Informatik IIb

Objektorientierte Programmierung mit Java

Oliver Jack

Fachhochschule Jena
Fachbereich Elektrotechnik und Informationstechnik

Wintersemester 2010/11

Vorlesung 7. Netzwerk, Datenbank, Multimedia

Lernziele dieser Vorlesung

Netzwerk

Client-Server-Anwendungen

Zusammenfassung

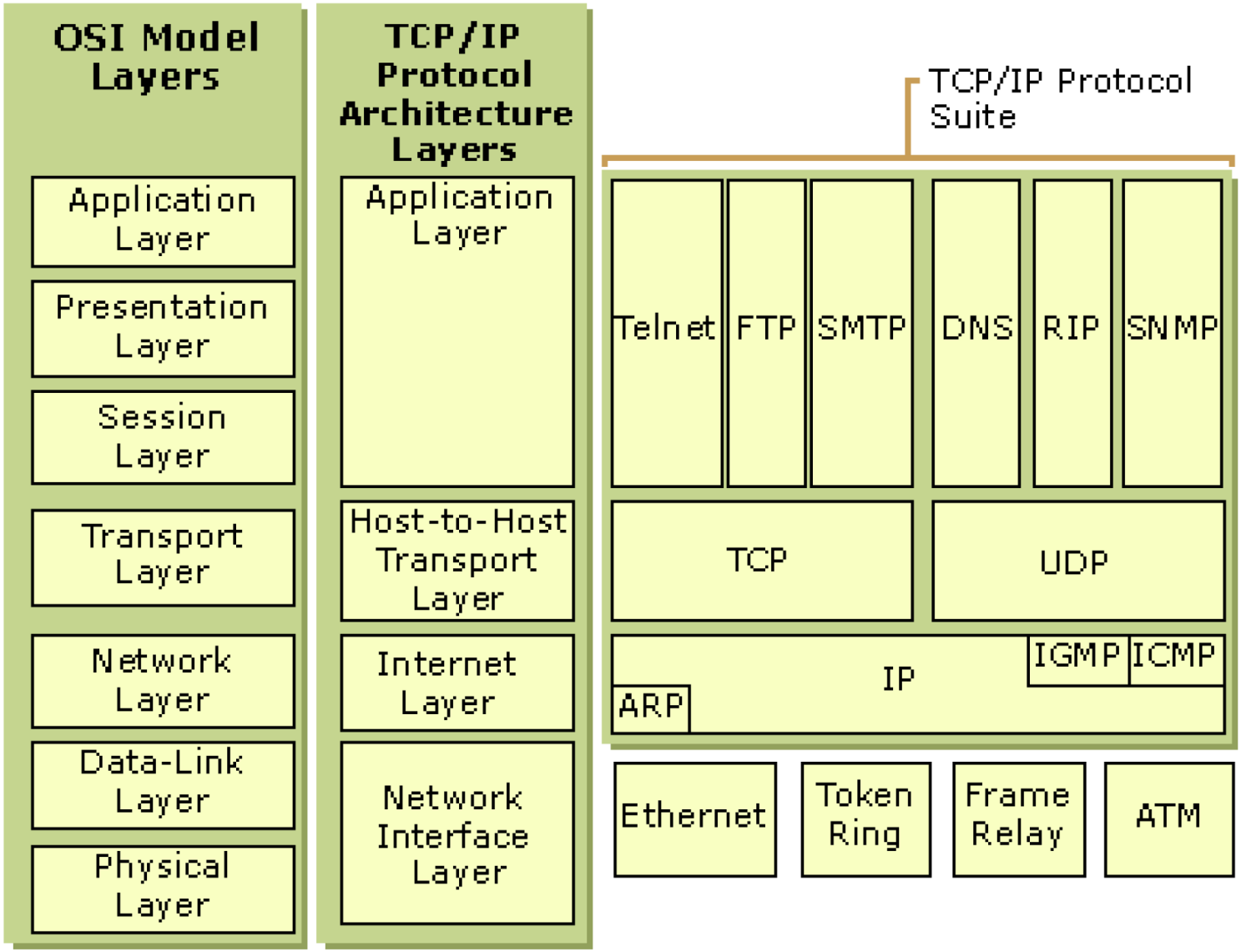
Lernziele

- Kenntnis von Netzwerkzugriffsmechanismen
- Kenntnis Client-Server-Anwendungen

Wichtige Begriffe

- IP steht für Internet Protocol, die Grundlage des heutigen Internets (vgl. OSI Referenz-Modell)
- IP-Adresse bezeichnet so etwas wie die Telefonnummer eines Computers, der in einem Netzwerk hängt
- TCP ist das Transmission Control Protocol, das eine zuverlässige Übertragung zwischen zwei Computern ermöglicht
- RFC steht für Request for Comment
- In RFCs sind zahlreiche de facto Internet-Standards definiert (siehe z. B. http://www.rfc-editor.org)
- Port ist ein spezielle Nummer, die einem Dienst auf einem Server zugeordnet ist, z. B. HTTP läuft gewöhnlich auf Port 80

Internet-Protokoll



URL

Uniform Resource Locator (URL) gemäß RFC 1738

- Benennt den Pfad, unter dem eine Datei im Internet zu finden ist, z. B. http://www.google.com In Java:

```
import java.net.*
URL url1 = new URL("http://www.fh-jena.de:80")
URL url2 =
   new URL("http://www.fh-jena.de/fhj_campus.pac");
```

- Java achtet darauf, dass die URL den Konventionen entspricht
- Eine URL enthält folgende Teile (oder kann enthalten):
  - Protokoll
  - Host (oder IP-Adresse)
  - Dateinamen (in url1 nicht vorhanden)
  - Port (in url2 nicht vorhanden)

Daten über das Netzwerk lesen

Einfache Möglichkeit über InputStreamReader

```
import java.io.*;
import java.net.*;
public class URLReader {
  public static void main(String[] args) throws Exception {
    URL url =
      new URL("http://www.fh-jena.de/fhj_campus.pac");
    BufferedReader in =
      new BufferedReader(new InputStreamReader(url
        .openStream()));
    String str;
    while ((str = in.readLine()) != null)
      if (str.trim().length() != 0)
        System.out.println(str);
    in.close();
  }
}
```

Eine Suchmaschine ansprechen

```
import java.net.URL;
import java.net.URLEncoder;
import java.util.Scanner;
public class URLReader {
  public static void main(String [] args)
  throws Exception {
    String search = "Java";
    search = "p="
      + URLEncoder.encode(search.trim(), "UTF-8");
    URL u = new URL(
      "http://de.search.yahoo.com/search?" + search );
    String r =
      new Scanner(u.openStream()).
        useDelimiter("\\Z").next();
    System.out.println(r);
  }
}
```

Schema POST über URL

```
try {
  // Construct data
  String data = URLEncoder.encode("key1", "UTF-8") + "="
   + URLEncoder.encode("value1", "UTF-8");
  data += "&" + URLEncoder.encode("key2", "UTF-8") + "="
   + URLEncoder.encode("value2", "UTF-8");

  // Send data
  URL url = new URL("http://hostname:80/cgi");
  URLConnection conn = url.openConnection();
  conn.setDoOutput(true);
  OutputStreamWriter wr =
   new OutputStreamWriter(conn.getOutputStream());
  wr.write(data);
  wr.flush();
```

Schema POST über URL (Forts.)

```
    // Get the response
    BufferedReader rd =
     new BufferedReader(
      new InputStreamReader(conn.getInputStream()));
    String line;
    while ((line = rd.readLine()) != null) {
        // Process line...
    }
    wr.close();
    rd.close();
} catch (Exception e) {
}
```

Schema POST über Socket (Forts.)

```
try {
  // Construct data
  String data = URLEncoder.encode("key1", "UTF-8") + "="
   + URLEncoder.encode("value1", "UTF-8");
  data += "&" + URLEncoder.encode("key2", "UTF-8") + "="
   + URLEncoder.encode("value2", "UTF-8");

  // Create a socket to the host
  String hostname = "hostname.com";
  int port = 80;
  InetAddress addr = InetAddress.getByName(hostname);
  Socket socket = new Socket(addr, port);
```

Schema POST über Socket (Forts.)

```
// Send header
String path = "/servlet/SomeServlet";
BufferedWriter wr =
 new BufferedWriter(
  new OutputStreamWriter(socket.getOutputStream(),
  "UTF8"));
wr.write("POST "+path+" HTTP/1.0\r\n");
wr.write("Content-Length: "+data.length()+"\r\n");
wr.write("Content-Type: " +
           "application/x-www-form-urlencoded\r\n");
wr.write("\r\n");

// Send data
wr.write(data);
wr.flush();
```

Schema POST über Socket (Forts.)

```
    // Get response
    BufferedReader rd =
     new BufferedReader(
      new InputStreamReader(socket.getInputStream()));
    String line;
    while ((line = rd.readLine()) != null) {
        // Process line...
    }
    wr.close();
    rd.close();
} catch (Exception e) {
}
```

Socket Client ohne Timeout

```
// Create a socket without a timeout
try {
   InetAddress addr = InetAddress.getByName("java.sun.com");
   int port = 80;

   // This constructor will block
   // until the connection succeeds
   Socket socket = new Socket(addr, port);
} catch (UnknownHostException e) {
} catch (IOException e) {
}
```

Socket Client mit Timeout

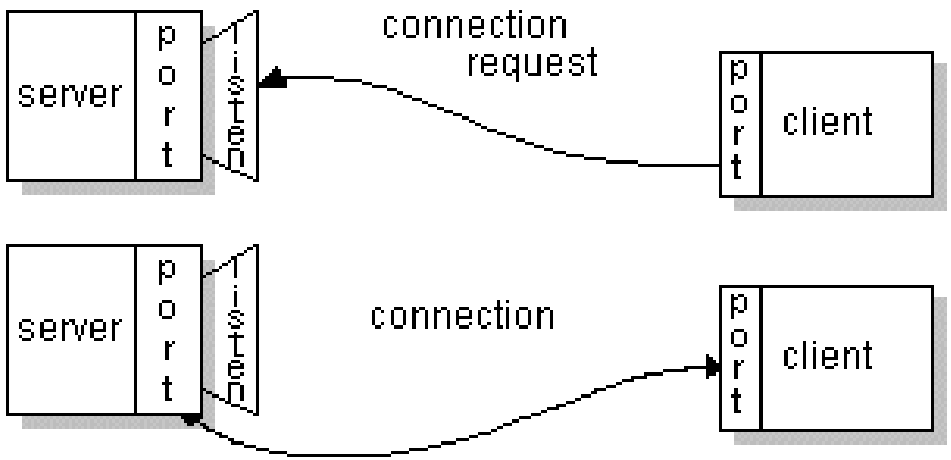
```
// Create a socket with a timeout
try {
   InetAddress addr = InetAddress.getByName("java.sun.com");
   int port = 80;
   SocketAddress sockaddr =
    new InetSocketAddress(addr, port);

   // Create an unbound socket
   Socket sock = new Socket();
   // This method will block no more than timeoutMs.
   // If the timeout occurs,
   // SocketTimeoutException is thrown.
   int timeoutMs = 2000;   // 2 seconds
   sock.connect(sockaddr, timeoutMs);
} catch (UnknownHostException e) {
} catch (SocketTimeoutException e) {
} catch (IOException e) {
}
```

Server und Client

- URL-Verbindung: High-Level — Socket-Verbindung: Low-Level
- Socket-Verbindungen sind geeignet für Server-Client-Anwendungen.
- Server bietet einen Dienst an, Client nutzt ihn.
- Server und Client bauen eine Punkt-zu-Punkt-Verbindung auf.
- Kommunikation erfolgt über TCP.
- Beide binden ein Socket an ihre Kommunikation.
- Ein Socket ist an eine Port-Nummer gebunden.

Socket

- Ein Socket ist ein Endpunkt einer bidirektionalen Kommunikationsverbindung.
- Die Verbindung besteht zwischen zwei Programmen, die in einem Netzwerk laufen.
- java.net bietet zwei Klassen: Socket und ServerSocket.



Kommunikationsprotokoll

Ein Server und ein Client, die mit einander reden

KnockKnock

```
Server: "Knock␣knock!"
Client: "Who's␣there?"
Server: "Dexter."
Client: "Dexter␣who?"
Server: "Dexter␣halls␣with␣boughs␣of␣holly."
```

KnockKnock-Protokoll in Java

Protokoll als Automat realisiert

```
public class KnockKnockProtocol {
  private static final int WAITING = 0;
  private static final int SENTKNOCKKNOCK = 1;
  private static final int SENTCLUE = 2;
  private static final int ANOTHER = 3;

  private static final int NUMJOKES = 5;

  private int state = WAITING;
  private int currentJoke = 0;
```

KnockKnock-Protokoll in Java (Forts.)

Die Kommunikationsworte

```
private String[] clues =
 { "Turnip", "Little␣Old␣Lady",
              "Atch", "Who", "Who" };
private String[] answers =
 { "Turnip␣the␣heat,␣it's␣cold␣in␣here!",
  "I␣didn't␣know␣you␣could␣yodel!",
  "Bless␣you!",
  "Is␣there␣an␣owl␣in␣here?",
  "Is␣there␣an␣echo␣in␣here?" };
```

KnockKnock-Protokoll in Java (Forts.)

Der Automat

```
public String processInput(String theInput) {
  String theOutput = null;
  if (state == WAITING) {
    theOutput = "Knock! Knock!";
    state = SENTKNOCKKNOCK;
  } else if (state == SENTKNOCKKNOCK) {
    if (theInput.equalsIgnoreCase("Who's there?")
      theOutput = clues[currentJoke];
      state = SENTCLUE;
    } else {
      theOutput =
      "You're supposed to say \"Who's there?\"! "
"Try again. Knock! Knock!";
    }
  }
```

KnockKnock-Protokoll in Java (Forts.)

Der Automat

```
    else if (state == SENTCLUE) {
      if (theInput.equalsIgnoreCase(
          clues[currentJoke] + "␣who?")) {
        theOutput = answers[currentJoke] +
         "␣Want␣another?␣(y/n)";
        state = ANOTHER;
      } else {
        theOutput = "You're␣supposed␣to␣say␣\"" +
clues[currentJoke] +
"␣who?\"" +
"!␣Try␣again.␣Knock!␣Knock!";
        state = SENTKNOCKKNOCK;
      }
    }
```

KnockKnock-Protokoll in Java (Forts.)

Der Automat

```
    else if (state == ANOTHER) {
      if (theInput.equalsIgnoreCase("y")) {
        theOutput = "Knock! Knock!";
        if (currentJoke == (NUMJOKES - 1))
          currentJoke = 0;
        else
          currentJoke++;
        state = SENTKNOCKKNOCK;
      } else {
        theOutput = "Bye.";
        state = WAITING;
      }
    }
    return theOutput;
    }
}
```

KnockKnock-Server

ServerSocket

```
import java.net.*;
import java.io.*;

public class KnockKnockServer {
  public static void main(String[] args)
  throws IOException {

    ServerSocket serverSocket = null;
    try {
      serverSocket = new ServerSocket(4444);
    } catch (IOException e) {
      System.err.println(
      "Could␣not␣listen␣on␣port:␣4444.");
      System.exit(1);
    }
```

KnockKnock-Server (Forts.)

Client request akzeptieren

```
Socket clientSocket = null;
try {
  clientSocket = serverSocket.accept();
} catch (IOException e) {
  System.err.println("Accept␣failed.");
  System.exit(1);
}
```

KnockKnock-Server (Forts.)

Lese- und Schreibobjekte erzeugen

```
PrintWriter out =
 new PrintWriter(
  clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(
    new InputStreamReader(
      clientSocket.getInputStream()));
```

KnockKnock-Server (Forts.)

Kommuniziren

```
String inputLine, outputLine;
KnockKnockProtocol kkp =
 new KnockKnockProtocol();
outputLine = kkp.processInput(null);
out.println(outputLine);

while ((inputLine = in.readLine()) != null) {
    outputLine = kkp.processInput(inputLine);
    out.println(outputLine);
    if (outputLine.equals("Bye."))
      break;
}
```

KnockKnock-Server (Forts.)

Kommunikation beenden

```
        out.close();
        in.close();
        clientSocket.close();
        serverSocket.close();
    }
}
```

KnockKnock-Client

Socket, Reader, Writer erzeugen

```
import java.io.*;
import java.net.*;

public class KnockKnockClient {
  public static void main(String[] args)
    throws IOException {

    Socket kkSocket = null;
    PrintWriter out = null;
    BufferedReader in = null;
```

KnockKnock-Client (Forts.)

Verbindung aufbauen

```
try {
  kkSocket = new Socket("bonsai", 4444);
  out = new PrintWriter(
   kkSocket.getOutputStream(), true);
  in = new BufferedReader(
   new InputStreamReader(
    kkSocket.getInputStream()));
} catch (UnknownHostException e) {
  System.err.println(
  "Don't know about host: bonsai.");
  System.exit(1);
} catch (IOException e) {
  System.err.println(
  "Couldn't get I/O for the connection to: bonsa
  System.exit(1);
}
```

KnockKnock-Client (Forts.)

Kommunizieren

```
    BufferedReader stdIn =
     new BufferedReader(new InputStreamReader(
      System.in));
    String fromServer;
    String fromUser;
    while ((fromServer = in.readLine()) != null) {
      System.out.println("Server: " + fromServer);
      if (fromServer.equals("Bye."))
        break;

      fromUser = stdIn.readLine();
      if (fromUser != null) {
        System.out.println("Client: " + fromUser);
        out.println(fromUser);
      }
    }
```

KnockKnock-Client (Forts.)

Verbindung beenden

```
        out.close();
        in.close();
        stdIn.close();
        kkSocket.close();
    }
}
```

Zusammenfassung

- Kommunikation über das Netzwerk erfolgt über TCP bzw. TCP/IP.
- Netzwerkzugriff kann über die URL- und Socket-Klassen in Verbindung mit InputStreamReader und OutputStreamWriter erfolgen.
- Client-Server-Applikationen werden über Sockets realisiert.