Netzwerkverbindung über einen Socket aufbauen

Ein Socket ist in Java ein **Objekt** der Klasse `Socket` (`java.net.Socket`), **das eine Netzwerkverbindung zwischen zwei Maschinen repräsentiert**. Die Maschinen erhalten dadurch gegenseitig Informationen bezüglich des Standorts im Netzwerk (IP-Adresse) und des TCP-Ports.

1) Socket-Verbindung zum Server herstellen

```
Socket chatSocket = new Socket("127.0.0.1",5000);
```

2) Daten von einem Socket lesen

Der Client liest Daten vom Server über den Eingabe-Strom des Sockets.

```
chatSocket.getInputStream();
```

Da der Eingabe-Strom ein Byte-Strom ist, müssen die Bytes zunächst in Zeichen umgewandelt werden. Hierfür kann ein Objekt der Klasse `InputStreamReader` verwendet werden.

```
InputStreamReader stream = new InputStreamReader(chatSocket.getInputStream());
```

Die Zeichen müssen schließlich noch gepuffert werden, falls der Chat-Client nicht schnell genug lesen kann. Hierfür wird ein Objekt der Klasse `BufferedReader` erzeugt.

```
BufferedReader reader = new BufferedReader(stream);
```

Jetzt können Daten mit der Operation `readLine()` der Klasse `BufferedReader` vom Socket gelesen werden.

```
String message = reader.readLine();
```

3) Daten in einen Socket schreiben

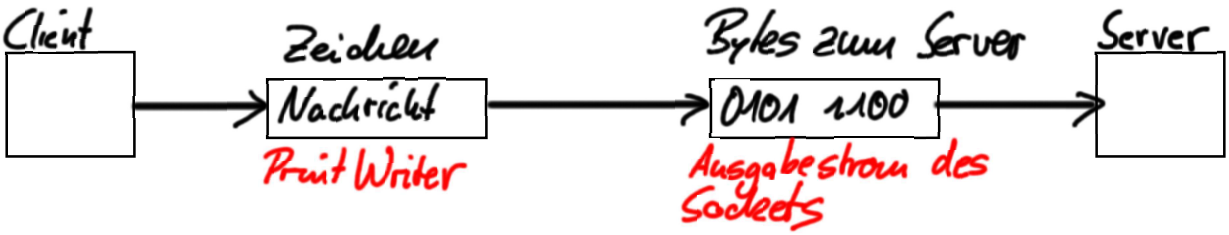
Daten könnten mit einem Objekt der Klasse `BufferedWriter` in den Socket geschrieben werden. Da aber die Daten String für String geschrieben werden, ist es besser, ein Objekt der Klasse `PrintWriter` zu verwenden. Die Klasse besitzt die Operationen `print` und `println`, die völlig ausreichend sind.

Der Client schickt Daten zum Server über den Ausgabe-Strom des Sockets.

```
chatSocket.getOutputStream();
```

Auch hier muss der Strom von Zeichen in einen Byte-Strom übersetzt werden. Dies wird wie oben beschrieben über das `PrintWriter`-Objekt realisiert.

```
PrintWriter writer = new PrintWriter(chatSocket.getOutputStream());
```



Jetzt können Daten mit der Operation `println` der Klasse `PrintWriter` in den Socket geschrieben werden.

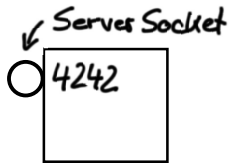
```
writer.println("Nachricht");
```

Eine Server-Anwendung braucht insgesamt zwei Sockets. Dies ist zunächst ein `ServerSocket`-Objekt. Auf diesem Socket wartet der Server auf Client-Anfragen, d.h. dass ein Client mit `new Socket()` einen Socket mit der IP-Adresse des Servers und dem im `ServerSocket`-Objekt angegebenen Port erstellt. Ist dies der Fall, so erzeugt der Server einen neuen Socket für die Kommunikation mit diesem Client (zweiter Socket).

1) ServerSocket des Servers erstellen (erster Socket)

```
ServerSocket serverSocket = new ServerSocket(4242);
while (true) {
     ...
}
```

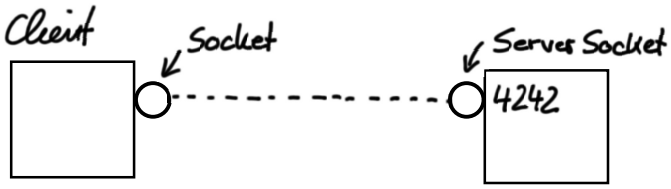
Damit beginnt die Server-Anwendung auf dem Port 4242 auf eingehende Client-Anfragen zu warten.



2) Server erzeugt neuen Socket zur Kommunikation mit dem Client (zweiter Socket)

Kennt ein Client die IP-Adresse des Servers (127.0.0.1) und die Portnummer (4242) der Serveranwendung, so kann er eine Socket-Verbindung zur Server-Anwendung herstellen.

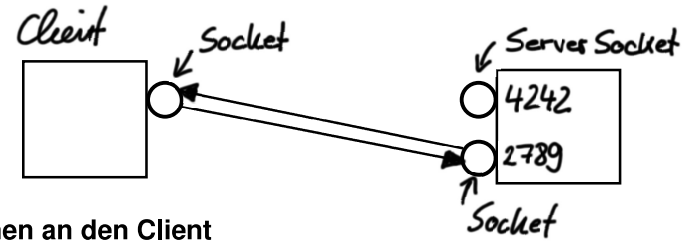
```
Socket clientSocket = new Socket("127.0.0.1",4242);
```



Darauf erzeugt der Server einen neuen Socket für die Kommunikation mit diesem Client.

```
Socket socket = serverSocket.accept();
```

Die `accept`-Operation blockiert während sie auf eine Client-Socket-Verbindung wartet. Wenn dann ein Client versucht, sich zu verbinden, gibt die Operation einen einfachen Socket auf einem anderen Port zurück. Anschließend wartet der ServerSocket wieder auf einen andere Client.



3) Versenden von Informationen an den Client

Wie bereits auf dem letzten Arbeitsblatt gelernt, werden Nachrichten über einen Ausgabe-Strom verschickt. Der Server muss sich also von der neuen Socket-Verbindung den Ausgabe-Strom geben lassen und diesen mit einem `PrintWriter`-Objekt zu verknüpfen, damit die Zeichen-Ströme in Byte-Ströme umgewandelt werden.

```
PrintWriter writer = new PrintWriter(socket.getOutputStream());
writer.println("Nachricht an den Client");
```

Schreibe eine Server-Anwendung, die auf dem Port 5050 läuft und einem Client, der sich über einen Socket mit der Server-Anwendung verbindet und dann einen coolen zufälligen Spruch erhält.

Programmiere den Server und den Client. Teste beide ausführlich.

Anmerkungen zu den Klassen `BufferedReader` und `PrintWriter`

- Ströme können mit der Operation `close()` geschlossen werden.
  ```
  writer.close()
  reader.close()
  ```
- Sollen alle ausstehenden Daten sofort geschrieben werden, bevor der nächste Befehl ausgeführt wird, so kann dies mit der Operation `flush()` erzwungen werden.
  ```
  writer.flush()
  ```

```
try {
    ServerSocket serverSock = new ServerSocket(4242);
    while (true) {
        Socket sock = serverSock.accept();
        PrintWriter writer = new PrintWriter(sock.getOutputStream());
        String tipp = getTipp();
        writer.println(tipp);
        writer.close();
    }
} catch (IOException ex) {
    ex.printStackTrace();
}
```