|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Escuela Politécnica Superior  Ingeniería Informática  Prácticas de Sistemas Informáticos 2 | | | |
| **Grupo** | **1391** | **Práctica** | **1B** | **Fecha** | 14/03/2022 |
| **Alumno/a** | | Gil Maroto, Lucía | | | |
| **Alumno/a** | | Varela Sánchez, Daniel | | | |
| **Alumno/a** | | Martín-Coello Juárez, Guillermo | | | |

# Práctica 1B: Arquitectura de JAVA EE (Segunda parte)

## 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.**

Tras comprobar el archivo, vemos que la librería que más nos llama la atención es “javax.ejb.Local” porque gracias a ella podemos utilizar la definición de la interfaz se hace con la anotación @Local.

Esta librería y anotación se utiliza para que el Bean de nuestra práctica sea accesible de manera local, por lo que cliente y servidor se ejecutarán desde el mismo servidor.

## Ejercicio número 1:

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

* **Hacer que la clase implemente la interfaz local y convertirla en un EJB stateless mediante la anotación Stateless.**

En el archivo VisaDAOBean:

Texto

Descripción generada automáticamente

* **Eliminar el constructor por defecto de la clase.**

En el archivo VisaDAOBean:

Texto

Descripción generada automáticamente

* **Ajustar el método getPagos() a la interfaz definida en VisaDAOLocal**

En el archivo VisaDAOBean:

Texto

Descripción generada automáticamente

En el archivo GetPagos:

Captura de pantalla de computadora

Descripción generada automáticamente

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

Las capturas de código están en cada sección del ejercicio.

## Ejercicio número 2:

**Modificar el servlet ProcesaPago para que acceda al EJB local. Para ello, modificar el archivo ProcesaPago.java de la siguiente manera:**

* **En la sección de preproceso, añadir las siguientes importaciones de clases que se van a utilizar:**

**...**

**import javax.ejb.EJB;**

**import ssii2.visa.VisaDAOLocal;**

**...**

Texto

Descripción generada automáticamente

* **Se deberán eliminar estas otras importaciones que dejan de existir en el proyecto, como por ejemplo:**

**~~import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente~~**

**~~import ssii2.visa.VisaDAOWS; // Stub generado automáticamente~~**

Texto

Descripción generada automáticamente

* **Añadir como atributo de la clase el objeto proxy que permite acceder al EJB local, con su correspondiente anotación que lo declara como tal:**

**...**

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

**private VisaDAOLocal dao;**

**...**

El código para esta parte se encuentra capturado en el apartado de capturas.

* **En el cuerpo del servlet, eliminar la declaración de la instancia del antiguo webservice VisaDAOWS, así como el código necesario para obtener la referencia remota**

**...**

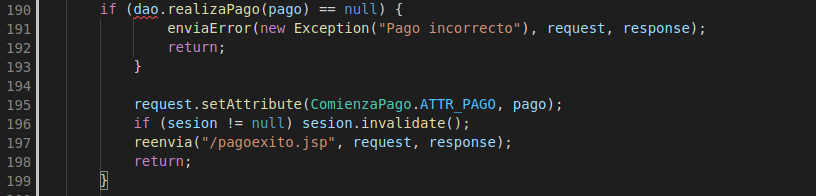
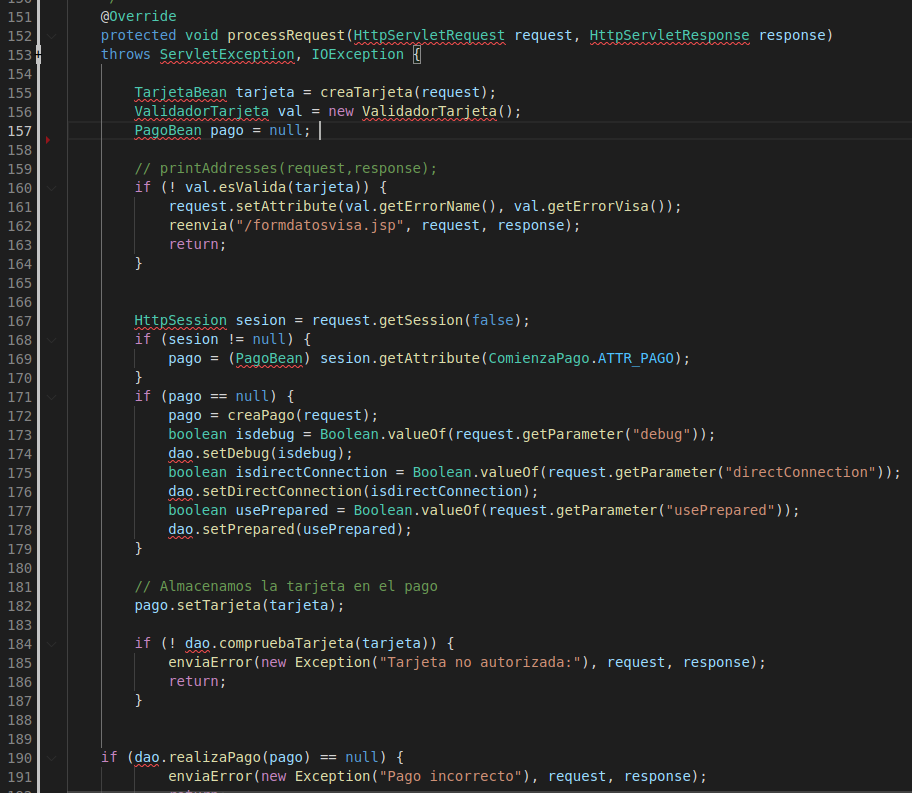
**~~VisaDAOWS dao = null;~~**

**~~VisaDAOWSService service = new VisaDAOWSService();~~**

**~~dao = service.getVisaDAOWSPort()~~**

El código para esta parte se encuentra capturado en el apartado de capturas.

* **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 código al cambiar la interfaz del antiguo VisaDAOWS (en particular, el método getPagos()).**

Por tanto, hemos repetido el mismo proceso en GetPagos.java y en DelPagos.java. Añadimos las capturas. Para GetPagos.java:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Para DelPagos.java:

Captura de pantalla de un celular

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## 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). Explique brevemente el contenido y ponga evidencias en la memoria.**

El archivo aplication.xml contiene el descriptor de despliegue de la aplicación de empresa. En su interior se describe cada uno de los módulos de la aplicación (ejb y web) acorde con las etiquetas definidas en el fichero XSD.

Texto

Descripción generada automáticamente

Para obtener los archivos .jar hay que empaquetar el servidor; para obtener los archivos .war hay que empaquetar el cliente; y para obtener los archivos .ear hay que empaquetar la aplicación. Después, los leemos por terminal con los siguientes comandos:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

## Ejercicio número 3:

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

* **Editar el archivo build.properties para que la propiedad as.host contenga la dirección IP del servidor de aplicaciones. Indica el valor y porqué es ese valor.**



Hemos cambiado as.host al valor 10.7.8.1 que es la IP de nuestra máquina virtual (PC2 de la Figura 2 del pdf de la práctica).

* **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.**

Texto

Descripción generada automáticamente

Hemos cambiado db.host al valor 10.10.108.134 que es la IP de PC (PC1 de la Figura 2 del pdf de la práctica) y es donde se albergará la base de datos.

* **Desplegar la aplicación de empresa (ant desplegar).**

Hemos comprobado que, tras desplegar, nuestra aplicación se encuentra bajo el apartado de Enterprise Applications

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Podemos acceder a nuestra aplicación desde el siguiente puerto:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Ejercicio número 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.**

**Si la base de datos no se ha generado previamente, será necesario crearla usando ant regenerar-bd.**

**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.**

Primero, realizamos un pago desde pago.html:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Comprobamos que se realiza correctamente:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Ahora probamos la funcionalidad de la aplicación con testbd.jsp. Comprobamos que el pago se realiza correctamente.

Interfaz de usuario gráfica, Aplicación

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

Descripción generada automáticamente

Consultamos los pagos del comercio 1:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Probamos a borrar los pagos del comercio 1.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

La aplicación se ha desplegado correctamente y cumple toda su funcionalidad.

## Ejercicio número 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.**

En VisaDAORemote.java:

Texto

Descripción generada automáticamente

En VisaDAOBean.java:

Texto

Descripción generada automáticamente

En PagoBean.java:

Texto

Descripción generada automáticamente

En Tarjeta Bean,java:

Texto

Descripción generada automáticamente

Desplegamos la aplicación satisfactoriamente.

Texto

Descripción generada automáticamente

## Ejercicio número 6:

**Realizar los cambios comentados en la aplicación P1-base para convertirla en P1-ejb-clienteremoto. Compilar, empaquetar y desplegar de nuevo la aplicación en otra máquina virtual distinta a la de la 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.**

Primero hemos copiado P1-base a P1-ejb-cliente remoto y hemos cambiado el nombre de la aplicación a P1-ejb-cliente-remoto. Después hemos eliminado el directorio src/sii2/visa/dao. Hemos marcado como serializables PagoBean y TarjetaBean e implementado la interfaz Serializable:

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

En los archivos procesaPago getPagos y delPagos hemos eliminado la declaración de VisaDAO dao por una declaración de la siguiente forma fuera del método dentro de la clase:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Hemos tenido que cambiar también la condición !dao.realizaPago(pago) en procesaPago ya que daba error, ahora es de la siguiente forma:

Texto

Descripción generada automáticamente

Hemos creado un archivo glassfish-web.xml en el siguiente directorio:

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

siguiendo la plantilla proporcionada y cambiando el 10.x.y.z1 por 10.8.7.3 la cual es la IP de nuestro servidor remoto.

Hemos cambiado las IP de los archivos properties sabiendo que nuestro cliente se halla en 10.8.7.1 y nuestro servidor en 10.8.7.3 :

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Hemos probado a realizar un pago y ha sido exitoso:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

La comprobación que el pago ha sido guardado en la base de datos:

Tabla

Descripción generada automáticamente

## Ejercicio número 7:

**Modificar la aplicación VISA para soportar el campo saldo:**

**Archivo TarjetaBean.java:**

* **Añadir el atributo saldo y sus métodos de acceso: private double saldo;**

**Archivo VisaDAOBean.java:**

* **Importar la definición de la excepción EJBException que debe lanzar el servlet para indicar que se debe realizar un rollback: import javax.ejb.EJBException;**
* **Declarar un prepared statement para recuperar el saldo de una tarjeta de la base de datos.**
* **Declarar un prepared statement para actualizar el nuevo saldo calculado en la base de datos.**
* **Modificar el método realizaPago con las siguientes acciones:**
* **Recuperar el saldo de la tarjeta a través del prepared statement declarado anteriormente.**
* **Comprobar si el saldo es mayor o igual que el importe de la operación. Si no lo es, retornar denegando el pago (idAutorizacion= null y pago retornado=null)**
* **Si el saldo es suficiente, decrementarlo en el valor del importe del pago y actualizar el registro de la tarjeta para reflejar el nuevo saldo mediante el prepared statement declarado anteriormente.**
* **Si lo anterior es correcto, ejecutar el proceso de inserción del pago y obtención del idAutorizacion, tal como se realizaba en la práctica anterior (este código ya debe estar programado y no es necesario modificarlo).**
* **En caso de producirse cualquier error a lo largo del proceso (por ejemplo, si no se obtiene el idAutorizacion porque la transacción está duplicada), lanzar una excepción EJBException para retornar al cliente.**
* **Modificar el servlet ProcesaPago para que capture la posible interrupción EJBException lanzada por realizaPago, y, en caso de que se haya lanzado, devuelva la página de error mediante el método enviaError (recordar antes de retornar que se debe invalidar la sesión, si es que existe).**
* **Incluye en la memoria cada fragmento de código donde se han ido añadiendo las modificaciones solicitadas**

TarjetaBean:

Texto

Descripción generada automáticamente

VisaDAOBean:

Texto

Descripción generada automáticamente

## Ejercicio número 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.**

Hemos probado a realizar un pago y ha sido exitoso:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

La comprobación que el pago ha sido guardado en la base de datos:

Tabla

Descripción generada automáticamente

Comprobamos que el valor del saldo se ha reducido con la compra



Tras intentar realizar exactamente la misma operación, con los mismos valores, comprobamos cómo el valora del saldo en la base de datos no ha variado:



## Ejercicio número 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.**

Abrimos la página correspondiente y añadimos los datos que nos piden.

Texto, Tabla

Descripción generada automáticamente

## Ejercicio número 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.**

Añadimos los cambios solicitados-

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## Ejercicio número 11:

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

Texto

Descripción generada automáticamente

* **Incluya en la clase VisaCancelacionJMSBean:**
* **Consulta SQL necesaria para obtener el código de respuesta del pago cuyo idAutorizacion coincida con lo recibido por el mensaje**



* **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**

Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media

* **Método onMessage() que obtenga el idAutorización de la cola de mensajes, compruebe si el pago con dicho idAutorizacon tiene código de respuesta 000 y en ese caso actualice el código de respuesta y rectifique el saldo de la tarjeta. Para ello tome de ejemplo el código SQL de ejercicios anteriores, de modo que se use un prepared statement que haga bind del idAutorizacion para cada mensaje recibido.**

Texto

Descripción generada automáticamente

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

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

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

## Ejercicio número 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

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Los estáticos, al ser estáticos, están anotados con nombres de recursos; mientras que los dinámicos no tienen por qué indicar su nombre antes de la compilación

## Ejercicio número 13:

**Automatice la creación de 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?**

**Borre desde la consola de administración de Glassfish la connectionFactory y la cola creadas manualmente y ejecute:**

cd P1-jms

ant todo

**Compruebe en la consola de administración del Glassfish que, efectivamente, los recursos se han creado automáticamente. Incluye una captura de pantalla, donde se muestre la consola de administración con los recursos creados. Revise el fichero jms.xml y anote en la memoria de prácticas cuál es el comando equivalente para crear una cola JMS usando la herramienta asadmin.**

Texto

Descripción generada automáticamente Imagen que contiene texto, teléfono, celular, tabla

Descripción generada automáticamente

## Ejercicio número 14:

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

**Para enviar mensajes a la cola de mensajes usando el cliente se debe usar el comando:**

<dir-glassfish>/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 se encuentra desplegado el MDB. Para garantizar que el comando funcione correctamente es necesario fijar la 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 a cabo desde la consola de administración. Será necesario reiniciar el servidor de aplicaciones para que surja efecto.**

**Para verificar el contenido de la cola se debe ejecutar:**

<dir-glassfish>/glassfish/bin/appclient –targetserver 10.X.Y.Z -client

dist/clientjms/P1-jms-clientjms.jar -browse

**Detenga la ejecución del MDB con la consola de administración para que no consuma los mensajes de la cola (check de ‘Enabled’ en Applications/P1-jms-mdb y guardar los cambios).**

**Envíe un mensaje a la cola, por ejemplo, el texto “idAutorizacion” y compruebe el contenido de la cola.**

**Incluya en la memoria de prácticas evidencias de todos los pasos, por ejemplo, capturas de pantalla.**

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

* **Realice un pago con la aplicación web (P1-ejb-transaccional)**
* **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**

**Importante: 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 el cliente 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 se encuentra la versión 8 de java exportando la variable JAVA\_HOME y ejecutar el cliente de colas con appclient 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 -targetserver 10.X.Y.Z2-client /tmp/P1-jms-clientjms.jar <idAutorizacion>

Texto

Descripción generada automáticamente