1. Vorlesung

Themen: Einführung und Wiederholung Kommunikationssysteme, See-Kabel, dynamische Webseiten und ihre Software Architektur (CGI; PHP; Server Applets

Willkommen im neuen Semester

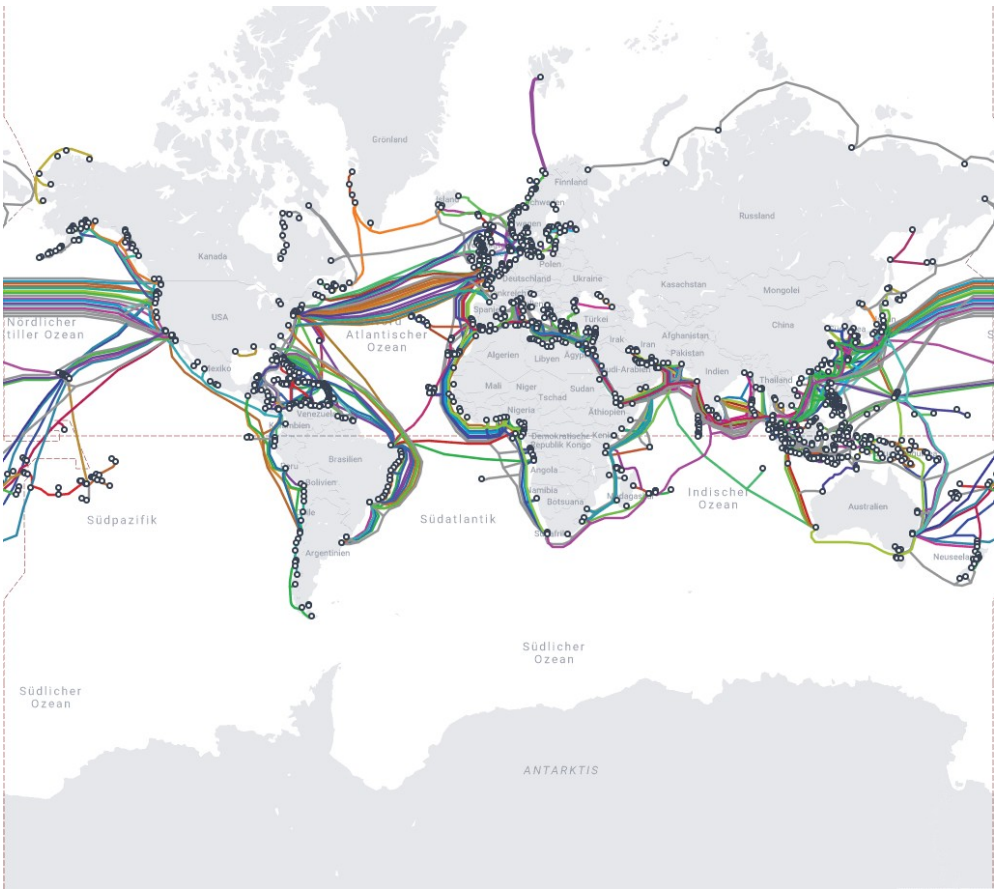
Switching und Routing

Switches sind dazu da, andere Geräte in ein LAN zu binden. Noch vor den Switches gab es Bridges, die die mac-Adressen vermerken und den Datenfluss filtern. Die Bridge überprüft die Zieladresse und leitet die Daten an das gewünschte Netzwerk weiter, wenn in einem Netzwerk ein Gerät etwas an ein anderes senden will.

Switches sind also ähnlich wie Switches, arbeiten aber etwas spezieller und leiten Daten an das gewünschte Gerät weiter, anstatt nur an das Netzwerk. Da die Daten direkt an das gewünschte Gerät versendet werden, ist die Datenübertragung recht effizient.

Seekabel

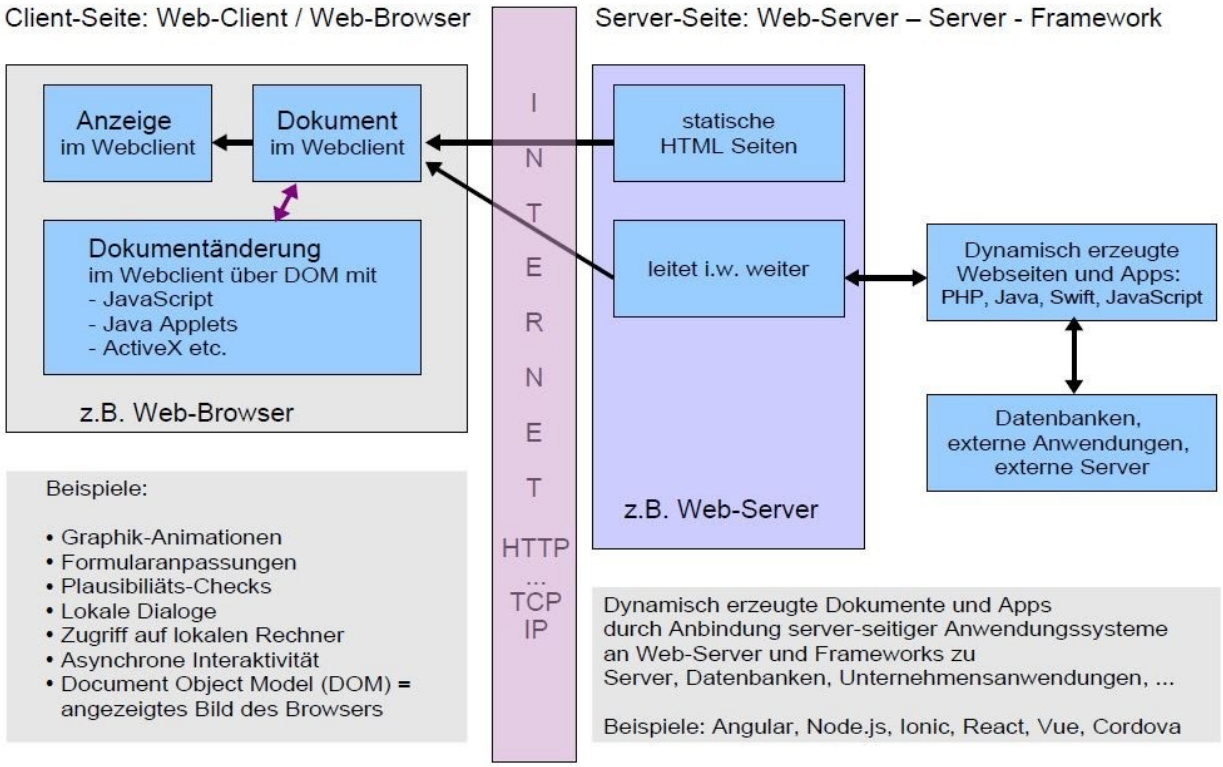
Um auf internationalem Raum vernetzt zu sein, gibt es Seekabel, die z.B. Glasfaserkabel unterirdisch (unterseeisch) verlegen und beinhalten, für den Datenaustausch zwischen Ländern und auf dem Globus. Diese Kabel haben eine extremst hohe Übertragungsrate, sind dabei auf dem Meeresboden verankert und mehrere tausend Kilometer lang. Hier kann man sie sehen:



Dynamische Webseiten und ihre Software Architektur

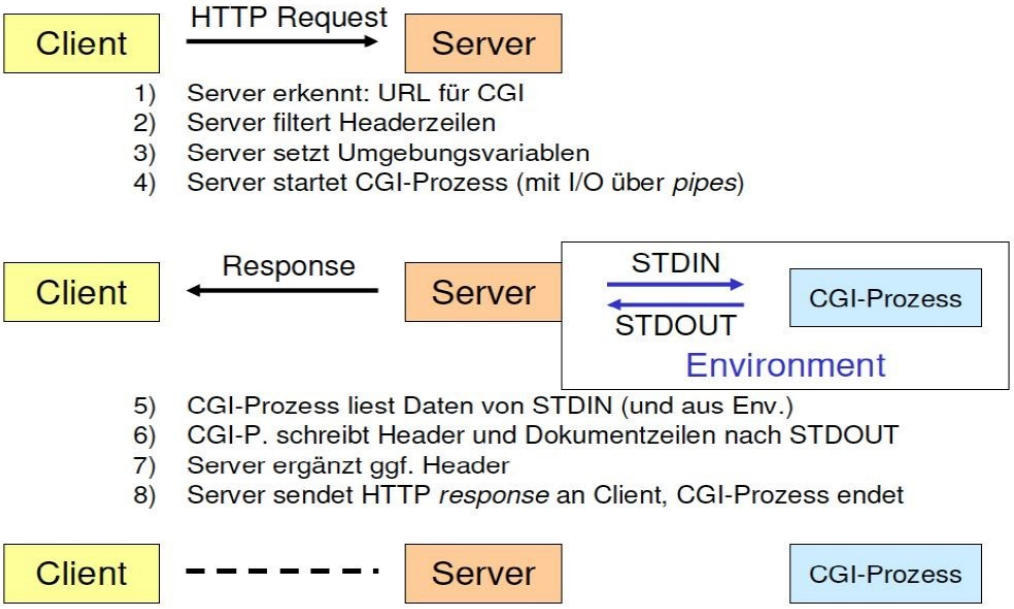
Dieses Semester geht es mehr um serverseitige Technologien, letztes Semester haben wir uns auf clientseitige Technologien fokussiert!

Hier sieht man die beiden Seiten nochmal genauer, auf ein paar Einzelheiten gehe ich gleich ein und erkläre sie.



Serverseitige Technologien sind nicht direkt sichtbar, es geht um die Weiterleitung von HTTP Requests. Hinter dieser Technologie steckt ein Großteil PHP, Java und Ruby (als Programmiersprachen).

Ein Start in serverseitige Technologien war CGI.



CGI

CGI ist eine serverseitige Technologie, die aufgrund von wenig Sicherheit und ihrer langsamen Geschwindigkeit kaum noch verwendet wird.

Dabei werden bestimmte URLs auf Programme anstatt auf Dokumenten abgebildet. Diese Programme werden anschließend vom Webserver gestartet und beendet. Die CGI-Anwendungen können mit praktisch allen Programmiersprachen erstellt werden und können nur Daten erhalten, die der Server passieren lässt.

Schwierig ist aber, dass bei jedem Aufruf einer CGI-URL ein eigener Prozess gestartet wird. Hier kann es schnell zu einer Überlastung kommen.

PHP

Heißt eigentlich Hypertext Preprocessor, denn PHP steht für Personal Home Page (des Entwicklers)!

PHP ist eine Open-Source-Entwicklung und zudem frei verfügbar. Der PHP-Code ist durch eine Start- und Ende-Tag gekennzeichnet, wenn er z.B. in HTML eingebettet ist.
<?php … ? > oder <> (short-open-tag Variante)
Es gibt noch weitere Alternativen.
Der PHP-Code wird vom PHP-Server interpretiert.

Um eine Überschrift zu erzeugen würde der Code wie folgt ausschauen:

** <h1><?php printf(„Überschrift“) ; ? ></h1> **

Java Applets

Java Applets sind wie der Name schon sagt in Java geschrieben und kleine Programme, die im Browser ausgeführt werden können. Sie waren früher sehr beliebt für Datenvisualisierungen oder kleine Spiele, denn sie konnten im Browser angezeigt werden und man konnte mit ihnen interagieren. Sie sind aber jetzt nichtmehr so beliebt und werden aus Sicherheitsgründen im Browser blockiert, da ist JavaScript einfach sicherer.

Es gibt auch Server Applets, die anstatt im Browser auf dem Server ausgeführt werden. Sie übernehmen Funktionen, bei denen man den Server benötigt, wie für komplexere Rechnungen.

Abgabe: Spielfigur/Speicherungstest

Spielfigur

```
package SpielTeam;

import java.io.*;
public class Spielfigur implements Serializable  {
    int staerke;
    String typ;
    String[] waffen;
    public Spielfigur(int staerke, String typ, String[] waffen) {
        this.staerke = staerke;
        this.typ = typ;
        this.waffen = waffen;
    }
    public int getStaerke() {
        return staerke;
    }
    public String getTyp() {
        return typ;
    }
    public String getWaffen() {
        String waffenListe = "";
        for (int i = 0; i < waffen.length; i++) {
            waffenListe += waffen[i] + ", ";
        }
        return waffenListe;
    }
}
```

SpielSpeicherungTest

```
package SpielTeam;

import SpielTeam.Spielfigur;
import java.io.*;
public class SpielSpeicherungTest {
    public static void main(String[] args) {
        Spielfigur eins = new Spielfigur(50, "Elb", new String[] {"Bogen",
"Schwert", "Staub"});
        Spielfigur zwei = new Spielfigur(40, "Troll", new String[] {"bloße
Hände", "Axt"});
        Spielfigur drei = new Spielfigur(150, "gute Fee", new String[]
{"Zaubersprüche", "Flügel"});
        Team MeinTeam = new Team("MeinTeam", new Spielfigur[]{eins, zwei,
drei});
        try {
```

```
            ObjectOutputStream os = new ObjectOutputStream(new
FileOutputStream("Spiel.ser"));
            os.writeObject(MeinTeam);
            os.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        MeinTeam = null;
        eins = null;
        zwei = null;
        drei = null;
        try {
            ObjectInputStream is = new ObjectInputStream(new
FileInputStream("Spiel.ser"));
            Team Wiederhergestellt = (Team) is.readObject();
            is.close();
            System.out.println("Teamname: " + Wiederhergestellt.getTeamname());
            System.out.println("Mitglieder: \n" +
Wiederhergestellt.getMitglieder());
        }catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Team

```
package SpielTeam;

import SpielTeam.Spielfigur;
import java.io.*;
public class Team implements Serializable{
    Spielfigur[] mitglieder;
    String teamname;
    public Team(String teamname, Spielfigur[] mitglieder) {
        this.teamname = teamname;
        this.mitglieder = mitglieder;
    }
    public String getMitglieder() {
        String mitgliederListe = "";
        for (int i = 0; i < mitglieder.length; i++) {
            mitgliederListe += (i+1) + ": ";
            mitgliederListe += "Typ: " + mitglieder[i].getTyp() + ", ";
            mitgliederListe += "Stärke: " + mitglieder[i].getStaerke() + ", ";
            mitgliederListe += "Waffen: " + mitglieder[i].getWaffen() + " \n";
        }
        return mitgliederListe;
    }
    public String getTeamname() {
        return teamname;
    }
}
```

```
Teamname: MeinTeam
Mitglieder:
1: Typ: Elb, Stärke: 50, Waffen: Bogen, Schwert, Staub,
2: Typ: Troll, Stärke: 40, Waffen: bloße Hände, Axt,
3: Typ: gute Fee, Stärke: 150, Waffen: Zaubersprüche, Flügel,
```

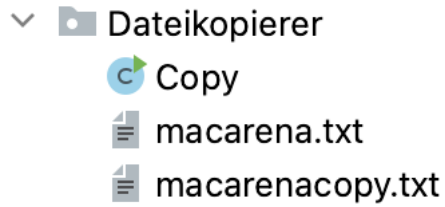
2. Vorlesung

Übung Dateikopierer

Relativ zu Beginn der Vorlesung haben wir eine „Kopierer“ Klasse erstellt, die eine gegebene Datei einliest, den Inhalt kopiert und eine weitere Kopier-Datei erstellt und dort den Inhalt einfügt.

Unten sieht man auch, wie das Ergebnis im gleichen Verzeichnis aussieht und ja, der Inhalt ist identisch.

```
package Dateikopierer;
import java.io.*;
public class Copy {
    public static void main(String[] args) {
        try (BufferedReader reader = new BufferedReader(new
FileReader("/Users/jasmin/IdeaProjects/Verteilte
Systeme/src/Dateikopierer/macarena.txt"));
             BufferedWriter writer = new BufferedWriter(new
FileWriter("/Users/jasmin/IdeaProjects/Verteilte
Systeme/src/Dateikopierer/macarenacopy.txt") )) {
            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```



Java.io File

Die java.io File Klasse ist eine Klasse in der Java-Standardbibliothek. Man kann mit ihrer Hilfe Dateien und Verzeichnisse verarbeiten und darstellen, unter anderem kann man Dateien und Verzeichnisse erstellen, löschen, verschieben und durchsuchen. Die Klasse ist weit verbreitet und die Grundlage für „Dateioperationen“, also um mit Dateien zu interagieren.

Reader & Writer

Um Textdateien in Java einzulesen und zu schreiben, gibt es die Reader und Writer. So sehen sie aus:

```
objectOutputStream.writeObject(MeineDatei);

fileWriter.write("Mein Text");
```

Hier brauchen wir aber try-catch Block, weil es immer eine IOException geben kann. Das sieht so aus:

```
try {

FileWriter writer = new FileWriter("helloworld.txt"); Der Filewriter ist aus java.io

writer.write("Hallo Welt");

writer.close();

}catch (IOException ex){

ex.printStackTrace();
```

Server

Zum Aufbau einer Verbindung zwischen einem Client und Server muss der Client den Server kennen und der Server muss alle Clients kennen, die sich mit ihm verbinden. Hierzu werden TCP/IP Sockets verwendet. Diese sorgen dafür, dass eine Verbindung hergestellt wird.

Die Verbindung zwischen Client und Server fundiert auf den IP-Adressen und den Port-Nummern. Ports sind sozusagen Datentüren beim Rechner und viele davon sind schon belegt. Später müssen wir bestimmte Port-Nummern angeben, um eine Verbindung zu erstellen, damit wir unseren Client und unseren Server zusammen nutzen / verbinden können.

TCP / IP Protokolle

TCP und IP sind Protokolle, die den Datenaustausch ermöglichen.

Das IP Protokoll weist jedem Gerät eine eindeutige IP-Adresse zu und ermöglicht dadurch die Weiterleitung von Datenpaketen vom Sender zum Empfänger.

Das TCP Protokoll ist ein zuverlässiges Übertragungsprotokoll, dass auf IP aufbaut. Hier wird sichergestellt, dass die Datenpakete in richtiger Reihenfolge ankommen, dass sie fehlerfrei sind und keine Paketverluste auftreten. Die Daten werden zuverlässig übertragen und der Empfang (vom Empfänger) bestätigt.

Zusammen sind sie grundlegende Protokolle für den Datenaustausch im Internet. Hier kann man die Protokolle auf den verschiedenen Ebenen des OSI-Modells sehen.

OSI model

Layer	Name	Example protocols
7	Application Layer	HTTP, FTP, DNS, SNMP, Telnet
6	Presentation Layer	SSL, TLS
5	Session Layer	NetBIOS, PPTP
4	Transport Layer	TCP, UDP
3	Network Layer	IP, ARP, ICMP, IPSec
2	Data Link Layer	PPP, ATM, Ethernet
1	Physical Layer	Ethernet, USB, Bluetooth, IEEE802.11

Port-Mapping

Bei Port-Mapping geht es darum, Netzwerkverbindungen von einem externen Netzwerk auf bestimmte Geräte (oder Dienste) in einem internen Netzwerk weiterzuleiten. Externe Nutzer können so auf Webserver zugreifen, indem die Verbindungen vom Router an die Geräte weitergeleitet werden.

Ergo: Da es interne und externe Ports gibt, entscheidet der Router, was wohin gehört.

Abgabe: Tipp des Tages

Tipp des Tages Clientseitigen

```
package TTS;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.Socket;
public class TippdesTagesClient {
    public void los() {
        try (Socket s = new Socket("127.0.0.1", 4242);
             InputStreamReader streamReader = new
InputStreamReader(s.getInputStream());
             BufferedReader reader = new BufferedReader(streamReader)) {
            String zeile;
            while ((zeile = reader.readLine()) != null) {
                System.out.println("Tipp des Tages: " + zeile);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    public static void main(String[] args) {
        TippdesTagesClient client = new TippdesTagesClient();
        client.los();
    }
}
```

Tipp des Tages Server

```
package TTS;
import java.io.IOException;
```

```
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
public class TippdesTagesServer {
    private String[] tippListe = {
            "Trink genug Wasser, sonst ist deine Birne irgendwann matschig.",
            "Plan deine Arbeit im Voraus, sonst bist du am Ende gestresster als
ein Gartenzwerg auf Speed.",
            "Geh raus in die Natur, um den Stress abzubauen und deinen Kopf mal
richtig durchzulüften.",
            "Steh früher auf, um produktiver zu sein - aber keine Sorge, das
heißt nicht, dass du auf Kaffee verzichten musst."
    };
    public void los() {
        try (ServerSocket serverSock = new ServerSocket(4242)) {
            while (true) {
                try (Socket sock = serverSock.accept();
                     PrintWriter writer = new
PrintWriter(sock.getOutputStream())) {
                    for (int i = 0; i < 3; i++) {
                        String tipp = getTipp();
                        writer.println(tipp);
                        System.out.println(tipp);
                    }
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
    private String getTipp() {
        int zufallszahl = (int) (Math.random() * tippListe.length);
        return tippListe[zufallszahl];
    }
    public static void main(String[] args) {
        TippdesTagesServer server = new TippdesTagesServer();
        server.los();
    }
}
```

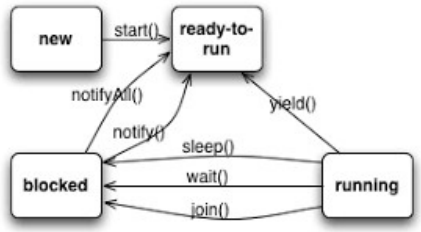
```
/Users/jasmin/Library/Java/JavaVirtualMachines/corretto-11.0.12/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/
Tipp des Tages: Trink genug Wasser, sonst ist deine Birne irgendwann matschig.
Tipp des Tages: Steh früher auf, um produktiver zu sein - aber keine Sorge, das heißt nicht, dass du auf Kaffee verzichten musst.
Tipp des Tages: Plan deine Arbeit im Voraus, sonst bist du am Ende gestresster als ein Gartenzwerg auf Speed.

Process finished with exit code 0
```

3. und 4. Vorlesung

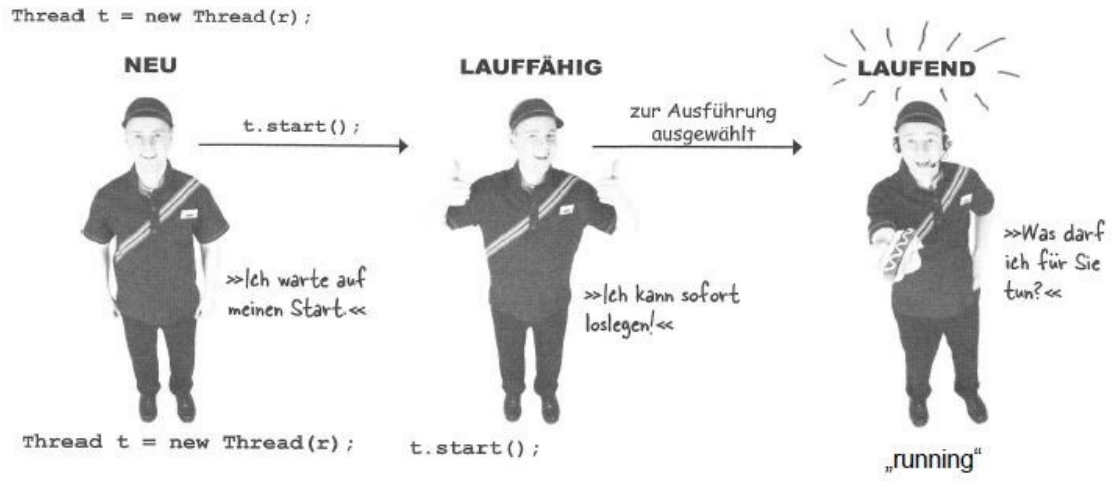
Threads

Threads sind unabhängige Ausführungseinheiten innerhalb eines Programms, die gleichzeitig arbeiten können. Durch sie kann man Aufgaben parallel ausführen und damit die Effizienz des Programmes verbessern! Threads teilen sich den Speicherbereich eines Prozesses. Sie können verschiedene Zustände annehmen:

Neu, Lauffähig, Laufend



Hier sieht man die verschiedenen Zustände, die die Threads annehmen können. Neu, Laufbereit, Laufend und Blockiert auf Deutsch. Der neue Thread kann gestartet werden, wodurch er dann „aktiv“ ist und dann ausgeführt werden kann, indem er in den Laufend Zustand gebracht wird. Threads die gerade nicht lauffähig sind, sind blockiert.



Thread Scheduler

Der Thread Scheduler versetzt lauffähige Threads in den laufenden Zustand. Der Thread Scheduler entscheidet, welcher Thread ausgeführt wird und für wie lange, basierend z.B. auf Prioritäten. Ziel von ihm ist es, den Prozessor so effizient wie möglich zu nutzen, man kann den Scheduler aber nicht kontrollieren, daher schreiben wir dazu unabhängige Programme, die Threads in den Zustand blockiert bringen können, sodass sie kurz aussetzen.

Übung Thread Testlauf

Hier haben wir eine Übung mit Threads gemacht:

ThreadTestLauf.class

```
package ThreadTestLauf;
public class ThreadTestLauf {
    public static void main(String[] args) {
```

```
        Runnable threadJob = new MeinRunnable();
        Thread meinThread = new Thread(threadJob);
        meinThread.start();
        System.out.println("zurück in main");
    }
}
```

MeinRunnable.class

```
class MeinRunnable implements Runnable
{
    public void run()
    {
        los();
    }
    public void los()
    {
        tuNochMehr();
    }
    public void tuNochMehr()
    {
        System.out.println("Oben auf dem Stack");
    }
}
```

ZweiThreads.class

```
package ZweiThreads;
public class ZweiThreads implements Runnable {
    public static void main(String[] args) {
        ZweiThreads aufgabe = new ZweiThreads();
        Thread alpha = new Thread(aufgabe);
        Thread beta = new Thread(aufgabe);
        alpha.setName("Alpha-Thread");
        beta.setName("Beta-Thread");
        alpha.start();
        try {
            Thread.sleep(20);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        beta.start();
    }
    public void run() {
        for (int i = 0; i < 25; i++) {
            String threadName = Thread.currentThread().getName();
            System.out.println("Jetzt läuft der " + threadName);
        }
    }
}
```

Nebenläufigkeitsprobleme

Nebenläufigkeitsprobleme sind ein Beispiel von Threadproblemen. Dabei geht es darum, dass zwei Objekte auf die gleiche Datei zugreifen wollen, wodurch die Daten verändert werden.

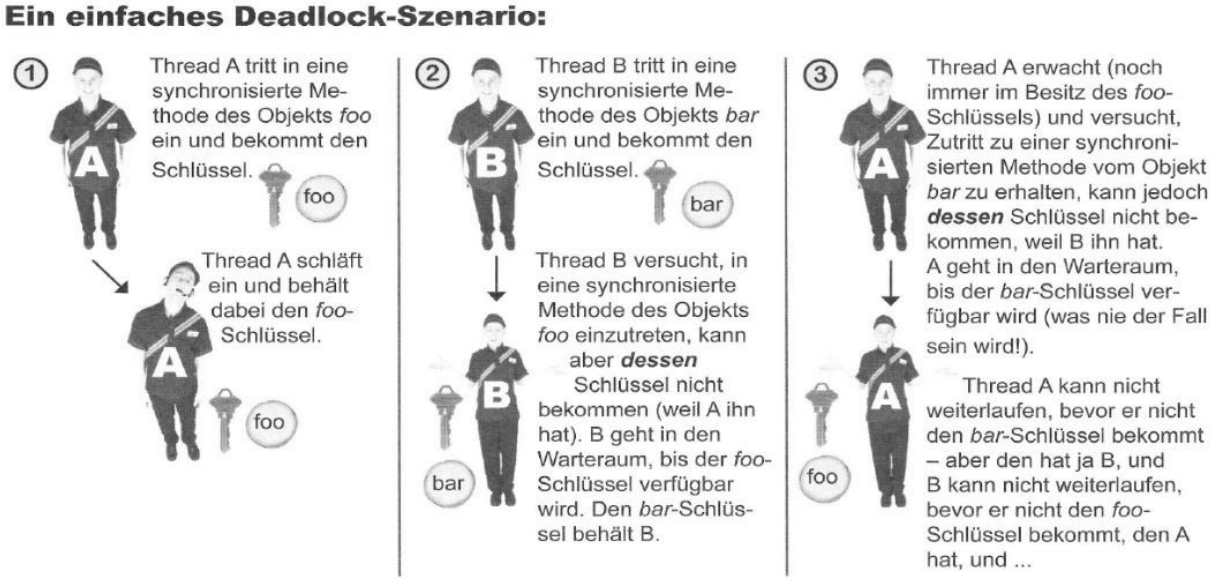
In der Vorlesung hatten wir das Beispiel von Reiner und Monika: Sie wollen beide Geld vom gemeinsamen Konto abheben, bei der Transaktion schläft Reiner aber ein, sodass beide denken, sie

haben den Anfangsbestand auf dem Konto zur Verfügung, dann kann es passieren, dass das Konto überzogen wird, weil beide gleichzeitig Geld abheben.

Als Lösung könnte man alle neuen Transaktionen blocken, wenn noch eine gerade durchgeführt wird. In Java nutzt man dafür „synchronized“. Nur ein Thread kann auf eine Methode zu einer Zeit zugreifen.

Deadlocks

Dazu gibt es auch noch Deadlocks als Threadprobleme, synchronisierte Methoden blockieren sich gegenseitig. Mehr dazu im Bild unten:



Erzeuger/ Verbraucher-Problem

Zu allerletzt gibt es noch das Erzeuger/Verbraucher Threadproblem. Zwei Threads tauschen Daten aus, der „Verbraucher“ will schon auf eine Datei zugreifen, die es aber noch nicht gibt oder noch aktualisiert werden muss. Dabei werden die Daten verändert. Die Kommunikation zwischen den Threads, wie durch die Methoden wait, join und notify, ermöglichen es den Threads untereinander sich auszutauschen.

Chat-Server

Unsere Übung war dieses Mal einen Chat-Server aufzubauen. Et voila:

EinfacherChatClient

```
package Chatserver;
import java.io.*;
import java.net.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class EinfacherChatClient {
    JTextArea eingehend;
    JTextField ausgehend;
```

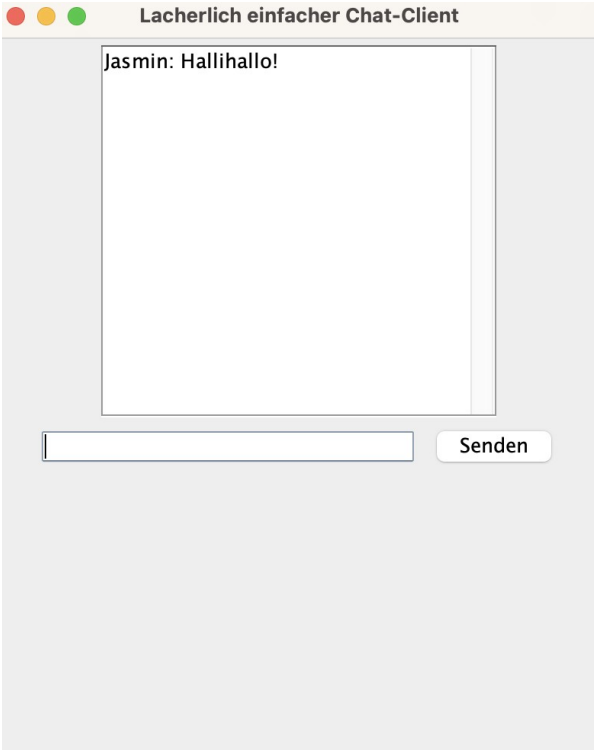
```
    BufferedReader reader;
    PrintWriter writer;
    Socket sock;
    private String senderName = "Jasmin";
    public static void main(String[] args) {
        EinfacherChatClient client = new EinfacherChatClient();
        client.los();
    }
    public void los() {
        JFrame frame = new JFrame("Lacherlich einfacher Chat-Client");
        JPanel hauptPanel = new JPanel();
        eingehend = new JTextArea(15, 20);
        eingehend.setLineWrap(true);
        eingehend.setWrapStyleWord(true);
        eingehend.setEditable(false);
        JScrollPane fScroller = new JScrollPane(eingehend);

fScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWA
YS);

fScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_
NEVER);
        ausgehend = new JTextField(20);
        ausgehend.addKeyListener(new AusgehendKeyListener());
        JButton sendenButton = new JButton("Senden");
        sendenButton.addActionListener(new SendenButtonListener());
        hauptPanel.add(fScroller);
        hauptPanel.add(ausgehend);
        hauptPanel.add(sendenButton);
        netzwerkEinrichten();
        Thread readerThread = new Thread(new EingehendReader());
        readerThread.start();
        frame.getContentPane().add(BorderLayout.CENTER, hauptPanel);
        frame.setSize(400, 500);
        frame.setVisible(true);
    }
    private void netzwerkEinrichten() {
        try {
            sock = new Socket("127.0.0.1", 5050);
            InputStreamReader streamReader = new
InputStreamReader(sock.getInputStream(),"UTF-8");
            reader = new BufferedReader(streamReader);
            writer = new PrintWriter(new
OutputStreamWriter(sock.getOutputStream(),"UTF-8"),true);
            System.out.println("Netzwerkverbindung steht");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    } // netzwerkEinrichten schliessen
    public class SendenButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            try {
                String message = senderName + ": " + ausgehend.getText();
                writer.println(message);
                writer.flush();
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            ausgehend.setText("");
            ausgehend.requestFocus();
        }
```

```
    }//innere Klasse SendenButtonListener schliessen
    public class EingehendReader implements Runnable {
        public void run() {
            String nachricht;
            try {
                while ((nachricht = reader.readLine()) != null) {
                    System.out.println("gelesen: " + nachricht);
                    eingehend.append(new String(nachricht.getBytes("UTF-8")) +
"\n");
                } //Ende der while-Schleife
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }//run schliessen
    }//innere Klasse EingehendReader schliessen
    public class AusgehendKeyListener implements KeyListener {
        @Override
        public void keyTyped(KeyEvent e) {
        }
        @Override
        public void keyPressed(KeyEvent e) {
            if(e.getKeyCode() == KeyEvent.VK_ENTER) {
                sendMessage();
            }
        }
        @Override
        public void keyReleased(KeyEvent e) {
        }
    }
    public void sendMessage() {
        String message = senderName + ": " + ausgehend.getText();
        writer.println(message);
        writer.flush();
        ausgehend.setText("");
        ausgehend.requestFocus();
    }
}//aussere Klasse schliessen
```

EinfacherChatServer

```
package Chatserver;
import java.io.*;
import java.net.*;
import java.util.*;
public class EinfacherChatServer{
    ArrayList<PrintWriter> clientAusgabeStroeme;
    public static void main (String[] args) {
        new EinfacherChatServer().los();
    }
    public class ClientHandler implements Runnable {
        BufferedReader reader;
        Socket sock;
        public ClientHandler(Socket clientSocket) {
            try {
                sock = clientSocket;
                InputStreamReader isReader = new
InputStreamReader(sock.getInputStream());
                reader = new BufferedReader(isReader);
            } catch(Exception ex) {ex.printStackTrace();}
```

```
        } // Konstruktor schliessen
        public void run() {
            String nachricht;
            try {
                while ((nachricht = reader.readLine()) != null) {
                    System.out.println("gelesen: " + nachricht);

esAllenWeitersagen(nachricht,"");//clientSocket.getInetAddress()
                } // Ende der while-Schleife
            } catch(Exception ex) {ex.printStackTrace();}
        } // run schliessen
    } // innere Klasse schliessen
    public void los() {
        clientAusgabeStroeme = new ArrayList<PrintWriter>();
        try {
            ServerSocket serverSock = new ServerSocket(5050);
            System.out.println("Server started on port:
"+serverSock.getInetAddress()+":"+serverSock.getLocalPort());
            while(true) {
                Socket clientSocket = serverSock.accept();
                PrintWriter writer = new
PrintWriter(clientSocket.getOutputStream());
                clientAusgabeStroeme.add(writer);
                Thread t = new Thread(new ClientHandler(clientSocket));
                t.start();
                System.out.println("habe eine Verbindung mit
"+clientSocket.getInetAddress());
            }
            // wenn wir hier angelangt sind, haben wir eine Verbindung
        }catch(Exception ex) {
            ex.printStackTrace();
        }
    }
    public void esAllenWeitersagen(String nachricht, String absender) {
        Iterator it = clientAusgabeStroeme.iterator();
        while(it.hasNext()) {
            try {
                PrintWriter writer = (PrintWriter) it.next();
                writer.println(nachricht);
                writer.flush();
            } catch(Exception ex) {
                ex.printStackTrace();
            }
        } // Ende der while-Schleife
//      Statt einen Iterator und eine while-Schleife zu benutzen,
//      koennten Sie in Java 5 auch wie folgt vorgehen (da wir aus der
//      ArrayList clientAusgabeStroeme hier eine parametrisierte
//      ArrayList<PrintWriter> clientAusgabeStroeme gemacht haben):
//
//      for(PrintWriter writer:clientAusgabeStroeme) {
//         try {
//            writer.println(nachricht);
//            writer.flush();
//         } catch(Exception ex) {
//            ex.printStackTrace();
//         }
//      } // Ende der for-Schleife
//
    } // esAllenWeitersagen schliessen
}
```



5. Vorlesung

Tomcat Server

Da wir uns mit Java Servlets beschäftigt haben, haben wir Tomcat implementiert. Tomcat ist ein Webserver/Servlet Container, damit können wir in einer Webserverumgebung Java Code ausführen. Als „Übung“ oder How-to haben wir dann gemeinsam in einer Session Tomcat installiert/laufen lassen, wie das geht, habe ich durch die folgenden Tutorials gesehen:

https://wolfpaulus.com/tomcat/

https://www.youtube.com/watch?v=2KD7L8j1tio&t=326s

Die Get- und Post-Methode

Die Kommunikation zwischen Server und Clientseitig

Wie funktionierte das nochmal: Der Client sendet eine HTTP-Anfrage (mit einer HTTP-Methode) und bekommt eine HTTP-Antwort. Die Methoden, die man dafür am meisten nutzt sind Get und Post!

Dabei ist die Get-Methode „nur“ ein Aufruf (sowie das Zurücksenden einer Datenressource) und die Post-Methode sendet dazu noch Benutzerdaten und man kann Formulardaten an den Server senden.

Wann benutze ich Get und wann Post? Eigentlich Simpel:

Man nutzt die Get-Methode immer, außer man will einen längeren Text im Suchfeld eingeben, sensible Daten wie Passwörter senden oder kein Lesezeichen für das Formular anlegen.

Apache

Apache ist ein Open-Source Webserver (ein sehr bekannter dazu). Der Apache HTTP Server ist ein leistungsstarker Webserver, der auf verschiedenen Betriebssystemen wie Linux, Windows, macOS und anderen läuft. Der Server unterstützt verschiedene Protokolle wie HTTP, HTTPS, FTP und mehr. Der Apache HTTP Server ermöglicht es, Websiten, dynamische Inhalte und verschiedene Arten von Webanwendungen zu hosten. Es gibt auch Erweiterungen wie Verschlüsselungen, Virtual Hosting, Skriptsprachenunterstützung und mehr.

Dazu gibt es auch die Apache Software Foundation, was eine gemeinnützige Organisation ist, die viele Open-Source Projekte entwickelt. Dazu gehören auch Tomcat (das ist ein Java-Servlet-Container, siehe oben), Maven, Spark und viele andere.

Übung Servlet

Diese Übung haben wir am Ende der 5. Vorlesung aber auch in der 6. Vorlesung behandelt, da sie länger dauert und ein paar kniffelige Stellen hat, also kann es manchmal etwas dauern, bis diese Übung funktioniert.

Ich werde hier mal die Details einsparen um komme zum Punkt: Wir installieren erstmal Tomcat richtig und „bringen es“ im localhost:8080 im Browser zum laufen. Jedenfalls sollte, wenn man in der URL-Leiste im Browser localhost:8080 eingibt und Enter drückt, die Tomcat Website erscheinen.

Danach legen wir ein Dateipfad an, so wie er in der Übung steht und können damit und mit einem Befehl in der Kommandozeile unsere gerade geschriebene (oder von der Tafel abgeschriebene) Datei von eine .java Datei in eine .class Datei umwandeln. Erkennt man als Laie auch an der Kaffeetasse. Diese Datei, zusammen mit der heruntergeladenen (und leicht abgeänderten) web.xml Datei, packen wir jetzt in das Tomcat Verzeichnis unter webapps und verschachteln sie dort in Unterordner wie gezeigt.

Öffnen wir jetzt localhost:8080 im Browser, sieht das so aus:

Kap01Servlet

Mon Jun 05 20:37:48 CEST 2023

Diese Übung hat mich seelisch und moralisch an meine Grenzen gebracht. Alles für die Katz', aber diesmal im wahrsten Sinne des Wortes.

Die Java-Datei:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Kap01Servlet extends HttpServlet {
    public Kap01Servlet() {
    }

    public void doGet(HttpServletRequest var1, HttpServletResponse var2) throws
IOException {
        PrintWriter var3 = var2.getWriter();
        Date var4 = new Date();
        var3.println("<html><body><h1 align=center>VKBF Kapitel 1
Servlet</h1><br>"
                + var4 + "</body></html>");
    }
}
```

Die web.xml Datei packe ich jetzt hier nicht rein, dazu ist sie zu lang. Und sie ist ja auf moodle (obwohl man sie noch ein klitzekleinesbisschen anpassen muss).

6. Vorlesung

Einführung Web-Anwendungen und Apps

Da sich mein Schwerpunkt auf dieses Themengebiet bezieht, bin ich hier nicht ganz so detailgetreu.

Apps für Smartphones und Tablets gibt es als native

Mittlerweile gibt es 3 Kategorien für Smartphone und Tablet Apps – Native Apps, Hybride Apps und Progressive Web-Apps:

Native Apps – werden für jede Plattform einzeln geschrieben, weil sie plattformspezifisch sind.

Hybride Apps – sind nur für mobile Geräte (also Smartphones und Tablets), können aber auf verschiedenen Plattformen laufen (wie bei den Nativen).

Progressive Apps – sehen zwar aus wie Native Apps, sind aber plattformübergreifend nutzbar. (Für Applegeräte also auf iOS&iPadOS gehen sie aber nicht)

Node.js

Um JavaScript auf jeder Plattform, ohne einen Browser, laufen zu lassen, gibt es node.js! Node.js ist eine Laufzeitumgebung für JavaScript. Dadurch kann man es für die serverseitige Entwicklung nutzen.

Asynchrone Programmierung

Asynchrone Programme geben dem Computer die Möglichkeit mehrere Threads gleichzeitig auszuführen, wenn der eine auf beispielsweise Daten von der Festplatte wartet. Wenn ein synchrones Programm Daten anfordern würde, würde der Thread so lange den Prozessor belegen, bis die Daten eingetroffen sind. Dies blockiert den Prozessor unnötig, da in der Wartezeit auf dem Prozessor nicht gearbeitet wird und andere Aufgaben diese Zeit nicht nutzen könnte. Mit synchronen Programmen kann die benötige Zeit nur verkürzt werden, wenn ein zweiter Prozessor oder ein Prozessor mit mehreren Kernen verwendet wird. Der zweite Kern kann parallel eigene Threads ausführen. Bei asynchronen Programmen wird das Problem anders gelöst. Hier kann der Prozessor in der Wartezeit von einem Thread einen anderen bearbeiten und mit dem Ursprünglichem weiter machen, wenn die benötigten Daten eingetroffen sind.

React

React wurde 2011 entwickelt und ist ein bekanntes JavaScript Framework zum Erstellen von Benutzeroberflächen. Es gibt zusätzliche Libraries, die für die React Apps benötigt werden, wie zum Beispiel Redux. React wurde von Facebook entwickelt und ermöglicht es, effiziente UI-Komponenten und komplexe Webanwendungen zu erstellen. Eine Besonderheit ist auch, dass React eine JSX-Syntax hat, also eine Syntaxerweiterung für HTML und JavaScript.

Pure Functions

Pure Functions geben für den Eingabeparameter immer dasselbe Ergebnis zurück und ändern keine der übergebenen Eingabeparameter. Sie liefern einen Wert zurück, der nur aus den Parametern besteht/entsteht.

Higher-Order-Functions - HOF

HOFs sind Funktionen, die andere Funktionen als Parameter nehmen und/oder Funktionen als Rückgabewerte liefert. Funktionen sind also für HOFs Werte und man kann sie dadurch ändern/manipulieren und zusammenführen/kombinieren.

bind:

Bind Funktionen sind Methoden in JavaScript, um Funktionen an Objekte zu „binden“.

Object Destructuring

Object Destructuring ermöglicht es, Werte oder Eigenschaften von Objekten (oder Arrays) zu lösen / in seperate Variablen zu „extrahieren“.

Object Literal Enhancement

Um entgegengesetzt zum Destrukturieren wie oben zu gehen, gibt es das Object Literal Enhancement, hierbei wird ein Objekt zusammengesetzt, da man Variablen direkt als Eigenschaftsnamen verwenden kann und Methoden zu Objekten hinzufügen kann.

Spread Operator

Der Spread Operator (…) wandelt alles iterierbare (also Strings, Maps und Sets) in einen Array um und kombiniert Objekte und Arrays.

Keine Übung

Da es die letzte Vorlesung war, haben wir nur noch die Übung Servlets von letzter Stunde besprochen.