

Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2					
Grupo	2401	Práctica	1	Fecha	dd/mm/aaaa
Alumno/a	Gómez, Martínez, Javier				
Alumno/a	Li, Hu, Carlos				

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

### EJERCICIO 1.

Prepare e inicie una máquina virtual a partir de la plantilla si2srv con: 1GB de RAM asignada, 2 CPUs. A continuación:

- Modifique los ficheros que considere necesarios en el proyecto para que se despliegue tanto la aplicación web como la base de datos contra la dirección asignada a la pareja de prácticas.

Estas son las modificaciones que hemos realizado para poder desplegar la aplicación web y la base de datos contra la dirección 10.1.2.2:

- en el fichero build.properties asignamos as.host=10.1.2.2 (ip donde vamos a desplegar, en este caso, el de la máquina virtual).
- En el fichero postgresql.properties
  - db.password=alumnodb
  - db.host=10.1.2.2 (el host de la base de datos es la máquina local)
  - db.client.host= 10.1.2.2 (el cliente será la máquina local)

Posteriormente empezamos preparando la base de datos, tal y como se ve en la imagen de abajo, mediante el comando ant setup-db:

```
e321083@localhost: ~/Desktop/si2/P1-base
File Edit View Search Terminal Help
Buildfile: /home/alumnos/e321083/Desktop/si2/P1-base/build.xml

create-pool-local:
[echo] Registering jdbc-connection-pool VisaPool.
[echo] ds=org.postgresql.ds.PGConnectionPoolDataSource

create-jdbc-connection-pool:
[exec] Command create-jdbc-connection-pool failed.
[exec] remote failure: Invalid property syntax, missing property value: Password=
[exec] Invalid property syntax, missing property value: Password=
[exec] Usage: create-jdbc-connection-pool [--datasourceclassname=datasourceclassname] [--restype=restype] [--steadypoolsize=8] [--maxpoolsize=32] [--maxwait=60000] [--poolresize=2] [--idletimeout=300] [--initsql=initsql] [--isolationlevel=isolationlevel] [--isolationguaranteed=true] [--isconnectvalidatereq=false] [--validationmethod=table] [--validationtable=validationtable] [--failconnection=false] [--allownoncomponentcallers=false] [--nontransactionalconnections=false] [--validateatmostonceperiod=0] [--leaktimeout=0] [--leakreclaim=false] [--creationretryattempts=0] [--creationretryinterval=10] [--sqltracelisteners=sqltracelisteners] [--statementtimeout=1] [--statementleaktimeout=0] [--statementleakreclaim=false] [--lazyconnectionlistment=false] [--lazyconnectionassociation=false] [--associatewiththread=false] [--driverclassname=driverclassname] [--matchconnections=false] [--maxconnectionusagecount=0] [--ping=false] [--pooling=true] [--statementcachesize=0] [--validationclassname=validationclassname] [--wrapjdbcobjects=true] [--description=description] [--property=property] jdbc_connection_pool_id
[exec] Result: 1

create-resource-local:
[echo] Registering jdbc resource jdbc/VisaDB.

create-jdbc-resource:
[exec] Command create-jdbc-resource failed.
[exec] remote failure: Attribute value (pool-name = VisaPool) is not found in list of jdbc connection pools.
[exec] Result: 1

delete-db:
[echo] driver=org.postgresql.Driver
[echo] url=jdbc:postgresql://10.1.2.2:5432/visa
[echo] user=alumnodb
[echo] password=
[exec] dropdb: database removal failed: ERROR: database "visa" does not exist
[exec] Result: 1

create-db:
[sql] Executing resource: /home/alumnos/e321083/Desktop/si2/P1-base/sql/create.sql
[sql] Executing resource: /home/alumnos/e321083/Desktop/si2/P1-base/sql/insert.sql
[sql] 1003 of 1003 SQL statements executed successfully

setup-db:
BUILD SUCCESSFUL
Total time: 4 seconds
```

Posteriormente compilamos los ficheros mediante el comando ant compilar:

```
e321083@localhost:~/Desktop/si2/P1-base$ ant compilar
Buildfile: /home/alumnos/e321083/Desktop/si2/P1-base/build.xml

montar-jerarquia:

compilar:
    [javac] Compiling 17 source files to /home/alumnos/e321083/Desktop/si2/P1-base/build/WEB-INF/classes

BUILD SUCCESSFUL
Total time: 1 second
e321083@localhost:~/Desktop/si2/P1-base$ █
```

A continuación empaquetamos los ficheros mediante ant empaquetar:

```
e321083@localhost:~/Desktop/si2/P1-base$ ant empaquetar
Buildfile: /home/alumnos/e321083/Desktop/si2/P1-base/build.xml

preparar-web-inf:
    [copy] Copying 11 files to /home/alumnos/e321083/Desktop/si2/P1-base/build

empaquetar:
    [jar] Building jar: /home/alumnos/e321083/Desktop/si2/P1-base/dist/P1.war

BUILD SUCCESSFUL
Total time: 0 seconds
e321083@localhost:~/Desktop/si2/P1-base$ █
```

Y finalmente desplegamos la aplicación mediante el comando ant desplegar:

```
e321083@localhost:~/Desktop/si2/P1-base$ ant desplegar
Buildfile: /home/alumnos/e321083/Desktop/si2/P1-base/build.xml

desplegar:
    [exec] Application deployed with name P1.
    [exec] Command deploy executed successfully.

BUILD SUCCESSFUL
Total time: 2 seconds
```

- Realice un pago contra la aplicación web empleando el navegador en la ruta <http://10.X.Y.Z:8080/P1> Conéctese a la base de datos (usando el cliente Tora por ejemplo) y obtenga evidencias de que el pago se ha realizado.

Empezamos eligiendo un unos Id's de transacción y de comercio, y un importe arbitrarios para comenzar el pago. Después añadimos los datos de un usuario cualquiera en la base de datos, tal y como vemos abajo.

## Pago con tarjeta

Numero de visa:	<input type="text" value="7772 8952 5915 5042"/>
Titular:	<input type="text" value="John Dominguez Lopez"/>
Fecha Emisión:	<input type="text" value="01/10"/>
Fecha Caducidad:	<input type="text" value="10/20"/>
CVV2:	<input type="text" value="317"/>
<input type="button" value="Pagar"/>	

---

Id Transacción: 2  
Id Comercion: 1  
Importe: 1.0

---

Prácticas de Sistemas Informáticos II

Al pulsar el botón de pagar, nos aparece esta pantalla, donde vemos que se ha realizado correctamente.

## Pago con tarjeta

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

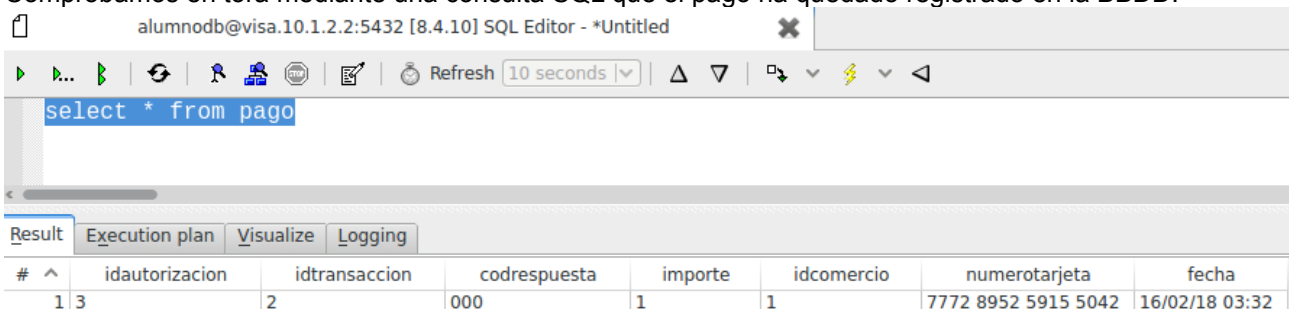
idTransaccion: 2  
idComercio: 1  
importe: 1.0  
codRespuesta: 000  
idAutorizacion: 3

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Comprobamos en tora mediante una consulta SQL que el pago ha quedado registrado en la BBDD.



The screenshot shows a SQL Editor window titled 'alumnodb@visa.10.1.2.2:5432 [8.4.10] SQL Editor - \*Untitled'. The query 'select \* from pago' is entered in the editor. Below the editor, the 'Result' tab is active, displaying a table with 8 columns: '#', 'idautorizacion', 'idtransaccion', 'codrespuesta', 'importe', 'idcomercio', 'numerotarjeta', and 'fecha'. The table contains one row of data.

#	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	3	2	000	1	1	7772 8952 5915 5042	16/02/18 03:32

- Acceda a la página de pruebas extendida, <http://10.X.Y.Z:8080/P1/testbd.jsp>. Compruebe que la funcionalidad de listado de y borrado de pagos funciona correctamente. Elimine el pago anterior. Incluya en la memoria de prácticas todos los pasos necesarios para resolver este ejercicio así como las evidencias obtenidas. Se pueden incluir por ejemplo capturas de pantalla.

Empezamos aplicando como filtro el id de comercio 1, pues es el que usamos en el apartado anterior, y se nos muestra tal y cómo esperábamos esto:

## Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
2	1.0	000	3

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Ahora borramos los pagos asociados al comercio con id 1, y se nos borra de la base de datos:

## Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Si tratamos de volver a listar los comercios con dicho id anterior, vemos que ya no aparece, y por tanto se ha realizado de forma correcta.

## Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

## EJERCICIO 2.

La clase VisaDAO implementa los dos tipos de conexión descritos anteriormente, los cuales son heredados de la clase DBTester. Sin embargo, la configuración de la conexión utilizando la conexión directa es incorrecta. Se pide completar la información necesaria para llevar a cabo la conexión directa de forma correcta. Para ello habrá que fijar los atributos a los valores correctos. En particular, el nombre del driver JDBC a utilizar, el JDBC connection string que se debe corresponder con el servidor postgresql, y el nombre de usuario y la contraseña. Es necesario consultar el apéndice 10 para ver los detalles de cómo se obtiene una conexión de forma correcta.

Para este apartado hemos modificado en el fichero DBTester, los atributos:

- JDBC\_DRIVER="org.postgresql.Driver";
- JDBC\_CONNSTRING="jdbc:postgresql://10.1.2.2:5432/visa";//create=true
- JDBC\_USER="alumnodb";
- JDBC\_PASSWORD="alumnodb";

Una vez completada la información, acceda a la página de pruebas extendida, <http://10.X.Y.Z:8080/P1/testbd.jsp> y pruebe a realizar un pago utilizando la conexión directa y pruebe a listarlo y eliminarlo. Adjunte en la memoria evidencias de este proceso.

Empezamos rellenando el formulario con estos datos (los únicos que no son arbitrarios son los datos del usuario, que están en la BBDD). Además marcamos el Direct Connection a True.

# 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="2347 4840 5058 7931"/>
Titular:	<input type="text" value="Gabriel Avila Locke"/>
Fecha Emisión:	<input type="text" value="11/09"/>
Fecha Caducidad:	<input type="text" value="01/20"/>
CVV2:	<input type="text" value="207"/>
Modo debug:	<input checked="" 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 checked="" type="radio"/> False
<input type="button" value="Pagar"/>	

Le damos al botón de pagar, y se realiza correctamente:

## Pago con tarjeta

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

idTransaccion: 3  
idComercio: 3  
importe: 3.0  
codRespuesta: 000  
idAutorizacion: 1

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Una vez hecho, listamos los pagos con id de comercio 3, y vemos que ha quedado bien registrado.

## Pago con tarjeta

Lista de pagos del comercio 3

idTransaccion	Importe	codRespuesta	idAutorizacion
3	3.0	000	1

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Después borramos el pago con dicho Id de comercio:

## Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 3

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Y comprobamos que ha sido borrado correctamente al volver a consultar la lista de pagos asociada:

## Pago con tarjeta

Lista de pagos del comercio 3

idTransaccion	Importe	codRespuesta	idAutorizacion

[Volver al comercio](#)

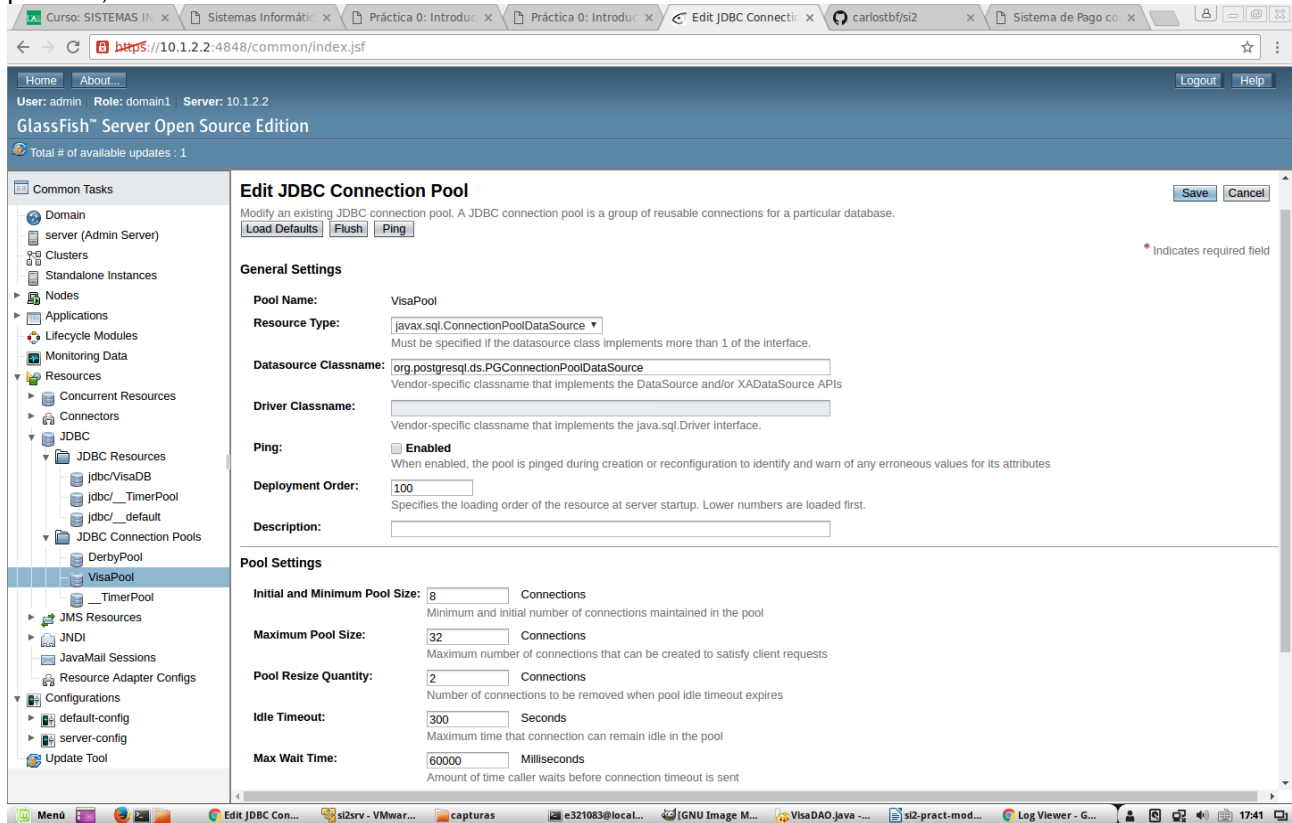
---

Prácticas de Sistemas Informáticos II

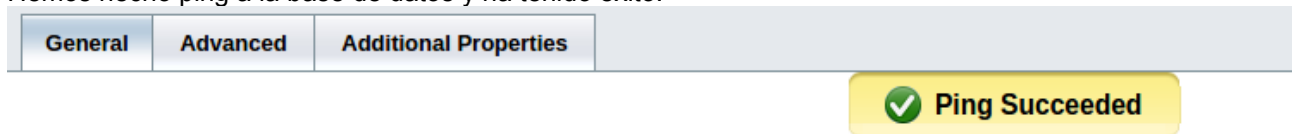
### EJERCICIO 3.

Examinar el archivo `postgresql.properties` para determinar el nombre del recurso JDBC correspondiente al `DataSource` y el nombre del pool. Acceda a la Consola de Administración. Compruebe que los recursos JDBC y pool de conexiones han sido correctamente creados. Realice un Ping JDBC a la base de datos. Anote en la memoria de la práctica los valores para los parámetros Initial and Minimum Pool Size, Maximum Pool Size, Pool Resize Quantity, Idle Timeout, Max Wait Time. Comente razonadamente qué impacto considera que pueden tener estos parámetros en el rendimiento de la aplicación.

Hemos accedido, en la consola de administración, a los recursos JDBC y como ilustramos en la captura de pantalla, tanto el recurso `visaDB` como `VisaPool` han sido creados correctamente.



Hemos hecho ping a la base de datos y ha tenido éxito.



### Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

[Load Defaults](#) [Flush](#) [Ping](#)

#### General Settings

**Pool Name:** VisaPool

Los valores de los parámetros son:

- Initial and Minimum Pool Size: 8
- Maximum Pool Size: 32
- Pool Resize Quantity: 2
- Idle Timeout: 300
- Max Wait Time: 60000

**Initial and Minimum Pool Size:** Número mínimo e inicial de conexiones mantenidas en el pool. Si aumenta,

podríamos atender a un mayor número de conexiones como mínimo pero si son demasiadas, podríamos acabar teniendo conexiones ociosas

**Maximum Pool Size:** Número máximo de conexiones que llegaremos a mantener. Si aumentan, podremos crear un mayor número de conexiones como máximo pero esto también conlleva mayor coste computacional. Si es demasiado bajo, las peticiones podrían congestionarse en colas de espera

**Pool Resize Quantity:** Número de conexiones a eliminar cuando expira el idle timeout. Cuanto mayor sea, las conexiones se irán eliminando de más en más. Esto puede conllevar que se tengan que crear conexiones más frecuentemente si el número es demasiado alto. Por lo contrario, si es demasiado bajo, si hay muchas conexiones ociosas, tardarán más en ser eliminadas

**Idle Timeout:** Tiempo máximo que una conexión puede permanecer ociosa. Si es demasiado alto, las conexiones que no ofrecen servicio permanecerán ocupando recursos durante más tiempo. De lo contrario, si es demasiado bajo, puede que se eliminen conexiones con demasiada frecuencia

**Max Wait Time:** Cantidad de tiempo que espera el invocador antes de que la conexión envíe un timeout. Si es muy alto, un método invocador puede permanecer esperando demasiado tiempo. Por lo contrario, si fuera muy bajo, es posible que el método invocador abandonara la conexión antes de que se procesara la consulta que invocó.

## EJERCICIO 4.

Localice los siguientes fragmentos de código SQL dentro del proyecto proporcionado (P1-base) correspondientes a los siguientes procedimientos:

- Consulta de si una tarjeta es válida.

- Ejecución del pago. Incluya en la memoria de prácticas dichas consultas.

En el fichero visaDAO.java, los metodos para consultar si una tarjeta es válida son:

1. GetQryCompruebaTarjeta, que genera la query sql para comprobar los datos de la tarjeta como una string

La query es:

```
"select * from tarjeta "  
    + "where numeroTarjeta=" + tarjeta.getNumero()  
    + " and titular=" + tarjeta.getTitular()  
    + " and validaDesde=" + tarjeta.getFechaEmision()  
    + " and validaHasta=" + tarjeta.getFechaCaducidad()  
    + " and codigoVerificacion=" + tarjeta.getCodigoVerificacion() + """
```

2. CompruebaTarjeta, que utiliza la query generada por el metodo anterior para comprobar en la base de datos que la tarjeta es valida

En el mismo fichero, los metodos para ejecutar el pago son:

1. getQryInsertPago, que genera la query sql para insertar un pago nuevo como una string

La query es:

```
"insert into pago("  
    + "idTransaccion,"  
    + "importe,idComercio,"  
    + "numeroTarjeta)"  
    + " values ("  
    + "" + pago.getIdTransaccion() + ","  
    + pago.getImporte() + ","  
    + "" + pago.getIdComercio() + ","  
    + "" + pago.getTarjeta().getNumero() + ""  
    + ")"
```

2. GetQryBuscaPagoTransaccion, que genera la query sql para asegurar que el pago ha funcionado como una string

La query es:

```
"select idAutorizacion, codRespuesta "  
    + " from pago "  
    + " where idTransaccion = " + pago.getIdTransaccion()  
    + " and idComercio = " + pago.getIdComercio() + """
```

3. realizaPago, que ejecuta las dos queries anteriores en la base de datos



## EJERCICIO 5:

🔧 Edite el fichero VisaDAO.java y localice el método errorLog. Compruebe en qué partes del código se escribe en log utilizando dicho método. Realice un pago utilizando la página testbd.jsp con la opción de debug activada. Visualice el log del servidor de aplicaciones y compruebe que dicho log contiene información adicional sobre las acciones llevadas a cabo en VisaDAO.java. Incluya en la memoria una captura de pantalla del log del servidor.

Hemos realizado el pago con test\_debug, direct\_connectio y use\_prepared a true.

Instance: server Log File: server.log

Log Viewer Results (40)					
Records before 1263	Log File Record Numbers 1263 through 1302	Records after 1302			
Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
1302	SEVERE	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = ? and ... (details)		20-feb-2018 08:34:08.542	[levelValue=1000, timeMillis=1519144448542]
1301	SEVERE	[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values (?, ?, ... (details)		20-feb-2018 08:34:08.538	[levelValue=1000, timeMillis=1519144448538]
1300	SEVERE	[directConnection=true] select * from tarjeta where numeroTarjeta=? and titular=? and validaDesde=... (details)		20-feb-2018 08:34:08.505	[levelValue=1000, timeMillis=1519144448505]
1299	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto./procesapago(details)	javax.enterprise.web	20-feb-2018 08:34:08.424	[levelValue=800, timeMillis=1519144448424]

## EJERCICIO 6:

Realícense las modificaciones necesarias en VisaDAOWS.java para que implemente de manera correcta un servicio web. Los siguientes métodos y todos sus parámetros deberán ser publicados como métodos del servicio.

🔧 compruebaTarjeta()

🔧 realizaPago()

🔧 isDebug() / setDebug() (Nota: VisaDAO.java contiene dos métodos setDebug que reciben distintos argumentos. Solo uno de ellos podrá ser exportado como servicio web)3 .

🔧 isPrepared() / setPrepared()

En la clase DBTester, de la que hereda VisaDAOWS.java, deberemos publicar así mismo:

🔧 isDirectConnection() / setDirectConnection()

Para ello, implemente estos métodos también en la clase hija. Es decir, haga un override de Java, implementando estos métodos en VisaDAOWS mediante invocaciones a la clase padre (super).

Los cambios en el código son esencialmente:

- poner @WebMethod antes de los métodos descritos en el enunciado y @WebParam antes de los parámetros de dichos métodos.
- Y poner @WebService() antes de la declaración de la clase VisaDAOWS

Modifique así mismo el método realizaPago() para que éste devuelva el pago modificado tras la correcta o incorrecta realización del pago:

🔧 Con identificador de autorización y código de respuesta correcto en caso de haberse realizado.

🔧 Con null en caso de no haberse podido realizar.

Tal y como se ve en el código, esencialmente cambiamos el prototipo de la función para que retorne un objeto de clase PagoBean y devolvemos el pago si se ha obtenido correctamente, o null si ha habido un error.

Por último, conteste a la siguiente pregunta:

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

En esta implementación, el servicio web y el cliente están desacoplados, y por tanto, el acceso a los datos también. Y por esta razón, necesitamos devolver el pago para poder visualizar los datos del pago correctamente en la aplicación.

## EJERCICIO 7.

Despliegue el servicio con la regla correspondiente en el build.xml. Acceda al WSDL remotamente con el navegador e inclúyalo en la memoria de la práctica (habrá que asegurarse que la URL contiene la dirección IP de la máquina virtual donde se encuentra el servidor de aplicaciones).

```
Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is Metro/2.3 (tags/2.3-7528; 2013-04-29T19:34:10+0000) JAXWS-RI/2.2.8 JAXWS/2.2 svn-revision
-->
<!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is Metro/2.3 (tags/2.3-7528; 2013-04-29T19:34:10+0000) JAXWS-RI/2.2.8 JAXWS/2.2 svn-revision
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://dao.visa.ssi2/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://dao.visa.ssi2/" name="VisaDAOWSService">
  <types>...</types>
  <message name="compruebaTarjeta">...</message>
  <message name="compruebaTarjetaResponse">...</message>
  <message name="realizaPago">
    <part name="parameters" element="tns:realizaPago"/>
  </message>
  <message name="realizaPagoResponse">...</message>
  <message name="getPagos">...</message>
  <message name="getPagosResponse">...</message>
  <message name="delPagos">...</message>
  <message name="delPagosResponse">...</message>
  <message name="isPrepared">...</message>
  <message name="isPreparedResponse">...</message>
  <message name="setPrepared">...</message>
  <message name="setPreparedResponse">...</message>
  <message name="errorLog">...</message>
  <message name="errorLogResponse">...</message>
  <message name="isDirectConnection">...</message>
  <message name="isDirectConnectionResponse">...</message>
  <message name="setDirectConnection">...</message>
  <message name="setDirectConnectionResponse">...</message>
  <message name="setDebug">...</message>
  <message name="setDebugResponse">...</message>
  <message name="isDebug">...</message>
  <message name="isDebugResponse">...</message>
  <portType name="VisaDAOWS">...</portType>
  <binding name="VisaDAOWSPortBinding" type="tns:VisaDAOWS">...</binding>
  <service name="VisaDAOWSService">
    <port name="VisaDAOWSPort" binding="tns:VisaDAOWSPortBinding">...</port>
  </service>
</definitions>
```

Comente en la memoria aspectos relevantes del código XML del fichero WSDL y su relación con los métodos Java del objeto del servicio, argumentos recibidos y objetos devueltos.

WSDL, las siglas de *Web Services Description Language*, es un formato del XML que se utiliza para describir servicios web. Describe la forma de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. Las operaciones y mensajes que soporta se describen en abstracto y se ligán después al protocolo concreto de red y al formato del mensaje.

Conteste a las siguientes preguntas:

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

En el fichero XSD referenciado en el WSDL

■ ¿Qué tipos de datos predefinidos se usan?

xsd:string, xsd:double y xsd:boolean

■ ¿Cuáles son los tipos de datos que se definen?

Se define uno por cada método del servicio y uno por cada respuesta de cada método. Además de un elemento por cada clase involucrada en la comunicación.

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

<operation>

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

<message>

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

En este fragmento del código:

<binding name="VisaDAOWSPortBinding" type="tns:VisaDAOWS">

<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>

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

En este fragmento de código con etiqueta soap:address:

<soap:address location="http://10.1.2.1:8080/P1-ws-ws/VisaDAOWSService"/>

## EJERCICIO 8.\_

Realícese las modificaciones necesarias en `ProcesaPago.java` para que implemente de manera correcta la llamada al servicio web mediante stubs estáticos. Téngase en cuenta que:

■ El nuevo método `realizaPago()` ahora no devuelve un boolean, sino el propio objeto `Pago` modificado.

Para llevar a cabo este cambio, hemos cambiado el método `realizaPago` de la clase `VisaDAOWS` para que en su prototipo devuelva un objeto de tipo `pago`. En el código del método, hemos sustituido los `return false;` por `return null;` Y al final del todo, hemos cambiado el retorno por las siguientes líneas de código:

```
if(ret == false){  
    return null;  
}  
return pago;
```

De manera que sin tener que modificar mucho de la función, si hubo un error se devuelve `null` mientras que si todo fue bien, devolvemos el pago realizado (con todos sus datos actualizados).

Después, en el método `processRequest` de la clase `ProcesaPago`, hemos cambiado la comprobación de error del retorno del método `realizaPago`:

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

■ Las llamadas remotas pueden generar nuevas excepciones que deberán ser tratadas en el código cliente.

Para gestionar las posibles excepciones debidas a las llamadas remotas, hemos rodeado la instanciación de `visaDAOWS` con un `try catch` de la siguiente forma:

```
VisaDAOWSService service = new VisaDAOWSService();  
VisaDAOWS dao = service.getVisaDAOWSPort();  
Try {  
    BindingProvider bp = (BindingProvider) dao;  
    bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, "http://10.1.2.1");  
} catch (Exception e){  
    enviaError(new Exception("servidor no encontrado"), request, response);  
    return;  
}
```

## EJERCICIO 9.\_

Modifique la llamada al servicio para que la ruta al servicio remoto se obtenga del fichero de configuración `web.xml`. Para saber cómo hacerlo consulte el apéndice 15.1 para más información y edite el fichero `web.xml` y analice los comentarios que allí se incluyen.

Con la siguiente invocación a método obtenemos la url de nuestro servidor:

```
String url = getServletContext().getInitParameter("url-server");
```

Después, hemos cambiado el parámetro "`http://10.1.2.1`" por esta url en la invocación al método `bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, "http://10.1.2.1");`

De esta forma, se el stub se conecta dinámicamente (en tiempo de ejecución) al servidor correspondiente sin necesidad de que su url esté codificada a fuego.

Adicionalmente, para que esas llamadas a métodos surtan efecto, en el fichero `web.xml` hemos añadido las siguientes líneas:

```
<context-param>  
    <param-name>url-server</param-name>  
    <param-value>http://10.1.2.1:8080/P1-ws-ws/VisaDAOWSService</param-value>  
</context-param>
```

## EJERCICIO 10.\_

Siguiendo el patrón de los cambios anteriores, adaptar las siguientes clases cliente para que toda la funcionalidad de la página de pruebas testbd.jsp se realice a través del servicio web. Esto afecta al menos a los siguientes recursos:

■ Servlet DelPagos.java: la operación dao.delPagos() debe implementarse en el servicio web.

■ Servlet GetPagos.java: la operación dao.getPagos() debe implementarse en el servicio web.

Tenga en cuenta que no todos los tipos de datos son compatibles con JAXB (especifica como codificar clases java como documentos XML), por lo que es posible que tenga que modificar el valor de retorno de alguno de estos métodos. Los apéndices contienen más información. Más específicamente, se tiene que modificar la declaración actual del método getPagos(), que devuelve un PagoBean[], por: public ArrayList<PagoBean> getPagos(@WebParam(name = "idComercio") String idComercio)

Hay que tener en cuenta que la página listapagos.jsp espera recibir un array del tipo PagoBean[]. Por ello, es conveniente, una vez obtenida la respuesta, convertir el ArrayList a un array de tipo PagoBean[] utilizando el método toArray() de la clase ArrayList.

En ambas clases, DelPagos y GetPagos, hemos instanciado remotamente la clase VisaDAOWS tal y como lo hicimos en ProcesaPago.

A continuación, hemos cambiado el método getPagos de la clase VisaDAOWS para que devuelva un arrayList <Pago> en lugar de un Pago[]. Para esto, hemos realizado los siguientes cambios (el código comentado es el que había antes que ya no es necesario porque ahora el metodo devolvera el ArrayList pagos directamente):

```
/*
ret = new PagoBean[pagos.size()];
ret = pagos.toArray(ret);
*/
// Cerramos / devolvemos la conexion al pool
pcon.close();
```

Y en el retorno del método:

```
return pagos;
```

Después, en la clase GetPagos, hemos añadido las siguientes líneas para mantener la funcionalidad del código:

```
List <PagoBean> pagosList = dao.getPagos(idComercio);
PagoBean[] pagos = pagosList.toArray(new PagoBean[pagosList.size()]);
```

## EJERCICIO 11.\_

Realice una importación manual del WSDL del servicio sobre el directorio de clases local. Anote en la memoria qué comando ha sido necesario ejecutar en la línea de comandos, qué clases han sido generadas y por qué. Téngase en cuenta que el servicio debe estar previamente desplegado.

El comando para importar manualmente el wsdl ha sido:

Wsimport -d build/client/WEB-INF/classes -p ssi2.visa <http://10.1.2.1:8080/P1-ws-ws/VisaDAOWSService?wsdl>

Y el resultado obtenido:

```
e321786@localhost:~/Desktop/si2/P1-ws$ wsimport -d build/client/WEB-INF/classes -p ssi
i2.visa http://10.1.2.1:8080/P1-ws-ws/VisaDAOWSService?wsdl
analizando WSDL...
```

```
Generando código...
```

```
Compilando código...
```

Las clases generadas han sido:

1. CompruebaTarjeta.class

2. CompruebaTarjetaResponse.class
3. IsDebug.class
4. IsDebugResponse.class
5. IsDirectConnection.class
6. IsDirectConnectionResponse.class
7. IsPrepared.class
8. IsPreparedResponse.class
9. ObjectFactory.class
10. Package-info.class
11. PagoBean.class
12. RealizaPago.class
13. RealizaPagoResponse.class
14. SetDebug.class
15. SetDebugResponse.class
16. SetDirectConnection.class
17. SetDirectConnectionResponse.class
18. SetPrepared.class
19. SetPreparedResponse.class
20. TarjetaBean.class
21. VisaDAOWS.class
22. VisaDAOWSService.class

Como podemos ver, se generan dos clases por cada uno de los métodos remotos de VisaDAOWS, siendo uno de ellos el input del método (sus argumentos) y el otro el output del método (sus datos de retorno). Además, se genera una clase por cada una de las clases VisaDAOWS, PagoBean y TarjetaBean.

## EJERCICIO 12.\_

Complete el target generar-stubs definido en build.xml para que invoque a wsimport (utilizar la funcionalidad de ant exec para ejecutar aplicaciones). Téngase en cuenta que:

- El raíz del directorio de salida del compilador para la parte cliente ya está definido en build.properties como \${build.client}/WEB-INF/classes
- El paquete Java raíz (ssii2) ya está definido como \${paquete}
- La URL ya está definida como \${wsdl.url}

En el fichero build.xml hemos añadido la siguiente directriz al target genera-stubs:

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

Esta directriz se encargará de ejecutar wsimport por nosotros cada vez que se quiera preparar el cliente.

## EJERCICIO 13:

■ Realice un despliegue de la aplicación completo en dos nodos tal y como se explica en la Figura 8. Habrá que tener en cuenta que ahora en el fichero build.properties hay que especificar la dirección IP del servidor de aplicaciones donde se desplegará la parte del cliente de la aplicación y la dirección IP del servidor de aplicaciones donde se desplegará la parte del servidor. Las variables as.host.client y as.host.server deberán contener esta información.

■ Probar a realizar pagos correctos a través de la página testbd.jsp. Ejecutar las consultas SQL necesarias para comprobar que se realiza el pago. Anotar en la memoria práctica los resultados en forma de consulta SQL y resultados sobre la tabla de pagos.

Una vez desplegada la aplicación correctamente, realizamos un pago en testbd.jsp con los parámetros siguientes.

# Pago con tarjeta

Numero de visa:	<input type="text" value="2347 4840 5058 7931"/>
Titular:	<input type="text" value="Gabriel Avila Locke"/>
Fecha Emisión:	<input type="text" value="11/09"/>
Fecha Caducidad:	<input type="text" value="01/20"/>
CVV2:	<input type="text" value="207"/>
<input type="button" value="Pagar"/>	

---

Id Transacción: 777  
Id Comercion: 777  
Importe: 999.0

---

Prácticas de Sistemas Informáticos II

Y vemos que se realiza con éxito tras darle clic a pagar:

# Pago con tarjeta

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

idTransaccion: 777  
idComercio: 777  
importe: 999.0  
codRespuesta: 000  
idAutorizacion: 3

[Volver al comercio](#)

---

Prácticas de Sistemas Informáticos II

Y si nos metemos en Tora a realizar una consulta, veremos que se ha registrado correctamente el pago:

The screenshot shows a SQL Editor window titled 'alumnodb@visa.10.1.2.1:5432 [8.4.10] SQL Editor - \*Untitled'. The query entered is 'select \* from pago where idautorizacion = 3'. The results are displayed in a table with columns: #, idautorizacion, idtransaccion, codrespuesta, importe, idcomercio, numerotarjeta, and fecha. The first row of results shows idautorizacion: 3, idtransaccion: 777, codrespuesta: 000, importe: 999, idcomercio: 777, numerotarjeta: 2347 4840 5058 7931, and fecha: 22/02/18 06:35.

#	idautorizacion	idtransaccion	codrespuesta	importe	idcomercio	numerotarjeta	fecha
1	3	777	000	999	777	2347 4840 5058 7931	22/02/18 06:35

## **Cuestiones:**

### ***Cuestión número 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.

Pago.html formulario con importe, idComercio e idTransacción

Lleva a comienzaPago que evalúa que los datos del formulario sean correctos. Si lo son:

te manda a formdatosvisa.jsp con otro formulario para los datos de la tarjeta de crédito. Son rellenados por el usuario

Lleva a procesaPago, que se da cuenta de que la tarjeta ha expirado

redirecciona a la página de error: error/muestraerror.jsp

### ***Cuestión número 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?

ComienzaPago y procesaPago

### ***Cuestión número 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?

ComienzaPago solicita idTransaccion, idComercio e importe.

ProcesaPago solicita el número de visa, el titular, la fecha de emisión, la fecha de caducidad y el CVV2

### ***Cuestión número 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 diferencia radica en que desde pago.html se necesitan los servlets comienzaPago y procesaPago para obtener la información de dos formularios independientes (en uno se pide idTransaccion, idComercio e importe y en el otro el resto de los datos) mientras que desde testbd.jsp se llama directamente a procesaPago con todos los datos en un único formulario (además, con campos como debug, prepared y direct connection para probar más a fondo la funcionalidad).