		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2321	Práctica	1A	Fecha	23/03/2021
Alumno/a		Arribas Gozalo, Francisco Javier			
Alumno/a		Sáenz Ferrero, Santos			

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

Ejercicio 1:

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

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

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

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

```

25 import javax.ejb.Stateless;
26
27 /**
28  * @author jaime
29  */
30
31 // @WebService()
32 @Stateless (mappedName = "VisaDAOBean")
33 public class VisaDAOBean extends DBTester implements VisaDAOLocal {
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83 /**
84  * Constructor de la clase
85  */
86 // public VisaDAOBean() {
87 //     return;
88 // }

```

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

```

332 public PagoBean[] getPagos
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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

Ejercicio 2:

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

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

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

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

Este proceso es el mismo para todos los archivos.

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

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

Ahora añadiremos la interfaz local a las clases:

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

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

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

Comentamos el código con referencias a `BindingProvider`:

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

Ejercicio 3:

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

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

Desplegar la aplicación de empresa

`ant desplegar`

En postgresql.properties, usaremos las siguientes IPs:

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

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

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

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

El despliegue de la aplicación se ha realizado correctamente en 10.7.8.2:

The screenshot shows the GlassFish Server Open Source Edition web console. The 'Edit Application' page for 'P1-ejb' is displayed. The page includes a sidebar with a tree view of the server structure. The main content area shows the configuration for the application. The 'Modules and Components (9)' table is as follows:

Module Name	Engines	Component Name	Type	Action
P1-ejb-cliente.war	[web]	default	Servlet	Launch
P1-ejb-cliente.war		jsp	Servlet	
P1-ejb-cliente.war		DelPagos	Servlet	
P1-ejb-cliente.war		ProcesaPago	Servlet	
P1-ejb-cliente.war		GetPagos	Servlet	
P1-ejb-cliente.war		ComienzaPago	Servlet	
P1-ejb.jar	[ejb, weld]			
P1-ejb.jar		VisaDAOBean	StatelessSessionBean	

Ejercicio 4:

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

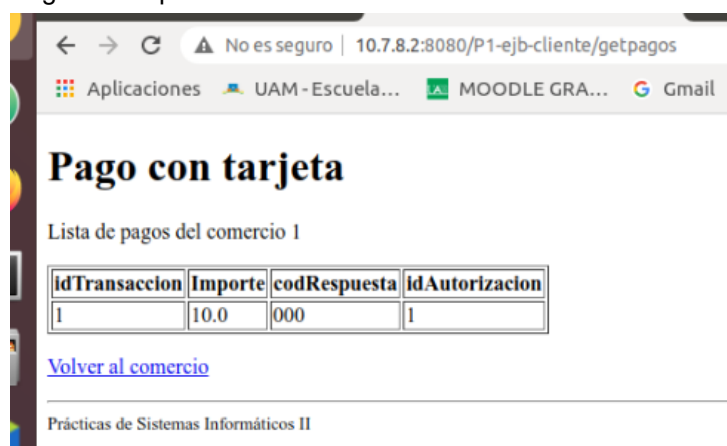
Incluya en la memoria de prácticas todos los pasos necesarios para resolver este ejercicio así como las evidencias obtenidas. Se pueden incluir por ejemplo capturas de pantalla.

Con la aplicación desplegada, hemos hecho las comprobaciones necesarias:

Pago con tarjeta:



Pago en la aplicación:



Pago borrado:



Ejercicio 5:

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

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

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

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

Además, hemos hecho que la clase VisaDAOBean implemente VisaDAORemote

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

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

```
9 import java.io.Serializable;
10
11 public class TarjetaBean implements Serializable {
12
13     // ...
14
15     /*
16     public class PagoBean implements Serializable {
17
```

Ejercicio 6:

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

P1-ejb-cliente-remoto desplegada:

The screenshot shows the GlassFish Server Open Source Edition web console. The left sidebar displays the application hierarchy, with 'P1-ejb-cliente-remoto' selected under the 'P1' application. The main area shows the 'Edit Application' configuration for 'P1-ejb-cliente-remoto'. The configuration includes the following details:

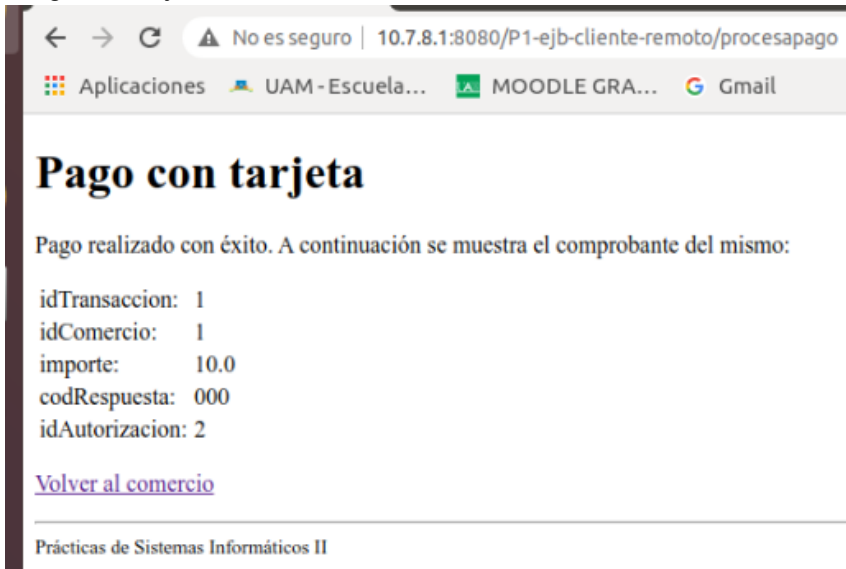
- Name:** P1-ejb-cliente-remoto
- Status:** Enabled
- Virtual Servers:** server
- Context Root:** /P1-ejb-cliente-remoto
- Implicit CDI:** Enabled
- Location:** \${com.sun.aas.instanceRootURL}/applications/P1-ejb-cliente-remoto/
- Deployment Order:** 100
- Libraries:** (empty)
- Description:** (empty)

Below the configuration, a table titled 'Modules and Components (7)' lists the components of the application:

Module Name	Engines	Component Name	Type	Action
P1-ejb-cliente-remoto	[web]	default	Servlet	Launch
P1-ejb-cliente-remoto		jsp	Servlet	
P1-ejb-cliente-remoto		DeiPagos	Servlet	
P1-ejb-cliente-remoto		ProcesaPago	Servlet	
P1-ejb-cliente-remoto		GetPagos	Servlet	
P1-ejb-cliente-remoto		ComienzaPago	Servlet	

Con la aplicación desplegada, hemos hecho las comprobaciones necesarias:

Pago con tarjeta:



The screenshot shows a web browser window with the address bar displaying "10.7.8.1:8080/P1-ejb-cliente-remoto/procesapago". The page title is "Pago con tarjeta". The main content area displays the message "Pago realizado con éxito. A continuación se muestra el comprobante del mismo:" followed by transaction details: idTransaccion: 1, idComercio: 1, importe: 10.0, codRespuesta: 000, and idAutorizacion: 2. A link "Volver al comercio" is provided at the bottom. The footer indicates "Prácticas de Sistemas Informáticos II".

Pago con tarjeta

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

idTransaccion: 1
idComercio: 1
importe: 10.0
codRespuesta: 000
idAutorizacion: 2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago en la aplicación:



The screenshot shows a web browser window with the address bar displaying "10.7.8.1:8080/P1-ejb-cliente-remoto/getpagos". The page title is "Pago con tarjeta". The main content area displays the message "Lista de pagos del comercio 1" followed by a table with the following data:

IdTransaccion	Importe	codRespuesta	IdAutorizacion
1	10.0	000	2

A link "Volver al comercio" is provided below the table. The footer indicates "Prácticas de Sistemas Informáticos II".

Pago con tarjeta

Lista de pagos del comercio 1

IdTransaccion	Importe	codRespuesta	IdAutorizacion
1	10.0	000	2

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Pago borrado:



The screenshot shows a web browser window with the address bar displaying "10.7.8.1:8080/P1-ejb-cliente-remoto/delpagos". The page title is "Pago con tarjeta". The main content area displays the message "Se han borrado 1 pagos correctamente para el comercio 1". A link "Volver al comercio" is provided at the bottom. The footer indicates "Prácticas de Sistemas Informáticos II".

Pago con tarjeta

Se han borrado 1 pagos correctamente para el comercio 1

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Ejercicio 7:

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

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

```
16      private double saldo;

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

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

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

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

```
245      // Registrar el pago en la base de datos
246      try {
247
248          // Obtener conexion
249          con = getConnection();
250
251          // Recuperar el saldo de la tarjeta a través del prepared statement declarado anteriormente
252          String aux_saldo = GET_SALDO_QRY;
253          errorLog(aux_saldo);
254          pstmt = con.prepareStatement(aux_saldo);
255          pstmt.setString(1, pago.getTarjeta().getNumero());
256          rs = pstmt.executeQuery();
257
258          if (rs.next()) { // Query correcta
259              saldo = rs.getDouble("saldo");
260              importe = pago.getImporte();
261
262              // Comprobar si el saldo es mayor o igual que el importe de la operación.
263              // Si no lo es, retornar denegando el pago (idAutorizacion= null y pago retornado=null)
264              if (saldo < importe)
265              {
266                  pago.setIdAutorizacion(null);
267                  return null;
268              } else {
269                  // Si el saldo es suficiente, decrementarlo en el valor del importe del pago y actualizar el registro
270                  // de la tarjeta para reflejar el nuevo saldo mediante el prepared statement declarado anteriormente.
271                  saldo = saldo - importe;
272                  String aux_saldo2 = SET_SALDO_QRY;
273                  errorLog(aux_saldo2);
274                  pstmt = con.prepareStatement(aux_saldo2);
275                  pstmt.setDouble(1, saldo);
276                  pstmt.setString(2, pago.getTarjeta().getNumero());
277                  if (!pstmt.execute()
278                      && pstmt.getUpdateCount() == 1){
279                      throw new EJBException("Error al actualizar el saldo");
280                  }
281              }
282          } else { // Excepcion
283              throw new EJBException("Error al obtener el saldo");
284          }
285      }
```

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

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

Ejercicio 8:

Desplegar y probar la nueva aplicación creada.

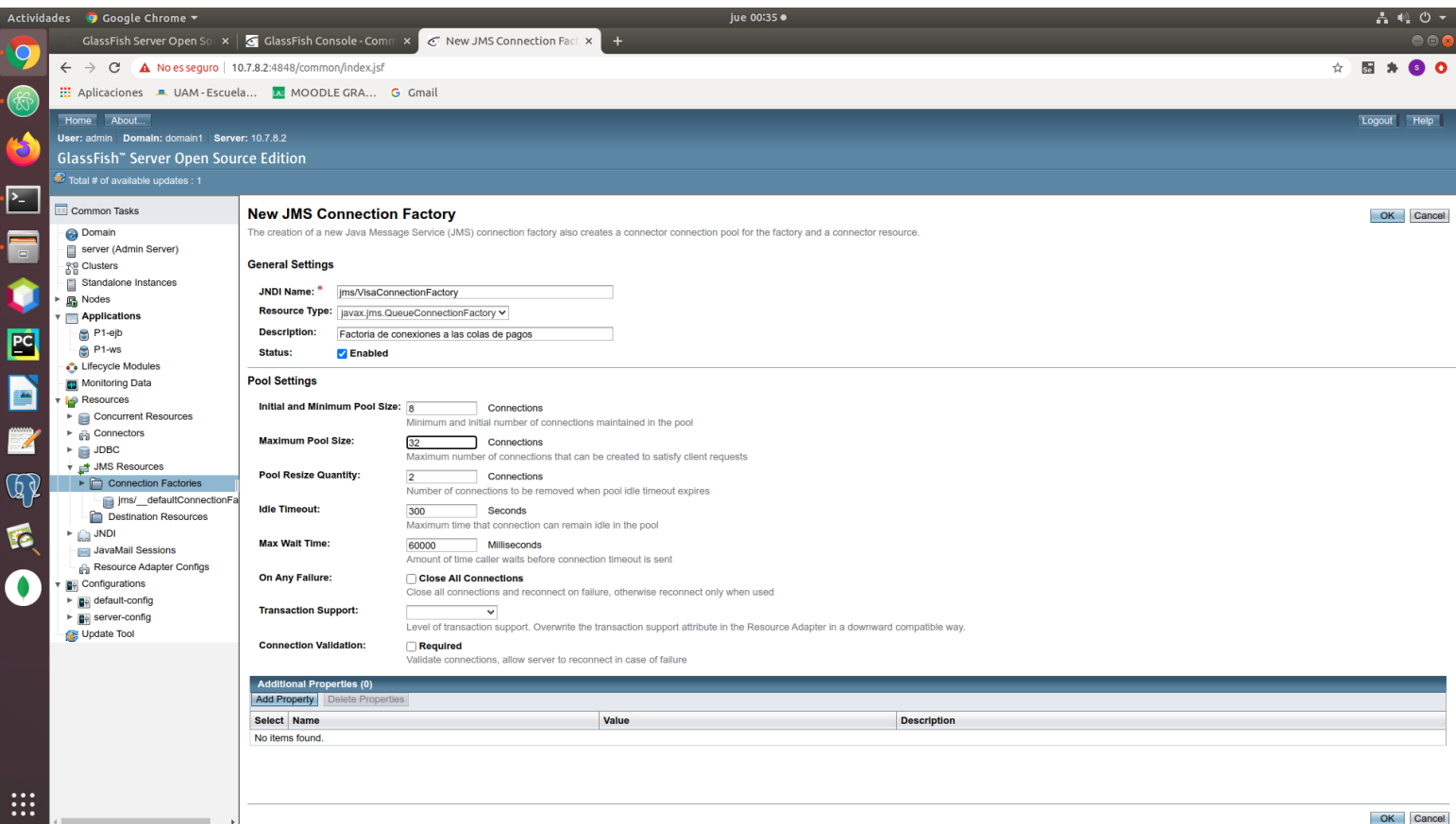
- Probar a realizar pagos correctos. Comprobar que disminuye el saldo de las tarjetas sobre las que realice operaciones. Añadir a la memoria las evidencias obtenidas.
- Realice una operación con identificador de transacción y de comercio duplicados. Compruebe que el saldo de la tarjeta especificada en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

(Texto de la respuesta)

Ejercicio 9:

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

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



Podemos encontrar la consola de administración en Resources→JMS Resources→ Connection Factories

Ejercicio 10:

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

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

The screenshot shows the GlassFish Server Open Source Edition administration console in a web browser. The page title is "New JMS Connection Factory". The left sidebar shows the navigation tree with "Connection Factories" selected. The main content area displays the configuration form for a new JMS Connection Factory.

General Settings

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

Pool Settings

- Initial and Minimum Pool Size: Connections
- Maximum Pool Size: Connections
- Pool Resize Quantity: Connections
- Idle Timeout: Seconds
- Max Wait Time: Milliseconds
- On Any Failure: ☐ Close All Connections
- Transaction Support: Level of transaction support.
- Connection Validation: ☐ Required

Additional Properties (0)

Select	Name	Value	Description
No items found.			

Ejercicio 11:

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

- Incluya en la clase VisaCancelacionJMSBean:
 - Consulta SQL necesaria para actualizar el código de respuesta a valor 999, de aquella autorización existente en la tabla de pagos cuyo idAutorizacion coincida con lo recibido por el mensaje.
 - Consulta SQL necesaria para rectificar el saldo de la tarjeta que realizó el pago.
 - Método onMessage() que implemente ambas actualizaciones. Para ello tome de ejemplo 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 y cierre de conexiones.

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

Modificación del código en sun-ejb-jar.xml:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!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">
3  <sun-ejb-jar>
4  <enterprise-beans>
5  <ejb>
6  <ejb-name>VisaCancelacionJMSBean</ejb-name>
7  <!-- TODO - definir el nombre de la connection factory -->
8  <mdb-connection-factory>
9  <jndi-name>jms/VisaConnectionFactory</jndi-name>
10 </mdb-connection-factory>
11 </ejb>
12 </enterprise-beans>
13 </sun-ejb-jar>
14
```

Modificaciones en VisaCancelacionJMSBean.java (Consultas SQL):

```
32 private static final String UPDATE_CANCELA_QRY =
33     "update pago" +
34     " set codRespuesta=999" +
35     " where idAutorizacion=?";
36 // TODO : Definir UPDATE sobre la tabla pagos para poner
37 // codRespuesta a 999 dado un código de autorización
38 private static final String UPDATE_SALDO_QRY =
39     "update tarjeta" +
40     " set saldo=saldo + pago.importe" +
41     " from pago" +
42     " where pago.idAutorizacion=? and pago.numeroTarjeta=tarjeta.numeroTarjeta";
43
```

Modificiaciones en VisaCancelaciónJMSBean.java (método onMessage):

```
52 public void onMessage(Message inMessage) {
53     TextMessage msg = null;
54
55     PreparedStatement pstmt = null;
56     Connection con = null;
57     int idAutorizacion = 0;
58
59     try {
60         if (inMessage instanceof TextMessage) {
61             msg = (TextMessage) inMessage;
62             logger.info("MESSAGE BEAN: Message received: " + msg.getText());
63
64             idAutorizacion = Integer.parseInt(msg.getText());
65             con = getConnection();
66
67             // Cancelar pago
68             String aux = UPDATE_CANCELA_QRY;
69             pstmt = con.prepareStatement(aux);
70             pstmt.setInt(1, idAutorizacion);
71             if (!pstmt.execute()
72                 && pstmt.getUpdateCount() == 1) {
73                 logger.warning("Error al cancelar pago usando id " + idAutorizacion);
74             }
75
76             // Update saldo
77             String aux2 = UPDATE_SALDO_QRY;
78             pstmt = con.prepareStatement(aux2);
79             pstmt.setInt(1, idAutorizacion);
80             if (!pstmt.execute()
81                 && pstmt.getUpdateCount() == 1) {
82                 logger.warning("Error al actualizar saldo para pago usando id " + idAutorizacion);
83             }
84
85         } else {
86             logger.warning(
87                 "Message of wrong type: "
88                 + inMessage.getClass().getName());
89         }
90     } catch (JMSEException e) {
91         e.printStackTrace();
92         mdc.setRollbackOnly();
93     } catch (Throwable te) {
94         te.printStackTrace();
95     } finally { // Control de errores en el método onMessage y cierre de conexiones.
96         try {
97             if (pstmt != null) {
98                 pstmt.close();
99                 pstmt = null;
100             }
101             if (con != null) {
102                 closeConnection(con);
103                 con = null;
104             }
105         } catch (SQLException e) {
106             }
107     }
108 }
109 }
```

Ejercicio 12:

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

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

En *VisaQueueMessageProducer.java*, presente en la carpeta *P1-jms/src/clientjms/ssii2*, haremos los siguientes cambios:

Hacer el mapping de los recursos *ConnectionFactory* y *Queue*:

```
17      // TODO: Anotar los siguientes objetos para
18      // conectar con la connection factory y con la cola
19      // definidas en el enunciado
20      @Resource(mappedName = "jms/VisaConnectionFactory")
21      private static ConnectionFactory connectionFactory;
22      @Resource(mappedName = "jms/VisaPagosQueue")
23      private static Queue queue;
```

Uso de JNDI:

```
69      try {
70          // TODO: Inicializar connectionFactory
71          // y queue mediante JNDI
72          /*
73          InitialContext jndi=new InitialContext();           // Creamos el initial ctxt
74          connectionFactory=(ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory"); // Nueva connection factory, con mappedName correspondiente
75          queue=(Queue)jndi.lookup("jms/VisaPagosQueue");    // Nueva cola de mensajes, con mappedName correspondiente
76          */
```

La principal ventaja de este método frente al otro es que con éste, no es necesario conocer los nombres de la cola al compilar. Esto significa que el nombre de la cola puede cambiar y no necesitaremos recompilar el código, ya que en su lugar es una referencia dinámica. Sin embargo, al ser una referencia, la localización de la cola cuesta más tiempo.

Ejercicio 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?

Este es el fichero *jms.properties*, con las IPs adecuadas:

```
1  as.home=${env.J2EE_HOME}
2  as.lib=${as.home}/lib
3  as.user=admin
4  as.host.client=10.7.8.1
5  as.host.server=10.7.8.2
6  as.port=4848
7  as.passwordfile=${basedir}/passwordfile
8  as.target=server
9  jms.factoryname=jms/VisaConnectionFactory
10 jms.name=jms/VisaPagosQueue
11 jms.physname=VisaPagosQueue
12
```

Ejercicio 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.

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

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

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

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

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

Verifique el contenido de la cola ejecutando:

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

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

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

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

Al realizar este ejercicio en los laboratorios surge un error indicando que no es posible resolver el nombre del host local a una dirección IP. Esto se debe a que no hay una entrada con dicho nombre en el fichero /etc/hosts asociado a una dirección IP. Como dicho fichero no se puede editar, la solución es ejecutar 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>
```

Modificación del código para implementar el envío de args[0] como un mensaje de texto:

```
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

try {
    // TODO: Inicializar connectionFactory
    // y queue mediante JNDI
    /*
    InitialContext jndi=new InitialContext();
    connectionFactory=(ConnectionFactory)jndi.lookup("jms/VisaConnectionFactory");
    queue=(Queue)jndi.lookup("jms/VisaPagosQueue");
    */
    // Creamos el initial context
    // Nueva connection factory, con mappedName correspondiente
    // Nueva cola de mensajes, con mappedName correspondiente

    connection = connectionFactory.createConnection();
    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
    if (args[0].equals("-browse")) {
        browseMessages(session);
    } else {
        // TODO: Enviar argv[0] como mensaje de texto
        messageProducer=session.createProducer(queue);
        message=session.createTextMessage();
        message.setText(args[0]);
        messageProducer.send(message);
        messageProducer.close();
        session.close();
        System.out.println("Mensaje enviado correctamente: " + message.getText());
    }
} catch (Exception e) {
    System.out.println("Excepcion : " + e.toString());
} finally {
    if (connection != null) {
        try {
            connection.close();
        } catch (JMSException e) {
        }
    }
} // if
```

Cuestiones:

Cuestión 1:

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

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

Cuestión 2:

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

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

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

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

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

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

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