Zusammenfassung Prog1 HS2012

Programmieren 1

Emanuel Duss
emanuel.duss@gmail.com

17. Dezember 2012

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Inhaltsverzeichnis

Emanuel Duss
http://emanuelduss.ch

1 Einführung

1.1 Überblick Java

Der Java Compiler (`javac`) macht aus dem Java-Quellcode Java Bytecode. Dieser läuft auf der Java Virtual Machine (JVM), welche auf der realen Maschine läuft.

Die Dokumentation zu Java findet man auf: http://docs.oracle.com/javase/7/docs/api/.

1.2 Schritte der Programmierung

1. Anforderungsanalyse: Das Problem fixieren
2. Entwurf: Lösungsweg definieren
3. Implementierung: Java Programm schreiben
4. Test: Funktion Prüfen
5. Betrieb: Programm verbessern und erweitern

1.3 Grundgerüst

```
//////////////////////////////////////////////////////////////
//
// HelloWorld.java
// Mein erstes Java Programm ;-)
//
//////////////////////////////////////////////////////////////

public class HelloWorld
{
  public static void main (String[] args)
  {
    // Deklarationen
    int n;

    // Initialisierung
    n = 23;
```

```
    System.out.println ("Die Zahl n ist: " + n);
  }
}

// EOF
```

1.4 Kompilieren und Ausführen

```
$ javac HelloWorld.java
$ ls HelloWorld.class
HelloWorld.class
$ java HelloWorld
```

1.5 Ausdrücke und Variabeln

- Literal: 523, true, false, w, 'Eris!'
- Variablenname: i, foo, fnord
- Unärer Operator: -523, +42, -i, i++
- Priorität: Klammern, von Links nach Rechts, Unäre Operationen, Binäre Operationen, Multiplikation/Division/Modulo, Addition/Subtraktion

1.6 Variabeln initialisieren und zuweisen

```
int i;        // Deklaration
i = 523;      // Initialisieren
int w = 23;   // Deklarieren und initialisieren

final int answer; // final = Konstante
answer = 42;      // Kann jetzt nicht mehr geaendert werden!
```

1.7 Unäre Operationen

Postfix-Inkrement Auslesen und dann inkrementieren: `x++`

Prefix-Inkrement Inkrementieren und dann auslesen: `++x`

1.8 Verkürzte Schreibweisen

```
x += y; // x = x + y
x -= 1; // x = x -1
x++     // x = x + 1 (Postfix-Inkrement)
        // (zuerst Anfangswert zurueckgeben, dann inkrementieren)
++x     // x = x + 1 (Prefix-Inkrement)
        // (zuerst Inkrementieren, dann Anfangswert zurueckgeben)
```

1.9 Logische Operatoren

```
a && b   // a UND b
a || b   // a ODER b
!b       // Nicht b
```

1.10 Kontrollstrukturen

1.10.1 Block

```
int foo = 523;
{
  init i, j; // i und j nur im Block verfuegbar
  i = 23;
  j = i + k; // k von aussen verfuegbar
}
```

1.10.2 If-Abfrage

```
if (celsius >= 30) {
  System.out.println("Hot");
}
else if (celsius >= 25) {
  System.out.println("Warm");
}
else {
  System.out.println("Fooooo");
}
```

Hat es nur eine Anweisung, können die Blöcke weggelassen werden.

```
// Das
x = a ? b : c;
// Ist das selbe wie das:
if (a) {
  x = b;
}
else {
  x = c;
}
```

1.10.3 Switch-Statemetn

```
switch (Ausdruck) {
  case Wert1:
    Anweisungen;
    break;
case Wert2:
    Anweisungen;
    break;
default:
    Anweisungen;
}
```

1.10.4 Schleifen

```
// While Schleife
while (Bedingung) {
  continue; // continue bricht den aktuellen Schleifenvorgang
            // ab und beginnt nochmals von oben
  Anweisungen
}

// do-while Schleife
do {
    ...
    break; // break bricht die ganze Schleife ab
} while (Bedingung);

// For Schleife
for (int i = 0; i >= 100; i++) {
  ...
}
```

```
// For-Each Schleife
for (String s: args) {
  System.out.println(s)
}
```

2 Datentypen und Referenzen

2.1 Definitionen

- Negative Zahl Binär = Zweierkomplement: Alle Bits invertieren und +1 rechnen.

```
// Ganzzahlen
byte  b = 30;      // Dezimal
short s = 034;     // Oktal (0 am Anfang)
int   i = 0x23;    // Hexadezimal (0x Am Anfang)
long  l = 0x23L    // Long braucht ein 'L' am Schluss!

// Fliesskommazahlen
double d = 4.3;     // Mit einem Punkt
double h = 4d;      // Oder mit einem d (double) bzw. f (float)
float  f = 4.3;
float  g = 23f;
double e = 23E5;    // Exponent: 23 * 10^5
       e = 5e-23;
```

Darstellung von einem Float -42E5:

Bit 0	Bit 23	Bit 31
Mantisse	Exponent	+ / -
42	5	-

2.2 Datentypen

Typ	**Inhalt**	**Genauigkeit**	**Wertebereich**
boolean			
char			
byte	8 Bit	Ganzzahlen	-2^7 bis 2^7-1 = -128 ... 127
short	16 Bit	Ganzzahlen	-2^{15} bis $2^{15}-1$ = -32k ... 32k
int	32 Bit	Ganzzahlen	-2^{31} bis $2^{31}-1$ = -2G ... 2G
long	64 Bit	Ganzzahlen	-2^{64} bis $2^{64}-1$
float	4 Bytes	23 Bit (7 Dez-Stellen)	$-3.4*10^{38}$ bis $3.4*10^{38}$
double	8 Bytes	52 Bit (15 Dez-Stellen)	$-1.7*10^{308}$ bis $1.7*10^{308}$

```
// Numerische Konstanten

// MAX_Value Werte
```

```
System.out.println("Byte.MAX_VALUE = " + Byte.MAX_VALUE);
System.out.println("Short.MAX_VALUE = " + Short.MAX_VALUE);
System.out.println("Integer.MAX_VALUE = " + Integer.MAX_VALUE);
System.out.println("Long.MAX_VALUE = " + Long.MAX_VALUE);
System.out.println("Float.MAX_VALUE = " + Float.MAX_VALUE);
System.out.println("Double.MAX_VALUE = " + Double.MAX_VALUE);

// MIN_Value Werte
System.out.println("Byte.MIN_VALUE = " + Byte.MIN_VALUE);
System.out.println("Short.MIN_VALUE = " + Short.MIN_VALUE);
System.out.println("Integer.MIN_VALUE = " + Integer.MIN_VALUE);
System.out.println("Long.MIN_VALUE = " + Long.MIN_VALUE);
System.out.println("Float.MIN_VALUE = " + Float.MIN_VALUE);
System.out.println("Double.MIN_VALUE = " + Double.MIN_VALUE);
```

```
Byte.MAX_VALUE = 127
Short.MAX_VALUE = 32767
Integer.MAX_VALUE = 2147483647
Long.MAX_VALUE = 9223372036854775807
Float.MAX_VALUE = 3.4028235E38
Double.MAX_VALUE = 1.7976931348623157E308
Byte.MIN_VALUE = -128
Short.MIN_VALUE = -32768
Integer.MIN_VALUE = -2147483648
Long.MIN_VALUE = -9223372036854775808
Float.MIN_VALUE = 1.4E-45
Double.MIN_VALUE = 4.9E-324
```

- Falsche Ergebnisse bei Bereichsüberschreitung
- 2147483647 + 1 gibt einen Fehler: -2147483648

2.2.1 Datentyp char

- Zeichen haben den Wert des jeweiligen ASCII-Codes
- a = 97 = 0x61; A = 65 = 0x41
- A + 32 = 97 = a
- Intern als Unicode UTF-16 abgelegt

- ’0̆061’ ersetzt der Compiler automatsich zu A

```
   2 3 4 5 6 7       30 40 50 60 70 80 90 100 110 120
  ------------      --------------------------------
0:   0 @ P ‘ p     0:    (  2  <  F  P  Z  d   n   x
1: ! 1 A Q a q     1:    )  3  =  G  Q  [  e   o   y
2: " 2 B R b r     2:    *  4  >  H  R  \  f   p   z
3: # 3 C S c s     3: !  +  5  ?  I  S  ]  g   q   {
4: $ 4 D T d t     4: "  ,  6  @  J  T  ^  h   r   |
5: % 5 E U e u     5: #  -  7  A  K  U  _  i   s   }
6: & 6 F V f v     6: $  .  8  B  L  V  ‘  j   t   ~
7: ’ 7 G W g w     7: %  /  9  C  M  W  a  k   u  DEL
8: ( 8 H X h x     8: &  0  :  D  N  X  b  l   v
9: ) 9 I Y i y     9: ’  1  ;  E  O  Y  c  m   w
A: * : J Z j z
B: + ; K [ k {
C: , < L \ l |
D: - = M ] m }
E: . > N ^ n ~
F: / ? O _ o DEL
```

Quelle: man 7 ascii

```
int i = ’A’
char c = (char)65
```

2.3 Typkonversion

```
short s; int i; long l;
...

// Implizite Typkonversion
// Wenn es ’reinpasst’, macht der Commpiler das automatisch:
l = i;
i = s;
l = s;
...
// Explizite Typkonversion
// Wenn es nicht ’reinpasst’, muss man das dem  Commpiler sagen (
   casten):
```

```
i = (int)l;
s = (short)i;
s = (short)l;
```

- Ganzzahl zu Ganzzahl: Nur untere Bits werden genommen: (byte)0x1234 = 0x34
- Gleitkommazahl zu Ganzzahl: NaN ⇒ 0; sonst wird ein möglichst naher Wert genommen: (int)3.5 = 3

2.4 Mathematische Methoden

```
Math.abs()
Math.sqrt()
Math.log10()
Math.exp() // e-Funktion
Math.sin()
Math.pow(double base, double exponent) // potenzieren
```

3 Klasse, Objekte, Methoden

3.1 Klasse

```
class Rectangle {
  // Instanz-Variablen (Zustand)
  int length;
  int width;

  void setSize(int length, int width) {
    // Namenskollision: Mit this.length wird direkt
    // auf die Instanzvariable zugegriffen
    // length greift auf die Variable der Methode zu
    this.length = length;
    this.width = width;
  }

  // Instanz-Methoden (Verhalten)
  void print() {
    System.out.println("Length: " + length); // oder this.length
    System.out.println("Width: " + width);
  }

  void printDetails() {
    int area = length * width;
    System.out.println("A: " + area);
  }

  void stretch(int factor) {
    length *= factor;
    width *= factor;
  }

  int getArea() {
    // Rueckgabewert, kompatibel zu Rueckgabetyp (int)
    return length * width;
  }
  void printAll() {
    print(); // oder this.print();
    printDetails();
  }
}
```

3.2 Objekt erzeugen

```
Rectangle r = new Rectangle();   // Objekt erzeugen
r.length = 5;      // Variabeln setzen
r.width = 23;
r.stretch(42);     // Methode aufrufen
r.print();         // Funktion aufrufen
```

3.3 Namenskollision

Mit `this.instanzvariable` kann man direkt auf die Instanzvariabeln zugreifen.

```
class Square {
  int length = 10;

  void test() {
    int length = 5;
    System.out.println(length);      // output: 5
    System.out.println(this.length); // output: 10
  }
}
```

3.4 Parameter-Übergabe

Wertetypen

```
void printDouble(int i) {
  i *= 2;
  System.out.println(i);
}

int i = 5;
printDouble(i);
// i == 5!
```

Referenztypen

```
void change(Point p) {
  p.x *= 2; p.y *= 2;
  p = null;
}
```

```
Point a = new Point();
a.x = 1;
a.y = 1;
change(a);
// a != null
// a.x == 2, a.y == 2
```

Mit `this.instanzvariable` kann man direkt auf die Instanzvariabeln zugreifen.

```
class Square {
  int length = 10;

  void test() {
    int length = 5;
    System.out.println(length);       // output: 5
    System.out.println(this.length); // output: 10
  }
}
```

3.5 Method Overloading (Überladung)

Mehrere Methoden können mit gleiche Namen in der Klasse vorkommen. Diese haben dann eine unterschiedliche Parameterliste.

```
lass Rectangle {
  int width, length;

  void set() {
   // Aufruf mit: r.set();
    width = 1; length = 1;
  }
  void set(int size) {
    // Aufruf mit: r.set(12);
    width = size; length = size;
  }
  void set(int length, int width) {
    // Aufruf mit: r.set(2, 3);
    this.length = length; this.width = width;
  }
}
```

3.6 Rückgabe von this

```
class Rectangle {
int length, width;
Rectangle setSize(int length, int width) {
this.length = length;
this.width = width;
return this;
}
void print() { ... }
}
new Rectangle().setSize(10, 20).print();
```

3.7 Sichtbarkeit

```
public class Rectangle {
  // Nur in dieser Klasse zugreifbar
  private int length;
  private int width;

  // In und ausserhalb Klasse aufrufbar
  public void set(int length, int width) {
    this.length = length; this.width = width;
  }

  public int getLength() { return length; }

  public int getWidth() { return width; }

  public void print() {
    System.out.println("Rectangle");
  printDetails();
  }
  private void printDetails() { ... }
}
```

3.8 Konstruktoren

Gleicher Name wie Klasse, kein Rückgabewert. Mit `this(foo,bar)` kann ein eigener anderer Konstruktor mit passenden Argumenten aufgerufen werden

```
Expliziter Aufruf mit \texttt{this(argumentlist);}:
public class Rectangle {
```

```
  private int length;
  private int width;

  public Rectangle() {
    this(10, 20); // Ruft den entsprechenden Konstruktor auf
  }
  public Rectangle(int length, int width) {
    this.length = length; this.width = width;
  }
}
```

3.9 Array

Ein Array ist ein Objekt mit fixer Anzahl von Elementen (alle vom gleichen Typ).

```
// Array deklarieren
int fnord[]; // oder
int[] foo;
// Array-Objekt erzeugen
fnord = new int[23];
// Direkt
int[] foo = {5, 23, 42};

// Elemnte verwenden
fnord[1] = 5;
fnord[2] = 23;
fnord[23] = 24;
```

Geschachtelte Arrays

```
int[][] m = new int[2][3];

int[][] m = new int[2][3];
  for (int i = 0; i < m.length; i++) {
    for (int j = 0; j < m[i].length; j++) {
    m[i][j] = ...;
  }
}

// Direkt initialisieren
int[][] m = {
    { 0 },
    { 1, 2 },
```

```
    { 3, 4, 5 }
};
```

Arrays vergleichen

4 Vererbung

4.1 Klassenvererbung

Die Klasse `Object` ist in Java die oberste Basisklasse aller Klassen. Diese Klasse hat bereits eingebaute Methoden wie `toString()`, `equals(Object obj)` oder `clone()`.

```
class Vehicle { ...}
class Car extends Vehicle {...}
```

Protected: Membervariablen und Methoden Nur in Subklassen verfügbar!

```
protected int currentSpeed;
```

Konstruktoren: Falls keinen eigenen: der Konstruktor der Superklasse wird automatisch aufgerufen. Mit `super()` kann man den Konstruktor der Superklasse aufrufen.

4.2 Polymorphismus

```
Car c = new Car();
Vehicle v = new Car();
Object o = new Car();
```

Der Datentyp spezifiziert, welche Operationen wir durchführen können (Methoden / Variabeln). Das Object o kann nicht auf die Membervariabeln der Klasse Car zugreifen.

4.3 Abstrakte Klassen und Methoden

Die Oberklasse `Vehicle` möchte ich gar nicht instanzieren.

```
abstract class Vehicle {...}
```

Ich möchte nur die Methode `report()` der Subklassen verwenden:

```
abstract class Vehicle {
  abstract void report();
}
```

5 Rekursion

```
long factorial(int number) {
  if (number <= 0) {
    return 1;
  }
  else {
    return number * factorial(number - 1);
  }
}
```

6 Garbage Collection

6.1 Finalizer

Wird beim Abräumen ausgeführt.

6.2 Static Keyword

Nicht objektorientiert. Variable lebt auf der Klasse und nicht auf den Instanzen. Diese Variabeln existieren nur einmal pro Klasse. Zugriff mit `Klassenname.variable`.

```
class Rectangle {
  static int maxLength;
    ...
  }
  Rectangle.maxLength = 100;
  r.maxLength // nicht empfohlen.
```

6.2.1 Static Methoden

Globale Methoden. Bei den static Methoden können nur static Variablen verwendet werden!

```
class Rectangle {
  static int maxLength;
  ...
  static void getMaxLength() {
    return maxLength;
  }
}
Rectangle.getMaxLength();
```

Konstanten mit `static final`. Darf `public` sein.

6.3 Final Schlüsselwort

- Final bei Variabeln: konstant bei Zuweisung

- Final bei Methoden: nicht überschreibbar
- Final bei Klassen: kann nicht mehr abgeleitet werden; keine Subklassen mehr

Konstanten-Klasse: public final class Constants private Constants() public static final int NO_ROWS = 8; public static final int NO_COLUMNS = 8; Standardkonstruktor private = keine instanzierung möglich.

6.4 Schnittstellen

Jede Schnittstelle definiert auch einen Typ. Gleichnamige Konstanten: Interface angeben! Methoden von zwei Interfaces mit unterschiedlichen Rückgabetypen sind nicht möglich!

7 Exceptions

```
String clip(String s) throws Exception {
if (s == null) {
  throw new Exception("String is null");
}
if (s.length() < 2) {
  throw new Exception("String is too short");
}
  return s.substring(1, s.length()-1);
}
```

Wirft eine Funkton eine Exception (direkt oder indirekt über einen Methodenaufruf), muss man das angeben:

```
void test() throws Exception {
  String s = null;
  String c = clip(s);
  System.out.println(c);
}
```

Exceptions behandeln: (throws ist hier nicht mehr nötig)

```
void test() {
  try {
  }
  catch (exception e) {
  }
}
```

Vordefinierte Exceptions:

```
String clip(String s) throws Exception {
  if (s == null) {
    throw new NullPointerException("no string");
  }
  if (s.length() < 2) {
    throw new IllegalArgumentException("short string");
  }
    return s.substring(1, s.length()-1);
  }
```

Mehrere Catch-Klauseln:

```
try {
```

```
  ... regular code ...
} catch (exceptionType1 variableName) {
  ... error handling ...
} catch (exceptionType2 variableName) {
  ... error handling ...
}
```

- ArithmeticException,
- NullPointerException,
- ArrayIndexOutOfBoundsException
- IllegalArgumentException

Eigene Stack Klasse:

```
public class StackException extends Exception {
  private static final long serialVersionUID = 1L;
  public StackException(String message) {
    super(message);
  }
}
...
public Stack(int capacity) throws StackException {
    if (capacity < 0 || capacity >= 65536) {
      throw new StackException("Gosche!");
    }
    elements = new Object[capacity];
    nextFreeIndex = 0;
  }
```

7.0.1 Assertion (Zusicherung)

Boolean-Ausdrucke, der immer zutreffen muss. Das wird während der Entwicklung eingesetzt und darf in der produktiven Version nicht mehr vorkommen.

```
assert expression: string;
```

```
int foo = foo(x, y);
assert foo > 0: "foo ist nicht positiv";
```

Run Configuration: VM arguments (-ea):

```
java -ea MyProgram
```

Externe Quellen (User, Files) nicht mit Assertions prüfen, sondern mit Exceptions.

8 Collections

8.1 Wrapper-Klassen

Verpacken eines Wertes in ein Objekt.

- char → Character
- int → Integer

Funktionen der Wrapper-Klassen:

```
Long.MAX_VALUE
int hex = Integer.parseInt(args[0], 16);
String bin = ndw Integer(i).toBinaryString();

// Boxing:
Integer iObj = new Integer(i);  // Manuell
Integer iObj = i;  // Automatisches Boxing

// Unboxing
int i = iObj.intValue();  // Manuell
int i = iObj;
```

8.2 Array und ArrayList

Arrays haben eine fixe Grösse mit effizientem Zugriff, jedoch ineffizient beim Hinzufügen/-Löschen. ArrayList hat eine dynamische Anzahl Elemente.

	Array	ArrayList
definieren	String[] a	ArrayList<String> a
erzeugen	a = new String[100]	a = new ArrayList<String>()
schreiben	a[i] = 'Fnord'	a.set(i, 'Fuu')
lesen	String x = a[i]	String x = a.get(i)

```
import java.util.ArrayList;
// Wrapper-Type (Objekt) verwenden, nicht den Datentyp
ArrayList<String> stringList = new ArrayList<String>();
```

```
// Kurzschreibweise
ArrayList<String> stringList = new ArrayList<>();
stringList.add("element #0");
stringList.get(0);
stringList.set(0,"Fuuuuuu"); // Element ersetzen
stringList.get(5); // IndexOutOfBoundsException
stringList.remove(0);
```

8.3 Iteratoren

```
// Traditionell
int i = 0;
while (i < stringList.size()) {
  String elem = stringList.get(i++);
  System.out.println(elem);
}

// Iterator
Iterator<Integer> it = stringList.iterator();
while (it.hasNext()) {
  String elem = it.next();
  System.out.println(elem);
}

List<String> stringList = new ArrayList<>();
stringList.add("Hi");
stringList.add("there");
stringList.add("How");
stringList.add("are");
stringList.add("you");
stringList.add("doing");
Collections.sort(stringList);
for (String s : stringList) {
  System.out.println(s);
}
```

8.4 HashMap

Die Equals-Methode und die Hashcode-Methode müssen korrekt überschrieben werden.

```
// Neue HashMap (Schluessel: Martikelnummer, Wert: Student)
Map<Integer, Student> map = new HashMap<>();
```

```
Student st1 = new Student("Andrea", "Meier", 20000);
Student st2 = new Student("Bertha", "Mueller", 70000);
Student st3 = new Student("Clara", "Schneider", 13000);

map.put(st1.getRegNumber(), st1);
map.put(st2.getRegNumber(), st2);
map.put(st3.getRegNumber(), st3);

// Finden nach Schluessel (konstante Zeit)
Student st = map.get(12345);

// Collection aller Werte (unsortiert als view)
for (Student s : map.values()) {
  System.out.println(s);
}
```

9 I/O und Streams

9.1 Byte Streams

Lesen:

```
FileInputStream fis = new FileInputStream("myFile.data");
int value = fis.read();
while (value >= 0) { // -1: end of file (wenn positiv)
  byte b = (byte)value; // Gelesenes Byte
  // work with b
  value = fis.read();
}
fis.close();
```

Schreiben:

```
FileInputStream fis = new FileInputStream("myFile.data");
try {
  int value = fis.read();
  while (value >= 0) {
    byte b = (byte)value;
    // work with b
    value = fis.read();
  }
}
finally {
  fis.close();
}
```

Klasse:

```
class FileOutputStream ... {
  FileOutputStream(String filePath) {...}
  FileOutputStream(String filePath, boolean append) {...}
  void write(int b);
  // writes only 8 lowest bits
  void close() {...}
  ...
}
Append = an File anfuegen; sonst wirds ueberschrieben.
```

Exceptions

- IOException

9.2 Character Streams

9.2.1 Reader

```
try (FileReader reader = new FileReader("quotes.txt")) {
 int value = reader.read();
 while (value >= 0) {
   -1 = end of file
   char c = (char)value; // 16-bit char
   // use character
   value = reader.read();
 }
}
```

9.2.2 Writer

```
try (FileWriter writer = new FileWriter("test.txt", true)) {
  writer.write("Hello!"); // schreiben
  writer.write('\n');
}
```

9.2.3 Bridge-Klasse

Umwandlung Character ⇔ Bytes.

```
InputStream byteStream = new FileInputStream("quotes.txt")
Reader reader = new InputStreamReader(byteStream, "UTF-8");
```

9.2.4 Buffered Streams

```
FileOutputStream fos = new FileOutputStream("data.bin");
BufferedOutputStream stream = new BufferedOutputStream(fos, 4096);
// write to stream
stream.flush(); // flush buffer
// write to stream
stream.close(); // closes file stream
```

9.3 Objekt-Serialisierung

9.3.1 Klasse implementiert `Serializable` Interface

```
class Person implements Serializable {
  private String firstName;
  private String lastName;
  private transient int age; // Nicht serialisiert
  private List<Person> colleagues = new ArrayList<Person>();
}
```

9.3.2 Serialisieren

```
Person person = new Person(...);
OutputStream fos = new FileOutputStream("serial.bin")
try (ObjectOutputStream stream = new ObjectOutputStream(fos)) {
  stream.writeObject(person);
}
```

9.3.3 Deserialisieren

```
InputStream fis = new FileInputStream("serial.bin")
try (ObjectInputStream stream = new ObjectInputStream(fis)) {
  // Der Stream stream.readObject() muss fuer Person p in Person
     gecastet werden!
  Person p = (Person)stream.readObject();
  // ...
}
```

9.3.4 `transient` Keyword

- Instanzvariabeln von Serialisierung ausschliessen
- Wird beim Deserialisieren mit Default initialisiert

9.3.5 Serial Version UID

```
class Person implements Serializable {
  private static final long serialVersionUID = -6583929648459736324
     L;
```

```
  // ...
}
```

- Klasse kann eigene Version angeben
  - Beim Deserialisieren wird Version überprüft
  - ClassNotFoundException, wenn Version nicht passt
- Klassen-Kompatibiltät steuerbar
  - Änderung an Klasse möglich, wenn Version gleichbleibt
  - Empfehlung: serialVersionUID generieren

9.3.6 Beispiel: Objekte serialisieren

```
// Klasse Contact
public class Contact implements Serializable { ... }

// Klasse PhoneEntry
public class PhoneEntry implements Serializable { ... }

// Klasse ContactBook
public class ContactBook {
  private Map<String, Contact> contactBook = new HashMap<>();
  // ...

  // contactBook serialisieren
  public void save() {
    try(ObjectOutputStream stream = new ObjectOutputStream(new
       FileOutputStream("serial.bin"))) {
      stream.writeObject(contactBook);
    }
    catch (IOException e) {
      e.printStackTrace();
    }
  }

  // contactBook deserialisieren
  public void load() {
    try (ObjectInputStream stream = new ObjectInputStream(new
       FileInputStream("serial.bin"))) {
```

```
      // Zu Map casten!
      contactBook = (Map<String, Contact>)stream.readObject();
    }
    catch (IOException | ClassNotFoundException e) {
      e.printStackTrace();
    }
  }
}
```

10 Packages und Nested Classes

10.1 Packages

- Klassen logisch strukturieren
- Sichtbarkeit einschränken
- Datei im selben Paket beginnt immer mit gleicher Paketdeklaration
- `public class` kann mit Import benutzt werden
- Geschachtelt mit Punkt getrennt: `package <name.>.<subname>.<subsubname>` (müssen auch importiert werden!!!)

```
package p1;
public class A {}

// ...

package p2;
import p1.A;
public class C {}
```

10.2 Geschachtelte Klassen

```
public class Polygon {
  private class Point {
    int x, y;
  }
  // Nur innerhalb LinkedList sichtbar
  private List<Point> points;
  ...
}
```

10.3 Anonyme innere Klasse

- Klasse implementiert FilterCriterion
- Klasse hat keinen eigenen Namen (anonym)

```
void test(File path);
  final String searchText = readFromConsole();
  searchFiles(new File(path), new FilterCriterion() {
    @Override
      public boolean matches(File file) {
      ... my logic using searchText ...
    }
  });
}
```

11 Generics

```
Class Node<T> {
  T value;
  Node<T> left;
  Node<T> right;
  Node(T value) { ... }
  // ...
  T getValue() { ... }
  ...
  Node<T> getLeft() { ... }
  // ...
}
```

11.1 Covarianz

```
Node<? extends Number> node;
Number n = node.getValue(); // wert holen ist erlaubt
node.setValue(3); // ERROR
```

11.2 Contravarianz

```
Node<? super Number> node;
Number n = node.getValue(); // ERROR
node.setValue(3); // OK
```

12 Testen und Dokumentieren

12.1 Testing

```
// Imports
import static org.junit.Assert.*;
import org.junit.Test;

// Asserts
assertEquals(expected, actual)     actual "equals" expected
assertSame(expected, actual)       actual == expected
assertNotSame(expected, actual)    expected != actual
assertTrue(condition)              condition
assertFalse(condition)             !condition
assertNull(value)                  value == null
assertNotNull(value)               value != null
fail()                             Immer verletzt

// Erklaerung angeben:
assertEquals("message", expected, actual);
```

Beispiel:

```
import static org.junit.Assert.*;
import org.junit.Test;

public class MultiSetTest {
  private static final int DEFAULT_TIMEOUT = 2000;

  @Test(timeout = DEFAULT_TIMEOUT)
  public void testEmptyMultiSet() {
    MultiSet<Integer> set = new MultiSet<>();

    assertFalse("no element", set.contains(0));
    assertEquals("empty set", 0, set.size());
    set.remove(0); // no exception
  }

  private static final String ELEM_NAME = "elem";
  @Test(timeout = DEFAULT_TIMEOUT)
  public void testSingleOccurrence() {
    MultiSet<String> set = new MultiSet<>();
    set.add(ELEM_NAME);
    assertTrue("contained", set.contains(ELEM_NAME));
```

```
    assertEquals("set size", 1, set.size());

    set.remove(ELEM_NAME);
    assertFalse("removed", set.contains(ELEM_NAME));
    assertEquals("original size", 0, set.size());
  }

  @Test(expected = IllegalArgumentException.class)
  public void testAddNullElementFails() {
    MultiSet<Double> set = new MultiSet<>();
    set.add(null);
  }
}
```

12.2 Dokumentieren

12.2.1 Klasse

```
/**
* Zusammenfassung mit Punkt am Schluss.
*
* Text
* mehrere
* Zeilen.
*
* @author foo
* @version 1.0
*
*/
```

12.2.2 Konstruktor

```
/**
* Custom constructor accepting Cartesian complex number
* components.
*
* @param real The real part. Must not be NaN or infinity.
* @param imaginary The imaginary part. Neither NaN nor infinity.
* @exception IllegalArgumentException. NaN or infinity value
* for a component.
*/
public Complex(double real, double imaginary) {
  ...
```

```
}
```

13 Enums

13.1 Enum

```
public enum Weekday {
  Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

  public boolean isWeekend() {
    return this == Saturday || this == Sunday;
  }
}
// ...
currentDay = Weekday.Monday;
if (currentDay == Weekday.Sunday) { ... }
// ...
WeekDay currentDay = ...
if (currentDay.isWeekend()) { ... }
```

13.2 Enum mit Konstruktor

```
public enum Weekday {
  Monday(true), Tuesday(true), Wednesday(true), Thursday(true),
    Friday(true), Saturday(false), Sunday(false);
  private boolean workDay;
  Weekday(boolean workDay) {
    this.workDay = workDay;
  }

  // Nur privater Konstruktor (implizit ohne Angabe)
  public boolean isWorkDay() {
    return workDay;
  }
}
```

Literatur

[1] Typo3, Typo3 Version 4 genau erklärt

Abbildungsverzeichnis

Tabellenverzeichnis