

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Übungen zum Java-Praxiskurs für Fortgeschrittene

Matthias Book
Malte Hülдер

Lehrstuhl für Angewandte Telematik / e-Business
WS 2005/2006

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Wiederholung: Interfaces in Java

```
public interface Verlaengerbar {
    ...
    int VERLAENGERUNGSFRIST = 14;
    int verlaengern (...);
    ...
}

public class Buch extends Medium
    implements Verlaengerbar, Vormerkbar {
    ...
    public int verlaengern (...) {...}
    public int vormerken (...) {...}
    ...
}
```

- Klassen können nur eine Oberklasse beerben, aber mehrere Interfaces implementieren, deren Methoden sie alle überschreiben müssen

Der Java-Praxiskurs II 2

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Wdh.: Modellierung des Dateisystems

- mit Objekten der Klasse `java.io.File` zum
 - Anlegen, Öffnen, Schließen und Löschen von Dateien
 - Anlegen und Löschen von Verzeichnissen
 - Abfragen/Setzen von Informationen über Dateien und Verzeichnisse
 - Blättern in der Verzeichnishierarchie
- nicht zum Lesen und Schreiben in Dateien!
 - wird mit Streams realisiert
- Beispiel: Länge ausgeben, wenn "java" existiert und eine Datei (kein Verzeichnis) ist


```
File datei = new File("java");
if (datei.exists() && datei.isFile()) {
    System.out.println(datei.length() + " Bytes");
}
```

Der Java-Praxiskurs II 3

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Ü1.1.1 Directory-Lister

- Schreiben Sie eine Klasse `DirectoryLister`, die eine Methode `void doList(String directory)` enthält, die alle Dateinamen aus einem Verzeichnis am Bildschirm auflistet.

```
import java.io.*;

public class DirectoryLister {
    public void doList(String directory) {
        File dir = new File(directory);
        String list[] = dir.list();
        for (int i = 0; i < list.length; i++)
            System.out.println(list[i]);
    }

    public static void main(String args[]) {
        DirectoryLister dl = new DirectoryLister();
        dl.doList(".");
    }
}
```

Der Java-Praxiskurs II 4

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Ü1.1.2 Directory-Lister mit Interface

- Das Interface `ListerCallback` soll ermöglichen, die Verzeichniseinträge beliebig zu verarbeiten. Zur Demonstration sollen alternativ mit der Klasse `CallbackTest` in `DirectoryLister` Dateinamen ausgegeben werden.

```
import java.io.*;

interface ListerCallback {
    public void perform(File theFile);
}

import java.io.*;

public class CallbackTest implements ListerCallback {
    public void perform(File theFile) {
        System.out.println(theFile.getName());
    }
}
```

Der Java-Praxiskurs II 5

UNIVERSITÄT LEIPZIG
LPZ E-BUSINESS
applied telematics

Ü1.1.2 Directory-Lister mit Interface

- Modifizieren Sie die Klasse `DirectoryLister` so, dass ihrem Konstruktor ein Objekt, das `ListerCallback` implementiert, übergeben werden kann.
- Das übergebene Objekt soll in seiner `perform`-Methode die Implementierung der Aktion enthalten, die für jeden Verzeichniseintrag auszuführen ist. Ändern Sie dazu den Rest der Klasse so, dass für jede Datei einmal `ListerCallback.perform(...)` aufgerufen wird.
- Ändern Sie Ihr Programm so, dass beim Erzeugen der Instanz von `DirectoryLister` eine Instanz von `CallbackTest` übergeben wird.

Der Java-Praxiskurs II 6

Ü1.1.2 Directory-Lister mit Interface

```

import java.io.*;

public class DirectoryLister {
    private ListerCallback lcb;
    public DirectoryLister(ListerCallback lcb) {
        this.lcb = lcb;
    }
    public void doList(String directory) {
        File dir = new File(directory);
        String list[] = dir.list();
        for (int i = 0; i < list.length; i++) {
            File f = new File(list[i]);
            lcb.perform(f);
        }
    }
    public static void main(String[] args) {
        DirectoryLister dl = new DirectoryLister(new CallbackTest());
        dl.doList(".");
    }
}
    
```

Der Java-Praxiskurs II 7

Wiederholung: Streams

- Unterklassen der abstrakten Klassen
 - `InputStream`, `OutputStream` (für Byte-Streams)
 - `Reader`, `Writer` (für Unicode-Zeichenstreams)
- zur Erfüllung bestimmter Aufgaben
 - Lesen/Schreiben aus/in Datenquellen/-senken
 - Verwendung anderer Streams als Datenquellen/-senken
 - Vor-/Nachverarbeitung der Daten im Stream
 - Ein-/Ausgabe verschiedener Datentypen
- i.d.R. müssen mehrere (aber nicht unbedingt alle) dieser Aufgaben gelöst werden
- Verkettung von Streams für verschiedene Funktionen

Der Java-Praxiskurs II 8

Wdh.: Verwendung von Medienstreams

- Beispiel: `FileOutputSteam`
- Konstruktor öffnet die Datei und
 - überschreibt ggf. zuvor bestehende Datei
 - `FileOutputSteam(File file)`
 - `FileOutputSteam(String name)`
 - hängt bei `append==true` an Datei an, überschreibt sonst ggf.
 - `FileOutputSteam(File file, boolean append)`
 - `FileOutputSteam(String name, boolean append)`
 - (alle können `FileNotFoundException` werfen)
- Methoden i.W. wie abstrakter `OutputStream`
 - `write` (Byte oder Byte-Array schreiben), `flush` (Puffer leeren), etc.
 - arbeiten auf der vom Stream gekapselten Datei

Der Java-Praxiskurs II 9

Wdh.: Verwendung von Datenstreams

- Beispiel: `DataOutputSteam`
- Konstruktor nimmt zugrunde liegenden Stream entgegen
 - `DataOutputSteam(OutputStream out)`
- Methoden wie abstrakter `OutputStream`, *zusätzlich auch*:
 - `void writeBoolean(boolean v)`
 - `void writeShort(int v), void writeByte(int v)`
 - `void writeInt(int v), void writeLong(long v)`
 - `void writeFloat(float v), void writeDouble(double v)`
 - `void writeChar(int v), void writeUTF(String str)`
 - `void writeBytes(String s), void writeChars(String s)`
 - schreiben primitive Typen in passender Byte-Darstellung in Stream

Der Java-Praxiskurs II 10

Wdh.: Schreiben in eine Textdatei

`FileWriter` kann Zeichen in eine Datei schreiben
`PrintWriter` kann Daten als Strings in Stream schreiben

- über mit div. Datentypen überladene `print/println`-Methoden
- Umwandlung von Referenzdatentypen in Strings mit `toString()`

```

Buch javaBuch = new Buch(...); float preis = 41.99;
try {
    FileWriter fw =
        new FileWriter("ausgabe.txt");
    PrintWriter pw =
        new PrintWriter(fw);
    pw.println(javaBuch);
    pw.println(preis);
}
catch (IOException e) {...}
    
```

Der Java-Praxiskurs II 11

Ü1.2.1 Directory-Lister für Java-Dateien

- Ändern Sie die Klasse `DirectoryLister` aus der ersten Aufgabe so, dass der `perform`-Methode des `ListerCallback` implementierenden Objekts nur Dateien mit der Endung `.java` übergeben werden.

```

public class DirectoryLister { ...
    public void doList(String directory) {
        File dir = new File(directory);
        String list[] = dir.list(
            new FilenameFilter() {
                public boolean accept(File dir, String name) {
                    return name.endsWith(".java");
                }
            });
        for (int i = 0; i < list.length; i++) {
            File f = new File(list[i]);
            lcb.perform(f);
        }
    }
}
    
```

anonyme innere Klasse

Der Java-Praxiskurs II 12

Ü1.2.1 Directory-Lister für Java-Dateien

- äquivalente Lösung mit expliziter Deklaration einer Klasse `JavaFilter`, die das Interface `FilenameFilter` implementiert:

```
public class JavaFilter implements FilenameFilter {
    public boolean accept(File dir, String name) {
        return name.endsWith(".java");
    }
}
```

```
public class DirectoryLister { ...
    public void doList(String directory) {
        File dir = new File(directory);
        String list[] = dir.list(new JavaFilter());
        for (int i = 0; i < list.length; i++) {
            File f = new File(list[i]);
            lcb.perform(f);
        }
    }
    ...
}
```

Der Java-Praxiskurs II

13

Ü1.2.2 Lines of Code

- Schreiben Sie eine Klasse `InspectCode`, die das Interface `ListerCallback` aus der ersten Aufgabe implementiert, und bei Aufruf ihrer `perform`-Methode die Anzahl der Zeilen in der übergebenen Datei am Bildschirm ausgibt.

```
public class InspectCode implements ListerCallback {
    public void perform(File theFile) {
        String line;
        int lines = 0;
        try {
            BufferedReader bfr = new BufferedReader(
                new FileReader(theFile));
            while ((line = bfr.readLine()) != null) { lines++; }
            bfr.close();

            System.out.println("File " + theFile.getName() +
                " has " + lines + " Lines.");
        } catch (IOException ioe) { ioe.printStackTrace(); }
    }
}
```

Der Java-Praxiskurs II

14

Ü1.2.2 Lines of Code

- Verwenden Sie die Klasse `InspectCode` statt `CallbackTest` im `DirectoryLister`.

```
public class DirectoryLister {
    ...
    public static void main(String[] args) {
        DirectoryLister dl = new DirectoryLister(
            new InspectCode());

        dl.doList(".");
    }
}
```

Der Java-Praxiskurs II

15

Ü1.2.3 Lines of Code ohne Kommentare

- Modifizieren Sie die Klasse `InspectCode` so, dass bei der Zählung der Zeilen Kommentare in Java-Programmen nicht berücksichtigt werden.

```
public class InspectCode implements ListerCallback {
    public void perform(File theFile) {
        String line;
        int lines = 0;
        int nesting = 0;
        try {
            BufferedReader bfr = new BufferedReader(
                new FileReader(theFile));
            while ((line = bfr.readLine()) != null) {
                ... // siehe nächste Folie
            }
            bfr.close();
            System.out.println("File " + theFile.getName() + " has " +
                lines + " Lines.");
        } catch (IOException ioe) { ioe.printStackTrace(); }
    }
}
```

Der Java-Praxiskurs II

16

Ü1.2.3 Lines of Code ohne Kommentare

```
...
while ((line = bfr.readLine()) != null) {
    if (nesting != 0) { // wir lesen einen Kommentarabschnitt
        int i = line.indexOf("*/");
        if (i != -1) {
            nesting = 0;
            line = line.substring(i + 2);
        } else line = "";
    } else { // wir lesen einen Quelltextabschnitt
        int i = line.indexOf("/*");
        if (i != -1) {
            nesting = 1;
            line = line.substring(0, i);
        }
        i = line.indexOf("*/");
        if (i != -1) line = line.substring(0, i);
    }
    line = line.trim();
    if (line.length() != 0) lines++;
}
...
}
```

Der Java-Praxiskurs II

17

Ü1.2.4 Lines of Code rekursiv

- Modifizieren Sie das gesamte Programm so, dass ausgehend von einem Startverzeichnis alle Unterverzeichnisse rekursiv durchlaufen werden und die Gesamtsumme der Zeilenzahlen aller Java-Dateien darin ermittelt wird.
- Um die Aufsummierung zu ermöglichen, muss im Interface `ListerCallback` die Zeilenanzahl des betrachteten Files an die aufrufende Methode zurückgegeben werden:

```
import java.io.*;

interface ListerCallback {
    public int perform(File theFile);
}
```

Der Java-Praxiskurs II

18

Ü1.2.4 Lines of Code rekursiv

- In der Klasse `InspectCode`, die das Interface implementiert, wird zusätzlich zur Ausgabe der Zeilenanzahl der Wert von `lines` auch zurückgegeben:

```
public int perform(File theFile) {
    int lines = 0;
    ...
    try {
        ...
        while { ... }
        ...
    } catch (...) { ... }
    return lines;
}
```

Ü1.2.4 Lines of Code rekursiv

- In der `doList`-Methode der Klasse `DirectoryLister`
 - werden die von `perform` ermittelten Zeilenzahlen aufsummiert
 - wird die `doList`-Methode rekursiv für Unterverzeichnisse aufgerufen

```
public int doList(String directory) {
    File dir = new File(directory);
    String list[] = dir.list(new FilenameFilter() {
        public boolean accept(File dir, String name) {
            return (new File(dir, name).isDirectory() ||
                name.endsWith(".java"));
        }
    });
    int result = 0;
    for (int i = 0; i < list.length; i++) {
        File f = new File(directory, list[i]);
        if (f.isDirectory()) result = result + fcb.perform(f);
        else result = result + doList(directory + "/" + list[i]);
    }
    return result;
}
```

Ü1.2.4 Lines of Code rekursiv

- In der `main`-Methode von `DirectoryLister` wird die von `doList` ermittelte Gesamtsumme ausgegeben:

```
public static void main(String args[]) {
    DirectoryLister dl = new DirectoryLister(
        new InspectCode());

    System.out.println("Result: " + dl.doList("."));
}
```

Ü1.3.1 Aufteilen von Dateien

- Schreiben Sie ein Java-Programm, das die gegebene Textdatei einliest und alle positiven Zahlen in einer Textdatei `positiv.txt` und alle negativen Zahlen in einer Datei `negativ.txt` speichert.

```
12 13 14
-12
-14
17
```

Ü1.3.1 Aufteilen von Dateien

```
import java.io.*;

public class Aufteilen {

    private PrintWriter positiv;
    private PrintWriter negativ;

    private void verarbeite(String inhalt) { ... }

    public void arbeite()
        throws FileNotFoundException, IOException { ... }

    public static void main(String[] args)
        throws FileNotFoundException, IOException {

        Aufteilen aufteilen = new Aufteilen();
        aufteilen.arbeite();
    }
}
```

Ü1.3.1 Aufteilen von Dateien

```
public void arbeite() throws FileNotFoundException, IOException {
    positiv = new PrintWriter(new FileWriter("positiv.txt"));
    negativ = new PrintWriter(new FileWriter("negativ.txt"));

    FileReader fr = new FileReader("eingabe.txt");

    int c;
    StringBuffer buf = new StringBuffer(); // Zeichen einlesen
    while ((c = fr.read()) != -1) {
        // Wenn Leer- oder Trennzeichen: verarbeiten!
        if ((c == '\n') || (c == '\t') || (c == ' ') || (c == '\r')) {
            verarbeite(buf.toString());
            buf = new StringBuffer();
        }
        else
            buf.append((char) c); // ansonsten Zeichen sammeln
    }

    verarbeite(buf.toString()); // ggf. restliche Zeichen verarbeiten
    fr.close(); positiv.close(); negativ.close();
}
```

Ü1.3.1 Aufteilen von Dateien

```
private void verarbeite(String inhalt) {
    inhalt = inhalt.trim(); // Leerzeichen von String entfernen

    if (inhalt.length() != 0) { // wenn String != Leerstring
        try {
            // String in int umwandeln
            int i = Integer.parseInt(inhalt);
            // je nach Vorzeichen in entsprechende Datei schreiben
            if (i < 0)
                negativ.println(i);
            else
                positiv.println(i);
        } catch (NumberFormatException nfe) {
            // bei Umwandlungsfehler Meldung ausgeben
            nfe.printStackTrace();
        }
    }
}
```

Der Java-Praxiskurs II

25

Ü1.3.2 Aufteilen und Lesen von Dateien

- Erweitern Sie Ihr Programm so, dass nach dem Aufteilen der Daten die Dateien `positiv.txt` und `negativ.txt` an der Konsole ausgegeben werden.

```
public static void main(String[] args)
    throws FileNotFoundException, IOException {

    Aufteilen aufteilen = new Aufteilen();
    aufteilen.arbeite();

    // ab hier beginnt die Ausgabe
    aufteilen.ausgabe("positiv.txt");
    aufteilen.ausgabe("negativ.txt");
}
```

Der Java-Praxiskurs II

26

Ü1.3.2 Aufteilen und Lesen von Dateien

```
private void ausgabe(String filename)
    throws IOException, FileNotFoundException {

    BufferedReader br = new BufferedReader(
        new FileReader(filename));

    String line;

    System.out.println("Ausgabe von " + filename);

    while ((line = br.readLine()) != null)
        System.out.println(line);

    br.close();
}
```

Der Java-Praxiskurs II

27

Wdh.: Serialisierung von Objekten

- `FileOutputStream` kann Bytes in eine Datei schreiben
- `ObjectOutputStream` kann Objekte in Bytes wandeln
 - über `write`-Methoden für diverse primitive und Referenzdatentypen
 - Umwandlung von Referenzdatentypen in Bytes passiert automatisch

```
Buch javaBuch = new Buch(...); float preis = 41.99;
try {
    FileOutputStream fos =
        new FileOutputStream("ausgabe.ser");
    ObjectOutputStream oos =
        new ObjectOutputStream(fos);
    oos.writeObject(javaBuch);
    oos.writeFloat(preis);
}
catch (IOException e) {...}
```



Der Java-Praxiskurs II

28

Ü1.4.1 Objektserialisierung

- Ergänzen Sie die vorgegebenen Klassen `Buch`, `Benutzer` und `BuchSchreiber` so, dass die von `BuchSchreiber` angelegten `Benutzer` und `Bücher` per Objektserialisierung in einer Datei gespeichert werden.

```
import java.io.Serializable;
public class Buch implements Serializable {
    private String autor; private String titel;
    private String verlag; private int erscheinungsjahr;
    public Buch(String autor, String titel, String verlag,
        int erscheinungsjahr) {
        this.autor = autor; this.titel = titel;
        this.verlag = verlag;
        this.erscheinungsjahr = erscheinungsjahr;
    }
    public String toString() {
        return autor + ": " + titel + ". " + verlag + ", " +
            erscheinungsjahr;
    }
}
```

Der Java-Praxiskurs II

29

Ü1.4.1 Objektserialisierung

```
import java.io.Serializable;

public class Benutzer implements Serializable {
    private String nachname;
    private String vorname;

    public Benutzer(String nachname, String vorname) {
        this.nachname = nachname;
        this.vorname = vorname;
    }

    public String toString() {
        return nachname + ", " + vorname;
    }
}
```

Der Java-Praxiskurs II

30

Ü1.4.1 Objektserialisierung

```

import java.util.*;
import java.io.*;

public class BuchSchreiber {
    public static void main(String[] args) {
        Benutzer benutzer = new Benutzer("Beydeda", "Sami");
        Vector buecher = new Vector();
        buecher.add(new Buch(...));
        ...
        try {
            ObjectOutputStream oos = new ObjectOutputStream(
                new FileOutputStream("objekte.ser"));
            oos.writeObject(benutzer);
            oos.writeObject(buecher);
            oos.close();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Der Java-Praxiskurs II 31

Ü1.4.2 Objektdeserialisierung

- Ergänzen Sie die vorgegebene Klasse `BuchLeser` so, dass sie per Deserialisierung die in `objekte.ser` gespeicherten Objekte rekonstruiert und am Bildschirm ausgibt.

```

import java.util.*; import java.io.*;

public class BuchLeser {
    public static void main(String[] args) {
        Benutzer benutzer;
        Vector buecher;
        try {
            ObjectInputStream ois = new ObjectInputStream(
                new FileInputStream("objekte.ser"));
            benutzer = (Benutzer) ois.readObject();
            buecher = (Vector) ois.readObject();
            ois.close();
            ... // Fortsetzung auf nächster Folie
        }
    }
}

```

Der Java-Praxiskurs II 32

Ü1.4.2 Objektdeserialisierung

- Ergänzen Sie die vorgegebene Klasse `BuchLeser` so, dass sie per Deserialisierung die in `objekte.ser` gespeicherten Objekte rekonstruiert und am Bildschirm ausgibt.

```

... // Fortsetzung von vorheriger Folie
System.out.println(benutzer);
Enumeration buecherListe = buecher.elements();
while (buecherListe.hasMoreElements()) {
    System.out.println(buecherListe.nextElement());
}
}
catch (IOException ioe) {
    ioe.printStackTrace();
}
catch (ClassNotFoundException cnfe) {
    cnfe.printStackTrace();
}
}
}

```

Der Java-Praxiskurs II 33

Ü1.4.3 Probleme bei Deserialisierung

- Wenn die Klasse `Buch` fehlt, wird zur Laufzeit von `BuchLeser` eine `ClassNotFoundException` geworfen, da bei der Deserialisierung des `Vectors` die darin enthaltenen `Buch`-Objekte ohne entsprechende Klassendefinition nicht rekonstruiert werden können.
- Wenn die Klasse `Benutzer` fehlt, schlägt bereits das Kompilieren von `BuchLeser` fehl, weil der entsprechende Bezeichner im Quelltext nicht aufgelöst werden kann.
- Beim Fehlen von `Buch` gibt es keinen Kompilierfehler, weil `Buch` nicht direkt von `BuchLeser` referenziert wird.
 - **Vorsicht:** Fehlerquelle!

Der Java-Praxiskurs II 34

Ü1.4.x Probleme bei Deserialisierung

- Das nachträgliche Hinzufügen/Entfernen von Attributen führt beim Deserialisieren zur `InvalidClassException: Klassenname; local class incompatible`, da die in der serialisierten Datei gespeicherte Klassenstruktur nicht mit der der neuen Instanz übereinstimmt.
- Die nachträgliche Änderung der Attributreihenfolge wirkt sich nicht auf die Deserialisierung aus, da Klassenstruktur in der Datei auf die der neuen Instanz abgebildet werden kann.
- Die nachträgliche Änderung der Klassenreihenfolge führt beim Deserialisieren zu einer `ClassCastException`, da zunächst die Benutzer-Instanz gelesen wird, die sich jedoch nicht zu einem `Vector` umwandeln lässt.

Der Java-Praxiskurs II 35