JV-107 JMS 1.1 with ActiveMQ

Assignment

Table of Contents

JMS Assignments

This assignment is a step-by-step guide on how to configure ActiveMQ/JBoss and create, send and receive messages using the JMS API. By the time you have worked your way through the tutorial material you will see how the constituent parts of the JMS API, namely Destination, session, provider etc. all fit together in the context of a messaging in an application.

As part of your deliverables, you must provide following:

1. Running programs using ActiveMQ or JBoss.
2. Compilable source code of the application
3. A small write-up explaining the features of JMS API.
4. Code should be refactored according to the coding standards and guidelines.

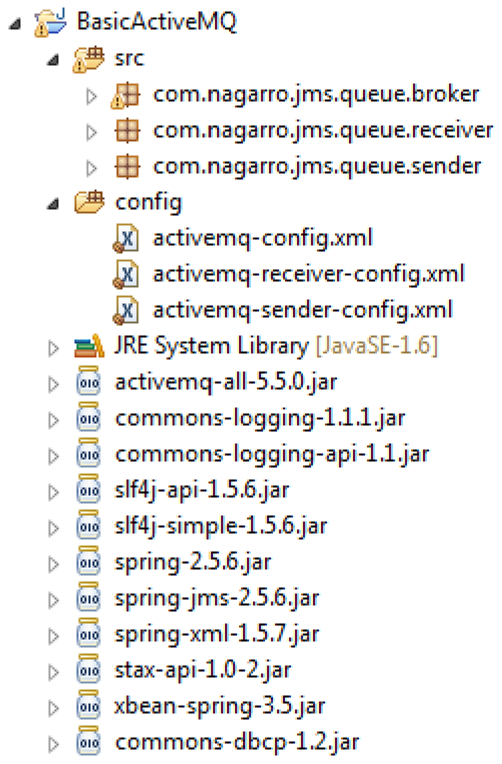
Prerequisite Software

The following prerequisite software and environment setup is assumed. You should also be reasonably comfortable using the following technologies:

- Java SDK 1.5 or above (http://www.oracle.com/technetwork/java/javase/downloads/index.html)
- Active MQ 5.5 (http://activemq.apache.org/download.html)
- Eclipse JavaEE 3.6 (http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/heliossr2)
- MySQL 5.5.14 (http://www.mysql.com/downloads/)
- JBoss 4.0.2(http://www.jboss.org/jbossas/downloads/)

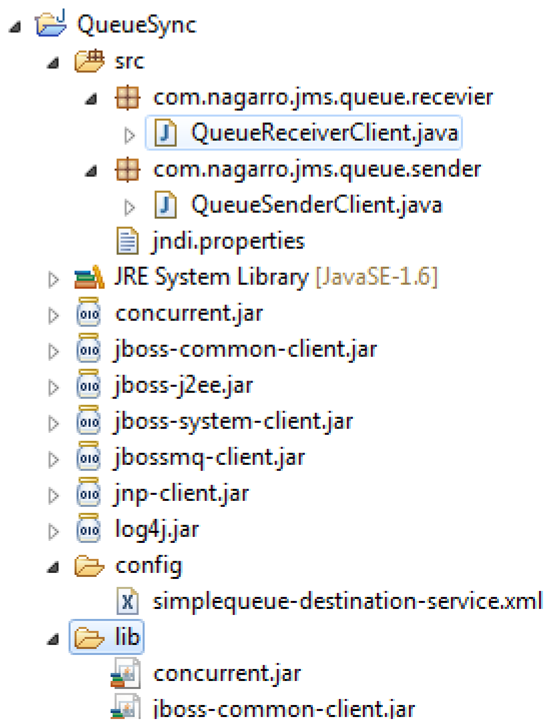
Application package Structure

Following is the final application structure for an Active MQ project which will be created step by step:



```
BasicActiveMQ
  src
    com.nagarro.jms.queue.broker
    com.nagarro.jms.queue.receiver
    com.nagarro.jms.queue.sender
  config
    activemq-config.xml
    activemq-receiver-config.xml
    activemq-sender-config.xml
  JRE System Library [JavaSE-1.6]
  activemq-all-5.5.0.jar
  commons-logging-1.1.1.jar
  commons-logging-api-1.1.jar
  slf4j-api-1.5.6.jar
  slf4j-simple-1.5.6.jar
  spring-2.5.6.jar
  spring-jms-2.5.6.jar
  spring-xml-1.5.7.jar
  stax-api-1.0-2.jar
  xbean-spring-3.5.jar
  commons-dbcp-1.2.jar
```

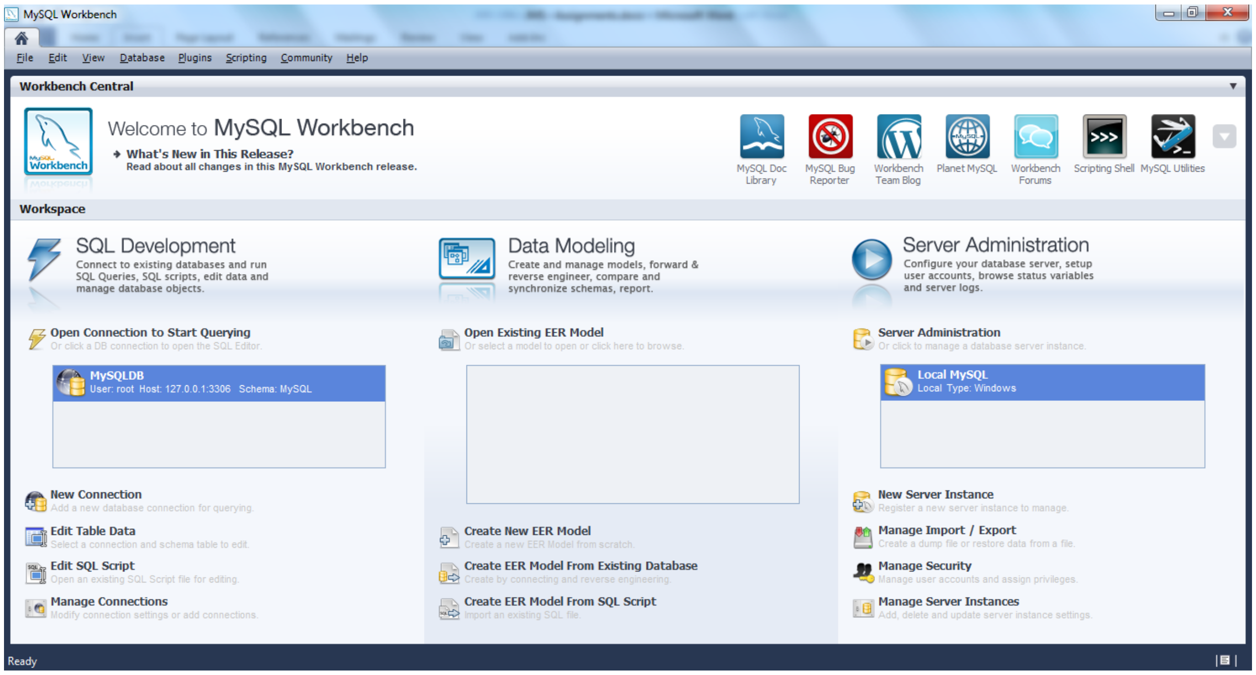
The package structure for a Java project using JBoss as a provider would be as shown below:



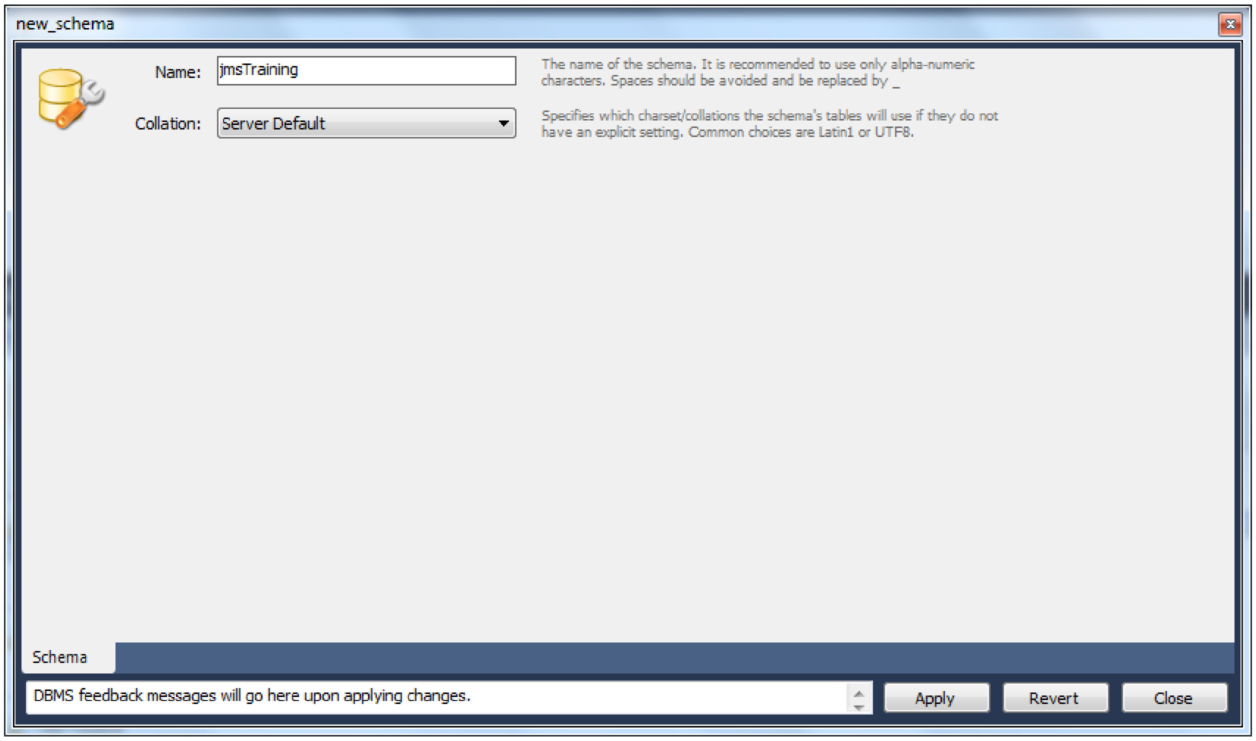
```
QueueSync
    src
        com.nagarro.jms.queue.recevier
            QueueReceiverClient.java
        com.nagarro.jms.queue.sender
            QueueSenderClient.java
        jndi.properties
    JRE System Library [JavaSE-1.6]
    concurrent.jar
    jboss-common-client.jar
    jboss-j2ee.jar
    jboss-system-client.jar
    jbossmq-client.jar
    jnp-client.jar
    log4j.jar
    config
        simplequeue-destination-service.xml
    lib
        concurrent.jar
        jboss-common-client.jar
```

MySQL Setup

1) Download MySQL installer and run the setup
2) Start the MySQL service using the MySQL workbench


MySQL Workbench
File Edit View Database Plugins Scripting Community Help
Workbench Central
Welcome to MySQL Workbench
What's New in This Release?
Read about all changes in this MySQL Workbench release.
MySQL Doc Library
MySQL Bug Reporter
Workbench Team Blog
Planet MySQL
Workbench Forums
Scripting Shell
MySQL Utilities
Workspace
SQL Development
Connect to existing databases and run SQL Queries, SQL scripts, edit data and manage database objects.
Data Modeling
Create and manage models, forward & reverse engineer, compare and synchronize schemas, report.
Server Administration
Configure your database server, setup user accounts, browse status variables and server logs.
Open Connection to Start Querying
Open Existing EER Model
Server Administration
MySQLDB
User: root Host: 127.0.0.1:3306 Schema: MySQL
Local MySQL
Local Type: Windows
New Connection
Edit Table Data
Edit SQL Script
Manage Connections
Create New EER Model
Create EER Model From Existing Database
Create EER Model From SQL Script
New Server Instance
Manage Import / Export
Manage Security
Manage Server Instances
Ready

3) Create a connection to the server instance using “Open connection to start Querying”
4) Click on Add schema and create a schema with the name jmsTraining as shown below:


new_schema

Name: jmsTraining

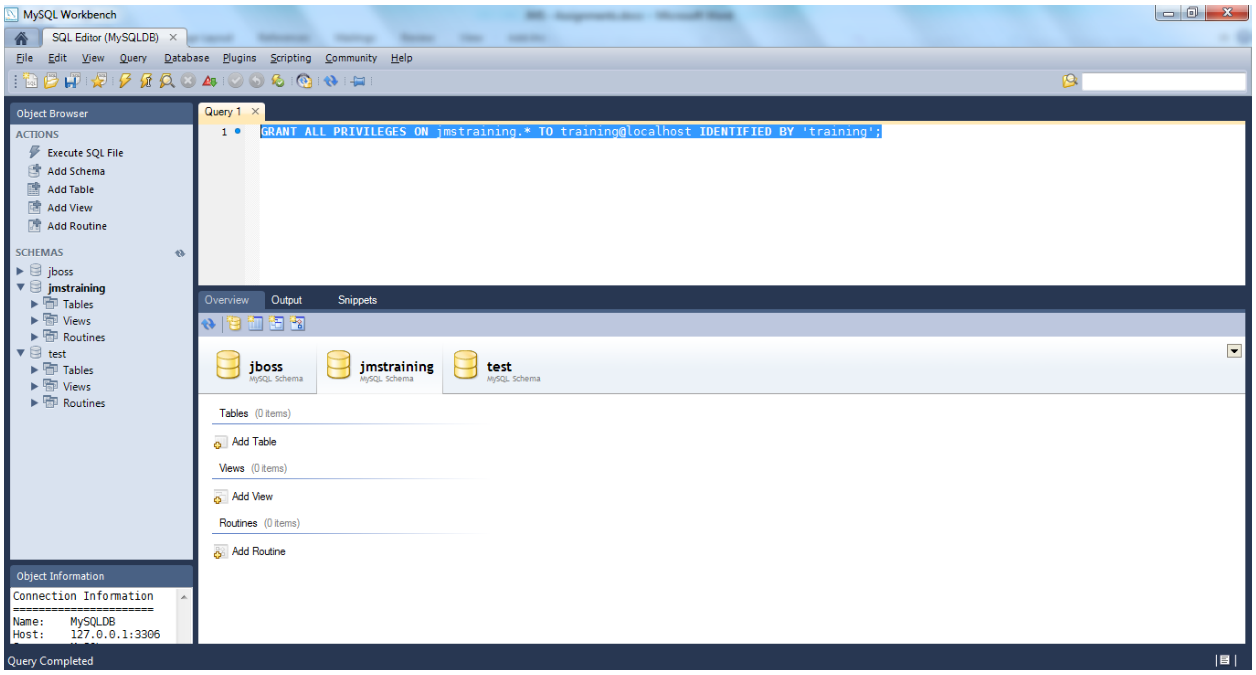
The name of the schema. It is recommended to use only alpha-numeric characters. Spaces should be avoided and be replaced by _

Collation: Server Default

Specifies which charset/collations the schema's tables will use if they do not have an explicit setting. Common choices are Latin1 or UTF8.

DBMS feedback messages will go here upon applying changes.

5) Run the below query to create a user training with password training:
GRANT ALL PRIVILEGES ON jmstraining.* TO training@localhost IDENTIFIED BY 'training';

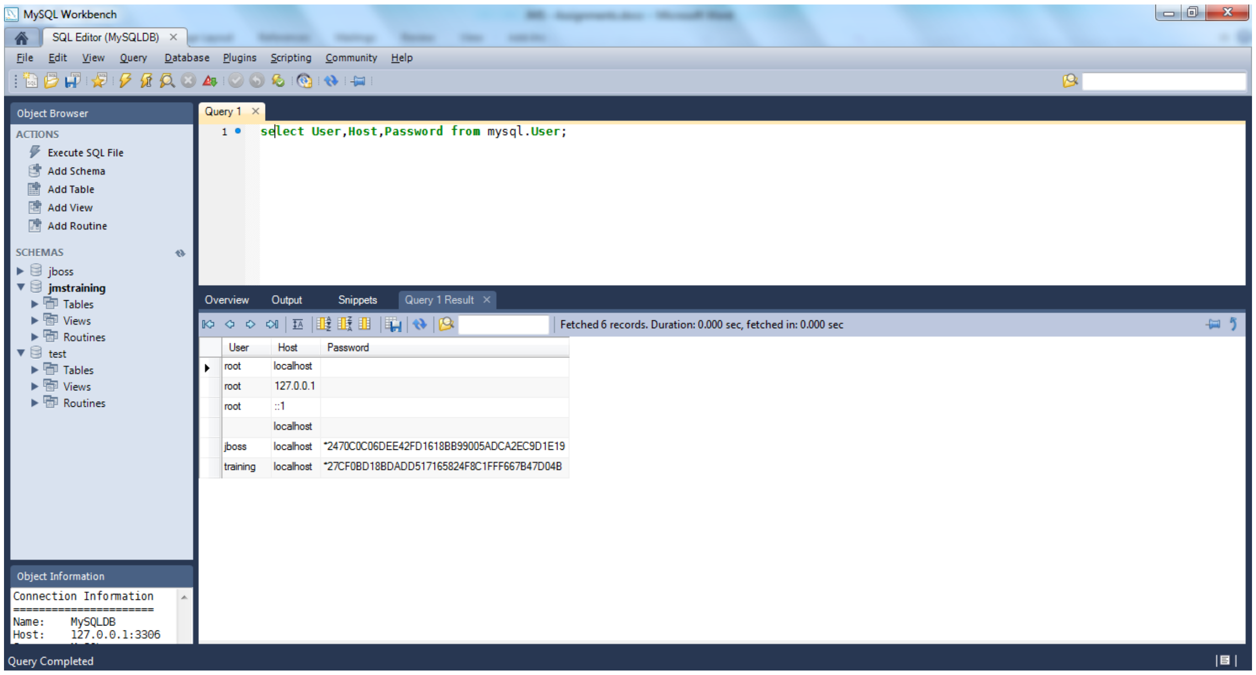


```
GRANT ALL PRIVILEGES ON jmstraining.* TO training@localhost IDENTIFIED BY 'training';
```

```
Connection Information
======================
Name:    MySQLDB
Host:    127.0.0.1:3306
```

Query Completed

6) Run the query as shown below to check that training user has been created in the jmstraining database.



```
select User,Host,Password from mysql.User;
```

Fetched 6 records. Duration: 0.000 sec, fetched in: 0.000 sec

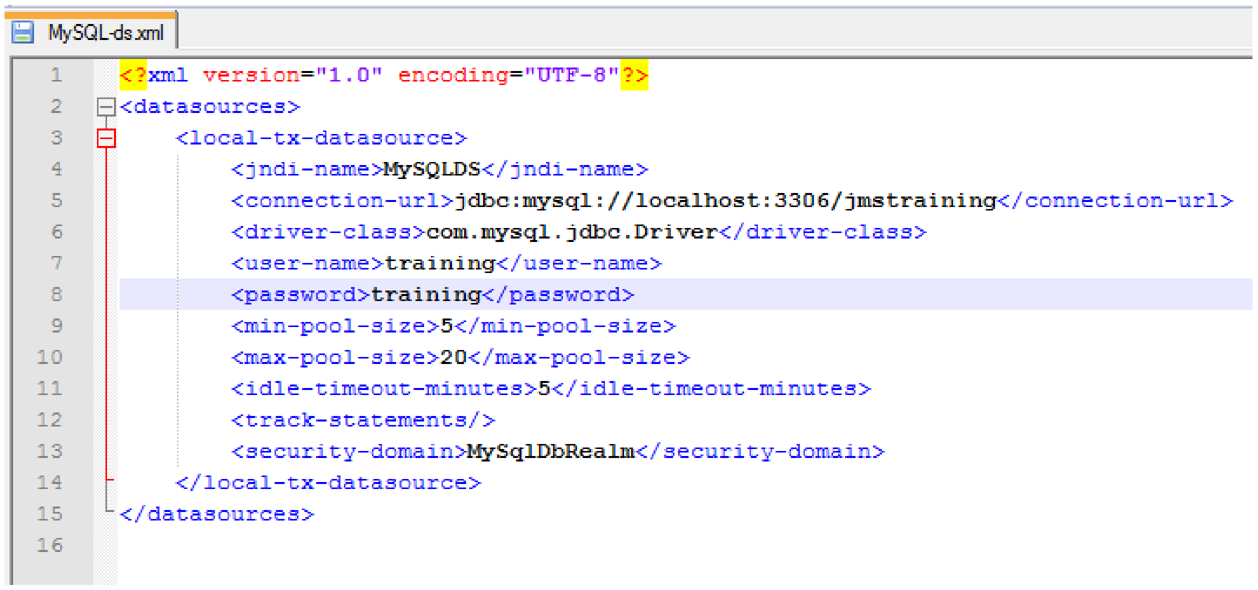
```
User      Host       Password
root      localhost
root      127.0.0.1
root      ::1
          localhost
jboss     localhost  *2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19
training  localhost  *27CF0BD18BDADD517165824F8C1FFF667B47D04B
```

```
Connection Information
======================
Name:    MySQLDB
Host:    127.0.0.1:3306
```

Query Completed

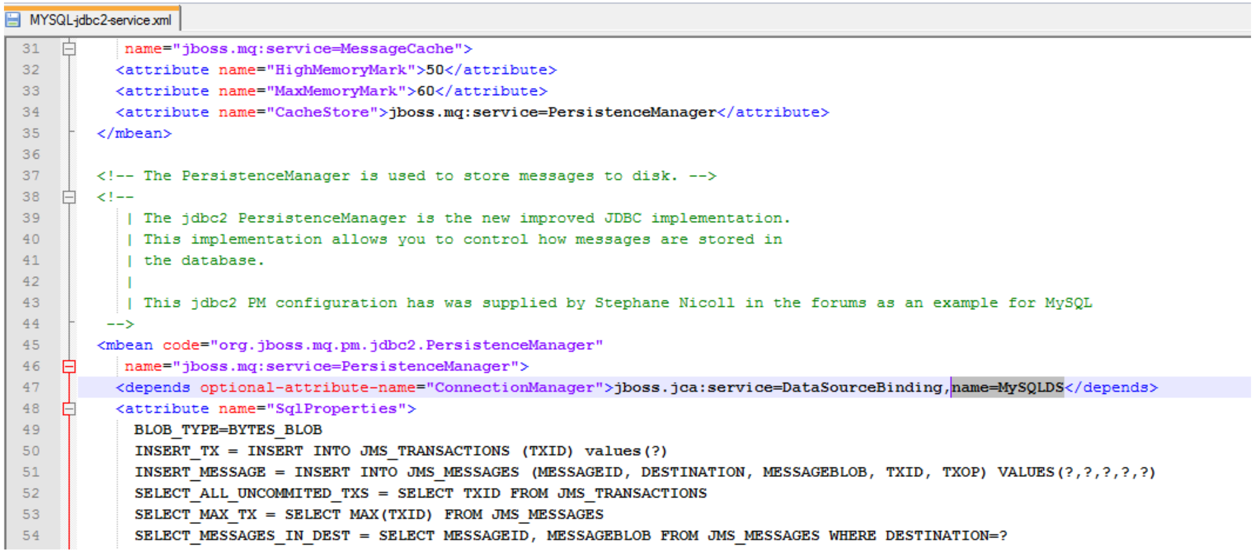
JBoss Setup

1) Download JBoss-4.0.2.zip
2) Extract JBoss into a directory
3) We will use the default configuration provided by JBoss
4) Configuring JMS
    a. By default JMS is configure on JBoss. But default JMS is configured to use hsql db (in memory database). Due to this if JBoss is re-started; all the unprocessed messages in the queue(s) will be lost. We will configure JBoss to use MySQL database.
    b. Create a file MySQL-ds.xml in Jboss-4.0.2\server\default\deploy. File name must end with –ds.xml. This will define the data source MySQLDS.


MySQL-ds.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
    <local-tx-datasource>
        <jndi-name>MySQLDS</jndi-name>
        <connection-url>jdbc:mysql://localhost:3306/jmstraining</connection-url>
        <driver-class>com.mysql.jdbc.Driver</driver-class>
        <user-name>training</user-name>
        <password>training</password>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>20</max-pool-size>
        <idle-timeout-minutes>5</idle-timeout-minutes>
        <track-statements/>
        <security-domain>MySqlDbRealm</security-domain>
    </local-tx-datasource>
</datasources>
```

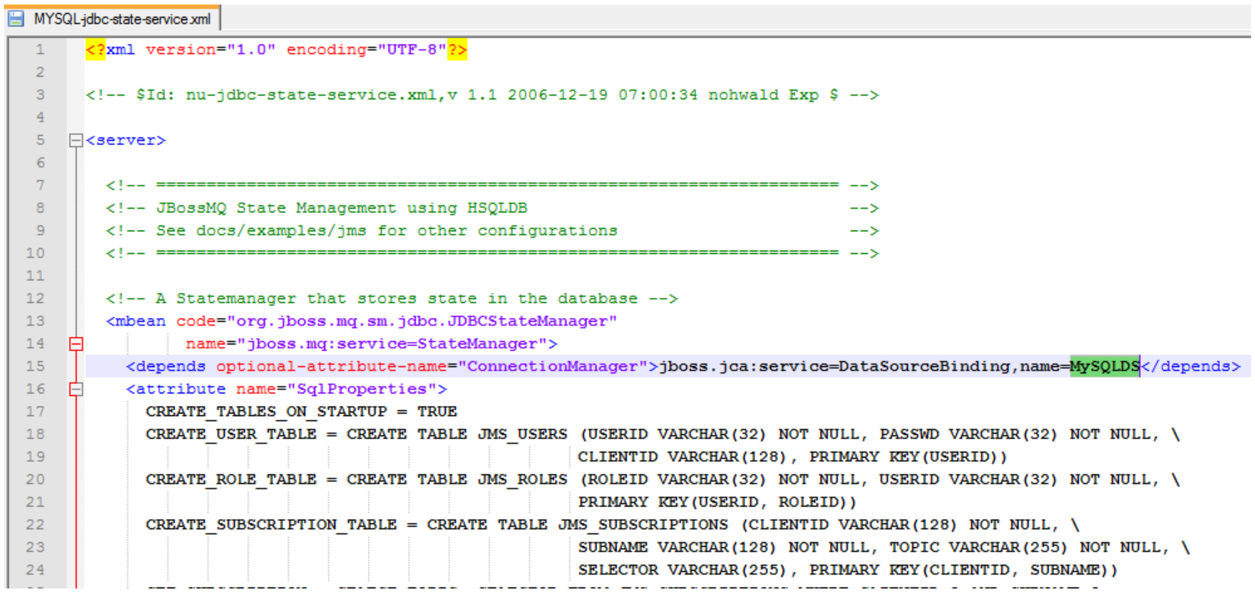
5) Rename following files from location jboss-4.0.2\server\default\deploy\jms
    a. hsqldb-jdbc2-service.xml to hsqldb-jdbc2-service.xml.old
    b. hsqldb-jdbc-state-service.xml to hsqldb-jdbc-state-service.xml.old
6) Create a file MYSQL-jdbc2-service.xml and paste the contents from the file hsqldb-jdbc2-service.xml in the directory jboss-4.0.2\server\default\deploy\jms
    a. Change the data source name from DefaultDS to MySQLDS (as created in the MySQL-ds.xml in step 4 above)



MYSQL-jdbc2-service.xml

```
     name="jboss.mq:service=MessageCache">
    <attribute name="HighMemoryMark">50</attribute>
    <attribute name="MaxMemoryMark">60</attribute>
    <attribute name="CacheStore">jboss.mq:service=PersistenceManager</attribute>
  </mbean>

  <!-- The PersistenceManager is used to store messages to disk. -->
  <!--
     | The jdbc2 PersistenceManager is the new improved JDBC implementation.
     | This implementation allows you to control how messages are stored in
     | the database.
     |
     | This jdbc2 PM configuration has was supplied by Stephane Nicoll in the forums as an example for MySQL
   -->
  <mbean code="org.jboss.mq.pm.jdbc2.PersistenceManager"
     name="jboss.mq:service=PersistenceManager">
    <depends optional-attribute-name="ConnectionManager">jboss.jca:service=DataSourceBinding,name=MySQLDS</depends>
    <attribute name="SqlProperties">
      BLOB_TYPE=BYTES_BLOB
      INSERT_TX = INSERT INTO JMS_TRANSACTIONS (TXID) values(?)
      INSERT_MESSAGE = INSERT INTO JMS_MESSAGES (MESSAGEID, DESTINATION, MESSAGEBLOB, TXID, TXOP) VALUES(?,?,?,?,?)
      SELECT_ALL_UNCOMMITED_TXS = SELECT TXID FROM JMS_TRANSACTIONS
      SELECT_MAX_TX = SELECT MAX(TXID) FROM JMS_MESSAGES
      SELECT_MESSAGES_IN_DEST = SELECT MESSAGEID, MESSAGEBLOB FROM JMS_MESSAGES WHERE DESTINATION=?
```

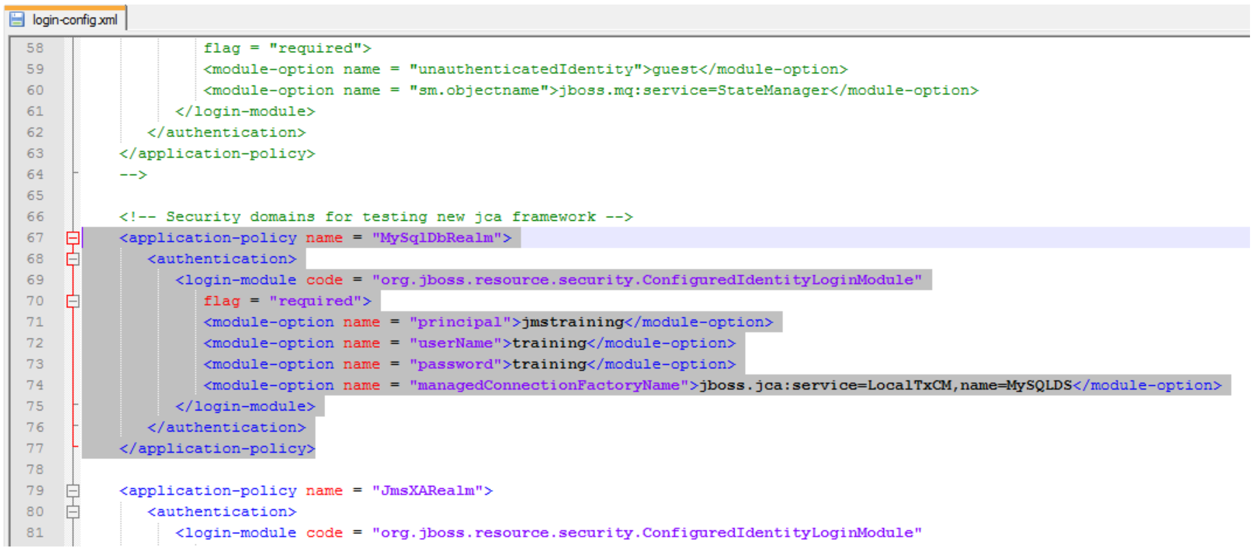
7) Create a file MYSQL-jdbc-state-service.xml and paste the contents from the file hsqldb-jdbc2-service.xml in the directory jboss-4.0.2\server\default\deploy\jms
   a. Change the data source name from DefaultDS to MySQLDS (as created in the MySQL-ds.xml in step 4 above)



MYSQL-jdbc-state-service.xml

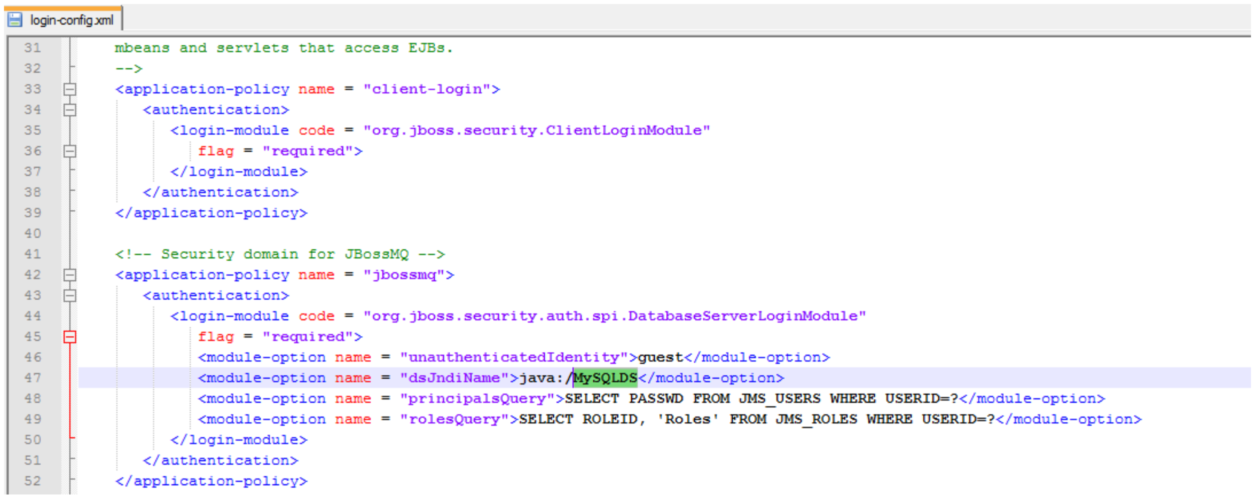
```
<?xml version="1.0" encoding="UTF-8"?>

<!-- $Id: nu-jdbc-state-service.xml,v 1.1 2006-12-19 07:00:34 nohwald Exp $ -->

<server>

  <!-- ==================================================================== -->
  <!-- JBossMQ State Management using HSQLDB                                -->
  <!-- See docs/examples/jms for other configurations                       -->
  <!-- ==================================================================== -->

  <!-- A Statemanager that stores state in the database -->
  <mbean code="org.jboss.mq.sm.jdbc.JDBCStateManager"
         name="jboss.mq:service=StateManager">
    <depends optional-attribute-name="ConnectionManager">jboss.jca:service=DataSourceBinding,name=MySQLDS</depends>
    <attribute name="SqlProperties">
      CREATE_TABLES_ON_STARTUP = TRUE
      CREATE_USER_TABLE = CREATE TABLE JMS_USERS (USERID VARCHAR(32) NOT NULL, PASSWD VARCHAR(32) NOT NULL, \
                                                  CLIENTID VARCHAR(128), PRIMARY KEY(USERID))
      CREATE_ROLE_TABLE = CREATE TABLE JMS_ROLES (ROLEID VARCHAR(32) NOT NULL, USERID VARCHAR(32) NOT NULL, \
                                                  PRIMARY KEY(USERID, ROLEID))
      CREATE_SUBSCRIPTION_TABLE = CREATE TABLE JMS_SUBSCRIPTIONS (CLIENTID VARCHAR(128) NOT NULL, \
                                                  SUBNAME VARCHAR(128) NOT NULL, TOPIC VARCHAR(255) NOT NULL, \
                                                  SELECTOR VARCHAR(255), PRIMARY KEY(CLIENTID, SUBNAME))
```

8) Copy the MySQL jdbc driver jar(mysql-connector-java-5.1.18-bin) to JBOSS_HOME/server/default/lib directory
9) Open file JBOSS_HOME/server/default/conf/login-config.xml and add an application-policy tag as shown below


login-config.xml

```
                    flag = "required">
                    <module-option name = "unauthenticatedIdentity">guest</module-option>
                    <module-option name = "sm.objectname">jboss.mq:service=StateManager</module-option>
                 </login-module>
              </authentication>
           </application-policy>
           -->

           <!-- Security domains for testing new jca framework -->
           <application-policy name = "MySqlDbRealm">
              <authentication>
                 <login-module code = "org.jboss.resource.security.ConfiguredIdentityLoginModule"
                    flag = "required">
                    <module-option name = "principal">jmstraining</module-option>
                    <module-option name = "userName">training</module-option>
                    <module-option name = "password">training</module-option>
                    <module-option name = "managedConnectionFactoryName">jboss.jca:service=LocalTxCM,name=MySQLDS</module-option>
                 </login-module>
              </authentication>
           </application-policy>

           <application-policy name = "JmsXARealm">
              <authentication>
                 <login-module code = "org.jboss.resource.security.ConfiguredIdentityLoginModule"
```

And change the data source name in the DataBaseServerLoginModule as shown below:


login-config.xml

```
           mbeans and servlets that access EJBs.
           -->
           <application-policy name = "client-login">
              <authentication>
                 <login-module code = "org.jboss.security.ClientLoginModule"
                    flag = "required">
                 </login-module>
              </authentication>
           </application-policy>

           <!-- Security domain for JBossMQ -->
           <application-policy name = "jbossmq">
              <authentication>
                 <login-module code = "org.jboss.security.auth.spi.DatabaseServerLoginModule"
                    flag = "required">
                    <module-option name = "unauthenticatedIdentity">guest</module-option>
                    <module-option name = "dsJndiName">java:/MySQLDS</module-option>
                    <module-option name = "principalsQuery">SELECT PASSWD FROM JMS_USERS WHERE USERID=?</module-option>
                    <module-option name = "rolesQuery">SELECT ROLEID, 'Roles' FROM JMS_ROLES WHERE USERID=?</module-option>
                 </login-module>
              </authentication>
           </application-policy>
```

10) Start JBoss using JBOSS_HOME/bin/run.bat

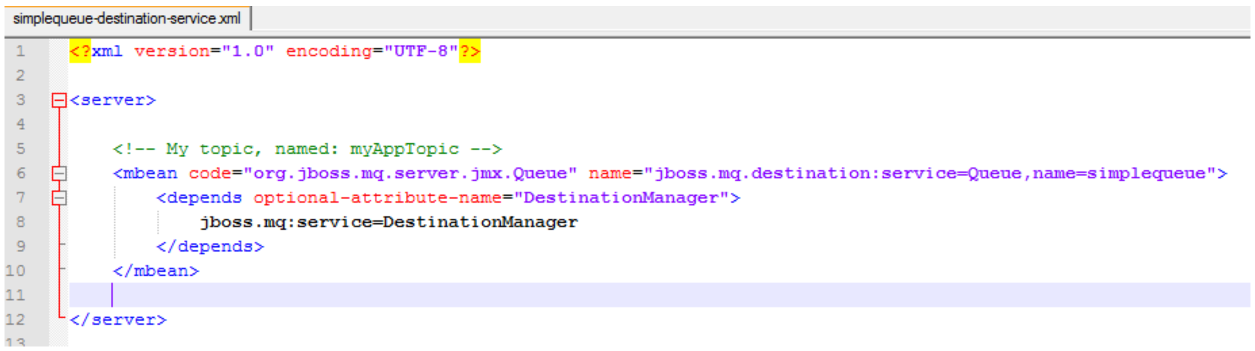
ActiveMQ-Spring Setup

1) Copy/Download the following jars

- activemq-all-5.5.0.jar
- commons-logging-1.1.1.jar
- commons-logging-api-1.1.jar
- slf4j-api-1.5.6.jar
- slf4j-simple-1.5.6.jar
- spring-2.5.6.jar
- spring-jms-2.5.6.jar
- spring-xml-1.5.7.jar
- stax-api-1.0-2.jar
- xbean-spring-3.5.jar
- commons-dbcp-1.2.jar
- commons-pool-1.5.jar
- sqljdbc4.jar
- mysql-connector-java-5.1.18-bin.jar

Queue Synchronized messaging example

1) Create a Java project in Eclipse and name it QueueSyncExample
2) Create a file simplequeue-destination-service.xml as shown below and paste it in the JBOSS_HOME/server/default/deploy directory. This will setup a queue with JNDI name queue/simplequeue in JBoss



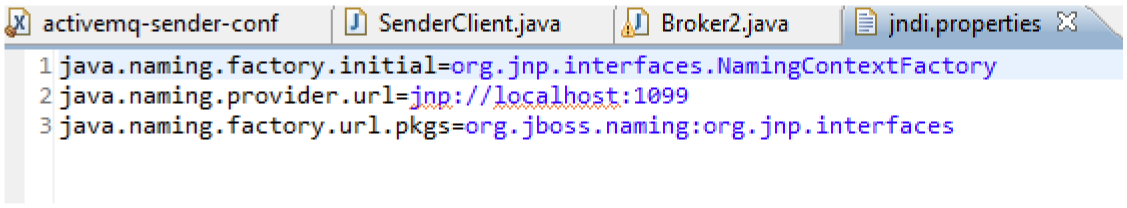
simplequeue-destination-service.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<server>

    <!-- My topic, named: myAppTopic -->
    <mbean code="org.jboss.mq.server.jmx.Queue" name="jboss.mq.destination:service=Queue,name=simplequeue">
        <depends optional-attribute-name="DestinationManager">
            jboss.mq:service=DestinationManager
        </depends>
    </mbean>

</server>
```

3) Copy the Jars shown below from the Jboss installation into the project's build path

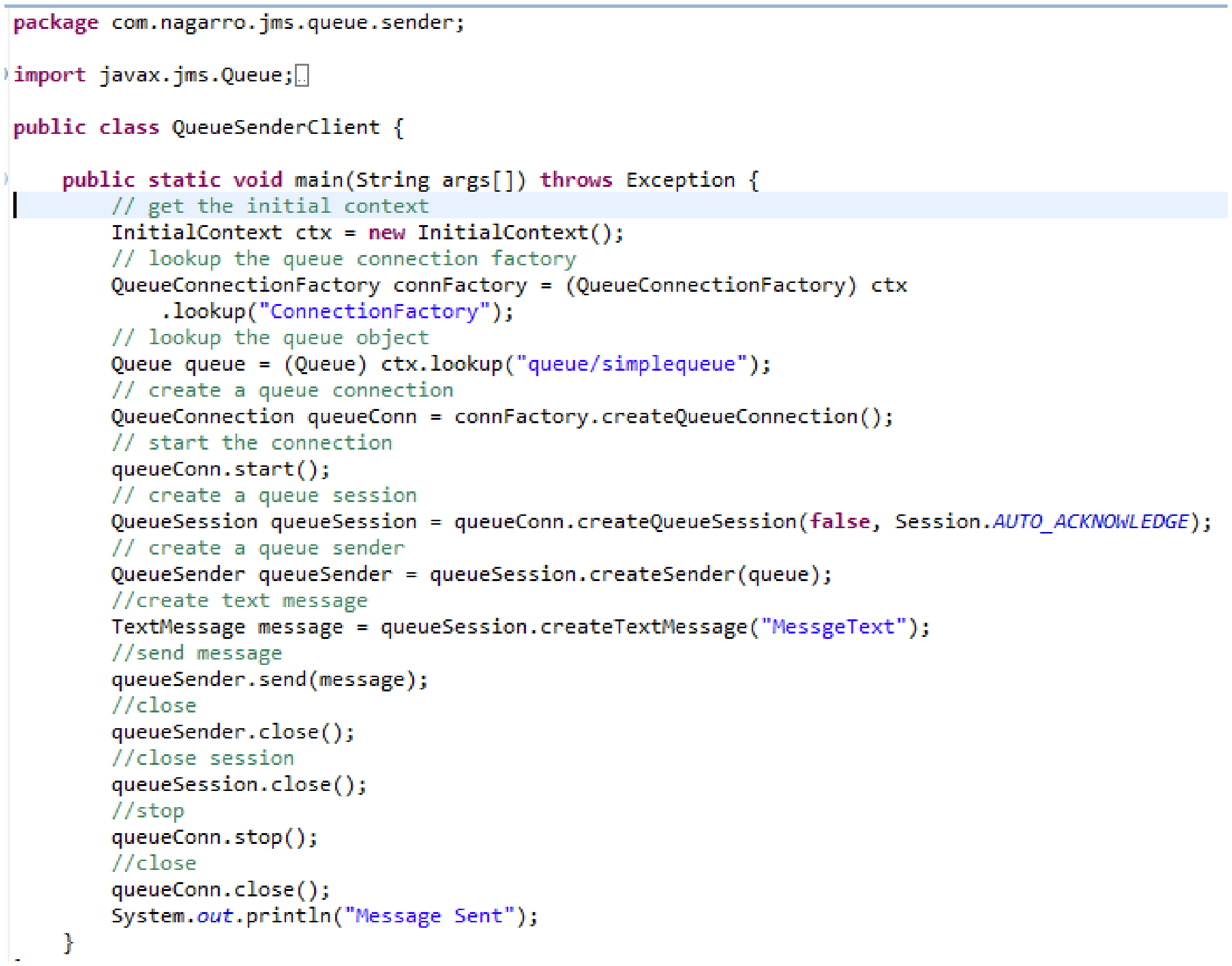
- lib
    - concurrent.jar
    - jboss-common-client.jar
    - jboss-j2ee.jar
    - jboss-system-client.jar
    - jbossmq-client.jar
    - jnp-client.jar
    - log4j.jar

4) Create a jndi.properties file as shown below in the src folder



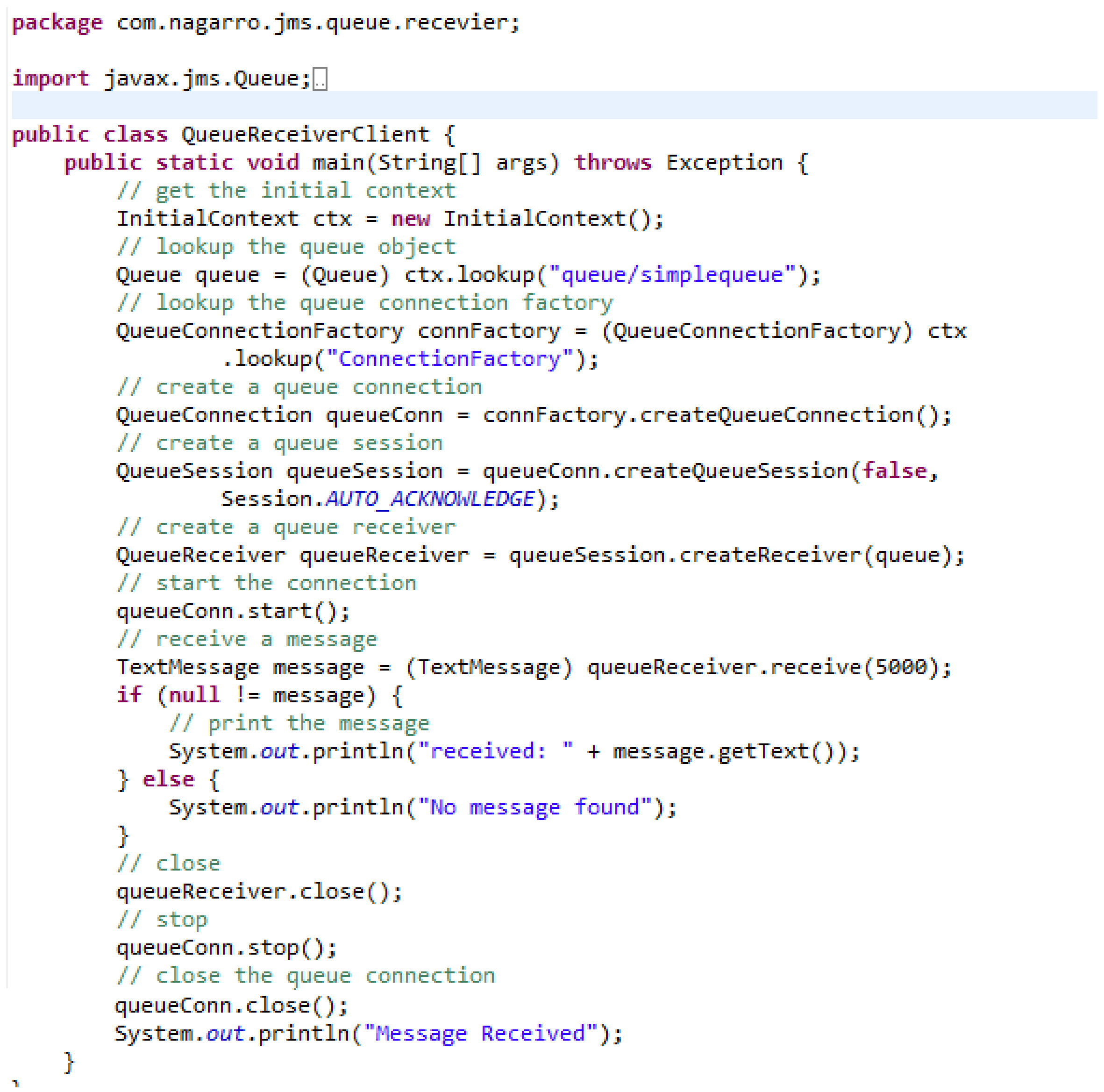
```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

5) Create a java file QueueSenderClient.java as shown below



```
package com.nagarro.jms.queue.sender;

import javax.jms.Queue;

public class QueueSenderClient {

    public static void main(String args[]) throws Exception {
        // get the initial context
        InitialContext ctx = new InitialContext();
        // lookup the queue connection factory
        QueueConnectionFactory connFactory = (QueueConnectionFactory) ctx
            .lookup("ConnectionFactory");
        // lookup the queue object
        Queue queue = (Queue) ctx.lookup("queue/simplequeue");
        // create a queue connection
        QueueConnection queueConn = connFactory.createQueueConnection();
        // start the connection
        queueConn.start();
        // create a queue session
        QueueSession queueSession = queueConn.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
        // create a queue sender
        QueueSender queueSender = queueSession.createSender(queue);
        //create text message
        TextMessage message = queueSession.createTextMessage("MessgeText");
        //send message
        queueSender.send(message);
        //close
        queueSender.close();
        //close session
        queueSession.close();
        //stop
        queueConn.stop();
        //close
        queueConn.close();
        System.out.println("Message Sent");
    }
```

6) Create a java file QueueReceiverClient.java as shown below

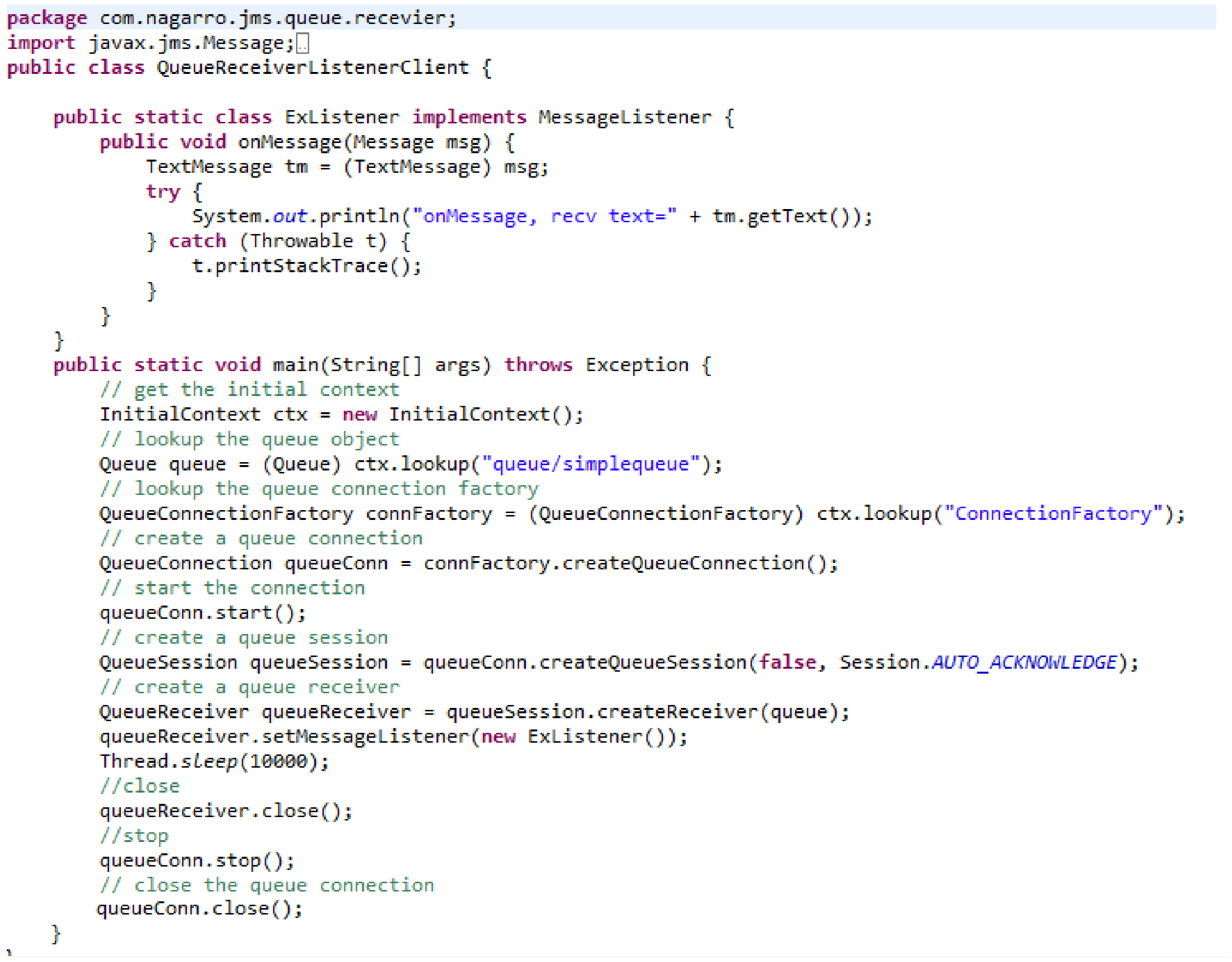


```
package com.nagarro.jms.queue.recevier;

import javax.jms.Queue;

public class QueueReceiverClient {
    public static void main(String[] args) throws Exception {
        // get the initial context
        InitialContext ctx = new InitialContext();
        // lookup the queue object
        Queue queue = (Queue) ctx.lookup("queue/simplequeue");
        // lookup the queue connection factory
        QueueConnectionFactory connFactory = (QueueConnectionFactory) ctx
                .lookup("ConnectionFactory");
        // create a queue connection
        QueueConnection queueConn = connFactory.createQueueConnection();
        // create a queue session
        QueueSession queueSession = queueConn.createQueueSession(false,
                Session.AUTO_ACKNOWLEDGE);
        // create a queue receiver
        QueueReceiver queueReceiver = queueSession.createReceiver(queue);
        // start the connection
        queueConn.start();
        // receive a message
        TextMessage message = (TextMessage) queueReceiver.receive(5000);
        if (null != message) {
            // print the message
            System.out.println("received: " + message.getText());
        } else {
            System.out.println("No message found");
        }
        // close
        queueReceiver.close();
        // stop
        queueConn.stop();
        // close the queue connection
        queueConn.close();
        System.out.println("Message Received");
    }
```

7) Run QueueSenderClient to send the message and QueueReceiverClient to receive it

Queue A-Synchronized messaging example

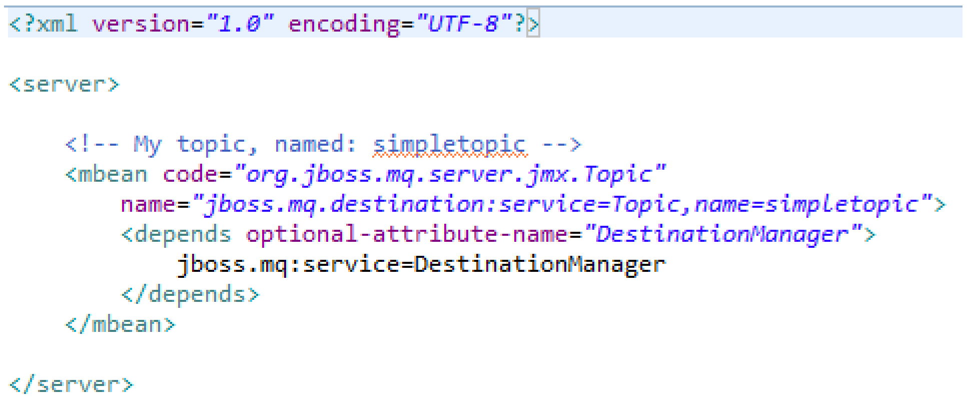
1) Everything else will remain same as the example above(Queue Synchronized messaging example) apart from the QueueReceiverClient.java
2) In place of QueueReceiverClient.java, create a file QueueReceiverListenerClient.java as shown below



```
package com.nagarro.jms.queue.recevier;
import javax.jms.Message;
public class QueueReceiverListenerClient {

    public static class ExListener implements MessageListener {
        public void onMessage(Message msg) {
            TextMessage tm = (TextMessage) msg;
            try {
                System.out.println("onMessage, recv text=" + tm.getText());
            } catch (Throwable t) {
                t.printStackTrace();
            }
        }
    }
    public static void main(String[] args) throws Exception {
        // get the initial context
        InitialContext ctx = new InitialContext();
        // lookup the queue object
        Queue queue = (Queue) ctx.lookup("queue/simplequeue");
        // lookup the queue connection factory
        QueueConnectionFactory connFactory = (QueueConnectionFactory) ctx.lookup("ConnectionFactory");
        // create a queue connection
        QueueConnection queueConn = connFactory.createQueueConnection();
        // start the connection
        queueConn.start();
        // create a queue session
        QueueSession queueSession = queueConn.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
        // create a queue receiver
        QueueReceiver queueReceiver = queueSession.createReceiver(queue);
        queueReceiver.setMessageListener(new ExListener());
        Thread.sleep(10000);
        //close
        queueReceiver.close();
        //stop
        queueConn.stop();
        // close the queue connection
        queueConn.close();
    }
```

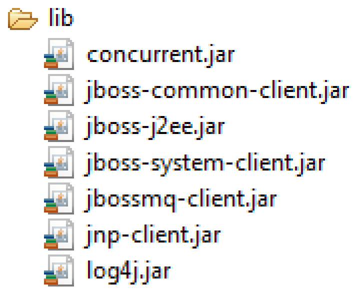
Topic Synchronized messaging example

1) Create a Java project in Eclipse and name it TopicSyncExample
2) Create a file simpletopic-destination-service.xml as shown below and paste it in the JBOSS_HOME/server/default/deploy directory. This will setup a queue with JNDI name queue/simpletopic in JBoss



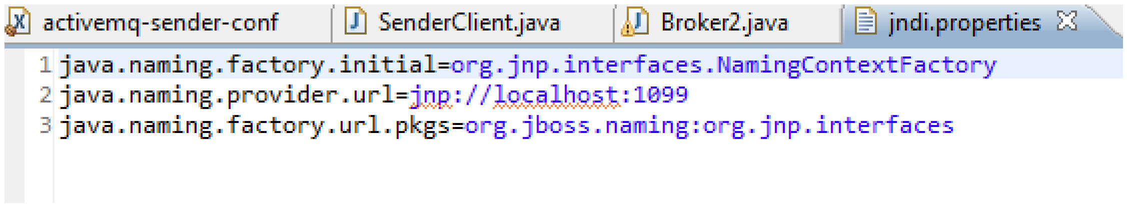
```xml
<?xml version="1.0" encoding="UTF-8"?>

<server>

    <!-- My topic, named: simpletopic -->
    <mbean code="org.jboss.mq.server.jmx.Topic"
        name="jboss.mq.destination:service=Topic,name=simpletopic">
        <depends optional-attribute-name="DestinationManager">
            jboss.mq:service=DestinationManager
        </depends>
    </mbean>

</server>
```

3) Copy the Jars shown below from the Jboss installation into the project's build path



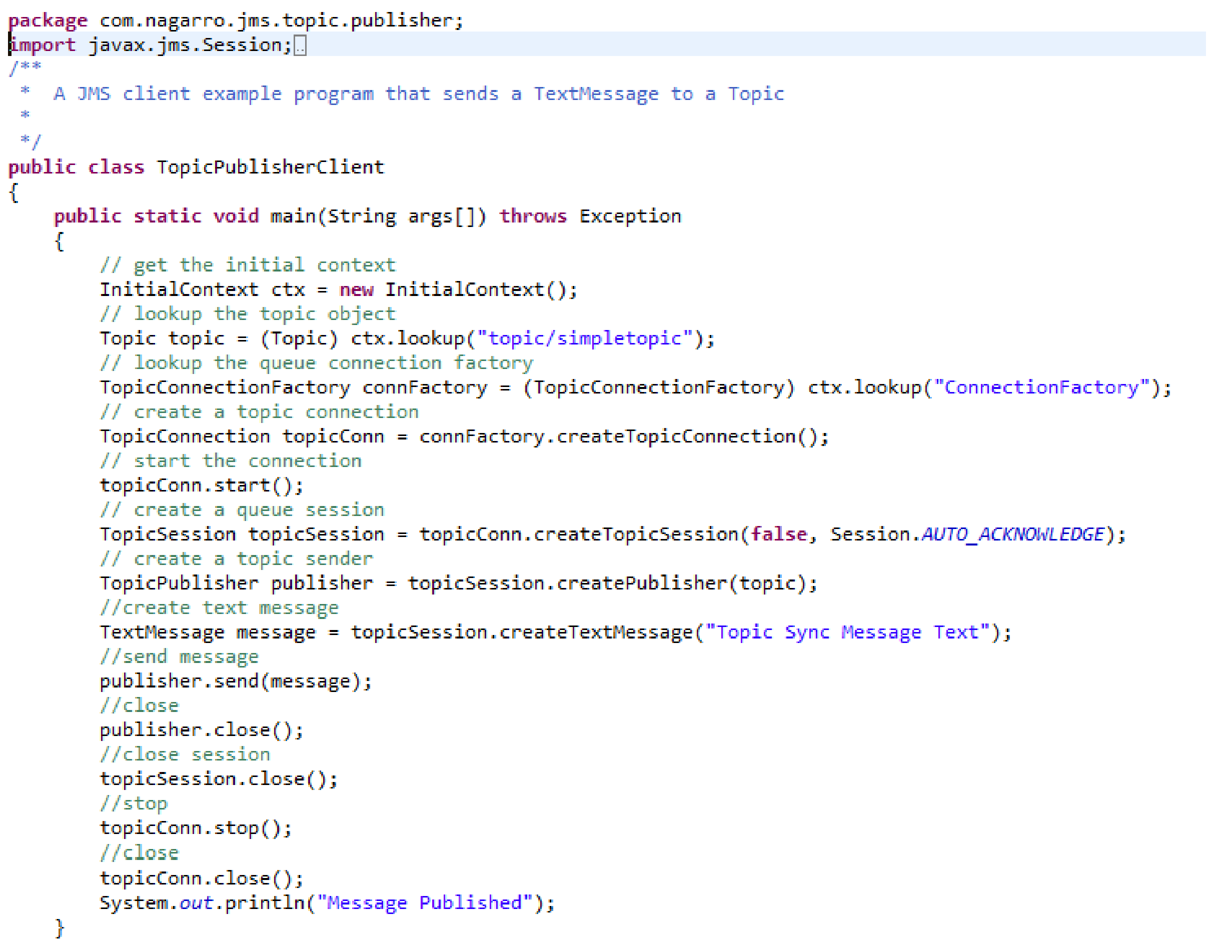
```
lib
    concurrent.jar
    jboss-common-client.jar
    jboss-j2ee.jar
    jboss-system-client.jar
    jbossmq-client.jar
    jnp-client.jar
    log4j.jar
```

4) Create a jndi.properties file as shown below in the src folder



```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
```

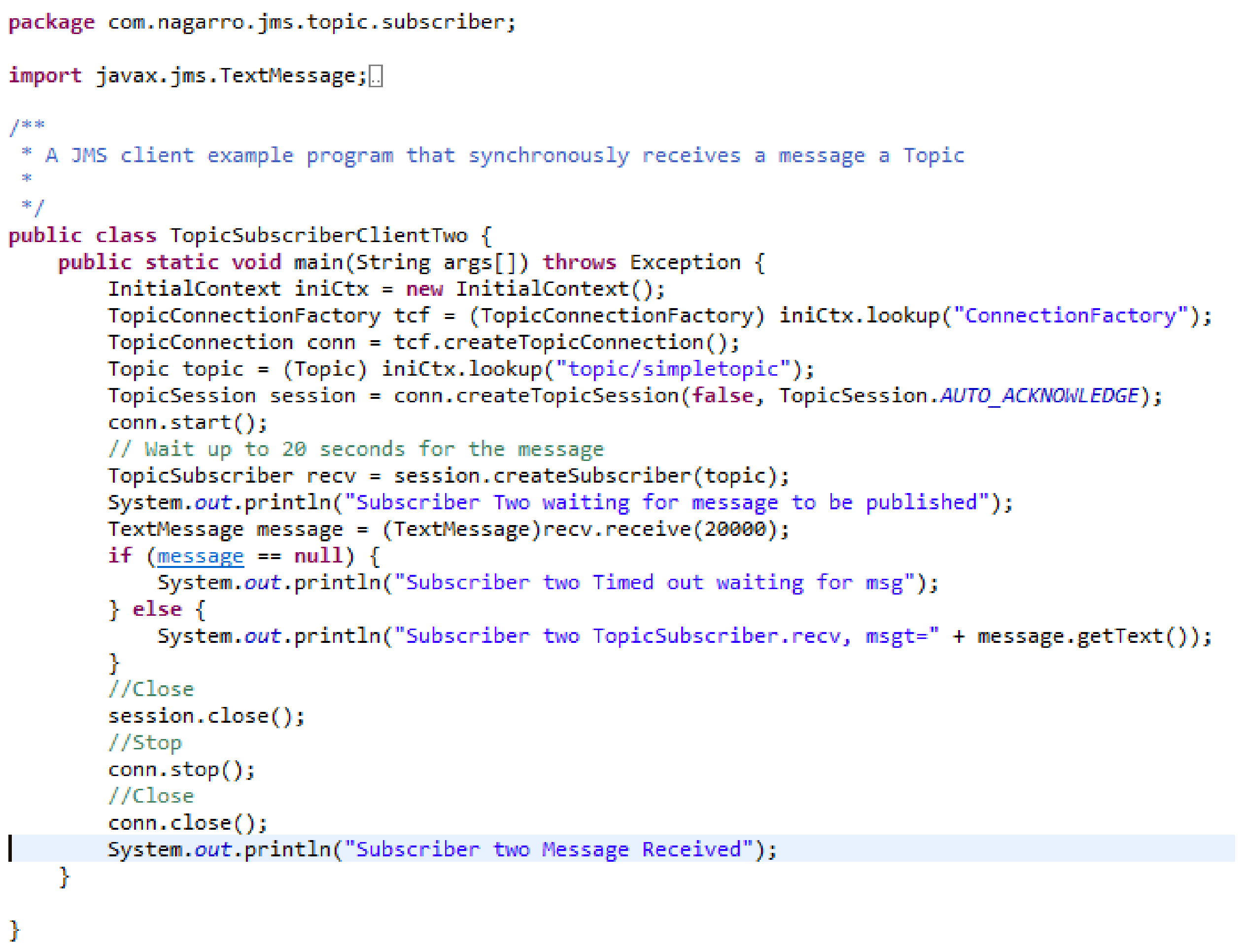
5) Create a java file TopicPublisherClient.java as shown below



```java
package com.nagarro.jms.topic.publisher;
import javax.jms.Session;
/**
 *  A JMS client example program that sends a TextMessage to a Topic
 *
 */
public class TopicPublisherClient
{
    public static void main(String args[]) throws Exception
    {
        // get the initial context
        InitialContext ctx = new InitialContext();
        // lookup the topic object
        Topic topic = (Topic) ctx.lookup("topic/simpletopic");
        // lookup the queue connection factory
        TopicConnectionFactory connFactory = (TopicConnectionFactory) ctx.lookup("ConnectionFactory");
        // create a topic connection
        TopicConnection topicConn = connFactory.createTopicConnection();
        // start the connection
        topicConn.start();
        // create a queue session
        TopicSession topicSession = topicConn.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        // create a topic sender
        TopicPublisher publisher = topicSession.createPublisher(topic);
        //create text message
        TextMessage message = topicSession.createTextMessage("Topic Sync Message Text");
        //send message
        publisher.send(message);
        //close
        publisher.close();
        //close session
        topicSession.close();
        //stop
        topicConn.stop();
        //close
        topicConn.close();
        System.out.println("Message Published");
    }
```

6) Create a java file TopicSubscriberClientOne.java and TopicSubscriberClientTwo.java as shown below

```
package com.nagarro.jms.topic.subscriber;

import javax.jms.TextMessage;

/**
 * A JMS client example program that synchronously receives a message a Topic
 *
 */
public class TopicSubscriberClientOne {

    public static void main(String args[]) throws Exception {
        InitialContext iniCtx = new InitialContext();
        TopicConnectionFactory tcf = (TopicConnectionFactory) iniCtx.lookup("ConnectionFactory");
        TopicConnection conn = tcf.createTopicConnection();
        Topic topic = (Topic) iniCtx.lookup("topic/simpletopic");
        TopicSession session = conn.createTopicSession(false, TopicSession.AUTO_ACKNOWLEDGE);
        conn.start();
        // Wait up to 20 seconds for the message
        TopicSubscriber recv = session.createSubscriber(topic);
        System.out.println("Subscriber one waiting for message to be published");
        TextMessage message = (TextMessage)recv.receive(20000);
        if (message == null) {
            System.out.println("Subscriber one Timed out waiting for msg");
        } else {
            System.out.println("Subscriber one TopicSubscriber.recv, msgt=" + message.getText());
        }
        //Close
        session.close();
        //Stop
        conn.stop();
        //Close
        conn.close();
        System.out.println("Subscriber one Message Received");
    }

}
```

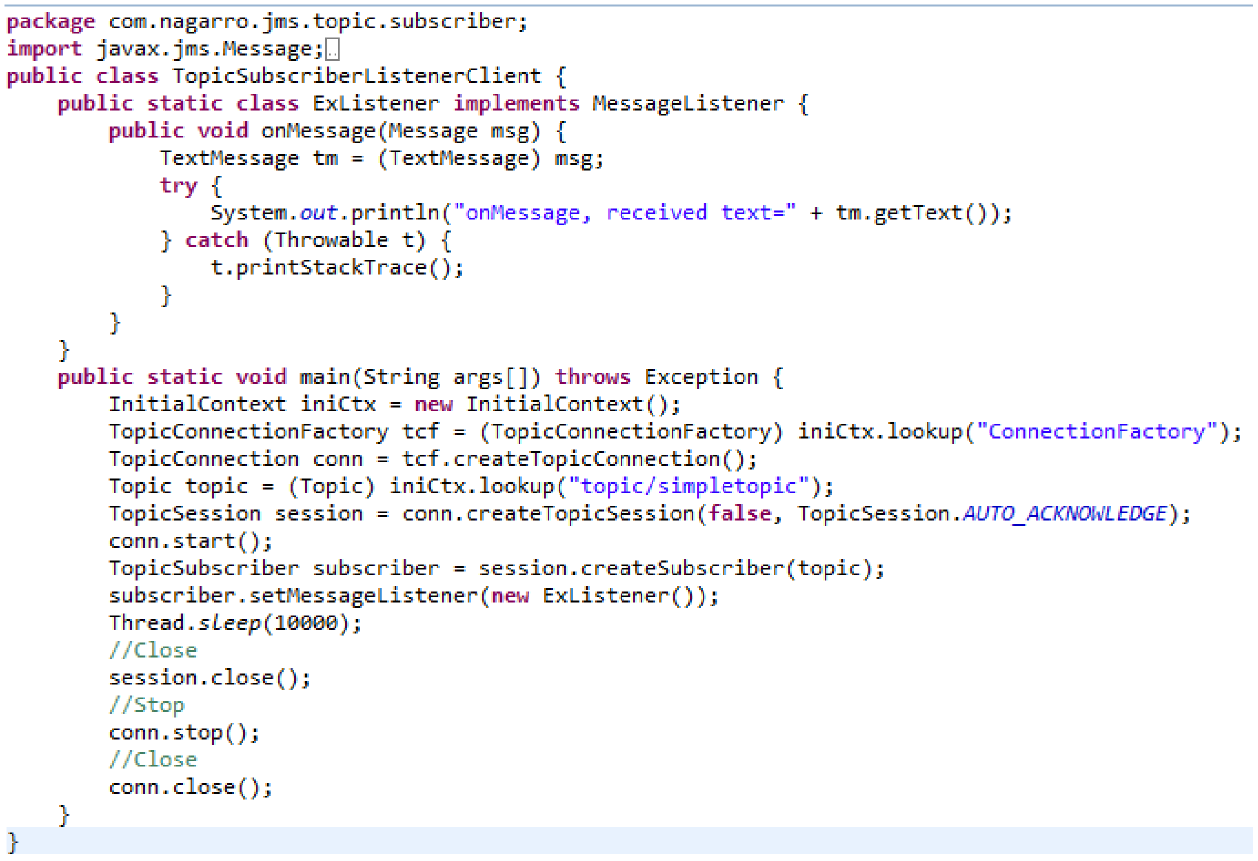


```
package com.nagarro.jms.topic.subscriber;

import javax.jms.TextMessage;

/**
 * A JMS client example program that synchronously receives a message a Topic
 *
 */
public class TopicSubscriberClientTwo {
    public static void main(String args[]) throws Exception {
        InitialContext iniCtx = new InitialContext();
        TopicConnectionFactory tcf = (TopicConnectionFactory) iniCtx.lookup("ConnectionFactory");
        TopicConnection conn = tcf.createTopicConnection();
        Topic topic = (Topic) iniCtx.lookup("topic/simpletopic");
        TopicSession session = conn.createTopicSession(false, TopicSession.AUTO_ACKNOWLEDGE);
        conn.start();
        // Wait up to 20 seconds for the message
        TopicSubscriber recv = session.createSubscriber(topic);
        System.out.println("Subscriber Two waiting for message to be published");
        TextMessage message = (TextMessage)recv.receive(20000);
        if (message == null) {
            System.out.println("Subscriber two Timed out waiting for msg");
        } else {
            System.out.println("Subscriber two TopicSubscriber.recv, msgt=" + message.getText());
        }
        //Close
        session.close();
        //Stop
        conn.stop();
        //Close
        conn.close();
        System.out.println("Subscriber two Message Received");
    }

}
```

7) Run TopicSubscriberClientOne and TopicSubscriberClientTwo first to wait for message to be published, run TopicPublisherClient to publish the message

Topic A-Synchronized messaging example

1) Everything else will remain same as the example above(Topic Synchronized messaging example) apart from the TopicSubscriberClientOne.java (remove TopicSubscriberClientTwo.java)
2) In place of TopicSubscriberClientOne.java, create a file TopicSubscriberListenerClient.java as shown below

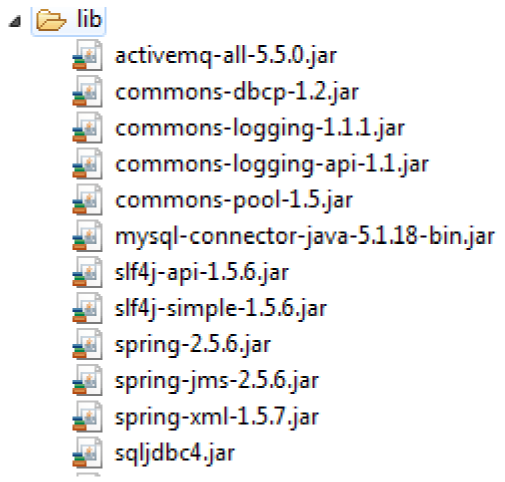


```java
package com.nagarro.jms.topic.subscriber;
import javax.jms.Message;
public class TopicSubscriberListenerClient {
    public static class ExListener implements MessageListener {
        public void onMessage(Message msg) {
            TextMessage tm = (TextMessage) msg;
            try {
                System.out.println("onMessage, received text=" + tm.getText());
            } catch (Throwable t) {
                t.printStackTrace();
            }
        }
    }
    public static void main(String args[]) throws Exception {
        InitialContext iniCtx = new InitialContext();
        TopicConnectionFactory tcf = (TopicConnectionFactory) iniCtx.lookup("ConnectionFactory");
        TopicConnection conn = tcf.createTopicConnection();
        Topic topic = (Topic) iniCtx.lookup("topic/simpletopic");
        TopicSession session = conn.createTopicSession(false, TopicSession.AUTO_ACKNOWLEDGE);
        conn.start();
        TopicSubscriber subscriber = session.createSubscriber(topic);
        subscriber.setMessageListener(new ExListener());
        Thread.sleep(10000);
        //Close
        session.close();
        //Stop
        conn.stop();
        //Close
        conn.close();
    }
}
```

3) Run TopicSubscriberListenerClient first to wait for message to be published, run TopicPublisherClient to publish the message

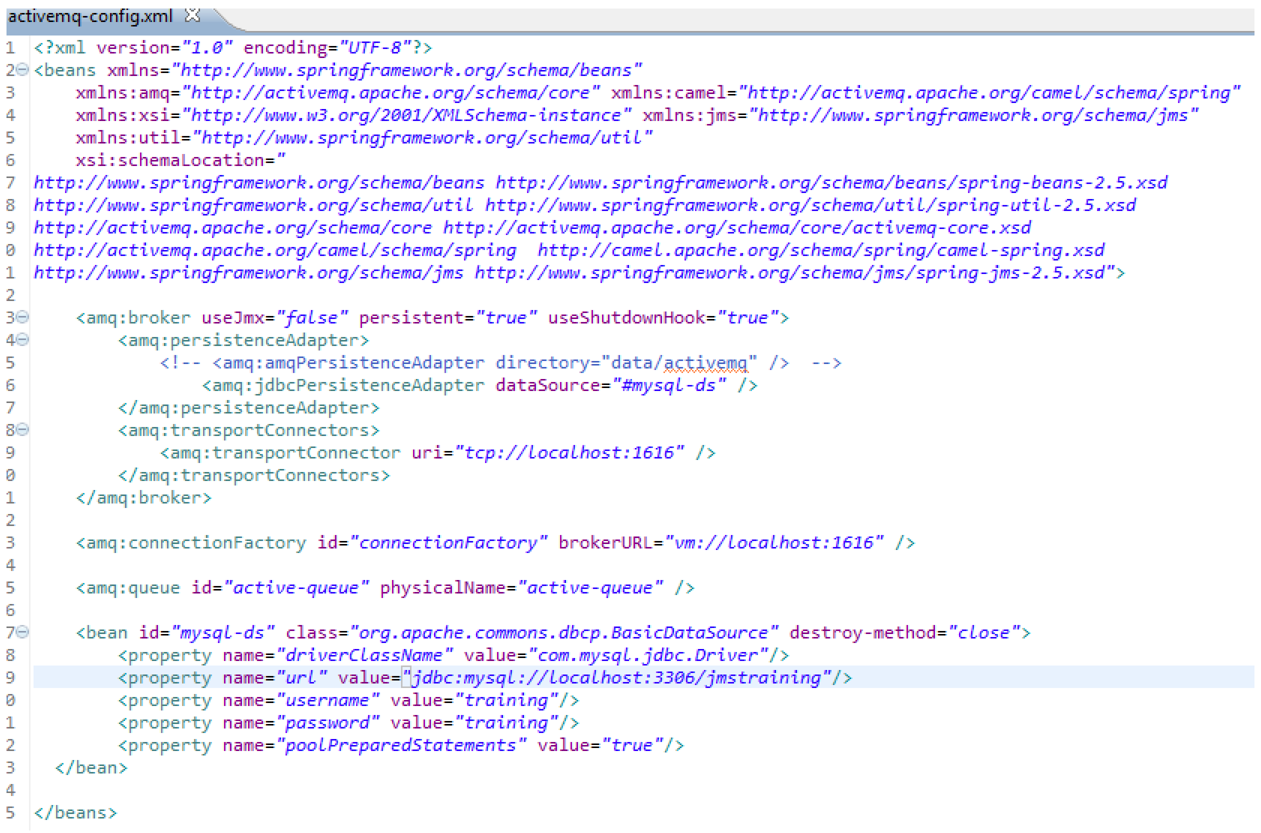
Queue with ActiveMQ/Spring example (With DB as persistence store)

1) Create a Java project and name it QueueActiveMQExample
2) Import the following Jars in the lib folder (as per ActiveMQ configuration section above)



```
lib
    activemq-all-5.5.0.jar
    commons-dbcp-1.2.jar
    commons-logging-1.1.1.jar
    commons-logging-api-1.1.jar
    commons-pool-1.5.jar
    mysql-connector-java-5.1.18-bin.jar
    slf4j-api-1.5.6.jar
    slf4j-simple-1.5.6.jar
    spring-2.5.6.jar
    spring-jms-2.5.6.jar
    spring-xml-1.5.7.jar
    sqljdbc4.jar
```

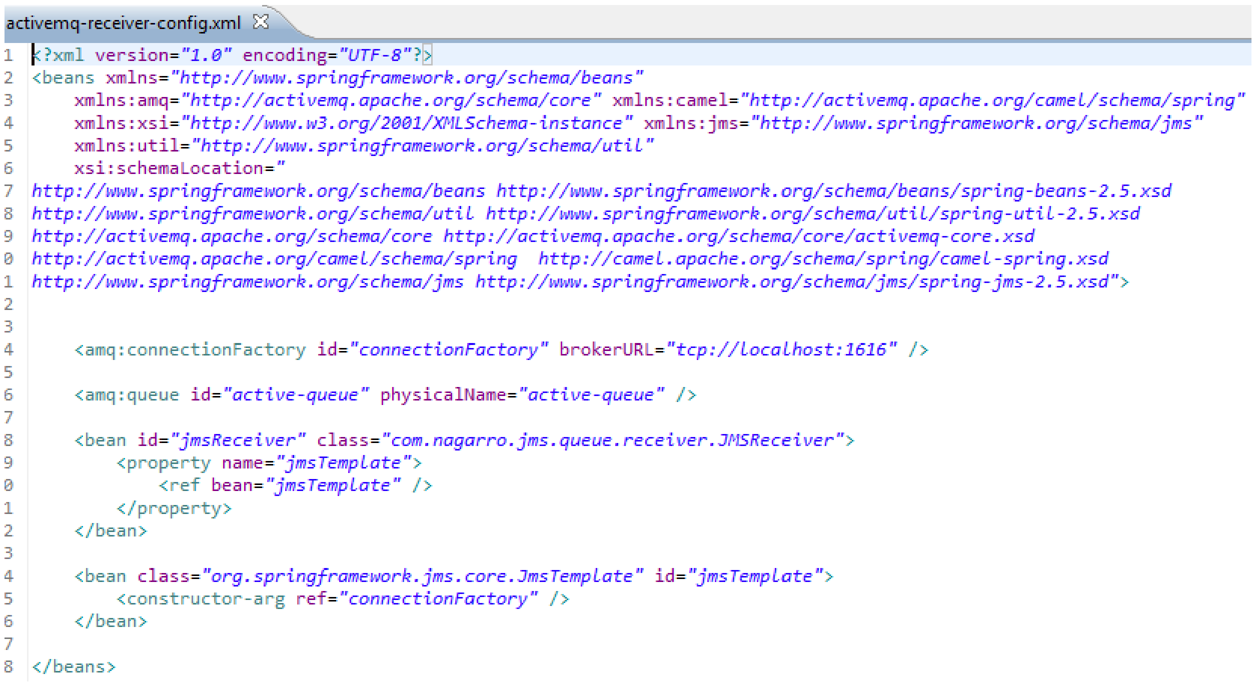
3) Create a config folder and create the 3 files(activemq-config.xml, activemq-receiver-config.xml, activemq-sender-config.xml) as shown below



activemq-config.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:broker useJmx="false" persistent="true" useShutdownHook="true">
        <amq:persistenceAdapter>
            <!-- <amq:amqPersistenceAdapter directory="data/activemq" />  -->
                <amq:jdbcPersistenceAdapter dataSource="#mysql-ds" />
        </amq:persistenceAdapter>
        <amq:transportConnectors>
            <amq:transportConnector uri="tcp://localhost:1616" />
        </amq:transportConnectors>
    </amq:broker>

    <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost:1616" />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url" value="jdbc:mysql://localhost:3306/jmstraining"/>
        <property name="username" value="training"/>
        <property name="password" value="training"/>
        <property name="poolPreparedStatements" value="true"/>
  </bean>

</beans>
```

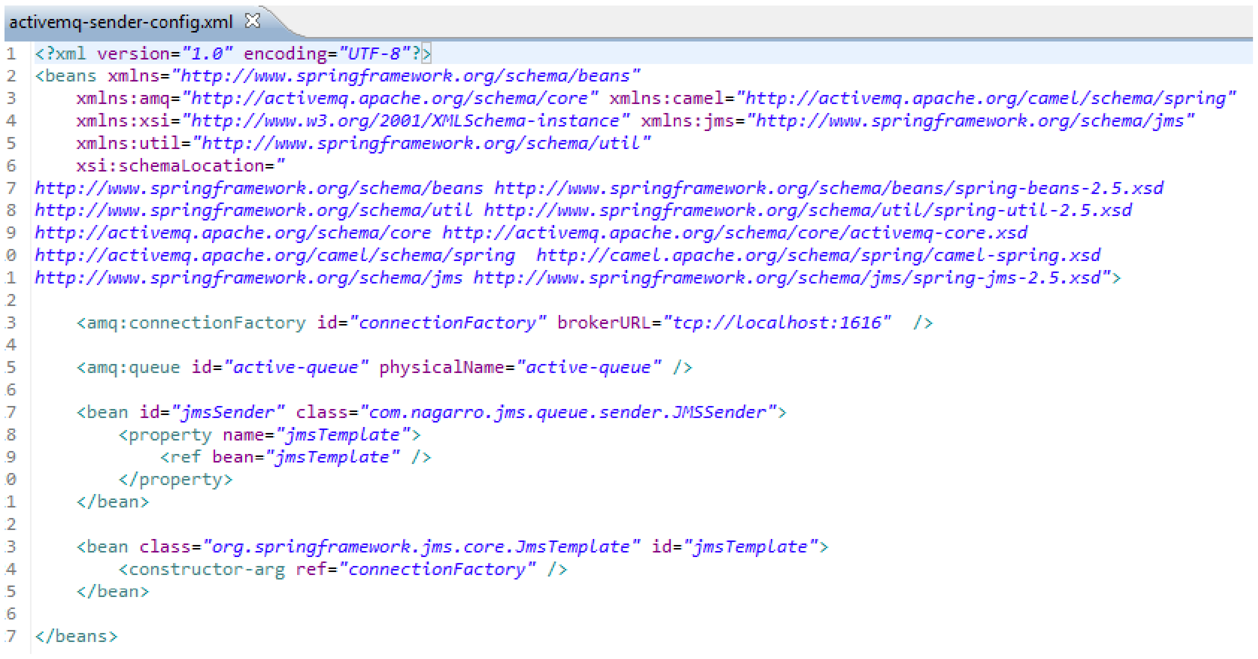
This file shown above configures the persistence store as our MySQL DB and sets up a queue and connectionFactory in ActiveMQ.



activemq-receiver-config.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">


    <amq:connectionFactory id="connectionFactory" brokerURL="tcp://localhost:1616" />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="jmsReceiver" class="com.nagarro.jms.queue.receiver.JMSReceiver">
        <property name="jmsTemplate">
            <ref bean="jmsTemplate" />
        </property>
    </bean>

    <bean class="org.springframework.jms.core.JmsTemplate" id="jmsTemplate">
        <constructor-arg ref="connectionFactory" />
    </bean>

</beans>
```

The file shown above configures a receiver, queue, Spring's jmsTemplate bean and our jmsReceiver bean.

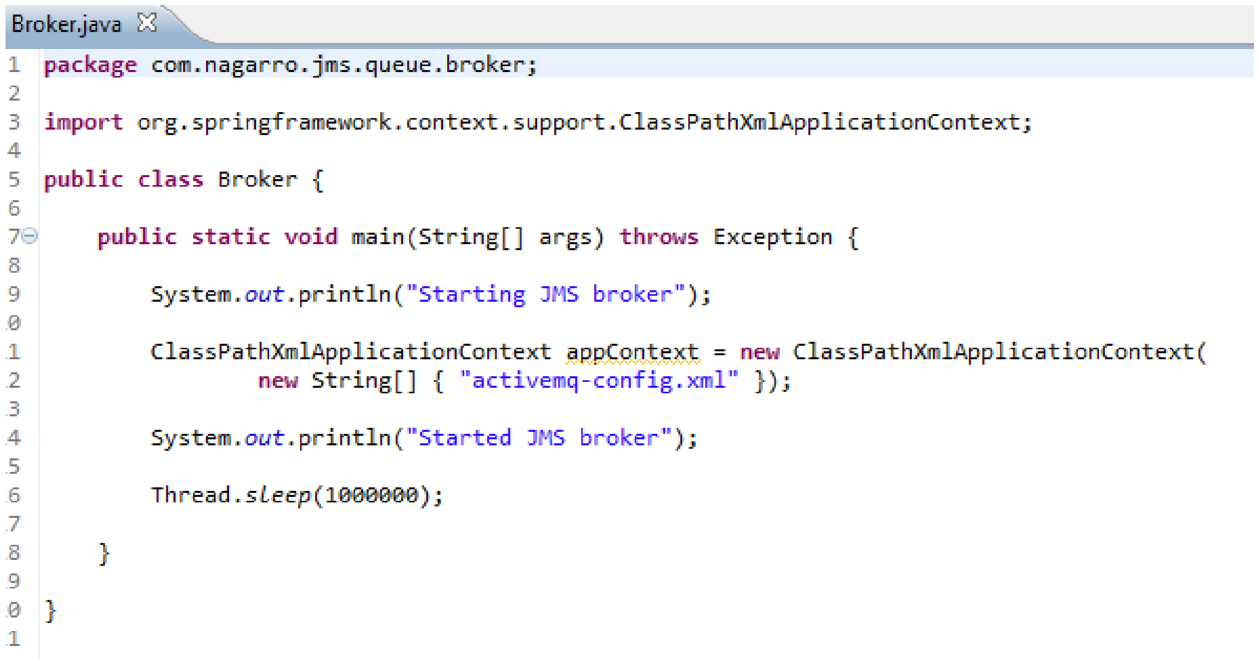


activemq-sender-config.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:connectionFactory id="connectionFactory" brokerURL="tcp://localhost:1616"  />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="jmsSender" class="com.nagarro.jms.queue.sender.JMSSender">
        <property name="jmsTemplate">
            <ref bean="jmsTemplate" />
        </property>
    </bean>

    <bean class="org.springframework.jms.core.JmsTemplate" id="jmsTemplate">
        <constructor-arg ref="connectionFactory" />
    </bean>

</beans>
```

The file shown above configures a sender, queue, Spring's jmsTemplate bean and our jmsReceiver bean.

4) Create a file Broker.java as shown below to start the ActiveMQ broker programmatically



Broker.java

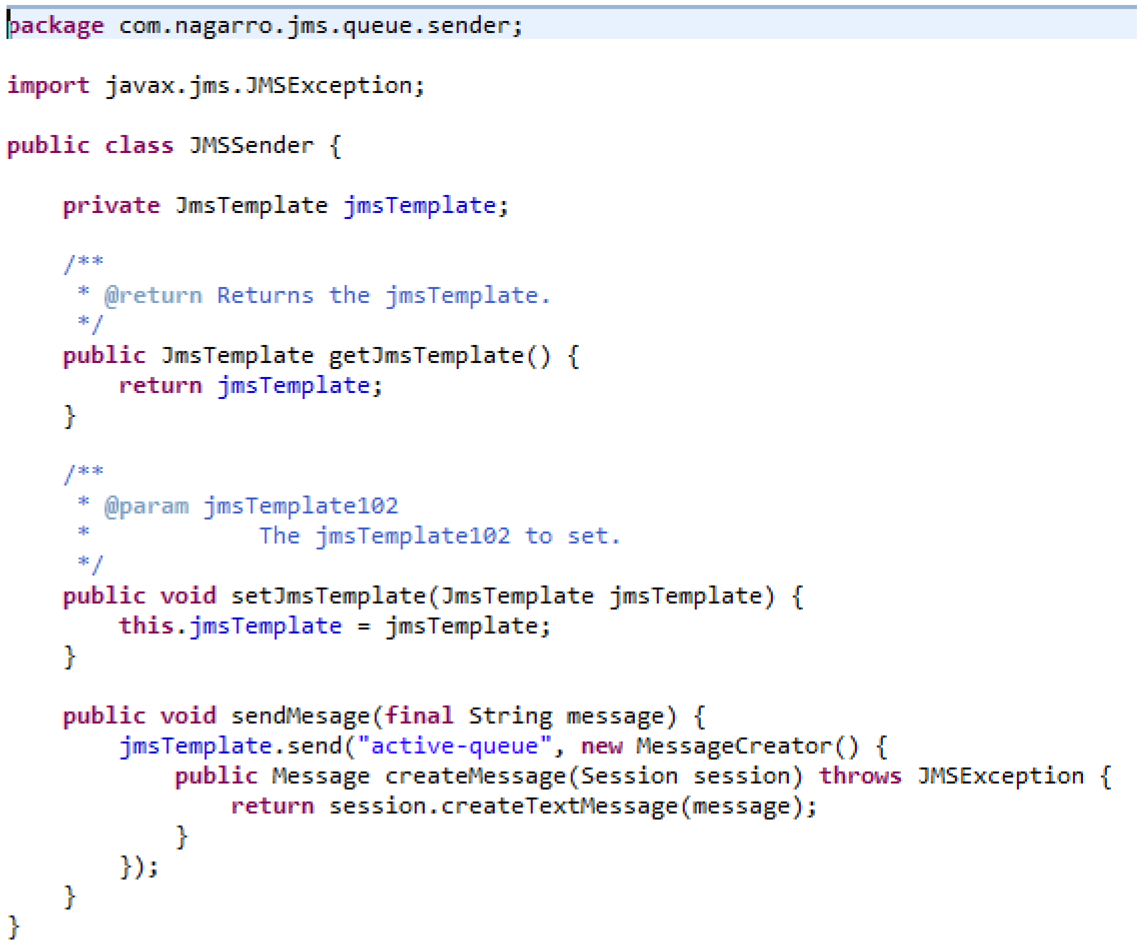
```java
package com.nagarro.jms.queue.broker;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Broker {

    public static void main(String[] args) throws Exception {

        System.out.println("Starting JMS broker");

        ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                new String[] { "activemq-config.xml" });

        System.out.println("Started JMS broker");

        Thread.sleep(1000000);

    }

}
```

5) Create a file JMSReceiver.java as shown below

```java
package com.nagarro.jms.queue.receiver;

import javax.jms.Message;

public class JMSReceiver {
    private JmsTemplate jmsTemplate;
    /**
     * @return Returns the jmsTemplate.
     */
    public JmsTemplate getJmsTemplate() {
        return jmsTemplate;
    }

    /**
     * @param jmsTemplate
     *            The jmsTemplate to set.
     */
    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    public void processMessage() {
        Message msg = jmsTemplate.receive("active-queue");
        try {
            TextMessage textMessage = (TextMessage) msg;
            if (msg != null) {
                System.out.println(" Message Received -->" + textMessage.getText());
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}
```

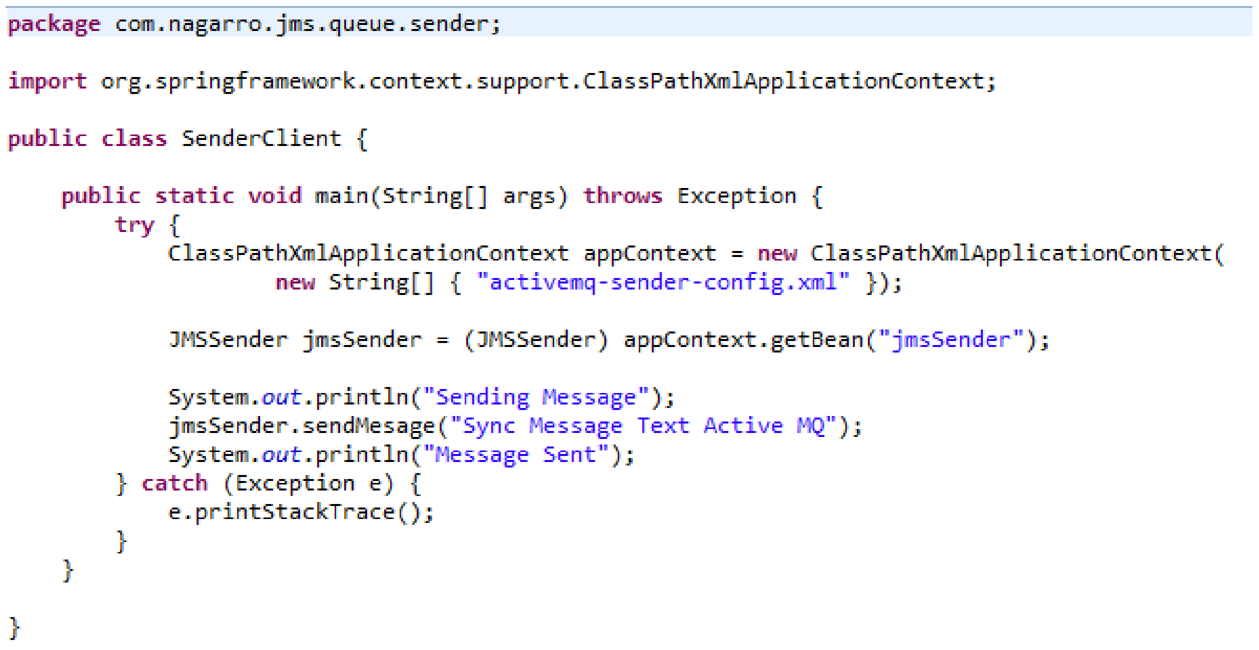
6) Create a file JMSSender.java as shown below

A s



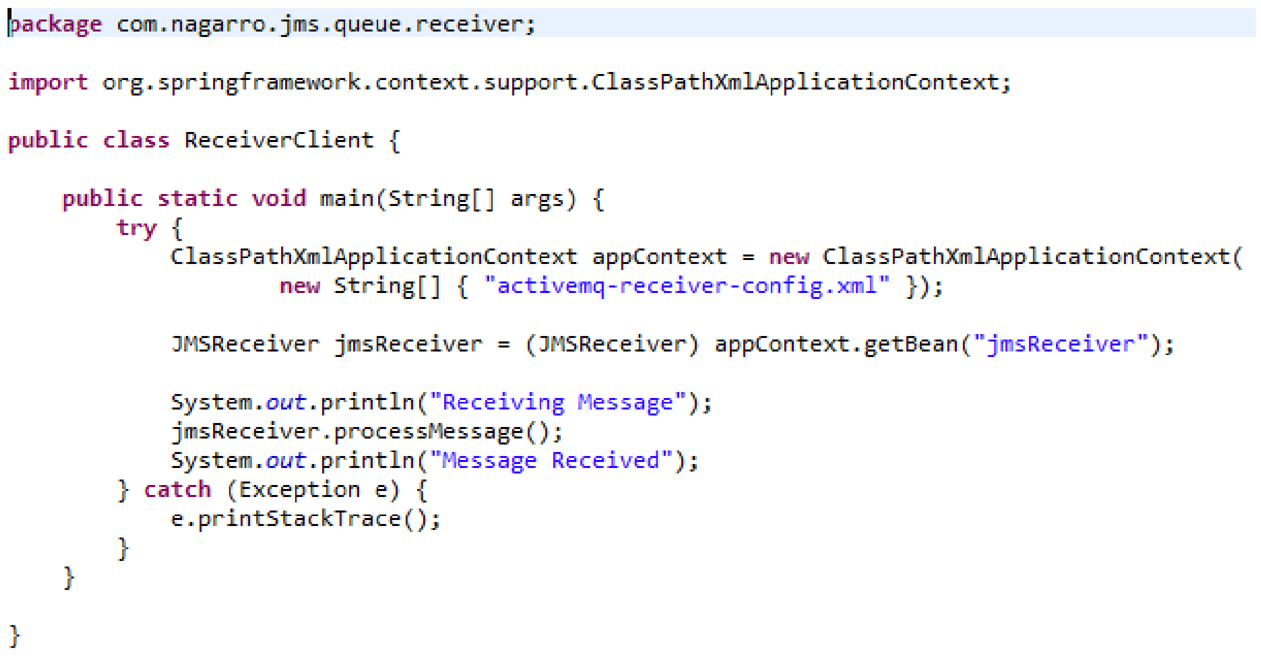
```java
package com.nagarro.jms.queue.sender;

import javax.jms.JMSException;

public class JMSSender {

    private JmsTemplate jmsTemplate;

    /**
     * @return Returns the jmsTemplate.
     */
    public JmsTemplate getJmsTemplate() {
        return jmsTemplate;
    }

    /**
     * @param jmsTemplate102
     *            The jmsTemplate102 to set.
     */
    public void setJmsTemplate(JmsTemplate jmsTemplate) {
        this.jmsTemplate = jmsTemplate;
    }

    public void sendMesage(final String message) {
        jmsTemplate.send("active-queue", new MessageCreator() {
            public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage(message);
            }
        });
    }
}
```

7) Create SenderClient.java as shown below:



```java
package com.nagarro.jms.queue.sender;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SenderClient {

    public static void main(String[] args) throws Exception {
        try {
            ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                    new String[] { "activemq-sender-config.xml" });

            JMSSender jmsSender = (JMSSender) appContext.getBean("jmsSender");

            System.out.println("Sending Message");
            jmsSender.sendMesage("Sync Message Text Active MQ");
            System.out.println("Message Sent");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

8) Create ReceiverClient.java as shown below



```java
package com.nagarro.jms.queue.receiver;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ReceiverClient {

    public static void main(String[] args) {
        try {
            ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                    new String[] { "activemq-receiver-config.xml" });

            JMSReceiver jmsReceiver = (JMSReceiver) appContext.getBean("jmsReceiver");

            System.out.println("Receiving Message");
            jmsReceiver.processMessage();
            System.out.println("Message Received");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

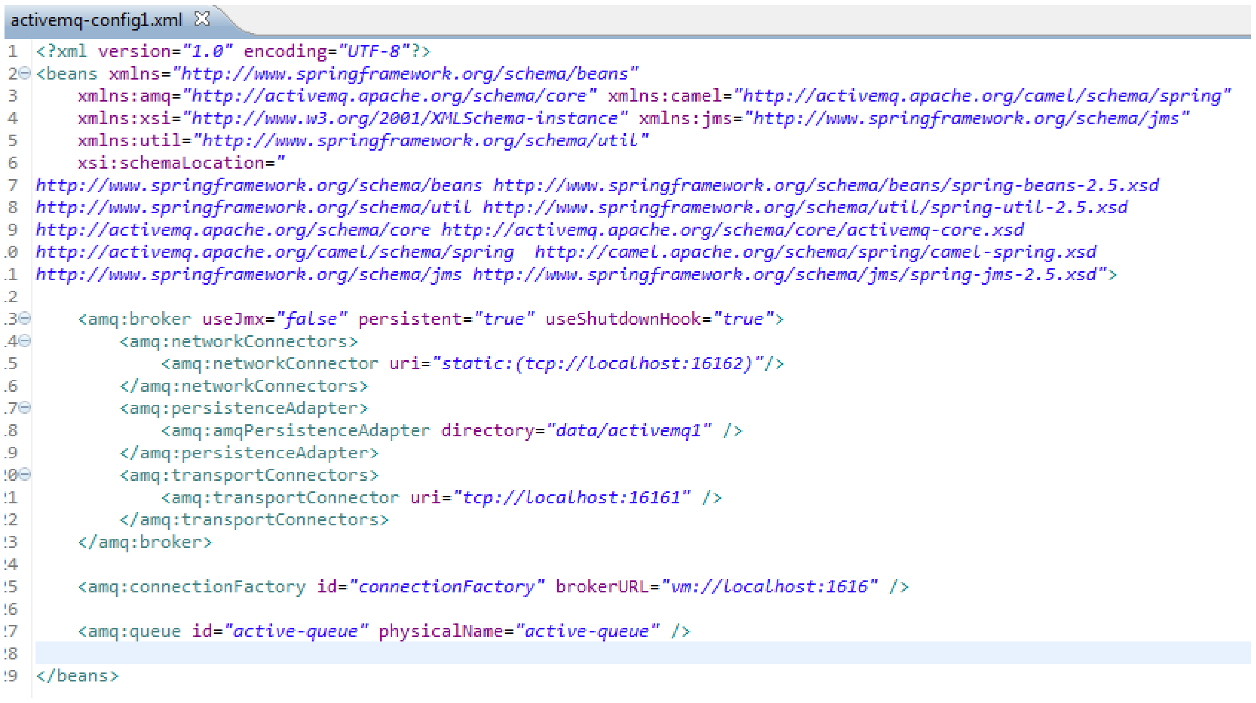
9) Run Broker class

10) Run SenderClient to send a message and ReceiverClient to receive a message

ActiveMQ clustering Example (with File as Persistence store)

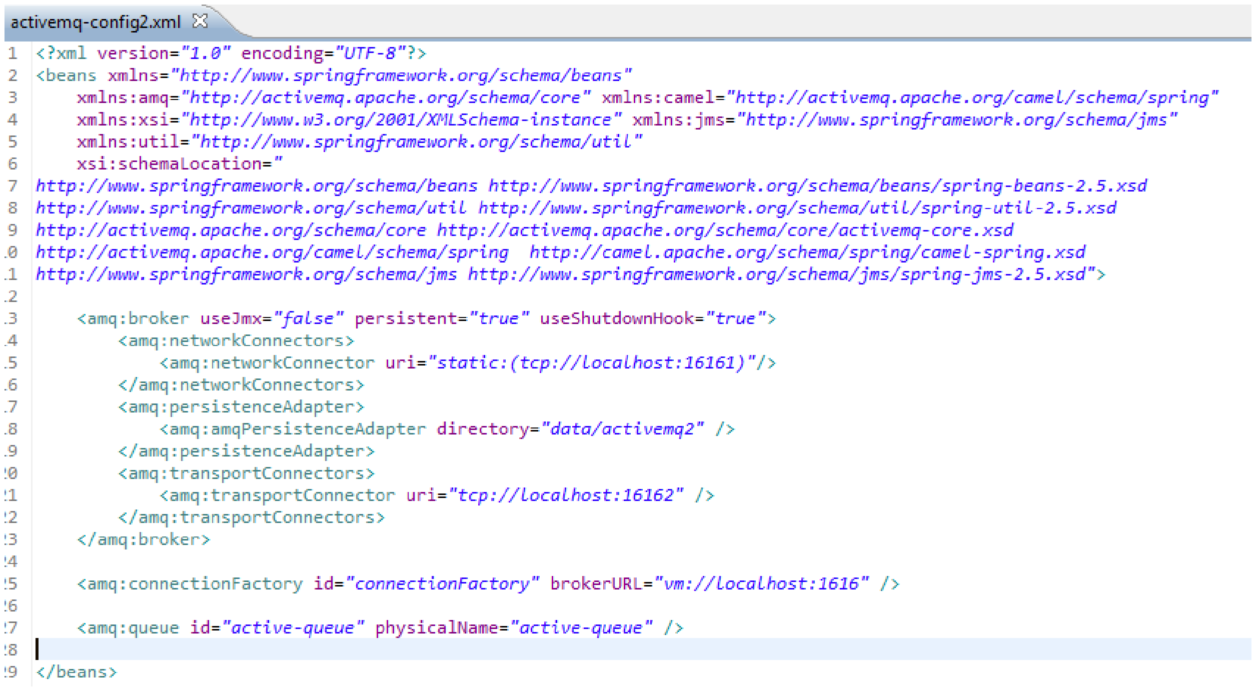
In this example, we will start 2 brokers in a cluster, send message to broker1 and receive from broker2.

1) Create a Java project and name it ActiveMQClusterExample
2) Copy the Spring and ActiveMQ jars as done in previous example
3) Create a config folder and create the 3 files(activemq-config1.xml, activemq-config2.xml, activemq-receiver-config1.xml, activemq-sender-config2.xml) as shown below


activemq-config1.xml

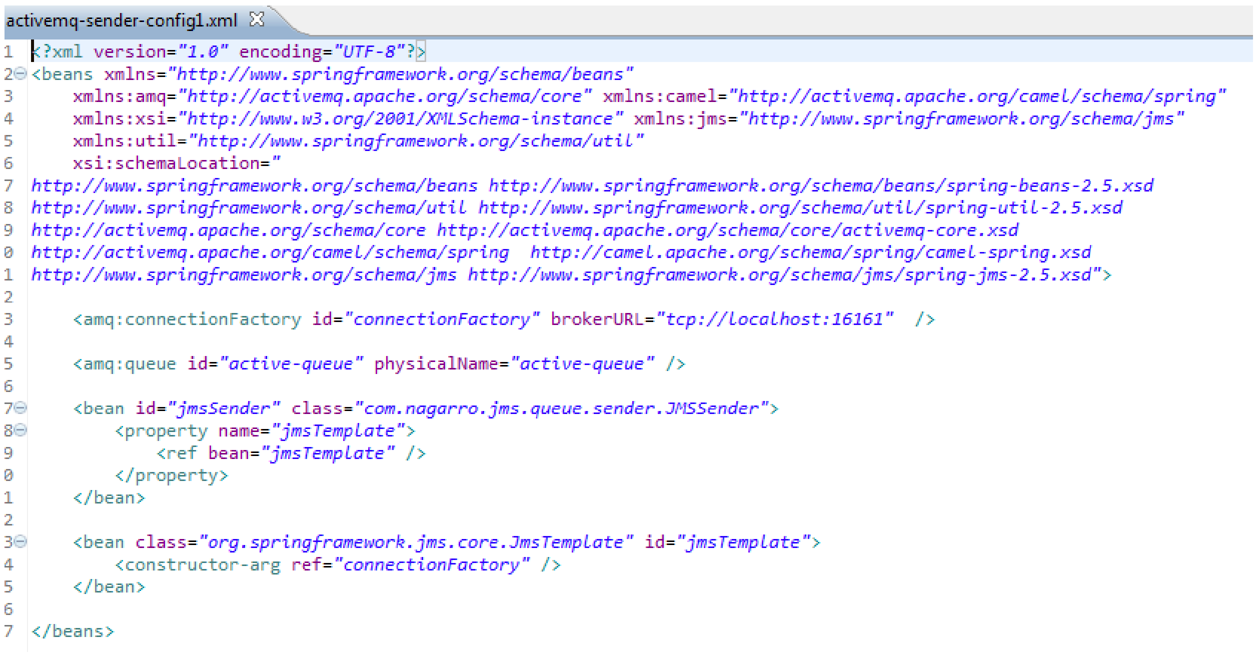
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:broker useJmx="false" persistent="true" useShutdownHook="true">
        <amq:networkConnectors>
            <amq:networkConnector uri="static:(tcp://localhost:16162)"/>
        </amq:networkConnectors>
        <amq:persistenceAdapter>
            <amq:amqPersistenceAdapter directory="data/activemq1" />
        </amq:persistenceAdapter>
        <amq:transportConnectors>
            <amq:transportConnector uri="tcp://localhost:16161" />
        </amq:transportConnectors>
    </amq:broker>

    <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost:1616" />

    <amq:queue id="active-queue" physicalName="active-queue" />

</beans>
```

The file above is configuration for broker1, notice the amq:networkConnector tag here, this will tell broker 1 to connect with broker2. It will also setup the file persistence store using amq:persistenceAdapter tag.



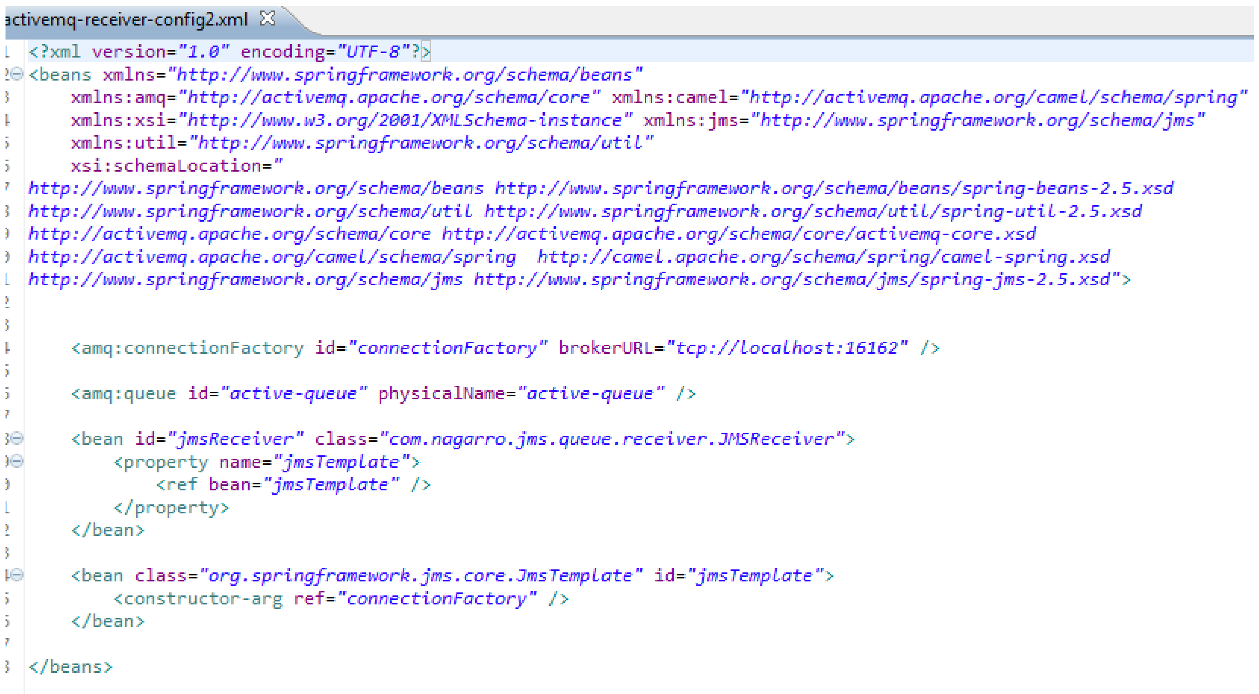
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:broker useJmx="false" persistent="true" useShutdownHook="true">
        <amq:networkConnectors>
            <amq:networkConnector uri="static:(tcp://localhost:16161)"/>
        </amq:networkConnectors>
        <amq:persistenceAdapter>
            <amq:amqPersistenceAdapter directory="data/activemq2" />
        </amq:persistenceAdapter>
        <amq:transportConnectors>
            <amq:transportConnector uri="tcp://localhost:16162" />
        </amq:transportConnectors>
    </amq:broker>

    <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost:1616" />

    <amq:queue id="active-queue" physicalName="active-queue" />

</beans>
```

The file above configures broker2 and using the networkConnector, creates a cluster with broker1.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:connectionFactory id="connectionFactory" brokerURL="tcp://localhost:16161"  />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="jmsSender" class="com.nagarro.jms.queue.sender.JMSSender">
        <property name="jmsTemplate">
            <ref bean="jmsTemplate" />
        </property>
    </bean>

    <bean class="org.springframework.jms.core.JmsTemplate" id="jmsTemplate">
        <constructor-arg ref="connectionFactory" />
    </bean>

</beans>
```

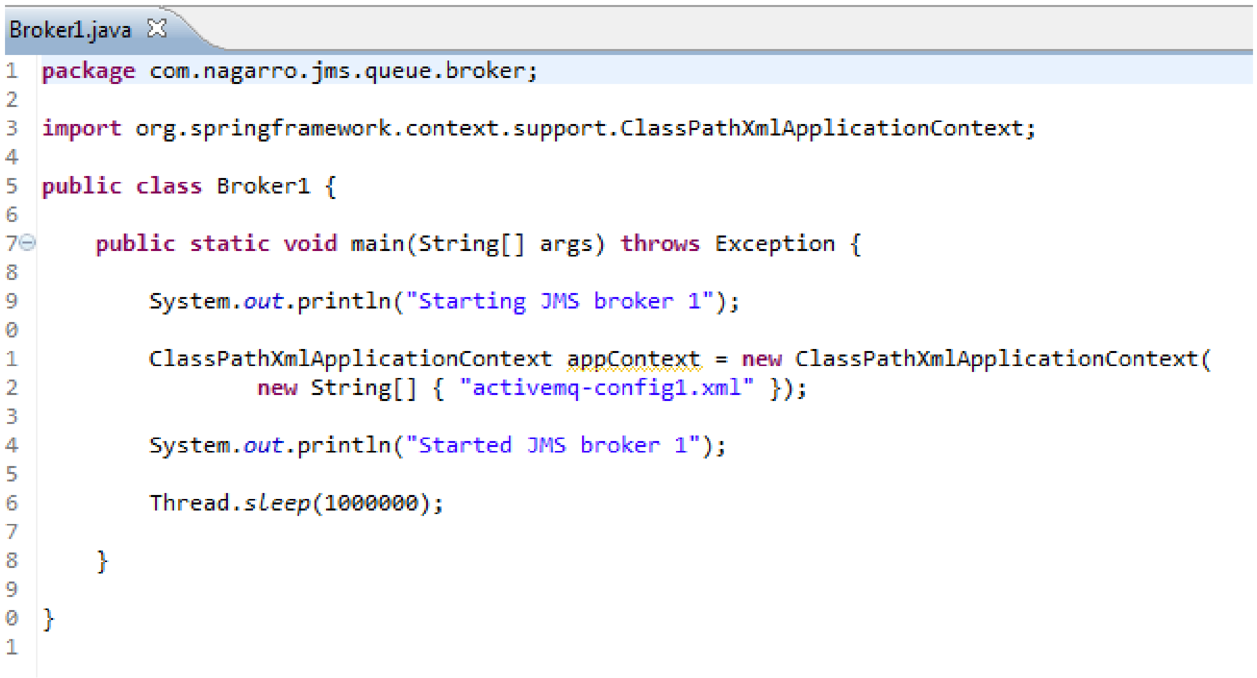
The file above will setup a sender to send message to broker1.


ctivemq-receiver-config2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">


    <amq:connectionFactory id="connectionFactory" brokerURL="tcp://localhost:16162" />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="jmsReceiver" class="com.nagarro.jms.queue.receiver.JMSReceiver">
        <property name="jmsTemplate">
            <ref bean="jmsTemplate" />
        </property>
    </bean>

    <bean class="org.springframework.jms.core.JmsTemplate" id="jmsTemplate">
        <constructor-arg ref="connectionFactory" />
    </bean>

</beans>
```

The file above will setup a receiver to receive from broker1.

4) Create Broker1.java as shown below:


Broker1.java

```java
package com.nagarro.jms.queue.broker;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Broker1 {

    public static void main(String[] args) throws Exception {

        System.out.println("Starting JMS broker 1");

        ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                new String[] { "activemq-config1.xml" });

        System.out.println("Started JMS broker 1");

        Thread.sleep(1000000);

    }

}
```

5) Create Broker2.java as shown below

```java
package com.nagarro.jms.queue.broker;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Broker2 {

    public static void main(String[] args) throws Exception {

        System.out.println("Starting JMS broker 2");

        ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                new String[] { "activemq-config2.xml" });

        System.out.println("Started JMS broker 2");

        Thread.sleep(1000000);

    }

}
```

6) Create JMSReceiver.java and JMSSender.java similar to how it was done in previous example
7) Create SenderClient.java as shown below

```java
package com.nagarro.jms.queue.sender;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SenderClient {

    public static void main(String[] args) throws Exception {
        try {
            ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                    new String[] { "activemq-sender-config1.xml" });

            JMSSender jmsSender = (JMSSender) appContext.getBean("jmsSender");

            System.out.println("Sending Message");
            jmsSender.sendMesage("Hello clustering Example");
            System.out.println("Message Sent");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

8) Create ReceiverClient.java as shown below

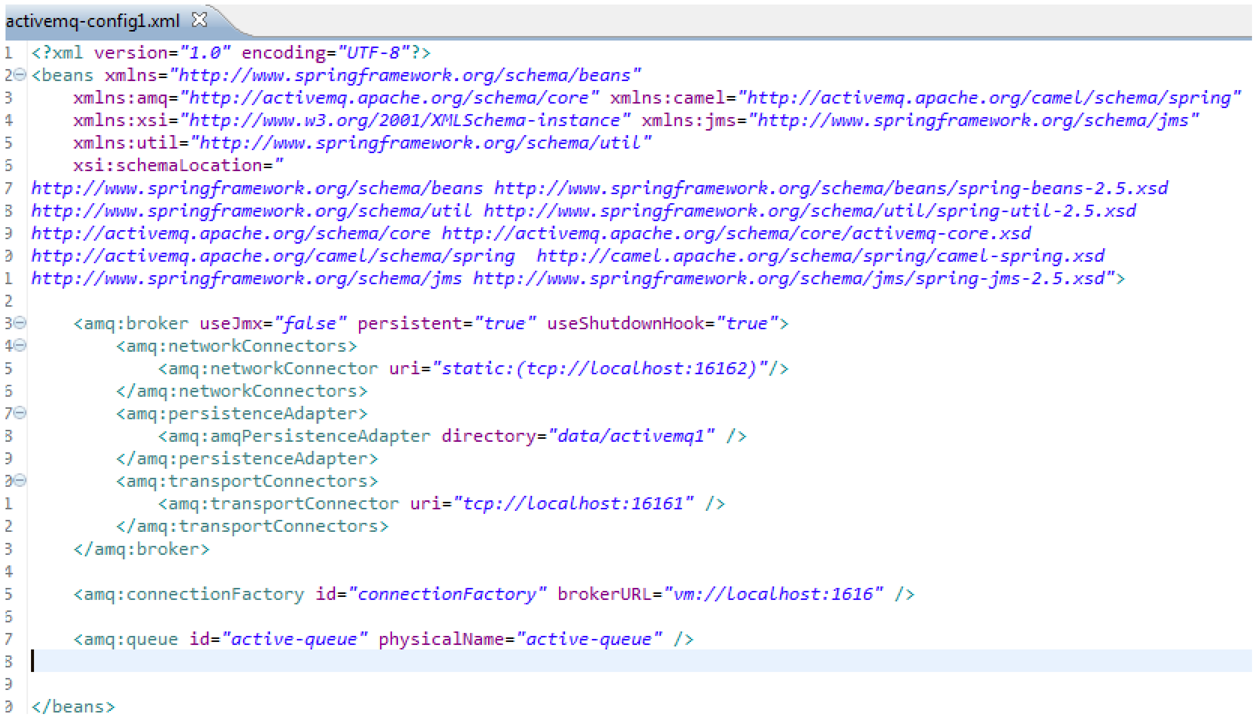
```java
package com.nagarro.jms.queue.receiver;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ReceiverClient {

    public static void main(String[] args) {
        try {
            ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                    new String[] { "activemq-receiver-config2.xml" });

            JMSReceiver jmsReceiver = (JMSReceiver) appContext.getBean("jmsReceiver");

            System.out.println("Receiving Message");
            jmsReceiver.processMessage();
            System.out.println("Message Received");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

9) Run Broker1 and Broker2
10) Run SenderClient to send message to broker1
11) Run ReceiverClient to receive message from broker2

ActiveMQ failover Example (with File as Persistence store)

This will demonstrate the failover feature provided by ActiveMQ.

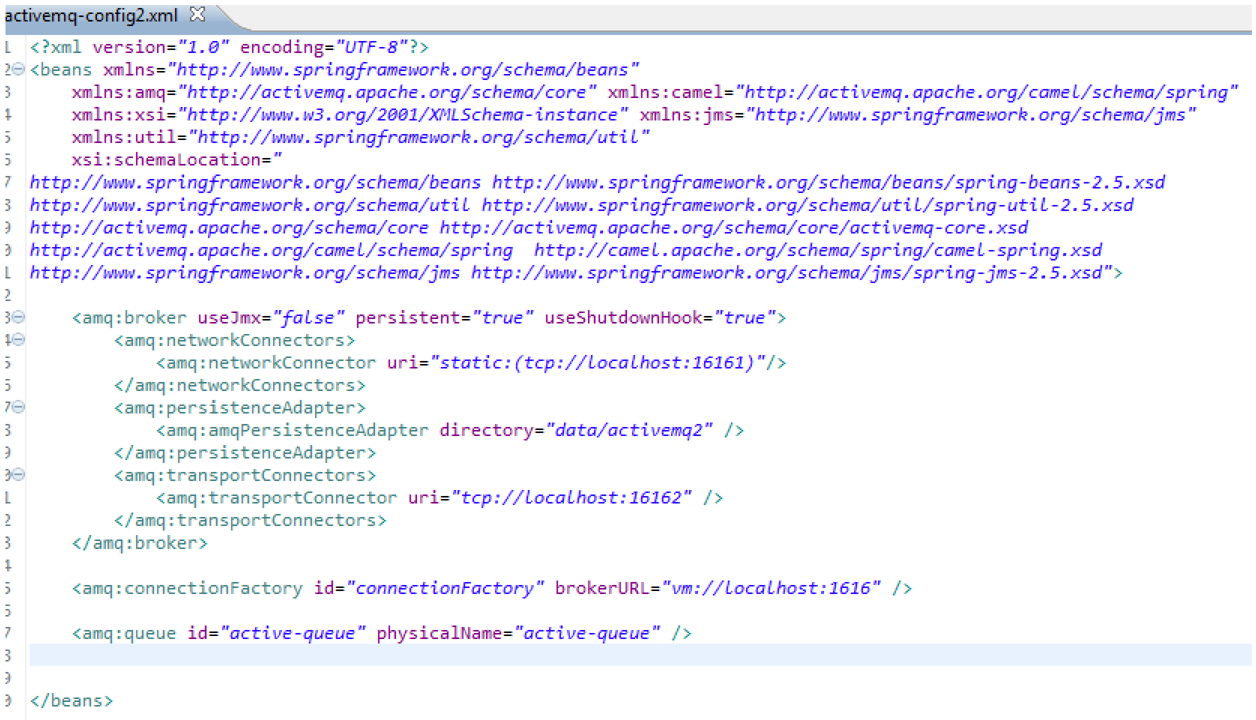
1) Create a Java project and name it ActiveMQFailoverClusterExample
2) Create a config folder and create the 3 files(activemq-config1.xml, activemq-config1.xml, activemq-receiver-config2.xml, activemq-sender-config1.xml) as shown below



activemq-config1.xml

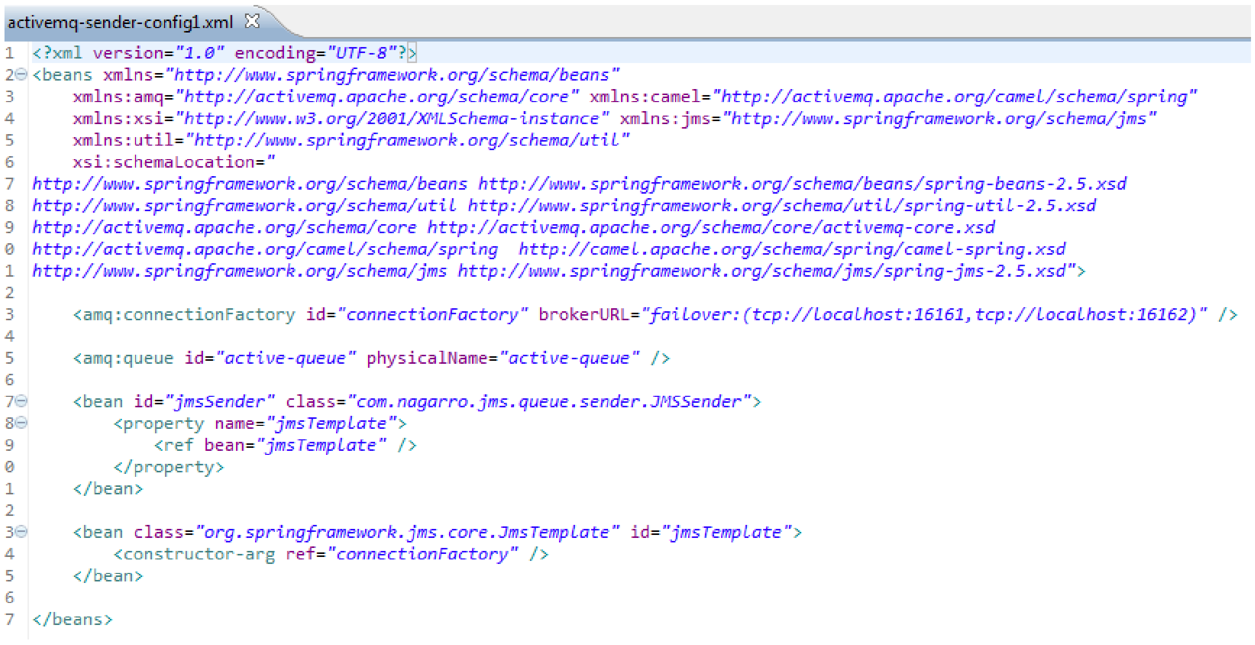
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:broker useJmx="false" persistent="true" useShutdownHook="true">
        <amq:networkConnectors>
            <amq:networkConnector uri="static:(tcp://localhost:16162)"/>
        </amq:networkConnectors>
        <amq:persistenceAdapter>
            <amq:amqPersistenceAdapter directory="data/activemq1" />
        </amq:persistenceAdapter>
        <amq:transportConnectors>
            <amq:transportConnector uri="tcp://localhost:16161" />
        </amq:transportConnectors>
    </amq:broker>

    <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost:1616" />

    <amq:queue id="active-queue" physicalName="active-queue" />


</beans>
```

Notice the networkConnector tag similar to the clustering example above.



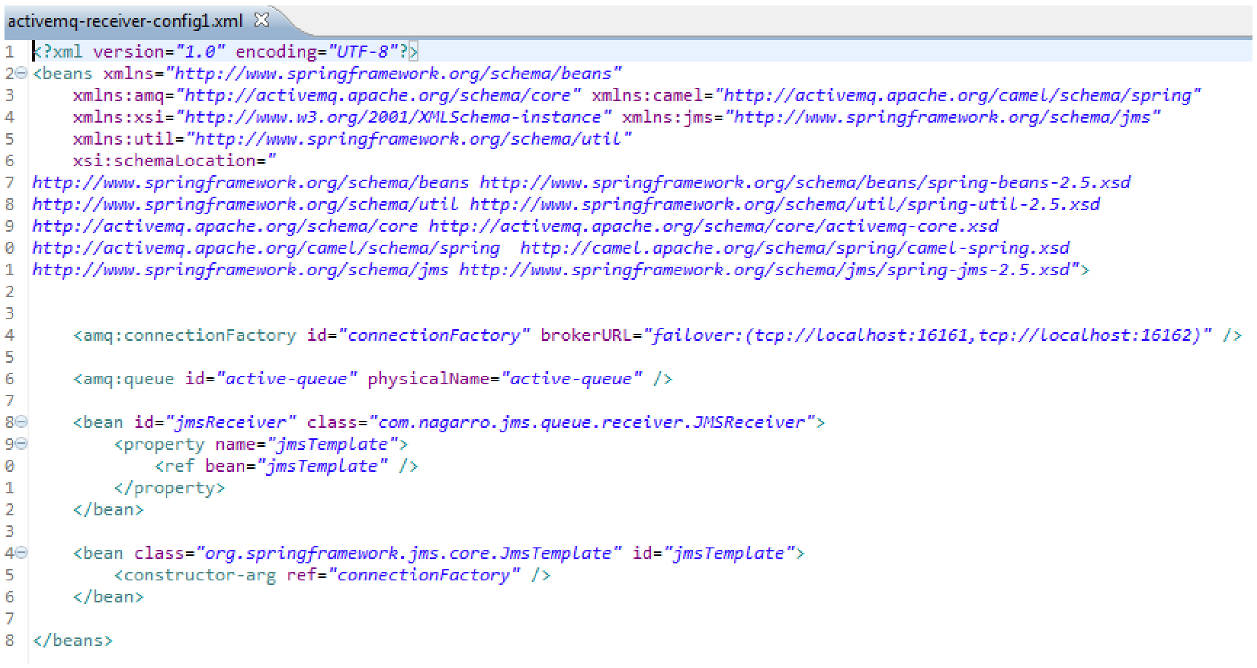
activemq-config2.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:broker useJmx="false" persistent="true" useShutdownHook="true">
        <amq:networkConnectors>
            <amq:networkConnector uri="static:(tcp://localhost:16161)"/>
        </amq:networkConnectors>
        <amq:persistenceAdapter>
            <amq:amqPersistenceAdapter directory="data/activemq2" />
        </amq:persistenceAdapter>
        <amq:transportConnectors>
            <amq:transportConnector uri="tcp://localhost:16162" />
        </amq:transportConnectors>
    </amq:broker>

    <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost:1616" />

    <amq:queue id="active-queue" physicalName="active-queue" />


</beans>
```


activemq-sender-config1.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">

    <amq:connectionFactory id="connectionFactory" brokerURL="failover:(tcp://localhost:16161,tcp://localhost:16162)" />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="jmsSender" class="com.nagarro.jms.queue.sender.JMSSender">
        <property name="jmsTemplate">
            <ref bean="jmsTemplate" />
        </property>
    </bean>

    <bean class="org.springframework.jms.core.JmsTemplate" id="jmsTemplate">
        <constructor-arg ref="connectionFactory" />
    </bean>

</beans>
```

Notice the failover: in the connectionFactory tag. This will enable the failover feature provided by ActiveMQ.


activemq-receiver-config1.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:amq="http://activemq.apache.org/schema/core" xmlns:camel="http://activemq.apache.org/camel/schema/spring"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-2.5.xsd
http://activemq.apache.org/schema/core http://activemq.apache.org/schema/core/activemq-core.xsd
http://activemq.apache.org/camel/schema/spring  http://camel.apache.org/schema/spring/camel-spring.xsd
http://www.springframework.org/schema/jms http://www.springframework.org/schema/jms/spring-jms-2.5.xsd">


    <amq:connectionFactory id="connectionFactory" brokerURL="failover:(tcp://localhost:16161,tcp://localhost:16162)" />

    <amq:queue id="active-queue" physicalName="active-queue" />

    <bean id="jmsReceiver" class="com.nagarro.jms.queue.receiver.JMSReceiver">
        <property name="jmsTemplate">
            <ref bean="jmsTemplate" />
        </property>
    </bean>

    <bean class="org.springframework.jms.core.JmsTemplate" id="jmsTemplate">
        <constructor-arg ref="connectionFactory" />
    </bean>

</beans>
```

3) Create only Broker2.java as shown below, even though Broker1 won't be running, Broker2 will serve the requests coming for broker1 as well

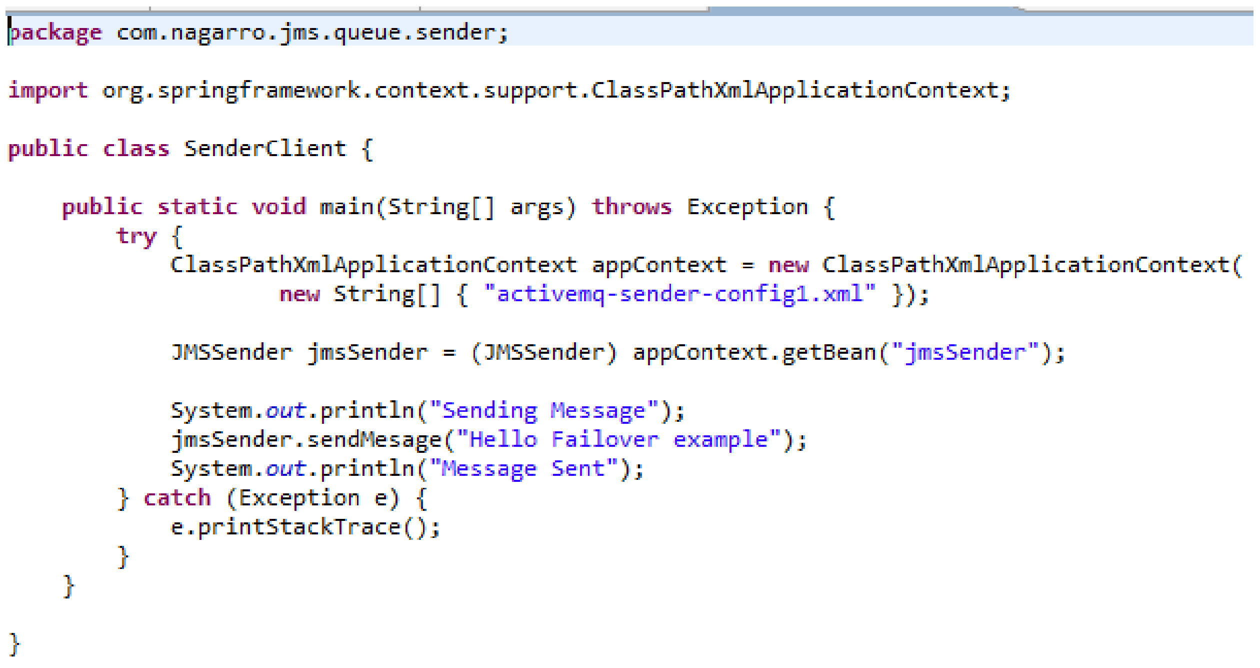
```
package com.nagarro.jms.queue.broker;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Broker2 {

    public static void main(String[] args) throws Exception {

        System.out.println("Starting JMS broker 2");

        ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                new String[] { "activemq-config2.xml" });

        System.out.println("Started JMS broker 2");

        Thread.sleep(1000000);

    }

}
```

4) Create JMSReceiver.java and JMSSender.java similar to how it was done in previous example

5) Create SenderClient.java as shown below

```
package com.nagarro.jms.queue.sender;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SenderClient {

    public static void main(String[] args) throws Exception {
        try {
            ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                    new String[] { "activemq-sender-config1.xml" });

            JMSSender jmsSender = (JMSSender) appContext.getBean("jmsSender");

            System.out.println("Sending Message");
            jmsSender.sendMesage("Hello Failover example");
            System.out.println("Message Sent");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

6) Create receiverClient.java as shown below



```java
package com.nagarro.jms.queue.sender;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SenderClient {

    public static void main(String[] args) throws Exception {
        try {
            ClassPathXmlApplicationContext appContext = new ClassPathXmlApplicationContext(
                    new String[] { "activemq-sender-config1.xml" });

            JMSSender jmsSender = (JMSSender) appContext.getBean("jmsSender");

            System.out.println("Sending Message");
            jmsSender.sendMesage("Hello Failover example");
            System.out.println("Message Sent");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

7) Run Broker2
8) Run SenderClient which is configured to send message to broker1 which is not running, but since failover is configured, broker2 will cater to requests coming for broker1 as well
9) Run ReceiverClient to receive message from Broker2