

	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2	
Grupo 2312 - Pareja 6	Práctica 1a	Fecha 7/02/2018
Alumno	Jorge Gutiérrez Díaz	
Alumno	Manuel Jiménez Sánchez	

Práctica P1·A

Sistemas Informáticos II

Índice

Ejercicio 1	3
Ejercicio 2	4
Ejercicio 3	6
Ejercicio 4	8
Ejercicio 5	9
Ejercicio 6	10
Ejercicio 7	13
Ejercicio 8	15
Ejercicio 9	16
Ejercicio 10	17
Ejercicio 11	18
Ejercicio 12	19
Ejercicio 13	20
Cuestiones	23

Antes de empezar necesitamos ejecutar el script de ip virtual en la máquina host:

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

Además en el host, tendremos que exportar la variable del entorno para saber la ubicación de Glassfish:

```
export J2EE_HOME=/usr/local/glassfish-4.1.1/glassfish
```

Ejercicio 1

Para poder desplegar la aplicación en la máquina remota necesitaremos primero levantar el dominio con asadmin:

```
asadmin start-domain domain1
```

En la máquina host ejecutaremos **ant todo**, esto creará la base de datos y subirá todos los archivos a la máquina virtual.

Para comprobar que el servidor funciona accedemos a la página <http://10.5.6.1:8080/P1> para crear un pago y ver que se ha añadido a la base de datos:

Id Transacción:

Id Comercio:

Importe:

Pago con tarjeta

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Id Transacción: 1
Id Comercio: 1
Importe: 1.0

Prácticas de Sistemas Informáticos II

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 1.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Edit Data - visa (10.5.6.1:5432) - visa - pago

File Edit View Tools Help

No limit

	idautorizac [PK] serial	idtrans charac	codrespues character(3)	importe double pre	idcon chara	numerotarjeta character(19)	fecha timestamp without time zone
1	1	1	000	1	1	0424 2375 8201 2251	2018-02-16 00:52:51.233626
*							

Ejercicio 2

Hasta que no se nos pida lo contrario, trabajaremos en la carpeta P1-base. Ahora intentaremos realizar las operaciones mediante conexión directa.

Para ello tenemos que especificar la conexión directa en la clase que vaya a acceder a la base de datos, en este caso es VisaDAO.java, que hereda de DBTester.java.

1. Importamos en VisaDAO.java las cabeceras:

```
import javax.sql.DataSource;
import javax.naming.InitialContext;
import java.sql.DriverManager;
```

2. Reescribimos las siguientes variables de DBTester.java como protected:

```
protected static final String JDBC_DSN = "jdbc/VisaDB";
protected boolean directConnection = false;
protected DataSource ds = null;
protected int dsConnectionCount = 0;
protected int directConnectionCount = 0;
```

3. Copiamos el constructor de DBTester.java en VisaDAO.java:

```
public VisaDAO() {

    try {

        // Para conexiones directas, instanciamos el driver
        Class.forName("org.postgresql.Driver").newInstance();

        // Para conexiones con pool, preparamos un datasource
        // Buscar el datasource por JNDI
        ds = (DataSource) new InitialContext().lookup(JDBC_DSN);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

4. Copiamos en VisaDAO.java la funcion Connection de DBTester.java con los valores correctos:

```
protected synchronized Connection getConnection()
    throws SQLException {

    if (!directConnection) {
        dsConnectionCount++;
        // Obtener una conexión del pool
        return ds.getConnection();
    } else {
        directConnectionCount++;
        // Obtener una nueva conexión
        return
DriverManager.getConnection("jdbc:postgresql://10.5.6.3:5432/visa",
                                "alumnodb",
                                "alumnodb");
    }
}
```

Aquí probamos que un pago se ha realizado correctamente con conexión directa contra la base de datos:

10.5.6.3:8080/P1/testbd.jsp

Pago con tarjeta

Proceso de un pago

Id Transacción:
Id Comercio:
Importe:
Numero de visa:
Titular:
Fecha Emisión:
Fecha Caducidad:
CVV2:
Modo debug: ☐ True ☐ False
Direct Connection: ☒ True ☐ False
Use Prepared: ☐ True ☐ False

10.5.6.3:8080/P1/procesapago

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 10.0
codRespuesta: 000
idAutorizacion: 4

[Volver al comercio](#)

10.5.6.3:8080/P1/getpagos

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	10.0	000	4

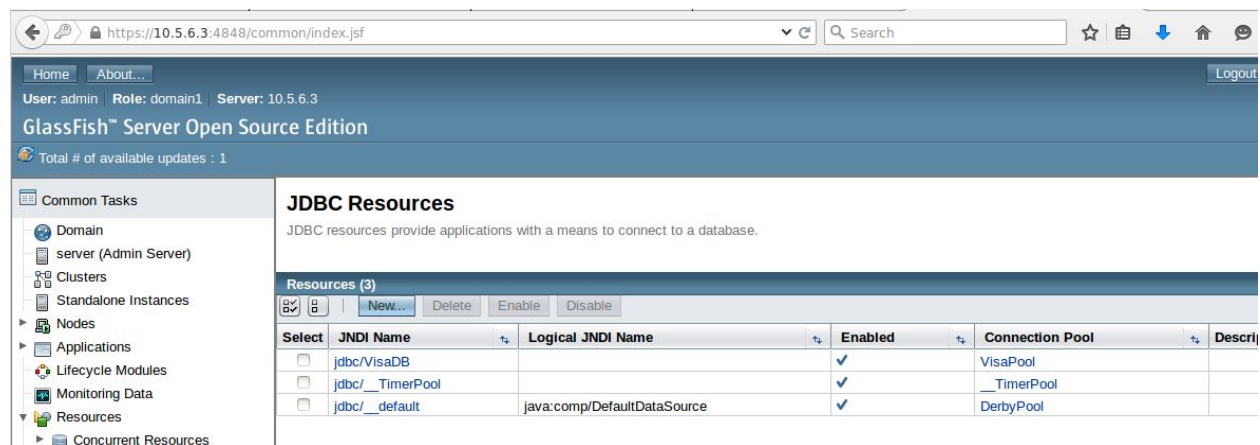
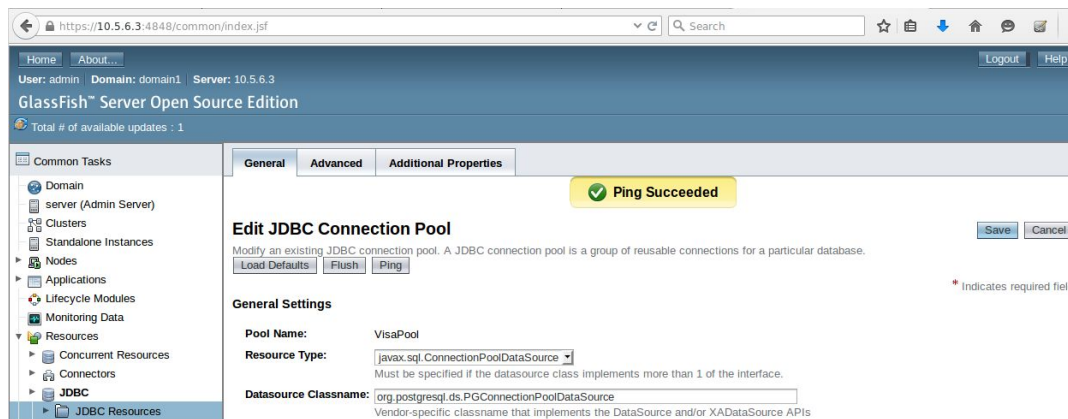
[Volver al comercio](#)

Ejercicio 3

En el archivo *postgresql.properties* encontramos estos datos:

```
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
db.client.host=10.5.6.1
db.client.port=4848
```

Y vemos que coincide en la interfaz de administración. Para ello entramos en Resources > JDBC Resources, localizado en el sidebar. Comprobamos que el pool de conexiones *VisaPool* ha sido creado.



Probamos a hacer un *ping* y comprobamos si ha funcionado:

The screenshot shows the GlassFish Server Open Source Edition web console. The browser address bar displays `https://10.5.6.3:4848/common/index.jsf`. The page header includes navigation links (Home, About...), user information (User: admin, Domain: domain1, Server: 10.5.6.3), and a Logout/Help button. The main content area is titled "JDBC Connection Pools" and includes a description: "To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access a database, it must get a connection." Below this, there is a section for "Pools (3)" with a table listing the configured pools.

Select	Pool Name	Resource Type	Classname	Description
<input type="checkbox"/>	DerbyPool	javax.sql.DataSource	org.apache.derby.jdbc.ClientDataSource	
<input type="checkbox"/>	VisaPool	javax.sql.ConnectionPoolDataSource	org.postgresql.ds.PGConnectionPoolDataSource	
<input type="checkbox"/>	TimerPool	javax.sql.XADataSource	org.apache.derby.jdbc.EmbeddedXADataSource	

Pool Settings

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

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

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

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

Max Wait Time: Milliseconds
Amount of time caller waits before connection timeout is sent

- **Initial and Minimum Pool Size:** Este número indica el número mínimo de conexiones que se establecen en el pool en un primer momento.
- **Maximum Pool Size:** Indica el tamaño máximo de conexiones que pueden crearse en el pool
- **Pool Resize Quantity:** Cuando el timeout del pool se acaba, especificar el número de conexiones que se cierran.
- **Idle Timeout:** Tiempo máximo en el que la conexión puede quedarse parada
- **Max Wait Time:** Tiempo máximo antes de mandar un timeout.

Estos parámetros pueden ser ajustados dependiendo de la demanda y de la cantidad de clientes conectados.

Ejercicio 4

Las consultas que se usan para comprobar si una tarjeta es válida y para realizar el pago son estas:

```
/**
 * 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;
}

/**
 * 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;
}
```

La primera consulta busca una tarjeta que coincida con la información especificada. Si se ha encontrado alguna que corresponda, devuelve una fila con todos los datos disponibles de esa tarjeta.

La segunda consulta consiste en crear una entrada en la tabla de pagos con la información de la tarjeta especificada y los datos del pago y de la transacción.

Ejercicio 5

Accedemos a la dirección 10.5.6.3:4848 y vemos el *log* de la aplicación en la interfaz de administración.

Log Viewer - Mozilla Firefox

https://10.5.6.3:4848/common/logViewer/logViewer.jsf?instanceName=server&logLevel=INFO&viewResults=true

Instance:

Log File:

Log Viewer Results (40)

Records before 1386 | Log File Record Numbers 1386 through 1425 | Records after 1425 |

Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
1425	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:testbd.jsp(details)	javax.enterprise.web	Feb 19, 2018 00:35:58.845	{levelValue=800, timeMillis=1519029358845}
1424	SEVERE	[directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '1' ... (details)		Feb 19, 2018 00:35:54.514	{levelValue=1000, timeMillis=1519029354514}
1423	SEVERE	[directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1... (details)		Feb 19, 2018 00:35:54.512	{levelValue=1000, timeMillis=1519029354512}
1422	SEVERE	[directConnection=false] select * from tarjeta where numeroTarjeta='2576 9060 0185 4562' and titular... (details)		Feb 19, 2018 00:35:54.501	{levelValue=1000, timeMillis=1519029354501}
1421	INFO	visiting unvisited references(details)	javax.enterprise.system.tools.deployment.dol	Feb 19, 2018 00:35:53.889	{levelValue=800, timeMillis=1519029353889}
1420	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:procesapago(details)	javax.enterprise.web	Feb 19, 2018 00:35:53.748	{levelValue=800, timeMillis=1519029353748}
1419	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:testbd.jsp(details)	javax.enterprise.web	Feb 19, 2018 00:33:11.229	{levelValue=800, timeMillis=1519029191229}
1418	INFO	P1 was successfully deployed in 179 milliseconds.(details)	javax.enterprise.system.core	Feb 19, 2018 00:31:46.349	{levelValue=800, timeMillis=1519029106349}

Ejercicio 6

En primer lugar, copiamos los ficheros necesarios en P1-base a P1-ws:

P1-base: Carpeta src/	P1-ws: Carpeta src/
src/ssii2/visa/error/*	src/client/ssii2/error/*
src/ssii2/visa/ValidadorTarjeta.java	src/client/ssii2/visa/ValidadorTarjeta.java
src/ssii2/filtros/CompruebaSesion.java	src/client/ssii2/filtros/CompruebaSesion.java
src/ssii2/controlador/ComienzaPago.java	src/client/ssii2/controlador/ComienzaPago.java
src/ssii2/controlador/GetPagos.java	src/client/ssii2/controlador/GetPagos.java
src/ssii2/controlador/DelPagos.java	src/client/ssii2/controlador/DelPagos.java
src/ssii2/controlador/ServletRaiz.java	src/client/ssii2/controlador/ServletRaiz.java
src/ssii2/controlador/ProcesaPago.java	src/client/ssii2/controlador/ProcesaPago.java
src/ssii2/visa/dao/DBTester.java	src/server/ssii2/visa/DBTester.java
src/ssii2/visa/dao/VisaDAO.java	src/server/ssii2/visa/VisaDAOWS.java
src/ssii2/visa/PagoBean.java	src/server/ssii2/visa/PagoBean.java
src/ssii2/visa/TarjetaBean.java	src/server/ssii2/visa/TarjetaBean.java

Para VisaDAOWS.java debemos eliminar la función getConnection() y el constructor de la clase VisaDAOWS si lo hemos copiado directamente de VisaDAO.java.

También será necesario copiar, si no ha sido copiado ya, el .jar con el driver para la base de datos postgresql, "postgresql-jdbc-4.jar", que se encontraba en la práctica P1-base al directorio P1-ws.

En segundo lugar, modificamos algunos valores de build.properties y postgresql.properties de P1-ws:

- Build.properties:
 - as.host.client=10.5.6.3
 - as.host.server=10.5.6.1
- Postgresql.properties:
 - db.host=10.5.6.1
 - db.client.host=10.5.6.1

Por último modificamos el fichero VISADOWS.java para que se convierta en un servicio web:

- Clase VisaDAOWS

```
/*@WebService(serviceName = "VisaDAO")*/
@WebService()
public class VisaDAOWS extends DBTester {

    private boolean debug = false;
```

- compruebaTarjeta()

```
@WebMethod(operationName = "compruebaTarjeta")
public boolean compruebaTarjeta(@WebParam(name="tarjeta") TarjetaBean tarjeta) {
```

- realizaPago()

```
@WebMethod(operationName = "realizaPago")
public synchronized PagoBean realizaPago(@WebParam(name="pago") PagoBean pago) {
```

- getPagos()

```
@WebMethod(operationName = "getPagos")
public ArrayList<PagoBean> getPagos(@WebParam(name="idComercio")String idComercio) {
```

- delPagos()

```
@WebMethod(operationName = "delPagos")
public int delPagos(@WebParam(name="idComercio")String idComercio) {
```

- isPrepared() / setPrepared()

```
@WebMethod(operationName="isPrepared")
public boolean isPrepared() {
    return prepared;
}

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

- isDirectConnection() / setDirectConnection()

```
@Override
@WebMethod(operationName = "isDirectConnection")
public boolean isDirectConnection(){
    return super.isDirectConnection();
}

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

- isDebug() / setDebug()

```
@WebMethod(operationName="isDebug")
public boolean isDebug() {
    return debug;
}

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

/**
 * @param debug the debug to set
 */
public void setDebugS(String debug) {
    this.debug = (debug.equals("true"));
}
}
```

Por último cambiamos la función getPagos() para que devuelva el pago realizado en lugar en boolean y añadimos las siguientes librerías restantes.

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

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

Antes teníamos un solo nodo y la llamada se realizaba como una clase normal de Java.

Como el método `pagoBean` ahora se ejecuta de manera remota, si devolvemos un booleano, los cambios del objeto no se 'propagan' de vuelta (ya que la función devolvería un *true*, que no presenta cambios respecto al estado anterior) y para ello es necesario devolver el objeto que ha cambiado.

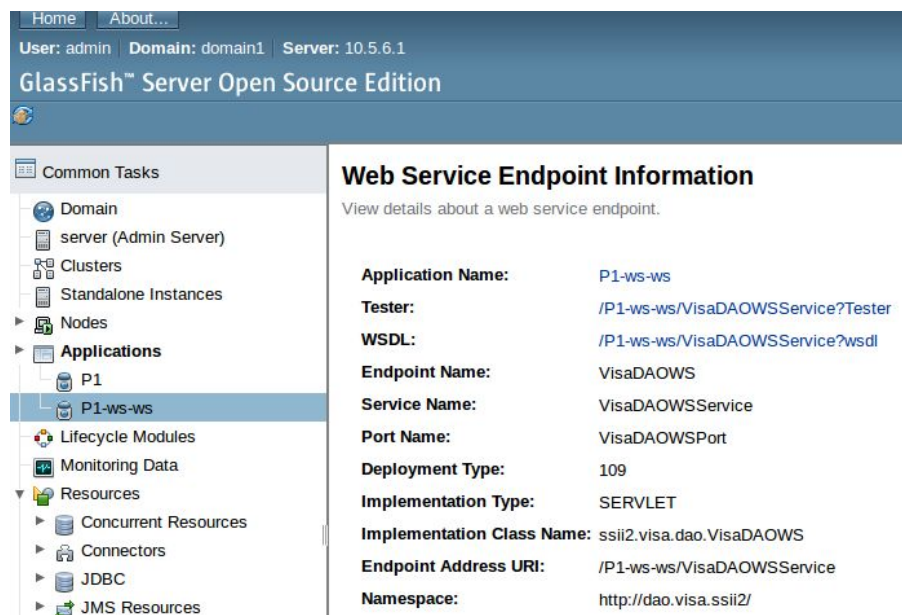
Ejercicio 7

ant empaquetar-servicio: genera estos archivos:

postgresql-jdbc4.jar

dist/server/P1-ws-ws.war

Además podemos ver que en la interfaz de administración el servicio se ha desplegado y que sus métodos han sido publicados como WebService. Podemos acceder al fichero .wsdl que analizaremos a continuación.



- **¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?**
Están definidos en el fichero WSDL, en los elementos que engloban la etiqueta <types>.
- **¿Qué tipos de datos predefinidos se usan?**
En este caso los tipos de datos predefinidos dentro de la etiqueta <types> son <xsd:schema>.
- **¿Cuáles son los tipos de datos que se definen?**
Los tipos de datos que se definen tienen que ver con los métodos que se han publicado al servicio web.
- **¿Qué etiqueta está asociada a los métodos invocados en el webservice?**
La etiqueta <message> contiene a los métodos publicados del WebService.
- **¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice?**
Las etiquetas input y output de la sección <operation> describen qué mensajes de entrada/salida se intercambian en cada uno de los métodos

- **¿En qué etiqueta se especifica el protocolo de comunicación con el webservice?**
En la sección <definitions> la etiqueta xmlns describe qué protocolo se va a usar, en nuestro caso corresponde a esto: xmlns="<http://schemas.xmlsoap.org/wsdl/>" que corresponde con el lenguaje WSDL.
- **¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice?**
Al final del todo, existe una etiqueta llamada
<soap:address
location="<http://10.5.6.1:8080/P1-ws-ws/VisaDAOWSService/>">
que nos dice cuál es la ubicación de acceso al webservice.

Ejercicio 8

Debemos importar las siguientes librerías en procesaPago.java:

```
import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente
import ssii2.visa.VisaDAOWS; // Stub generado automáticamente
import javax.xml.ws.WebServiceRef;
```

Una vez importadas las librerías, somos capaces de instanciar las clases remotas:

```
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;
}

|

////////////////////////////////////
VisaDAOWSService service = new VisaDAOWSService();
VisaDAOWS dao = service.getVisaDAOWSPort ();
```

Ahora que realizaPago() devuelve un objeto de tipo Pago debemos modificar el código para que las llamadas a la función no cause problemas y tratar de forma correcta las excepciones que puedan ocurrir.

```
if (pago == null) {
    pago = creaPago(request);
    boolean isdebug = Boolean.valueOf(request.getParameter("debug"));
    dao.setDebug(isdebug);
    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;
}

pago = dao.realizaPago(pago);

if (pago == null) {
    enviaError(new Exception("Pago incorrecto"), request, response);
    return;
}
```


Ejercicio 9

Accedemos al fichero web.xml y descomentamos la sección donde llamamos a la ruta al servidor remoto. Buscamos la ruta del servidor remoto dentro del fichero VisaDAOWService.wsdl y lo copiamos en la sección descomentada de web.xml.

```
<context-param>
  <param-name>webmaster</param-name>
  <param-value>http://10.5.6.1:8080/P1-ws-ws/VisaDAOWService</param-value>
</context-param>
```

Así podremos acceder con el alias webmaster al Servicio VisaDAOS sin tener que incrustar direcciones IP al código .java:

```
BindingProvider bp = (BindingProvider) dao;
```

```
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, getServletContext().getInitParameter("webmaster"));
```

Ejercicio 10

Para que nuestro sistema funcione con WebServices, hemos adaptado la función `getPagos` para que devuelva un `ArrayList` de objetos `PagoBean`.

```
//PagoBean[]
@WebMethod(operationName = "getPagos")
public ArrayList<PagoBean> getPagos(@WebParam(name="idComercio")String idComercio) {

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

En segundo lugar, importamos las siguientes librerías en las clases `DelPagos.java` y `GetPagos.java`:

import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente

import ssii2.visa.VisaDAOWS; // Stub generado automáticamente

import javax.xml.ws.WebServiceRef;

Por último, debemos instanciar las clases remotas y obtener la dirección al servicio en ambos ficheros:

```
VisaDAOWSService service = new VisaDAOWSService();
VisaDAOWS dao = service.getVisaDAOWSPort ();

BindingProvider bp = (BindingProvider) dao;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, getServletContext().getInitParameter("webmaster"));

//VisaDAO dao = new VisaDAO();

/* Se recoge de la petición el parámetro idComercio*/
String idComercio = request.getParameter(PARAM_ID_COMERCIO);

/* Petición de los pagos para el comercio */
int ret = dao.delPagos(idComercio);
```

```
VisaDAOWSService service = new VisaDAOWSService();
VisaDAOWS dao = service.getVisaDAOWSPort ();

BindingProvider bp = (BindingProvider) dao;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, getServletContext().getInitParameter("webmaster"));

//VisaDAO dao = new VisaDAO();

/* Se recoge de la petición el parámetro idComercio*/
String idComercio = request.getParameter(PARAM_ID_COMERCIO);

/* Petición de los pagos para el comercio */

List<PagoBean> listaPagos = dao.getPagos(idComercio);
ArrayList<PagoBean> arrayPagos = new ArrayList<PagoBean>(listaPagos.size());
arrayPagos.addAll(listaPagos);

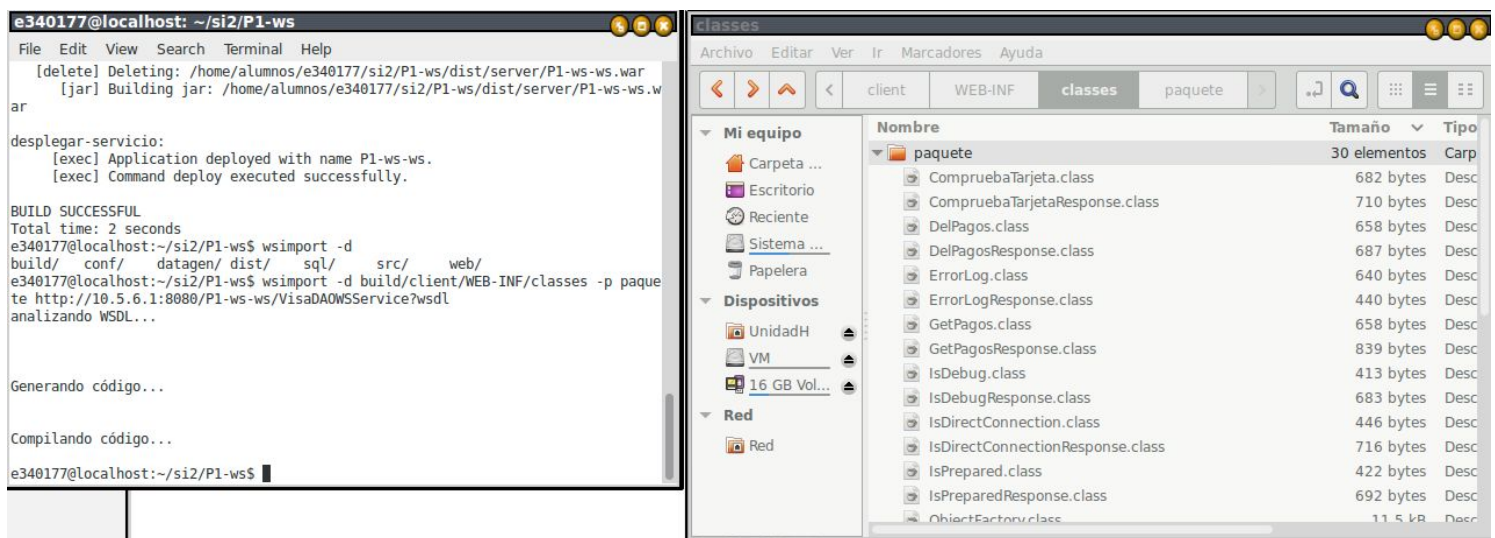
PagoBean[] pagos = new PagoBean[arrayPagos.size()];
pagos = arrayPagos.toArray(pagos);
```

Ejercicio 11

Importamos el WDSL en nuestro directorio de clases local. Para ello no es necesario descargarse directamente el archivo, simplemente ejecutamos el siguiente comando:

```
wsimport -d build/client/WEB-INF/classes -p paquete  
http://10.5.6.1:8080/P1-ws-ws/VisaDAOWSService?wsdl
```

Con esto conseguimos que aparezca en el directorio WEB-INF/classes distintos archivos .class



Las clases que han sido generadas corresponden a las funciones publicadas en VisaDAOWS que a su vez están definidas en el fichero WDSL.

Ejercicio 12

En el fichero build.xml, dentro del target generar-stubs añadimos lo siguiente:

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

```
e340177@localhost:~/si2/P1-ws$ ant generar-stubs
Buildfile: /home/alumnos/e340177/si2/P1-ws/build.xml


montar-jerarquia:

generar-stubs:
    [jar] Building jar: /home/alumnos/e340177/si2/P1-ws/build/P1-ws-clientstubs.jar
    [move] Moving 1 file to /home/alumnos/e340177/si2/P1-ws/build/client/WEB-INF/lib
    [exec] analizando WSDL...
    [exec]
    [exec]
    [exec]
    [exec] Generando código...
    [exec]
    [exec]
    [exec] Compilando código...
    [exec]

BUILD SUCCESSFUL
Total time: 1 second
e340177@localhost:~/si2/P1-ws$
```

Ejercicio 13

Para comprobar que todo funciona correctamente, desplegamos la aplicación en dos ordenadores: El ordenador server (el de la base de datos) va a tener la IP 10.5.6.1 y el ordenador cliente (que interactúa con el usuario) va a tener la IP 10.5.6.3.



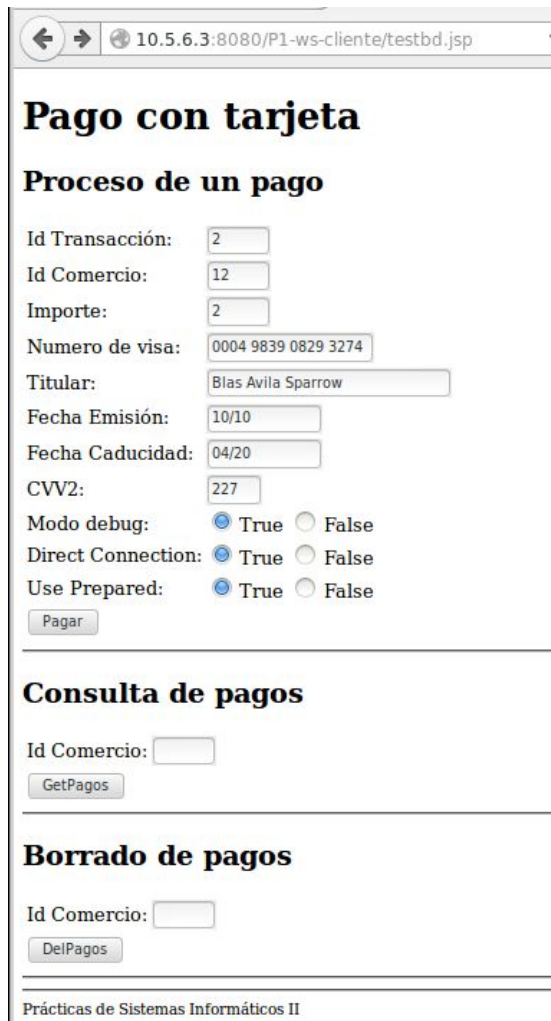
Ubuntu Start Page x http://

10.5.6.3:8080/P1-ws-cliente/

Id Transacción:

Id Comercio:

Importe:



10.5.6.3:8080/P1-ws-cliente/testbd.jsp

Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug: ☒ True ☐ False

Direct Connection: ☒ True ☐ False

Use Prepared: ☒ True ☐ False

Consulta de pagos

Id Comercio:

Borrado de pagos

Id Comercio:

Prácticas de Sistemas Informáticos II

Para probar que todas las consultas se han realizado correctamente, vamos a ejecutarlas a partir de la página web, que contactará con la base de datos PostgreSQL localizada en 10.5.6.1

10.5.6.3:8080/P1-ws-cliente/procesapago

Sea

Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el c

idTransaccion: 2
idComercio: 12
importe: 2.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Edit Data - fdg (10.5.6.1:5432) - visa - pago							
File Edit View Tools Help							
				No limit			
	idautorizac [PK] serial	idtransacci character(3)	codrespue: character(3)	importe double pre	idcomercio character(3)	numerotarjeta character(19)	fecha timestamp without time zone
1	1	2	000	2	12	0004 9839 0829 3274	2018-02-26 02:50:38.506793

Como vemos arriba, hacemos un pago de prueba y comprobamos que los datos de ese pago se ven reflejados en la base de datos del otro ordenador.

Para que getpagos funcionase correctamente, ya que ahora devuelve un ArrayList de pagos tuvimos que modificar el método processRequest() en GetPagos.java:

Creamos una lista y llamamos a getPagos. Luego creamos un arraylist de pagos a partir de esa lista de pagos. Luego creamos un array estático y lo mandamos a la función setAttribute.

```
List<PagoBean> listaPagos = dao.getPagos(idComercio);
ArrayList<PagoBean> arrayPagos = new ArrayList<PagoBean>(listaPagos.size());
arrayPagos.addAll(listaPagos);
```

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

Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug: ☒ True ☐ False

Direct Connection: ☒ True ☐ False

Use Prepared: ☒ True ☐ False

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	1.0	000	1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

En estas últimas imágenes hacemos la comprobación de que getPagos funciona correctamente.

Cuestiones

Cuestión 1. Teniendo en cuenta el diagrama de la Figura 3, indicar las páginas html, jsp y servlets por los que se pasa para realizar un pago desde pago.html, pero en el caso de uso en que se introduce una tarjeta cuya fecha de caducidad ha expirado.

Primero se entra en la página testbd.jsp. Si la tarjeta introducida ha expirado se lanza un error, error que se comprueba en la clase ValidadorTarjeta.java, en el método esValidaFechaCaducidad. Es importante recalcar que esta comprobación se hace en la aplicación cliente, ya que sólo se compara con la fecha actual y se evitan operaciones innecesarias en el servidor.

Cuestión 2. De los diferentes servlets que se usan en la aplicación, ¿podría indicar cuáles son los encargados de solicitar la información sobre el pago con tarjeta cuando se usa pago.html para realizar el pago?

El servlet ProcesaPago en el cliente llama a la función realizaPago del servidor que se encuentra dentro del modulo “Visa DAO” en el diagrama. Concretamente en VisaDAOWS están definidas las consultas necesarias para hacer estas comprobaciones.

Cuestión 3. Cuando se accede a pago.html para hacer el pago, ¿qué información solicita cada servlet? Respecto a la información que manejan, ¿cómo la comparten? ¿dónde se almacena?

ProcesaPago solicita los datos de la tarjeta al dao. Si los datos se han comprobado correctamente, se invoca al servlet ComienzaPago, que creará un objeto pago con los datos de la transacción que haya definido el usuario. Una vez se ha hecho esto, ProcesaPago redirige a pagoexito.jsp que confirmará al usuario la transacción. La información que manejan se transfiere mediante objetos, que se comparten entre los distintos servlets mediante llamadas a métodos.

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?

Pago.html simplemente es un formulario estático que llama al servlet comienzaPago sin ningún tipo de comprobación con la base de datos.

Testbd.jsp llama al ValidadorTarjeta.jsp para comprobar distintos datos del pago. Una vez hecho esto, puede a través de una conexión directa a la base de datos crear una nueva entrada con el pago, hacer nuevas consultas y borrar pagos.