|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Escuela Politécnica Superior  Ingeniería Informática  Prácticas de Sistemas Informáticos 2 | | | |
| **Grupo** | **2401** | **Práctica** | 1b | **Fecha** | 19/03/2021 |
| **Alumno/a** | | De las Heras Moreno, Martín | | | |
| **Alumno/a** | | Valderrábano Zamorano, Santiago Manuel | | | |

# Práctica 1b: Arquitectura de JAVA EE

## Cuestión número 1:

Las librerías son:

***import java.sql.Connection;***

***import java.sql.PreparedStatement;***

***import java.sql.ResultSet;***

***import java.sql.SQLException;***

***import java.sql.Statement;***

***import java.util.ArrayList;***

***import javax.ejb.Local;***

Las única anotación que hay es ***@Local*** encima de la clase VisaDAOLocal, que indica que la clase va a estar en el mismo emplazamiento que el cliente.

## Ejercicio 1:

Introducimos en VisaDAOBean las siguientes modificaciones como se indica en el enunciado:

***import javax.ejb.Stateless;***

***@Stateless(mappedName="VisaDAOBean")***

Aquí indicamos que la clase VisaDAO además de extender a DBTester, implementa la interfaz VisaDAOLocal:

***public class VisaDAOBean extends DBTester implements VisaDAOLocal {***

y además borramos el constructor por defecto y modificamos el método getPagos para que devuelva un array y así coincida con la definición VisaDAOLocal de la siguiente forma:

public PagoBean[] getPagos(String idComercio)

***PagoBean[] ret = null;***

***ret = new PagoBean[pagos.size()];***

***ret = pagos.toArray(ret);***

***return ret;***

## Ejercicio número 2:

En este ejercicio realizamos las siguientes modificaciones en **todos los servelets**:

1. Añadimos los nuevos imports:

***import javax.ejb.EJB;***

***import ssii2.visa.\*;***

(Ya que esto incluye VisaDAOLocal)

1. Añadimos el atributo para acceder al EJB Local:

***@EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)***

***private VisaDAOLocal dao;***

1. Comentamos los siguientes imports:

***// import javax.xml.ws.BindingProvider;***

***//*** ***import javax.xml.ws.WebServiceRef***

***// import javax.xml.ws.WebServiceException;***

1. Comentamos las siguientes líneas:

***// String rutaServicio = getServletContext().getInitParameter("rutaServicio");***

***//***

***// BindingProvider bp = (BindingProvider) dao;***

***// bp.getRequestContext().put(BindingProvider.ENDPOINT\_ADDRESS\_PROPERTY, rutaServicio);***

## Cuestión número 2:

El archivo application.xml se encuentra en ./conf/application/META-INF/application.xml. En este fichero se encuentra la información necesaria para el despliegue de la aplicación.

Para ver el contenido de los archivos .ear .jar y .war, ejecuutamos los siguientes comandos:

*jar -tvf dist/P1-ejb.ear* ***//*** *jar -tvf dist/client/P1-ejb-cliente.war*

*Texto

Descripción generada automáticamente*

*jar -tvf dist/server/P1-ejb.jar*

*Texto

Descripción generada automáticamente*

## Ejercicio número 3:

## Para desplegar nuestra aplicación, debemos indicar en los ficheros .properties donde se encuentran alojados BD, cliente y sevidor.

## En build.properties indicamos que tanto as.host.client como as.host.server son igual a 10.1.7.2 ya que tanto cliente como servidor se encontrarán en el mismo servidor. Esto se debe a que hora estamos accediendo a la interfaz local del Enterprise JavaBean (VisaDAOLocal).

## En postgresql.properties indicamos donde se encuentra la base de datos (db.host) y donde se encuentra la interfaz de acceso a la ella (db.client.host). Según está descrito en el esquema del enunciado, la base de datos se encuentra en db.host = 10.1.7.1 y la interfaz de acceso en db.client.host = 10.1.7 2

## Ejercicio número 4:

## En primer lugar, comprobamos que se ha desplegado correctamente todo. Para ello accedemos a la interfaz de acceso a la BD que se encuentra en <http://10.1.7.2:4848> como hemos definido en el ejercicio anterior.

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

## Después, comprobamos el correcto funcionamiento de la aplicación:

## Desde pago.html

***Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente***

## Interfaz de usuario gráfica Descripción generada automáticamente

## 

## Desde testbd.jsp:

## Interfaz de usuario gráfica, Aplicación Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico Descripción generada automáticamente

## Captura de pantalla de un celular Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico Descripción generada automáticamente

## Ejercicio número 5:

## Para este ejercicio vamos a implementar un cliente remoto en EJB. Para ello, como se indica en el enunciado, usaremos como material de partida P1-ejb para implementar el servidor remoto.

## En primer lugar, ejecutamos el comando cp P1-ejb –pr P1-ejb-servidor-remoto.

## A continuación, realizamos los siguientes cambios escritos en el enunciado:

## Copiar el fichero VisaDAOLocal.java al fichero VisaDAORemote.java.

## Cambiar, en VisaDAORemote.java, el nombre de la interfaz a VisaDAORemote y cambiar la anotación @Local por @Remote.

## Añadir import javax.ejb.Remote; y quitar import javax.ejb.Local

* Hacer que VisaDAOBean implemente ambas interfaces, la local y la remota.

## Después, tenemos que serializar tanto TarjetaBean con PagoBean. Para ello, modificamos los ficheros .java correspondientes añadiendo:

## import java.io.serializable

## public class TarjetaBean implements Serializable

## public class PagoBean implements Serializable

## Por último, compilamos, empaquetamos y desplegamos la aplicación P1-ejb de la misma forma que en el ejercicio 3.

## Ejercicio número 6:

## En este ejercicio vamos a implementar el cliente remoto de EJB. Para ello, como se indica en el enunciado, usaremos como material de partida P1-base. A continuación, realizamos las siguientes modificaciones:

## Cambiar en build.properties el nombre de la aplicación a P1-ejb-cliente-remoto

1. Eliminar el directorio P1-ejb-cliente-remoto/src/ssii2/visa/dao puesto que a partir de ahora la lógica se invocará de forma remota
2. Serializar tanto TarjetaBean como PagoBean de la misma forma que en el ejercicio anterior.
3. Copiar la interfaz VisaDAORemote.java que hemos creado en el ejercicio anterior a P1-ejb-cliente-remoto/src/ssii2/visa
4. Insertar en cada servelet la declaración del VisaDAORemote junto con la anotación en la que se especifica que la variable conecta con EJB como se indica en el enunciado, además de los imports necesarios.
5. Comentar la declaración VisaDAO dao en los servelets.
6. Crear el fichero glassfish-web.xml en web/WEB\_INF con el contenido especificado en el enunciado únicamente modificando la dirección IP por 10.1.7.2
7. Cambiar las direcciones IP de build.properties por as.host=10.1.7.1 y postgresql.properties por db.host=10.1.7.1 y db.client.host=10.1.8.2
8. **Extra:** En ProcesaPago.java insertamos un if(pago == null) tras hacer dao.realizaPago(pago) ya que generaba un error al compilar.

Para terminar, compilamos y desplegamos y nos conectamos a la aplicación del cliente para comprobar el correcto funcionamiento de todo.

A continuación incluimos las capturas que prueba dicho funcionamiento:

***Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente***Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

## Ejercicio número 7:

## En este ejercicio, vamos a modificar la aplicación Visa para que soporte el campo saldo. Para ello, realizamos los siguientes pasos que se indican en el enunciado:

1. cp P1-ejb –r P1-ejb-transaccional
2. cd P1-ejb-transaccional
3. tar -xzvf /ruta/a/P1-ejb-transaccional-base.tgz
4. ant limpiar-todo
5. En TarjetaBean.java añadidos el campo ***private double saldo*** además de sus ***métodos get y set***.
6. En VisaDAOBean.java,
   1. Importamos el nuevo tipo de excepción a tener en cuenta (import javax.ejb.EJBException;)
   2. Programamos las siguientes cosas:
      1. Prepared statements que se piden

Texto

Descripción generada automáticamente

* + 1. Modificar la función realizaPago:

Texto

Descripción generada automáticamente

1. En ProcesaPago.java:

Texto

Descripción generada automáticamente

## Ejercicio número 8:

## En este ejercicio comprobamos el correcto funcionamiento de lo realizado en el ejercicio 7. A continuación, adjuntamos las capturas que lo demuestran:

## Prueba 1:

## Pagos correctos

## Comprobar saldo

## Realizar pago

## Comprobar saldo

## Interfaz de usuario gráfica, Texto Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

## Interfaz de usuario gráfica, Aplicación Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico Descripción generada automáticamente

## Captura de pantalla de un celular Descripción generada automáticamente

## Identificador de transacción y de comercio duplicados

## Comprobar saldo

## Realizar pagos

## Comprobar saldo

## Interfaz de usuario gráfica, Aplicación Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación Descripción generada automáticamente

## Ejercicio número 9:

En este ejercicio declaramos una factoría de conexiones de forma manual accediendo al formulario mostrado en la captura. Accedemos a este desde la consola de administración y lo rellenamos según se indica en el enunciado.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

## Ejercicio número 10:

En este ejercicio creamos una cola de mensajes manualmente de la misma forma que en el ejercicio 9:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Ejercicio número 11:

En este ejercicio vamos a implementar un servicio que utilice la factoría de conexiones y la cola de mensajes para cancelar pagos a partir del idAutorizacion.

Partimos del código que se encuentra en Moodle llamado P1-jms-base.

En primer lugar, modificamos el fichero sun-ejb-jr.xml añadiendo lo especificado en el enunciado para conectarse correctamente a la conexión Factory.

**CAPTURA**

A continuación, añadimos 2 queries a ***VisaCancelacionJMSBean.java*** que se encargan una poner el codRepuesta del pago a 999 y otra de modificar el saldo de la tarjeta con la que se realizó el pago que se ha cancelado.

**CAPTURA**

Después, modificamos la función onMessage de ***VisaCancelacionJMSBean.java*** para que haga las llamadas necesarias a las queries declaradas anteriormente.

**CAPTURA**

Estas modificaciones realizan lo siguiente.

1. Actualizar el código de respuesta a 999.
2. Rectificar el saldo del cliente que realizó el pago que se ha cancelado.

## Ejercicio número 12:

En este ejercicio desarrollamos los dos métodos posiles de acceder a los objetos de factoría de conexiones y cola de mensajes de clientes, creados en los ejercicios 9 y 10 respectivamente, desde el cliente.

El primer método consiste en declarar ambos objetos y utilizar anotaciones.

**CAPTURA**

El segundo método consiste usar la API JNDI de la siguiente forma:

**CAPTURA**

Las ventajas del método JMS dinámico, respecto al estático, son las siguientes:

1. Al ser dinámico, se podrían introducir cambios en tiempo de ejecución
2. Aunque en ambos hay que definir en la instancia los nombres de los recursos, en el primer método es una etiqueta fija mientras que en el segundo se busca el recurso durante la ejecución por lo que puedes declarar varios recursos e ir usándolos según se necesiten.

La principal desventaja de JMS dinámico a través de la API JNDI es que introduce una mayor complejidad en el sistema.

## Ejercicio número 13:

En este ejercicio vamos a automatizar la creación de los recursos JMS. Para ello, modificamos en primer lugar los ficheros

* jms.properties
  + as.host.server = 10.1.7.2
  + as.host.client = 10.1.7.2
  + jms.factoryname = jms/VisaConnectionFactory
  + jms.name = jms/VisaPagosQueue
  + jms.physname = VisaPagosQueue
* build.properties
  + as.host.mdb = 10.1.7.2

Para probar que esto funciona correctamente, eliminamos manualmente los recursos a través de la consola de administración y después ejecutamos ***ant todo***  y comprobamos que se han creado bien:

**CAPTURA**

## Ejercicio número 14:

En este ejercicio