


| | | | | | |
|---|-------------|---|----|--------------|------------|
|  | | Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2 | | | |
| Grupo | 2402 | Práctica | 1A | Fecha | 01/03/2021 |
| Alumno/a | | Rivas, Molina, Lucía | | | |
| Alumno/a | | Santo-Tomás, López, Daniel | | | |

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

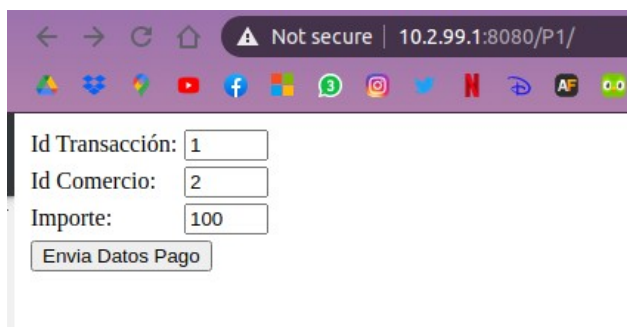
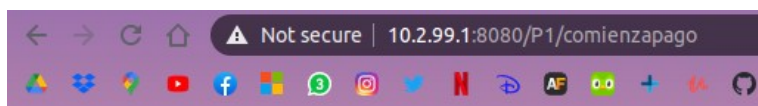
Ejercicio número 1:

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.

Básicamente hemos modificado los cambios indicados en el pdf de la actividad, poniendo la IP del host como 10.11.10.1 en el build.properties, la IP del host en el postgresql.properties como 10.11.10.1 y en el mismo fichero la IP del cliente como 10.11.10.2.

Realice un pago contra la aplicación web empleando el navegador en la ruta <http://10.X.Y.Z:8080/P1>.

Primero creamos una transacción y luego pagamos con la tarjeta indicada en las fotos, realizaremos dos pagos más (como podremos observar más abajo en la fotografía de la base de datos) para el apartado siguiente:

Pago con tarjeta

| | |
|--------------------------------------|--|
| Numero de visa: | <input type="text" value="1111 2222 3333 4444"/> |
| Titular: | <input type="text" value="Jose Garcia"/> |
| Fecha Emisión: | <input type="text" value="11/09"/> |
| Fecha Caducidad: | <input type="text" value="11/22"/> |
| CVV2: | <input type="text" value="123"/> |
| <input type="button" value="Pagar"/> | |

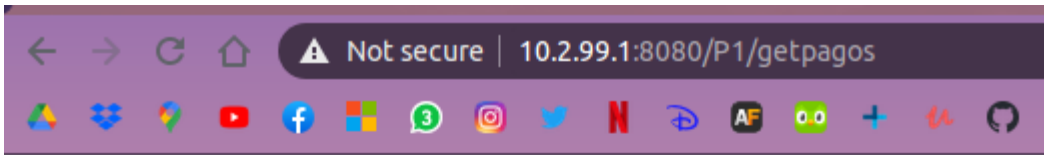
Id Transacción: 1
Id Comercion: 2
Importe: 100.0

Además, en la base de datos vemos que se ha añadido el pago luego se ha ejecutado correctamente:

| | idautorizacion | idtransaccion | codrespuesta | importe | idcomercio | numerotarjeta | fecha |
|---|----------------|---------------|--------------|---------|------------|---------------------|-----------------|
| 1 | 1 | 1 | 000 | 100 | 2 | 1111 2222 3333 4444 | 2/22/21 9:51 AM |
| 2 | 2 | 2 | 000 | 10 | 2 | 4055 0999 2100 8562 | 2/22/21 9:53 AM |
| 3 | 3 | 3 | 000 | 1232 | 2 | 7279 6014 6893 2592 | 2/22/21 9:54 AM |

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.

Primero listamos los pagos del comercio 2 por ejemplo:



Pago con tarjeta

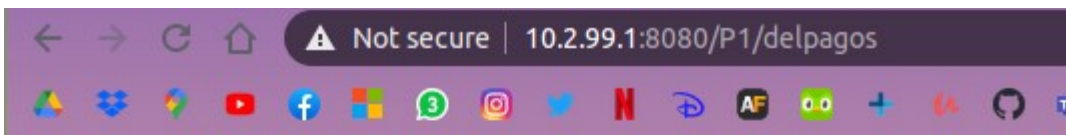
Lista de pagos del comercio 2

| idTransaccion | Importe | codRespuesta | idAutorizacion |
|---------------|---------|--------------|----------------|
| 1 | 100.0 | 000 | 1 |
| 2 | 10.0 | 000 | 2 |
| 3 | 1232.0 | 000 | 3 |

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

A continuación eliminamos esos pagos y los volvemos a listar para ver que se ha ejecutado correctamente:

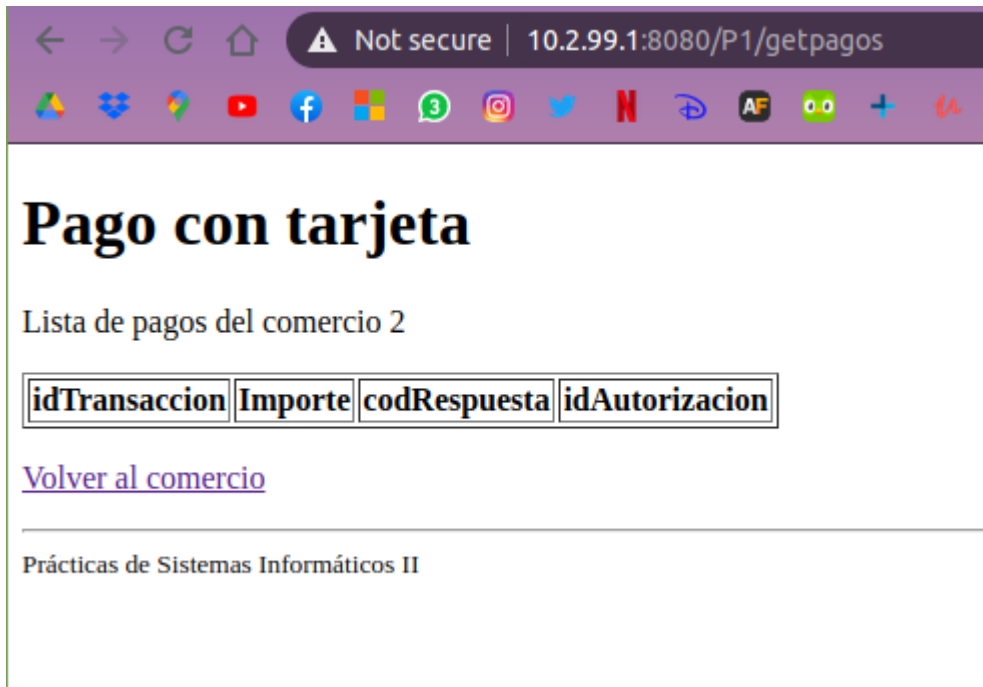


Pago con tarjeta

Se han borrado 3 pagos correctamente para el comercio 2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II



Ejercicio número 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. 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, incluyendo capturas de pantalla

Primero, comprobamos que ejecutando la prueba con conexión directa sin cambiar el código, esta no funciona. Efectivamente, salta el mensaje de error de tarjeta no autorizada. Seguidamente, modificamos el siguiente fragmento de código :

```
/*private static final String JDBC_CONNSTRING =
    "jdbc:derby://10.1.1.1:1527/visa;create=true";
    /***/
/* private static final String JDBC_USER = "APP";
private static final String JDBC_PASSWORD = "APP";*/

private static final String JDBC_CONNSTRING =
    "jdbc:postgresql://10.2.99.1:5432/visa";
    /***/
private static final String JDBC_USER = "alumnodb";
private static final String JDBC_PASSWORD = "****";
```

La parte comentada es lo que había antes, lo hemos sustituido por los datos de acceso a la base de datos de la máquina virtual con IP 10.2.99.1. Una vez realizado este cambio, se ejecuta la prueba con conexión directa, y esta funciona.

Pago con tarjeta

Proceso de un pago

| | |
|--------------------------------------|---|
| Id Transacción: | <input type="text" value="1"/> |
| Id Comercio: | <input type="text" value="1"/> |
| Importe: | <input type="text" value="1"/> |
| Numero de visa: | <input type="text" value="1111 2222 3333 4444"/> |
| Titular: | <input type="text" value="Jose Garcia"/> |
| Fecha Emisión: | <input type="text" value="11/09"/> |
| Fecha Caducidad: | <input type="text" value="11/22"/> |
| CVV2: | <input type="text" value="123"/> |
| Modo debug: | <input type="radio"/> True <input type="radio"/> False |
| Direct Connection: | <input checked="" type="radio"/> True <input type="radio"/> False |
| Use Prepared: | <input type="radio"/> True <input type="radio"/> False |
| <input type="button" value="Pagar"/> | |

Pago con tarjeta

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

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

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

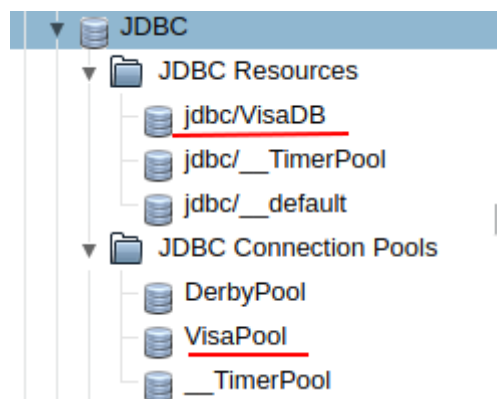
Ejercicio número 3:

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.

Obtenemos los nombres de `postgresql.properties`:

```
db.pool.name=VisaPool  
db.jdbc.resource.name=jdbc/VisaDB
```

Desde la consola, comprobamos que se han creado correctamente:



También desde la consola, obtenemos los datos pedidos

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

La ventaja del uso del pool de conexiones es que se optimiza el tiempo de acceso, ya que no hay que estar abriendo y cerrando constantemente conexiones. Por ello ,es útil tener un número razonable de pools al inicio (Initial and Minimum Pool Size). Cuando se llega al número máximo de conexiones, se aumenta el número de conexiones, y si no se puede porque se ha llegado al máximo(Maximum Pool Size), el hilo que pide la conexión se pone en espera hasta que se libere alguna. Al revés, si las conexiones llevan un cierto tiempo sin ser usadas(dle Timeout), pueden ser cerradas.

Estos valores tienen que alcanzar un equilibrio donde se optimice el acceso al tener muchos pools, pero sin usar tantos como para que el acceso sea lento. Las sinergias entre estos valores son las que generan este equilibrio.

Ejercicio número 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.

```
private static final String SELECT_TARJETA_QRY =
    "select * from tarjeta " +
    "where numeroTarjeta=? " +
    " and titular=? " +
    " and validaDesde=? " +
    " and validaHasta=? " +
    " and codigoVerificacion=? ";
```

- Ejecución del pago.

```
private static final String INSERT_PAGOS_QRY =
    "insert into pago(" +
    "idTransaccion,importe,idComercio,numeroTarjeta)" +
    " values (?,?,?,?)";
```

Ejercicio 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 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.

Podemos observar que el método errorLog se invoca en dos escenarios distintos: cuando ocurre una excepción imprimiendo el error que hay ocurrido; y también antes de ejecutar las consultas y de asociar sus parámetros.

A continuación realizamos un pago y entramos en el administrador para ver el fichero log:

Log Viewer

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

Search Close

[Advanced Search](#)

Search Criteria

Text search:

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

Timestamp: ☒ Most Recent
☐ Specific Range:

Log Level: ☐ Do not include more severe messages

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

Search Close

[Modify Search](#)

Instance:

Log File:

Log Viewer Results (40)

Records before 283 Log File Record Numbers 283 through 322

Records after 322



| Record Number | Log Level | Message | Logger | Timestamp | Name-Value Pairs |
|---------------|-----------|---|--|--------------------------|---|
| 322 | INFO | WebModule[null] ServletContext.log():[INFO] Acceso correcto:/testbd.jsp(details) | javax.enterprise.web | 01-mar-2021 02:26:40.866 | {levelValue=800, timeMillis=1614594400866} |
| 321 | SEVERE | [directConnection=false] select idAutorizacion, codRespuesta from pago where idTransaccion = '1' ... (details) | | 01-mar-2021 02:26:36.304 | {levelValue=1000, timeMillis=1614594396304} |
| 320 | SEVERE | [directConnection=false] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1... (details) | | 01-mar-2021 02:26:36.295 | {levelValue=1000, timeMillis=1614594396295} |
| 319 | SEVERE | [directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular... (details) | | 01-mar-2021 02:26:36.241 | {levelValue=1000, timeMillis=1614594396241} |
| 318 | INFO | visiting unvisited references(details) | javax.enterprise.system.tools.deployment.dol | 01-mar-2021 02:26:34.366 | {levelValue=800, timeMillis=1614594394366} |
| 317 | INFO | WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago(details) | javax.enterprise.web | 01-mar-2021 02:26:33.831 | {levelValue=800, timeMillis=1614594393831} |

Si por ejemplo entramos en una de las entradas relacionadas con el pago que acabamos de realizar, en concreto la cuarta entrada por arriba en la imagen anterior, podemos observar el mensaje log con los detalles del pago:

Log Entry Detail

Timestamp 01-mar-2021 02:26:36.241

Log Level SEVERE

Logger

Name-Value Pairs {levelValue=1000, timeMillis=1614594396241}

Record Number 319

Message ID

Complete Message [directConnection=false] select * from tarjeta where numeroTarjeta='1111 2222 3333 4444' and titular='Jose Garcia' and validaDesde='11/09'

Ejercicio número 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()`
- `isPrepared()` / `setPrepared()`

De 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). En ningún caso se debe añadir ni modificar nada de la clase DBTester.

En primer lugar hemos incluido los respectivos `@WebService()`, `@WebMethod(operationName = "nombreMetodo")` y `@WebParam(name = "nombreArgumento")` en los métodos indicados en el enunciado.

Para la clase solamente hemos añadido @WebService encima de la clase DAOWS.

```
@WebService()  
public class VisaDAOWS extends DBTester {  
  
    private boolean debug = false;
```

Se incluyen capturas de pantalla de algunos de los métodos y parámetros:

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

Además se incluye la captura de los métodos heredados de la clase DBTester:

```
@Override  
@WebMethod(operationName = "setDirectConnection")  
public void setDirectConnection(@WebParam(name = "directConnection") boolean directConnection) {  
    super.setDirectConnection(directConnection);  
}
```

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.

Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones requeridas.

Hemos modificado las líneas 214 para que el método sea de tipo PagoBean; 228 para que devuelva null en caso de error y las últimas líneas para que devuelva el pago o null.

```
@WebMethod(operationName = "realizaPago")  
public synchronized PagoBean realizaPago(@WebParam(name = "pago") PagoBean pago) {  
    Connection con = null;  
    Statement stmt = null;  
    ResultSet rs = null;  
    boolean ret = false;  
    String codRespuesta = "999"; // En principio, denegado  
  
    // TODO: Utilizar en funcion de isPrepared()  
    PreparedStatement pstmt = null;  
  
    // Calcular pago.  
    // Comprobar id.transaccion - si no existe,  
    // es que la tarjeta no fue comprobada  
    if (pago.getIdTransaccion() == null) {  
        return null;  
    }  
}
```

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

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?

Porque ahora el cliente es independiente del servidor y del servicio web, de modo que es mejor devolver el pago para que el cliente pueda comprobarlo con la respuesta y el identificador. Con un boolean solo recibiría si el pago fue correcto o no, sin recibir ningún tipo de dato más.

Ejercicio número 7:

Despliegue el servicio con la regla correspondiente en el build.xml.

Desplegamos el servicio con los comandos “ant empaquetar-servicio” y “ant desplegar-servicio”

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).

← → ↻ 🏠 No es seguro | 10.2.99.1:4848/common/index.jsf

Home About...

User: admin Role: domain1 Server: 10.2.99.1

GlassFish™ Server Open Source Edition

Total # of available updates : 1

Tree

- Common Tasks
 - Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
 - Nodes
 - Applications
 - P1
 - P1-ws-ws
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
 - Configurations
 - default-config
 - server-config
 - Update Tool

Web Service Endpoint Information

View details about a web service endpoint.

| | |
|----------------------------|-----------------------------------|
| Application Name: | P1-ws-ws |
| Tester: | /P1-ws-ws/VisaDAOWSService?Tester |
| WSDL: | /P1-ws-ws/VisaDAOWSService?wsdl |
| Endpoint Name: | VisaDAOWS |
| Service Name: | VisaDAOWSService |
| Port Name: | VisaDAOWSPort |
| Deployment Type: | 109 |
| Implementation Type: | SERVLET |
| Implementation Class Name: | ssii2.visa.dao.VisaDAOWS |
| Endpoint Address URI: | /P1-ws-ws/VisaDAOWSService |
| Namespace: | http://dao.visa.ssii2/ |

← → ↻ 🏠 No es seguro | 10.2.99.1:8080/P1-ws-ws/VisaDAOWSService?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-
-->
<!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.2-b608 (trunk-
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-util
xmlns:wspl_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/
xmlns:tns="http://dao.visa.ssii2/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://s
">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://dao.visa.ssii2/" schemaLocation="http://10.2.99.1:8080/P1-ws
      </xsd:schema>
    </types>
    <message name="setDebug">
      <part name="parameters" element="tns:setDebug"/>
    </message>
    <message name="setDebugResponse">
      <part name="parameters" element="tns:setDebugResponse"/>
    </message>
    <message name="isDebug">
      <part name="parameters" element="tns:isDebug"/>
    </message>
    <message name="isDebugResponse">
      <part name="parameters" element="tns:isDebugResponse"/>
    </message>
    <message name="compruebaTarjeta">
      <part name="parameters" element="tns:compruebaTarjeta"/>
    </message>
    <message name="compruebaTarjetaResponse">
      <part name="parameters" element="tns:compruebaTarjetaResponse"/>
    </message>
  </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.

Podemos observar que, a parte de las cabeceras indicando las definiciones, el documento WSDL indica con las definiciones de cada método de la clase así como sus inputs (argumentos) y outputs (return), es decir, las operaciones del servidor, sus respectivos mensajes y el puerto y la dirección de cada servicio.

Conteste a las siguientes preguntas:

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

Se encuentra en el link “<http://10.2.99.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1>” definido por la etiqueta wsdl en el archivo WSDL

- ¿Qué tipos de datos predefinidos se usan?

String, boolean, int y double.

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

Son compruebaTarjeta, compruebaTarjetaResponse, delPagos, delPagosResponse, errorLog, errorLogResponse, getPagos, getPagosResponse, isDebug, isDebugResponse, isDirectConnection, isDirectConnectionResponse, isPrepared, isPreparedResponse, realizaPago, realizaPagoResponse, setDebug, setDebugResponse, setDirectConnection, setDirectConnectionResponse, setPrepared, setPreparedResponse.

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

Es la etiqueta portType que define una etiqueta operation por cada método.

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

La etiqueta message que describe los mensajes de entrada y de salida.

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

En la etiqueta binding que se especifica que es SOAP.

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

En la etiqueta service.

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

En las siguientes imágenes se pueden observar los cambios. Primero, hemos añadido las líneas dadas por el enunciado para la llamada al servicio web:

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

Después hemos introducido todo el código desde estas líneas en un try...catch que gestiona las posibles excepciones. Como realizaPago ahora devuelve el pago modificado o null, lo sobrescribimos con lo que devuelva la función , y comprobamos en un if si es null (antes el if comprobaba si esto era false):

```

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

    request.setAttribute(ComienzaPago.ATTR_PAGO, pago);
    if (sesion != null) sesion.invalidate();
    reenvia("/pagoexito.jsp", request, response);

```

Ejercicio número 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.

Modificamos el fichero XML para que incluya la url:

```

<context-param>
    <param-name>service-url</param-name>
    <param-value>http://10.2.99.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>

```

Y por otro lado, modificamos ProcesaPago.java para cogerla:

```

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

String url = getServletContext().getInitParameter("nombre-parámetro");
BindingProvider bp = (BindingProvider) dao;
bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,url);

```

Ejercicio número 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(...)

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**.

Incluye en la memoria una captura con las adaptaciones realizadas.

Hemos modificado el método `processRequest` tanto de `GetPagos.java` como de `DelPagos.java`, adaptándolos a `VisaDAOWS`. Básicamente, se ha cambiado en ambos la llamada al servicio web (cambio igual que en la captura del ejercicio anterior). Además, en `GetPagos` hay que tener en cuenta que la lista de pagos es ahora un `ArrayList`. En la siguiente captura se muestra como se transforma a un array de `PagoBean`:

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

Las otras modificaciones son en `VisaDAOWS`, donde `getPagos` y `delPagos` se declaran como web methods.

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

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

Además en `getPagos`, devolvemos directamente el `ArrayList<PagoBeans>` que se genera con los pagos, en vez de transformarlo en `PagoBeans[]`

Ejercicio 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. Téngase en cuenta que el servicio debe estar previamente desplegado.

El comando es:

`wsimport -d build/client/WEB-INF/classes -p ssii2.visa http://10.2.99.1:8080/P1-ws-ws/VisaDAOWSService?wsdl`

Nos genera las clases:

```
CompruebaTarjeta.class
CompruebaTarjetaResponse.class
DelPagos.class
DelPagosResponse.class
ErrorLog.class
ErrorLogResponse.class
GetPagos.class
GetPagosResponse.class
IsDebug.class
IsDebugResponse.class
IsDirectConnection.class
IsDirectConnectionResponse.class
IsPrepared.class
IsPreparedResponse.class
```

```
ObjectFactory.class
package-info.class
PagoBean.class
RealizaPago.class
RealizaPagoResponse.class
SetDebug.class
SetDebugResponse.class
SetDirectConnection.class
SetDirectConnectionResponse.class
SetPrepared.class
SetPreparedResponse.class
TarjetaBean.class
VisaDAOWS.class
VisaDAOWSService.class
```


Estas clases se corresponden con los elementos etiquetados como *message* en el fichero WSDL. Son necesarias para implementar los métodos del web service, en particular, las clases necesarias para implementar la entrada y salida (parámetros y retornos). Cada par corresponde a una función.

Ejercicio número 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}`

El cambio se ve en la siguiente captura:

```
<target name="generar-stubs" depends="montar-jerarquia" description="Genera los stubs del cliente a partir del archivo WSDL">
  <exec executable="${wsimport}">
    <arg line="-d ${build.client}/WEB-INF/classes" />
    <arg line="-p ${paquete}.visa" />
    <arg line="-s ${wsdl.url}" />
  </exec>
  <delete file="${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>
```

Ejercicio número 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. Incluye evidencias en la memoria de la realización del ejercicio.

Desplegamos todo y realizamos un pago desde `testdb.jsp`:

Pago con tarjeta

Proceso de un pago

| | |
|--------------------------------------|--|
| Id Transacción: | <input type="text" value="1"/> |
| Id Comercio: | <input type="text" value="1"/> |
| Importe: | <input type="text" value="12"/> |
| Numero de visa: | <input type="text" value="1111 2222 3333 4444"/> |
| Titular: | <input type="text" value="Jose Garcia"/> |
| Fecha Emisión: | <input type="text" value="11/09"/> |
| Fecha Caducidad: | <input type="text" value="11/22"/> |
| CVV2: | <input type="text" value="123"/> |
| Modo debug: | <input type="radio"/> True <input type="radio"/> False |
| Direct Connection: | <input type="radio"/> True <input type="radio"/> False |
| Use Prepared: | <input type="radio"/> True <input type="radio"/> False |
| <input type="button" value="Pagar"/> | |

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 12.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Desde Tora accedemos a la base de datos, y realizamos la siguiente consulta:

*SELECT * FROM pago WHERE idautorizacion=1*

Obteniendo:

| # | ^ | idautorizacion | idtransaccion | codrespuesta | importe | idcomercio | numerotarjeta | fecha |
|---|---|----------------|---------------|--------------|---------|------------|---------------------|----------------|
| 1 | 1 | 1 | 1 | 000 | 12 | 1 | 1111 2222 3333 4444 | 04/03/21 07:44 |

Pregunta 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.

Pues en primer lugar pasaríamos por el archivo pago.html para poder efectuar algún pago. A continuación se llamaría al servlet ComienzaPago y luego al formdatosvisa.jsp. Por último se llamaría al servlet ProcesaPago, el cual detectaría el error de la tarjeta expirada, y luego se llamaría al formdatosvisa.jsp.

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

Pregunta 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, y cuáles son los encargados de procesarla?

El servlet encargado de solicitar y procesar la información es ProcesaPago.

Pregunta 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?

En primer lugar, ComienzaPago se encarga de solicitar los datos del Id de transacción, el Id del comercio y el importe. Por otro lado, ProcesaPago solicita los datos de la tarjeta, como el titular, el número de tarjeta, las fechas de caducidad y de primer uso y el código ccv2. Ambos servlets comparten dicha información a través de la variable de sesión sesion.getAttribute(ComienzaPago.ATTR_PAGO).

Pregunta 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 está en que, por un lado, en pago.html primero se pide la información del comercio, transacción e importe, llamando a ComienzaPago, y después se pide la información de la tarjeta, llamando a ProcesaPago; mientras que en testbd.jsp se pide todo a la vez, llamando directamente a ProcesaPago. Sin embargo, ProcesaPago sigue recibiendo la misma información, luego ambos formatos son válidos.