UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior** **Ingeniería Informática** **Prácticas de Sistemas Informáticos 2**			
Grupo	**2401**	**Práctica**	1b	**Fecha**	26/02/2019
Alumno/a		Cea Tassel, Claudia			
Alumno/a		Riera Gómez, Juan			

Práctica 1: Arquitectura de JAVA EE (Segunda parte)

Cuestión número 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?

La librería Java EE importada en el archivo VisaDAOLocal.java es la librería “Local” (import javax.ejb.Local). En este fichero también se utiliza la anotación @Local, la cual indica que la EJB correspondiente se accede desde la misma máquina virtual.

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**

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

```
@Stateless(mappedName="VisaDAOBean")
public class VisaDAOBean extends DBTester implements VisaDAOLocal {
```

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

  Hemos eliminado el constructor por defecto de la clase.

- **Ajustar los métodos getPagos() a la interfaz definida en VisaDAOLocal**

```
@WebMethod(operationName = "getPagos")
public public PagoBean[] getPagos(@WebParam(name = "idComercio")String idComercio) {
```

```
    //return new ArrayList<PagoBean>(Arrays.asList(ret));
    return ret
```

Ejercicio número 2:

Modificar el servlet ProcesaPago para que acceda al EJB local.

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

```
/*try{
    VisaDAOWSService service = new VisaDAOWSService();
    dao = service.getVisaDAOWSPort ();
    BindingProvider bp = (BindingProvider) dao;
    bp.getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
        getServletContext().getInitParameter("serverAddress"));
}
catch (javax.xml.ws.WebServiceException e){
    e.printStackTrace();
}*/
```

Hemos reemplazado toda la declaración anterior y toda la apertura del servicio por la declaración de la variable con la instrucción anterior

Lo mismo hemos hecho en getPagos(), donde antes llamábamos a toArray de la respuesta del servidor esperando que nos enviara un ArrayList, y ahora simplemente tomamos el array de vuelta:

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

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

También lo hemos hecho en delPagos():

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

Cuestión número 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).

El archivo application.xml es el descriptor de despliegue de la aplicación, el cual describe cada uno de los módulos de la aplicación. La etiqueta <application> es el elemento raíz del descriptor. La etiqueta <display-name> indica el nombre que queremos que aparezca como nombre de la aplicación. A continuación mediante la etiqueta <module> se indican cuáles son los módulos de la aplicación. En este caso, la aplicación cuenta con 2 módulos: un módulo de tipo EJB indicado con la etiqueta <ejb>, que es P1-ejb.jar y un módulo de tipo web indicado con la etiqueta <web> cuya URI es P1-ejb-cliente.war y cuya raíz es P1-ejc-cliente. Estos datos específicos del segundo módulo se indican con sus respectivas etiquetas <web-uri> y <context-root>.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<application version="5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/jav
  <display-name>P1-ejb</display-name>
  <module>
    <ejb>P1-ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>P1-ejb-cliente.war</web-uri>
      <context-root>/P1-ejb-cliente</context-root>
    </web>
  </module>
</application>
```

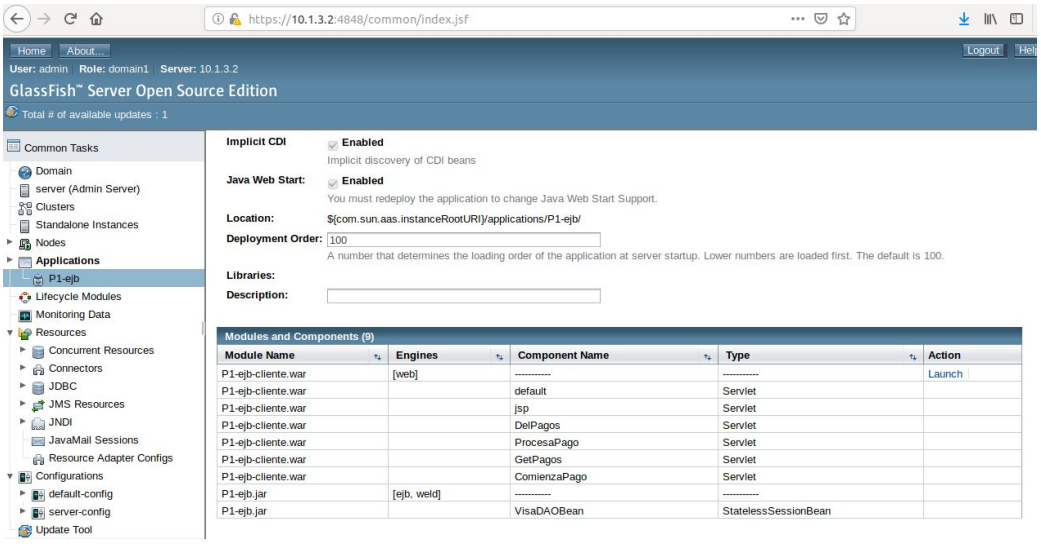
El único archivo .jar construido hasta el momento es el archivo postgresql-jdbc4.jar, el cual contiene todas las clases de la aplicación referentes a postgresql. Para ver el contenido de este archivo jar hemos utilizado el comando “jar -tvf postgresql-jdbc4.jar”.

Ejercicio número 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 por qué 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 por qué son esos valores.**
- **Desplegar la aplicación de empresa**

Tras editar los ficheros build.properties y postgresql.properties desplegamos la aplicación. Podemos ver que en este caso, la aplicación no se ha desplegado, como en las prácticas anteriores, como Web Application, sino que se encuentra bajo el apartado de Enterprise Applications:



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. 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.

En primer lugar, compilamos y desplegamos la aplicación. A continuación accedemos a testbd.jsp, rellenamos el formulario con los datos de un cliente válido y realizamos el pago:

El pago ha sido realizado con éxito:

Listamos los pagos realizados al comercio con ID 5. Observamos que aparece la transacción que acabamos de realizar, demostrando así que, efectivamente, el pago fue exitoso:

Pago con tarjeta

Lista de pagos del comercio 5

idTransaccion	Importe	codRespuesta	idAutorizacion
5	5.0	000	1

Volver al comercio

Prácticas de Sistemas Informáticos II

Probamos ahora a borrar las transacciones del comercio con ID 5:

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 5

Volver al comercio

Prácticas de Sistemas Informáticos II

Ahora intentaremos mostrar los pagos asociados a ese comercio para ver que efectivamente se han borrado. No aparece ningún pago, luego se han borrado correctamente:

Pago con tarjeta

Lista de pagos del comercio 5

idTransaccion	Importe	codRespuesta	idAutorizacion

Volver al comercio

Prácticas de Sistemas Informáticos II

Ahora queremos realizar un pago pero desde pago.html. Comenzamos rellenando los datos asociados al comercio y a la transacción:

A continuación, rellenamos el formulario con los datos de un cliente válido:

Nos aparece la confirmación del pago:

Comprobamos desde testbd.jsp que efectivamente el pago se ha realizado correctamente

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.

Declaramos VisaDAORemote con la etiqueta @Remote, tras haber importado javax.ejb.Remote**:**

```
import javax.ejb.Remote;

@Remote
public interface VisaDAORemote {
```

Ahora la clase VisaDAOBean implementa también la interfaz VisaDAORemote:

```
@Stateless(mappedName="VisaDAOBean")
public class VisaDAOBean extends DBTester implements VisaDAOLocal, VisaDAORemote {
```

Importamos la interfaz Serializable y hacemos que PagoBean la implemente, convirtiéndose en un objeto serializable:

```
import java.io.Serializable;

public class PagoBean implements Serializable {
```

Hacemos lo mismo con TarjetaBean:

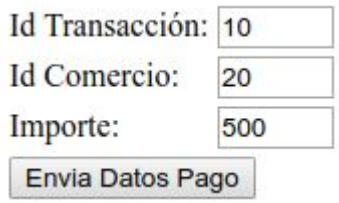
```
import java.io.Serializable;

public class TarjetaBean implements Serializable {
```

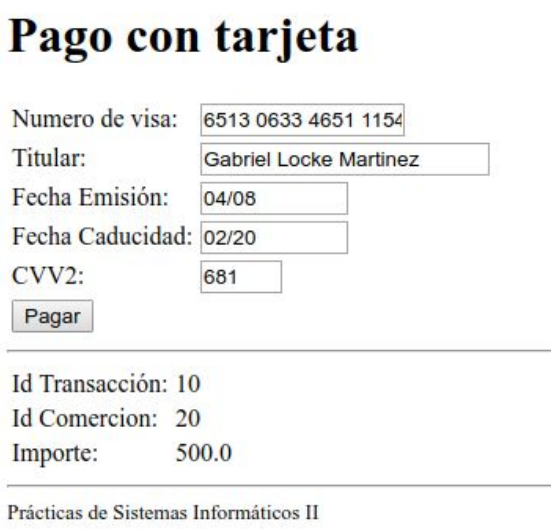
Ejercicio número 6:

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 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.

Ya compilada, empaquetada y desplegada la aplicación vamos a probar a realizar un pago. Para ello entramos en la página pago.html y rellenamos el formulario:



Continuamos rellenando el formulario de pago tomando los datos de un usuario válido:



Se nos muestra la página de pago con éxito. Y tras entrar en testbd.jsp comprobamos que efectivamente se ha realizado con éxito:

Ejercicio número 7:

Modificarla aplicación VISA para soportar el campo saldo.

Empezamos añadiendo los prepared statements SELECT_SALDO_QRY y UPDATE_SALDO_QRY para la obtención y la actualización del saldo de una tarjeta:

```
private static final String SELECT_SALDO_QRY =
    "select saldo from tarjeta " +
    "where numeroTarjeta=? ";

private static final String UPDATE_SALDO_QRY =
    "update tarjeta set saldo=? " +
    "where numeroTarjeta=?";
/**
```

En el método realizarPago añadimos la comprobación de que la tarjeta tiene saldo suficiente para realizar el pago. Si no lo tiene, devuelve un código de respuesta determinado para indicar esa falta de saldo:

```
    /*************************************************/
    /* Vamos a comprobar si hay saldo disponible */
    String saldos  = SELECT_SALDO_QRY;
    errorLog(saldos);
    pstmt = con.prepareStatement(saldos);
    pstmt.setString(1, pago.getTarjeta().getNumero());
    rs = pstmt.executeQuery();
    Double saldo = 0.0;
    if(rs.next()) {
        saldo = rs.getDouble("saldo");
        pstmt.close();
    } else {
        pago.setIdAutorizacion(null);
        pstmt.close();
        pago.setCodRespuesta("997");
        return pago;
    }
    if (saldo<pago.getImporte()){
        pago.setIdAutorizacion(null);
        pstmt.close();
        pago.setCodRespuesta("997");
        return pago;
    }
```

Por último, al final del proceso de pago, añadimos la actualización de pago. Pensamos que este es el punto del código en el que se debe realizar, ya que solo debemos quitar dinero de la tarjeta cuando estemos seguros de que el pago se completa correctamente, es decir, al llegar a este punto sin errores:

```
String insertSaldo  = UPDATE_SALDO_QRY;
errorLog(insertSaldo);
pstmt = con.prepareStatement(insertSaldo);
pstmt.setDouble(1, saldo-pago.getImporte());
pstmt.setString(2, pago.getTarjeta().getNumero());
if(pstmt.execute() || pstmt.getUpdateCount() != 1){
    pstmt.close();
    pago.setIdAutorizacion(null);
    pago.setCodRespuesta("996");
    return pago;
}
pstmt.close();      You, a few seconds ago • Uncommit
```

Ejercicio número 8:

Desplegar y probar la nueva aplicación creada.

Vamos a comenzar realizando un pago correctamente. Para ello comprobamos el saldo inicial de la tarjeta que vamos a utilizar, que debería ser 1000:

```
visa=# select * from tarjeta where numeroTarjeta = '7557 9649 3955 5976';
    numerotarjeta    |       titular       | validadesde | validahasta | codigoverificacion | saldo
---------------------+---------------------+-------------+-------------+--------------------+-------
 7557 9649 3955 5976 | Irene Avila Morales | 01/10       | 03/20       | 185                |  1000
(1 row)
```

Continuamos entrando en testbd.jsp y rellenando el formulario de pago:

Pago con tarjeta

Proceso de un pago

Id Transacción:	1
Id Comercio:	1
Importe:	1
Numero de visa:	7557 9649 3955 5976
Titular:	Irene Avila Morales
Fecha Emisión:	01/10
Fecha Caducidad:	03/20
CVV2:	185
Modo debug:	● True ○ False
Direct Connection:	○ True ● False
Use Prepared:	○ True ● False

Pagar

Tras haber rellenado el formulario realizamos el pago y recibimos un mensaje de éxito:

Finalmente comprobamos que el saldo de la tarjeta se ha visto reducido en una unidad:

```
visa=# select * from tarjeta where numeroTarjeta = '7557 9649 3955 5976';
    numerotarjeta    |       titular       | validadesde | validahasta | codigoverificacion | saldo
---------------------+---------------------+-------------+-------------+--------------------+-------
 7557 9649 3955 5976 | Irene Avila Morales | 01/10       | 03/20       | 185                |   999
(1 row)
```

Efectivamente, el saldo actual es el esperado: 999. Ahora intentemos realizar un pago repitiendo el mismo id de comercio e id de pago:

Tras obtener el mensaje de error, volvemos a mirar la tarjeta a ver si se ha restado el precio dos veces. En la siguiente captura vemos las tres consultas, las dos anteriores y la última recién mencionada. Como observamos, el saldo no se ha reducido y se ha mantenido en 999:

```
visa=# select * from tarjeta where numeroTarjeta = '7557 9649 3955 5976';
    numerotarjeta    |       titular       | validadesde | validahasta | codigoverificacion | saldo
---------------------+---------------------+-------------+-------------+--------------------+-------
 7557 9649 3955 5976 | Irene Avila Morales | 01/10       | 03/20       | 185                |  1000
(1 row)

visa=# select * from tarjeta where numeroTarjeta = '7557 9649 3955 5976';
    numerotarjeta    |       titular       | validadesde | validahasta | codigoverificacion | saldo
---------------------+---------------------+-------------+-------------+--------------------+-------
 7557 9649 3955 5976 | Irene Avila Morales | 01/10       | 03/20       | 185                |   999
(1 row)

visa=# select * from tarjeta where numeroTarjeta = '7557 9649 3955 5976';
    numerotarjeta    |       titular       | validadesde | validahasta | codigoverificacion | saldo
---------------------+---------------------+-------------+-------------+--------------------+-------
 7557 9649 3955 5976 | Irene Avila Morales | 01/10       | 03/20       | 185                |   999
(1 row)
```

Ahora intentemos realizar un pago superior al saldo del usuario. El saldo es de 999 y estamos intentando realizar un pago de 1000:

Pago con tarjeta

Proceso de un pago

Id Transacción: 2
Id Comercio: 2
Importe: 1000
Numero de visa: 7557 9649 3955 5976
Titular: Irene Avila Morales
Fecha Emisión: 01/10
Fecha Caducidad: 03/20
CVV2: 185
Modo debug: (•) True ( ) False
Direct Connection: ( ) True (•) False
Use Prepared: ( ) True (•) False
Pagar

Efectivamente recibimos un mensaje de error:

Pago con tarjeta

Pago incorrecto 997

Prácticas de Sistemas Informáticos II

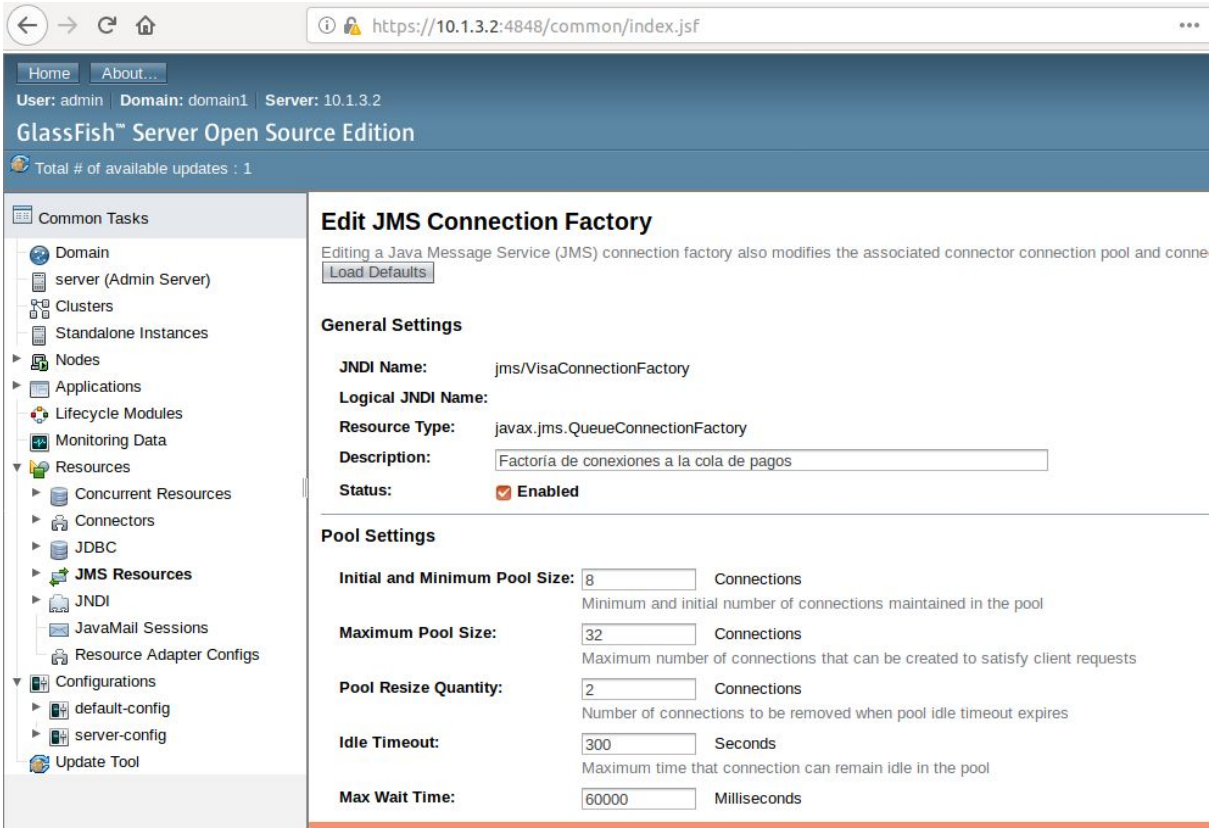
Y además comprobamos que el saldo no se ha visto afectado. Sigue siendo el esperado, 999:

```
visa=# select * from tarjeta where numeroTarjeta = '7557 9649 3955 5976';
    numerotarjeta    |       titular       | validadesde | validahasta | codigoverificacion | saldo
---------------------+---------------------+-------------+-------------+--------------------+-------
 7557 9649 3955 5976 | Irene Avila Morales | 01/10       | 03/20       | 185                |   999
(1 row)
```

Ejercicio número 9:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.2), declare manualmente la factoría de conexiones empleando la consola de administración.

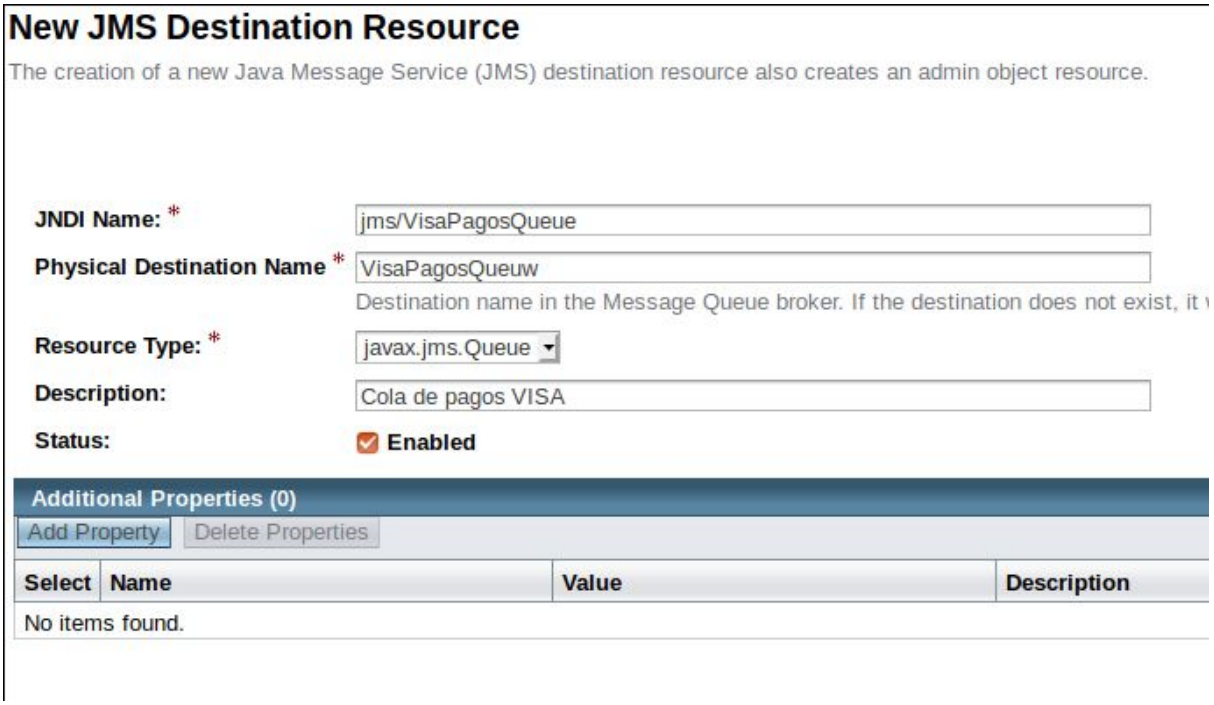
Hemos declarado manualmente la factoría de conexiones con los datos mencionados en el enunciado:

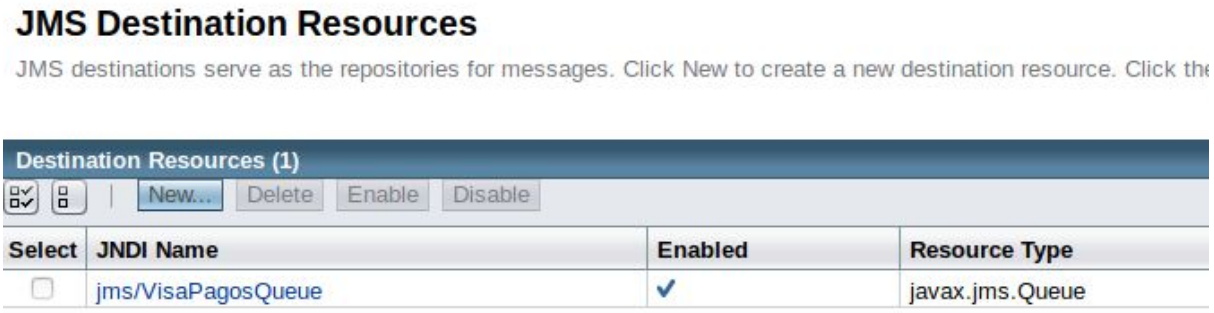


Ejercicio número 10:

En la máquina virtual donde se encuentra el servidor de aplicaciones (10.X.Y.2), declare manualmente la conexión empleando la consola de administración.

Hemos declarado manualmente la conexión con los datos mencionados en el enunciado:





Ejercicio número 11:

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

Añadimos al fichero sun-ejb-jar.xml el nombre de la factoría de conexiones que en nuestro caso es jms/VisaConnectionFactory:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Application Ser
<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>
```

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.

Declaramos el prepared statement UPDATE_CANCELA_QRY que se encarga de actulizar ambos, el código de respuesta y el saldo de la tarjeta que realizó el pago:

```
private static final String UPDATE_CANCELA_QRY = "update tarjeta set saldo = saldo + pago.importe from pago " +
"where pago.numerotarjeta=tarjeta.numerotarjeta and idautorizacion=? and codrespuesta = \'000\';"+
"UPDATE pago " +
  "set codrespuesta = \'999\' where idautorizacion = ? and codrespuesta = \'000\'; ";
```

Método onMessage() que implemente ambas actualizaciones. 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. Control de errores en el método onMessage.

Aquí mostramos todas las modificaciones realizadas al método tomando como ejemplo el código SQL de ejercicios anteriores:

```
public void cancelarPago(int id) {
  PreparedStatement pstmt = null;
  ResultSet rs = null;
  Connection con=null;
  try {
      con = getConnection();
      pstmt = con.prepareStatement(UPDATE_CANCELA_QRY);
      pstmt.setInt(1, id);
      pstmt.setInt(2, id);
      rs = pstmt.executeQuery();
      if (!rs.next()){
          logger.warning("The cancelation query did not work correctly");
      }
  } catch (Exception e) {
      e.printStackTrace();
      logger.warning(e.toString());
  } finally {
      try {
          if (rs != null) {
              rs.close(); rs = null;
          }
          if (pstmt != null) {
              pstmt.close(); pstmt = null;
          }
          if (con != null) {
              closeConnection(con);
              con = null;
          }
      } catch (SQLException e) {
          e.printStackTrace();
          logger.warning(e.toString());
      }
  }
}
```

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 qué ventajas podrían tener uno u otro método.

En primer lugar implementamos el método que utiliza anotaciones JMS estáticas:

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

A continuación, implementamos el método que utiliza la búsqueda dinámica del recurso mediante la API de JNDI y lo dejamos comentado tal y como indica el enunciado:

```
//InitialContext jndi = new InitialContext();
//connectionFactory = (ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");
//queue = (Queue)jndi.lookup("jms/VisaPagosQueue");      You, a few seconds ago • U
// Método de prueba
```

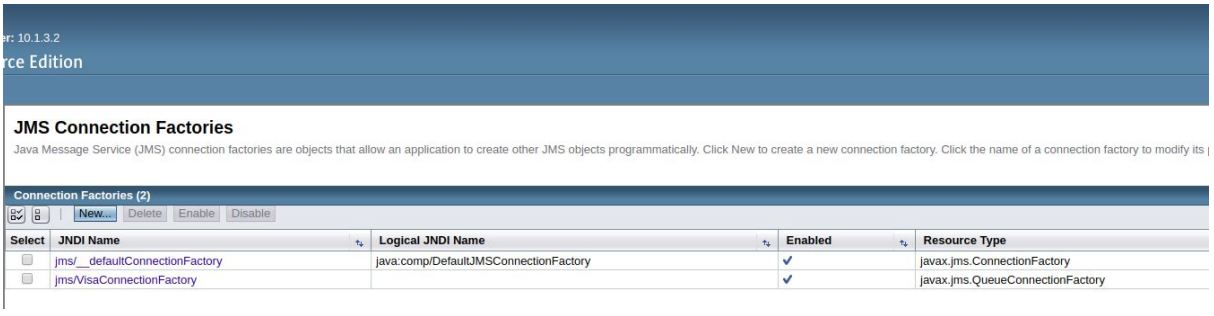
JMS mejora el rendimiento ya que se lleva a cabo en tiempo de compilación. Una desventaja que tiene es que requiere saber el nombre del recurso de antemano. Por otro lado, JNDI es más flexible a la hora de buscar el recurso, pues proporciona transparencia de ubicación.

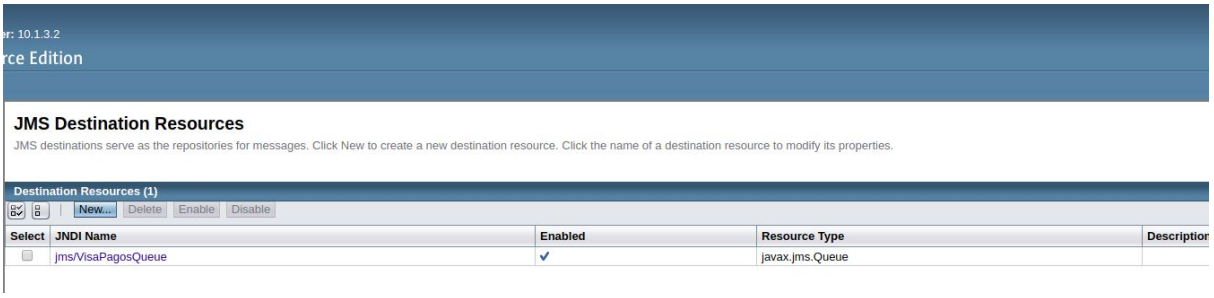
Ejercicio número 13:

Automatice la creación de los recursos JMS (cola y factoría de conexiones) en el build.xml y jms.xml. 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?

Para automatizar la creación de los recursos JMS indicamos en jms.properties los nombres de la cola y factoría de conexiones. También debemos asignar las direcciones IP adecuadas a las variables as.host.mdb y as.host.server en los ficheros build.properties y jms.properties respectivamente. Esta dirección IP es, en nuestro caso, 10.1.3.2 que indica en el fichero build.properties cual es el modelo de la cola y en el fichero jms.properties donde se encuentra la cola.

Comprobamos que al ejecutar el comando ant todo efectivamente, los recursos se han creado automáticamente:





Ejercicio número 14:

Modifique el cliente, VisaQueueMessageProducer.java, implementando el envío de args[0] como mensaje de texto.

Así hemos modificado el cliente, VisaQueueMessageProducer.java, para que envíe args[0] como mensaje de texto:

```
} else {
   // TODO: Enviar args[0] como mensaje de texto
   messageProducer = session.createProducer(queue);
   message = session.createTextMessage();
   message.setText(args[0]);
   messageProducer.send(message);
   messageProducer.close();
   session.close();
}
```

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

Ejecutamos el cliente con el comando indicado:

```
e336799@1-32-64-81:~/si2claujuani/p2/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -target
server 10.1.3.2 -client dist/clientjms/P1-jms-clientjms.jar idAutorizacion
mar 11, 2019 3:50:28 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 11, 2019 3:50:28 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1.1  (Build 2-c)
 Compile:  March 17 2015 1045
mar 11, 2019 3:50:28 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, c
onnection mode is TCP
mar 11, 2019 3:50:28 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: idAutorizacion
```

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

Verificamos el contenido de la cola con el comando indicado:

```
e336799@1-32-64-81:~/si2claujuani/p2/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -target
server 10.1.3.2 -client dist/clientjms/P1-jms-clientjms.jar -browse
mar 11, 2019 3:53:49 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 11, 2019 3:53:50 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1.1  (Build 2-c)
 Compile:  March 17 2015 1045
mar 11, 2019 3:53:50 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, c
onnection mode is TCP
mar 11, 2019 3:53:50 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Mensajes en cola:
idAutorizacion
```

A continuación, volver a habilitar la ejecución del MDB y realizar los siguientes pasos:
Realice un pago con la aplicación web.

Empezamos comprobando que el saldo inicial de un cliente es 1000:

```
    numerotarjeta    |        titular         | validadesde | validahasta | codigoverificacion | saldo
---------------------+------------------------+-------------+-------------+--------------------+-------
 6513 0633 4651 1154 | Gabriel Locke Martinez | 04/08       | 02/20       | 681                |  1000
(1 row)
```

Ahora procedemos a realizar un pago de un importe de 9€ con los mismos datos de usuario cuyo saldo acabamos de comprobar:

Pago con tarjeta

Proceso de un pago

Id Transacción:	9
Id Comercio:	9
Importe:	9
Numero de visa:	6513 0633 4651 1154
Titular:	Gabriel Locke Martinez
Fecha Emisión:	04/08
Fecha Caducidad:	02/20
CVV2:	681
Modo debug:	● True ○ False
Direct Connection:	○ True ● False
Use Prepared:	○ True ● False

Pagar

Obtenga evidencias de que se ha realizado.

Comprobamos que se ha realizado el pago buscando todos los pagos cuyo id de comercio sea 9:

Pago con tarjeta

Lista de pagos del comercio 9

idTransaccion	Importe	codRespuesta	idAutorizacion
9	9.0	000	1

Volver al comercio

Prácticas de Sistemas Informáticos II

Además, comprobamos que el saldo del cliente ha disminuido en 9 unidades, que es el importe del pago que hemos realizado. Su saldo actual debería ser 991:

```
    numerotarjeta    |        titular         | validadesde | validahasta | codigoverificacion | saldo
---------------------+------------------------+-------------+-------------+--------------------+-------
 6513 0633 4651 1154 | Gabriel Locke Martinez | 04/08       | 02/20       | 681                |   991
(1 row)
```

Cancélelo con el cliente.

Procedemos a la cancelación del pago con idAutorización 1:

```
e336799@1-32-64-81:~/si2claujuani/p2/P1-jms$ /opt/glassfish4/glassfish/bin/appclient -targetserver 10.1
.3.2 -client dist/clientjms/P1-jms-clientjms.jar 1
mar 11, 2019 4:09:59 PM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 5.1.2.Final
mar 11, 2019 4:09:59 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter: Version:  5.1.1  (Build 2-c) Compile:
March 17 2015 1045
mar 11, 2019 4:09:59 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter starting: broker is REMOTE, connection m
ode is TCP
mar 11, 2019 4:09:59 PM com.sun.messaging.jms.ra.ResourceAdapter start
INFORMACIÓN: MQJMSRA_RA1101: GlassFish MQ JMS Resource Adapter Started:REMOTE
Enviando el siguiente mensaje: 1
```

Obtenga evidencias de que se ha cancelado y de que el saldo se ha rectificado.

Comprobamos que el pago se ha cancelado. Ahora el código de respuesta ha cambiado a 999:

Pago con tarjeta

Lista de pagos del comercio 9

idTransaccion	Importe	codRespuesta	idAutorizacion
9	9.0	999	1

Volver al comercio

Prácticas de Sistemas Informáticos II

Finalmente, comprobamos que el saldo del usuario vuelve a ser 1000:

```
visa=# select * from tarjeta where numerotarjeta = '6513 0633 4651 1154';
    numerotarjeta    |        titular         | validadesde | validahasta | codigoverificacion | saldo
---------------------+------------------------+-------------+-------------+--------------------+-------
 6513 0633 4651 1154 | Gabriel Locke Martinez | 04/08       | 02/20       | 681                |  1000
(1 row)
```