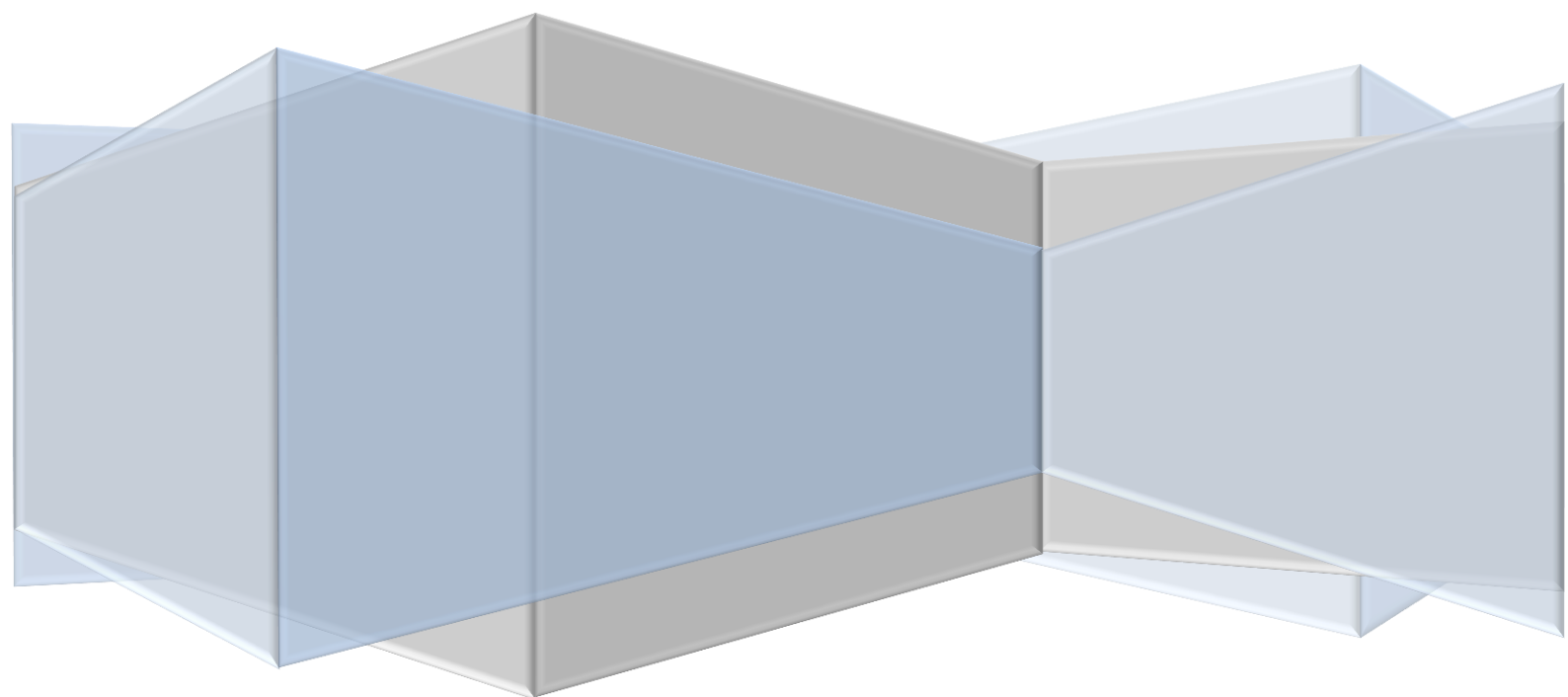


TGM Wien

JMS Chat

DEZSYS-06

Thomas Taschner & Michael Weinberger 4AHITT 2014/15



Inhaltsverzeichnis

| | |
|-------------------------------------------------------------------------------|---|
| Aufgabenstellung | 2 |
| Beschreibung auf Moodle | 2 |
| Designüberlegung | 3 |
| Erster Ansatz | 3 |
| Konkrete Idee | 3 |
| Umsetzung | 4 |
| Detaillierte Arbeitsaufteilung (Aufwandsabschätzung, Endzeitaufteilung) | 5 |
| Aufgabentrennung | 5 |
| Aufwandabschätzung | 5 |
| Endzeitaufteilung | 5 |
| Fazit | 5 |
| Arbeitsdurchführung (Resultate/Niederlagen) | 6 |
| Resultate | 6 |
| Niederlagen | 6 |
| Testbericht | 7 |
| Coverage | 7 |
| Beschreibung | 7 |
| Quellenangaben | 8 |

Aufgabenstellung

Beschreibung auf Moodle

Implementieren Sie eine Chatapplikation mit Hilfe des Java Message Service. Verwenden Sie Apache ActiveMQ 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.

Designüberlegung

Erster Ansatz

Mit den Erkenntnissen des KnockKnock-Servers aus den vorherigen Einheiten haben wir bereits einen guten Einblick bekommen, wie Nachrichtenübermittlung ohne Middleware funktioniert. Mithilfe von Apache ActiveMQ wollen wir es nun auch völlig unterschiedlichen Systemen möglich machen, miteinander zu kommunizieren. Die MOM (,Message Oriented Middleware') handelt als Broker ,in der Mitte' die Kommunikation ab, sodass die Clients nur mehr Befehle zum Senden & Empfangen geben müssen.

Konkrete Idee

Bestehend auf dem Wissen der vorigen Beispiele, gab es die Idee, dass die 2 Mechanismen Sender & Receiver in einer kontrollierten ,Endlosschleife' Nachrichten entgegennehmen und den Parametern entsprechend verschicken. Auf einem System müssen, um beide Funktionen zu erfüllen, sprich die eines vollwertigen Chats mit Senden & Empfangen von Nachrichten zu gewährleisten, also beide Instanzen ausgeführt werden. Wie beschrieben in der Aufgabenstellung soll ein Chatraum (JMS Topic) erstellt werden und während der Laufzeit sollen private Nachrichten verschickt werden können, indem nach einer Abfrage der Text über die JMS Queue verschickt wird, und diese erst zu einem beliebigen Zeitpunkt empfangen werden sollen. Die Funktionsweise der Queue lässt sich so erklären, die Nachricht wird weggeschickt (in die Queue geschrieben), und erst mit receive() Nachricht für Nachricht ausgelesen.

```
// Create the message
TextMessage message = session.createTextMessage(input);
producer.send(message);

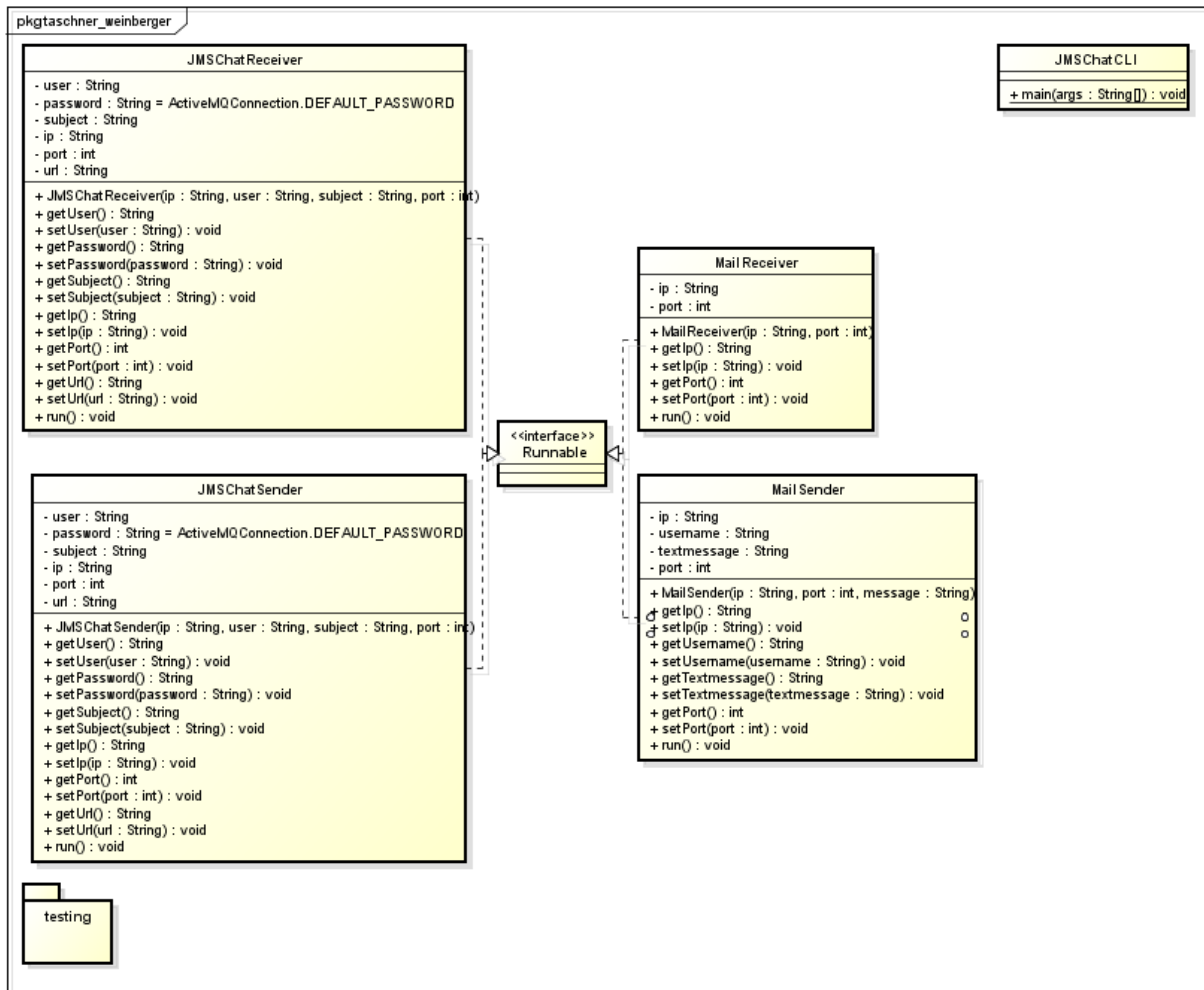
// Start receiving
TextMessage message = (TextMessage) consumer.receive();

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

message.acknowledge();
```

Umsetzung

Unsere Implementierung baut auf exakt dem gleichem Prinzip auf der konkreten Idee auf.



Das von uns erstellte UML-Diagramm zur Aufgabe.

Detaillierte Arbeitsaufteilung (Aufwandsabschätzung, Endzeitaufteilung)

Aufgabentrennung

| WEINBERGER | TASCHNER |
|------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| CLI, Einlesen der Parameter Erhebliche Mithilfe beim Implementieren, Beratung und Lösungsansätze suchen Protokoll, Sourcecode-Dokumentation | Sender/Receiver Implementierung Mailbox Senden/Empfangen Implementierung Testfälle Absicherung der Eingaben, Ausfallsicherheit |

Aufwandabschätzung

Dieses Projekt sollte in einer Gesamtzahl von jeweils *10 Stunden* (gemeinsam) fertiggestellt werden.

Endzeitaufteilung

Das Projekt von Anfang bis Ende nahm im Resümee gut *12,5 Stunden* ein. Besonders bei der Implementierung der Funktionen gab es hin und wieder Fehler, die einige Zeit in Anspruch nahmen und wertvolle Zeit genommen haben.

| CLI | Implementierung | Protokoll | Sourcecode-Doc | Implementierung Mailbox | Testfälle | Debuggen | Endergebnis |
|-----|-----------------|-----------|----------------|-------------------------|-----------|----------|-------------|
| 0,5 | 5 | 1 | 1 | 1,5 | 2 | 2 | 13 h |

Gesamt: 26 Arbeitsstunden (verteilt auf 2 Personen)

Fazit

Die Einschätzung der benötigten Zeit lag *in etwa* an der Menge der tatsächlich verbrauchten.

Arbeitsdurchführung (Resultate/Niederlagen)

Bei der Durchführung stießen wir hin und wieder auf einige mehr oder minder schwerwiegende Fehler.

Resultate

Apache ActiveMQ ist ein (sofern man der Handhabung mächtig ist) sehr nützliches Tool, um Nachrichten einfach und sicher (TCP!) von A nach B zu bringen. Als ‚fauler Programmierer‘ bin ich jedoch Freund der unsicheren, aber schneller zu implementierenden Version, wo direkt mit dem Client kommuniziert wird, was natürlich nur funktioniert, wenn dieser PC unter denselben Umständen agiert. Ein Broker wiederum nimmt diese Hürde, sodass es egal ist, welcher Architektur, welchem Betriebssystem usw. der Client angehört, was im logischen Sinne der Hauptgrund ist, um MOM-Systeme im Einsatz zu haben.

Niederlagen

Wo es Code gibt, gibt es auch Niederlagen: Speziell beim Verbindungs-Aspekt und den erfordernten genauen Angaben der Parameter war es nicht gerade schwierig, Exceptions auszulösen.

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

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

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

Irrtümlicherweise wurden beim Erstellen der Factory die Strings *password* und *url* vertauscht, was leider etwas Zeit kostete.

Nun zu eher technischen Problemen:











Wenn eine fehlerhafte IP-Adresse eingegeben wird, so wechselt das Programm in einen Deadlock-Status. Es war uns unmöglich herauszufinden, ob eine Instanz am (entfernten) Rechner läuft, auch wenn simples anpingen möglich war, jedoch nichts absicherte, dass auch das Programm läuft.

Bitte um genaue Angabe Ihrer Eingaben! Alle zur Laufzeit, nicht aber die Konsolenparameter können zu 100% abgesichert werden.

Ansonsten gab es nur kleine triviale, schnell behebbare Niederlagen, die an dieser Stelle nicht weiter zu erwähnen sind und sich meist nach etwas Hineinlesen erübrigt haben.

Testbericht

Coverage

| | | | | | |
|-------------------------------|-----------------------------------------------------------------------------------|---------|-----|-----|-------|
| ▲ A06 - JMS Chat |  | 61,4 % | 731 | 460 | 1.191 |
| ▲ src |  | 61,4 % | 731 | 460 | 1.191 |
| ▲ taschner_weinberger |  | 42,6 % | 342 | 460 | 802 |
| ▶ JMSChatSender.java |  | 33,3 % | 83 | 166 | 249 |
| ▶ MailSender.java |  | 24,4 % | 40 | 124 | 164 |
| ▶ JMSChatReceiver.java |  | 46,8 % | 81 | 92 | 173 |
| ▶ MailReceiver.java |  | 22,8 % | 23 | 78 | 101 |
| ▶ JMSChatCLI.java |  | 100,0 % | 115 | 0 | 115 |
| ▲ taschner_weinberger.testing |  | 100,0 % | 389 | 0 | 389 |
| ▶ TestJMSChat.java |  | 100,0 % | 389 | 0 | 389 |

Beschreibung

Leider konnte nur eine Testabdeckung des Codes von 61.4% erreicht werden, da die run() Methoden der einzelnen Klassen in JUnit, aufgrund von mangelndem Knowhow, nicht getestet werden konnten. Für erfolgreiche Tests wären wahrscheinlich Mock-Objekte von Nöten, die wir letztes Jahr nur angestreift haben und es daher an der praktischen Anwendung dieser scheitert.

Das Testen der CLI selber und der Getter- und Setter-Methoden verlief jedoch ohne größere Probleme.

Quellenangaben

<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>

<http://www.straub.as/java/jms/basic.html>

<http://activemq.apache.org/hello-world.html>

& einige einzelne Tipps von Kollegen aus der Klasse