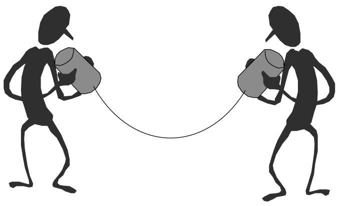
Netzwerkprogrammierung mit Java



Überblick

- URL
- Sockets
- TCP
- UDP
- verteilte Objekte: (nur kurz)
  – Object Serialisation
  – RMI
- Implementierung des Netzwerkprotokolls von Miss Connectivity

In den Beispielen verwenden wir zur Vereinfachung kein Exception-Handling!!!

URL (über HTTP)

- **Laden von HTML-Dokumenten, Gif, MPEG oder Audio-Dateien**
- **Aufruf von CGI Skripten**
  – **mit Parameterübergabe und Empfang der Antwortseite**

- **Erzeugen**

```
URL base = new URL("http://www.gamelan.com/pages/");
URL docs = new URL(base, ”docs/index.html");
```

- **Direktes Lesen**

```
BufferedReader in =
 new BufferedReader(new InputStreamReader(docs.openStream()));
String line;
while ((line = in.readLine()) != null) System.out.println(line);
in.close();
```

URL (über HTTP)

- **Öffnen** (doc wie zuvor erzeugt)

```
URLConnection uc = doc.openConnection();
```

- **Lesen**

```
BufferedReader in =
 new BufferedReader(new InputStreamReader(uc.openStream()));
String line;
while ((line = in.readLine()) != null) System.out.println(line);
in.close();
```

- **Schreiben**

```
uc.setDoOutput(true);
PrintWriter out = new PrintWriter(uc.getOutputStream());
out.println(”Eine Nachricht für Mister X”);
out.close();
```

TCP

- **Punkt-zu-Punkt-Verbindung für die gesamte Dauer der Übertragung**

```
Socket s=new Socket(“cscip.uni-sb.de”,3000)

PrintWriter out = new PrintWriter(s.getOutputStream(),true);

BufferedReader in =
 new BufferedReader(new InputStreamReader(s.getInputStream()));

…
in.close();
out.close();
```

TCP - Client/Server

```
ServerSocket ss = new ServerSocket(3000);
Socket client = ss.accept();
```

- **mehrere Clients ---> Threads**

```
while (true)
{
      warte auf Verbindung mit accept();
      erzeuge Thread für die Verbindung zum Client über neuen Socket
}
```

UDP

- Datagram = in sich geschlossene Nachricht, korrekte Übertragung nicht garantiert

```
DatagramSocket socket = new DatagramSocket();

byte[] buf = new byte[256];
… schreibe Daten in buf
InetAddress address =
   InetAddress.getByName("dings.com");
DatagramPacket packet =
    new DatagramPacket(buf,buf.length,address,3000);
socket.send(packet);
```

UDP

- **Empfange auf Port 3000:**

```
DatagramSocket socket = new DatagramSocket(3000);
packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);
String msg = new String(packet.getData());
```

- **Anworten auf eine Anfrage:**

```
InetAddress address = packet.getAddress();
int port = packet.getPort();
DatagramPAcket packet =
  new DatagramPacket(buf, buf.length, address, port);
socket.send(packet);
```

- **Am Ende der Kommunikation nicht vergessen:**

```
socket.close();
```

Multicasting über UDP

- **Senden an eine Gruppe:** (224.0.0.0 bis 239.255.255.255 , Class-D IP Adressen)

```
InetAddress group = InetAddress.getByName("230.0.0.1");
DatagramPacket packet
 = new DatagramPacket(buf, buf.length,group,3000);
socket.send(packet);
socket.close();
```

- **Empfangen von Nachrichten an eine Gruppe:**

```
MulticastSocket socket = new MulticastSocket(4446);
InetAddress group = InetAddress.getByName("230.0.0.1");
socket.joinGroup(group);
DatagramPacket packet
 = new DatagramPacket(buf, buf.length);
socket.receive(packet);
String msg = new String(packet.getData());
socket.leaveGroup(group);
socket.close();
```

Object Serialisation

- Zustand des Objekts (und seiner referenzierten Teilobjekte) wird in eine Bytefolge umgewandelt.
  - Mit `transient` markierte Variablen werden nicht übertragen (z.B. Filehandle).
  - Nur Objekte von Klassen, die `java.io.Serializable` Schnittstelle implementieren (fast alle Standardklassen)

```
s = new ObjectOutputStream(new FileOutputStream(filename));
s.writeObject(new Date());
s.flush();
s = new ObjectInputStream(new FileOutputStream(filename));
Date d = (Date) s.readObject();
```

Remote Methode Invocation RMI

- Aufruf von Methoden eines Objekts auf einer anderen JVM, die auf einem anderen Rechner laufen kann.

```
interface Oint extends java.rmi.Remote { void meth(); }
class O extends java.rmi.UnicastRemoteObject
        implements Oint { … }
```

- Server:

```
myinst=new O();
Naming.rebind("//localhost/myinstServer",myinst);
```

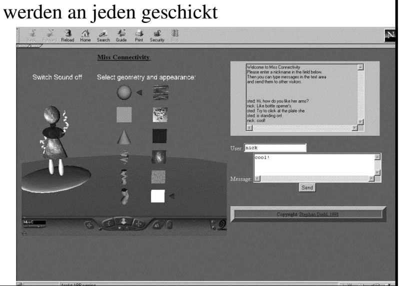
- Client:

```
Oint stub = (Oint)
   Naming.lookup("http://myserver.com/javastuff/myinstServer");
stub.meth();
```

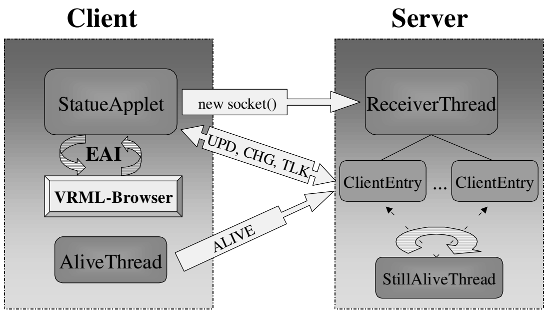
- Wir schauen uns als Alternative später CORBA genauer an!

Java Networking in einer einfachen verteilten, virtuellen Welt

- Miss Connectivity
  - mehrere Benutzer können Statue gleichzeitig ändern
  - alle Änderungen werden an jeden geschickt
  - Textchat



Architektur von MissC



StatueApplet.java

```
public void connect() {
 InetAddress host;
 try { host = InetAddress.getByName(hostname); }
 catch (UnknownHostException e)
   { errout("Unknown host "+hostname); return; }
 try { tcpsock = new Socket(host,port); }
 catch (IOException e)
   { errout("No connection to server!"); return; }
 try { tcpinput = new BufferedReader(new
       InputStreamReader(tcpsock.getInputStream())); }
 catch (IOException e)
    { errout("No input stream from server!"); return; }
 try { tcpoutput =
         new PrintWriter(tcpsock.getOutputStream(),true); }
 catch (IOException e)
  { errout("No output stream to server!"); return; }
}
```

StatueApplet.java

```
public void init()
{  // fordere Handle auf VRML-Browser und Szenenknoten der Statue an
   connect();
   // starte Threads
   // * receiverThread: empfange Nachrichten vom Server
   // * aliveThread: schicke periodisch ALIVE Nachrichten an Server
   receiverThread = new ReceiverThread(this);
   aliveThread = new AliveThread(this);
   running=true;
}
```

ReceiverThread.java

```
public class ReceiverThread extends Thread
{ MUTech mutech;
  public ReceiverThread(MUTech m) { mutech=m; start();}

  public void run()
  { while (true)
      { if (mutech.isRunning())
            mutech.processUpdate(mutech.getUpdate());
        //Thread.yield();
        try{ Thread.sleep(500); }
        catch (InterruptedException e) {}
        }
  }
}
```

StatueApplet.java

```
public void sendUpdate(String upd)
{ tcpoutput.println(upd); }

public String getUpdate()
{ String str;
  try { str=tcpinput.readLine(); }
  catch (IOException e)
   { errout("No response from server!"); str=""; }
  return str;
}
```

StatueApplet.java

```
public void processUpdate(String str)
  { if (str.indexOf("UPD")==0)
       { neue Werte für alle Teile der Statue }
    if (str.indexOf("CHG")==0)
       { neuer Wert für ein Teil der Statue }
    if (str.indexOf("TLK")==0)
       { neue Textzeile für Chatfenster }
```

- **Protokoll:**
- `UPD APP1;GEO1;APP1;GEO1; …;APP2;GEO3;`
- `CHG 1:APP1;GEO3;`
- `TLK sted: Try to click at the plate!`
- `KILL ME`

StatueServer.java

```
public void run() {
 StillAliveThread checkThread=new StillAliveThread(this);
 try { acceptsock=new ServerSocket(port); }
 catch (IOException e)
   { errout("Unable to start server!"); return; }
 while(true)
  { try { newsock = acceptsock.accept(); }
    catch (IOException e)
     { errout("Not accepting requests!"); newsock=null; }
       clients.addElement(new ClientEntry(newsock));
        Thread.yield();
      }
  }
```

StatueServer.java

```
static void sendUpdates(ClientEntry sender, String str)
{ int i;
  synchronized (clients)
  { for(i=0; i<clients.size();i++)
    { ClientEntry cl=(ClientEntry) clients.elementAt(i);
      if (cl!=sender) cl.tcpoutput.println(str);
    }
  }
}
```

ClientEntry.java

ohne Exception-Handling

```
public ClientEntry(Socket newsock)
 { tcpsock=newsock;
   connect();
   this.start();
 }

public  void connect()
{ tcpinput = new BufferedReader(
              new InputStreamReader(tcpsock.getInputStream()));
  tcpoutput = new PrintWriter(tcpsock.getOutputStream(),true);
  String update="UPD ";
  update = … // aktueller Zustand
  tcpoutput.println(update);
}

public void disconnect()
 { this.stop();
   tcpinput.close(); tcpoutput.close(); tcpsock.close();
 }
```

ClientEntry.java

ohne Exception-Handling

```
public void run() {
 boolean alive=true;
 lastAlive=System.currentTimeMillis();
 String str;
 while (alive)
 { str= tcpinput.readLine();
   if (str==null) { Thread.yield(); continue; }
   if (str.equals("ALIVE")) lastAlive=System.currentTimeMillis();
   if (str.indexOf("TLK")==0)
      StatueServer.sendUpdates(null,str);
   if (str.indexOf("CHG")==0)
       { StatueServer.sendUpdates(this,str);
         ändere im aktuellen Zustand }
   if (str.equals("NO RESPONSE") &&
       (System.currentTimeMillis()-lastAlive)>10000)
     { alive=false;}
   if (str.equals("KILL ME")) { lastAlive=(long) 0; alive=false;}
   Thread.yield(); }
}
```

StillAliveThread.java

ohne Exception-Handling

```
public StillAliveThread(StatueServer serv)
  { serverThread=serv; this.start();}

public void run()
{ while (true){
   sleep(10000);
   synchronized(serverThread.clients)
   { for(int i=0; i<serverThread.clients.size();i++)
     { long now=System.currentTimeMillis();
       ClientEntry cl
         =(ClientEntry) serverThread.clients.elementAt(i);
       if (now-cl.lastAlive>10000)
        { cl.disconnect();
          serverThread.clients.removeElementAt(i);
          i--; }
     }
   }
}
```