Kommunikation

Kommunikation mit TCP / IP

IP-basierte Netzwerke versenden die Nachricht in **Paketen**. Alle IP-Pakete enthalten Zieladresse (IP-Adresse), Absenderadresse (IP-Adresse) und den Dienst, an den sie weitergereicht werden müssen (z. B. TCP).

Wichtigster Dienst über IP ist **TCP** (transmission control protocol). Er realisiert **bidirektionale Datenströme**. Um zwischen zwei Rechnern mehrere Datenströme unabhängig realisieren zu können, werden für Sender und Empfänger zusätzliche Kennungen, die „**Ports**“ angegeben. Der Datenstrom wird durch Zieladresse, Absenderadresse, Zielport und Absenderport eindeutig identifiziert.

Portnummern bis 255 sind fest vorgegeben, die Nummern von 256 bis 65535 können frei vergeben werden.

Server „lauschen“ auf einem von Server definierten Port auf Anfragen. Bei jeder neuen Anfrage wird ein neuer Serverprozess erzeugt, der die Anfrage behandelt.

TCP / IP - Unterstützung in Java

Java realisiert TCP/IP-Verbindungen über so genannte Sockets, die jeweils einen Eingabedatenstrom der Klasse InputStream und einen Ausgabestrom der Klasse OutputStream verwalten. Jeder Socket wird über ein Objekt der Klasse Socket realisiert.

Ein Objekt der Klasse ServerSocket erlaubt das Lauschen an einem angegebenen Port; für jede neue Verbindung wird ein Objekt der Klasse Socket erzeugt.

Die in Java vorhandenen Threads (später) ermöglichen ein einfaches Anlegen neuer Prozesse zur Weiterverarbeitung einer erzeugten Verbindung.

Die Klasse InetAddress implementiert eine Hilfsklasse zur Verwaltung von Internetadressen.

Die Klasse ServerSocket

ServerSocket hat folgende wichtige Methoden:

ServerSocket (int port)	Erzeugt einen ServerSocket, der am angegebenen Port auf Verbindungswünsche wartet
Socket accept ()	wartet blockierend auf einen Verbindungswunsch. Das zurückgegebenen Objekt ist der Socket für die neue Verbindung
void close ()	schließt den ServerSocket

Die Klasse Socket

Socket hat folgende wichtige Methoden:

Socket (String host, int port)	Erzeugt einen Socket, der mit „host“ auf „port“ Verbindung aufnehmen will. „host“ kann einen Rechnernamen oder eine IP-Adresse enthalten. Der lokale Port wird automatisch vergeben.
InputStream getInputStream ()	Liefert den Eingabestrom
OutputStream getOutputStream ()	Liefert den Ausgabestrom
void close ()	schließt die Verbindung
InetAddress getInetAddress ()	Liefert die Adresse des Kommunikationspartners
InetAddress getLocalAddress ()	Liefert die eigene Adresse
int getPort ()	Liefert den Port des Kommunikationspartners
int getLocalPort ()	Liefert den eigenen Port

Server, der jeweils nur einen Klienten bedient

Das Serverprogramm wartet in der Regel einfach auf Anrufe.

```
void ServerKern ()
```

```
    {
        try
        {
            ServerSocket server = new ServerSocket (<zu überwachender Port>);
            while (true)
            {
                Socket anrufSocket = server. accept ();
                System.out.println ("Anruf von " + anrufSocket. toString ());
                Bearbeiten (anrufSocket);
            }
        }
        catch (Exception e)
        {
            e. printStackTrace ();
        };
    }
```

Die Bearbetung des Auftrags erfolgt in einer eigenen Methode.

```
void Bearbeiten (Socket socket)
{
    InputStream in;
    OutputStream out;

    out = socket. getOutputStream ();
    out. flush ();
    in = socket. getInputStream ();

    try {
        <was zu machen ist>
    }
    catch (IOException e) {
        e. printStackTrace ();
    }
    try {
        in. close ();
        out. close ();
        socket. close ();
    }
    catch (IOException e) {
        e. printStackTrace ();
    };
}
```

Typischer Client

Das Clienthauptprogramm ruft den Server an.

```
    void Anruf ()
    {
        Socket s;
        InputStream in;
        OutputStream out;
        try {
            s = new Socket (<Servername>, <Serverport>);
            out = new ObjectOutputStream (s. getOutputStream ());
            out. flush ();
            in = new ObjectInputStream (s. getInputStream ());
            <Jetzt geht die Arbeit los.>
        }
        catch (UnknownHostException e) {
            System. out. println ("Server nicht gefunden!");
            System.exit(-1);
        }
        catch (IOException e) {
            e. printStackTrace ();
            System. out. println ("Server antwortet nicht!");
            System.exit(-1);
```

```
        };
    }
```

Aufgaben

1. Kommunikation mit Telnet

Bauen Sie mithilfe des Programms Telnet eine Verbindung zu einem Webserver auf (Welcher Port muss verwendet werden?) und fordern Sie eine Seite an.

2. Webseite „absaugen“

Entwickeln und implementieren Sie eine Klasse deren Methode(n) eine Verbindung zu einem Webserver aufbauen und eine Seite anfordern. Die Seite soll in das Ausgabefenster ausgegeben werden.

Hinweis: Arbeiten Sie mit zeilenorientierten Zeichenströmen.

3. Chatserver Vorbereitung

a) Entwickeln Sie einen einfachen Server, der auf einen Verbindungswunsch wartet und dann alle an ihn gesendeten Zeichen wieder zurück an den Client sendet. Der Server soll beendet werden, wenn der Client die Verbindung schließt. Testen Sie den Server mit dem Programm Telnet.

b) Entwickeln Sie ein kleines Client-Programm für den Server aus a).

Hinweis: Sie können direkt mit Byteströmen arbeiten.

4. Chatserver Anmeldung

Entwerfen Sie ein möglichst einfaches Protokoll, mit dem sich der Client beim Server an- und wieder abmelden kann. Nach der Abmeldung soll der Server auf den nächsten Anruf warten. Das Protokoll soll eine spezielle Anweisung enthalten, um den Server zu beenden.

Testen Sie zunächst mit Telnet und ergänzen Sie dann Ihren Client entsprechend.

Hinweis: Keine Authentifizierung