# **Java Message Service**

**JMS** is a API that allows a Java application to send messages to another application. It is highly scalable and allows us to loosely couple applications using asynchronous messaging.it means applications which have to communicate are not connected directly they are connected through a common destination Using JMS we can read, send and read messages.

Also, The Java Message Service ([JMS](https://en.wikipedia.org/wiki/Java_Message_Service)) API is a Java message-oriented middleware API for sending messages between two or more clients

## 1. Why We Need JMS?

When one application wants to send a message to another application in such a way that both applications do not know anything about each other; and even they may NOT be deployed on the same server.

For example, one application A is running in India and another application B is running in the USA, and A is interested in sending some updates/messages to B – whenever something unique happens on A. There may be N number of such applications that are interested in such updates to B.

In this scenario, Java provides one of the solutions in form of JMS – and solves the exact same problem discussed above.

JMS is also useful when we are writing any event-based application like a [chat server](https://howtodoinjava.com/java/library/java-aiml-chatbot-example/) where it needs a publish event mechanism to send messages between the server and the clients.

AD

Note that JMS is different from RMI so there is no need for the destination object to be available online while sending a message. The publisher publishes the message and forgets it, whenever the receiver comes online, it will fetch the message. It’s a very powerful solution for very common problems in today’s world.

AD

## 2. Benefits of JMS

#### 2.1. Asynchronous

JMS is asynchronous by default. So to receive a message, the client is not required to initiate the communication. The message will arrive automatically to the client as they become available.

AD

#### 2.2. Reliable

JMS provides the facility of assurance that the message will be delivered once and only once. We know that duplicate messages create problems. JMS helps you avoid such problems.

### **Elements**

The following are JMS elements:

* **JMS provider**  
  An implementation of the JMS interface for message-oriented middleware (MOM). Providers are implemented as either a Java JMS implementation or an adapter to a non-Java MOM.
* **JMS client**  
  An application or process that produces and/or receives messages.
* **JMS producer/publisher**  
  A JMS client that creates and sends messages.
* **JMS consumer/subscriber**  
  A JMS client that receives messages.
* **JMS message**  
  An object that contains the data being transferred between JMS clients.
* J**MS queue**  
  A staging area that contains messages that have been sent and are waiting to be read (by only one consumer). Contrary to what the name *queue* suggests, messages don’t have to be received in the order in which they were sent. A JMS queue only guarantees that each message is processed only once.
* **JMS topic**  
  A distribution mechanism for publishing messages that are delivered to multiple subscribers.

### To use JMS, one must have a JMS provider that can manage the sessions, queues and topics. Starting from Java EE version 1.4, a JMS provider has to be contained in *all* Java EE application servers. This can be implemented using the message inflow management of the Java EE Connector Architecture, which was first made available in that version.

### **Models**

The JMS API supports two distinct models:

* Point-to-point
* Publish-and-subscribe

### 3.1. Point-to-Point Messaging

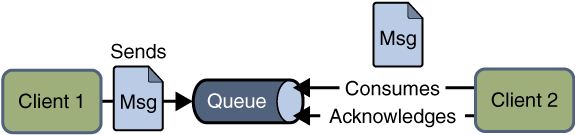
In the [point-to-point messaging](https://howtodoinjava.com/jms/jms-point-to-point-message-example/) domain the application is built on the basis of **message queues, senders and receivers**.

AD

Each and every message is addressed to a particular [**queue**](https://docs.oracle.com/javaee/6/api/javax/jms/Queue.html). Queues retain all messages sent to them until the messages are consumed or expired.

There are some characteristics of PTP messaging:

1. There is only one client for each message.
2. There is no timing dependency for sender and receiver of a message.
3. The receiver can fetch message whether it is running or not when the sender sends the message.
4. The receiver sends the acknowledgement after receiving the message.

Point to point JMS Messaging

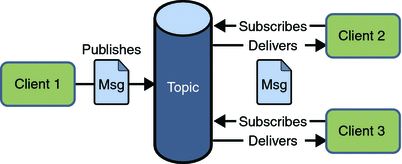
### 3.2. Publish/Subscribe Messaging

In [publish/subscribe messaging](https://howtodoinjava.com/jms/jms-publish-subscribe-message-example/) domain, only one message is published which is delivered to all clients through [**Topic**](https://docs.oracle.com/javaee/6/api/javax/jms/Topic.html) which acts as a bulletin board.

Publishers and subscribers are generally anonymous and can dynamically publish or subscribe to the topic. The Topic is responsible to hold and deliver messages. The topic retains messages as long as it takes to distribute to the present clients.

Some of the characteristics are:

1. There can be multiple subscribers for a message.
2. The publisher and subscribe have a timing dependency. A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages.

Publish Subscribe JMS Messaging

## 4. Receiving a Message

In JMS, the message consumption can be done in two ways:

AD

### 4.1. Synchronous

In synchronous message consumption, the subscriber/receiver requests the message from the destination by calling the [receive()](https://docs.oracle.com/javaee/7/api/javax/jms/MessageConsumer.html#receive--) method.

The receive() method will block till the message arrives or time out if the message does not arrive within a given time. Just like normal java method calls with some return value.

AD

### 4.2. Asynchronous

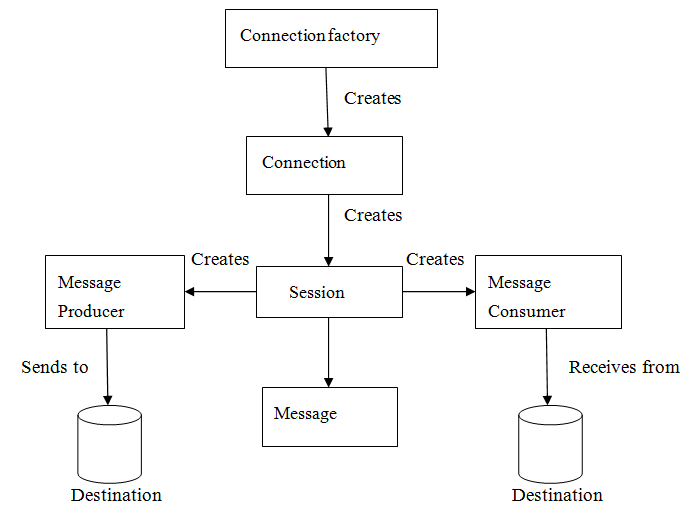
In asynchronous message consumption, a subscriber can register (or subscribe) as a message listener with the consumer.

The message listener is the same as the event listener, whenever the message arrives at the destination the JMS provider will deliver the message by calling the listener’s [onMessage()](https://docs.oracle.com/javaee/7/api/javax/jms/MessageListener.html" \l "onMessage-javax.jms.Message-" \t "_blank) method which will act on the content of the message.

## 5. JMS Participating Objects (Architecture)

JMS application has some basic building blocks, which are:

1. Administered objects – Connection Factories and Destination
2. Connections
3. Sessions
4. Message Producers
5. Message Consumers
6. Message Listeners



### 5.1. JMS Administered Objects

JMS application provides two types of administered objects:

* Connection Factories
* Destinations

These two administered objects are created by the JMS system administrator in JMS Provider by using the Application Server the admin console. Both objects are stored in the Application server JNDI Directory or JNDI Registry.

AD

### 5.2. Connection Factories

The client uses an object which is a [connection factory](https://docs.oracle.com/javaee/6/api/javax/jms/ConnectionFactory.html) used to create a connection to a provider. It creates a connection between JMS Provider and JMS Client.

When JMS Client such as sender or receiver search out for this object in JNDI Registry, then the JMS Client receives one connection object which is nothing just a physical connection between JMS Provider and JMS Client. Using this connection, the client can have communication with the destination object to send or receive messages into Queue or Topic.

Let us have an example to understand it to send the message:

QueueConnectionFactory queueConnFactory = (QueueConnectionFactory) initialCtx.lookup ("primaryQCF");

Queue purchaseQueue = (Queue) initialCtx.lookup ("Purchase\_Queue");

Queue returnQueue = (Queue) initialCtx.lookup ("Return\_Queue");

### 5.3. Destination

The client uses an object known as a destination which is used to specify the target of messages it produces and the source of message who consumes it.

The JMS application uses two types of destinations:

* Queue
* Topic

The code examples to create a queue or a topic:

**Create queue session**

QueueSession ses = con.createQueueSession (false, Session.AUTO\_ACKNOWLEDGE); *//get the Queue object*

Queue t = (Queue) ctx.lookup ("myQueue"); *//create QueueReceiver*

QueueReceiver receiver = ses.createReceiver(t);

**Create topic session**

TopicSession ses = con.createTopicSession (false, Session.AUTO\_ACKNOWLEDGE); *// get the Topic object*

Topic t = (Topic) ctx.lookup ("myTopic"); *//create TopicSubscriber*

TopicSubscriber receiver = ses.createSubscriber(t);

### 5.4. JMS Connection

The connection encapsulates the virtual connection with a JMS Provider. The connection implements the Connection interface, when it will have a ConnectionFactory object then we can use this to create a connection.

Connection connection = connectionFactory.createConnection();

### 5.5. JMS Session

The [session](https://docs.oracle.com/javaee/7/api/javax/jms/Session.html) is a single-threaded context that is used for producing and consuming messages.

The sessions are used to create the following:

* Message Producers
* Message Consumers

The session implements the Session interface and after creating a Connection object we use this to create a Session.

Session session = connection.createSession(false, Session.AUTO\_ACKNOWLEDGE);

### 5.6. Message Producer

A message producer is an object which is created by a session and is used for sending messages to the destination. This implements the [MessageProducer](https://docs.oracle.com/javaee/7/api/javax/jms/MessageProducer.html" \t "_blank) interface.

We use Session to create a MessageProducer for the destination, queue or topic object.

MessageProducer producer = session.createProducer(dest);

MessageProducer producer = session.createProducer(queue);

MessageProducer producer = session.createProducer(topic);

### 5.7. Message Consumer

A message consumer is an object which is created by a session and is used to receive messages sent at a destination. It will implement the [MessageConsumer](https://docs.oracle.com/javaee/7/api/javax/jms/MessageConsumer.html" \t "_blank) interface.

We use a session to create a MessageConsumer for a destination, queue or topic object.

MessageConsumer consumer = session.createConsumer(dest);

MessageConsumer consumer = session.createConsumer(queue);

MessageConsumer consumer = session.createConsumer(topic);

#### 5.8. Message Listener

A message listener is an object which acts as an asynchronous event handler for messages. The message listener implements the MessageListener interface which contains the one method onMessage().

In onMessage() method, we define the actions to be performed when the message arrives. By using setMessageListener() we define the message listener with a specific MessageConsumer.

Listener myListener = **new** Listener();

consumer.setMessageListener(myListener);

## 6. Message Components

The JMS Messages are used by the JMS Clients to have communication between systems. The JMS messages have a simple format but are highly flexible, which allows creating messages matching different formats.

The JMS message is divided into three parts:

### 6.1. Message Header

The JMS message header contains the number of predefined fields which contain those values which are used by the clients and providers to identify and send messages. The predefined headers are:

– JMSDestination  
– JMSDeliveryMode  
– JMSMessageID  
– JMSTimestamp  
– JMSCorrelationID  
– JMSReplyTo  
– JMSRedelivered  
– JMSType  
– JMSExpiration  
– JMSPriority

### 6.2.Message Properties

In message properties we can create and set properties for messages. The message properties are custom name value pairs which are set or read by applications. The message properties are useful for supporting filtering messages. The JMS API provides some predefined property that a provider can support. The message property is optional.

### 6.3. Message Body

In message bodies the JMS API defines five message body formats which are also called as message types which allow us to send and receive data in many different forms and also provides compatibility with the existing messaging formats. It basically consists of the actual message sent from JMS sender to receiver.

The different message types are:

* **Text**: Represented by javax.jms.TextMessage. It is used to represent a block of text.
* **Object**: Represented by javax.jms.ObjectMessage. It is used to represent a java object.
* **Bytes**: Represented by javax.jms.BytesMessage. It is used to represent the binary data.
* **Stream**: Represented by javax.jms.StreamMessage. It is used to represent a list of java primitive values.

‘

Ref:

* https://howtodoinjava.com/jms/jms-java-message-service-tutorial/
* <https://www.geeksforgeeks.org/spring-jms-integration/>
* https://www.javacodegeeks.com/jms-tutorials