exitosa:

← → ↻ ⓘ Not secure | 10.7.13.1:8080/P1/delpagos

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 25

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Si volvemos a utilizar la funcionalidad de listado, comprobamos que el pago ya no aparece:

← → ↻ ⓘ Not secure | 10.7.13.1:8080/P1/getpagos

Pago con tarjeta

Lista de pagos del comercio 25

idTransaccion	Importe	codRespuesta	idAutorizacion
---------------	---------	--------------	----------------

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Ejercicio 3:

Para llevar a cabo este ejercicio, lo primero que hacemos es examinar el archivo *postgresql.properties* para determinar el nombre del recurso JDBC correspondiente al DataSource y el nombre del pool.

```
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
```


- **Timeout Settings**

- **Max Wait Time**

Tiempo que el cliente espera hasta llegar al timeout, en nuestro caso 300 segundos. Para mejoras en el rendimiento, se recomienda establecer este valor a 0, de modo que el cliente esperará indefinidamente, hasta que se le conceda acceso a la conexión.

- **Idle Timeout**

Tiempo máximo en milisegundos durante el cual una conexión puede permanecer en el pool, después del cual se cerrará dicha conexión, en nuestro caso 60.000 segundos. Para mejor rendimiento, se debe establecer este valor por debajo del timeout, para prevenir la acumulación de conexiones no utilizadas en el servidor. Es más, se recomienda establecer dicho valor a 0, de manera que no se eliminen las conexiones ociosas y se eviten las penalizaciones por la creación de nuevas conexiones.

En general, a menor tamaño de pool, el establecimiento de la conexión será más rápido, aunque si el número es muy bajo pueden no satisfacerse todas las peticiones o tener que esperar mucho. Por otro lado, un tamaño grande de pool permite satisfacer más peticiones, pero el acceso a la tabla de conexiones será más lento.

Fuente: <https://docs.oracle.com/cd/E19159-01/819-3681/abehq/index.html>

Ejercicio 4:

En la clase *VisaDAO* encontramos el código SQL que comprueba si una tarjeta existe en la base de datos. Esto lo realiza la función `public boolean compruebaTarjeta(TarjetaBean tarjeta)`, la cual utiliza la siguiente consulta:

```
/**
 * getQryCompruebaTarjeta
 */
String getQryCompruebaTarjeta(TarjetaBean tarjeta) {
    String qry = "select * from tarjeta "
        + "where numeroTarjeta='" + tarjeta.getNumero()
        + "' and titular='" + tarjeta.getTitular()
        + "' and validaDesde='" + tarjeta.getFechaEmision()
        + "' and validaHasta='" + tarjeta.getFechaCaducidad()
        + "' and codigoVerificacion='" + tarjeta.getCodigoVerificacion() + "'";
    return qry;
}
```

También en la clase *VisaDAO* encontramos la función `public synchronized boolean realizaPago(PagoBean pago)`, la cual se encarga de la ejecución del pago a través de la siguiente consulta:

```
/**
 * getQryInsertPago
 */
String getQryInsertPago(PagoBean pago) {
    String qry = "insert into pago("
        + "idTransaccion,"
        + "importe,idComercio,"
        + "numeroTarjeta)"
        + " values ("
        + "'" + pago.getIdTransaccion() + "',"
        + pago.getImporte() + ","
        + "'" + pago.getIdComercio() + "',"
        + "'" + pago.getTarjeta().getNumero() + "'"
        + ")";
    return qry;
}
```


Ejercicio 5:

Encontramos el método `errorLog` dentro de la clase `VisaDAO`:

```
/**
 * Imprime traza de depuracion
 */
public void errorLog(String error) {
    if (isDebug())
        System.err.println("[directConnection=" + this.isDirectConnection() + "] " +
                           error);
}
```

Dicho método devuelve información de depuración en caso de que la variable `debug` valga `true`. A lo largo del código, se utiliza como control de errores tras la ejecución de queries y excepciones. En concreto, se utiliza en los métodos `compruebaTarjeta`, `realizaPago`, `getPagos` y `delPagos`.

A continuación, realizamos un pago con la opción de debug activada:

Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="1"/>
Id Comercio:	<input type="text" value="2"/>
Importe:	<input type="text" value="11"/>
Numero de visa:	<input type="text" value="0004 9839 0829 3274"/>
Titular:	<input type="text" value="Blas Avila Sparrow"/>
Fecha Emisión:	<input type="text" value="10/10"/>
Fecha Caducidad:	<input type="text" value="04/21"/>
CVV2:	<input type="text" value="227"/>
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 type="radio"/> True <input checked="" type="radio"/> False
<input type="button" value="Pagar"/>	

Y observamos que dicho pago se realiza correctamente:

Pago con tarjeta

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

idTransaccion: 1
idComercio: 2
importe: 11.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Al acceder a la consola de administración, podemos visualizar los logs del servidor, comprobando

que en esta página encontramos información adicional sobre las clases de nuestra aplicación:

Log Viewer

View, search, and filter a server log file using basic and advanced options. Refer to the Log Levels page for information about log levels you can filter here.

Search Close

Advanced Search

Search Criteria

Text search:

Only log entries containing the specified text will be displayed. Search is case sensitive.

Timestamp: ☒ Most Recent
☐ Specific Range:

Log Level: ☐ Do not include more severe messages

Log entries are limited to those stored in the log file. Set appropriate log level in the Log Level page to ensure data is logged.

Search Close

Modify Search

Instance:

Log File:

Log Viewer Results (40)						
Records before 640		Log File Record Numbers 640 through 679		Records after 679		⬆⬇⬆
Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs	
679	INFO	Admin Console: Initializing Session Attributes...(details)	org.glassfish.admingui	21-feb-2020 04:15:22.932	{levelValue=800, timeMillis=1582287322932}	
678	INFO	Redirecting to /index.jsf(details)	org.glassfish.admingui	21-feb-2020 04:15:22.779	{levelValue=800, timeMillis=1582287322779}	
677	INFO	Listening to REST requests at context: /management/domain.(details)	javax.enterprise.admin.rest	21-feb-2020 04:15:22.746	{levelValue=800, timeMillis=1582287322746}	
676	SEVERE	The SSL certificate has expired: [[Version: V3 Subject: CN=Entrust net Secure Server Certifica (details)	javax.enterprise.system.security.ssl	21-feb-2020 04:15:21.346	{levelValue=1000, timeMillis=1582287321346}	

Ejercicio 6:

Lo primero que vamos a añadir en el código de *VisaDAOWS.java* son las librerías Java necesarias para implementar las nuevas funcionalidades. Para ello importamos de la librería *javax.jws* las clases *WebServices*, *WebMethod* y *WebParam* tal y como se aprecia en la imagen.

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

Después, añadimos a la clase *VisaDAOWS* la anotación de *@WebService* para indicar que es un servicio web.

```
/**
 * @author jaime
 */
@WebService
public class VisaDAOWS extends DBTester {
```

Por otro lado, añadimos en las funciones que se comportarán como un servicio la anotación *@WebMethod* y las variables que se pasan por parámetro se le añade la anotación *@WebParam*.

```
/**
 * Comprobacion de la tarjeta
 * @param tarjeta Objeto con toda la informacion de la tarjeta
 * @return true si la comprobacion contra las tarjetas contenidas en
 * en la tabla TARJETA fue satisfactoria, false en caso contrario */
@WebMethod
public boolean compruebaTarjeta(@WebParam TarjetaBean tarjeta) {
```

```

/**
 * Realiza el pago
 * @param pago
 * @return
 */
@WebMethod
public synchronized PagoBean realizaPago(@WebParam PagoBean pago) {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    PagoBean ret = null;
}

```

```

/**
 * TODO: Metodos isPrepared() y setPrepared()
 */
@WebMethod
public boolean isPrepared() {
    return prepared;
}

@WebMethod
public void setPrepared(@WebParam boolean prepared) {
    this.prepared = prepared;
}

```

```

/**
 * @return the debug
 */
@WebMethod
public boolean isDebug() {
    return debug;
}

/**
 * @param debug the debug to set
 */
@WebMethod
public void setDebug(@WebParam boolean debug) {
    this.debug = debug;
}

```

Añadimos las funciones que se encuentran en la clase *DBTester* mencionadas en el enunciado con la anotación `@Override` para que se realice las funciones de la clase *VisaDAOWS* antes que la de la clase *DBTester* que es de la que extiende. Además se le añade la anotación de `@WebMethod` y a la variable que se pasa por parámetro se le añade la anotación `@WebParam` al ser un método con servicio.

```

/**
 * @return the pooled
 */
@Override
@WebMethod
public boolean isDirectConnection() {
    return super.isDirectConnection();
}

/**
 * @param directConnection valor de conexión directa o indirecta
 */
@Override
@WebMethod
public void setDirectConnection(@WebParam boolean directConnection) {
    super.setDirectConnection(directConnection);
}

```

Dentro de la función *realizaPago* realizamos las modificaciones que nos indican en el enunciado. Para ello cambiamos la clase de valor que retorna la función a *PagoBean*. También modificamos la variable **ret** para que sea una variable de tipo *PagoBean* y las asignaciones a *false* las reemplazamos por *null* y las *true* por **pago**.

```
/**
 * Realiza el pago
 * @param pago
 * @return
 */
@WebMethod
public synchronized PagoBean realizaPago(@WebParam PagoBean pago) {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    PagoBean ret = null;
    String codRespuesta = "999"; // En principio, denegado

    // TODO: 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;
    }
}
```

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

```
if (rs.next()) {
    pago.setIdAutorizacion(String
    pago.setCodRespuesta(rs.getSt
    } else {
        ret = null;
    }
}

} catch (Exception e) {
    errorLog(e.toString());
    ret = null;
}
```

¿Por qué se ha de alterar el parámetro de retorno del método realizaPago() para que devuelva el pago el lugar de un boolean?

Se realiza dicha modificación ya que es necesario devolver el objeto *PagoBean* y así obtener toda la información que se quiere mostrar por la página de la web.

Ejercicio 7:

¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?

En el fichero XML-schemas que se encuentra en la URL: <http://10.7.13.1:8080/P1-ws-ws/VisaDAOWSService?xsd1>

¿Qué tipos de datos predefinidos se usan?

Se usan: *xs:string*, *xs:boolean*, *xs:int* y *xs:double*.

¿Cuáles son los tipos de datos que se definen?

Se definen Clases Java con la anotación *tns:NombreClaseJava* y métodos con la anotación *tns:NombreMetodo*

¿Qué etiqueta está asociada a los métodos invocados en el webservice?

Se asocia a los métodos la etiqueta *operations*. Dentro tenemos las etiquetas *input* y *ouput* para identificar los mensajes de entrada y salida.

¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice?

La etiqueta *message* se utiliza. Dentro tenemos la etiqueta *part* que hace referencia a los parámetros del método

¿En qué etiqueta se especifica el protocolo de comunicación con el webservice?

En la etiqueta *binding* donde se indica mediante la etiqueta *soap:binding*.

¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice?

La etiqueta *soap:address* donde contiene la URL dentro de la campo *location*.

```
<service name="VisaDAOWSService">
  <port name="VisaDAOWSPort" binding="tns:VisaDAOWSPortBinding">
    <soap:address location="http://10.7.13.1:8080/P1-ws-ws/VisaDAOWSService"/>
  </port>
</service>
```

Ejercicio 8:

Inicialmente introducimos los import de las clases nuevas necesarias para el funcionamiento de *ProcesaPago.java* tal y como se aprecian en la imagen.


```
//import ssii2.visa.dao.VisaDAO;
import ssii2.visa.VisaDAOWSService;
import ssii2.visa.VisaDAOWS;
import javax.xml.ws.WebServiceRef;
import javax.xml.ws.BindingProvider;
```

Después añadimos la implementación mediante la llamada al servicio web mediante stubs estáticos con el siguiente código.

```
VisaDAOWS dao = null;
try{
    VisaDAOWSService service = new VisaDAOWSService();
    dao = service.getVisaDAOWSPort();
}catch (Exception e){
    enviaError(e, request, response);
    return;
}
```

Ejercicio 9:

Tal y como se menciona en el enunciado de la práctica modificamos primero el fichero *web.xml* en la que se añade un *context-param* para indicar la URL del servidor.

```
<context-param>
    <param-name>url</param-name>
    <param-value>http://10.7.13.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

Además añadimos las siguientes líneas en la clase java *ProcesaPago.java* para conseguir esta ruta y poder implementar la nueva funcionalidad. Para ello lo recuperamos con el valor introducido en el fichero *web.xml* *param-name* (*url*) como se aprecia en la imagen.

```
import javax.xml.ws.BindingProvider;
```

```
public final static String URL = "url";
```

```
VisaDAOWS dao = null;
try{
    VisaDAOWSService service = new VisaDAOWSService();
    dao = service.getVisaDAOWSPort();
    BindingProvider binding = (BindingProvider) dao;
    binding.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                                   getServletContext().getInitParameter(URL));
}catch (Exception e){
    enviaError(e, request, response);
    return;
}
```


Ejercicio 10:

Para que toda la funcionalidad de la página de pruebas *testbd.jsp* se realice a través del servicio web se deben modificar las clases Java *DelPagos.java* y *GetPagos.java*. Para ello introducimos el mismo código en que se ha realizado en los ejercicios 8 y 9.

```
//import ssii2.visa.dao.VisaDAO;  
import ssii2.visa.VisaDAOWSService;  
import ssii2.visa.VisaDAOWS;  
import javax.xml.ws.WebServiceRef;  
import javax.xml.ws.BindingProvider;
```

```
VisaDAOWS dao = null;  
try{  
    VisaDAOWSService service = new VisaDAOWSService();  
    dao = service.getVisaDAOWSPort();  
    BindingProvider binding = (BindingProvider) dao;  
    binding.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,  
                                   getServletContext().getInitParameter(URL));  
}catch (Exception e){  
    enviaError(e, request, response);  
    return;  
}
```

Además se deben añadir como se realizó en el ejercicio 6 las etiquetas *@WebMethod* y *@WebParam* para las funciones *getPagos()* y *delPagos()* de la clase Java *VisaDAOWS.java* tal y como se aprecia en la imagen.

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

```
/**  
 *  
 * @param idComercio  
 * @return numero de registros afectados  
 */  
@WebMethod  
public int delPagos(@WebParam String idComercio) {
```

Por otro lado, modificamos el resultado retornado en la función *getPagos()* tal y como recomienda el enunciado.

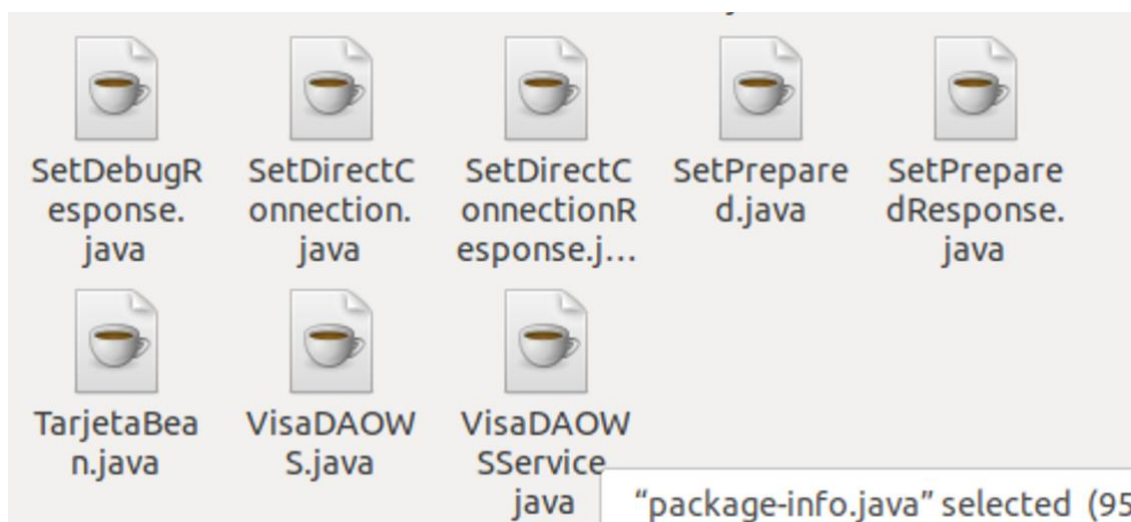
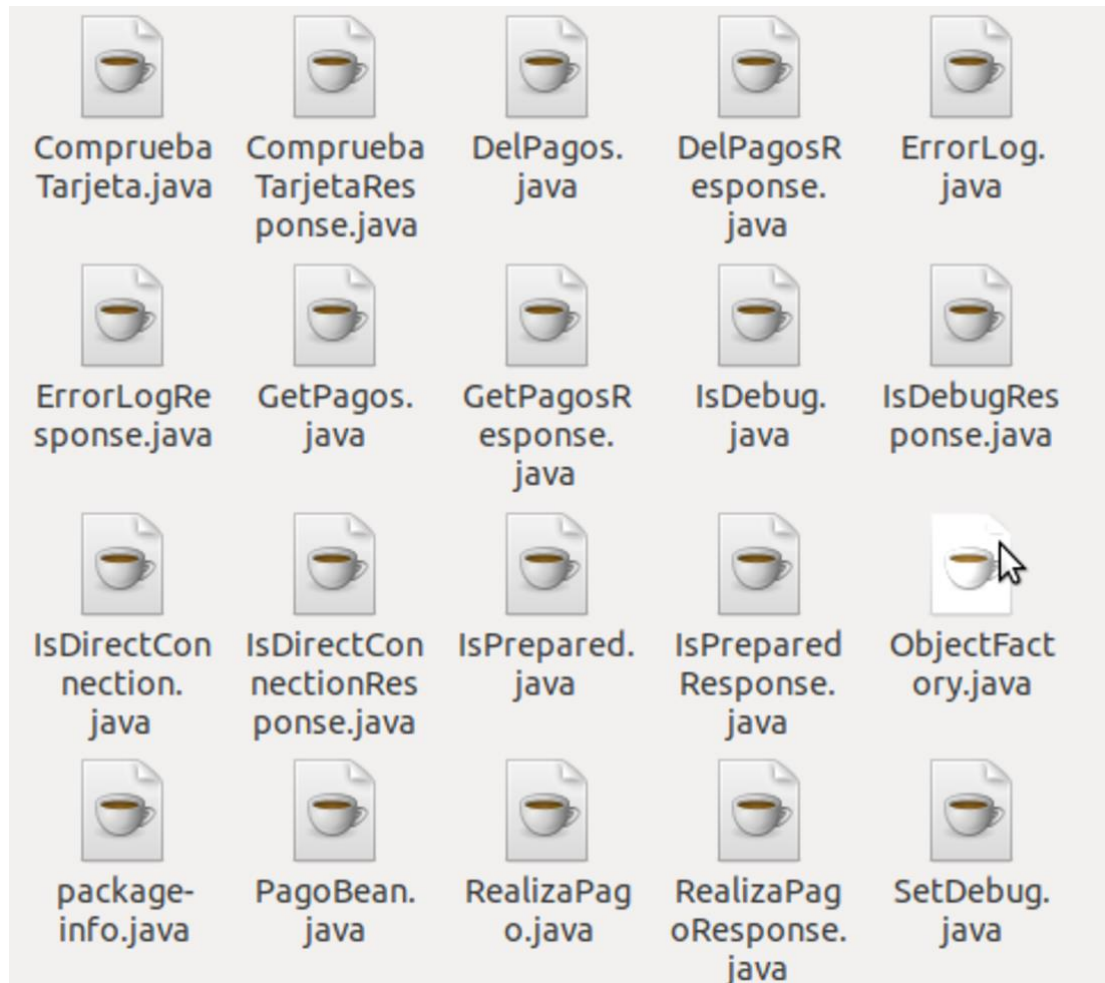
```
/* Petición de los pagos para el comercio */  
List<PagoBean> pagosAux = dao.getPagos(idComercio);  
PagoBean[] pagos = pagosAux.toArray(new PagoBean[pagosAux.size()]);
```

Ejercicio11:

Tras realizar la siguiente comando por la terminal:

wsimport -Xnocompile -d ./ -p ssid.visa <http://10.7.13.1:8080/P1-ws-ws/VisaDAOWSService?wsdl>

Se generan los stubs del cliente. En la carpeta /ssid/visa una clase Java por cada @WebService y cada @WebMethod declarados en el servidor.



Ejercicio 12:

Para completar el target generar-stubs de forma dinámica, modificamos el fichero build.xml de la siguiente manera para que se invoque a wsimport.

```
<!-- TODO - Implementar llamada wsimport -->
<exec executable="${wsimport}">
  <arg line="-d ${build.client}/WEB-INF/classes"/>
  <arg line="-p ${paquete}.visa"/>
  <arg line="${wsdl.url}"/>
</exec>
```

Ejercicio 13:

Tras todas las modificaciones desarrolladas realizamos un ant compilar-cliente, ant empaquetar-cliente y ant desplegar-cliente, y accedemos a la url para realizar las pruebas <http://10.7.13.1:8080/P1-ws-cliente>.

Pago con tarjeta

Proceso de un pago

Id Transacción:	<input type="text" value="3"/>
Id Comercio:	<input type="text" value="3"/>
Importe:	<input type="text" value="3"/>
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 checked="" 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"/>	

Pago con tarjeta

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

idTransaccion: 3
idComercio: 3
importe: 3.0
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

Utilizando el gestor de bases de datos *Tora*, somos capaces de comprobar en la base de datos que dicho pago efectivamente se ha realizado:

```
select * from pago
```

Result	Execution plan	Visualize	Logging					
#	▲	Id Autorizac	Id Transacc	Cod Respue	Importe	comerc	numerotarjeta	fecha
1	▲	6	3	000	3	3	1111 2222 3333 4444	2/24/20 10:25 AM

Finalmente, podemos acceder a la página de pruebas extendida *testbd.jsp* y comprobar la funcionalidad de listado y de borrado de pagos.

Consultamos el pago con el id de comercio utilizado anteriormente:

Consulta de pagos

Id Comercio:

Y observamos que el pago aparece listado:

Pago con tarjeta

Lista de pagos del comercio 3

idTransaccion	Importe	codRespuesta	idAutorizacion
3	3.0	000	6

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Ahora borramos dicho pago utilizando el mismo id:

Borrado de pagos

Id Comercio:

El sistema nos informa de que la operación ha sido exitosa:



Si volvemos a utilizar la funcionalidad de listado, comprobamos que el pago ya no aparece:



Cuestión 1:

Teniendo en cuenta el siguiente diagrama, vamos a estudiar el comportamiento de la aplicación para el caso de uso en que se introduce una tarjeta cuya fecha de caducidad ha expirado.

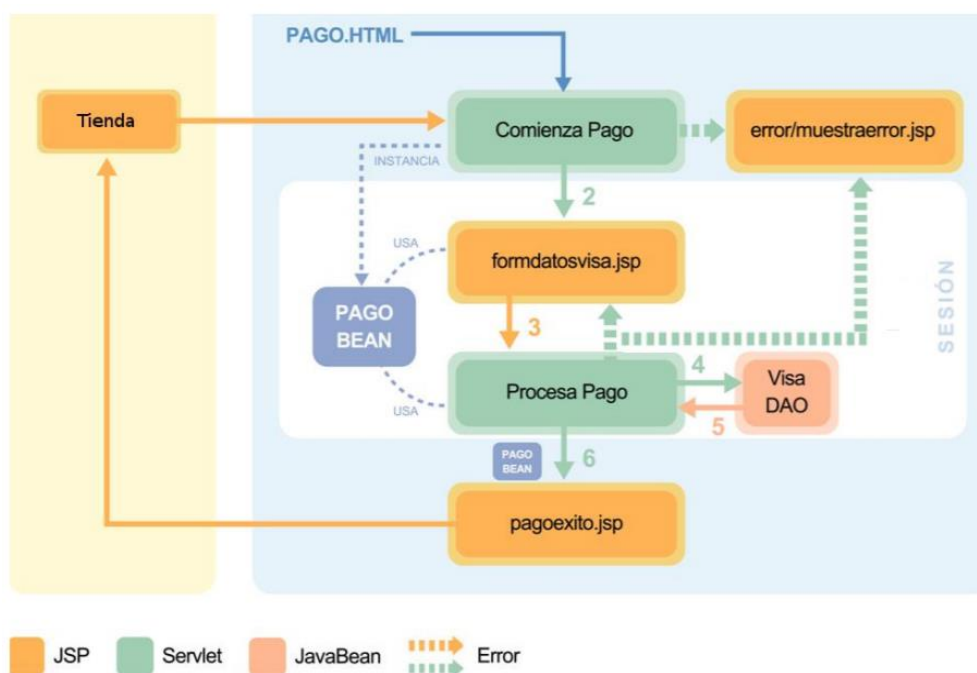


Figura 3. Navegación a través de la aplicación de la práctica 1

Partiendo desde `pago.html`, el cliente accede a `Comienza Pago`. Dicho servlet envía los datos introducidos por el usuario a `formatosvisa.jsp`, el cual se encarga de comprobar que la tarjeta tiene un formato válido. Si es así, se envían los datos del pago a `Procesa Pago`, el servlet que va a conectar con la base de datos `VisaDAO` para comprobar que la información existe en el servidor y es correcta. Sin embargo, al ser incorrecto el formato de la visa, la query ejecutada devuelve un error y se redirige al usuario a `error/muestraerror.jsp`, mostrando un mensaje por pantalla.

Cuestión 2:

De los diferentes servlets que se usan en la aplicación, los encargados de solicitar la información sobre el pago con tarjeta cuando se usa `pago.html` son `VisaDAOWS`, en el lado del servidor, y `Comienza Pago`, en el lado del cliente. En cambio, el encargado de procesar dicha información es el servlet `Procesa Pago`, en el lado del cliente, el cual accede a métodos del servidor para comprobar en la base de datos la existencia de dicha información.

Cuestión 3:

Partiendo desde `pago.html`, el servlet `Comienza Pago` solicita un identificador de transacción, un identificador de comercio y un importe. Dicha información se almacena en una sesión HTTP en el navegador del cliente, y así se comparte con `Procesa Pago`. Este servlet solicita el número de tarjeta, el titular, la fecha de emisión y de caducidad y el código de seguridad. Una vez validada dicha información, se crean las instancias `VisaDAOWS` y `VisaDAOWSService`, y ahora sí se comparte con el servidor web y se actualiza la base de datos.

Cuestión 4:

Enumere las diferencias que existen en la invocación de servlets, a la hora de realizar el pago, cuando se utiliza la página de pruebas extendida `testbd.jsp` frente a cuando se usa `pago.html`. ¿Podría indicar por qué funciona correctamente el pago cuando se usa `testbd.jsp` a pesar de las diferencias observadas?

La principal diferencia entre realizar un pago utilizando `pago.html` y la página de pruebas extendida `testdb.jsp`, reside en que la primera comienza utilizando el servlet `Comienza Pago`, mientras que la segunda accede directamente a `Procesa Pago`, y además utiliza el servlet `VisaDAOWS`. La página de pruebas extendida funciona correctamente debido a que esta inicia su ejecución en `Procesa Pago`, donde ha de pedir toda la información y cotejar en la base de datos que esta sea correcta.