Aufgabe 2
TCP- Chat

Von den Studenten:
Emanuela Giovanna Calabi, 13186
Angelo Rosace, 13386

1. Beschreiben Sie Ihren Algorithmus und zeigen Sie auf, welche Interaktionen zwischen den Clients und dem Server stattfinden.

Die Algorithmus verwaltet die Kommunikation zwischen ein Client und ein Server auf Grundlage des TCP-Protokolls.
Client und Server sind in unserem Fall in zwei verschiedene Java-Klassen implementiert.
Das Server benutzt ein Port, das vom Benutzer ausgewählt wird, auf dem die Kommunikation durchgeführt werden kann. Auf diesem Port wird ein ServerSocket hergestellt, das wartet auf eingehende Verbindungen von Clients. Dies wird mithilfe eines Thread (ServerListener) durchgeführt, so dass die Kommunikation während des Wartens nicht blockiert wird.
Das ServerSocket bietet methode um die Kommunikation zu starten an. Die Erstellung der Kommunikation hängt von die Java-Klasse “Connection”, die Thread -extendiert- ab. Jeder Client hat seinen eigenen Verbindungsthread, so dass der Server von jedem von ihnen gleichzeitig lesen und schreiben kann.
Diese Klasse handelt die Kommunikation indem sie zwei DataStreams vom Socket öffnet: ein DataInputStream und ein DataOutputStream. Das erste lässt der Client am Server schreiben und das zweite teilt die Nachrichten, die ein Client gesendet hat, zu alle die anderen Clients, die mit dem Server verbunden sind, weiter. Der Server überträgt die Nachricht von einem Client zu allen anderen über die Methode Broadcast, die über alle Verbindungen zu den Clients iteriert.
Das Client verwendet eine vom Benutzer eingegebene Serveradresse und einen Port, an dem die Verbindung hergestellt wird. Nach dem Erstellen der Verbindung wird der Benutzer nach einem eindeutigen Benutzernamen gefragt, der im Chatraum identifiziert werden soll. Der Benutzername wird auch vom Server als Schlüssel für die HashMap verwendet, die alle Verbindungen zu den Clients enthält.
jeder Client verwendet zwei Threads: ein Receiver zum Lesen von Eingaben des Benutzers und ein Sender zum Senden an den Server und einen zum Empfangen von Eingaben vom Server und zum Drucken auf der Konsole.
Wenn die Kommunikation gestartet wird das Client kann eine Nachricht schreiben. Ein Client kann die Kommunikation enden wenn er “/quit” sendet.

3. Testen Sie Ihre Implementierung und berichten Sie mögliche Bugs und/oder unerwartetes Verhalten, das Sie nicht erklären können.

Wenn der Benutzer nach einer Portnummer für den Server gefragt wird und eine Nummer kleiner als 1024 eingegeben wird, wird eine Ausnahme ausgelöst. Dies liegt daran, dass Java standardmäßig die Verwendung vordefinierter Ports verhindert, da dies die Systemleistung beeinträchtigen könnte. Dies könnte durch die Angabe einer Sicherheitsrichtlinie (security policy) vermieden werden, obwohl sie immer noch eine Ausnahme auslöst, wenn der Port bereits verwendet wird.

Das Trennen der Verbindung von einer Client- oder Serverseite erzeugt keine hässlichen Stack-Traces, aber es war nicht leicht zu unterscheiden, ob die Verbindung freiwillig geschlossen wurde oder ob sie verloren ging oder abrupt geschlossen wurde. Daher sind die Botschaften sehr vage, wenn dies geschieht.

Es gab ein Problem mit dem Lesen der Eingaben von der Konsole, was uns einige Tage lang Rätsel aufgab. Wenn ein von System.in gelesenes Scanner-Objekt geschlossen wird, wird eigentlich auch System.in geschlossen. Dies führte dazu, dass die Clientseite nicht von der Konsole in ihren Child-Threads scannen konnte, wenn ein anderer Scanner in der Main-Methode geschlossen wurde. Außerdem wollten wir das Console Singleton-Objekt nicht verwenden, da es aufgrund eines Bugs in Eclipse nicht funktioniert. Der Workaround, den wir gefunden haben, war nicht, den Scanner in der Hauptmethode zu schließen, sondern seine Referenz auf seine Unterklassen und Threads zu übergeben und ihn zu schließen, wenn die Socket-Verbindung schließlich geschlossen wird. Aufgrund dieses Fehlers in unserem Code haben wir tatsächlich viele Stunden Arbeit verschwendet, ohne zu verstehen, warum wir die Eingaben des Benutzers nicht lesen konnten.

4. Diskutieren Sie kurz, wie eine alternative Implementierung auf Grundlage des UDP-Protokolls aussehen würde und vergleichen Sie die beiden Möglichkeiten.

UDP, wie TCP, ist ein Protokoll, das zum Senden von Datenpaketen über das Internet verwendet wird.

Das UDP-Protokoll funktioniert ähnlich wie TCP, ist aber weniger zuverlässig. UDP-Pakete werden einfach an den Empfänger gesendet, ohne darauf zu warten, dass der Empfänger das Paket erhält. Auch UDP bietet keine Bestell- und Fehlerprüfung. Es gibt keine Garantie, dass Sie alle Pakete erhalten, und Sie könnten sie in der falschen Reihenfolge erhalten. Auch gibt es keine Möglichkeit, nach einem Paket zu fragen, wenn es fehlt. Die von TCP angebotene Sicherheit erzeugt jedoch eine Menge Overhead. Ohne sie können die Computer schneller kommunizieren und daher wird UDP verwendet, wenn die Geschwindigkeit wichtiger ist und keine Fehlerkorrektur erforderlich ist.

Die Implementierung eines UDP-Chats würde sich nicht sehr von der eines TCP-Chats unterscheiden, aber es wäre unzuverlässig. Das Risiko, ein Paket nicht

zu empfangen oder Pakete in der falschen Reihenfolge zu empfangen, ist zu hoch. Dies würde zu Kommunikationsproblemen zwischen den Clients führen. Für einen textbasierten Chat wird udp nicht empfohlen, aber es wird für Live-Streaming bevorzugt, wo Geschwindigkeit wichtiger ist. Der Hauptunterschied besteht jedoch darin, dass der Server nicht zu jedem Client eine Verbindung halten muss und es daher nicht für jede Verbindung einen eigenen Thread geben muss.