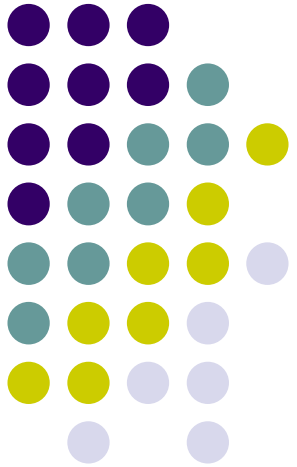
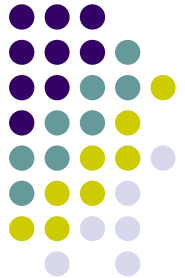
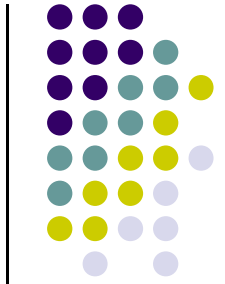
Message Oriented Middleware JMS



Agenda



- MOM
- JMS
- Ejemplos

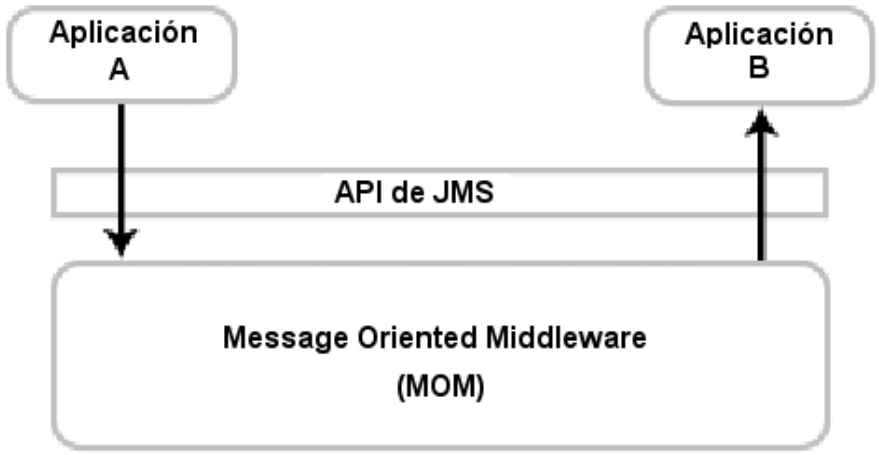


MOM

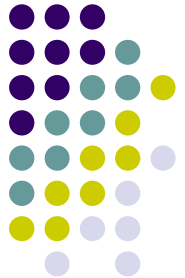
Message Oriented Middleware

Un ejemplo de sistema de mensajería que todos conocemos es el e-mail. El e-mail es un sistema de comunicación persona-persona.

Los MOM’s tratan de comunicaciones aplicación-aplicación. Estos sistemas permiten que las aplicaciones intercambien información en forma de mensajes, un mensaje esta compuesto por: Cabeceras, Datos



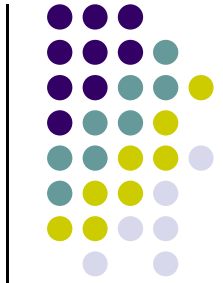
Message Oriented Middleware



Los MOM’s aseguran que los mensajes son distribuidos adecuadamente entre las distintas aplicaciones, por lo general suelen proporcionar otras características importantes como:

-Tolerancia a fallos
-Transacciones
-Escalabilidad

Message Oriented Middleware

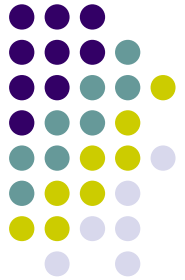


- En un MOM las aplicaciones intercambian mensajes a través de canales virtuales: **destinations**

- Cuando se envía un mensaje no se envía a una aplicación concreta sino a un determinado **destination**

- Las aplicaciones receptoras de los mensajes deben registrar su interés por recibir los mensajes dirigidos a un **destination**

Message Oriented Middleware



-En un MOM los mensajes son enviados de forma **asíncrona**.

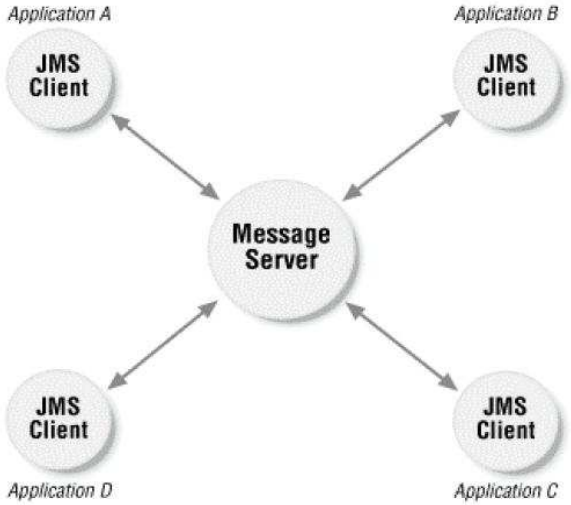
-El encargado de enviar el mensaje no tiene que esperar una respuesta, envía el mensaje y sigue procesando.

-Los mensajes son tratados como unidades auto-contenidas. Contienen todos los datos necesarios para que puedan ser procesados.

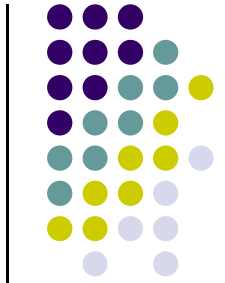
Message Oriented Middleware

- Las distintas implementaciones actuales de MOM están basadas en diferentes arquitecturas, desde arquitecturas con un servidor de mensajes centralizado a arquitecturas descentralizadas que distribuyen el proceso entre los clientes.

- Un sistema de mensajería esta compuesto por los clientes y el propio MOM. Un cliente es cualquier aplicación que envié o reciba mensajes del MOM.



Message Oriented Middleware



Las Ventajas

-Balanceos de carga y prioridad permitiendo la recuperación de mensajes de la cola en cualquier orden.

-Los clientes están libres a realizar otras operaciones mientras que esperan una respuesta del servidor.

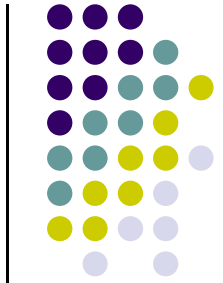
-Permite muchas respuestas a una petición o muchas peticiones a una respuesta.

-Es conveniente para aplicaciones con transacciones largas, tales como workflows.

-Los productos de mensajes soportan una tolerancia a fallas: las colas persistentes permiten que los mensajes sean recuperados cuando el sistema deja de funcionar.

-Escalable: en contraposición al modelo RPC

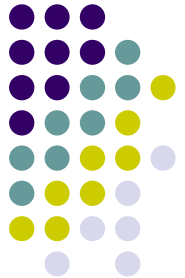
Message Oriented Middleware



Los siguientes productos son ejemplos de distintas implementaciones de un sistema MOM:

IBM: MQSeries
Microsoft: MSMQ
Jboss: JBossMQ
Progress: SonicMQ
Fiorano: FioranoMQ
Softwired: iBus
Sun Microsystems: Java Message Queue
BEA: WebLogic Server
ExoLab:OpenJMS
Apache: ActiveMQ

JMS - Java Message Service



-JMS es un API estándar de la plataforma java para construir aplicaciones que utilicen sistemas de mensajería.

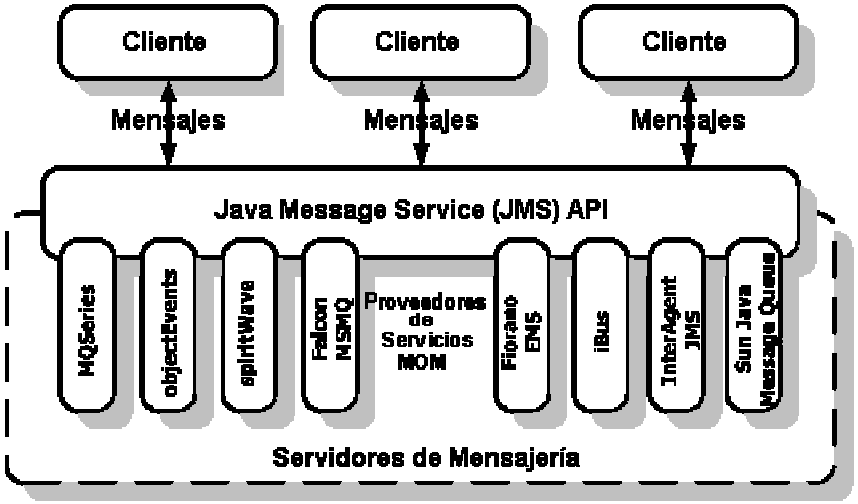
-JMS no es un sistema de mensajería, es un conjunto de las clases y interfaces que necesita un cliente para comunicarse con un sistema de mensajería.

-De forma análoga a como el API **JDBC** permite acceder a diferentes bases de datos JMS permite acceder a diferentes sistemas de mensajería.

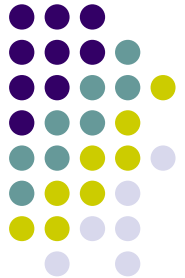
JMS - Java Message Service

-Gracias a JMS se puede construir aplicaciones portables entre distintos MOM's.

-La gran mayoría de implementaciones de MOM soportan el API JMS, lo que en la practica permite construir aplicaciones que usen un paradigma de comunicación basado en mensajes independientemente del producto concreto que se use.



JMS - Modelos



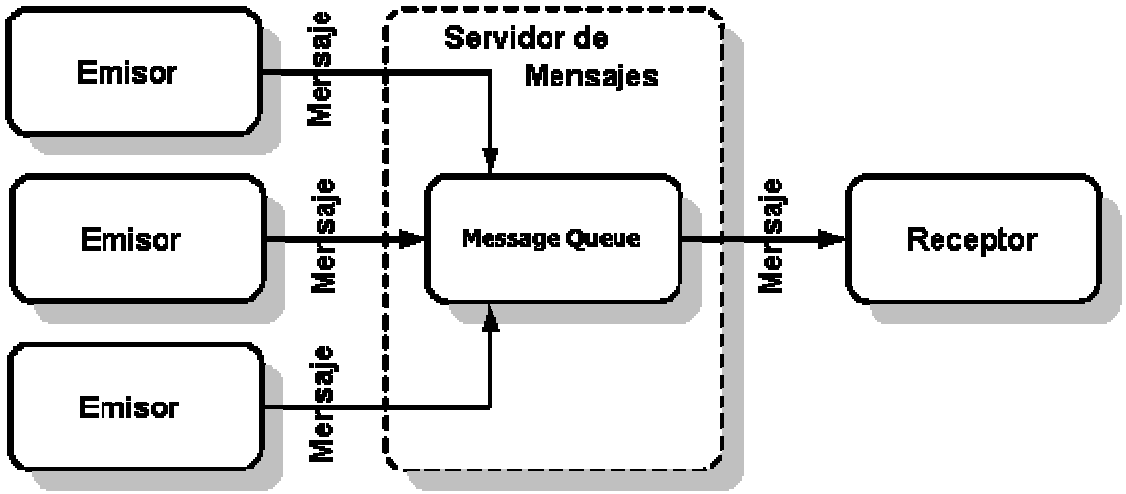
JMS nos proporciona dos modelos distintos de mensajería:
-publicación/subscripción abreviadamente "**pub/sub**”
-comunicación punto a punto abreviadamente "**p2p**“

Como primera aproximación: el modelo publicación/subscripción está pensado para una comunicación "uno a muchos" mientras que el modelo punto a punto lo está para comunicaciones "uno a uno".

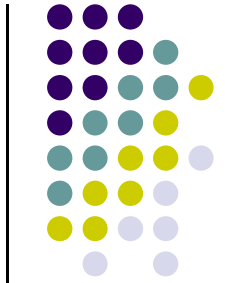
JMS – Modelo P2P

Cuando un proceso necesita mandar un mensaje a otro proceso, se utiliza principalmente la mensajería *Point-to-Point*.

Los sistemas Point-to-Point trabajan con colas de mensajes. Estos se encaminan a un consumidor individual que mantenga una cola de mensajes entrantes. Las aplicaciones de mensajerías envían mensajes a una coa especifica, y los clientes extraen mensajes de esa cola.



JMS – Modelo P2P



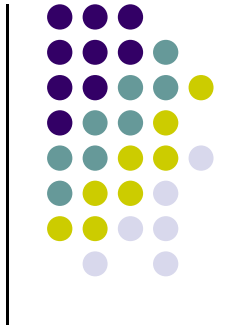
Una aplicación Point-to-Point tiene las siguientes características:

-Un Productor (producer) es el emisor.
-Un Consumidor (consumer) es el receptor.
-Un Destino (destination) es una cola.

-**Un Mensaje solo puede ser usado por un solo receptor y antes puede determinar si consume o no el mensaje mediante algun mecanismo de consulta.**

Por ejemplo: Una aplicación de un CAU (Central de Atención a Usuarios) puede usar un dominio Point-to-Point. Una llamada de teléfono entra en una cola (queue) y la operadora atiende a la llamada. La llamada solo va una operadora, no a todas.

JMS – Modelo P2P



•**Queue**: Es el nombre del objeto por el cual el usuario puede identificar la cola.

•**TemporaryQueue**: Es un objeto queue pero tiene una duración limitada, depende de cuanto dure la QueueConnection. Solo es usada por la QueueConnection que la ha creado.

•**QueueConnectionFactory**: Un cliente usa esto para crear QueueConnection por medio de un proveedor JMS.

•**QueueConnection**: Es una conexión abierta a un proveedor JMS. Se usa el QueueConnection para crear una o más QueueSession para crear o recibir mensajes.

•**QueueSession**: Lo que hace esto es proveer métodos a los queue como puede ser el QueueReceiver, TemporaryQueue, por ejemplo.

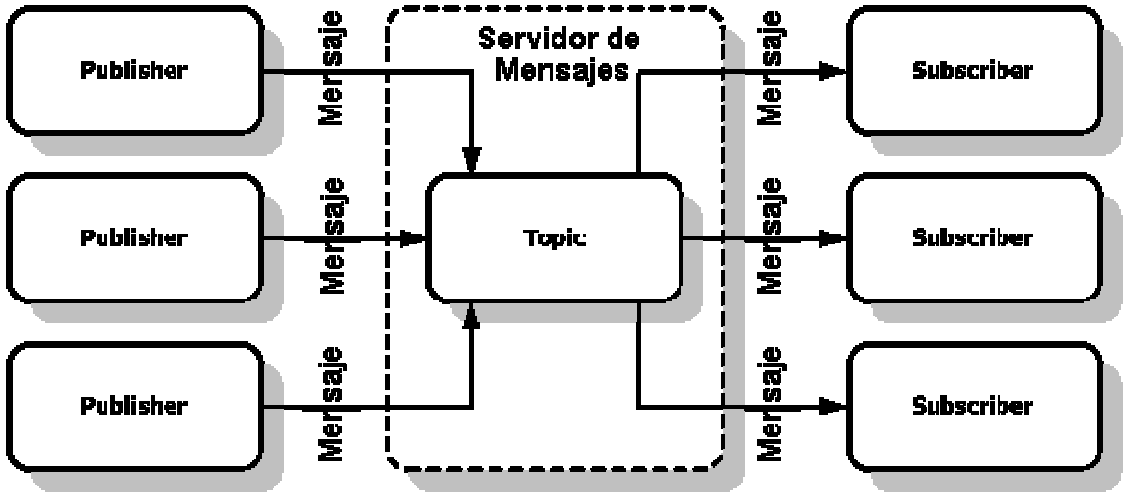
•**QueueReceiver**: Recoge los mensajes que se dejan en el queue (cola).

•**QueueSender**: Manda mensajes a la queue.

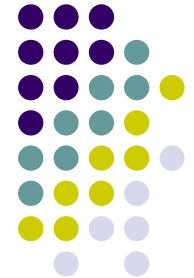
•**QueueBrowser**: Mira en el queue a ver si hay mensajes sin que los retire de ella.

JMS – Modelo pub/sub

-En el modelo pub/sub un productor envía un mensaje a un canal virtual llamado tópico.
-Los consumidores pueden subscribirse a dicho tópico, con lo que recibirían una copia del mensaje.
-Todos los mensajes enviados a un tópico son entregados a **todos** los receptores.
-En este modelo se conoce al productor como publicador y al consumidor como subscriptor.



JMS – Modelo pub/sub



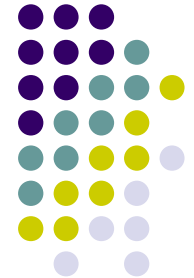
Se usa este modelo de mensajería cuando múltiples aplicaciones quieren recibir el mismo mensaje a la vez. Es muy útil cuando un grupo de aplicaciones quieren notificar a cada uno de las otras aplicaciones una ocurrencia en particular. Puede haber muchos que manden mensajes y muchos también en recibir los mensajes, se puede decir que es de orden M-N.

Una aplicación Publish-Subscribe tiene las siguientes características:

-Un Productor (Producer) es un publicador (publisher).
-Un Consumidor (Consumer) es un subscriptor (subscriber).
-Un Destino (Destination) es un Tema (Topic).
-Un Mensaje puede tener varios subscriptores.

Por Ejemplo: Una aplicación de News puede ser un modelo publish/subscribe. Cualquier persona que este interesada a esa noticia se inscribe y cuando un nuevo mensaje es publicado, todos los inscritos lo reciben.

JMS – Modelo pub/sub



Se usa este modelo de mensajería cuando múltiples aplicaciones quieren recibir el mismo mensaje a la vez. Es muy útil cuando un grupo de aplicaciones quieren notificar a cada uno de las otras aplicaciones una ocurrencia en particular. Puede haber muchos que manden mensajes y muchos también en recibir los mensajes, se puede decir que es de orden M-N.

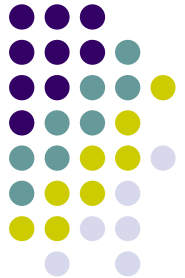
Una aplicación Publish-Subscribe tiene las siguientes características:

-Un Productor (Producer) es un publicador (publisher).
-Un Consumidor (Consumer) es un subscriptor (subscriber).
-Un Destino (Destination) es un Tema (Topic).
-Un Mensaje puede tener varios subscriptores.

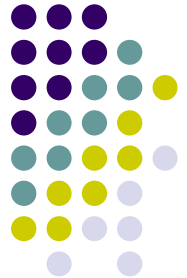
Por Ejemplo: Una aplicación de News puede ser un modelo publish/subscribe. Cualquier persona que este interesada a esa noticia se inscribe y cuando un nuevo mensaje es publicado, todos los inscritos lo reciben.

JMS – Objetos

-Objetos administrados en el servidor de aplicaciones
-Connections
-Destinations
-Sessions
-Message producers
-Message consumers
-Message Listeners
-Messages
-Message Selectors



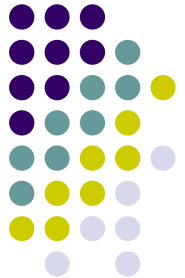
JMS – Objetos



-Objetos administrados en el servidor de aplicaciones: Estos objetos nos administrados fuera del contexto de la programación en si con JMS.
Proveen un mecanismo de abstracción independiente de la implementación de servicios JMS que este utilizando.
Estos objetos serán localizados mediante JNDI

Connections
Destinations

JMS – Connections

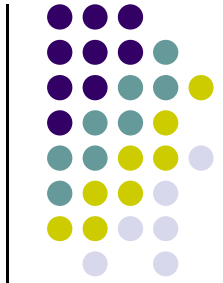


Una conexión es un objeto que permite conectar a un cliente JMS con el servicio de mensajeria correspondiente.

Para poder obtener una conexión es necesario utilizar una factoría de conexiones según el tipo de arquitectura que uno quiera utilizar.

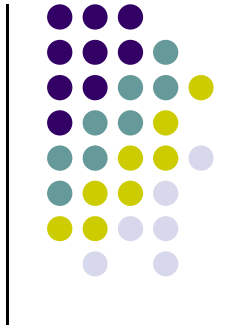
```
Context ctx = new InitialContext();

QueueConnectionFactory queueConnectionFactory =
(QueueConnectionFactory) ctx.lookup("QueueConnectionFactory");

TopicConnectionFactory topicConnectionFactory =
(TopicConnectionFactory) ctx.lookup("TopicConnectionFactory");
```

JMS – Connections



```
QueueConnection queueConnection =
queueConnectionFactory.createQueueConnection();


TopicConnection topicConnection =
topicConnectionFactory.createTopicConnection();

...

// siempre realizar el close....
queueConnection.close();

topicConnection.close();
```

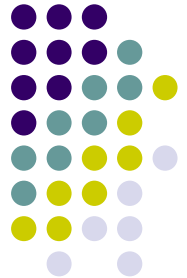
JMS – Destinations



Un destino es un objeto donde un cliente JMS envia o recupera mensajes provienentes de otros sistemas o clientes
Una aplicacion tipica de JMS consume multiples colas y topicos mediante JNDI.

```
Topic topicoProductos = (Topic)
ctx.lookup("TopicoDeNuevosProductos");

....

Queue colaDeVentas = (Queue) ctx.lookup(“ColaDeVentas");
```

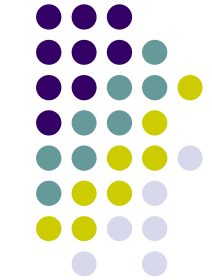
JMS – Sessions



Un objeto session se utiliza para generar y consumir mensajes. Tambien es el punto de control para el manejo de transacciones en el contexto de una transaccion.

```
//false sin soporte de transaciones
//AUTO_ACKNOWLEDGE envia automaticamente que se consumio el mensaje exitosamente

TopicSession topicSession =
topicConnection.createTopicSession(false,
Session.AUTO_ACKNOWLEDGE);

// true con soporte de transacciones y sin respuesta automatica
QueueSession queueSession =
queueConnection.createQueueSession(true, 0);
```

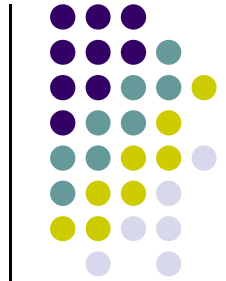
JMS – Message producers



Un message producer es un objeto creado por una un objeto session y es utilizado para crear mensajes que luego seran enviados a un destino determinado.

```
QueueSender queueSender =
queueSession.createSender(colaDeProductos);


TopicPublisher topicPublisher =
topicSession.createPublisher(topicoProductos);

.....

queueSender.send(message);

topicPublisher.publish(message);
```

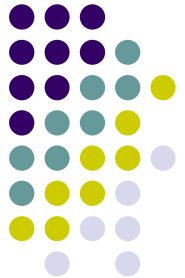
JMS – Message consumers



Un message consumer es un objeto creado por una un objeto session y es utilizado para recivir mensajes.

```
QueueReceiver queueReceiver = queueSession.createReceiver(myQueue);

TopicSubscriber topicSubscriber =
topicSession.createSubscriber(myTopic);

...

queueConnection.start();
Message m = queueReceiver.receive();

topicConnection.start();
Message m = topicSubscriber.receive(1000);
// time out despues de 1 segundo
```

JMS – Mensajes

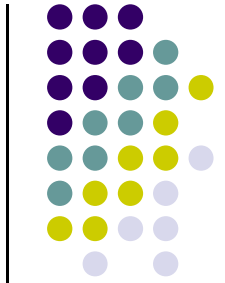


El objeto Message es el elemento mas importante en JMS y representa la información que es intercambiada entre todos los participantes en un sistema de mensajeria.

Existen 5 tipos de mensajes:

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.

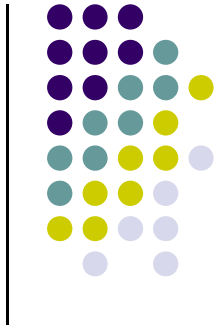
JMS – Mensajes



Un mensaje esta compuesto por:

-Una cabecera
-Una serie de propiedades (clave par-valor)
-Un cuerpo que contiene la información del mensaje

JMS – Mensajes



JMSMessageID (String)
Un numero que identifica univocamente al mensaje. Solo se puede consultar una vez que esta enviado el mensaje.

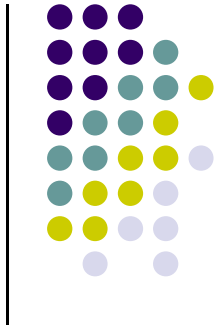
JMSDestination (Destination)
El destino a donde se envia el mensaje.

JMSDeliveryMode(int)
Puede ser de tipo PERSISTENT, entonces se envía una única vez, o de tipo NON_PERSISTEN, de manera que se envía como mucho una vez, lo cual incluye también que no sea enviado nunca.

JMSTimestamp (long)
la hora a la que se envió el mensaje.

JMSExpiration(long)
La hora hasta la cual el mensaje es valido, si es 0 quiere decir que no caduca nunca.

JMS – Mensajes



JMSPriority(int)
Prioridad del mensaje de 0 a 9, siendo 0 la mas baja.

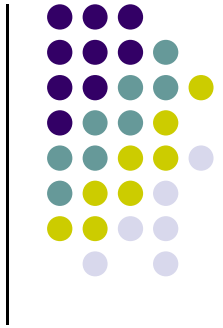
JMSCorrelationID(String)
Este campo se usa para relacionar una respuesta a un mensaje, se copia aqui el id de mensaje del mensaje al que se esta respondiendo.

JMSReplyTo(Destination)
Especifica el lugar a donde se deben enviar las respuestas al mensaje actual.

JMSType(String)
Este campo lo puede usar el programa de mensajeria 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.

JMS – Mensajes



JMSPriority(int)
Prioridad del mensaje de 0 a 9, siendo 0 la mas baja.

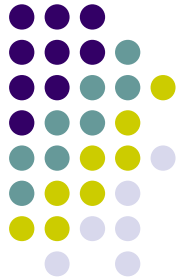
JMSCorrelationID(String)
Este campo se usa para relacionar una respuesta a un mensaje, se copia aqui el id de mensaje del mensaje al que se esta respondiendo.

JMSReplyTo(Destination)
Especifica el lugar a donde se deben enviar las respuestas al mensaje actual.

JMSType(String)
Este campo lo puede usar el programa de mensajeria 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.

JMS – Message Listeners

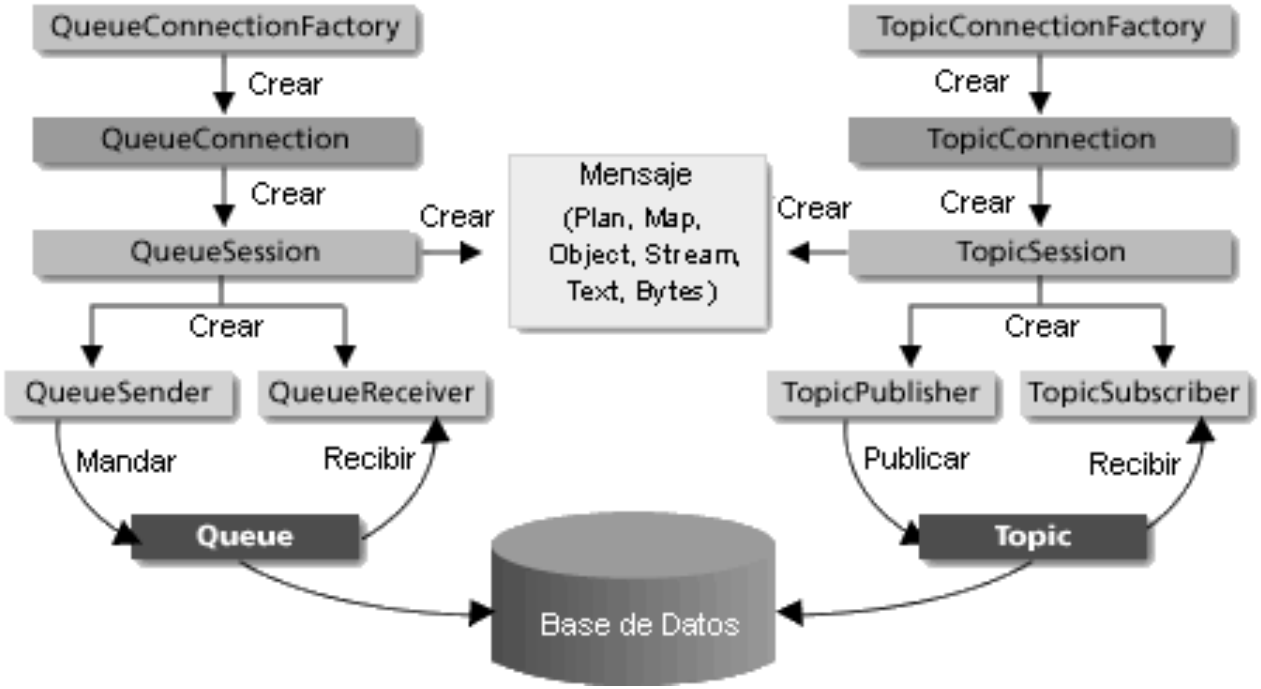


Un message listener es un objeto que permite recibir mensajes asincronicamente mediante eventos.

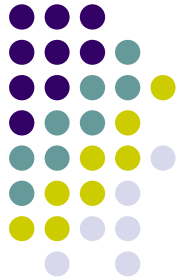
```
TopicListener topicListener = new TopicListener();
topicSubscriber.setMessageListener(topicListener);
```

```
public class ……. implements MessageListener

public void onMessage(Message mensaje) …….
```

JMS – Modelo general



¿Preguntas?



Gracias por su atención