

MOM - JMS-Chat

SYT / DEZSYS06



25. November 2014

Erceg, Kritzl

4AHITT

Inhalt

[1. Aufgabenstellung 2](#_Toc404703925)

[2. detaillierte Arbeitsaufteilung mit Aufwandsabschätzung 3](#_Toc404703926)

[2.1 Aufwandabschätzung 3](#_Toc404703927)

[2.2 Arbeitsaufteilung für die Implementierung des Programms 3](#_Toc404703928)

[2.2.1 Package „connection“ 3](#_Toc404703929)

[2.2.2 Package „display“ 3](#_Toc404703930)

[2.2.3 Package „handler“ 3](#_Toc404703931)

[3. anschließende Endzeitaufteilung 4](#_Toc404703932)

[3.1 Erceg 4](#_Toc404703933)

[3.2 Kritzl 4](#_Toc404703934)

[3.3 Gesamtsumme 4](#_Toc404703935)

[4. Designüberlegung 5](#_Toc404703936)

[4.1 Abbildung 5](#_Toc404703937)

[4.2 Überlegungen zur Struktur 6](#_Toc404703938)

[4.2.1 Package „connection“ 6](#_Toc404703939)

[4.2.2 Package „handler“ 7](#_Toc404703940)

[4.2.3 Package „display“ 7](#_Toc404703941)

[5. Arbeitsdurchführung 8](#_Toc404703942)

[6. Testfälle 10](#_Toc404703943)

[6.1 Starten des Programms 10](#_Toc404703944)

[6.2 Befehl „HELP“ 10](#_Toc404703945)

[6.3 Fehlermeldung 10](#_Toc404703946)

[6.4 Programm beenden 10](#_Toc404703947)

[6.5 Einloggen 11](#_Toc404703948)

[6.6 Beitreten eines Chatraums 11](#_Toc404703949)

[6.7 Senden und Empfangen von persönlichen Nachrichten 12](#_Toc404703950)

[6.8 Benutzer wechseln 12](#_Toc404703951)

[7. Lessons learned 13](#_Toc404703952)

[8. Quellenangaben 14](#_Toc404703953)

*Github-Link:* [*https://github.com/serceg-tgm/DezSys06-JMS\_Chat*](https://github.com/serceg-tgm/DezSys06-JMS_Chat)

*Github-Tag: erceg\_kritzl\_dezsys06\_v1*

# 1. Aufgabenstellung

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 dem Schlüsselwort EXIT verlassen. Der Benutzer verlässt den Chatraum, die anderen Teilnehmer sind davon nicht betroffen.

# 2. detaillierte Arbeitsaufteilung mit Aufwandsabschätzung

## 2.1 Aufwandabschätzung

|  |  |
| --- | --- |
| Teilaufgabe | benötigte Gesamtzeit |
| UML-Diagramm erstellen | 180 Minuten (3 Stunden) |
| Implementierung des Programms inkl. JavaDoc | 420 Minuten (7 Stunden) |
| Testen des Programms | 180 Minuten (3 Stunden) |
| Protokoll schreiben | 120 Minuten (2 Stunden) |
|  |  |
| *Gesamt* | **900 Minuten (15 Stunden)** |

## 2.2 Arbeitsaufteilung für die Implementierung des Programms

### 2.2.1 Package „connection“

|  |  |  |
| --- | --- | --- |
| Klassen/Interfaces | Erceg | Kritzl |
| Connectable | x |  |
| ConnectTopic | x |  |
| JMSClient |  | x |
| JMSServer | x |  |
| Message |  | x |
| MessageBehavior |  | x |

### 2.2.2 Package „display“

|  |  |  |
| --- | --- | --- |
| Klassen/Interfaces | Erceg | Kritzl |
| CLI | x |  |
| Display | x |  |
| Executor | x |  |

### 2.2.3 Package „handler“

|  |  |  |
| --- | --- | --- |
| Klassen/Interfaces | Erceg | Kritzl |
| CommandType |  | x |
| Handler |  | x |
| InputHandler |  | x |

# 3. anschließende Endzeitaufteilung

## 3.1 Erceg

|  |  |  |
| --- | --- | --- |
| Arbeit | Datum | Zeit in Minuten |
| UML | 18.11.2014 | 120 Minuten |
| Protokoll | 20.11.2014 | 40 Minuten |
| JavaDoc | 22.11.2014 | 30 Minuten |
| Protokoll | 23.11.2014 | 120 Minuten |
| Programm testen | 25.11.2014 | 120 Minuten |
|  |  |  |
| *Gesamt* | *25.11.2014* | **430 Minuten (7 h 10 min)** |

## 3.2 Kritzl

|  |  |  |
| --- | --- | --- |
| Arbeit | Datum | Zeit in Minuten |
| UML | 18.11.2014 | 120 Minuten |
| Protokoll | 20.11.2014 | 40 Minuten |
| Implementierung | 20.11.2014 | 180 Minuten |
| Implementierung | 21.11.2014 | 230 Minuten |
| Programm testen | 25.11.2014 | 260 Minuten |
|  |  |  |
| *Gesamt* | *25.11.2014* | **830 Minuten (13 h 50 min)** |

## 3.3 Gesamtsumme

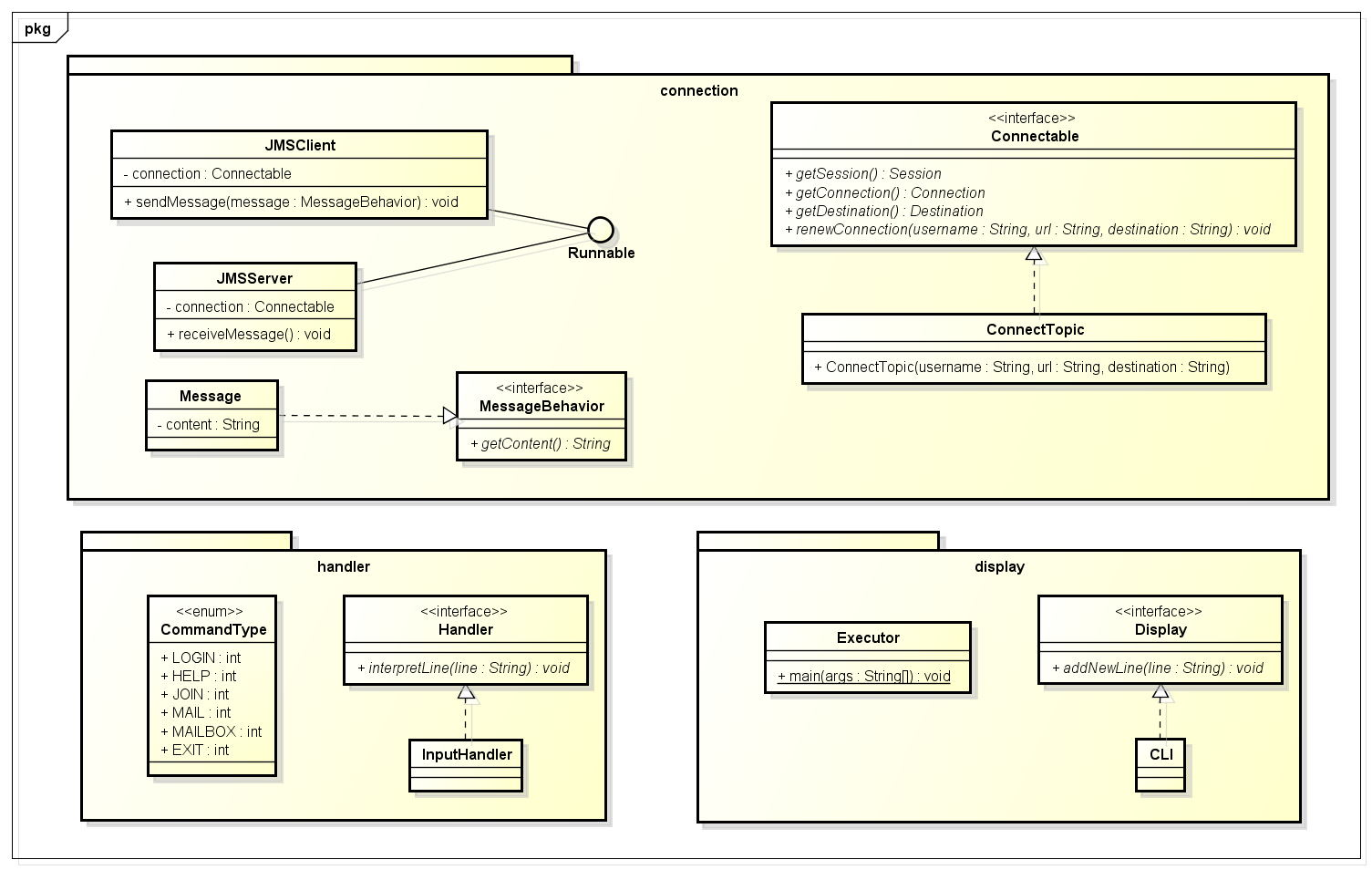
Insgesamt haben wir für diese Übung **21 Stunden** benötigt. Geschätzt wurden 15 Stunden, daher lag unsere Einschätzung ziemlich daneben.

Aufgrund einer plötzlichen Krankheit vom Teammitglied Erceg hatte Kritzl eine etwas höhere Arbeitszeit als Erceg.

# 4. Designüberlegung

## 4.1 Abbildung

Das UML-Diagramm wurde mit dem Programm „Astah“ erstellt.



## 4.2 Überlegungen zur Struktur

Wir haben uns überlegt, unser Programm in 3 Packages unterzuordnen:

**1.) connection**

In diesem Package befinden sich die Klassen, die für die Verbindung zwischen dem Client und des JMS-Systems benötigt werden. Ebenfalls ist eine Message vorhanden, die den entsprechenden Nachrichteninhalt enthält.

**2.) handler**

Die Eingaben des Benutzers werden auf spezielle Befehle geprüft, wie z.B. dem Beitreten und dem Verlassen eines Chatraums.

**3.) display**

Die Anzeige der Eingaben des Benutzers (Hostname, Nachrichteninhalt usw.) und der erhaltenen Nachrichten wird dargestellt. Ebenfalls ist in diesem Package die Main-Klasse vorhanden.

### 4.2.1 Package „connection“

**Klassen, die im Package enthalten sind:**

1.) JMSClient

* implementiert das Interface Runnable
* ist für das Senden der Nachricht zuständig

2.) JMSServer

* implementiert das Interface Runnable
* ist für das Erhalten der Nachricht zuständig

3.) Connectable (Interface)

* beinhaltet die nötigen Bestandteile einer Verbindung zu dem JMS-System

4.) ConnectTopic

* implementiert das Interface Connectable
* im Konstruktor werden die notwendigen Attribute (Username, URL, Ziel) im Parameter angegeben

5.) MessageBehavior (Interface)

* die Methode „getContent()“ wird vorgeschrieben, welche den Inhalt einer Nachricht zurückgeben soll

6.) Message

* implementiert das Interface MessageBehavior

### 4.2.2 Package „handler“

**Klassen, die im Package enthalten sind:**

1.) CommandType (Enum-Klasse)

* beschreibt, welche Kommandos zur Verfügung stehen

2.) Handler (Interface)

* schreibt die Methode „interpretLine“ vor, welche das Interpretieren der Eingabe ermöglicht, z.B. das Eintreten in einen Chat, das Schreiben einer Mail

3.) InputHandler

* implementiert das Interface Handler

### 4.2.3 Package „display“

**Klassen, die im Package enthalten sind:**

1.) Display (Interface)

* schreibt die Methode „addNewLine“ vor, welche einen Text in der Commandline hinzufügt

2.) CLI

* implementiert das Interface Display

3.) Executor

* enthält die Main-Methode und ist dadurch für den Aufruf des Servers und Clients zuständig

# 5. Arbeitsdurchführung

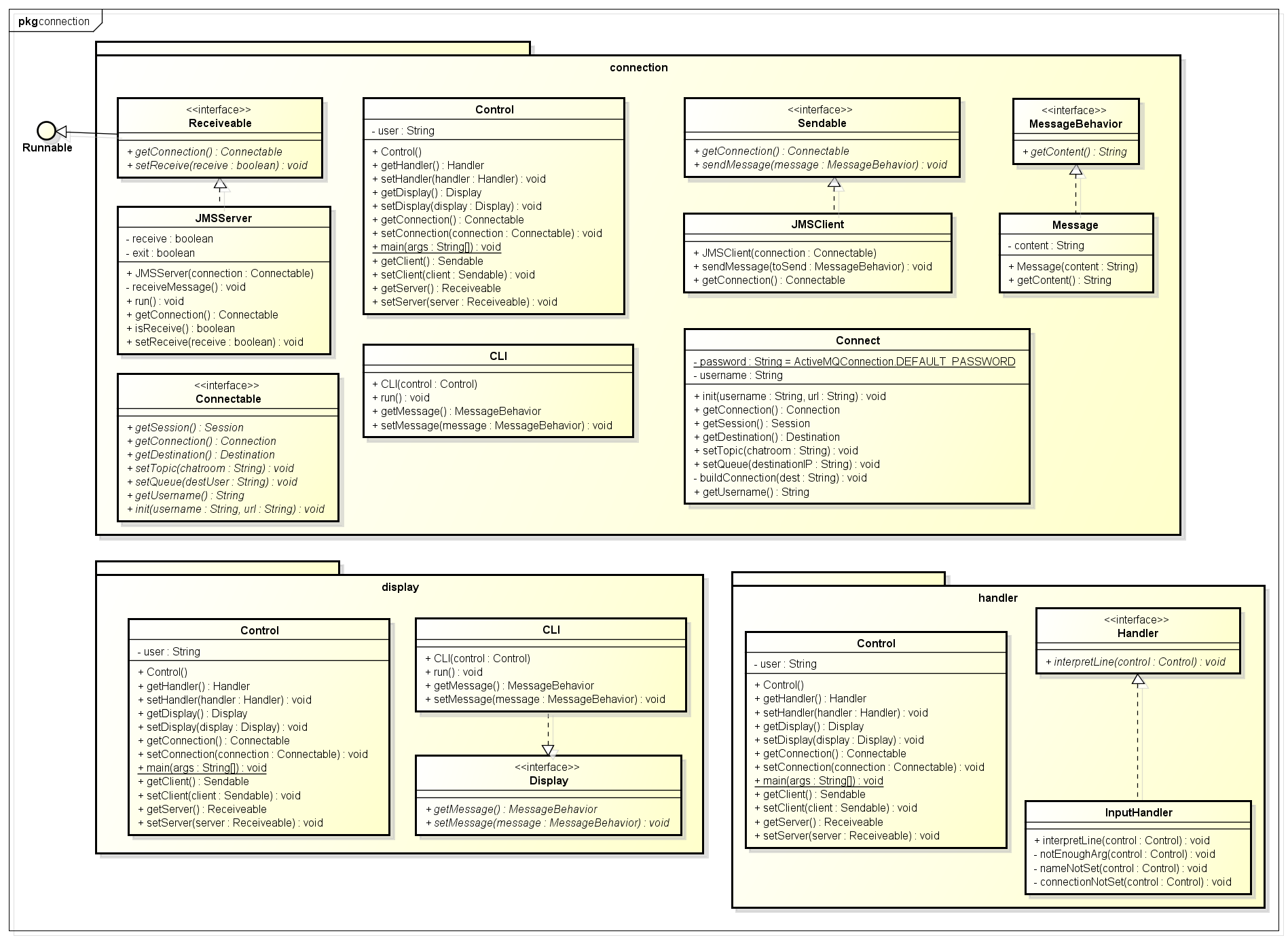
Um die Middleware ActiveMQ als Message Broker zu verwenden, luden wir uns die Version 5.7 der Snapshot-Binary herunter [1].

Die gedownloadete ZIP-Datei wurde entpackt und daraufhin wurde die Windows-Batchdatei „activemq“ im Unterordner „bin“ gestartet. Somit konnte die Webseite in einem beliebigen Browser unter „localhost:8181“ aufgerufen werden.

Da die Chatapplikation mit Hilfe des Java Message Service implementiert wird, wurde das vom Herr Professor Micheler auf e-Learning zur Verfügung gestellte JAR-File heruntergeladen und in unser Projekt importiert.

Nach der Designüberlegung konnte das Programm implementiert werden. Folgende Schritte wurden in folgender Reihenfolge durchgeführt:

* Erstellung der Interfaces
* Implementierungen:
  + Message
  + InputHandler
  + CLI
  + Connection
  + JMSClient
  + JMSServer
  + Control
* Funktionalität überprüfen (während der Implementierungen)
* „System.out.println“-Zeilen auf Logger umgeändert [2]

Da wir während der Implementierung auf einige Verbesserungen bezüglich der Struktur gekommen sind, sieht unser finales UML-Diagramm folgendermaßen aus:

# 6. Testfälle

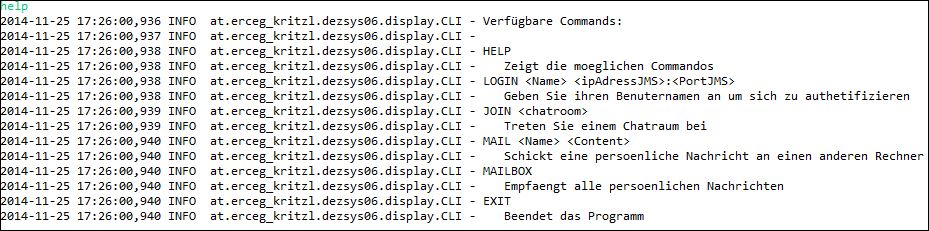
## 6.1 Starten des Programms

Beim Starten des Programms wird folgende Meldung angezeigt:



## 6.2 Befehl „HELP“

Möchte der Benutzer wissen, welche Befehle ihm zur Verfügung stehen, kann er diese unter Eingabe von „HELP“ (Befehle sind nicht case-sensitive) einsehen:



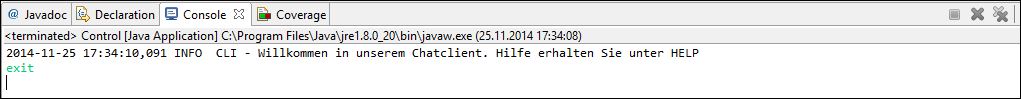
## 6.3 Fehlermeldung

Der Benutzer kann die Befehle „JOIN“ (Chatraum beitreten), „MAIL“ (persönliche Nachricht schicken) und „MAILBOX“ (Empfangen der persönlichen Nachrichten) nicht ausführen, wenn er sich vorher nicht mit dem Befehl „LOGIN“ eingeloggt hat. Falls er die vorher genannten Befehle trotzdem ausprobiert, wird eine Fehlermeldung angezeigt (Screenshot zeigt Fehlermeldung bei Eingabe von MAIL):



## 6.4 Programm beenden

Das Programm kann jederzeit mit dem Befehl „EXIT“ beendet werden:



## 6.5 Einloggen

Das Einloggen geschieht durch die Eingabe „LOGIN <Benutzername> <IP-Adresse von JMS>:<Port von JMS>“:



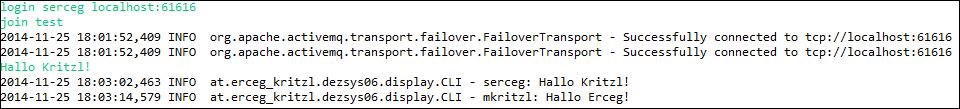
## 6.6 Beitreten eines Chatraums

Nach dieser Eingabe kann der Benutzer einem Chatraum beitreten oder eine persönliche Nachricht senden. Getestet wird nun das Beitreten eines Chatraums durch den Befehl „JOIN <Chatraumname>“:



Beliebig viele Nachrichten können nun gesendet werden. Alle Benutzer, die in diesem Chatraum angemeldet sind, erhalten diese. Bei folgendem Beispiel befinden sich die Benutzer „serceg“ und „mkritzl“ in dem Chatraum „test“ und senden untereinander Nachrichten.

*Benutzer „serceg“:*



*Benutzer „mkritzl“:*



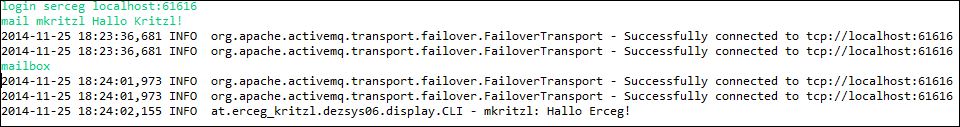
Wie man sehen kann, erhalten beide die Nachrichten vom jeweils anderen Benutzer im Chatraum.

## 6.7 Senden und Empfangen von persönlichen Nachrichten

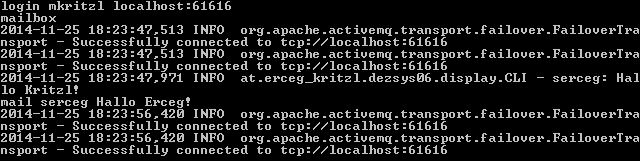
In dem vorherigen Testfall konnten sich mehrere Benutzer in einem Chatraum befinden. Möchten „serceg“ und „mkritzl“ jedoch nur untereinander Nachrichten senden, damit niemand außer sie sie empfangen können, wird der Befehl „MAIL <Name des Benutzers, an den die Nachricht gesendet werden soll> <Nachrichteninhalt>“ verwendet. Mit dem Befehl „MAILBOX“ können die vom anderen Benutzer gesendeten Nachrichten eingesehen werden.

In folgendem Beispiel senden „serceg“ und „mkritzl“ sich jeweils persönliche Nachrichten:

*Benutzer „serceg“:*



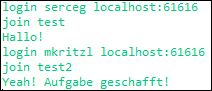
*Benutzer „mkritzl“:*



Wie man auch hier sehen kann, erhalten beide die persönlichen Nachrichten vom jeweils anderen Benutzer.

## 6.8 Benutzer wechseln

Der Benutzer kann jederzeit gewechselt werden. Dazu muss man nur die Eingabe „LOGIN „LOGIN <Benutzername> <IP-Adresse von JMS>:<Port von JMS>“ erneut durchführen und der vorher aktive Benutzer ist abgemeldet und der neue nun eingeloggt:



# 7. Lessons learned

* Die Verwendung von Interfaces ist für die Erweiterbarkeit des Programms ausschlaggebend.
* Queues und Topics sind von der Verwendung unter JMS nicht sehr unterschiedlich. Der einzige große Unterschied ist, dass Topics Subscribers haben können und Queues nicht. Wenn aus einer Queue die Nachricht geholt wurde, ist sie in der Middleware nicht mehr vorhanden.
* Bei den Topics gibt es einen Unterschied zwischen Subscribers und Consumers. Cosumers erhalten Nachrichten nur, wenn diese auch verbunden sind und Subscribers erhalten Nachrichten auch, wenn diese erst später wieder eine Verbindung aufbauen
* Ein Subscriber bekommt Nachrichten im Nachhinein nur, wenn er schon einmal eingeloggt war.
* Wenn Subscribers oder Consumers nicht geschlossen werden, kommt es zum Versuch eine weitere Verbindung aufzubauen, was bei den Subscribers zu einem Error führt und bei Consumers unnötige Verbindungen verbleiben.
* Der Server darf für die Zeit, in der die Verbindung neu gesetzt wird, keine Nachrichten empfangen, weil es sonst dazu kommen kann über eine falsche Verbindung Nachrichten empfangen zu wollen, was in einer endlosen Warteschleife endet.
* Wenn eine Connection während dem Receiven der Nachrichten geändert wird, wird diese Wartehaltung unterbrochen.
* Messages können Attribute, wie z.B. Sendername, hinzufügen.

# 8. Quellenangaben

[1] The Apache Software Foundation (2004, 2011). SNAPSHOT Binaries [Online]. Available at: <https://repository.apache.org/content/repositories/snapshots/org/apache/activemq/apache-activemq/> [abgerufen am 14.11.2014].

[2] The Apache Software Foundation (1999, 2014). Configuration [Online]. Available at: <http://logging.apache.org/log4j/2.0/manual/configuration.html>

[abgerufen am 21.11.2014]