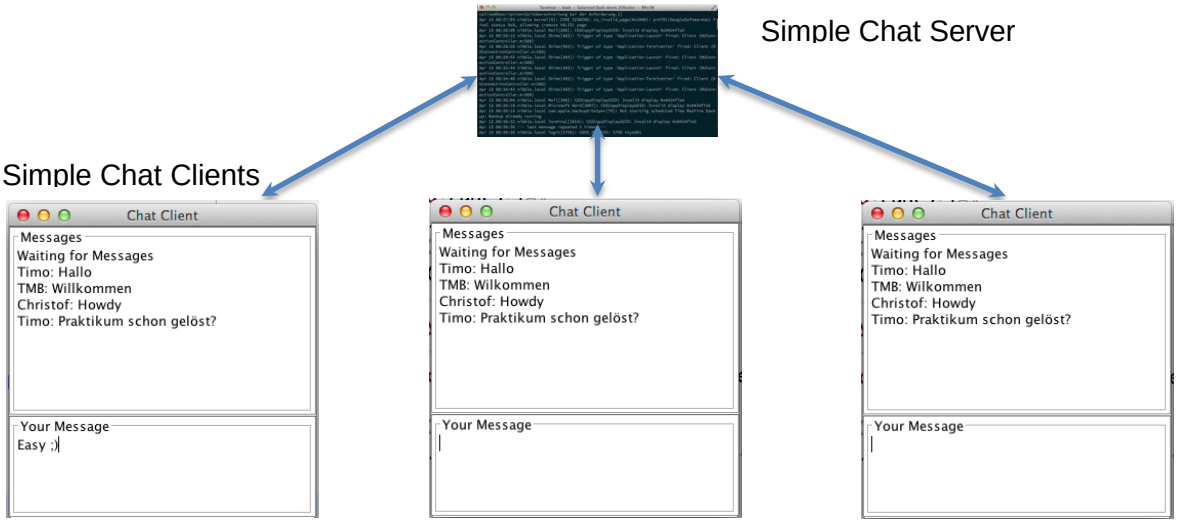
Praktikum Networking

In diesem Praktikum üben Sie, wie Programme übers Netzwerk kommunizieren können.
Am Ende des Praktikums haben Sie

- einen Netzwerkclient programmiert
- einen Netzwerkserver programmiert (Single- & Multi-Threaded)
- ein Anwendungsprotokoll implementiert
- das Java Logging-System angewendet

Simple Chat

In dieser Aufgabe erstellen Sie einfaches Chat-System, in welchem mehrere Benutzer miteinander Nachrichten austauschen können. Jede Nachricht, die ein Benutzer mittels Chat-Client an den Server schickt, wird unmittelbar an alle verbundenen Clients verteilt (inkl. dem ursprünglichen Absender).

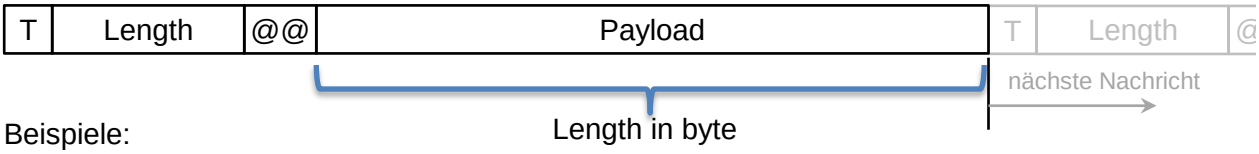


Simple Chat Protokoll

Damit die unterschiedlichen Clients und Server zueinander kompatibel sind, ist das folgende Anwendungsprotokoll vorgegeben.

Als Transportprotokoll wird TCP verwendet.

Nachrichten sind *textbasiert* und bestehen aus **Header** und **Payload**, welche durch ‚@@' voneinander getrennt sind (siehe Grafik). Der Header enthält einen ***Typ*** (1 Charakter ohne Umlaute = 1 Byte) und die ***Länge*** (in Bytes) des Payload als String. Der ***Payload*** *besteht aus UTF-8 Text* und enthält die eigentliche Nachricht. Je Nach Typ wird diese unterschiedlich interpretiert (siehe Tabelle)



Beispiele:

- Message: `M5@@Hallo`
- Command: `C14@@REGISTER Peter`
- Confirm: `C0@@`
- Error: `E23@@You are not registered!`

Folgende Nachrichtentypen sind vorgesehen:

Typ	Richtung	Bedeutung	Payload
'C'	Client → Server	**Command**	Kommando an den Server: • **`REGISTER`** *Chatname* • **`QUIT`**
	Server → Client	**Confirm**	Kommando erfolgreich (kein Payload / leer)
'E'	Server → Client	**Error**	Fehlermeldung
'M'	Client → Server	**Message**	Gesendete Nachricht
	Server → Client		Empfangene Nachricht

Um die Implementierung einfach zu halten, können alle Kommandos jederzeit geschickt werden. Nach dem Kommando (**Command**) muss auf eine Bestätigung (**Confirm**) oder Fehlermeldung (**Error**) gewartet werden.

Solange der Benutzer seinen Chatnamen nicht registriert hat (**REGISTER**), wird er auf dem Server als **`Anonymous-`*`Nr`*** geführt, wobei die Nummer für verschiedene Benutzer/Verbindungen unterschiedlich sein muss. Chatnamen, die mit „Anonymous" beginnen, sind nicht erlaubt (→ Fehlermeldung). Mit **`QUIT`** wird die Verbindung beendet. Das **Confirm** sollte jedoch noch abgewartet werden.

Nachrichten (**Message**) eines Clients werden vom Server an alle zurzeit verbundenen Clients verschickt (inkl. Absender). Wie der Client Benutzereingaben als Kommando und Nachricht unterscheidet, ist Ihnen überlassen.

Beim Verteilen der Nachrichten fügt der Server der Nachricht als **Prefix** den Namen des Absenders plus einen Doppelpunkt hinzu (z.B. „`Peter Muster: Hallo`" bzw. „`Anonymous-Nr:…`", wenn nicht registriert).

Aufgaben:

a) Implementieren Sie den **Chat-Client**. Da die Kommandozeile sich nicht gut eignet um gleichzeitig Ein- und Ausgaben durchzuführen, stellen wir ein fertiges, einfaches GUI mit Ein- und Ausgabefeld zur Verfügung (`Client` bzw. `ClientView`). Beim Start von Client wird automatisch eine Instanz der Klasse `ClientNetworkHandler` in einem separaten Thread gestartet. In dieser Klasse implementieren Sie die Netzwerk-Kommunikation mit dem Server. Server-Adresse und -Port werden beim Start als Kommandozeilenparameter mitgegeben. Text im „Your Message"-Feld wird beim Drücken von Enter/Return an die Methode `sendMessage()` geschickt. Wenn Sie eine Nachricht im Messages-Textfeld ausgeben wollen, rufen Sie die Methode `recievedMessage()` auf.

b) Implementieren Sie einen einfachen **Single-Threaded Chat-Server**, welcher das obige Simple Chat Protokoll implementiert. Den Server können Sie als normales Java-Programm in der Konsole/Eclipse starten. Als Startpunkt können Sie z.B. die `TCPEchoServer` Beispiele aus der Vorlesung verwenden.
Verwenden Sie für die Ausgabe auf der Konsole ein Java-Logger-Objekt, wie in der Vorlesung gezeigt. Versehen Sie die ausgegebenen Log-Meldungen mit dem passenden Log-Level (z.B. Fehler → SEVERE oder WARNING, Status → INFO, Commands → FINE, Meldungen → FINEST)
   - Arbeitet Ihr Client-Programm mit dem Server zusammen?
   - Wie viele Clients können Sie bedienen?

   Führen Sie das Resultat dem Betreuer vor und erklären Sie die wesentlichen Merkmale.

c) Erweitern Sie ihren Server zu einem **Multi-Threaded Chat-Server**, der „beliebig“ viele Clients bedienen kann. Auch hier können Sie von den Unterrichtsbeispielen ausgehen.
   - Wie organisieren Sie die Kommunikation mit den anderen Threads?
   - Müssen Sie den Client auch anpassen?
   - Funktioniert Ihr Server auch mit den Clients anderer Studierender?

   Führen Sie das Resultat dem Betreuer vor und erklären Sie die wesentlichen Merkmale.

d) Konfigurieren Sie Ihren **Logger** so, dass auf der Konsole nur noch Status und Fehlermeldungen ausgegeben werden. Kommandos und Meldungen sollten jedoch auch noch in eine Log-Datei (z.B. ChatLog) ausgegeben werden.
   Fügen Sie dazu einen `java.logging.FileHandler` dem Logger hinzu. Evtl. müssen Sie diesem auch einen geeigneten `java.logging.Formater` zur Seite stellen, um ein passendes Log-Format zu erhalten. Welchen würden Sie wählen?
   Zeigen Sie das Resultat (Konsole & Logfile) dem Betreuer.

e) (Optional) Passen Sie Ihren Server so an, dass er nur noch Meldungen an einen Client weiterleitet, wenn dieser sich registriert hat (**REGISTER**). Ansonsten wird eine Fehlermeldung zurückgegeben. Das bedeutet, dass Sie verschiedene **Zustände** (States) einführen (z.B. ***Unregistered*** & ***Registered***), in welchen der Server unterschiedlich reagiert.
   - Welche Nachrichten sind in welchem Zustand noch sinnvoll/erlaubt?
   - Wie reagieren Sie wenn für einen Zustand illegale Nachrichten eintreffen?