7.1.2015

Hampl & Kritzl

Rückwärtssalto

A05 - Metadata

Inhalt

[1. Angabe 1](#_Toc408385561)

[2. detaillierte Arbeitsaufteilung mit Aufwandsabschätzung 1](#_Toc408385562)

[2.1 Aufwandabschätzung 2](#_Toc408385563)

[2.2 Arbeitsaufteilung für die Implementierung des Programms 2](#_Toc408385564)

[2.2.1 Package „connection“ 2](#_Toc408385565)

[2.2.2 Package „display“ 2](#_Toc408385566)

[2.2.3 Package „handler“ 2](#_Toc408385567)

[3. anschließende Endzeitaufteilung 3](#_Toc408385568)

[3.1 Erceg 3](#_Toc408385569)

[3.2 Kritzl 3](#_Toc408385570)

[3.3 Gesamtsumme 3](#_Toc408385571)

[4. Designüberlegung 4](#_Toc408385572)

[4.1 Abbildung 4](#_Toc408385573)

[4.2 Überlegungen zur Struktur 5](#_Toc408385574)

[4.2.1 Package „connection“ 5](#_Toc408385575)

[4.2.2 Package „handler“ 6](#_Toc408385576)

[4.2.3 Package „display“ 6](#_Toc408385577)

[5. Arbeitsdurchführung 7](#_Toc408385578)

[6. Testfälle 9](#_Toc408385579)

[6.1 Starten des Programms 9](#_Toc408385580)

[6.2 Befehl „HELP“ 9](#_Toc408385581)

[6.3 Fehlermeldung 9](#_Toc408385582)

[6.4 Programm beenden 9](#_Toc408385583)

[6.5 Einloggen 10](#_Toc408385584)

[6.6 Beitreten eines Chatraums 10](#_Toc408385585)

[6.7 Senden und Empfangen von persönlichen Nachrichten 11](#_Toc408385586)

[6.8 Benutzer wechseln 11](#_Toc408385587)

[7. Lessons learned 12](#_Toc408385588)

[8. Quellenangaben 13](#_Toc408385589)

# 1. Angabe

Erstelle ein Java-Programm, dass Connection-Parameter und einen Datenbanknamen auf der Kommandozeile entgegennimmt und die Struktur der Datenbank als EER-Diagramm und Relationenmodell ausgibt (in Dateien geeigneten Formats, also z.B. PNG für das EER und TXT für das RM)

Verwende dazu u.A. das ResultSetMetaData-Interface, das Methoden zur Bestimmung von Metadaten zur Verfügung stellt.

Zum Zeichnen des EER-Diagramms kann eine beliebige Technik eingesetzt werden für die Java-Bibliotheken zur Verfügung stehen: Swing, HTML5, eine WebAPI, ... . Externe Programme dürfen nur soweit verwendet werden, als sich diese plattformunabhängig auf gleiche Weise ohne Aufwand (sowohl technisch als auch lizenzrechtlich!) einfach nutzen lassen. (also z.B. ein Visio-File generieren ist nicht ok, SVG ist ok, da für alle Plattformen geeignete Werkzeuge zur Verfügung stehen)

Recherchiere dafür im Internet nach geeigneten Werkzeugen.

Die Extraktion der Metadaten aus der DB muss mit Java und JDBC erfolgen.

Im EER müssen zumindest vorhanden sein:

* korrekte Syntax nach Chen, MinMax oder IDEFIX
* alle Tabellen der Datenbank als Entitäten
* alle Datenfelder der Tabellen als Attribute
* Primärschlüssel der Datenbanken entsprechend gekennzeichnet
* Beziehungen zwischen den Tabellen inklusive Kardinalitäten soweit durch Fremdschlüssel nachvollziehbar. Sind mehrere Interpretationen möglich, so ist nur ein (beliebiger) Fall umzusetzen: 1:n, 1:n schwach, 1:1
* Kardinalitäten

Fortgeschritten (auch einzelne Punkte davon für Bonuspunkte umsetzbar)

* Zusatzattribute wie UNIQUE oder NOT NULL werden beim Attributnamen dazugeschrieben, sofern diese nicht schon durch eine andere Darstellung ableitbar sind (1:1 resultiert ja in einem UNIQUE)
* optimierte Beziehungen z.B. zwei schwache Beziehungen zu einer m:n zusammenfassen (ev. mit Attributen)
* Erkennung von Sub/Supertyp-Beziehungen

# 2. detaillierte Arbeitsaufteilung mit Aufwandsabschätzung

## 2.1 Aufwandabschätzung

|  |  |  |  |
| --- | --- | --- | --- |
| Arbeitspaket | Person | Schätzung | Erledigt |
| **Usereingabe** | Hampl |  |  |
| Parsen | Hampl |  |  |
| Überprüfen | Hampl |  |  |
| Hilfe | Kritzl |  |  |
| **Datenbankverbindung** | Kritzl |  |  |
| MySQL | Kritzl |  |  |
| **Auslesen Metadaten** | Kritzl |  |  |
| Datenbank | Kritzl |  |  |
| Tabelle | Kritzl |  |  |
| Attribut | Kritzl |  |  |
| Referenz | Kritzl |  |  |
| **Export in EER** | Hampl |  |  |
| Syntaktisch | Hampl |  |  |
| optisch | Hampl |  |  |
| **Export in RM** | Hampl |  |  |
| Syntaktisch | Hampl |  |  |

## 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

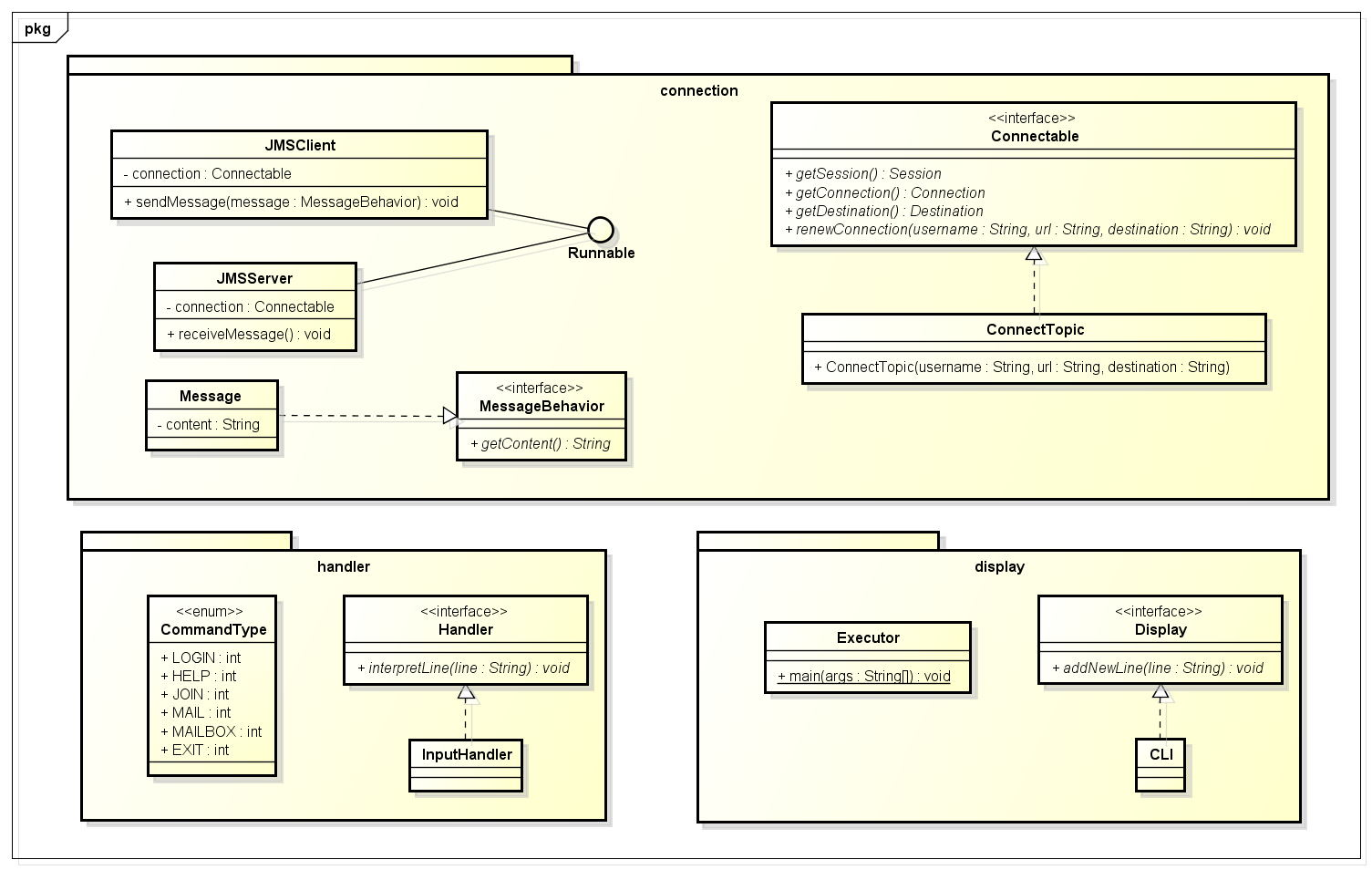
Insgesamt haben wir für diese Übung **21 Stunden** benötigt. Geschätzt wurden 15 Stunden, daher lag unsere Einschätzung ziemlich daneben.

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

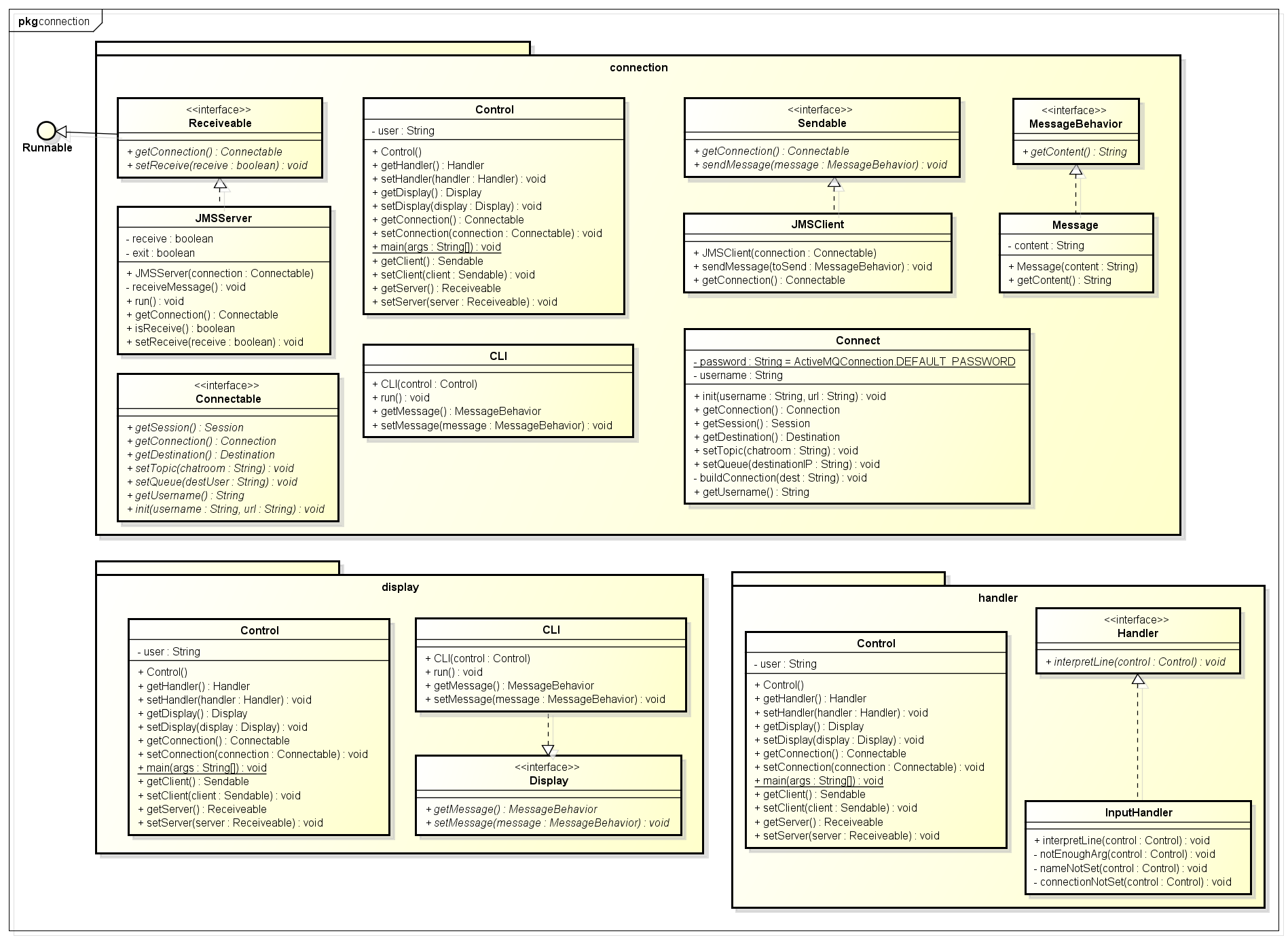
Um die Middleware ActiveMQ als Message Broker zu verwenden, luden wir uns die Version 5.7 der Snapshot-Binary herunter [1].

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

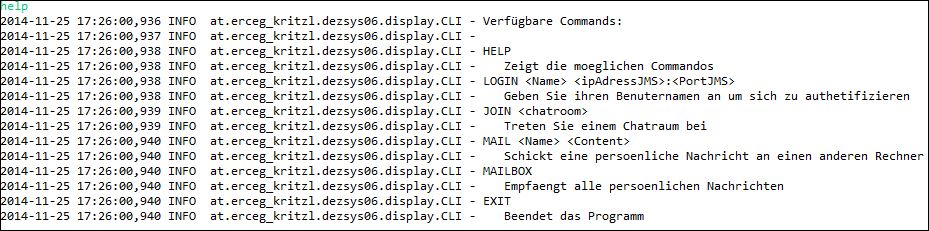
## 6.1 Starten des Programms

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:



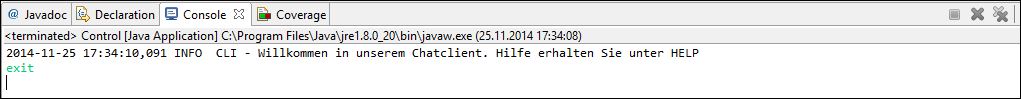
## 6.3 Fehlermeldung

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>“:



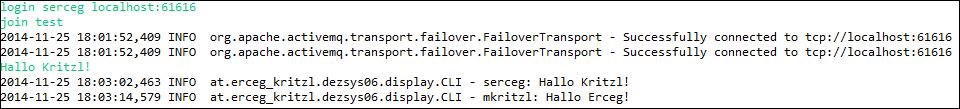
## 6.6 Beitreten eines Chatraums

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“:*



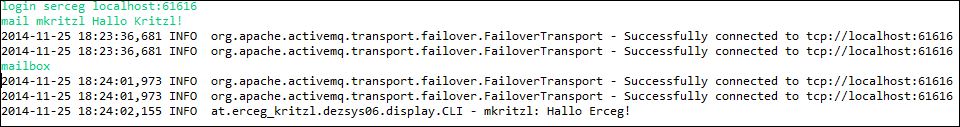
Wie man sehen kann, erhalten beide die Nachrichten vom jeweils anderen Benutzer im Chatraum.

## 6.7 Senden und Empfangen von persönlichen Nachrichten

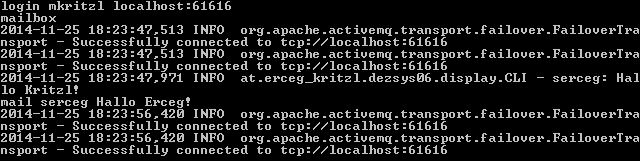
In dem vorherigen Testfall konnten sich mehrere Benutzer in einem Chatraum befinden. Möchten „serceg“ und „mkritzl“ jedoch nur untereinander Nachrichten senden, damit niemand außer sie sie empfangen können, wird der Befehl „MAIL <Name des Benutzers, an den die Nachricht gesendet werden soll> <Nachrichteninhalt>“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



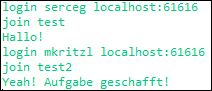
*Benutzer „mkritzl“:*



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]