Client-Server – TCP/IP - Kodierung

(1.) (a.) Testen Sie das erste Beispiel aus dem Ordner 01: Starten Sie Client und Server auf dem gleichen PC, aber den Server zuerst!

(b.) In welchem Bereich müssen die Ports liegen? Testen Sie mit einem Port, von dem Sie meinen, dass dieser nicht funktioniert und verifizieren Sie die „Nichtfunktionalität“!

(c.) Welche Daten werden übertragen? Zeichenketten? Große Zahlen? Negative Zahlen?

(d.) Der Server läuft nicht, der Client wird gestartet. Erklären Sie die Ausgabe auf der Konsole und ändern Sie diese ab, dass der Text „Fehler bei…..“ aufgegeben wird!

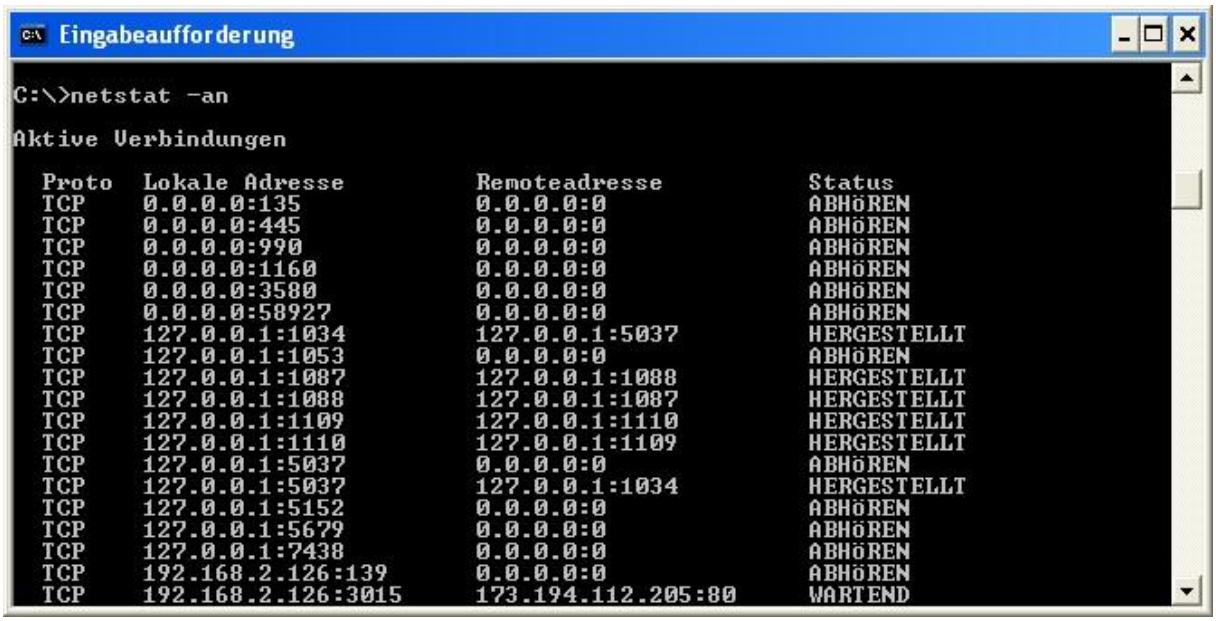
(e.) Der Server läuft bereits und wird ein zweites Mal gestartet! Erklären Sie die Ausgabe auf der Konsole und ändern Sie gegebenenfalls die Fehlermeldung in eine verständliche Form!

(f.) Ändern Sie in der Klasse Server die Programmzeile outputstream.write(256); Begründen Sie, warum jetzt der Client einen falschen Wert auf der Konsole ausgibt. Ändern Sie anschließend den Programmcode, so dass eine fehlerfreie Übertragung erfolgt. Hinweis für die Klasse Server: Verwenden Sie die JDK-Klasse DataOutputStream.

(2.) Ein Client schickt an einen Server 2 Zahlen, welche vorher mittels Tastatur eingegeben wurden. Der Server addiert beide Zahlen und schickt das Ergebnis an den Client zurück. Danach beendet er die Verbindung und wartet auf die nächste Verbindungsanfrage. Der Client gibt das vom Server erhaltene Ergebnis auf der Konsole aus und wird danach beendet.

Erstellen Sie den Quellcode für den Client und den Server.

(3.) Auf der Konsole den Befehl „netstat –an“ eingeben. Es werden die aktiven offenen TCP-Ports angezeigt, die im „Listen-Modus“ abhören, ob sich jemand mit Ihnen verbinden möchte:



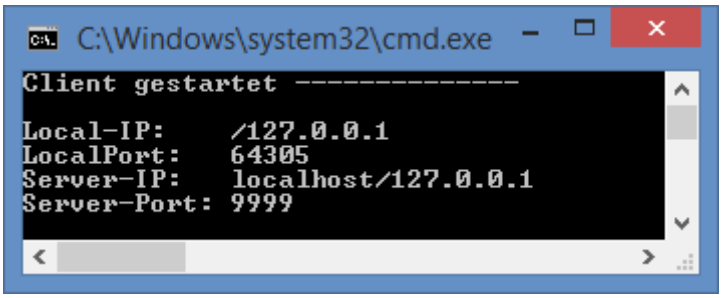
Gegeben ist nun folgender Quellkode:

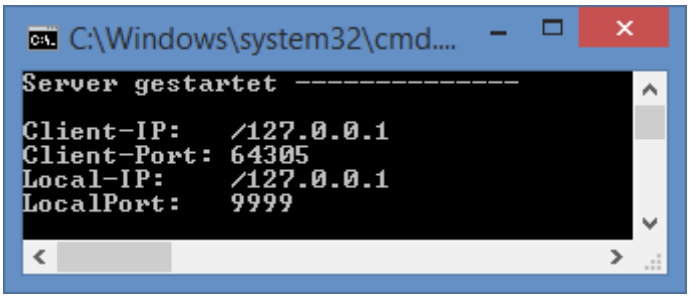
```
import java.io.*;
import java.net.*;
public class PortScanner {
  public static void main(String[] args) {
    try
    {
      Socket socket = new Socket ("localhost",135 );
      System.out.println ("Es läuft ein Server auf Port 135");
      socket.
      socket.close();
    }
    catch (IOException e){
      System.out.println(e.getMessage());
    }
  }
}
```

(a.) Analysieren Sie den Quellcode und erläutern Sie kurz die Funktion!

(b.)Erstellen Sie ein Java-Programm, welches nach Eingabe von zwei Portnummern alle dazwischen liegenden Portnummern des eigenen Rechners (URL = localhost) scannen kann. Das Programm soll, falls ein Server auf einer Portnummer erreichbar ist, eine entsprechende Information ausgeben!

(4.) Erstellen Sie zwei Programme(Server und Client), welches folgende Ausgabe leisten:





Verwenden Sie die Methoden:
getLocalAddress(), getLocalPort(), getInetAddress(), getPort()
der Socket-Klasse!

(5.) (a.) Ergänzen Sie die vorgegebenen Quellkodes von einem Echoserver und seinem Klienten!

(b.) Ändern sie den Server so ab, so dass dieser den Text in Großbuchstaben zurückgibt!

(c.) Ändern sie den Server so ab, so dass dieser nach dem Beenden des Clienten weiter läuft aber auch zu beenden ist! Stichwort: Einmalige und Dauerverbindung!

(d.) Erweitern Sie das Programm aus (a.) wie folgt: Der Server gibt auf seiner Konsole die folgenden Meldungen aus, welche vom aktuellen Status abhängig sind:

- *"Ich warte ..." (Server läuft, ist aber mit keinem Client verbunden)*
- *"Bin mit Client verbunden." (Server ist gerade mit einem Client verbunden)*
- *"Der Client hat die Verbindung beendet." (Client hatte gerade den String ".." gesendet)*
- *Sollte der Client die Verbindung zum Server "unsauber" (also nicht durch Eingabe des Strings "ende") abbrechen, so muss der Server auf seiner Konsole ausgeben: "Verbindung zum Client verloren ..." In diesem Fall muss der Server für einen neuen Client verfügbar sein.*

(e.) Erweitern Sie den Server so, dass dieser den Text zurück sendet, allerdings ersetzt er alle Vokale durch einen per Zufallsgenerator ausgewählten anderen Vokal!
Kodieren Sie dazu folgende Funktion:

```
public static String cryptStr(String sourceStr, String cryptStr){
}
```

sourceStr ist die Zeichenkette, wo die Vokale ersetzt werden, cryptStr die Zeichenkette aus der wahllos Buchstaben zum Ersetzen gewählt werden(hier wird immer mit „aeiou“ derAufruf gestaltet) und es wird die neue Zeichenkette zurückgegeben!

(6.) (a.) Ändern Sie die Quelltextvorlage aus (5.), so dass ein Multi-Echo-Server möglich ist, der den jeweiligen Klienten seine Nachricht zurücksendet!

(b.) Erweitern Sie das Programm, so dass der Server die Anzahl der Klienten ausgibt!

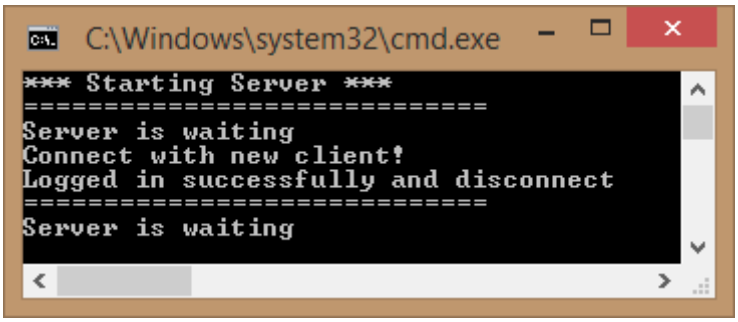
(7.) Kodieren Sie den Multi-Klienten-Chat-Server und einen passenden Klienten!

(8.) Kodieren Sie Augabe 6 und 7 mit Hilfe der Landesabiturklassen Socket und ServerSocket „neu“

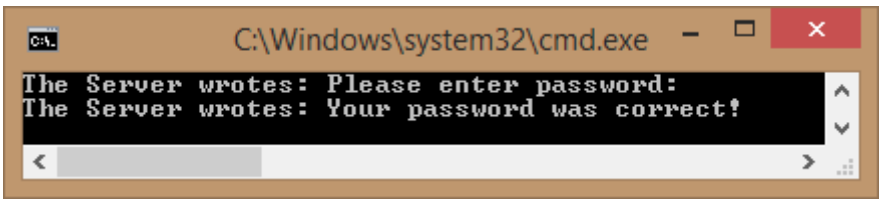
(9.) Mögliche Abituraufgabe: Erstellen Sie einen Server, welcher die nachfolgend beschriebenen Anforderungen erfüllt:

Verbindet sich der Client mit dem Server, so schickt der Server folgende Aufforderung an den Client: "Please enter Password:" Der Client sendet anschließend das Passwort als reinen Text ohne Verschlüsselung , welches vorher mittels Tastatur einzugeben ist. Ist das Passwort korrekt, so sendet der Server eine kurze Nachricht (z. B. " Your password was correct!"), welche der Client auf der Konsole ausgibt. Danach beendet der Server die Verbindung und gibt eine entsprechende Meldung auf der Konsole aus. Anschließend wartet der Server auf die nächste Verbindungsanfrage eines Clients. Der Server soll den jeweils aktuellen Status auf der Konsole ausgeben. Die Konsolenausgaben für den oben beschriebenen Ablauf sollen etwa so aussehen:

Server:

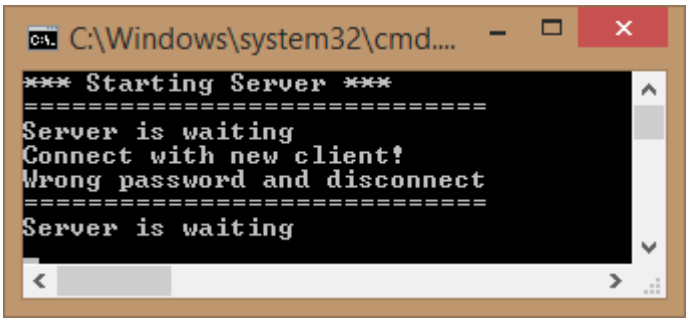


Client:

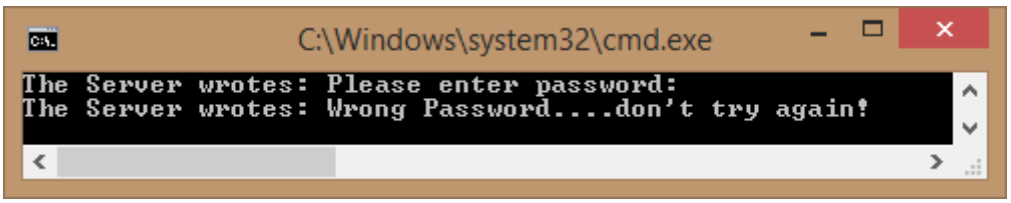


Ist das Passwort falsch, so sendet der Server eine entsprechende Meldung und beendet ebenfalls die Verbindung. Für diesen Vorgang sollen die Konsolenausgaben etwa so aussehen:

Server:

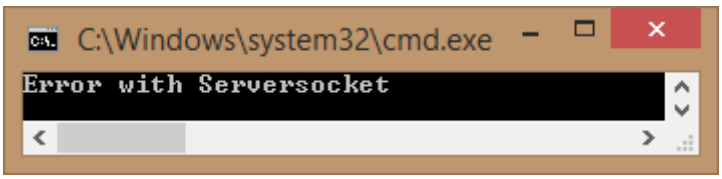


Client:



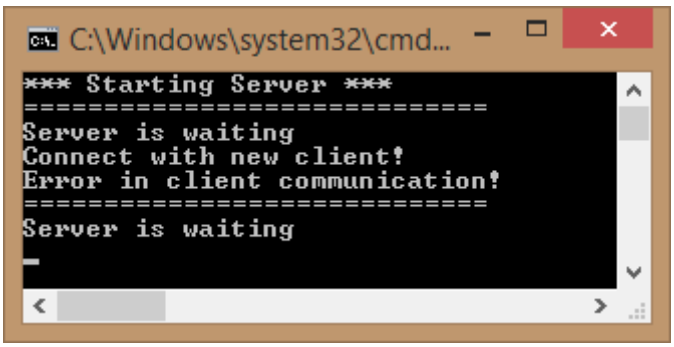
Verhalten des Servers bei Fehlern:

Sollte beim Start des Servers der Serversocket nicht erzeugt werden können (weil z.B. der Port bereits von einem anderen Server genutzt wird), so wird die folgende Meldung ausgegeben und das Programm wird beendet:



Sollte der Client die Verbindung abrupt trennen (wenn z.B. der Client vor Eingabe des Passwortes durch Schließung des Konsolenfensters beendet wird), so darf der Server nicht abstürzen. Er gibt eine eindeutige Fehlermeldung aus und muss weiterhin auf Verbindungsanfragen von Clients warten können.

Die Konsolenausgabe für den beschriebenen Ablauf soll etwa so aussehen:



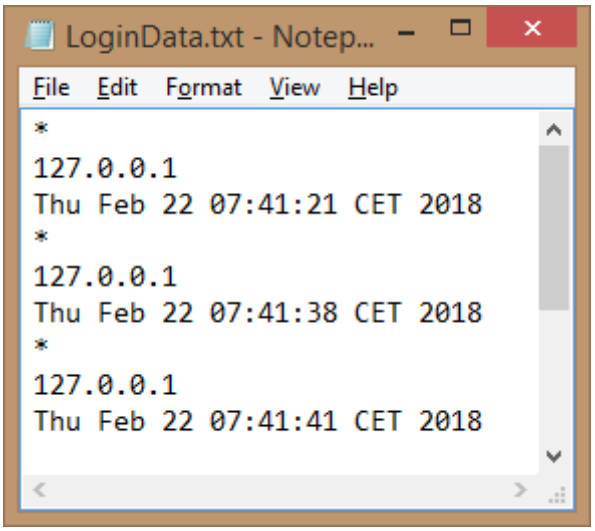
Wichtige Hinweise:

- In der Abiturklausur werden die Klassendiagramme gegeben!

- Denken Sie an die „Fußangeln“ bei der Kodierung insbesondere das Zufügen von „\n“ bei der Methode write um ganze Zeilen zu schreiben!

Verwenden Sie zum Testen die beiden Programme unter Templates!

(10.) Ein Server soll die Verbindungsdaten von Clients speichern. Wenn sich ein Client mit dem Server verbindet, so speichert der Server sofort die IP-Adresse in der Dotted-Quad-Notation (Dotted-Decimal-Notation) sowie Datum und Uhrzeit des Verbindungsaufbaus in einer Textdatei mit diesem Format:



Anschließend findet der (hypotetische)Datenaustausch zwischen Client und Server statt. Es würde auch genügen, den Server mit Thread.sleep(10) kurz anzuhalten und danach die Verbindung zum Client sauber zu trennen

(11.) Abituraufgabe

(c.) Vergleichen Sie die Musterlösung von (b.) und geben Sie allgemeine Kriterien an, wie die(und ihre) Kodierung optimiert werden kann!

Lösung:

```
import java.net.*;
import java.io.*;

public class Server
{
  public static void main(String[] args)
  {
     try
    {
        System.out.println("Ich warte auf einen Client.");
        ServerSocket servsock = new ServerSocket(1234);
        Socket socket = servsock.accept();
        System.out.println("Die Verbindung zu einem Client wurde hergestellt.");

     OutputStream out = socket.getOutputStream();

     FileInputStream fis = new FileInputStream("video.wmv");

     byte[] buffer = new byte[1024];

     while (fis.available()>0)
     {
       out.write(buffer, 0, fis.read(buffer));
     }

     fis.close();
     out.close();
       socket.close();
        servsock.close();
        System.out.println("Die Verbindung zum Client wurde beendet.");
     }
     catch (Exception e)
     {
        e.printStackTrace();
     }
    }
}
```

(12.) Abituraufgabe:

Client-Server – TCP/IP - Kodierung - Lösungen

(1.)
(b.) 0 bis 65535; z. B. 80 für Internet belegt
(c.) Zahlen von 0..255; d.h. Datetyp byte
(f.)
Server:

```
dos = new DataOutputStream(outputstream);
dos.writeInt(257);
dos.close();
```

Client:

```
dis = new DataInputStream(inputstream);
zahl = dis.readInt();
System.out.println(zahl);
dis.close();
```