

Einführung in die Programmierung mit Java

Dozent: Prof. Dr. Michael Eichberg

Kontakt: michael.eichberg@dhbw-mannheim.de, Raum 149B

Version: 1.0

Die Folien sind teilweise inspiriert von oder basierend auf Lehrmaterial von Prof. Dr. Michael Matt.



1

Folien: <https://delors.github.io/prog-java-basics/folien.de.rst.html>

<https://delors.github.io/prog-java-basics/folien.de.rst.html.pdf>

Fehler melden:

<https://github.com/Delors/delors.github.io/issues>

1. EINFÜHRUNG

"Hello World" - das erste Java-Programm^[1]

```
void main(){  
    println("Hello World!");  
}
```

[1] Die Datei *HelloWorld.java* kann [hier](#) heruntergeladen werden und mit `java --enable-preview HelloWorld.java` ausgeführt werden.

3

Die Datei enthält ein einfaches Java-Programm, das den Text **Hello World!** auf der Konsole ausgibt.

In der ersten Zeile wird die Methode `main` definiert. Diese die Einstiegsmethode in das Programm. Der Text **Hello World!** wird mit der Methode `println` auf der Konsole ausgegeben. Die Methoden `print`, und `println` sind in Java Skripten immer verfügbar und geben den übergebenen Text auf der Konsole aus (ohne bzw. mit Zeilenumbruch am Ende).

Von der Konsole lesen[2]

```
void main() {  
    println("Hello "+ readln("What is your name? "));  
}
```

[2] HelloWorld.java

4

Mit Hilfe von `readln` können Sie von der Konsole lesen. In Java Skripten ist `readln` immer verfügbar. Das Programm gibt den Text **Hello** gefolgt von dem eingegebenen Text aus. Die Methode `readln` gibt erst den übergebenen String aus und liest dann eine Zeile von der Konsole ein. Der eingelesene Text wird dann an das Wort "Hello " angehängt (mittels des "+" Operators) und als ganzes zurückgegeben.

Lesen von und Schreiben auf die Konsole

Schreiben Sie ein Java-Programm (**GutenMorgen.java**), das erst nach dem Namen des Nutzers **X** fragt und dann **Guten Morgen X!** auf der Konsole ausgibt. Beachten Sie dabei, dass der Text **X** durch den eingegebenen Namen ersetzt wird und am Ende ein Ausrufezeichen steht.

Als zweites soll das selbe Programm dann nach dem Wohnort **Y** des Nutzers fragen und dann **Y ist wirklich schön!** auf der Konsole ausgeben.

Schreiben Sie das Programm und führen Sie es aus!

Lesen von und Schreiben auf die Konsole

Schreiben Sie ein Java-Programm (**GutenMorgen.java**), das erst nach dem Namen des Nutzers **X** fragt und dann **Guten Morgen X!** auf der Konsole ausgibt. Beachten Sie dabei, dass der Text **X** durch den eingegebenen Namen ersetzt wird und am Ende ein Ausrufezeichen steht.

Als zweites soll das selbe Programm dann nach dem Wohnort **Y** des Nutzers fragen und dann **Y ist wirklich schön!** auf der Konsole ausgeben.

Schreiben Sie das Programm und führen Sie es aus!

2. EINFACHE PROZEDURALE PROGRAMMIERUNG

Prozedurale Elemente

Kommentare:

Dienen der Codedokumentation und werden vom Compiler ignoriert.

primitive Datentypen:

ganze Zahlen (byte, int, long), Fließkommazahlen (float, double), Zeichen (char, byte), Wahrheitswerte (boolean)

Literale: Konstante Wert (z.B. 42, 3.14, 'A', "Hello World!")

Zuweisungen:

Speichern eines Wertes in einer Variable

(Eine benannten Stelle im Speicher)

Ausdrücke:

dienen der Berechnung von Werten mit Hilfe von Variablen, Literalen und Operatoren

Kontrollstrukturen:

dienen der Ablaufsteuerung mit Hilfe von Schleifen (`while`, `do-while`, `for``) und Verzweigungen (`if-else`, `switch-case`)

Unterprogramme:

Methoden (*Prozeduren* und *Funktionen*), die eine bestimmte Funktionalität wiederverwendbar bereitstellen

Kommentare

- Kommentare dienen der Dokumentation des Codes und helfen anderen Entwicklern den Code zu verstehen.

- In Java unterscheiden wir folgende Arten von Kommentaren:

- Einzeilige Kommentare, die mit `//` beginnen und bis zum Ende der Zeile gehen.
- Mehrzeilige Kommentare, die mit `/*` beginnen und mit `*/` enden.

Kommentare, die mit `/**` beginnen und mit `*/` enden, sind so genannte JavaDoc Kommentare und dienen der Erzeugung von Dokumentation.

- [ab Java 23] Mehrzeilige Kommentare, bei der jede Zeile mit `///` beginnt, werden ebenfalls als JavaDoc Kommentare interpretiert.

1

Beispiel (ab Java 1.0 - spezifische Tags und HTML)

```
/**
 * Berechnet die Fakultät von n.
 *
 * @param n die Zahl, von der die Fakultät berechnet werden soll; (0 <= n <= 20).
 * @return die Fakultät von n.
 */
long fak(long n){ // TODO mögliche Fehlerfälle abfangen
    /* Die Verwendung von long als Datentyp limitiert uns auf n <= 20;
       durch den Wechsel von long auf double könnten wir bis n <= 170 rechnen;
       sind aber unpräziser. */
    if (n == 0) return 1;
    else return n * fak(n-1);
}
```

2

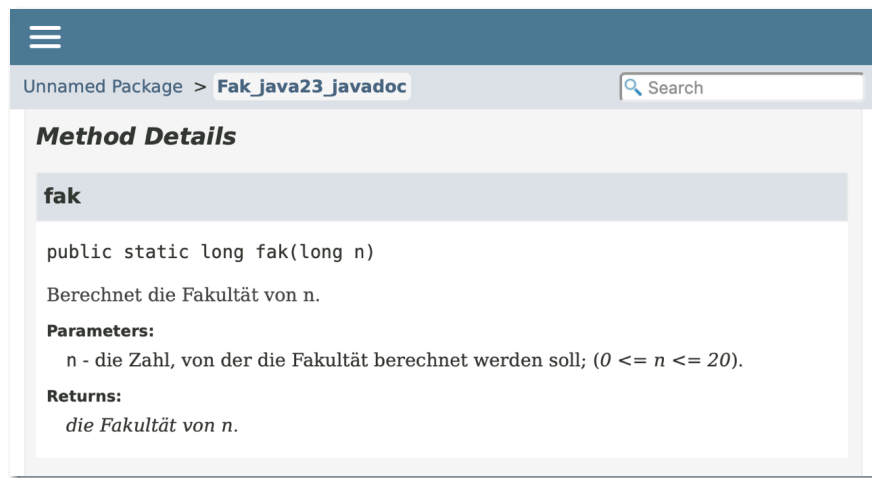
Beispiel (ab Java 23 - spezifische Tags und Markdown)

```
/// Berechnet die Fakultät von n.
///
/// @param n die Zahl, von der die Fakultät berechnet werden soll;
///         (*0 <= n <= 20*).
/// @return _die Fakultät von n_.
long fak(long n){ // TODO mögliche Fehlerfälle abfangen
    /* Die Verwendung von long als Datentyp limitiert uns auf n <= 20;
       durch den Wechsel von long auf double könnten wir bis n <= 170 rechnen;
```

```
    sind aber unpräziser. */  
    if (n == 0) return 1;  
    else return n * fak(n-1);  
}
```

3

Erzeugte Dokumentation (mit Java 23)



4

8

JavaDoc tags

@param <name descr>:

Dokumentiert einen Parameter einer Methode.



@return <descr>:

Dokumentiert den Rückgabewert einer Methode.

3. PRIMITIVE DATENTYPEN

Verwendung von Datentypen

- Um die erlaubten Werte von Parametern, Variablen und Rückgabewerten genauer spezifizieren zu können, werden Datentypen verwendet.
- Java stellt hierzu primitive Datentypen, Aufzählungen (**enum**), Klassen und Interfaces zur Verfügung

Ein primitiver Datentyp ist z. B. **int** (d. h.  *integer* bzw.  *Ganzzahl*)

Dieser Datentyp legt fest, dass ein Wert eine Ganzzahl mit dem Wertebereich:

10

Bitte beachten Sie, dass in Code für Zahlen immer die Englische Schreibweise verwendet wird. D. h. das Dezimalkomma wird durch einen Punkt ersetzt.

Java kennt neben den primitiven Datentypen auch noch Arrays, Aufzählungen (**enum**) sowie Klassen und Interfaces. Diese werden wir später behandeln.

Ganzzahlige Datentypen - Hintergrund

- Ganzzahlige Werte werden im Speicher als Binärzahlen gespeichert; d. h. als Folge von Nullen und Einsen.
- Um verschieden große Werte zu speichern, stellen Programmiersprachen ganzzahlige Werte mit einer unterschiedlichen Zahl von Bits dar.

Zahlen werden immer mit 8 Bit (1 Byte), 16 Bit (2 Byte), 32 (4 Byte) oder 64 Bit (8 Byte) gespeichert.

Hinweis

In Java werden Zahlen immer vorzeichenbehaftet gespeichert. D. h. ein Bit wird für das Vorzeichen verwendet; auch wenn es nicht immer benötigt wird.

1

Umrechnung Binär-Dezimal

Binär	Dezimal
0000 0000	+0
0000 0001	+1
...	...
0111 1111	+127
1000 0000	-128
...	...
1111 1111	-1

2

Datentyp	Genauigkeit (in Bit)	Wertebereich	Anzahl Werte
byte	8	-128 bis 127	2^8
short	16	-32768 bis 32767	2^{16}
int	32	-2147483648 bis 2147483647	2^{32}
long	64	-922337022036854775808 bis 922337022036854775807	2^{64}

3

11

Die Größenwahl für `long` und `int` ist teilweise historisch bedingt. Auf gängigen Prozessoren sind jedoch 64 Bit und 32 Bit die natürlichen Größen für Ganzzahlen und können effizient verarbeitet werden.

Gleitkommatypen - Hintergrund (Konzeptionell)

- Gleitkommazahlen werden in **Java nach Norm IEEE 754** (Seit Java 15 Version 2019) durch die Mantisse m und den Exponent e dargestellt: $z = m \times 2^e$

Für das Vorzeichen wird das erste Bit verwendet, für Mantisse und Exponent werden zusammen 31- (bei **float**) bzw. 63-Bit (bei **double**) verwendet.

Die Mantisse und der Exponent sind vorzeichenbehaftete Ganzzahlen.

Beispiel (vereinfacht)

12

Ganzzahlen $< 2^{24}$ können bei Verwendung des Datentyps **float** exakt dargestellt werden; bei **double** sind es Ganzzahlen $< 2^{53}$.

In beiden Fällen gibt es noch die Möglichkeit +/- Unendlich und NaN (Not a Number) zu repräsentieren.

Gleitkommatypen - Verwendung

Warnung

Bei Berechnungen mit Gleitkommazahlen treten Rundungsfehler auf, da nicht alle Werte in beliebiger Genauigkeit dargestellt werden können

Beispiel: Der Wert `0.123456789f (float)` wird durch die Darstellung mit Mantisse und Exponent ($m \times 2^e$) zu `0.12345679`.

Gleitkommazahlen sind somit nicht für betriebswirtschaftliche Anwendungen geeignet.

Gleitkommazahlen sind z. B. für wissenschaftliche Anwendungen geeignet.

Für betriebswirtschaftliche Anwendungen gibt es den Datentyp `BigDecimal`. Dieser ist aber kein primitiver Datentyp und wird später behandelt.

Zeichen - Hintergrund

- einzelne Zeichen (z. B. 'a') werden in Java mit dem Datentyp `char` dargestellt
- ein `char` ist (intern) eine vorzeichenlose Ganzzahl mit 16 Bit (d. h. eine Zahl im Bereich `[0, 65536]`), die den Unicode-Wert des Zeichens repräsentiert

Alle gängigen (westeuropäischen) Zeichen können mit einem `char` dargestellt werden.

Warnung

Seit Java eingeführt wurde, wurde der Unicode Standard mehrfach weiterentwickelt und heute gibt es Zeichen, die bis zu 32 Bit benötigen. Diese können mit nur einem `char` nicht dargestellt werden und benötigen ggf. zwei `chars`.

- Für Zeichenketten (z. B. "Hello World") existiert ein nicht-primitiver Datentyp `String`.

14

Unicode Zeichen und chars

Hinweise: - `0x1F60E` ist der Unicode Codepoint von 🤪 und `Character.toChars(<Wert>)` rechnet den Wert um. - In Java ist die Länge (`<String>.length()`) einer Zeichenkette (🇩🇪 *String*) die Anzahl der benötigten `chars` und entspricht somit nicht notwendigerweise der Anzahl der (sichtbaren) Zeichen.

```
1 jshell> var smiley = Character.toChars(0x1F60E)
2 smiley ==> char[2] { '?', '?' }
3
4 jshell> var s = new String(smiley)
5 s ==> "🤪"
6
7 jshell> s.length()
8 $1 ==> 2
9
10 jshell> s.getBytes(StandardCharsets.UTF_8)
11 $2 ==> byte[4] { -16, -97, -104, -114 }
12
13 jshell> s.codePointCount(0,s.length())
14 $3 ==> 1
```

Wahrheitswerte (Boolesche) - Hintergrund

- die Wahrheitswerte wahr (**true**) und falsch (**false**) werden in Java mit dem Datentyp **boolean** dargestellt
- häufigste (explizite) Verwendung ist das Speichern des Ergebnisses einer Bedingungsüberprüfung

(Wahrheitswerte sind zentral für Bedingungsüberprüfungen und Schleifen, werden dort aber selten explizit gespeichert; z. B. beim Test von **n** auf 0 im Algorithmus für die Berechnung der Fakultät.)

Konvertierung von Datentypen

- Die (meist verlustfreie), implizite Konvertierung von Datentypen ist nur in eine Richtung möglich:

`((byte → short) | char) → int → long → float → double`

- Konvertierungen in die andere Richtung sind immer explizit anzugeben, da es zu Informationsverlust kommen kann

Beispiel: `int` zu `byte` (Wertebereich `[-128, 127]`)

Bei der Konvertierung von `int` zu `byte` werden die höherwertigen Bits (9 bis 32) einfach abgeschnitten.

`(byte) 128 ⇒ -128`

`(byte) 255 ⇒ -1`

`(byte) 256 ⇒ 0`

16

- Beispiel für die verlustbehaftete implizite Konvertierung

```
jshell> long l = Long.MAX_VALUE - 1;
l ==> 9223372036854775806

jshell> float f = l
f ==> 9.223372E18

jshell> f == l
$1 ==> true // Warum?

jshell> ((long) f) == l
$2 ==> false

jshell> ((long) f)
$3 ==> 9223372036854775807 // == Long.MAX_VALUE
```

- Wahrheitswerte können nicht konvertiert werden.

4. LITERALE

Literale - Übersicht

Literale stellen konstante Werte eines bestimmten Datentyps dar

Datentyp	Literal (Beispiele)
<code>int</code>	Dezimal: 127 ; Hexadezimal: 0xcafebabe ^[3] ; Oktal: 010 ; Binär: 0b1010
<code>long</code>	123_456_789l oder 123456789L ("_" dient nur der besseren Lesbarkeit)
<code>float</code>	0.123456789f oder 0.123456789F
<code>double</code>	0.123456789 oder 0.123456789d oder 0.123456789D
<code>char</code>	'a' (Zeichen-Darstellung) oder 97 (Zahlen-Darstellung) oder 'u0061' (Unicode-Darstellung) oder Sonderzeichen (siehe nächste Folie)
<code>String</code>	"Hallo"
<code>boolean</code>	true oder false

^[3] 0xcafebabe ist der Header aller kompilierten Java-Klassen-Dateien.

Literale - Sonderzeichen ("\\" ist das Escape-Zeichen)

Datentyp	Literal (Beispiele)
\'	Einfaches Hochkomma
\"	Doppeltes Hochkomma
\\	Backslash
\b	Rückschrittaste (backspace)
\f	Seitenvorschub (form feed)
\n	Zeilenschaltung (line feed)
\t	Tabulator
\r	Wagenrücklauf

5. VARIABLEN UND KONSTANTEN

Variablen - Übersicht

- Variablen stellen einen logischen Bezeichner für einen Wert eines bestimmten Datentyps dar.
- Variablen müssen erst deklariert werden. Danach können sie weiter initialisiert werden, wenn der Standardwert nicht ausreicht.

Deklaration:

Variablennamen und Datentyp werden festgelegt

Initialisierung (optional):

Variablen werden mit einem bestimmten Wert versehen

- der Wert einer Variablen kann jederzeit geändert werden

1

Beispieldeklaration und -initialisierung

```
void main() {  
    // Deklaration (Datentyp muss konkret angegeben werden)  
    int alter;  
    // Deklaration und Initialisierung inkl. Datentyp (Standardfall)  
    String name = "Asta Mueller";  
  
    // vereinfachte Deklaration mittels var und Initialisierung (seit Java 10)  
    // (Datentyp wird automatisch erkannt)  
    var geburtsOrt = "Berlin";  
    var wohnort = "Schönau";  
    var geschlecht = 'd';  
  
    alter = 25; // späte Initialisierung  
    println(name + "(" + geschlecht + ")", " + alter + " Jahre, aus " + wohnort);  
}
```

2

Konstanten - Übersicht

- Konstanten sind Variablen, die nach der Initialisierung nicht mehr verändert werden können
- Konstanten werden in Java mit dem Schlüsselwort **final** deklariert
- Es wird überprüft, dass keine weitere Zuweisung erfolgt
- Konvention: Konstanten werden in Großbuchstaben geschrieben

1

Beispieldeklaration und -initialisierung

```
void main() {  
    // Deklaration und Initialisierung inkl. Datentyp (Standardfall)  
    final String NAME = "Asta Mueller";  
  
    // vereinfachte Deklaration mittels var und Initialisierung (seit Java 10)  
    // (Datentyp wird automatisch erkannt)  
    final var WOHNORT = "Schönau";  
    final var GESCHLECHT = 'd';  
  
    println(NAME + "(" + GESCHLECHT + ")", " + " Jahre, aus " + WOHNORT);  
  
    // name = "Berta"; // error: cannot assign a value to final variable name  
}
```

2

Bezeichner (*Identifier*) - Übersicht

- Bezeichner sind Namen für Variablen, Konstanten, Methoden, Klassen, Interfaces, Enums, etc.
- Erstes Zeichen: Buchstabe, Unterstrich (`_`) oder Dollarzeichen (`$`);
- Folgende Zeichen: Buchstaben, Ziffern, Unterstrich oder Dollarzeichen
- Groß- und Kleinschreibung wird unterschieden
- Schlüsselworte (z. B. `var`, `int`, etc.) dürfen nicht als Bezeichner verwendet werden
- Konvention:

- Variablen (z. B. `aktuellerHerzschlag`) und Methoden (z. B. `println`) verwenden *lowerCamelCase*
- Konstanten verwenden *UPPER_CASE* und Unterstriche (z. B. `GEWICHT_BEI_GEBURT`)
- Klassen, Interfaces und Enums verwenden *UpperCamelCase* (z. B. `BigDecimal`)

23

In Java ist es unüblich, das Dollarzeichen (\$) in eigenem Code zu verwenden und es wird in der Regel nur von der JVM (der Java Virtual Machine; d. h. der Ausführungsumgebung) verwendet; ein Unterstrich am Anfang des Bezeichners sollte ebenfalls vermieden werden. Ganz insbesondere ist darauf zu verzichten den Unterstrich als alleinige Variablennamen zu verwenden, da der "reine" Unterstrich auch andere Bedeutungen hat bzw. bekommen könnte/wird.

Java Shell

- Die Java Shell (**jshell**) ist ein interaktives Werkzeug, das es ermöglicht, Java-Code direkt auszuführen.
- Starten Sie die Java Shell mit dem Befehl **jshell --enable-preview** in der Konsole.
- Den gültigen Java-Code können Sie direkt in der Java Shell eingeben oder über **/edit** als Ganzes bearbeiten.
- Sie beenden die Java Shell mit dem Befehl **/exit**.
- Die Java Shell eignet sich insbesondere für das Ausprobieren von Code-Schnipseln und das Testen von Methoden.

1

```
# jshell --enable-preview

| Welcome to JShell -- Version 23
| For an introduction type: /help intro

jshell> var x = "X";
x ==> "X"

jshell> x + "Y"
$2 ==> "XY"

jshell> $2.length()
$3 ==> 2
```

2

Hinweis

Für diese Aufgabe können Sie sowohl die Java Shell verwenden als auch Ihren Code in eine Datei schreiben. Denken Sie in diesem Fall daran, dass der Code in einer Methode `main` stehen muss (`void main(){ <IHRE CODE> }`).

- Deklarieren und initialisieren Sie eine Variable `x` mit dem Ganzzahlwert 42.
 - Welche Datentypen können Sie verwenden, wenn eine präzise Darstellung des Wertes notwendig ist?
 - Welcher Datentyp wird verwendet, wenn Sie keinen Typ angeben (d. h. wenn Sie `var` schreiben bzw. anders ausgedrückt welchen Typ hat das Literal 42)?
- Weisen Sie den Wert der Variable `x` einer Variable `f` vom Typ `float` zu.
- Ändern Sie den Wert der Variablen `x`. Welche Auswirkungen hat das auf die Variable `f` vom Typ `float`?
- Deklarieren und initialisieren Sie die Konstante π (Wert 3.14159265359).

- Deklarieren und initialisieren Sie eine Variable `x` mit dem Ganzzahlwert 42.
 - Welche Datentypen können Sie verwenden, wenn eine präzise Darstellung des Wertes notwendig ist?
 - Welcher Datentyp wird verwendet, wenn Sie keinen Typ angeben (d. h. wenn Sie `var` schreiben bzw. anders ausgedrückt welchen Typ hat das Literal `42`)?
- Weisen Sie den Wert der Variable `x` einer Variable `f` vom Typ `float` zu.
- Ändern Sie den Wert der Variablen `x`. Welche Auswirkungen hat das auf die Variable `f` vom Typ `float`?
- Deklarieren und initialisieren Sie die Konstante π (Wert 3.14159265359).

6. AUSDRÜCKE UND OPERATOREN

Ausdrücke und Operatoren - Übersicht

- Berechnungen erfolgen über Ausdrücke, die sich aus Variablen, Konstanten, Literalen, Methodenaufrufen und Operatoren zusammensetzen.
- Jeder Ausdruck hat ein Ergebnis (d. h. Rückgabewert).
Beispiel: `(age + 1)` addiert zwei Werte und liefert das Ergebnis der Addition zurück.
- Einfache Ausdrücke sind Variablen, Konstanten, Literale und Methodenaufrufe.
- Komplexe Ausdrücke werden aus einfachen Ausdrücken und Operatoren (z. B. +, -, *, /, %, >, <, >=, <=) zusammengesetzt
- Ergebnisse von Ausdrücken können insbesondere Variablen zugewiesen werden (z.B. `int newAge = age + 1` oder `var isAdult = age >= 18`)
- Ausdrücke, die einen Wahrheitswerte ergeben können zusätzlich in Bedingungen (z. B. `if(age + 5 >= 18) ...`) verwendet werden.

Ausdrücke und Operatoren - Beispiele

```
void main() {  
    String s = readln("Enter your age: ");  
    int age = Integer.parseInt(s);  
  
    if (age >= 18) {  
        println("You are an adult.");  
    } else {  
        println("You are a minor.");  
    }  
  
    var yearsUntil100 = 100-age;  
    println("You will be 100 in " + yearsUntil100 + " years.");  
}
```


Operatoren

- Operatoren sind spezielle Zeichen, die auf Variablen, Konstanten und Literale angewendet werden, um Ausdrücke zu bilden.
- Die Auswertungsreihenfolge wird durch die Priorität der Operatoren bestimmt.
(Wie aus der Schulmathematik bekannt gilt auch in Java: * oder / vor + und –.)
- Runde Klammern können verwendet werden, um eine bestimmte Auswertungsreihenfolge zu erzwingen bzw. dienen zur Strukturierung
- Es gibt Operatoren, die auf eine, zwei oder drei Operanden angewendet werden: diese nennt man dann ein-, zwei- oder dreistellige Operatoren.
- Für einstellige Operatoren wird die Präfix- oder Postfix-Notation (z.B. `++a` oder `a++`) verwendet,
- Für mehrstellige Operatoren wird die Infix-Notation (z.B. `a + b`) verwendet