

B.1 Überblick über die 1. Übung

B.1 Überblick über die 1. Übung

- Streams (Ein-/Ausgabe)
- Sockets (Netzwerkprogrammierung)
- Serialisierung

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

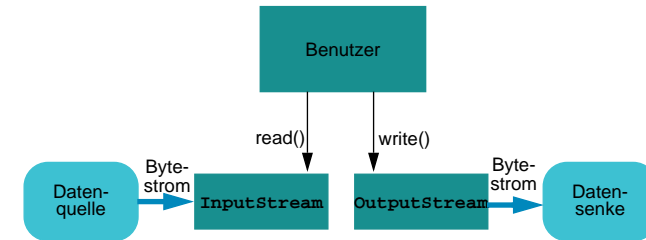
B-Übung1.fm 2008-10-22 12:23

B.1

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Byteströme

B.2 Das Java Ein-/Ausgabesystem



MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

B.3

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.2 Das Java Ein-/Ausgabesystem

B.2 Das Java Ein-/Ausgabesystem

- Grundlegendes Konzept: Ströme (Streams)
 - ◆ Byteströme (InputStream/OutputStream)
 - ◆ Zeichenströme (Reader/Writer)

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

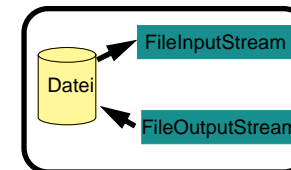
B.2

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Spezialisierungen von Strömen

B.2 Das Java Ein-/Ausgabesystem

- Wo kommen die Daten her, wo gehen sie hin?



MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

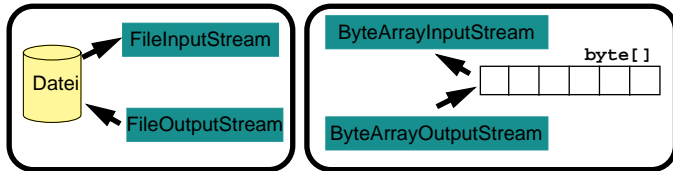
B-Java-Streams.fm 2008-10-22 12:20

B.4

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Spezialisierungen von Strömen (2)

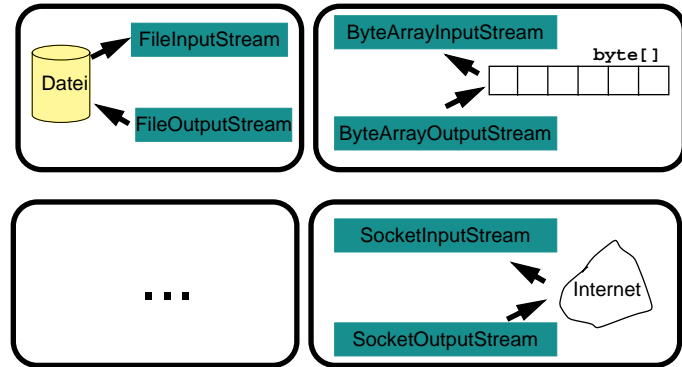
■ Wo kommen die Daten her, wo gehen sie hin?



MW - Übung

2 Spezialisierungen von Strömen (4)

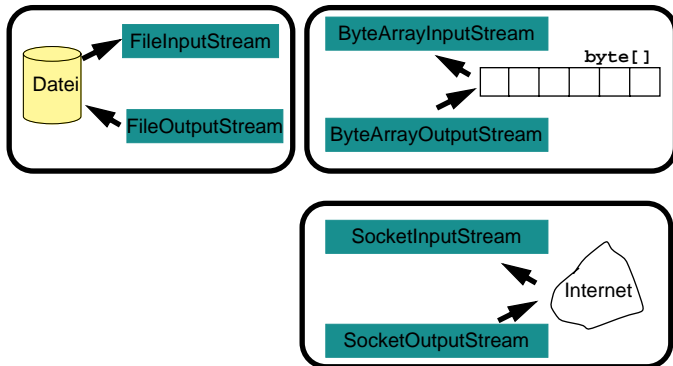
■ Wo kommen die Daten her, wo gehen sie hin?



MW - Übung

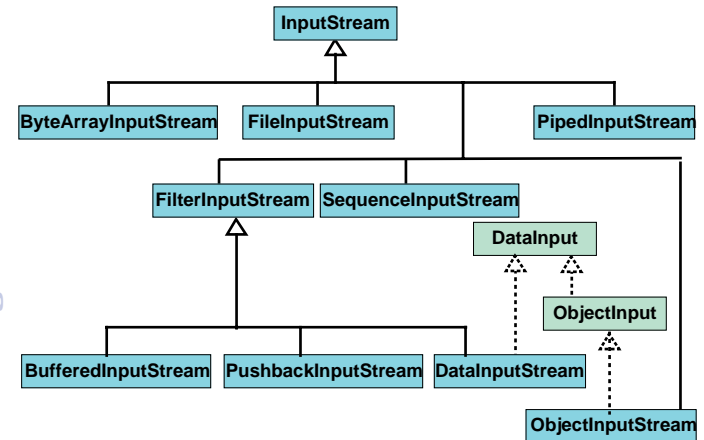
2 Spezialisierungen von Strömen (3)

■ Wo kommen die Daten her, wo gehen sie hin?



MW - Übung

3 Klassendiagramm der Eingabeströme



MW - Übung

4 FileInputStream

- Aus einer Datei lesen:

```
import java.io.*;

public class InTest {
    public static void main (String argv[] throws IOException {
        FileInputStream f = new FileInputStream ("/tmp/test");
        byte buf[] = new byte[4];
        f.read(buf);
    }
}
```

5 Kombinieren von Strömen

- Aus einfachen Strömen "komfortable" Ströme generieren
- Der komfortable Strom umhüllt den einfachen Strom
- → Decorator Design-Pattern

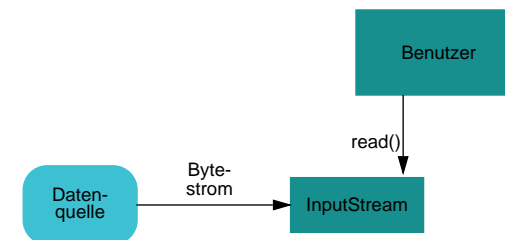
4 FileOutputStream

- In eine Datei schreiben:

```
import java.io.*;

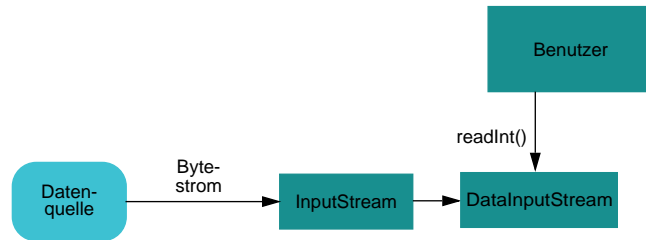
public class OutTest {
    public static void main (String argv[] throws IOException {
        FileOutputStream f = new FileOutputStream ("/tmp/test");
        byte buf[] = new byte[4];
        for (byte i=0; i < buf.length; i++) buf[i]=i;
        f.write(buf);
    }
}
```

5 Kombinieren von Strömen (2)



5 Kombinieren von Strömen (3)

B.2 Das Java Ein-/Ausgabesystem



MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

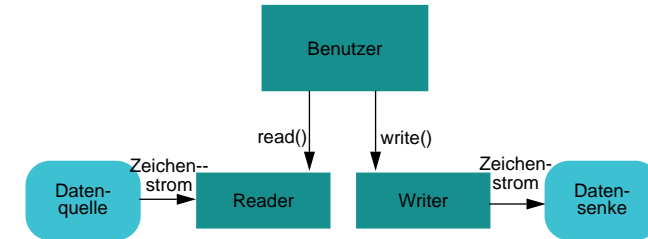
B.13

Regelmäßig jeder Art oder Verwendung dieser Vorlesung, oder zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Reader/Writer

B.2 Das Java Ein-/Ausgabesystem

- Zeichenströme zur Ein- und Ausgabe (**Reader**, **Writer**)



MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

B.15

Regelmäßig jeder Art oder Verwendung dieser Vorlesung, oder zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

- Zeichenströme enthalten Unicodezeichen (16 bit)

6 DataInputStream

B.2 Das Java Ein-/Ausgabesystem

- **InputStream** ist relativ unkomfortabel
- **DataInputStream** wird verwendet um eine *binäre Darstellung* der Daten zu lesen (int, float,...)
- Ein **DataInputStream** kann aus jedem **InputStream** erzeugt werden:

```
InputStream in = new FileInputStream("/tmp/test");
DataInputStream dataIn = new DataInputStream(in);
float f = dataIn.readFloat();
```

- **readLine()** kann verwendet werden um ganze Zeilen zu lesen (Deprecated):

```
for(;;) {
    String s = dataIn.readLine();
    System.out.println(s);
}
```

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

B.14

Regelmäßig jeder Art oder Verwendung dieser Vorlesung, oder zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 Reader

B.2 Das Java Ein-/Ausgabesystem

- wichtige Methoden:
 - ◆ **int read()**
liest ein Zeichen und gibt es als **int** zurück
 - ◆ **int read(char buf[])**
liest Zeichen in ein Array. Liefert die Anzahl der gelesenen Zeichen zurück oder -1 falls ein Fehler aufgetreten ist
 - ◆ **int read(char buf[], int offset, int len)**
liest **len** Zeichen in den Puffer **buf**, beginnend ab **offset**
 - ◆ **long skip(long n)**
überspringt **n** Zeichen
 - ◆ **void close()**
schließt den Strom

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

B.16

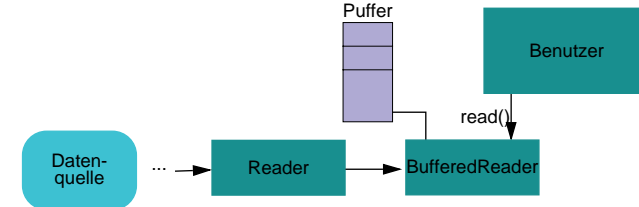
Regelmäßig jeder Art oder Verwendung dieser Vorlesung, oder zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

8 FileReader

- Wird verwendet um aus einer Datei zu lesen
- Konstruktoren:
 - ◆ `FileReader(String fileName)`
 - ◆ `FileReader(File file)`
 - ◆ `FileReader(FileDescriptor fd)`
- Keine weitere Methoden (nur die von `InputStreamReader` geerbt)
- Was ist ein `InputStreamReader`?

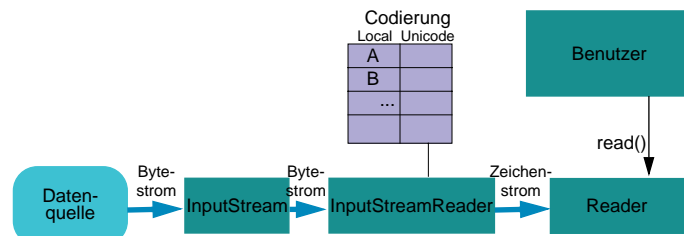
10 Gepufferte Ein-/Ausgabe

- Lesen/Schreiben von einzelnen Zeichen kann teuer sein.
- Umrechnung der Zeichenkodierung kann teuer sein.
- Falls möglich `BufferedReader`, `BufferedWriter` verwenden.
- `BufferedReader` kann aus jedem anderen Reader erzeugt werden.
- Wichtige Methoden von `BufferedWriter`: `void flush()`:
Leert den Puffer - schreibt den Puffer zum ungepufferten Writer:



9 Byte- und Zeichenströme

- Umwandeln von Byteströmen in Zeichenströme mit Hilfe einer Codierung



- einige Codierungen: "Basic Latin", "Greek", "Arabic", "Gurmukhi"

10 Gepufferte Ein-/Ausgabe (2)

- `BufferedReader` kann ganze Zeilen lesen: `string readLine()`

```

BufferedReader in = new BufferedReader(new FileReader("test.txt"));
String line = in.readLine();

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String line = in.readLine();
  
```

11 PrintWriter

B.2 Das Java Ein-/Ausgabesystem

- Kann von jedem `OutputStream` oder `Writer` erzeugt werden.
- `println(String s)`: schreibt den String und das/die EOL Zeichen.
- Beispiel: Datei einlesen und auf der Standardausgabe ausgeben:

```
import java.io.*;

public class CopyStream {
    public static void main(String a[]) throws Exception {
        BufferedReader in = new BufferedReader(
            new FileReader("test.txt"));
        PrintWriter out = new PrintWriter(System.out);
        for(String line; (line = in.readLine())!=null;) {
            out.println(line);
        }
        out.close();
    }
}
```

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Streams.fm 2008-10-22 12:20

B.21

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

12 FileWriter

B.2 Das Java Ein-/Ausgabesystem

- `FileWriter` wird verwendet um Zeichen in eine Datei zu schreiben.
 - ◆ Nachdem das Schreiben beendet ist sollte `close()` aufgerufen werden!

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

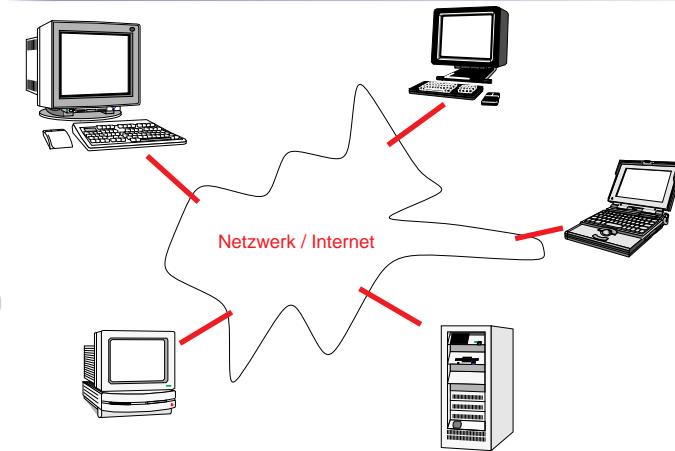
B-Java-Streams.fm 2008-10-22 12:20

B.22

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.3 Netzwerkprogrammierung

B.3 Netzwerkprogrammierung



MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10:00

B.23

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Adressierung: InetAddress

B.3 Netzwerkprogrammierung

- IP-Adresse:
 - ◆ Hostname (Umwandlung durch DNS): `www4.informatik.uni-erlangen.de`
 - ◆ durch Punkte getrenntes Quartupel: `131.188.34.200`
- `java.net.InetAddress` enthält IP-Adressen
- `InetAddress` hat keinen öffentlichen Konstruktor. Instanzen können folgendermassen erzeugt werden:
 - ◆ `getLocalHost()`
 - ◆ `getByName(String hostname)`
 - ◆ `getAllByName(String hostname)`
- `InetAddress` stellt Konvertierungsmethoden zur Verfügung:
 - ◆ `byte[] getAddress()`: IP-Adresse in Bytearray
 - ◆ `String.getHostAddress()`: Stringdarstellung
 - ◆ `String.getHostByName()`: Rechnername (DNS-Form)

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

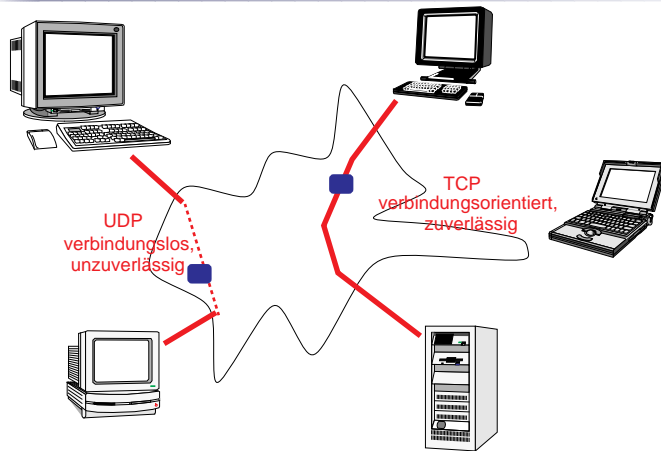
B-Java-Sockets.fm 2008-10-23 10:00

B.24

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Sockets

B.3 Netzwerkprogrammierung



MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.25

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 ServerSocket

B.3 Netzwerkprogrammierung

- `java.net.ServerSocket`
 - ◆ wird serverseitig verwendet um auf Verbindungsanfragen von Clients zu warten
- `accept` wartet auf Verbindungsanfragen
- für eine neue Verbindung wird ein neues `socket`-Objekt zurückgegeben:

```
ServerSocket serverSocket = new ServerSocket(10412);  
Socket socket = serverSocket.accept();
```
- `close` schließt den Port.

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.27

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Verbindungsorientierte Sockets

B.3 Netzwerkprogrammierung

- `java.net.Socket`
 - ◆ TCP/IP
 - ◆ zuverlässig
 - ◆ Repräsentiert einen Kommunikationsendpunkt bei einem Client oder einem Server
- Erzeugen eines neuen Sockets:

```
socket = new Socket("www4.informatik.uni-erlangen.de", 80);
```
- Ein Kommunikationsendpunkt ist definiert durch *Rechnername und Port* (Ports: 16 bit, < 1024 privilegiert)
- `close` schließt den Socket.

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.26

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Ein- / Ausgabe über Sockets

B.3 Netzwerkprogrammierung

- Lesen von einem Socket mittels `InputStream`:

```
InputStream inStream = socket.getInputStream();
```
- Schreiben auf einen Socket mittels `OutputStream`:

```
OutputStream outStream = socket.getOutputStream();
```
- aus diesen Strömen können leistungsfähigere Ströme erzeugt werden:

```
DataOutputStream out =  
    new DataOutputStream(new BufferedOutputStream(outStream));
```

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.28

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 TCP Client/Server

B.3 Netzwerkprogrammierung



Server

```
ServerSocket serverSocket = new ServerSocket(5124);
```

Client

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10:00 B.29
Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 TCP Client/Server (3)

B.3 Netzwerkprogrammierung



Server

```
ServerSocket serverSocket = new ServerSocket(5124);  
Socket socket = serverSocket.accept(); // accept blockiert
```

Client

```
Socket socket = new Socket("hal", 5124);
```

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10:00 B.31
Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 TCP Client/Server (2)

B.3 Netzwerkprogrammierung



Server

```
ServerSocket serverSocket = new ServerSocket(5124);  
Socket socket = serverSocket.accept(); // accept blockiert
```

Client

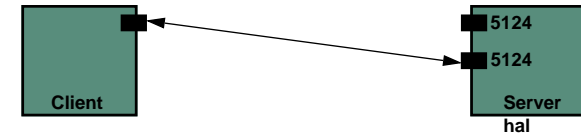
MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10:00 B.30
Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 TCP Client/Server (4)

B.3 Netzwerkprogrammierung



Server

```
ServerSocket serverSocket = new ServerSocket(5124);  
Socket socket = serverSocket.accept(); // accept kehrt zurück
```

Client

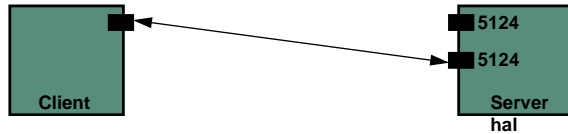
```
Socket socket = new Socket("hal", 5124);
```

MW - Übung

Übungen zu Middleware
© Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10:00 B.32
Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 TCP Client/Server (5)



Server

```
ServerSocket serverSocket = new ServerSocket(5124);
Socket socket = serverSocket.accept();
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
```

Client

```
Socket socket = new Socket("hal", 5124);
InputStream in = socket.getInputStream();
OutputStream out = socket.getOutputStream();
```

4 UDP Empfänger

- Pakete an einem bestimmten Port empfangen:

```
DatagramSocket socket = new DatagramSocket(10412);
byte[] buf = new byte[1024];
DatagramPacket packet = new DatagramPacket(buf, buf.length);

socket.receive(packet);

InetAddress from = packet.getAddress();
int bytesReceived = packet.getLength();
```

4 Verbindungslose Sockets

- `java.net.DatagramSocket`

- ◆ UDP/IP
- ◆ unzuverlässig: **Datagramme können verloren gehen!**
- ◆ geringe Latenzzeit
- ◆ Konstruktoren:

```
DatagramSocket(int port)
bindet an den lokalen Port port
```

```
DatagramSocket()
bindet an irgendeinen lokalen Port
```

- ◆ Methoden:

```
send(DatagramPacket packet)
sendet ein Paket, die Zieladresse muss im Paket eingetragen sein
```

```
receive(DatagramPacket packet)
empfängt ein Paket, die Absenderadresse ist im Paket enthalten
```

4 UDP Sender

- Pakete von einem beliebigen Port verschicken:

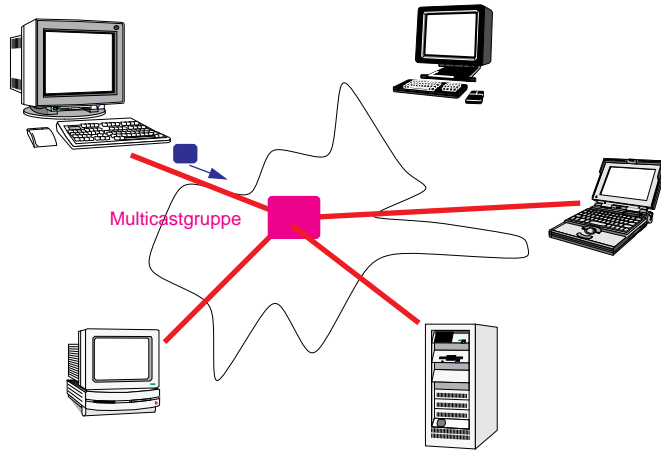
```
DatagramSocket socket = new DatagramSocket();
byte[] buf = new byte[1024];
buf[0] = ...

InetAddress addr = InetAddress.getByName("faii40");
int port = 10412;
DatagramPacket packet;
packet = new DatagramPacket(buf, buf.length, addr, port);

socket.send(packet);
```

5 Multicast-Sockets

B.3 Netzwerkprogrammierung



MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.37

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Multicast-Sockets

B.3 Netzwerkprogrammierung

- `java.net.MulticastSocket`
 - ◆ verbindungslos (Unterklasse von `DatagramSocket`)
 - ◆ verwendet IP-Adressen der Klasse D (224.0.0.1 bis 239.255.255.255)
 - ◆ nach dem Erzeugen des Sockets kann man Pakete verschicken
 - ◆ Um Pakete zu empfangen muss man der Gruppe beitreten mittels `joinGroup()`
 - ◆ Die Paketverbreitung wird mit Hilfe des Parameters "time-to-live" (TTL) gesteuert.

■ Beispiel:

```
InetAddress group = InetAddress.getByName("228.5.6.7");
MulticastSocket socket = new MulticastSocket(6789);
socket.setTimeToLive((byte)2);
socket.joinGroup(group);
```

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

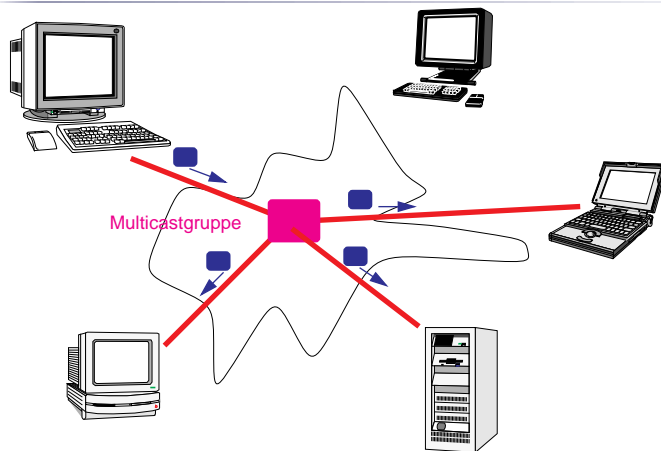
B-Java-Sockets.fm 2008-10-23 10.00

B.39

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Multicast-Sockets

B.3 Netzwerkprogrammierung



MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.38

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Zusammenfassung

B.3 Netzwerkprogrammierung

- `socket`: Endpunkt (Server oder Client) einer TCP-Kommunikation
 - ◆ enthält Zieladresse / -port und lokalen Port
 - ◆ Daten werden mittels Strömen (Streams) gelesen und geschrieben
- `serverSocket`: ein Server-Endpunkt, erzeugt Instanzen von `socket`
- `DatagramSocket`: UDP-Kommunikation
 - ◆ `send()`/`receive()` um Daten zu verschicken / empfangen.
 - ◆ Zieladresse ist in einem `DatagramPacket`-Objekt enthalten.
- `MulticastSocket`: Multicast-UDP-Kommunikation
 - ◆ verwendet einen reservierten Bereich von IP-Adressen
 - ◆ bevor man Daten empfängt, muss man mittels `joinGroup()` einer Multicastgruppe beitreten

MW - Übung

Übungen zu Middleware
©Universität Erlangen-Nürnberg • Informatik 4, 2008

B-Java-Sockets.fm 2008-10-23 10.00

B.40

Reproduktion jeder Art oder Verwendung dieser Vorlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

B.4 Objekt Serialisierung

- Motivation:
 - ◆ Objekte sollen von der Laufzeit einer Anwendung unabhängig sein
 - ◆ Objekte sollen zwischen Anwendungen ausgetauscht werden können

1 Objektströme

- Mit Objektströmen können Objekte in Byteströme geschrieben werden und von dort gelesen werden.
- Klasse `java.io.ObjectOutputStream`
 - ◆ `void writeObject(Object o)`: Serialisierung eines Objekts (transitiv) (`java.io.NotSerializableException`)
- Klasse `java.io.ObjectInputStream`
 - ◆ `Object readObject()`: Lesen eines serialisierten Objekts (`ClassNotFoundException`)

2 Beispiel

- Speichern eines Strings und eines `Date`-Objekts:

```
FileOutputStream f = new FileOutputStream("/tmp/objects");
ObjectOutput s = new ObjectOutputStream(f);
s.writeInt(42);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();
f.close();
```

- Lesen der Objekte:

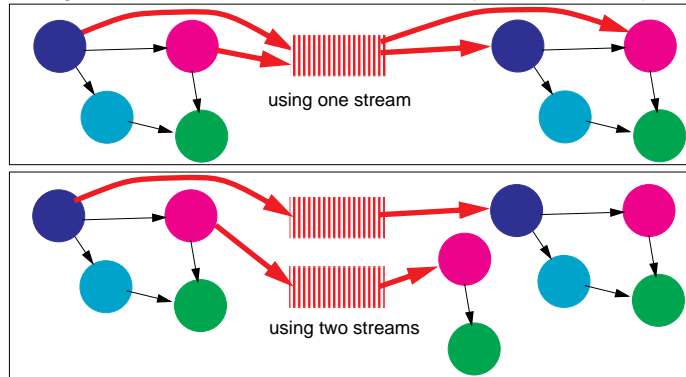
```
FileInputStream in = new FileInputStream("/tmp/objects");
ObjectInputStream s = new ObjectInputStream(in);
int i = s.readInt();
String today = (String)s.readObject();
Date date = (Date)s.readObject(); //! ClassNotFoundException
in.close();
```

3 Schnittstellen

- "Marker Interface" `java.io.Serializable`:
 - ◆ Instanzvariablen werden automatisch gesichert
 - ◆ Variablen, die mit `transient` deklariert wurden, werden nicht gesichert (Klassenvariablen (`static`) auch nicht)
- Interface `java.io.Externalizable`:
 - ◆ Ein Objekt kann seine Serialisierung selbst vornehmen
 - ◆ folgende Methoden müssen implementiert werden:
 - `writeExternal(ObjectOutput out)`
 - `readExternal(ObjectInput in)`

4 Probleme

- Alle Objekte eines Objekt-Graphen sollten in den gleichen Strom geschrieben werden, ansonsten werden die Objekte beim Lesen dupliziert



5 Versionskontrolle der Klassen

- serialisierte Objekte müssen mit der "richtigen" Klasse gelesen werden
- serialisierte Objekte enthalten dazu eine Klassenreferenz, welche den Namen und eine Versionsnummer der Klasse enthält
- Die Versionsnummer wird durch einen Hash-Wert repräsentiert, der über den Klassennamen, die Schnittstellen und die Namen der Instanzvariablen und Methoden gebildet wird.
- Problem: kleine Änderungen an der Klasse führen dazu, dass alte serialisierte Objekte unlesbar sind.
- Lösung:
 - ◆ eine Klasse kann ihre Versionsnummer festlegen:


```
static final long serialVersionUID = 1164397251093340429L;
```
 - ◆ initiale Versionsnummer kann mittels `serialver` berechnet werden.
 - ◆ ⇒ Versionsnummer nur nach inkompatiblen Änderungen verändern

4 Probleme (2)

- der Objekt-Graph muss atomar geschrieben werden
- Klassen werden nicht gespeichert: sie müssen verfügbar sein, wenn ein Objekt später wieder eingelesen wird
- statische Elemente werden nicht gesichert
 - ◆ Lösung: Serialisierung beeinflussen:
 - `private void writeObject(java.io.ObjectOutputStream out)`
 - `private void readObject(java.io.ObjectInputStream in)`

B.5 Hinweise zur 1. Aufgabe

- Lesen von Kommandozeile

```
InputStreamReader is = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(is);
String myline = br.readLine();
```

- StringTokenizer
 - ◆ Schneidet Strings in Tokens
 - ◆ Definiert in: `java.util`
 - ◆ Beispiel:

```
String str = "Hello this is a test"
StringTokenizer tokenizer = new StringTokenizer(str);

// Token einlesen bis zum Ende des Strings
while(tokenizer.hasMoreTokens()) {
    System.out.println(tokenizer.nextToken());
}
```