Sistemas Distribuidos
Grado en Ingeniería Informática en Sistemas de Información

Enseñanzas de Prácticas y Desarrollo
EPD 10: Java Message Service

Objetivos

1. Aprender a utilizar un middleware orientado a mensajes
2. Aprender a utilizar JMS
3. Implementar aplicaciones utilizando JMS

Conceptos

1. Introducción

JMS es la solución proporcionada por Java para implementar un middleware orientado a mensajes (MOM).

Hemos visto que, en las comunicaciones cliente servidor, los datos que se intercambian entre las dos partes necesitan de una comunicación síncrona, y que las dos partes estén activas en el momento de la comunicación. Los sistemas de mensajes aportan una serie de mejoras a la comunicación entre aplicaciones que no tienen por qué residir en la misma máquina.

JMS se sitúa como middleware en medio de la comunicación de dos aplicaciones. En entornos cliente servidor, cuando la aplicación A quiere comunicarse con la Aplicación B, necesita saber dónde está B (su IP por ejemplo) y que B esté escuchando en ese momento. Cuando se usa JMS (o cualquier otro sistema de mensajes), la aplicación A envía un mensaje, el sistema de mensajes lo recibe y se lo envía a B cuando se conecte al servicio. De esta manera se consigue una comunicación asíncrona entre A y B, y no hace falta que B este activo en el momento del envío del mensaje, y no por ello va a dejar de recibirlo.

La anterior es una de las ventajas de JMS, pero no la única. La comunicación anterior tiene dos extremos, el productor (A) y el consumidor (B). JMS soporta otro tipo de comunicaciones que se denomina Publisher/Subscriber (publicación/subscripción). En este tipo de comunicación, la aplicación A publica su mensaje en el servicio JMS y lo reciben todas las aplicaciones que entes suscritas al servicio JMS al que se envió el mensaje.

Otra de las ventajas de usar JMS (o cualquier otro MOM) es que las aplicaciones se pueden cambiar simplemente asegurándose que la nueva aplicación entiende los mensajes que se intercambian.

2. Arquitectura de JMS

Una aplicación JMS consta de los siguientes elementos:

- **Clientes JMS**: Aplicaciones que envían o reciben mensajes a través de JMS
- **Mensajes**: Los mensajes que se intercambian
- **Objetos administrados**: Los objetos JMS a los que se dirigen las comunicaciones

2.1 Objetos administrados

Los objetos administrados son el punto al que se comunican los clientes JMS para enviar o recibir mensajes, se denominan objetos administrados por que los crea el administrador (en la implementación de referencia mediante GlassFish). Implementan las interfaces JMS y se sitúan en el espacio de nombres de JNDI (Java Naming and Directory Interface) para que los clientes puedan solicitarlos. Así que estos objetos serán utilizados para intercambiar mensajes entre aplicaciones. JNDI proporciona la posibilidad de encontrar objetos especificando sus nombres en el espacio gestionado por JNDI, así que permite organizar y localizar componentes de un sistema distribuido.

Hay dos tipos de objetos administrados en JMS:

`ConnectionFactory`: se usa para crear una conexión al proveedor del sistema de mensajes.

`Destination`: es el destino de los mensajes que se envían y el recipiente de los mensajes que se reciben.

Más adelante veremos cómo podemos crear estos objetos. Cada objeto tendrá un nombre, que tendrá que ser conocido por las aplicaciones que necesitan utilizar ese objeto.

Los objetos administrados implementan interfaces JMS. Desde la versión 1.1., JMS define un conjunto común de interfaces que incorporan varios conceptos de mensajería. En versiones anteriores hay que utilizar interfaces especializadas para suportar las dos formas de comunicación: Point to Point (PTP) y publicación/suscripción.

Las interfaces comunes son:

- `ConnectionFactory`: Un objeto administrado que crea una conexión `Connection`.
- `Connection`: Una conexión activa con un proveedor de mensajes.
- `Destination`: Un objeto administrado que incorpora la identidad de un destino para los mensajes, tal como donde se envían los mensajes o desde donde se reciben.
- `Session`: Es un contexto que se utiliza para enviar y recibir mensajes. Solo se puede utilizar desde un solo thread.
- `MessageProducer`: Utilizado para enviar mensajes.
- `MessageConsumer`: Utilizado para recibir mensajes.

La siguiente tabla muestra las especializaciones de las interfaces para los tipos de comunicación:

Interfaz común	**Dominio PTP**	**Dominio pub/sub**
`ConnectionFactory`	`QueueConnectionFactory`	`TopicConnectionFactory`
`Connection`	`QueueConnection`	`TopicConnection`
`Destination`	`Queue`	`Topic`
`Session`	`QueueSession`	`TopicSession`
`MessageProducer`	`QueueSender`	`TopicPublisher`
`MessageConsumer`	`QueueReceiver`, `QueueBrowser`	`TopicSubscriber`

Nosotros utilizaremos las interfaces comunes.

ConnectionFactory

ConnectionFactory es un objeto administrado que se consigue de la JNDI con el fin de crear un proveedor de mensajes. Contiene un método createConnection(), que devuelve un objeto Connection.

Connection

Connection incorpora una conexión activa con un proveedor de mensajes. Algunos métodos son:

- createSession(boolean, int): Devuelve un objeto Session. El parámetro boolean indica si la sesión soporta transacciones o no, mientras que el parámetro int indica el modo de reconocimiento de mensajes (volveremos sobre estos dos puntos más adelante).
- start(): Activa la entrega de mensajes desde el proveedor.
- stop(): Para temporalmente la entrega de mensajes desde el proveedor. La entrega puede ser reanudada utilizando start().
- close(): Cierra la conexión con el proveedor.

Session

Session es el contexto para enviar y recibir mensajes. Sólo se puede utilizar desde un solo *thread.* Algunos de sus métodos son:

- createProducer(Destination): Devuelve un objeto MessageProducer que se puede utilizar para enviar mensaje al destino especificado por Destination.
- createConsumer(Destination): Devuelve un objeto MessageConsumer que se puede utilizar para recibir mensajes desde el destino especificado por Destination.
- commit(): Se utiliza si la sesión admite transacciones, en este caso termina la transacción y aplicando todas las operaciones para la transacción actual. .
- rollback(): Se utiliza si la sesión admite transacciones, en este caso deshace una transacción actual.
- create<MessageType>Message(...): Una variedad de métodos que devuelven un <MessageType>Message – por ejemplo, MapMessage, TextMessage, ...

Destination

Destination incorpora un destino para los mensajes. Es un objeto administrado y se consigue de la JNDI.

MessageProducer

MessageProducer se utiliza para enviar mensajes. Algunos de sus métodos son:

- send(Message): Envía el mensaje Message.
- setDeliveryMode(int): Configura la forma de entrega para los siguiente mensajes enviados; DeliveryMode.PERSISTENT y DeliveryMode.NON_PERSISTENT son valores aceptados, que especifican una comunicación persistente o no persistente, respectivamente.
- setPriority(int): Configura la prioridad para los mensajes enviados; la prioridad puede ir desde 0 hasta 9.
- setTimeToLive(long): Configura la caducidad, en milisegundos, de los mensajes enviados.

MessageConsumer

MessageConsumer se utiliza para recibir mensajes. Algunos de sus métodos son:

- receive(): Devuelve el siguiente mensaje en entrada. Este método bloquea el thread si no hay mensajes en cola.
- receive(long): Devuelve el siguiente mensaje en entrada que llega en los próximos long milisegundos; este método devuelve null si no llega ningún mensaje dentro del tiempo límite establecido.
- receiveNoWait(): Devuelve el siguiente mensaje en entrada. Si no hay mensajes disponibles devuelve null.
- setMessageListener(MessageListener): Configura un MessageListener; un objeto MessageListener recibe los mensajes conforme lleguen, es decir, de forma asíncrona.

MessageListener

MessageListener es una interfaz con un sólo método: onMessage(Message) que proporciona una forma de recibir y procesar mensaje de manera asíncrona.

Este interfaz debe ser implementada por una clase cliente y se pasará una instancia de esa clase al objeto MessageConsumer utilizando el método setMessageListener(MessageListener). Cuando llega un mensaje en un destino, el mensaje se pasa al objeto con el método onMessage(Message).

2.2 Mensajes

Los mensajes representan el núcleo del sistema de mensajes. Están formados por tres elementos:

`Header`: Es la cabecera del mensaje, contiene una serie de campos que le sirven a los clientes y proveedores a identificar a los mensajes. La lista de campos es la siguiente:

Campo	*Tipo de Dato*	*Descripción*
`JMSMessageID`	`String`	Un número que identifica unívocamente al mensaje. Sólo se puede consultar una vez que está enviado el mensaje.
`JMSDestination`	`Destination`	El destino al cual se envía el mensaje.
`JMSDeliveryMode`	`int`	Puede ser de tipo PERSISTENT, entonces se envía una única vez, o de tipo NON_PERSISTENT, de manera que se envía como mucho una vez, lo cual incluye también que no sea enviado nunca. PERSISTENT y NON_PERSISTENT están definidas como constantes.
`JMSTimestamp`	`long`	La hora a la que se envió el mensaje.
`JMSExpiration`	`long`	La hora hasta la cual el mensaje es válido, si es 0 quiere decir que no caduca nunca.
`JMSPriority`	`int`	Prioridad del mensaje de 0 a 9, siendo 0 la más baja.
`JMSCorrelationID`	`String`	Este campo se usa para relacionar una respuesta a un mensaje, se copia aquí el ID de mensaje del mensaje al que se está respondiendo.
`JMSReplyTo`	`Destination`	Especifica el lugar al cual se deben enviar las respuestas al mensaje actual.
`JMSType`	`String`	Este campo lo puede usar el programa de mensajería para almacenar el tipo del

		mensaje.
JMSRedelivered	boolean	Indica que el mensaje ha sido enviado con anterioridad pero el destino no lo ha procesado, por lo que se reenvía.

Properties: Son propiedades personalizadas para un mensaje en particular. Estas propiedades proporcionan la posibilidad de añadir campos de cabecera adicionales a los mensajes. Si una aplicación necesita categorizar o clasificar un mensaje de una forma que no es proporcionada por los campos de cabeceras estándar podemos añadir una propiedad al mensaje para lograr la categorización o clasificación requerida. Existen métodos set<Type>Property(...) y get<Type>Property(...) para establecer u obtener propiedades.

La lista de propiedades es la siguiente:

Propiedad	*Tipo de Dato*	*Descripcion*
JMSXUserID	String	El usuario que envía el mensaje.
JMSXApplID	String	La aplicación que envía el mensaje.
JMSXDeliveryCount	int	El número de veces que se ha intentado enviar el mensaje
JMSXGroupID	String	Identificador del grupo al que pertenece el mensaje.
JMSXGroupSeq	int	Número de secuencia en el grupo de mensajes.
JMSXRcvTimestamp	long	La hora a la que JMS le entregó el mensaje al/los destinatario/s.
JMSXState	int	Para uso del proveedor de mensajería.
JMSX_<nombre_del_proveedor>	-	Reservado para propiedades particulares del proveedor.

Body: Es el mensaje en sí, hay distintos tipos:

- StreamMessage: Contiene un stream de datos que se escriben y leen de manera secuencial.
- MapMessage: Contiene pares nombre-valor.
- TextMessage: Contiene un String.
- ObjectMessage: Contiene un objeto que implemente la interfaz Serializable.
- BytesMessage: Contiene un stream de bytes.

2.3 Transacciones

Las transacciones (transactions) son un mecanismo por el cual una serie de operaciones se consideran como un todo, lo que quiere decir que o se ejecutan todas correctamente, o no se ejecuta ninguna. Un ejemplo típico de uso de transacciones son las bases de datos de los bancos, cuando se hace un ingreso hay que actualizar varias tablas correctamente o si no la información contenida en ellas no sera consistente. Imaginemos un traspaso de una cuenta a otra, hay dos operaciones básicas, sacar de una cuenta e ingresar el dinero en otra, si se saca de una cuenta y no se llega a meter en otra tenemos un problema, por lo que el traspaso se considera como una transacción, o se completan todos los pasos con éxito o se vuelve atrás, al punto donde se inició la transacción. Las transacciones tienen dos operaciones básicas, commit, que crea una nueva transacción (y termina correctamente la anterior), y rollback, que anula una transacción y deshace las operaciones realizadas durante la misma.
Los objetos Session controlan las transacciones, y podemos indicar que una Session soporta transacciones en el momento en que se crea el objeto. Una Session que soporta transacciones siempre tiene una transacción actual, es decir, no hay un begin(), las operaciones commit() y rollback() terminan una transacciones y automáticamente crean otra.

2.4 Acknowledgment

Mediante el reconocimiento de mensajes (o en inglés *Message Acknowledgement*) un mensaje enviado no se elimina de donde lo tenga guardado la JMS hasta que no sea reconocido por el consumidor de los mensajes. Este reconocimiento se realiza llamando al método acknowledge() del mensaje. Para que sea el cliente el que reconozca los mensajes y no la JMS, hay que crear una sesión con la constante CLIENT_ACKNOWLEDGE, como parámetro. En general hay tres tipos de reconocimiento de mensajes:

1. Session.DUPS_OK_ACKNOWLEDGE: reconocimiento de entrega de mensaje *lazy*. Se debe utilizar sólo si se pueden gestionar mensajes duplicados.
2. Session.AUTO_ACKNOWLEDGE: Se envía un reconocimiento automático de entrega de mensajes al ejecutar un método que reciba un mensaje.

3. Session.CLIENT_ACKNOWLEDGE: La recepción de mensaje se comunica explícitamente llamando al método acknowledge() de Message.

2.3 Clientes JMS

Son clientes de JMS tanto el que suministra mensajes como el que los recibe, así que no tenemos la distinción cliente servidor. Todos los clientes tienen que seguir una serie de pasos antes de lograr enviar o recibir un mensaje:

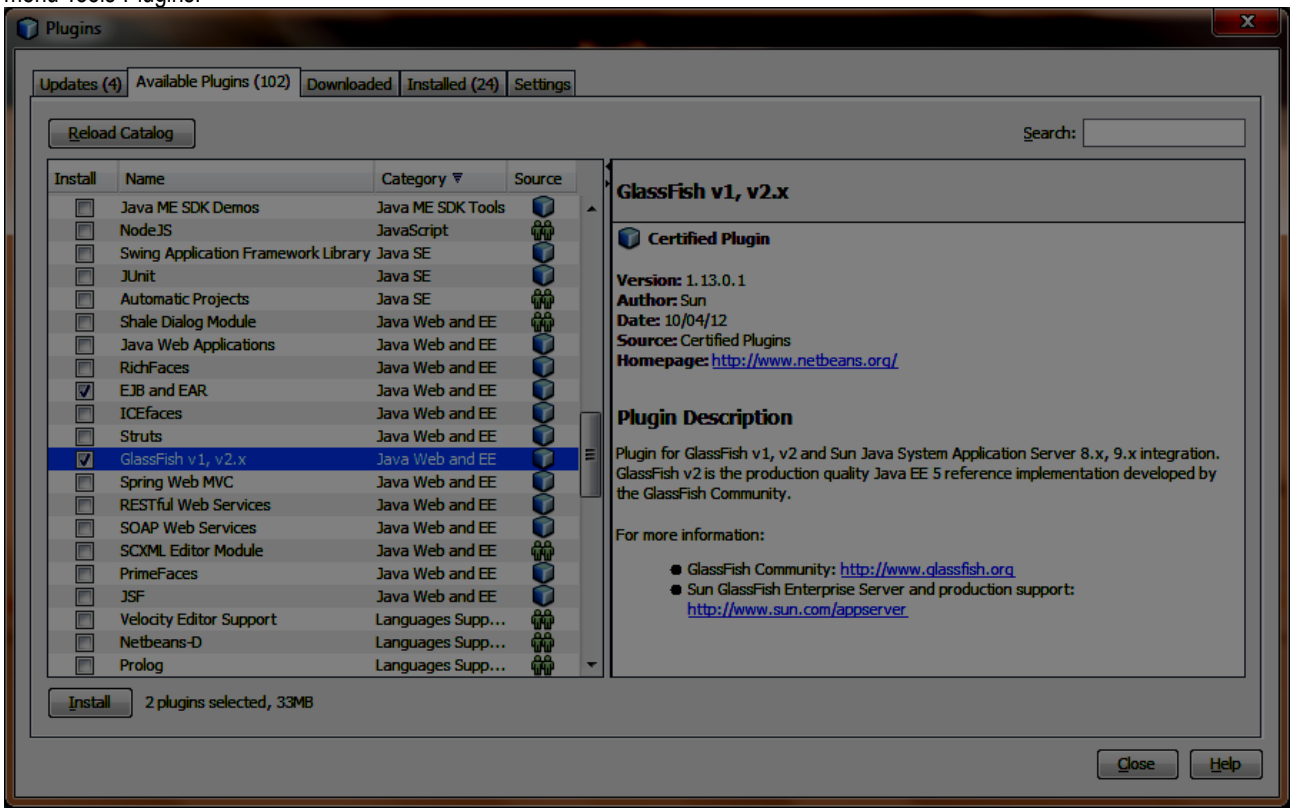
- Conseguir un objeto `ConnectionFactory` a través de `JNDI`.
- Conseguir un destino, mediante el objeto `Destination` a través de `JNDI`.
- Usar `ConnectionFactory` para conseguir un objeto `Connection`
- Usar `Destination` para crear un objeto `Session`.
- Utilizar `Session` y `Destination` para crear los MessageProducer y MessageConsumer necesarios.
- Arrancar la conexión Connection.

Después de ejecutar estos pasos, se podrán intercambiar mensajes, y la aplicación podrá recibir, procesar y enviar mensajes.

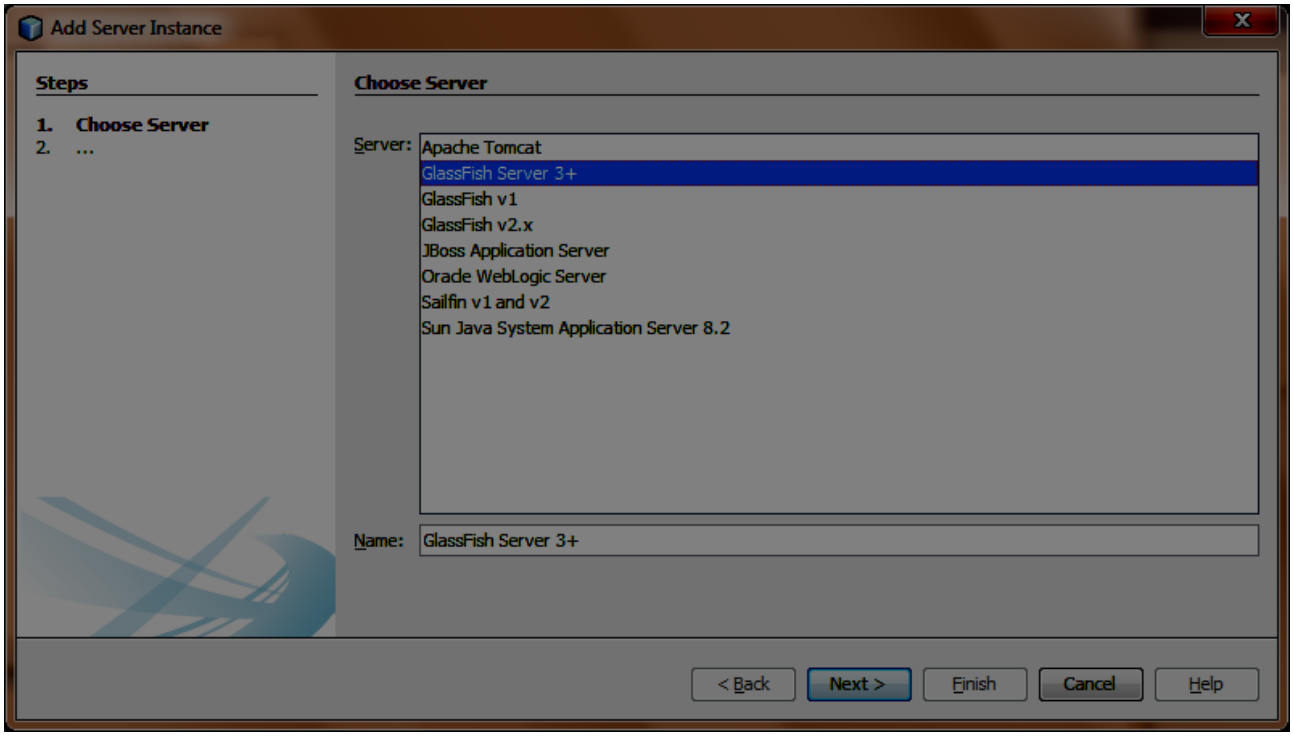
4. Configuración Netbeans

En esta práctica, utilizaremos el entorno de programación Netbeans (se puede utilizar tanto en Windows como en Linux). Hay dos posibilidades, o instalar el IDE que ya incluya soporte para Java EE y el servidor Glassfish (la versión para Windows se puede encontrar en la sección "Recursos" de la WebCT) o configurar una copia estándar.

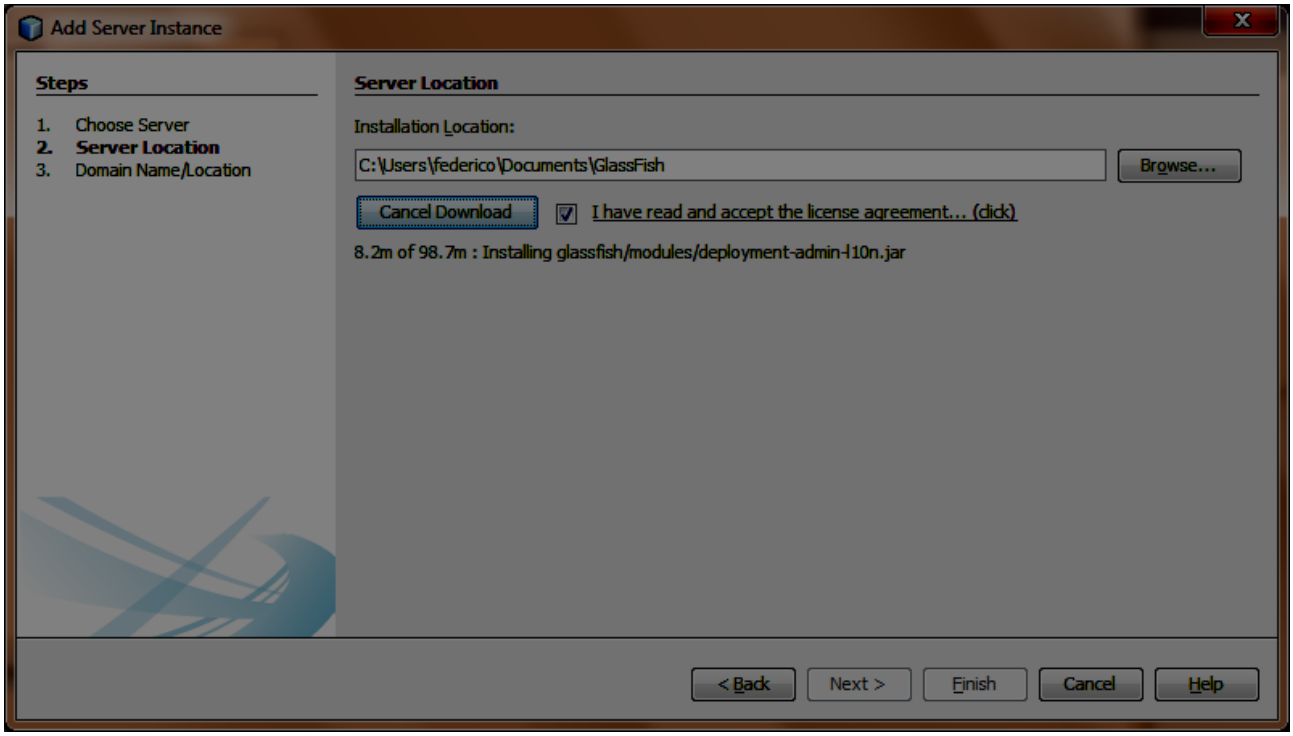
Estas instrucciones se han probado en Netbeans 7.1.2, ya que la versión 7.2 todavía tiene problemas con el plugins de GlassFish. Para configurar un copia de Netbeans, primero tendremos que instalar los plugins "EJB y EAR" y "Glassfish v1, v2.x", desde el menú Tools-Plugins.



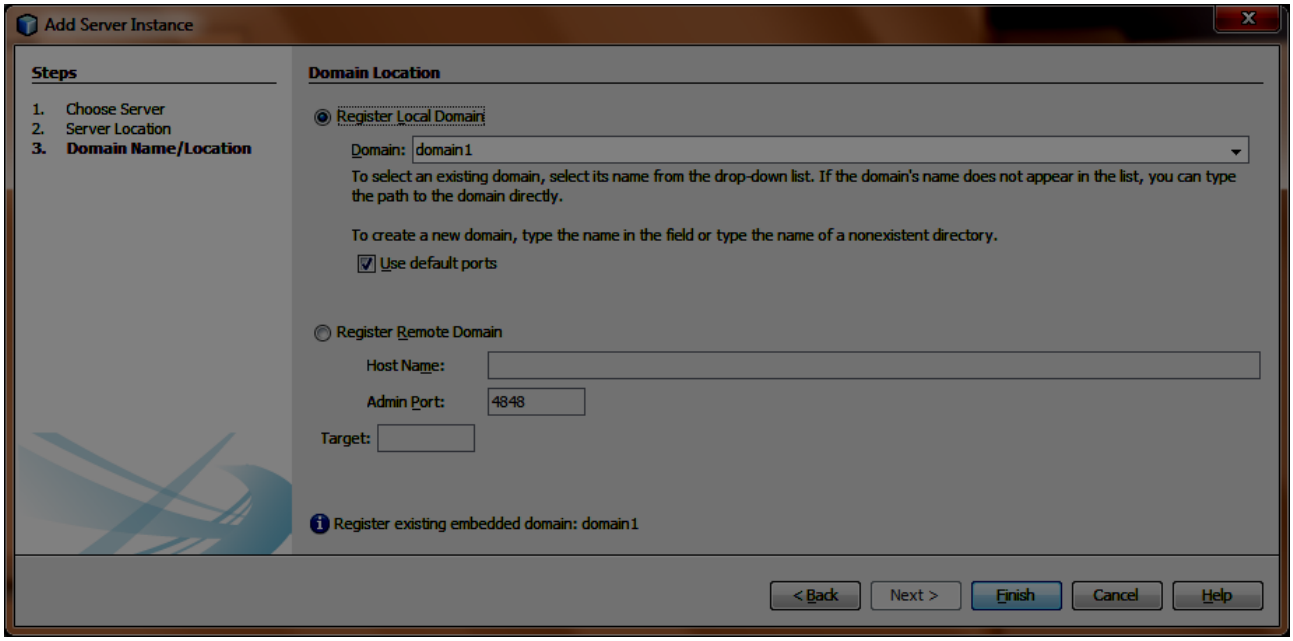
Tendremos luego que añadir el servidor GlassFish, desde el menú Tools-Servers. Pulsamos el boton "Add Server..." y seleccionamos la opción "GlassFish Server 3+"



En la siguiente pantalla elegimos un directorio de destino y elegimos descargar la versión 3.0.1, que se instalará automáticamente en la carpeta especificada en “Installation Location”.



Al finalizar la instalación nos aparecerá la siguiente ventana:



Dejamos todo igual y pulsamos al botón “Finish”.

5. Aplicaciones PTP

En las conexiones punto a punto sólo hay dos extremos en la comunicación, es decir, el que envía y el que recibe. La comunicación es gestionada por medio de una **Cola de mensajes** (en inglés *Queue*) y actúa como una cola FIFO (el primer mensaje que llega es el primer mensaje que se recoge). El que envía enviará su mensaje a la cola, y el que recibe sacará los mensajes desde la cola.

Vamos a seguir los pasos que necesita un cliente JMS para conseguir una conexión a una cola de mensajes.

- Nuestra aplicación deberá importar los paquetes `javax.jms.*` y `javax.naming.*`;
- Primero debemos crear un contexto inicial para `JNDI`:

```
InitialContext contextoInicial = new InitialContext();
```

- Recuperamos de este contexto el objeto `ConnectionFactory`

ConnectionFactory factory = `(ConnectionFactory)contextoInicial.lookup(factoryName);`
Donde `factoryName` es una cadena que representa el nombre de la factoría de conexión utilizada para esta aplicación.

- Recuperamos de la `JNDI` la cola de mensajes

```
Destination cola = (Destination)contextoInicial.lookup(queueName);
```

Donde `queueName` es el nombre de la cola de mensajes que vamos a utilizar.

- Ahora la aplicación deberá crear los objetos JMS necesario para el envío de mensajes. Mediante el método `createConnection()` de `factory` (objeto de tipo `ConnectionFactory`) conseguimos un objeto `Connection`.

```
Connection conexion = factory.createConnection();
```

- El objeto anterior nos sirve ahora para crear una sesión:

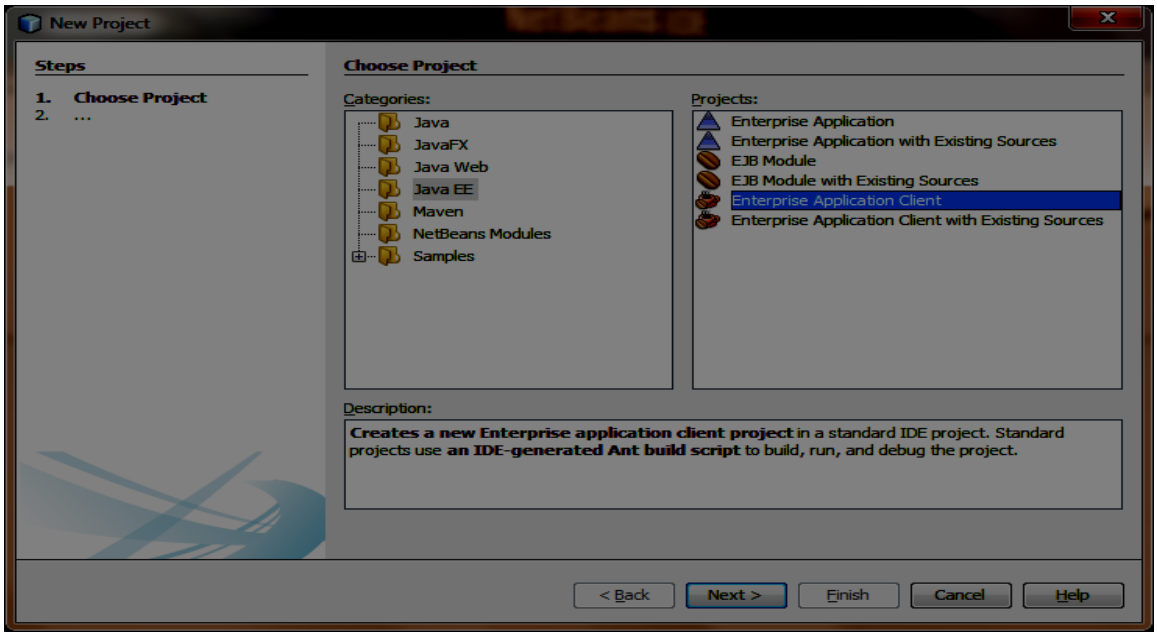
```
Session sesion =
          conexion.createSession(false,sesion.AUTO_ACKNOWLEDGE);
```

con el primer parámetro establecemos que la sesión no soporta transacciones, y que los mensajes son reconocidos automáticamente.

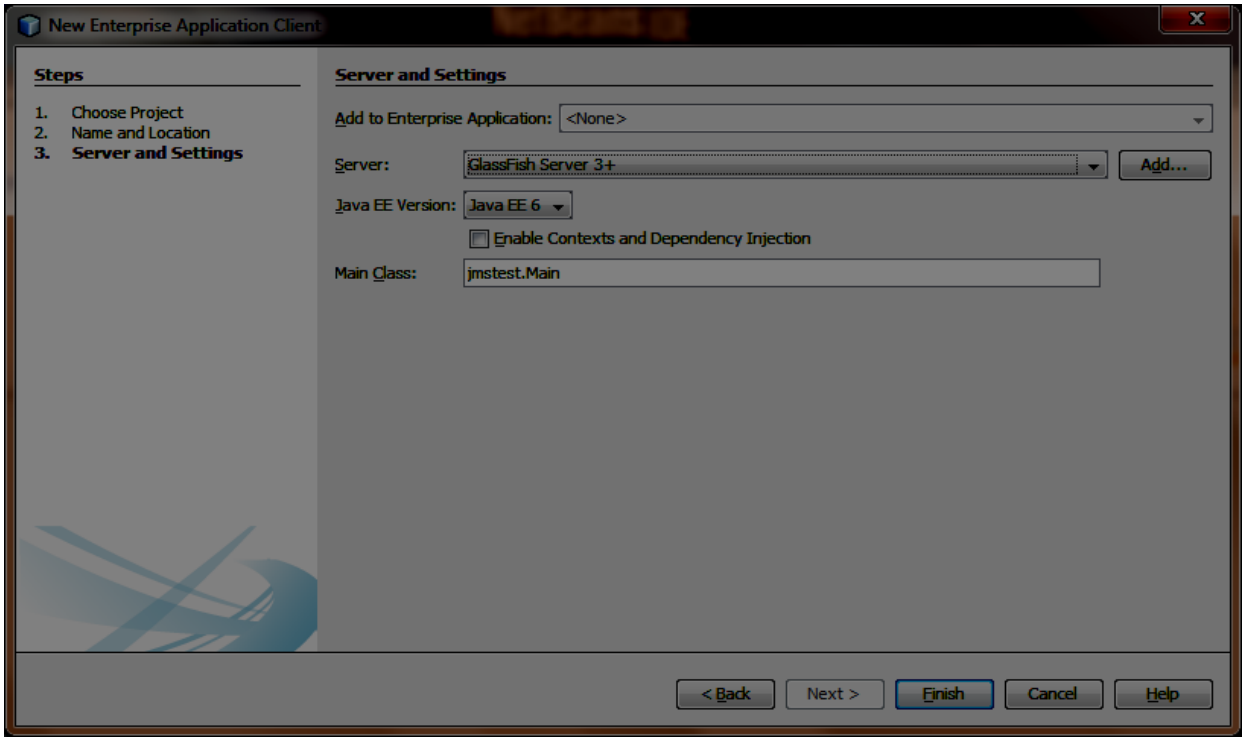
A partir de ahora ya podríamos crear mensajes y enviarlos. El experimento 1 proporciona un ejemplo de aplicación que envía mensaje en un dominio PTP.

7. Crear Aplicaciones con Netbeans

Con Netbeans, podemos crear una aplicación que use JMS con File -> New Project -> Java EE -> Enterprise Application Client



Después de haber elegido el nombre del proyecto, nos aparecerá la siguiente ventana:



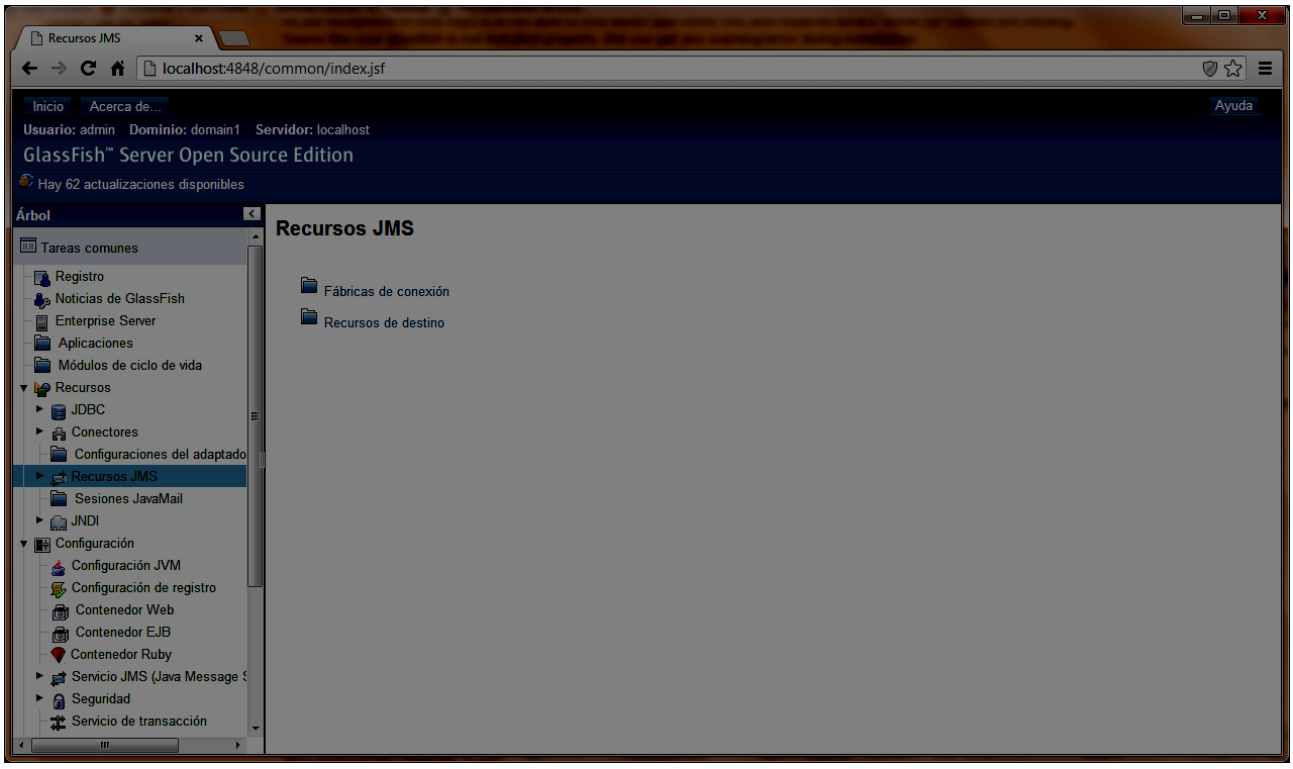
Pulsamos el botón Finish.

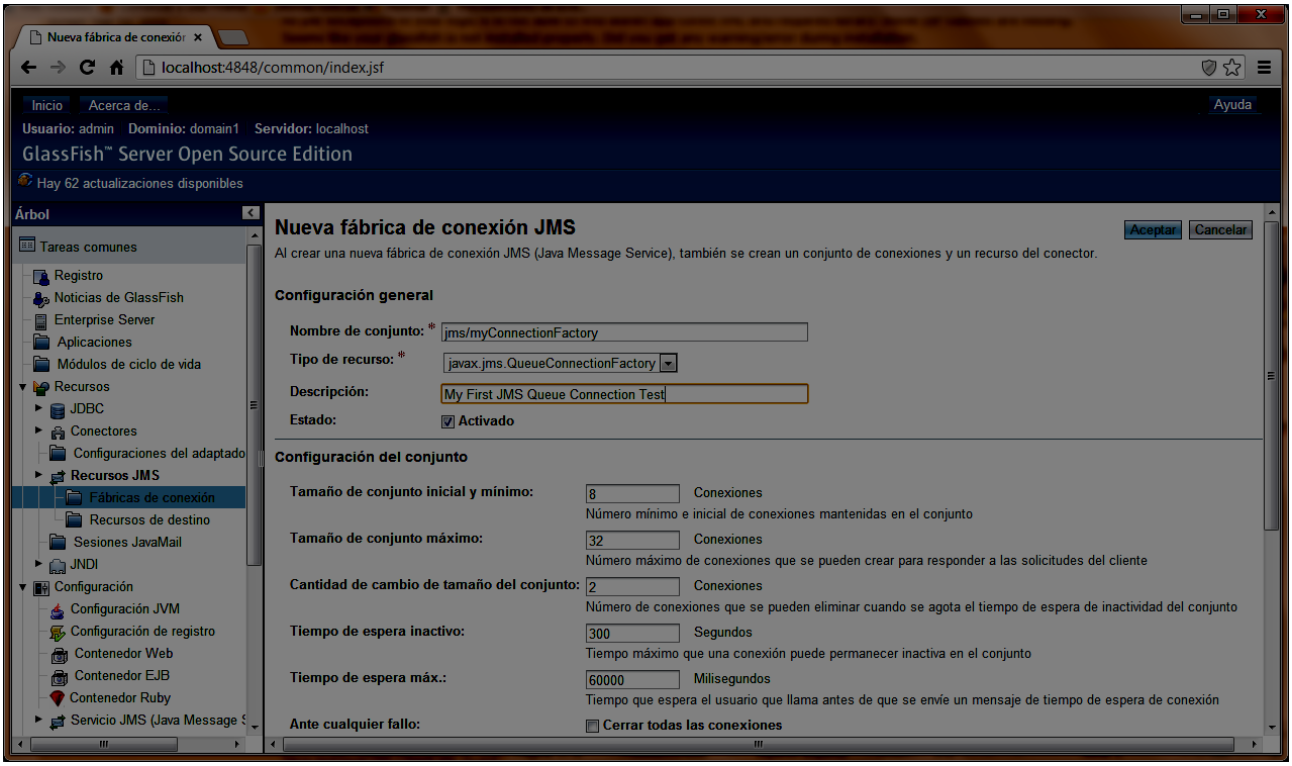
Para una aplicación PTP hay que crear por lo menos una fábrica de conexiones y una cola de mensajes. Podemos hacerlo utilizando la consola de administración de GlassFish.

Para arrancar la consola de administración, desde la pestaña “Services” de Netbeans, pulsamos con el botón derecho sobre "GlassFish Server 3+" y luego en "Start", lo cual arrancará el servidor.

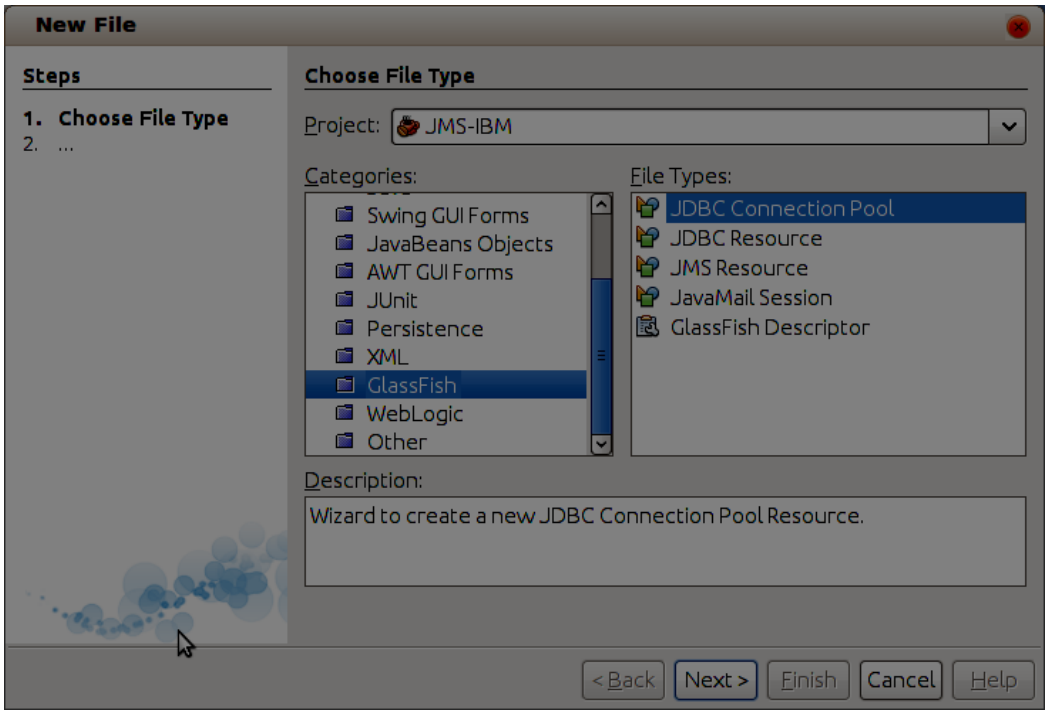
Luego seguimos estos pasos:

- Pulsamos otra vez con el botón derecho sobre "GlassFish Server 3+" y luego pulsamos en "View Domain Admin Console"
- Accedemos a la consola
- Seleccionamos Recursos -> Recursos JMS-> Fábricas de conexión
- Pulsar [Nuevo...] para crear una nueva Fábrica de conexión JMS
- Nombre de conjunto: miFabrica
- tipo de recursos: javax.jms.QueueConnectionFactory
- Descripción: My First JMS Queue Connection Test
- Estado: Activado (checked)
- Pulsar en [Aceptar]
- Seleccionamos Recursos -> Recursos JMS → Recursos de destino
- Pulsar [Nuevo...] para crear un nuevo Recursos de destino JMS
- Nombre JNDI: myDestination
- Nombre de destino fisico: miDestino
- Tipo de recurso: javax.jms.Queue
- Descripción: My First JMS Queue Test
- Estado: Activado (checked)
- Pulsar [Aceptar]





Los objetos administrados se pueden crear también desde Netbeans. Pulsando con el botón derecho sobre el proyecto que hemos creado, seleccionamos “New File...” y a continuación seleccionamos “Other...” Nos aparecerá está ventana:



Seleccionamos GlassFish y JMS Resource. Pulsamos en Next y luego el procedimiento es similar al descrito anteriormente.

Una aplicación deberá tener por lo menos dos clases, una para recibir mensajes y otra que envíe mensajes. Estas clases podrán tener un método main que ejecuta los métodos de envío y recepción, o se utilizarán para crear objetos y luego invocando

explícitamente los métodos de envío y recepción de mensajes. El experimento 1 proporciona un ejemplo de clase que envía mensajes.

En los laboratorios puede que el puerto 8080 se esté utilizando, y en este caso GlassFish no arrancaría. Se puede comprobar poniendo la dirección http://localhost:8080/ en un navegador y comprobando si hay un servicio ejecutándose. Puede que haya un servidor de openERP, en este caso tendremos que parar el servicio.

8. Aplicaciones Pub/Sub

Una aplicación pub/sub se implementa de forma similar a una aplicación PTP. La diferencia es que los objetos administrados que hay que crear son de tipo TopicConnectionFactory y Topic.

La diferencia entre una aplicación pub/sub y una PTP es que en las PTP, si hay más de un receptor de mensaje para la misma cola, sólo uno de ellos recibirá el mensaje, mientras que en una aplicación pub/sub todos los receptores subscritos a esa cola recibirán copia del mensaje.

Bibliografía

1. Tutorial JMS http://docs.oracle.com/javaee/1.3/jms/tutorial/
2. Introducción de IMB a JMS http://www.ibm.com/developerworks/java/tutorials/j-jms/
3. JMS API http://docs.oracle.com/javaee/1.4/api/javax/jms/package-summary.html

Experimentos

E1. Considere la siguiente clase que envía mensaje a una cola de mensajes:

```
import java.io.*;
import javax.jms.*;
import javax.naming.*;

public class Sender  {

    public static void main(String[] args) {

        new Sender().send();
    }


    public void send() {

        BufferedReader reader = new BufferedReader(new InputStreamRead-
er(System.in));

        try {
            String destinationName = "jms/myDestination";
            String factoryName = "jms/myConnectionFactory";

            //Look up administered objects
            InitialContext initContext = new InitialContext();
            ConnectionFactory factory =
                    (ConnectionFactory) initContext.lookup(factoryName);
            Destination destination = (Destination) initCon-
text.lookup(destinationName);
            initContext.close();

            //Create JMS objects
            Connection connection = factory.createConnection();
```

```
            Session session =
                    connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
            //MessageProducer sender = session.createProducer(queue);
            //cambiato queue con destination
            MessageProducer sender = session.createProducer(destination);

            //Send messages
            String messageText = null;
            while (true) {
                System.out.println("Enter message to send or 'quit':");
                messageText = reader.readLine();
                if ("quit".equals(messageText)) {
                    break;
                }
                TextMessage message = session.createTextMessage(messageText);
                sender.send(message);
            }

            //Exit
            System.out.println("Exiting...");
            reader.close();
            connection.close();
            System.out.println("Goodbye!");

        } catch (Exception e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Analice el código para entender el procedimiento utilizado. Recuerde que para que este código funcione correctamente, tenemos que crear primero una fábrica de conexión y un recurso de destino JMS. Además los nombres de los objetos administrados tendrán que ser utilizados para inicializar las cadenas `destinationName` y `factoryName`, respectivamente.

Problemas

P1) Implementar una clase que reciba mensajes desde la clase implementada en el experimento 1. La clase deberán implementar la interfaz `MessageListener` y utilizar el método `setMessageListener` de `MessageConsumer` de esta forma `receiver.setMessageListener(this);` donde `receiver` es un objeto de tipo `MessageConsumer`.

P2) Crear los objetos administrados necesarios para la ejecución del código implementado en el experimento 1 y en el problema 1. Ejecutar la aplicación.

Ampliación de Problemas

AP1) Implementar una aplicación publicación/subscripción. Esta aplicación simulará una chat, donde los clientes recibirán solo los mensajes sobre temas en los cuales están interesados.

Consideraciones de Desarrollo

Se pide desarrollar la aplicación descrita la ampliación de problemas 1. La entrega de la aplicación resultante se efectuará utilizando la tarea en Campusvirtual creada para tal propósito. El código tendrá que ser entregado siguiendo las normas de entregas especificadas a continuación.

NORMAS DE ENTREGA

LEA DETENIDAMENTE LAS NORMAS DE LA PRUEBA

Cualquier incumplimiento de las normas significará una respuesta nula en el examen y por tanto una valoración de 0 puntos.

1. La resolución de los ejercicios propuestos se entregará en un único archivo formato ZIP que contendrá EXCLUSIVAMENTE los archivos que compongan la solución a los problemas planteados.

2. El nombre del fichero tendrá un formato específico dictado por el nombre de cada alumno. Por ejemplo, para un alumno llamado "José María Núñez Pérez" el fichero se nombrará como NunyezPerezJM.zip. Obsérvese que las tildes son ignoradas y las eñes sustituidas.

3. El fichero se subirá utilizando la correspondiente tarea en CampusVirtual

IMPORTANTE: Cualquier envío que no respete el formato de compresión o el nombre adecuado será ignorado y, por tanto, valorado con cero puntos.