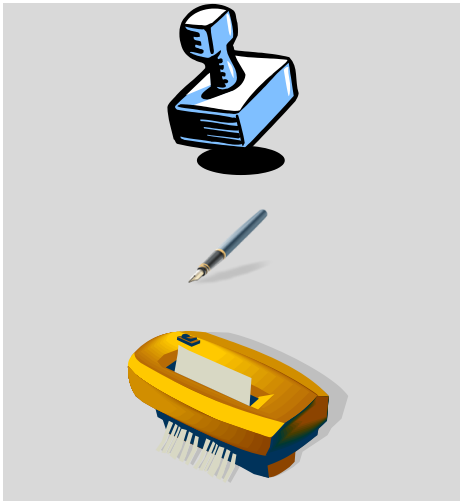
Informatik: Einführung Netzwerkprogrammierung, Exceptions

Ziele

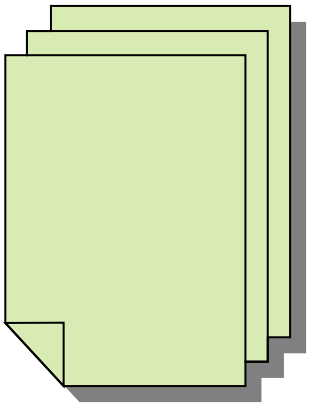
- Das Client-/Serverkonzept verstehen
- Einfache Server- und Client-Sockets zum gegenseitigen Datenaustausch programmieren können
- Auf Laufzeitfehler reagieren können
- Eigene Fehlerklassen schreiben können
- Laufzeitfehler werfen können

Server-Client-Modell

Server



Clients



Host

Server

SMTP E-Mail-Versand
Port 25

pop3 E-Mail-Empfang
Port 110

WWW Web-Server
Port 80

Telnet
Port 23

Clients

Server
Programm, das mit einem anderen Programm, dem Client kommuniziert, um ihm Zugang zu speziellen Diensten zu verschaffen.

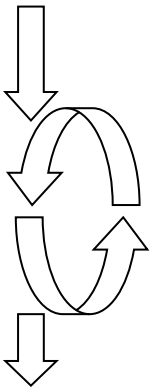
Client
Programm, welches mit Server Verbindung aufnimmt und Dienste von diesem anfordert.

Port
Nummer welche am Host laufende Server unterscheiden lässt.

Allgemein gültige Ports	Reservierbare Ports	frei verwendbar
0 bis 1023	1024 bis 49151	49152 bis 65535

Socket
streambasierte (≈ dateibasierte) Programmierschnittstelle zur Kommunikation zweier Rechner in einem **TCP/IP-Netz** (Transmission Control Protocol/Internet Protocol, Identifikation der Rechner erfolgt über IP-Adressen).

Der Server und seine Aufgaben



1. Erstellen eines Serversockets
2. Warten auf Verbindungsanfragen des Clients
3. Behandlung der Clientdaten und Zurückschicken des Ergebnisses
4. Schließen des Client
5. Schließen des Servers

Package `java.net` enthält die Klassen

`ServerSocket` zum Erstellen eines Servers.

`Socket` zum Erstellen der Clientanfrage am Server bzw. des Clients selbst.

Package `java.io` enthält die Klasse

`IOException` definiert ein Fehlerobjekt das zur Laufzeit generiert wird.

Konstruktor zum Erstellen des Servers ist folgendermaßen definiert:

```
public ServerSocket(int port) throws IOException
```

Exception-Signatur

Beim Erstellen des Servers möglicherweise auftretende Laufzeitfehler müssen behandelt werden:

```
ServerSocket server = null;
try {
   server = new ServerSocket(PORT);
} catch (IOException e) {
   System.out.println("Exc.klasse: " + e.getClass().getName());
   System.out.println("Fehlermeldung: " + e.getMessage());
} finally {
   try { server.close(); } catch (Exception e) { ; }
}
```

try leitet geschützten Block ein.

catch unterschiedliche Laufzeitfehler können abgefragt werden.

finally wird immer durchgeführt.

Die Klasse Exception

`public final Class getClass()`	Typ des Fehlerobjektes
`public String getMessage()`	Fehlermeldung
`public Exception(String msg)`	Konstruktor mit Fehlermeldung

Alle Exceptions – auch eigene – von `Exception` abgeleitet.

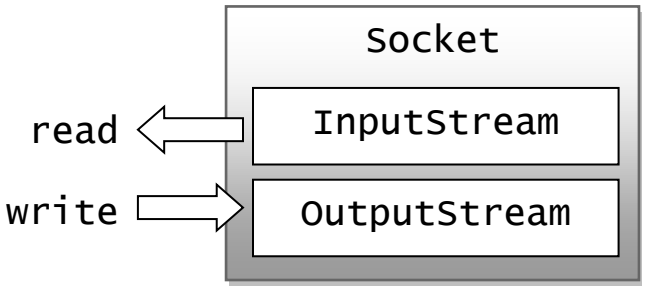
Verbindungsanfrage des Clients am Server

In Klasse `ServerSocket`
`public Socket accept() throws IOException`

wartet auf einen Verbindungswunsch des Clients. Blockiert sich solange bis sich Client am Server meldet.

```
…
server = new ServerSocket(PORT);
while (true) {
   Socket client = null;
   try {
      client = server.accept();
      behandleClient(client);
   } catch (IOException e) {
      behandleException(e);
   } finally {
      try { client.close();   } catch (Exception e) {  ;    }
   }
}
…
```

Behandlung des Clients am Server



In Klasse Socket
`public InputStream getInputStream() throws IOException`
`public OutputStream getOutputStream() throws IOException`

InputStream und OutputStream im Package java.io

In Klasse InputStream
`public int read() throws IOException`
liest ein Byte und schreibt es nach **int**. Ist blockierend und wartet bis ein Byte ankommt.

In Klasse OutputStream
`public void write(int b) throws IOException`
schreibt ein Byte.

```
private static void behandleClient(Socket client)
   throws IOException {
   InputStream in = client.getInputStream();
   OutputStream out = client.getOutputStream();
   int faktor1 = (byte)[1]in.read();
   int faktor2 = (byte)in.read();
   out.write(faktor1 * faktor2);
}
```

Methode behandleClient reagiert selbst nicht auf Exceptions sondern gibt diese unbehandelt an den Aufrufer weiter.

[1] Konvertierung bei neg. Werten unbedingt notwendig da sich Zweierkomplementdarstellung von int und byte unterscheidet

Zusammenfassung

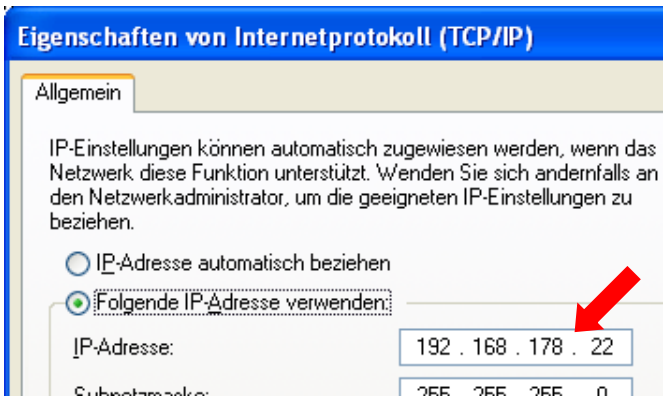
```
import java.io.*;
import java.net.*;
public class TestServer
{
  private static final int PORT = 65535;
  public static void main(String[] args) {
    ServerSocket server = null;
    try {
      server = new ServerSocket(PORT);
      while (true) {
        Socket client = null;
        try {
          client = server.accept();
          behandleClient(client);
        } catch (IOException e) {
          behandleException(e);
        } finally {
          try { client.close(); } catch (Exception e) { ; }
        }
      }
    } catch (IOException e) {
      behandleException(e);
    } finally {
      try { server.close(); } catch (Exception e) { ; }
    }
  }
  private static void behandleClient(Socket client)
    throws IOException {
    InputStream in = client.getInputStream();
    OutputStream out = client.getOutputStream();
    int faktor1 = (byte)in.read();
    int faktor2 = (byte)in.read();
    out.write(faktor1 * faktor2);
  }
  private static void behandleException(Exception e) {
    System.out.println("Exception Objekt: " +
      e.getClass().getName());
    System.out.println("Fehlermeldung: " + e.getMessage());
  }
}
```

Der Client

Konstruktor zum Erstellen des Clients ist folgendermaßen definiert:

```
public Socket(String host, int port)
    throws IOException, UnknownHostException
```

`String host = "localhost"; oder "192.168.178.22";`



Durch **Netzwerkumgebung**

oder

`ipconfig.exe`

```
import java.io.*;
import java.net.*;
public class TestClient
{
  private static final String HOST = "192.168.178.22";
  private static final int PORT = 65535;
  public static void main( String[] args ) {
    Socket client = null;
    try {
      client = new Socket(HOST, PORT);
      InputStream in = client.getInputStream();
      OutputStream out = client.getOutputStream();
      out.write(4);
      out.write(9);
      int ergebnis = (byte)in.read();
      System.out.println(ergebnis);
    } catch (UnknownHostException e) {
      behandleException(e);
    } catch (IOException e) {
      behandleException(e);
    } finally {
      try { client.close(); } catch ( Exception e ) { ; }
    }
  }
  private static void behandleException(Exception e) {
    …
  }
}
```

Eigene Exception-Klassen schreiben

Methode kuerzen soll einen ihr übergebenen String vom ersten und letzten Zeichen befreien. Mögliche Fehlerfälle:

- Der übergebene String ist null
- Der String ist kürzer als zwei Zeichen.

```
public static String kuerzen(String s)
   throws StringKuerzenException {
   String ret = null;
   if (s == null)
      throw new StringKuerzenException("Kein String");
   if (s.length() < 2)
      throw new StringKuerzenException("String zu kurz");
   ret = s.substring(1, s.length() - 1);
   return ret;
}
```

Exception-Signatur

Exception auslösen

```
public class StringKuerzenException extends Exception
{
  public StringKuerzenException(String msg) {
    super(msg);
  }
}
```

Es hat sich eingebürgert, dass eine wichtige Information über einen Laufzeitfehler im Typ des Exception-Objekts steckt.

Durchreichen von Exceptions

```
public static String dreifachKuerzen(String s)
  throws StringKuerzenException {
  String ret = kuerzen(kuerzen(kuerzen(s)));
  return ret;
}
```

```
try {
  System.out.println(kuerzen(null));
  System.out.println(kuerzen("x"));
  System.out.println(dreifachKuerzen("x"));
} catch (StringKuerzenException e) {
  System.out.println(e.getClass().getName());
  System.out.println(e.getMessage());
}
```

Die speziellen RuntimeExceptions

RuntimeException ist Vaterklasse aller Laufzeitfehler die behandelt werden *können*, aber nicht *müssen*. Sie tauchen deshalb auch nicht in der Exception-Signatur auf.

Zu diesen gehören u. a.

- ArithmeticException
  wenn beispielsweise ein int-Wert durch 0 dividiert wird.
- ClassCastException
  wenn zur Laufzeit die gewünschte Typkonvertierung nicht möglich ist.
- IllegalArgumentException
  wenn einer Methode ein ungültiges Argument übergeben wird.
- IndexOutOfBoundsException
- NullPointerException

```
public static void main(String[] args) {
   String s = null;
   int i = s.length();
   s = "hallo";
   s = s.substring(10);
}
```

NullPointerException

StringIndexOutOfBoundsException