Programmieren mit

Java

Lars Knipping

Februar 2002

Java nach Sun

Java ist eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutral portable, hochleistungsfähige, multithread-fähige und dynamische Sprache.

Sun Microsystems Inc. über Java (frei übersetzt)

Ausgewählte Eigenschaften

- *objektorientiert (OO)*
- *einfach*
  kleiner Sprachumfang, umfangreiche Bibliotheken
- *interpretiert, architekturneutral portabel*
  „*write once, run anywhere*“
  - Compiler erzeugt Bytecode
  - Bytecode wird von Virtueller Maschine (JVM) interpretiert
- *robust und sicher*
  - Ausnahmebehandlung (*exceptions*)
  - Keine Zeigerarithmetik
  - Autom.Speicherverwaltung (*garbage collection*)
  - Sog. Sandkastenprinzip für Applets
  - ...

Werkzeuge

- JRE (Java Runtime Environment)
  Java Bytecode Interpreter

- JDK (Java Development Kit)/Java SDK (Software Development Kit)
  - *java*, *jre*: Java Bytecode Interpreter
  - *javac*: Java Compiler (Übersetzer)
  - *javadoc*: Generator für API Dokumentationen
  - *appletviewer*
  - ...

Gratis ...

- JDK /Java SDK von Sun
- JDK von IBM
- Kaffe von TransVirtual (GPL)
- Jikes von IBM, Compiler (Open Source)
- guavac, Compiler (GPL)
- Japhar, Java Bytecode Interpreter (LGPL)

IDEs

- Borland JBuilder
- Forte for Java von NetBeans.com/Sun (Open Source)

  `http://www.sun.com/forte/ffj/`
  `http://www.netbeans.org`

- IBM Visual Age for Java
- WebGain Visual Cafe
- Sybase PowerJ

Hello, World!

```
public class HelloWorld {

  public static void main(String[] argv) {

    System.out.println("Hello, world!");

  }
}
```

Hello?

- `public class HelloWorld { ... }`

  definiert die *Klasse* mit Namen *HelloWorld*

  in Klammern: was zu der Klasse gehört

- `public static void main(String[] argv) { ... }`

  Einsprungspunkt beim Ausführen der Klasse

  Im Innern: auszuführende *Anweisungen* (*statements*)

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

  Anweisung, schreibt *Hello, world!*

  Anweisungen werden mit Semikolon abgeschlossen

No News is Good News

- Mit Texteditor **HelloWorld.java** erstellen

```
public class HelloWorld {
  public static void main(String[] argv) {
    System.out.println("Hello, world!");
  }
}
```

- Übersetzen mit $>$ **javac HelloWorld.java**

no news is good news

- Ausführen

```
> java HelloWorld
Hello, world!
>
```

Anweisungssequenz

```
public class HelloGoodbye {

  public static void main(String[] argv) {
    System.out.println("You say yes, I say no");
    System.out.println("You say stop,");
    System.out.println("  I say go, go, go");
    System.out.println("Oh no");
  }
}
```

It was hard to write, so it should be hard to read

- `//` Kommentar bis Zeilenende
- `/*` und `*/` klammern Kommentar, nicht schachtelbar
- `/**` und `*/` wie oben, Inhalt für *javadoc*

```
/**
 * Gibt Anfang von <I>Hello Goodbye</I> aus.
 * @author Lars (Programm, nicht Lied)
 */
public class HelloGoodbye { //heisst wie Lied
  /** Der Einsprungspunkt ins Programm. */
  public static void main(String[] argv) {
    System.out.println("You say yes, I say no");
    System.out.println("You say stop,");
    System.out.println("  I say go, go, go");
    System.out.println("Oh no"); /* Ende... */
  }
}
```

Variablen

Variablen (*variables*) bezeichnen Speicherplätze („Behälter “) für Werte eines Datentyps.

Variablen werden vor Verwendung deklariert: Festlegung von Name (Bezeichner, *identifier*) und Datentyp.

```
// Integer (ganze Zahl) mit Namen x:
int x; // Deklarationen sind Anweisungen
```

Zuweisungen

Variablenwerte können mit Zuweisungsanweisung gesetzt werden

```
int x;
x = 5;
int y;
y = x;
x = 7;
System.out.println(x); // Ausgabe: 7
System.out.println(y); // Ausgabe: 5
```

Mehr Deklarationen

Bei Deklaration kann Variable initialisiert werden

```
int x = 42;
```

Mehrere Variablen gleichen Typs können mit einer Anweisung deklariert werden

```
int i=23, j=5, k, l, m=-7;
```

Dreieckstausch

Vertauschen zweier Variablen mittels einer Hilfsvariable

```
// i, j Integervariablen
int h = i;
i = j;
j = h;
```

Datentypen

- einfache (in Java auch *Primitive Datentypen*)
  - Wert unteilbar
  - kann nur als ganzes manipuliert werden
- zusammengesetzte
  - mehrere, einzeln manipulierbare Teilwerte
  - rekursives Bauprinzip

Primitive Datentypen

Typ	Bits	Werte
Ganze Zahlen		
`byte`	8*	-128 bis 127
`short`	16*	-32768 bis 32767
`int`	32	-2^{31} bis $2^{31}-1$
`long`	64	-2^{63} bis $2^{63}-1$
Fließkommazahlen (IEEE 754 single/double precision floating points)		
`float`	32	ca. $-3,4*10^{38}$ bis $3,4*10^{38}$ Betragsmin. $\neq 0$ ca. $\pm 1,4*10^{-45}$
`double`	64	ca. $-3,4*10^{38}$ bis $3,4*10^{38}$ Betragsmin. $\neq 0$ ca. $\pm 4,9*10^{-324}$
Logikwerte		
`boolean`	1*	`true` oder `false`
Buchstaben (Unicode)		
`char`	16	`\u0000` bis `\uFFFF`

Schall und Rauch

In Bezeichnern (*identifier*) erlaubte Zeichen

- Buchstaben (*java letters*, u.a. `'a'`-`'z'` und `'A'`-`'Z'`)
- Unterstrich `'_'`
- Ziffern `'0'`-`'9'`
- `'$'` (historisch - sollte nicht verwendet werden)

Erstes Zeichen muß ein *Java Letter* sein.

Literale (literals) I

- Integer
  - dezimal, Bsp.: `23`, `-15`
  - oktal, beginnt mit `0`
    Bsp.: `023`, `-017`
  - hexadezimal, beginnt mit `0x`
    Bsp.: `0x17`, `-0xF`

- Long
  wie Integer, aber mit angehängtem `l`/`L`
  Bsp: `23L`

- Double
  Dezimalpunkt, Exponent `e`/`E` oder angehängtes `d`/`D`
  Bsp: `1.39e-47` ($= 1,39 * 10^{-47}$)

- Float
  angehängtes `f`/`F`, Dezimalpunkt/Exponent optional
  Bsp: `1.39e-47f`

Literale II

- Char
  Zeichen in einfachen Anführungszeichen
  - druckbare Zeichen
    Bsp.: `'K'`
  - Escapesequenzen

`'\b'`	backspace
`'\t'`	(horizontaler) Tabulator
`'\n'`	Zeilenvorschub (newline)
`'\f'`	Seitenvorschub (form feed)
`'\r'`	Wagenrücklauf (carriage return)
`'\''`	Anführungszeichen
`'\"'`	doppeltes Anführungszeichen
`'\\'`	backslash

  - Oktalcodes `'\000'` bis `'\377'`
  - Hexcode `'\u0000'` bis `'\uFFFF'`

Literale III

- Boolean

  `true`, `false`

- String

  Zeichenfolge in doppelten Anführungszeichen

  `"\114ars \u004Bnipping\n"`

- Objektreferenz

  `null`

Operatoren und Ausdrücke

Ausdrücke:

- mit Operatoren verknüpfte Teilausdrücke
- geklammerte Ausdrücke
- Variablen
- Literale

```
5*(y+1)
```

Arithmetische Operatoren

operieren auf Zahlen

- binäre

  `+`, `-`, `*`, `/` und `%` (Modulo, Divisionsrest)

```
int x = 7 / 2;         // ergibt 3
double y = 7.0 / 2.0; // ergibt 3.5
```

- unäre
  - Vorzeichen `+` und `-`
  - `++` (Prä-/Postinkrement, nur auf Variablen)
  - `--` (Prä-/Postdekrement, nur auf Variablen)

```
int i = 5;
int j = +i;  // j = 5
int k = i++; // k = 5, i = 6
int l = ++i; // l = 7, i = 7
```

Vergleichsoperatoren

liefern `boolean`-Wert

- gleich `==` und ungleich `!=`

  auf bel. Datentypen

- Vergleiche `<`, `<=`, `>`, `>=`

  auf arithmetischen Typen (Zahlen und `char`s)

- `instanceof` Typüberprüfung für Objekte

Boolsche Operatoren

operieren auf `boolean`-Werten

- `!` Nicht (*NOT*, unär)
- `&&` bedingtes Und, `||` bedingtes Oder
- `&` Und, `|` Oder, $\wedge$ Exklusiv Oder (*XOR*)

```
// erhoeht i:
boolean b1 = true | (++i == 5);
// veraendert i nicht:
boolean b2 = true || (++i == 5);
```

Bitoperatoren

operieren auf *integralen Typen* (`byte`, `short`, `int`, `long`, `char`)

- $\sim$ bitweises Komplement (unär)
- `&` bitweises Und, `|` bitweises Oder, $\wedge$ bitweises *XOR*
- `<<` Linksshift, `>>` arithm. Rechtsshift, `>>>` logischer Rechtsshift

```
int i = 25 >> 2; // ergibt 6
```

Zuweisungsoperatoren

- `=` Zuweisung
- Zuweisung mit Operation

  `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=`, `>>>=`

  „*var op*`=` *expr* “entspricht „*var* `=` *var op expr* “

  ```
  i *= 2; // verdoppelt i
  ```

liefern zugewiesenen Wert zurück

```
i = j = 7;
k = 5 + 6*(i += j/2);
```

Operatoren und kein Ende ...

- `(`*type*`)` „*cast*“ auf Datentyp

  explizite Typumwandlung

```
short s = (short) 157;
int i = 255;
byte b = (byte) i; // => b = -1
```

- `? :` Ternärer Auswahloperator

```
max = (i>j) ? i : j;
```

- `+` Aneinanderhängen von Strings

```
"Hello, "+ "world!"
```

Implizite Typumwandlung I

- „*widening* “

  $$\left.\begin{matrix}\texttt{byte} \\ \texttt{short} \\ \texttt{char}\end{matrix}\right\} \longrightarrow \texttt{int} \longrightarrow \texttt{long}$$

  $$\left.\begin{matrix}\texttt{char} \\ \texttt{byte} \\ \texttt{short} \\ \texttt{int} \\ \texttt{long}\end{matrix}\right\} \longrightarrow \left\{\begin{matrix}\texttt{float} \\ \texttt{double}\end{matrix}\right.$$

  $$\texttt{float} \longrightarrow \texttt{double}$$

  Bsp.: `5/2.0 // => double 2.5`

- Konst. `int`-Wert einem `byte`, `short` oder `char` zuweisen (wenn ohne Über-/Unterlauf)

  Bsp.: `byte b = 5; // kein cast noetig`

Implizite Typumwandlung II

- Umwandlung nach `String`

```
int i = 5;
System.out.println("i ist "+i);
```