JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Protokoll

MOM

Praktische Prüfung

SYT
4CHIT 2015/16

Borsos Robert

Version 1.0

Note:

Betreuer: T. Micheler

Begonnen am 16. Juni 2016

Beendet am 16. Juni 2016

Inhaltsverzeichnis

1 Einführung

Es soll eine Administrationsseite zum Verwalten einer Warteschlange mittels JMS für ein Krankenhaus erstellt werden.

1.1 Ziele

Ziel dieser Übung ist JMS kennen zu lernen und damit arbeiten zu können.

1.2 Voraussetzungen

- Grundlagen zu Java
- Grundlagen zu Apache ActiveMQ
- Grundlagen zu JMS

1.3 Aufgabenstellung

Implementieren Sie ein MOM-System, das zur Administration von Warteschlangen von Patienten bei der Aufnahme in einem Spital herangezogen werden kann. Jeder neue Patient bekommt eine eindeutige Nummer wird einer Warteschlange zugeordnet. Die Zuordnung der Warteschlange kann nach 2 verschiedenen Arten erfolgen: Round-Robin oder Least-Connection (wobei wir hier als Connection die Anzahl an Patienten in einer Warteschlange ansehen). Der Verteilungsmodus (Round-Robin / Least-Connection) kann mittels einem Befehl umgestellt werden. Der Arzt kann jeweils eine Warteschlange wählen und bekommt die Nummer des nächsten Patienten in der Warteschlange -> mit dieser Aktion wird der Patient aus der Warteschlange entfernt. Überlegen Sie sich eine passende Syntax zu den notwendigen Befehlen. Die Anzahl der Warteschlangen ist nicht begrenzt und kann jederzeit mit einem Befehl erhöht werden. Wenn eine Warteschlange leer ist, dann kann diese Warteschlange auch wieder gelöscht werden. Solange eine Warteschlange nicht leer ist, kann diese auch nicht gelöscht werden.

Punkteverteilung:
1 Punkte: Implementierung Befehle für Patient / Arzt
1 Punkte: Administration der Warteschlangen (Hinzufügen, Löschen und Umstellung des Zuordnungsmodus)
3 Punkte: Implementierung Verteilungsmethode Round-Robin
3 Punkte: Implementierung Verteilungsmethode Least-Connection

Verwenden Sie Apache-Active-MQ als Message Broker.

Persönlich Abnahme inklusive Abnahmegespräch erfolgt am 17.06.2016 um 12:30 Uhr. Wenn Sie an diesem Termin keine Zeit haben, dann bitte ich Sie mit mir Kontakt aufzunehmen, damit wir einen Ersatztermin suchen. Ein Abnahmegespräch ist unbedingt erforderlich!

Abzugeben sind (es wird ein eigene Abgabe in Moodle geben):
- Protokoll (wie bei Laboruebungen, inkl. UML Klassendiagramm)
- Ausfuehrbare JAR-Dateien inklusive Quellcode
- API mittels Javadoc

2 Ergebnisse

2.1 Theorie

Mit **JMS** hat eine **Schnittstelle** zum Verbinden und Ansteuern einer Message Oriented Middleware (MOM). Hierbei werden die Funktionen Senden und Empfangen von Nachrichten von einem Client aus angeboten.
JMS kann auf zwei verschiedene Arten Nachrichten Empfangen:

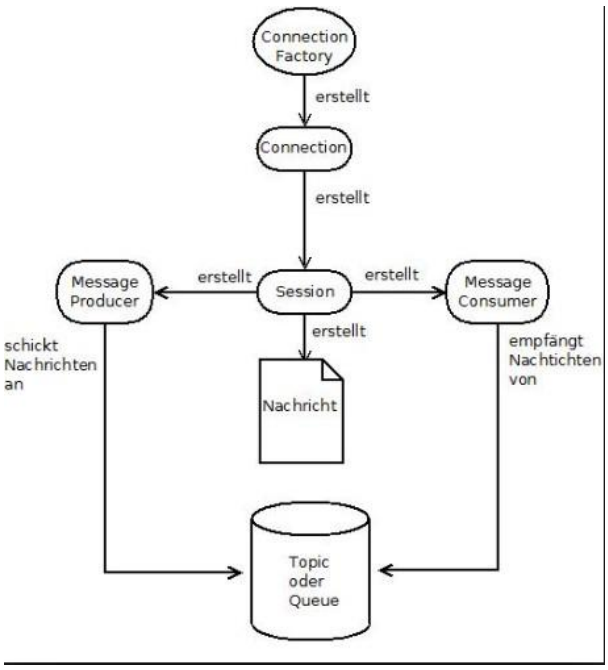
- **Asynchronously**

  Der Empfänger registriert einen Message-Listener, der sich wie ein EventListener verhält. Wenn die Nachricht ihn dann erreicht, wird die Methode onMessage sofort aufgerufen. Mit der wird dann die Nachricht weiterverarbeitet.

- **Synchronously**

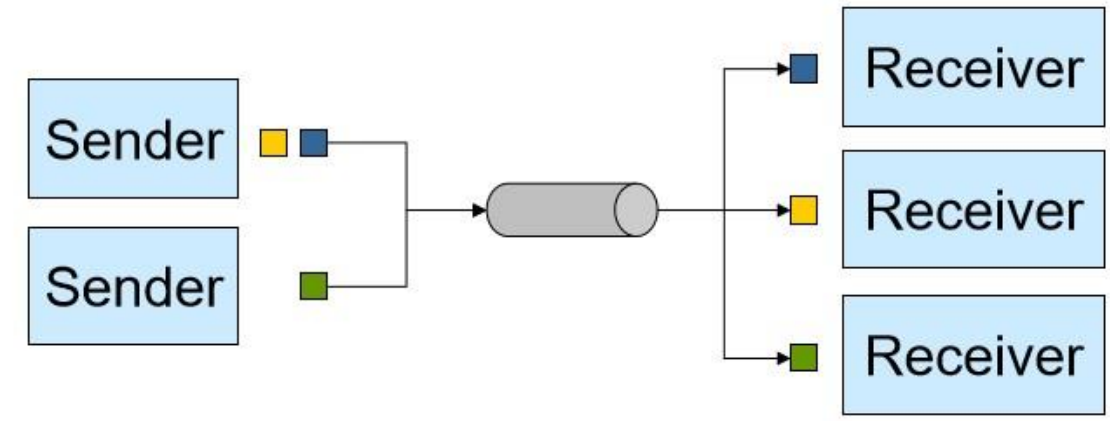
  Der Empfänger ruft eine receive-Methode auf, um seine Nachrichten zu erhalten. Diese übernimmt den Empfang und kann ein Time-Out für den Empfang festlegen.

Aufbau von Java Message Service (JMS)



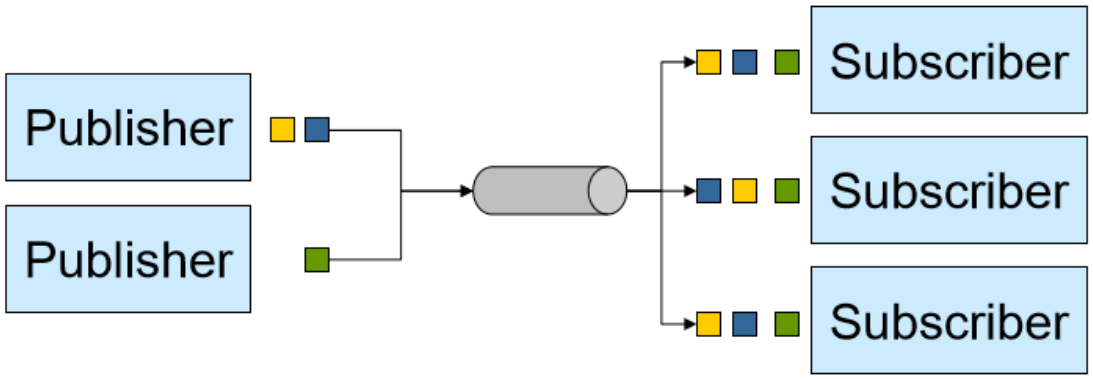
[1]

Point-To-Point (PTP) Messaging Domain [3]



PTP funktioniert mit dem Konzept der Warteschlangen (Queue). Jede Nachricht hat nur einen Empfänger und die Empfänger holen die Nachrichten aus der Queue nach den First-In-First-Out (FIFO). Diese Warteschlangen behalten die Nachrichten bis zum Abruf durch einen Empfänger oder wenn sie ihre Ablaufzeit erreicht haben. Daher ist der Sender unabhängig davon, ob der Empfänger zu dieser Zeit zu erreichen ist. (Queue.java)

Publish/Subscribe Messaging Domain [3]



Bei dieser Methode wird die Nachricht an ein bestimmtes Topic gesendet. Diese Topic System verteilt die Nachrichten an die Empfänger, die das Topic als Empfänger haben. Diese Nachrichten werden dann hier so lange behalten, bis der Empfänger die Nachricht erhält. So kann im Gegensatz zu der PTP Methode die Nachrichten an mehrere Empfnänger verteilt werden. Zum Empfangen und Versenden ist die Verfügbarkeit des „Chatters" notwendig. Daher können nur die Nachrichten von einem Topic empfangen werden. (Topic.java)

Least Connection

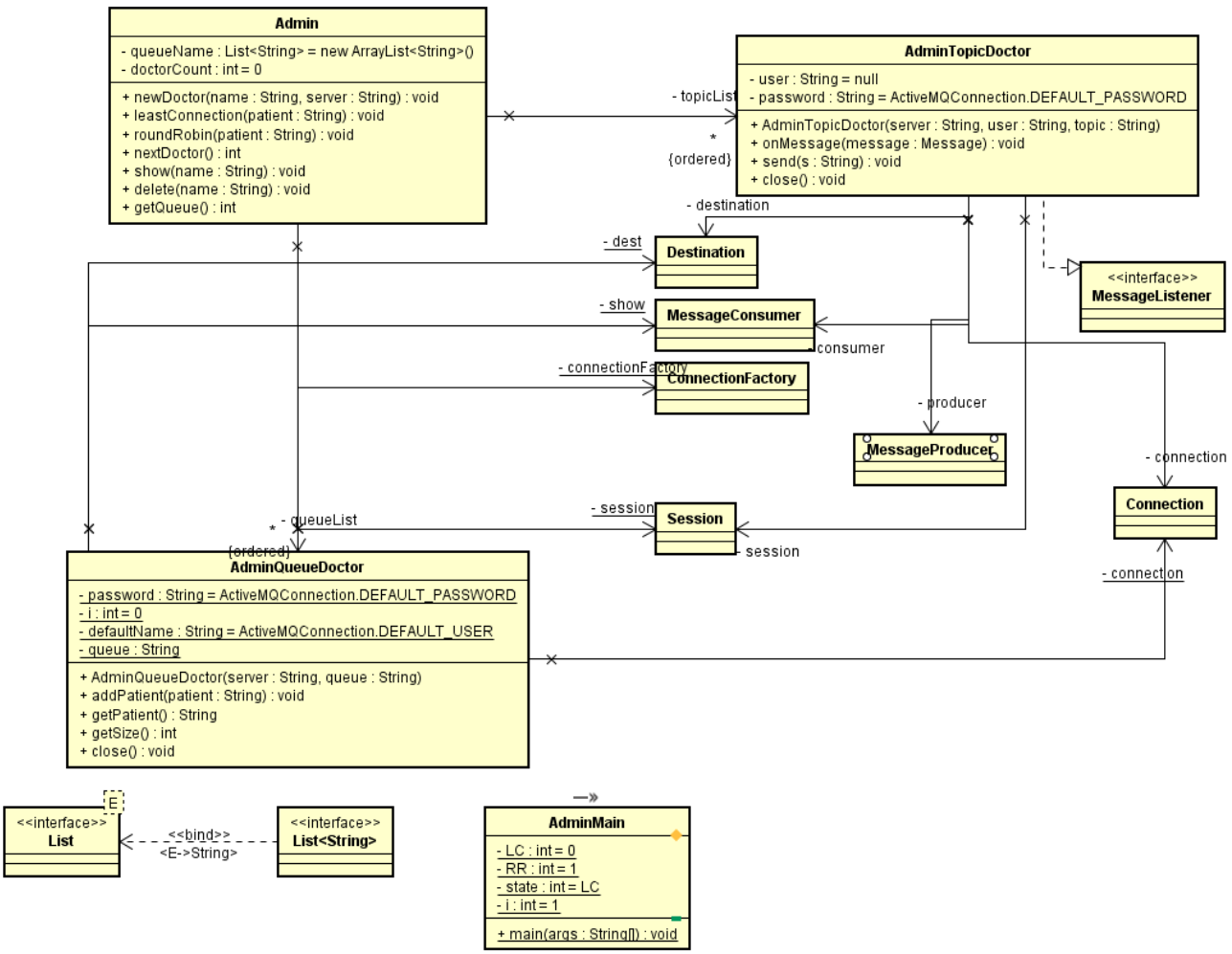
Bei der Least Connection wird die Queue in der am wenigsten Patienten sind ausgesucht.

Round Robin

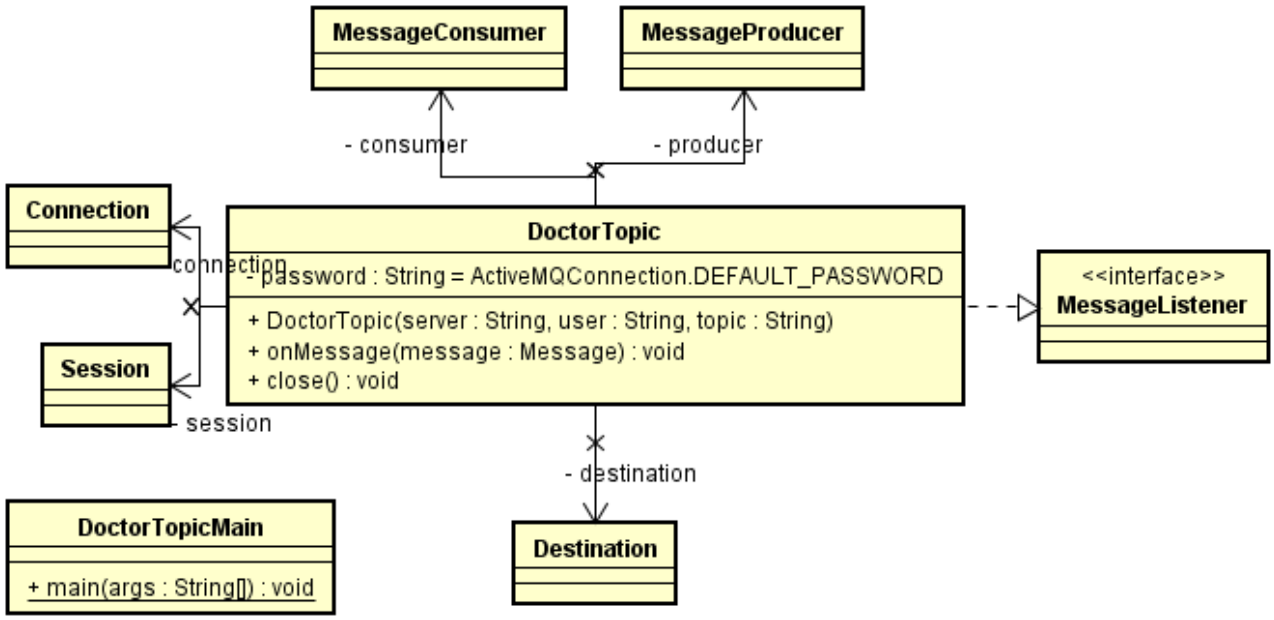
Bei dem Round Robin wird immer die nächste Queue in der Reihenfolge ausgesucht.

2.2 UML

AdminMain.java
Admin.java
AdminQueueDoctor.java
AdminTopicDoctor.java



DoctorTopic.java
DoctorTopicMain.java



2.3 Programmstart

Damit das Programm verwendet werden kann, muss Apache ActiveMQ gestartet werden. (\apache-activemq-5.13.3\bin\win64\wrapper.exe)

Eclipse

Zum starten des Programmes muss die Klasse **AdminMain.java** gestartet werden. Diese verlangt Parameter, die bei den Run Configurations (Daten, die die Main Methode mit den args speichert) gesetzt werden.

In unserem Fall muss dabei nur die IP Adresse des Brokers (Apache ActiveMQ) gesetzt werden.

<ip_message_broker> //Z.b.: localhost

Weiteres muss

JAR Datei

Es wurde mit Eclipse (Export -> Java -> Runnable Jar) von der Klasse Main.java eine Runnable Jar erstellt. Nun wird eine Eingabeaufforderung im Verzeichnis dieser .jar Datei geöffnet. Nun zum Starten dieser Datei wird

folgendes hingeschrieben:

java -jar Datei.jar <ip_message_broker>

//Z.b.: java -jar AdminMain.jar localhost

```
C:\Users\Robert\Desktop\TGM\4CHIT\SYT\Micheler\MOM_>java -jar AdminMain.jar localhost
Queuing is activated
```

Nun muss noch das zweite Programm gestartet werden, welches nur als Subscriber dient. Davon wurde ebenfalls eine JAR Datei erstellt welche nun wie folgt mit dem Namen des Doctors/Queue gestartet wird:

java -jar Datei.jar <ip_message_broker> <Doctor/Queue>

//Z.b.: java -jar DoctorTopicMain.jar localhost Dr.Borsos

Damit dies funktioniert, muss der Name des in AdminMain.jar erstellten Doctors/Queue mit dem Namen von DoctorTopicMain.jar übereinstimmen.

2.4 Programmverwendung

Nach dem starten stehen folgende Befehle **in AdminMain.jar** zur Verfügung:

Syntax	**Erklärung**
NEW <queue/topic>	Erstellen einer neuen Queue mit dem angegebenen Namen.
DELETE <queue/topic>	Löschen der angegebenen Queue.
SHOW <queue>	Anzeigen des nächsten Patienten der angegebenen Queue.
<Name>	Einfügen eines neuen Patienten in eine Queue.
LC	Wechseln zum Zuordnungsmodus Least Connection
RR	Wechseln zum Zuordnungsmodus Round Robin

Nach dem starten stehen folgende Befehle **in DoctorTopicMain.jar** zur Verfügung:

Syntax	Erklärung
EXIT	Verlassen des Topics

2.5 Testen

Das Programm hat nicht die vollständig verlangte Funktion, wodurch die Tests jeweils auf eine Queue bezogen sind. Nähere Problembeschreibung in Punkt 2.7.

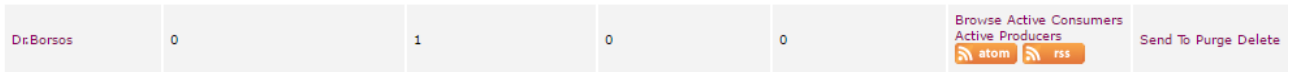
Es können nur einnamige Queues/Topics als auch Patienten erstellt werden!

Neue Queue/Topic erstellen

```
NEW Dr.Borsos
Queue created
```

Über **http://localhost:8161/admin/** (User: admin ; Passwort: admin) kann dies überprüft werden.

Queue:



Topic:

Queue/Topic löschen

Queue/Topic löschen, wenn Queue leer ist:

```
DELETE Dr.Borsos
Queue is closed, no more inputs!
```

Queue/Topic löschen, wenn Queue leer ist:

```
DELETE Dr.Borsos
Dr.Borsos not empty
```

Zuordnungsmodus ändern

Das Programm startet mit dem Zuordnungsmodus LC (Least Connection).

Wechseln zu Round Robin (RR):

```
RR
State changed to Round Robin
```

Wechseln zu Least Connection (LC):

```
LC
State changed to Least Connection
```

Patienten hinzufügen

```
Gabriel
Patient added
```

2.6 Codeerklärung

Es gibt insgesamt 2 .jar Dateien. Die AdminMain.jar Datei ist komplett zum Verwalten der Queues und beinhaltet die Funktionalität. Die DoctorMain.jar Datei wird gestartet, in welcher dann für den Doktor die nächsten Patienten angezeigt werden.

AdminMain.java

In der AdminMain.java Klasse ist die Main Methode beinhaltet. Diese liest mit den args[] die IP Adresse des Message Brokers ein. Weiteres wird in einer while-Schleife die Eingabe eingelesen und demnach eine Funktion durchgeführt.

Admin.java

Die Klasse Admin.java beinhaltet die Funktionalität des Programmes. Beim Aufruf von der Methode **newDoctor()** wird eine neues Objekt von **AdminQueueDoctor** und **AdminTopicDoctor** erzeugt. Diese werden dann einer Liste hinzugefügt.

```
private List<AdminQueueDoctor> queueList = new ArrayList<AdminQueueDoctor>(); //List for the queue objects
private List<AdminTopicDoctor> topicList = new ArrayList<AdminTopicDoctor>(); //List for the topic objects
private List<String> queueName = new ArrayList<String>(); //List for the queue names
private int doctorCount = 0; //Counter for Round Robin

/**
 * Make new Doctor/Queue and adding it to the lists above
 * @param name
 * @param server
 */
public void newDoctor(String name, String server){
    AdminQueueDoctor queue = new AdminQueueDoctor("tcp://" + server + ":61616", name);
    AdminTopicDoctor topic = new AdminTopicDoctor("tcp://" + server + ":61616", name, name);
    queueList.add(queue);
    topicList.add(topic);
    queueName.add(name);
    System.out.println("Queue created");
}
```

Beim Aufruf von der Methode **leastConnection()** wird ein Patient nach dem

Least Connection Prinzip einer Queue hinzugefügt.

```
/**
 * Add Patient to a queue through Least connection
 * @param patient
 */
public void leastConnection(String patient){
    int tempSize = 100;
    int tempQueue = 0;
    for(int i = 0; i < queueName.size(); i++){ //search through all queues
        if(queueList.get(i).getSize() < tempSize){
            tempSize = queueList.get(i).getSize();
            tempQueue = i;
        }
    }
    queueList.get(tempQueue).addPatient(patient); //add Patient to queue
}
```

Beim Aufruf von der Methode **roundRobin()** wird ein Patient nach dem Round Robin Prinzip einer Queue hinzugefügt.

```
/**
 * Adding Patient to a queue through Round Robin
 * @param patient
 */
public void roundRobin(String patient){
    queueList.get(this.nextDoctor()).addPatient(patient); //add Patient to queue
    doctorCount++;
}

/**
 *
 * @return next Queue for Round Robin
 * @return if no more queue, than start at 0 again
 */
public int nextDoctor(){
    if(doctorCount <= queueList.size()){
        return doctorCount;
    }
    else{
        doctorCount = 0;
        return doctorCount;
    }
}
```

Mit der Methode **show()** wird ein Patient der angegebenen Queue angezeigt.

```
/**
 * Get the next patient for a specific queue
 * @param name
 */
public void show(String name) {
    boolean found = false;
    String tempPatient;
    for(int i = 0; i < this.queueName.size(); i++){
        if(queueName.get(i).equals(name) && queueList.get(i).getSize() != 0){
            tempPatient = queueList.get(i).getPatient();
            System.out.println("Next Patient: " + tempPatient);
            topicList.get(i).send(tempPatient);
            found = true;
        } else if (queueList.get(i).getSize() == 0){
            System.out.println("Queue is empty");
            found = true;
        }
    }
    if(found == false){
        System.out.println("Queue not found"); // If no queue named like that is found
    }
}
```

Mit der Methode delete() wird eine Queue/Topic gelöscht, wenn die Queue leer ist.

```
/**
 * delete queue if not empty
 * @param name
 */
public void delete(String name) {
    for(int i = 0; i < this.queueName.size(); i++){
        if(queueName.get(i).equals(name)){
            if(queueList.get(i).getSize() == 0){ //.getSize() from Queue which knows about the queue content
                queueList.get(i).close();
                topicList.get(i).close();
            }
            else {
                System.out.println(name + " not empty");
            }
        }
    }
}
```

AdminQueueDoctor.java

In AdminQueueDoctor.java wird eine neue Queue für einen Doktor erzeugt.

```java
public class AdminQueueDoctor {
    private static ConnectionFactory connectionFactory;
    private static String password = ActiveMQConnection.DEFAULT_PASSWORD;
    private static Connection connection = null;
    private static Session session = null;
    private static Destination dest;
    private static MessageConsumer show = null;
    private static int i = 0;
    private static String defaultName = ActiveMQConnection.DEFAULT_USER;
    private static String queue;

    /**
     * Class-Method from Queue
     *
     * @param url ip for connection
     * @param user username
     * @param topic chattopic
     */
    public AdminQueueDoctor(String server, String queue) {
        this.queue = queue;
        try {
            connectionFactory = new ActiveMQConnectionFactory(defaultName, password, server);
            connection = connectionFactory.createConnection();
            connection.start();

            // Create the session
            session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

            // setting destination address
            dest = session.createQueue(queue);

            // Create the consumer
            show = session.createConsumer(dest);

        } catch (Exception e) {
            System.out.println("Queue connection failed");
        }
    }
```

Weiteres gibt es die Methode **addPatient()** und **getPatient()** welche zum Hinzufügen und Auslesen eines Patienten dient.

```java
/**
 * Add Patient to Queue
 * @param patient
 */
public void addPatient(String patient){
    TextMessage msg;
    try {
        //Destination dest = session.createQueue(queue);
        // Create the producer.
        MessageProducer mail = session.createProducer(dest);
        System.err.println("Patient added");
        msg = session.createTextMessage(patient);
        mail.send(msg);
        i++;
    } catch (JMSException e) {
        e.printStackTrace();
    }
}

/**
 * Receives the TextMessage (Patient) from the queue
 */
public String getPatient(){
    String temp = null;
    try {
        TextMessage msg = (TextMessage) show.receive(300);
        //System.out.println("" + msg);
        //System.out.println("Next Patient: " + msg.getText());
        temp = msg.getText();
        i--;
    } catch (JMSException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return temp;
}
```

AdminTopicDoctor.java

In AdminTopicDoctor.java wird eine neues Topic für einen Doktor erzeugt.

```
public class AdminTopicDoctor implements MessageListener {
    private String user = null;
    private String password = ActiveMQConnection.DEFAULT_PASSWORD;

    private Session session = null;
    private Connection connection = null;
    private MessageProducer producer = null;
    private Destination destination = null;
    private MessageConsumer consumer = null;

    /**
     * Class-Method from Topic
     *
     * @param url ip for connection
     * @param user username
     * @param topic topic
     */
    public AdminTopicDoctor(String server, String user, String topic) {
        this.user = user;

        try {
            ConnectionFactory connectionFactory = new ActiveMQConnectionFactory(user, password, server);
            connection = connectionFactory.createConnection();
            connection.start();

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

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

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

        } catch (Exception e) {
            System.out.println("Topic connection failed");
        }
    }
```

Diese Klasse beinhaltet weiteres NUR eine Funktion **send()** zum Einfügen eines neuen Patienten.

```
/**
 * to send a notification to the doctor
 *
 * @param s message
 */
public void send(String s) {
    TextMessage message;
    try {
        message = session.createTextMessage(s); // rborsos: "text"
        producer.send(message);
    } catch (JMSException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

DoctorTopicMain.java

In der DoctorTopicMain.java Klasse ist die Main Methode beinhaltet. Diese liest mit den args[] die IP Adresse des Message Brokers und den Doktor/Topic auf dem man horchen möchte ein. Weiteres wird in einer while-Schleife auf das Wort EXIT gehorcht, wobei das Topic danach geschlossen wird.

DoctorTopic.java

DoctorTopic.java ist der Subscriber, der den nächsten Patienten ausgibt.

2.7 Problembeschreibung

Das Programm wurde bis auf das Problem abgebaut!

Trotz dieses Aufwandes, konnte das Problem festgestellt, aber nicht gelöst werden.

Folgendes Problem im abgespeckten Programm, welches sich bei dem gesamten Programm zu einem Fehler entwickelte, wodurch nur **eine** Queue/Topic erzeugt werden konnte.

```
public Admin(){
    queueList = new ArrayList<AdminQueueDoctor>(); //List for the queue objects
}

/**
 * mit dieser methode soll eine neue queue erstellt werden.
 * Diese queue soll einer liste (wird im konstruktor erzeugt) hinzugefügt werden
 *
 * Problem: Ich füge mit "NEW <queueName1>" eine neue queue einer liste hinzu.
 * Nun lasse ich mir zum überprüfen den Namen dieser queue ausgeben (queueList.get(0).queueName())
 * Bis dahin passt alles.
 * WENN
 * ich aber dann noch eine queue hinzufüge "NEW <queueName1>" und nochmals den Name der zuvor
 * erstellten queue (queueList.get(0).queueName()) ausgeben möchte,
 * dann wird mir aber der Name der zweiten erzeugten queue zurückgegeben.
 *
 * Probiers aus:
 * NEW <queueName1> ---> ausgabe: queueName1
 * NEW <queueName2> ---> ausgabe: queueName2
 * @param server
 * @param queueName
 */
public void newDoctor(String server, String queueName) {
    AdminQueueDoctor queue = new AdminQueueDoctor("tcp://" + server + ":61616", queueName);
    queueList.add(queue);
    System.out.println(queueList.get(0).queueName() + "    " + queueList.size() + "    " + queueList.get(0));
    System.out.println(queueList.get(0).queueName() + "    " + queueList.size() + "    " + queueList.get(0));

}
```

2.8 Zeitaufzeichnung

Übung	Zeitaufwand	Ort/Datum
Theoretisches Verständnis	200 min	Zuhause 12.06.2016
Umsetzung in Java	960 min	Zuhause 12.06.2016 Zuhause 13.06.2016 Zuhause 14.06.2016 Zuhause 15.06.2016 Zuhause 16.06.2016
Protokoll	200 min	Zuhause 16.06.2016
Gesamt	1360 min	

2.9 GitHub Repository

https://github.com/rborsos-tgm/MOM_z

2.10 Quellen

[1] http://www.java-programmieren.com/bilder/jms-architektur.jpeg
[2] http://www.itwissen.info/definition/lexikon/load-balancing-Lastausgleich.html
[3] http://docs.oracle.com/javaee/1.4/tutorial/doc/JMS3.html#wp78636