

Vorlesung



University of Applied Sciences

Programmieren I und II

Unit 4

Einfache I/O Programmierung

Serialisierung von Objekten

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

1



University of Applied Sciences



**Prof. Dr. rer. nat.
Nane Kratzke**

*Praktische Informatik und
betriebliche Informationssysteme*

- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: kratzke@fh-luebeck.de



@NaneKratzke

Updates der Handouts auch über Twitter #prog_inf
und #prog_itd

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

2

**Handout zu den Vorlesungen
Programmieren I und II sowie Grundlagen und Vertiefung der Programmierung (Unit 4)**

Units

Unit 1 Einleitung und Grundbegriffe	Unit 2 Grundelemente imperativer Programme	Unit 3 Selbstdefinierbare Datentypen und Collections	Unit 4 Einfache I/O Programmierung
Unit 5 Rekursive Programmierung und rekursive Datenstrukturen	Unit 6 Einführung in die objektorientierte Programmierung und UML	Unit 7 Konzepte objektorientierter Programmiersprachen	Unit 8 Testen (objektorientierter) Programme
Unit 9 Generische Datentypen	Unit 10 Objektorientierter Entwurf und objektorientierte Designprinzipien	Unit 11 Graphical User Interfaces	Unit 12 Multithread Programmierung
Unit 13 Einführung in die Funktionale Programmierung			

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke | 3
Praktische Informatik und betriebliche Informationssysteme

Abgedeckte Ziele dieser UNIT

Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmusbegriffs, Implementieren einfacher Algorithmen	Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)	Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Multithread Programmierung sowie funktionale Programmierung
Am Beispiel der Sprache JAVA			

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke | 4
Praktische Informatik und betriebliche Informationssysteme

Zum Nachlesen ...



University of Applied Sciences

Kapitel 19

Ein- und Ausgabe über Streams

Abschnitt 19.1

Grundsätzliches zu Streams in Java

Abschnitt 19.2

Dateien und Verzeichnisse

Abschnitt 19.3

Ein- und Ausgabe über Character-Streams

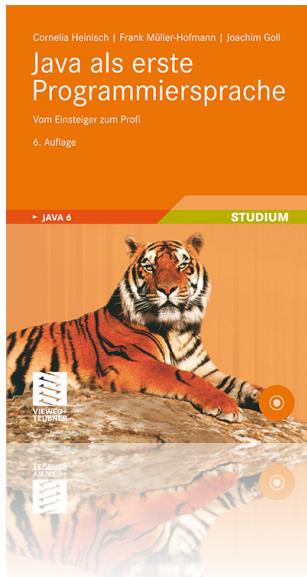
Abschnitt 19.4.2

Die Serialisierung und Deserialisierung von Objekten

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

5

Noch mehr zum Nachlesen ...



University of Applied Sciences

Kapitel 16

Ein-/Ausgabe und Streams

Abschnitt 16.2 Klassifizierung von Str.

Abschnitt 16.3 Das Stream-Konzept

Abschnitt 16.4 ByteStream-Klassen

Abschnitt 16.5 ByteStream-Klassen

Abschnitt 16.7 Ein- und Ausgabe von Objekten

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

6

I/O erscheint mit Java recht komplex



University of Applied Sciences

- Das Lesen und Schreiben von Daten in Dateien in Java ist nicht ganz einfach.
- I/O Programming wird in Java mittels sogenannter Datenströme (Streams) realisiert.
- Lese- und Schreiboperationen auf Dateien lassen sich in anderen Programmiersprachen häufig wesentlich einfacher realisieren.
- In Java muss man hierzu erst einmal einige Konzepte verstehen.
- Dafür wird man dann aber auch mit einiger Flexibilität belohnt.

... ist es aber nicht, wenn man über die Zusammenhänge weiß!



University of Applied Sciences

- Zugriff auf das **Dateisystem** erfolgt mittels **File Objekten**
- Definition von **Datenquellen und –senken** erfolgt mittels **Input- und OutputStreams**
 - Datenquellen und –senken sind dabei nicht auf Dateien beschränkt
 - Weitere Quellen und Senken können bspw. beliebige URLs oder die Systemausgabe auf einer Konsole sein.
 - Durch objektorientierte Erweiterungen der **InputStream** und **OutputStream** Klassen lassen sich beliebige Quellen und Senken erschließen
- **Lese- und Schreiboperationen** auf diesen Streams erfolgen zeichenweise mittels **Reader** und **Writer** Objekten
- Lese- und Schreiboperationen lassen sich mittels **BufferedReader** und **BufferedWriter** Objekten **puffern** (um bspw. eine zeilenweise Verarbeitung von Textdateien zu ermöglichen)

Themen dieser Unit



University of Applied Sciences



Dateien und Verzeichnisse

- File Objekt
- Dateien/Verzeichnisse erzeugen, löschen, umbenennen
- Zustände von Dateien/Verzeichnissen abfragen

I/O Streams

- Das Stream Konzept
- Aus Quellen lesen
- In Senken schreiben

Serialisierung

- Objektzustände speichern
- Objektzustände wiederherstellen

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

9

Auf das Dateisystem aus Java zugreifen



University of Applied Sciences

Java bietet, um auf Elemente des Dateisystems zugreifen zu können, Objekte der Klasse **File** an (sowohl für Dateien als auch für Verzeichnisse).

File

Erzeugen/löschen

- boolean createNewFile()
- boolean mkdir()
- boolean delete()
- boolean renameTo(File f)

File

Rechte und Typ

- boolean canRead()
- boolean canWrite()
- boolean isDirectory()
- boolean isFile()
- long length()
- ...

File

Abfragen

- boolean exists()
- String getName()
- String getAbsolutePath()
- File[] listFiles()
- ...

```
import java.io.File;  
File f = new File("/Users/Nane/Desktop");
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

10

Beispiel: Typische Nutzung von File Objekten



University of Applied Sciences

```
File f = new File("/Users/None");

// Anlegen und loeschen einer neuen Datei
File newFile = new File(f.getAbsolutePath() + File.separator + "newFile.test");
newFile.createNewFile();
newFile.delete();

// Anlegen und loeschen eines neuen Verzeichnisses
File newDir = new File(f.getAbsolutePath() + File.separator + "directory");
newDir.mkdir();
newDir.delete();

// Auflisten aller Dateien in einem Verzeichnis
for (File file : f.listFiles()) {
    if (file.isFile()) {
        System.out.println(file.getAbsolutePath() + " Size: " + file.length() + " bytes");
    }
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

11

Themen dieser Unit



University of Applied Sciences



Dateien und Verzeichnisse

- File Objekt
- Dateien/Verzeichnisse erzeugen, löschen, umbenennen
- Zustände von Dateien/Verzeichnissen abfragen

I/O Streams

- Das Stream Konzept
- Aus Quellen lesen
- In Senken schreiben

Serialisierung

- Objektzustände speichern
- Objektzustände wiederherstellen

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

12

Worum geht es nun?



University of Applied Sciences


Das Streamkonzept

**Beispiel:
Textdatei
zeilenweise
auslesen**

**Beispiel:
Textdatei
zeilenweise
schreiben**

**Flexibel
kombinieren um
Daten zu lesen
und zu speichern**

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

13

Das Stream Konzept



University of Applied Sciences

Mit **Datenströmen** (englisch: data streams) bezeichnet man kontinuierliche Abfolgen von Datensätzen, deren Ende nicht im Voraus abzusehen ist.

Die einzelnen Datensätze sind dabei üblicherweise von beliebigem, aber festem Typ. Die Menge der Datensätze pro Zeiteinheit (Datenrate) kann variieren. Im Gegensatz zu anderen Datenquellen werden Datenströme daher nicht als ganzes, sondern fortlaufend verarbeitet.

Insbesondere ist im Gegensatz zu Datenstrukturen mit wahlfreiem Zugriff (wie z. B. Arrays) **nur ein sequentieller Zugriff** auf die einzelnen Datensätze möglich.

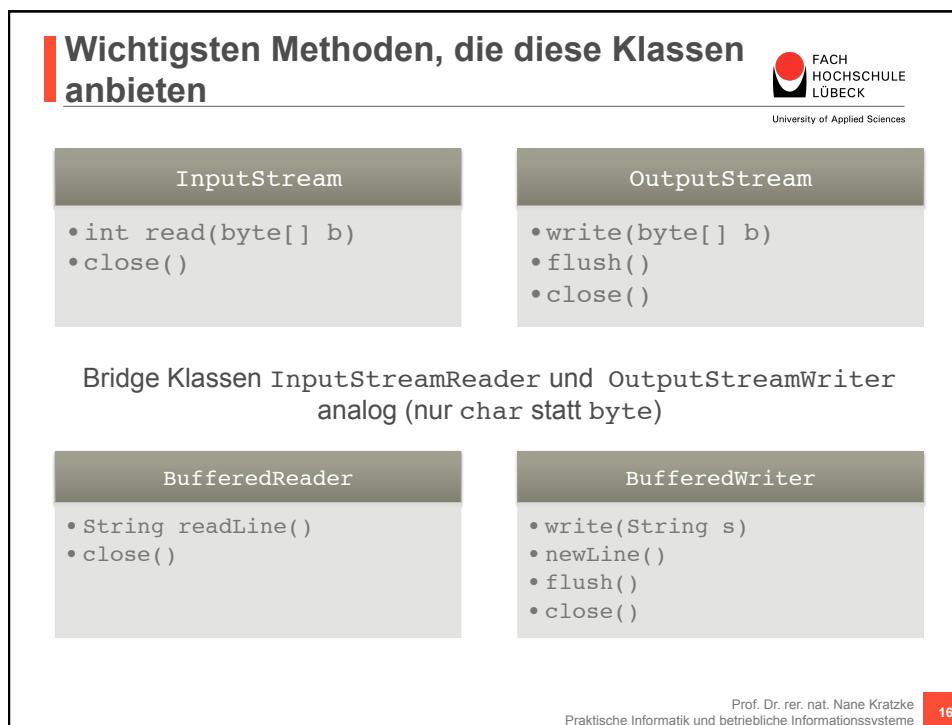
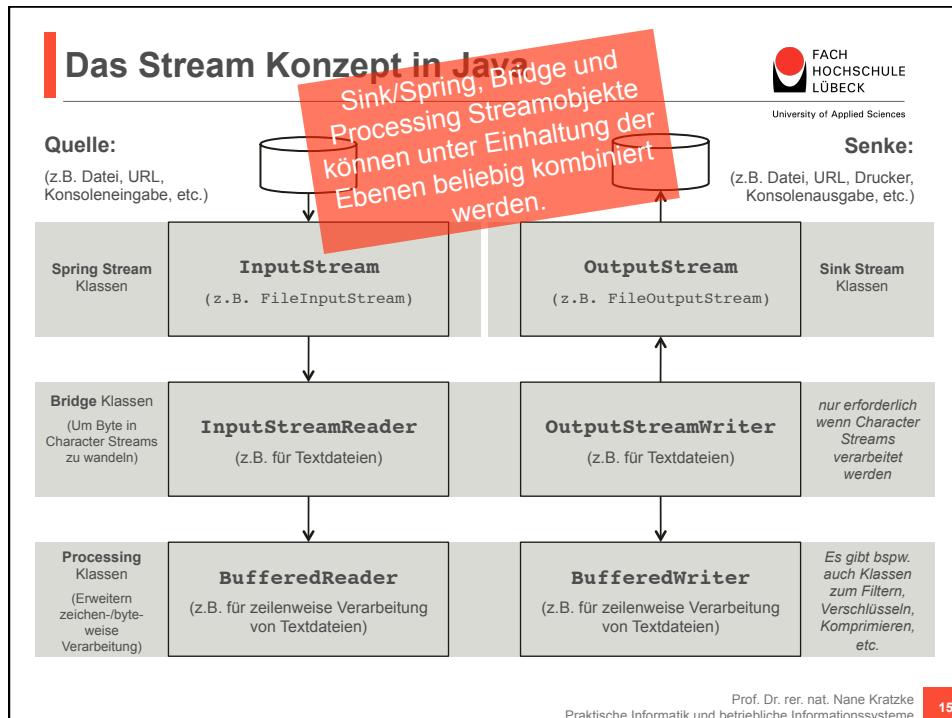
Datenströme werden häufig zur Interprozesskommunikation verwendet, sowohl zur Übertragung von Daten über Netzwerke als auch zur Kommunikation zwischen Prozessen auf einem Rechner.

In Java werden mit diesem Konzept auch lokale Dateien verarbeitet.

Quelle: Wikipedia DE (Datenstrom)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

14



Worum geht es nun?



University of Applied Sciences

Das
Streamkonzept

Beispiel:
Textdatei
zeilenweise
auslesen

Beispiel:
Textdatei
zeilenweise
schreiben

Flexibel
kombinieren um
Daten zu lesen
und zu speichern

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

17

Beispiel: Datei zeilenweise einlesen



University of Applied Sciences

```
/**  
 * Öffnet eine Textdatei und gibt diese auf der Konsole zeilenweise aus.  
 * Z.B. durch den Aufruf printFile("./Desktop/test.txt");  
 * @param path Zu öffnende Datei  
 */  
  
public static void printFile(String path) throws Exception {  
    InputStream is = new FileInputStream(path); // Spring Stream  
    InputStreamReader bridge = new InputStreamReader(is); // Bridge Stream  
    BufferedReader reader = new BufferedReader(bridge); // Processing Stream  
  
    // Stream mittels Processing Stream auslesen  
    String line = "";  
    while ((line = reader.readLine()) != null) { System.out.println(line); }  
  
    // Schließen der Streams nicht vergessen  
    reader.close(); bridge.close(); is.close();  
}  
  
import java.io.*;
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

18

Worum geht es nun?



University of Applied Sciences

Das
Streamkonzept

Beispiel:
Textdatei
zeilenweise
auslesen

Beispiel:
Textdatei
zeilenweise
schreiben

Flexibel
kombinieren um
Daten zu lesen
und zu speichern

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

19

Beispiel: Datei zeilenweise schreiben



University of Applied Sciences

```
/*
 * Schreibt einhundert mal "Hallo mein Name ist Hase {x}" in eine Textdatei.
 * Z.B. durch den Aufruf writeFile("./Desktop/test.txt");
 * @param path Pfad der Datei in die geschrieben werden soll
 */
public static void writeFile(String path) throws Exception {
    OutputStream os = new FileOutputStream(path); // Sink Stream
    OutputStreamWriter bridge = new OutputStreamWriter(os); // Bridge Stream
    BufferedWriter writer = new BufferedWriter(bridge); // Writer Stream

    // Stream mittels Processing Stream befüllen
    for (int i = 1; i <= 100; i++) {
        writer.write("Hallo mein Name ist Hase " + i + "\n");
    }

    // Schließen der Streams nicht vergessen
    writer.close(); bridge.close(); os.close();
}
```

import java.io.*;

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

20

Worum geht es nun?



University of Applied Sciences

Das
Streamkonzept

Beispiel:
Textdatei
zeilenweise
auslesen

Beispiel:
Textdatei
zeilenweise
schreiben

Flexibel
kombinieren um
Daten zu lesen
und zu speichern

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

21

Kann man damit auch aus dem Internet lesen?



University of Applied Sciences

Streams sind Datenströme, die aus beliebigen Datensunken und –quellen entstammen können. Es müssen also nicht immer nur Dateien seien, aus denen man Daten einliest oder hineinschreibt.

Sie sollen jetzt den HTML Code der Website der Fachhochschule Lübeck (<http://www.fh-luebeck.de>) auslesen und in einer Datei speichern.

Tipp: Einen Sink Stream von einer über einer URL adresierbare Remote Quelle können Sie wie folgt erzeugen.

```
import java.net.URL;  
  
URL url = new URL("http://www.fh-luebeck.de");  
InputStream i = url.openStream();
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

22

Beispiel: HTML der FH-Lübeck speichern



University of Applied Sciences

```
/**  
 * Liest den Inhalt von einer Url (bspw. Webseite) und kopiert diesen in eine Datei.  
 * Z.B. durch Aufruf von  
 * copyFromUrlToFile("http://www.fh-luebeck.de", "/Users/Nane/Desktop/fhl.html");  
 */  
  
public static void copyFromUrlToFile(String src, String to) throws Exception {  
    URL url = new URL(src);  
  
    // Diesmal bauen wir die Stream Kaskaden jeweils in einer Zeile auf  
    BufferedReader input = new BufferedReader(new InputStreamReader(url.openStream()));  
    BufferedWriter output = new BufferedWriter(new OutputStreamWriter(  
        new FileOutputStream(to)  
    ));  
  
    // Input Stream in Output Stream kopieren  
    String line = "";  
    while ((line = input.readLine()) != null) { output.write(line + "\n"); }  
  
    // Schließen der Streams nicht vergessen  
    input.close(); output.close();  
}  
  
import java.io.*;  
import java.net.URL;
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

23

Beispiel: HTML der FH-Lübeck speichern



University of Applied Sciences

```
/**  
 * Liest den Inhalt von einer Url (bspw. Webseite) und kopiert diesen in eine Datei.  
 * Z.B. durch Aufruf von  
 * copyFromUrlToFile("http://www.fh-luebeck.de", "/Users/Nane/Desktop/fhl.html");  
 */  
  
public static void copyFromUrlToFile(String src, String to) throws Exception {  
    URL url = new URL(src);  
  
    // Diesmal bauen wir die Stream Kaskaden jeweils in einer Zeile auf  
    BufferedReader input = new BufferedReader(new InputStreamReader(url.openStream()));  
    BufferedWriter output = new BufferedWriter(new OutputStreamWriter(  
        System.out  
    ));  
  
    // Input Stream in Output Stream kopieren  
    String line = "";  
    while ((line = input.readLine()) != null) {  
        System.out.println(line);  
    }  
  
    // Schließen der Streams nicht vergessen  
    input.close(); output.close();  
}  
  
import java.io.*;  
import java.net.URL;
```

Was müssen wir ändern,
wenn wir die Quelle auf der
Konsole ausgeben wollen?

Hat denselben Effekt wie ...

System.out.println(line);

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

24

Fazit:



- Durch die Trennung von
 - Sink und Spring Streams (Quelle und Ziel)
 - Bridge Streams (Zeichen oder Bytes)
 - und Processing Streams (Inhalt)
- hat man sehr flexible Möglichkeiten um Datenströme definieren zu können.
- Das tatsächliche Speicher-, Transfer- oder Ausgabemedium kann erst sehr spät festgelegt und auch jederzeit geändert werden, ohne den Rest der Logik anpassen zu müssen ☺
- Dafür ist das einfache Öffnen und Auslesen einer Datei leider etwas komplizierter als in anderen Sprachen üblich ☹

Themen dieser Unit



Serialisierung

- | Dateien und Verzeichnisse | I/O Streams | Serialisierung |
|--|--|--|
| <ul style="list-style-type: none">• File Objekt• Dateien/ Verzeichnisse erzeugen, löschen, umbenennen• Zustände von Dateien/ Verzeichnissen abfragen | <ul style="list-style-type: none">• Das Stream Konzept• Aus Quellen lesen• In Senken schreiben | <ul style="list-style-type: none">• Objektzustände speichern• Objektzustände wiederherstellen |

(De-)serialisierung



University of Applied Sciences

Serialisierung bezeichnet eine Abbildung von **strukturierten Daten** auf eine sequenzielle Darstellungsform. Serialisierung wird hauptsächlich für die Speicherung von Objektzuständen in **Dateien** und für die Übertragung von Objektzuständen über das **Netzwerk** verwendet.

Hierzu wird der komplette **Zustand des Objektes**, inklusive aller referenzierten Objekte, in einen speicherbaren **Datenstrom** umgewandelt.

Die Umkehrung der Serialisierung, also die Umwandlung eines Datenstroms in Objekte, wird als Deserialisierung bezeichnet.

Quelle: Wikipedia DE (Serialisierung)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

27

(De-)serialisierung in Java



University of Applied Sciences

In Java ist es sehr einfach Objekte zu serialisieren. Hierdurch können Objektzustände nicht nur im Hauptspeicher (also zur Laufzeit eines Programms) existieren, sondern auch persistent in Streams geschrieben, bzw. aus diesen gelesen werden.

Hierzu existieren die beiden Bytestrom basierten Processing Streams

`ObjectInputStream`

`ObjectOutputStream`

Sie lassen sich gem. der gezeigten Stream Systematik einsetzen, um Objektzustände zu speichern und wiederherstellen zu können.

Alle Objekte von Klassen, die die Schnittstelle `Serializable` implementiert haben, lassen sich auf diese Weise persistent speichern.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

28

Beispiel einer serialisierbaren Klasse



University of Applied Sciences

```
public class Person implements Serializable {  
  
    // Diese UID wird benötigt, um Objekte wiederherzustellen.  
    // Sie sollte geändert werden, wenn sich Datenfelder einer Klasse ändern.  
    private static final long serialVersionUID = -9006175784695176582L;  
  
    public String vorname = "";  
    public String nachname = "";  
    public int alter;  
  
    public Person(String vn, String nn, int a) {  
        this.vorname = vn;  
        this.nachname = nn;  
        this.alter = a;  
    }  
  
    public String toString() {  
        return vorname + " " + nachname + " (" + alter + ")";  
    }  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

29

Ausnehmen von Datenfelder aus dem Serialisierungsprozess



University of Applied Sciences

Ggf. soll bei Objekten, nicht der komplette Objektzustand serialisiert werden (z.B. weil Passwortinformationen nicht im Klartext in Dateien gespeichert werden sollen).

Dann können diese Datenfelder als **transient** markiert werden, um sie aus dem Serialisierungsprozess auszunehmen.

Bei der Deserialisierung werden diese Datenfelder mit der Default-Initialisierung belegt.

```
public class Person implements Serializable {  
  
    public String vorname = "";  
    public String nachname = "";  
    public transient String password = "";  
    public int alter;  
  
    [...]  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

30

Serialisieren von Objekten



University of Applied Sciences

Folgender Code erzeugt 100 zufällige Personen zufälligen Alters und speichert diese in einer Datei.

```
Random rand = new Random();  
  
String[] vornamen = { "Maren", "Max", "Moritz", "Magda", "Momme" };  
String[] nachnamen = { "Mustermann", "Musterfrau", "Mommsen", "Meier", "Müller" };  
  
ObjectOutputStream out = new ObjectOutputStream(  
    new FileOutputStream("/Users/Nane/Desktop/personen.ser")  
);  
  
for (int i = 1; i <= 100; i++) {  
    String vn = vornamen[rand.nextInt(vornamen.length)];  
    String nn = nachnamen[rand.nextInt(nachnamen.length)];  
    Person p = new Person(vn, nn, rand.nextInt(100));  
    out.writeObject(p);  
}  
  
out.close();
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

31

Deserialisieren von Objekten



University of Applied Sciences

Folgender Code liest in einer Datei gespeicherte Personen ein und gibt diese aus.

```
ObjectInputStream in = new ObjectInputStream(  
    new FileInputStream("/Users/Nane/Desktop/personen.ser")  
);  
  
Person p;  
while ((p = (Person)in.readObject()) != null) {  
    System.out.println(p);  
}  
in.close();
```

De-/Serialisierung funktioniert, dabei nicht nur auf Einzelobjektbene, sondern auch komplexe Objektabhängigkeiten lassen sich serialisieren. Beim Serialisieren eines Objekts werden dabei Referenzen auf Objekte rekursiv durchlaufen. Beim deserialisieren entsprechend anders herum.

Zum Beispiel sind Collection Klassen in Java serializable.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

32

Serialisieren von Collections



University of Applied Sciences

Folgender Code erzeugt 100 zufällige Personen zufälligen Alters und speichert diese in einer Datei als Collection.

```
Random rand = new Random();  
  
String[] vornamen = { "Maren", "Max", "Moritz", "Magda", "Momme" };  
String[] nachnamen = { "Mustermann", "Musterfrau", "Mommsen", "Meier", "Müller" };  
  
ObjectOutputStream out = new ObjectOutputStream(  
    new FileOutputStream("/Users/Nane/Desktop/personen.ser")  
);  
  
List<Person> personen = new LinkedList<Person>();  
  
for (int i = 1; i <= 100; i++) {  
    String vn = vornamen[rand.nextInt(vornamen.length)];  
    String nn = nachnamen[rand.nextInt(nachnamen.length)];  
    personen.add(new Person(vn, nn, rand.nextInt(100)));  
}  
  
out.writeObject(personen);  
out.close();
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

33

Deserialisieren von Collections



University of Applied Sciences

Folgender Code liest diese mittels einer Collection in einer Datei gespeicherten Personen ein und gibt diese aus.

```
ObjectInputStream in = new ObjectInputStream(  
    new FileInputStream("/Users/Nane/Desktop/personen.ser")  
);  
  
List<Person> personen = (List<Person>)in.readObject();  
in.close();  
  
for (Person p : personen) { System.out.println(p); }
```

Es lassen sich somit auch komplexe Objektstrukturen sehr einfach einer Serialisierung unterwerfen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

34

Zusammenfassung

- **File Objekt**
 - Zugriff auf das Dateisystem
 - Erzeugen und Zugreifen auf Dateien und Verzeichnisse
- **Streams**
 - Datenströme sequentiell erzeugen/verarbeiten
 - Sink und Spring Streams
 - Bridge Streams
 - Processing Streams
 - Flexibel kombinieren
- **Objekte Serialisieren**
 - Schnittstelle Serializable
 - Serialisieren (Objektzustand/Collections speichern)
 - Deserialisierungen (Objektzustand/Collections wiederherstellen)

