|  |
| --- |
| 4AHIT |
| DezSys06 |
| JMSChat |
|  |
| **Paul Kalauner, Simon Wortha** |
| **21.11.2014** |

|  |
| --- |
| GitHub Repolink: https://github.com/pkalauner-tgm/DezSys06 |

Inhaltsverzeichnis

[Aufgabenstellung/Requirementsanalyse 2](#_Toc404336495)

[Arbeitsaufteilung 3](#_Toc404336496)

[Zeitabschätzung & Zeitaufzeichnung 3](#_Toc404336497)

[Zeitschätzung 3](#_Toc404336498)

[Zeitaufzeichnung 3](#_Toc404336499)

[Gesamt 3](#_Toc404336500)

[Klassendiagramm 4](#_Toc404336501)

[Abweichung der Angabe 5](#_Toc404336502)

[Funktion 5](#_Toc404336503)

[Unittests 6](#_Toc404336504)

[Lessons Learned 7](#_Toc404336505)

[Quellen 8](#_Toc404336506)

# Aufgabenstellung/Requirementsanalyse

Implementieren Sie eine Chatapplikation mit Hilfe des Java Message Service. Verwenden Sie Apache ActiveMQ ([http://activemq.apache.org](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:   
    
  vsdbchat <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> [<ip\_des\_benutzers>]: <Nachricht>
* Zusätzlich zu dem Chatroom kann jedem Benutzer eine Nachricht in einem persönlichen Postfach (JMS Queue) hinterlassen werden. Der Name des Postfachs ist die IP Adresse des Benutzers (Eindeutigkeit).  
    
  Nachricht an das Postfach senden:   
  MAIL <ip\_des\_benutzers> <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.

Gruppenarbeit: Die Arbeit ist in einer 2er-Gruppe zu lösen und über das Netzwerk zu testen! Abnahmen, die nur auf localhost basieren sind unzulässig und werden mit 6 Minuspunkten benotet!

# Arbeitsaufteilung

|  |  |  |
| --- | --- | --- |
| **Aufgabe** | **Name** | **geschätzter Zeitaufwand** |
| UML-Diagramm | Paul Kalauner, Simon Wortha | 20 Minuten |
| Konsolenverarbeitung | Paul Kalauner | 20 Minuten |
| Chatrooms | Paul Kalauner, Simon Wortha | 30 Minuten |
| private Nachrichten | Paul Kalauner, Simon Wortha | 20 Minuten |
| Testing | Paul Kalauner, Simon Wortha | 40 Minuten |
| Protokoll | Paul Kalauner, Simon Wortha | 30 Minuten |

# Zeitabschätzung & Zeitaufzeichnung

## Zeitschätzung

Schätzung: 2 Stunden, 40 Minuten

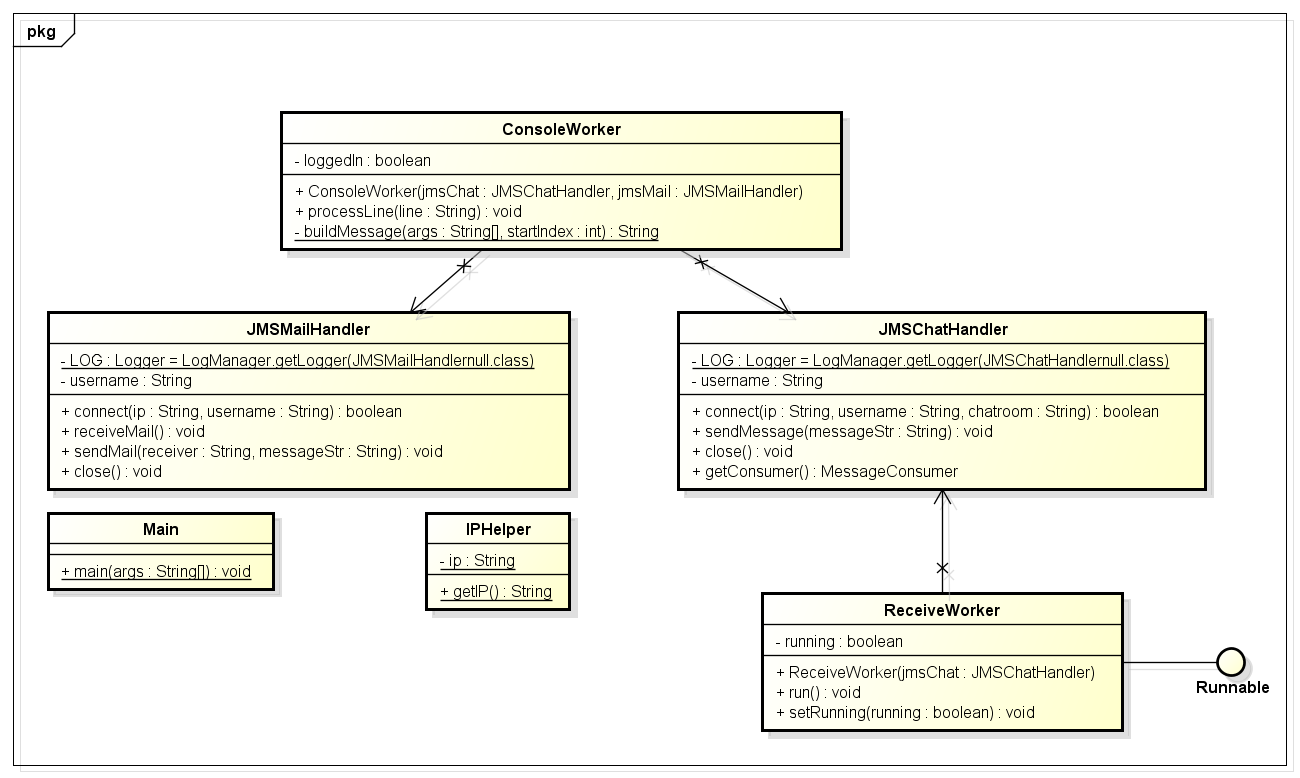
## Zeitaufzeichnung

|  |  |  |
| --- | --- | --- |
| **Aufgabe** | **Name** | **Zeitaufwand** |
| UML-Diagramm | Paul Kalauner, Simon Wortha | 15 Minuten |
| Konsolenverarbeitung | Paul Kalauner | 15 Minuten |
| Chatrooms | Paul Kalauner, Simon Wortha | 35 Minuten |
| private Nachrichten | Paul Kalauner, Simon Wortha | 30 Minuten |
| Testing | Paul Kalauner, Simon Wortha | x Minuten |
| Protokoll | Paul Kalauner, Simon Wortha | x Minuten |

## Gesamt

x Stunden, x Minuten

# Klassendiagramm



# Abweichung der Angabe

Die in unserem Programm verwendeten IPs der Benutzer sind public. Daher können verschiedene Benutzer die gleiche IP haben.

Um trotzdem einem eindeutigen Benutzer eine Mail senden zu können, muss beim Mail-Kommando Benutzername **und** IP angegeben werden.

**Beispiel:**

MAIL Name@IP Das ist eine Nachricht!

# Funktion

1. Start des Programmes: java –jar Filename.jar

2. Verbinden mit Chatroom:

vsdbchat <ip\_message\_broker> <benutzername> <chatroom>

3. private Nachrichten an Benutzer senden:

MAIL Name@IP Das ist eine Nachricht!

4. eigene private Nachrichten abfragen: MAILBOX

5. Chatroom verlassen und Programm beenden: EXIT

# Unittests

Die Unittests befinden sich im JAR-File und wurden mit Javadoc ausführlich kommentiert.

# Lessons Learned

**- Verwendung von JMS und ActiveMQ [1]**

Verbindung herstellen:

ConnectionFactory connectionFactory = **new** ActiveMQConnectionFactory(username, ActiveMQConnection.***DEFAULT\_PASSWORD***, ip);

connection = connectionFactory.createConnection();

connection.start();

session = connection.createSession(**false**, Session.***AUTO\_ACKNOWLEDGE***);

Destination destination = session.createTopic(chatroom);

consumer = session.createConsumer(destination);

producer = session.createProducer(destination);

producer.setDeliveryMode(DeliveryMode.***NON\_PERSISTENT***);

Nachricht senden:

TextMessage message;

**try** {

message = session.createTextMessage(username + "@" + IPHelper.*getIP*() + " " + messageStr);

producer.send(message);

} **catch** (JMSException e) {

***LOG***.error("Fehler beim Senden der Nachricht");

}

Nachrichten empfangen:

TextMessage message;

**try** {

message = (TextMessage) jmsChat.getConsumer().receive();

**if** (message != **null**) {

System.***out***.println(message.getText());

message.acknowledge();

}

} **catch** (JMSException e) {

***LOG***.error("Fehler beim Empfangen der Nachricht", e);

}

# Quellen

[1]: Apache ActiveMQ. Verfügbar unter:

<http://activemq.apache.org/>

[abgerufen am 21.11.2014]