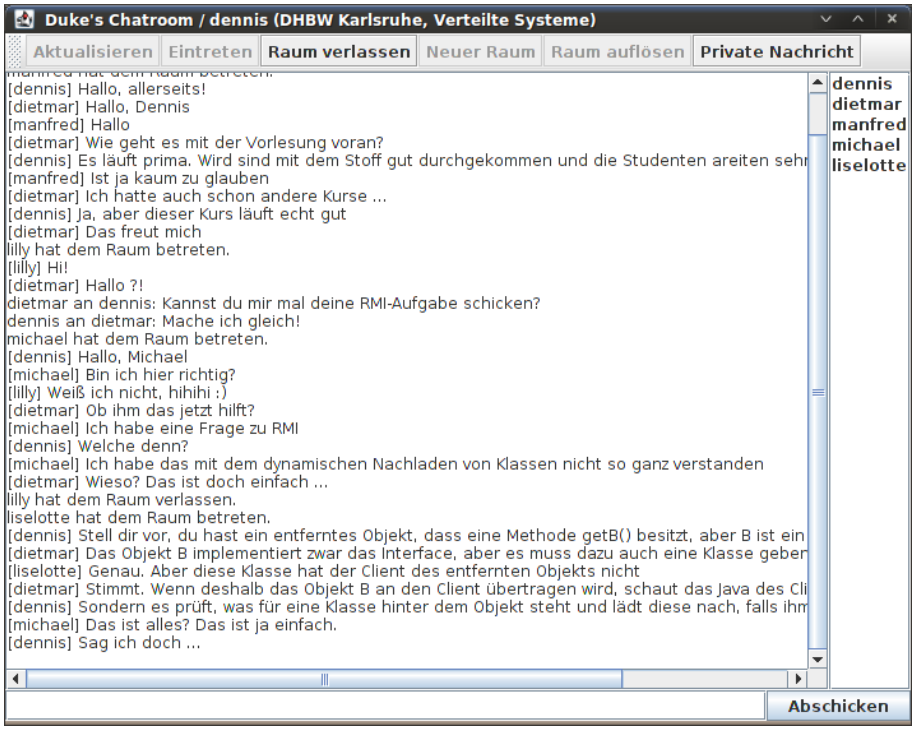
— Gruppenarbeit 3 (P): Duke's Chatroom —

Aufgabe 1: Alle **(Einleitung)**

Da alle etablierten Anbieter von Instant Messangern inzwischen dazu übergegangen sind, ihre Dienste durch aufdringliche Werbung zu finanzieren, hat Ihr Kurs beschlossen, derartige Anwenungen während den Vorlesungen nicht mehr einzusetzen. Stattdessen wollen Sie künftig Ihre eigene Chatanwendung benutzen und somit den Herstellern ein Schnippchen schlagen. Diese Anwendung soll ganz im Sinne klassischer Chatprotokolle wie `IRC`[1] stehen und daher folgende Funktionen bieten:

- Die Anwendung soll von beliebig vielen Benutzern gleichzeitig nutzbar sein
- Jeder Benutzer besitzt ein Pseudonym, unter dem er schreibt
- Die Serveranwendung soll mehrere Chat Rooms verwalten, die von den Benutzern beliebig angelegt und gelöscht werden können
- Ein Chat beschränkt sich daher immer nur auf einen Chat Room
- Zusätzlich sollen sich die Benutzer noch private Nachrichten schicken können.

Somit ist klar, dass es sich bei der Anwendung um eine Client/Server-Anwendung handelt, wobei Sie in dieser Aufgabe sowohl den Client als auch den Server entwickeln müssen. Als kleine Draufgabe sollen Sie dann noch einen Chatbot entwickeln, mit dem man sich unterhalten kann, wenn sonst niemand in einem Chat Room anwesend ist. Eine typische Sitzung könnte so aussehen:



[1] Internet Relay Chat, vgl. http://de.wikipedia.org/wiki/Internet_Relay_Chat

Die Serveranwendung produziert keine nennenswerte Bildschirmausgaben. Sie erbringt einfach nur stillschweigend ihre Dienste. Die Clientanwendung jedoch, noch bevor sie das Hauptfenster zeigt, frägt den Benutzer nach der Adresse des Chatservers und seinem Pseudonym. Anschließend erscheint das Hauptfenster, in dem eine Liste aller verfügbaren Chat Rooms angezeigt wird. Über die Toolbar können dann weitere Räume angelegt oder gelöscht werden. Ebenso kann ein Raum ausgewählt werden, um ihm beizutreten. In diesem Fall wechselt das Fenster auf die oben abgebildete Anzeige, so dass Nachrichten ausgetauscht werden können. Über die Toolbar kann dann der aktuelle Raum wieder verlassen werden, um zur Raumübersicht zu gelangen.

Vorbereitung: Laden Sie sich von Moodle die Aufgabe `Duke's Chatroom` herunter. Dieses Archiv beinhaltet in den drei Ordnern `Client`, `Server` und `Chatbot` die vorprogrammierten Anwendungen. Ihre Aufgabe besteht nun darin, die Anwendungen auf Basis von Sun/Oracle `RMI` fertigzuprogrammieren.

Aufgabe 2: Alle **(Der Chatserver)**

Die Serveranwendung besteht im Wesentlichen aus zwei verschiedenen Arten von entfernt aufrufbaren Objekten: Einem `ChatServer`, der die zentrale Verwaltung der Chat Rooms übernimmt, und beliebig vielen `ChatRoom`-Objekten, welche die Räume darstellen, in denen gechattet wird. Dabei ist das `ChatServer`-Objekt als Quasi-Singleton in der `RMI` Registry abgelegt, damit die Clients darauf zugreifen können. Die `ChatRoom`-Objekte können jedoch nicht mit dem Namensdienst verwaltet werden, da sie erst zur Laufzeit erzeugt werden und es daher keinen festen Namen gibt, unter dem sie abgelegt werden könnten.[2]

Zunächst müssen Sie die Serveranwendung `RMI`-fähig machen. Nehmen Sie daher folgende Anpassungen vor:

1. Sorgen Sie dafür, dass die beiden Interfaces `ChatServer` und `ChatRoom` als Schnittstelle für entfernte Objekte verwendet werden können.
2. Passen Sie dann die beiden Klassen `ChatServerService` und `ChatRoomService` so an, dass sie das jeweilige Interface implementieren und somit entfernt aufrufbar werden.
3. Ändern Sie die `main()`-Methode der Klasse `RunServer`, indem Sie darin ein `ChatServer`-Objekt erzeugen und unter dem Namen `ChatServer` im Namensdienst ablegen.

Nun programmieren Sie die Klasse `ChatServerService` aus:

1. Fügen Sie dem `ChatServer`-Interface und der `ChatServerService`-Klasse folgende Methoden hinzu:

   ```
   public ChatRoom getChatRoom(int nr)
   public ChatRoom createChatRoom(String name)
   public void deleteChatRoom(int nr)
   ```

2. Die Methode `getChatRoom()` soll einfach nur das angeforderte Objekt aus `this.chatRooms` zurückgeben.

[2] Natürlich könnte man schon den Namensdienst hierfür verwenden, aber so sehen Sie den Unterschied zwischen der Kopiersemantik und der Referenzsemantik besser.

3. In `createChatRoom()` müssen Sie erst in einer Schleife über alle Chat Rooms prüfen, ob es bereits einen Raum mit dem übgergebenen Namen gibt. Falls ja, soll einfach `null` zürckgegeben werden. Andernfalls müssen Sie sich durch Aufruf der Methode `getNewId()` eine neue Raumnummer besorgen und ein neues `ChatRoom`-Objekt erzeugen. Übergeben Sie dem Objekt sowohl die Raumnummer als auch seinen Namen und legen Sie das Objekt unter seiner Nummer in der Liste der Chat Rooms ab.

4. In der Methode `deleteChatRoom()` besorgen Sie sich das angeforderte Objekt und prüfen somit, ob es überhaupt in der Liste enthalten ist. Falls ja, rufen Sie die Methode `getUsers().isEmpty()` des Objekts auf, um zu prüfen, ob sich noch jemand in dem Raum befindet. Nur wenn dies nicht zutrifft, können Sie das Objekt aus `chatRooms`-Liste entfernen.

Achtung: Damit Sie die Anwendung kompilieren können, müssen Sie erst noch den nächsten Punkt abarbeiten, weil es im Interface `ChatRoom` die Methode `getUsers()` noch nicht gibt.

Nehmen Sie folgende Änderungen an der Klasse `ChatRoomService` vor:

1. Fügen Sie dem `ChatRoom`-Interface und der `ChatRoomService`-Klasse folgende Methoden hinzu:

   ```
   public int enterRoom(String username)
   public void leaveRoom(String username)
   public void postMessage(String username, String message)
   public Map<Integer, String> getMessages(int lastKnownMessage)
   public List<String> getUsers()
   ```

2. Bevor ein Benutzer mit `enterRoom()` den Raum betritt, prüfen Sie mit der Methode `this.users.contains()`, ob es bereits einen Benutzer mit selbem Namen gibt und fügen den Benutzer der Liste hinzu, falls die Prüfung fehlschlägt. Dann rufen Sie die Methode `addMessage()` auf, um einen Text wie `Dennis betritt den Saal ...` zu posten. Dabei erhalten Sie eine Message ID zurück. Von diesem Wert ziehen Sie eins ab, bevor Sie ihn zurückgeben.

3. In der Methode `leaveRoom()` entfernen Sie den übergebenen Benutzer aus der Benutzerliste und posten einen Text wie `Dannis hat das Gebäude verlassen`. Da diese Methode keinen Rückgabewert bestizt, müssen Sie auch die Message ID nicht berücksichtigen.

4. Die Methode `postMessage()` ruft intern eigentlich nur `addMessage()` auf. Jedoch soll dem geposteten Text in eckigen Klammern der Name des Benutzers vorangestellt werden. Zum Beispiel: `[Dennis] Hallo allerseits!`

5. Die Methode `getMessages()` ist etwas aufwendiger. Erzeugen Sie erst ein `Map<Integer, String>`-Objekt und merken Sie sich den Wert `this.previousId` in einer lokalen Variable. Dann prüfen Sie, ob der übergebene Wert `lastKnownMessage` größer oder gleich -1 ist. Falls ja, führen Sie eine Schleife über alle Zahlen zwischen `lastKnownMessage + 1` und der gemerkten `previousId` aus, in der Sie aus `this.messages` die entsprechenden Nachrichten auslesen und in der eben erzeugten `Map` speichern. Zum Schluss geben Sie die `Map` an den Aufrufer zurück.

6. In `getUsers()` geben Sie einfach nur das `users`-Objekt zurück.

Das war's. Der Server ist nun fertig. Achten Sie darauf, dass Sie den Befehl `rmiregistry` ausführen, um den Namensdienst zu starten, bevor Sie den Server starten. Dabei muss die `RMI` Registry aus demselben Verzeichnis gestartet werden, in dem die kompilierten Serverklassen liegen und solange laufen, wie Client und Server aktiv sind.

Aufgabe 3: Alle **(Swing-Client)**

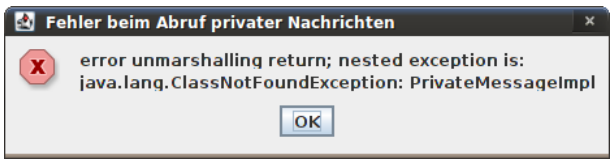
Das war ja nicht so schwer. Gehen wir daher gleich zur Clientanwendung über, die natürlich auch noch nicht ganz fertig programmiert wurde. Nehmen Sie folgende Änderungen an den Klassen der Clientanwendung vor:

1. Ergänzen Sie den Konstruktor der Klasse `MainWindowController` so, dass aus der `RMI` Registry ein `ChatServer`-Stub gelesen wird.
2. Ergänzen Sie die Methode `createRoom()` derselben Klasse so, dass in der vorletzten Zeile die Methode `createChatRoom()` des entfernten `ChatServers` aufgerufen wird.
3. Fügen Sie der Methode `writePrivateMessage()` derselben Klasse einen Aufruf der Methode `sendPrivateMessage()` des `ChatServers` hinzu, damit die eingegebene, private Nachricht auch wirklich verschickt wird.
4. Erweitern Sie die Methode `postMessage()` so, dass die eingegebene Nachricht an den gerade aktiven Chat Room geschickt wird.

Das war's! Damit ist auch die Clientanwendung fertig. Allerdings können Sie diese noch nicht kompilieren, weil Ihnen dazu die Class-Dateien der entfernten Schnittstellen fehlen. Kopieren Sie daher folgende Dateien der kompilierten Serveranwendung in das Verzeichnis der Clientanwendung:

- `ChatRoom.class`
- `ChatServer.class`
- `PrivateMessage.class`

Jetzt klappt's auch mit dem Compiler. Kompilieren Sie daher die Anwendung und führen Sie diese aus. Anschließend probieren Sie die Anwendung aus, indem Sie sie mehrmals starten und sich jeweils mit unterschiedlichen Benutzernamen anmelden. Legen Sie ein Paar Chat Rooms an und chatten Sie miteinander. Was passiert, wenn Sie einem Benutzer eine private Nachricht schicken? Was hat die Fehlermeldung zu bedeuten?



Offensichtlich ist hier was schief gegangen. Sowohl die Serveranwendung als auch die Clientanwendung konnten Sie ohne Probleme kompilieren, dennoch funktioniert der Austausch privater Nachrichten nicht. Das liegt daran, dass die Clientanwendung mit dem Objekt, dass es als private Nachricht erhält,

nichts anfangen kann, obwohl ihr das Interface bekannt ist, das dieses Objekt implementiert. Das genügt aber nicht. Damit das empfangene Objekt wieder deserialisiert werden kann, muss seine Klasse bekannt sein und lokal vorliegen. In diesem Fall handelt es sich um die Klasse `PrivateMessageImpl`, welche der Clientanwendung fehlt. Eine Möglichkeit ist nun, die fehlende Class-Datei ins Verzeichnis der Clientanwendung zu kopieren. Das würde im speziellen Fall funktionieren, allerdings könnte der Server unter bestimmten Umständen auch noch andere Objekte schicken, die ebenfalls das `PrivateMessage`-Interface umsetzen, wodurch derselbe Fehler entstünde.

Rufen Sie daher die Anwendung mit folgendem Befehl auf, der Java mitteilt, wo es fehlende Klassen nachladen kann:

```
$ java -Djava.rmi.server.codebase=http://www.windows3.de/misc/DHBW-VertSys/
       RunClient
```

Damit erreichen Sie, dass fehlende Klassen aus dem Internet von der Adresse `http://www.windows3.de/misc/DHBW-VertSys/` nachgeladen werden. Probieren Sie es aus. Starten Sie die Anwendung mit dem veränderten Befehl und schicken Sie sich private Nachrichten. Wenn der Server jetzt ein `PrivateMessageImpl`-Objekt an den Client überträgt, lädt dieser aus dem Internet die Datei `http://www.windows3.de/misc/DHBW-VertSys/PrivateMessageImpl.class` mit der kompilierten Klasse runter und erzeugt ein Objekt davon. Voraussetzung dafür ist natürlich, dass Sie keine Änderungen am Interface `PrivateMessage` und seiner Implementierung in `PrivateMessageImpl` vorgenommen haben. Denn dann würde die Anwendung eine nicht passende Klasse nachladen und ebenfalls eine Fehlermeldung erzeugen.

Aufgabe 4: Alle **(Chatbot)**

Nach der Pflicht nun die Kür. Weil nicht immer jemand von Ihrem Kurs im Chat anwesend ist, sollen Sie hier einen Chatbot schreiben, der sich mit dem Server verbindet und als normaler Teilnehmer an einem Chat teilnimmt. Der Bot soll daher einfach in einem Raum abwarten, bis ein echter Benutzer eintritt und sich mit ihm unterhalten will. Ein Gespräch könnte dann so aussehen, wobei man sich mit dem Bot auf englisch unterhalten muss:

```
[Eliza Weizenbaum] Hey folks.
[Dennis] How are you?
[Eliza Weizenbaum] why do you ask that?
[Dennis] Because I feel fine
[Eliza Weizenbaum] how does that make you feel?
[Dennis] Why are you saying that?
[Eliza Weizenbaum] perhaps the answer lies within yourself?
[Dennis] Are you sure?
[Eliza Weizenbaum] please consider whether you can answer your own question.
[Dennis] I need to go. Bye
[Eliza Weizenbaum] Thank you, that will be $150.  Have a good day!
```

Nehmen Sie folgende Änderungen am Hauptprogramm vor:

1. Ändern Sie die Klasse `RunChatbot` so ab, dass aus dem Namensdienst ein `ChatServer`-Objekt ausgelesen und in der Variable `server` gespeichert wird.

2. Weiter unten rufen Sie den Server auf, um alle vorhandenen Chat Rooms zu erfahren. Zeigen Sie die Räume zusammen mit ihrer Nummer in einer Liste an, so dass der Anwender einen Raum auswählen kann. Falls keine Räume vorhanden sind, soll eine Fehlermeldung erscheinen und das Programm beendet werden.

3. Am Ende derselben Methode erzeugen Sie einen neuen `ElizaThread` und führen diesen nebenläufig aus. Achten Sie darauf, dass Sie dem Thread ein neues `Eliza`-Objekt sowie die Variable `lastMessage` übergeben.

Jetzt programmieren Sie noch den `ElizaThread` zuende. Zunächst soll der Thread in einer Endlosschleife so lange laufen, wie das `quit`-Kennzeichen nicht gesetzt ist. Dabei soll zwischen jedem Durchlauf der Schleife mindestens eine Sekunde gewartet werden. Davon abgesehen programmieren Sie in der Schleife folgenden Algorithmus:

1. Abruf aller Teilnehmer im Chat Room sowie aller neuen Nachrichten. Hierfür müssen Sie sich die ID der zuletzt bearbeiteten Nachricht in der Variable `lastKnownMessage` speichern und diese der Methode `getMessages()` des `ChatRoom`-Objekts übergeben.

2. Wenn keine Benutzer im Raum sind, nichts unternehmen.

3. Wenn Benutzer im Raum sind und sich der Bot noch nicht vorgestellt hat, soll eine Begrüßung gepostet werden. Diese erhalten Sie durch Aufruf der Methode `getFirstMessage(String username)` des `Eliza`-Objekts. Der Methode müssen Sie den Benutzernamen des Bots übergeben.

4. Wenn neue Nachrichten vorliegen, soll nur die jüngste Nachricht tatsächlich beantwortet werden. Dabei müssen Sie sich die ID der jüngsten Nachricht in `lastKnownMessage` merken, damit Sie im nächsten Schleifendurchlauf nur die neuen Nachrichten vom Server abholen.

5. Wenn mehr als sechzig mal zwar Teilnehmer im Raum anwesend sind, aber keiner was gesagt hat, soll der Bot eine Nachricht schreiben, dass er sich langweilt. Diese Nachricht erhalten Sie durch Aufruf der Methode `getIdleMessage()` des `Eliza`-Objekts. Anschließend beginnt die Zählung wieder bei null.

6. Falls der Bot auf eine Nachricht antworten will, muss erst geprüft werden, ob es sich um eine Nachricht von einem anderen Benutzer handelt. Hierfür können Sie folgende Prüfung verwenden (`message` beinhaltet dabei die Nachricht, auf die geantwortet werden soll):

   ```
   !message.startsWith(this.ownPrefix) && message.startsWith("[")
   ```

   Mit folgenden Befehlen erhalten Sie dann die eigentliche Nachricht, die Sie an die Methode `talk()` von `Eliza` übergeben müssen. Die Antwort posten Sie natürlich im Chat Room.

   ```
   int split = text.indexOf("]") + 2;
   String text = message.substring(split);
   ```

Jetzt haben Sie es geschafft. Starten Sie alle drei Anwendungen dieser Aufgabe und unterhalten Sie sich ein wenig mit Eliza. Wenn Sie wollen, können Sie sich auch den Quellcode der `Eliza`-Klasse anschauen, um zu verstehen, warum der Bot die eine oder andere Antwort gibt. Schaffen Sie es, dem Bot weitere Schlüsselbegriffe und Antworten hinzuzufügen?