Prácticas de laboratorio
JMS: Java Message Service
(3 sesiones)

Concurrencia y Sistemas Distribuidos

Introducción

Esta práctica tiene como objetivo desarrollar una aplicación simple utilizando el servicio de mensajería **Java Message Service** (JMS), así como instalar, configurar y ejecutar un proveedor JMS. En particular, se desarrollará el componente de comunicación de una aplicación de mensajería instantánea, a la que hemos llamado "CSD Messenger".

Dado el carácter introductorio de esta práctica, se tratarán aspectos básicos tales como la conexión con el proveedor JMS y el envío y recepción de mensajes. Otros aspectos que serían fundamentales para el desarrollo de una aplicación real, tales como seguridad y tolerancia ante fallos, se han dejado intencionadamente al margen en aras de la simplicidad.

La duración estimada de esta práctica es de tres semanas. Cada semana debe asistir a una sesión de laboratorio donde podrá comentar con el profesorado de prácticas sus progresos y resolver las dudas que le surjan. Tenga en cuenta que necesitará destinar algo de tiempo de su trabajo personal para concluir la práctica.

Esta practica está preparada para realizarla desde los sistemas **Windows** de los laboratorios del DSIC. Si desea realizarla en el sistema Windows de su ordenador personal, se requiere tener instalada la versión 1.8 del JDK Java de Oracle (la versión openJDK que viene en muchas distribuciones es probable que no funcione) y adaptar los nombres de los directorios a su instalación.

Actividad 1

En esta actividad se pretende instalar, configurar y ejecutar un proveedor JMS. El proveedor JMS que se instalará es Apache ActiveMQ Artemis (simplemente *Artemis* en el resto de este documento).

Artemis es un conjunto de librerías y aplicaciones desarrolladas en Java que se distribuye como un archivo comprimido. Para esta práctica, lo tiene disponible en el Lessons de la práctica 5 en el sitio PoliformaT de la asignatura (fichero "apache-artemis-1.4.0-bin.zip").

Para evitar problemas de cuotas de espacio en disco, el contenido de este archivo ya está descomprimido en la unidad asig(\\fileserver.dsic.upv.es)(M:)

```
M:\ETSINF\csd\jms\artemis
```

Como hemos comentado, en esta actividad vamos a instalar, configurar y ejecutar el proveedor JMS Artemis. Así, crearemos primero una instancia del bróker de mensajería, definiremos los puertos a utilizar por el bróker y lanzaremos el bróker a ejecución. Para ello, seguiremos las instrucciones que se indican a continuación.

NOTA: Como Anexo a este documento se incluye una breve descripción de los comandos de Artemis que nos van a ser necesarios para nuestra práctica.

IMPORTANTE: En todas las modificaciones de fichero que se indican a continuación, dichas modificaciones se deben realizar "a mano" (escribiendo directamente en el fichero) y no mediante copia-pega desde el boletín de la práctica (pues entonces se podrían copiar caracteres no visibles que generarán errores en la creación y ejecución del bróker).

Instrucciones

1. Abra un explorador de archivos y sitúese en M:\ETSINF\csd\jms\artemis\bin
2. Cree una instancia del bróker de mensajería en un directorio de tu disco (W:), para ello cree la estructura de directorios:
   - `W:\csd\jms`
   - Desde el explorador de archivos abierto en el paso 1, utilizando el menú "Archivo" abra un "símbolo del sistema". Se abrirá una nueva ventana con el intérprete de órdenes básico de Windows, ya situado en el directorio de trabajo `M:\ETSINF\csd\jms\artemis\bin`
   - Escriba la orden: `artemis create W:\csd\jms\csdbroker` *(este mandato debe escribirlo en una sola línea)*
   - Conteste **admin**, **admin**, **admin** e **Y** a las cuatro preguntas que realizará el mandato anterior. *(Definirá así el usuario, contraseña y rol que se utilizará para las conexiones anónimas, las cuales a su vez deberá permitir. Tenga en cuenta que no se mostrará ningún carácter en pantalla cuando se introduzca la contraseña)*
3. Como se va a realizar esta práctica en un ordenador compartido con otros usuarios (al realizarse a través del Escritorio Remoto del DSIC), es imprescindible decidir de forma coordinada qué puertos utilizará su bróker con el fin de que no sean los mismos que los de otros brókeres lanzados por otros usuarios. El profesor de prácticas le indicará qué puertos utilizar. Concretamente le facilitará un puerto al que denominaremos NUMPUERTO y su bróker utilizará este puerto y el siguiente al que llamaremos NUMPUERTO +1

4. En el fichero "W:\csd\jms\csdbroker\etc\broker.xml" cambie en la siguiente línea el puerto 61616 por NUMPUERTO:

```
<acceptor
name="artemis">tcp://0.0.0.0:61616?tcpSendBufferSize=1048576;tcpReceiveBuffer
Size=1048576</acceptor>
```

El fichero broker.xml es el archivo de configuración del servidor central que contiene el elemento central o core, con la configuración del servidor principal. Entre otros aspectos, permite definir los denominados aceptadores (acceptors). Cada aceptador define una forma en la que se pueden hacer conexiones al servidor Artemis. En nuestro caso, hemos definido un aceptador que usa Artemis para escuchar las conexiones en el puerto NUMPUERTO, indicando también los tamaños del buffer de envío del TCP y del buffer de recepción del TCP en bytes (`1048576bytes = 1Mbyte`)

5. En el mismo fichero anterior, elimine el resto de líneas con elementos `<acceptor … </acceptor>`

   Eliminamos estas líneas, ya que solamente vamos a utilizar el aceptador definido en el paso anterior.

6. En el fichero "W:\csd\jms\csdbroker\etc\bootstrap.xml" modifique en la siguiente línea el puerto `8161` por NUMPUERTO +1:

   ```
   <web bind="http://localhost:8161" path="web">
   ```

   El fichero bootstrap.xml es el fichero de configuración por defecto que se utiliza para arrancar el servidor. Entre otros aspectos, en dicho fichero se indica cuál es el fichero de configuración del bróker (broker.xml).

7. Ahora que ya ha configurado su bróker, puede ejecutarlo con el siguiente mandato en un intérprete de órdenes situado en W:\csd\jms\csdbroker\bin (*abra un "símbolo del sistema" desde el menú "Archivo" del explorador de archivos una vez situado en el directorio W:\csd\jms\csdbroker\bin)*
   - `artemis run`

Verá en pantalla el bróker en ejecución:

```
           _                _
     /\   | |              | |
    /  \  | |__   ___ _ __ | |_ ___
   / /\ \ | '_ \ / _ \ '_ \| __/ _ \
  / ____ \| | | |  __/ | | | ||  __/
 /_/    \_\_| |_|\___|_| |_|\__\___|
Apache ActiveMQ Artemis 1.4.0
```

Comprobación

Al lanzar a ejecución el bróker, si todo ha funcionado de forma correcta, nos aparecerá en pantalla que el servidor Artemis está en funcionamiento.

Si no fuera así, deberá repetir el proceso anterior, asegurándose de que se escribe todo de forma correcta. En caso de dudas, consulte a su profesor.

Actividad 2

En esta actividad se pretende comprobar el funcionamiento del sistema de mensajería instantánea proporcionado en la práctica, denominado CSD Messenger.

Conceptos previos

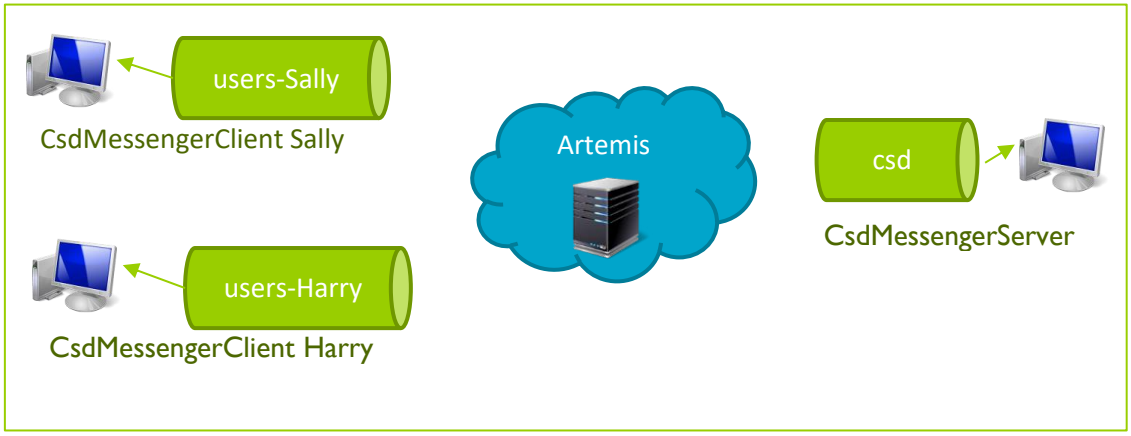
CSD Messenger está formado principalmente por dos aplicaciones, una aplicación servidor (CsdMessengerServer) y una aplicación cliente (CsdMessengerClient). CsdMessengerServer se encarga de la gestión de las colas JMS para los clientes y de informar a cada cliente los nombres del resto de clientes que están utilizando el servicio.

En un momento dado solo debe haber activa una aplicación CSDMessengerServer, pero pueden estar activas varias CsdMessengerClient, una por cada usuario conectado al sistema. Cuando se ejecuta CsdMessengerClient es necesario pasarle como argumento el nombre del usuario que se ha conectado.

La aplicación CsdMessengerServer consume mensajes de la cola llamada “csd”, mientras que cada aplicación CsdMessengerClient consume de la cola “users-NOMBREDELUSUARIO” correspondiente.

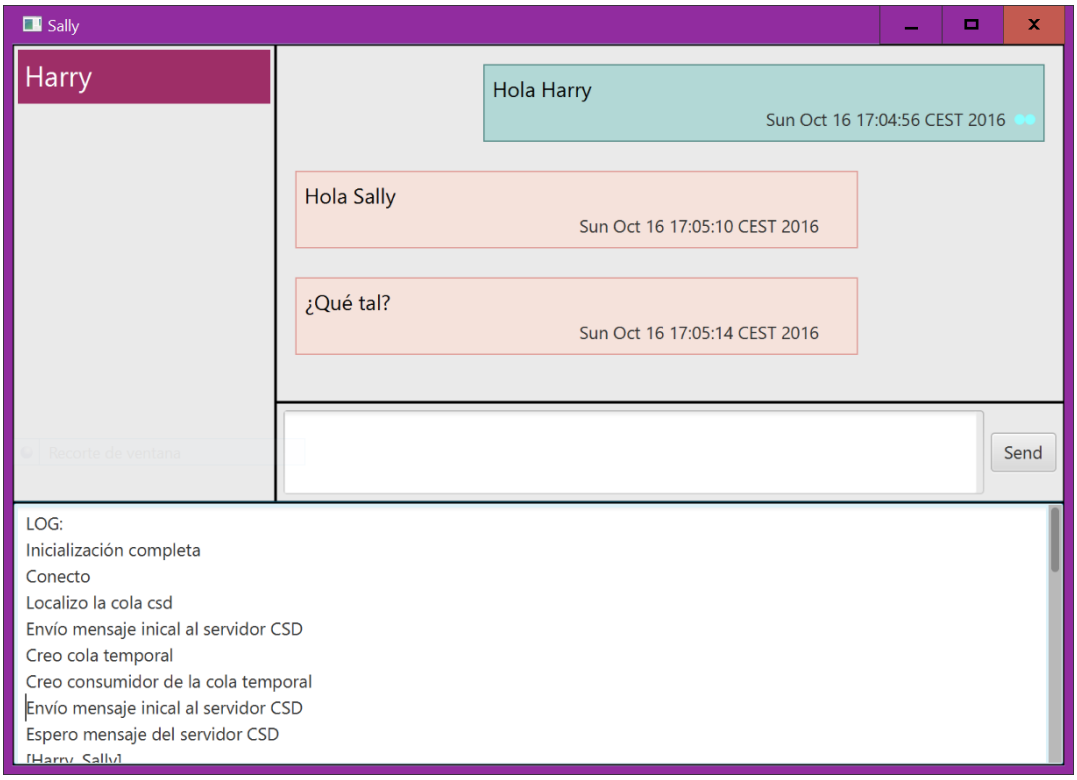
La cola “csd” debe existir previamente, por lo que hay que crearla en la configuración de Artemis. Las colas “users-NOMBREDELUSUARIO” son creadas por CSDMessengerServer bajo demanda, la primera vez que un usuario se conecta al sistema.

La siguiente figura resume los componentes previamente citados, asumiendo los usuarios Sally y Harry:



CsdMessengerServer es una aplicación Java en modo consola y muestra en pantalla un registro de las peticiones que recibe en la cola “csd”.

CsdMessengerClient es una aplicación Java con interfaz gráfica de usuario, desarrollada con JavaFX. Su aspecto es el siguiente:



Los elementos de esta interfaz son:

- Panel izquierdo. Se muestran los usuarios conocidos por el sistema. El primer usuario que se conecte verá este panel vacío. Se requiere que se conecte al menos otro usuario para que empiece a mostrarse la lista de usuarios conocidos.
- Panel derecho: Muestra la lista de mensajes de la conversación con el usuario seleccionado en el panel izquierdo. Dispone de un área de texto para componer un mensaje y un botón para enviarlo (también puede enviarse pulsando la tecla Entrar)
- Panel inferior: Muestra un registro de mensajes informativos de la aplicación.

Instrucciones

1. Descargue del Lessons de la práctica 5 los archivos "CsdMessengerServer.jar", "CsdMessengerClient.jar" y "jndi.properties". Guárdelos en: W:\csd\jms

2. Los ficheros "CsdMessengerServer.jar" y "CsdMessengerClient.jar" contienen las aplicaciones servidor y cliente previamente mencionadas. Antes de poder ejecutarlas, es necesario configurar el puerto en el que está escuchando el bróker JMS. Para ello haga lo siguiente.

   2.1. Edite el fichero "jndi.properties" y cambie el puerto 61616 por NUMPUERTO en la línea:

   ```
   connectionFactory.ConnectionFactory=tcp://localhost:61616
   ```

   2.2. Añada el fichero "jndi.properties" modificado a los archivos .jar de las aplicaciones servidor y cliente *(abra un "símbolo del sistema" desde el menú "Archivo" del explorador de archivos una vez situado en el directorio W:\csd\jms y escriba)*:

```
jar uvf CsdMessengerServer.jar jndi.properties
jar uvf CsdMessengerClient.jar jndi.properties
```

3. Ahora es necesario cambiar la configuración del bróker para adecuarla a la aplicación que queremos desarrollar. Para ello, haga lo siguiente:

   3.1. Detenga Artemis, pulsando Control-C en la ventana que se está ejecutando.

   3.2. Cree la cola "csd". Para ello, en el fichero "W:\csd\jms\csdbroker\etc\broker.xml" añada un elemento <queue> tal como se muestra a continuación:

```
<jms xmlns="urn:activemq:jms">
...
   <queue name="csd"><durable>true</durable></queue>
</jms>
```

   Con esta configuración se crea una cola llamada "csd" que perdura incluso cuando el bróker JMS se detiene (i.e. propiedad "durable").

   3.3. Deshabilite la creación automática de colas. Para ello, añada en el mismo fichero que en el paso anterior un elemento <auto-create-jms-queues> con el contenido `false` tal como se muestra a continuación:

```
<address-settings>
   <address-setting>
      ...
      <auto-create-jms-queues>false</auto-create-jms-queues>
   </address-setting>
</address-settings>
```

   Por defecto, en Artemis cuando un cliente referencia una cola JMS, ésta se crea inmediatamente de forma automática. Pero dicha característica resulta inadecuada para el servicio de mensajería que queremos implementar, ya que la creación de las colas JMS debe estar gobernada por CsdMessengerServer. Por eso, hemos requerido eliminar la auto-creación automática de colas.

   3.4. Ejecute de nuevo el bróker, lanzándolo con:

```
artemis run
```

Comprobación

A partir de este punto ya está todo preparado para probar las aplicaciones. Para ello:

1. Ejecute en otra terminal el servidor de CsdMessenger
```
java -jar CsdMessengerServer.jar
```

2. Ejecute en otra terminal un cliente, por ejemplo Sally:
```
java -jar CsdMessengerClient.jar Sally
```

3. Y ejecute en otra terminal otro cliente, por ejemplo Harry:
```
java -jar CsdMessengerClient.jar Harry
```

4. Puede ejecutar más clientes con otros nombres si así lo desea.

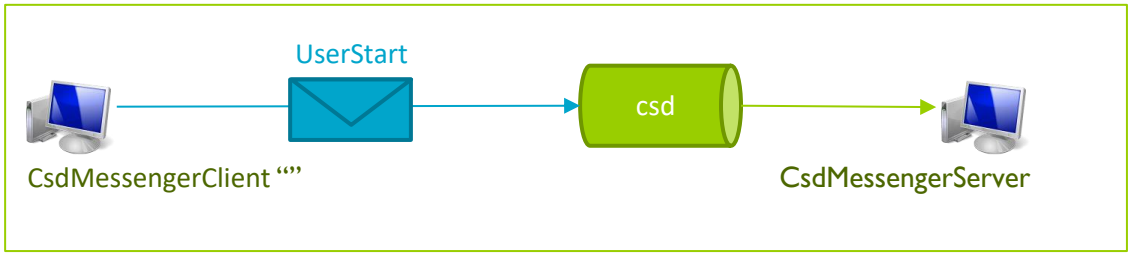
Utilice las aplicaciones cliente para conversar. Compruebe que es posible enviar mensajes a un usuario, aunque el usuario no esté conectado en ese momento y que los mensajes se entregan cuando el usuario se conecta posteriormente. Esto es gracias a que los mensajes se guardan en las colas del proveedor JMS. Por ejemplo, cierre la aplicación cliente de Sally, envíele como Harry algunos mensajes, vuelva a abrir la aplicación cliente de Sally y debería observar que en ese momento recibe los mensajes enviados mientras no estaba conectada.

Actividad 3

Tiene a su disposición una implementación parcial de la aplicación CsdMessengerClient, llamada **FuentesCsdMessenger**. En este caso, el componente de interfaz de usuario está totalmente implementado, pero en cambio el componente de comunicación está pendiente de implementar, por lo que no se produce ningún tipo de envío o recepción de mensajes. En esta actividad se pretende conseguir que la implementación FuentesCsdMessenger envíe un mensaje a CsdMessengerServer.

La primera tarea que debe realizar el componente de comunicación es enviar un mensaje JMS a CsdMessengerServer, para notificar que se conecta un usuario al sistema.

Por el momento, con el fin de no complicar en exceso esta actividad, en el mensaje no se va a indicar el nombre del usuario. De esta forma, CsdMessengerServer no contestará a dicho mensaje, por lo que no es necesario que prepare su código para tratar esa respuesta (esto ya se hará en la actividad siguiente).



Instrucciones

1. Descargue del Lessons de la práctica los archivos “**FuentesCsdMessenger.jar**” y “**ArtemisLib.jar**”
2. Si está usando BlueJ, ábralo y añada en el apartado Herramientas/Preferencias/Librerías la librería “ArtemisLib.Jar”. Reinicie BlueJ para aplicar este cambio. Si utiliza otro entorno de programación, añada esta librería como se requiera en dicho entorno.
3. Abra su archivo “FuentesCsdMessenger.jar” con su entorno de programación Java. Se habrá creado una carpeta llamada “FuentesCsdMessenger”.
4. Con un editor externo a BlueJ edite el fichero “jndi.properties” (ej. con un editor de texto) y cambie el puerto 61616 por NUMPUERTO, tal como se había hecho en la actividad anterior.
5. De nuevo, empleando su entorno de programación Java (ej. BlueJ), edite la clase **comm.Communication**.
6. En dicha clase, implemente en el método **initialize()** lo siguiente:

*(NOTA: en el código de la clase CsdMessengerServer tiene ejemplos muy similares a lo que se pide a continuación y entre paréntesis dispone de enlaces a la documentación de Java del **método que debe utilizar**)*

6.1. Inicialice un contexto JNDI (javax.naming.InitialContext()).

6.2. Establezca la conexión con el broker (javax.naming.InitialContext.lookup()).

6.3. Cree un contexto JMS (javax.jms.ConnectionFactory. createContext()).

6.4. Cree un *producer* asociado al contexto anterior (javax.jms.JMSContext.createProducer())

6.5. Cree un objeto de la clase *messageBodies.UserStart*, pasando al constructor del objeto el argumento "". Por ejemplo:

```
UserStart us=new UserStart("");
```

6.6. Construya un mensaje JMS y asígnele el objeto anterior (javax.jms.JMSContext.createObjectMessage())

6.7. Busque la cola asociada al nombre "dynamicQueues/csd" (utilizando el método lookup(): javax.naming.InitialContext.lookup()).

6.8. Envíe el mensaje creado en el punto 6.6 a la cola obtenida en el punto 6.7 utilizando el *producer* creado en el punto 6.4 (javax.jms.JMSProducer.Send())

Comprobación

Asegúrese de que está en ejecución *CsdMessengerServer*.

Ejecute el método `ui.CsdMessengerClient.main()`, pasando como argumento el nombre de usuario que quiera.

Debería aparecer una ventana con varios avisos y también la interfaz gráfica del cliente vacía salvo el mensaje "`LOG: Inicialización Completa`".

Por otra parte, en la terminal donde ha ejecutado `CsdMessengerServer` debería ver la siguiente línea:

```
"He recibido una petición UserStart anónima. No hago nada"
```

Actividad 4

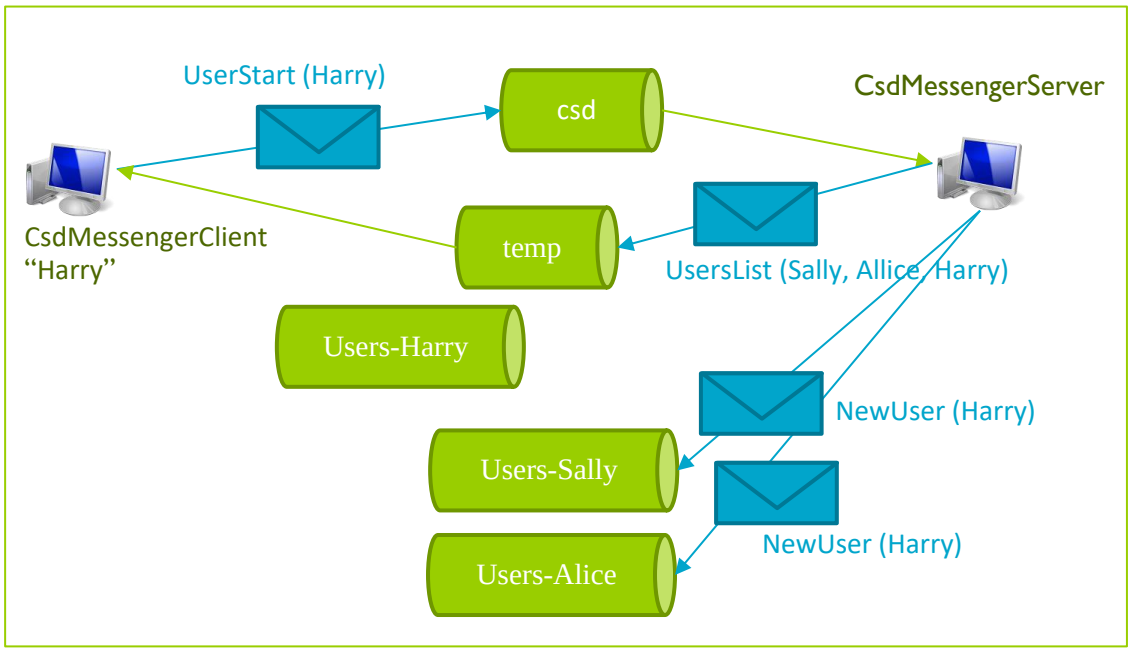
En esta actividad, se pretende completar el protocolo de intercambio de mensajes iniciado en la actividad anterior, atendiendo la respuesta de CsdMessengerServer ante la notificación de que se conecta un usuario.

Conceptos previos

Cuando CsdMessengerServer recibe un mensaje del tipo UserStart y se le indica correctamente el nombre del usuario, realiza las siguientes acciones:

- Si es la primera vez que el usuario indicado se conecta:
  - Solicita la creación de la cola JMS "users-NOMBREDEUSUARIO".
  - Notifica al resto de usuarios conocidos la existencia del nuevo usuario.

- Contesta a la cola especificada en la propiedad "ReplyTo" del mensaje recibido con una lista de usuarios conocidos, formada por todos los usuarios que ya han ejecutado la aplicación cliente previamente.



La finalidad de los mensajes que envía CsdMessengerServer es que las aplicaciones cliente actualicen en su panel izquierdo la lista de usuarios conocidos.

El uso de una cola temporal es un patrón habitual en este tipo de aplicaciones. En este caso se ha utilizado porque la primera vez que un usuario se conecta no dispone todavía de su cola permanente "users-NOMBREDEUSUARIO", pero necesita de una cola donde esperar una respuesta.

Instrucciones

1. Sustituya lo que había codificado para resolver el punto 6.6 de la actividad anterior por lo siguiente (debe mantener el resto de puntos):

   1.1. Construya un mensaje JMS con un objeto de la clase messageBodies.UserStart, pasando al constructor del objeto el nombre del usuario que se ha conectado (resultado del método ui.API.getMyName()).

   1.2. Cree una cola JMS temporal (javax.jms.JMSContext.createTemporaryQueue())

   1.3. Asigne dicha cola temporal a la propiedad `ReplyTo` del mensaje JMS (javax.jms.Message.setJMSReplyTo())

2. Tras lo codificado para la actividad anterior, realice lo siguiente:

   2.1. Cree un *consumer* asociado a la cola temporal (para ello, puede consultar cómo utilizar el método javax.jms.JMSContext.createConsumer()).

   2.2. Espere la llegada de un mensaje en la cola temporal (javax.jms.JMSConsumer. receive()). Ese mensaje contendrá un objeto de la clase `UsersList` que contiene un array de strings con la lista de usuarios conocidos, recuperable con el método `getUsers`();

   2.3. Llame al método `ui.API.updateUserList()` pasando el array recibido como argumento.

Comprobación

Asegúrese de que está en ejecución `CsdMessengerServer`.

Ejecute el método `ui.CsdMessengerClient.main()`, pasando como argumento el nombre de usuario que quiera. En el panel izquierdo se mostrará la lista de usuarios conocidos.

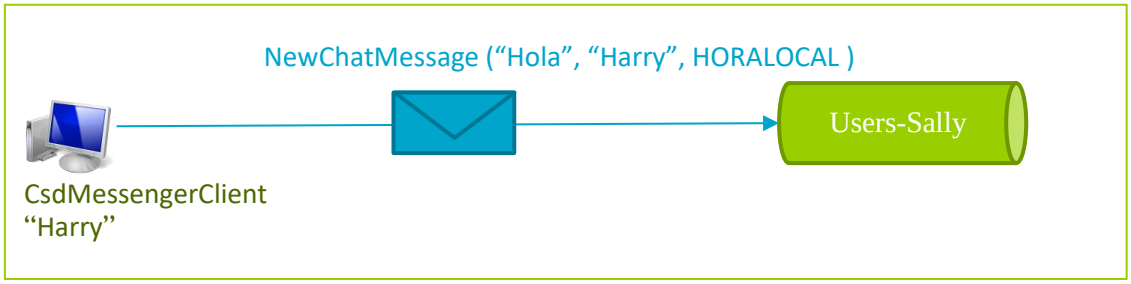
Actividad 5

En esta actividad se pretende enviar un mensaje JMS a la cola del usuario que está seleccionado en el panel izquierdo cuando se pulsa el botón de envío de mensaje o la tecla Entrar.

Conceptos previos

La interfaz de usuario, cuando detecta que se pulsa el botón de envío o la tecla Entrar, solicita al componente de comunicación el envío de un mensaje del tipo NewChatMessage, en el que se indica el texto del mensaje, el nombre del usuario emisor y el valor del reloj local del emisor.

Este mensaje se entrega a la cola JMS del usuario seleccionado en el panel izquierdo. El mensaje será consumido de inmediato o permanecerá en la cola, dependiendo de si el usuario está o no conectado en estos momentos.



Instrucciones

1. Implemente en el método `comm.Communication.sendChatMessage` lo siguiente:
   1.1. Busque la cola asociada al nombre "dynamicQueues/users-" + destUser, siendo *destUser* un parámetro del método `sendChatMessage` (javax.naming.InitialContext.lookup()).
   1.2. Cree un objeto de la clase `messageBodies.NewChatMessage`, pasando al constructor del objeto los argumentos `text, ui.API.getMyName()` y `timestamp`.
   1.3. Construya un mensaje JMS y asígnele el objeto anterior, utilizando el método createObjectMessage (javax.jms.JMSContext.createObjectMessage()).
   1.4. Envíe el mensaje creado en el paso anterior utilizando el mismo *producer* que se había empleado en la actividad 3 (javax.jms.JMSProducer.Send()).

Comprobación

Ejecute la aplicación cliente completa que le hemos facilitado:

```
java –jar CsdMessengerClient.jar "Sally"
```

Ejecute el método `ui.CsdMessengerClient.main()`, pasando como argumento el nombre de otro usuario distinto a Sally. En su aplicación seleccione a Sally y envíele un mensaje. En la aplicación de Sally debería aparecer el mensaje que ha enviado.
Si Sally contesta no podrá ver su mensaje, ya que en la aplicación del alumno todavía no se trata la llegada de mensajes de chat de otros usuarios. Esto se va a realizar en la siguiente actividad.

Actividad 6

En esta actividad se procesará la llegada de mensajes de chat provenientes de otros usuarios.

Conceptos previos

Tal como se ha visto en la actividad anterior, un mensaje de chat llega a la cola JMS del usuario destinatario. La aplicación CsdMessengerClient debe estar permanentemente a la escucha de llegadas de mensaje a esta cola, y cuando llega un mensaje debe notificar al componente interfaz de usuario el nuevo mensaje de chat para que se pueda mostrar en el panel derecho.

Dado que la aplicación debe estar permanentemente a la escucha de nuevos mensajes, pero al mismo tiempo poder atender la interacción del usuario, la recepción debe llevarse a cabo en un hilo de ejecución distinto al de la interfaz de usuario. Además, al tratarse de un hilo distinto, es necesario crear un nuevo contexto JMS para dicho hilo, así como un nuevo *producer* y un nuevo *consumer*, ya que los contextos JMS están pensados para ser utilizados desde un único hilo de forma simultánea.

Instrucciones

1. Implemente en el método `comm.Communication.run()` lo siguiente:
   1.1. Cree el contexto JMS a partir del contexto JMS creado en la actividad 3, con el método createContext(), pasando como parámetro JMSContext.AUTO_ACKNOWLEDGE (javax.jms.JMSContext.createContext())
   1.2. Cree un *producer* asociado al contexto anterior (javax.jms.JMSContext.createProducer()).
   1.3. Busque una cola asociada al nombre "dynamicQueues/users-" + ui.API.getMyName() (javax.naming.InitialContext.lookup()).
   1.4. Cree un *consumer* para la cola anterior asociado al contexto creado en el punto 1.1. (javax.jms.JMSContext.createConsumer()).

1.5. En un bucle, esperar permanentemente la llegada de mensajes a la cola creada previamente (javax.jms.JMSConsumer.receive()).

1.6. Si el objeto contenido en el mensaje recibido es de la clase: `messageBodies.NewChatMessage`, entonces:

1.6.1. Notifique a la interfaz de usuario la llegada de un nuevo mensaje, con el método chatMessageReceived de nuestro API (`ui.API.chatMessageReceived`());

1.7. Si el objeto contenido en el mensaje es de otra clase diferente:

1.7.1. Muestre en el Log el nombre de dicha clase

Comprobación

Ejecute la aplicación cliente completa que hemos facilitado:

```
java –jar CsdMessengerClient.jar "Sally"
```

Ejecute el método `ui.CsdMessengerClient.main`(), pasando como argumento el nombre de otro usuario distinto al anterior. Ahora, además de que Sally vea los mensajes que la aplicación del alumno le envía, dicha aplicación debería ver también los mensajes enviados por Sally.

Actividad 7

En esta actividad se pretende completar la aplicación cliente. Los aspectos que han quedado pendientes son los siguientes:

- Tratar el mensaje `NewUser` enviado por CsdMessengerServer a las aplicaciones cliente cuando se conecta un usuario nuevo.
- Implementar el reconocimiento de llegada de mensajes, para mostrar así un segundo círculo gris en el estado de cada mensaje enviado.
- Implementar el reconocimiento de lectura de mensajes, para mostrar así los dos círculos asociados a cada mensaje en color azul.

El alumno deberá aplicar los conocimientos adquiridos a lo largo de esta práctica para realizar esta actividad.

Una explicación detallada sobre la configuración del servidor Apache ActiveMQ Artemis está disponible en:

https://activemq.apache.org/components/artemis/documentation/1.5.3/using-server.html

A continuación, describiremos los aspectos más relevantes de este servidor relativos a esta práctica.

Creación de un broker Artemis:

```
artemis create csdbroker
```

¿Cómo iniciar Artemis?

Artemis se puede iniciar con el siguiente mandato, que nos lanzará el broker en background:

```
artemis-service start
```

O bien con el mandato:

```
artemis run
```

Que lanzará el broker, mostrando todos los mensajes explicativos de configuración.

En nuestra práctica, se recomienda iniciarlo con el mandato "run", pues en caso de que se haya configurado de forma errónea el broker, nos mostrará por pantalla aquellos puntos de la configuración donde existen problemas. Y podremos así tratar de solucionarlos.

Por tanto, para nuestra práctica deberemos utilizar:

```
W:\csd\jms\csdbroker\bin\artemis run
```

¿Cómo detener Artemis?

Artemis se detiene con el mandato:

```
artemis-service stop
```

Por tanto, en nuestra práctica debemos utilizar:

```
W:\csd\jms\csdbroker\bin\artemis-service stop
```

¿Cómo reinicializar Artemis?

Para eliminar todos los buzones creados y poder así volver a usar Artemis como si acabara de ser instalado, debemos detener el servicio Artemis, eliminar los ficheros del directorio "data" del broker instalado y volver a reactivar el servicio Artemis.

En concreto, en nuestra práctica debemos ejecutar:

```
W:\csd\jms\csdbroker\bin\artemis-service stop
```

Eliminaremos todos los ficheros de la carpeta:

```
W:\csd\jms\csdbroker\data
```

Y volveremos a lanzar Artemis:

```
W:\csd\jms\csdbroker\bin\artemis-service start
```

¿Cómo eliminar el historial de conversaciones?

Esta práctica está configurada para que las conversaciones se guarden en un directorio denominado "csdmessenger". Para eliminar el historial de conversaciones, basta con eliminar los ficheros de dicho directorio.

En concreto, en nuestra práctica debemos eliminar todos los ficheros del directorio:

```
W:\csdmessenger
```

¿Cómo ejecutar y detener el servidor CsdMessengerServer?

Para iniciarlo, abra un terminal y ejecute:

```
java -jar CsdMessengerServer.jar
```

Para detenerlo, pulse Control-C

¿Cómo ejecutar y detener mi aplicación CsdMessengerClient?

Para ejecutarla, en BlueJ seleccione la clase `ui.CsdMessengerClient`, pulse el botón derecho y elija el método `main()`, pasando como argumento el nombre de usuario que quiera.
Para detenerla simplemente cierre la ventana.

¿Cómo tratar las excepciones de JMS?

Varios métodos de la API de JMS lanzan excepciones y por ello cuando los use debería encerrar su código en una sentencia try-catch como la siguiente:

```
try {
      ……
} catch (NamingException | JMSException e) {
e.printStackTrace();
}
```

¿Cómo mostrar un mensaje en el panel de información de la aplicación cliente?

```
ui.API.addToLog(0, "mensaje a mostrar);
```

¿Qué clases y métodos de los proporcionados son los que debo modificar?

- Solo los métodos de la clase comm.Communication

¿Qué clases y métodos de los proporcionados son los que puedo llamar?

- Todos los métodos públicos estáticos de la clase `ui.API`
- Todas las clases y sus métodos públicos del paquete `messageBodies`

La documentación de estos métodos está disponible en el propio código de cada clase.