DEZSYS-04 „Interprozesskommunikation“ von Stefan Erceg

Inhalt

Aufgabenstellung

Lesen Sie das Oracle-Online-Tutorial "**All About Sockets**" und implementieren Sie das angeführte **KnockKnock Beispiel**. Hierfür empfiehlt es sich, die Quelldateien von der Webseite herunterzuladen und bei Ihnen lokal durchzuführen. Verfassen Sie ein Abgabedokument, indem Sie einen Testfall dokumentieren und jene Quellcodestellen hervorheben, die für die **Inter Process Communication** von Bedeutung sind.

Bewertungskriterien

- 2 Punkte: Quelldateien herunterladen und lokal ausführen. Vorgenommene Schritte dokumentieren
- 2 Punkte: Testfall dokumentieren (Aufruf Server, Aufruf Client, Ausgabe, Eingabe etc.)
- 2 Punkte: Dokumentieren der für die Kommunikation wichtigen Quellcodezeilen
- 2 Punkte: Erweiterung des KnockKnock-Servers zu einem Multiclient-Server

Quelldateien herunterladen und lokal ausführen

1.) Eclipse starten -> neues Projekt erstellen

2.) Die Quelldateien für das KnockKnock-Beispiel werden von den von Oracle zur Verfügung gestellten Webseiten heruntergeladen und in das neu erstellte Projekt importiert.

KnockKnockClient-Klasse:
http://docs.oracle.com/javase/tutorial/networking/sockets/examples/KnockKnockClient.java

KnockKnockProtocol-Klasse:
http://docs.oracle.com/javase/tutorial/networking/sockets/examples/KnockKnockProtocol.java

KnockKnockServer-Klasse:
http://docs.oracle.com/javase/tutorial/networking/sockets/examples/KnockKnockServer.java

3.) Führt man den KnockKnockServer nun aus, liefert die Konsole folgende Ausgabe:

```
Usage: java KnockKnockServer <port number>
```

Es muss somit noch eine Portnummer eingegeben werden (siehe Schritt 1 bei Testfall dokumentieren).

4.) Führt man den KnockKnockClient aus, liefert die Konsole folgende Ausgabe:

```
Usage: java EchoClient <host name> <port number>
```

Beim KnockKnockClient ist daher die Eingabe der IP-Adresse eines Servers und der Portnummer notwendig (siehe Schritt 1 bei Testfall dokumentieren).

Testfall dokumentieren

Damit der KnockKnockServer und der KnockKnockClient erfolgreich gestartet werden und eine Kommunikation durchführen, müssen noch die entsprechenden Argumente eingegeben werden. Dazu führt man folgenden Schritt in Eclipse aus:

1.) Run -> Run Configurations -> KnockKnockServer / KnockKnockClient -> Arguments -> Program arguments

- beim Server: <port number>
  - 50000
- beim Client: <host name> <port number>
  - 127.0.0.1 50000

-> Apply

Nun kann die Kommunikation gestartet werden. Wichtig hierbei ist, dass der Server vor dem Client gestartet wird, da der Client sich ansonsten mit keinem Server verbinden kann und eine Fehlermeldung (`Couldn't get I/O for the connection to localhost`) wirft:

2.) Server starten
3.) Client starten: Konsole öffnet sich und gibt „`Server: Knock! Knock!`“ aus.
4.) Wenn der Client die Frage „Who's there?“ stellt, gibt der Server „`Server: Turnip`“ aus.
5.) Nachdem der Client nach „Turnip who?“ fragt, werden die Zeilen „`Client: Turnip who?`“ und „`Server: Turnip the heat, it's cold in here! Want another? (y/n)`“ zum Dialog hinzugefügt.
6.) Gibt man „y“ (yes) ein, startet der Server die Kommunikation mit „`Server: Knock! Knock!`“ erneut, jedoch antwortet der Server bei der Frage „Who's there?“ vom Client mit einem anderen Namen (5 stehen zur Verfügung, nach dem 5. fängt der Server wieder beim 1. Namen an).

    Bei der Eingabe von „n“ antwortet der Server mit „`Bye.`“ und die Verbindung wird abgebrochen.

Gibt der Client eine der Fragen aus Schritt 4 - 6 falsch ein, antwortet der Server mit „`Server: You're supposed to say "..."!`“ und startet die Kommunikation erneut.

Wichtige Quellcodezeilen für die Kommunikation

KnockKnockServer

```
/* neuer Server-Socket wird initialisert und mit einer Portnummer versehen */
ServerSocket serverSocket = new ServerSocket(portNumber);

/* ein Client-Socket verbindet sich mit dem Server-Socket */
Socket clientSocket = serverSocket.accept();

/* ein Output-Stream wird ausgegeben */
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

/* der Input des Clients wird in einem BufferedReader zwischengelagert */
BufferedReader in =
      new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

/* Kommunikation mit dem Client wird gestartet */
KnockKnockProtocol kkp = new KnockKnockProtocol();
```

KnockKnockClient

```
/* neuer Socket wird initialisiert und mit einer IP-Adresse und einer Portnummer
   versehen */
Socket kkSocket = new Socket(hostName, portNumber);

/* ein Output-Stream des Sockets wird erzeugt */
PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);

/* der Input des Clients wird in einem BufferedReader zwischengelagert */
BufferedReader in =
      new BufferedReader(new InputStreamReader(kkSocket.getInputStream()));
```

Erweiterung des KnockKnock-Servers zu einem Multiclient-Server

Für diese Erweiterung wurden die Quelldateien KKMultiServer und KKMultiServerThread, welche ebenfalls von Oracle zur Verfügung gestellt wurden, heruntergeladen und in das Eclipse-Projekt importiert.

KKMultiServer-Klasse:
http://docs.oracle.com/javase/tutorial/networking/sockets/examples/KKMultiServer.java

KKMultiServerThread-Klasse:
http://docs.oracle.com/javase/tutorial/networking/sockets/examples/KKMultiServerThread.java

Durchführung

1.) Führt man den KKMultiServer nun aus, liefert die Konsole folgende Ausgabe:

```
Usage: java KKMultiServer <port number>
```

2.) Beim KKMultiServer wird bei den Run Configurations daher dasselbe Argument wie beim KnockKnockServer, nämlich die Portnummer, eingetragen.

Run -> Run Configurations -> KKMultiServer -> Arguments -> Program arguments

- beim KKMultiServer: <port number>
  - 50000

-> Apply

3.) Nun ist es möglich, dass sich mehrere Clients zu dem KKMultiServer verbinden. Die Funktionsweise des KKMultiServers bleibt im Gegensatz zum KnockKnockServer gleich, nur das hier Threads für die Nebenläufigkeit verwendet werden. Solange der KKMultiServer läuft, d.h. solange das Programm nicht beendet wird, können beliebig viele KnockKnockClients eine Kommunikation mit ihm starten. Damit dies so erfolgt, wird eine `while(true)` im Quellcode eingebaut.

Runnable JAR-File erzeugen

Damit ein Runnable JAR-File erstellt wird, wurden die Klasse „ServerThread“, „ClientThread“ und „Simulation“ geschrieben.

- „ServerThread“
  ➔ implementiert Runnable und startet in der run-Methode einen KKMultiServer. Diesem wird eine Portnummer voreingestellt.
- „ClientThread“
  ➔ implementiert ebenfalls Runnable und startet in der run-Methode einen KnockKnockClient. Diesem werden eine IP-Adresse und eine Portnummer voreingestellt.
- „Simulation“
  ➔ In „Simulation“ werden 2 Threads erstellt, einer führt den ServerThread aus, der andere den ClientThread.

Nun kann „Simulation“ als Main-Class vom JAR-File verwendet und das JAR-File über die CMD gestartet werden.