

 UNIVERSIDAD AUTÓNOMA DE MADRID	Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2				
Grupo	2401	Práctica	1A	Fecha	22/02/2022
Alumno/a	Villa, Rodríguez, Juan Carlos				
Alumno/a	Sánchez, Signorini, Martín				

Práctica 1A:

Cuestión número 1:

Incluya en la memoria los pasos necesarios para iniciar la máquina virtual, desplegar la aplicación y realizar un pago contra la aplicación web, comprobando su correcto funcionamiento.

Una vez descargada la máquina virtual y asegurándose de que las propiedades de red estén configuradas correctamente estableciendo las direcciones MAC, seguimos los pasos iniciales para configurar la dirección de bridge en la máquina virtual.

Comprobamos en el host que las variables de entorno necesarias (J2EE_HOME, JAVA_HOME, etc.) están presentes. Ejecutamos el script virtualip.sh para establecer una interfaz de red que permita la conexión con la máquina virtual. Finalmente, establecemos una conexión SSH con la máquina virtual para facilitar la introducción de comandos.

Comenzamos modificando los ficheros *build.properties* y *postgresql.properties* de la carpeta *P1-base* en el host para especificar así la dirección del servidor donde desplegar la aplicación, en particular 10.1.7.1. Cambiamos la variable *as.host* en *build.properties*, y las variables *db.host*, *db.client.host* en *postgresql.properties*, y empezamos con el despliegue.

En primer lugar, introducimos el comando *asadmin start-domain domain1* en la máquina virtual, a continuación desde el host utilizamos *ant regenerar-bd* seguido de *ant replegar limpiar-todo unsetup-db todo* con lo cual nos dirigimos en el navegador a la dirección <http://10.1.7.1:8080/P1> donde realizamos un pago (para ello, previamente vimos la base de datos en pgadmin para encontrar una tarjeta válida y datos correctos).

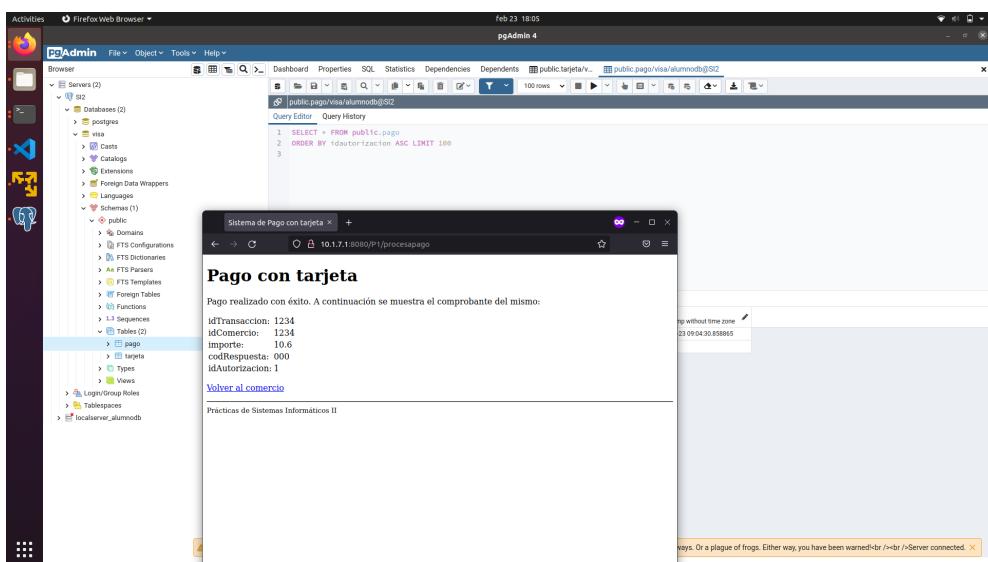


Fig. 1: Resultado obtenido tras realizar el pago.

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers (2)', 'S12' is selected, and 'Tables (2)' is expanded to show 'pago'. A 'Query Editor' tab is open with the following SQL query:

```

1 SELECT * FROM public.pago
2 ORDER BY idautorizacion ASC LIMIT 100
3

```

The results table has columns: idautorizacion, bResumen, cComprador, dImporte, fTimestamp, and fFecha. One row is displayed with the following values:

	1	1 1234	000	10.6	1234	0004 9839 0829 3274	2022-02-23 09:04:10.85865
--	---	--------	-----	------	------	---------------------	---------------------------

A message at the bottom of the interface states: 'You have connected to a server version that is older than is supported by pgAdmin. This may cause pgAdmin to break in strange and unpredictable ways. Or a plague of frogs. Either way, you have been warned!

Server connected.'

Fig. 2: Comprobación de que el pago se ha realizado en la base de datos.

Posteriormente, nos dirigimos a <http://10.1.7.1:8080/P1/testdb.jsp> para listar los pagos con id 1234, comprobando que existe el nuestro.

The screenshot shows a Firefox browser window with the URL '10.1.7.1:8080/P1/getpagos'. The page title is 'Pago con tarjeta'. It displays a table titled 'Lista de pagos del comercio 1234' with columns: idTransaccion, Importe, codRespuesta, and idAutorizacion. One row is highlighted with the value 1234. Below the table is a link 'Volver al comercio'.

idTransaccion	Importe	codRespuesta	idAutorizacion
1234	10.600000381469727	000	1

A message at the bottom of the browser window states: 'You have connected to a server version that is older than is supported by pgAdmin. This may cause pgAdmin to break in strange and unpredictable ways. Or a plague of frogs. Either way, you have been warned!

Server connected.'

Fig. 3: Listado de pagos, aparece el realizado previamente.

Proseguimos con su eliminación especificando los datos del pago y comprobamos que en efecto se ha eliminado.

The screenshot shows a Firefox browser window with the URL '10.1.7.1:8080/P1/delpagos'. The page title is 'Pago con tarjeta'. It displays a message 'Se han borrado 1 pagos correctamente para el comercio 1234' and a link 'Volver al comercio'.

A message at the bottom of the browser window states: 'You have connected to a server version that is older than is supported by pgAdmin. This may cause pgAdmin to break in strange and unpredictable ways. Or a plague of frogs. Either way, you have been warned!

Server connected.'

Fig. 4: Borrado del pago.



Fig. 5: Comprobación de que la eliminación es correcta.

Cuestión número 2:

Completar en VisaDAO la información correcta para llevar a cabo la conexión directa de forma correcta, y comprobarlo realizando un pago mediante conexión directa.

Realizamos los cambios pertinentes en DBTester, clase de la cual VisaDAO hereda. En particular, corregimos el driver, conexión, contraseña y usuario.

```
    public DBTester() {
        try {
            // Para conexiones directas, instanciamos el driver
            Class.forName(DBC_DRIVER).newInstance();
            // Para conexiones con pool, preparamos un datasource
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Fig. 6: Datos modificados.

Realizamos un pago con conexión directa y comprobamos los resultados:

Activities Firefox Web Browser Sistema de Pago con tarjeta Feb 25 18:28 + 10.1.7.1:8080/testbd.jsp

Pago con tarjeta

Proceso de un pago

Id Transacción:
Id Comercio:
Importe:
Número de vista:
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

Fig. 7: Pago mediante conexión directa

A screenshot of a Firefox browser window. The title bar says "Sistema de Pago con tarjeta". The address bar shows the URL "10.1.7.1:8080/pt1/procesapago". The main content area displays a success message: "Pago realizado con éxito. A continuación se muestra el comprobante del mismo:" followed by transaction details: idTransaccion: 1112, idComercio: 1112, importe: 20.0, codRespuesta: 000, idAutorizacion: 1. Below this, there is a blue link "Volver al comercio". At the bottom, a footer reads "Prácticas de Sistemas Informáticos II". The browser interface includes standard navigation buttons (back, forward, search) and a menu bar.

Fig. 8: Realización correcta del pago por conexión directa.



Fig. 9: Listado de pagos.



Fig. 10: Borrado del pago.



Fig. 11: Listado tras el borrado.

Cuestión número 3:

Examinar el archivo `postgresql.properties` para determinar el nombre del recurso **JDBC** correspondiente al **DataSource** y el nombre de **pool**. Acceda a la Consola de Administración y compruebe que los recursos **JDBC** y **pool de conexiones** han sido correctamente creados. Realice un ping y anote los valores de los parámetros **Initial and Minimum Pool Size**, **Maximum Pool Size**, **Pool Resize Quantity**, **Idle Timeout**, **Max Wait Time**. Comente el impacto de dichos parámetros en el rendimiento de la aplicación.

En primer lugar adjuntamos 2 capturas de pantalla, la primera de ellas de un Ping JDBC realizado con éxito.

La segunda, del pool de conexiones de la Consola de Administración, viendo que las conexiones han sido correctamente creadas.

The screenshot shows the GlassFish Administration Console interface. The URL is https://10.1.7.1:4848/common/index.jsf. The top navigation bar includes Home, About, Logout, and Help. The left sidebar menu shows the tree structure: tree, Common Tasks, Domain, Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources, Concurrent Resources, Connectors, JDBC, JDBC Resources, JDBC Connection Pools, DerbyPool, VistaPool, TimerPool, JMS Resources, JNDI, JavaMail Sessions, Resource Adapter Configs, Configurations, default-config, server-config, and Update Tool. The JDBC Connection Pools section is selected. A sub-menu under JDBC Connection Pools shows DerbyPool and VistaPool. The VistaPool item is selected and highlighted with a blue border. The main content area is titled "Edit JDBC Connection Pool" and shows the configuration for the VistaPool. A yellow button at the top right says "Ping Succeeded". The General Settings section includes fields for Pool Name (VistaPool), Resource Type (javax.sql.ConnectionPoolDataSource), Datasource Classname (org.postgresql.ds.PGConnectionPoolDataSource), Driver Classname (Vendor-specific classname that implements the DataSource and/or XADatasource APIs), Ping (Enabled), Deployment Order (100), and Description. The Pool Settings section includes fields for Initial and Minimum Pool Size (8), Maximum Pool Size (32), Pool Resize Quantity (2), Idle Timeout (300), and Max Wait Time (60000). The Transaction section has a checkbox for Non Transactional Connections which is unchecked. At the bottom right are Save and Cancel buttons. A note at the bottom right indicates that * indicates required fields.

Fig. 12: Ping JDBC realizado con éxito a la base de datos.

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

General Settings

Pool Name:	VisaPool
Resource Type:	<input type="button" value="javax.sql.ConnectionPoolDataSource"/>
Must be specified if the datasource class implements more than 1 of the interface.	
Datasource Classname:	<input type="text" value="org.postgresql.ds.PGConnectionPoolDataSource"/>
Vendor-specific classname that implements the DataSource and/or XADatasource APIs	
Driver Classname:	<input type="text"/>
Vendor-specific classname that implements the java.sql.Driver interface.	
Ping:	<input checked="" type="checkbox"/> Enabled
When enabled, the pool is pinged during creation or reconfiguration to identify and warn of any erroneous values for its attributes	
Deployment Order:	<input type="text" value="100"/>
Specifies the loading order of the resource at server startup. Lower numbers are loaded first.	
Description:	<input type="text"/>

Pool Settings

Initial and Minimum Pool Size:	<input type="text" value="8"/> Connections
Minimum and initial number of connections maintained in the pool	
Maximum Pool Size:	<input type="text" value="32"/> Connections
Maximum number of connections that can be created to satisfy client requests	
Pool Resize Quantity:	<input type="text" value="2"/> Connections
Number of connections to be removed when pool idle timeout expires	
Idle Timeout:	<input type="text" value="300"/> Seconds
Maximum time that connection can remain idle in the pool	
Max Wait Time:	<input type="text" value="60000"/> Milliseconds
Amount of time caller waits before connection timeout is sent	

Transaction

Non Transactional Connections:	<input checked="" type="checkbox"/> Enabled
Returns non-transactional connections	
Transaction Isolation:	<input type="button"/>
If unspecified, use default level for JDBC Driver	

Fig. 13: Pool de conexiones visto desde la Consola de Administración de Glassfish.

Por otro lado, adjuntamos también los valores asignados a los diferentes parámetros en la sección de pool settings:

- Initial and Minimum Pool Size: **8 connections**, cuando el servidor es creado, se crea por defecto con 8 conexiones, además, este será el mínimo número de conexiones que tendrá activas.
- Maximum Pool Size: **32 connections**, es el número máximo de conexiones activas que se mantendrán a la vez en el pool.
- Pool Resize Quantity: **2 connections**, número de conexiones a cerrar cuando el idle timeout expira.

Con todo esto, podemos concluir cuáles son algunos de los pros y contras de tomar decisiones con respecto al tamaño del pool de conexiones:

- Un pool de conexiones pequeño, hará que los accesos a la tabla de conexiones sean sustancialmente más rápidos. Por otra parte puede provocar que en ocasiones no haya conexiones suficientes como para responder a todas las peticiones y que la cola de peticiones se vaya saturando.
- Un pool de conexiones grande, tendrá más conexiones mediante las cuales procesar peticiones, de esta manera, se evitará en gran medida que las peticiones pasen mucho tiempo en la cola. Como principal contra, la ralentización de los accesos a la tabla de conexiones.

En cuanto en lo que las propiedades relacionadas con el timeout de las conexiones tenemos:

- Idle Timeout: **300 seconds**, tiempo que máximo que permitiremos que una conexión permanezca inactiva en el pool de conexiones, después de este tiempo, el pool puede cerrarla.

Para un mejor rendimiento, establecer este valor a 0, de manera que las conexiones inactivas no serán eliminadas. Esto asegura que prácticamente no haya penalización por crear nuevas conexiones además de que desactiva el hilo encargado de monitorizar las conexiones inactivas. El servidor de la base de datos, es cierto, podrá reiniciar conexiones que no se hayan usado por mucho tiempo.

- Max Wait Time: **60000ms**, tiempo que el cliente esperará a que el servidor genere un connection timeout.
Para mejorar el rendimiento lo máximo posible, establecer el valor a 0, de esta manera las peticiones esperarán indefinidamente a que alguna de las conexiones pase a estar disponible.

Cuestión número 4:

Localiza los fragmentos de código SQL dentro del proyecto proporcionado correspondientes a la consulta de si una tarjeta es válida y la ejecución del pago.

En la captura adjunta a este ejercicio, encontramos los fragmentos de código propios de las consultas requeridas. El primero de estos fragmentos, es el que se encargará de buscar si en la base de datos existe alguna tarjeta de crédito con los datos concretos introducidos en el formulario, verificando por tanto si la tarjeta es válida. El segundo de los fragmentos de código, es la función que se encarga de realizar un pago, para ellos simplemente utilizará los datos de pago introducidos por el usuario y mediante la consulta SQL introducirá una entrada nueva en la tabla pago, de la base de datos.

```
* 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;
}
```

Fig. 14: Consultas para verificar si una tarjeta es válida y para realizar un pago.

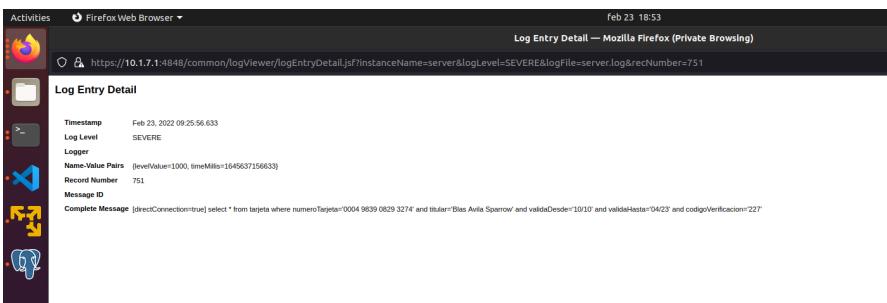
Cuestión número 5:

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 desde la página de pruebas extendida usando la opción de debug. Visualice el log del servidor de aplicaciones y compruebe que dicho log contiene información adicional. Incluya capturas accediendo al log tanto desde el terminal como del portal web.

Para este ejercicio, adjuntamos las capturas de las diferentes impresiones en el log realizadas por las diferentes llamadas al método errorLog a lo largo del proceso de procesaPago del fichero VisaDAO.java.

Las tres primeras capturas son de la sección de vista detallada de las entradas del log de la Consola de Administración de Glassfish. Por otro lado la cuarta captura es la información que se va escribiendo en el fichero encontrado en la dirección “/opt/glassfish4.1.2/glassfish/domains/domain1/logs/server.log”.

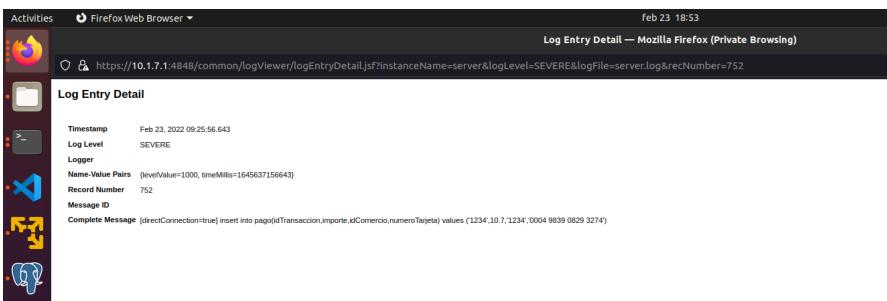
En ambos casos, vemos cuáles son los procesos que se van llevando a cabo por parte del servidor, en su interacción con la base de datos a lo largo de la realización de un pago. En primer lugar se verifica si la tarjeta introducida es válida, viendo si existe una entrada con dicha información en la base de datos. Después se inserta la transacción en la tabla pago de la base de datos y por último se obtiene el id de autorización y el código de respuesta de la transacción.



The screenshot shows a Firefox browser window with the URL <https://10.1.7.1:4848/common/logViewer/logEntryDetail.jsf?instanceName=server&logLevel=SEVERE&logFile=server.log&recNumber=751>. The title bar says "Log Entry Detail — Mozilla Firefox (Private Browsing)". The log entry details are as follows:

Timestamp	Feb 23, 2022 09:25:56.633
Log Level	SEVERE
Logger	
Name-Value Pairs	{levelValue=1000, timeMillis=1645637156633}
Record Number	751
Message ID	
Complete Message	[directConnection=true] select * from tarjeta where numeroTarjeta='0004 9839 0829 3274' and titular='Bles Avila Sperow' and validaDesde='10/10' and validaHasta='04/23' and codigoVerificacion='227'

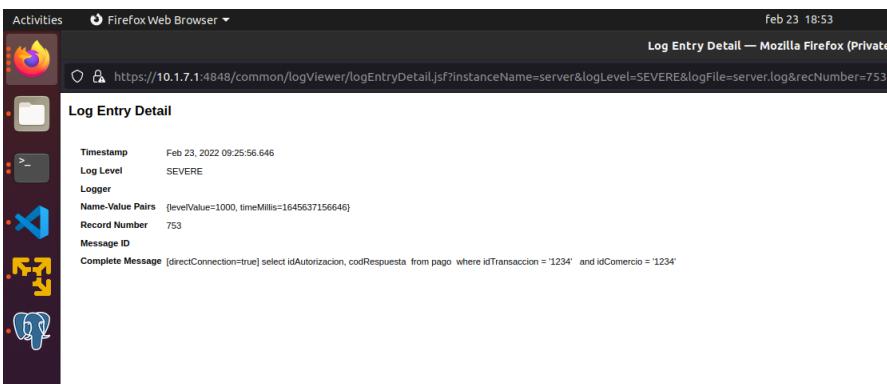
Fig. 15: Entrada en el log para la comprobación de si una tarjeta es válida.



The screenshot shows a Firefox browser window with the URL <https://10.1.7.1:4848/common/logViewer/logEntryDetail.jsf?instanceName=server&logLevel=SEVERE&logFile=server.log&recNumber=752>. The title bar says "Log Entry Detail — Mozilla Firefox (Private Browsing)". The log entry details are as follows:

Timestamp	Feb 23, 2022 09:25:56.643
Log Level	SEVERE
Logger	
Name-Value Pairs	{levelValue=1000, timeMillis=1645637156643}
Record Number	752
Message ID	
Complete Message	[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values (1234,10.7,1234,'0004 9839 0829 3274')

Fig. 16: Entrada en el log para la inserción de una entrada en la tabla pago.

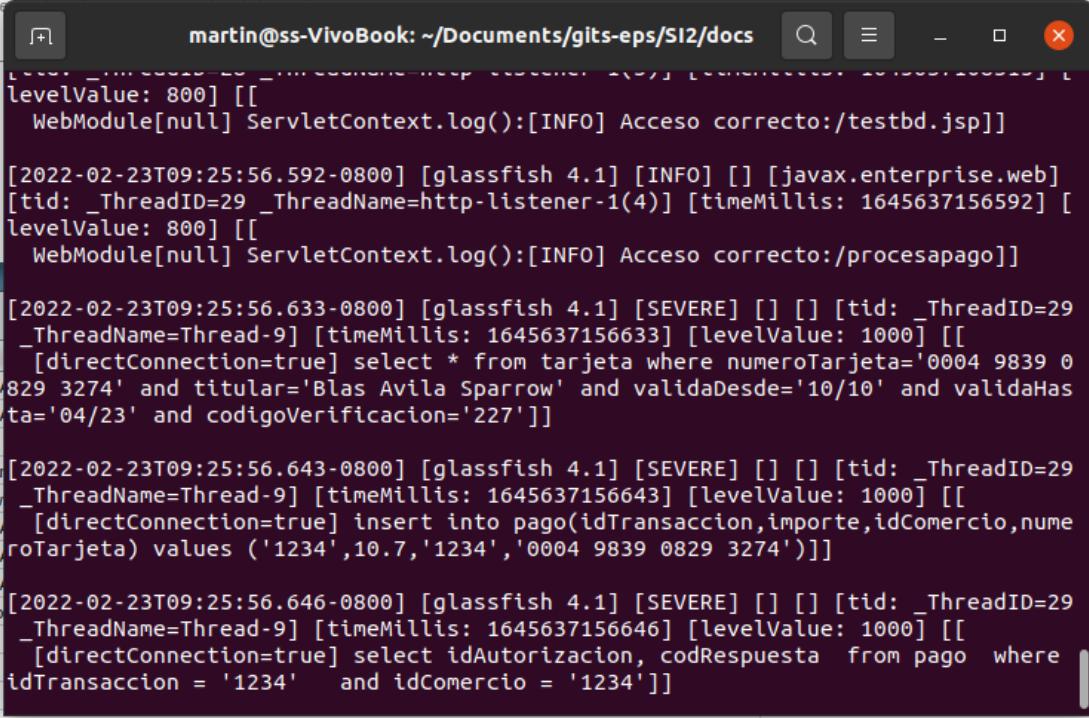


The screenshot shows a Firefox browser window with the URL <https://10.1.7.1:4848/common/logViewer/logEntryDetail.jsf?instanceName=server&logLevel=SEVERE&logFile=server.log&recNumber=753>. The title bar says "Log Entry Detail — Mozilla Firefox (Private Browsing)". The log entry details are as follows:

Timestamp	Feb 23, 2022 09:25:56.646
Log Level	SEVERE
Logger	
Name-Value Pairs	{levelValue=1000, timeMillis=1645637156646}
Record Number	753
Message ID	
Complete Message	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = '1234' and idComercio = '1234'

Fig. 17: Entrada en el log para la obtención del id de autorización de una transacción.

vel in the



```
martin@ss-VivoBook: ~/Documents/gits-eps/SI2/docs
[2022-02-23T09:25:56.592-0800] [glassfish 4.1] [INFO] [] [javax.enterprise.web]
[tid: _ThreadID=29 _ThreadName=http-listener-1(4)] [timeMillis: 1645637156592] [
levelValue: 800] [[
WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp]

[2022-02-23T09:25:56.633-0800] [glassfish 4.1] [INFO] [] [javax.enterprise.web]
[tid: _ThreadID=29 _ThreadName=http-listener-1(4)] [timeMillis: 1645637156633] [levelValue: 800] [[
WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago]

[2022-02-23T09:25:56.633-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=29
(ThreadName=Thread-9] [timeMillis: 1645637156633] [levelValue: 1000] [[
[directConnection=true] select * from tarjeta where numeroTarjeta='0004 9839 0
829 3274' and titular='Blas Avila Sparrow' and validaDesde='10/10' and validaHas
ta='04/23' and codigoVerificacion='227']]]

[2022-02-23T09:25:56.643-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=29
(ThreadName=Thread-9] [timeMillis: 1645637156643] [levelValue: 1000] [[
[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numero
Tarjeta) values ('1234',10.7,'1234','0004 9839 0829 3274')]]]

[2022-02-23T09:25:56.646-0800] [glassfish 4.1] [SEVERE] [] [] [tid: _ThreadID=29
(ThreadName=Thread-9] [timeMillis: 1645637156646] [levelValue: 1000] [[
[directConnection=true] select idAutorizacion, codRespuesta from pago where
idTransaccion = '1234' and idComercio = '1234']]]

javax.enterprise.system.tools.deployment.common
```

Fig. 16: Entradas del log de las capturas anteriores pero vistas desde el fichero local.

Cuestión número 6:

Realíicense las modificaciones necesarias en VisaDAOWS.java para que implemente de manera correcta un servicio web. Publique los métodos indicados como métodos del servicio. En el caso de *isDirectConnection()* e *setDirectConnection()* que son métodos heredados de la clase *DBTester*. Para ello, implemente estos métodos también en la clase hija. Modifique así mismo el método *realizaPago()* para que éste devuelva el pago modificado tras la correcta o incorrecta realización del pago.

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 primer lugar, lo que haremos será hacer los imports pertinentes para disponer de las anotaciones para poder convertir la clase en un Web Service.

Una vez realizados estos imports, ya estamos en disposición de añadir la anotación `@WebService()` justo antes de definir la clase `VisaDAOWS`, para indicar que se trata de una clase que implementa un servicio web.

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Activities > Visual Studio Code - VisaDAOWS.java - SI2 - Visual Studio Code
- File Explorer:** Shows the project structure under 'SI2'. The 'src' folder contains Java files: DBTester.java, PagoBean.java, TarjetaBean.java, and VisaDAOWS.java (the active file). It also lists build properties, XML files, and JDBC drivers.
- Code Editor:** The active file is `VisaDAOWS.java`. The code implements a `WebService` for managing payments. It includes imports for `java.sql.Connection`, `java.sql.PreparedStatement`, `java.sql.ResultSet`, `java.sql.SQLException`, `java.sql.Statement`, `java.util.ArrayList`, `javax.jws.WebMethod`, `javax.jws.WebParam`, and `javax.jws.webservice`. The class `VisaDAOWS` extends `DBTester`. It has a private boolean `debug` set to `false`. A private boolean `prepared` is set to `true`. The class contains several annotations and methods, including `@WebMethod(operationName = "realizarPago")` and `DELETE_PAGO_QRY`.
- Bottom Status Bar:** Shows file navigation (main, C:\1, 10, 5, Martin), live share, and Java projects.

Fig. 17: imports y definición de la clase como servicio web.

Ahora lo que haremos será ir añadiendo a los diferentes métodos las anotaciones correspondientes, en primer lugar `@webMethod(operationName = "nombreMetodo")` para indicar que el método será exportado como método público del servicio. Además, en caso de que estos método reciban argumentos como parámetros habrá que indicar con una anotación `@WebParam(name = "nombreArgumento")`, para indicar que es un argumento del método del servicio.

```

    /**
     * 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(operationName = "compruebaTarjeta")
    public boolean compruebaTarjeta(@WebParam(name = "tarjeta") TarjetaBean tarjeta) {
        Connection con;
        Statement stat = null;
        ResultSet rs = null;
        boolean ret = false;
        String qry = null;

        try {
            // Crear una conexion u obtenerla del pool
            con = getConnection();
            // Se busca la ocurrencia de la tarjeta en la tabla
            /* TODO User.prepared_statement_si
            isPrepared() == true */
            *****
            if (isPrepared() == true) {
                String select = "SELECT TARJETA_ORY";
                errorLog(select);
                PreparedStatement stmt = con.prepareStatement(select);
                stmt.setString(1, tarjeta.getNumero());
                stmt.setString(2, tarjeta.getTitular());
                stmt.setString(3, tarjeta.getFechaEmision());
                stmt.setString(4, tarjeta.getFechaCaducidad());
                stmt.setString(5, tarjeta.getCodigoVerificacion());
                rs = stmt.executeQuery();
            } else {
                *****
                Connection con = createStatement();
                qry = getQueryCompruebaTarjeta(tarjeta);
                errorLog(qry);
                rs = stmt.executeQuery(qry);
            }
        } catch (SQLException e) {
            return ret;
        }
    }

    /**
     * Realiza el pago
     * @param pago
     * @return
     */
    @WebMethod(operationName = "realizaPago")
    public synchronized PagoBean realizaPago(@WebParam(name = "pago") PagoBean pago) {
        Connection con = null;
        Statement stat = null;
        ResultSet rs = null;
        boolean ret = false;
        String codResuesta = "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;
        }

        // Registrar el pago en la base de datos
        try {
            // Obtener conexion
            con = getConnection();

            // Insertar en la base de datos el pago
            /* TODO User.prepared_statement_si
            isPrepared() == true */
            *****
            if (isPrepared() == true) {
                String insert = INSERT_PAGOS_QRY;
                errorLog(insert);
                pstmt = con.prepareStatement(insert);
                pstmt.setString(1, pago.getIdTransaccion());
            }
        } catch (SQLException e) {
            return ret;
        }
    }
}

```

Fig. 18: anotación para indicar que el método será exportado como método público de servicio y especificación de que el parámetro tarjeta es un parámetro del mismo.

```

    /**
     * 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(operationName = "compruebaTarjeta")
    public boolean compruebaTarjeta(@WebParam(name = "tarjeta") TarjetaBean tarjeta) {
        Connection con;
        Statement stat = null;
        ResultSet rs = null;
        boolean ret = false;
        String qry = null;

        try {
            // Crear una conexion u obtenerla del pool
            con = getConnection();
            // Se busca la ocurrencia de la tarjeta en la tabla
            /* TODO User.prepared_statement_si
            isPrepared() == true */
            *****
            if (isPrepared() == true) {
                String select = "SELECT TARJETA_ORY";
                errorLog(select);
                PreparedStatement stmt = con.prepareStatement(select);
                stmt.setString(1, tarjeta.getNumero());
                stmt.setString(2, tarjeta.getTitular());
                stmt.setString(3, tarjeta.getFechaEmision());
                stmt.setString(4, tarjeta.getFechaCaducidad());
                stmt.setString(5, tarjeta.getCodigoVerificacion());
                rs = stmt.executeQuery();
            } else {
                *****
                Connection con = createStatement();
                qry = getQueryCompruebaTarjeta(tarjeta);
                errorLog(qry);
                rs = stmt.executeQuery(qry);
            }
        } catch (SQLException e) {
            return ret;
        }
    }

    /**
     * Realiza el pago
     * @param pago
     * @return
     */
    @WebMethod(operationName = "realizaPago")
    public synchronized PagoBean realizaPago(@WebParam(name = "pago") PagoBean pago) {
        Connection con = null;
        Statement stat = null;
        ResultSet rs = null;
        boolean ret = false;
        String codResuesta = "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;
        }

        // Registrar el pago en la base de datos
        try {
            // Obtener conexion
            con = getConnection();

            // Insertar en la base de datos el pago
            /* TODO User.prepared_statement_si
            isPrepared() == true */
            *****
            if (isPrepared() == true) {
                String insert = INSERT_PAGOS_QRY;
                errorLog(insert);
                pstmt = con.prepareStatement(insert);
                pstmt.setString(1, pago.getIdTransaccion());
            }
        } catch (SQLException e) {
            return ret;
        }
    }
}

```

Fig. 19: hacemos lo mismo que en el caso anterior para exportar el método como método público de servicio, además cambiamos el tipo del retorno por un objeto de la clase PagoBean.

```

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

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

```

Fig. 20: modificaciones pertinentes del retorno del método realizaPago, para devolver un objeto del tipo PagoBean o null en caso de error.

```

    ...
    } catch (SQLException e) {
        ...
    }
    return ret;
}

< /**
 * _T000...Metodos isPrepared() y setPrepared()
 */
*****
@WebMethod(operationName = "isPrepared")
public boolean isPrepared() {
    return prepared;
}
*****

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

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

< /**
 * @param debug the debug to set
 */
*****
@WebMethod(exclude=true)
public void setDebugEnabled(String debug) {
    this.debug = (debug.equals("true"));
}
*****

```

Fig. 21: exportamos los métodos correspondientes como métodos públicos de servicio. Además, se evita que el método setDebugEnabled sea incluido en el WSDL mediante el uso de (exclude=true).

```

    481     * @param debug the debug to set
    482     */
    483     @WebMethod(exclude=true)
    484     public void setDebug(String debug) {
    485         this.debug = (debug.equals("true"));
    486     }
    487
    488     /**
    489     * Imprime traza de depuración
    490     */
    491     public void errorLog(String error) {
    492         if (isDebugEnabled())
    493             System.err.println("[" + directConnection() + "] " +
    494                             error);
    495     }
    496
    497     /**
    498      * Override methods from parent class
    499      * Note: Annotations are not needed
    500      */
    501     @Override
    502     @WebMethod(operationName = "isDirectConnection")
    503     public boolean isDirectConnection() {
    504         return super.isDirectConnection();
    505     }
    506
    507     @Override
    508     @WebMethod(operationName = "setDirectConnection")
    509     public void setDirectConnection(@WebParam(name = "directConnection") boolean directConnection) {
    510         super.setDirectConnection(directConnection);
    511     }
    512
    513 }

```

Fig. 22: Igual que en los casos anteriores, pero sobreescritiendo los métodos de la clase padre.

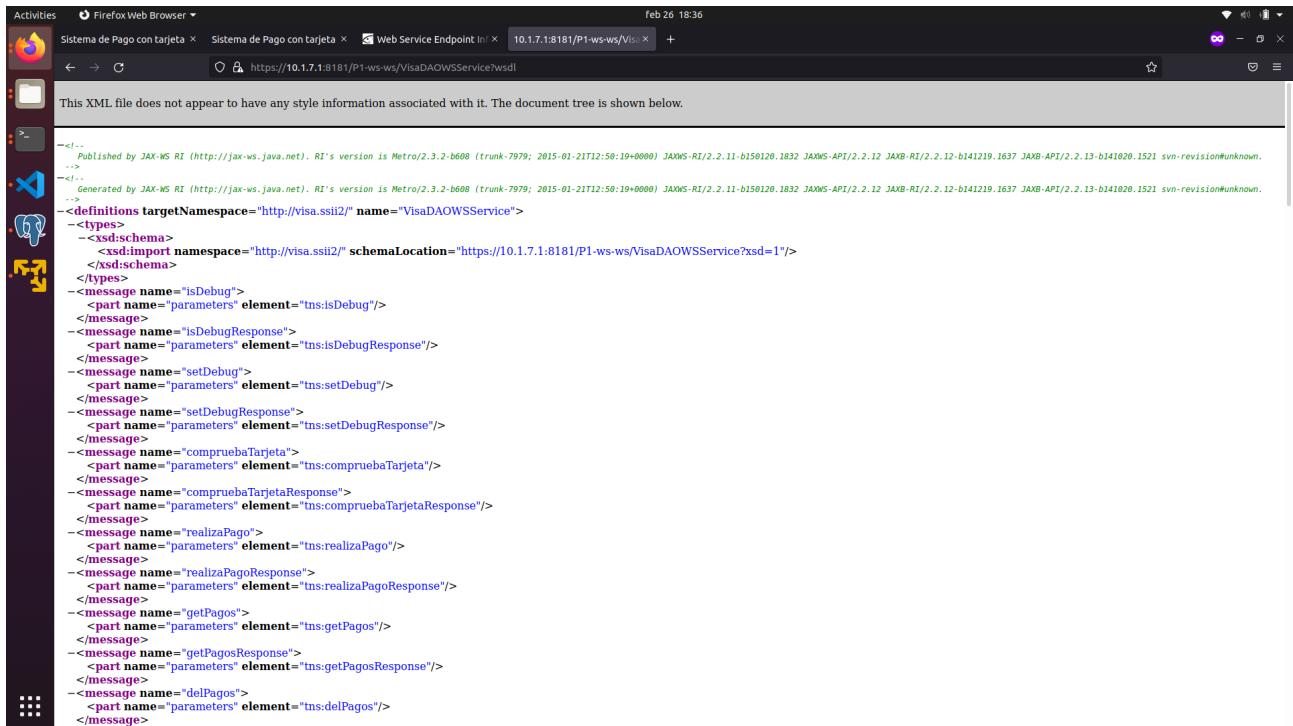
- ¿Por qué se ha de alterar el parámetro de retorno del método realizarPago() para que devuelva el pago en lugar de un booleano?

Esto se debe a que anteriormente todo el código de cliente y de servidor era común, de manera que en el propio método el servidor realizaba el pago a petición del cliente, de esta manera tenía sentido que este devolviese un booleano a modo de que el servidor indicase si la operación se había realizado con éxito o no. Por otro lado, al hacer la distinción entre el cliente y el servidor, ahora se deberá de generar un objeto de tipo PagoBean que será enviado, con la información correspondiente, al lado del servidor para que este realice los cambios pertinentes.

Cuestión número 7:

Desplegar el servicio con la regla correspondiente en build.xml. Acceder al WSDL remotamente con el navegador e inclúyalo en la memoria. Comente los aspectos relevantes del código XML del fichero WSDL y su relación con los métodos Java del objeto del servicio, argumentos recibidos y objetos devueltos. Conteste a las cuestiones listadas.

En primer lugar comentar cual es la información principal que encontraremos en el fichero WSDL. Como podemos ver en la captura adjunta a la parte inferior de esta cuestión, en el fichero WDSL, encontramos un listado con los diferentes métodos exportados, salvo aquellos que tuviesen la especificación “export=false” en sus argumentos. En este listado encontramos información acerca de cada uno de estos métodos, en forma de mensajes, tanto el nombre de dicho método como sus parámetros de entrada. Por otro lado, en la parte superior del fichero XML, encontramos la parte debida a la definición del servicio web, como el nombre y la localización del mismo.



The screenshot shows a Firefox browser window with the title bar "Activities Firefox Web Browser". Below it, there are several tabs: "Sistema de Pago con Tarjeta", "Sistema de Pago con tarjeta", "Web Service Endpoint Info", and "10.1.7.1:8181/P1-ws-ws/VisaDAOWSService?wsdl". The main content area displays the WSDL XML code. The code starts with a header indicating it was published by JAX-WS RI (version Metro/2.3.2-b608) and generated by JAX-WS RI (version Metro/2.3.2-b608). It defines a target namespace "http://visa.ssl2/" for the service. The service contains various messages such as "isDebugEnabled", "setDebugResponse", "compruebaTarjeta", "realizaPago", "getPagos", and "delPagos", each with their respective parameters and responses. The XML uses namespaces like "tns" and "xsd".

```
<!--
Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revisionUnknown.
-->
<!--
Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-7979; 2015-01-21T12:50:19+0000) JAXWS-RI/2.2.11-b150120.1832 JAXWS-API/2.2.12 JAXB-RI/2.2.12-b141219.1637 JAXB-API/2.2.13-b141020.1521 svn-revisionUnknown.
-->
<definitions targetNamespace="http://visa.ssl2/" name="VisaDAOWSService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://visa.ssl2/" schemaLocation="https://10.1.7.1:8181/P1-ws-ws/VisaDAOWSService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="isDebugEnabled">
    <part name="parameters" element="tns:isDebugEnabled"/>
  </message>
  <message name="isDebugEnabledResponse">
    <part name="parameters" element="tns:isDebugEnabledResponse"/>
  </message>
  <message name="setDebug">
    <part name="parameters" element="tns:setDebug"/>
  </message>
  <message name="setDebugResponse">
    <part name="parameters" element="tns:setDebugResponse"/>
  </message>
  <message name="compruebaTarjeta">
    <part name="parameters" element="tns:compruebaTarjeta"/>
  </message>
  <message name="compruebaTarjetaResponse">
    <part name="parameters" element="tns:compruebaTarjetaResponse"/>
  </message>
  <message name="realizaPago">
    <part name="parameters" element="tns:realizaPago"/>
  </message>
  <message name="realizaPagoResponse">
    <part name="parameters" element="tns:realizaPagoResponse"/>
  </message>
  <message name="getPagos">
    <part name="parameters" element="tns:getPagos"/>
  </message>
  <message name="getPagosResponse">
    <part name="parameters" element="tns:getPagosResponse"/>
  </message>
  <message name="delPagos">
    <part name="parameters" element="tns:delPagos"/>
  </message>
</definitions>
```

Fig. 23: fichero WSDL.

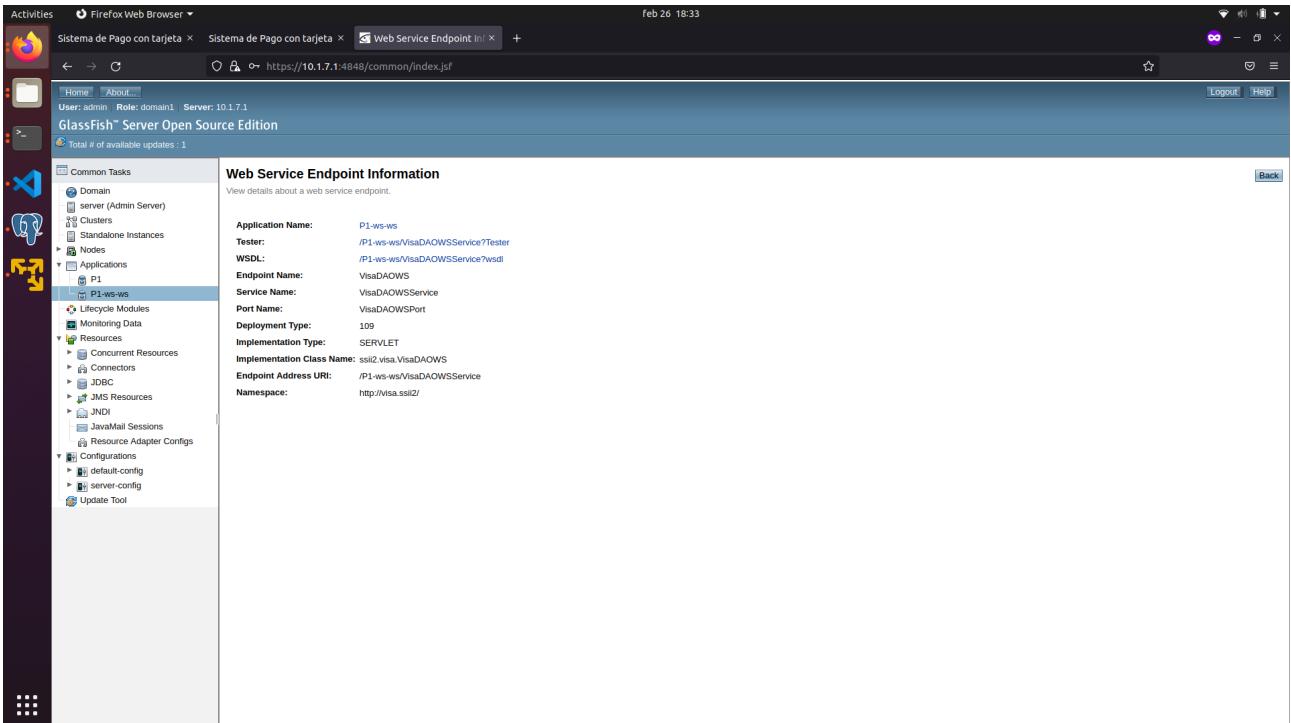


Fig. 24: página asociada a la gestión del servicio web en cuestión.

Conteste a las siguientes cuestiones:

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

Los tipos de datos vienen especificados en el fichero XSD, el cual se importa en el WSDL, en particular nuestra URL es <http://10.1.7.1:8080/P1-ws-ws/VisaDAOSService?xsd=1>

- **¿ Qué tipos de datos predefinidos se usan ?**

Dentro de los tipos de datos predefinidos, hay 3 que están presentes en nuestro fichero WSDL, estos son, xs:boolean, xs:string y xs:int. Estos tipos aparecen en los argumentos de métodos como getPagos o setDebug y en retornos como el de delPagosResponse.

- **¿ Cuáles son los tipos de datos que se definen ?**

Los tipos de datos que se definen, son aquellos los cuales, en el fichero XSD aparecen indicados mediante la etiqueta complexType. Así realizando un análisis del fichero, podemos elaborar un listado de cuáles son estos tipos de datos que se definen:

- comprouebaTarjeta
- comprouebaTarjetaResponse
- delPagos
- delPagosResponse
- isDebug
- isDebugResponse
- isDirectConnection
- isDirectConnectionResponse
- isPrepared
- isPreparedResponse
- setPrepared
- setPreparedResponse
- getPagos
- getPagosResponse
- pagoBean
- tarjetaBean

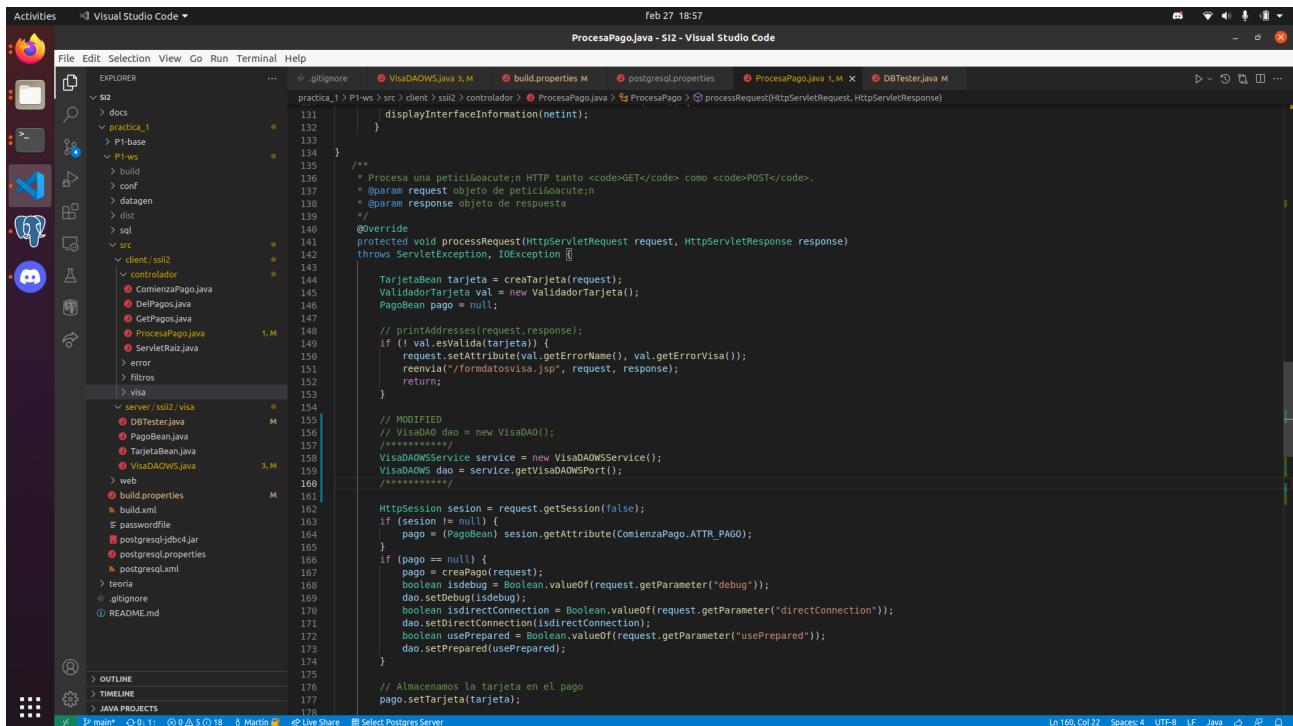
Los cuales, salvo pagoBean y tarjetaBean, que son clases de java, se tratan de métodos públicos del servicio web.

- **¿ Qué etiqueta está asociada a los métodos invocados en el webservice ?**
Es la etiqueta operation, existiendo 4 primitivas de transmisión, One-way, Request-response, Solicit-response y Notification.
- **¿ Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice ?**
En la etiqueta binding, es donde se describen los mensajes intercambiados en la invocación de los métodos del webservice.
- **¿ En qué etiqueta se especifica el protocolo de comunicación con el webservice ?**
En la etiqueta soap, se especifica el protocolo de comunicación con el webservice.
- **¿ En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice ?**
La etiqueta service, contiene información de la dirección en la que se localiza el servicio. Para ser más exactos, la dirección del servicio web podremos encontrarla en el nodo hijo con etiqueta soap:address.

Cuestión número 8:

Realice las modificaciones necesarias en *ProcesaPago.java* para que implemente de manera correcta la llamada al servicio web mediante *stubs estáticos*.

En primer lugar adjuntamos una captura en la cual observamos cómo cambiamos la declaración del objeto VisaDAO dao, antes era mediante su constructor por defecto, pero ahora, lo haremos mediante un objeto de la clase VisaDAOWSService que es una referencia al servicio remoto, que nos permitirá obtener el stub local. Para ello lo invocamos una única vez en el método. Una vez teniendo la instancia de este servicio remoto, obtenemos el objeto dao como instancia de un objeto VISADAOWS obtenido mediante el método getVisaDAOWSPort.

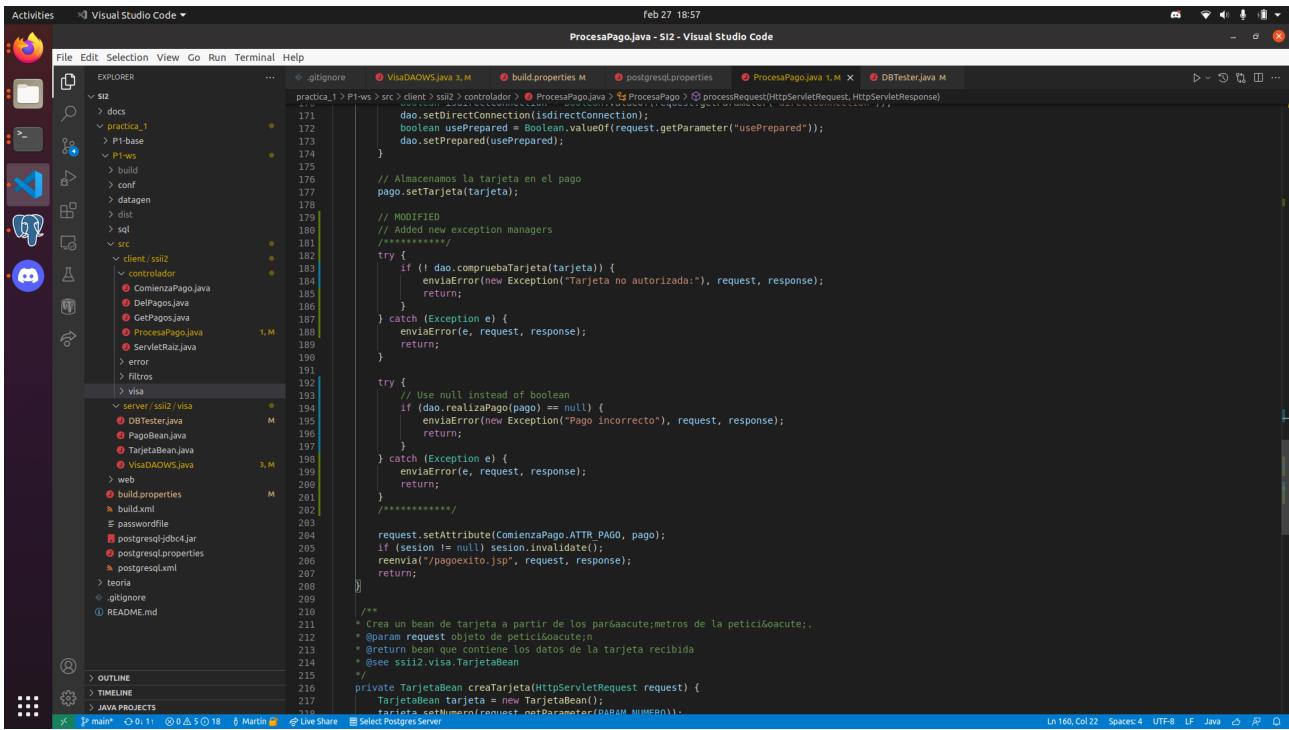


The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Activities, Visual Studio Code, ProcesaPago.java - SI2 - Visual Studio Code, feb 27 18:57.
- File Explorer:** Shows the project structure under 'SI2'. Key files include: .gitignore, VisaDAO.java, build.properties, postgresql.properties, Prueba.java, processRequest(HttpServletRequest, HttpServletResponse), DBTester.java, and several Java files like ComienzaPago.java, PagoBean.java, TarjetaBean.java, and VisaDAOWS.java.
- Code Editor:** The main view displays the Java code for 'ProcesaPago.java'. The code is annotated with comments and includes imports for javax.servlet, javax.servlet.http, java.util, and other packages. It defines a class 'ProcesaPago' with methods 'processRequest' and 'createTarjeta'. The 'processRequest' method uses a static reference to 'VisaDAOWSService' to get an instance of 'VisaDAOWS', which it then uses to call 'getVisaDAOWSPort()' to obtain a local stub object 'dao'. This 'dao' is used to handle payment logic.
- Status Bar:** Shows file statistics (Ln 160, Col 22, Spaces: 4, UTF-8, LF, Java), a 'Select Postgres Server' button, and other standard status icons.

Fig. 25: modificaciones realizadas para llamar al servicio web mediante stubs estáticos.

Por otro lado, como se nos indicaba en el enunciado, se han tomado las medidas pertinentes para controlar las nuevas excepciones que son inducidas por realizar las llamadas a servicios web a través de stubs estáticos.



The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Activities, Visual Studio Code, ProcesaPago.java - SI2 - Visual Studio Code, feb 27 18:57
- Code Editor:** The main area displays Java code for the `ProcesaPago.java` file. The code handles payment processing, including connection setup, card storage, exception handling for card validation, and session management. It includes annotations like `@Param` and `@Return`. Lines 193-217 show the creation of a `TarjetaBean` from a `HttpServletRequest`.
- Explorer:** On the left, the Explorer sidebar shows the project structure under the `SI2` folder, including subfolders like `client`, `docs`, `practica_1`, `P1-ws`, `server/ssl2`, and `web`, along with various Java files and configuration files.
- Status Bar:** At the bottom, it shows file statistics (Ln 160, Col 22, Spaces:4, UTF-8), language (LF, Java), and other UI elements.

Fig. 26: adiciones a los controladores de excepciones.

Cuestión número 9:

Modifique la llamada al servicio para que la ruta (URL) al servicio remoto se obtenga del fichero de configuración web.xml.

Siguiendo las pautas dadas en el apéndice 15.1, lo primero que haremos será añadir un nuevo parámetro de inicialización al archivo web.xml, el cual tendrá como valor la ruta URL al servicio remoto en cuestión.

```
<context-param>
    <param-name>visadaows</param-name>
    <param-value>http://10.1.7.1:8080/P1-ws-ws/VisaDAOSService</param-value>
</context-param>
```

Fig. 27: parámetro de inicialización con la ruta URL del servicio remoto.

Una vez teniendo esto, lo que haremos será pedir dicha URL desde ProcesaPago.java.

```
 |   /*****
|   |   BindingProvider bp = (BindingProvider) dao;
|   |   String remote_server_url = getServletContext().getInitParameter("visadaows");
|   |   bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, remote_server_url);
|   |   *****/
|   |
```

Fig. 28: Llamadas para obtener el URL del fichero web.xml.

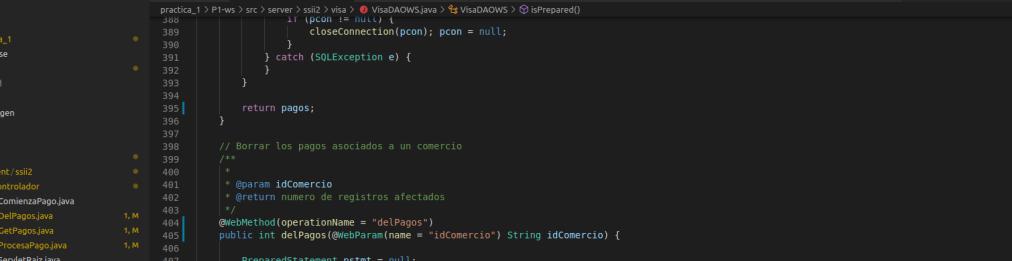
Cuestión número 10:

Siguiendo el patrón de los pagos anteriores, adaptar las clases DelPagos.java u GetPagos.java para que toda la funcionalidad de la página de pruebas testbd.jsp se realice a través del servicio web.

Ahora lo que haremos, será seguir los pasos tomados en los apartados 8 y 9, pero para las clases DelPagos y GetPagos, es decir, modificar estas como ya previamente habíamos modificado ProcesaPago.

En primer lugar, no se han tenido que realizar cambios en web.xml, pues el parámetro de inicialización con el URL, ya estaba creado previamente para el apartado 9.

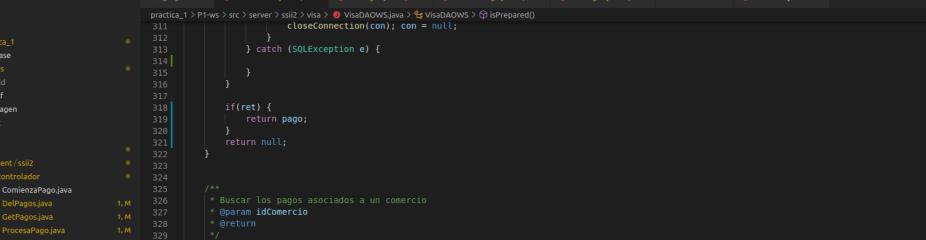
En primer lugar, modificará el fichero VisaDAOWS, modificando los métodos delPagos y getPagos:



The screenshot shows the Visual Studio Code interface with the following details:

- Activity Bar:** Activities, Visual Studio Code, Notifications, Taskbar.
- File Explorer:** Shows the project structure under the `siz` folder:
 - `docs`
 - `practica_1`
 - `P1-base`
 - `P1-ws`
 - `build`
 - `conf`
 - `datagen`
 - `dist`
 - `sql`
 - `src`
 - `client/ssl2`
 - `controlador`
 - `ComienzaPago.java`
 - `DelPago.java`
 - `GetPagos.java`
 - `ProcesaPago.java`
 - `ServletRaiz.java`
 - `error`
 - `filtros`
 - `visa`
 - `server/ssl2/visa`
 - `DBTester.java`
 - `PadroBase.java`
 - `TarjetaBuenPool.java`
 - `VISAOWS.java`
 - `web`
 - `error`
 - `WEB-INF`
 - `web.xml`
 - `borradorerror.jsp`
 - `borradorok.jsp`
 - `cabecera.jsp`
 - `formatovisita.jsp`
 - `listpagos.jsp`
 - `pago.html`
 - `pagoexitoso.jsp`
 - `pie.html`
 - `testbd.jsp`
- Editor:** The `VISAOWS.java` file is open, showing Java code for a DAO class. The code includes methods for inserting, updating, and deleting payment records, as well as handling connections and statements.
- Bottom Status Bar:** Shows file paths like `practica_1>P1-ws>src>server>ssl2>visa>VISAOWS.java`, line numbers (e.g., 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435), and other status indicators like `JAVA PROJECTS`.

Fig. 29: añadimos las anotaciones pertinentes al método delPagos, exportando el método como método público de servicio e indicando lo mismo para los argumentos.

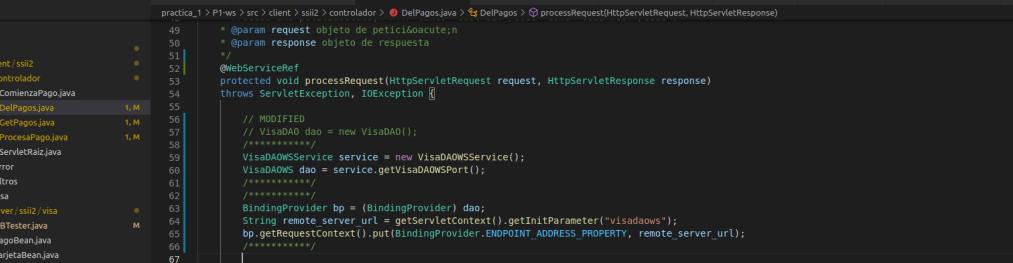


The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Activities, Visual Studio Code, Feb 27 19:34, VisaDAOWS.java - S12 - Visual Studio Code
- File Explorer:** Shows the project structure for 'S12'. The 'src' folder contains packages like 'com.practica_1' and 'com.practica_1.P1ws'. Under 'com.practica_1.P1ws', there are several Java files: 'ComercioPago.java', 'Delfago.java', 'GetPago.java', 'ProcesarPago.java', 'ServidorRaiz.java', 'Visa.java', and 'VisaDAOWS.java'. Other files like 'client/ssl2/client', 'Filtros.java', 'visa', 'server/ssl2/visa/DBTester.java', 'PagoBean.java', 'TarjetaBean.java', and 'web.xml' are also listed.
- Code Editor:** The main editor window displays the Java code for 'VisaDAOWS.java'. The code handles database operations using JDBC, including preparing statements, executing queries, and handling results. It includes annotations like @Param and @WebParam.
- Bottom Status Bar:** Shows file paths (VisaDAOWS.java, web.xml, DBTester.java), line numbers (Ln 453, Col 8), and other status indicators.

Fig. 30: al igual que en el caso anterior se añaden las anotaciones pertinentes, pero en este caso, también cambiamos el tipo de retorno del método, puesto que necesitamos que este sea compatible con JAXB.

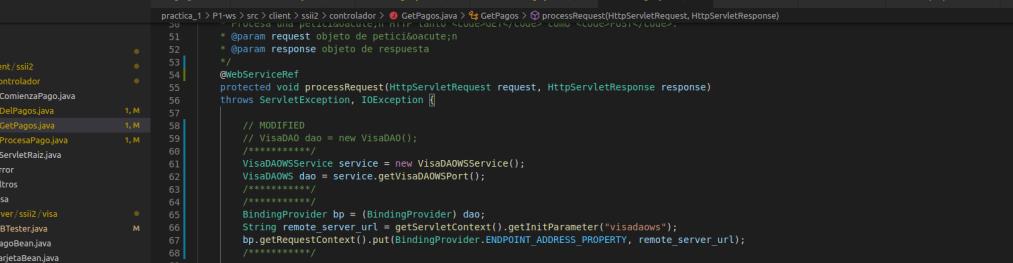
Por otro lado, también debemos realizar cambios en las propias clases de los servicios, DelPagos.java y GetPagos.java. Lo primero que haremos será añadir la anotación a los métodos correspondientes, @WebServiceRef, usada para definir una referencia a un servicio web. Una vez hecho esto, lo que hacemos es seguir, como ya mencionamos antes, los pasos marcados por los apartados anteriores. En primer lugar, realizamos las modificaciones para que las llamadas al servicio se produzcan mediante stubs estáticos, con sus correspondientes controles de las nuevas excepciones que se puedan producir. Por último, añadimos el código encargado de que la ruta al servicio remoto se obtenga de web.xml.



The screenshot shows the Visual Studio Code interface with the following details:

- Activity Bar:** Activities, Visual Studio Code, Notifications, Taskbar.
- Header:** feb 27 21:29, DelPagos.java - S12 - Visual Studio Code.
- Sidebar:** Explorer (File tree), Search, Open, Terminal, Help.
- Editor:** The main editor window displays the Java code for `DelPagos.java`. The code handles a `processRequest` method, which interacts with a `VisaDAO` and a `VisaAOWSService` to process payment requests. It also manages session attributes and sends responses back to the client.
- Bottom Status Bar:** Shows file paths like `src/main/java/com/comercia/pago/DelPagos.java`, line numbers (e.g., 1, M), and icons for build, run, and terminal.

Fig. 31: cambios realizados en DelPagos.java.



The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** Activities > Visual Studio Code - feb 27 21:29
- File Explorer:** Shows the project structure under 'S12' (a folder). The 'src' folder contains several Java files: `Practica_1_P1ws>src>client/ssl2/controlador`, `GetPagos.java`, `ProcesarPago.java`, `ProcesarRequest.java`, `DelPagos.java`, `ComercioPago.java`, `DelPagos.java`, `GetPagos.java`, `ProcesarPago.java`, `ServletRaiz.java`, `error`, `filtros`, `visa`, `server/ssl2/visa`, `DBTester.java`, `PagoBean.java`, `TarjetaBean.java`, `VisaDAOWS.java`, `web`, `error`, and `WEB-INF/web.xml`. There are also some binary files like `borradeor.jsp`, `horadok.jsp`, `cabeceira.jsp`, `formdatosvisita.jsp`, `listpagos.jsp`, `pagos.html`, `pagowebto.jsp`, `pie.html`, `testbd.jsp`, `build.properties`, `build.xml`, `passwordfile`, `postgresql-jdbc4.jar`, `postgresql-properties`, `postgresql.xml`, `teoria`, and `gitignore`.
- Code Editor:** The main editor window displays the `GetPagos.java` file. The code implements a `BindingProvider` named `bp` to interact with a `VisaDAO` service. It handles parameters like `remote_server_url` and `idComercio`. The code includes annotations for `param request` and `param response`, and an `override` method for `doGet`.

Fig. 32: cambios realizados en GetPagos.java.

Cuestión número 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.

En primer lugar, adjuntamos el comando utilizado para generar las clases así como una captura de las mismas:

```
wsimport -d build/client/WEB-INF/classes/ -p ssii2.visa  
http://10.1.7.1:8080/P1-ws-ws/VisaDAOSService?wsdl
```

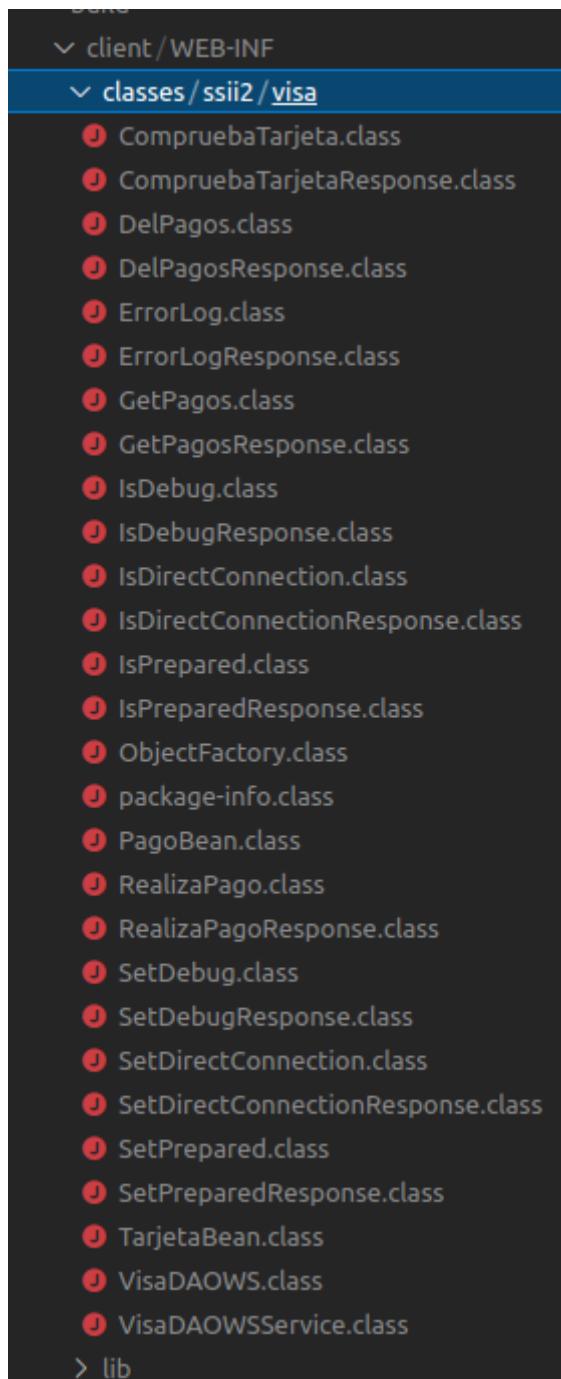


Fig. 33: clases generadas tras la ejecución del comando.

Estas clases generadas son los métodos que tenían la anotación de webMethod. Estas clases se generan para que se puedan llevar a cabo llamadas a los procedimientos.

Cuestión número 12:

Complete el target generar-stubs definido en build.xml para que invoque a wsimport.

Realizamos las modificaciones pertinentes en build.xml.

```
<target name="generar-stubs" depends="montar-jerarquia" description="Genera los stubs del cliente a partir del archivo WSDL">
    <!-- TODO - Implementar llamada wsimport -->
    <exec executable="${wsimport}">
        <arg line="-d ${build.client}/WEB-INF/classes" />
        <arg line="-p ${paquete}.visa ${wsdl.url}" />
    </exec>
    <delete file="${build}/${tmpvisaclientjar}" />
    <jar jarfile="${build}/${tmpvisaclientjar}" >
        <fileset dir="${build.client}/WEB-INF/classes" />
    </jar>
    <move file="${build}/${tmpvisaclientjar}" todir="${build.client}/WEB-INF/lib" />
</target>
```

Fig. 34: Modificaciones realizadas en build.xml.

Cuestión número 13:

Realice un despliegue de la aplicación completo en dos nodos. Pruebe a realizar pagos correctos a través de la página testbd.jsp y ejecute las consultas SQL necesarias para comprobar que se realiza el pago. Anote en la memoria práctica los resultados en forma de consulta SQL y resultados sobre la tabla de pagos, además de evidencia de la realización.

Incluimos una serie de capturas con un pago realizado con éxito y el lisado de pagos:

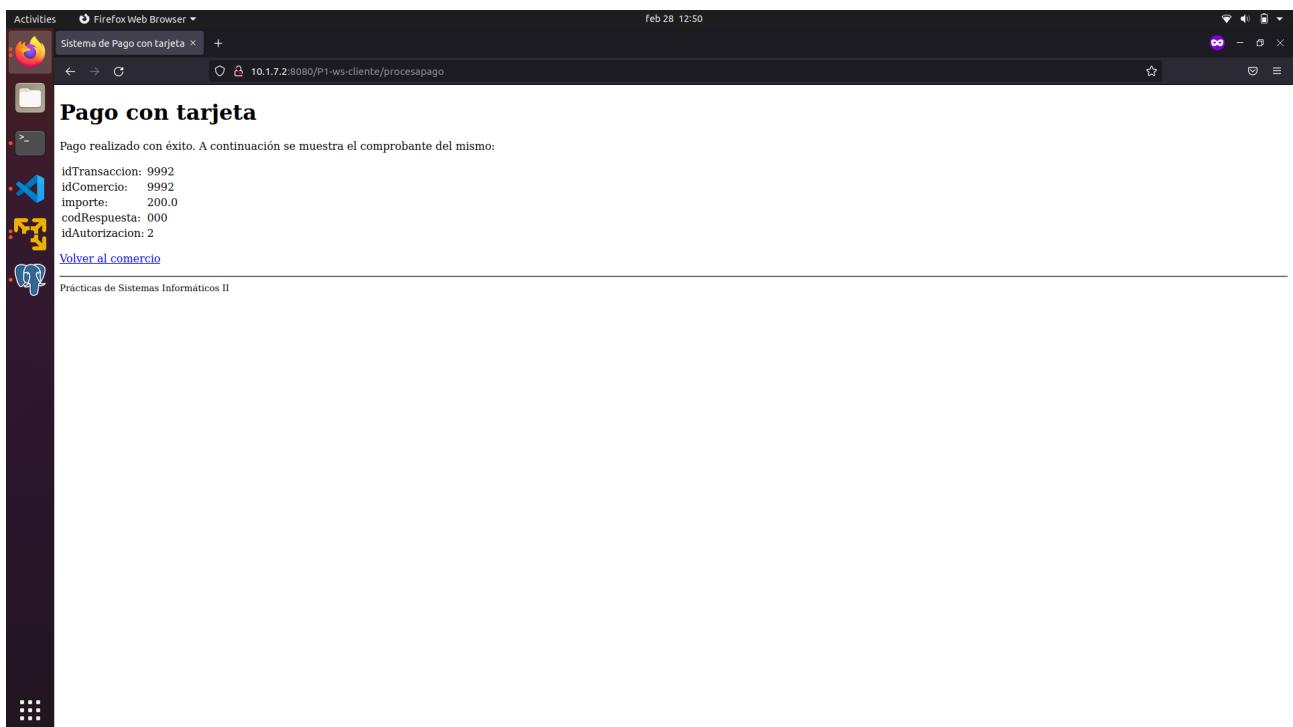


Fig. 35: Pago realizado con éxito.

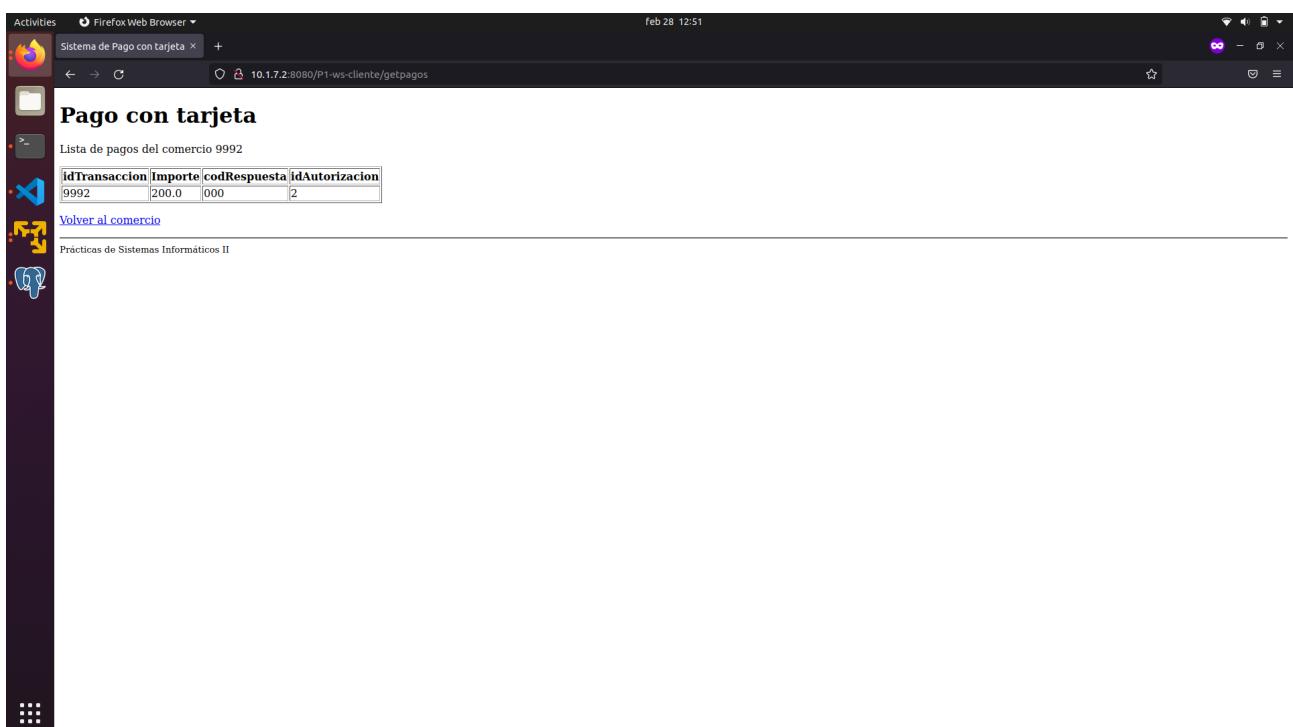


Fig. 35: Lista de pagos una vez realizado.

Cuestión número 14:

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.

En primer lugar se envía el formulario desde pago.htm, que se trata de la primera página que se le muestra al usuario que quiere realizar un pago, la información del mismo será procesada por el servlet ComienzaPago.java que una vez finaliza, le envía al usuario formdatosvisa.jsp, formulario para recoger los datos de la tarjeta VISA. Una vez completado y enviado el formulario, éste se le envía al servlet ProcesaPago.java, el cual interactuando con el servlet VisaDAO.java, verifica que la fecha de caducidad ha expirado y devuelve al usuario el formulario formdatosvisa.jsp, pero esta vez mostrándole al mismo un mensaje de error.

Cuestión número 15:

Cuestión 2) De los diferentes servlets (recuerde que las páginas jsp también se compilan a un servlet) que se usan en la aplicación, ¿podría indicar cuáles son los encargados de obtener la información sobre el pago con tarjeta cuando se usa pago.html para realizar el pago, y cuáles son los encargados de procesarla? ¿Qué información obtiene y procesa cada uno?

Obtener información:

- ComienzaPago.java: procesa el formulario que el usuario envía desde pago.html que tiene la información acerca de la transacción.
- formdatosvisa.jsp: este es el que es enviado una vez procesada la información por parte de ComienzaPago.java, se encargará de recopilar los datos de la cuenta bancaria.

Procesar información:

- ProcesaPago.java: a este se le enviará la información obtenida previamente y será el encargado de realizar las modificaciones pertinentes en la base de datos. Para realizar estos cambios, hará uso de los diferentes servicios web del servlet VisaDAO.java.

Cuestión número 16:

Cuestión 3) ¿Dónde se crea la instancia de la clase pago cuando se accede por pago.html? ¿Y cuándo se accede por testbd.jsp? Respecto a la información que manejan, ¿cómo la comparte entre los distintos servlets? ¿dónde se almacena? ¿dónde se crea ese almacén?

A partir de la dirección de la acción de los formularios, encontramos en el fichero web.xml el servlet encargado de la gestión de dicho formulario. Así pues, tenemos que:

- pago.html: El formulario se gestiona en el servlet ComienzaPago, donde en su método processRequest se obtiene el objeto pago con una llamada a creaPago, función donde se instancia la clase PagoBean
- testbd.jsp: El formulario es gestionado por el servlet ProcesaPago, y de manera similar, en el método processRequest de su clase es donde se obtiene el pago. Para ello, o bien se utiliza el pago ya existente en la sesión o bien se instancia uno nuevo con el método creaPago. Dado que se hace uso de la página testbd.jsp, no se pasa por el servlet ComienzaPago y por tanto ocurre lo segundo, instanciando un nuevo objeto en el cuerpo de la función processRequest de ProcesaPago.

La información que manejan se comparte mediante sesiones HTTP, utilizando objetos HttpServletRequest y HttpServletResponse. Esta sesión se almacena en el navegador el cuál gestiona su almacén. Una vez validada la información se almacena a su vez en objetos Java que hacen uso de ella.

Cuestión número 17:

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 diferencia principal surge de que al utilizar la página pago.html, se hace uso del servlet ComienzaPago, el cual prosigue con el flujo estándar del servicio: instancia un pago, luego se redirige a formdatosvisa.jsp de

donde se prosigue con la validación de los datos para ya continuar empleando el servlet ProcesaPago. Sin embargo, al acceder mediante testbd.jsp, se hace uso directo del servlet ProcesaPago, sin pasar previamente por formdatosvisa.jsp.

El pago funciona correctamente pese a dicha diferencia porque el servlet encargado de acceder a la base de datos y realizar las comprobaciones necesarias es ProcesaPago. Por ello, basta con especificar toda la información requerida en testbd.jsp para llevar a cabo un pago, pues éste hace uso del servlet ProcesaPago que es el que realmente realiza el pago.