Informatik II
Objektorientierte SW-Entwicklung, Algorithmik, Nebenläufigkeit
Sommersemester 2004

Martin Hofmann, Jan Johannsen, Ralph Matthes

Einführung

- Organisatorisches
- Geschichte von Java
- Die Java Virtual Machine
- “Hello World” in Java
- Verwendung einfacher vordefinierter Objekte
- Dokumentation mit Javadoc

Inhalt der Vorlesung

- Einführung in Java: Datentypen, Kontrollstrukturen
- Objektorientierte Programmierung: Klassen, Objekte, Vererbung, Schnittstellen
- Algorithmen für Listen und Bäume, insbes. Balancierung von binären Suchbäumen
- Nebenläufigkeit und Threads
- Entwicklung von grafischen Benutzeroberflächen
- Softwaretechnik: Testen, Modellierung mit UML, Entwurfsmuster, Verifikation mit Hoare Logik

Die Programmiersprache Java

Entstand Anfang der 1990 aus einem Projekt bei Sun Microsystems zur Programmierung elektronischer Geräte (Set top boxen, Waschmaschinen, etc.). Leiter: James Gosling.

Wurde dann zur plattformunabhängigen Ausführung von Programmen in Webseiten (“Applets”) verwendet.

Seitdem stark expandiert, mittlerweile neben C++ die beliebteste Sprache.

Vorteile von Java

- Sicherheit
- Portierbarkeit (plattformunabhängig durch JVM)
- (Relativ) sauberes Sprachkonzept (Objektorientierung von Anfang an eingebaut)
- Verfügbarkeit großer Bibliotheken

Nachteile von Java

- Teilweise kompliziert und technisch, da nicht für Studenten entworfen (wie Pascal)
- Zu groß für ein Semester
- Ausführung relativ langsam und speicherplatzintensiv.

Die Java Virtual Machine

Java Programme werden i.a. nicht in Maschinensprache übersetzt, sondern in eine Folge von elementaren Befehlen (*Bytecodes*), die von einer *virtuellen Maschine* (JVM) ausgeführt werden.

Die JVM verwendet einen Stapel (*stack*) zur Speicherung von Zwischenergebnissen.

Beispiel:

```
iload        40
bipush      100
if_icmpgt   240
```

1. Lege den Inhalt der Speicherstelle 40 auf den Stapel.
2. Lege den Wert 100 auf den Stapel.
3. Falls der erste Wert größer als der zweite ist, dann springe zur Speicherstelle 240 (ansonsten führe den nächsten Befehl aus).

Plattformunabhängigkeit

Die JVM, die den Bytecode ausführt, ist auf unterschiedlichen Plattformen (Windows, Unix, Mac, Mobiltelefon, PDA) implementiert.

Damit kann Java Bytecode auf all diesen Plattformen ausgeführt werden.

Eine Windows `.exe` Datei kann dagegen nur unter Windows ausgeführt werden.

Dadurch wird es insbesondere möglich Programme über das WWW zu verschicken (“*Applets*”).

Das erste Java Programm

```
public class Hello
{
    public static void main(String[] args)
    {
        /* Hier findet die Ausgabe statt */
        System.out.println("Hello, World!");
    }
}
```

Durchführung am Rechner

Um es auszuführen müssen wir

- Eine Datei `Hello.java` anlegen
- In die Datei den Programmtext schreiben
- Den Java Compiler mit dieser Datei aufrufen. Er erzeugt dann eine Datei `Hello.class`, die die entsprechenden JVM Befehle enthält. Bei Unix: `javac Hello.java`.
- Diese Datei mit der JVM ausführen. Bei Unix: `java Hello`.

Anatomie unseres Programms

Einrückungen etc. spielen keine Rolle (wie in OCAML).

Kommentare werden in `/*...*/` eingeschlossen. Sie werden von `javac` ignoriert.

Zwischen den Mengenklammern steht die Definition der Klasse.

Das Schlüsselwort `public` besagt, dass diese Klasse “öffentlich” sichtbar ist, im Gegensatz zu `private`.

Wir brauchen uns im Moment nur zu merken, dass ein Programm in so eine Klassendefinition eingeschlossen werden muss und dass Klassenname und Dateiname übereinstimmen müssen.

`public static void main(String[] args){···}`

definiert eine Methode des Namens `main` in der Klasse `Hello`.

Zwischen den Mengenklammern steht die Definition der Methode.

Die Methode des Namens `main` wird automatisch beim Programmstart ausgeführt. Andere Methoden, werden innerhalb des Programms aufgerufen. Etwa `berechneZinsen`, `verschiebeRaumschiff`, `openConnection`, ...

Das Schlüsselwort `static` bedeutet, dass `main` im Prinzip eine Funktion (und keine “richtige” Methode) ist. Mehr dazu später.

Der Parameter `String[] args` erlaubt es, dem Programm beim Aufruf Daten, etwa einen Dateinamen mitzugeben. Man darf ihn nicht weglassen.

Keine Angst

All das erklären wir erst viel später. Für den Anfang merken wir uns, dass Programme immer so aussehen müssen:

```
public class Klassenname
{
  public static void main(String[] args)
  {
    Hier geht's los
  }
}
```

und in einer Datei *Klassenname*`.java` stehen müssen.

Statements

Die Methodendefinition (der Programmrumpf) besteht aus einer Folge von Statements (deutsch: “Befehlen”).

Hier haben wir nur ein Statement:

```
System.out.println("Hello, World!");
```

Statements enden immer mit Semikolon (Strichpunkt, `;`).

Dieses Statement ruft die eingebaute Methode `println` des Objektes `out` in der Klasse `System` mit dem Argument (Parameter) `"Hello, World!"` auf.

Mehrere Statements

```
System.out.println("Guten Tag.");
System.out.println("Urlaubsbeginn 18. Urlaubsende 31. Das ma
System.out.println(31-18+1);
System.out.println("Urlaubstage.");
System.out.println("Auf Wiedersehen.");
```

Hier ist `31-18+1` ein arithmetischer Ausdruck. Sein Wert wird berechnet und von `println` ausgegeben.

Will man keine Zeilenumbrüche, kann man auch die Methode `print` verwenden.

```
System.out.println("Guten Tag.");
System.out.println("Urlaubsbeginn 18ter Urlaubsende 31ter.");
System.out.print("Das macht ");
System.out.print(31-18+1);
System.out.println(" Urlaubstage.");
System.out.println("Auf Wiedersehen.");
```

Ergebnis:

```
Guten Tag.
Urlaubsbeginn 18. Urlaubsende 31.
Das macht 14 Urlaubstage.
Auf Wiedersehen.
```

Escapesequenzen

Ein " beginnt und endet eine Zeichenkette. Will man das Symbol " selbst drucken, so muss man \" verwenden.

Das Symbol \ selbst kriegt man durch \\.

Es gibt noch andere solche Escapesequenzen, z.B. \n: Zeilenumbruch.

```
System.out.println("Die Zeichen \" und \\ erhält man
durch Vorausstellen eines \\.\n Das war’s.");
```

Vergleich mit OCAML

Java ist eine *imperative Sprache*:

Ein Programm besteht aus einer Folge von Statements, die den Programmzustand verändern.

Statements können aber auch Ausdrücke enthalten, wie `31-18+1`.

In OCAML entsprechen Statements Ausdrücke vom Typ `unit`. Man verwendet Statements in OCAML nur relativ selten.

Auf der anderen Seite unterstützt die OCAML Syntax die Bildung komplexer Ausdrücke, z.B. durch `let` und `if`.

Klassen und Objekte

Objekte enthalten Werte und Methoden (um aus diesen Werten Ergebnisse zu berechnen *und* um diese Werte zu verändern).

Klassen dienen als Muster für Objekte.

Die Klasse spezifiziert die Formate der Werte, und definiert die Methoden.

Beispiel: die eingebaute Klasse `Rectangle`. Der Ausdruck

```
new Rectangle(5, 10, 20, 30)
```

erzeugt ein Objekt der Klasse `Rectangle` mit linker oberer Ecke (5,10) und Breite/Höhe 20/30.

Das Statement

```
System.out.println(new Rectangle(5,10,20,30));
```

gibt das Objekt aus:

```
java.awt.Rectangle[x=5,y=10,width=20,height=30]
```

Beispielprogramm

```
import java.awt.Rectangle;

public class Rechteck
{
    public static void main(String[] args)
    {
        System.out.println("Guten Tag.");
        System.out.println(new Rectangle(5,10,20,30));
    }
}
```

Die Deklaration `import java.awt.Rectangle;` importiert den Klassennamen aus der package `java.awt`.

Alternativ kann man auch `java.awt.Rectangle` schreiben.

Programmvariablen

Durch das Statement

```
Rectangle cornflakesPackung;
```

wird eine Programmvariable (kurz Variable) des Typs `Rectangle` deklariert.

Man kann der Variablen einen Wert zuweisen durch `=`. Z.B.

```
cornflakesPackung = new Rectangle(5,10,20,30);
```

Und dann ausgeben:

```
System.out.println(cornflakesPackung);
```

Die Programmvariable enthält einen Verweis auf ein Objekt.

Schreiben wir

```
Rectangle frostiesPackung = cornflakesPackung;
```

(Beachte, Deklaration und Zuweisung gehen in einem.)

Dann ist `frostiesPackung` auch ein Verweis auf das erzeugte Objekt.

Es gibt aber nach wie vor nur eins!

Methoden

Die Klasse `Rectangle` enthält die Methode `translate` zum Verschieben eines Rechtecks. So verwenden wir sie:

```
cornflakesPackung.translate(15,25);
```

Geben wir jetzt `cornflakesPackung` aus, dann erhalten wir

```
java.awt.Rectangle[x=20,y=35,width=20,height=30]
```

Was kommt ’raus, wenn wir `frostiesPackung` ausgeben?

Antwort: Genau dasselbe, da ja `frostiesPackung` und `cornflakesPackung` auf dasselbe Objekt verweisen.

Dokumentation mit Javadoc

Mit `javadoc` können bestimmte Kommentare zur Erzeugung von HTML (mit Internet Browser lesbarer) Dokumentation verwendet werden:

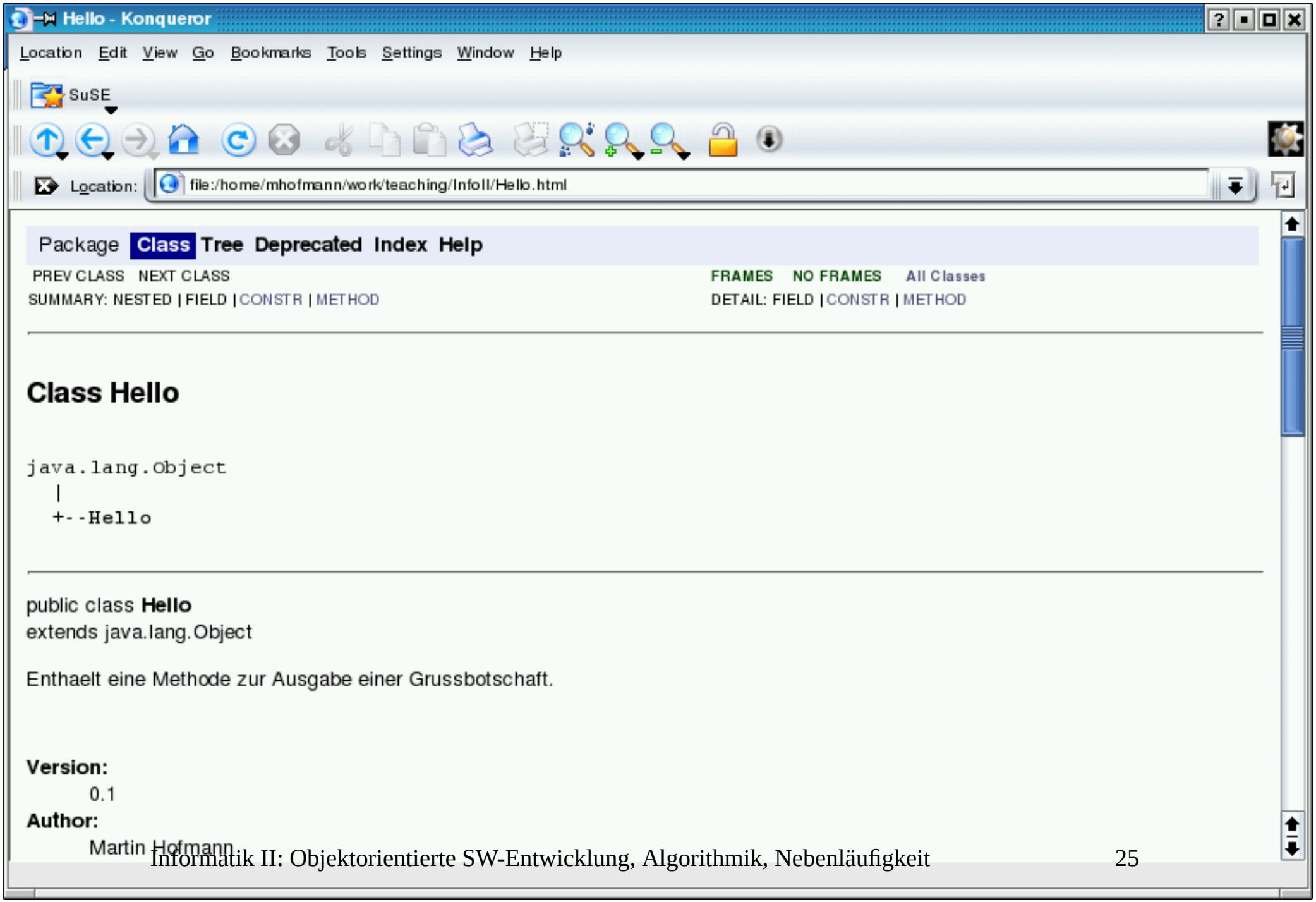
- Ein Javadoc Kommentar beginnt immer mit `/**` und endet mit `*/`.
- Zeilen innerhalb eines Javadoc Kommentars dürfen mit `*` beginnen.
- Ein Javadoc Kommentar soll immer vor einer Deklaration stehen (Klasse, Methode, ...)
- weitere Regeln: siehe Beispiele und `man javadoc`.

Der Befehl `javadoc -version -author` *Datei*`.java` erzeugt eine Datei *Datei*`.html`, die eine schön formatierte Dokumentation enthält.

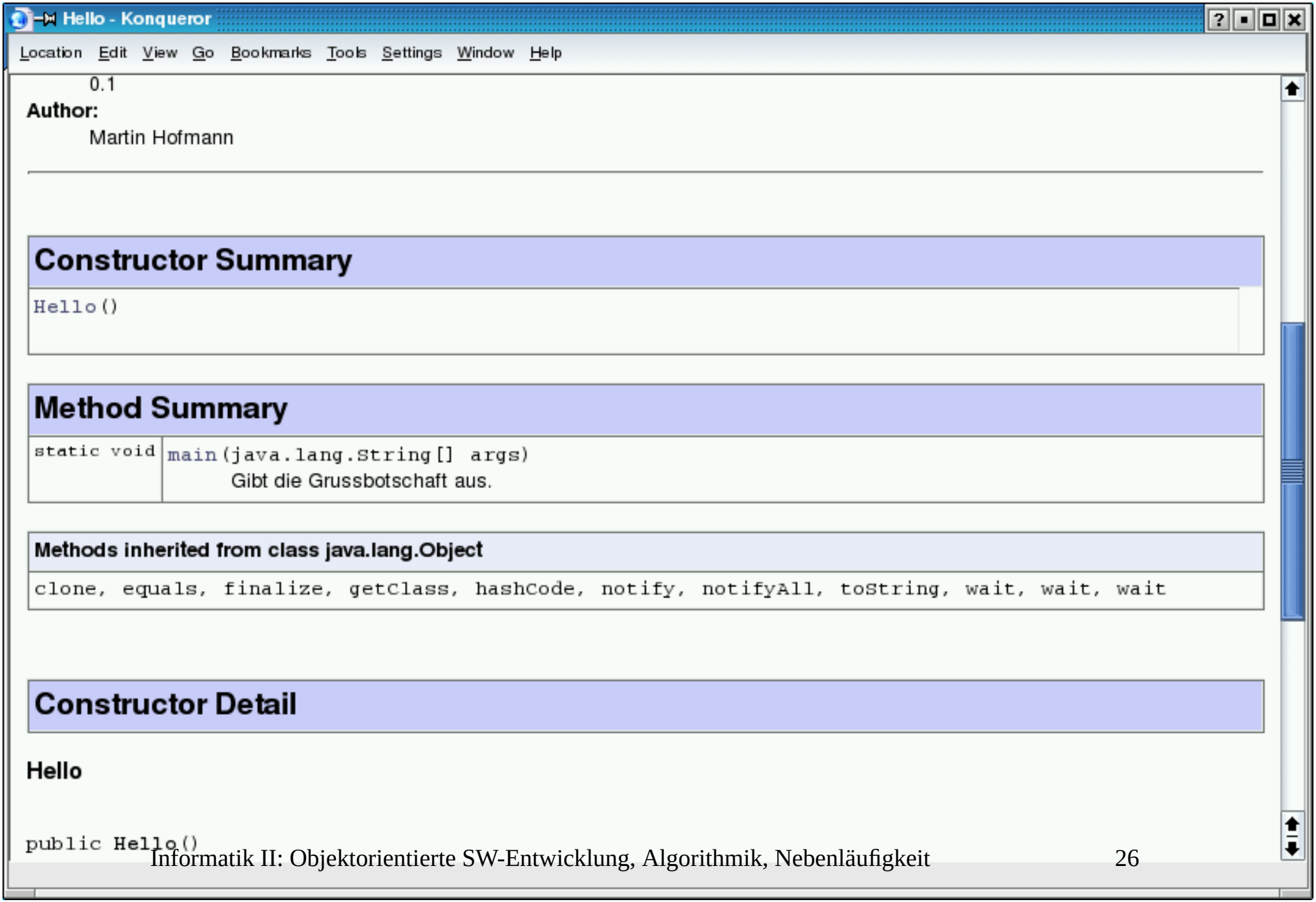
Beispiel

```
/**
 * Enthaelt eine Methode zur Ausgabe einer Grussbotschaft.
 * @author Martin Hofmann
 * @version 0.1
 */
public class Hello
{    /** Gibt die Grussbotschaft aus.
     * @param args Kommandozeilenparameter
     */
    public static void main(String[] args)
    {
        /* Hier findet die Ausgabe statt */
        System.out.println("Hello, World!");
    }
}
```

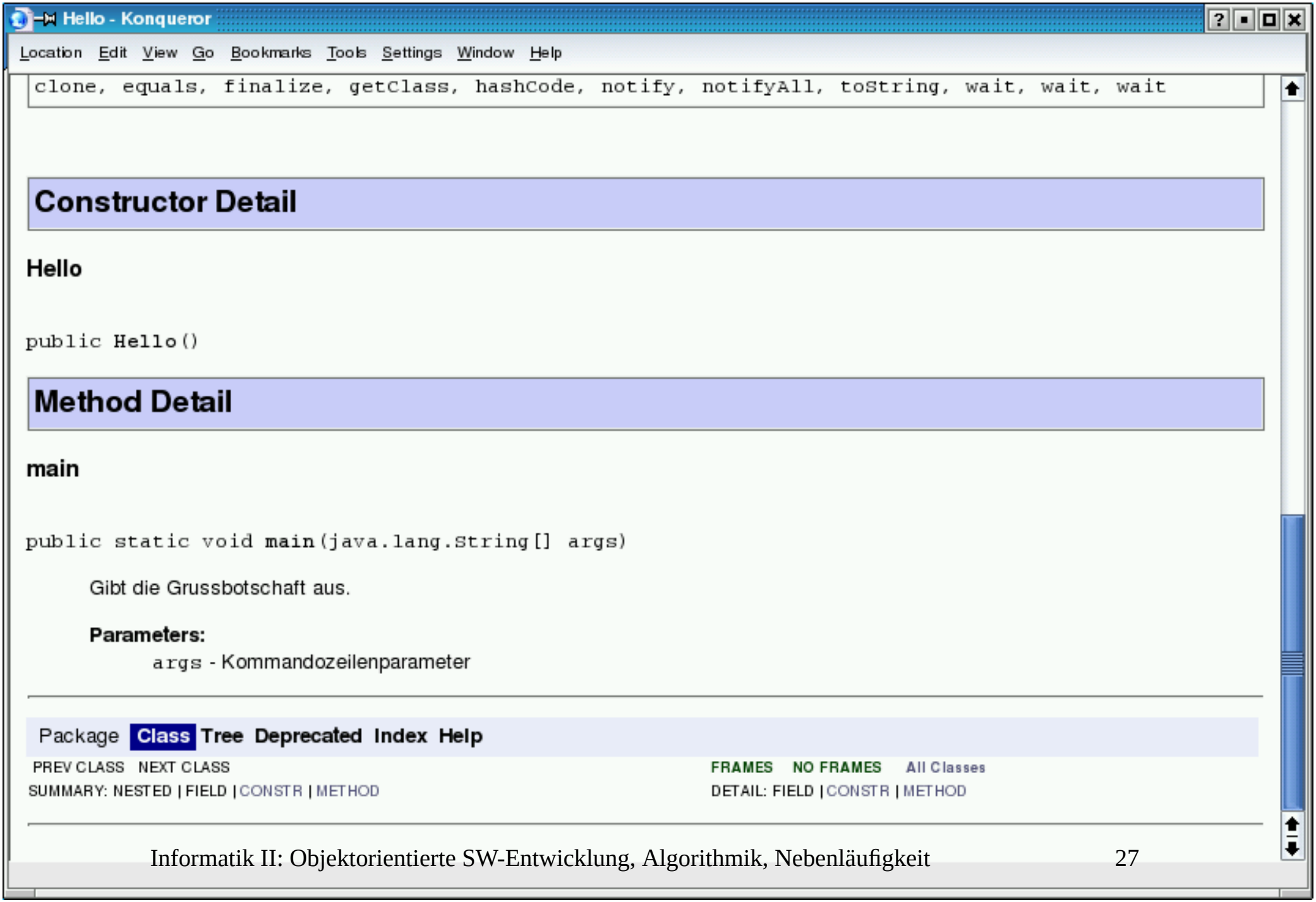
Beispiel


Hello - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
SuSE
Location: file:/home/mhofmann/work/teaching/InfoII/Hello.html
Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
Class Hello
java.lang.Object
|
+--Hello
public class Hello
extends java.lang.Object
Enthaelt eine Methode zur Ausgabe einer Grussbotschaft.
Version:
0.1
Author:
Martin Hofmann

Beispiel


Hello - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
0.1
Author:
Martin Hofmann
Constructor Summary
Hello()
Method Summary
static void
main(java.lang.String[] args)
Gibt die Grussbotschaft aus.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
Hello
public Hello()

Beispiel


Hello - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
Constructor Detail
Hello
public Hello()
Method Detail
main
public static void main(java.lang.String[] args)
Gibt die Grussbotschaft aus.
Parameters:
args - Kommandozeilenparameter
Package Class Tree Deprecated Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

Fundamentale Datentypen

- Die Datentypen int, double, String, boolean
- Variablen und Zuweisung
- Fallunterscheidungen

Münzwerte

```
public class Muenzen1
{
    public static void main(String[] args) {
        int zehnerl = 8; // Anzahl 10 ct Muenzen
        int zwanzgerl = 4; // Anzahl 20 ct Muenzen
        int fuchzgerl = 3; // Anzahl 50 ct Muenzen

        double gesamt = zehnerl * 0.10 +
            zwanzgerl * 0.20 + fuchzgerl * 0.50;

        System.out.print("Gesamtwert = ");
        System.out.println(gesamt);
    }
}
```

Die Typen `int` und `double`

Ganze Zahlen bilden den Typ `int`; Fließkommazahlen den Typ `double`.

In Java werden ints automatisch in doubles konvertiert.

Variablen

Das Statement

```
int zehnerl = 8;
```

deklariert eine ganzzahlige Variable des Typs `int` mit dem Wert 8.

Man kann in Java einer schon deklarierten Variablen neue Werte zuweisen:

```
zwanzgerl = 5;
int zw = zwanzgerl;
zw = 6;
System.out.print("Wert von \"zwanzgerl\": ");
System.out.println(zwanzgerl);
System.out.print("Wert von \"zw\": ");
System.out.println(zw);
```

Was wird gedruckt?

Antwort

```
Wert von "zwanzgerl": 5
Wert von "zw": 6
```

Der Grund ist, dass Integer- und Double-Variablen keine Verweise sind (wie Objektvariablen) sondern den jeweiligen Wert direkt enthalten.

Mit anderen Worten: eine Integer-Variable enthält einen Integer-Wert, eine Objekt-Variable enthält eine Speicheradresse (unter der sich ein Objekt befindet).

Vergleich mit OCAMLs let

Eine Variablendeklaration entspricht ziemlich genau dem let-Konstrukt von OCAML:

```
let zwanzgerl = 5 in
let zw = zwanzgerl in
let zw = 6 in (zwanzgerl,zw)
```

Der Wert ist (5,6).

Ein wichtiger Unterschied ist, dass in Java der Typ einer Variablen bei ihrer Deklaration immer explizit angegeben werden muss.

Ein weiterer wichtiger Unterschied ist, dass Wertzuweisung in Java ein Statement ist; in OCAML dagegen nur in Form des geschachtelten Let Konstrukts auftreten kann.

Auch die Deutung unterscheidet sich: in OCAML wird das zweite Vorkommen von `zw` als eine neue Variable gedeutet, die die erste überschattet.

Initialisierung

Man muss Variablen nicht initialisieren:

```
int a;
int b = 4;
a = b + 2;
```

Sie müssen aber vor der ersten Verwendung einen Wert bekommen:

```
int a;
int b = 4;
System.out.println(a);
```

ist ein Programmierfehler (den Java schon beim Compilieren erkennt.)

Nochmal Wertzuweisung

Man kann auch schreiben:

```
a = a + 1;
```

Dadurch wird der Wert von `a` um eins erhöht.

Manche Programmierer verwenden dafür die Kurzform

```
a++;
```

Daher auch der Name C++ für „Nachfolger von C“.

Rechengenauigkeit

Wie in OCAML gibt es nur 2^{32} verschiedene int-Werte. Ebenso hat `double` begrenzte Genauigkeit.

Es gibt den Datentyp `long`, der insgesamt 2^{64} verschiedene Werte aufweist. Long-Konstanten schreibt man so: `1340000000000000L`.

Schließlich gibt es die Klassen `BigInteger` und `BigDecimal` die praktisch unlimitiert sind. Sie repräsentieren Zahlen als Listen von Ziffern.

Nachteil: Langsam und umständlich zu benutzen.

```
import java.math.BigInteger;

...

BigInteger b = new BigInteger("100000000");
b = b.multiply(b) ; b = b.multiply(b) ; b = b.multiply(b);
b = b.subtract(new BigInteger("1"));
System.out.println(b);
```

Rechengenauigkeit

Druckt:
99999999999999999999999999999999999999999999999999999999999999999999999999999

Typkonversion

```
int euros = 2;
double gesamt = euros; // ok

double euros = 2.0;
int anzahlEuros = euros; // geht nicht
```

Im ersten Beispiel wird der Integer-Wert automatisch in Double konvertiert.

Im zweiten Beispiel nicht.

Typkonversion

Man kann aber schreiben:

```
double euros = 2.50;
int anzahlEuros = (int)euros;
System.out.println(anzahlEuros);
```

Das ist eine explizite Typkonversion (*typecast*).

Hier werden einfach alle Dezimalstellen abgeschnitten.

Will man runden, so verwende man

```
double a = 3.759;
System.out.println((int)Math.round(a));
```

Die (statische) Methode `Math.round` berechnet den nächstgelegenen ganzzahligen Double-Wert.

Rundungsfehler

```
double f = 4.35;
int n = (int)(100 * f);
System.out.print(n);
```

Druckt 434.

Grund: In Binärdarstellung ist 4,35 ein echt periodischer Bruch.

```
(int)Math.round(100 * f);
```

hat Wert 435.

Konstanten

```
int flaschen = 3;
int dosen = 5;
int mengeFanta = flaschen * 0.5 + dosen * 0.33;
```

ist unschön, da `0.5` und `0.33` einfach so dastehen.

Besser:

```
final double FLASCHEN_INHALT = 0.5;
final double DOSEN_INHALT = 0.33;
int mengeFanta = flaschen * FLASCHEN_INHALT + dosen * DOSEN_
```

Werte, die mit `final` deklariert werden, können nur einmal initialisiert und danach nicht mehr verändert werden.

Vorteil gegenüber Variablen: Effizienz + Dokumentation.

Numerische Konstanten wie `0.5` mitten im Programm sind schlechter Stil.

Vordefinierte Double-Konstanten: `Math.PI` und `Math.E`.

Arithmetik

Plus `+` und Mal `*` hatten wir schon.

Division wird als `/` notiert.

Vorsicht: Sind beide Operanden von `/` Integers, so wird abgerundet.

```
int s1 = 5;
int s2 = 6;
int s3 = 3;
double mittelwert = (s1 + s2 + s3) / 3;
```

`mittelwert` hat den Wert `4` (statt `4.666666...`)

Beliebter Programmierfehler.

Richtig:

```
double mittelwert = (s1 + s2 + s3) / 3.0;
```

Was gibt’s sonst noch

- Die „Punkt vor Strich“ Regel gilt.
- Mathematische Funktionen sind in der Klasse `Math` definiert.
- Automatische Typkonversionen erfolgen von innen nach außen.

Beispiel: Die „Lösungsformel“:

```
double a;
double b;
double c;
/* Wertzuweisung */
x1 = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);
x2 = (-b - Math.sqrt(b * b - 4 * a * c)) / (2 * a);
```

Ebenso: `a * a + b * b - 2 * a * b * Math.sin(phi);`

Zeichenketten

Der Datentyp `String` besteht aus Zeichenketten, d.h. Folgen von Buchstaben und Sonderzeichen.

```
String name = "Matthias";
       name = "Johanna";
System.out.println(name);
```

Druckt: `Johanna`.

```
int n = name.length();
```

Die Variable `n` hat den Wert 7.

Teile einer Zeichenkette

Der Ausdruck

```
s.substring(anfang, endePlusEins)
```

bezeichnet die Teilzeichenkette von `s` angefangen vom Zeichen an der Position `anfang` bis (ausschließlich) zum Zeichen an der Position `endePlusEins`.

Positionen beginnen immer bei Null.

```
String s = "Hello, World!";
String sub1 = s.substring(0,5);
String sub2 = s.substring(4,8);
```

Was sind die Werte von `sub1` und `sub2`?

Welcher `substring`-Ausdruck hat den Wert `World` ?

Teile einer Zeichenkette

Antwort: `sub1` den Wert `Hello` und `sub2` den Wert `o, W`. Der Ausdruck `s.substring(7, 12)` hat den Wert `World`.

Will man alle Zeichen von `anfang` bis zum Ende der Zeichenkette, dann kann man

```
s.substring(anfang, s.length())
```

schreiben. Das letzte Zeichen hat nämlich die Position `s.length()` – 1.

Eine Kurzform dafür ist `s.substring(anfang)`.

Fehlerbehandlung

Ruft man `s.substring` mit unpassenden Argumenten auf, so gibt es einen Fehler. Z.B.: `s.substring(4,30)` führt zu:

```
Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
String index out of range: 20
        at java.lang.String.substring(String.java:1473)
        at Namen.main(Namen.java:5)
```

Man sagt: der Ausdruck wirft eine Ausnahme (*throws an exception*).

Es ist möglich, so eine Ausnahme im Programm “aufzufangen” und benutzerdefinierte Befehle auszuführen, z.B. eine ordentliche Fehlermeldung.

Noch besser ist es, das Auftreten solcher Ausnahmen von vornherein zu vermeiden.

Verkettung

Zeichenketten kann man konkatenieren.

In Java verwendet man dafür das Pluszeichen (vgl. OCAML: ^)

Der Ausdruck `"Matthias" + "Johanna"` hat den Wert `MatthiasJohanna`.

Der Ausdruck

```
"Euro" + "s".substring(0,n)
```

hat den Wert `Euro` oder `Euros` ja nachdem, ob `n` gleich 0 oder 1 ist. Alle anderen Werte von `n` sind nicht erlaubt.

Was “sind” Zeichenketten?

Eine Zeichenkette ist ein Objekt.

Es versteht u.a. die Methoden `length` und `substring`.

Die Methode `length` liefert die Länge zurück.

Die Methode `substring` ein neues String-Objekt, das den jeweiligen Teilstring enthält.

Man kann eine Zeichenkette nie verändern (im Gegensatz zu veränderlichen Objekten wie `Rectangles`).

Im Computer ist ein String eine Speicheradresse. Unter dieser Adresse befindet sich die Länge, z.B. n. In den n darauffolgenden Speicherstellen befinden sich dann die Zeichen.

Wie in OCAML gibt es in Java keine Möglichkeit, diese Zeichen oder die Länge zu verändern, obwohl die Maschinensprache das im Prinzip zuließe.

Verkettung mit Zahlwerten

```
double betrag = 34.99;
int nummerMahnung = 2;
String anweisung = nummerMahnung + ". Mahnung: " +
        "Bitte zahlen Sie " + betrag + " EUR.";

System.out.println(anweisung);
```

Druckt: `2. Mahnung: Bitte zahlen Sie 34.99 EUR.`

Verkettung mit Zahlwerten

Ist ein Operand von + eine Zeichenkette, so wird der andere automatisch in eine Zeichenkette umgewandelt. Das ist keine Typkonversion:

```
String betrag = 34.99 * 2;
```

löst aus:

```
incompatible types
found   : double
required: java.lang.String
        String betrag = 34.99 * 2;
```

Vielmehr hat das +-Zeichen je nach Typ der Operanden leicht unterschiedliche Bedeutung. Man spricht von *overloading* (Überladung).

Verkettung mit Zahlwerten

Man kann schreiben `""+x` um `x` in eine Zeichenkette umzuwandeln.

Aber Vorsicht:

```
String anweisung = nummerMahnung + ". Mahnung: " +
    "Bitte zahlen Sie " + betrag/3 + " EUR.";
System.out.println(anweisung);
```

Ergebnis:

```
2. Mahnung: Bitte zahlen Sie 11.663333333333334 EUR.
```

Abhilfe: Formatierte Ausgabe

```
import java.text.NumberFormat;

...

NumberFormat formatierer = NumberFormat.getNumberInstance();

formatierer.setMaximumFractionDigits(2);
formatierer.setMinimumFractionDigits(2);

double betrag = 34.99;
int nummerMahnung = 2;
String anweisung = nummerMahnung + ". Mahnung: " +
   "Bitte zahlen Sie " + formatierer.format(betrag/3) + " EU
```

Ergebnis:

```
2. Mahnung: Bitte zahlen Sie 11,66 EUR.
```

Sogar Tausenderpunkte werden eingesetzt, z.B.: `1.192.279,33 EUR`.

Benutzernamen

Wir möchten aus dem ersten und letzten Buchstaben des Namens und einer laufenden Nummer einen Benutzernamen erzeugen:

```
String name = "Johanna";
int lfdNo = 1728;
```

Der Benutzername sollte `Ja1728` sein.

Kein Problem

```
String benutzerName;
benutzerName = name.substring(0,1) +
     name.substring(name.length() - 1) + lfdNo;
```

Parsing von Zeichenketten

Wie erhalten wir aus einem Benutzernamen die laufende Nummer?

```
benutzerName.substring(2)
```

enthält zwar die Ziffern der lfd. Nr. ist aber immer noch ein String.

Die Lösung:

```
int nummer = Integer.parseInt(benutzerName.substring(2));
```

`parseInt` ist eine statische Methode der Klasse `Integer` und dient dazu, eine Zeichenkette in einen integer umzuwandeln.

Statische Methoden werden nicht an ein Objekt geschickt, sondern können "einfach so" ausgeführt werden.

Details später.

Parsing von Doubles

...geht analog mit `Double.parseDouble`, z.B.:

```
double c = Double.parseDouble("2.97E9"); /* oder so */
```

Aber Vorsicht: eine “deutsche” Zahl, wie 1.234,59 kann `parseDouble` nicht verarbeiten.

Wie das geht, siehe Java Reference Manual

Fallunterscheidungen

Oft will man ein Statement nur dann ausführen, wenn eine bestimmte Bedingung gilt. Das geht mit dem `if` Statement:

```
if (zinsSatz > 100.0) {
     System.out.println("Fehler.");
} else
     rate = restschuld * zinsSatz/100.0 / 12.0 + tilgung;
```

Bemerkung: Ausgaben an `System.out` sind dilettantisch.

Professionellere Lösung

```
if (zinsSatz > 100.0) {
     Fehlerbearbeitung.fehler(falscherZinsSatz);
} else
     rate = restschuld * zinsSatz/100.0 / 12.0 + tilgung;
```

- `Fehlerbearbeitung.fehler` ist eine benutzerdefinierte Methode, die Fehlerobjekte anzeigt und entsprechend verfährt.
- Ein (benutzerdefiniertes) Fehlerobjekt (hier `falscherZinsSatz`) beinhaltet den Anzeigetext (üblicherweise in verschiedenen Sprachen) und Instruktionen wie bei seinem Auftreten zu verfahren ist.

Bedingungen

. . . stehen immer in Klammern und sind von der Form x_1 *op* x_2 wobei

- *op* ist `<, >, <=, >=, ==`
- Die Operanden sind Zahlen des gleichen Typs, evtl. wird implizite Typkonversion vorgenommen.
- Später lernen wir noch andere Bedingungen kennen.

Vorsicht beim Testen von Doubles auf Gleichheit. Besser

```
double final EPSILON = 1E-10; // zum Beispiel
if ( Math.abs(x - y) <= EPSILON )
```

Die Klammern

Formal steht nach der Bedingung ein Statement.

Braucht man mehrere, so fasst man sie mit geschweiften Klammern (`{}`) zu einem Block zusammen.

Das ganze `if-else` Konstrukt wird als ein einziges Statement aufgefasst, ein zusammengesetztes Statement.

Man kann den `else` Teil auch weglassen.

Was ist hier falsch?

```
if (betrag <= kontostand)
     double neuerKontostand = kontostand - betrag;
     kontostand = neuerKontostand;
```

Antwort

Die geschweiften Klammern fehlen.

Einrückungen dienen der Lesbarkeit werden aber vom Compiler ignoriert.

Richtig:

```
if (betrag <= kontostand) {
     double neuerKontostand = kontostand - betrag;
     kontostand = neuerKontostand;
}
```

Oder so:

```
if (betrag <= kontostand)
{
     double neuerKontostand = kontostand - betrag;
     kontostand = neuerKontostand;
}
```

Mehrere if-Statements

Natürlich können in den Zweigen eines if-Statements wiederum solche stehen:

```
if ( richter >= 8.0 )
   s = "Grosse Verwuestung";
else if ( richter >= 7.0 )
   s = "Viele Gebaeude zerstoert";
else if ( richter >= 6.0 )
   s = "Viele Gebaeude beschaedigt";
else if ( richter >= 4.5 )

...
```

Dangling else

Ein `else` bezieht sich immer auf das nächstgelegene `if`.

Ungeachtet der Einrückung.

Wir wollen aus `int a, b, c` as größte und das kleinste berechnen.

Maximum und Minimum

```
if (a >= b)
    if (a >= c) {
        max = a;
        if (c >= b)
            min = b;
        else
            min = c;
    } else {
        max = c; min = b;
    }
else
```

Maximum und Minimum

```
if (b >= c) {
    max = b;
    if (c >= a)
        min = b;
    else
        min = c;
} else {
    max = c; min = a;
}
```

Der Datentyp `boolean`

```
int x = 5;
System.out.println(x < 10);
```

Gibt aus: `true`

In Java ist `x < 10` ein Ausdruck des Typs `boolean`, genauso wie `x + 12` einer vom Typ `int` ist.

Werte des Typs `boolean` sind (nur) `true` und `false`.

Logische Verknüpfung

Auf dem Typ `boolean` sind die zweistelligen Verknüpfungen && und || erklärt:

- e_1 && e_2 bedeutet: "e_1 und e_2"; "sowohl e_1, als auch e_2"; "e_1 und e_2 beide `true`".
- e_1 || e_2 bedeutet: "e_1 oder e_2"; "mindestens eins von beiden, e_1 oder e_2, ist `true`".

Außerdem gibt es die einstellige Operation `!`:

- `!`e bedeutet: "nicht e", "das Gegenteil von e".

Beachte: wie in OCAML wird bei e_1 || e_2 zunächst e_1 ausgewertet. Ist das Ergebnis `true`, so wird e_2 gar nicht erst ausgewertet. Analog für &&.

Anwendung

Johannas Geburtstag ist der 21.12.

Angenommen, wir haben `int`-Variablen `day` und `month`.

Welcher Boole’sche Ausdruck ist `true` genau dann, wenn die Werte der beiden Variablen Johannas Geburtstag entsprechen?

Antwort

Der Ausdruck

```
day == 21 && month == 12
```

So können wir ihn verwenden:

```
if (day == 21 && month == 12)
     System.out.println("Happy birthday, Johanna!");
```

Anwendung

Wie drücken wir aus, dass die `double` Variable `heat` zwischen `100.0` und `120.0` liegt?

Antwort

```
100.0 <= heat && heat <= 120.0
```

Äquivalent ist

```
!(heat < 100.) && !(heat > 120.)
```

Äquivalent ist auch

```
!(heat < 100. || heat > 120.)
```

De Morgan’s Gesetz:

```
!(a && b)  =  !a || !b
!(a || b)  =  !a && !b
```

Andere Boole’sche Ausdrücke

Methoden können einen `boolean` Wert zurückliefern:

Die Klasse `String` enthält die Methode `equals`, die einen `String` Parameter hat und einen `boolean` zurückliefert.

```
String answer;

System.out.println("Koennen Sie mir helfen?");
/* Eingabe von answer */
if (answer.equals("ja") || answer.equals("Ja")) {
    System.out.println("Danke!");
} else {
    System.out.println("Schade.");
}
```

Stringvergleiche

Es gibt auch die Methode `equalsIgnoreCase`, die Groß/Kleinschreibung ignoriert.

```
if (antwort.equalsIgnoreCase("ja"))
   System.out.println("Danke!");
```

Zum Vergleich von Strings soll man nicht `==` verwenden.

```
String x = "a";
System.out.println("ja" == "ja");
System.out.println("ja" == "j" + "a");
System.out.println("ja" == "j" + x);
```

Gibt aus:

```
true true false
```

Grund: `==` bezeichnet hier Identität, d.h. Repräsentierung an derselben Speicherstelle.

Internalisierung

Der Inhalt dieser Folie ist nicht prüfungsrelevant.

```
String x = "a";
System.out.println("ja" == "ja");
System.out.println("ja" == "j" + "a");
System.out.println("ja" == ("j" + x).intern());
```

Gibt aus:

```
true true true
```

Grund: die Methode `intern` schaut nach, ob ein identischer String schon vorhanden ist und gibt ggf. diesen zurück.

Stringvergleiche

Die Methode `compareTo` vergleicht nach der lexikographischen Ordnung, liefert aber einen `int` zurück:

s_1`.compareTo(`s_2`)` ist

- < 0, wenn s_1 alphabetisch vor s_2 kommt
- $= 0$, wenn s_1 und s_2 gleich sind
- > 0, wenn s_1 alphabetisch nach s_2 kommt.

Beispiele:

`"AAAaaaaaa".compareTo("meinSchluesseldienst")` ist < 0

`"Vorlesung".compareTo("Vorlesen")` ist > 0

Boole’sche Variablen

Man kann auch Variablen des Typs `boolean` deklarieren:

```
boolean mitBedienung;
boolean tischGedeckt;
boolean draussen;

/* Initialisierung von draussen und tischGedeckt */

mitBedienung = !draussen || tischGedeckt;

if (mitBedienung)
   System.out.println("Hier keine Selbstbedienung!");
```

Übungen

Was ist an den folgenden Statements falsch ?

```
if cents > 0 then System.out.println(cents + " Cents");

if (1 + x > Math.pow(x, Math.sqrt(2)) y = y +x;

if (x = 1) y++; else if (x = 2) y = y + 2;

if (x && y == 0) p = new Point(x,y);

if (1 <= x <= 10) {System.out.println("Danke.");}

if (!antwort.equalsIgnoreCase("Ja ") ||
            !antwort.equalsIgnoreCase("Nein"))
    System.out.println("Antworten Sie mit Ja oder Nein.");
```

Schaltjahre

Jedes vierte Jahr ist ein Schaltjahr, es sei denn, die Jahreszahl ist durch hundert teilbar. In diesem Fall liegt ein Schaltjahr nur vor, wenn die Jahreszahl durch 400 teilbar ist.

Beispiel: 2000 war ein Schaltjahr, 1900 war keins.

Man schreibe einen Boole'schen Ausdruck, der `true` ist, genau dann wenn `jahr` ein Schaltjahr ist.

Hilfe: `x % y` ist der Rest der ganzzahligen Division von `x` durch `y`. Z.B.: `12%5`=2.

Lösung

```
jahr % 4 == 0 && !(jahr % 100) == 0 || jahr % 400 == 0
```