

---

## Aufgaben zu Kapitel 3

---

### Aufgabe 1

- a.) Betrachten Sie die .NET-Klasse `TcpClient` aus dem Namensraum `System.Net.Sockets`. Sie setzt auf der klassischen Socket-Schnittstelle auf. Welche Socket-Aufrufe werden getätigt, wenn der Konstruktor von `TcpClient` mit `TcpClient("www.swt.hs-mannheim.de", "80");` aufgerufen wird?



**Lösung:** Der Aufruf bewirkt zunächst die Auflösung des FQDN `www.swt.hs-mannheim.de` mit der Methode `Dns.GetHostByName()` in eine IP-Adresse. Danach kann eine Instanz der Klasse `System.Net.Sockets.Socket` als Stream-Socket erzeugt werden. Schließlich wird die Methode `Socket.Connect()` aufgerufen, um eine TCP-Verbindung zum Server aufzubauen.

- b.) Geben Sie an, was an der Socket-Schnittstelle passiert, wenn der Konstruktor von `System.Net.Sockets.UdpClient` mit `UdpClient()` aufgerufen wird.



**Lösung:** Es wird eine Instanz der Klasse `System.Net.Sockets.Socket` als Datagram-Socket erzeugt.

- c.) Wie ist der Zusammenhang mit dem Socket-API, wenn die Methode `Start()` der Klasse `System.Net.Sockets.TcpListener` aufgerufen wird.



**Lösung:** Es wird die Methode `Socket.Bind()` aufgerufen, um den lokalen Kommunikationsendpunkt (insbesondere den well-known Port) mit dem Socket zu assoziieren. Anschließend wird die Methode `Socket.Listen()` aufgerufen, um den Socket in den passiven („listening“) Zustand zu überführen.

### Aufgabe 2

Ein stream-basierter Client benutzt die Methode `GetResponse()`, um eine Zeile Text (mit `\r\n` terminiert) von einem Server zu empfangen und auf den Bildschirm zu schreiben.



- a.) Erläutern Sie, warum die folgende Version der Methode `GetResponse()` in der Klasse `TcpStringClient` nicht immer funktioniert.




```
using System;
using System.IO;
using System.Net.Sockets;

public class TcpStringClient
{
    private TcpClient tcpClient;
    private NetworkStream networkStream;
```

```
private StreamReader reader;

public TcpStringClient()
{
    try
    {
        tcpClient = new TcpClient("127.0.0.1", 2345);
        networkStream = tcpClient.GetStream();
        reader = new StreamReader(networkStream);
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

private void GetResponse()
{
    try
    {
        string response;
        int count;
        byte[] bytes = new byte[8192];
        count = networkStream.Read(bytes, 0, 8192); 
        response = System.Text.Encoding.ASCII.GetString(
            bytes, 0, count);
        Console.WriteLine(response);
        networkStream.Close();
        tcpClient.Close();
    }
    catch(Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

public static void Main()
{
    TcpStringClient tcpStringClient = new TcpStringClient();
    tcpStringClient.GetResponse();
    return;
}
```

**Lösung:** Da ein Stream-Socket benutzt wird, kann die Zeile Text in einem oder mehreren TCP-Segmenten ankommen. Wenn `networkStream.Read()` aufgerufen wird, ist möglicherweise nur ein Teil des Antwortstrings erhältlich. Es gibt keine Garantie, dass die Terminierung `\r\n` eingetroffen ist.

Es ist zu beachten, dass diese Version manchmal funktionieren wird. Das ist dann der Fall, wenn alle Daten durch den Server abgeschickt wurden und vom Client zum Zeitpunkt des Aufrufs von `networkStream.Read()` empfangen wurden.

- b.) Überlegen Sie, unter welcher Voraussetzung die folgende Version von `GetResponse()` funktionieren wird. Was passiert, wenn diese Voraussetzung nicht erfüllt ist?

```
private void GetResponse()  
{  
    try  
    {  
        StreamReader reader = new StreamReader(networkStream);  
        string response;  
        response = reader.ReadToEnd();  
        Console.WriteLine(response);  
        networkStream.Close();  
        tcpClient.Close();  
    }  
    catch(Exception e)  
    {  
        Console.WriteLine(e.Message);  
    }  
}
```

**Lösung:** Damit diese Version funktioniert, muss der Server nach der Übertragung die Verbindung schließen. Dann kehrt der blockierende Aufruf von `reader.ReadToEnd()` zurück. Ist dies nicht der Fall, wird der Client nach Empfang des Strings weiterhin im Aufruf von `reader.ReadToEnd()` blockieren, da auf das Ende des Streams gewartet wird.

- c.) Schreiben Sie die Methode `GetResponse()` von Teil b.) so um, dass die Textzeile sicher empfangen wird.


**Lösung:**

Variante 1:

```
private void GetResponse()  
{  
    try  
    {  
        string response;  
        int count;  
        byte[] bytes = new byte[8192];  
        do  
        {  
            count = networkStream.Read(bytes, 0, 8192);  
            response = System.Text.Encoding.ASCII.GetString(  
                bytes, 0, count);  
            Console.Write(response);  
        } while (!response.EndsWith("\r\n"));  
        networkStream.Close();  
        tcpClient.Close();  
    }  
    catch(Exception e)  
    {  
        Console.WriteLine(e.Message);  
    }  
}
```

Hier wird `networkStream.Read()` solange aufgerufen bis das terminierende `\r\n` eingetroffen ist

Variante 2 (eleganter):

```
private void GetResponse()  
{  
    try  
    {  
           
        StreamReader reader = new StreamReader(networkStream);  
        string response;  
        response = reader.ReadLine();  
        Console.WriteLine(response);  
        networkStream.Close();  
        tcpClient.Close();  
    }  
    catch(Exception e)  
    {  
        Console.WriteLine(e.Message);  
    }  
}
```

Hier stellt der Aufruf der Methode `reader.ReadLine()`; sicher, dass solange Daten empfangen werden, bis eine komplette Zeile Text empfangen wurde. Dazu muss eine Instanz der Klasse `StreamReader` über den Netzwerkstrom gelegt werden.

### Aufgabe 3

- a.) Beschreiben Sie Vor- und Nachteile der Programmiermodelle Boss-Worker, Peer Thread und Thread Pool für das Multithreading in Server-Programmen.

**Lösung:** Siehe Skript.

- b.) Ein Server wird wie folgt betrieben.

Eine einzige Instanz `tcpListener` der Klasse `System.Net.Sockets.TcpListener` wird durch den Master-Thread erzeugt.

- Zehn Slave Threads werden erzeugt.
- Jeder Slave Thread führt einen Aufruf von `tcpListener.AcceptSocket()` durch, um eine Verbindung anzunehmen.
- Sobald ein Thread eine Verbindung annimmt, bedient er die Verbindung auf dem neuen Socket `activeSock`, der von `tcpListener.AcceptSocket()` zurückgegeben wird und schließt dann die Verbindung.
- Der Thread kehrt zu Schritt (iii) zurück, um auf eine neue Verbindung zu warten.

Wie nennt man den in diesem Fall gewählten Ansatz für das Multithreading?

**Lösung:** Es handelt sich um das Peer Thread - Modell. In diesem Fall werden 10 Threads statisch allokiert für die parallele Verarbeitung von bis zu 10 Verbindungen.



Ist es in Schritt (iii) möglich, dass zwei oder mehrere Slave Threads dieselbe Verbindungsanforderung von einem Client annehmen? Begründen Sie Ihre Antwort.

**Lösung:** Nein. Die Methode `AcceptSocket()` ist thread-safe. Letztendlich sorgt das Betriebssystem für gegenseitigen Ausschluss (mutual exclusion) zwischen den Threads, um sicherzustellen, dass nur ein Thread vom Aufruf `AcceptSocket()` zurückkehrt, nachdem eine Verbindung mit einem Client zustande gekommen ist.