Sanyam Agrawal SE21UCSE192 CSE3

DS - Assignment 5 - ***Indirect Communication using Java***

Q1) Message Passing using Message Queues

MessageSender.java

```
package sanyam_192.jms;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;

/*
 * This class is used to send a text message to the queue.
 */
public class MessageSender
{

    /*
     * URL of the JMS server. DEFAULT_BROKER_URL will just mean that JMS server
is on localhost
     *
     * default broker URL is : tcp://localhost:61616"
     */
    private static String url = ActiveMQConnection.DEFAULT_BROKER_URL;

    /*
     * Queue Name.You can create any/many queue names as per your requirement.
     */
    private static String queueName = "MESSAGE_QUEUE";

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

```
        System.out.println("url = " + url);

        /*
         * Getting JMS connection from the JMS server and starting it
         */
        ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(url);
        Connection connection = connectionFactory.createConnection();
        connection.start();

        /*
         * Creating a non-transactional session to send/receive JMS messages.
         */
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

        /*
         * The queue will be created automatically on the server.
         */
        Destination destination = session.createQueue(queueName);

        /*
         * Destination represents our queue 'MESSAGE_QUEUE' on the JMS server.
         *
         * MessageProducer is used for sending messages to the queue.
         */
        MessageProducer producer = session.createProducer(destination);
        TextMessage message = session.createTextMessage("Hi Peter, How are
you?");

        /*
         * Here we are sending our message!
         */
        producer.send(message);

        System.out.println("Message '" + message.getText() + ", Sent
Successfully to the Queue");
        connection.close();
    }
}
```

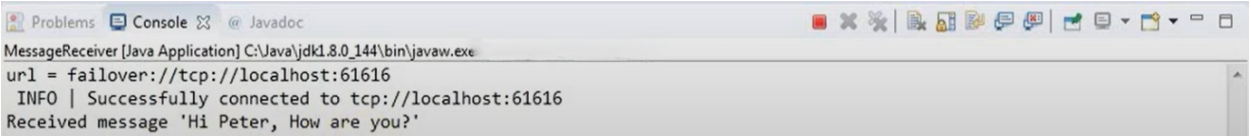
MessageReciver.java

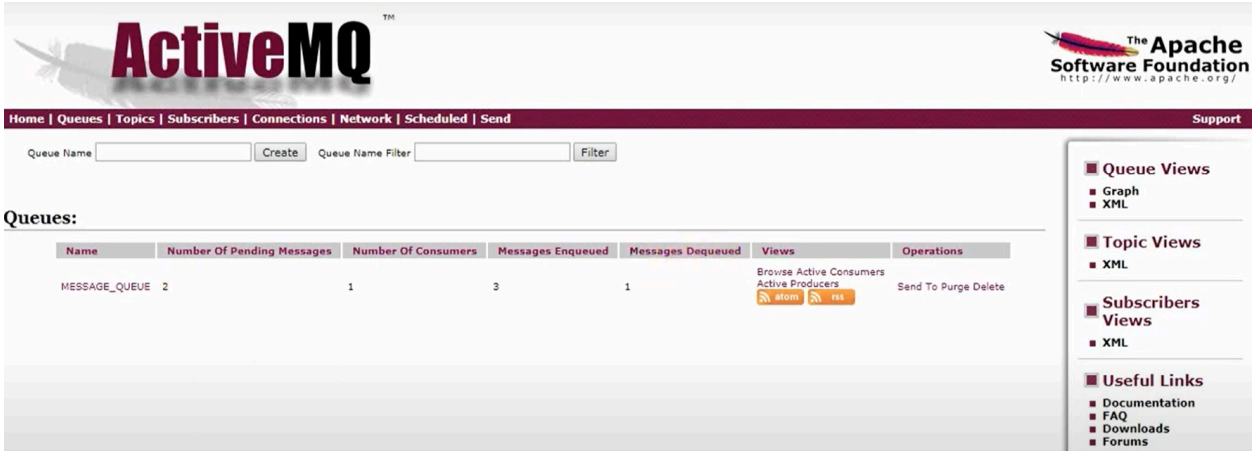
```
package sanyam_192.jms;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;

/*
 * This class is used to receive text message from the queue.
 */
public class MessageReceiver
{
    /*
     * URL of the JMS server. DEFAULT_BROKER_URL will just mean that JMS server
is on localhost
     *
     * default broker URL is : tcp://localhost:61616"
     */
    private static String url = ActiveMQConnection.DEFAULT_BROKER_URL;

    /*
     * Name of the queue we will receive messages from
     */
    private static String queueName = "MESSAGE_QUEUE";

    public static void main(String[] args) throws JMSException
    {
        System.out.println("url = " + url);

        /*
         * Getting JMS connection from the JMS server and starting it
         */
        ConnectionFactory connectionFactory = new
ActiveMQConnectionFactory(url);
        Connection connection = connectionFactory.createConnection();
        connection.start();
```

```
        /*
         * Creating session for receiving messages
         */
        Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

        /*
         * Destination represents our queue 'MESSAGE_QUEUE' on the JMS server.
         *
         * MessageConsumer is used for receiving messages from the queue.
         */
        Destination destination = session.createQueue(queueName);

        /*
         * MessageConsumer is used for receiving (consuming) messages
         */
        MessageConsumer consumer = session.createConsumer(destination);

        /*
         * Here we receive the message.
         */
        Message message = consumer.receive();

        if (message instanceof TextMessage)
        {
            Text Message text Message = (TextMessage) message;
            System.out.println("Received message '" + textMessage.getText() +
"'");
        }
        connection.close();
    }
}
```

Output:

```
Problems  Console  Javadoc
<terminated> MessageSender [Java Application] C:\Java\jdk1.8.0_144\bin\javaw.exe
url = failover://tcp://localhost:61616
 INFO | Successfully connected to tcp://localhost:61616
Message 'Hi Peter, How are you?, Sent Successfully to the Queue
```





In ActiveMQ, it's clear that three messages have been enqueued by the client. After the server processes these messages and sends a response back to the client, one message is dequeued, leaving two messages still pending.

Q2) Publish-Subscribe Model using JMS (Java Message Service)

TopicProducer.java

```
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.jms.DeliveryMode;

import org.apache.activemq.ActiveMQConnectionFactory;

public class TopicProducer implements Runnable {
```

```java
    private ActiveMQConnectionFactory connectionFactory;

    public TopicProducer(ActiveMQConnectionFactory connectionFactory)
{
        this.connectionFactory = connectionFactory;
    }

    public void run() {
        try {
            // First, create a connection
            Connection connection =
connectionFactory.createConnection();
            connection.start();

            // Now, create a Session
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            // Let's create a topic. If the topic exists, it will
return that.
            Destination destination =
session.createTopic("ComputerLab-IX");

            // Create a MessageProducer from the Session to the Topic
            MessageProducer producer =
session.createProducer(destination);
            producer.setDeliveryMode(DeliveryMode.PERSISTENT);

            // Create a message for the current topic
            String text = "Complete JMS Expt";
            TextMessage message = session.createTextMessage(text);

            // Send the message to the topic
            producer.send(message);

            // Cleanup
            connection.close();
        } catch (JMSException jmse) {
            System.out.println("Exception: " + jmse.getMessage());
```

```
        }
    }
}
```

TopicConsumer.java

```
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnectionFactory;

public class TopicConsumer implements Runnable {

    private ActiveMQConnectionFactory connectionFactory;

    public TopicConsumer(ActiveMQConnectionFactory connectionFactory)
{
        this.connectionFactory = connectionFactory;
    }

    public void run() {
        try {
            // First, create a connection
            Connection connection =
connectionFactory.createConnection();
            connection.start();

            // Now, create a Session
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            // Let's create a topic. If the topic exists, it will
```

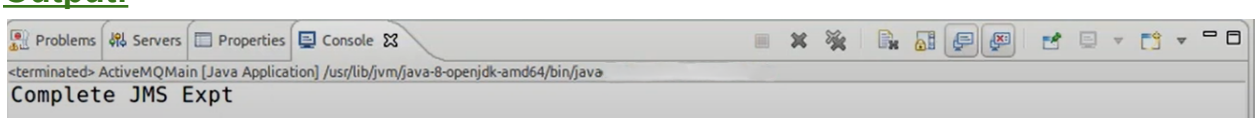
```
return that.
            Destination topicDestination =
session.createTopic("ComputerLab-IX");

            // Create a MessageConsumer for the topic
            MessageConsumer messageConsumer =
session.createConsumer(topicDestination);

            // Get the message
            Message message = messageConsumer.receive();
            if (message instanceof TextMessage) {
                TextMessage textMessage = (TextMessage) message;
                System.out.println("Received: " +
textMessage.getText());
            }

            // Cleanup
            session.close();
            connection.close();
        } catch (JMSException jmse) {
            System.out.println("Exception: " + jmse.getMessage());
        }
    }
}
```

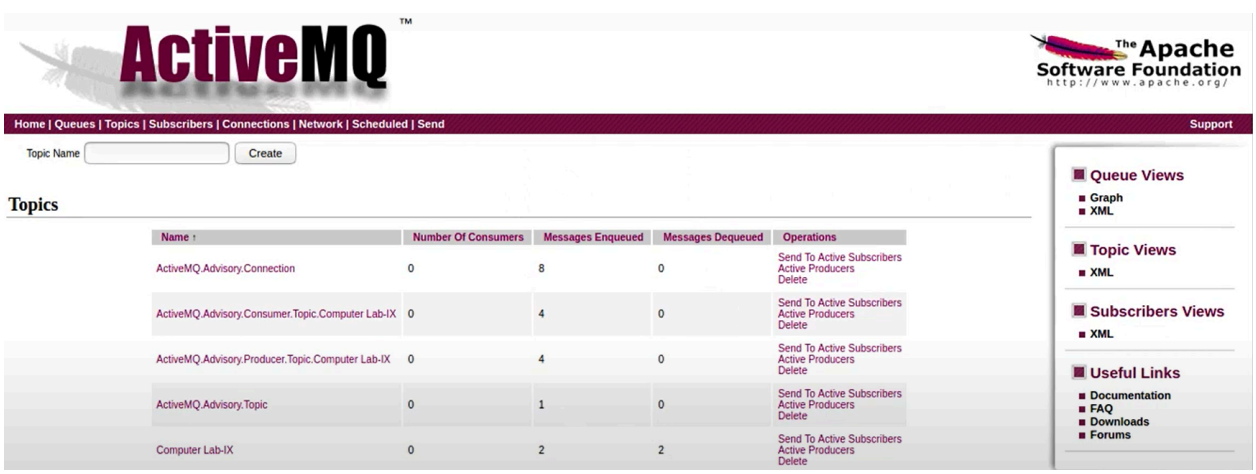
ActiveMQMain.java

```
import javax.jms.JMSException;
import org.apache.activemq.ActiveMQConnectionFactory;

public class ActiveMQMain {

    public static void main(String[] args) throws JMSException {
        // Create the connection factory
        ActiveMQConnectionFactory connectionFactory = new
ActiveMQConnectionFactory("tcp://localhost:61615");
```

```
        // Create the consumer. It will listen to the topic.
        Thread topicConsumerThread = new Thread(new
TopicConsumer(connectionFactory));
        topicConsumerThread.start();

        try {
            Thread.sleep(1000);  // Delay to ensure consumer is
ready
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Create a producer. As soon as the message is
published on the topic, it will be consumed.
        Thread topicProducerThread = new Thread(new
TopicProducer(connectionFactory));
        topicProducerThread.start();
    }
}
```

Output:





The topic "Computer Lab-IX" shows 2 messages enqueued and dequeued, indicating that messages were sent and consumed twice, reflecting successful publish-subscribe interactions.

TopicProducer.java ensures that a publisher can send messages to the specified topic (`ComputerLab-IX` in this case).

TopicConsumer.java ensures that a subscriber (consumer) listens to the topic and receives the message published by the producer.

Q3) Chat Application using Publish-Subscribe

CommandLineChat.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Properties;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicPublisher;
import javax.jms.TopicSession;
import javax.jms.TopicSubscriber;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class CommandLineChat implements MessageListener {

    public static final String TOPIC = "topic/ZAPubSubChatTopic";

    // Listener to receive messages
```

```java
    @Override
    public void onMessage(Message message) {
        try {
            // Cast the message to TextMessage and print it
            System.out.println(((TextMessage) message).getText());
        } catch (JMSException e) {
            e.printStackTrace();
        }
    }

    // Main method
    public static void main(String[] args) throws JMSException,
NamingException {
        if (args.length == 0) {
            System.out.println("a username is required");
        } else {
            String username = args[0];
            new CommandLineChat().startChat(username);
        }
    }

    // Starts the chat application
    public void startChat(String username) throws JMSException,
NamingException {
        CommandLineChat commandLineChat = new CommandLineChat();

        // Get initial context
        Context initialContext = CommandLineChat.getInitialContext();

        // Lookup topic and connection factory
        Topic topic = (Topic)
initialContext.lookup(CommandLineChat.TOPIC);
        TopicConnectionFactory topicConnectionFactory =
            (TopicConnectionFactory)
initialContext.lookup("ConnectionFactory");

        // Create a connection and subscribe to the topic
        TopicConnection topicConnection =
topicConnectionFactory.createTopicConnection();
```

```
        commandLineChat.subscribe(topicConnection, topic,
commandLineChat);

        // Start publishing messages
        commandLineChat.publish(topicConnection, topic, username);
    }

    // Method to subscribe to a topic
    public void subscribe(TopicConnection topicConnection, Topic
topic, CommandLineChat commandLineChat)
            throws JMSException {
        // Create a session for subscribing
        TopicSession subscribeSession =
topicConnection.createTopicSession(false,
TopicSession.AUTO_ACKNOWLEDGE);

        // Create a subscriber for the topic and set the listener
        TopicSubscriber topicSubscriber =
subscribeSession.createSubscriber(topic);
        topicSubscriber.setMessageListener(commandLineChat);
    }

    // Method to publish messages to the topic
    public void publish(TopicConnection topicConnection, Topic topic,
String username)
            throws JMSException, IOException {
        // Create a session for publishing
        TopicSession publishSession =
topicConnection.createTopicSession(false,
TopicSession.AUTO_ACKNOWLEDGE);
        TopicPublisher topicPublisher =
publishSession.createPublisher(topic);

        // Start the connection
        topicConnection.start();

        // Read messages from the console and publish them
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
```

```
        while (true) {
            String messageToSend = reader.readLine();

            // If the user types 'exit', close the connection and
exit
            if (messageToSend.equalsIgnoreCase("exit")) {
                topicConnection.close();
                System.exit(0);
            } else {
                // Create a text message, format it with the
username, and publish it
                TextMessage message =
publishSession.createTextMessage();
                message.setText("[" + username + "]: " +
messageToSend);
                topicPublisher.publish(message);
            }
        }
    }

    // Method to get the initial context for JNDI lookup
    public static Context getInitialContext() throws NamingException
{
        Properties props = new Properties();
        props.setProperty("java.naming.factory.initial",
"org.jnp.interfaces.NamingContextFactory");
        props.setProperty("java.naming.factory.url.pkgs",
"org.jboss.naming");
        props.setProperty("java.naming.provider.url",
"localhost:1099");

        return new InitialContext(props);
    }
}
```

Output:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\DS_192>cd work
C:\Users\DS_192\work>cd test
C:\Users\DS_192\work\test>java -jar pubsub.jar user1
[User2: 2222222222]
1111111111
[User1: 1111111111]
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\DS_192>cd work
C:\Users\DS_192\work>cd test
C:\Users\DS_192\work\test>java -jar pubsub.jar user2
2222222222
[User2: 2222222222]
[User1: 1111111111]
```

In this output, both users (`user1` and `user2`) send messages to a shared chat room. Each user's messages are formatted with their username and broadcasted to all participants. User1 sees User2's message first, and vice versa, illustrating real-time communication using the Publish-Subscribe model.