Übung 05: NIO und Networking

Abgabetermin: 9. 5. 2022, 8:30

Übung 5: FileShare (24 Points)

In dieser Aufgabe soll eine Client-Server-Anwendung realisiert werden, mit dem Teilnehmer über einen Server Files downloaden (bzw. optional auch uploaden) können.

Die Clients melden sich mit einem eindeutigen Namen beim Server an und können folgende Operationen auswählen:

- `x` – Abmelden vom Server und Beenden der Applikation
- `l` – Auflisten aller Directories am Server
- `f` – Auflisten aller Files in einem Directory
- `d` – Download einer Datei aus einem Directory
- `u` – Upload einer Datei vom lokalen Verzeichnis dieses Clients (Operation ist optionale Zusatzaufgabe)

Folgend ist ein Dialog als Beispiel gezeigt:

```
------- Welcome to FileShare -------
Please, input your name: Fritz
------------------------------------
Hello Fritz!
Your commands:
  x - exit
  l - list directories
  f - files in directory
  u - upload file
  d - download file
------------------------------------

Input command: l
Directories:
  Fritz
  Hans

Input command: f
 Input directory: Hans
Files in Hans:
  hans1.txt
  hans2.txt

Input command: d
 Input directory: Hans
 Input file name: hans2.txt
File hans2.txt downloaded from Hans

Input command: x
--- Thanks for using FileShare -----
```

Implementieren Sie die Client-Server-Anwendung unter Verwendung von:

- `Files`, `Path`, `Paths` aus Package `java.nio` für den Dateizugriff
- Sockets (`ServerSocket`, `Socket`) aus `java.net` für die Client-Server-Kommunikation
- `InputStream` und `OutputStream` und `Reader` und `Writer` aus Package `java.io` für den Austausch von Daten und Messages

Hinweis zu Übung 6:

Diese Übung 5 wird mit Übung 6 kombiniert, indem in Übung 6 die Applikation mit Channels aus Package `java.nio` implementiert wird. Es wird dabei **keine** Musterlösung zu dieser Übung 5 ausgegeben. Es ist daher ratsam, diese Übung 5 sorgsam zu machen.

Anleitungen zum Programmdownload

Zur Übung gibt es ein Programmdownload, in dem Sie bereits folgende Programmteile finden:

- Die Klasse `Constants` enthält einige globale Konstanten, wie Servername oder Port.
- Die Klasse `Msg` mit der inneren Enumerationsklasse `Kind` dient zum Erstellen und Empfangen von Nachrichten. `Kind` ist eine Enumeration für die Arten der Messages, z.B. `LOGIN`, `DOWNLOAD` (fügen Sie bei Bedarf neue Enumerationswerte hinzu). Eine `Msg` besteht aus einem Kind und einer Liste von Parametern vom Typ `String`. Für das Senden und Empfangen von Nachrichten sind in `Msg` folgende statische Methoden bereitgestellt:
  - `static String sendMsg(PrintWriter writer, Kind kind, String...params)` – Schickt eine Nachrichtenzeile mit Kind und Parametern über den gegebenen `PrintWriter` (verwendet `Msg.line`).
  - `static String sendMsg(PrintWriter writer, String line)` – Schickt eine Nachrichtenzeile über den gegebenen `PrintWriter`.
  - `static Msg receiveMsg(BufferedReader reader)` – Liest eine Nachrichtenzeile über den gegebenen `BufferedReader`. Rückgabewert ist ein `Msg`-Objekt (verwendet `Msg.from`).
  - `public static String line(Kind kind, String...params)` – Erstellt eine Nachrichtenzeile aus Kind und Parameterwerten, abgeschlossen mit `END_OF_LINE`-Zeichen.
  - `public static Msg from(String line)` – Erstellt aus einer Nachrichtenzeile ein `Msg`-Objekt.
- Klasse `FileShareClient` enthält einen Rahmen für die Implementierung des Clients mit entsprechenden TODOs.
- Klasse `FileShareServer` enthält einen Rahmen für die Implementierung des Servers mit entsprechenden TODOs.
- Im Projekt sind bereits zwei Directories `"local"` und `"global"` angelegt. Im Directory `"global"` befinden sich die Directories mit den Files zum Download. Im Directory `"local"` hat jeder Client ein Directory (das Directory soll beim ersten Anmelden eines Clients erstellt werden). In dieses Directory sollen die Files beim Download eingefügt werden.

Herausforderungen und Lösungsansätze:

<u>Übertragen von Messages und Filedaten</u>

Eine wesentliche Herausforderung bei der Aufgabe ist, dass über die gleiche Socket-Verbindung einmal Messages und dann Filedaten übertragen werden müssen. Damit:

- sollen Messages in einer Zeile geschrieben und gelesen werden können (setzen Sie `sendMsg` und `receiveMsg` mit `PrintWriter` und `BufferedReader` ein)
- müssen Filedaten als Byte-Daten geschrieben und gelesen werden (setzen Sie Methoden `read` und `write` von `OutputStream` und `InputStream` ein).

Um das zu ermöglichen, gehen Sie bei einem Download am besten folgend vor:

- Schicken Sie mit der Message zum Download die Größe der Datei mit.
- Damit weiß der Client, wie viele Bytes er empfangen wird und wann der File vollständig übertragen ist.
- Mit den überladenen Methoden `read` von `InputStream` können Sie dabei exakt ermitteln, wie viele Bytes gelesen wurden und damit bestimmen, wann alle Daten übertragen wurden.

<u>Gleichzeitiger Zugriff von mehreren Clients</u>

Die Clients müssen am Server in eigenen Threads behandelt werden. Damit müssen die Zugriffe auf gemeinsame Daten unter wechselseitigem Ausschluss erfolgen.

<u>Beenden</u>

Achten Sie darauf, dass Client- und Server-Programme korrekt beendet werden.

Optionale Zusatzaufgabe: Upload (6 Zusatzpunkte)

Für die Implementierung eines Uploads vom Client gibt es 6 Zusatzpunkte.

Für ein Upload vom Client soll beim Login eines Clients im Directory `"global"` ein Unterverzeichnis mit dem Namen des Clients angelegt werden. Hier sollen die Files beim Upload eingefügt werden.