UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2**			
Grupo	**2401**	**Práctica**	1A	**Fecha**	27/02/2018
Alumno/a		Marcos Manchón, Pablo			
Alumno/a		Nevado Catalán, David			

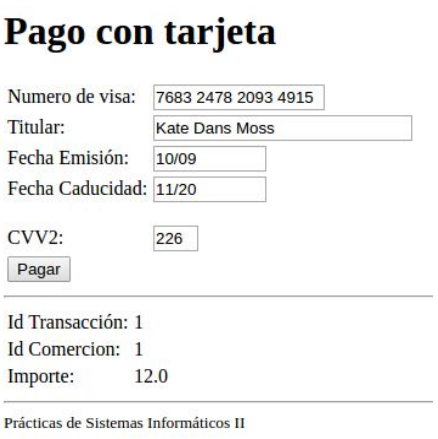
Ejercicio 1

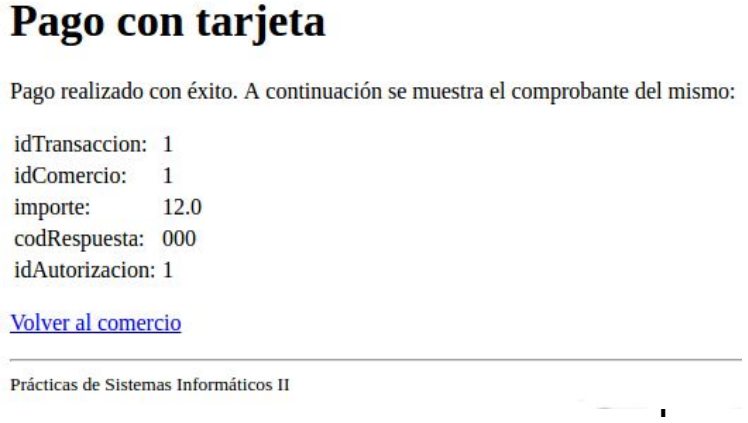
En primer lugar para iniciar la máquina virtual configuramos la interfaz de red ejecutando el script **si2fixMAC.sh** y utilizando VMware player ejecutamos la máquina virtual y configuramos su IP, en nuestro caso 10.1.9.2.
Posteriormente configuramos la interfaz de red eth0 ejecutando el script situado en **/opt/si2/virtualip.sh** y exportamos la variable J2EE_HOME para el posterior despliegue de la aplicación. Mediante ssh iniciamos glassfish en la máquina virtual con el comando **asadmin start-domain domain1**.

Modificamos en los archivos **build.properties** y **postgres.properties** de P1-base, para poder desplegar el servicio, las variables **as.host**, **db.host** y **db.client.host**. Las asignamos a la dirección IP de nuestra aplicación y nuestra base de datos:

build.properties	**postgres.properties**
as.host =10.1.9.2	db.host = 10.1.9.2
	db.client.host = 10.1.9.2

Tras realizar estas modificaciones, utilizamos la herramienta ant para **compilar**, **empaquetar** y **desplegar** la aplicación en nuestro servidor. También utilizamos ant **regenerar-bd** para generar la base de datos que usa la aplicación. Capturas del pago realizado:





Nos conectamos a la base de datos utilizando psql para ver si la transacción se ha efectuado correctamente, para ello ejecutamos
psql -U alumnodb -h 10.1.9.2 -p 5432 -b visa

Y efectuamos una consulta sql para ver los pagos registrados en la base de datos

```
 idautorizacion |  idtransaccion   | codrespuesta | importe |    idcomercio    |    numerotarjeta    |           fecha
----------------+------------------+--------------+---------+------------------+---------------------+----------------------------
              1 | 1                | 000          |      12 | 1                | 7683 2478 2093 4915 | 2018-02-20 07:09:06.618923
(1 row)
```

Desde la página **testbd.jsp** procedemos a listar los datos de la aplicación y posteriormente a borrar el pago realizado.

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
1	12.0	000	1

Volver al comercio

Prácticas de Sistemas Informáticos II

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

Volver al comercio

Prácticas de Sistemas Informáticos II

Ejercicio 2

En el fichero **DBTester.java** cambiamos las variables necesarias para que la aplicación se conecte a nuestra base de datos, que en nuestro caso está en la dirección 10.1.9.2:

```
private static final String JDBC_DRIVER = "org.postgresql.Driver";
private static final String JDBC_CONNSTRING = "jdbc:postgresql://10.1.9.2:5432/visa";
private static final String JDBC_USER = "alumnodb";
private static final String JDBC_PASSWORD = "****";
```

Tras volver a compilar y desplegar el servicio con la configuración correcta realizamos un pago utilizando conexión directa desde la página **testbd.jsp**. Vemos que el nuevo pago aparece en la lista y se elimina correctamente.

Pago con tarjeta

Lista de pagos del comercio 23

idTransaccion	Importe	codRespuesta	idAutorizacion
1112	14.0	000	1

Volver al comercio

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 23

Volver al comercio

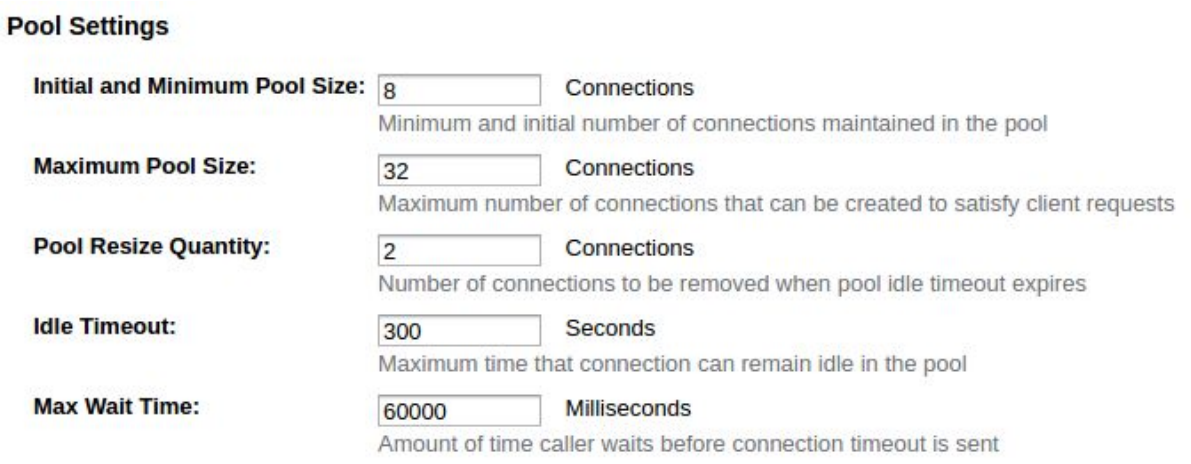
Ejercicio 3

En el fichero **postgres.properties** vemos que se encuentran definidas las variables db.pool.name y db.datasource con los nombres correspondientes de los recursos.

```
db.pool.name=VisaPool
db.datasource=org.postgresql.ds.PGConnectionPoolDataSource
```

Desde la consola de administración realizamos un PING JDBC al recurso visa pool.

Desde la consola de administración podemos acceder a la configuración del pool de conexiones.



Veamos cómo afecta al servidor la variación de los parámetros del pool:

- **Initial and Minimum Pool Size**
  El servidor crea conexiones a la base de datos reutilizables, gracias al cual al solicitar una nuevo conexión un cliente no tiene que esperar a la creación de ésta si hay conexiones disponibles en el pool. Esto supone una mejora de rendimiento ya que se evita crear una nueva conexión (algo relativamente costoso) con cada petición.

  Un tamaño inicial y mínimo no muy elevado de conexiones propicia un arranque más rápido del servidor, además en caso de no haber demanda de conexiones permite que no se desperdicien recursos del sistema. En general en una aplicación al crecer de forma dinámica en base a las peticiones no se verá afectada por un tamaño pequeño de conexiones mínimo, a excepción de picos de demanda grande.

- **Maximum Pool size**
  Un número máximo de conexiones elevado mejorará la eficiencia del servidor, siempre y cuando los recursos del sistema sean suficientes para soportar la apertura en paralelo de todas las conexiones y cuando las consultas en la base de datos sean lo suficientemente complejas para que el coste de acceso a la tabla de conexiones, que se incrementa al aumentar su número, sea despreciable con el coste de la consulta.

- **Pool Resize Quantity.**
  Este parámetro permite regular la prioridad entre velocidad de adaptación y precisión. Una cantidad elevada permitirá al sistema adaptarse rápidamente a la demanda necesaria y también liberar recursos en cuanto esta demanda haya pasado. Sin embargo, un parámetro más bajo permite utilizar solo los recursos necesarios (lo que hemos llamado precisión) aunque con un tiempo de adaptación algo mayor.

- **Idle Timeout**
  Un idle timeout elevado favorece el mantenimiento de un mayor número de conexiones abiertas, pero tiene el lado negativo de que pueden mantenerse abiertos recursos innecesarios.

- **Max wait Time.**
  Cuanto menor sea el tiempo máximo de espera mayor será el aprovechamiento de recursos del sistema. Por otra parte, un tiempo demasiado pequeño podría causar timeouts evitables elevando así el número fallos de la aplicación.

Ejercicio 4.

El código SQL se encuentra en **VisaDao.java** (P1-Base/src/sii2/visa/dao/VisaDao.java), tanto para la realización de sentencias preparadas como sin preparar.

Consultas preparadas

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

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

Consultas sin preparar

```
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;
 }
```

Ejercicio 5

La función **errorLog** se encuentra en P1-base/src/sii2/controlador/ServletRaiz.java

```
protected void errorLog(String error) {
    getServletContext().log("[ERROR] : " + error);
}
```

Realizamos un pago con el modo debug activado, y vemos cómo se generan logs correspondientes a la transacción, en este caso efectuada correctamente.

Log Viewer Results (40)

Records before 1329 | Log File Record Numbers 1329 through 1368 | Records after 1368

Record Number	Log Level	Message	Logger	Timestamp	Name-Value Pairs
1368	SEVERE	[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = '1' ... (details)		20-feb-2018 08:37:25.352	{levelValue=1000, timeMillis=1519144645352}
1367	SEVERE	[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1'... (details)		20-feb-2018 08:37:25.348	{levelValue=1000, timeMillis=1519144645348}
1366	SEVERE	[directConnection=true] select * from tarjeta where numeroTarjeta='7683 2478 2093 4915' and titular=... (details)		20-feb-2018 08:37:25.342	{levelValue=1000, timeMillis=1519144645342}
1365	INFO	WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago(details)	javax.enterprise.web	20-feb-2018 08:37:25.328	{levelValue=800, timeMillis=1519144645328}

1368

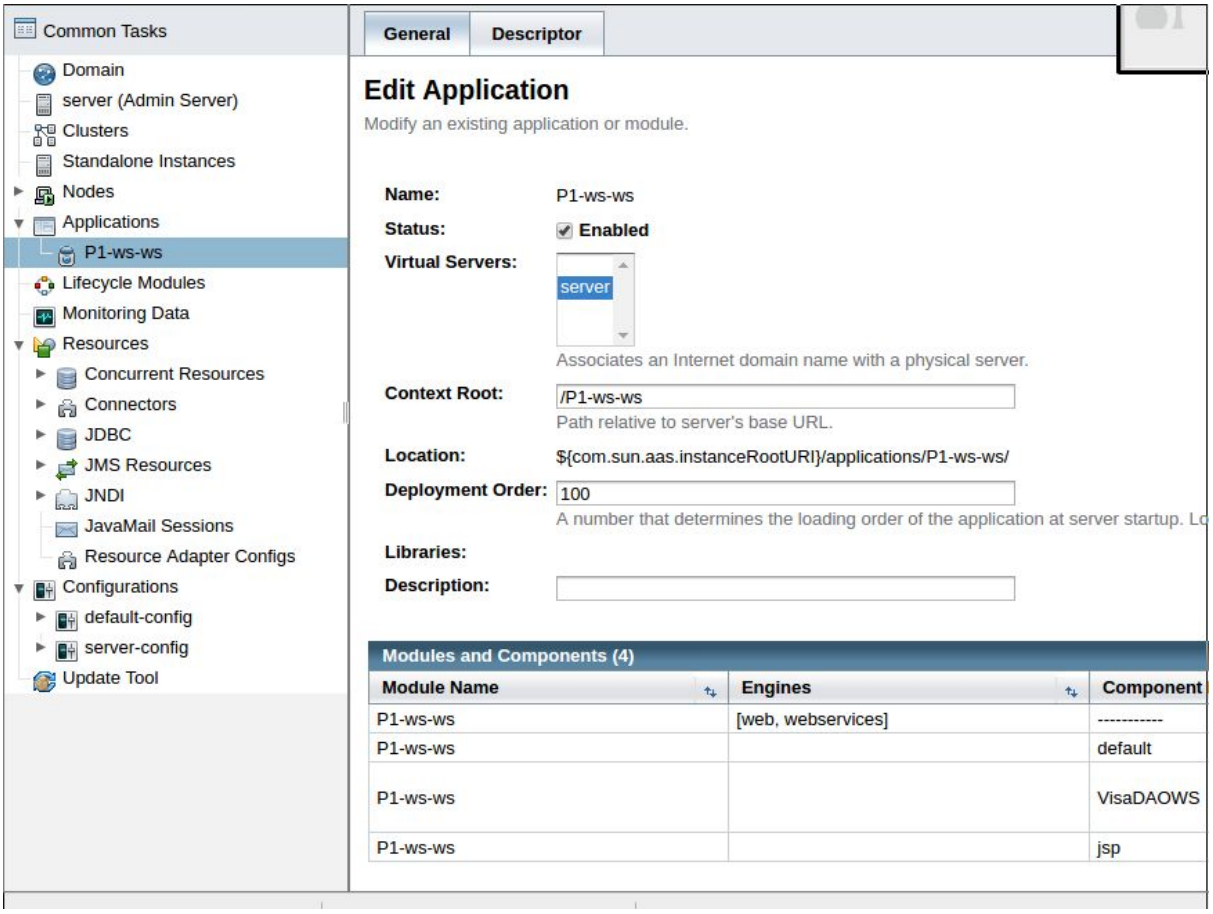
Message
[directConnection=true] select idAutorizacion, codRespuesta from pago where idTransaccion = '1' ... (details)
[directConnection=true] insert into pago(idTransaccion,importe,idComercio,numeroTarjeta) values ('1'... (details)
[directConnection=true] select * from tarjeta where numeroTarjeta='7683 2478 2093 4915' and titular=... (details)
WebModule[null] ServletContext.log():[INFO] Acceso correcto:/procesapago(details)

Ejercicio 6

Los cambios se encuentran en el archivo **VisaDAOWS.java** y **DBTester.java**.
Para publicar el servicio web se ha utilizado la anotación **@WebService** para la clase **VisaDAOWS** y las anotaciones **@WebMethod** y **@WebParam** para los métodos correspondientes y sus respectivos parámetros.
Además hemos exportado como servicio web solo la versión de SetDebug que recibía un bool como argumento.

Se ha tenido que modificar el método **realizaPago** porque ahora el cliente es independiente del servicio web y por tanto desde el cliente no puede accederse a los datos, por eso es necesario que **realizaPago** devuelva los datos del pago para poder comprobar desde el lado del cliente el código de respuesta y el identificador de la autorización.

Tras estas modificaciones podemos desplegar el servicio web, el cual se registra como P1-ws-ws.



Ejercicio 7

El fichero **WDSL** se encuentra adjunto en **VisaDAOWSService-wdsl.xml**.

1. **¿En qué fichero están definidos los tipos de datos intercambiados con el webservice?**
   En el fichero XSD que se especifica en el WSDL:

```
<xsd:import namespace="http://visa.ssii2.server/" schemaLocation="http://10.1.9.1:8080/P1-ws-ws/VisaDAOWSService?xsd=1"/>
```

2. **¿Qué tipos de datos predefinidos se usan?**
   xs:boolean, xs:string y xs:double

3. **¿Cuáles son los tipos de datos que se definen?**
   Se definen los métodos del sistema que se pueden llamar así como sus respuestas.

4. **¿Qué etiqueta está asociada a los métodos invocados en el webservice?**
   Se utiliza la etiqueta <operation>

5. **¿Qué etiqueta describe los mensajes intercambiados en la invocación de los métodos del webservice?**
   Se utiliza la etiqueta <message>

6. **¿En qué etiqueta se especifica el protocolo de comunicación con el webservice?**
   En <binding>. Se especifica el protocolo HTTP

```
-<binding name="VisaDAOWSPortBinding" type="tns:VisaDAOWS">
   <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  -<operation name="compruebaTarjeta">
```

7. **¿En qué etiqueta se especifica la URL a la que se deberá conectar un cliente para acceder al webservice?**
   Dentro de la etiqueta <service> en <soap:address>

```
-<service name="VisaDAOWSService">
  -<port name="VisaDAOWSPort" binding="tns:VisaDAOWSPortBinding">
     <soap:address location="http://10.1.9.1:8080/P1-ws-ws/VisaDAOWSService"/>
```

Ejercicio 8:
En el método **realizaPago()** se cambió el tipo de retorno a PagoBean. Simplemente se realizaron modificaciones para que se devolviera null cuando antes devolvía false, y que devolviera el objeto PagoBean que contiene la información sobre el pago realizado cuando antes devolvía true.

Rodeamos las llamadas remotas en un bloque try-catch en el código del cliente ya que ahora se podrían producir errores al tratar de conectarse con el servicio.

```
155     // Realiza una instanciacion del servicio VisaDaoWS
156     VisaDAOWSService service = new VisaDAOWSService();
157     // Obtenemos el dao a partir de la llamada al servicio
158     VisaDAOWS dao = service.getVisaDAOWSPort();
159     String direccion;
160     try{
161         // Obtenemos la direccion del xml
162         direccion =getServletContext().getInitParameter("service-url");
163         BindingProvider bp = (BindingProvider) dao;
164         bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, direccion);
165     } catch (Exception e){
166         enviaError(new Exception("Error. Server unreacheable"), request, response);
167         return;
168     }
169
```

Ejercicio 9:
Se editó el fichero web.xml para añadir al contexto de los servlets el parámetro *service-url*. En este parámetro se almacena la url del servicio.

```
<context-param>
    <param-name>webmaster</param-name>
    <param-value>david.nevadoc@estudiante.uam.es</param-value>
    <param-name>service-url</param-name>
    <param-value>http://10.1.9.1:8080/P1-ws-ws/VisaDAOWSService</param-value>
</context-param>
```

Los comentarios del fichero nos indican que llamada se debe utilizar para acceder a los parámetros del contexto: **getServletContext().getInitParameter("nombre");**

Ejercicio 10:
Para que la funcionalidad de testbd.jsp se implemente por el servicio web, se modificaron los métodos **processRequest** de los siguientes ficheros.

- ProcesaPago.java
- DelPagos.java
- GetPagos.java

Ejercicio 11:
Para realizar la importación de las clases se utilizó el comando

```
wsimport -d build/client/WEB-INF/classes/
http://10.1.9.1:8080/P1-ws-ws/VisaDAOWSService?wsdl
```

Se observa que tras la ejecución de este comando se generan dos clases (fichero .class) por cada clase método. Un fichero contiene la información de entrada del método y el otro con la salida:
(p.e. `CompruebaTarjeta.class, CompruebaTarjetaResponse.class`)

Ejercicio 12:
Para conseguir que **`ant generar-stubs`** genere las clases necesarias para la compilación del cliente modificamos el fichero **build.xml**

```
85
86      <target name="generar-stubs" depends="montar-jerarquia"
87          description="Genera los stubs del cliente a partir del archivo WSDL">
88      <!-- TODO - Implementar llamada wsimport -->
89
90      <delete file="${build}/${tmpvisaclientjar}" />
91      <jar jarfile="${build}/${tmpvisaclientjar}" >
92      <fileset dir="${build.client}/WEB-INF/classes" />
93      </jar>
94      <move file="${build}/${tmpvisaclientjar}" todir="${build.client}/WEB-INF/lib" />
95      <exec executable="${wsimport}">
96          <arg line=" -d ${build.client}/WEB-INF/classes"/>
97          <arg line="${wsdl.url}"/>
98      </exec>
99      </target>
100
```

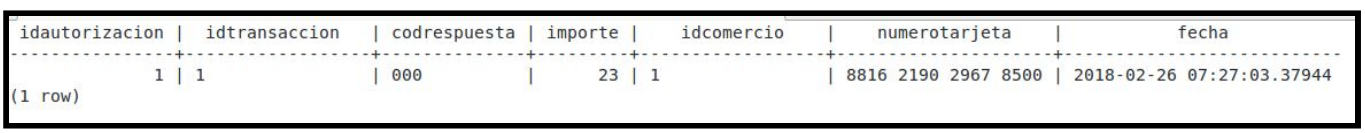
La modificación consiste en hacer que se ejecute el comando visto en el ejercicio anterior mediante la etiqueta **exec** de ant.

Ejercicio 13:
Para realizar este ejercicio desplegamos el servidor y la base de datos en una máquina virtual y el cliente en otra diferente. El servicio y la BD tienen dirección: `10.1.9.1` , el servicio tiene direccion `10.1.9.2`.
Los ficheros build.properties y postgresql.properties se modificaron para reflejar estos cambios.

Realización de un pago:

La operación termina con éxito.

Se comprueba desde el cliente y desde la base de datos que la operación es persistente en la base de datos.



```
 idautorizacion |  idtransaccion   | codrespuesta | importe |    idcomercio    |    numerotarjeta    |           fecha
----------------+------------------+--------------+---------+------------------+---------------------+----------------------------
              1 | 1                | 000          |      23 | 1                | 8816 2190 2967 8500 | 2018-02-26 07:27:03.37944
(1 row)
```

Cuestiones:

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

Primero se accede al formulario de **pago.html** , una vez completo se llama al servlet **ComienzaPago**, se pasa después a **formdatosvisa.jsp.** Luego se pasa al servlet **ProcesaPago** donde se detecta el error, por lo que se pasa a **formdatosvisa.jsp.**

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

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

La información de la tarjeta se solicita en ProcesaPago.

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

ComienzaPago solicita los primeros datos (Id Transacción , Id Comercio e Importe) y ProcesaPago solicita el resto de datos (Número de tarjeta, Titular, etc.)

Los servlets comparten la información mediante una variable de sesión http. En processRequest:

```
pago = (PagoBean) sesion.getAttribute(ComienzaPago.ATTR_PAGO);
```

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 única diferencia es que pago.html solicita la información en dos pasos, invocando primero a ComienzaPago, que recibe información sobre el pago; y después a ProcesaPago que recibe la información sobre la tarjeta por el formulario web y la información del pago por la variable de sesión. Testbd.jsp solicita toda la información a la vez y por lo tanto solo se ejectua ProcesaPago.

Ambos métodos son válidos porque ProcesaPago es el encargado de procesar el pago y en ambos casos recibe la información necesaria para ello (aunque por vías distintas)