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

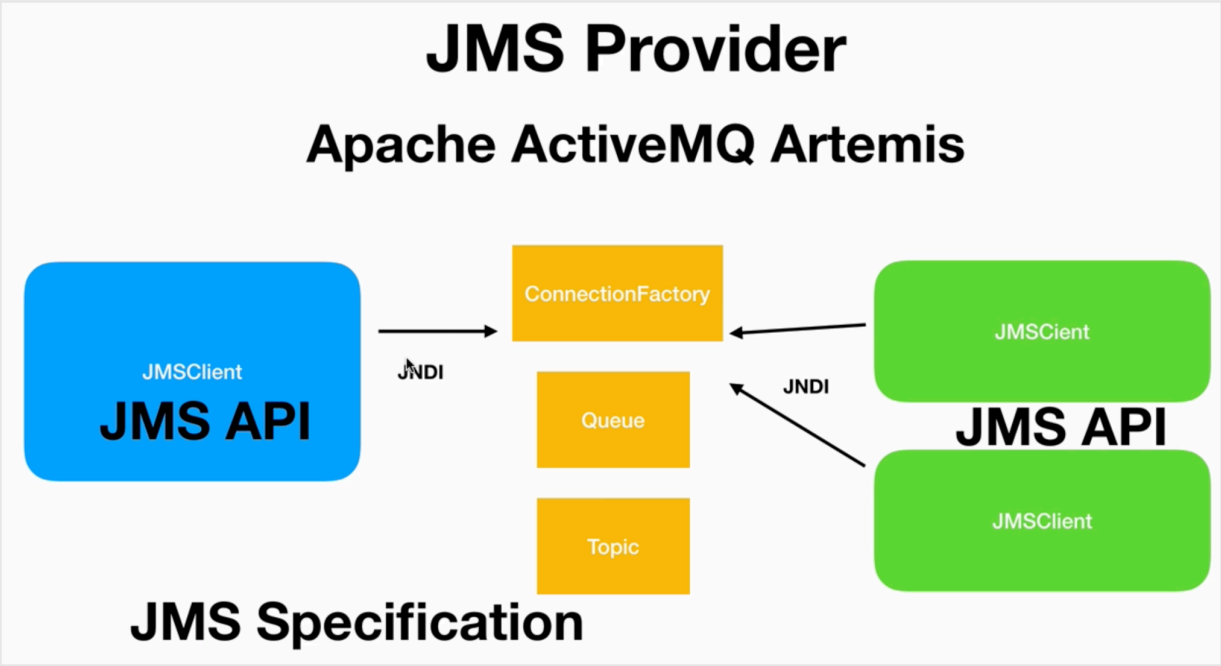
Publish/Subscribe:
One publisher, multiple subscribers
Push model, no poll needed
JMS providers pushes message to all subscriber applications who subscribed that particular topic.

News paper subscription

JMS Provider:
Apache ActiveMQ Artemis:
Open source:

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

Accessed using JNDI:



jmsprojects.zip
Zip archive
252 KB

ZIP

Software set up:
Use JDK - not JRE

Install Apache ActiveMQ Artemis:
/JMS_HOME/bin/artemis
/JMS_HOME/examples/

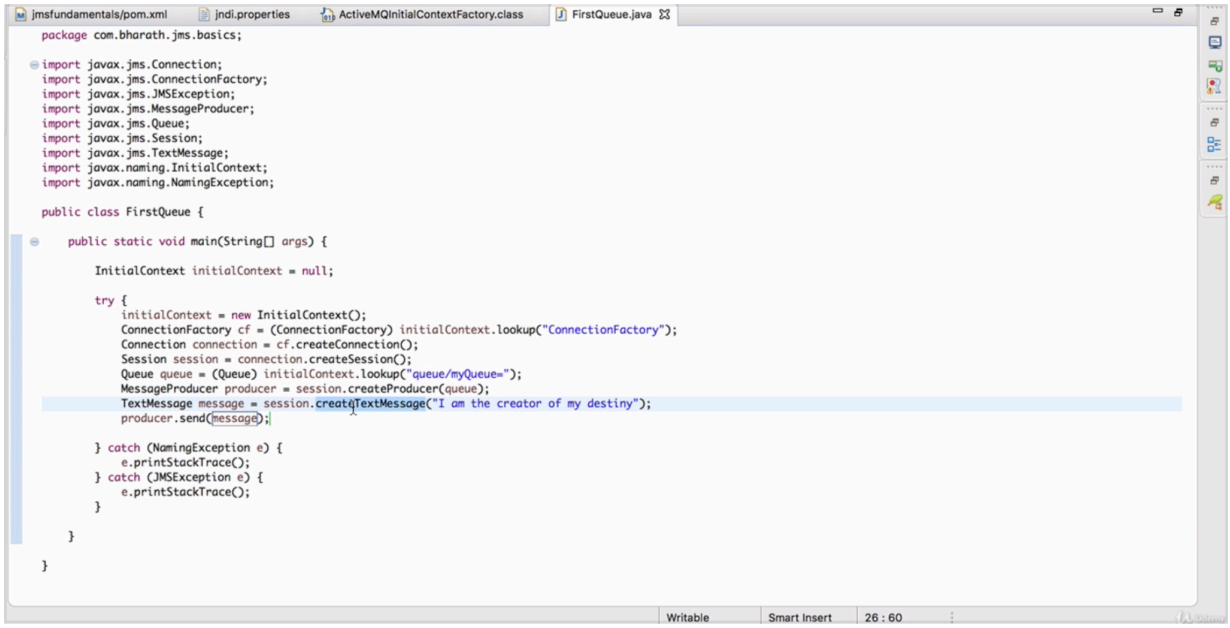
Creating the Messaging broker/JMS Provider:
/JMS_HOME/bin/>./artemis create /Users/suvendu/Documents/mybroker
admin
password -> admin
Allow anonymous access? Y
>cd /Users/suvendu/Documents/mybroker/bin
>ls
>artemis artemis-service
>./artemis run
>
>vi /Users/suvendu/Documents/mybroker/etc/broker.xml

>

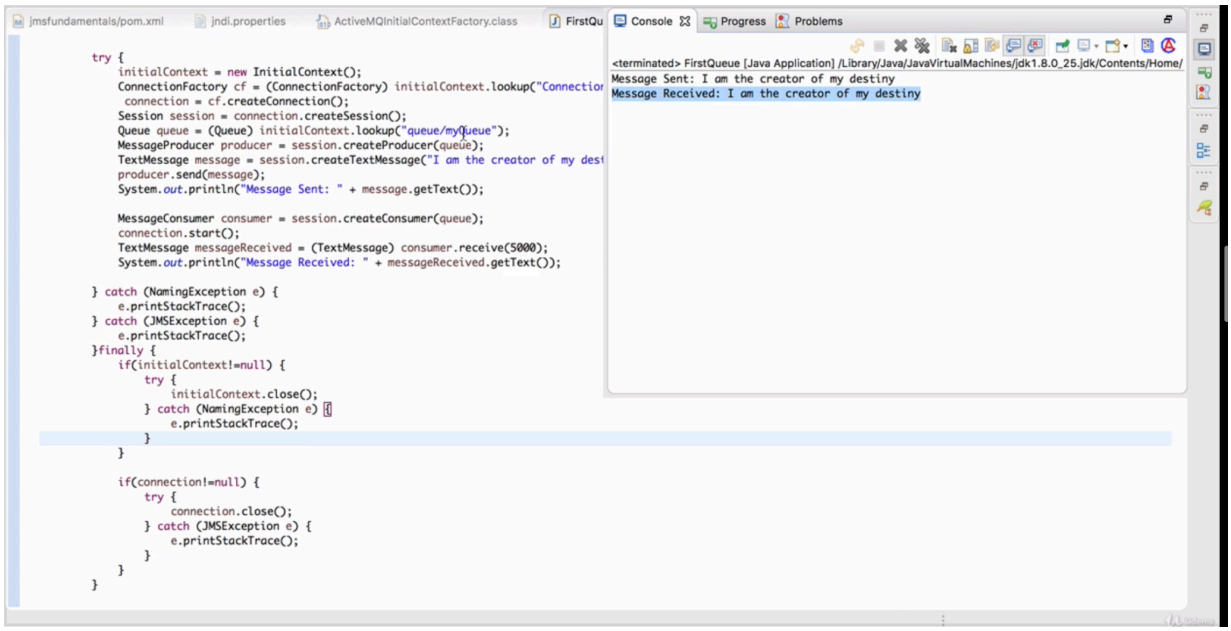
Messaging in action:
JMS 1.X API: 7 Components
ConnectionFactory - Registered with JNDI
Destination - Queue/Topic - Registered with JNDI
Connection
Session - n sessions from a connection
Message - Can be created once we have session
MessageProducer - QueueProducer/TopicProducer
MessageConsumer - QueueConsumer/TopicConsumer

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

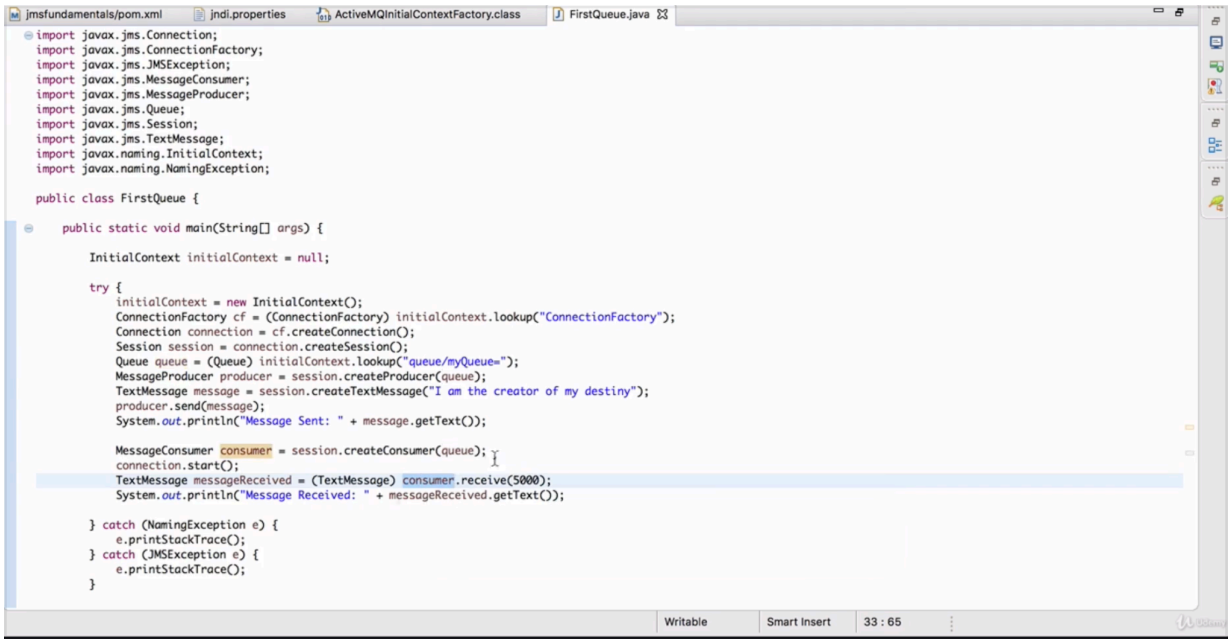


Consume a message from the Queue: P2P in action:

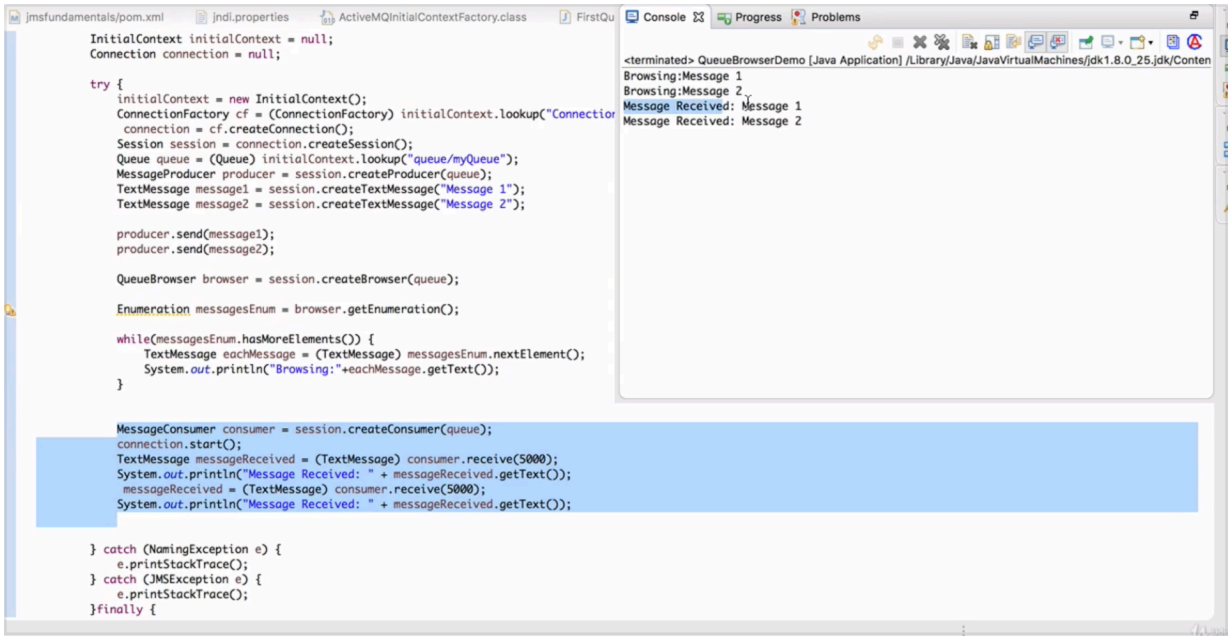


Code will create a queue if not initialized already.

Publish/Subscribe:
Add below to jndi.properties:
topic.**topic/myTopic**=myTopic



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.



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.

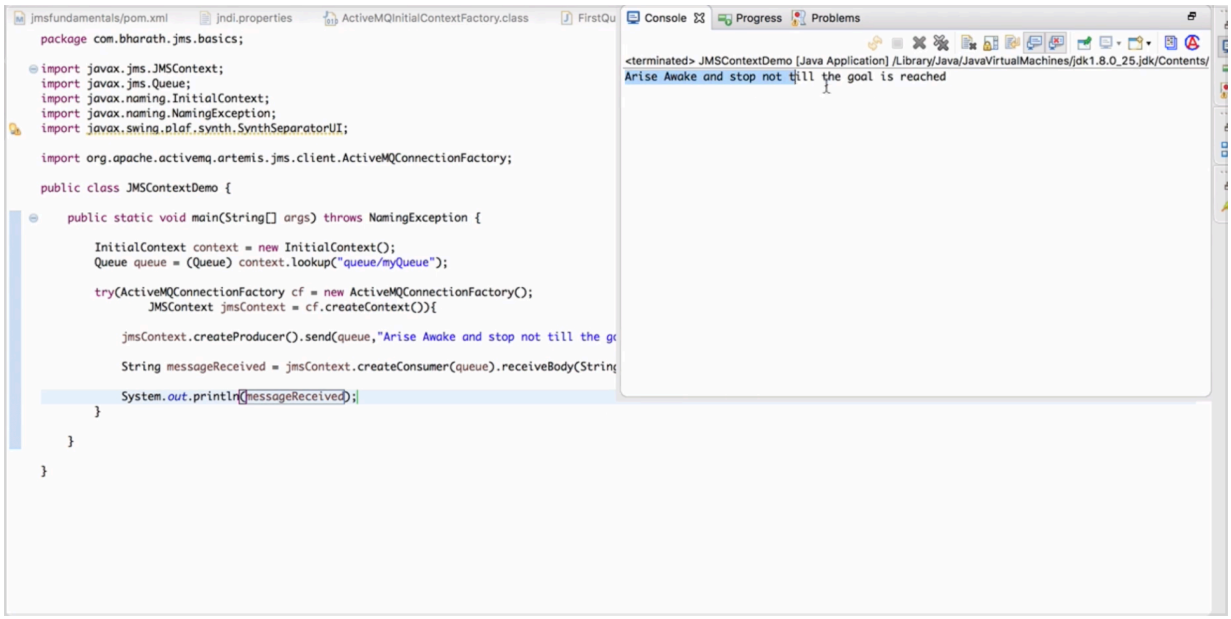
With application server having JEE 7,

@Inject
@JMSConnectionFactory(“jms/connectionFactory”)
private JMSContext context;

@Resource(lookup=“jms/dataQueue”)
private Queue dataQueue;

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

JMS 2.0:



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.

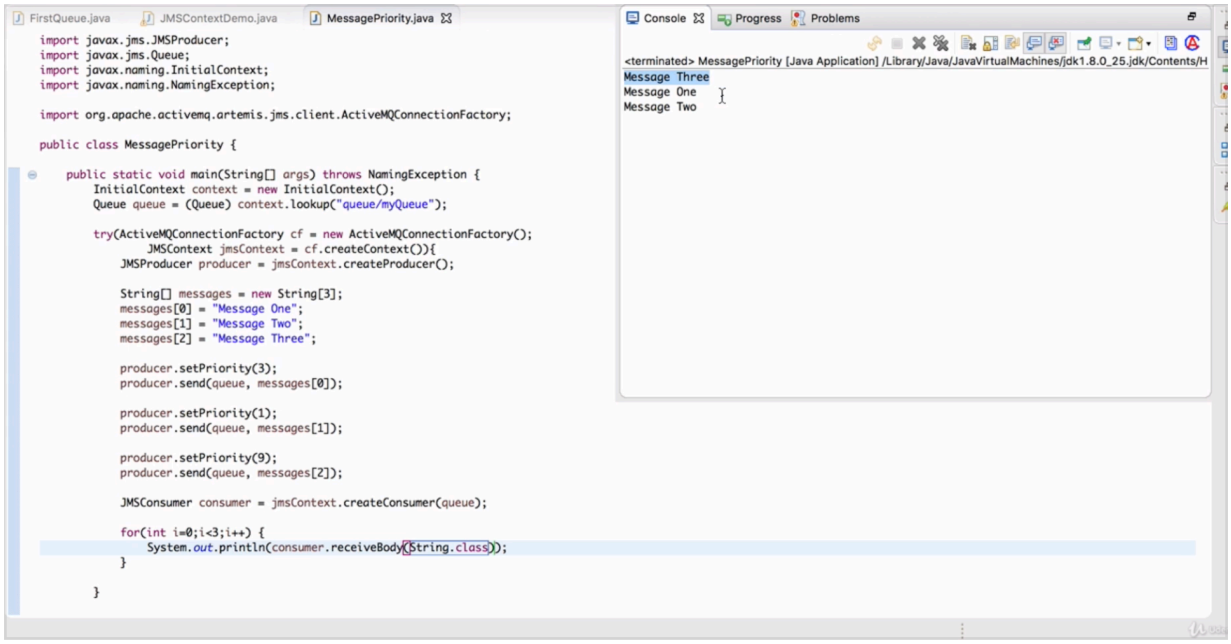
Properties:
1. Application specific
2. JMS defined specific
3. Provider specific

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

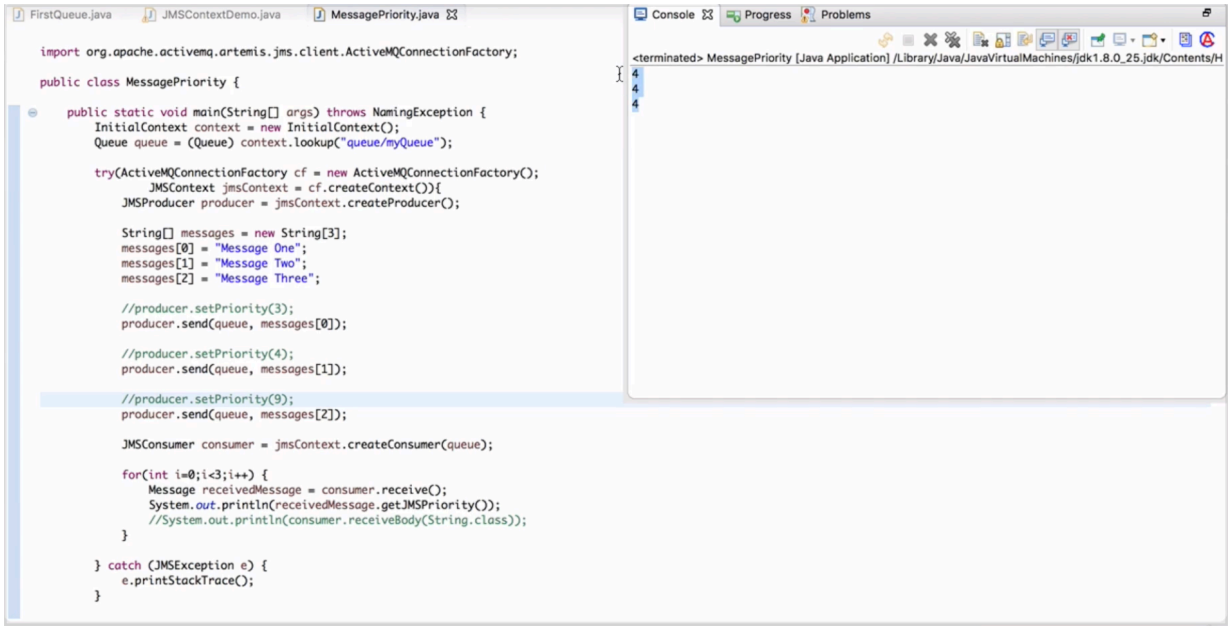
Application specific properties:
Mostly we will use application specific properties.
setXXXProperty
getXXXProperty
XXX can be Integer/String etc.

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:



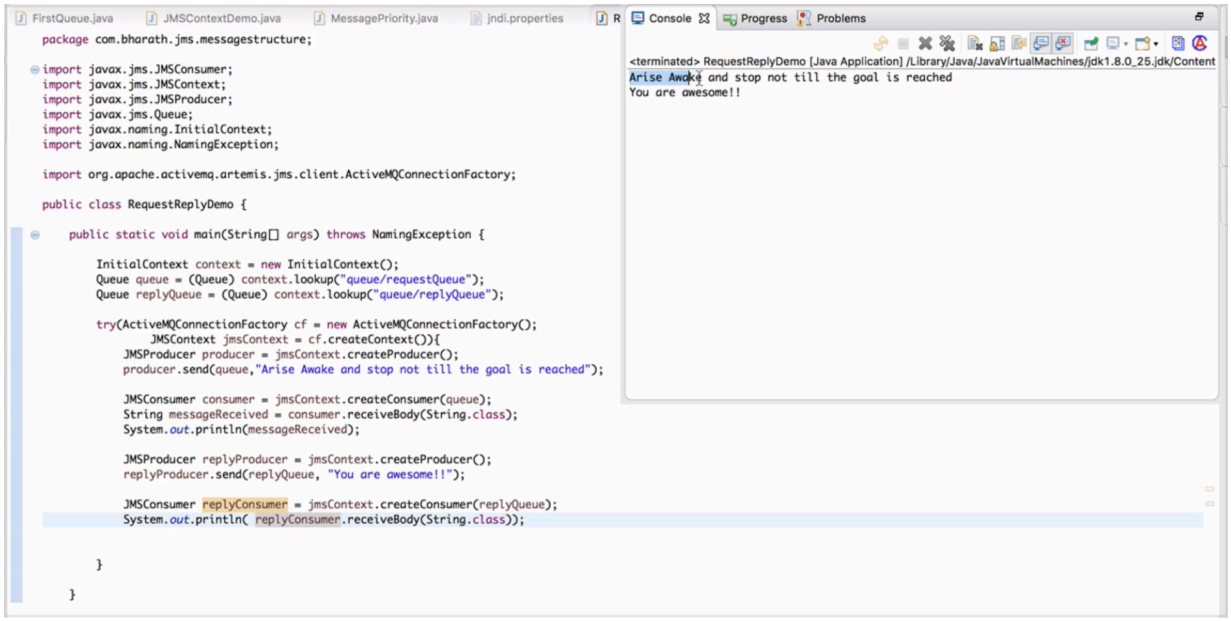
Default priority: 4



Request reply messaging:
request-reply:
replyTo
messageId
correlationID

FirstQueue.java | JMSContextDemo.java | MessagePriority.java | jndi.properties

```
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
```

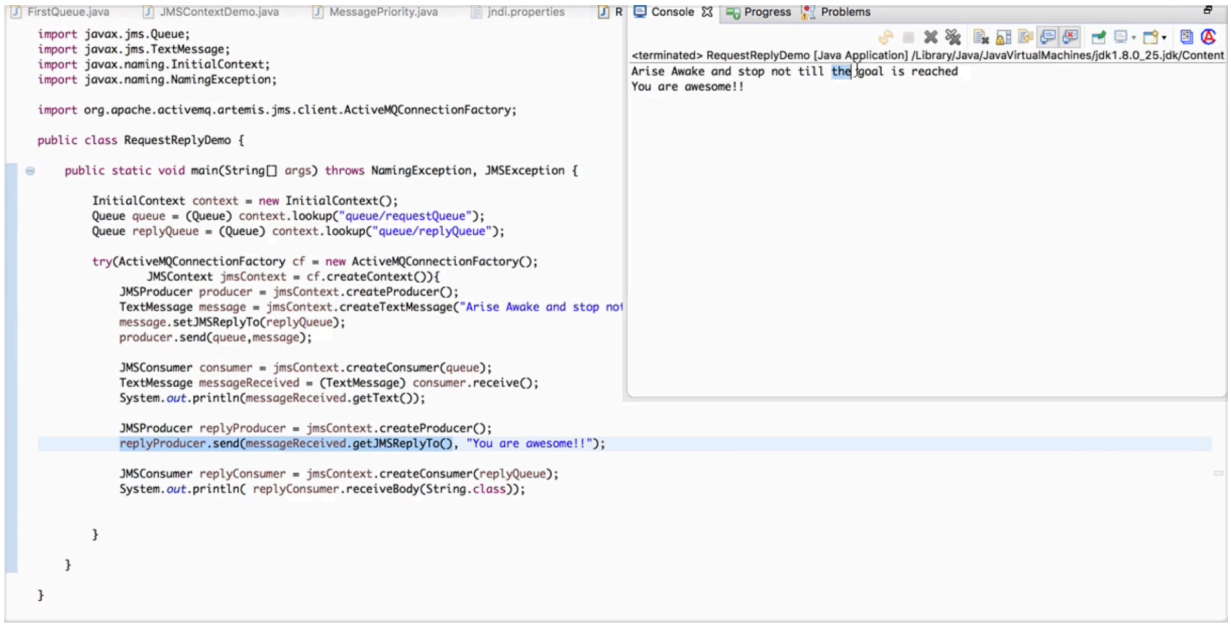


```
package com.bharath.jms.messagestructure;

import javax.jms.JMSConsumer;
import javax.jms.JMSContext;
import javax.jms.JMSProducer;
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();
                JMSContext jmsContext = cf.createContext()){
            JMSProducer producer = jmsContext.createProducer();
            producer.send(queue,"Arise Awake and stop not till the goal is reached");

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

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

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


        }

    }
```

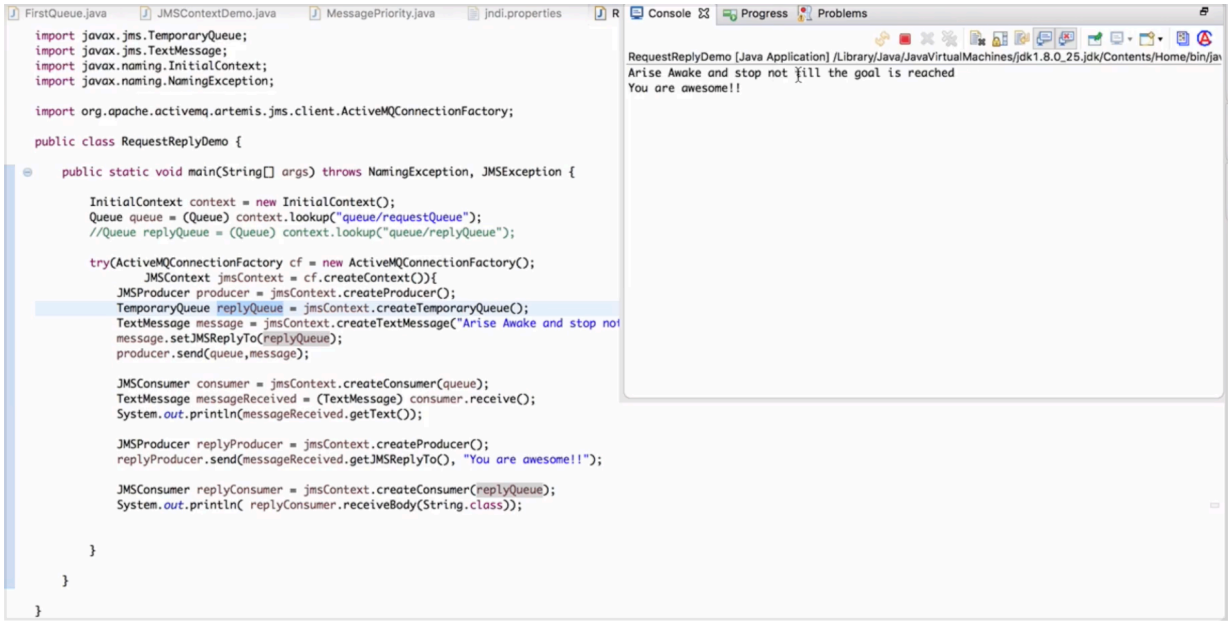
Console | Progress | Problems

```
<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!!
```

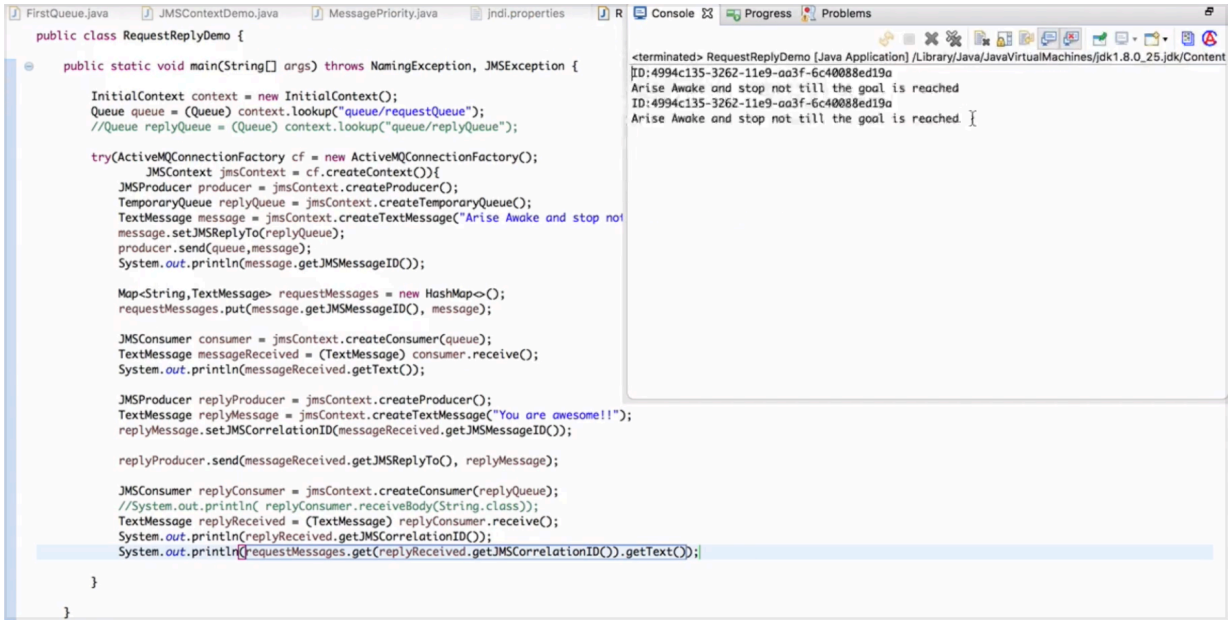
Use replyTo JMS Header:
We can set it in JMSProducer and Message header object. Better convention is to set it in Message header object.
setJMSReplyTo
getJMSReplyTo



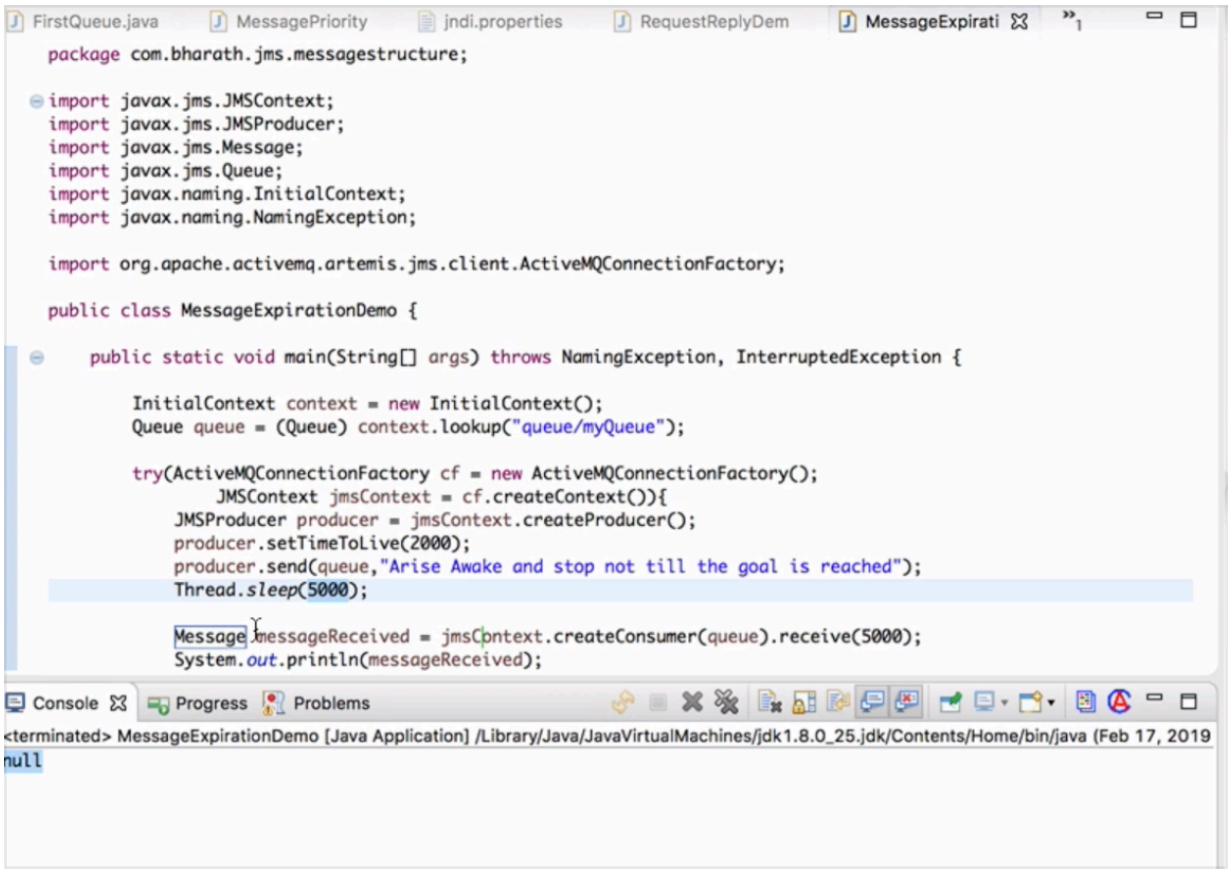
When to use replyTo header:
TemporaryQueue replyQueue = jmsContext.createTemporaryQueue();



Use messageId and correlationID headers:
Use these to link request and response.



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



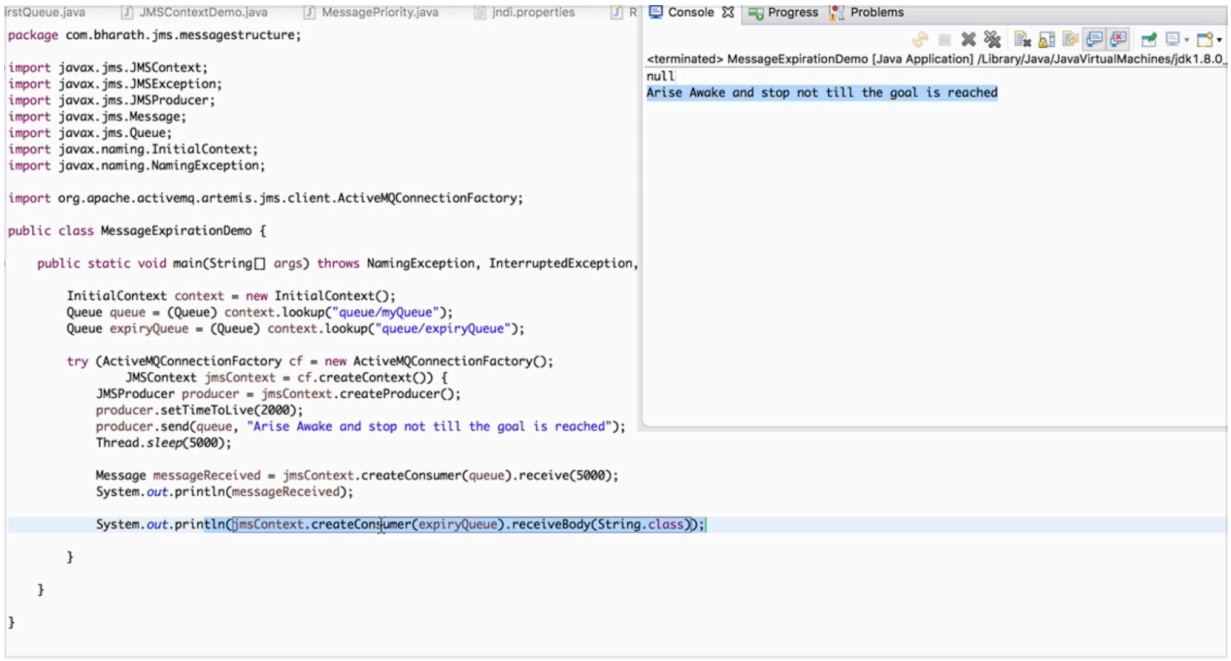
Expired messages are not lost. They just get moved to ExpiryQueue which is

already configured in broker.xml.

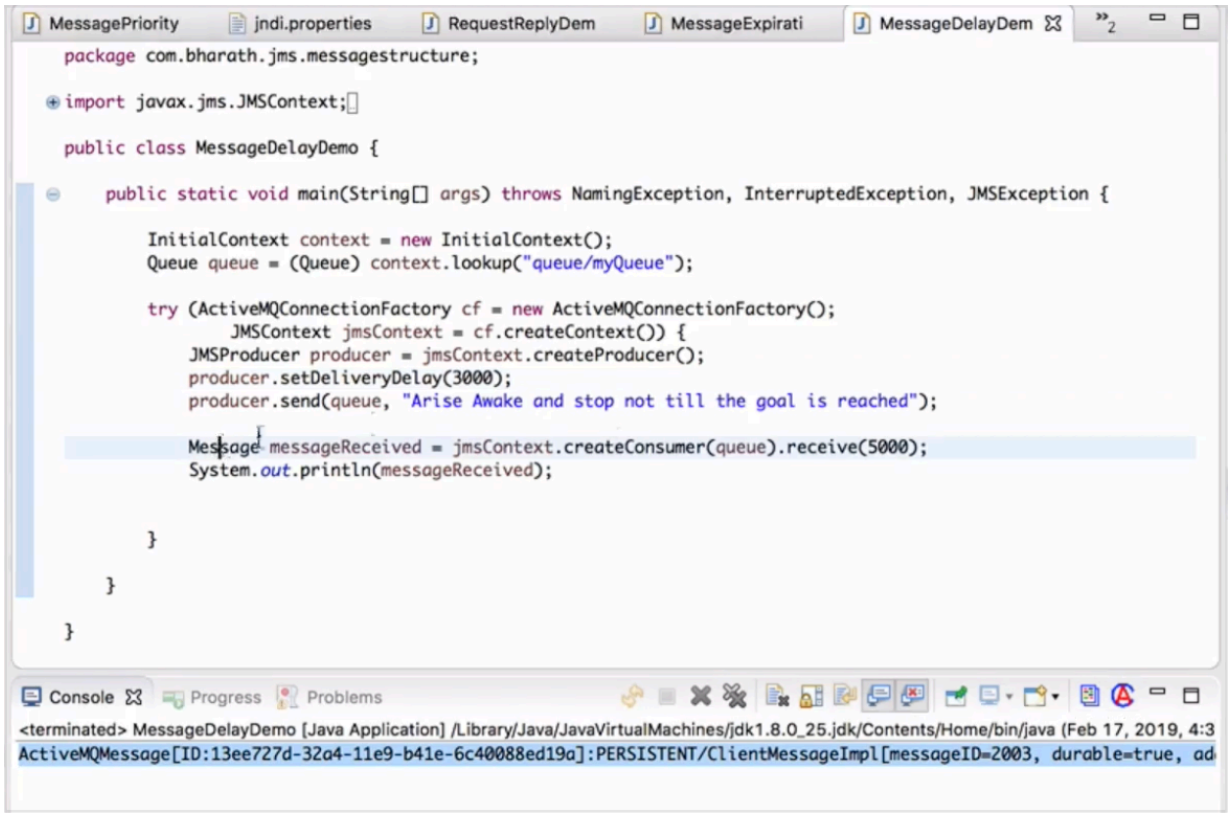
We can retrieve those by configuring ExpiryQueue in jndi.properties. Note that E is in uppercase.

queue.queue/expiryQueue = ExpiryQueue

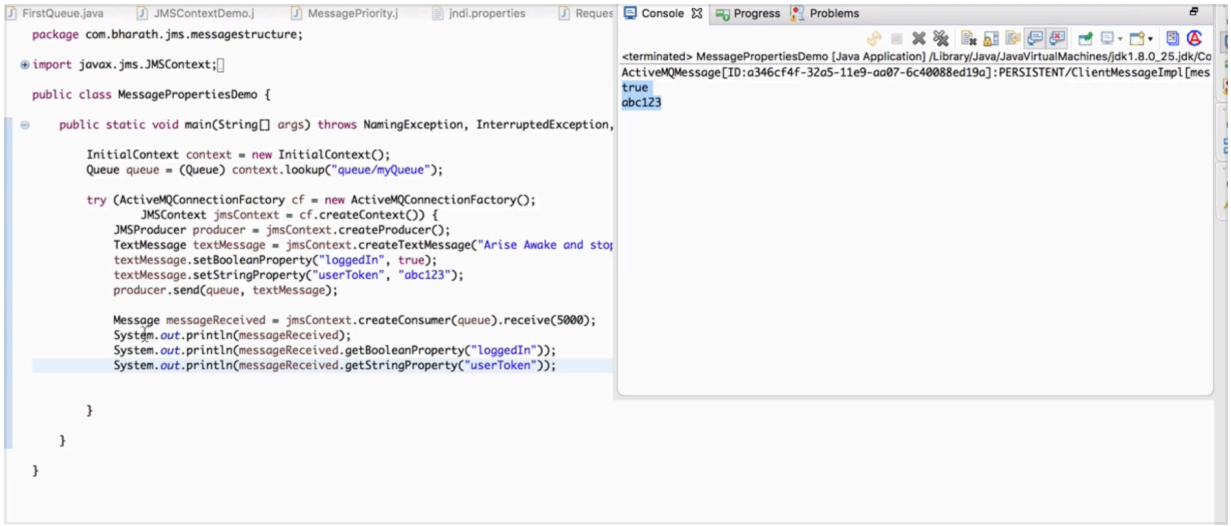
Queue expiryQueue = (Queue) initialContext.lookup(“queue/expiryQueue”);
System.out.println(jmsContext.createConsumer(expiryQueue).receiveBody(String.class));



Delay the message delivery:
jmsProducer.setDeliveryDelay(n milliseconds);



Add custom message properties:
message.setXXXProperty("", ...);
message.setBooleanProperty("loggedIn", true);
message.setStringProperty("userToken", "user123");

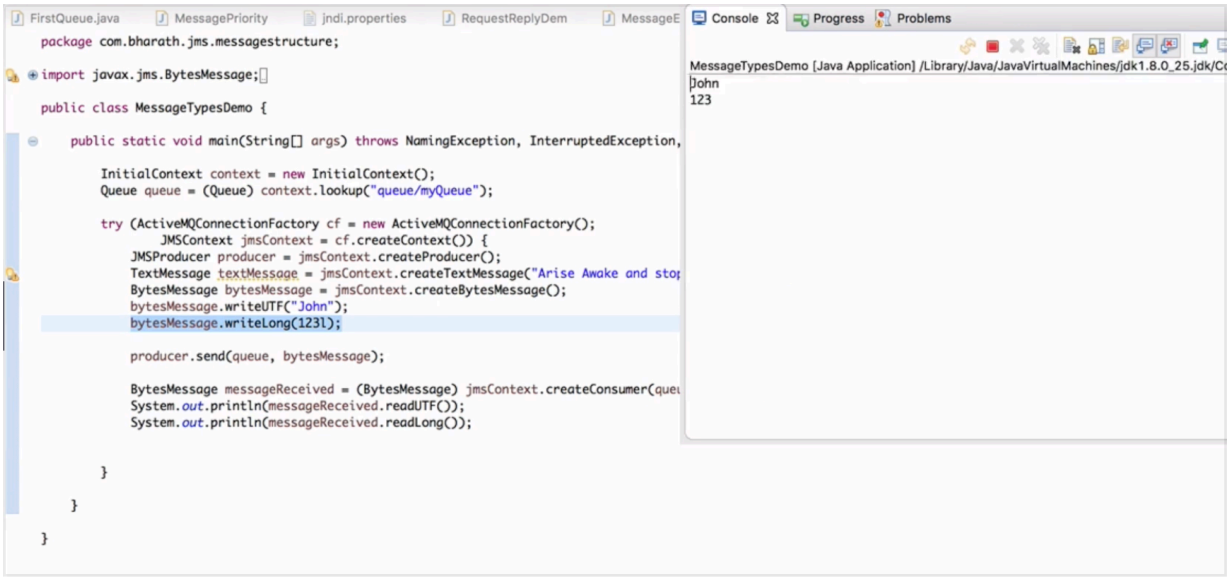


Message types:

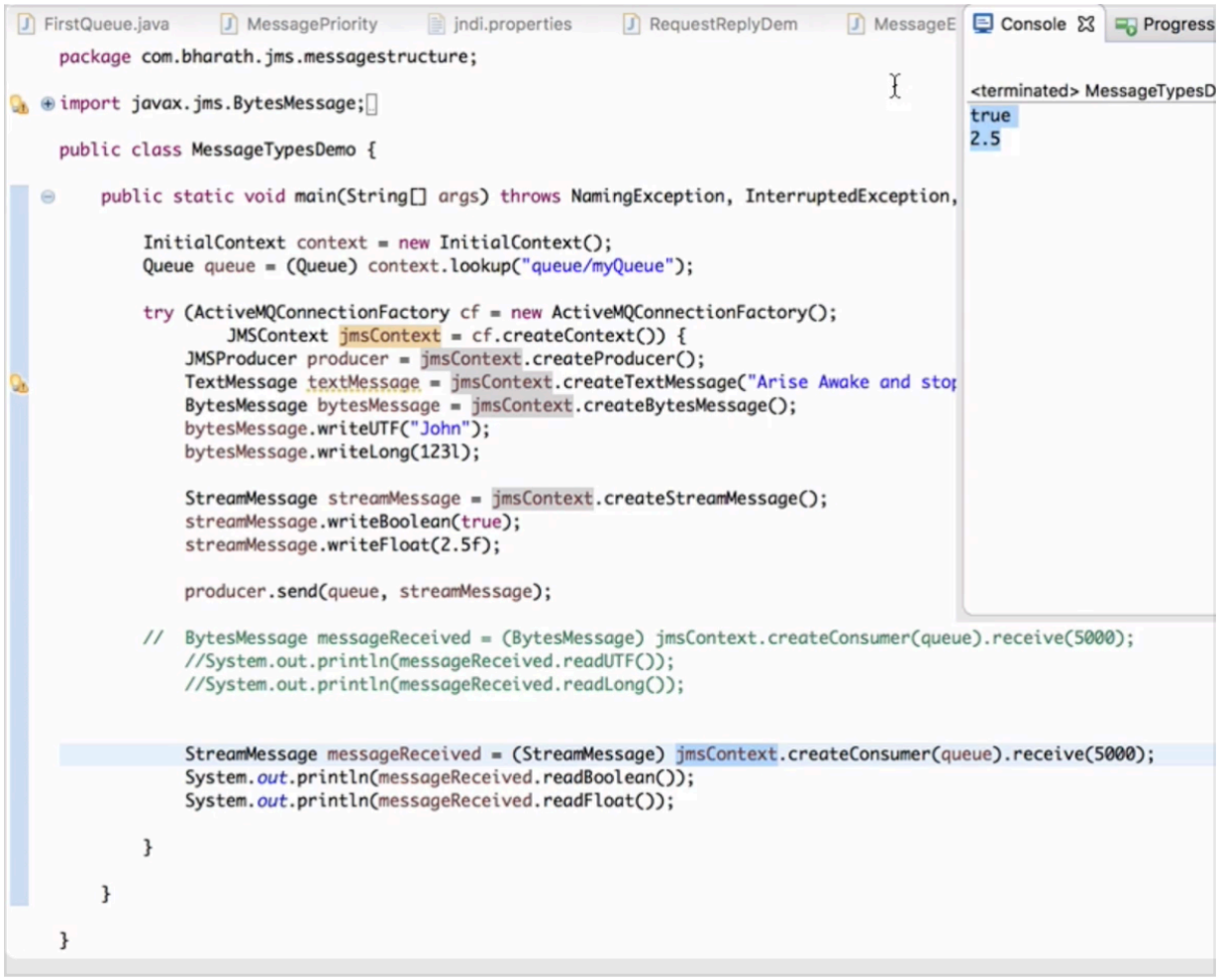
1. TextMessage
2. ByteMessage

3. ObjectMessage: Should be serialized
4. StreamMessage
5. MapMessage

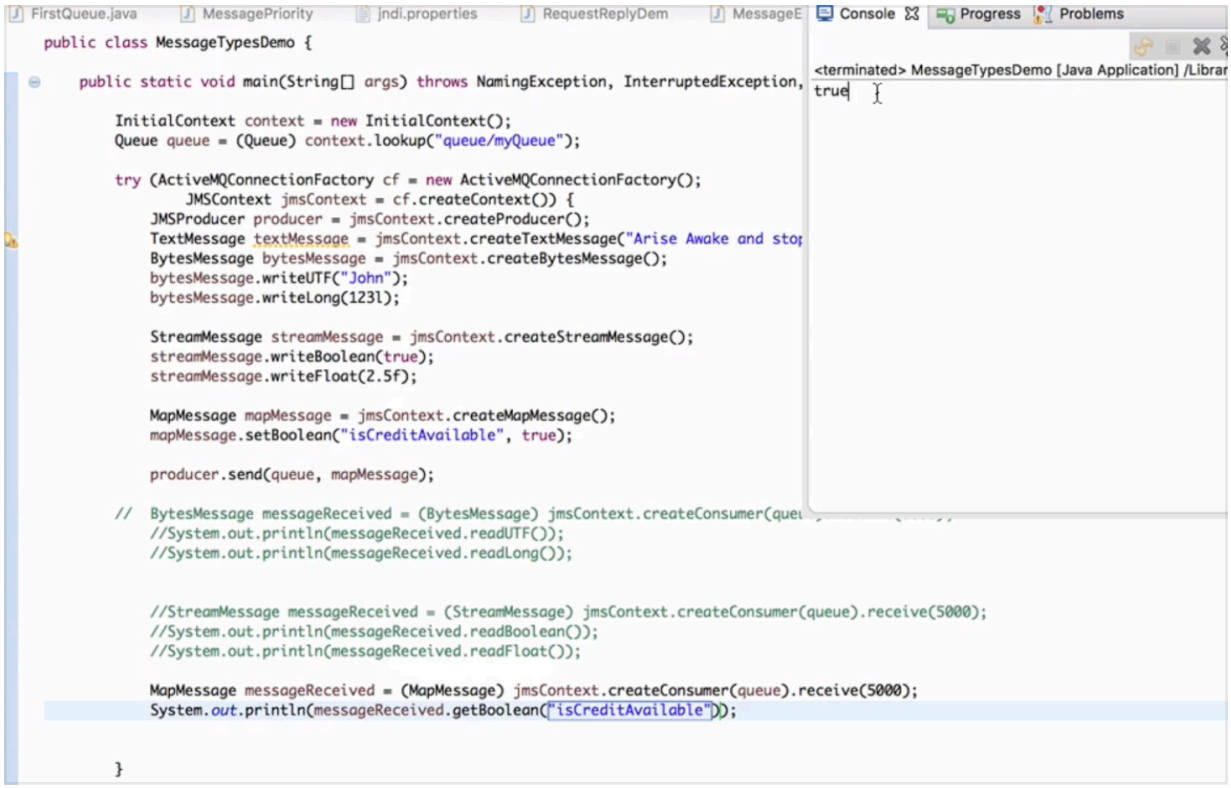
Message types in action:
ByteMessage: Generally methods start with write and read



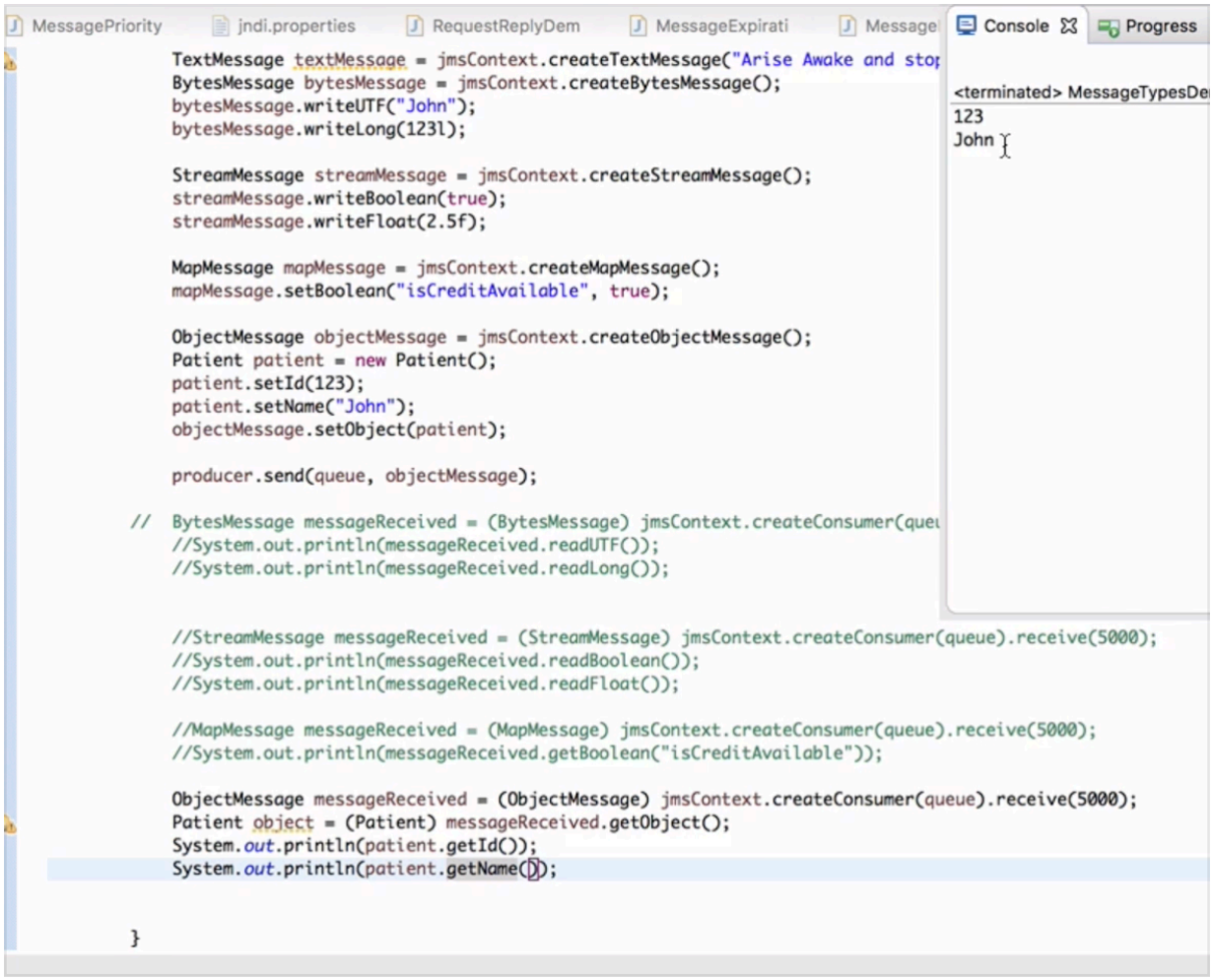
StreamMessage: Generally methods start with write and read



MapMessage: Generally methods start with set and get



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



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

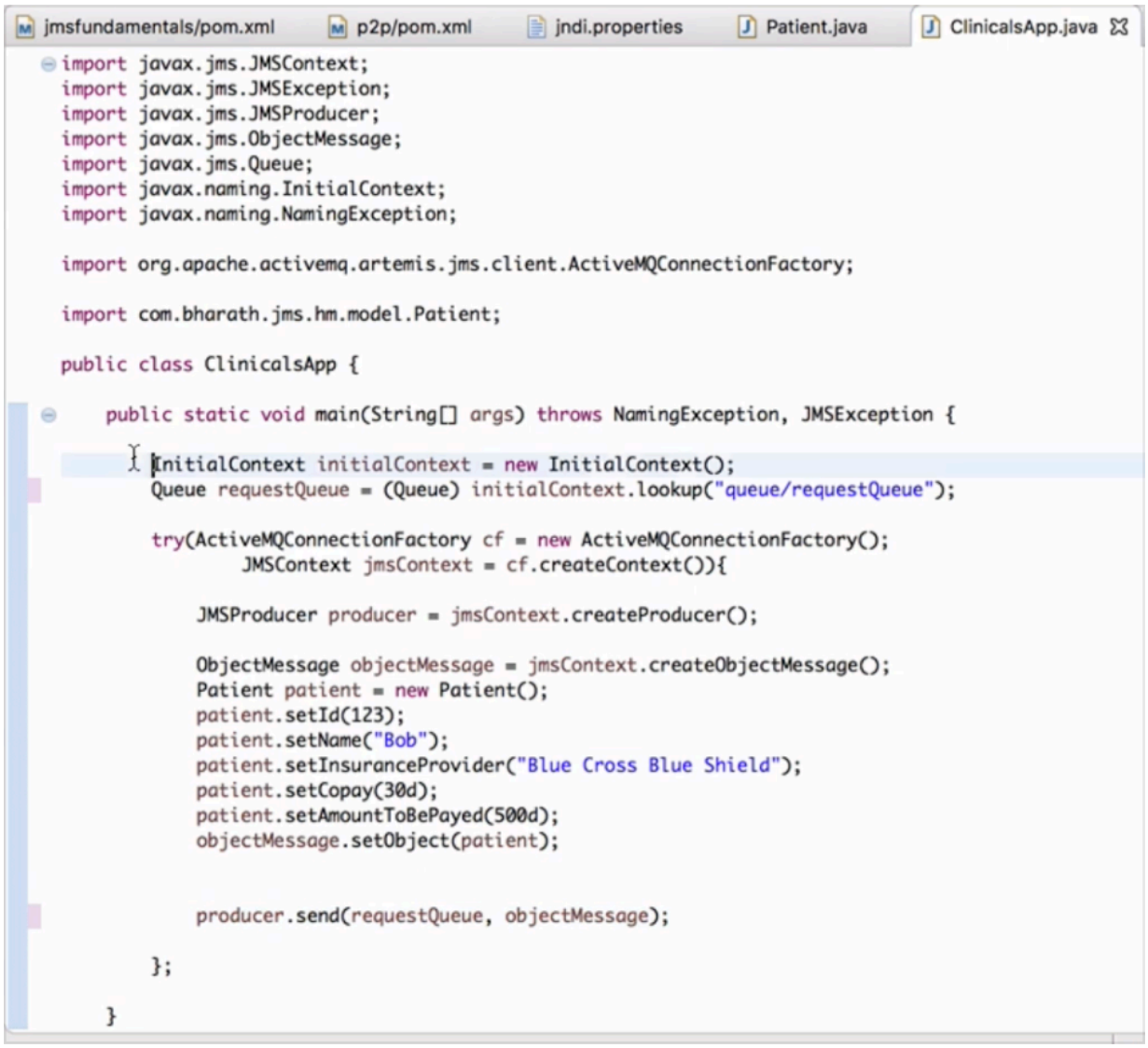
2. src/main/resources/jndi.properties:

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

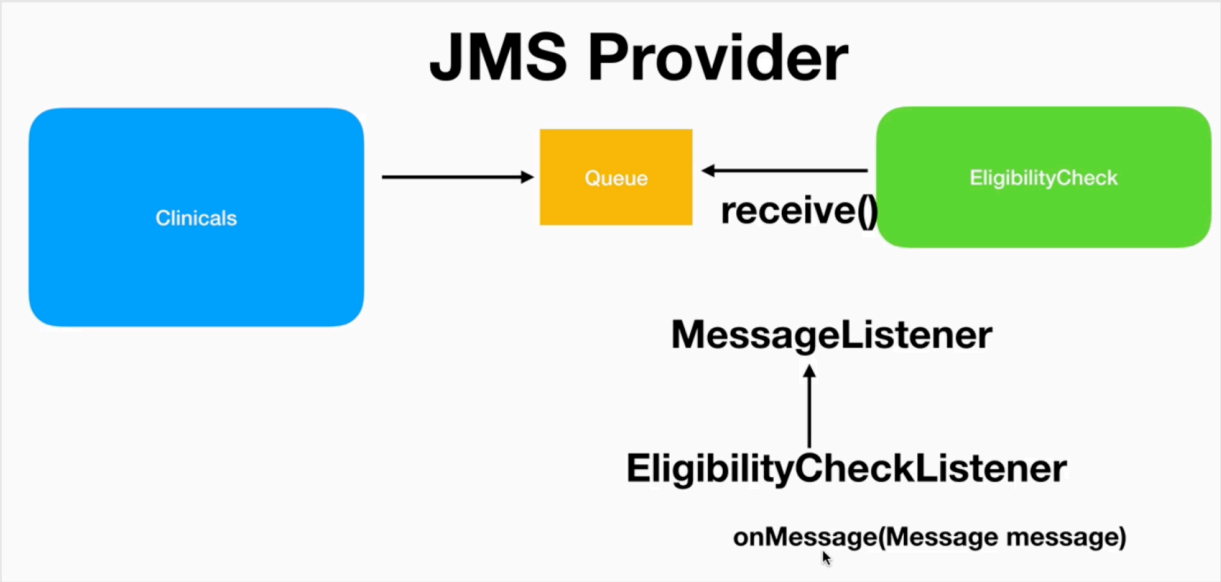
3. public class Patient implements Serializable.

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

ClinicalsApp.java: Provider App



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.



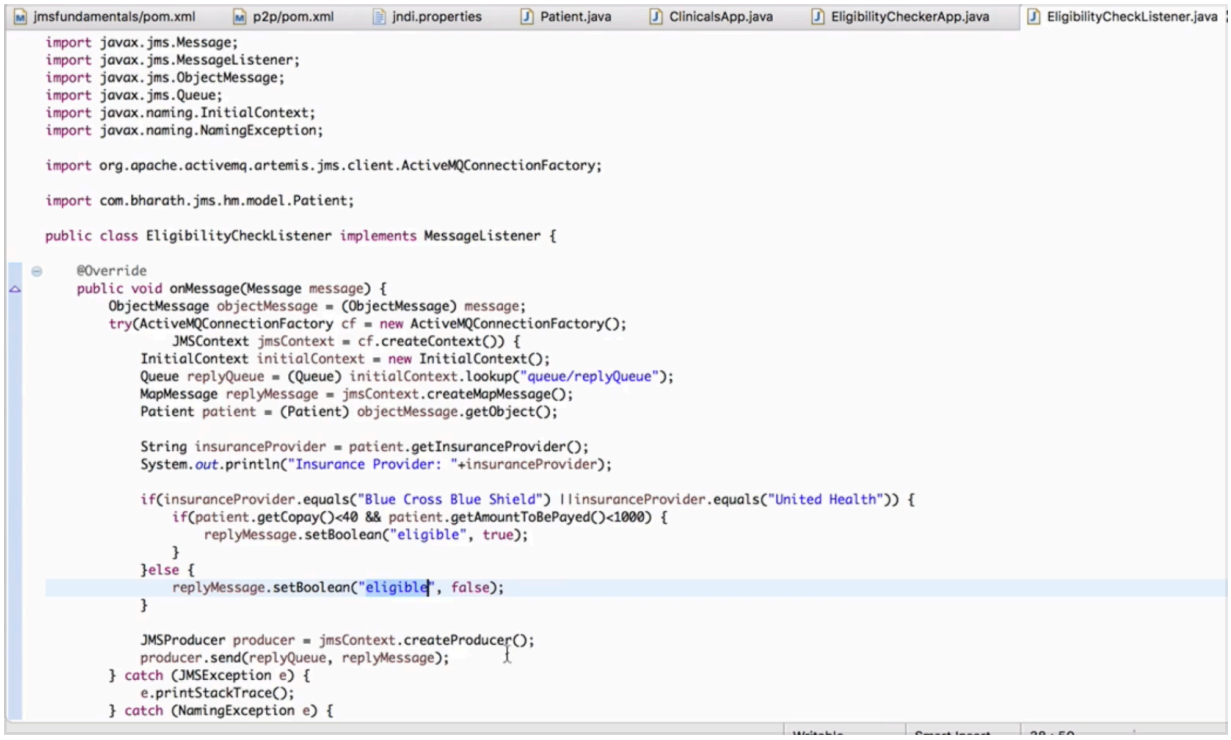
EligibilityCheckerApp.java: Consumer App

```java
package com.bharath.jms.hm.eligibilitycheck;

import javax.jms.JMSConsumer;
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());

        };

    }

}
```

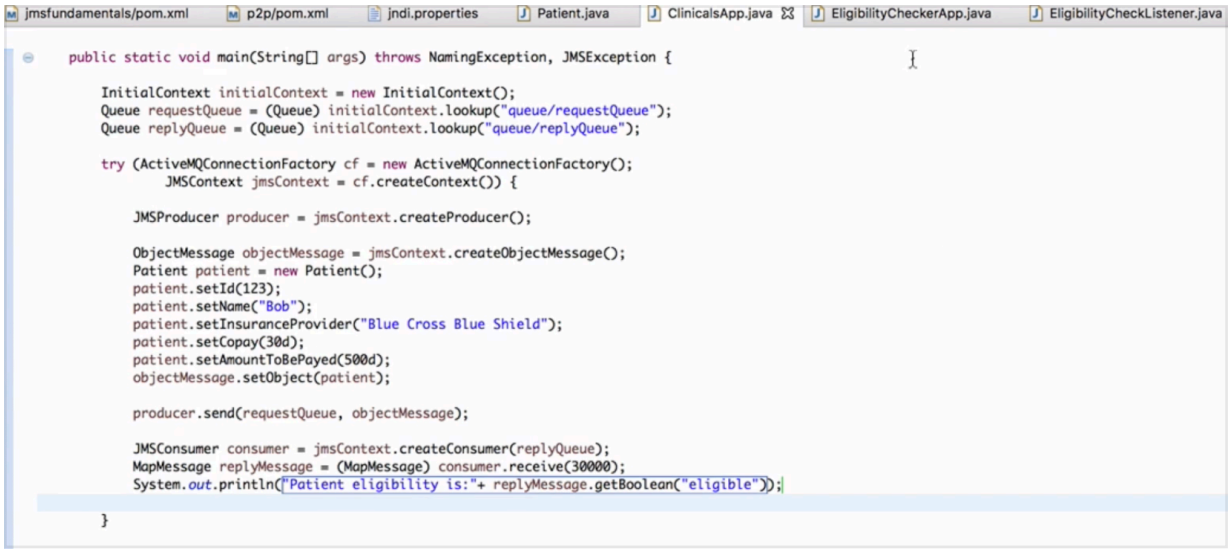
Put Thread.sleep(30000); in the past to keep this standalone consumer app running for testing if we need to run and test both provider and consumer app together without server.

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

import javax.jms.JMSException;

public class EligibilityCheckListener implements MessageListener {

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

}
```

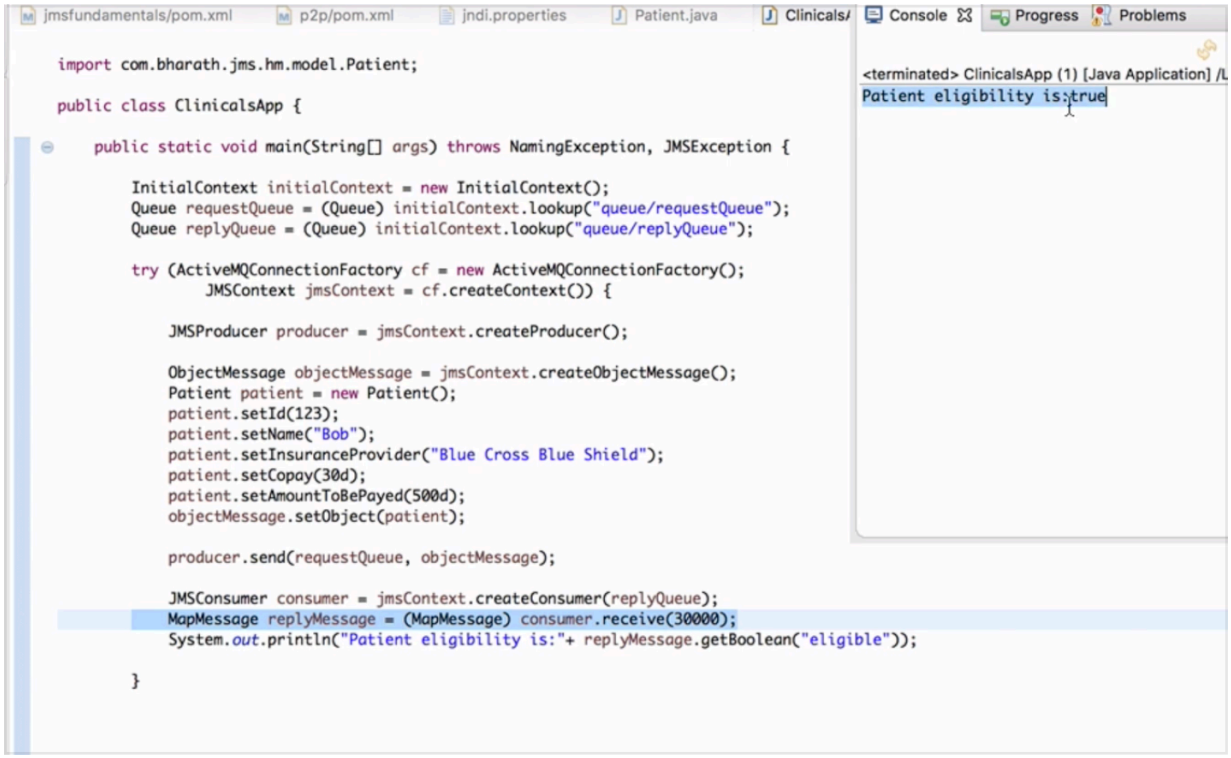
Check eligibility and send reply:



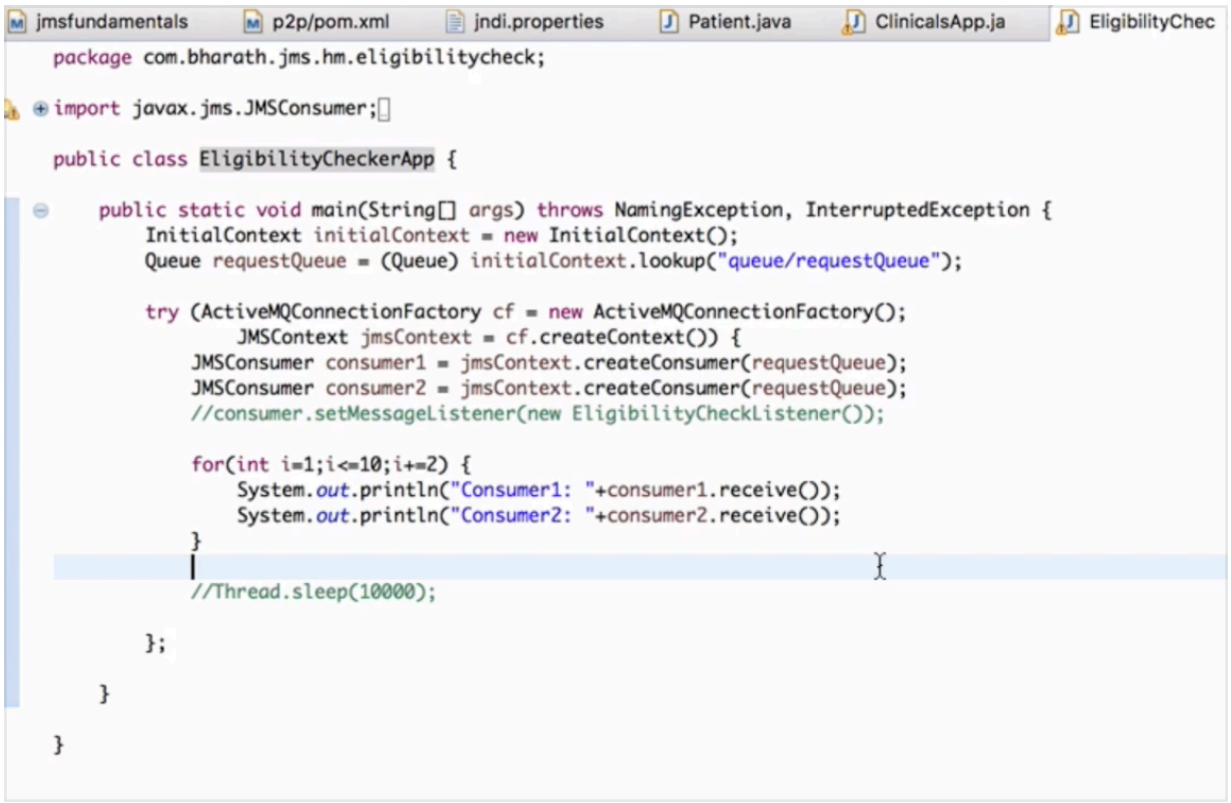
Process the response:



Testing:



Load balancing:
Load will be distributed among consumers.



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;
}
```

pubsub/pom.xml | jmsfundamentals/pom.xml | jndi.properties | HRApp.java | ClinicalsApp.java | Employee.java

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

import javax.jms.JMSContext;
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();
                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");
            jmsContext.createProducer().send(topic, employee);
            System.out.println("Message Sent");

        }

    }

}
```

Create the subscribers:
Similar codes in 3 subscribers PayrollApp, SecurityApp and WellnessApp

Package Explorer

- jmsfundamentals
  - src/main/java
  - src/test/java
  - src/main/resources
    - jndi.properties
  - Maven Dependencies
  - JRE System Library [Java SE 8 [1
  - src
  - target
  - pom.xml
- p2p
- pubsub
  - src/main/java
    - com.bharath.jms.payroll
      - PayrollApp.java
    - com.bharath.jms.security
      - SecurityApp.java
    - com.bharath.jms.welness
      - WelnessApp.java
    - com.bharath.jsm.hr
      - Employee.java
      - HRApp.java
  - src/test/java
  - src/main/resources
    - jndi.properties
  - JRE System Library [JavaSE-1.7]

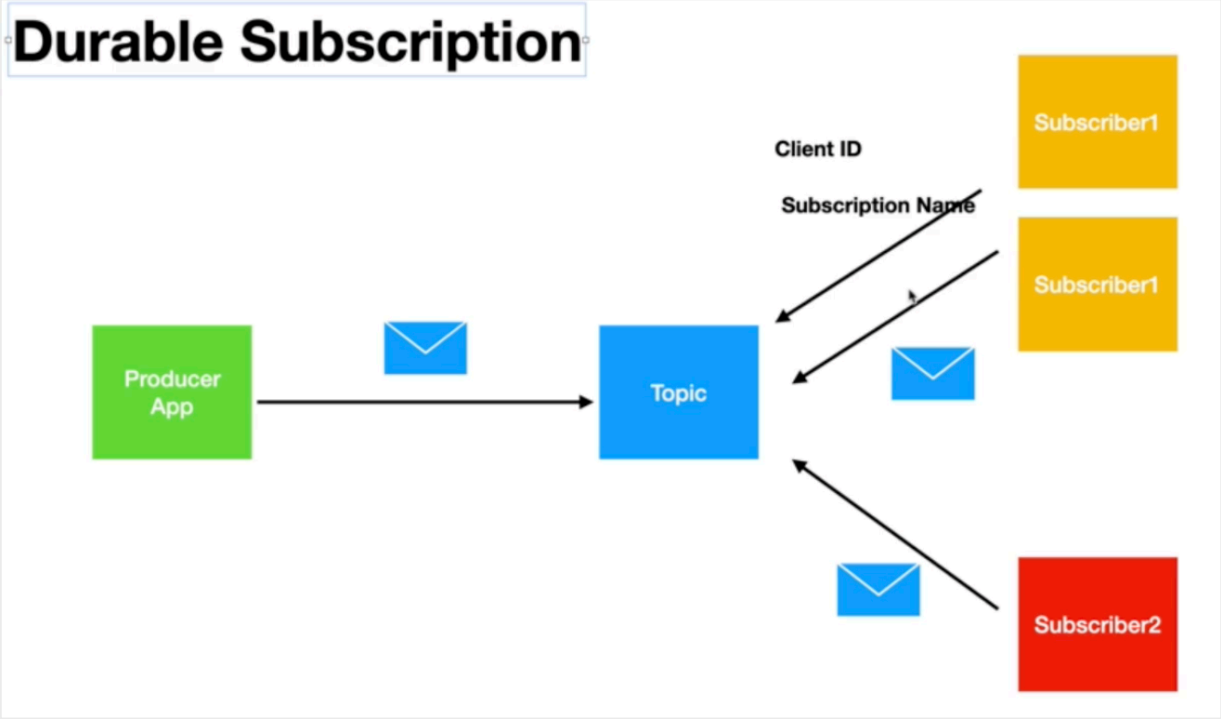
HRApp.java | ClinicalsApp.ja | Employee.java | SecurityApp.jav | PayrollApp.java

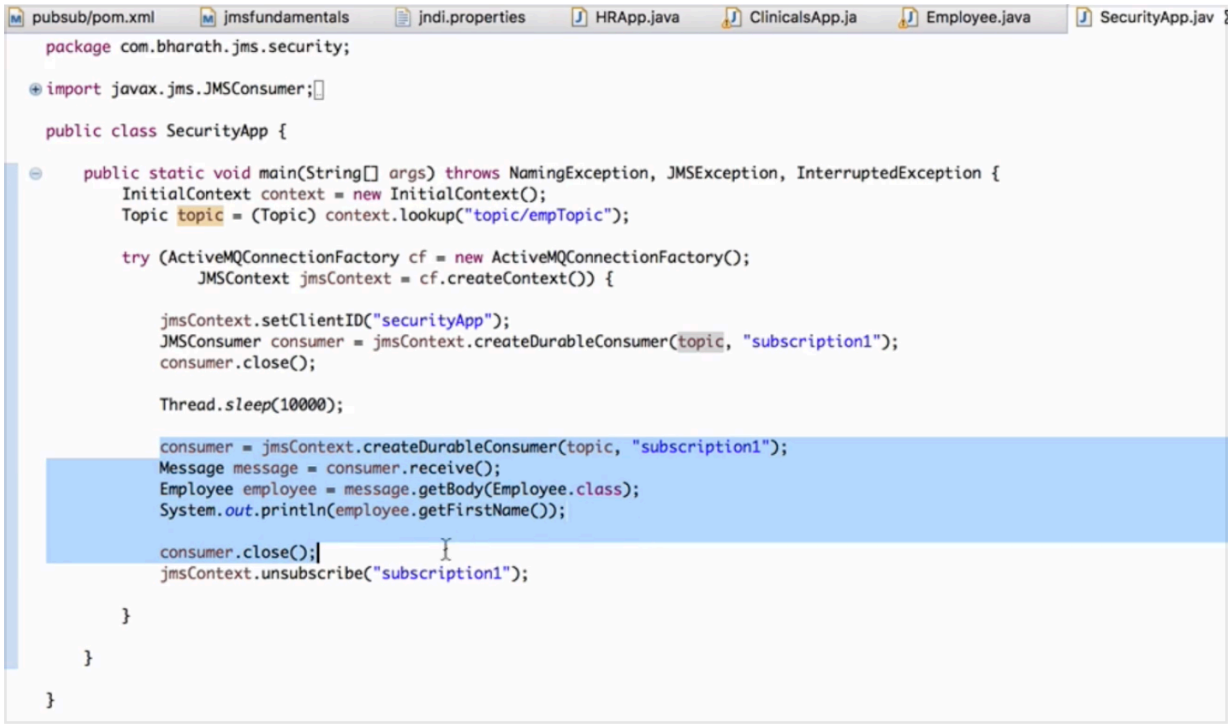
```java
package com.bharath.jms.payroll;

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

        }

    }

}
```

Console | Progress | Problems

<terminated> EligibilityCheckerApp [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Hom

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.

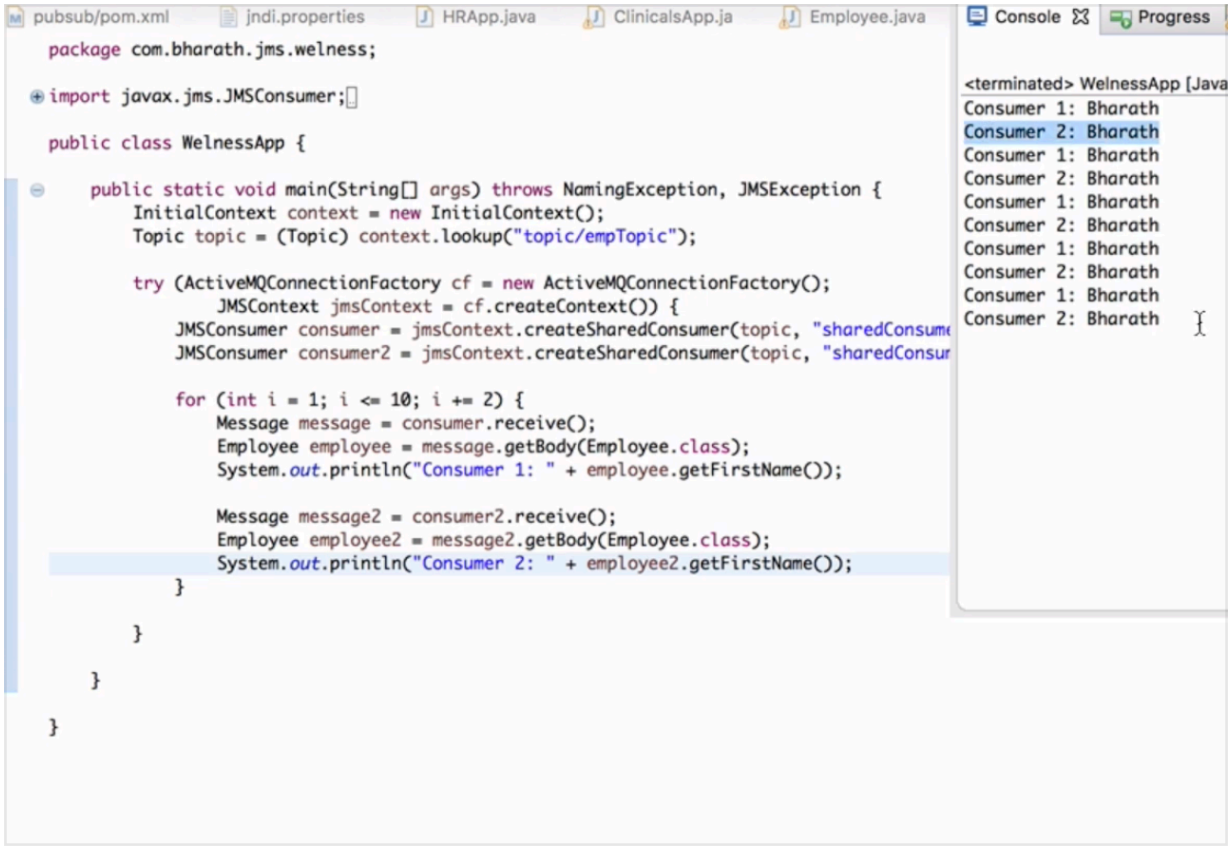




Shared subscription in action:
Create shared consumer to not consume same message by multiple subscribers of same topic. Only one subscriber will take it.

pubsub/pom.xml | jndi.properties | HRApp.java | ClinicalsApp.ja | Employee.java

```java
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");

        }

    }

}
```



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:

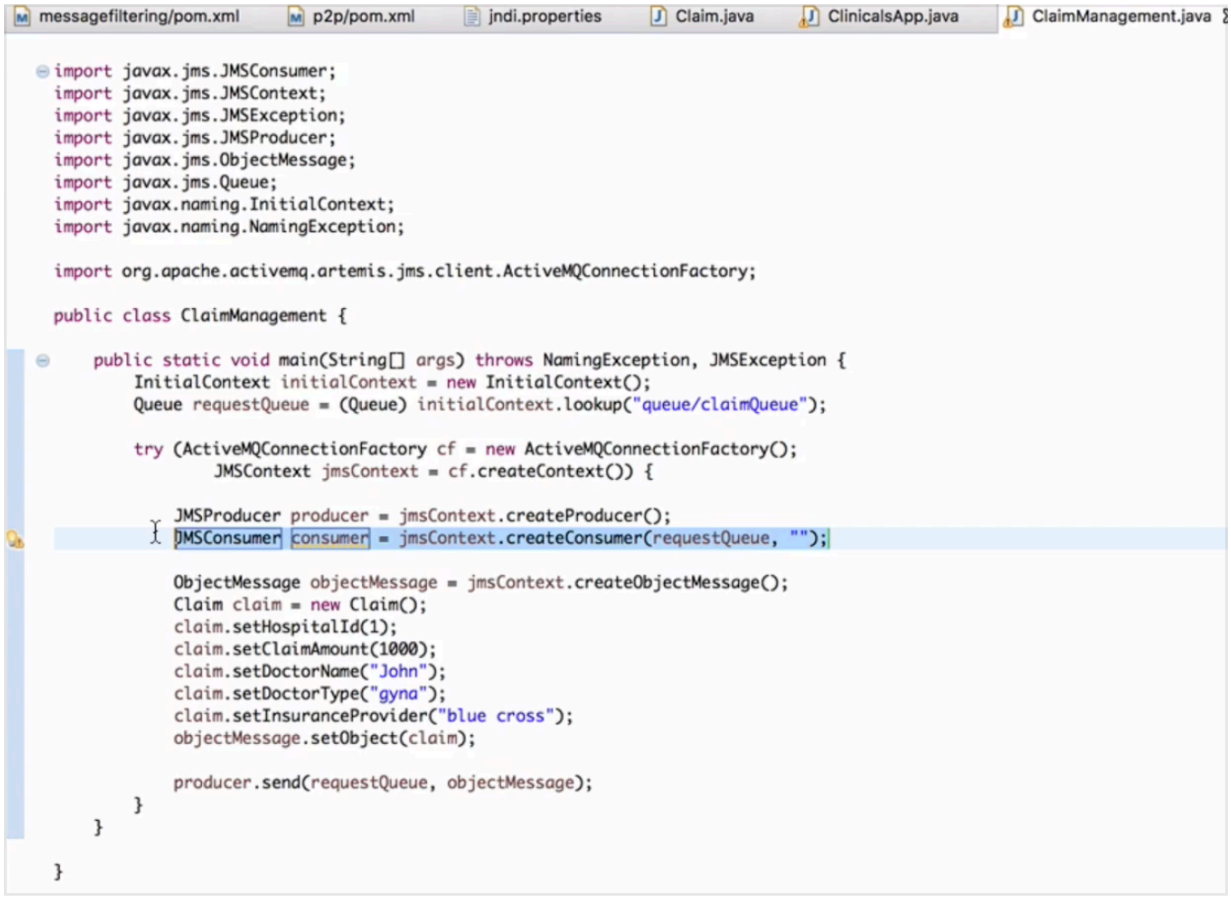
pom

jndi.properties:
claimQueue

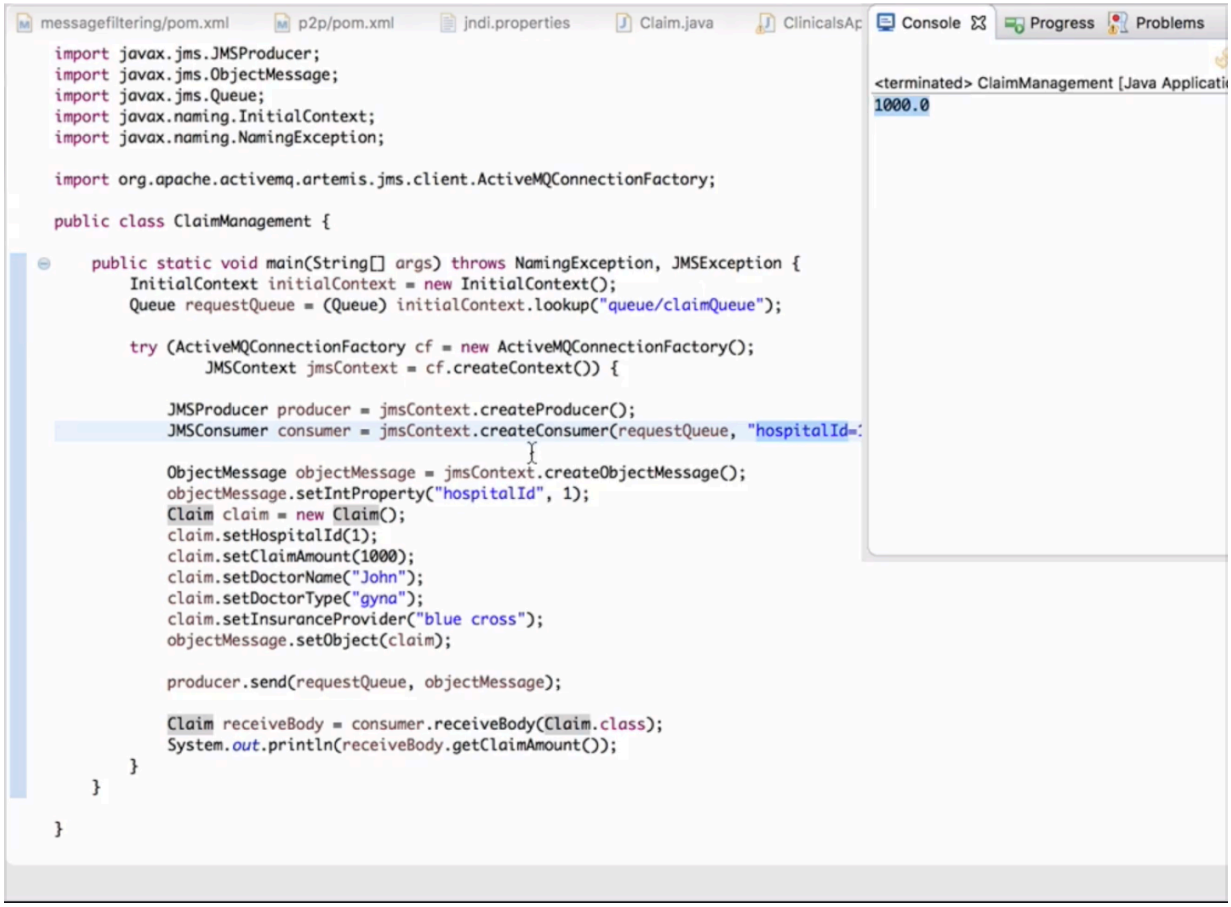
Model:

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

ClaimManagement:



Filters in action:



Use other operators:

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

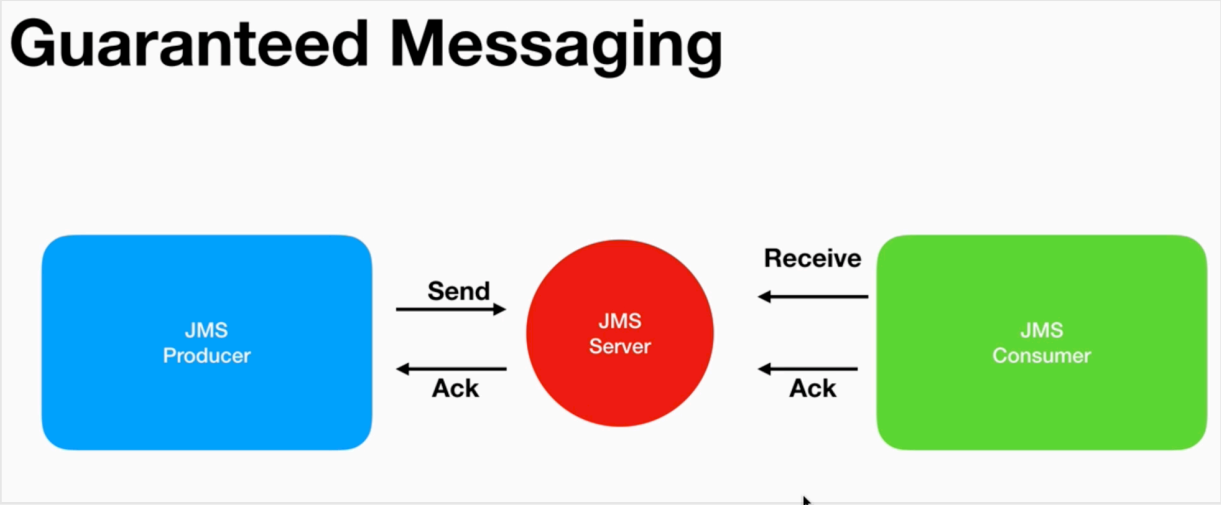
```java
import javax.jms.JMSProducer;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

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();
                JMSContext jmsContext = cf.createContext()) {

            JMSProducer producer = jmsContext.createProducer();
            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

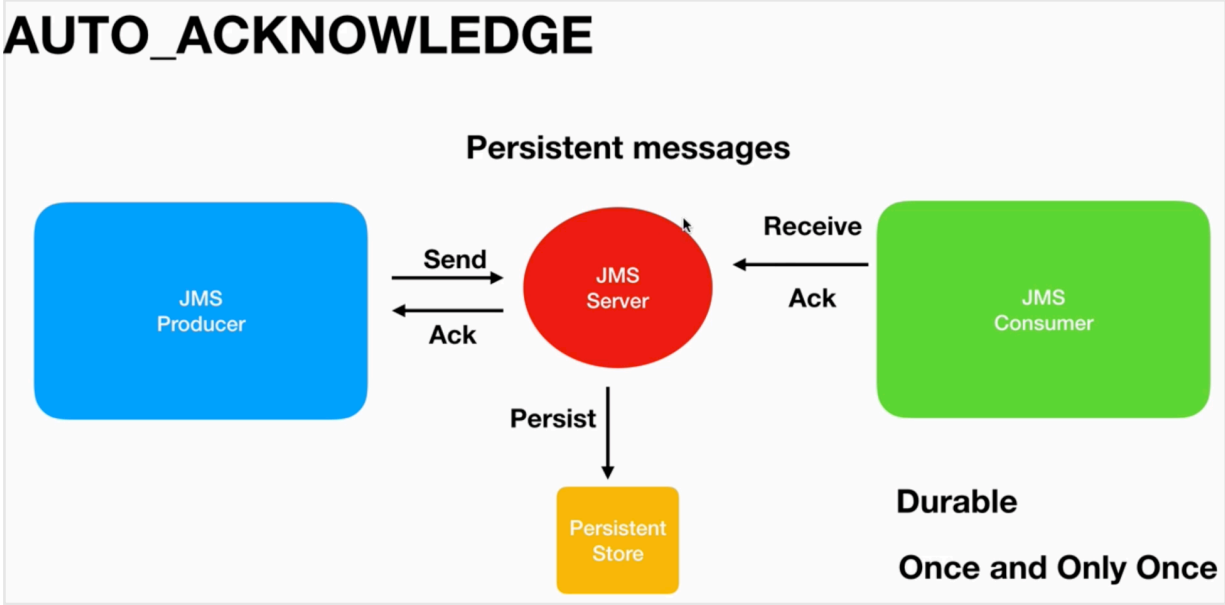
```java
import javax.jms.JMSProducer;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.naming.InitialContext;
import javax.naming.NamingException;

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();
                JMSContext jmsContext = cf.createContext()) {

            JMSProducer producer = jmsContext.createProducer();
            JMSConsumer 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);

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

Guaranteed messaging:



Message acknowledgements:
AUTO_ACKNOWLEDGE
CLIENT_ACKNOWLEDGE
DUPS_OK_ACKNOWLEDGE

AUTO_ACKNOWLEDGE:



JMS Producer sends/publishes a synchronous message and then wait 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);

guaranteedmessaging/pom.xml | p2p/pom.xml | jndi.properties | MessageProducer.java | Mes

guaranteedmessaging/src/mai
bharath/jms/guaranteedmessa
MessageProducer.java

```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");

        }

    }

}
```

MessageConsumer:

guaranteedmessagi | p2p/pom.xml | jndi.properties | MessageProducer.ja | MessageConsumer.ja

```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()) {

            JMSConsumer consumer = jmsContext.createConsumer(requestQueue);
            TextMessage message = (TextMessage) consumer.receive();
            System.out.println(message.getText());

        }

    }

}
```

Console | Progress | Problems

<terminated> MessageConsumer [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java (Feb 24, 2019,

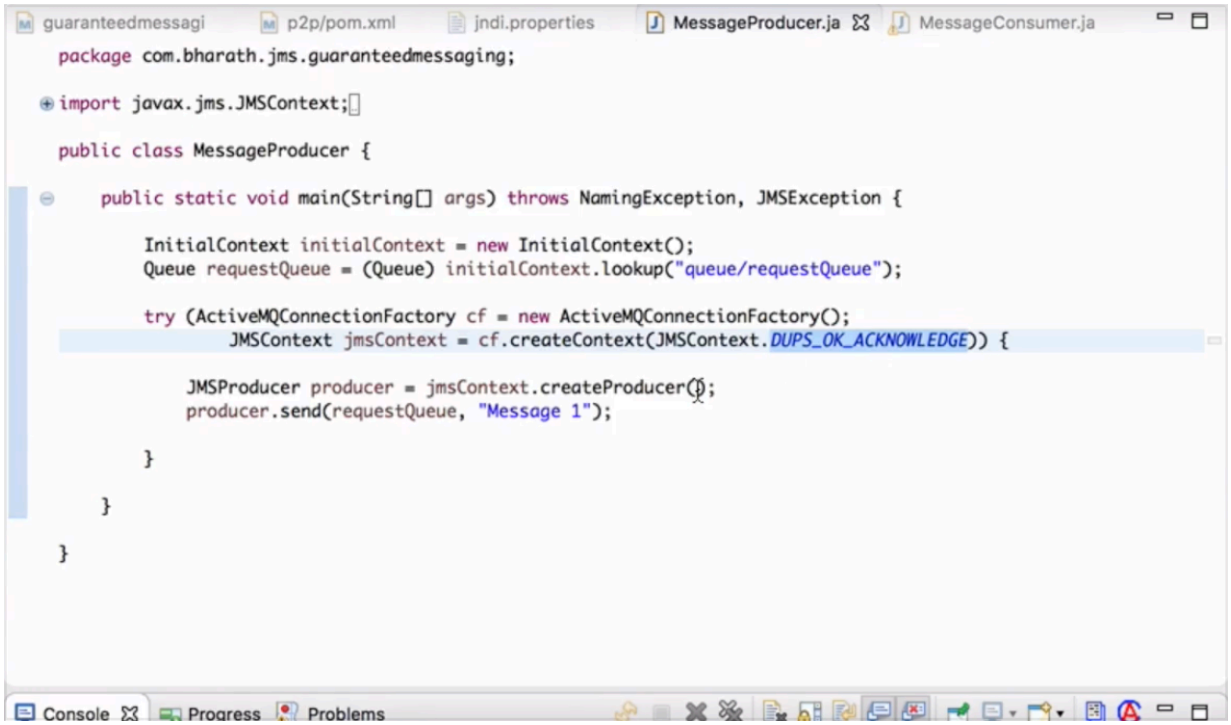
```
Message 1
```

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.

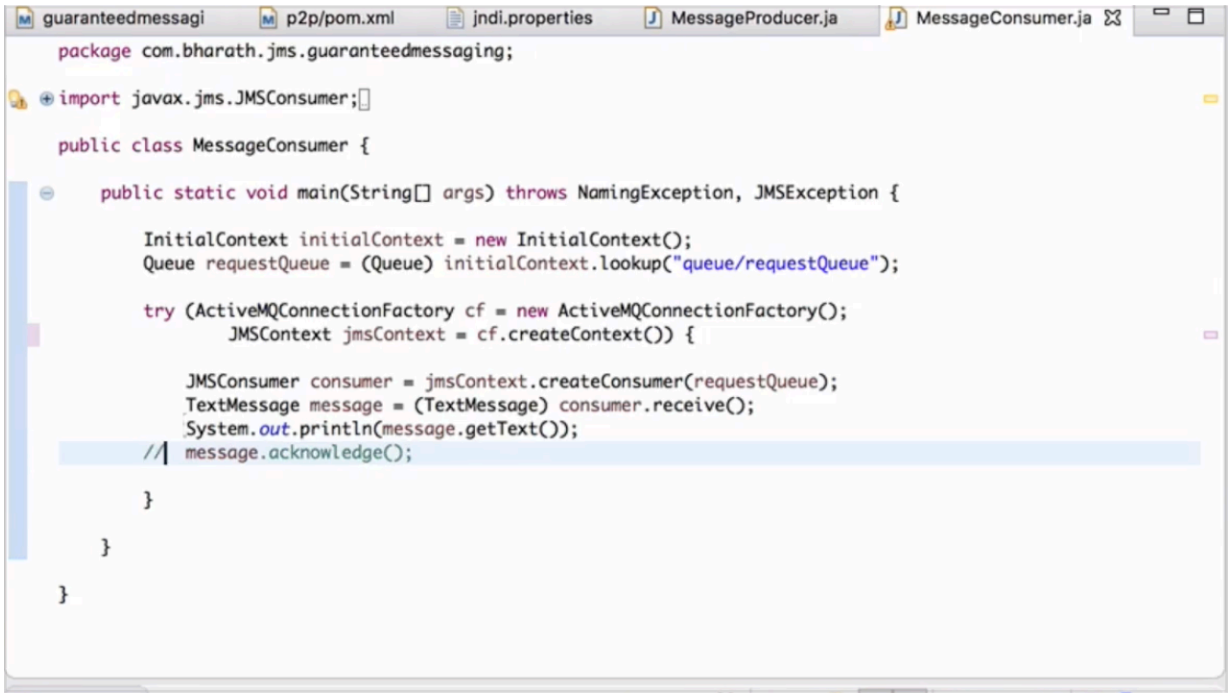
DUPS_OK_ACKNOWLEDGE:
JMS server can send a message to one consumer multiple times. For AUT_ACKNOWLEDGE there is an overhead by JMS server to make sure a message gets delivered to a consumer only once. In this DUPS_OK_ACKNOWLEDGE that overhead is not there because it does not need to worry about message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

Consumer tracks JMSMessageId to identify duplicate messages received.

Code for DUPS_OK_ACKNOWLEDGE:
Producer:



Consumer:



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.ja | MessageConsumer.ja

```
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

Consumer:

guaranteedmessagi | p2p/pom.xml | jndi.properties | MessageProducer.ja | MessageConsumer.ja

```
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:

Message 1

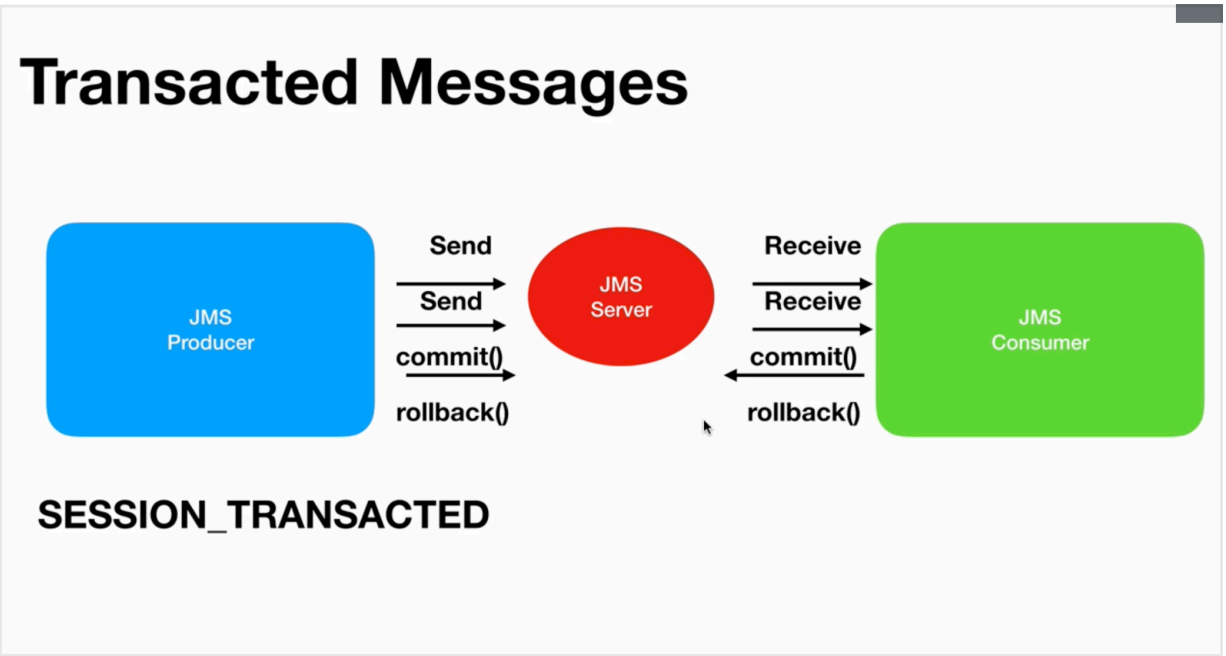
JMSContext jmsContext =

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 cliet 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



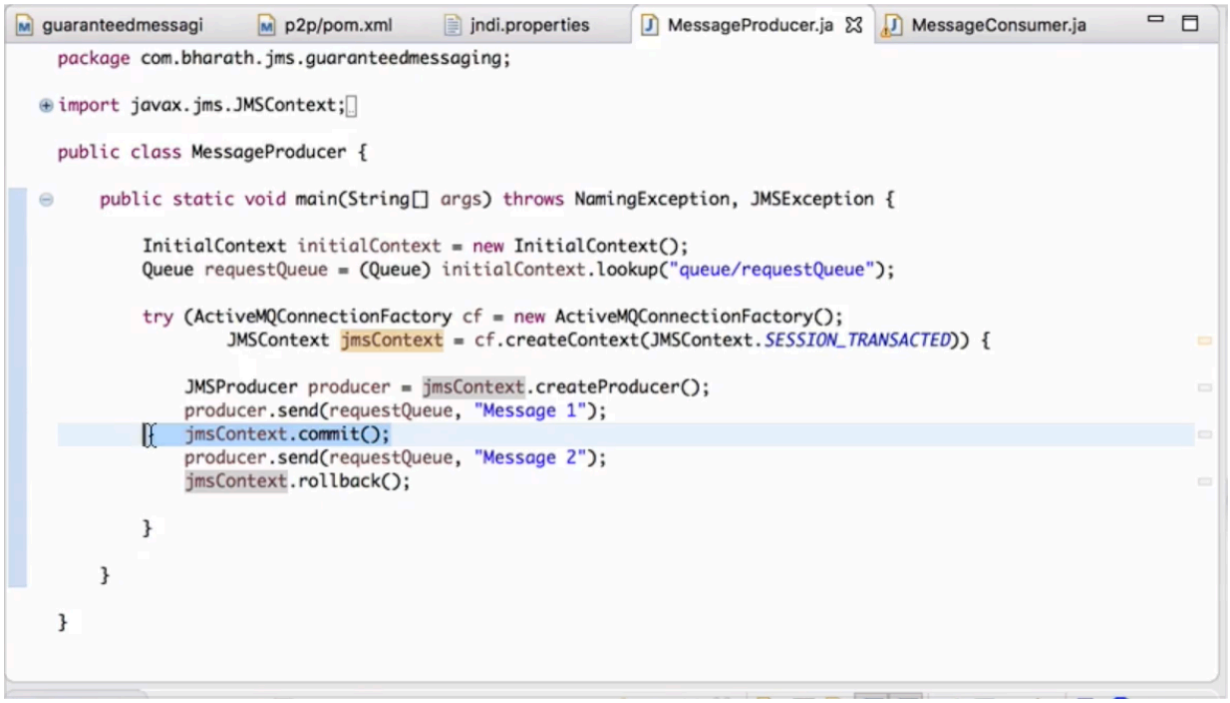
MessageProducer to JMS server:
MessageProducer keeps on sending messages to JMS server. JMS server keeps on adding messages to cache - not into queue/topic.
If MessageProducer sends rollback() then JMS server would have removed messages from cache.
Once MessageProducer sends commit(), then only all those 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:



Consumer:

guaranteedmessaging/pom.xml | p2p/pom.xml | jndi.properties | MessageProducer.java | MessageConsumer.java

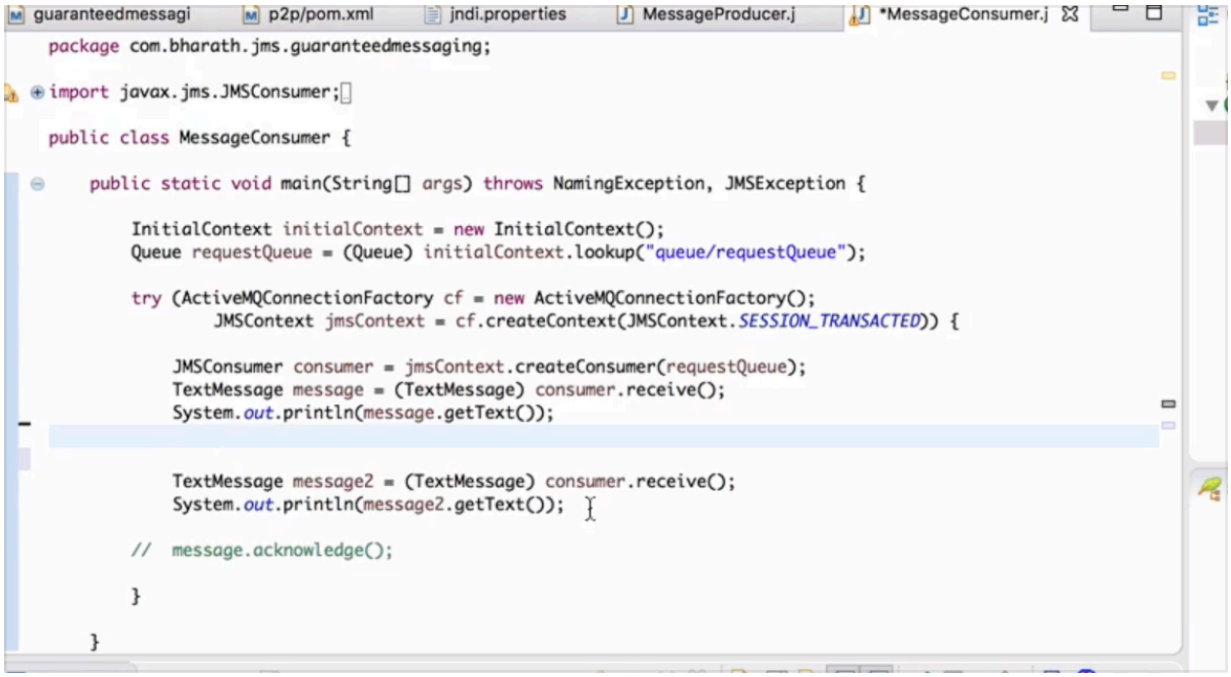
```
package com.bharath.jms.guaranteedmessaging;

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()) {

            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:

guaranteedmessagi | p2p/pom.xml | jndi.properties | MessageProducer.ja | MessageConsumer.ja

```
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.



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:

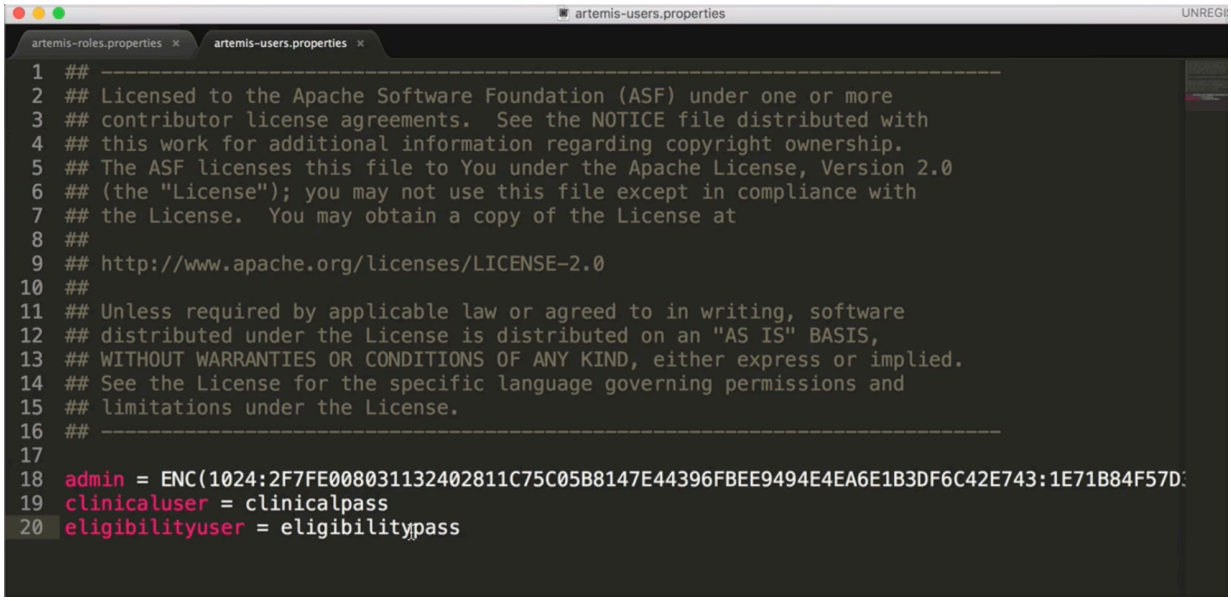
```
java.naming.factory.initial =
org.apache.activemq.artemis.jndi.ActiveMQInitialContextFactory
connectionFactory.ConnectionFactory=tcp://localhost
61616
```

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

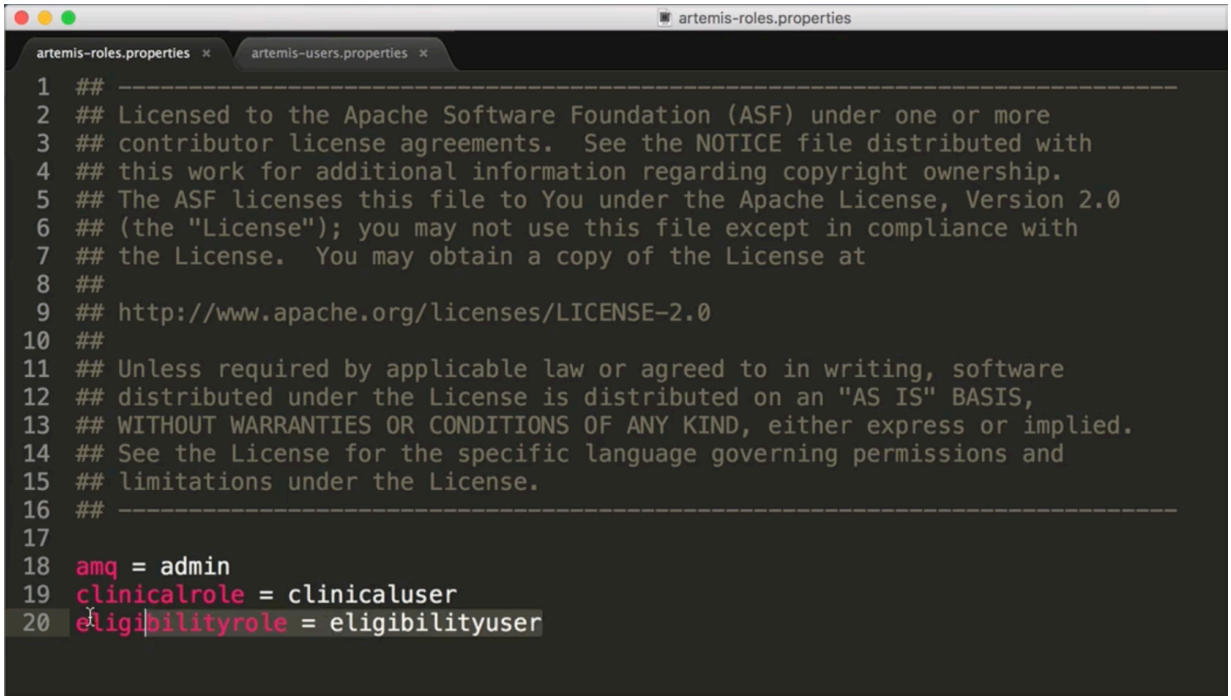
Producer:
JMSContext jmsContext = cf.createContext(“clinicalUser”, “clinicalPass”);

Consumer:
JMSContext jmsContext = cf.createContext(“eligibilityUser”, “eligibilityPass”);

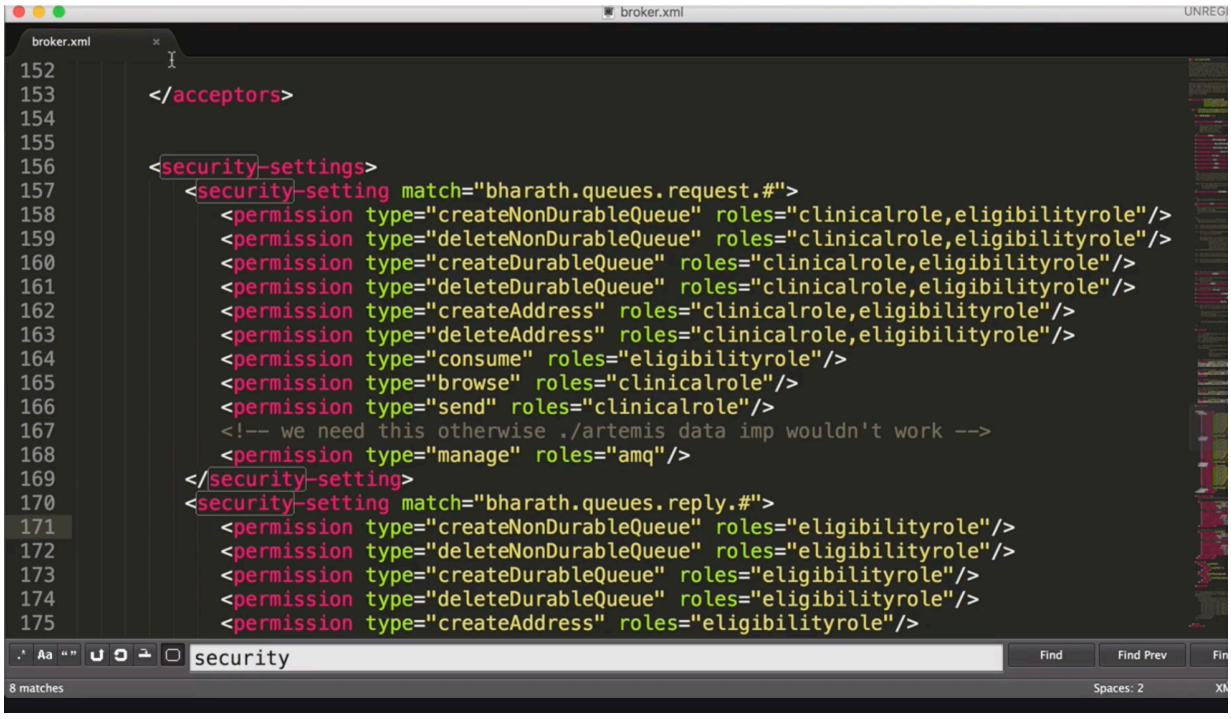
Code:
artemis-user.properties:



artemis-roles.properties:



broker.xml:



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.

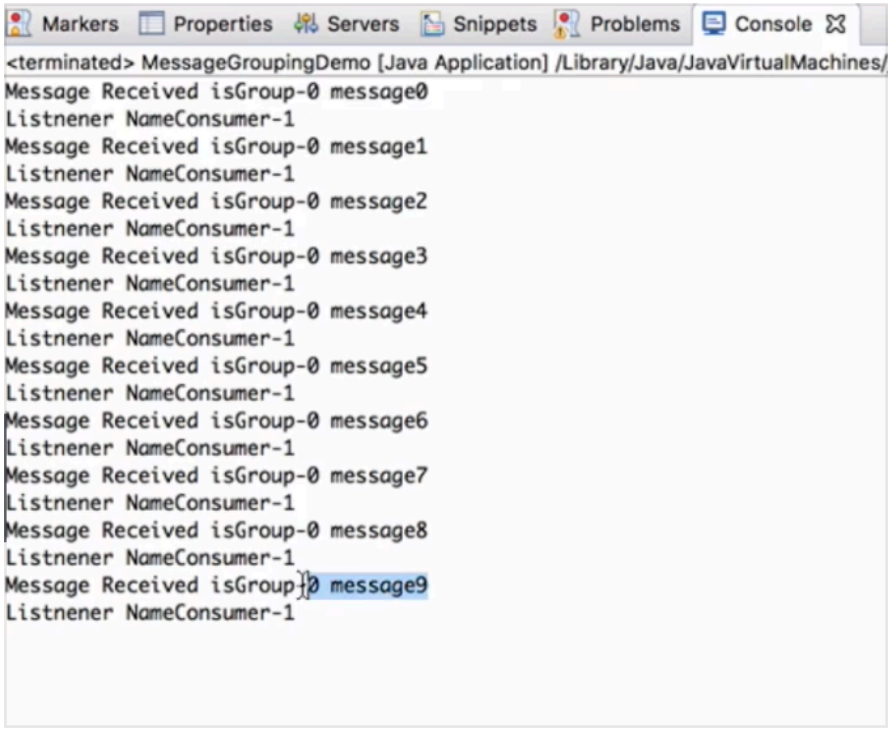
MessageGroupingDemo.java

```
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");
                }
            }

        }

    }

}
```

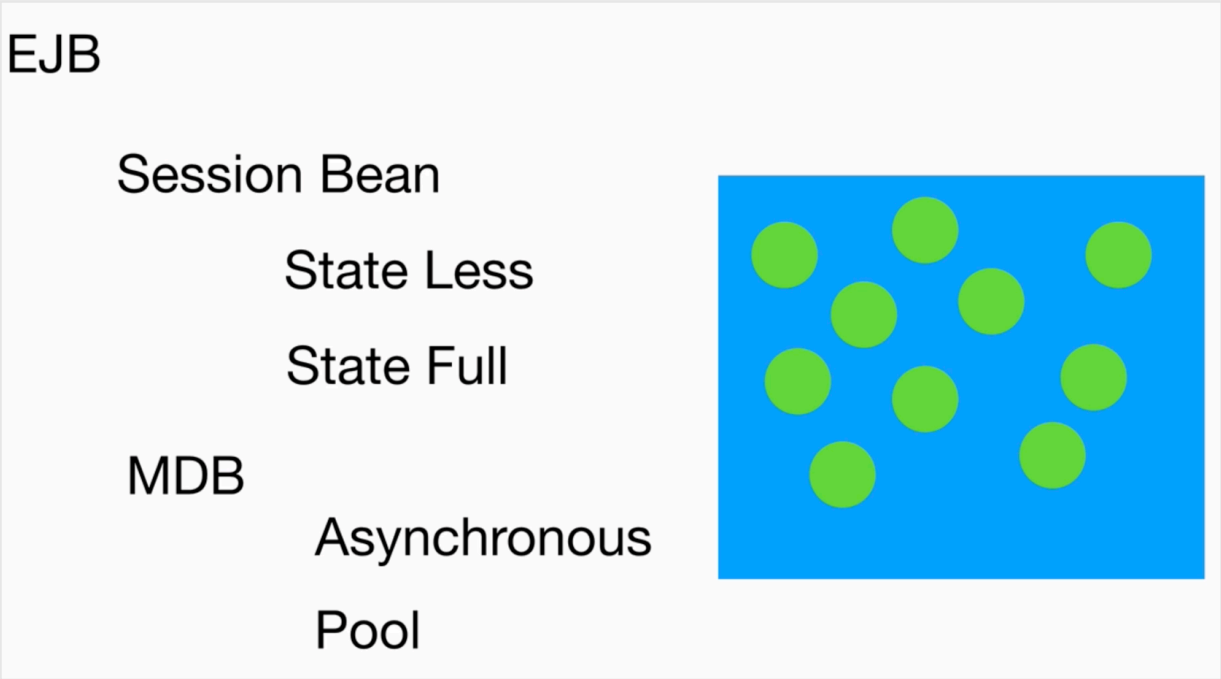
Put a Thread.sleep(2000); before checking Consumer-1 or 2.

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();
        }
    }

}
```



Java EE and Message Driven Beans:



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

Create a maven project:

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

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

Create a Message Producer:

jeejms/pom.xml | MyMessageProducer.java

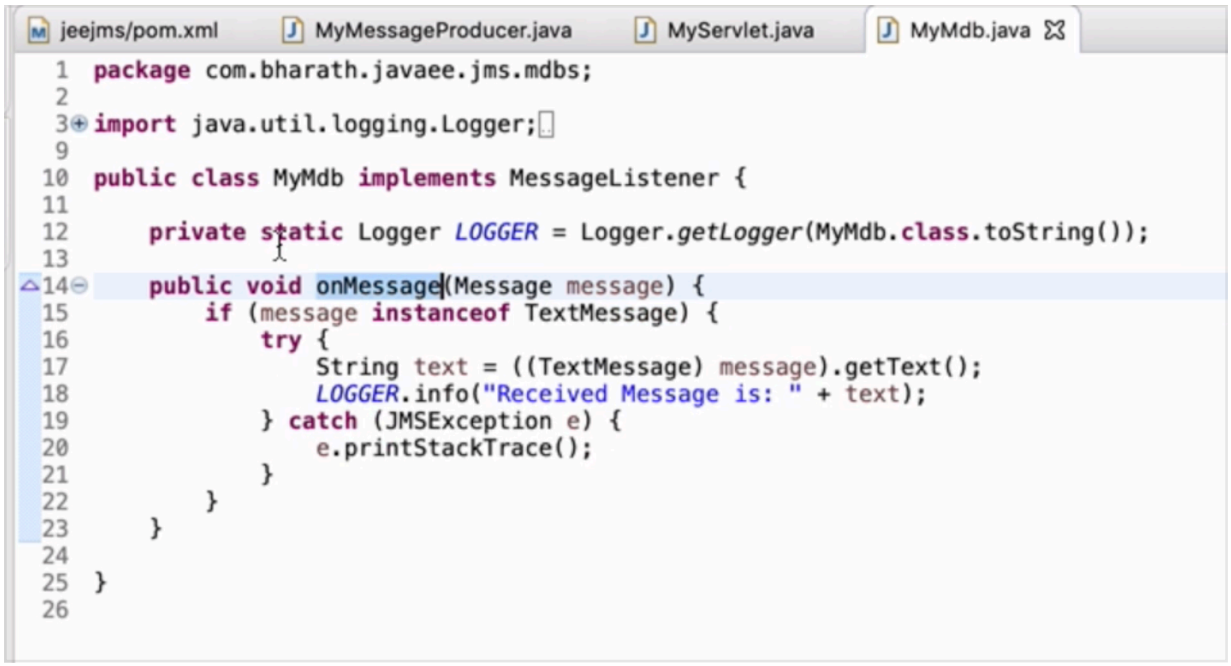
```java
package com.bharath.javaee.jms;

import javax.annotation.Resource;
import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.inject.Inject;
import javax.jms.JMSContext;
import javax.jms.Queue;

@Stateless
@LocalBean
public class MyMessageProducer {

    @Resource(mappedName = "java:/queue/myQueue")
    Queue myQueue;

    @Inject
    JMSContext jmsContext;

    public void sendMessage(String message) {
        jmsContext.createProducer().send(myQueue, message);
    }

}
```

Create servlet:

jeejms/pom.xml | MyMessageProducer.java | *MyServlet.java

```java
package com.bharath.javaee.servlets;

import java.io.IOException;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.bharath.javaee.jms.MyMessageProducer;

@WebServlet(urlPatterns = "/")
public class MyServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;
    @EJB
    MyMessageProducer producer;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String message = "Hello Message from JavaEE Server using JMS!!";
        producer.sendMessage(message);
        resp.getWriter().write("Published the message" + message);

    }

}
```

Create MDB:



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")
})

jeejms/pom.xml | MyMessageProducer.java | MyServlet.java | MyMdb.java

```java
package com.bharath.javaee.jms.mdbs;

import java.util.logging.Logger;

import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

@MessageDriven(name = "MyMdb", activationConfig = {
        @ActivationConfigProperty(propertyName = "destination", propertyValue = "queue/myQueue"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "aknowledgeMode", propertyValue = "Auto-acknowledge") })
public class MyMdb implements MessageListener {

    private static Logger LOGGER = Logger.getLogger(MyMdb.class.toString());

    public void onMessage(Message message) {
        if (message instanceof TextMessage) {
            try {
                String text = ((TextMessage) message).getText();
                LOGGER.info("Received Message is: " + text);
            } catch (JMSException e) {
                e.printStackTrace();
            }
        }
    }

}
```

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.

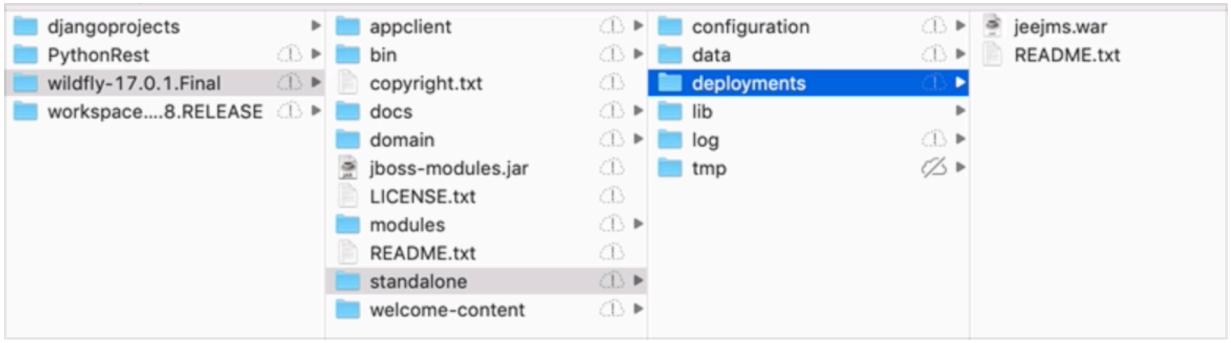
jeejms/pom.xml | MyMessageProducer.java | MyServlet.java | MyMdb.java

```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>
```

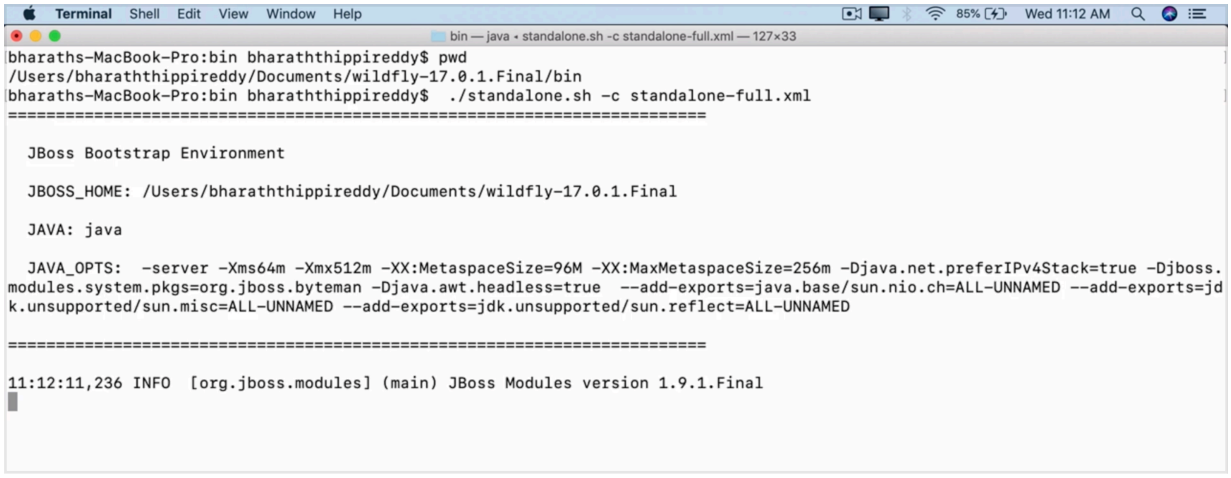
Text version of the updated pom.xml:

```
<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>
```

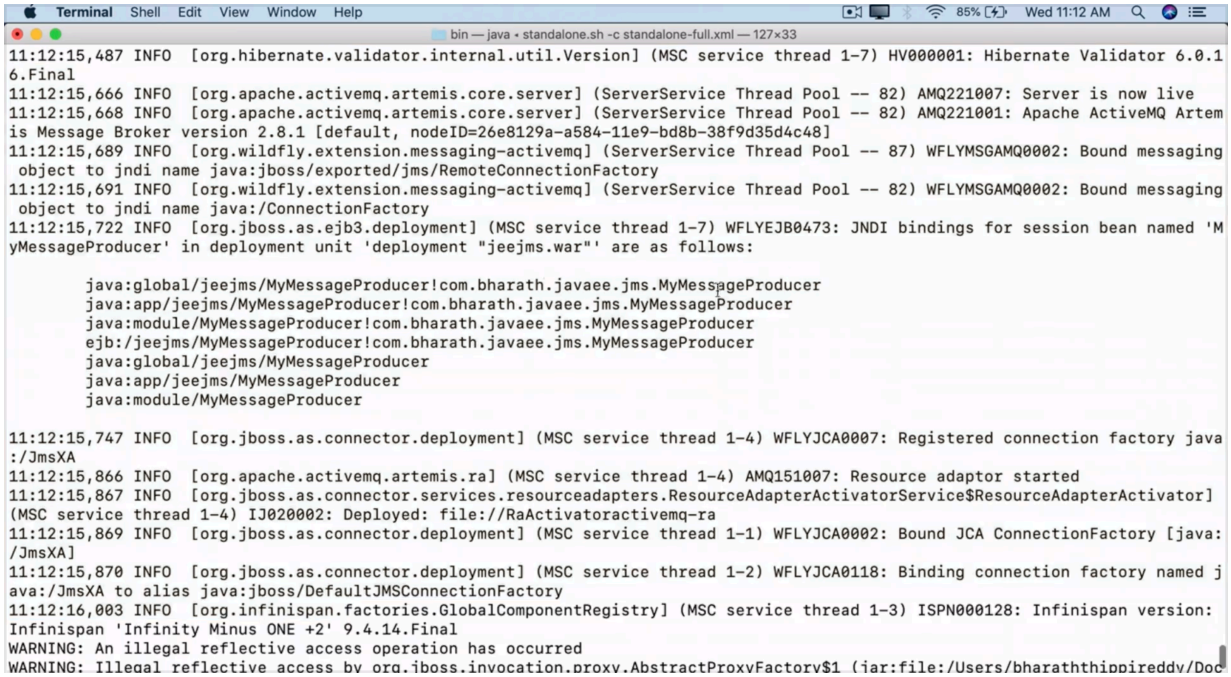
Deploy and Run:
Maven clean and build to get the deployable .war file.
Put the .war into wildFly_Home/standalone/deployments/ path.



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

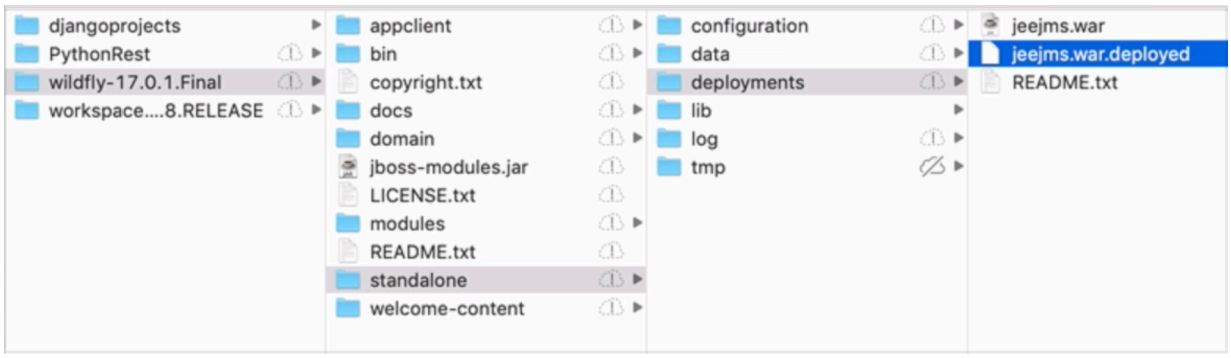


It will deploy the message bean and all as in below log



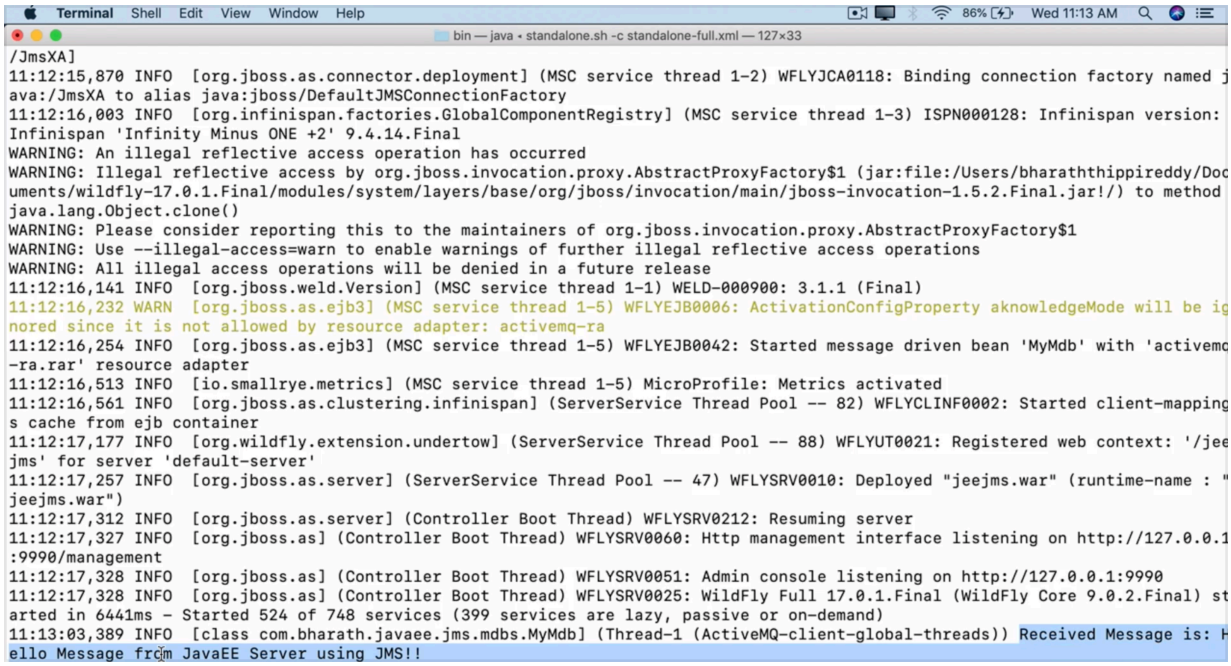
It will create warFileName.war.deployed in JBoss wildFly deployments

directory.

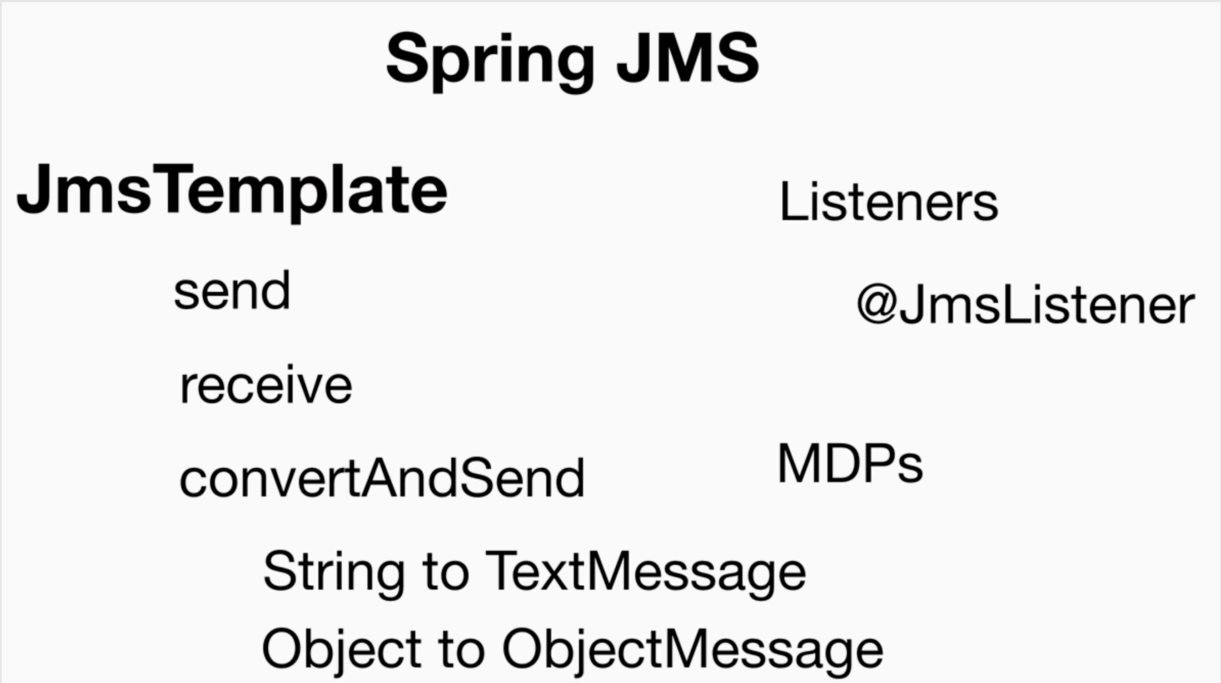


It will run successfully if hit with JBoss default port 8080 means localhost:8080/warFileContextName/ as below.

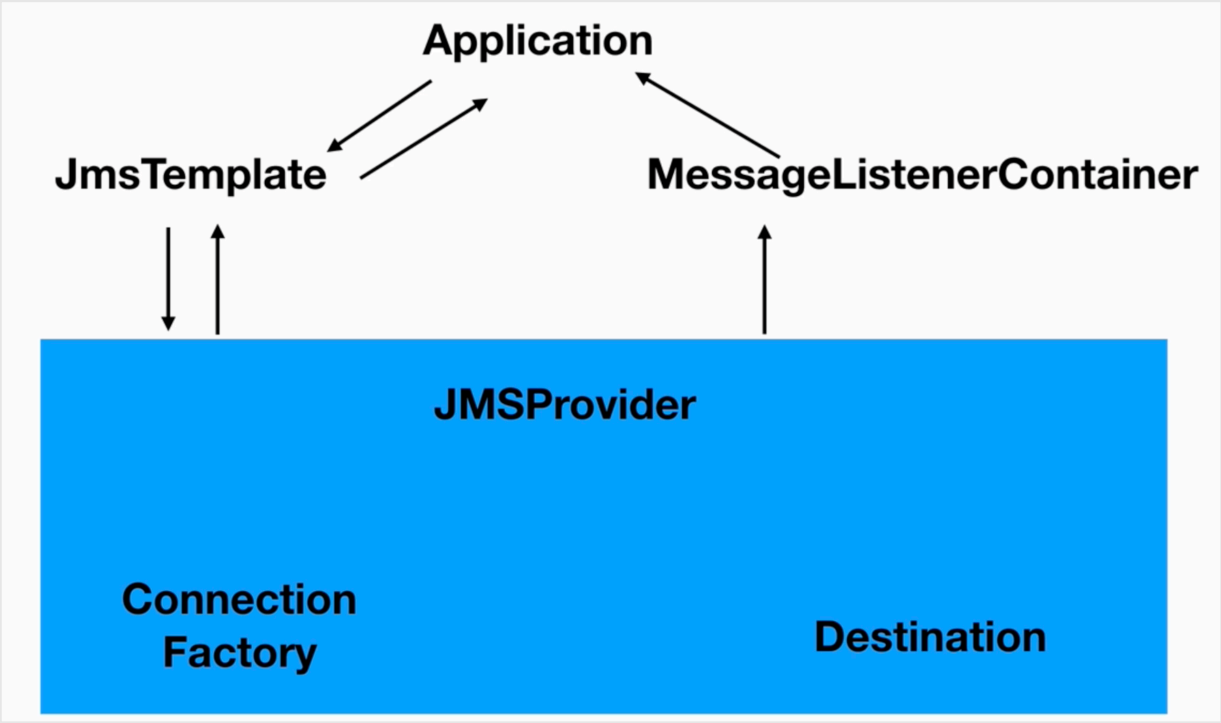
It also posts JMS message and receives as highlighted in below log:



Spring JMS Overview:



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:

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

springjms/pom.xml

```
        <version>2.1.9.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.bharath.springjms</groupId>
    <artifactId>springjms</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springjms</name>
    <description>Spring JMS Demo</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-artemis</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
```

Add @EnableJms to spring boot starter main class.

springjms/pom.xml | SpringjmsApplication.java

```java
package com.bharath.springjms;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@EnableJms
public class SpringjmsApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringjmsApplication.class, args);
    }

}
```

Message Sender:

springjms/pom.xml | SpringjmsApplication.java | MessageSender.java | application.properties

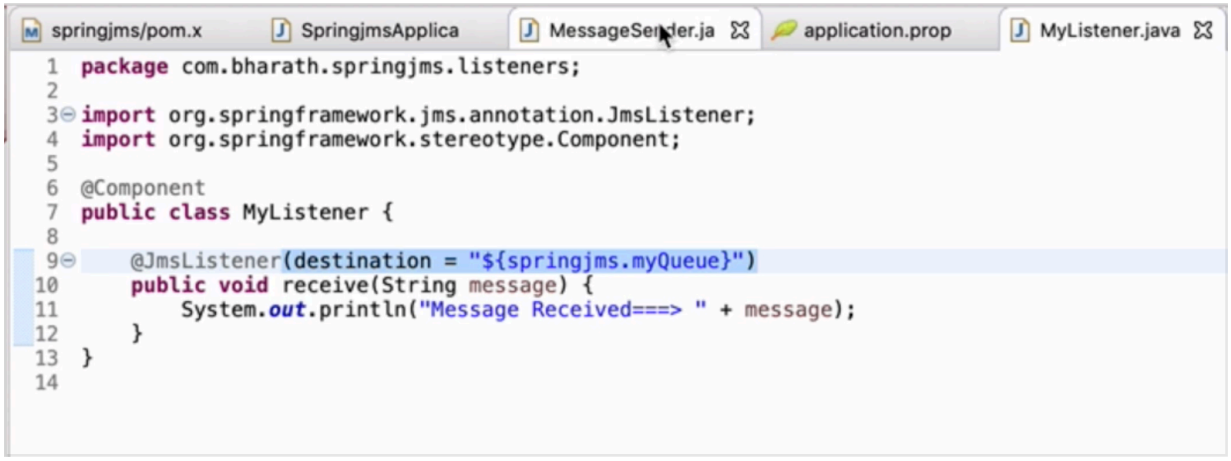
```
springjms.myQueue=myQueue
```

springjms/pom.x | SpringjmsApplica | MessageSender.ja | applicati

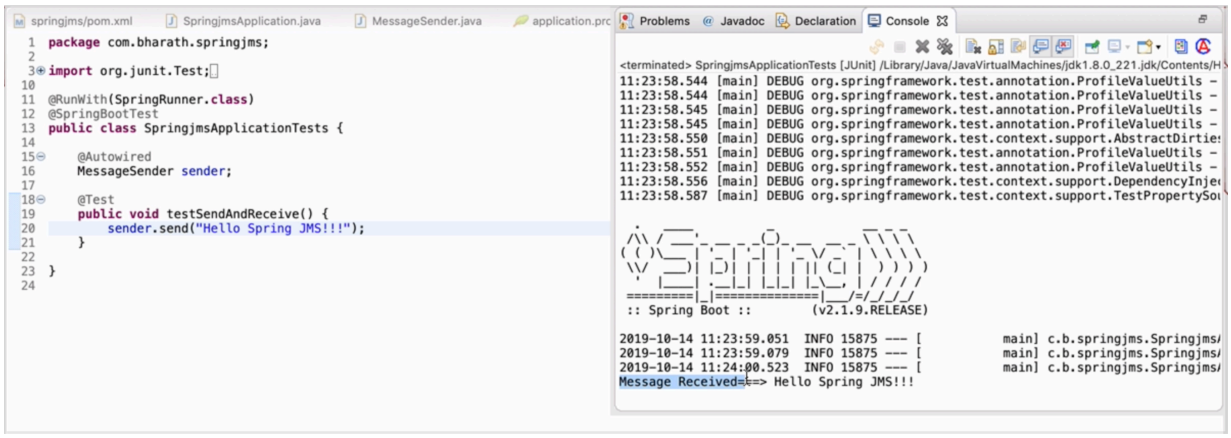
```java
package com.bharath.springjms.senders;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Component;

@Component
public class MessageSender {

    @Autowired
    private JmsTemplate jmsTemplate;

    @Value("${springjms.myQueue}")
    private String queue;

    public void send(String message) {
        jmsTemplate.convertAndSend(queue, message);
    }
}
```

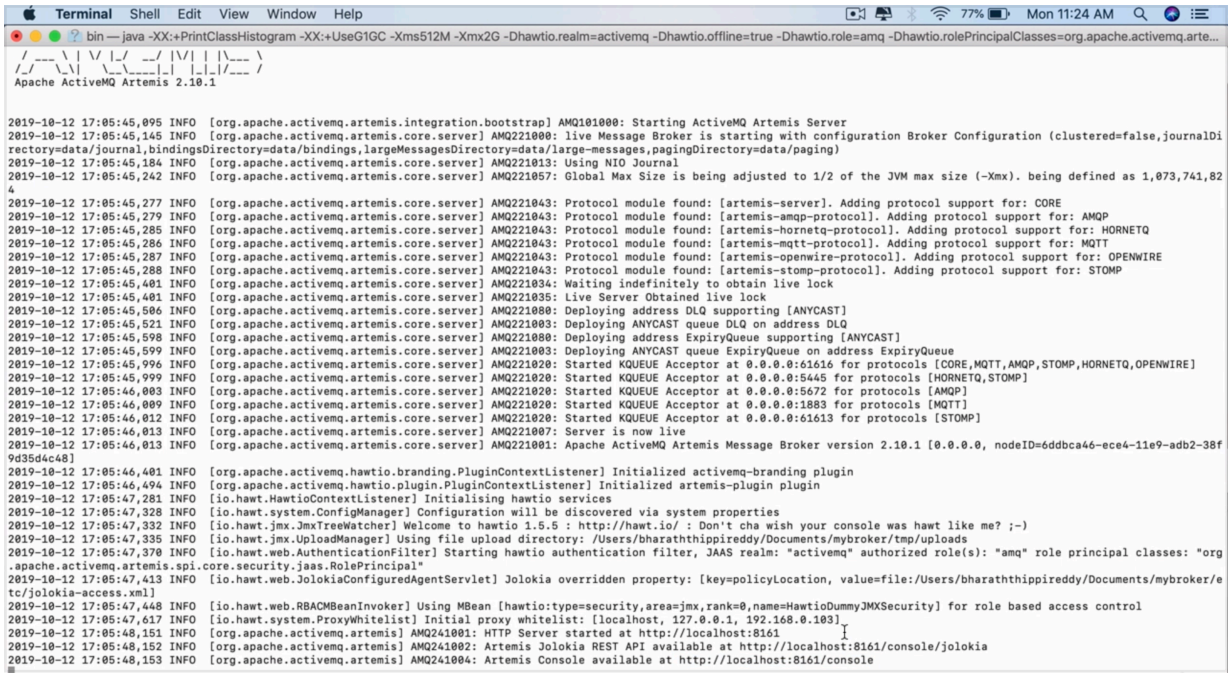
Message Receiver:

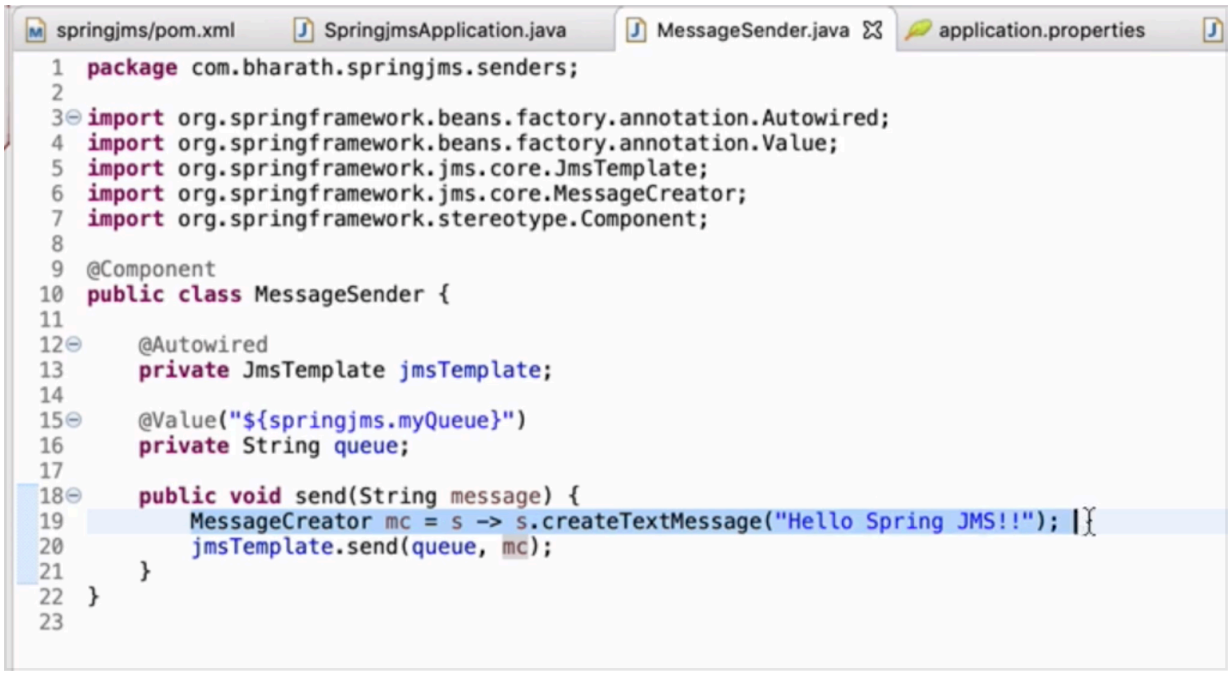
jmsTemplate.receive gets blocked until there is any message. Better to use @JMSListener



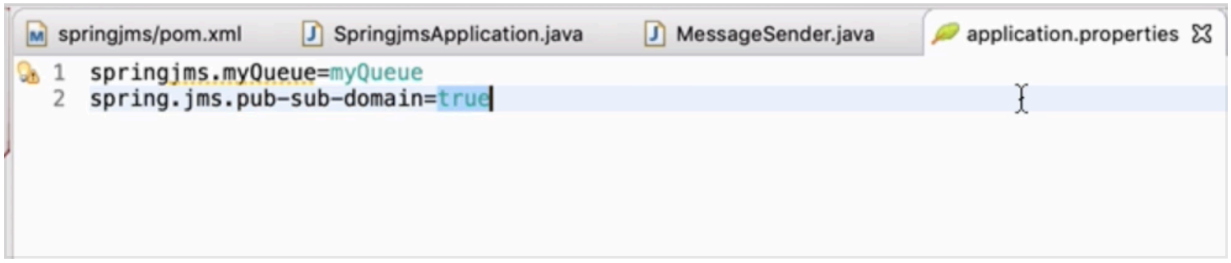
Test:
Make sure artemis is up.

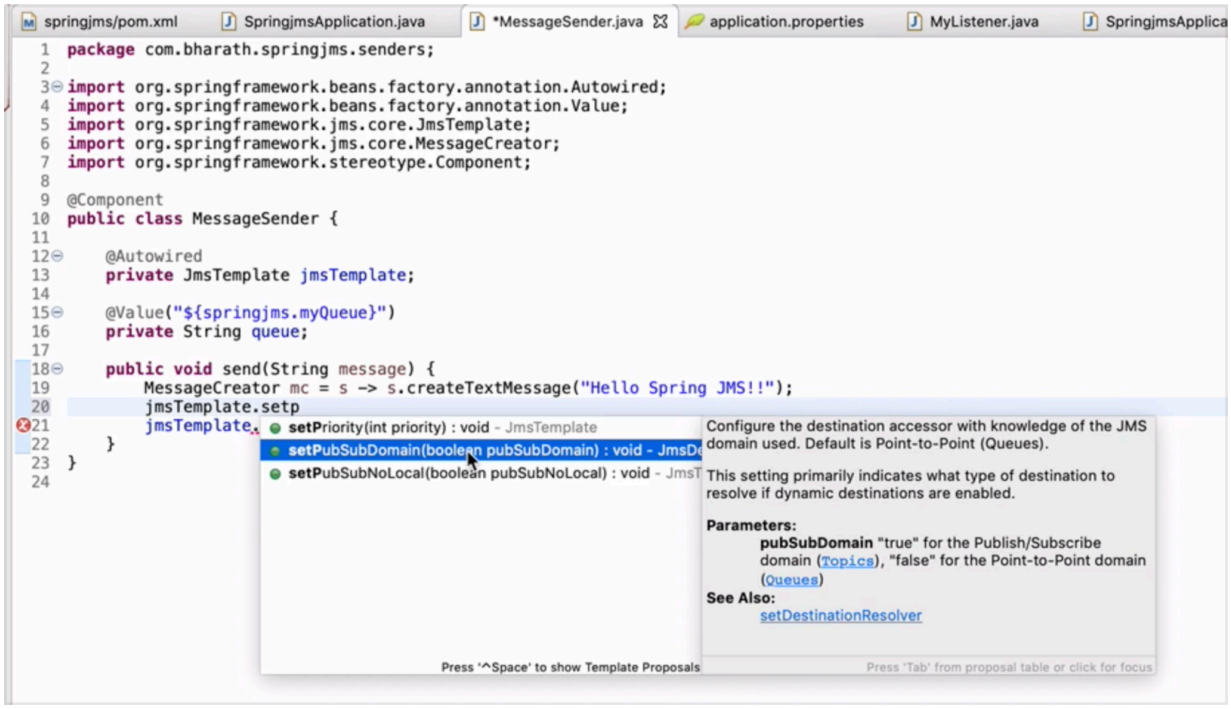


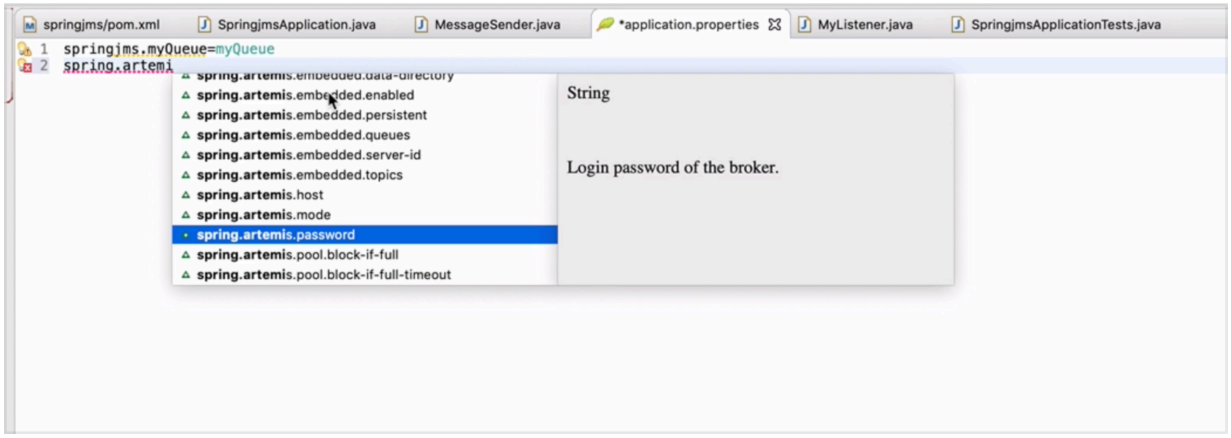




Default value of spring.jms.pub-sub-domain is false means Spring configures queue for P2P communication - not topic for publish subscribe communication. We can configure in application.properties file for all JMS communication or for a particular jmsTemplate by using jmsTemplate.setPubSubDomain(true)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







Bonus Lecture

Connect with me :

http://www.bharaththippireddy.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