JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Message Oriented Middleware

Systemtechnik Labor
4CHITT 2015/16, Gruppe A

Gabriel Frassl

Version 1.0

Note:

Betreuer: M.Borko

Begonnen am 26. Februar 2016

Beendet am 08. März 2016

Inhalt

1 Aufgabenstellung

Implementieren Sie eine Chatapplikation mit Hilfe des Java Message Service. Verwenden Sie Apache ActiveMQ (http://activemq.apache.org) als Message Broker Ihrer Applikation. Das Programm soll folgende Funktionen beinhalten:

- Benutzer meldet sich mit einem Benutzernamen und dem Namen des Chatrooms an.
  Beispiel für einen Aufruf:

  jmschat <ip_message_broker> <benutzername> <chatroom>

- Der Benutzer kann in dem Chatroom (JMS Topic) Nachrichten an alle Teilnehmer eine Nachricht senden und empfangen.
  Die Nachricht erscheint in folgendem Format:

  <benutzername>: <Nachricht>

- Zusätzlich zu dem Chatroom kann jedem Benutzer eine Nachricht in einem persönlichen Postfach (JMS Queue) hinterlassen werden. Der Name des Postfachs und der Benutzername sind ident!

  Nachricht an das Postfach senden:
  MAIL <benutzername> <nachricht>

  Eignes Postfach abfragen:
  MAILBOX

- Der Chatraum wird mit den Schlüsselwort EXIT verlassen. Der Benutzer verlaesst den Chatraum, die anderen Teilnehmer sind davon nicht betroffen.

Die Applikation ist über das Netzwerk zu testen! Abnahmen, die nur auf localhost basieren sind unzulässig.

2 Quellen

http://activemq.apache.org/index.html
http://www.academictutorials.com/jms/jms-introduction.asp
http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS.html#wp84181
http://www.openlogic.com/wazi/bid/188010/How-to-Get-Started-with-ActiveMQ
http://jmsexample.zcage.com/index2.html
http://www.onjava.com/pub/a/onjava/excerpt/jms_ch2/index.html
http://www.oracle.com/technetwork/systems/middleware/jms-basics-jsp-135286.html
http://java.sun.com/developer/technicalArticles/Ecommerce/jms

3 Durchführung der Übung

3.1 Setup

Download:

Bevor mit dem Programmieren begonnen werden kann muss zuerst der Apache Message Broker ActiveMQ herunter geladen werden.

http://activemq.apache.org/download.html

Broker starten

Nach dem Download muss der ActiveMQ Service auf dem Gerät welches als Message Broker dienen soll gestartet werden.

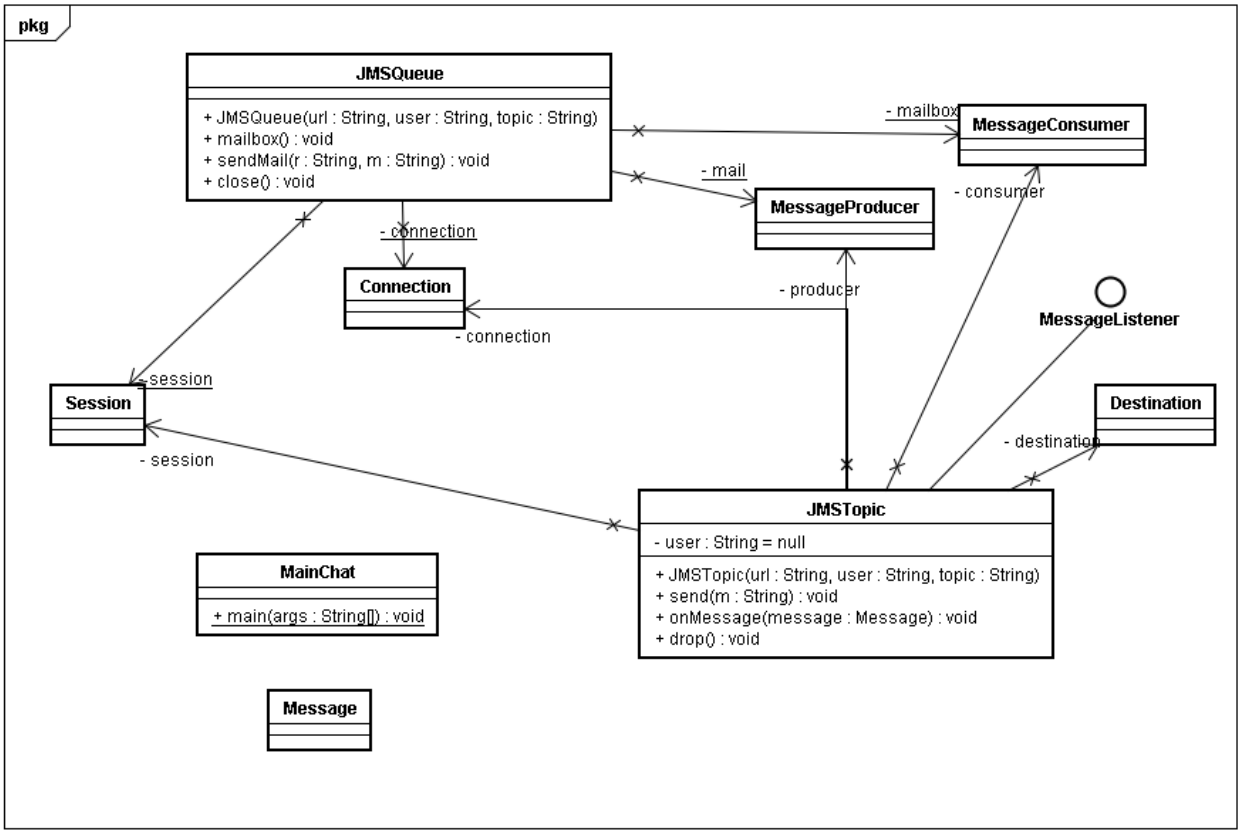
apache-activemq-5.13.1\bin\win64 -> Broker starten

ActiveMQ in Eclipse Buildpath

Als letzter Schritt vor dem Durchführen der eigentlichen Aufgabenstellung muss die in ActiveMQ vorhandene jar Datei im Build Path des Arbeitsprojekts eingebunden werden.

3.2 Designüberlegung

Für die beiden Funktionen JMSQUEUE und JMSTOPIC wird jeweils eine Klasse erstellt. Der User Input und der dadurch bestimmte Ablauf des Chatprogrammes werden in der Klasse MainChat geregelt.



3.3 Implementieren der Funktionalität

JMSTopic

Als erste Klasse erstellen wir JMSTopic. Das heißt die Klasse welche für Standard Broadcast Nachrichten verwendet wird. Sie implementiert MessageListener.

Attribute:

Als Attribute benötigt diese Klasse alles was für die Connection und für die Topic Session nötig ist. Des Weiteren sollte ein Attribut des Usernamen erstellt werden, da dieses in mehreren Methoden benötigt wird.

```
public class JMSTopic implements MessageListener {
    private String user = null;
    private Session session = null;
    private Connection connection = null;
    private MessageProducer producer = null;
    private Destination destination = null;
    private MessageConsumer consumer = null;
```

Constructor:

Im Constructor wird mit den als Parameter mitgelieferten Informationen eine Verbindung erstellt. Mit dieser Verbindung werden dann Session, Destination, Producer und Consumer festgelegt.

```
ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, ActiveMQConnection.DEFAULT_PASSWORD, url);
connection = connectionFactory.createConnection();
connection.start();

session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
destination = session.createTopic(topic);

producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

consumer = session.createConsumer(destination);
consumer.setMessageListener(this);

catch (Exception e) {
System.out.println("Connection failed: \n " + e.getMessage());
```

SendMessage:
Nun wird eine Methode geschrieben durch den producer eine Nachricht verschickt.

```
public void sendMessage(String m) {
    try {
        TextMessage message;
        message = session.createTextMessage(user + ": " + m);
        producer.send(message);
```

OnMessage:
On Message gibt Nachrichten aus wenn welche von anderen Usern verschickt worden sind. Diese Methode wird von selbst aufgerufen wenn eine Nachricht verschickt wurde(MessageListener).

```
public void onMessage(Message message) {
    try {
        System.out.println(((TextMessage) message).getText());
    } catch (JMSException e) {
```

JMSQueue

Nun wird die Klasse welche für private Nachrichten zwischen zwei Usern verwendet wird erstellt. Sie hat die beiden Funktionen eine Mail zu senden und seine Mailbox abzufragen.

Attribute:

Als Attribute benötigt diese Klasse alles was für die Connection und für die Queue Session nötig ist.

```
private static Session session = null;
private static Connection connection = null;
private static MessageConsumer mailbox = null;
private static MessageProducer mail = null;
```

Constructor:

Wie auch bei JMSTopic wird hier erstmal eine Verbindung aufgebaut auf welcher dann Session, Consumer und Producer aufgebaut werden.

```
ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, ActiveMQConnection.DEFAULT_PASSWORD, url);
connection = connectionFactory.createConnection();
connection.start();
session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

sendMail:

Der Methode zum Senden einer Mail muss im Gegensatz zu JMSTopic nicht nur die Nachricht sondern auch ein Zieluser mitgegeben werden, da ja nur dieser die Nachricht erhält.

```
public void sendMail(String r, String m) {
    TextMessage message;
    try {
        Destination destination = session.createQueue(r);//
        System.out.println("mail sent");
        message = session.createTextMessage(m);
        mail.send(message);
```

Mailbox:

Beim Aufruf von Mailbox wird eine Liste mit allen Nachrichten ausgegeben die der User erhalten hat.

```
System.out.println("Mailbox:");
TextMessage message;
while ((message = (TextMessage) mailbox.receive(500)) != null) {
    System.out.println(message.getText());
}
```

MainChat

MainChat enthält nur die main-Methode welche den User Input auswertet und somit denn Programmablauf regelt.

Auslesen der Argumente

Beim Programmstart müssen die Informationen für die Chatverbindung als CLI Argumente eingegeben werden. In der Main Methode wird nun überprüft ob die richtige Anzahl an Argumenten vorhanden ist. Danach wird jeweils ein Objekt für JMSTopic und JMSQueue, mit den Verbindungsdaten erstellt.

```
if (args.length != 3) {
    System.err.println("missing arguments! <ip_message_broker> <benutzername> <chatroom>");
} else {
    JMSTopic topic = new JMSTopic("tcp://" + args[0] + ":61616", args[1], args[2]);
    JMSQueue queue = new JMSQueue("tcp://" + args[0] + ":61616", args[1], args[2]);
```

Sequence of Actions

Da der User durch seine Eingaben steuert ob er Topic oder Queue verwenden will, wird auch der Input des Users als Basis für den Ablauf des Programmes genommen. Dafür wird ein BufferedReader verwendet. Dannach wird durch mehrere If-Klauseln überprüft welche Aktion der User vornehmen will.

```
BufferedReader commandLine = new java.io.BufferedReader(new InputStreamReader(System.in)); //Reads CLI input of
    while (true) { //programm sequence until /exit is being called
        String input;
        input = commandLine.readLine();
        String[] splitInput = input.split("\\s+");
        String[] ms = null;

        if (splitInput[0].equalsIgnoreCase("/exit")) { //when /exit close topic,queue and exit programm
            topic.drop();
            queue.close();
            System.exit(0);
        } else if (splitInput[0].equalsIgnoreCase("/mail")) {//when /mail call queue methods to send mail
            if (input.matches("/\\S+\\s+\\S+\\s+\\S+.+")) {
                ms = input.split("/\\S+\\s+\\S+\\s+");
                queue.sendMail(splitInput[1], ms[1]);
            } else {//if /mail is not correctly used
                System.err.println("Wrong Usage! \n"
            +"do : /mail <destination_username> <message>");
            }

        } else if (splitInput[0].equalsIgnoreCase("/mailbox")) {//if /mailbox call queue mailbox method
            queue.mailbox();

        } else { // in any other cases its a simple topic message -> call topic send method
            topic.sendMessage(input.toString());
        }
    }
```

3.4 Testen

Zum Testen des Programmes sollte dieses auf zumindest 2 Hosts gestartet werden. Einer der beiden muss wie bereits im Kapitel Setup beschrieben als Message Broker dienen und den ActiveMQ Broker gestartet haben.

1)Starten des Brokers auf meinem Host

2)Verbinden der beiden Hosts

10.0.104.2018<user> und 10.0.105.148<rborsos> und testen der Chatanwendung.

```
Welcome to JMS chat
Chat usage:
to check mailbox: /mailbox
to send private mail: /mail <destination_username> <message>
hi
user: hi
rborsos:
rborsos: hallo
das ist ein Test für das Protokoll. Sag hallo
user: das ist ein Test für das Protokoll. Sag hallo
rborsos: hallo


user: das ist ein Test für das Protokoll. Sag hallo
hallo
rborsos: hallo
```

4 Github

Das Projekt befindet sich in dem Repository:

https://github.com/gfrassl-tgm/MessageOrientedMiddleware.git

5 Zeitaufzeichnung

Anmerkung:

Das größte Problem an dieser Aufgabe war für mich das Verständnis der Verwendung von ActiveMQ und JMS. Es fiel mir schwer mich in das Beispiel einzulesen. Durch die Hilfe eines Klassenkameraden habe ich dann erst verstanden wie das Beispiel zu lösen ist, was auch zu der verspäteten Fertigstellung geführt hat.

Übung	Zeitaufwand geschätzt	Zeitaufwand real	Datum/Ort
JMSTopic	180	120	Zuhause 05.03.2016
JMSQueue	180	120	Zuhause 06.03.2016
Main	60	60	Zuhause 07.03.2016
Protokoll	120	180	Zuhause 08.03.2016
Gesamt	540min	480min	