		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2402	Práctica	1 B	Fecha	13/03/2019
Alumno/a		Gómez, Borzdynski, Óscar			
Alumno/a		Polanía, Bernárdez, Pablo Alejo			

Práctica 1B: Arquitectura de Java EE (Segunda parte)

**** Las etiquetas requeridas en esta entrega se escriben sin argumentos:**

Ejemplo: `@Stateless(mappedName = "NombreMetodo")` pasaría a ser `@Stateless` **

Cuestión 1:

Importamos `javax.ejb.Local` y lo usamos para anotar la interfaz (etiqueta `@Local`). Esto indica que esta interfaz está destinada a procesar peticiones dentro del mismo contenedor (en este caso máquina virtual) donde la aplicación ha sido desplegada. Por tanto la transparencia de ubicación del *EJB* es baja.

Ejercicio número 1:

Siguiendo el enunciado se han añadido las siguientes líneas al código de `VisaDAOBean.java`:

```
import javax.ejb.EJBException;
import javax.ejb.Stateless;

/**
 * @author jaime
 */
@Stateless
public class VisaDAOBean extends DBTester implements VisaDAOLocal {
```

El primer `import` será necesario posteriormente. El segundo sirve para indicar que la clase es un *EJB* de sesión sin estado. Además, la clase ha sido renombrada y se ha hecho que implemente la interfaz `VisaDAOLocal`.

Al implementar esta interfaz, se debe ajustar el retorno del método `getPagos()` de la clase `VisaDAOBean` al retorno definido en `VisaDAOLocal`. Por tanto se ha cambiado al tipo array de `PagoBean`:

```
public PagoBean[] getPagos(String idComercio) {
```

Por último se ha comentado el constructor predefinido de la clase `VisaDAOBean`.

Ejercicio número 2:

Para este ejercicio se va a modificar la parte de la aplicación del cliente. Se accede al archivo *ProcesaPago* y se eliminan todos los *imports* y referencias a la implementación *Web Service (WS)*, tales como el *Binding Provider* o los *stubs* de la práctica 1A.

Para poder implementar la funcionalidad de *EJB local* se añaden los siguientes *imports*:

```
// import ssii2.visa.dao.VisaDAO;  
// import ssii2.visa.VisaDAOWSService; // Stub generado automáticamente  
// import ssii2.visa.VisaDAOWS; // Stub generado automáticamente  
// import javax.xml.ws.WebServiceRef;  
// import javax.xml.ws.BindingProvider;  
import javax.ejb.EJB;  
import ssii2.visa.VisaDAOLocal;
```

El primero para poder hacer uso de la etiqueta *@EJB* y el segundo para indicar la clase que se implementará como *EJB*.

Además se añade un atributo privado a la clase con la etiqueta *@EJB*. Esto permitirá acceder al *EJB* local del servidor:

```
@EJB(name="VisaDAOBean", beanInterface=VisaDAOLocal.class)  
private VisaDAOLocal dao;
```

También, debido al cambio de retorno del método *getPagos()* del apartado anterior se modifica la siguiente línea en el fichero:

```
/* Petición de los pagos para el comercio */  
PagoBean[] pagos = dao.getPagos(idComercio);
```

Se llevan a cabo modificaciones similares en los ficheros *GetPagos()* y *DelPagos()* ya que ambos hacían uso de la clase *VisaDAOWS*.

Cuestión 2:

Siguiendo la ruta `/conf/application/META-INF/` accedemos al fichero `application.xml`. Al principio del fichero aparece la etiqueta `xml` que indentifica el fichero como de tipo `xml` (indica la versión y que soporta la codificación `UTF-8`). Siguiendo las definiciones de la documentación del archivo http://java.sun.com/xml/ns/javaee/application_5.xsd (proporcionado en el enunciado) vemos que se usa la etiqueta `application`, la cual hace referencia al elemento raíz del descriptor de despliegue de la aplicación Java EE (este fichero). Esta etiqueta viene asociada a ciertos parámetros que indican de donde se ha cogido la definición de la etiqueta y el esquema que se seguirá. Define el despliegue gracias a etiquetas como:

- `display-name`: nombre de la aplicación
- `module`: define cada módulo de la aplicación, con las etiquetas internas se define el tipo:
 - `web`: define el módulo con interfaz web, define su uri y su contexto.

En cuanto a los ear y los jar:

- `Ear`: tiene los dos jar dentro (es la aplicación completa)
 - `Jar`: contiene las clases del servidor
 - `War`: contiene las clases del cliente, así como la interfaz web

Además podemos ver que cliente y servidor están definidos como dos módulos diferentes en el `application.xml`. En el primero aparece la etiqueta `ejb` en la cual se especifica la `URI` principal de un fichero `ejb-jar` asociado al nivel más alto de la aplicación. En este caso el fichero es `P1-ejb.jar`. Vemos, con el comando `find`, que se encuentra en el directorio `./dist/server/`. Ejecutando el comando `jar -tvf P1-ejb.jar` se visualiza el contenido del `.jar`:

```
gjo@gjo-Lenovo-ideapad-310-15ISK:~/Documentos/Universidad/Cuarto/Informatica/SI2
/SI2/P1b/P1-ejb/dist/server$ jar -tvf P1-ejb.jar
 0 Tue Mar 05 01:16:42 CET 2019 META-INF/
105 Tue Mar 05 01:16:40 CET 2019 META-INF/MANIFEST.MF
 0 Tue Mar 05 00:07:28 CET 2019 ssi2/
 0 Tue Mar 05 00:07:28 CET 2019 ssi2/visa/
 0 Tue Mar 05 00:07:28 CET 2019 ssi2/visa/dao/
255 Tue Mar 05 00:07:36 CET 2019 META-INF/sun-ejb-jar.xml
1464 Tue Mar 05 00:07:28 CET 2019 ssi2/visa/PagoBean.class
 856 Tue Mar 05 00:07:28 CET 2019 ssi2/visa/TarjetaBean.class
 593 Tue Mar 05 00:07:28 CET 2019 ssi2/visa/VisaDAOLocal.class
1745 Tue Mar 05 01:16:40 CET 2019 ssi2/visa/dao/DBTester.class
6947 Tue Mar 05 01:16:40 CET 2019 ssi2/visa/dao/VisaDAOBean.class
```

En el segundo módulo se especifica, con la etiqueta `web`, que el módulo es del tipo `webType` de la definición `javaee`.

Con la etiqueta `context-root` se hace referencia al contexto raíz de la aplicación web, en este caso `P1-ejb-cliente`. La etiqueta `web-uri` sirve para especificar la `URI` del fichero de aplicación web asociado al nivel más alto de la aplicación. Se ve su contenido con el comando `jar -tvf P1-ejb-cliente.war`:

```

gjo@gjo-Lenovo-ideapad-310-15ISK:~/Documentos/Universidad/Cuarto/Informatica/SI2/SI2/P1b/P1-
ejb/dist/client$ jar -tvf P1-ejb-cliente.war
 0 Tue Mar 05 01:16:42 CET 2019 META-INF/
105 Tue Mar 05 01:16:40 CET 2019 META-INF/MANIFEST.MF
 0 Tue Mar 05 00:07:56 CET 2019 WEB-INF/
 0 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/
 0 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/
 0 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/controlador/
 0 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/filtros/
 0 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/visa/
 0 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/visa/error/
 0 Tue Mar 05 00:07:26 CET 2019 WEB-INF/lib/
 0 Tue Mar 05 00:07:56 CET 2019 error/
2844 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/controlador/ComienzaPago.class
1513 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/controlador/DelPagos.class
1365 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/controlador/GetPagos.class
4919 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/controlador/ProcesaPago.class
1894 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/controlador/ServletRaiz.class
2608 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/filtros/CompruebaSesion.class
3170 Tue Mar 05 00:07:46 CET 2019 WEB-INF/classes/ssii2/visa/ValidadorTarjeta.class
 616 Tue Mar 05 01:16:42 CET 2019 WEB-INF/classes/ssii2/visa/error/ErrorVisa.class
 198 Tue Mar 05 01:16:42 CET 2019 WEB-INF/classes/ssii2/visa/error/ErrorVisaCVV.class
 209 Tue Mar 05 01:16:42 CET 2019 WEB-INF/classes/ssii2/visa/error/ErrorVisaFechaCaducidad
.class
 207 Tue Mar 05 01:16:42 CET 2019 WEB-INF/classes/ssii2/visa/error/ErrorVisaFechaEmision.c
lass
 201 Tue Mar 05 01:16:42 CET 2019 WEB-INF/classes/ssii2/visa/error/ErrorVisaNumero.class
 202 Tue Mar 05 01:16:42 CET 2019 WEB-INF/classes/ssii2/visa/error/ErrorVisaTitular.class

```

Ejercicio número 3:

Para desplegar el entorno de manera correcta se modifica en primer lugar el fichero *build.properties*, en particular, las propiedades *as.host.client* y *as.host.server*. Ambas pertenecerán a la máquina virtual 2 debido a que se utiliza el *EJB* de manera local. Es necesario por tanto que la definición de las clases que el cliente usará y el despliegue de éstas se produzca en el mismo módulo que el del cliente:

```
as.host.client=10.2.5.2
as.host.server=10.2.5.2
```

Por el contrario los datos usados por la aplicación residen en el lado del servidor de la base de -datos. Por tanto, las propiedades *db.host* y *db.client.host* del fichero *postgresql.properties* harán referencia a la máquina virtual 1:

```
db.host=10.2.5.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}:${db.port}/${db.name}
db.client.host=10.2.5.1
db.client.port=4848
```

Se comprueba que el despliegue de la aplicación se ha efectuado correctamente:

The screenshot shows the GlassFish Server Open Source Edition web console. The browser address bar displays `https://10.2.5.2:4848/common/index.jsf`. The page header includes navigation links for 'Home' and 'About...', and a status bar showing 'User: admin', 'Role: domain1', and 'Server: 10.2.5.2'. The main content area is titled 'GlassFish™ Server Open Source Edition' and shows 'Total # of available updates : 1'. On the left, a tree view under 'Common Tasks' shows the hierarchy: Domain > server (Admin Server) > Clusters > Standalone Instances > Nodes > Applications > P1-ejb. The right pane is titled 'Edit Application' and contains the following configuration details:

- Name:** P1-ejb
- Status:** ☒ Enabled
- Virtual Servers:** A dropdown menu showing 'server'.
- Implicit CDI:** ☒ Enabled. Description: Implicit discovery of CDI beans.
- Java Web Start:** ☒ Enabled. Description: You must redeploy the application to change Java Web Start Support.
- Location:** `${com.sun.aas.instanceRootURI}/applications/P1-ejb/`
- Deployment Order:** 100. Description: A number that determines the loading order of the application at server startup. Lower numbers are loaded first. The de
- Libraries:** (Empty field)
- Description:** (Empty field)

Ejercicio número 4:

Se comprueba el correcto despliegue de la aplicación realizando las tareas habituales. Primero se consiguen los datos de una tarjeta accediendo a la base de datos con *PgAdmin*:



Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

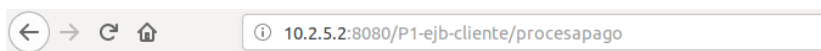
CVV2:

Modo debug: ☐ True ☐ False

Direct Connection: ☐ True ☐ False

Use Prepared: ☐ True ☐ False

Vemos que se ha ejecutado de manera correcta:



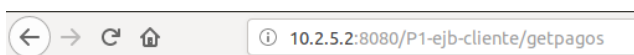
Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 100.0
codRespuesta: 000
idAutorizacion: 7

[Volver al comercio](#)

Enlistamos los pagos realizados:



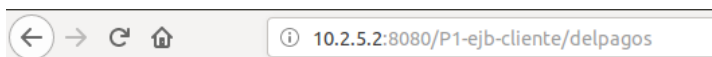
Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
13	24.0	000	6
1	100.0	000	7

[Volver al comercio](#)

Y eliminamos el pago realizado:



Pago con tarjeta

Se han borrado 2 pagos correctamente para el comercio 1

[Volver al comercio](#)

Se puede apreciar en la *url* que todos los procesos se han llevado a cabo a través de *P1-ejb-cliente* como se especificaba en el fichero *application.xml*.

Ejercicio número 5:

Siguiendo los pasos mencionados en el enunciado se ha creado el fichero *VisaDAORemote.java* copiando la información del fichero *VisaDAOLocal.java* y cambiando el *import* por *Remote* al igual que la etiqueta *@Remote* para la nueva interfaz (*VisaDAORemote*):

```
import javax.ejb.Remote;

@Remote
public interface VisaDAORemote {
```

Además en los ficheros *PagoBean.java* y *TarjetaBean.java* se ha añadido la implementación de la interfaz *java.io.Serializable*:

```
import java.io.Serializable;

/**
 * @author jaime
 */
public class PagoBean implements Serializable{

import java.io.Serializable;

public class TarjetaBean implements Serializable{
```

Por último se especifica que la nueva clase *VisaDAORemote* implemente tanto la clase *VisaDAOLocal* y la clase *VisaDAORemote*:

```
public class VisaDAOBean extends DBTester implements VisaDAOLocal, VisaDAORemote {
```

Las modificaciones que se llevaron a cabo en los ficheros *GetPagos.java* y *DelPagos.java* no han sido necesarias ya que estos han sido copiados del directorio *P1-ejb* al directorio *P1-ejb-cliente-remoto*.

Ejercicio número 6:

En este apartado se construirá el cliente remoto del EJB. Para ello se siguen todos los pasos indicados en el enunciado:

Se crea el cliente remoto (*P1-ejb-cliente-remoto*), tomando como referencia P1-base. Se eliminan los directorios con el path *ssii2/visa/dao* de la carpeta *src*. Se copia el fichero *VisaDAORemote.java* de la parte de servidor y añadimos la nueva variable *dao* de tipo *VisaDAORemote*. Esta modificación se lleva a cabo en todos los ficheros que hacían uso del objeto *VisaDAO*: *ProcesaPago.java*, *Getpagos.java* y *DelPagos.java*. Para indicarle que se trata de un EJB se utiliza la etiqueta *@EJB* importada con los *imports* siguientes:

```
import javax.ejb.EJB;
import ssii2.visa.VisaDAORemote;

/**
 *
 * @author phaya
 */
public class ProcesaPago extends ServletRaiz {

    @EJB(name = "VisaDAOBean", beanInterface = VisaDAORemote.class)
    private VisaDAORemote dao;
```

Como se había hecho en el anterior apartado hacemos que las clases *PagoBean* y *TarjetaBean* implementen la clase *Serializable* de *java.io*. Además se vuelven a modificar *realizaPago* y *ProcesaPago* para que devuelvan objetos del tipo *array* de *PagoBean*.

(No se adjuntan capturas ya que son cambios que se han hecho con anterioridad)

Por último se lleva a cabo la creación del fichero *glassfish-web.xml* y se añaden las líneas indicadas en el enunciado modificando la *ip* para que coincida con nuestra máquina virtual:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD
GlassFish Application Server 3.1 Servlet 3.0//EN" "http://
glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app>
  <ejb-ref>
    <ejb-ref-name>VisaDAOBean</ejb-ref-name>
    <jndi-name>corbaname:iiop:10.2.5.2:3700#java:global/
P1-ejb/P1-ejb/VisaDAOBean!ssii2.visa.VisaDAORemote</
jndi-name>
  </ejb-ref>
</glassfish-web-app>
```

Se han modificado las direcciones *ip* de *build.properties* y de *postgresql.properties* para que apunten a la otra máquina virtual: 10.2.5.1. Así se consigue que tanto el servidor se despliegue de manera remota (apartado anterior) y el cliente acceda a este despliegue de manera remota también (este apartado). Se recuerda que el EJB se despliega en la máquina virtual 2.

Para terminar se comprueba que los cambios realizados funcionan. Tras desplegar la aplicación se llevan a cabo las comprobaciones habituales.

Datos del pago y confirmación de su éxito:

← → ↺ 🏠

10.2.5.1:8080/P1-ejb-cliente-remoto/testbd.jsp

← → ↺ 🏠

10.2.5.1:8080/P1-ejb-cliente-remoto/procesapago

Pago con tarjeta

Proceso de un pago

Id Transacción:

11

Id Comercio:

1

Importe:

100

Numero de visa:

0039 2818 1198 8592

Titular:

Hugo Linus Sparrow

Fecha Emisión:

01/10

Fecha Caducidad:

09/20

CVV2:

971

Modo debug:

☐ True ☐ False

Direct Connection:

☐ True ☐ False

Use Prepared:

☐ True ☐ False

Pagar

Pago con tarjeta

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

idTransaccion: 11

idComercio: 1

importe: 100.0

codRespuesta: 000

idAutorizacion: 8

[Volver al comercio](#)

Listado de pagos y borrado del mismo:

← → ↺ 🏠

10.2.5.1:8080/P1-ejb-cliente-remoto/getpagos

← → ↺ 🏠

10.2.5.1:8080/P1-ejb-cliente-remoto/delpagos

Pago con tarjeta

Lista de pagos del comercio 1

idTransaccion	Importe	codRespuesta	idAutorizacion
11	100.0	000	8

[Volver al comercio](#)

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Ejercicio número 7:

Hasta ahora al utilizar *EJBs* se estaban llevando a cabo transacciones. Para reflejar el uso de éstas y llevar a cabo *commits* o *rollbacks* se modificará la funcionalidad del método *realizaPago*. Para ello se han llevado a cabo los cambios y ediciones de directorios y ficheros descritos en el enunciado para crear *P1-ejb-transaccional*.

Además se ha modificado la base de datos añadiendo a la tabla *Tarjetas* el campo *Saldo*. Para que la aplicación soporte y maneje este nuevo campo se llevan a cabo las siguientes modificaciones:

- Se añade el campo saldo al archivo *TarjetaBean.java*:

```
public class TarjetaBean {  
    private double saldo;
```

En *VisaDAOBean.java*:

- Como ya se llevo a cabo en el ejercicio 1 el *import* de *javax.ejb.EJBException* no se ha tenido que volver a importar.
- Se añade el segmento de código que realiza las consultas a la base de datos para conseguir el saldo y para actualizarlo. Para ello se ha seguido la estructura de las consultas ya instanciadas:

```
private static final String GET_SALDO_QRY =  
    "select saldo from tarjeta "  
    "where numeroTarjeta=?";  
  
private static final String SET_SALDO_QRY =  
    "update tarjeta "  
    "set saldo = ?" +  
    "where numeroTarjeta=?";
```

- Se modifica el código de *realizaPago* siguiendo las instrucciones de la práctica:

```
// Comprobar que tenemos saldo suficiente  
String getSaldo = GET_SALDO_QRY;  
errorLog(getSaldo);  
pstmt = con.prepareStatement(getSaldo);  
pstmt.setString(1, pago.getTarjeta().getNumero());  
ret = null;  
rs = pstmt.executeQuery();  
if(rs.next()){  
    double saldo_actual = rs.getDouble("saldo");  
    double saldo_final = saldo_actual - pago.getImporte();  
    if (saldo_final < 0){  
        pago.setIdAutorizacion(null);  
        return null;  
    }  
    else{  
        String setSaldo = SET_SALDO_QRY;  
        errorLog(setSaldo);  
        pstmt = con.prepareStatement(setSaldo);  
        pstmt.setDouble(1, saldo_final);  
        pstmt.setString(2, pago.getTarjeta().getNumero());  
        ret = null;  
        if (!pstmt.execute() && pstmt.getUpdateCount() == 1) {  
            ret = pago;  
        }else{  
            throw new EJBException();  
        }  
    }  
}  
else{  
    throw new EJBException();  
}
```

- En *ProcesaPago.java*, manejamos la llamada al método *realizaPago* con la estructura *try catch* para capturar la excepción que lanzamos en el DAO.

```
try{
    pago = dao.realizaPago(pago);
}
catch (Exception e) {
    if (sesion != null) sesion.invalidate();
    enviaError(new Exception("Pago incorrecto"), request, response);
}
```

La captura de la excepción es más general que la lanzada en *realizaPago* por si ocurre una de cualquier tipo.

Ejercicio número 8:

En este ejercicio se lleva a cabo la comprobación del apartado anterior. Para ello seguimos dos procedimientos:

- Realizamos un pago correcto, comprobando que el saldo disminuye:

10.2.5.2:8080/P1-ejb-transaccional-cliente/testbd.jsp
10.2.5.2:8080/P1-ejb-transaccional-cliente/procesapago

Pago con tarjeta

Proceso de un pago

Id Transacción:

Id Comercio:

Importe:

Numero de visa:

Titular:

Fecha Emisión:

Fecha Caducidad:

CVV2:

Modo debug: ☐ True ☐ False

Direct Connection: ☐ True ☐ False

Use Prepared: ☐ True ☐ False

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 100.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

3	0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/20	126	1000
4	0039 2818 1198 8592	Hugo Linus Sparrow	01/10	09/20	971	900
5	0060 4909 1343 5346	Pedro Cozar Martinez	02/10	05/20	187	1000

- Se realiza un pago repetido (el anterior), que devuelve error y comprobamos que el saldo no se ha modificado en la base de datos:

10.2.5.2:8080/P1-ejb-transaccional-cliente/procesapago

Pago con tarjeta

Pago incorrecto

3	0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/20	126	1000
4	0039 2818 1198 8592	Hugo Linus Sparrow	01/10	09/20	971	900
5	0060 4909 1343 5346	Pedro Cozar Martinez	02/10	05/20	187	1000

Se ve que no se ha vuelto a restar la cantidad de 100 al saldo de la tarjeta del usuario que ha efectuado el pago.

Ejercicio número 9:

A partir de este apartado se tiene el objetivo de poder administrar peticiones de manera asíncrona. Para ello se hará uso de la *JMS API* utilizando la funcionalidad de cola de mensajes. Por tanto accediendo al servidor de aplicaciones a través de *glashfish* se rellenan los siguientes campos (copiados del enunciado):

The screenshot shows the GlassFish Server Open Source Edition web console. The left sidebar contains a tree view with categories like Domain, Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources, and Configurations. The 'Resources' category is expanded, showing 'JMS Resources' and 'Connection Factories'. The main content area is titled 'New JMS Connection Factory' and contains the following settings:

General Settings

- JNDI Name:
- Resource Type:
- Description:
- Status: ☒ Enabled

Pool Settings

- Initial and Minimum Pool Size: Connections
Minimum and initial number of connections maintained in the pool
- Maximum Pool Size: Connections
Maximum number of connections that can be created to satisfy client requests
- Pool Resize Quantity: Connections
Number of connections to be removed when pool idle timeout expires
- Idle Timeout: Seconds
Maximum time that connection can remain idle in the pool
- Max Wait Time: Milliseconds
Amount of time caller waits before connection timeout is sent
- On Any Failure: ☐ Close All Connections
Close all connections and reconnect on failure, otherwise reconnect only when used
- Transaction Support: Level of transaction support. Overwrite the transaction support attribute in the Resource Adapter in a downward compatible way.
- Connection Validation: ☐ Required
Validate connections, allow server to reconnect in case of failure

Así se crea una *connection factory*, encargada de administrar las diferentes colas de mensaje o temas de mensajes asociados a nuestro servidor de aplicaciones.

Ejercicio número 10:

Para crear una cola de mensajes específica (*destination resource*) asociada a nuestro servidor de aplicaciones seguimos el enunciado:

The screenshot shows the GlassFish Server Open Source Edition web console. The left sidebar is the same as in the previous screenshot. The main content area is titled 'New JMS Destination Resource' and contains the following settings:

General Settings

- JNDI Name:
- Physical Destination Name:
Destination name in the Message Queue broker. If the destination does not exist, it will be created automatically when needed.
- Resource Type:
- Description:
- Status: ☒ Enabled

Additional Properties (0)

Ejercicio número 11:

Para proseguir con la actualización del servidor de aplicaciones se deben crear dos clases nuevas. Pero primero (tras haber descomprimido el paquete *P1-jms-base.tar*) se tiene que indicar el *connection factory* al que se va a conectar. Esto se consigue modificando el fichero *sun-ejb-jar.xml* introduciendo el nombre del recurso *jms* que acabamos de crear como *connection factory*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server 9.0 EJB 3.0//EN" "http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-0.dtd">
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>VisaCancelacionJMSBean</ejb-name>
      <mdb-connection-factory>
        <jndi-name>jms/VisaConnectionFactory</jndi-name>
      </mdb-connection-factory>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

Se han utilizado las líneas dadas por el enunciado.

En este apartado la clase que se crea es *VisaCancelacionJMSBean*. Además se implementan las *queries* que actualizan el código de respuesta de un pago cancelado y la que rectifican el saldo de la tarjeta que ha llevado a cabo el pago. Esto se hace conociendo la estructura de la base de datos:

```
private static final String UPDATE_CANCELA_QRY =
    "update pago " +
    "set codrespuesta=999 " +
    "where idautorizacion=?";

private static final String RECTIFICA_QRY =
    "update tarjeta " +
    "set saldo=saldo+pago.importe " +
    "from pago " +
    "where pago.idAutorizacion=? and tarjeta.numerotarjeta = pago.numerotarjeta";
```

Por último se codifica el método que se encarga de llamar a estas consultas(*onMessage()*):

```
public void onMessage(Message inMessage) {
    TextMessage msg = null;
    Connection con = null;

    try {
        if (inMessage instanceof TextMessage) {
            msg = (TextMessage) inMessage;
            logger.info("MESSAGE BEAN: Message received: " + msg.getText());
            int idautorizacion = Integer.parseInt(msg.getText());
            con = getConnection();

            PreparedStatement pstmt = con.prepareStatement(UPDATE_CANCELA_QRY);
            pstmt.setInt(1, idautorizacion);
            if (pstmt.execute() || pstmt.getUpdateCount() != 1) {
                logger.warning("Error en update cancelacion de pago");
            }
            pstmt.close();

            pstmt = con.prepareStatement(RECTIFICA_QRY);
            pstmt.setInt(1, idautorizacion);
            if (pstmt.execute() || pstmt.getUpdateCount() != 1) {
                logger.warning("Error en rectificacion saldo en cancelacion de pago");
            }
            pstmt.close();
        } else {
            logger.warning(
                "Message of wrong type: "
                + inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        e.printStackTrace();
        mdc.setRollbackOnly();
    } catch (Throwable te) {
        te.printStackTrace();
    }
}
```

Ejercicio número 12:

Para el acceso al recurso JMS estático al declarar los atributos de la clase *VisaQueueMessageProducer* se les añade las siguientes anotaciones:

```
@Resource(mappedName = "jms/VisaConnectionFactory")
private static ConnectionFactory connectionFactory;
@Resource(mappedName = "jms/VisaPagosQueue")
private static Queue queue;
```

Por otro lado, si se quiere conseguir este recurso de manera dinámica, se debe añadir en el método principal de la clase las siguientes líneas (se han dejado comentadas siguiendo las instrucciones del enunciado):

```
// InitialContext jndi = new InitialContext();
// connectionFactory = (ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");
// queue = (Queue)jndi.lookup("jms/VisaPagosQueue");
```

Ejercicio número 13:

Para automatizar la creación de recursos JMS se usan los ficheros *jms.xml* y *build.xml*. Para ello se debe modificar antes el fichero *jms.properties* y *build.properties* de la siguiente forma:

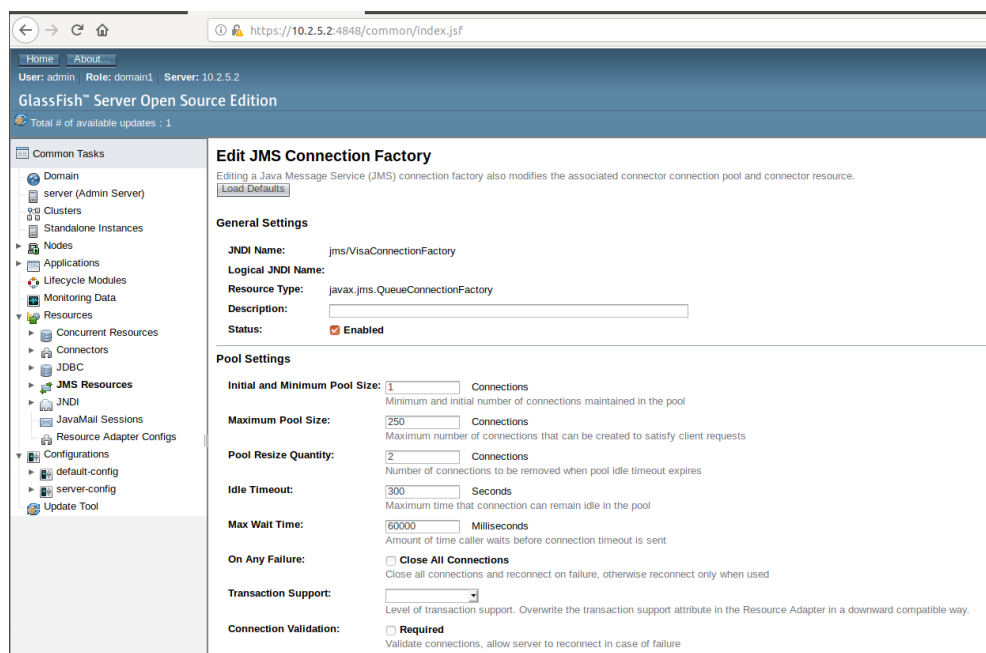
```
as.home=${env.J2EE_HOME}
as.lib=${as.home}/lib
as.user=admin
as.host.client=10.2.5.1
as.host.server=10.2.5.2
as.port=4848
as.passwordfile=${basedir}/passwordfile
as.target=server
jms.factoryname=jms/VisaConnectionFactory
jms.name=jms/VisaPagosQueue
jms.physname=VisaPagosQueue
```

```
as.host.mdb=10.2.5.2
```

Estas nuevas *ips* son necesarias porque indican donde se encuentran los recursos JMS. Además, siguiendo las indicaciones de los ejercicios 7 y 8, se han añadido los nombres de estos recursos en el fichero *jms.properties*.

Tras ello, se borran los recursos creados anteriormente de manera manual y se ejecuta el comando *ant todo*. Se puede comprobar que los recursos se han creado correctamente de manera dinámica:

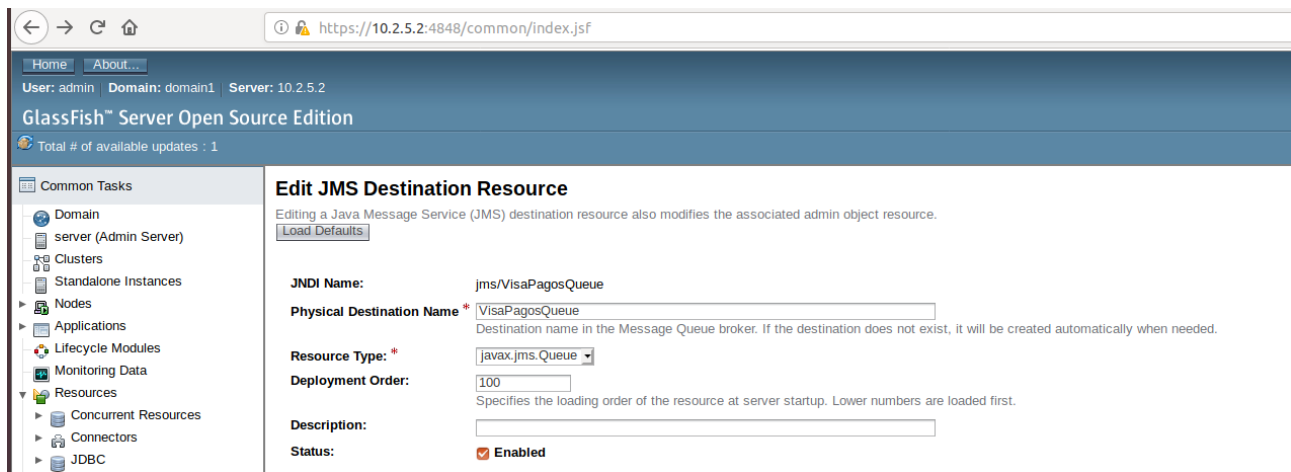
Connection factory:



The screenshot shows the GlassFish Server Open Source Edition web console. The left sidebar contains a tree view of the server's configuration, with 'JMS Resources' expanded. The main content area displays the 'Edit JMS Connection Factory' configuration page for the 'jms/VisaConnectionFactory' resource. The page includes a 'Load Defaults' button and a 'General Settings' section with fields for JNDI Name, Logical JNDI Name, Resource Type, Description, and Status. The 'Pool Settings' section includes fields for Initial and Minimum Pool Size, Maximum Pool Size, Pool Resize Quantity, Idle Timeout, Max Wait Time, On Any Failure, Transaction Support, and Connection Validation.

Section	Property	Value
General Settings	JNDI Name	jms/VisaConnectionFactory
	Logical JNDI Name	
	Resource Type	javax.jms.QueueConnectionFactory
	Description	
	Status	Enabled
Pool Settings	Initial and Minimum Pool Size	1 Connections
	Maximum Pool Size	250 Connections
	Pool Resize Quantity	2 Connections
	Idle Timeout	300 Seconds
	Max Wait Time	60000 Milliseconds
	On Any Failure	<input type="checkbox"/> Close All Connections
	Transaction Support	<input type="checkbox"/> Required
	Connection Validation	<input type="checkbox"/> Required

Cola de mensajes:



Tras revisar el fichero *jms.xml* se puede apreciar que el comando para crear los recursos *JMS* es:

```
<exec executable="${asadmin}">
  <arg line="--user ${as.user}" />
  <arg line="--passwordfile ${as.passwordfile}" />
  <arg line="--host ${as.host.server}" />
  <arg line="--port ${as.port}" />
  <arg line="create-jms-resource" />
  <arg line="--restype ${jms.restype}" />
  <arg line="--enabled=true" />
  <arg line="--property ${jms.resource.property}" />
  <arg line="--name ${jms.resource.name}" />
</exec>
```

Siendo *asadmin* el comando principal y cada *line* los argumentos de este comando. Los cuatro primero indican el servidor al que se asocia esta cola de mensajes. El comando que indica la creación del recurso es *create-jms-resource* seguido de los argumentos que especifican su tipo, propiedades y si está activo o no. Por último se indica el nombre del nuevo recurso.

Ejercicio número 14:

En este apartado se comprobará el correcto funcionamiento de la cola de mensajes. Para ello modificamos el método principal de la clase *VisaQueueMessageProducer*. Esta modificación consiste en conseguir el texto del mensaje recibido por argumento:

```
messageProducer = session.createProducer(queue);
message = session.createTextMessage();
message.setText(args[0]);
messageProducer.send(message);
messageProducer.close();
session.close();
```

Primero modificamos, a través del administrador de *glashfish* la variable mencionada en el enunciado. Después se ejecuta el primer comando tras reiniciar el servidor. Se comprueba el pago y se aprecia, con la ejecución del segundo comando que el pago con *idAutorizacion* con valor "12" se ha quedado en la cola de mensajes:

```
si2@si2srv01:~$ /opt/glassfish-4.1.2/glassfish/bin/appclient -targetserver 10.2.5.2 -client P1-jms-clientjms.jar 12
Mar 05, 2019 4:44:48 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 05, 2019 4:44:49 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
Mar 05, 2019 4:44:49 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
Mar 05, 2019 4:44:49 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
si2@si2srv01:~$ /opt/glassfish-4.1.2/glassfish/bin/appclient -targetserver 10.2.5.2 -client P1-jms-clientjms.jar -browse
Mar 05, 2019 4:45:39 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
Mar 05, 2019 4:45:40 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version: 5.1.1 (Build 2-c) Compile: March 17 2015 1045
Mar 05, 2019 4:45:40 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection mode is TCP
Mar 05, 2019 4:45:40 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFO: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensajes en cola:
12
```

A continuación se vuelve a activar el *MDB* y se realiza un pago con la aplicación transaccional:

10.2.5.2:8080/P1-ejb-transaccional-cliente/procesapago

Pago con tarjeta

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

idTransaccion: 12
idComercio: 12
importe: 100.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

Se comprueba que en la base de datos el saldo ha disminuido y aparece el pago:

3	0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/20	126	1000
4	0039 2818 1198 8592	Hugo Linus Sparrow	01/10	09/20	971	900
5	0060 4909 1343 5346	Pedro Cozar Martinez	02/10	05/20	187	1000

	[PK] serial	character(16)	character(3)	double precision	character(16)	character(19)	timestamp without time zone
1	1	12	000	100	12	0039 2818 1198 8592	2019-03-05 16:55:27.553615
*							

Se cancela el pago y vemos que el saldo vuelve al original:

3	0029 0099 6642 8003	Enjuto Vallejo Coll	03/09	10/20	126	1000
4	0039 2818 1198 8592	Hugo Linus Sparrow	01/10	09/20	971	1000
5	0060 4909 1343 5346	Pedro Cozar Martinez	02/10	05/20	187	1000

Por último se comprueba que el código de respuesta del pago ha pasado a 999:

	[PK] serial	character(16)	character(3)	double precision	character(16)	character(19)	timestamp without time zone
1	1	12	999	100	12	0039 2818 1198 8592	2019-03-05 16:55:27.553615
*							