

# Java Message Service - JMS Fundamentals

Messaging basics:

What is messaging?

Sender sends message-> MOM/Message Oriented Middleware/Messaging

Server <- Receiver receive/read message

1. Messaging
2. Fault tolerance
3. Load Balancing
4. Scalability
5. Durability
6. Acknowledgement
7. Clustering
8. Transaction management etc.

ActiveMQ/SonicMQ/Websphere MQ/TIBCO MQ

Why messaging?

Heterogeneous integration of services oriented architecture/microservices

Other option for heterogeneous integration is web services but messages will be lost in that.

Reduce system bottlenecks

Can increase number of asynchronous listener.

Scalable.

Flexibility and agility: Replace third party app by in-house app keeping integration contract same.

What is JMS:

Previously developer had to use particular messaging software's API. Now common API came from Java called JMS. No need to learn vendor specific API

JMS 1.2 -> JMS 2.0

Messaging models:

1. Point to point (P2P)
2. Publish/subscribe

Point to point:

Send/receive message synchronously/asynchronously

One message consumed only once by only one application.

Async fire and forget

Synchronous request/reply messaging

receive puts reply in a different queue

Publish/Subscribe:

One publisher, multiple subscribers

Push model, no poll needed

JMS providers pushes message to all subscriber applications who subscribed that particular topic.

News paper subscription

JMS Provider:

Apache ActiveMQ Artemis:

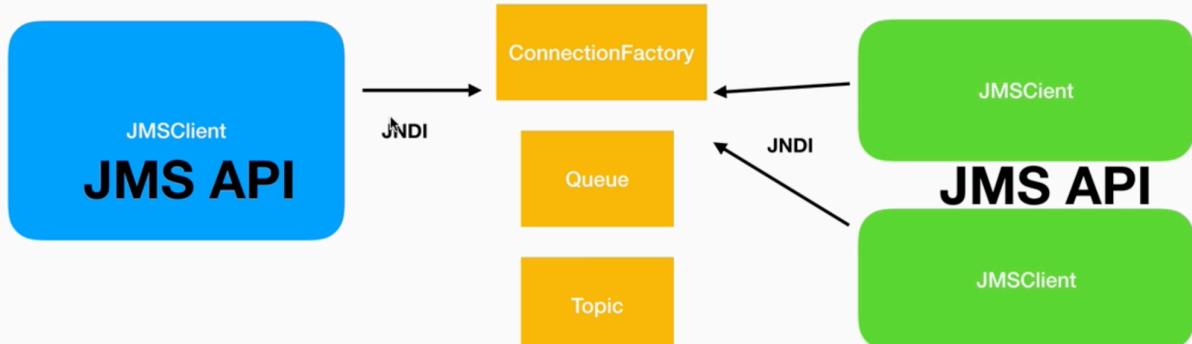
Open source:

Administered Objects - JNDI, Queue (P2P), Topic (P/S)

Accessed using JNDI:

# JMS Provider

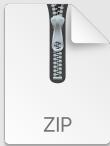
## Apache ActiveMQ Artemis



jmsprojects.zip

Zip archive

252 KB



Software set up:

Use JDK - not JRE

Install Apache ActiveMQ Artemis:

/JMS\_HOME/bin/artemis

/JMS\_HOME/examples/

Creating the Messaging broker/JMS Provider:

```
/JMS_HOME/bin/>./artemis create /Users/suvendu/Documents/mybroker  
admin
```

password -> admin

Allow anonymous access? Y

```
>cd /Users/suvendu/Documents/mybroker/bin
```

```
>ls
```

```
>artemis artemis-service
```

```
>./artemis run
```

```
>
```

```
>vi /Users/suvendu/Documents/mybroker/etc/broker.xml
```

>

Messaging in action:

JMS 1.X API: 7 Components

ConnectionFactory - Registered with JNDI

Destination - Queue/Topic - Registered with JNDI

Connection

Session - n sessions from a connection

Message - Can be created once we have session

MessageProducer - QueueProducer/TopicProducer

MessageConsumer - QueueConsumer/TopicConsumer

1. Create the project: Maven project

2. Add the artemis client dependency:

artemis-jms-client-all:2.6.4 latest version from org.apache.activemq

3. Create jndi.properties: src/main/resources/jndi.properties

java.naming.factory.initial =

org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory(ctrl+shift+t  
for jar class search in STS) This is entry point to JNDI

connectionFactory.**ConnectionFactory**=tcp://localhost

61616

queue.queue/**myQueue**=myQueue

Write a message to the Queue:

1. Create a Connection

2. Create a Session

3. Look up for the Destination

4. Send/Receive Message

```

jmsfundamentals/pom.xml jndi.properties ActiveMQInitialContextFactory.class FirstQueue.java
package com.bharath.jms.basics;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class FirstQueue {

    public static void main(String[] args) {
        InitialContext initialContext = null;

        try {
            initialContext = new InitialContext();
            ConnectionFactory cf = (ConnectionFactory) initialContext.lookup("ConnectionFactory");
            Connection connection = cf.createConnection();
            Session session = connection.createSession();
            Queue queue = (Queue) initialContext.lookup("queue/myQueue");
            MessageProducer producer = session.createProducer(queue);
            TextMessage message = session.createTextMessage("I am the creator of my destiny");
            producer.send(message);
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}

```

## Consume a message from the Queue: P2P in action:

```

jmsfundamentals/pom.xml jndi.properties ActiveMQInitialContextFactory.class FirstQu
try {
    initialContext = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory) initialContext.lookup("ConnectionFactory");
    connection = cf.createConnection();
    Session session = connection.createSession();
    Queue queue = (Queue) initialContext.lookup("queue/myQueue");
    MessageProducer producer = session.createProducer(queue);
    TextMessage message = session.createTextMessage("I am the creator of my destiny");
    producer.sendMessage();
    System.out.println("Message Sent: " + message.getText());

    MessageConsumer consumer = session.createConsumer(queue);
    connection.start();
    TextMessage messageReceived = (TextMessage) consumer.receive(5000);
    System.out.println("Message Received: " + messageReceived.getText());
} catch (NamingException e) {
    e.printStackTrace();
} catch (JMSException e) {
    e.printStackTrace();
} finally {
    if(initialContext!=null) {
        try {
            initialContext.close();
        } catch (NamingException e) {
            e.printStackTrace();
        }
    }
}

if(connection!=null) {
    try {
        connection.close();
    } catch (JMSException e) {
        e.printStackTrace();
    }
}
}

```

Console output:

```

<terminated> FirstQueue [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/
Message Sent: I am the creator of my destiny
Message Received: I am the creator of my destiny

```

Code will create a queue if not initialized already.

Publish/Subscribe:

Add below to jndi.properties:

topic.topic/myTopic=myTopic

```

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.JMSException;
import javax.jms.MessageConsumer;
import javax.jms.MessageProducer;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class FirstQueue {

    public static void main(String[] args) {
        InitialContext initialContext = null;
        try {
            initialContext = new InitialContext();
            ConnectionFactory cf = (ConnectionFactory) initialContext.lookup("ConnectionFactory");
            Connection connection = cf.createConnection();
            Session session = connection.createSession();
            Queue queue = (Queue) initialContext.lookup("queue/myQueue");
            MessageProducer producer = session.createProducer(queue);
            TextMessage message = session.createTextMessage("I am the creator of my destiny");
            producer.send(message);
            System.out.println("Message Sent: " + message.getText());
            MessageConsumer consumer = session.createConsumer(queue);
            connection.start();
            TextMessage messageReceived = (TextMessage) consumer.receive(5000);
            System.out.println("Message Received: " + messageReceived.getText());
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}

```

## Use QueueBrowser:

QueueBrowser is just to browse the message in Queue. It does not consume. So, message will not be removed from the queue even after QueueBrowser browses through the messages in Queue.

```

InitialContext initialContext = null;
Connection connection = null;

try {
    initialContext = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory) initialContext.lookup("ConnectionFactory");
    connection = cf.createConnection();
    Session session = connection.createSession();
    Queue queue = (Queue) initialContext.lookup("queue/myQueue");
    MessageProducer producer = session.createProducer(queue);
    TextMessage message1 = session.createTextMessage("Message 1");
    TextMessage message2 = session.createTextMessage("Message 2");

    producer.send(message1);
    producer.send(message2);

    QueueBrowser browser = session.createBrowser(queue);
    Enumeration messagesEnum = browser.getEnumeration();

    while(messagesEnum.hasMoreElements()) {
        TextMessage eachMessage = (TextMessage) messagesEnum.nextElement();
        System.out.println("Browsing:" + eachMessage.getText());
    }

    MessageConsumer consumer = session.createConsumer(queue);
    connection.start();
    TextMessage messageReceived = (TextMessage) consumer.receive(5000);
    System.out.println("Message Received: " + messageReceived.getText());
    messageReceived = (TextMessage) consumer.receive(5000);
    System.out.println("Message Received: " + messageReceived.getText());
}

```

## JMS 2.X:

### Simple API

JMSContext = Connection + Session  
JMSProducer and JMSConsumer are java.lang.AutoCloseable.  
MessageProducer and MessageConsumer are obsolete.  
JMSProducer and JMSConsumer can easily access to  
Message  
Body  
Headers  
Properties

Directly get TextMessage/StreamMessage/ObjectMessage. No need to convert.

With application server having JEE 7,

```
@Inject  
@JMSSessionFactory("jms/connectionFactory")  
private JMSContext context;  
  
@Resource(lookup="jms/dataQueue")  
private Queue dataQueue;
```

Custom ConnectionFactory  
JBoss, Websphere, Weblogic or any MQ implementation provides default ConnectionFactory.  
@JMSSessionFactoryDefinitions  
@JMSSessionFactoryDefinition  
<jms-connection-factory>

JMS 2.0:

The screenshot shows the Eclipse IDE interface. On the left, the code editor displays a Java file named JMSContextDemo.java. The code imports various JMS-related classes and defines a main method that creates an InitialContext, looks up a queue, and sends/receives a message. On the right, the 'Console' tab shows the output of the program, which includes the message "Arise Awake and stop not till the goal is reached".

```
package com.bharath.jms.basics;

import javax.jms.JMSText;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.swing.plaf.synth.SynthSeparatorUI;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class JMSContextDemo {

    public static void main(String[] args) throws NamingException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){

            jmsContext.createProducer().send(queue,"Arise Awake and stop not till the goal is reached");
            String messageReceived = jmsContext.createConsumer(queue).receiveBody(String.class);
            System.out.println(messageReceived);
        }
    }
}
```

## Anatomy of JMS Message:

Message is divided into 3 parts.

1. Message Header
2. Message Properties
3. Message Body/Payload

## Message Header:

It is metadata of messages.

2 categories:

1. Provider set headers
2. Developer set headers

## Provider (ActiveMQ etc.) set headers:

JMSDestination: The queue/topic to which it will be delivered set by JMS provider like ActiveMQ

JMSDeliveryMode: Persistent/Non-Persistent

JMSMessageId: Unique message id for each message to identify uniquely

JMSTimestamp: Timestamp at which message is received by JMS provider

JMSExpiration: Sets a time for expiry after which message will be expired

JMSRedelivered: Set by provider if a message was not delivered in a prior try and trying to redeliver to consumer.

JMSPriority: To prioritize message delivery. 0-4 for normal, 5-9 for higher priority.

## Developer set headers

JMSReplyTo: Producer application developers sets this to inform consumer application to inform to which queue or topic consumer application should reply to.

JMSCorrelationID: Consumer application sets this copying JMSMessageID from request and sets in response to inform producer app that this is reply message of related request with JMSMessageID.

JMSType: Producer applications sets this to inform the JMS message type.

Properties:

1. Application specific
2. JMS defined specific
3. Provider specific

When we work on filter, we will use properties to filter.

Application specific properties:

Mostly we will use application specific properties.

setXXXProperty

getXXXProperty

XXX can be Integer/String etc.

Provider specific properties:

Generally we don't change this. Not mandatory for provider to support all of the following properties. JMSXGroupID and JMSXGroupSeq used when sent group message instead of individual message.

JMSXUserID

JMSXAppID

JMSXProducerTXID

JMSXConsumerTXID

JMSXRcvTimestamp

JMSXDeliveryCount

JMSXState

JMSXGroupID

JMSXGroupSeq

Prioritize message:

The screenshot shows an IDE interface with two tabs open: 'MessagePriority.java' and 'Console'. The code in 'MessagePriority.java' demonstrates sending three messages with priorities 3, 1, and 9 respectively, and then receiving them back in order. The 'Console' tab shows the output: 'Message Three', 'Message One', and 'Message Two'.

```

import javax.jms.JMSPublisher;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class MessagePriority {

    public static void main(String[] args) throws NamingException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSPublisher producer = jmsContext.createPublisher();

            String[] messages = new String[3];
            messages[0] = "Message One";
            messages[1] = "Message Two";
            messages[2] = "Message Three";

            producer.setPriority(3);
            producer.send(queue, messages[0]);

            producer.setPriority(1);
            producer.send(queue, messages[1]);

            producer.setPriority(9);
            producer.send(queue, messages[2]);

            JMSConsumer consumer = jmsContext.createConsumer(queue);

            for(int i=0;i<3;i++) {
                System.out.println(consumer.receiveBody(String.class));
            }
        }
    }
}

```

## Default priority: 4

The screenshot shows an IDE interface with two tabs open: 'MessagePriority.java' and 'Console'. The code is identical to the previous one but includes comments indicating the default priority is set to 4. The 'Console' tab shows the output: '4', '4', and '4'.

```

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class MessagePriority {

    public static void main(String[] args) throws NamingException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSPublisher producer = jmsContext.createPublisher();

            String[] messages = new String[3];
            messages[0] = "Message One";
            messages[1] = "Message Two";
            messages[2] = "Message Three";

            //producer.setPriority(3);
            producer.send(queue, messages[0]);

            //producer.setPriority(4);
            producer.send(queue, messages[1]);

            //producer.setPriority(9);
            producer.send(queue, messages[2]);

            JMSConsumer consumer = jmsContext.createConsumer(queue);

            for(int i=0;i<3;i++) {
                Message receivedMessage = consumer.receive();
                System.out.println(receivedMessage.getJMSPriority());
                //System.out.println(consumer.receiveBody(String.class));
            }
        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}

```

## Request reply messaging:

request-reply:

replyTo

messageId

correlationID

```
java.naming.factory.initial=org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connectionFactory.ConnectionFactory=tcp://localhost:61616
queue.queue/myQueue=myQueue
topic.topic/myTopic=myTopic
queue.queue/requestQueue=requestQueue
queue.queue/replyQueue=replyQueue
```

```
package com.bharath.jms.messagestructure;

import javax.jms.JMSSession;
import javax.jms.JMSText;
import javax.jms.JMSPublisher;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class RequestReplyDemo {

    public static void main(String[] args) throws NamingException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/requestQueue");
        Queue replyQueue = (Queue) context.lookup("queue/replyQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSSession jmsContext = cf.createContext()){
            JMSPublisher producer = jmsContext.createPublisher();
            producer.send(queue, "Arise Awake and stop not till the goal is reached");

            JMSSession consumer = jmsContext.createConsumer(queue);
            String messageReceived = consumer.receiveBody(String.class);
            System.out.println(messageReceived);

            JMSPublisher replyProducer = jmsContext.createPublisher();
            replyProducer.send(replyQueue, "You are awesome!!");

            JMSSession replyConsumer = jmsContext.createConsumer(replyQueue);
            System.out.println( replyConsumer.receiveBody(String.class));
        }
    }
}
```

<terminated> RequestReplyDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_25.jdk/Content  
Arise Awake and stop not till the goal is reached  
You are awesome!!

### Use replyTo JMS Header:

We can set it in JMSPublisher and Message header object. Better convention is to set it in Message header object.

setJMSReplyTo  
getJMSReplyTo

The screenshot shows the Eclipse IDE interface with several tabs at the top: FirstQueue.java, JMSContextDemo.java, MessagePriority.java, jndi.properties, and RequestReplyDemo.java. The RequestReplyDemo.java tab is active, displaying the following Java code:

```

import javax.jms.Queue;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class RequestReplyDemo {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/requestQueue");
        Queue replyQueue = (Queue) context.lookup("queue/replyQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSProducer producer = jmsContext.createProducer();
            TextMessage message = jmsContext.createTextMessage("Arise Awake and stop not
message.setJMSReplyTo(replyQueue);
System.out.println(messageReceived.getText());

            JMSProducer replyProducer = jmsContext.createProducer();
            replyProducer.send(messageReceived.getJMSTo(), "You are awesome!!");

            JMSConsumer replyConsumer = jmsContext.createConsumer(replyQueue);
            System.out.println( replyConsumer.receiveBody(String.class));
        }
    }
}

```

The Console view on the right shows the output of the application:

```

<terminated> RequestReplyDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Content
Arise Awake and stop not till the goal is reached
You are awesome!!

```

When to use replyTo header:

TemporaryQueue replyQueue = jmsContext.createTemporaryQueue();

The screenshot shows the Eclipse IDE interface with several tabs at the top: FirstQueue.java, JMSContextDemo.java, MessagePriority.java, jndi.properties, and RequestReplyDemo.java. The RequestReplyDemo.java tab is active, displaying the following Java code:

```

import javax.jms.TemporaryQueue;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class RequestReplyDemo {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/requestQueue");
        //Queue replyQueue = (Queue) context.lookup("queue/replyQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSProducer producer = jmsContext.createProducer();
            TemporaryQueue replyQueue = jmsContext.createTemporaryQueue();
            TextMessage message = jmsContext.createTextMessage("Arise Awake and stop not
message.setJMSReplyTo(replyQueue);
producer.send(queue, message);

            JMSConsumer consumer = jmsContext.createConsumer(queue);
            TextMessage messageReceived = (TextMessage) consumer.receive();
            System.out.println(messageReceived.getText());

            JMSProducer replyProducer = jmsContext.createProducer();
            replyProducer.send(messageReceived.getJMSTo(), "You are awesome!!");

            JMSConsumer replyConsumer = jmsContext.createConsumer(replyQueue);
            System.out.println( replyConsumer.receiveBody(String.class));
        }
    }
}

```

The Console view on the right shows the output of the application:

```

RequestReplyDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Content/Home/bin/java
Arise Awake and stop not till the goal is reached
You are awesome!!

```

Use messageID and correlationID headers:

Use these to link request and response.

```

public class RequestReplyDemo {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/requestQueue");
        //Queue replyQueue = (Queue) context.lookup("queue/replyQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSProducer producer = jmsContext.createProducer();
            TemporaryQueue replyQueue = jmsContext.createTemporaryQueue();
            TextMessage message = jmsContext.createTextMessage("Arise Awake and stop not");
            message.setJMSReplyTo(replyQueue);
            producer.send(queue,message);
            System.out.println(message.getJMSMessageID());

            Map<String,TextMessage> requestMessages = new HashMap<>();
            requestMessages.put(message.getJMSMessageID(), message);

            JMSConsumer consumer = jmsContext.createConsumer(queue);
            TextMessage messageReceived = (TextMessage) consumer.receive();
            System.out.println(messageReceived.getText());
            System.out.println(messageReceived.getJMSMessageID());

            JMSProducer replyProducer = jmsContext.createProducer();
            TextMessage replyMessage = jmsContext.createTextMessage("You are awesome!!");
            replyMessage.setJMSCorrelationID(messageReceived.getJMSMessageID());

            replyProducer.send(messageReceived.getJMSReplyTo(), replyMessage);

            JMSConsumer replyConsumer = jmsContext.createConsumer(replyQueue);
            //System.out.println( replyConsumer.receiveBody(String.class));
            TextMessage replyReceived = (TextMessage) replyConsumer.receive();
            System.out.println(replyReceived.getJMSCorrelationID());
            System.out.println(requestMessages.get(replyReceived.getJMSCorrelationID()).getText());
        }
    }
}

```

<terminated> RequestReplyDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_25.jdk/Content  
ID:4994c135-3262-11e9-aa3f-6c40088ed19a  
Arise Awake and stop not till the goal is reached  
ID:4994c135-3262-11e9-aa3f-6c40088ed19a  
Arise Awake and stop not till the goal is reached

Set message expiry:  
`jmsProducer.setTimeToLive(n milliseconds);`

```

package com.bharath.jms.messagestructure;

import javax.jms.JMSContext;
import javax.jms.JMSProducer;
import javax.jms.Message;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class MessageExpirationDemo {

    public static void main(String[] args) throws NamingException, InterruptedException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSProducer producer = jmsContext.createProducer();
            producer.setTimeToLive(2000);
            producer.send(queue,"Arise Awake and stop not till the goal is reached");
            Thread.sleep(5000);

            Message messageReceived = jmsContext.createConsumer(queue).receive(5000);
            System.out.println(messageReceived);
        }
    }
}

```

<terminated> MessageExpirationDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_25.jdk/Contents/Home/bin/java (Feb 17, 2019)  
null

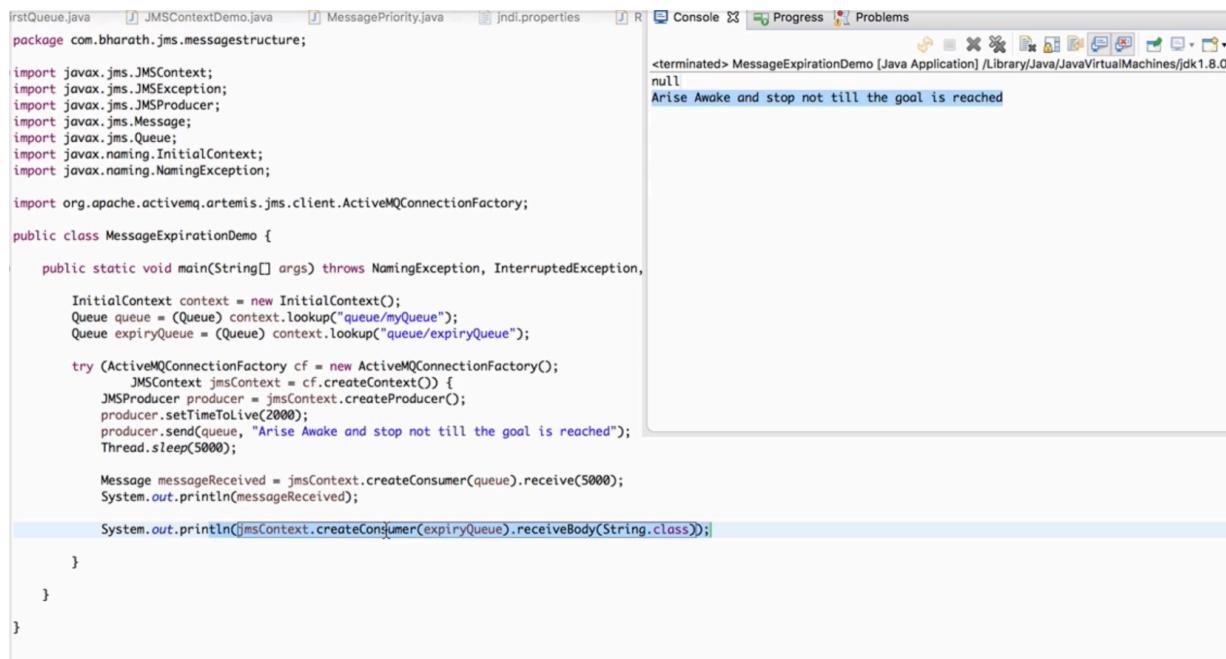
Expired messages are not lost. They just get moved to ExpiryQueue which is

already configured in broker.xml.

We can retrieve those by configuring ExpiryQueue in jndi.properties. Note that E is in uppercase.

queue.queue/expiryQueue = ExpiryQueue

```
Queue expiryQueue = (Queue) initialContext.lookup("queue/expiryQueue");
System.out.println(jmsContext.createConsumer(expiryQueue).receiveBody(S
tring.class));
```



The screenshot shows the Eclipse IDE interface. On the left, there is a package explorer with files like FirstQueue.java, JMSContextDemo.java, MessagePriority.java, and jndi.properties. The main editor area contains Java code for a JMS application. The code imports various JMS classes and uses ActiveMQConnectionFactory to create a producer and consumer. It sends a message with a specific text and receives it back. The terminal window on the right shows the output of the application's execution, which includes the message text and a timestamp.

```
FirstQueue.java JMSContextDemo.java MessagePriority.java jndi.properties R Console Progress Problems
<terminated> MessageExpirationDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_
null
Arise Awake and stop not till the goal is reached
```

```
package com.bharath.jms.messagestructure;

import javax.jms.JMSContext;
import javax.jms.JMSException;
import javax.jms.JMSProducer;
import javax.jms.Message;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class MessageExpirationDemo {

    public static void main(String[] args) throws NamingException, InterruptedException,
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");
        Queue expiryQueue = (Queue) context.lookup("queue/expiryQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory() {
            JMSContext jmsContext = cf.createContext();
            JMSproducer producer = jmsContext.createProducer();
            producer.setTimeToLive(2000);
            producer.send(queue, "Arise Awake and stop not till the goal is reached");
            Thread.sleep(5000);

            Message messageReceived = jmsContext.createConsumer(queue).receive(5000);
            System.out.println(messageReceived);
            System.out.println(jmsContext.createConsumer(expiryQueue).receiveBody(String.class));
        }
    }
}
```

Delay the message delivery:

```
jmsProducer.setDeliveryDelay(n milliseconds);
```

```

package com.bharath.jms.messagestructure;

import javax.jms.JMSContext;

public class MessageDelayDemo {

    public static void main(String[] args) throws NamingException, InterruptedException, JMSException {
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSProducer producer = jmsContext.createProducer();
            producer.setDeliveryDelay(3000);
            producer.send(queue, "Arise Awake and stop not till the goal is reached");

            Message messageReceived = jmsContext.createConsumer(queue).receive(5000);
            System.out.println(messageReceived);
        }
    }
}

```

Console output:

```

<terminated> MessageDelayDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Feb 17, 2019, 4:3
ActiveMQMessage[ID:13ee727d-32a4-11e9-b41e-6c40088ed19a]:PERSISTENT/ClientMessageImpl[messageID=2003, durable=true, ad
true
abc123

```

Add custom message properties:

```

message.setXXXProperty("", ...);
message.setBooleanProperty("loggedin", true);
message.setStringProperty("userToken", "user123");

```

```

package com.bharath.jms.messagestructure;

import javax.jms.JMSContext;

public class MessagePropertiesDemo {

    public static void main(String[] args) throws NamingException, InterruptedException,
        InitialContext context = new InitialContext();
        Queue queue = (Queue) context.lookup("queue/myQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSProducer producer = jmsContext.createProducer();
            TextMessage textMessage = jmsContext.createTextMessage("Arise Awake and stop not till the goal is reached");
            textMessage.setBooleanProperty("loggedin", true);
            textMessage.setStringProperty("userToken", "abc123");
            producer.send(queue, textMessage);

            Message messageReceived = jmsContext.createConsumer(queue).receive(5000);
            System.out.println(messageReceived.getBooleanProperty("loggedin"));
            System.out.println(messageReceived.getStringProperty("userToken"));
        }
    }
}

```

Console output:

```

<terminated> MessagePropertiesDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Feb 17, 2019, 4:3
ActiveMQMessage[ID:a346cf4f-32a5-11e9-aa07-6c40088ed19a]:PERSISTENT/ClientMessageImpl[mes
true
abc123

```

Message types:

1. TextMessage
2. ByteMessage

3. ObjectMessage: Should be serialized

4. StreamMessage

5. MapMessage

Message types in action:

ByteMessage: Generally methods start with write and read

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files like FirstQueue.java, MessagePriority, jndi.properties, RequestReplyDem, and MessageE.
- Code Editor:** Displays the Java code for `MessageTypesDemo`. The code uses ActiveMQ to send and receive a `BytesMessage`. It creates a `BytesMessage`, writes a string ("John") and a long value (123) to it, and then sends it to a queue. On the receiving side, it reads the UTF-8 string and the long value back from the message.
- Console View:** Shows the output of the application's execution. The output is:

```
MessageTypesDemo [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java -Djava.ext.dirs=/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/lib/ext -Dfile.encoding=UTF-8 com.bharath.jms.messagestructure.MessageTypesDemo
John
123
```

StreamMessage: Generally methods start with write and read

The screenshot shows an IDE interface with several tabs at the top: FirstQueue.java, MessagePriority, jndi.properties, RequestReplyDem, MessageE, Console, and Progress. The main area displays Java code for a class named MessageTypesDemo. The code uses ActiveMQConnectionFactory to create a JMSContext, which is then used to create a producer and send two types of messages (TextMessage and StreamMessage) to a queue. It also creates a consumer to receive these messages and prints their contents to the console. The 'Console' tab shows the output of the program, which includes the string "Arise Awake and stop", the boolean value true, and the float value 2.5.

```
package com.bharath.jms.messagestructure;

import javax.jms.BytesMessage;
import javax.jms.Message;
import javax.jms.TextMessage;
import javax.naming.InitialContext;
import javax.jms.Queue;
import org.apache.activemq.ActiveMQConnectionFactory;
import javax.jms.JMSContext;
import javax.jms.JMSProducer;
import javax.jms.TextMessage;
import javax.jms.BytesMessage;
import javax.jms.writeUTF;
import javax.jms.writeLong;
import javax.jms.StreamMessage;
import javax.jms.writeBoolean;
import javax.jms.writeFloat;
import javax.jms.producer.send;
import javax.jms.consumer.receive;
import javax.jms.consumer.createConsumer;
import javax.jms.consumer.readUTF;
import javax.jms.consumer.readLong;
import javax.jms.consumer.readBoolean;
import javax.jms.consumer.readFloat;
import java.out.println;
```

MapMessage: Generally methods start with set and get

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows files like FirstQueue.java, MessagePriority, jndi.properties, RequestReplyDem, MessageE.
- Console View:** Displays the output of the application.

```
<terminated> MessageTypesDemo [Java Application] /Library
true
```
- Code Editor:** Shows the Java code for `MessageTypesDemo`. The code demonstrates sending different message types (TextMessage, BytesMessage, StreamMessage, MapMessage) to a queue and printing their contents to the console.

```
public class MessageTypesDemo {  
    public static void main(String[] args) throws NamingException, InterruptedException,  
        InitialContext context = new InitialContext();  
        Queue queue = (Queue) context.lookup("queue/myQueue");  
  
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();  
            JMSContext jmsContext = cf.createContext()) {  
            JMSProducer producer = jmsContext.createProducer();  
            TextMessage textMessage = jmsContext.createTextMessage("Arise Awake and stop  
BytesMessage bytesMessage = jmsContext.createBytesMessage();  
bytesMessage.writeUTF("John");  
bytesMessage.writeLong(123L);  
  
StreamMessage streamMessage = jmsContext.createStreamMessage();  
streamMessage.writeBoolean(true);  
streamMessage.writeFloat(2.5f);  
  
MapMessage mapMessage = jmsContext.createMapMessage();  
mapMessage.setBoolean("isCreditAvailable", true);  
  
producer.send(queue, mapMessage);  
  
// BytesMessage messageReceived = (BytesMessage) jmsContext.createConsumer(queue).receive();  
//System.out.println(messageReceived.readUTF());  
//System.out.println(messageReceived.readLong());  
  
//StreamMessage messageReceived = (StreamMessage) jmsContext.createConsumer(queue).receive(5000);  
//System.out.println(messageReceived.readBoolean());  
//System.out.println(messageReceived.readFloat());  
  
MapMessage messageReceived = (MapMessage) jmsContext.createConsumer(queue).receive(5000);  
System.out.println(messageReceived.getBoolean("isCreditAvailable"));  
}  
}
```

**ObjectMessage:** Method starts with `setObject` and `getObject`  
Object's class need to implement Serializable interface.

The screenshot shows an IDE interface with several tabs at the top: MessagePriority, jndi.properties, RequestReplyDem, MessageExpirati, Message, Console, and Progress. The code in the editor is as follows:

```

TextMessage textMessage = jmsContext.createTextMessage("Arise Awake and stor
BytesMessage bytesMessage = jmsContext.createBytesMessage();
bytesMessage.writeUTF("John");
bytesMessage.writeLong(123l);

StreamMessage streamMessage = jmsContext.createStreamMessage();
streamMessage.writeBoolean(true);
streamMessage.writeFloat(2.5f);

MapMessage mapMessage = jmsContext.createMapMessage();
mapMessage.setBoolean("isCreditAvailable", true);

ObjectMessage objectMessage = jmsContext.createObjectMessage();
Patient patient = new Patient();
patient.setId(123);
patient.setName("John");
objectMessage.setObject(patient);

producer.send(queue, objectMessage);

// BytesMessage messageReceived = (BytesMessage) jmsContext.createConsumer(queue)
// System.out.println(messageReceived.readUTF());
// System.out.println(messageReceived.readLong());

// StreamMessage messageReceived = (StreamMessage) jmsContext.createConsumer(queue).receive(5000);
// System.out.println(messageReceived.readBoolean());
// System.out.println(messageReceived.readFloat());

// MapMessage messageReceived = (MapMessage) jmsContext.createConsumer(queue).receive(5000);
// System.out.println(messageReceived.getBoolean("isCreditAvailable"));

ObjectMessage messageReceived = (ObjectMessage) jmsContext.createConsumer(queue).receive(5000);
Patient object = (Patient) messageReceived.getObject();
System.out.println(patient.getId());
System.out.println(patient.getName());
}

```

The 'Console' tab shows the output:

```

<terminated> MessageTypesDe
123
John

```

JMS 2.X makes it simple:

JMS 2.X has overloaded methods which directly accepts byte[]/Map<String, Object>/Message/Serializable/String for respective message types like ByteMessage, MapMessage/Generic message, ObjectMessage and TextMessage.

```
jmsProducer.send(queue, patientObject);
```

```
Patient patientObjectReceived =
jmsContext.createConsumer(queue).receiveBody(Patient.class);
```

P2P Messaging:

When to use P2P Messaging:

One to one communication.

Once consumed, won't be available in the queue.

QueueBrowser facility is available only in P2P.

Throughput/performance can be increased by creating multiple queue and each queue for each instance of sender/receiver application.

Usecase:

  Async processing

  Load balancing

Create project p2p:

  1. pom.xml: artemis-jms-client-all dependency

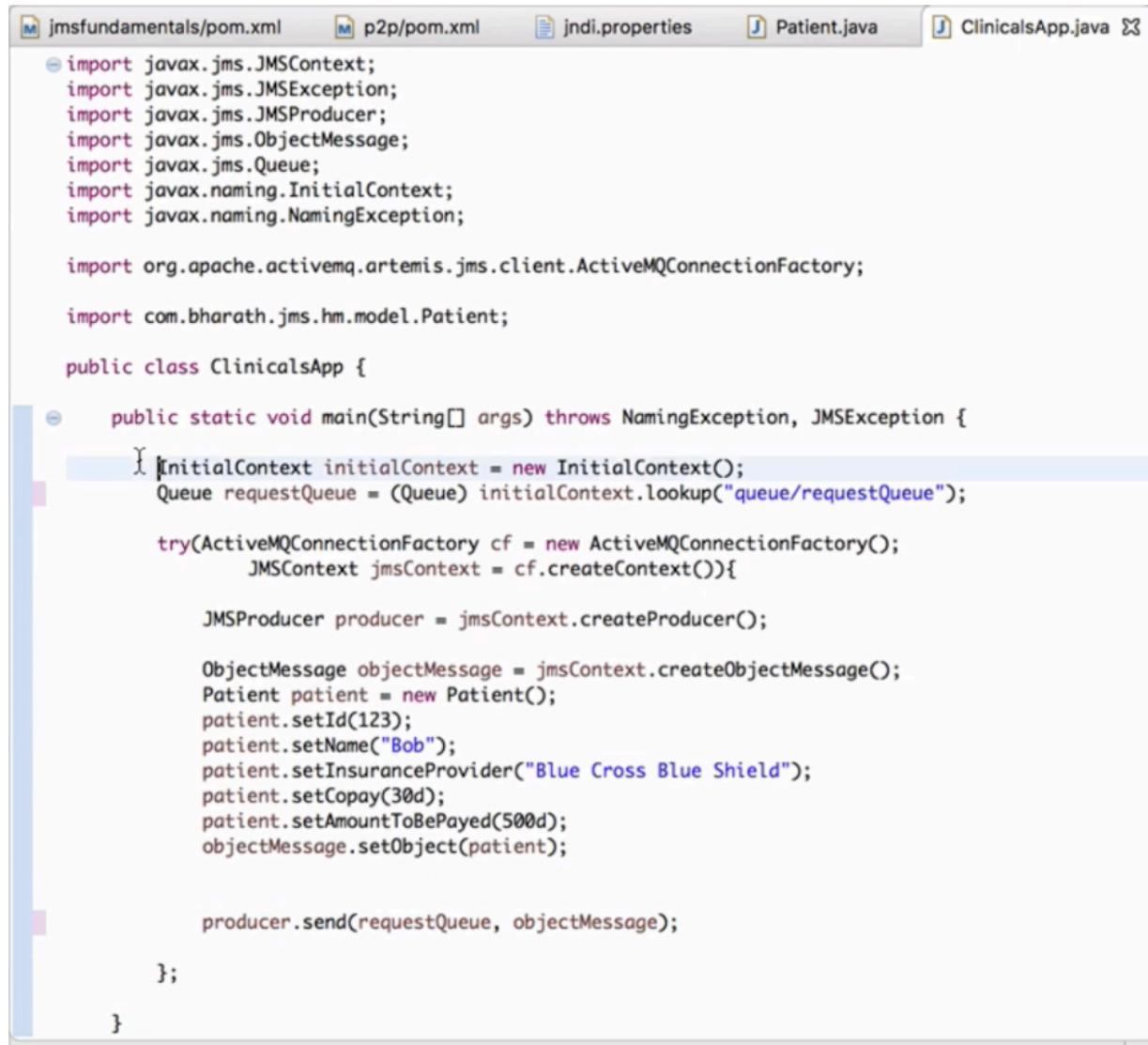
  2. src/main/resources/jndi.properties:

```
java.naming.factory.initial =
org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connectionFactory.ConnectionFactory=tcp://localhost
61616
queue.queue/requestQueue=requestQueue
queue.queue/replyQueue=replyQueue
```

  3. public class Patient implements Serializable.

```
int id;
String name;
String insuranceProvider;
Double copay;
Double amountToBePaid;
```

ClinicalsApp.java: Provider App



The screenshot shows a Java code editor with the file `ClinicalsApp.java` open. The code implements a producer for a JMS queue. It imports various JMS and naming packages, along with the `Patient` class from a local package. The `main` method creates an initial context, looks up a queue, and then tries to create a connection factory and a JMS context. It then creates a producer, an object message, and a patient object, setting its ID and name. Finally, it sends the object message to the request queue.

```
import javax.jms.JMSText;
import javax.jms.JMSException;
import javax.jms.JMSPublisher;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

import com.bharath.jms.hm.model.Patient;

public class ClinicalsApp {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

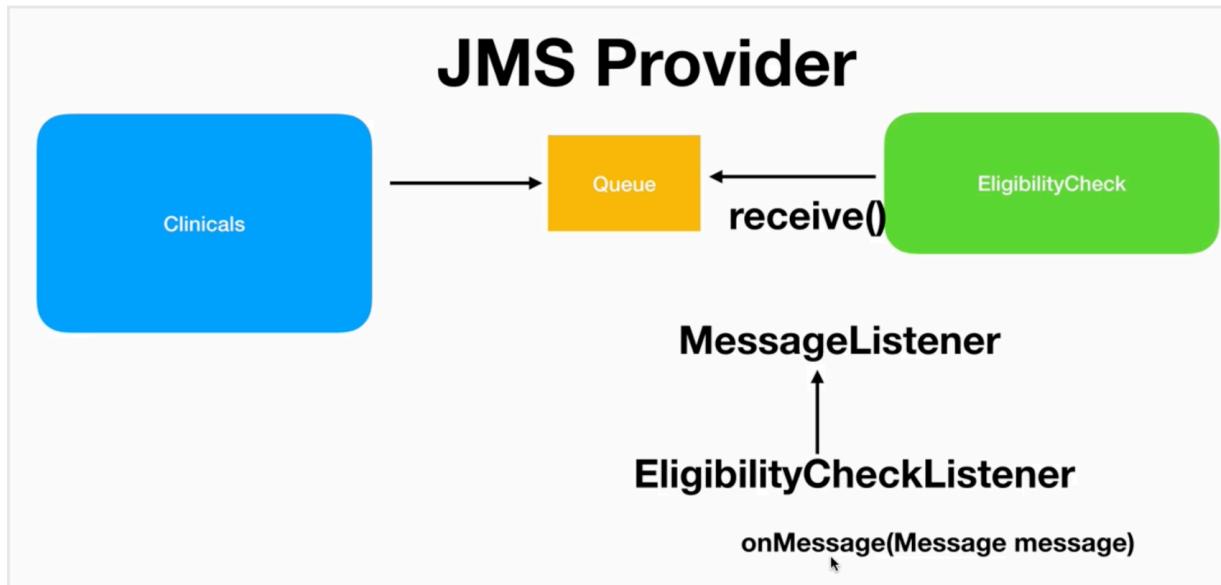
        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSText jmsContext = cf.createContext()){
            JMSPublisher producer = jmsContext.createPublisher();

            ObjectMessage objectMessage = jmsContext.createObjectMessage();
            Patient patient = new Patient();
            patient.setId(123);
            patient.setName("Bob");
            patient.setInsuranceProvider("Blue Cross Blue Shield");
            patient.setCopay(30d);
            patient.setAmountToBePayed(500d);
            objectMessage.setObject(patient);

            producer.send(requestQueue, objectMessage);
        };
    }
}
```

### Asynchronous processing:

Till now we were using synchronous processing. It makes consumer application stuck/wait. We should use asynchronous processing such that it won't wait when there is no message in queue.



### EligibilityCheckerApp.java: Consumer App

```

package com.bharath.jms.hm.eligibilitycheck;

import javax.jms.JMSSession;
import javax.jms.JMSContext;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

import com.bharath.jms.hm.eligibilitycheck.listeners.EligibilityCheckListener;

public class EligibilityCheckerApp {

    public static void main(String[] args) throws NamingException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
            JMSContext jmsContext = cf.createContext()){
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            consumer.setMessageListener(new EligibilityCheckListener());
        }
    }
}

```

Put Thread.sleep(30000); in the past to keep this standalone consumer app running for testing if we need to run and test both provider and consumer app together without server.

```

package com.bharath.jms.hm.eligibilitycheck.listeners;

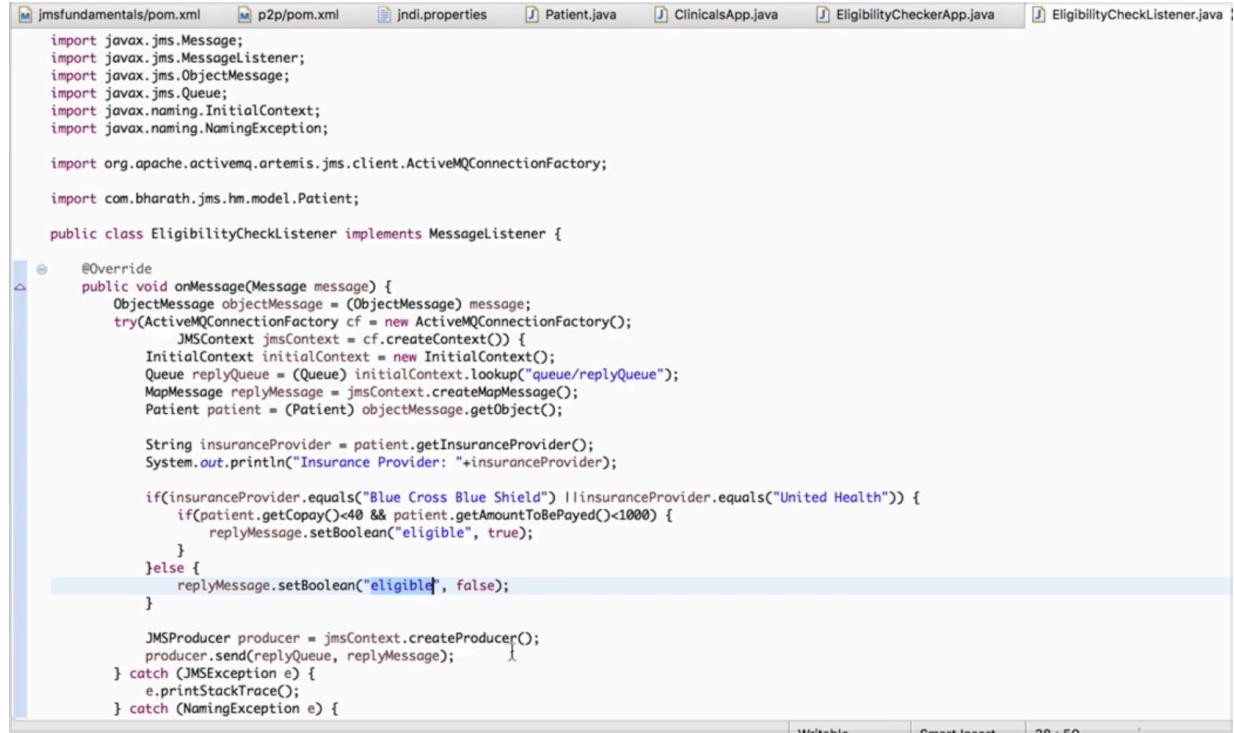
import javax.jms.JMSEException;

public class EligibilityCheckListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        ObjectMessage objectMessage = (ObjectMessage) message;
        try {
            Patient patient = (Patient) objectMessage.getObject();
        } catch (JMSEException e) {
            e.printStackTrace();
        }
    }
}

```

## Check eligibility and send reply:



```
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

import com.bharath.jms.hm.model.Patient;

public class EligibilityCheckListener implements MessageListener {

    @Override
    public void onMessage(Message message) {
        ObjectMessage objectMessage = (ObjectMessage) message;
        try(ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory()) {
            JMSContext jmsContext = cf.createContext();
            InitialContext initialContext = new InitialContext();
            Queue replyQueue = (Queue) initialContext.lookup("queue/replyQueue");
            MapMessage replyMessage = jmsContext.createMapMessage();
            Patient patient = (Patient) objectMessage.getObject();

            String insuranceProvider = patient.getInsuranceProvider();
            System.out.println("Insurance Provider: " + insuranceProvider);

            if(insuranceProvider.equals("Blue Cross Blue Shield") || insuranceProvider.equals("United Health")) {
                if(patient.getCopy() < 0 && patient.getAmountToBePayed() < 1000) {
                    replyMessage.setBoolean("eligible", true);
                } else {
                    replyMessage.setBoolean("eligible", false);
                }
            }

            JMSProducer producer = jmsContext.createProducer();
            producer.send(replyQueue, replyMessage);
        } catch (JMSException e) {
            e.printStackTrace();
        } catch (NamingException e) {
        }
    }
}
```

## Process the response:



```
public static void main(String[] args) throws NamingException, JMSException {
    InitialContext initialContext = new InitialContext();
    Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");
    Queue replyQueue = (Queue) initialContext.lookup("queue/replyQueue");

    try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
         JMSContext jmsContext = cf.createContext()) {

        JMSProducer producer = jmsContext.createProducer();

        ObjectMessage objectMessage = jmsContext.createObjectMessage();
        Patient patient = new Patient();
        patient.setId(123);
        patient.setName("Bob");
        patient.setInsuranceProvider("Blue Cross Blue Shield");
        patient.setCopy(30d);
        patient.setAmountToBePayed(500d);
        objectMessage.setObject(patient);

        producer.send(requestQueue, objectMessage);

        JMSConsumer consumer = jmsContext.createConsumer(replyQueue);
        MapMessage replyMessage = (MapMessage) consumer.receive(30000);
        System.out.println("Patient eligibility is:" + replyMessage.getBoolean("eligible"));
    }
}
```

## Testing:

The screenshot shows the Eclipse IDE interface with the ClinicalsApp.java file open. The code implements a JMS producer and consumer. It creates a Patient object, sets its attributes, and sends it via the requestQueue. Then, it receives a MapMessage from the replyQueue and prints the value of the "eligible" key.

```
import com.bharath.jms.hm.model.Patient;
public class ClinicalsApp {
    public static void main(String[] args) throws NamingException, JMSEException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");
        Queue replyQueue = (Queue) initialContext.lookup("queue/replyQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSProducer producer = jmsContext.createProducer();

            ObjectMessage objectMessage = jmsContext.createObjectMessage();
            Patient patient = new Patient();
            patient.setId(123);
            patient.setName("Bob");
            patient.setInsuranceProvider("Blue Cross Blue Shield");
            patient.setCopay(30d);
            patient.setAmountToBePayed(500d);
            objectMessage.setObject(patient);

            producer.send(requestQueue, objectMessage);

            JMSConsumer consumer = jmsContext.createConsumer(replyQueue);
            MapMessage replyMessage = (MapMessage) consumer.receive(30000);
            System.out.println("Patient eligibility is:" + replyMessage.getBoolean("eligible"));
        }
    }
}
```

The right side of the screen shows the Console view with the output: <terminated> ClinicalsApp (1) [Java Application] /L Patient eligibility is:true

## Load balancing:

Load will be distributed among consumers.

The screenshot shows the Eclipse IDE interface with the EligibilityCheckerApp.java file open. This application acts as a load balancer, creating two consumers (consumer1 and consumer2) to handle requests from the requestQueue. It prints the messages received by each consumer in a loop.

```
package com.bharath.jms.hm.eligibilitycheck;
import javax.jms.JMSConsumer;

public class EligibilityCheckerApp {
    public static void main(String[] args) throws NamingException, InterruptedException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSConsumer consumer1 = jmsContext.createConsumer(requestQueue);
            JMSConsumer consumer2 = jmsContext.createConsumer(requestQueue);
            //consumer.setMessageListener(new EligibilityCheckListener());

            for(int i=1;i<=10;i+=2) {
                System.out.println("Consumer1: "+consumer1.receive());
                System.out.println("Consumer2: "+consumer2.receive());
            }
            //Thread.sleep(10000);
        };
    }
}
```

PubSub Messaging:  
One publisher, multiple subscribers.

Use case:  
Publisher App:  
HR App

Subscriber Apps:  
Security App  
Payroll App  
Wellness App

Create project:

1. pom.xml: artemis-jms-client-all:2.6.4
2. src/main/resources/jndi.properties

```
java.naming.factory.initial =  
org.apache.activemq.jndi.ActiveMQInitialContextFactory  
connectionFactory.ConnectionFactory = tcp://localhost:61616  
topic.topic/empTopic = empTopic
```

Create the EventPublisher:

```
public class Employee implements Serializable {  
    private int id;  
    private String firstName;  
    private String lastName;  
    private String email;  
    private String designation;  
    private String phone;  
}
```

```

package com.bharath.jsm.hr;

import javax.jms.JMSEContext;
import javax.jms.Topic;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class HRApp {

    public static void main(String[] args) throws NamingException {
        InitialContext context = new InitialContext();
        Topic topic = (Topic) context.lookup("topic/empTopic");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSEContext jmsContext = cf.createContext()) {

            Employee employee = new Employee();
            employee.setId(123);
            employee.setFirstName("Bharath");
            employee.setLastName("Thippireddy");
            employee.setDesignation("Software Architect");
            employee.setEmail("bharath@bharath.com");
            employee.setPhone("123456");
            jmsContext.createProducer().send(topic, employee);
            System.out.println("Message Sent");
        }
    }
}

```

Create the subscribers:

Similar codes in 3 subscribers PayrollApp, SecurityApp and WellnessApp

```

package com.bharath.jms.payroll;

import javax.jms.JMSConsumer;

public class PayrollApp {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext context = new InitialContext();
        Topic topic = (Topic) context.lookup("topic/empTopic");

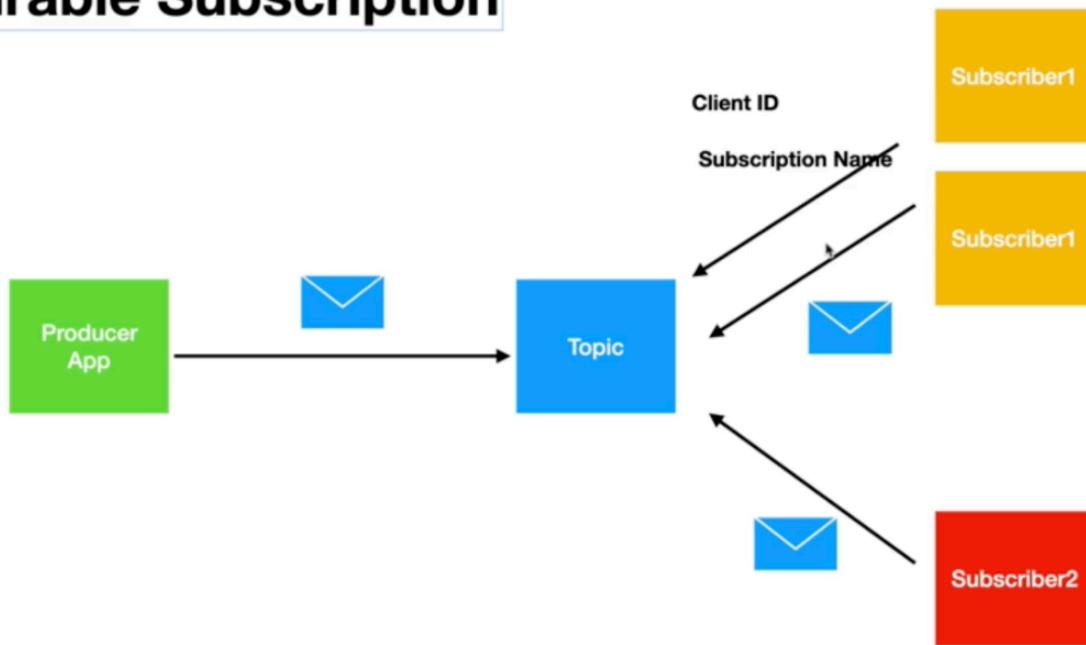
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSEContext jmsContext = cf.createContext()) {
            JMSConsumer consumer = jmsContext.createConsumer(topic);
            Message message = consumer.receive();
            Employee employee = message.getBody(Employee.class);
            System.out.println(employee.getFirstName());
        }
    }
}

```

Durable subscription:

By default, MoM/messaging server does not keep the messages. It serves to current subscribers and remove from topic. But by using durable subscription option, a subscriber can subscribe to a topic with it's client ID and subscription name, so that messaging server keeps message stored until the subscriber consumes it. Even messaging server provides persistent option to database.

## Durable Subscription



The screenshot shows a Java code editor with several tabs at the top: pubsub/pom.xml, jmsfundamentals, jndi.properties, HRApp.java, ClinicalsApp.java, Employee.java, and SecurityApp.java. The code in the editor is for a class named SecurityApp. It imports javax.jms.JMSConsumer and defines a main method. Inside the main method, it creates an InitialContext, looks up a Topic, and creates a JMSContext. It then creates a durable consumer on the topic with the name "subscription1". The consumer receives a message, which is cast to an Employee object, and its firstName is printed. After receiving the message, the consumer is closed and the subscription is unsubscribed.

```
package com.bharath.jms.security;

import javax.jms.JMSConsumer;

public class SecurityApp {

    public static void main(String[] args) throws NamingException, JMSEException, InterruptedException {
        InitialContext context = new InitialContext();
        Topic topic = (Topic) context.lookup("topic/empTopic");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {

            jmsContext.setClientID("securityApp");
            JMSConsumer consumer = jmsContext.createDurableConsumer(topic, "subscription1");
            consumer.close();

            Thread.sleep(10000);

            consumer = jmsContext.createDurableConsumer(topic, "subscription1");
            Message message = consumer.receive();
            Employee employee = message.getBody(Employee.class);
            System.out.println(employee.getFirstName());

            consumer.close(); }
            jmsContext.unsubscribe("subscription1");
        }
    }
}
```

Shared subscription in action:

Create shared consumer to not consume same message by multiple subscribers of same topic. Only one subscriber will take it.

The screenshot shows a Java code editor with several tabs at the top: pubsub/pom.xml, jndi.properties, HRApp.java (selected), ClinicalsApp.java, and Employee.java. The HRApp.java tab contains the following Java code:

```
package com.bharath.jsm.hr;

import javax.jms.JMSContext;

public class HRApp {

    public static void main(String[] args) throws NamingException {
        InitialContext context = new InitialContext();
        Topic topic = (Topic) context.lookup("topic/empTopic");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {

            Employee employee = new Employee();
            employee.setId(123);
            employee.setFirstName("Bharath");
            employee.setLastName("Thippireddy");
            employee.setDesignation("Software Architect");
            employee.setEmail("bharath@bharath.com");
            employee.setPhone("123456");
            for (int i = 1; i <= 10; i++) {
                jmsContext.createProducer().send(topic, employee);
            }

            System.out.println("Message Sent");
        }
    }
}
```

The screenshot shows an IDE interface with several tabs at the top: pubsub/pom.xml, jndi.properties, HRApp.java, ClinicalsApp.java, Employee.java, Console, and Progress.

The code in the main editor is as follows:

```

package com.bharath.jms.welness;

import javax.jms.JMSConsumer;

public class WellnessApp {

    public static void main(String[] args) throws NamingException, JMSEException {
        InitialContext context = new InitialContext();
        Topic topic = (Topic) context.lookup("topic/empTopic");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSConsumer consumer = jmsContext.createSharedConsumer(topic, "sharedConsumer");
            JMSConsumer consumer2 = jmsContext.createSharedConsumer(topic, "sharedConsumer");

            for (int i = 1; i <= 10; i += 2) {
                Message message = consumer.receive();
                Employee employee = message.getBody(Employee.class);
                System.out.println("Consumer 1: " + employee.getFirstName());

                Message message2 = consumer2.receive();
                Employee employee2 = message2.getBody(Employee.class);
                System.out.println("Consumer 2: " + employee2.getFirstName());
            }
        }
    }
}

```

The **Console** tab shows the output of the application:

```

<terminated> WellnessApp [Java]
Consumer 1: Bharath
Consumer 2: Bharath

```

Filter the messages:

If consumer app wants to get the message only on a conditional basis, then we need filter. After consuming if we filter the message, then that message will not be available for the actual intended application which needs it. In this case we need filter. Consumer app will inform messaging server about which filter it wants to subscribe. Provider application is responsible to put those properties in the message such that it can be used at the time of filtering.

Selectors	SQL
Algebraic	is null    +*/
NOT	IN
BETWEEN	LIKE

Properties and few headers of message

JMS Headers:

JMSDeliveryMode

JMSPriority

JMSMessageID

JMSCorrelationID

## JMSType

Create project:

pom

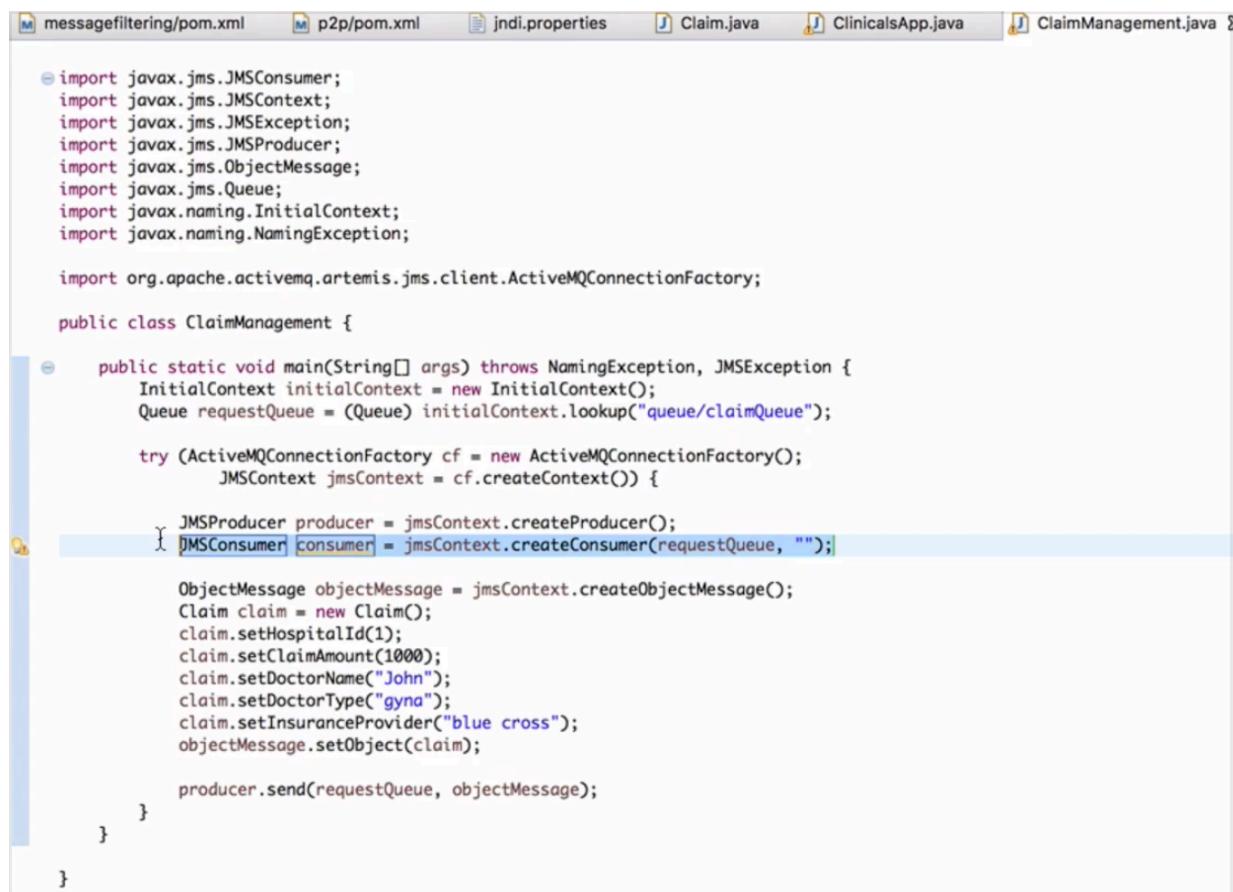
jndi.properties:

claimQueue

Model:

```
class Claim implements Serializable {  
    int hospitalId;  
    String doctorName;  
    String doctorType;  
    String insuranceProvider;  
    double claimAmount;  
}
```

ClaimManagement:



The screenshot shows an IDE interface with several tabs at the top: messagefiltering/pom.xml, p2p/pom.xml, jndi.properties, Claim.java, ClinicalsApp.java, and ClaimManagement.java. The ClinicalsApp.java tab is currently active. Below the tabs, there is a code editor containing Java code for a JMS consumer. The code imports various JMS-related classes and tries to connect to an ActiveMQConnectionFactory. It creates a JMSContext and a JMSConsumer named 'consumer'. The consumer is then used to send an object message containing a 'Claim' object to a queue named 'queue/claimQueue'. The 'Claim' class is defined in the code.

```
import javax.jms.JMSCConsumer;  
import javax.jms.JMSCContext;  
import javax.jms.JMSException;  
import javax.jms.JMSProducer;  
import javax.jms.ObjectMessage;  
import javax.jms.Queue;  
import javax.naming.InitialContext;  
import javax.naming.NamingException;  
  
import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;  
  
public class ClaimManagement {  
  
    public static void main(String[] args) throws NamingException, JMSException {  
        InitialContext initialContext = new InitialContext();  
        Queue requestQueue = (Queue) initialContext.lookup("queue/claimQueue");  
  
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();  
             JMSCContext jmsContext = cf.createContext()) {  
  
            JMSProducer producer = jmsContext.createProducer();  
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue, "");  
  
            ObjectMessage objectMessage = jmsContext.createObjectMessage();  
            Claim claim = new Claim();  
            claim.setHospitalId(1);  
            claim.setClaimAmount(1000);  
            claim.setDoctorName("John");  
            claim.setDoctorType("gyna");  
            claim.setInsuranceProvider("blue cross");  
            objectMessage.setObject(claim);  
  
            producer.send(requestQueue, objectMessage);  
        }  
    }  
}
```

## Filters in action:

The screenshot shows an IDE interface with several tabs at the top: messagefiltering/pom.xml, p2p/pom.xml, jndi.properties, Claim.java, ClinicalsAp, Console, Progress, and Problems. The main area displays Java code for a `ClaimManagement` class. The code uses JMS to send and receive messages. A specific line of code is highlighted: `JMSConsumer consumer = jmsContext.createConsumer(requestQueue, "hospitalId=1");`. In the terminal window (Console tab), the output is shown as `<terminated> ClaimManagement [Java Application]` followed by `1000.0`.

```
import javax.jms.JMSPublisher;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class ClaimManagement {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/claimQueue");

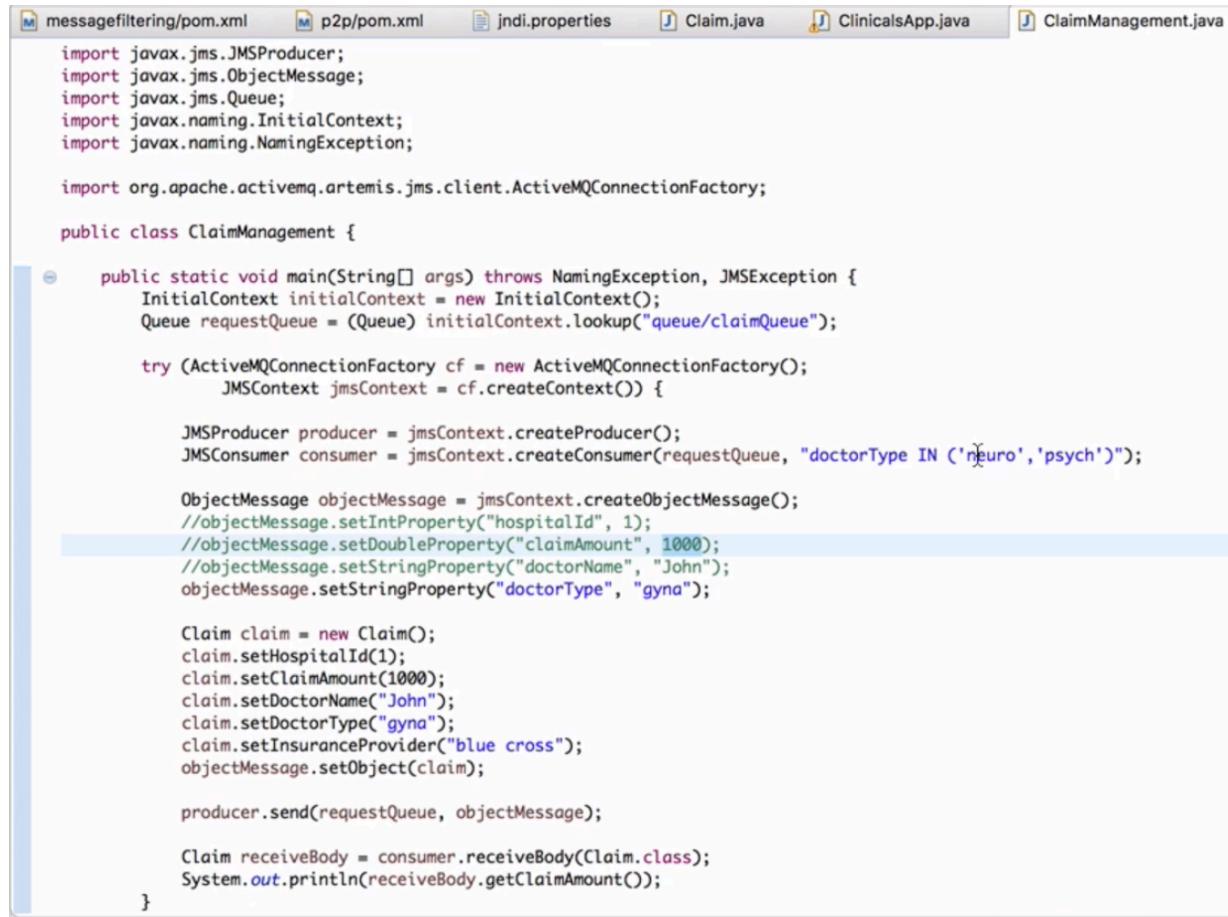
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {

            JMSProducer producer = jmsContext.createProducer();
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue, "hospitalId=1");
            ObjectMessage objectMessage = jmsContext.createObjectMessage();
            objectMessage.setIntProperty("hospitalId", 1);
            Claim claim = new Claim();
            claim.setHospitalId(1);
            claim.setClaimAmount(1000);
            claim.setDoctorName("John");
            claim.setDoctorType("gyna");
            claim.setInsuranceProvider("blue cross");
            objectMessage.setObject(claim);

            producer.send(requestQueue, objectMessage);

            Claim receiveBody = consumer.receiveBody(Claim.class);
            System.out.println(receiveBody.getClaimAmount());
        }
    }
}
```

Use other operators:



```
import javax.jms.JMSPublisher;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class ClaimManagement {

    public static void main(String[] args) throws NamingException, JMSEException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/claimQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {

            JMSPublisher producer = jmsContext.createPublisher();
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue, "doctorType IN ('neuro','psych')");

            ObjectMessage objectMessage = jmsContext.createObjectMessage();
            //objectMessage.setIntProperty("hospitalId", 1);
            //objectMessage.setDoubleProperty("claimAmount", 1000);
            //objectMessage.setStringProperty("doctorName", "John");
            objectMessage.setStringProperty("doctorType", "gyna");

            Claim claim = new Claim();
            claim.setHospitalId(1);
            claim.setClaimAmount(1000);
            claim.setDoctorName("John");
            claim.setDoctorType("gyna");
            claim.setInsuranceProvider("blue cross");
            objectMessage.setObject(claim);

            producer.send(requestQueue, objectMessage);

            Claim receiveBody = consumer.receiveBody(Claim.class);
            System.out.println(receiveBody.getClaimAmount());
        }
    }
}
```

Filter by header:

```

messagefiltering/pom.xml p2p/pom.xml jndi.properties Claim.java ClinicalsApp.java ClaimManagement.java

import javax.jms.JMSPublisher;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory;

public class ClaimManagement {

    public static void main(String[] args) throws NamingException, JMSEException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/claimQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSPublisher jmsContext = cf.createContext()) {

            JMSPublisher producer = jmsContext.createPublisher();
            JMSPublisher consumer = jmsContext.createConsumer(requestQueue, "doctorType IN ('neuro', 'psych') OR JMSPriority BETWEEN 5 AND 9");

            ObjectMessage objectMessage = jmsContext.createObjectMessage();
            //objectMessage.setIntProperty("hospitalId", 1);
            //objectMessage.setDoubleProperty("claimAmount", 1000);
            //objectMessage.setStringProperty("doctorName", "John");
            objectMessage.setStringProperty("doctorType", "gyna");

            Claim claim = new Claim();
            claim.setHospitalId(1);
            claim.setClaimAmount(1000);
            claim.setDoctorName("John");
            claim.setDoctorType("gyna");
            claim.setInsuranceProvider("blue cross");
            objectMessage.setObject(claim);

            producer.send(requestQueue, objectMessage);

            JMSPublisher receiveBody = consumer.receiveBody(Claim.class);
            System.out.println(receiveBody.getClaimAmount());
        }
    }
}

```

Guaranteed messaging:

## Guaranteed Messaging

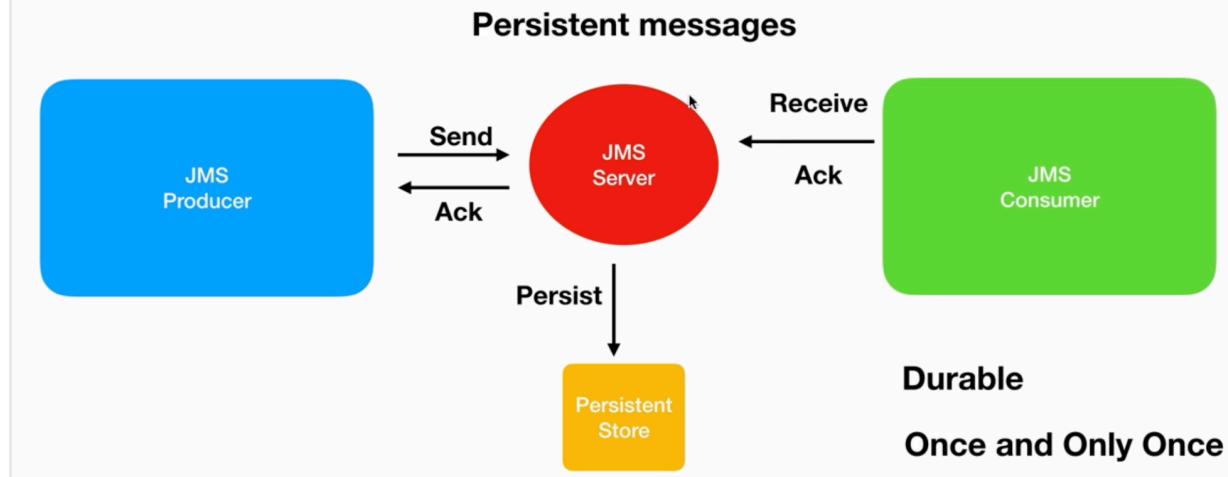


Message acknowledgements:

AUTO\_ACKNOWLEDGE  
CLIENT\_ACKNOWLEDGE  
DUPS\_OK\_ACKNOWLEDGE

AUTO\_ACKNOWLEDGE:

# **AUTO\_ACKNOWLEDGE**



JMS Producer sends/publishes a synchronous message and then waits for JMS server to acknowledge the message and waits/lock until acknowledgement is received.

For P2P, acknowledgement will be for only once delivery. Once and only once delivery.

Creating the project:

pom.xml: artemis

jndi.properties: one queue

MessageProducer:

```
JMSContext jmsContext =  
cf.createContext(JMSContext.AUTO_ACKNOWLEDGE);
```

The screenshot shows an IDE interface with several tabs at the top: guaranteedmessaging/pom.xml, p2p/pom.xml, jndi.properties, MessageProducer.java, and another tab whose name is partially visible. The main area displays the Java code for MessageProducer:

```
package com.bharath.jms.guaranteedmessaging;
import javax.jms.JMSContext;

public class MessageProducer {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.AUTO_ACKNOWLEDGE)) {
            JMSProducer producer = jmsContext.createProducer();
            producer.send(requestQueue, "Message 1");
        }
    }
}
```

## MessageConsumer:

The screenshot shows an IDE interface with several tabs at the top: guaranteedmessaging/pom.xml, p2p/pom.xml, jndi.properties, MessageProducer.java, and MessageConsumer.java. The main area displays the Java code for MessageConsumer:

```
package com.bharath.jms.guaranteedmessaging;
import javax.jms.JMSConsumer;

public class MessageConsumer {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            TextMessage message = (TextMessage) consumer.receive();
            System.out.println(message.getText());
        }
    }
}
```

At the bottom, the Console tab shows the output: "Message 1".

If we just run the MessageProducer.java then it will run and then terminate. and then when we run the MessageConsumer.java, it receives the message from JMS server. But still the producer gets the auto acknowledgement whenever it sent the message to JMS server even though the consumer was not started running. Now if we run the MessageConsumer again, it will run and get blocked until it receives a message. As the message was received in previous run, so the message is not in the Queue anymore. It got removed.

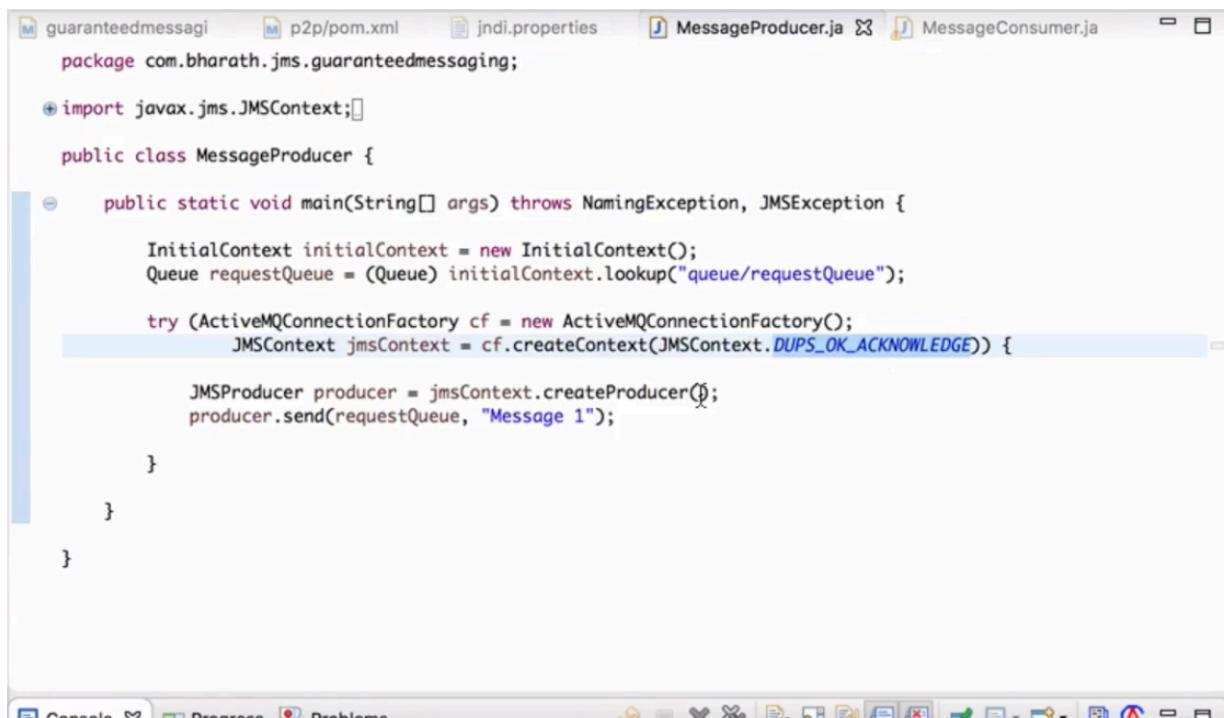
#### DUPS\_OK\_ACKNOWLEDGE:

JMS server can send a message to one consumer multiple times. For AUT\_ACKNOWLEDGE there is an overhead by JMS server to make sure a message gets delivered to a consumer only once. In this DUPS\_OK\_ACKNOWLEDGE that overhead is not there because it does not need to worry about message delivery duplication until message is really delivered.

Consumer tracks JMSMessageId to identify duplicate messages received.

#### Code for DUPS\_OK\_ACKNOWLEDGE:

Producer:



The screenshot shows the Eclipse IDE interface with the Java editor open. The code for MessageProducer.java is displayed, showing the creation of a JMSContext with DUPS\_OK\_ACKNOWLEDGE and the sending of a message to a queue.

```
guaranteedmessagi p2p/pom.xml jndi.properties MessageProducer.java MessageConsumer.java
package com.bharath.jms.guaranteedMessaging;

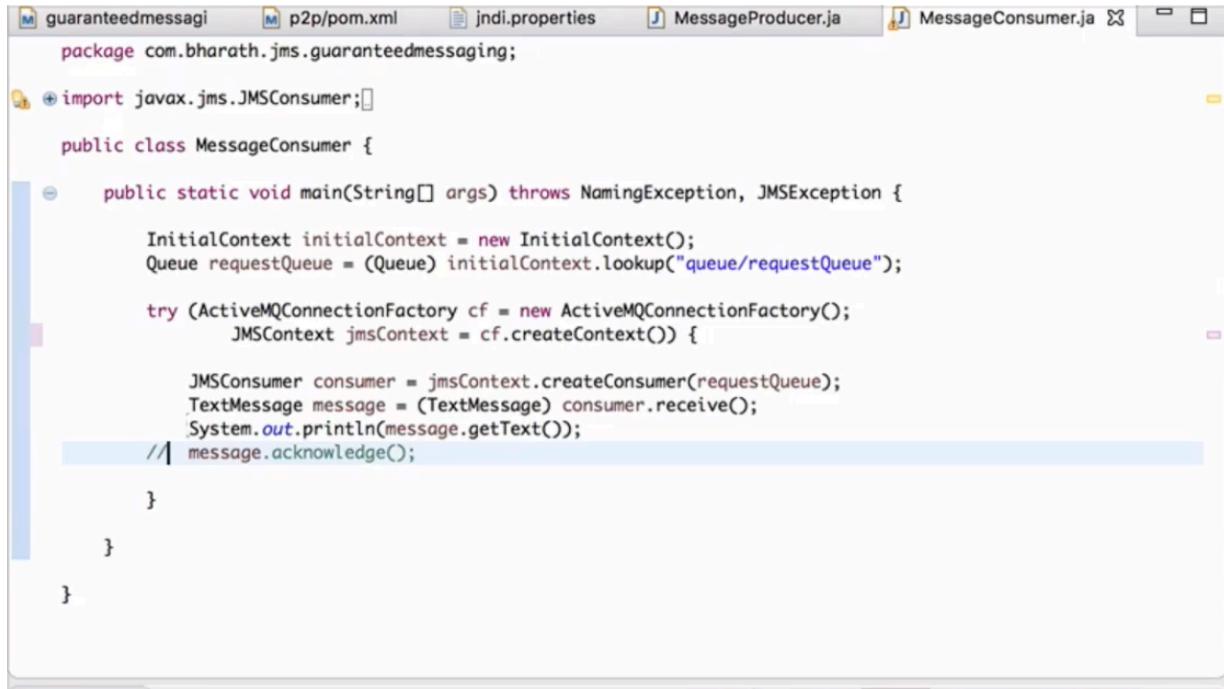
import javax.jms.JMSContext;

public class MessageProducer {

    public static void main(String[] args) throws NamingException, JMSEException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.DUPS_OK_ACKNOWLEDGE)) {
            JMSProducer producer = jmsContext.createProducer();
            producer.send(requestQueue, "Message 1");
        }
    }
}
```

Consumer:



The screenshot shows a Java code editor with several tabs at the top: 'guaranteedmessagi', 'p2p/pom.xml', 'jndi.properties', 'MessageProducer.java', and 'MessageConsumer.java'. The 'MessageConsumer.java' tab is active. The code is as follows:

```
package com.bharath.jms.guaranteedmessaging;
import javax.jms.JMSConsumer;
public class MessageConsumer {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            TextMessage message = (TextMessage) consumer.receive();
            System.out.println(message.getText());
            message.acknowledge();
        }
    }
}
```

### CLIENT\_ACKNOWLEDGE:

JMS server can continue doing other work instead of waiting for acknowledgement of a message after sending it.

JMS consumer will call `Message.acknowledge()` which will inform JMS server about acknowledgement.

If a JMS consumer is configured for `CLIENT_ACKNOWLEDGE` but never does `Message.acknowledge()` then the JMS server will not remove the message and it will keep it persisted as it didn't receive acknowledgement from JMS consumer. So, every time JMS consumer runs it will still get JMS message.

### Code for CLIENT\_ACKNOWLEDGE:

Producer:

Producer can have `AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE`. Does not matter.

```
guaranteedmessagi p2p/pom.xml jndi.properties MessageProducer.java MessageConsumer.java
package com.bharath.jms.guaranteedmessaging;
import javax.jms.JMSContext;
public class MessageProducer {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.AUTO_ACKNOWLEDGE)) {
            JMSProducer producer = jmsContext.createProducer();
            producer.send(requestQueue, "Message 1");
        }
    }
}
```

```
Console Progress Problems
terminated> MessageConsumer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Feb 24, 2019, 8:15:30 AM)
```

Consumer:

```
guaranteedmessagi p2p/pom.xml jndi.properties MessageProducer.java MessageConsumer.java
package com.bharath.jms.guaranteedmessaging;
import javax.jms.JMSConsumer;
public class MessageConsumer {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.CLIENT_ACKNOWLEDGE)) {
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            TextMessage message = (TextMessage) consumer.receive();
            System.out.println(message.getText());
            message.acknowledge();
        }
    }
}
```

```
Console Progress Problems
terminated> MessageConsumer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Feb 24, 2019, 8:15:30 AM)
Message 1
```

JMSContext jmsContext =

```
cf.createContext(JMSContext.AUTO_ACKNOWLEDGE);
JMSContext jmsContext =
cf.createContext(JMSContext.DUPS_OK_ACKNOWLEDGE);
```

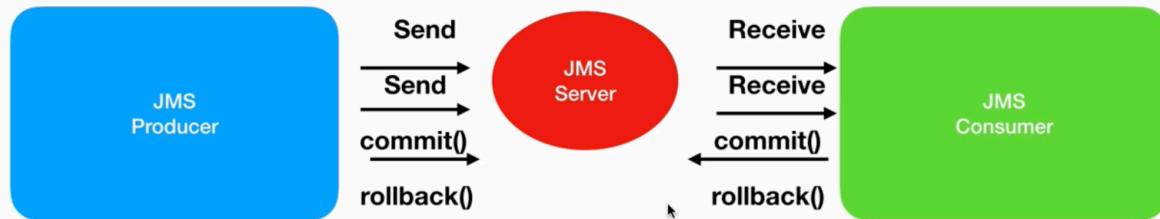
These 2 will be written in producer side and no configuration needed in client side for these 2 acknowledgement types.

```
JMSContext jmsContext =
cf.createContext(JMSContext.CLIENT_ACKNOWLEDGE);
```

This needs to be configured in consumer side for this type of acknowledgement. Producer side can be configured to AUTO\_ACKNOWLEDGE or CLIENT\_ACKNOWLEDGE.

JMS Transactions:  
SESSION\_TRANSACTED

## Transacted Messages



### SESSION\_TRANSACTED

MessageProducer to JMS server:

MessageProducer keeps on sending messages to JMS server. JMS server keeps on adding messages to cache - not into queue/topic.

If MessageProducer sends `rollback()` then JMS server would have removed messages from cache.

Once MessageProducer sends `commit()`, then only all those messages will be put into queue/topic.

JMS server to MessageConsumer:

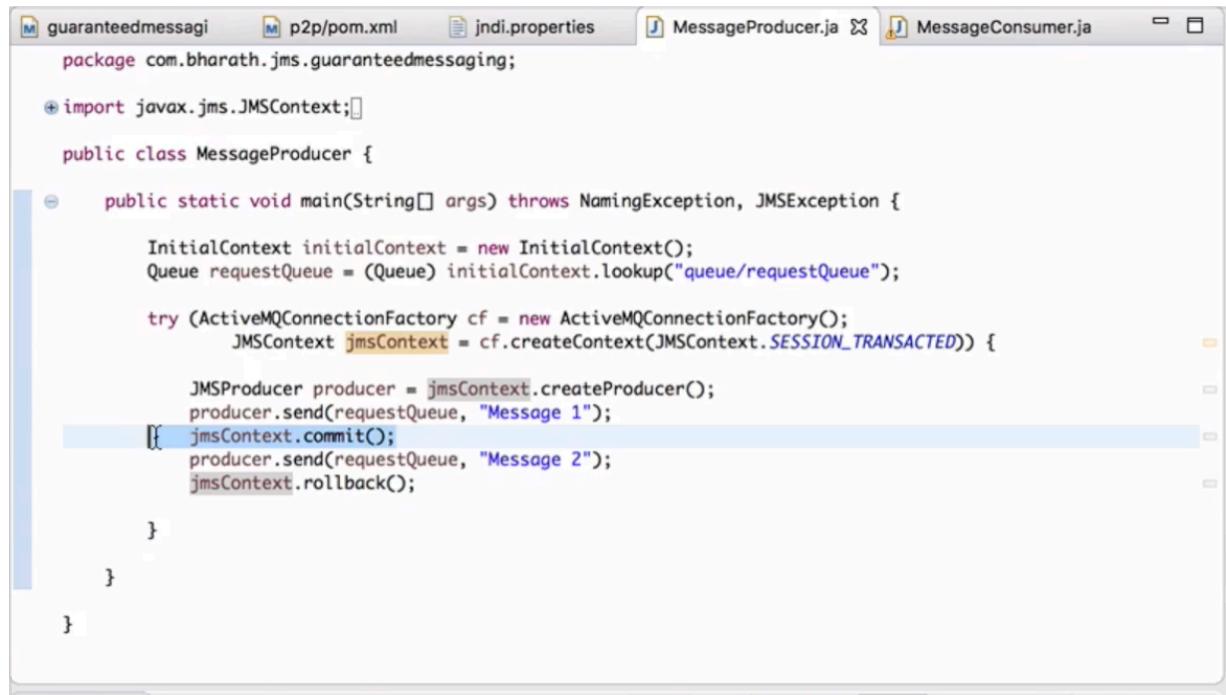
MessageConsumer keeps on receiving messages from JMS server.  
JMS server won't remove messages from it's persistence means keeps retaining messages until it receives commit().

If it gets rollback() then it sends those messages again to MessageConsumer.

Once it receives commit() then only will remove messages from server persistence storage.

If MessageProducer sends commit with 3 messages and MessageConsumer consumes 1 message and then sends commit then JMS server will remove only first message from its persistent storage but will retain second and third message for sending again.

Producer:



```
guaranteedmessagi    p2p/pom.xml    jndi.properties    MessageProducer.java    MessageConsumer.java
package com.bharath.jms.guaranteedmessaging;

import javax.jms.JMSContext;

public class MessageProducer {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.SESSION_TRANSACTED)) {
            JMSProducer producer = jmsContext.createProducer();
            producer.send(requestQueue, "Message 1");
            jmsContext.commit();
            producer.send(requestQueue, "Message 2");
            jmsContext.rollback();
        }
    }
}
```

Consumer:

```

package com.bharath.jms.guaranteedmessaging;

import javax.jms.JMSConsumer;

public class MessageConsumer {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext()) {
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            TextMessage message = (TextMessage) consumer.receive();
            System.out.println(message.getText());
            // message.acknowledge();
        }
    }
}

```

Transaction on consumer side:

Producer with 2 messages:

```

package com.bharath.jms.guaranteedmessaging;

import javax.jms.JMSContext;

public class MessageProducer {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

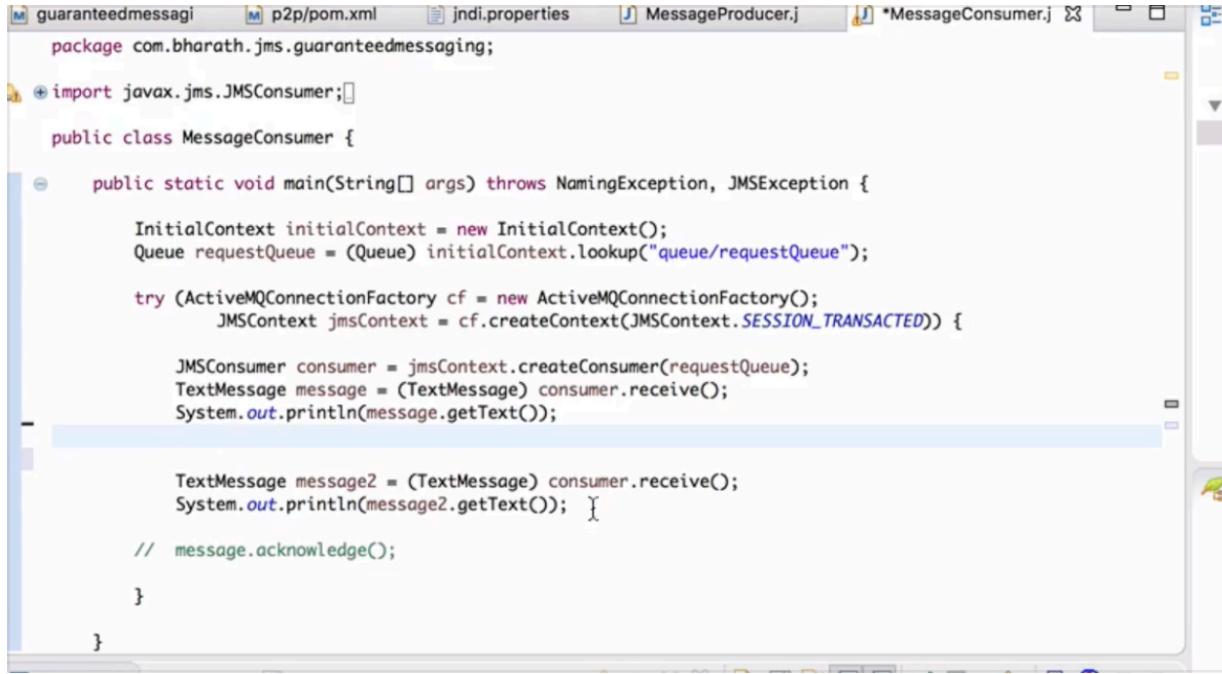
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.SESSION_TRANSACTED)) {
            JMSProducer producer = jmsContext.createProducer();
            producer.send(requestQueue, "Message 1");

            producer.send(requestQueue, "Message 2");
            jmsContext.commit();
            //jmsContext.rollback();
        }
    }
}

```

Consumer with 1 message commit:

It will receive first time 2 messages. But only first message is committed. So, in second run it will receive second message. In third run it will be blocked as there won't be any message in JMS server to consume.



The screenshot shows a Java code editor with the following code:

```
package com.bharath.jms.guaranteedmessaging;

import javax.jms.JMSConsumer;

public class MessageConsumer {

    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext(JMSContext.SESSION_TRANSACTED)) {
            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            TextMessage message = (TextMessage) consumer.receive();
            System.out.println(message.getText());

            TextMessage message2 = (TextMessage) consumer.receive();
            System.out.println(message2.getText());
        }
    }
}
```

## Security:

artemis-user.properties  
artemis-roles.properties  
broker.xml

**artemis-user.properties:** username, password like username with password  
**artemis-roles.properties:** role-based authorization like role vs users  
**broker.xml:** role will be defined with privileges like role vs privileges

**artemis-user.properties:** admin= ENC(1024:\*\*abx74grj\*) (Encrypted password for admin)  
test=testpassword (non-encrypted)

—  
clinicaluser = clinicalpass  
eligibilityuser=eligibilitypass

**artemis-roles.properties:** amq = admin, test (where amq is role and admin, test are users)

—  
clinicalrole=clinicaluser  
eligibilityrole=eligibilityuser

**broker.xml:**

```

<security-settings>
    <security-setting match="#">
        <permission type="createNonDurableQueue" roles="amq"/>
        ...
        ...
        ...
        <permission type="consume" roles="amq"/>
        <permission type="browse" roles="amq"/>
        <permission type="send" roles="amq"/>
        ...
    <security-setting>
    ...
    <security-setting match="suvendu.queues.request.#">
        <permission type="createNonDurableQueue"
roles="clinicalrole,eligibilityrole"/>
        ...
        ...
        ...
        <permission type="consume" roles="eligibilityrole"/>
        <permission type="browse" roles="clinicalrole"/>
        <permission type="send" roles="clinicalrole"/>
        ...
    <security-setting>
    <security-setting match="suvendu.queues.response.#">
        <permission type="createNonDurableQueue" roles="eligibilityrole"/>
        ...
        ...
        ...
        <permission type="consume" roles="clinicalrole"/>
        <permission type="browse" roles="eligibilityrole"/>
        <permission type="send" roles="eligibilityrole"/>
        ...
    <security-setting>
</security-settings>

```

### **src/main/resources/jndi.properties:**

```

java.naming.factory.initial =
org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connectionFactory.ConnectionFactory=tcp://localhost
61616

```

```
queue.queue/requestQueue=suvendu.queues.request.requestQueue  
queue.queue/replyQueue=suvendu.queues.reply.replyQueue
```

Producer:

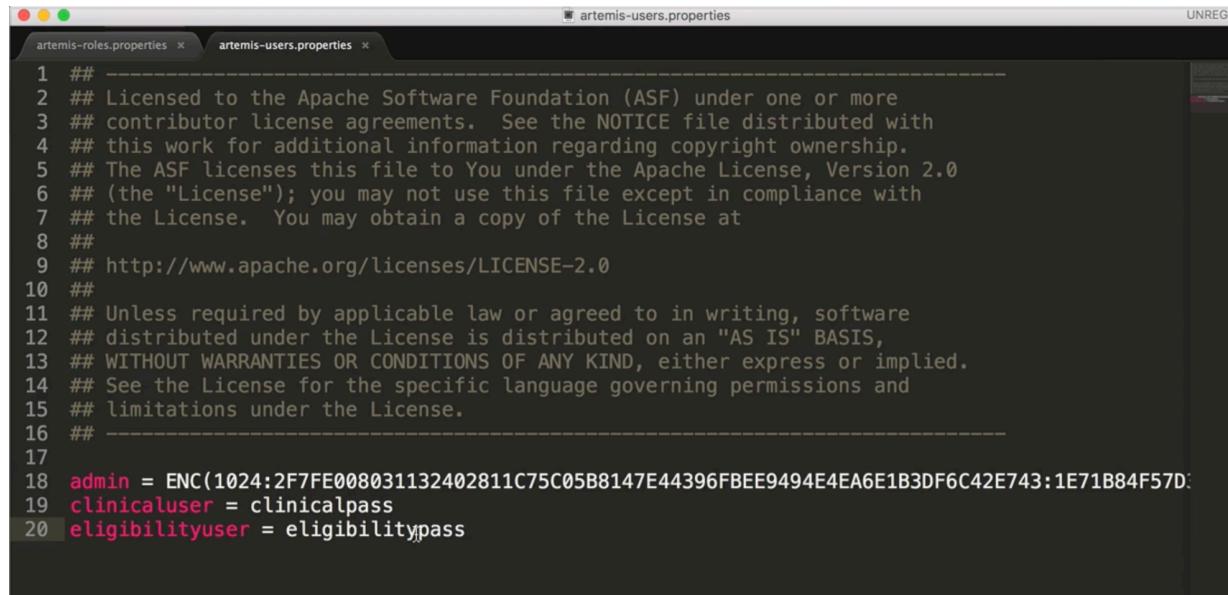
```
JMSContext jmsContext = cf.createContext("clinicalUser", "clinicalPass");
```

Consumer:

```
JMSContext jmsContext = cf.createContext("eligibilityUser", "eligibilityPass");
```

Code:

**artemis-user.properties:**



```
artemis-users.properties  
-----  
1 ## -----  
2 ## Licensed to the Apache Software Foundation (ASF) under one or more  
3 ## contributor license agreements. See the NOTICE file distributed with  
4 ## this work for additional information regarding copyright ownership.  
5 ## The ASF licenses this file to You under the Apache License, Version 2.0  
6 ## (the "License"); you may not use this file except in compliance with  
7 ## the License. You may obtain a copy of the License at  
8 ##  
9 ## http://www.apache.org/licenses/LICENSE-2.0  
10##  
11## Unless required by applicable law or agreed to in writing, software  
12## distributed under the License is distributed on an "AS IS" BASIS,  
13## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
14## See the License for the specific language governing permissions and  
15## limitations under the License.  
16## -----  
17  
18 admin = ENC(1024:2F7FE008031132402811C75C05B8147E44396FBEE9494E4EA6E1B3DF6C42E743:1E71B84F57D:  
19 clinicaluser = clinicalpass  
20 eligibilityuser = eligibilitypass
```

**artemis-roles.properties:**

```
artemis-roles.properties * artemis-users.properties * artemis-roles.properties
1 ## -----
2 ## Licensed to the Apache Software Foundation (ASF) under one or more
3 ## contributor license agreements. See the NOTICE file distributed with
4 ## this work for additional information regarding copyright ownership.
5 ## The ASF licenses this file to You under the Apache License, Version 2.0
6 ## (the "License"); you may not use this file except in compliance with
7 ## the License. You may obtain a copy of the License at
8 ##
9 ## http://www.apache.org/licenses/LICENSE-2.0
10 ##
11 ## Unless required by applicable law or agreed to in writing, software
12 ## distributed under the License is distributed on an "AS IS" BASIS,
13 ## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 ## See the License for the specific language governing permissions and
15 ## limitations under the License.
16 ##
17
18 amq = admin
19 clinicalrole = clinicaluser
20 eligibilityrole = eligibilityuser
```

## broker.xml:

```
broker.xml * broker.xml
152
153     </acceptors>
154
155
156     <security-settings>
157         <security-setting match="bharath.queues.request.#">
158             <permission type="createNonDurableQueue" roles="clinicalrole,eligibilityrole"/>
159             <permission type="deleteNonDurableQueue" roles="clinicalrole,eligibilityrole"/>
160             <permission type="createDurableQueue" roles="clinicalrole,eligibilityrole"/>
161             <permission type="deleteDurableQueue" roles="clinicalrole,eligibilityrole"/>
162             <permission type="createAddress" roles="clinicalrole,eligibilityrole"/>
163             <permission type="deleteAddress" roles="clinicalrole,eligibilityrole"/>
164             <permission type="consume" roles="eligibilityrole"/>
165             <permission type="browse" roles="clinicalrole"/>
166             <permission type="send" roles="clinicalrole"/>
167             <!-- we need this otherwise ./artemis data imp wouldn't work -->
168             <permission type="manage" roles="amq"/>
169         </security-setting>
170         <security-setting match="bharath.queues.reply.#">
171             <permission type="createNonDurableQueue" roles="eligibilityrole"/>
172             <permission type="deleteNonDurableQueue" roles="eligibilityrole"/>
173             <permission type="createDurableQueue" roles="eligibilityrole"/>
174             <permission type="deleteDurableQueue" roles="eligibilityrole"/>
175             <permission type="createAddress" roles="eligibilityrole"/>
```

## MessageProducer:

```
ClinicalsApp.java  EligibilityCheckerApp.java  jndi.properties
public class ClinicalsApp {
    public static void main(String[] args) throws NamingException, JMSException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");
        Queue replyQueue = (Queue) initialContext.lookup("queue/replyQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext("clinicaluser", "clinicalpass")) {
            JMSProducer producer = jmsContext.createProducer();

            ObjectMessage objectMessage = jmsContext.createObjectMessage();
            Patient patient = new Patient();
            patient.setId(123);
            patient.setName("Bob");
            patient.setInsuranceProvider("Blue Cross Blue Shield");
            patient.setCopay(100d);
            patient.setAmountToBePayed(500d);
            objectMessage.setObject(patient);

            for (int i = 1; i <= 10; i++) {
                producer.send(requestQueue, objectMessage);
            }

            //JMSConsumer consumer = jmsContext.createConsumer(replyQueue);
            //MapMessage replyMessage = (MapMessage) consumer.receive(30000);
            //System.out.println("Patient eligibility is: " + replyMessage.getBoolean("eligible"));
        }
    }
}
```

### MessageConsumer:

```
ClinicalsApp.java  EligibilityCheckerApp.java  jndi.properties
package com.bharath.jms.hm.eligibilitycheck;
import javax.jms.JMSConsumer;

public class EligibilityCheckerApp {
    public static void main(String[] args) throws NamingException, InterruptedException {
        InitialContext initialContext = new InitialContext();
        Queue requestQueue = (Queue) initialContext.lookup("queue/requestQueue");

        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
             JMSContext jmsContext = cf.createContext("eligibilityuser", "eligibilitypass")) {
            JMSConsumer consumer1 = jmsContext.createConsumer(requestQueue);
            JMSConsumer consumer2 = jmsContext.createConsumer(requestQueue);
            //consumer.setMessageListener(new EligibilityCheckListener());

            for(int i=1;i<=10;i+=2) {
                System.out.println("Consumer1: "+consumer1.receive());
                System.out.println("Consumer2: "+consumer2.receive());
            }

            //Thread.sleep(10000);
        };
    }
}
```

### Message Grouping:

Send multiple messages as a group of message having a group message id

to a particular consumer.

The screenshot shows a Java code editor window with the title bar "MessageGroupingDemo.java". The code itself is as follows:

```
public class MessageGroupingDemo {  
    public static void main(String[] args) throws Exception {  
        InitialContext context = new InitialContext();  
        Queue queue = (Queue) context.lookup("queue/myQueue");  
        Map<String, String> receivedMessages = new ConcurrentHashMap<>();  
  
        try (ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();  
             JMSContext jmsContext = cf.createContext();  
             JMSContext jmsContext2 = cf.createContext() {  
            JMSProducer producer = jmsContext.createProducer();  
            JMSConsumer consumer1 = jmsContext2.createConsumer(queue);  
            consumer1.setMessageListener(new MyListener("Consumer-1", receivedMessages));  
            JMSConsumer consumer2 = jmsContext2.createConsumer(queue);  
            consumer2.setMessageListener(new MyListener("Consumer-2", receivedMessages));  
  
            int count = 10;  
            TextMessage[] messages = new TextMessage[count];  
            for (int i = 0; i < count; i++) {  
                messages[i] = jmsContext.createTextMessage("Group-0 message" + i);  
                messages[i].setStringProperty("JMSXGroupID", "Group-0");  
                producer.send(queue, messages[i]);  
            }  
  
            for (TextMessage message : messages) {  
                if (!receivedMessages.get(message.getText()).equals("Consumer-1")) {  
                    throw new IllegalStateException(  
                        "Group Message" + message.getText() + "has gone to the wrong receiver");  
                }  
            }  
        }  
    }  
}
```

Put a Thread.sleep(2000); before checking Consumer-1 or 2.

```
MessageGroupingDemo.java
```

```
for (TextMessage message : messages) {
    if (!receivedMessages.get(message.getText()).equals("Consumer-1")) {
        throw new IllegalStateException(
            "Group Message" + message.getText() + "has gone to the wrong receiver");
    }
}

}

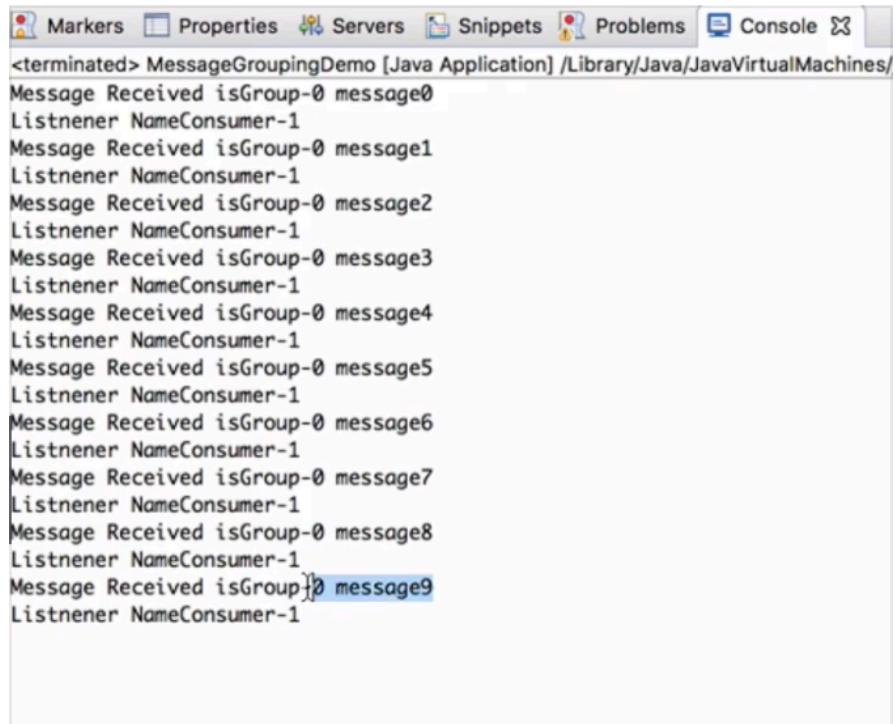
}

class MyListener implements MessageListener {

    private final String name;
    private final Map<String, String> receivedMessages;

    MyListener(String name, Map<String, String> receivedMessages) {
        this.name = name;
        this.receivedMessages = receivedMessages;
    }

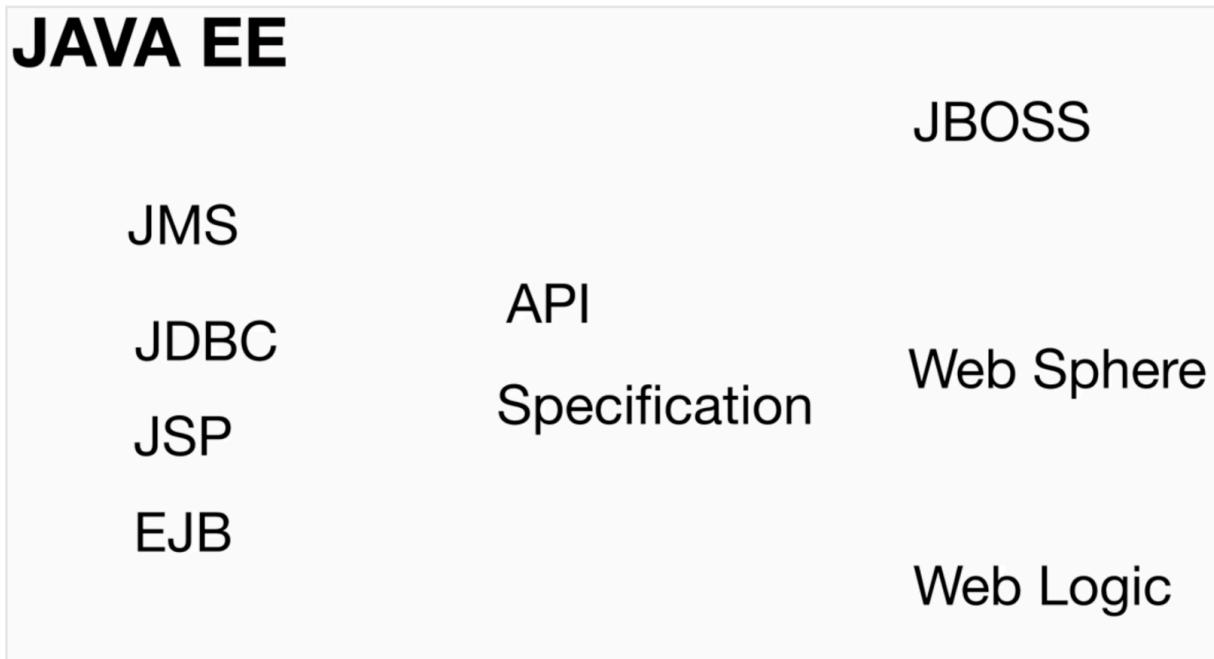
    @Override
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println("Message Received is" + textMessage.getText());
            System.out.println("Listnener Name" + name);
            receivedMessages.put(textMessage.getText(), name);
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }
}
```



The screenshot shows a Java application running in an IDE. The console tab is active, displaying the following log output:

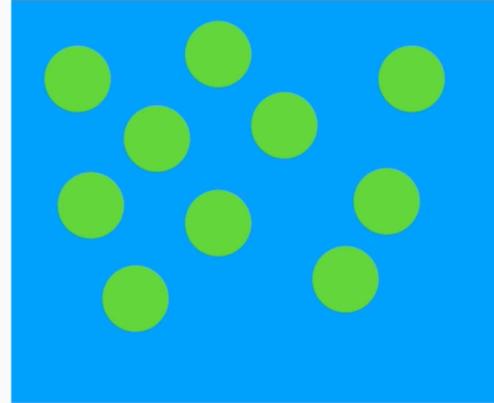
```
<terminated> MessageGroupingDemo [Java Application] /Library/Java/JavaVirtualMachines/  
Message Received isGroup-0 message0  
Listnener NameConsumer-1  
Message Received isGroup-0 message1  
Listnener NameConsumer-1  
Message Received isGroup-0 message2  
Listnener NameConsumer-1  
Message Received isGroup-0 message3  
Listnener NameConsumer-1  
Message Received isGroup-0 message4  
Listnener NameConsumer-1  
Message Received isGroup-0 message5  
Listnener NameConsumer-1  
Message Received isGroup-0 message6  
Listnener NameConsumer-1  
Message Received isGroup-0 message7  
Listnener NameConsumer-1  
Message Received isGroup-0 message8  
Listnener NameConsumer-1  
Message Received isGroup-0 message9  
Listnener NameConsumer-1
```

Java EE and Message Driven Beans:



# EJB

Session Bean  
State Less  
State Full  
MDB  
Asynchronous  
Pool



Auto-wiring in recent EE versions for beans available in application servers.

```
@Resource(mappedName="jndi name of the queue here")
```

```
Queue myQueue
```

Application server will create from scratch and inject it.

```
@Inject  
JMSContext jmsContext;
```

Inject EJB to servlet or in any other place.

```
@EJB  
MyMessageProducer producer;
```

Install Jboss WildFly server:

WildFly download Zip for "Java EE Full and Web Distribution".

WldFly\_Home/standalone/configuration/standalone-full.xml

WldFly\_Home/standalone/deployments/war files here

Creating the queue:

Start the server

Define the queue

Check if the queue is created.

Commands:

Start JBoss with standalone-full.xml to start with JMS queue support.

```
./standalone.sh -c standalone-full.xml
```

```
./jboss-cli.sh
```

Above command will start in disconnected mode. Type command 'connect' to connect it.

```
jms-queue add --queue-address=myQueue --entries=queue/  
myQueue,java:jboss/exported/jms/queue/myQueue
```

If above command does not give any error then it created JMS queue named myQueue

To delete above command use command: jms-queue delete —queue-address=myQueue

To check whether myQueue is created or not use below command.

```
/subsystem=messaging-activemq/server=default/jms-queue=myQueue:read-  
resource
```



The screenshot shows a macOS Terminal window with the following session:

```
Terminal Shell View Window Help 21% Tue 12:04 PM  
bin — java --add-exports=java.base/sun.nio.ch=ALL-UNNAMED --add-exports=jdk.unsupported/sun.misc=ALL-UNNAMED --add-exports=dk.unsupported/sun.reflect=ALL-UNNAMED -Djboss.modul...  
Last login: Tue Jul 16 11:59:11 on ttys000  
bharaths-MBP:~ bharaththippireddy$ cd Documents/wildfly-17.0.1.Final/bin  
bharaths-MBP:bin bharaththippireddy$ ./jboss-cli.sh  
You are disconnected at the moment. Type 'connect' to connect to the server or 'help' for the list of supported commands.  
[disconnected ] connect  
[standalone@localhost:9990 /] jms-queue add --queue-address=myQueue --entries=queue/myQueue,java:jboss/exported/jms/queue/myQueue  
[standalone@localhost:9990 /] /subsystem=messaging-activemq/server=default/jms-queue=myQueue:read-resource  
{  
    "outcome" => "success",  
    "result" => {  
        "durable" => true,  
        "entries" => [  
            "queue/myQueue",  
            "java:jboss/exported/jms/queue/myQueue"  
        ],  
        "legacy-entries" => undefined,  
        "selector" => undefined  
    }  
}  
[standalone@localhost:9990 /]
```

Create a maven project:

Create a maven archetype web app project with javaee-api dependency in pom.xml. This dependency gives API facilities for

# Steps

Create a maven project

Create the producer

Create the Servlet

Create the MDB

Do the maven stuff

Deploy and Run the App

STS

In maven pom.xml scope=provided means it is not required when deployed because server will have that library.



```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.bharath.jeejms</groupId>
    <artifactId>jeejms</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>jeejms Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <finalName>jeejms</finalName>
    </build>
</project>
```

Create a Message Producer:

The screenshot shows an IDE interface with two tabs: "jeejms/pom.xml" and "MyMessageProducer.java". The "MyMessageProducer.java" tab is active, displaying Java code for a JMS producer. The code includes imports for javax.annotation.Resource, javax.ejb.LocalBean, javax.ejb.Stateless, javax.inject.Inject, javax.jms.JMSContext, and javax.jms.Queue. It uses annotations @Stateless and @LocalBean. The class MyMessageProducer has a method sendMessage that sends a message to a queue named "myQueue". The code is annotated with line numbers from 1 to 25.

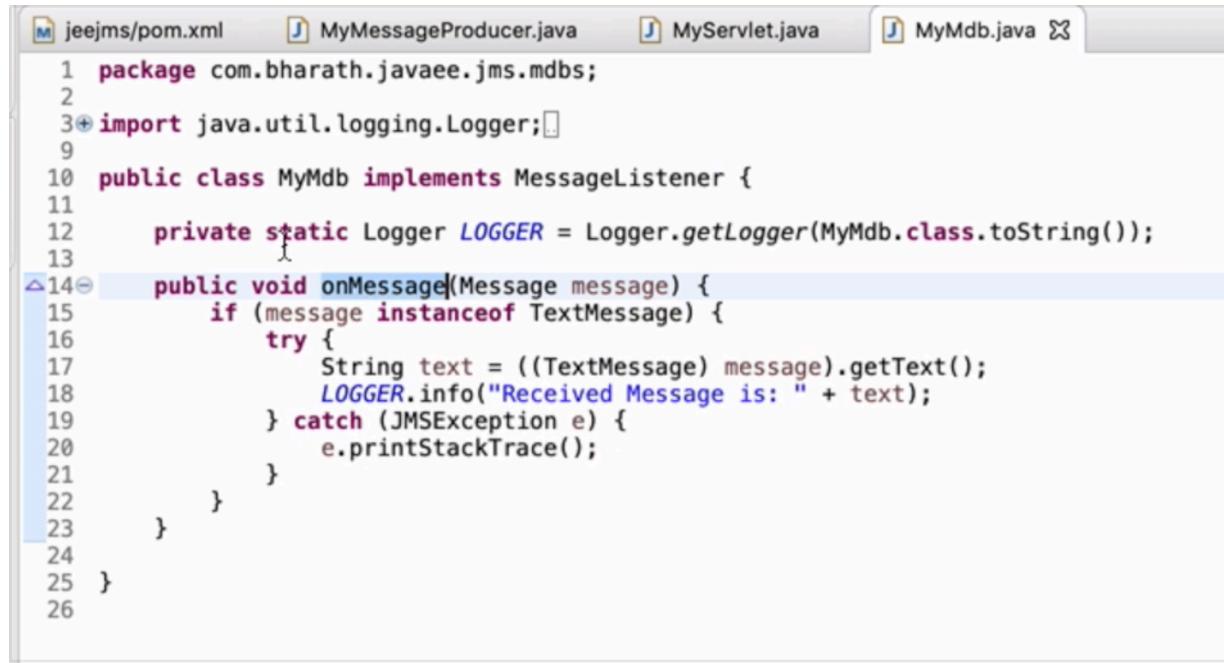
```
1 package com.bharath.javaee.jms;
2
3 import javax.annotation.Resource;
4 import javax.ejb.LocalBean;
5 import javax.ejb.Stateless;
6 import javax.inject.Inject;
7 import javax.jms.JMSContext;
8 import javax.jms.Queue;
9
10 @Stateless
11 @LocalBean
12 public class MyMessageProducer {
13
14     @Resource(mappedName = "java:/queue/myQueue")
15     Queue myQueue;
16
17     @Inject
18     JMSContext jmsContext;
19
20     public void sendMessage(String message) {
21         jmsContext.createProducer().send(myQueue, message);
22     }
23
24 }
25
```

Create servlet:

The screenshot shows an IDE interface with three tabs: "jeejms/pom.xml", "MyMessageProducer.java", and "\*MyServlet.java". The "\*MyServlet.java" tab is active, displaying Java code for a servlet. The code includes imports for java.io.IOException, javax.ejb.EJB, javax.servlet.ServletException, javax.servlet.annotation.WebServlet, javax.servlet.http.HttpServlet, javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse, and com.bharath.javaee.jms.MyMessageProducer. It uses annotations @WebServlet(urlPatterns = "/") and @EJB. The class MyServlet extends HttpServlet and overrides the doGet method. The doGet method retrieves a message from the JMS producer and writes it to the response. The code is annotated with line numbers from 1 to 30.

```
1 package com.bharath.javaee.servlets;
2
3 import java.io.IOException;
4
5 import javax.ejb.EJB;
6 import javax.servlet.ServletException;
7 import javax.servlet.annotation.WebServlet;
8 import javax.servlet.http.HttpServlet;
9 import javax.servlet.http.HttpServletRequest;
10 import javax.servlet.http.HttpServletResponse;
11
12 import com.bharath.javaee.jms.MyMessageProducer;
13
14 @WebServlet(urlPatterns = "/")
15 public class MyServlet extends HttpServlet {
16
17     private static final long serialVersionUID = 1L;
18     @EJB
19     MyMessageProducer producer;
20
21     @Override
22     protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
23         String message = "Hello Message from JavaEE Server using JMS!!";
24         producer.sendMessage(message);
25         resp.getWriter().write("Published the message" + message);
26     }
27 }
28
29 }
30
```

Create MDB:



The screenshot shows a Java code editor with several tabs at the top: jeejms/pom.xml, MyMessageProducer.java, MyServlet.java, and MyMdb.java. The MyMdb.java tab is active. The code is a Java class named MyMdb that implements the MessageListener interface. It contains a static logger and an onMessage method that logs the received message text if it is a TextMessage. The code is numbered from 1 to 26.

```
1 package com.bharath.javaee.jms.mdbc;
2
3+import java.util.logging.Logger;
4
5 public class MyMdb implements MessageListener {
6
7     private static Logger LOGGER = Logger.getLogger(MyMdb.class.toString());
8
9
10    public void onMessage(Message message) {
11        if (message instanceof TextMessage) {
12            try {
13                String text = ((TextMessage) message).getText();
14                LOGGER.info("Received Message is: " + text);
15            } catch (JMSException e) {
16                e.printStackTrace();
17            }
18        }
19    }
20
21}
22
23}
24
25}
26
```

### Configure MDB:

```
@MessageDriven(name="MyMdb", activationConfig= {
    @ActivationConfigProperty(propertyName="destination",
    propertyValue="queue/myQueue"), // receive message from this queue, java:
    is not needed, that is implicit.
    @ActivationConfigProperty(propertyName="destinationType",
    propertyValue="javax.jms.Queue"),
    @ActivationConfigProperty(propertyName="acknowledgeMode",
    propertyValue="Auto-acknowledge")
})
```

```

1 package com.bharath.javaee.jms.mdbc;
2
3 import java.util.logging.Logger;
4
5 import javax.ejb.ActivationConfigProperty;
6 import javax.ejb.MessageDriven;
7 import javax.jms.JMSException;
8 import javax.jms.Message;
9 import javax.jms.MessageListener;
10 import javax.jms.TextMessage;
11
12 @MessageDriven(name = "MyMdb", activationConfig = {
13     @ActivationConfigProperty(propertyName = "destination", propertyValue = "queue/myQueue"),
14     @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
15     @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge") })
16 public class MyMdb implements MessageListener {
17
18     private static Logger LOGGER = Logger.getLogger(MyMdb.class.toString());
19
20     public void onMessage(Message message) {
21         if (message instanceof TextMessage) {
22             try {
23                 String text = ((TextMessage) message).getText();
24                 LOGGER.info("Received Message is: " + text);
25             } catch (JMSException e) {
26                 e.printStackTrace();
27             }
28         }
29     }
30 }
31

```

Update the project and pom.xml:

Changed version to 1.8, failOnMissingWebXml to false, war build plugin and in last IDE JDK version to 1.8.

```

1<project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <groupId>com.bharath.jee.jms</groupId>
6   <artifactId>jeejms</artifactId>
7   <packaging>war</packaging>
8   <version>0.0.1-SNAPSHOT</version>
9   <name>jeejms Maven Webapp</name>
10  <url>http://maven.apache.org</url>
11
12  <properties>
13      <maven.compiler.source>1.8</maven.compiler.source>
14      <maven.compiler.target>1.8</maven.compiler.target>
15      <failOnMissingWebXml>false</failOnMissingWebXml>
16  </properties>
17
18  <dependencies>
19      <dependency>
20          <groupId>javax</groupId>
21          <artifactId>javaee-api</artifactId>
22          <version>8.0</version>
23          <scope>provided</scope>
24      </dependency>
25  </dependencies>
26  <build>
27      <finalName>jeejms</finalName>
28  <plugins>
29      <plugin>
30          <artifactId>maven-war-plugin</artifactId>
31          <version>3.2.3</version>
32      </plugin>
33  </plugins>
34 </build>
35 </project>
36

```

Text version of the updated pom.xml:

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.bharath.jee.jms</groupId>
    <artifactId>jeejms</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>jeejms Maven Webapp</name>
    <url>http://maven.apache.org</url>

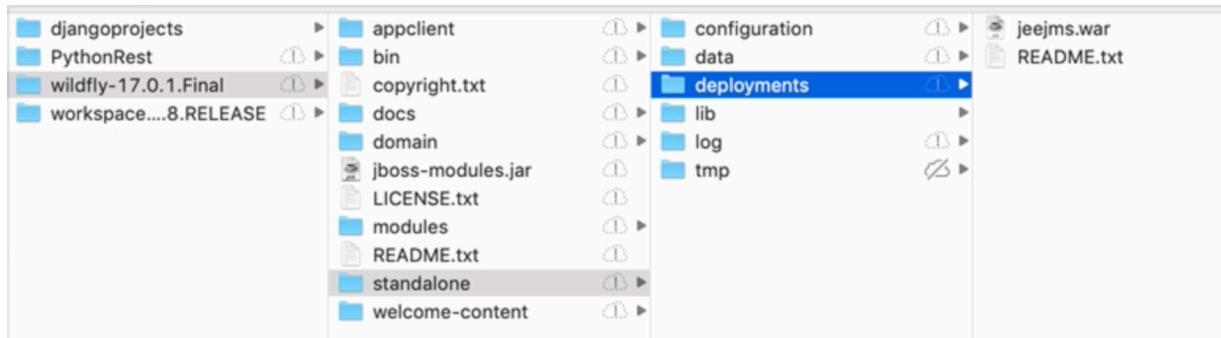
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <failOnMissingWebXml>false</failOnMissingWebXml>
    </properties>

    <dependencies>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-api</artifactId>
            <version>8.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <finalName>jeejms</finalName>
        <plugins>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.2.3</version>
            </plugin>
        </plugins>
    </build>
</project>
```

Deploy and Run:

Maven clean and build to get the deployable .war file.

Put the .war into wildFly\_Home/standalone/deployments/ path.



To run JBoss WildFly server go to WildFly\_Home/bin/ directory and run command: ./standalone.sh -c standalone-full.xml

```

Terminal Shell Edit View Window Help
bin — java - standalone.sh -c standalone-full.xml — 127x33
bharaths-MacBook-Pro:bin bharaththippireddy$ pwd
/Users/bharaththippireddy/Documents/wildfly-17.0.1.Final/bin
bharaths-MacBook-Pro:bin bharaththippireddy$ ./standalone.sh -c standalone-full.xml
=====
JBoss Bootstrap Environment
JBoss_HOME: /Users/bharaththippireddy/Documents/wildfly-17.0.1.Final
JAVA: java
JAVA_OPTS: -server -Xms64m -Xmx512m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true --add-exports=java.base/sun.nio.ch=ALL-UNNAMED --add-exports=jdk.unsupported/sun.misc=ALL-UNNAMED --add-exports=jdk.unsupported/sun.reflect=ALL-UNNAMED
=====
11:12:11,236 INFO [org.jboss.modules] (main) JBoss Modules version 1.9.1.Final

```

It will deploy the message bean and all as in below log

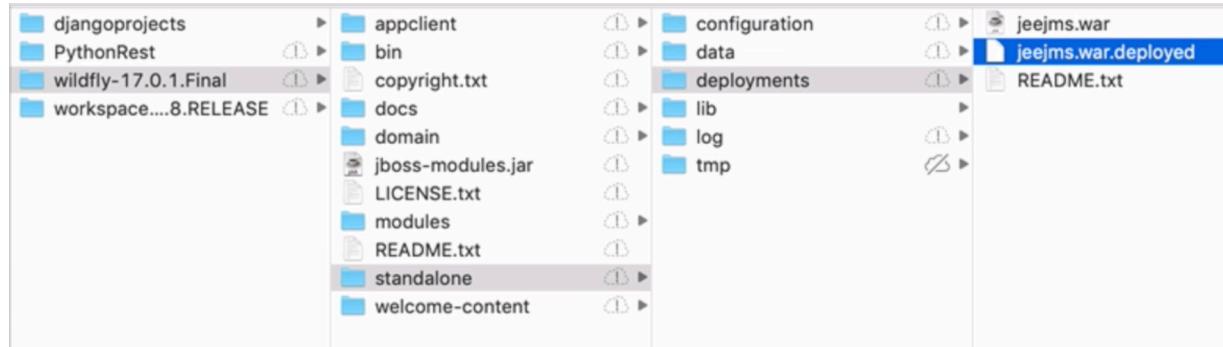
```

Terminal Shell Edit View Window Help
bin — java - standalone.sh -c standalone-full.xml — 127x33
11:12:15,487 INFO [org.hibernate.validator.internal.util.Version] (MSC service thread 1-7) HV000001: Hibernate Validator 6.0.1
6.Final
11:12:15,666 INFO [org.apache.activemq.artemis.core.server] (ServerService Thread Pool -- 82) AMQ221007: Server is now live
11:12:15,668 INFO [org.apache.activemq.artemis.core.server] (ServerService Thread Pool -- 82) AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.8.1 [default, nodeID=26e8129a-a5b4-11e9-bd8b-38f9d35d4c48]
11:12:15,689 INFO [org.wildfly.extension.messaging-activemq] (ServerService Thread Pool -- 87) WFLYMSGAMQ0002: Bound messaging object to jndi name java:jboss/exported/jms/RemoteConnectionFactory
11:12:15,691 INFO [org.wildfly.extension.messaging-activemq] (ServerService Thread Pool -- 82) WFLYMSGAMQ0002: Bound messaging object to jndi name java:/ConnectionFactory
11:12:15,722 INFO [org.jboss.as.ejb3.deployment] (MSC service thread 1-7) WFLYEJB0473: JNDI bindings for session bean named 'MyMessageProducer' in deployment "jeejms.war" are as follows:
java:global/jeejms/MyMessageProducer!com.bharath.javaee.jms.MyMessageProducer
java:app/jeejms/MyMessageProducer!com.bharath.javaee.jms.MyMessageProducer
java:module/MyMessageProducer!com.bharath.javaee.jms.MyMessageProducer
ejb:/jeejms/MyMessageProducer!com.bharath.javaee.jms.MyMessageProducer
java:global/jeejms/MyMessageProducer
java:app/jeejms/MyMessageProducer
java:module/MyMessageProducer

11:12:15,747 INFO [org.jboss.as.connector.deployment] (MSC service thread 1-4) WFLYJCA0007: Registered connection factory java:/JmsXA
11:12:15,866 INFO [org.apache.activemq.artemis.ra] (MSC service thread 1-4) AMQ151007: Resource adaptor started
11:12:15,867 INFO [org.jboss.as.connector.services.resourceadapters.ResourceAdapterActivatorService$ResourceAdapterActivator] (MSC service thread 1-4) I03020002: Deployed: file://RaActivatoractivemq-ra
11:12:15,869 INFO [org.jboss.as.connector.deployment] (MSC service thread 1-1) WFLYJCA0002: Bound JCA ConnectionFactory [java:/JmsXA]
11:12:15,870 INFO [org.jboss.as.connector.deployment] (MSC service thread 1-2) WFLYJCA0118: Binding connection factory named java:/JmsXA to alias java:jboss/DefaultJMSSConnectionFactory
11:12:16,003 INFO [org.infinispan.factories.GlobalComponentRegistry] (MSC service thread 1-3) ISPN000128: Infinispan version: Infinispan 'Infinity Minus ONE +2' 9.4.14.Final
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.jboss.invocation.proxy.AbstractProxyFactory$1 (jar:file:/Users/bharaththippireddy/Doc
```

It will create warFileName.war.deployed in JBoss wildFly deployments

directory.



It will run successfully if hit with JBoss default port 8080 means localhost:8080/warFileContextName/ as below.



It also posts JMS message and receives as highlighted in below log:

```
Terminal Shell Edit View Window Help
bin — java - standalone.sh -c standalone-full.xml — 127x33
/JmsXA]
11:12:15,870 INFO [org.jboss.as.connector.deployment] (MSC service thread 1-2) WFLYJCA0118: Binding connection factory named j
ava:/JmsXA to alias java:jboss/DefaultJMSConnectionFactory
11:12:16,003 INFO [org.infinispan.factories.GlobalComponentRegistry] (MSC service thread 1-3) ISPN000128: Infinispan version:
Infinispan 'Infinity Minus ONE +2' 9.4.14.Final
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.jboss.invocation.proxy.AbstractProxyFactory$1 (jar:file:/Users/bharathhippireddy/Doc
uments/wildfly-17.0.1.Final/modules/system/layers/base/org/jboss/invocation/main/jboss-invocation-1.5.2.Final.jar!/) to method
java.lang.Object.clone()
WARNING: Please consider reporting this to the maintainers of org.jboss.invocation.proxy.AbstractProxyFactory$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
11:12:16,141 INFO [org.jboss.weld.Version] (MSC service thread 1-1) WELD-000900: 3.1.1 (Final)
11:12:16,232 WARN [org.jboss.as.ejb3] (MSC service thread 1-5) WFLYEJB0006: ActivationConfigProperty acknowledgeMode will be ig
ored since it is not allowed by resource adapter: activemq-ra
11:12:16,254 INFO [org.jboss.as.ejb3] (MSC service thread 1-5) WFLYEJB0042: Started message driven bean 'MyMdb' with 'activem
-ra.rar' resource adapter
11:12:16,513 INFO [io.smallrye.metrics] (MSC service thread 1-5) MicroProfile: Metrics activated
11:12:16,561 INFO [org.jboss.as.clustering.infinispan] (ServerService Thread Pool -- 82) WFLYCLINF0002: Started client-mapping
s cache from ejb container
11:12:17,177 INFO [org.wildfly.extension.undertow] (ServerService Thread Pool -- 88) WFLYUT0021: Registered web context: '/jee
jms' for server 'default-server'
11:12:17,257 INFO [org.jboss.as.server] (ServerService Thread Pool -- 47) WFLYSRV0010: Deployed "jeejms.war" (runtime-name : "
jeejms.war")
11:12:17,312 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0212: Resuming server
11:12:17,327 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0060: Http management interface listening on http://127.0.0.1
:9990/management
11:12:17,328 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990
11:12:17,328 INFO [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: WildFly Full 17.0.1.Final (WildFly Core 9.0.2.Final) st
arted in 6441ms - Started 524 of 748 services (399 services are lazy, passive or on-demand)
11:13:03,389 INFO [class com.bharath.javaee.jms.mdb.MyMdb] (Thread-1 (ActiveMQ-client-global-threads)) Received Message is: H
ello Message from JavaEE Server using JMS!!
```

Spring JMS Overview:

# Spring JMS

## JmsTemplate

send

Listeners

receive

@JmsListener

convertAndSend

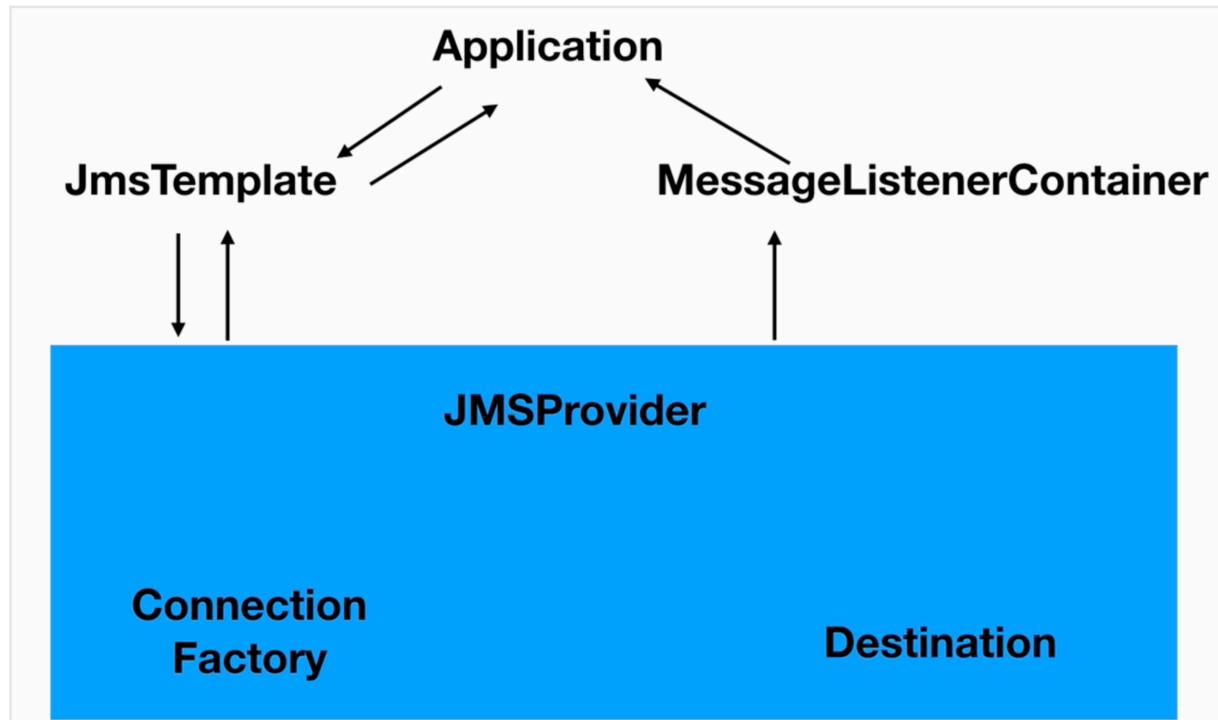
MDPs

String to TextMessage

Object to ObjectMessage

convertAndSend also provides facility for byte array to ByteMessage, Map to MapMessage or even with custom converter for JSON/XML etc.

@JMSListeners are like JBoss MessageDrivenBean but more like MessageDrivenPojo (MDP instead of MDB)



Create project:

# Steps

## Create the Project

### Create a Sender JMSTemplate

### Create a Receiver/Listener

Create a Spring boot project with artemis support i.e. with spring-boot-starter-artemis dependency.

```
springjms/pom.xml ✘
8      <version>2.1.9.RELEASE</version>
9      <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.bharath.springjms</groupId>
12     <artifactId>springjms</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>springjms</name>
15     <description>Spring JMS Demo</description>
16
17     <properties>
18         <java.version>1.8</java.version>
19     </properties>
20
21     <dependencies>
22         <dependency>
23             <groupId>org.springframework.boot</groupId>
24             <artifactId>spring-boot-starter-artemis</artifactId>
25         </dependency>
26
27         <dependency>
28             <groupId>org.springframework.boot</groupId>
29             <artifactId>spring-boot-starter-test</artifactId>
30             <scope>test</scope>
31         </dependency>
32     </dependencies>
33
34     <build>
35         <plugins>
```

Add @EnableJms to spring boot starter main class.

The screenshot shows a Java IDE interface with the following tabs at the top: 'springjms/pom.xml', 'SpringjmsApplication.java', and another tab that is partially visible. The code editor displays the following Java code:

```
1 package com.bharath.springjms;
2
3+ import org.springframework.boot.SpringApplication;[]
4
5 @SpringBootApplication
6 @EnableJms
7 public class SpringjmsApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringjmsApplication.class, args);
11     }
12
13 }
14
15 }
16
```

Message Sender:

The screenshot shows a Java IDE interface with the following tabs at the top: 'springjms/pom.xml', 'SpringjmsApplication.java', 'MessageSender.java', and 'application.properties'. The code editor displays the following content in the 'application.properties' file:

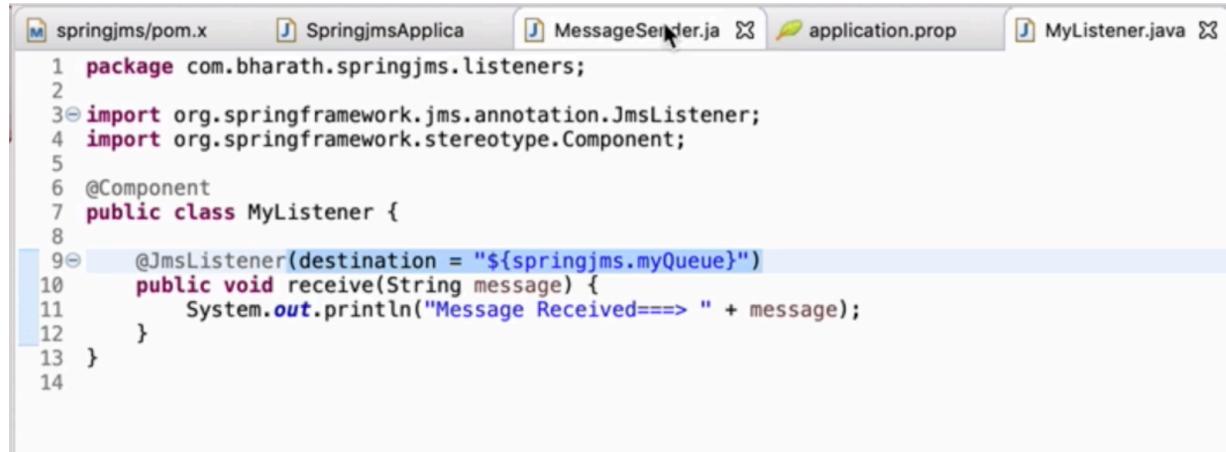
```
1 springjms.myQueue=myQueue
```

The screenshot shows a Java IDE interface with the following tabs at the top: 'springjms/pom.x', 'SpringjmsAplica', 'MessageSender.ja', and 'application'. The code editor displays the following Java code:

```
1 package com.bharath.springjms.senders;
2
3+ import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Value;
5 import org.springframework.jms.core.JmsTemplate;
6 import org.springframework.stereotype.Component;
7
8 @Component
9 public class MessageSender {
10
11     @Autowired
12     private JmsTemplate jmsTemplate;
13
14     @Value("${springjms.myQueue}")
15     private String queue;
16
17     public void send(String message) {
18         jmsTemplate.convertAndSend(queue, message);
19     }
20 }
21
```

Message Receiver:

jmsTemplate.receive gets blocked until there is any message. Better to use @JMSListener

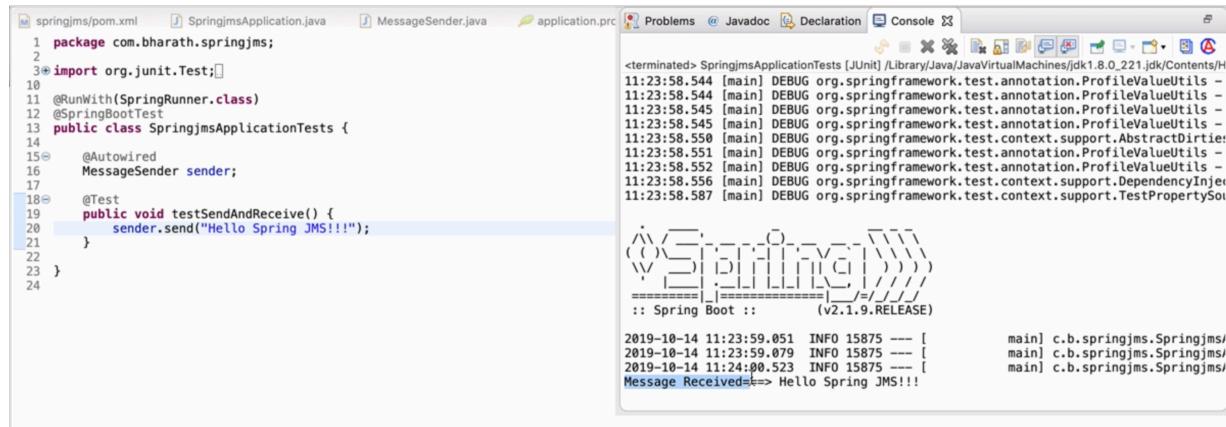


A screenshot of an IDE showing Java code for a JMS listener. The code is in a file named MyListener.java. It imports org.springframework.jms.annotation.JmsListener and org.springframework.stereotype.Component. The class is annotated with @Component and has a method receive that is annotated with @JmsListener(destination = "\${springjms.myQueue}"). The code prints the received message to System.out.

```
1 package com.bharath.springjms.listeners;
2
3 import org.springframework.jms.annotation.JmsListener;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class MyListener {
8
9     @JmsListener(destination = "${springjms.myQueue}")
10    public void receive(String message) {
11        System.out.println("Message Received====> " + message);
12    }
13 }
14
```

Test:

Make sure artemis is up.



A screenshot of an IDE showing a test class SpringjmsApplicationTests. The test method testSendAndReceive sends a message to the queue and prints it. The output shows the message being received by the test.

```
1 package com.bharath.springjms;
2
3 import org.junit.Test;
4 import org.springframework.boot.test.context.SpringBootTest;
5 import org.springframework.test.context.junit4.SpringRunner;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.jms.core.JmsTemplate;
8
9 @RunWith(SpringRunner.class)
10@SpringBootTest
11 public class SpringjmsApplicationTests {
12
13     @Autowired
14     JmsTemplate sender;
15
16     @Test
17     public void testSendAndReceive() {
18         sender.convertAndSend("Hello Spring JMS!!!");
19     }
20
21 }
```

Output:

```
<terminated> SpringjmsApplicationTests [JUnit] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/H...
11:23:58.544 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.544 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.545 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.545 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.545 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.546 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.546 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.546 [main] DEBUG org.springframework.test.context.support.AbstractDirtie...
11:23:58.551 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.552 [main] DEBUG org.springframework.test.annotation.ProfileValueUtils -
11:23:58.556 [main] DEBUG org.springframework.test.context.support.DependencyInje...
11:23:58.587 [main] DEBUG org.springframework.test.context.support.TestPropertyS...
:: Spring Boot ::      (v2.1.9.RELEASE)
=====
2019-10-14 11:23:59.051 INFO 15875 --- [        main] c.b.springjms.Springjms/
2019-10-14 11:23:59.079 INFO 15875 --- [        main] c.b.springjms.Springjms/
2019-10-14 11:24:00.523 INFO 15875 --- [        main] c.b.springjms.Springjms/
Message Received====> Hello Spring JMS!!!
```

```

bin -- java -XX:+PrintClassHistogram -XX:+UseG1GC -Xms512M -Xmx2G -Dhawtio.realm=activemq -Dhawtio.offline=true -Dhawtio.role=amq -Dhawtio.rolePrincipalClasses=org.apache.activemq.artemis
Apache ActiveMQ Artemis 2.10.1

2019-10-12 17:05:45,095 INFO [org.apache.activemq.artemis.integration.bootstrap] AMQ210100: Starting ActiveMQ Artemis Server
2019-10-12 17:05:45,145 INFO [org.apache.activemq.artemis.core.server] AMQ221000: live Message Broker is starting with configuration Broker Configuration (clustered=false,journalDirectory=data/journal,bindingsDirectory=data/bindings,largeMessageDirectory=data/large-messages,pagingDirectory=data/paging)
2019-10-12 17:05:45,184 INFO [org.apache.activemq.artemis.core.server] AMQ221013: Using NIO Journal
2019-10-12 17:05:45,242 INFO [org.apache.activemq.artemis.core.server] AMQ221057: Global Max Size is being adjusted to 1/2 of the JVM max size (-Xmx). being defined as 1,073,741,824
2019-10-12 17:05:45,277 INFO [org.apache.activemq.artemis.core.server] AMQ221043: Protocol module found: [artemis-server]. Adding protocol support for: CORE
2019-10-12 17:05:45,279 INFO [org.apache.activemq.artemis.core.server] AMQ221043: Protocol module found: [artemis-amp-protocol]. Adding protocol support for: AMQP
2019-10-12 17:05:45,285 INFO [org.apache.activemq.artemis.core.server] AMQ221043: Protocol module found: [artemis-hornetq-protocol]. Adding protocol support for: HORNETQ
2019-10-12 17:05:45,286 INFO [org.apache.activemq.artemis.core.server] AMQ221043: Protocol module found: [artemis-mqtt-protocol]. Adding protocol support for: MQTT
2019-10-12 17:05:45,287 INFO [org.apache.activemq.artemis.core.server] AMQ221043: Protocol module found: [artemis-openwire-protocol]. Adding protocol support for: OPENWIRE
2019-10-12 17:05:45,288 INFO [org.apache.activemq.artemis.core.server] AMQ221043: Protocol module found: [artemis-stomp-protocol]. Adding protocol support for: STOMP
2019-10-12 17:05:45,401 INFO [org.apache.activemq.artemis.core.server] AMQ221034: Waiting indefinitely to obtain live lock
2019-10-12 17:05:45,401 INFO [org.apache.activemq.artemis.core.server] AMQ221035: Live Server Obtained live lock
2019-10-12 17:05:45,566 INFO [org.apache.activemq.artemis.core.server] AMQ221080: Deploying address DLQ supporting [ANYCAST]
2019-10-12 17:05:45,521 INFO [org.apache.activemq.artemis.core.server] AMQ221080: Deploying ANYCAST queue DLQ on address DLQ
2019-10-12 17:05:45,598 INFO [org.apache.activemq.artemis.core.server] AMQ221080: Deploying address ExpiryQueue supporting [ANYCAST]
2019-10-12 17:05:45,599 INFO [org.apache.activemq.artemis.core.server] AMQ221080: Deploying ANYCAST queue ExpiryQueue on address ExpiryQueue
2019-10-12 17:05:45,996 INFO [org.apache.activemq.artemis.core.server] AMQ221029: Started KUEUE Acceptor at 0.0.0.0:61616 for protocols [CORE,MQTT,AMQP,STOMP,HORNETQ,OPENWIRE]
2019-10-12 17:05:45,999 INFO [org.apache.activemq.artemis.core.server] AMQ221029: Started KUEUE Acceptor at 0.0.0.0:61645 for protocols [HORNETQ,STOMP]
2019-10-12 17:05:44,883 INFO [org.apache.activemq.artemis.core.server] AMQ221029: Started KUEUE Acceptor at 0.0.0.0:5672 for protocols [AMQP]
2019-10-12 17:05:44,883 INFO [org.apache.activemq.artemis.core.server] AMQ221029: Started KUEUE Acceptor at 0.0.0.0:1883 for protocols [MQTT]
2019-10-12 17:05:44,812 INFO [org.apache.activemq.artemis.core.server] AMQ221029: Started KUEUE Acceptor at 0.0.0.0:61613 for protocols [STOMP]
2019-10-12 17:05:44,813 INFO [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
2019-10-12 17:05:44,813 INFO [org.apache.activemq.artemis.core.server] AMQ221001: Apache ActiveMQ Artemis Message Broker version 2.10.1 [0.0.0.0, nodeID=6ddbca46-ece4-11e9-adb2-38f9d8504c48]
2019-10-12 17:05:45,401 INFO [org.apache.activemq.hawtio.branding.PluginContextListener] initialized activemq-branding plugin
2019-10-12 17:05:46,494 INFO [org.apache.activemq.hawtio.plugin.PluginContextListener] initialized artemis-plugin plugin
2019-10-12 17:05:47,281 INFO [io.hawt.HawtioContextListener] Initialising hawtio services
2019-10-12 17:05:47,328 INFO [io.hawt.system.ConfigManager] Configuration will be discovered via system properties
2019-10-12 17:05:47,332 INFO [io.hawt.jmx.JmxTreeWatcher] Welcome to hawtio 1.5.5 : http://hawt.io/ : Don't cha wish your console was hawt like me? ;-)
2019-10-12 17:05:47,335 INFO [io.hawt.jmx.UploadManager] Using file upload directory: /Users/bharaththippireddy/Documents/mybroker/tmp/uploads
2019-10-12 17:05:47,378 INFO [io.hawt.web.AuthenticationFilter] Starting hawtio authentication filter, JAAS realm: "activemq" authorized role(s): "amq" role principal classes: "org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal"
2019-10-12 17:05:47,413 INFO [io.hawt.web.JolokiaConfiguredAgentServlet] Jolokia overridden property: [key=policyLocation, value=file:/Users/bharaththippireddy/Documents/mybroker/etc/jolokia-access.xml]
2019-10-12 17:05:47,448 INFO [io.hawt.web.RBACMBeanInvoker] Using MBean [hawtio:type=security,area=jmx,rank=0,name=HawtioDummyJMXSecurity] for role based access control
2019-10-12 17:05:47,617 INFO [io.hawt.system.ProxyWhitelist] Initial proxy whitelist: [localhost, 127.0.0.1, 192.168.0.103]
2019-10-12 17:05:48,151 INFO [org.apache.activemq.artemis] AMQ241001: HTTP Server started at http://localhost:8161
2019-10-12 17:05:48,152 INFO [org.apache.activemq.artemis] AMQ241002: Artemis Jolokia REST API available at http://localhost:8161/console/jolokia
2019-10-12 17:05:48,153 INFO [org.apache.activemq.artemis] AMQ241004: Artemis Console available at http://localhost:8161/console

```

```

springjms/pom.xml      SpringjmsApplication.java      MessageSender.java      application.properties
1 package com.bharath.springjms.senders;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Value;
5 import org.springframework.jms.core.JmsTemplate;
6 import org.springframework.jms.core.MessageCreator;
7 import org.springframework.stereotype.Component;
8
9 @Component
10 public class MessageSender {
11
12     @Autowired
13     private JmsTemplate jmsTemplate;
14
15     @Value("${springjms.myQueue}")
16     private String queue;
17
18     public void send(String message) {
19         MessageCreator mc = s -> s.createTextMessage("Hello Spring JMS!!!");
20         jmsTemplate.send(queue, mc);
21     }
22 }
23

```

Default value of `spring.jms.pub-sub-domain` is false means Spring configures queue for P2P communication - not topic for publish subscribe communication. We can configure in `application.properties` file for all JMS communication or for a particular `jmsTemplate` by using `jmsTemplate.setPubSubDomain(true)` as in below screenshot.

A screenshot of an IDE showing the `application.properties` file. The file contains the following configuration:

```
1 springjms.myQueue=myQueue
2 spring.jms.pub-sub-domain=true
```

A screenshot of an IDE showing Java code. The code is as follows:

```
1 package com.bharath.springjms.senders;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.beans.factory.annotation.Value;
5 import org.springframework.jms.core.JmsTemplate;
6 import org.springframework.jms.core.MessageCreator;
7 import org.springframework.stereotype.Component;
8
9 @Component
10 public class MessageSender {
11
12     @Autowired
13     private JmsTemplate jmsTemplate;
14
15     @Value("${springjms.myQueue}")
16     private String queue;
17
18     public void send(String message) {
19         MessageCreator mc = s -> s.createTextMessage("Hello Spring JMS!");
20         jmsTemplate.setp
21     }
22 }
```

A tooltip for the `jmsTemplate.setp` method is displayed, providing documentation for the `setPubSubDomain` method:

Configure the destination accessor with knowledge of the JMS domain used. Default is Point-to-Point (Queues).

This setting primarily indicates what type of destination to resolve if dynamic destinations are enabled.

Parameters:  
pubSubDomain "true" for the Publish/Subscribe domain ([Topics](#)), "false" for the Point-to-Point domain ([Queues](#))

See Also:  
[setDestinationResolver](#)

A screenshot of an IDE showing Java code. The code includes a configuration entry:

```
1 springjms.myQueue=myQueue
2 spring.artemis
```

A tooltip for the `spring.artemis` configuration section is displayed, specifically for the `password` property:

String  
Login password of the broker.

**Bonus Lecture**  
**Connect with me :**  
<http://www.bharathhippireddy.com/>

## **Maximum Discounts on my Other TOP Courses:**

**Complete Python Stack from core Python to Django REST Framework**

**Angular and React project creation with Java or Node backend**

**Docker , Kubernetes, Maven, Jenkins ,GIT , AWS EC2 ,Elastic Beanstalk,ELB,Auto Scaling and more in easy steps**

## **Hot and New Courses:**

**Docker , Kubernetes, Maven, Jenkins ,GIT , AWS EC2 ,Elastic Beanstalk,ELB,Auto Scaling and more in easy steps**

## **Serverless Using AWS Lambda For Java Developers**

### **Other Top Courses on Sale:**

**Spring Security**

**Gradle for Java Developers**

**Devops Tools and AWS for Java Microservice Developers**

**Yoga For Beginners**

**Java Courses: Java Design Patterns, Java Web Services, Java Messaging Service, JUnit and Mockito and Spring Frameworks in Easy Steps, Spring Boot Fundamentals, Spring Data JPA using Hibernate, Spring Data REST.**

**Core Java Made Easy**

**JDBC Servlets and JSP - Java Web Development Fundamentals**

**Spring Framework in Easy Steps**

**Spring Boot Fundamentals**

**Spring Data JPA Using Hibernate**

**Full Stack Development Using Spring Boot Angular and React**

**Java Web Services**

**Java Web Service Part 2**

**Junit and Mockito Crash Course**

**Spring Cloud Fundamentals**

**Java Design Patterns**

**REST APIs using Spring Data REST**

**Java Message Service - JMS Fundamentals**

## **Devops Tools:**

[\*\*Devops Tools and AWS for Java Microservice Developers\*\*](#)  
[\*\*Maven Crash Course\*\*](#)  
[\*\*Gradle For Java Developers\*\*](#)

## **Java Script Courses:**

[\*\*Node JS Made Easy for MEAN and MERN Stack\*\*](#)  
[\*\*Full Stack React with Node and Java Backend\*\*](#)  
[\*\*Full Stack Angular with Node and Java Backend\*\*](#)  
[\*\*TypeScript for beginners\*\*](#)

## **XML Courses:**

[\*\*XML and XML Schema Definition in Easy Steps\*\*](#)  
[\*\*XSLT XPATH and XQuery Fundamentals\*\*](#)

## **Python:**

[\*\*Django for Python Developers\*\*](#)  
[\*\*Django REST Framework\*\*](#)  
[\*\*Python Core and Advanced\*\*](#)

## **Free Courses:**

[\*\*JavaScript Fundamentals\*\*](#)  
[\*\*Advanced and Object Oriented JavaScript and ES6\*\*](#)