UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2**			
Grupo	**2321**	**Práctica**	1A	**Fecha**	23/03/2021
Alumno/a		Arribas Gozalo, Francisco Javier			
Alumno/a					

Práctica 1B: Arquitectura de JAVA EE (Segunda Parte)

Ejercicio 1:

Introduzca las siguientes modificaciones en el bean VisaDAOBean para convertirlo en unEJB de sesión stateless con interfaz local:

- **Hacer que la clase implemente la interfaz local y convertirla en un EJB stateless mediante la anotación Stateless**
- **Eliminar el constructor por defecto de la clase.**
- **Ajustar el método getPagos() a la interfaz definida en VisaDAOLocal**
- **Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas.**

En VisaDAOBean, que podemos encontrar en P1-ejb/src/server/ssii2/visa/VisaDAOBean.java, debemos realizar los siguientes cambios:

Importar el paquete Stateless, cambiar la clase VisaDAOBean a stateless, hacer que extienda VisaDAOLocal y eliminar el constructor:

```
25  import javax.ejb.Stateless;
26
27  /**
28   * @author jaime
29   */
30
31  // @WebService()
32  @Stateless (mappedName = "VisaDAOBean")
33  public class VisaDAOBean extends DBTester implements VisaDAOLocal {
```

```
83      /**
84       * Constructor de la clase
85       */
86      // public VisaDAOBean() {
87      //     return;
88      // }
```

Tambien debemos modificar la función getPagos() para que devuelva PagoBean[]

```
332     public PagoBean[] getPagos
```

```
372             ret = new PagoBean[pagos.size()];
373             ret = pagos.toArray(ret);
```

```
396         return ret;
397     }
```

También ha sido importante eliminar todas las etiquetas @WebService y los @WebParam correspondientes

Ejercicio 2:

Modificar el servlet ProcesaPago para que acceda al EJB local. Para ello, modificar el archivo ProcesaPago.java.

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

Eliminar también las referencias a BindingProvider. Importante: Esta operación deberá ser realizada para todos los servlets del proyecto que hagan uso del antiguo VisaDAOWS. Verifique también posibles errores de compilación y ajustes necesarios en el códigoal cambiar la interfaz del antiguo VisaDAOWS (en particular, el método getPagos()).

Aquí es necesario cambiar los archivos DelPagos, GetPagos y ProcesaPago, que se encuentran en P1-ejb/ src/client/ssii2/controlador/

Este proceso es el mismo para todos los archivos.

Como se especifica en el enunciado, debemos eliminar los import correspondientes a VisaDAO, a WebService y a BindingProvider, añadiendo los import a las librerías de EJB y la nueva implementación con VisaDAOLocal:

```
21  // import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente
22  // import ssii2.visa.VisaDAOWS; // Stub generado automáticamente
23  // import javax.xml.ws.WebServiceRef;
24  // import javax.xml.ws.BindingProvider;
25  import javax.ejb.EJB;
26  import ssii2.visa.VisaDAOLocal;
```

Ahora añadiremos la interfaz local a las clases:

```
32  public class DelPagos extends ServletRaiz {
33      @EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)
34      private VisaDAOLocal dao;
```

```
29  public class GetPagos extends ServletRaiz {
30      @EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)
31      private VisaDAOLocal dao;
```

```
56  public class ProcesaPago extends ServletRaiz {
57      @EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)
58      private VisaDAOLocal dao;
```

Comentamos el código con referencias a BindingProvider:

```
59      /*added*/
60      /*
61      VisaDAOWSService service = new VisaDAOWSService();
62      VisaDAOWS dao = service. getVisaDAOWSPort ();
63      BindingProvider bp = (BindingProvider) dao;
64      bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, getServletContext().getInitParameter("urlservidor")
65      */
66      /**/
67          /*VisaDAO dao = new VisaDAO();*/
```

Ejercicio 3:

Preparar los PCs con el esquema descrito y realizar el despliegue de la aplicación:

- **Editar el archivo build.properties para que las propiedades *as.host.client* y *as.host.server* contengan la dirección IP del servidor de aplicaciones. Indica qué valores y porqué son esos valores.**
- **Editar el archivo *postgresql.properties* para la propiedad *db.client.host* y *db.host* contengan las direcciones IP adecuadas para que el servidor de aplicaciones se conecte al postgresql, ambos estando en servidores diferentes. Indica qué valores y porqué son esos valores. Desplegar la aplicación de empresa**

```
ant desplegar
```

En postgresql.properties, usaremos las siguientes IPs:

```
7  db.port=5432
8  db.host=10.7.8.1
```

```
13 db.client.host=10.7.8.2
14 db.client.port=4848
```

Al ser local, estas serán las IPs en build.properties:

```
24 as.host.client=10.7.8.2
25 as.host.server=10.7.8.2
26 as.port=4848
```

Ejercicio 4:

Comprobar el correcto funcionamiento de la aplicación mediante llamadas directas a través de las páginas *pago.html* y *testbd.jsp* (sin *directconnection*). Realice un pago. Lístelo. Elimínelo. Téngase en cuenta que la aplicación se habrá desplegado bajo la ruta /P1-ejb-cliente.

Incluya en la memoria de prácticas todos los pasos necesarios para resolver este ejercicio así como las evidencias obtenidas. Se pueden incluir por ejemplo capturas de pantalla.

(Texto de la respuesta)

Ejercicio 5:

Realizar los cambios indicados en P1-ejb-servidor-remoto y preparar los PCs con el esquema de máquinas virtuales indicado. Compilar, empaquetar y desplegar de nuevo la aplicación P1-ejb como servidor de EJB remotos de forma similar a la realizada en el Ejercicio 3 con la Figura 2 como entorno de despliegue. Esta aplicación tendrá que desplegarse en la máquina virtual del PC2.

Se recomienda replegar la aplicación anterior (EJB local) antes de desplegar ésta.

Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas así como detallando los pasos realizados.

Hemos añadido el archivo VisaDAORemote.java a P1-ejb-servidor-remoto/src/server/ssii2/visa.

Además, hemos hecho que la clase VisaDAOBean implemente VisaDAORemote

```
31  // @WebService()
32  @Stateless (mappedName = "VisaDAOBean")
33  public class VisaDAOBean extends DBTester implements VisaDAOLocal, VisaDAORemote  {
34
```

Por otro lado, es necesario modificar PagoBean y TarjetaBean para hacer las clases serializables:

```
10  import java.io.Serializable;
11
12  /**
13   *
14   * @author jaime
15   */
16  public class PagoBean implements Serializable  {
```

```
 9  import java.io.Serializable;
10
11  public class TarjetaBean implements Serializable  {
12
```

Ejercicio 6:

Realizar los cambios comentados en la aplicación P1-base para convertirla en P1-ejb-cliente-remoto. Compilar, empaquetar y desplegar de nuevo la aplicación en otra máquina virtual distinta a la dela aplicación servidor, es decir, esta aplicación cliente estará desplegada en la MV del PC1 tal y como se muestra en el diagrama de despliegue de la Figura 2. Conectarse a la aplicación cliente y probar a realizar un pago. Comprobar los resultados e incluir en la memoria evidencias de que el pago ha sido realizado de forma correcta.

(Texto de la respuesta)

Ejercicio 7:

Modificar la aplicación VISA para soportar el campo saldo. Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas.

En P1-ejb-transaccional/src/server/ssii2/visa/TarjetaBean.java debemos añadir el atributo saldo:

```
16      private double saldo;
```

```
98      /**
99       * Establece el saldo
100      * @param saldo saldo
101      */
102     public void setSaldo (double saldo) {
103         this.saldo = saldo;
104     }
105
106     /**
107      * Devuelve el saldo
108      * @return saldo saldo
109      */
110     public double getSaldo () {
111         return saldo;
112     }
```

En P1-ejb-transaccional/src/server/ssii2/visa/VisaDAOBean.java añadiremos los dos prepared statements en el código, para poder acceder a los datos de saldo de sql:

```
82      private static final String GET_SALDO_QRY =
83                      "select saldo from tarjeta" +
84                      " where numeroTarjeta=?";
85
86      private static final String SET_SALDO_QRY =
87                      "update tarjeta " +
88                      " set saldo=? " +
89                      " where numeroTarjeta=?";
90      /*****************************************************/
```

Ahora, modificaremos el método realizarPago() con la para que pueda obtener el valor del saldo y actuar acorde:

```
245         // Registrar el pago en la base de datos
246         try {
247
248             // Obtener conexion
249             con = getConnection();
250
251             // Recuperar el saldo de la tarjeta a través del prepared statement declarado anteriormente
252             String aux_saldo = GET_SALDO_QRY;
253             errorLog(aux_saldo);
254             pstmt = con.prepareStatement(aux_saldo);
255             pstmt.setString(1, pago.getTarjeta().getNumero());
256             rs = pstmt.executeQuery();
257
258             if (rs.next()) { // Query correcta
259                 saldo = rs.getDouble("saldo");
260                 importe = pago.getImporte();
261
262                 // Comprobar si el saldo es mayor o igual que el importe de la operación.
263                 // Si no lo es, retornar denegando el pago (idAutorizacion= null y pago retornado=null)
264                 if (saldo < importe)
265                 {
266                     pago.setIdAutorizacion(null);
267                     return null;
268                 } else {
269                     // Si el saldo es suficiente, decrementarlo en el valor del importe del pago y actualizar el registro
270                     // de la tarjeta para reflejar el nuevo saldo mediante el prepared statement declarado anteriormente.
271                     saldo = saldo - importe;
272                     String aux_saldo2  = SET_SALDO_QRY;
273                     errorLog(aux_saldo2);
274                     pstmt = con.prepareStatement(aux_saldo2);
275                     pstmt.setDouble(1, saldo);
276                     pstmt.setString(2, pago.getTarjeta().getNumero());
277                     if (!(!pstmt.execute()
278                             && pstmt.getUpdateCount() == 1)){
279                         throw new EJBException("Error al actualizar el saldo");
280                     }
281                 }
282             } else { // Excepcion
283                 throw new EJBException("Error al obtener el saldo");
284             }
```

Por último, en la parte de cliente, en P1-ejb-transaccional/src/client/ssii2/controlador/ProcesaPago.java, añadimos la funcionalidad para capturar la excepción posible causada por esto:

```
186         try {
187             if (dao.realizaPago (pago) == null) {
188                 if (sesion != null) sesion.invalidate();
189                 enviaError(new Exception("Pago incorrecto"), request, response);
190                 return;
191             }
192         } catch (EJBException e) {
193             if (sesion != null) sesion.invalidate();
194             enviaError(e, request, response);
195         }
196
```

Ejercicio 8:

Desplegar y probar la nueva aplicación creada.

- **Probar a realizar pagos correctos. Comprobar que disminuye el saldo de las tarjetas sobre las que realice operaciones. Añadir a la memoria las evidencias obtenidas.**
- **Realice una operación con identificador de transacción y de comercio duplicados. Compruebe que el saldo de la tarjeta especificada en el pago no se ha variado.**
- **Incluya en la memoria de prácticas todos los pasos necesarios para resolver este ejercicio así como las evidencias obtenidas. Se pueden incluir por ejemplo capturas de pantalla.**

(Texto de la respuesta)

Ejercicio 9:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la factoría de conexiones empleando la consola de administración, tal y como se adjunta en la Figura 4.

Incluye una captura de pantalla donde se muestre dicha consola de administración con los cambios solicitados.

Ejercicio 10:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.Z2), declare manualmente la conexión empleando la consola de administración, tal y como se adjunta en la Figura 5.

Incluye una captura de pantalla donde se muestre dicha consola de administración con los cambios solicitados.

(Texto de la respuesta)

Ejercicio 11:

Modifique el fichero sun-ejb-jar.xml para que el MDB conecte adecuadamente a su *connection factory*.

- **Incluya en la clase VisaCancelacionJMSBean:**
  - **Consulta SQL necesaria para actualizar el código de respuesta a valor 999, de aquella autorización existente en la tabla de pagos cuyo idAutorizacion coincida con lo recibido por el mensaje.**
  - **Consulta SQL necesaria para rectificar el saldo de la tarjeta que realizó el pago.**
  - **Método onMessage() que implemente ambas actualizaciones. Para ello tome de ejemplo elcódigo SQL de ejercicios anteriores, de modo que se use un prepared statement que hagabind del idAutorizacion para cada mensaje recibido.**

- **Control de errores en el método onMessage y cierre de conexiones.**

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

(Texto de la respuesta)

Ejercicio 12:

Implemente ambos métodos en el cliente proporcionado. Deje comentado el método de acceso por la clase *InitialContext* de la API de JNDI. Indique en la memoria de prácticas qué ventajas podrían tener uno u otro método.

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

(Texto de la respuesta)

Ejercicio 13:

Automatice la creaciónde los recursos JMS (cola y factoría de conexiones) en el build.xml y jms.xml. Para ello, indique en jms.properties los nombres de ambos y el Physical Destination Name de la cola de acuerdo a los valores asignados en los ejercicios 9 y 10. Recuerde también asignar las direcciones IP adecuadas a las variables *as.host.mdb* (build.properties) y *as.host.server* (jms.properties). ¿Por qué ha añadido esas IPs?

(Texto de la respuesta)

Ejercicio 14:

Modifique el cliente, VisaQueueMessageProducer.java, implementando el envío de args[0] como mensajede texto (consultar los apéndices). Incluye en la memoria el fragmento de código que ha tenido que modificar.

Ejecute el cliente en el PC del laboratorio mediante el comando:

```
/opt/glassfish-4.1.2/glassfish/bin/appclient –targetserver 10.X.Y.Z -client
dist/clientjms/P1-jms-clientjms.jar idAutorizacion
```

Donde 10.X.Y.Z representa la dirección IP de la máquina virtual en cuyo servidor de aplicaciones seencuentra desplegado el MDB. Para garantizar que el comando funcione correctamente es necesario fijarla variable

```
(web console->Configurations->server-config->Java Message Service→JMS
Hosts→default_JMS_host)
```

que toma el valor "localhost" por la dirección IP de dicha máquina virtual. El cambio se puede llevar acabo desde la consola de administración. Será necesario reiniciar el servidor de aplicaciones para quesurja efecto.

Verifique el contenido de la cola ejecutando:

```
/opt/glassfish-4.1.2/glassfish/bin/appclient –targetserver 10.X.Y.Z -client
dist/clientjms/P1-jms-clientjms.jar -browse
```

Indique la salida del comando e inclúyala en la memoria de prácticas.

A continuación, volver a habilitar la ejecución del MDB y realizar los siguientes pasos:

- **Realice un pago con la aplicación web**
- **Obtenga evidencias de que se ha realizado**
- **Cancélelo con el cliente**
- **Obtenga evidencias de que se ha cancelado y de que el saldo se ha rectificado**

Al realizar este ejercicio en los laboratorios surge un error indicando que no es posible resolver el nombre del host local a una dirección IP. Esto se debe a que no hay una entrada con dicho nombre en el fichero /etc/hosts asociado a una dirección IP. Como dicho fichero no se puede editar, la solución es ejecutar elcliente de colas de mensajes desde la máquina virtual 1, para que se conecte a la máquina virtual 2. Basta con copiar el .jar del cliente a la máquina virtual, iniciar sesión de forma remota. Indicar donde seencuentra la versión 8 de java exportando la variable JAVA_HOME y

ejecutar el cliente de colas conappclient desde la máquina virtual 1. Para ello, hay que ejecutar la siguiente secuencia de comandos:

Desde PC1 host:

```
$ scp dist/clientjms/P1-jms-clientjms.jar si2@10.X.Y.Z1:/tmp
```

Desde la máquina virtual 10.X.Y.Z1:

```
si2@si2srv01:~$  export JAVA_HOME=/usr/lib/jvm/java-8-oracle/si2@si2srv01:~$
/opt/glassfish4/glassfish/bin/appclient     -targetserver10.X.Y.Z2 -client /tmp/
P1-jms-clientjms.jar <idAutorizacion>
```

(Texto de la respuesta)

Cuestiones:

Cuestión 1:

Abrir el archivo VisaDAOLocal.java y comprobar la definición de dicha interfaz. Anote en la memoria comentarios sobre las librerías Java EE importadas y las anotaciones utilizadas. ¿Para qué se utilizan? Comparar esta interfaz con el fichero de configuración del web service implementado en la práctica P1A.

Se usa el tag @Local, que emplea la librería de Java javax.ejb.Local. Esto significa que las precisiones se procesarán de forma local en la máquina virtual donde la aplicación ha sido desplegada. En comparación, en la práctica anterior se usa el tag @WebService, que hace que todas las peticiones se procesen usando la librería javax.jws.WebService.

Cuestión 2:

Abrir el archivo application.xml y explicar su contenido. Verifique el contenido de todos los archivos .jar/ .war / .ear que se han construido hasta el momento (empleando el comando jar -tvf). Anote sus comentarios y evidencias en la memoria.

En P1-ejb/conf/application/META-INF/application.xml podemos encontrar la información básica de la aplicación. Aquí, la etiqueta display-name nos indica el módulo de la aplicación:

```
<application version="5" xmlns="http://
  <display-name>P1-ejb</display-name>
```

Después, podemos observar la etiqueta ejb, que es el módulo correspondiente a los .jar

```
<module>
  <ejb>P1-ejb.jar</ejb>
</module>
```

También está la etiqueta web, que define la interfaz web, mediante la URI correspondiente a la aplicación y la raíz de la misma:

```
<module>
  <web>
      <web-uri>P1-ejb-cliente.war</web-uri>
      <context-root>/P1-ejb-cliente</context-root>
  </web>
</module>
```