



Java I/O, Serialisierung und Netzwerkprogrammierung

Grundlagen Java I/O

- Datenströme

- Erweiterung java.nio

Serialisierung

- Grundlagen Serialisierung

- Erweiterungen

Netzwerkprogrammierung

- Verbindungsorientierter Ansatz

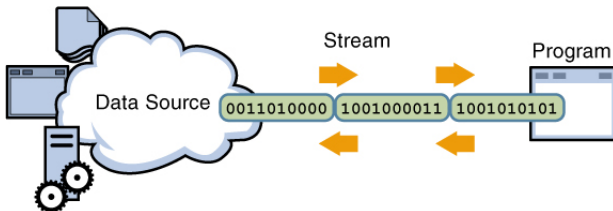
- Verbindungsloser Ansatz

- höhere Abstraktionsebene

Literatur

Datenströme

- ▶ Abstraktion von Ein- und Ausgabe von Daten
- ▶ byteorientierte Streams (`java.io.InputStream/OutputStream`)
- ▶ textorientierte Streams (`java.io.Reader/Writer`)
- ▶ Umwandlung über Adapterklassen (`java.io.InputStreamReader/OutputStreamWriter`)



Beispiele für Datenströme

- ▶ Print-, Buffered-, File-, Objekt-Streams, ...

```
1  BufferedReader in =
2      new BufferedReader(
3      new InputStreamReader(System.in));
4
5  BufferedWriter out =
6      new BufferedWriter(
7      new OutputStreamWriter(System.out));
8
9  String text = in.readLine();
10 out.write(text, 0, text.length());
11 in.close(); out.close();
```

RandomAccessFile

- ▶ wahlfreien Zugriffe innerhalb einer Datei
- ▶ Verhalten wie `ByteArray` mit Indexposition
- ▶ Lesen und Schreiben (`r`, `rw`, `..`) mit gleichem Objekt möglich
- ▶ Byte wie auch Character-Operationen
- ▶ `java.io.File` für Dateireferenz und Operationen auf dem Dateisystem (`mkdir`, `rename`)

Probleme Streams

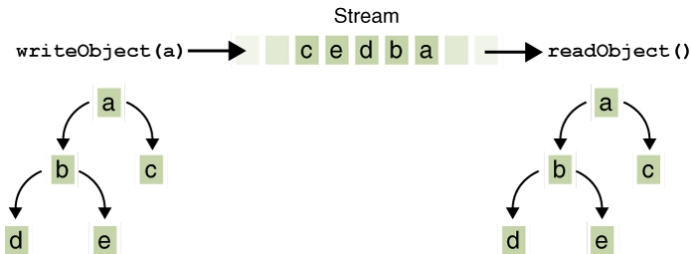
- ▶ unterschiedliche Ströme für Eingabe/Ausgabe
- ▶ strombasierte IO blockiert Ausführung
- ▶ `read()` springt erst zurück, wenn Daten gelesen (`write()` analog)
- ▶ Optimierung durch z.B. `BufferedReader`, Einsatz von Threads
- ▶ kaum Kontrolle über verwendete Buffer
- ▶ ineffizient durch Kopiervorgänge (BS/JVM)
- ▶ Sperrung von Dateien nicht in `java.io` vorhanden

Erweiterung java.nio

- ▶ **Buffer**
 - ▶ Container für Sequenzen von primitive Datentypen mit begrenzter Größe
 - ▶ direkte Zuweisung außerhalb des JVM-Heaps möglich
- ▶ **Channel**
 - ▶ repräsentieren offene Verbindungen zu Datenentitäten (Hardware, File, Socket)
 - ▶ Ein- und Ausgabe kann von einem Channel behandelt werden
 - ▶ Streams basieren auf Bytes, Channels auf Buffers
 - ▶ FileChannel, DatagramChannel, SocketChannel
 - ▶ Möglichkeit nicht-blockierender Operationen durch Selektoren

Serialisierung

- ▶ *Persistente* Zustände von Objekten
- ▶ Objektstruktur und Variablenbelegung erhalten
- ▶ Speicherung in Datei oder Transfer über Netzwerk
- ▶ Wiederherstellung von Objektzuständen



Was kann serialisiert werden?

- ▶ nicht-statische primitive Datentypen
- ▶ nicht-statische Objekte, Objekthierarchien
- ▶ mehrfach referenzierte Objekte nur einmal serialisiert
- ▶ nicht-statische geerbte Attribute und Basisklassen
- ▶ `transient` erlaubt expliziten Ausschluß
- ▶ `serialVersionUID` zur Kompatibilitätsicherung
- ▶ Warum implementiert `java.lang.Object` nicht `Serializable`?

Wie wird serialisiert?

- ▶ einfacher Fall: automatische Serialisierung
- ▶ betreffende Klasse implementiert `java.io.Serializable`

```
1 ObjectOutputStream stream =  
2     new ObjectOutputStream(  
3     new BufferedOutputStream(  
4     new FileOutputStream("serial.dat"))));  
5  
6 stream.writeObject(myObject);  
7 stream.close();
```

Eigene Mechanismen

- ▶ eigene Serialisierungsmethoden für aktuelles Objekt

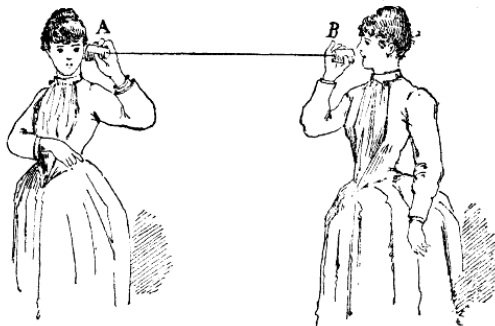
```
1 private void writeObject(ObjectOutputStream out)
2     throws IOException {
3
4     out.writeUTF(name);
5     out.writeLong(id);
6 }
7 private void readObject(ObjectInputStream in)
8     throws IOException, ClassNotFoundException {
9
10    name = in.readUTF();
11    id = in.readLong();
12 }
```

Eigene Mechanismen

- ▶ Interface `java.io.Externizable`
- ▶ Implementierung von `writeExternal/readExternal`-Methoden
- ▶ vollständige Kontrolle, kleinere Datenformate möglich
- ▶ nicht `ObjectStream` sondern Objekt selber behandelt
Serialisierung
- ▶ Beispiel: Objektdaten in PDF

Grundlagen Netzwerkprogrammierung

- ▶ `java.net` stellt IP-basierte Kommunikation
- ▶ Sockets als logische Endpunkte einer Verbindung
- ▶ einfaches Senden und Empfangen von Nachrichten
- ▶ verschiedene Sockettypen für Verbindungsarten



Sockets und ServerSockets

- ▶ sichere Übertragung, persistente Verbindung, TCP
- ▶ Sockets werden an Ziel-IP und -Port gebunden

```
1 Socket s = new Socket(HOSTorIP, toPort);
2     out = new PrintWriter(s.getOutputStream(), true)
3     in = new BufferedReader(
4         new InputStreamReader(s.getInputStream()));
5
6 out.println("hello world");
7 String response = in.readLine();
```

ServerSocket

- ▶ ServerSockets für Verbindungsannahme auf Quell-Port gebunden
- ▶ ServerSocket wartet auf Anfrage und erzeugt neuen Socket für Antwort

```
1 ServerSocket serverSocket = new ServerSocket(myPort);
2
3 while (true) {
4     Socket clientSocket = serverSocket.accept();
5     new ClientSocketThread(clientSocket).start();
6 }
```

DatagramSocket und DatagramPacket

- ▶ unsichere Übertragung, transistente Verbindung, UDP
- ▶ DatagramSocket wird auf Quell-Port gebunden
- ▶ DatagramPacket enthält eigentliche Ziel-IP und -Port

```
1 DatagramSocket socket = new DatagramSocket(myPort);
2     byte [] buf = requestString.getBytes();
3 DatagramPacket packet =
4 DatagramPacket(buf, buf.length(), toAddress, toPort);
5     socket.send(packet);
```

MulticastSocket

- ▶ Sender schickt Paket, das während des Transfers verteilt wird
- ▶ 224.0.0.0-239.255.255.255 für Mehrfachadressierung reserviert
- ▶ `MulticastSocket ms` wird nur auf *Port* gebunden
- ▶ über `ms.joinGroup(multicastGruppe)` beitreten
- ▶ über `ms.leave(multicastGruppe)` Empfang beenden
- ▶ Sender schickt *ein* `DatagramPacket` an Multicastgruppe

URL

- ▶ einfache Informationsabfrage (z.B. Webanfrage)
- ▶ http(s), ftp, file, etc. in sun.net.www.protocol.*

```
1 URL url = new URL("http://www.uni-ulm.de/home.html");
2 BufferedReader in =
3     new BufferedReader(
4     new InputStreamReader(url.openStream()));
5
6 String inputLine;
7
8 while ((inputLine = in.readLine()) != null)
9     System.out.println(inputLine);
10
11 in.close();
```

Links

- ▶ Java I/O
 - ▶ [Java Tutorials: Basic I/O](#)
 - ▶ [A Taste of Java's I/O Package: Streams, Files, and ...](#)
 - ▶ [Java New I/O](#)
- ▶ Serialisierung
 - ▶ [Java Object Serialization Specification](#)
 - ▶ [Serialisieren von Objekten \(Uni Köln\)](#)
- ▶ Netzwerkprogrammierung
 - ▶ [Netzwerke unter Java](#)
 - ▶ [Java Tutorials: Custom Networking](#)
 - ▶ [Felix von Leitner: Multicast](#)