Aplicaciones con Enfoque Orientado a Servicios

Tema Nº4:Java Message Service (JMS)

Indicador de logro Nº4:Implementa middleware JMS utilizando un modelo específico

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº4:**

Java Message Service (JMS)

**Subtema 4.1:**

Arquitectura JMS

El Java Message Service (JMS) se diseñó para facilitar el desarrollo de aplicaciones comerciales que envían y reciben de forma asincrónica datos y eventos comerciales. Define una API de mensajería empresarial común que está diseñada para ser compatible de manera fácil y eficiente con una amplia gama de productos de mensajería empresarial.

JMS define un conjunto de interfaces y semánticas que permiten que las aplicaciones Java se comuniquen con otras implementaciones de mensajería. JMS facilita la curva de aprendizaje minimizando el conjunto de conceptos que un desarrollador de Java debe aprender a utilizar productos de mensajería empresarial y, al mismo tiempo, maximiza la portabilidad de las aplicaciones de mensajería.

JMS admite los siguientes modelos de mensajería:

* Punto a punto (P2P o cola)

Bajo este modelo, un mensaje se consume por un único consumidor (1:1), pero puede haber varios emisores. El destino del mensaje es una cola definida que será atendido siguiendo el modelo FIFO, en el cual el mensaje encolado será el primero en salir de la cola.

* Publicación - suscripción (Pub/Sub)

En este modelo, un mensaje puede consumirse por múltiples consumidores (1:N). El destino de un mensaje se conoce como tópico. Un tópico no funciona como una pila, ya que los mensajes en los tópicos no se encolan. De hecho, un nuevo mensaje en el tópico sobrescribirá cualquier mensaje existente. Así pues, bajo este modelo de difusión, los productores/emisores/publicadores publican el mensaje en un tópico, y los consumidores se subscriben al tópico.

Para estandarizar el API, JMS define de un modo formal muchos conceptos y elementos del mundo de la mensajería:

* Cliente JMS

Una aplicación 100% Java que envía y recibe mensajes. Cualquier componente JavaEE puede actuar como un cliente JMS.

* + Clientes No-JMS: una aplicación escrita en un lenguaje que no es Java que envía y recibe mensajes.
  + Productor JMS: una aplicación cliente que crea y envía mensajes JMS.
  + Consumidor JMS: una aplicación cliente que recibe y procesa mensajes JMS.
* Proveedor JMS

Implementación de los interfaces JMS el cual está idealmente escrito 100% en Java. El proveedor debe ofrecer prestaciones tanto de administración como de control de los recursos JMS. Toda implementación de la plataforma Java incluye un proveedor JMS.

* Mensaje JMS

Es el elemento principal de JMS; objeto (cabecera + propiedades + cuerpo) que contiene la información y que es enviado y recibido por clientes JMS.

* Dominio JMS

Los dos estilos de mensajería: P2P y Pub/Sub.

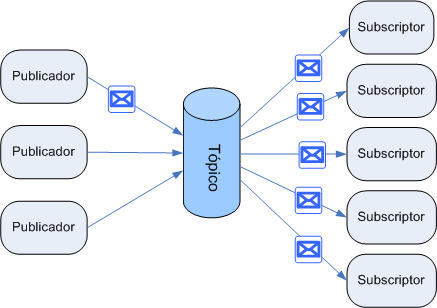
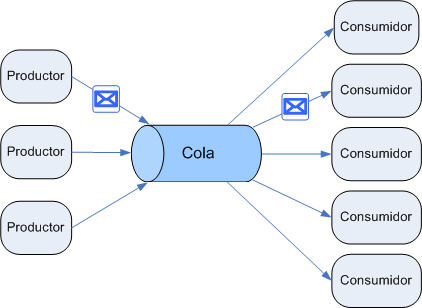
* Objetos Administrados

Objetos JMS preconfigurados que contienen datos de configuración específicos del proveedor, los cuales utilizarán los clientes. Los clientes acceden a estos objetos mediante JNDI.

* + Factoría de Conexión: los clientes utilizan una factoría para crear conexiones al proveedor JMS.
  + Destino: objeto (cola/tópico) al cual se direccionan y envían los mensajes, y desde donde se reciben los mensajes.

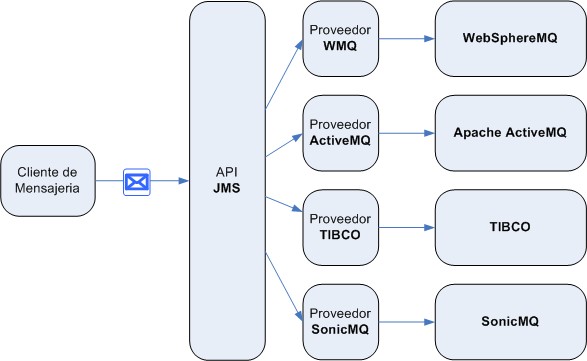
**Ejemplos:**

**Modelo Punto a Punto (cola o P2P) vs Publicación – Suscripción (Pub/Sub)**

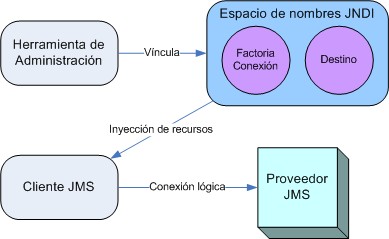


Modelo P2P Modelo Pub/Sub

**Forma de trabajo con JMS**



**Arquitectura JMS**



**Subtema 4.2:**

Modelo de programación JMS

Una aplicación JMS consta de un conjunto de mensajes definidos por la aplicación y un conjunto de clientes que los intercambian. Los clientes JMS interactúan enviando y recibiendo mensajes utilizando la API JMS. Un mensaje se compone de tres partes: encabezado, propiedades y cuerpo.

* El encabezado, que es necesario para cada mensaje, contiene información que se utiliza para enrutar e identificar mensajes. Algunos de estos campos los establece automáticamente el proveedor de JMS durante la producción y entrega de un mensaje, y otros los establece el cliente mensaje por mensaje.
* Las propiedades, que son opcionales, proporcionan valores que los clientes pueden utilizar para filtrar mensajes. Proporcionan información adicional sobre los datos, como qué proceso los creó y la hora en que se crearon. Las propiedades se pueden considerar como una extensión del encabezado y constan de pares de nombre / valor de propiedad. Con las propiedades, los clientes pueden ajustar su selección de mensajes especificando ciertos valores que actúan como criterios de selección.
* El cuerpo, que también es opcional, contiene los datos reales que se intercambiarán. La especificación JMS definió seis tipos o clases de mensajes que un proveedor JMS debe admitir:
  + *Message*

Esto representa un mensaje sin cuerpo de mensaje.

* + *StreamMessage*

Un mensaje cuyo cuerpo contiene una secuencia de tipos primitivos de Java. Está escrito y leído secuencialmente.

* + *MapMessage*

Un mensaje cuyo cuerpo contiene un conjunto de pares de nombre / valor. El orden de las entradas no está definido.

* + *TextMessage*

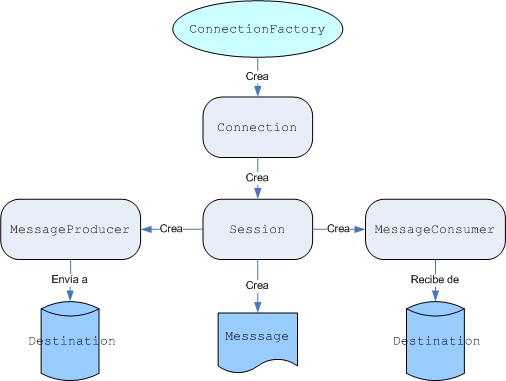
Un mensaje cuyo cuerpo contiene una cadena Java, como un mensaje XML.

* + *ObjectMessage*

Un mensaje cuyo cuerpo contiene un objeto Java serializado.

* + *BytesMessage*

Un mensaje cuyo cuerpo contiene un flujo de bytes no interpretados.



Esquema de trabajo con las interfaces JMS

**Subtema 4.3:**

Modelos específicos

Los dos modelos específicos son API Punto a Punto y Publicación / Suscripción, que veremos más a detalle.

**API Punto a Punto**

Se refiere de manera específica a los interfaces basados en el uso de colas, de modo que los interfaces de este API son:

* QueueConnectionFactory
* Queue
* QueueConnection
* QueueSession
* QueueSender
* QueueReceiver

Igual que en modelo general JMS, obtendremos tanto los objetos QueueConnectionFactory como Queue del proveedor JMS via JNDI (mediante la inyección de código vía anotaciones). Como puede observarse, la mayoria de los interfaces añaden el sufijo Queue al nombre del interfaz JMS. Las excepciones son el interfaz Destination, que se llama Queue, y los interfaces MessageProducer y MessageConsumer que pasan a ser QueueSender y QueueReceiver, respectivamente.

Las aplicaciones que usan un modelo de mensajería punto a punto normalmente utilizan este API específico en vez del API general.

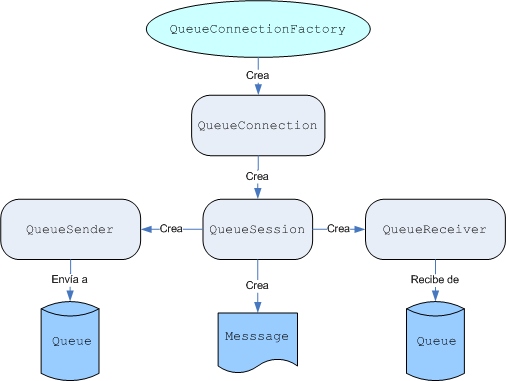
**API Publicación/Subscripción**

Cada elemento específico para las colas, tiene su correspondencia con el uso de tópicos. Así pues, tendremos los siguientes interfaces:

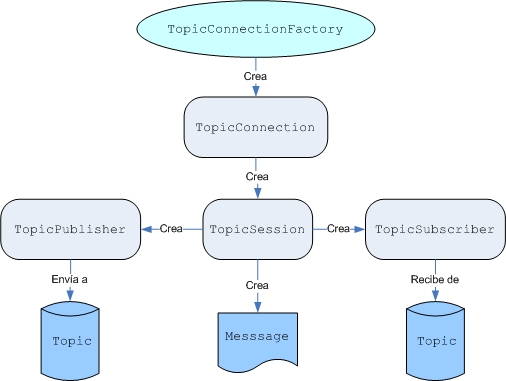
* TopicConnectionFactory
* Topic
* TopicConnection
* TopicSession
* TopicPublisher
* TopicSubscriber

**Ejemplos:**

**API Punto a Punto**



**API Punto a Punto**



**Subtema 4.4:**

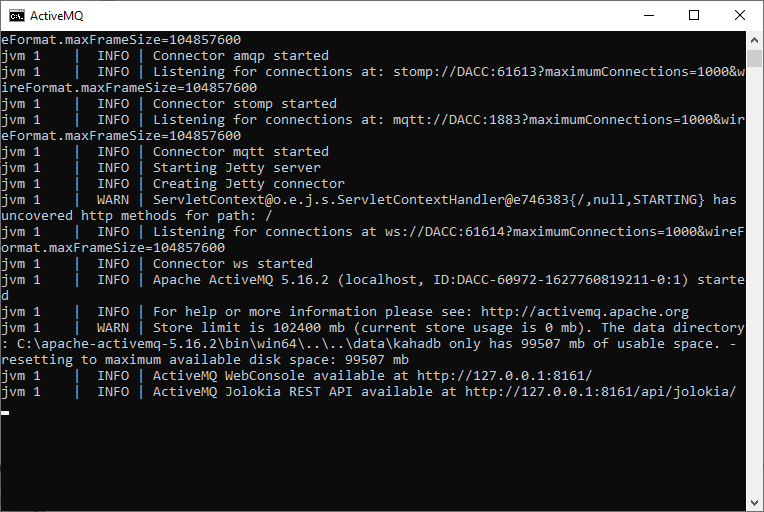
Apache ActiveMQ

ActiveMQ es un bróker de mensajería de código abierto de Apache escrito en Java junto con un cliente de servicio de mensajes Java completo (JMS). En otras palabras, es un programa que ayuda a traducir, dirigir, persistir y entregar mensajes que una aplicación fuente (productor) envía a un destino (consumidor). Es como un middleware de mensajería. Esto resulta muy útil cuando se trata de aplicaciones de comunicación de grandes infraestructuras en crecimiento. Existen diferentes tipos de bróker de mensajería, tales como RabbitMQ, IBM MQ, JBoss Messaging, ActiveMQ, etc.

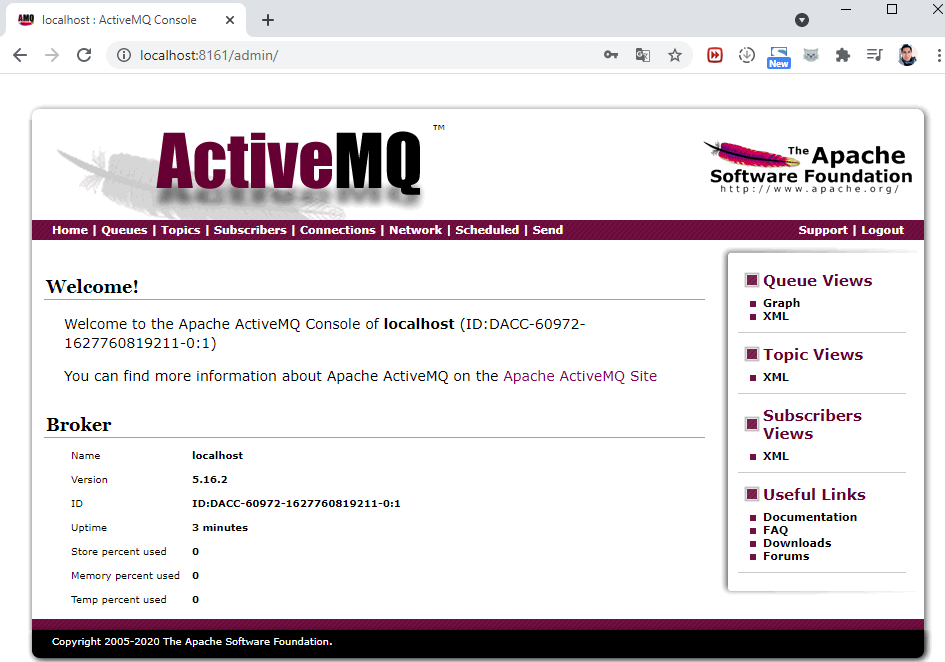
**Ejemplos:**

**Inicio de ActiveMQ**

1. Ingrese a la carpeta del terminal */ bin /* winXX.
2. Abra la terminal de ActiveMQ haciendo doble clic en el archivo **activemq.bat**

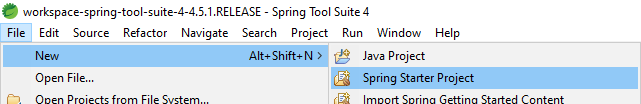


1. En base a la información que muestra la consola, acceda a la IP [**http://127.0.0.1:8161/admin**](http://127.0.0.1:8161/admin) a través de su navegador. En caso de solicitarle credenciales de usuario y contraseña, ingrese **admin** para ambos casos.

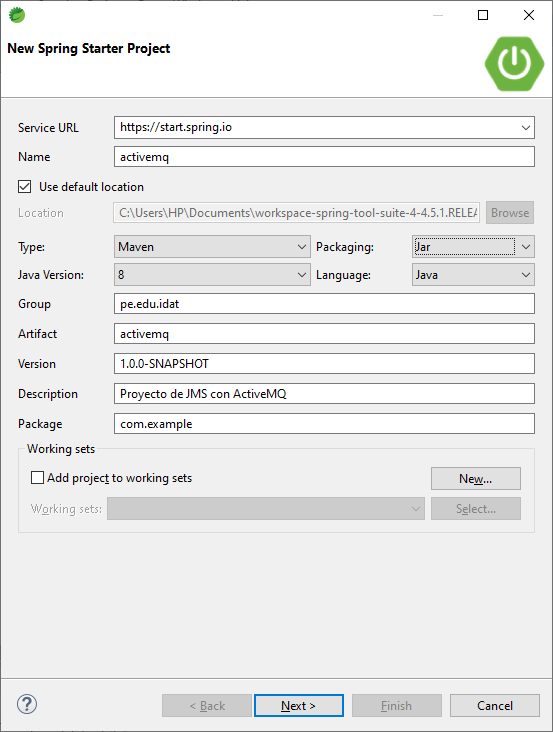


**Creación y configuración de proyecto**

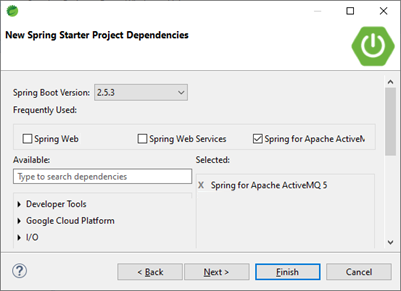
1. En Spring Tool Suite vaya al menú *File > New* y seleccione la opción *Spring Starter Project*.



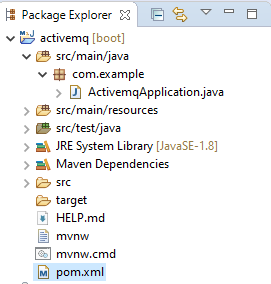
1. Cree el proyecto **restws** con la siguiente configuración y haga clic en Siguiente:



1. Agregue las dependencias **Spring for Apache ActiveMQ**. Luego, haga clic en Siguiente.



1. En la opción de *Site Info*, haga clic en Finalizar y verifique su proyecto.

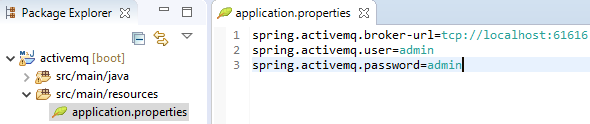


1. En el archivo **pom.xml**, añada la dependencia a Gson.



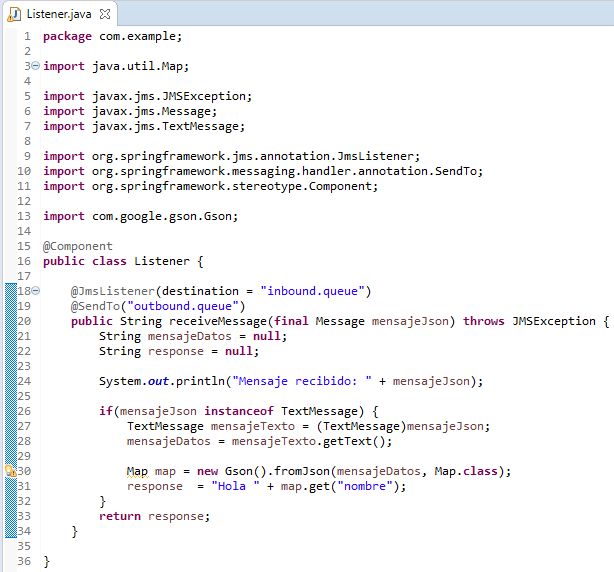
**Configuración de ActiveMQ en Spring Boot**

1. En el archivo **application.properties** añada la URL (a través del protocolo TCP), el usuario y contraseña.



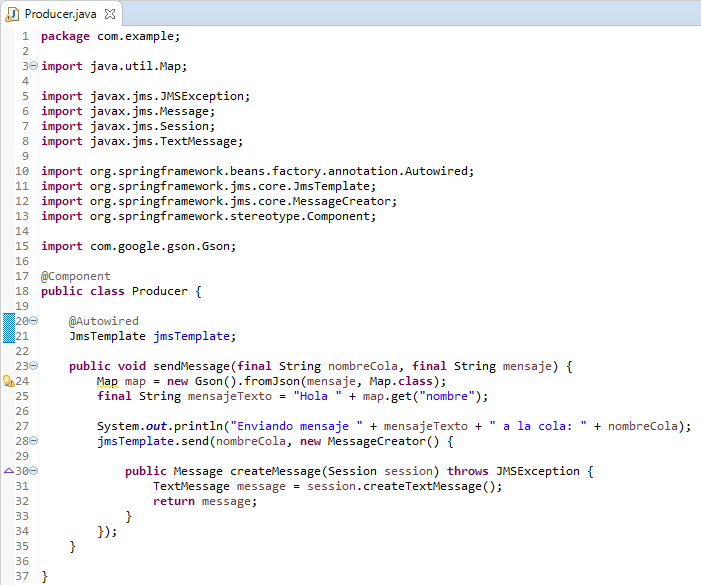
**JMS Listener**

1. Agregue la clase java **Listener** que servirá para implementar el método de escucha de mensajes JMS de la cola de entrada y procesarlo. A través de la anotación @SendTo enviaremos un nuevo mensaje a la cola de salida



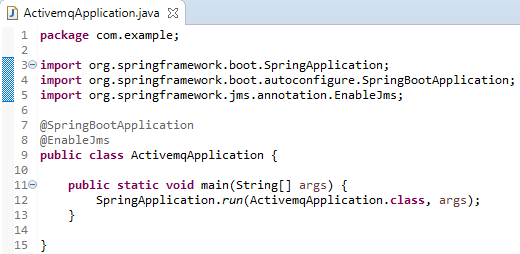
**JMS Producer**

1. Agregue la clase java **Producer** que generará un mensaje de salida.



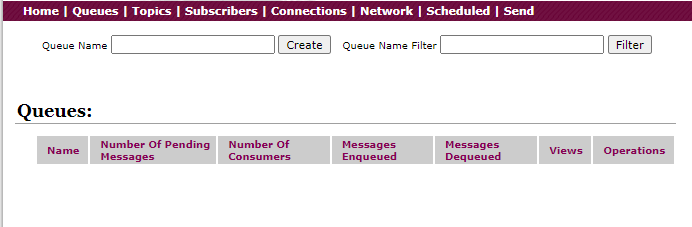
**Configuración de Application**

1. Agregue la notación @EnableJms a la clase **ActivemqApplication** para habilitar la mensajería.

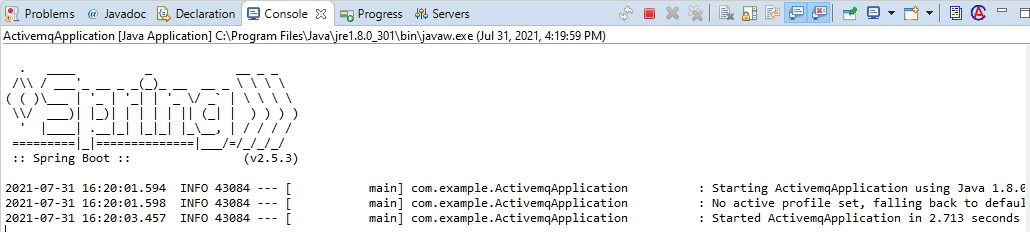


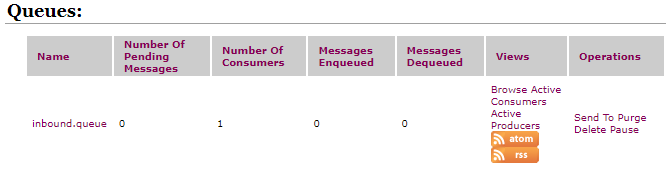
**Pruebas**

1. En la consola web de ActiveMQ verifique que no existan colas (*Queues*).



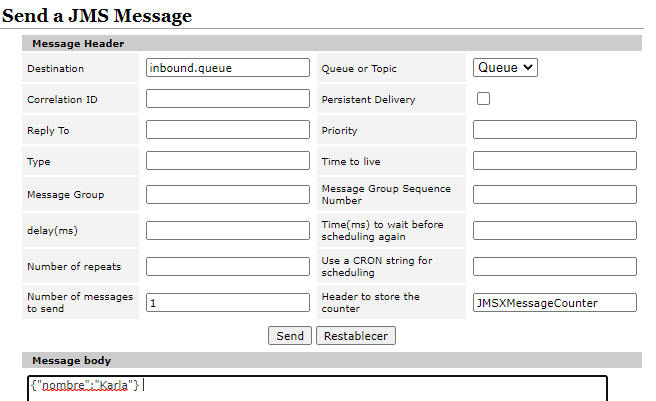
1. Ejecuta la aplicación como Java Application y actualice la vista de colas.



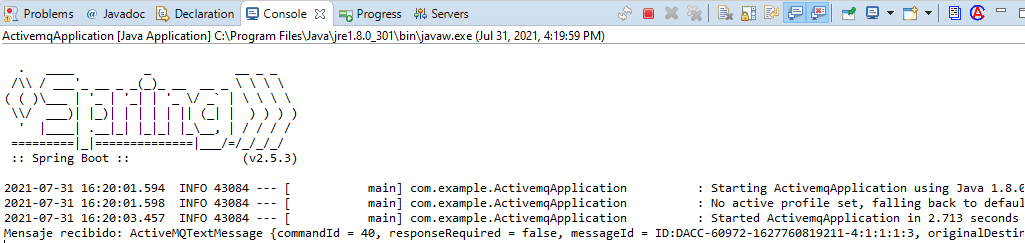


1. Ahora nuestra aplicación java está lista para escuchar a la cola. Para enviar un mensaje, haga clic en la opción **Send To**. En el mensaje ingrese un contenido JSON similar a: {"nombre":"Karla"} y haga clic en *Send*.

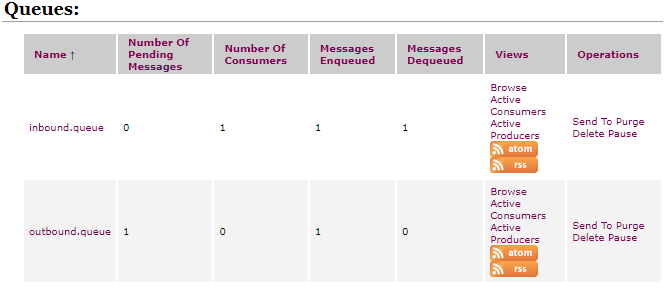


****

1. Verifique en la consola el mensaje recibido.



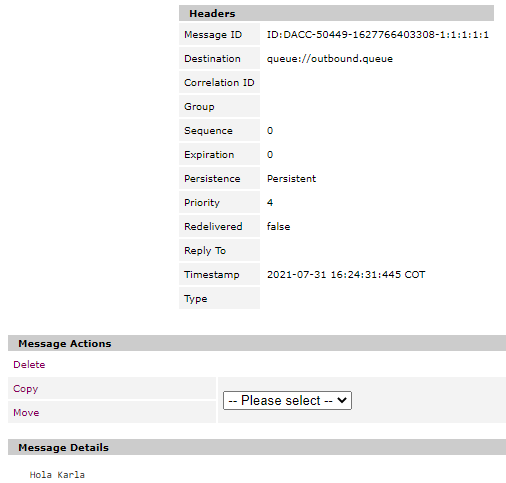
1. Como ha configurado que al recibir el mensaje también se envíe uno de salida, refresca la consola web de colas y verifique que exista un mensaje en la cola de salida.



1. Ingrese a la cola de salida.



1. Ingrese al mensaje recibido.



**Actividad:**

1. Implemente un sistema de punto a punto (colas) que envíe y reciba los datos de una entidad en formato JSON utilizando JMS y ActiveMQ