Übungsblatt 3
Sockets

Aufgabe 1: Sockets in Java

Folgende Aufgabe können Sie alleine oder in einem 2er-Team bearbeiten:

Es soll ein Client und ein Server entwickelt werden, die mittels TCP-Sockets Nachrichten austauschen. Der Client sendet den Text einer Usereingabe an den Server, dieser gibt sie auf der Konsole aus und sendet sie mit einem entsprechenden Präfix und/oder Postfix quittiert über die bestehende Verbindung zurück.

Binding-Port und Hostname werden über die Konsole bei Prozessstart mit übergeben, z. B.:

serverhost:VS$ `java Server 1234`
Dieser Aufruf startet den Server und bindet in an den angegebenen, lokalen Port (hier `1234` für alle Netzwerkschnittstellen des Rechners).

clienthost:VS$ `java Client rfhpc123 1234`
Dieser Aufruf startet die Client-Anwendung und baut eine Verbindung zum angegebenen Host und Port auf (hier zu Host `rfhpc123` am Port `1234`).

Beispiele für Konsolenausgaben:

Server an `rfhpc123`	Client an `rfhpc201`

serverhost:VS$ `java Server 1234`
`Warte auf Verbindungen...`
 	
`Neue Verbindung von rfhpc201:62107`
`Hallo Welt!`
	clienthost:VS$ `java Client rfhpc123 1234`
	
	`Bitte geben Sie eine Nachricht ein:`
	`Hallo Welt!`
	
	`ECHO:*Hallo Welt!*`

Die Verbindung zum Client wird nach dem ersten Nachrichtenaustausch geschlossen.

Tipp:
Die Eingabe von der Konsole funktioniert in Java wie folgt:

```
Scanner scanner = new Scanner(System.in);
String eingabe  = scanner.nextLine();
```

Aufgabe 2: Sockets in Java mit Threadpool

Entwickeln Sie eine Webserver-Anwendung in Java, die TCP-Verbindungen auf Port[1] `80` entgegennimmt. Clients können sich mit dem Server verbinden und den Inhalt von Dateien anfordern. Als Protokoll werden Ausschnitte des *Hypertext Transfer Protokol* (HTTP) verwendet.

Anfragen, die Ihr Webserver entgegennimmt, sollen nicht sequentiell abgearbeitet werden, sondern parallel mittels einzelner Threads innerhalb eines Threadpools (`ExecutorService`) abgearbeitet werden. Definieren Sie hierzu eine Klasse, die die Schnittstelle `Runnable` implementiert und das Socket als Parameter im Konstruktor erhält. Sobald sie im Threadpool abgearbeitet wird, antwortet das Objekt über das `Socket`-Objekt.

HTTP-Request eines Client:

```
GET /pfad/index.html HTTP/1.1
```

Der blau dargestellte, relative Pfad gibt den Dateinamen an, dessen Inhalt zurückgesendet werden soll.

HTTP-Response des Servers:

```
HTTP/1.1 200 OK\r\n
Content-Type: text/html\r\n
Content-Length: ___\r\n
\r\n
Dateiinhalt ab hier...
...
```

Die zugehörige Antwort folgt ebenfalls dem HTTP-Protokoll. Die obigen Angaben müssen im Header der Antwort enthalten sein. Der Header wird vom Body (dem Dateiinhalt) durch eine leere Zeile getrennt. Die Angabe `Content-Length` im Header gibt die Anzahl der Bytes (Zeichen) des Bodies an.

Zum Testen verwenden Sie als Client einen Browser. Eine Beispieldatei `index.html` für den Server finden Sie in GRIPS. Diese muss in den Workspace mit importiert werden.

[1] Alternativ kann der Port 8080 oder jeder andere freie Port gewählt werden.

Aufgabe 3: Planung eines Chat-Systems

Es soll die Architektur eines Chat-System geplant und mit Hilfe eines UML-Klassendiagramms dokumentiert werden, das die wichtigsten Klassen darstellt.

Folgende Rahmenbedingungen und Anforderungen sollen dabei berücksichtigt werden:

- Der Server bildet einen „Hub“, der eingehende Chat-Nachrichten an alle anderen, angemeldeten Clients weiterleitet
- Es gibt keine privaten Chats
- Es gibt keine Chats innerhalb von Teilgruppen
- Neu angemeldete Clients erhalten nur Nachrichten, die nach ihrer Anmeldung abgesetzt werden
- Der Server soll auf maximalen Durchsatz optimiert werden
- Clients können sich ggf. hinter einem NAT[2]-Router befinden, ggf. ist hier eine alternative Architekturentscheidung zu dokumentieren
- Geben Sie im Klassendiagramm durch einen Stereotyp <<synchronized>> an, welche Methoden durch einen Monitor abgesichert werden müssen
- Aktive Klassen sind als solche darzustellen

[2] NAT: Network Address Translation