**In diesem Kapitel:**
- *Was ist ein Server Socket?*
- *Die ServerSocket Klasse*
- *Aufbau eines Servers*
- *einige Server Beispiele*

# 8

# *Sockets für Server*

## *8.1. Was ist ein Server Socket?*

In diesem Kapitel beschreiben wir nun die Server Seite der Sockets. Ein wesentliches Problem, welches Sie im Detail verstehen müssen, ist die Frage, wie der Server mehrere Anfragen gleichzeitig behandeln kann:
was passiert, wenn der Server Port besetzt ist und weitere Anfragen ankommen? Werden diese noch bearbeitet, oder kann en Socket Server nur je einen Client haben? Wie sieht das Ganze auf der Stufe IP Adresse und Port Adresse aus?

## *8.2. Lernziele*

Sie können einfache Socket Programme realisieren und mit Ihren Kollegen mit Hilfe einer Socket Client / Server Verbindung kommunizieren.

## *8.3. Aufbau*

Der Aufbau ist wie in den vorangehenden Kapiteln:
- was ist ein Server Socket:
  kurze Erklärung des Begriffes (Wiederholung?)
- die Socket Server Klasse
  Konstruktor
  Methoden
  Datenelemente
- einführende Beispiele (im obigen Text eingestreut)
- einige brauchbare, komplexere Beispiele

### *8.4. Was ist ein Server Socket?*

Wir haben gesehen, dass ein Client oft Probleme hat einen passenden Server zu finden. Jetzt wollen wir diese Server oder auch nur Client Tester selber erstellen.

Ein Server hat zwangsweise viel mehr Aufgaben als ein Client, der einfach Angaben und Informationen abfragt. Der Server muss bereit sein, Auskunft zu erteilen, und dies in der Regel an mehrere Clients gleichzeitig und an nicht vorbestimmte Clients.

Was kann nun ein Socket Server?
1) Verbindung zu einem entfernten Rechner aufbauen
2) Daten senden
3) Daten empfangen
4) eine Verbindung beenden
5) einen Port binden
6) auf ankommende Daten hören
7) Verbindungsaufforderungen von andern Rechnern verarbeiten

Die Java Socket Server Klasse Sie besitzt Methoden, die man braucht , um auf Client Anfragen reagieren zu können. Jeder Socket Server hört auf einen bestimmten Port und muss reagieren, wenn ein Client eine Verbindung aufbauen möchte.

### *8.5. Die Server Socket Klasse*

Die Server Socket Klasse enthält alles, was man benötigt, um Socket Server in Java schreiben zu können.

Wie sieht nun eine Socket Verbindung aus Server Seite aus?
1. der Server kreiert einen neuen `ServerSocket` mit Hilfe eines Konstruktors, für einen bestimmten Port
2. der `ServerSocket` hört auf eingehende Anfragen auf dem definierten Port. Dazu verwendet der `ServerSocket` die Methode `accept`(). Die Methode `accept`() blockiert den Zugang bis eine Clientanfrage vorliegt. Sobald eine Anfrage vorliegt, liefert `accept`() ein Socket Objekt nun findet die Kommunikation zwischen zwei normalen Sockets statt
3. dazu öffnet der Server Eingabe- und Ausgabeströme mit Hilfe der `getInputStream`() und / oder `getOutputStream`() Methoden. Damit wird die Kommunikation mit dem / den Client Sockets ermöglicht. Nachdem die Verbindung aufgebaut ist, senden und empfangen der lokale und der remote Host Daten aus einem Input beziehungsweise Output Stream.
   Die Verbindung ist *voll-duplex* , Host und Client können Daten senden und empfangen. Die Bedeutung der Daten selber ergibt sich aus der Anwendung. Eine FTP oder ein HTTP sehen unterschiedlich aus. Für die Kommunikation verwenden beide Seiten, Client und Server normale Sockets.
4. nachdem die Datenübermittlung abgeschlossen ist, schliesst der Host oder der Client die Verbindung. Einige Protokolle, wie HTTP verlangen, dass nach jeder Datenübermittlung die Verbindung abgebaut wird (HTTP : GET Befehl holt sich Daten; der HTTP Server merkt sich aber nicht wer was angefordert hat.).
5. der Server wartet auf weitere Verbindungsanfragen

Da in der Regel Server stark belastet werden, muss nach dem Verbindungsaufbau dafür gesorgt werden, dass der Server wieder frei wird und für weitere Verbindungsanfragen zur Verfügung steht.

In Unix geschah dies in der Regel dadurch, dass pro Verbindung ein neuer Prozess gestartet wurde. In Java wird der Overhead reduziert, indem lediglich Threads generiert werden, in der Regel jeweils ein Thread pro Client.

Auf Betriebssystem Ebene werden eingehende Verbindungsanforderungen in eine First In / First Out Warteschlange eingetragen, normalerweise bis zu 50 Anfragen pro Warteschlange. Sobald die maximale Anzahl Einträge erreicht wird, schliesst der Server seine Türe, es sind keine weiteren Verbindungen mehr möglich. Sie kennen dies von fast allen MP3 Sites, von denen Sie versucht haben Dateien herunter zu laden.

Die Warteschlange wird auf Stufe Betriebsystem verwaltet. Wir werden versuchen, heraus zu finden, was konkret auf dieser Ebene geschieht.

Die Systemparameter können auch aus Java heraus verändert werden, falls sie veränderlich sind.

Schauen wir uns mal die Klasse etwas genauer an.

## *8.6. Die ServerSocket Klasse*

### 8.6.1. Lernziele

- Sie kennen die Konstruktoren der ServerSocket Klasse und können diese anwenden.
- Sie kennen die Methoden der ServerSocket Klasse und diese in einfachen Programmen einsetzen
- Sie die Parameter der ServerSocket Konstruktoren und wissen, wie diese mit Hilfe der Methoden abgefragt werden können

### 8.6.2. ServerSocket Klasse Übersicht

Schauen wir jetzt die einzelnen Grössen an.

#### 8.6.2.1. java.net Class ServerSocket

```
java.lang.Object
   |
   +--java.net.ServerSocket
```

---

public class **ServerSocket**
extends Object
Diese Klasse implementiert Server Sockets. Ein Server Socket wartet auf Anfragenaus dem Netzwerk. Der Server Socket führt basierend auf dieser Anfrage verschiedene Operationen aus und liefert deren Ergebnisse an den Client. Die aktuelle Arbeit wird von einer Instanz der `SocketImpl` Klasse ausgeführt. Eine Applikation kann die Socket Factory, mit deren Hilfe Socket Implementation realisiert werden, konfigurieren und passende Sockets bauen.

**Seit:**
JDK1.0
**Siehe Auch:**
`SocketImpl`, `setSocketFactory(java.net.SocketImplFactory)`

---

8.6.2.1.1. Konstruktor Übersicht

| | |
|---|---|
| **`ServerSocket`**`(int port)`<br>Kreiert einen Socket für einen bestimmten Port. | |
| **`ServerSocket`**`(int port, int backlog)`<br>Kreiert einen ServerSocket zum Port . | |
| **`ServerSocket`**`(int port, int backlog, InetAddress bindAddr)`<br>Kreiert einen Server mit spezifiziertem Port, Backlog, und lokaler IP Adresse | |

8.6.2.1.2. Methoden Übersicht

| | |
|---|---|
| `Socket` | **`accept`**`()`<br>hört auf Verbindungsanfragen und nimmt diese entgegen. |
| `void` | **`close`**`()`<br>schliesst diesen Socket. |
| `InetAddress` | **`getInetAddress`**`()`<br>liefert die lokale Adresse dieses Server Sockets. |
| `int` | **`getLocalPort`**`()`<br>liefert den Port auf dem dieser Socket Anfragen erwartet. |
| `int` | **`getSoTimeout`**`()`<br>ermittelt den Wert des SO_TIMEOUT Parameters. |
| `protected void` | **`implAccept`**`(Socket s)`<br>Subklassen von ServerSocket verwenden diese Methode um accept(). |
| `static void` | **`setSocketFactory`**`(SocketImplFactory fac)`<br>definiert die Server Socket Implementation Factory für die Applikation. |
| `void` | **`setSoTimeout`**`(int timeout)`<br>Enable/disable SO_TIMEOUT ; timeout, in Millisekunden. |
| `String` | **`toString`**`()`<br>übliche toString Methode. |

### 8.6.3. Die Konstruktoren

Schauen wir uns nun einige der Konstruktoren genauer an, in Form einfacher Beispiele.

#### 8.6.3.1. public ServerSocket(int port) throws IOException, BindException

Dieser Konstruktor baut einen TCP Server Socket zu einem spezifizierten Port. Falls der Port den Wert 0 hat, dann wählt das darunter liegende System automatisch einen freien Port aus, einen sogenannten *anonymen Port*.

So praktisch wie das aussieht ist das aber nicht:
damit ein Port sinnvoll genutzt werden kann, muss dem Client bekannt sein, auf welchem Port er welche Dienste erwarten kann.

##### 8.6.3.1.1. Programm Fragment

```
try (
     Socket sokHTTP = new ServerSocket(80);
} catch(IOException ioe) {
     System.err.println(ioe);
}
```

Mit diesem Konstruktor wird ein ServerSocket für den Port 80 (HTTP) kreiert. Eine `IOException` wird dann geworfen, falls der Port bereits besetzt ist, oder der Benutzer keine Privilegien hat.

##### 8.6.3.1.2. Instanzierung der ServerSocket Klasse mit Port-Prüfung

Dieses Beispiel benutzt den Konstruktor und überprüft gleichzeitig, ob bestimmte Ports noch frei sind.

##### 8.6.3.1.3. Programm Code

```
//Titel:        public ServerSocket(int port) throws IOException, BindEXception
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  ein einfacher Port Scanner
package Beispiel8_1;
import java.net.*;
import java.io.*;

public class KlasseBeispiel8_1 {

  public static void main(String[] args) {

    ServerSocket ssokServer;

    for (int i = 1024; i <= 65535; i++) {
      try {
        // falls der Port besetzt ist,
        // führt die nächste Zeile zu einem Fehler
```

```
        ssokServer = new ServerSocket(i);
        ssokServer.close();
      }
      catch (IOException e) {
        System.out.println("Port besetzt : Server auf Port " + i + ".");
      } // end try
    } // end for
  }
}
```

#### 8.6.3.1.4. Ausgabe

Es bietet sich an, nicht alle `IOExceptions` anzuzeigen.

```
Kapitel 8 Server Sockets : Beispiel 8_1
Port besetzt : Server auf Port 1025.
Port besetzt : Server auf Port 1026.
Port besetzt : Server auf Port 1027.
Port besetzt : Server auf Port 1028.
Ende Beispiel 8_1
```

#### 8.6.3.1.5. Selbsttestaufgabe

Schreiben Sie ein Java Programm, mit dessen Hilfe Sie einen Server Socket für einen sicher besetzten Port kreieren wollen. Welche Fehlermeldung wird produziert.

### 8.6.3.2. public ServerSocket(lint port int queuelength) throws IOException, BindException

Dieser Konstruktor baut einen ServerSocket für den Port port mit einer Warteschlangenlänge von Maximal queuelength.

#### 8.6.3.2.1. Programm Fragment

```
try {
     ssokHTTP = new ServerSocket(5776, 100);
} catch(IOException ioe) {
     System.err.println(ioe);
}
```

#### 8.6.3.2.2. Selbsttest Aufgabe

Versuchen Sie einen Server Socket zu programmieren, der eine Queue Länge von 2048 hat und auf Port 12 hört.
Ändern Sie das Programm so ab, dass das Programm auch einen Server Socket für den Port 80 kreiert.

#### 8.6.3.3. public ServerSocket((int port, int queuelength, InetAddress bindAddress) throws IOException

Dieser Konstruktor gestattet die Konstruktion eines Server Sockets, der neben Port und Warteschlange auch noch die lokale IP Adresse angibt. Dies erlaubt es , Server Sockets zu konstruieren, die auf Servern eingesetzt werden , die über mehrere IP Adressen verfügen, aber nur über einen Kanal Socket Clients empfangen wollen. Über andere Adressen können weitere Dienste angeboten werden.

##### 8.6.3.3.1. Programm Fragment

```
try {
  ServerSocket ssokHTTP = new ServerSocket(8080, 2048, InetAddress.getByNamet(host) );
} catch (IOException ioe) {
...
}
```

### 8.6.4. Methoden : Informationen über ServerSockets, Kommunikation mit Sockets

Grundlegend für die Server Sockets ist, dass die eigentliche Kommunikation mit Hilfe normaler Sockets statt findet. Bevor diese Kommunikation geschieht, muss der Server Socket mit Hilfe der `accept`() Methode in einen abwartenden Zustand gebracht werden. `accept`() liefert beim Verbindungsaufbau ein Socket, über den dann die Kommunikation statt finden kann.

#### 8.6.4.1. public Socket accept() throws IOException

Der Server Socket muss vorallem auf Verbindungsanfragen warten. Diese Methode macht genau das. Sobald eine Verbindung verlangt wird, liefert sie einen Socket, über den dann die Kommunikation mit dem Client statt finden kann.

##### 8.6.4.1.1. Programm Fragment

```
try {
     ServerSocket ssokServer = new ServerSocket(80);
     while (true)   // Endlosschleife {
          Socket sokVerbindung = ssokServer.accept();
          PrintStream ps = new
               PrintStream(sokVerbindung.getOutputStream() );
          ...
          sokVerbindung.close(); // Ende, aus!
     ...
} catch (IOException ioe) {
     // Fehlerbehandlung
     ...
}
```

#### 8.6.4.1.2. Programmbeispiel : Daytime Server

```
//Titel:       Daytime Server
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einfacher Socket Server, der das
//Daytime Protokoll gemäss RFC 867
//implementiert.
package Beispiel8_2;
import java.net.*;
import java.io.*;
import java.util.Date;

public class KlasseBeispiel8_2 {
  public final static int daytimePort = 13;
  public static void main(String[] args) {
    System.out.println("Kapitel 8 Beispiel 8_2");
    ServerSocket theServer;
    Socket theConnection;
    PrintStream p;

   try {
     theServer = new ServerSocket(daytimePort);
     try {
      System.out.println("Daytime Server wird gestartet...");
       while (true) {
         theConnection = theServer.accept();
         p = new PrintStream(theConnection.getOutputStream());
         System.out.println("Daytime Server läuft");
         p.println(new Date());
         theConnection.close();
         System.out.println("Daytime Server beendet");
       }
     }
     catch (IOException e) {
       theServer.close();
       System.err.println(e);
     }

   }  // end try
   catch (IOException e) {
     System.err.println(e);
   }
   System.out.println("Ende Beispiel 8_2");
  }
```

}

#### 8.6.4.1.3. Ausgabe

```
Kapitel 8 Beispiel 8_2
Daytime Server wird gestartet...
```

Bei erneutem Starten des selben Servers findet das Programm den bereits besetzten Port:

```
Kapitel 8 Beispiel 8_2
java.net.BindException: Address in use: bind
Ende Beispiel 8_2
```

#### 8.6.4.1.4. Selbsttest Aufgabe

Testen Sie den Daytime Server mit dem Daytime Client des letzten Kapitels.

### 8.6.4.2. public InetAddress getInetAddress()

Diese Methode liefert die lokale Adresse, also InetAddress.getLocalHost(), falls der lokale Host nur eine IP Adresse hat. Falls der lokale Host mehrere IP Adressen hat, dann liefert die Methode eine der IP Adressen, aber es ist nicht klar welche.

#### 8.6.4.2.1. Programm Fragment

```
try {
     Socket http = new Socket(80);
     InetAddress ia = http.getInetAddress();
     ...
} catch (IOException ioe) {
     // Fehlerbehandlung
     ...
}
```

### 8.6.4.3. public int getLocalPort()

Falls Sie den Server Socket mit dem Port 0 konstruiert haben, dann sucht das System einen leeren Port und accept() hört an diesem Port auf Verbindungsanfragen.

Damit Sie nicht ganz verloren sind, stellt Ihnen die Klasse eine Methode zur Verfügung, mit deren Hilfe Sie diesen zufälligen Port erfragen können.

#### 8.6.4.3.1. Programmcode Zufallsport

```
//Titel:       public int getLocalPort()
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Bestimmen des zufällig zugeteilten Ports eines Server Sockets
package Beispiel8_3;
import java.net.*;
```

```
import java.io.*;

public class KlasseBeispiel8_3 {

  public static void main(String[] args) {
    System.out.println("Kapitel 8 Beispiel 8_3");
    ServerSocket theServer;

    for (int i = 1024; i <= 1124; i++) {
      try {
        // testen ob der Port frei ist
        // falls nicht : Exception
        theServer = new ServerSocket(i);
        System.out.println("Server an Port "+i+" gestartet");
        theServer.close();
        System.out.println("Server an Port "+i+" geschlossen");
      }
      catch (IOException e) {
        System.out.println("An Port " + i + " ist bereits ein Server.");
      } // end try
    } // end for
  System.out.println("Ende Beispiel 8_3");
  }
}
```

#### 8.6.4.3.2. Ausgabe

```
Kapitel 8 Beispiel 8_3
Server an Port 1024 gestartet
Server an Port 1024 geschlossen
An Port 1025 ist bereits ein Server.
An Port 1026 ist bereits ein Server.
An Port 1027 ist bereits ein Server.
An Port 1028 ist bereits ein Server.
Server an Port 1029 gestartet
Server an Port 1029 geschlossen
...
Server an Port 1123 gestartet
Server an Port 1123 geschlossen
Server an Port 1124 gestartet
Server an Port 1124 geschlossen
Ende Beispiel 8_3
```

Die Details des Sockets werden mit Hilfe des Interfaces SocketOptions spezifiziert, wie bei den Client Sockets. Wir können also auf eine Besprechung verzichten.

### 8.6.5. java.net Interface SocketOptions

**wird implementiert von folgenden Klassen:**

DatagramSocketImpl, SocketImpl

---

public interface **SocketOptions**

Interface für Methoden der get/set socket Optionen. Diese Interface wird implementiert durch: **SocketImpl** and **DatagramSocketImpl**. Subklassen dieser Klassen sollten die Methoden überschreiben, um eigene Optionen zu implementieren.

Die Methoden und Konstanten sind für die Socket Implementation gedacht. Daher sollten Subklassen von SocketImpl oder DatagramSocketImpl nicht verändert werde, ausser dies ist unbedingt nötig..

Ein Subset der Standard BSD (Berkeley) Socket Optionen werden in den JDK Basisklassen, **PlainSocketImpl** und **PlainDatagramSocketImpl** implementiert. Hier eine Kurzbeschreibung dieser Felder.

#### 8.6.5.1. Datenfelder Übersicht

| | |
|---|---|
| static int | **IP_MULTICAST_IF**<br>Interface über das Multicast Pakete versandt werden. |
| static int | **SO_BINDADDR**<br>Lies die lokale Adressbindung eines Socket. |
| static int | **SO_LINGER**<br>Spezifikation eines linger-on-close Timeout Intervalls. |
| static int | **SO_RCVBUF**<br>Buffergrösse für ankommenden Netzwerk IO |
| static int | **SO_REUSEADDR**<br>Setzt SO_REUSEADDR für einen Socket. |
| static int | **SO_SNDBUF**<br>Buffergrösse für zu senden Netzwerk IO. |
| static int | **SO_TIMEOUT**<br>Setzt Timeout für Socket Operationen |
| static int | **TCP_NODELAY**<br>Nagle's Algorithmus ausschalten. |

#### 8.6.5.2. Methoden Übersicht

| | |
|---|---|
| Object | **getOption**(int optID)<br>liest den Wert einer Option. |
| void | **setOption**(int optID, Object value)<br>Enable/disable die Option *optID*. |

## *8.7. Einige komplexere Beispiele*

Nach den grundlegenden Konstruktoren, die alle recht einfach sind, wollen wir nun einige Anwendungsbeispiele ansehen, die etwas komplexer sind:

Im letztes Kapitel haben wir verschiedene Clients kennen gelernt. Viele konnten Sie lokal nicht testen, weil Ihnen der Server dazu fehlt. Hier wollen wir nun einige weitere dieser Server, zumindest die Grundfunktionen davon, implementieren.

### 8.7.1. Lernziele

Sie kennen einige der bekannteren TCP basierten Netzwerk Werkzeuge

- Client Tester
- HTTP File Server
- HTTP Redirector Server
- Java HTTP Server

und können deren Funktionsweise an Hand eines Java Programmes nach vollziehen.

### 8.7.2. Client Tester

Im letztes Kapitel haben wir verschiedene Server mit Hilfe von Telnet getestet. Der Client Tester soll nun die Möglichkeit schaffen, unterschiedliche Clients zu testen, also analog zum Test der Server im letzten Kapitel mit Hilfe von Telnet, jetzt mit Hilfe eines Java basierten Servers, dem Client Tester.

#### 8.7.2.1. Programm Code

```
//Titel:       Client Tester
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einfacher Client Tester, mit zwei Threads,
//je einem für Input und einem für Output
package Beispiel8_4;
import java.net.*;
import java.io.*;

public class KlasseBeispiel8_4 {

  public static void main(String[] args) {
    System.out.println("Kapitel 8 Beispiel 8_4");
    int thePort;
    ServerSocket ss;
    Socket theConnection;

    try {
      thePort = Integer.parseInt(args[0]);
    }
```

```
    catch (Exception e) {
      thePort = 0;
    }

    try {
      ss = new ServerSocket(thePort);
      System.out.println("Warte auf eine Verbindungsanforderung an Port " +
ss.getLocalPort());

      while (true) {
        theConnection = ss.accept();
        System.out.println("Verbindung aufgenommen " + theConnection);
        InputThread it = new InputThread(theConnection.getInputStream());
        it.start();
        OutputThread ot = new OutputThread(theConnection.getOutputStream(), it);
        ot.start();
        // warte bis ot und it beendet sind
        try {
          ot.join();
          it.join();
        }
        catch (InterruptedException e) {
        }
      }
    }
    catch (IOException e) {

    }
    System.out.println("Beispiel 8_4 : Client Tester beendet");
  }

}

class InputThread extends Thread {

  InputStream is;

   public InputThread(InputStream is) {
     this.is = is;
   }

   public void run()  {

     try {
       while (true) {
         int i = is.read();
         if (i == -1) break;
         char c = (char) i;
         System.out.print(c);
       }
```

```
    }
    catch (IOException e) {
      System.err.println(e);
    }

  }

}

class OutputThread extends Thread {

  PrintStream ps;
  DataInputStream is;
  InputThread it;

  public OutputThread(OutputStream os, InputThread it) {
    ps = new PrintStream(os);
    this.it = it;
    is = new DataInputStream(System.in);
  }

  public void run() {

    String line;
    try {
      while (true) {
        line = is.readLine();
        if (line.equals(".")) break;
        ps.println(line);
      }
    }
    catch (IOException e) {

    }
     it.stop();
  }

}
```

#### 8.7.2.2. Ausgabe

```
Kapitel 8 Beispiel 8_4
Warte auf eine Verbindungsanforderung an Port 1034

...
```

#### 8.7.2.3. Selbsttestaufgabe

Testen Sie einen der Clients aus Kapitel 8 mit Hilfe des obigen Client Testers.

### 8.7.3. HTTP Server Stufe 1 : File Server

Schauen wir in mehreren Stufen, die Arbeitsweise eines typischen Webservers an.

#### 8.7.3.1. Programm Code

```
//Titel:       HTTP File Server
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einfacher File Server auf HTTP Basis
package Beispiel8_5;
import java.net.*;
import java.io.*;
import java.util.*;

public class KlasseBeispiel8_5 extends Thread {

  static String theData = "";
  static String ContentType;
  Socket theConnection;

  public static void main(String[] args) {

    int thePort;
    ServerSocket ss;
    Socket theConnection;
    FileInputStream theFile;

    // cache the file
    try {
      theFile = new FileInputStream(args[0]);
      DataInputStream dis = new DataInputStream(theFile);
      if (args[0].endsWith(".html") || args[0].endsWith(".htm")) {
        ContentType = "text/html";
      }
      else {
        ContentType = "text/plain";
      }

      try {
        String thisLine = "";
        while ((thisLine = dis.readLine()) != null) {
          theData += thisLine + "\n";
        }
      }
      catch (Exception e) {
```

```
      System.err.println("Error: " + e);
    }

  }
  catch (Exception e) {
    System.err.println(e);
    System.err.println("Programmaufruf: java KlasseBeispiel8_5 Dateinamen Port");
    System.exit(1);
  }

  //Port setzen
  try {
    thePort = Integer.parseInt(args[1]);
    if (thePort < 0 || thePort > 65535) thePort = 80;
  }
  catch (Exception e) {
    thePort = 80;
  }

  try {
    ss = new ServerSocket(thePort);
    System.out.println("verbindung mit Port "
      + ss.getLocalPort());
    System.out.println("Daten:");
    System.out.println(theData);
    while (true) {
      KlasseBeispiel8_5 fs = new KlasseBeispiel8_5(ss.accept());
      fs.start();
    }
  }
  catch (IOException e) {

  }

}

public KlasseBeispiel8_5(Socket s) {
  theConnection = s;
}

public void run() {

  try {
    PrintStream os = new PrintStream( theConnection.getOutputStream());
    DataInputStream is = new DataInputStream( theConnection.getInputStream());
    String request = is.readLine();
    // falls HTTP/1.0 oder später :  MIME header senden
    if (request.indexOf("HTTP/") != -1) {
      while (true) {  // Test des MIME header
        String thisLine = is.readLine();
```

```
        if (thisLine.trim().equals("")) break;
      }

      os.print("HTTP/1.0 200 OK\r\n");
      Date now = new Date();
      os.print("Date: " + now + "\r\n");
      os.print("Server: KlasseBeispiel8_5 1.0\r\n");
      os.print("Content-length: " + theData.length() + "\r\n");
      os.print("Content-type: " + ContentType + "\r\n\r\n");
    } // end if
    os.println(theData);
    theConnection.close();
  }  // end try
  catch (IOException e) {

  }

 }
}
```

#### 8.7.3.2. Erläuterungen

Die Datei, die an den Client geschickt wird, wird in einer Schleife eingelesen und im Memory zwischen gespeichert, so dass man diese Daten nicht bei jedem Client neu ab der Harddisk lesen muss.

```
try {
    theFile = new FileInputStream(args[0]);
    DataInputStream dis = new DataInputStream(theFile);
    if (args[0].endsWith(".html") || args[0].endsWith(".htm")) {
      ContentType = "text/html";
    }
    else {
      ContentType = "text/plain";
    }

    try {
      String thisLine = "";
      while ((thisLine = dis.readLine()) != null) {
        theData += thisLine + "\n";
      }
    }
    catch (Exception e) {
      System.err.println("Error: " + e);
    }
```

Pro Anfrage wird ein eigener Thread kreiert:

```
while (true) {
      KlasseBeispiel8_5 fs = new KlasseBeispiel8_5(ss.accept());
      fs.start();
    }
```

Falls der Benutzer keinen Port angibt, dann versucht der Server Port 80 zu verwenden:

```
//Port setzen
   try {
     thePort = Integer.parseInt(args[1]);
     if (thePort < 0 || thePort > 65535) thePort = 80;
   }
   catch (Exception e) {
     thePort = 80;
   }
```

Der Thread kreiert in seiner `run`() Methode einen `PrintStream.` Als nächstes muss analysiert werden, um welche Version Client es sich handelt und anschliessen werden die Daten, inklusive allen Zusatzinformationen gemäss HTTP Definition, an den Client gesandt.

```
if (request.indexOf("HTTP/") != -1) {
       while (true) {  // Test des MIME header
         String thisLine = is.readLine();
         if (thisLine.trim().equals("")) break;
       }

       os.print("HTTP/1.0 200 OK\r\n");
       Date now = new Date();
       os.print("Date: " + now + "\r\n");
       os.print("Server: KlasseBeispiel8_5 1.0\r\n");
       os.print("Content-length: " + theData.length() + "\r\n"); //Daten aus der Datei
       os.print("Content-type: " + ContentType + "\r\n\r\n");
     } // end if
     os.println(theData);
```

### 8.7.4. HTTP Server Stufe 2 : Redirector

Jetzt wollen wir einen Redirector Server bauen. Das dies auch nicht sehr schwierig ist, dürfte Ihnen aus dem letzten Kapitel bereits klar sein.

#### 8.7.4.1. Programm Code

```
//Titel:       HTTP KlasseBeispiel8_6
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:
package Beispiel8_6;
import java.net.*;
import java.io.*;
```

```
import java.util.*;

public class KlasseBeispiel8_6 extends Thread {

  Socket theConnection;
  static String theSite;

  public KlasseBeispiel8_6(Socket s) {
    theConnection = s;
  }

  public static void main(String[] args) {

    int thePort;
    ServerSocket ss;

    try {
      theSite = args[0];
    }
    catch (Exception e) {
      theSite = "http://www.hta.fhz.ch";
    }

    // slash entfernen
    if (theSite.endsWith("/")) {
      theSite = theSite.substring(0, theSite.length()-1);
    }

    try {
      thePort = Integer.parseInt(args[1]);
    }
    catch (Exception e) {
      thePort = 80;
    }

    try {
      ss = new ServerSocket(thePort);
      System.out.println("Redirection der Verbindung zu Port " + ss.getLocalPort()
        + " to " + theSite);

      while (true) {
        KlasseBeispiel8_6 rd = new KlasseBeispiel8_6(ss.accept());
        rd.start();
      }
    }
    catch (IOException e) {

    }

  }  // end main
```

```
  public void run() {
    try {
      PrintStream os = new PrintStream( theConnection.getOutputStream());
      DataInputStream is = new DataInputStream( theConnection.getInputStream());
      String get = is.readLine();
      //StringTokenizer zum Parsen des verlangten Dokuments
      // damit die korrekte URL Adresse aufgebaut werden kann
      StringTokenizer st = new StringTokenizer(get);
      st.nextToken();  //Mthod,  "GET" oder "POST"
      String theFile = st.nextToken();

      //  HTTP/1.0 oder neuere Version?
      try {
        if (st.nextToken().startsWith("HTTP/")) {
          //Leerzeile
          while (true) {
            String thisLine = is.readLine();
            if (thisLine.trim().equals("")) break;
          }

          // HTTP 1.0 Header
          os.print("HTTP/1.0 302 FOUND\r\n");
          Date now = new Date();
          os.print("Date: " + now + "\r\n");
          os.print("Server: MiniKlasseBeispiel8_6 1.0\r\n");
          os.print("Location: " + theSite + theFile + "\r\n");
          os.print("Content-type: text/html\r\n\r\n");
        } // end if
      } // end try
      catch (NoSuchElementException e) {
        // Client versteht
        // HTTP/1.0 nicht
        // MIME header nicht senden
      }
      // Browser Konformität herstellen
      os.println("<HTML><HEAD><TITLE>Dokument wurde
verschoben</TITLE></HEAD>");
      os.println("<BODY><H1>Dokument wurde verschoben</H1>");
      os.println("Das Dokument " + theFile +
        " befindet sich neu an  <A HREF=\"" + theSite + "\">" + theSite +
        "</A>. Bitte modifizieren Sie Ihr Bookmark!<P>");
      os.println("</BODY></HTML>");
      theConnection.close();
    }
    catch (IOException e) {
    }
  }  // end run
}
```

#### 8.7.4.2. Selbsttestaufgabe

Testen Sie das Programm, wobei Sie die Parameter nicht vergessen dürfen, mit Hilfe von Telnet!
Testen Sie das Programm mit einem Browser:

- Sie müssen im Programm den neuen Server angeben (oder als Parameter)
- beim Zugriff auf diesen Server werden Sie dann (bezw. der Browser) an diesen "neuen" Host weiter geleitet

### 8.7.5. HTTP Server Stufe 3 : Java Web Server

Dieser Server ist in der Lage, Dokumente, Applets, Bilder, ... zu liefern, aus einem Root Verzeichnis oder aus Unterverzeichnissen.

#### 8.7.5.1. Programm Code

```
//Titel:       Java HTTP Server
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  ein einfacher HTTP Server mit den Grundfunktionen
package Beispiel8_7;
import java.net.*;
import java.io.*;
import java.util.*;

public class KlassenBeispiel8_7 extends Thread {

  Socket theConnection;
  static File docroot;
  static String indexfile = "index.html";

  public KlassenBeispiel8_7(Socket s) {
    theConnection = s;
  }

  public static void main(String[] args) {

    int thePort;
    ServerSocket ss;

    // get the Document root
```

```
  try {
    docroot = new File(args[0]);
  }
  catch (Exception e) {
    docroot = new File(".");
  }

  // set the port to listen on
  try {
    thePort = Integer.parseInt(args[1]);
    if (thePort < 0 || thePort > 65535) thePort = 80;
  }
  catch (Exception e) {
    thePort = 80;
  }

  try {
    ss = new ServerSocket(thePort);
    System.out.println("Accepting connections on port "
      + ss.getLocalPort());
    System.out.println("Document Root:" + docroot);
    while (true) {
      KlassenBeispiel8_7 j = new KlassenBeispiel8_7(ss.accept());
      j.start();
    }
  }
  catch (IOException e) {
    System.err.println("Server aborted prematurely");
  }

}

public void run() {

  String method;
  String file="";
  String ct;
  String version = "";
  File theFile;

  try {
    PrintStream os = new PrintStream(theConnection.getOutputStream());
    DataInputStream is = new DataInputStream(theConnection.getInputStream());
    String get = is.readLine();
    StringTokenizer st = new StringTokenizer(get);
    method = st.nextToken();
    if (st.hasMoreTokens()) {
      file = st.nextToken();
      if (file.endsWith("/")) file += indexfile;
    }
```

```
    if (st.hasMoreTokens()) version = st.nextToken();
    if (method.equals("GET")) {
      ct = guessContentTypeFromName(file);
      // loop through the rest of the input lines
      while ((get = is.readLine()) != null) {
        if (get.trim().equals("")) break;
      }

      try {
        theFile = new File(docroot, file.substring(1,file.length()));
        FileInputStream fis = new FileInputStream(theFile);
        byte[] theData = new byte[(int) theFile.length()];
        // need to check the number of bytes read here
        fis.read(theData);
        fis.close();

        if (version.startsWith("HTTP/")) {  // send a MIME header
          os.print("HTTP/1.0 200 OK\r\n");
          Date now = new Date();
          os.print("Date: " + now + "\r\n");
          os.print("Server: KlassenBeispiel8_7 1.0\r\n");
          os.print("Content-length: " + theData.length + "\r\n");
          os.print("Content-type: " + ct + "\r\n\r\n");
        }  // end try

        // send the file
        os.write(theData);
        os.close();
      }  // end try
      catch (IOException e) {  // can't find the file
        if (version.startsWith("HTTP/")) {  // send a MIME header
          os.print("HTTP/1.0 404 File Not Found\r\n");
          Date now = new Date();
          os.print("Date: " + now + "\r\n");
          os.print("Server: KlassenBeispiel8_7 1.0\r\n");
          os.print("Content-type: text/html" + "\r\n\r\n");
        }
        os.println("<HTML><HEAD><TITLE>File Not Found</TITLE></HEAD>");
        os.println("<BODY><H1>HTTP Error 404: File Not
Found</H1></BODY></HTML>");
        os.close();
      }
    }
    else {  // method does not equal "GET"
      if (version.startsWith("HTTP/")) {  // send a MIME header
        os.print("HTTP/1.0 501 Not Implemented\r\n");
        Date now = new Date();
        os.print("Date: " + now + "\r\n");
        os.print("Server: KlassenBeispiel8_7 1.0\r\n");
        os.print("Content-type: text/html" + "\r\n\r\n");
```

```
        }
        os.println("<HTML><HEAD><TITLE>Not Implemented</TITLE></HEAD>");
        os.println("<BODY><H1>HTTP Error 501: Not
Implemented</H1></BODY></HTML>");
        os.close();
      }
    }
    catch (IOException e) {

    }
    try {
      theConnection.close();
    }
    catch (IOException e) {
    }

  }

  public String guessContentTypeFromName(String name) {
    if (name.endsWith(".html") || name.endsWith(".htm")) return "text/html";
    else if (name.endsWith(".txt") || name.endsWith(".java")) return "text/plain";
    else if (name.endsWith(".gif") ) return "image/gif";
    else if (name.endsWith(".class") ) return "application/octet-stream";
    else if (name.endsWith(".jpg") || name.endsWith(".jpeg")) return "image/jpeg";
    else return "text/plain";
  }

}
```

#### 8.7.5.2. Selbsttestaufgabe

Testen Sie das Programm, wobei Sie die Parameter nicht vergessen dürfen, mit Hilfe von Telnet!
Testen Sie das Programm mit einem Browser:

- Sie müssen im Programm den neuen Server angeben (oder als Parameter)
- beim Zugriff auf diesen Server werden Sie dann (bezw. der Browser) an diesen "neuen" Host weiter geleitet

### *8.8. Schlussbemerkungen*

Was sind ServerSockets und was kann man damit machen?
Diese Grundfrage lässt sich vereinfacht wie folgt beantworten:

- ServerSockets bieten Dienste an
- sobald ein Client diesen Dienst verlangt, wird die Verbindung zwischen Client und Server mit Hilfe normaler Sockets abgewickelt

- Mit Hilfe von Sockets können zwei Rechner leicht und effizient mit einander kommunizieren. Die Komplexität der Kommunikation ist dabei dem Benutzer nicht sichtbar; er kann einfach lesen und schreiben