SoSe 2017

Nichtsequentielle und verteilte Programmierung

8. Übungsblatt

1. Aufgabe (4 P.) Implementierung eines Client/Server in Java unter Verwendung des TCP- und UDP-Protokolls.

a) (1 P.) Implementieren Sie einen Echo-Server. Dieser soll mit einem *java.net.ServerSocket* auf eingehende Verbindungen von Clients warten. Sobald ein Client verbunden ist, sollen alle Daten, die von diesem empfangen wurden, unverändert an den Client zurückgesendet werden. Es darf davon ausgegangen werden, dass sich immer nur ein Client zeitgleich verbinden möchte.

b) (1 P.) Implementieren Sie einen Client um den Echo-Dienst zu nutzen. Der Client soll Texteingaben des Nutzers entgegennehmen und danach an den Server senden. Die Antwort des Servers soll anschließend ausgegeben werden. Für die Umsetzung des Clients können Sie das Framework "*Chat Client*" nutzen.

c) (1 P.) Implementieren Sie einen Echo-Server. Dieses Mal auf Basis von UDP. Dieser soll eingehende Datagramme wieder zurück zu ihrem Sender verschicken. Es darf davon ausgegangen werden, dass immer nur ein Client zeitgleich den Service nutzt.

d) (1 P.) Implementieren Sie einen Client, um den Echo-Dienst zu nutzen. Der Client soll Texteingaben des Nutzers entgegennehmen und anschließend an den Server senden. Die Antwort des Servers soll ausgegeben werden. Für die Umsetzung des Clients können Sie auch hier das Framework "Chat Client" nutzen.

2. Aufgabe (14 P.) Implementierung eines Client/Server Chats unter Verwendung des TCP-Protokolls.

a) (0 bis 4 Bonuspunkte) Importieren Sie die Frameworks "*Chat Client*" und "*Chat Server*" in eine IDE Ihrer Wahl und lesen Sie die dazugehörige Dokumentation. –ODER- Implementieren Sie Ihre eigene grafische Oberfläche für Server und/oder Client. Dabei muss nicht derselbe Funktionsumfang erreicht werden. Grundlegende Funktionen eines Chat-Programms sollten aber vorhanden sein.

b) (10 P.) In dieser Aufgabe soll ein einfaches Chat-Programm implementiert werden, dem eine Client-Server Architektur zugrunde liegt. Die Übertragung von Daten soll dabei direkt mittels TCP realisiert werden. Weitere Einschränkungen bzgl. des Lösungswegs werden nicht gemacht. Allerdings soll Ihre Lösung folgende Kriterien erfüllen:

- Beliebig viele Clients können sich zu einem Server verbinden und über diesen Nachrichten miteinander austauschen. Wer der Autor einer Nachricht ist, soll für die Empfänger ersichtlich sein.
- Das Trennen von Clients soll weder beim Server noch bei den Clients zu Fehlern führen oder sie unbrauchbar machen.
- Das Stoppen des Servers soll bei den Clients zu keinen Fehlern führen. Clients sollen danach dazu fähig sein, sich zu einem neuen Server zu verbinden.

- Der Server soll die derzeitig verbundenen Clients anzeigen. Namensänderungen müssen dabei vorerst nicht berücksichtigt werden.

c) (4 P. und 4 Bonuspunkte.) Des Weiteren soll **>eins<** der folgenden Features umgesetzt werden:

- Sobald ein Client seinen Namen ändert, werden der Server und alle anderen Clients sofort über diese Namensänderung informiert. Der Server aktualisiert die Liste der derzeit verbundenen Clients entsprechend.
- Clients haben die Möglichkeit anderen Clients zuzuflüstern – also eine private Nachricht abzusenden, die sich gezielt an einen einzelnen Empfänger richtet und nicht für den globalen Chat bestimmt ist. Der Empfänger soll private und globale Nachrichten voneinander unterscheiden können.
  Beispiel: "/*private* **bob** Hi. I love you <3". Der Server schickt diese Nachricht nur an den Client mit dem Namen **bob** weiter.
- Über die Server-Konsole besteht die Möglichkeit, einzelne Clients auszuschliessen und für eine bestimmte Zeit stumm zu schalten.
  Beispiel 1: "/*exclude* **bob**" der Client mit dem Namen **bob** wird vom Server ausgeschlossen.
  Beispiel 2: "/*mute* **bob** 60" – der Client mit dem Namen **bob** kann für eine Minute keine Nachrichten mehr versenden.

Bitte machen Sie in Ihrer Abgabe kenntlich, welches der Features Sie umgesetzt haben. (Für freiwillig implementierte Features gibt es je 2 Bonuspunkte.)

Geben Sie Ihren Client und Ihren Server jeweils als ausführbare .jar Datei ab. (Die Datei sollte sich also in einem Terminal mit dem Befehl "`java -jar MyChatServer.jar`" starten lassen.) Ihren Quellcode können Sie mit in die .jar Datei packen oder in einem separates .zip-Archive abgeben.

3. Aufgabe (14 P.) Client/Server Chat UDP oder Multicast.

Für das Lösen dieser Aufgabe ist wieder das fünfte Framework zu verwenden. Alternativ können natürlich auch selbst erstellte Oberflächen benutzt werden.

a) (10 P.) Erneut soll ein Chat-Programm implementiert werden, das die auf Übungsblatt 7 in Aufgabe 5b) genannten Kriterien erfüllt. Dieses Mal jedoch mit **UDP oder Multicast**.

Sollten Sie sich für **UDP** entscheiden, soll der gesamte Verkehr (wie zuvor) über den Server abgewickelt werden. Verwenden Sie die Klassen java.net.DatagramSocket und java.net.DatagramPacket.

Wenn Sie die Aufgabe mit **Multicast** lösen, so sollen sich die Clients initial mittels TCP an den Server wenden, um von diesem die verwendete MulticastAdresse zu beziehen. (Die MulticastAdresse darf hardcoded sein.) Für die anschließende Kommunikation der Clients sollte der Server nicht mehr zwingend notwendig sein. Verwenden Sie die Klassen java.net.MulticastSocket (ggf. *java.net.DatagramSocket*) und *java.net.DatagramPacket*.

Hinweis: Der MulticastSocket leitet sich vom DatgramSocket ab und arbeitet ebenfalls mit Datagrammen. Besonders wissbegierige Kursteilnehmer-Innen können daher

beruhigt zu dieser Lösung greifen, ohne Gefahr zu laufen, etwas zu verpassen. Zeitgleich lässt sich mit Multicast die wohl elegantere Lösung finden.

b) (4 P. + 4 Bonuspunkte.) Des Weiteren soll **>eins<** der folgenden Features umgesetzt werden:

- Über den Server soll es möglich sein mit dem Kommando "/status" eine Liste aller verbundenen Clients auszugeben. Für jeden Client werden dabei die Verbindungsdauer (in Stunden, Minuten, Sekunden), die Anzahl der gesendeten Nachrichten und seine IP+Port ausgegeben.
- Sollten Nachrichten unerlaubte Wörter enthalten, so werden diese Wörter zensiert - durch Sternchen ersetzt. Werden von einem Client mehr als drei Nachrichten mit verbotenen Wörtern empfangen, so wird er dauerhaft stumm geschaltet (neue Nachrichten von ihm werden nicht mehr angezeigt). Diese Funktionalität kann entweder im Server (bei UDP) oder im Client (bei Multicast) umgesetzt werden. Die Liste der verbotenen Wörter kann in einem Array festgelegt werden und sollte mindestens folgende Wörter enthalten: framework, synchronisation, javafx.
- Über die Server-Konsole besteht die Möglichkeit, einzelne Clients zu beenden (inklusive Benutzeroberfläche) und Clients haben die Möglichkeit für die Textfarbe ihres Chats aus mindestens 5 verschiedenen Farben zu wählen. Für diese Funktionen sind Änderungen in der Klasse ClientController und dem Interface ClientGUI nötig. Sollten Sie Probleme mit einer IllegalStateException bekommen, so geben Sie "platform.runlater" in eine Suchmaschine Ihrer Wahl ein (… und beheben das Problem).
  Beispiel 1: (Beenden von Clients über Server): /terminate **bob**
  Beispiel 2: (Änderung der Textfarbe): /color RED

Bitte machen Sie in Ihrer Abgabe kenntlich, welches der Features Sie umgesetzt haben. (Für freiwillig implementierte Features gibt es je 2 Bonuspunkte.)

Geben Sie Ihren Client und Ihren Server jeweils als ausführbare `.jar` Datei ab. (Die Datei sollte sich also in einem Terminal mit dem Befehl "`java –jar MyChatServer.jar`" starten lassen.) Ihren Quellcode können Sie mit in die .jar Datei packen oder in einem separaten .zip-Archiv abgeben.

Hinweis: Um Daten für die Übertragung leichter zu serialisieren, können allerlei Dateiformate verwendet werden.