

Objektorientierte Programmierung in Java

Vorlesung 2 - Imperative Konzepte

Emily Lucia Antosch

HAW Hamburg

09.10.2024

Inhaltsverzeichnis

1. Einleitung	3
2. Einfache Datentypen	7
3. Kommentare und Bezeichner	29
4. Operatoren	41
5. Typkonvertierung	53
6. Kontrollstrukturen	64
7. License Notice	88

1. Einleitung

1.1 Wo sind wir gerade?

- In der Einführung habe ich Ihnen einen Überblick über die Themen der bevorstehenden Vorlesung gegeben.
- Sie haben außerdem Ihr erstes Programm in Java geschrieben!
- Heute geht es um **Imperative Konzepte**.

1.1 Wo sind wir gerade?

1. **Imperative Konzepte**
2. Klassen und Objekte
3. Klassenbibliothek
4. Vererbung
5. Schnittstellen
6. Graphische Oberflächen
7. Ausnahmebehandlung
8. Eingaben und Ausgaben
9. Multithreading (Parallel Computing)

1.2 Das Ziel dieses Kapitels

- Wir sprechen über imperative Konzepte in der Programmierung mit Java.
- Sie verstehen die einfachen Datentypen in Java.
- Sie steuern den Programmfluss mit Kontrollstrukturen und Schleifen.
- Sie wenden den korrekten Coding Style an.

2. Einfache Datentypen

? Frage

Wie kann sich sein Program Zustand merken?

? Frage

Wie kann sich sein Program Zustand merken?

- Variablen, die den Zustand im Speicher des Computers speichern.
- Inhalt des Speichers auf dem Computer wird anhand des **Datentyps** interpretiert.

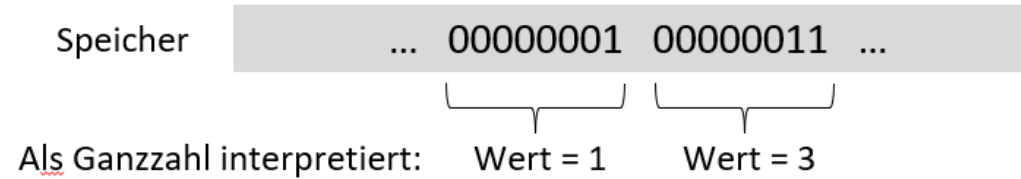


Abbildung 1: Speicher im Computer mit Werten aus dem Programm

? Frage

Welche Datentypen kennen Sie schon aus C?

? Frage

Welche Datentypen kennen Sie schon aus C?

- **int, char, float, double**
- **struct, enum, union**
- **void, bool**
- **Arrays mit [] und Zeiger mit ***

2.2 Datentypen in Java

2. Einfache Datentypen


Folgende Datenstrukturen sind in Java verfügbar:

Wahrheitswert:


boolean (1 Bit) 

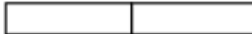
Ganzzahlen:

byte (1 Byte) 


short (2 Byte) 

int (4 Byte) 

long (8 Byte) 

char (2 Byte) 

Gleitkommazahlen:

float (4 Byte) 

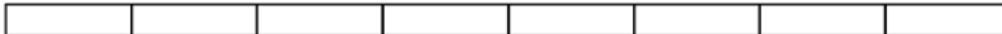
double (8 Byte) 

Abbildung 2: Datentypen in Java

2.2 Datentypen in Java

2. Einfache Datentypen

- Speichergrößen und die entsprechenden Wertebereiche:

Art	Datentyp	Größe	Werte	
Ganzzahl (Zeichen)	byte	1 Byte	-2^7 bis $2^7 - 1$	entspricht -128 bis 127
	short	2 Byte	-2^{15} bis $2^{15} - 1$	entspricht -32.768 bis 32.767
	int	4 Byte	-2^{31} bis $2^{31} - 1$	
	long	8 Byte	-2^{63} bis $2^{63} - 1$	
	char	2 Byte	0 bis $2^{16} - 1$	entspricht 0 bis 65.535
Fließkomma	float	4 Byte	$1,4 \cdot 10^{-45}$ bis $3,4 \cdot 10^{38}$	ungefährer Wertebereich
	double	8 Byte	$4,9 \cdot 10^{-324}$ bis $1,8 \cdot 10^{308}$	ungefährer Wertebereich
Wahrheit	boolean	1 Bit	true, false	

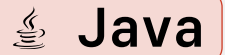
Abbildung 3: Wertebereiche der Datentypen in Java

! Merke

Variablen müssen deklariert werden, bevor sie benutzt werden können.

- Ein Datentyp wird vor dem Variablennamen geschrieben.
- Eine Deklaration könnte so aussehen:

```
1 int a;  
2 float b;  
3 char c;
```



! Merke

Im Anschluss an die Deklaration kann ein Wert zugewiesen werden. Das nennt man Initialisierung.

- Der Variable wird mittels des Zuweisungsoperators = ein Wert zugewiesen:

```
1  a = 5;  
2  b = 3.5;  
3  c = 'A';
```

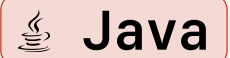


! Merke

Die Deklaration und Initialisierung kann auch in einem Schritt erfolgen. Das wird dann als Definition bezeichnet.

- Beide Schritte werden direkt hintereinander geschrieben.
- Deklaration und Initialisierung (Definition):

```
1 int a = 5;  
2 float b = 3.5;  
3 char c = 'A';
```



2.6 Gültigkeitsbereich von Variablen

2. Einfache Datentypen

- Variablen haben einen Gültigkeitsbereich, der durch die geschweiften Klammern definiert wird.
- Variablen können an beliebiger Stelle im Code deklariert werden.
- Der Compiler verhindert die Verwendung von Variablen, die nicht initialisiert wurden.

2.7 Typkorrektheit

- Typen müssen korrekt sein, um Fehler zu vermeiden.
 - ▶ Anders als in C müssen Werte dem korrektem Datentyp zugewiesen werden.
 - ▶ Folgendes würde nicht funktionieren:

```
1 int a = 5;  
2 float b = a;
```

**Java**

Inkorrekter Typ

? Frage

Welche Unterschiede sehen Sie zwischen C und Java, wenn es um Datentypen geht?

- Keine Zusammengesetzten Datentypen in Java.
- Kein `unsigned` in Java.
- Speichergrößen sind festgelegt und garantiert.
- Zeichen werden mit 2 Byte kodiert.
 - ▶ 65.536 Zeichen können dargestellt werden anstatt von 256.

! Merke

Ein **Literal** ist eine konstante, unveränderliche Zahl oder Zeichenfolge, die direkt im Code steht.

- Wenn Sie also einen bestimmten Wert direkt in Code schreiben, verwenden Sie einen Literal.
- Dieser wird dann nicht von einer Variablen repräsentiert.

? Frage

Warum glauben Sie, dass der folgende Code nicht funktioniert?

```
1 float point = 3.1416;
```



Java

? Frage

Warum glauben Sie, dass der folgende Code nicht funktioniert?

```
1 float point = 3.1416;
```



Java

- Der Zahl ist eine feste Fließkommazahl, die von Java als **double** interpretiert wird.
- Wegen der Typkorrektheit wird der Wert nicht in eine **float** Variable gespeichert. Der Java Compiler gibt einen Fehler aus.

? Frage

Wie würden Sie den Code korrigieren?

? Frage

Wie würden Sie den Code korrigieren?

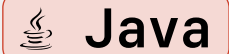
- Sie können den Wert als **float** Literal schreiben:

```
1 float point = 3.1416f;
```



- Alternativ können Sie den Wert in eine **double** Variable speichern:

```
1 double point = 3.1416d;
```

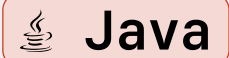


2.9 Konstanten

2. Einfache Datentypen

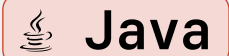
- Wir haben gerade bereits das Beispiel der Kreiszahl π gehabt.
- In Java gibt es das Schlüsselwort `final`, um Konstanten zu definieren.
- Diese können dann nicht mehr verändert werden.

```
1  final double PI = 3.1416;
```



- Nachdem eine Konstante deklariert wurde, kann sie nicht mehr verändert werden. Der folgende Code würde also einen Fehler erzeugen:

```
1  PI = 3;
```

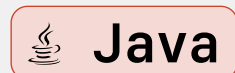


☰ Aufgabe 1

Wir wollen jetzt einmal eine Konsolenausgabe erzeugen:

- Öffnen Sie IntelliJ IDEA und öffnen oder erstellen Sie eine neue ausführbare Klasse.
- Probieren Sie den folgenden Code:

```
1 int age = 24;  
2 System.out.println(24);  
3 System.out.println(age);
```



☰ Aufgabe 2

- Mithilfe des „+“ Operators können Sie Text und Variablen kombinieren:

```
1 int age = 24;  
2 System.out.println("Mein Alter ist " + 24);  
3 System.out.println("Mein Alter ist " + age);
```



Java



Tipp

- Geben Sie einmal in IntelliJ IDEA `sout` ein und drücken Sie die Tab-Taste. Das spart Zeit beim Schreiben von `System.out.println()`!

? Frage

Was ist ein **Coding Style**? Was sagt Ihnen der Begriff?

? Frage

Was ist ein **Coding Style**? Was sagt Ihnen der Begriff?

- Der Coding Style ist eine Sammlung von Regeln, die bestimmen, wie Code geschrieben werden sollte.
- Einheitlicher Code ist leichter zu lesen und zu warten.

! Merke

Die Einhaltung des Coding Styles wird in der Klausur bewertet!

2.12 Coding Style: Namenskonventionen

2. Einfache Datentypen

- Alle Namen, und das gilt für alle Bezeichner, sind in der englischen Sprache zu schreiben!
- Folgende Namenskonventionen sollten eingehalten werden:
 - ▶ Klassen: **CamelCase**
 - ▶ Methoden und Variablen: **camelCase**
 - ▶ Konstanten: **UPPER_CASE**
 - ▶ Pakete: **lowercase**



Tipp

Aus meiner Erfahrung: Machen Sie Ihre Variablen so aussagekräftig wie möglich! Dann darf der Name auch länger sein.

3. Kommentare und Bezeichner

- Wie bereits erwähnt, verwendet Java den Unicode-Zeichensatz.
- Das heißt, es sind mehr Zeichen möglich (65.536 um genau zu sein).
- So können Sie Ihre Kommentare ohne größere Einschränkungen auf Deutsch, Englisch oder Chinesisch schreiben.
- Ich würde Sie jedoch bitten, Ihre Kommentare in **Deutsch** oder **Englisch** zu verfassen.

! Merke

Da Ihre Tastatur keine 65.536 Zeichen hat, können Sie die Zeichen auch kopieren und einfügen. Alternativ für ☒:

```
1 System.out.println("\u{1F600}");
```



Java

? Frage

Was denken Sie zu der folgenden Aussage? Warum sind Kommentare wichtig?

” Zitat

Den Code lesbar machen? Wer soll das denn sonst lesen?

— Viele Entwickler

- Kommentare sind wichtig, um den Code zu dokumentieren und die Wartbarkeit zu verbessern.
- Sowohl Nutzer des Codes als auch die Entwickler werden den Code verstehen müssen. Dafür sind Kommentare unerlässlich.

! Merke

Nicht die Menge, sondern die Qualität der Kommentare ist entscheidend! Kommentieren Sie immer direkt während Sie auch programmieren!

? Frage

Was ist der Unterschied zwischen einem **Blockkommentar** und einem **Zeilenkommentar**?

? Frage

Was ist der Unterschied zwischen einem **Blockkommentar** und einem **Zeilenkommentar**?

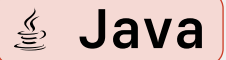
- **Zeilenkommentare** beginnen mit `//` und enden am Ende der Zeile.
- **Blockkommentare** beginnen mit `/*` und enden mit `*/`.

3.2 Kommentare

3. Kommentare und Bezeichner

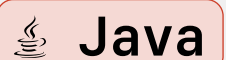
- Beispiel für einen Zeilenkommentar:

```
1 // Dies ist ein Zeilenkommentar
2 int distance; // Euklidischer Abstand zwischen a und b
```



- Beispiel für einen Blockkommentar:

```
1 /* Die Berechnung des euklidischen Abstands
   läuft über folgende Schritte ab:
2     1. Berechnung der Differenz der Koordinaten
3     2. Quadrieren der Differenz
4     ... */
```



- Alle Dinge, die sie in Java benennen, werden als **Bezeichner** bezeichnet. Viele Dinge, die Sie schreiben brauchen einen Namen!

! Merke

- Beachten Sie folgende Regeln für Bezeichner:
 - ▶ Erlaubt sind Buchstaben, Zahlen, Unterstriche und Dollarzeichen.
 - ▶ Das erste Zeichen darf keine Zahl sein.
 - ▶ Groß- und Kleinschreibung wird unterschieden.
 - ▶ Keine Leerzeichen oder Schlüsselworte.
 - ▶ Nicht die Literale `true`, `false` oder `null`.

3.3 Bezeichner

3. Kommentare und Bezeichner

- Alle reservierten Schlüsselworte in Java:

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp	

? Frage

Welche der Bezeichner sind aus Ihrer Sicht erlaubt und warum?

1 `int` length;

2 `int` länge;

3 `int` maxLength;

4 `int` max_length;

5 `int` _max_length;

6 `int` max-length;



```
7      int !maxLength;  
8  
9      int 3dlength;  
10     String öpnrKosten;  
11     String €kosten;  
12     String kostenin€  
13     String €;  
14     int long;  
15     int c.o.s.t;  
16     String @cost;
```


4. Operatoren

4.1 Operatoren

- Es gibt die üblichen arithmetischen Operatoren.
- Generell werden Operatoren auch von links nach rechts ausgewertet.

Operator	Bezeichnung	Beispiel	Priorität
+	Vorzeichen	<code>a = +7</code>	1
-	Vorzeichen	<code>a = -7</code>	1
++	<u>Inkrementierung</u>	<code>++count, count++</code>	1
--	<u>Dekrementierung</u>	<code>--count, count--</code>	1
*	Multiplikation	<code>area = length * width</code>	2
/	Division	<code>mean = sum / count</code>	2
%	Rest bei Division	<code>11 % 4</code> (ergibt 3)	2
+	Addition	<code>a = b + c</code>	3
-	Subtraktion	<code>a = b - c</code>	3

Abbildung 1: Arithmetische Operatoren in Java

4.2 Inkrement und Dekrement

- Es gibt auch die gleichen Operatoren zum Inkrementieren und Dekrementieren wie in C.

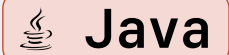
Operator	Art	Wert des Ausdrucks	Änderung von a
<code>++a</code>	Präfix	<code>a + 1</code>	<code>a = a + 1</code>
<code>a++</code>	<u>Postfix</u>	<code>a</code>	<code>a = a + 1</code>
<code>--a</code>	Präfix	<code>a - 1</code>	<code>a = a - 1</code>
<code>a--</code>	<u>Postfix</u>	<code>a</code>	<code>a = a - 1</code>

Abbildung 2: Operatoren für das Inkrement und Dekrement in Java

? Frage

Zum Mitdenken: Was wird hier auf der Konsole erscheinen?

```
1  int a = 1;
2  System.out.println("a      : " + a);
3  System.out.println("++a    : " + ++a);
4  System.out.println("a++    : " + a++);
5  System.out.println("--a    : " + --a);
6  System.out.println("a--    : " + a--);
```



4.3 Vergleichsoperatoren

- Es gibt auch die gleichen Vergleichsoperatoren wie in C!

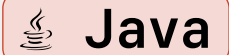
Operator	Bezeichnung	Priorität
<	kleiner	5
<=	kleiner oder gleich	5
>	größer	5
>=	größer oder gleich	5
==	gleich	6
!=	ungleich	6

Abbildung 3: Vergleichsoperatoren in Java

? Frage

Zum Mitdenken: Was passiert hier?

```
1 int a = 7, b = 4;  
2 boolean parentheses = (a > b) == (a <= b);  
3 boolean priorities = a > b == a <= b;  
4 System.out.println(parentheses);  
5 System.out.println(priorities);
```



4.4 Logische Operatoren

- Das Ergebnis der logischen Operatoren ist immer ein Wahrheitswert, der in Java als `boolean` dargestellt wird.

Operator	Bezeichnung	Priorität
!	NICHT	1
^	Exklusives ODER (XOR)	8
&&	UND	10
	ODER	11

Abbildung 4: Logische Operatoren in Java

! Merke

- Bei den logischen Operatoren wird der rechte Operand nicht ausgeführt, wenn das Ergebnis bereits feststeht. Im folgenden Beispiel wird `a` nicht ausgewertet.
- Beispiel: (`true || a`)
- Das nennt man dann **Short Circuit**.
- Wird dann interessant, wenn der rechte Operand bspw. eine Funktion/Methode ist.

? Frage

Wieder zum Mitdenken: Was passiert im folgenden Code?

```
1 int a = 3, b = 4;  
2 System.out.println((++a == b) || (a++ > b));  
3 System.out.println("a = " + a);
```



4.5 Zuweisungsoperatoren

- Wie in C gibt es auch in Java Zuweisungsoperatoren. Diese können auch mit anderen Operatoren kombiniert werden.
- Der Platzhalter `<op>` steht unter anderem für `*`, `/`, `+` und `-`.

Operator	Bezeichnung	Priorität
<code>=</code>	Zuweisung	13
<code><op>=</code>	Kombinierte Zuweisung: <code>a <op>= b</code> entspricht <code>a = a <op> b</code>	13

Abbildung 5: Zuweisungsoperatoren in Java

? Frage

Ein letztes Mal: Was passiert in diesem Code?

```
1  int a = 1;  
2  a += 2;  
3  System.out.println(a);  
4  System.out.println(a *= --a);  
5  System.out.println(a *= -a++);  
6  System.out.println(a /= 10);
```



5. Typkonvertierung

5.1 Typkonvertierung

- Zur Erinnerung: Typkorrektheit verhindert, dass Variablen einen Wert bekommen, der nicht ihrem Datentyp entspricht.
- Das verhindert Fehler und macht den Code sicherer.



Achtung

Eine Variable, die vom Typ `int` ist passt allerdings in eine Variable vom Typ `byte`. Wie können Sie den Wert in `byte` von trotzdem in eine `int` Variable speichern?



Idee

Sie können einfach schreiben, dass Sie das explizit wollen!

```
1 int a = 80;  
2 byte b = (byte) a;  
3 System.out.println(b);
```



Java

? Frage

Was passiert in dem folgenden Code?

```
1 double a = 128.38;  
2 int b = (int) a;  
3 byte c = (byte) a;  
4 System.out.println("double: " + a);  
5 System.out.println("int : " + b);  
6 System.out.println("byte : " + c);
```



Java

? Frage

Was passiert, wenn Sie den Wert von 128 in eine `byte` Variable speichern?

? Frage

Was passiert, wenn Sie den Wert von 128 in eine `byte` Variable speichern?

- Da der Datentyp nur Werte von -128 bis 127 speichern kann, wird der Wert überlaufen.
- Das Ergebnis wird eine negative Zahl sein. In diesem Fall wird es -128 sein.

! Merke

- Prinzip der impliziten Typkonvertierung:
 - ▶ Kein Datenverlust bei Zuweisung von einem kleineren in einen größeren Typ.
 - ▶ Der Cast-Operator ist nicht notwendig.
 - ▶ Es erfolgt eine automatische Konvertierung.



Beispiel

- `short` (-32.768 bis 32.767) passt in `int` (-2.147.483.648 bis 2.147.483.647).

```
1 short a = 71;
```

```
2 int b = (int) a;
```

```
3 int c = a;
```



Java

? Frage

Zum Mitdenken: Welche der folgenden Zeilen werden kompilieren?

```
1 short a = 1024;
```

```
2 long b = a;
```

```
3 float c = b;
```



5.2 Implizite Typkonvertierung

5. Typkonvertierung

```
1 char d = 'A';  
2 short e = d;  
3 int f = d;
```



Java

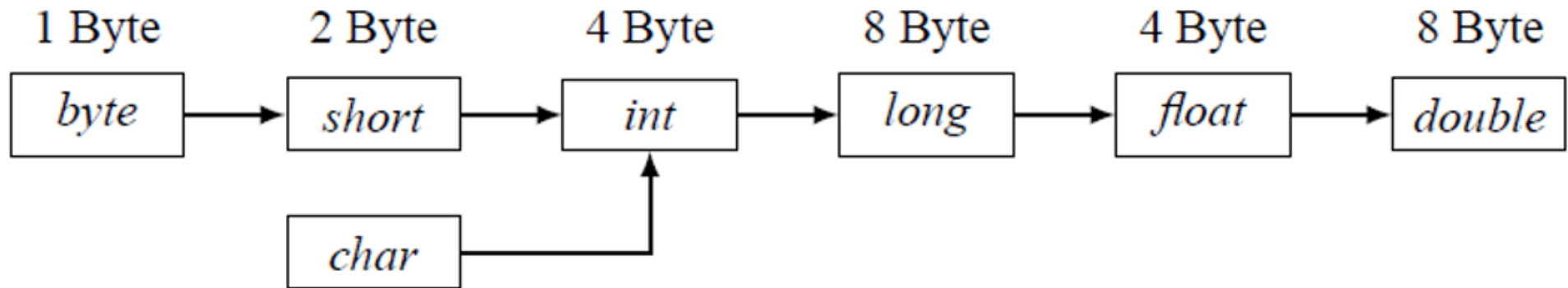


Abbildung 6: Implizite Typkonvertierung in Java

! Merke

- Ganzzahlen `char` und `short` besitzen jeweils 2 Byte, `char` ist aber ein **unsigned** Datentyp.
 - ▶ Wertebereich `char`: 0 bis 65.535
 - ▶ Wertebereich `short`: -32.768 bis 32.767
- Nicht alle `long`-Werte in `float` darstellbar (Potenzieller Datenverlust!).

6. Kontrollstrukturen

! Merke

If-Anweisungen sind die einfachste Form der Kontrollstrukturen. Sie erlauben es, Anweisungen nur auszuführen, wenn eine Bedingung erfüllt ist.

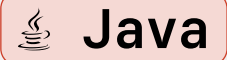
```
1  if (Bedingung) {  
2      Anweisungen  
3  }
```



6.1 if-Anweisung

- Die Bedingung muss, anders als in C, immer ein `boolean` sein.
- Anweisungen werden nur ausgeführt, wenn die Bedingung wahr (`true`) ist.
- Bei nur einer Anweisung können die geschweiften Klammern weggelassen werden.

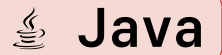
```
1  int a = 4, b = 8;  
2      int maximum = a;  
3  
4  if (b > maximum) {  
5      maximum = b;  
6  }
```

**Java**

6.2 if-else-Anweisung

Mittels einer `else`-Anweisung kann ein Block angegeben werden, der ausgeführt wird, wenn die Bedingung nicht erfüllt ist.

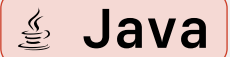
```
1  if (Bedingung) {  
2      Anweisungen 1  
3  } else {  
4      Anweisungen 2  
5  }
```



6.2 if-else-Anweisung

Die Anweisung 2 im obigen Beispiel wird eben ausgeführt, wenn die Bedingung `false` ist.

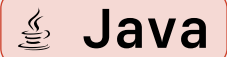
```
1  int a = 4, b = 8;  
2  int maximum;  
3  
4  if (a > b) {  
5      maximum = a;  
6  } else {  
7      maximum = b;  
8  }
```



6.3 Der ?-Operator

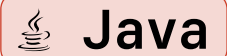
Für einfache Zuweisung mittels `if-else`-Anweisungen kann ein Ausdruck in dieser Form verwendet werden:

```
1 (Bedingung) ? Ausdruck 1 : Ausdruck 2;
```



- Bedingung `true`: Ausdruck 1 wird eingesetzt
- Bedingung `false`: Ausdruck 2 wird eingesetzt

```
1 int a = 4, b = 8;  
2 int maximum = (a > b) ? a : b;
```



Aufgabe 3

- Gegeben ist eine Ganzzahl `weekDay` zwischen 1 und 7.
- Es entspricht: 1 = Montag, 2 = Dienstag, 3 = Mittwoch usw.

Erzeugen Sie in Abhängigkeit des Wertes folgende Konsolenausgaben:

- Montag bis Freitag: „Arbeiten“
- Samstag: „Einkaufen“
- Sonntag: „Ausruhen“



Beispiel

```
1 byte weekDay = 3;  
2  
3 if (weekDay <= 5) {  
4     System.out.println("Arbeiten");  
5 } else if (weekDay == 6) {  
6     System.out.println("Einkaufen");  
7 } else if (weekDay == 7) {  
8     System.out.println("Ausruhen");  
9 }
```

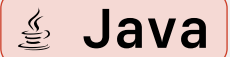


Java

6.5 switch-Anweisung

Mit der `switch`-Anweisung lassen sich `if-else`-Anweisungen vereinfachen.

```
1  switch (Ausdruck) {  
2      case Wert 1:  
3          Anweisungen  
4      break;  
5      case Wert 2:  
6          ...  
7      default:  
8          Anweisungen  
9  }
```



6.5 switch-Anweisung

- Ausdruck ist z.B. eine ganzzahlige Variable (außer Typ `long`) oder ein `String` (ab Java 7).
- Anweisungen, `break` und `default` sind optional.
- Mehrere `case`-Sprungmarken direkt hintereinander sind erlaubt.
- Sprung zu ...
 - ▶ `case`-Sprungmarke, falls diese den Wert von Ausdruck hat
 - ▶ `default`, falls keine passende `case`-Sprungmarke
 - ▶ Ende des `switch`-Blocks, falls keine passende `case`-Sprungmarke und kein `default`
- Von `case`-Sprungmarke oder `default` weiter bis `break` oder Ende des `switch`-Blocks

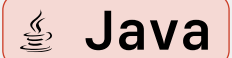
Aufgabe 4

Implementieren Sie eine Lösung für die Aufgabe 3 als `switch`-Anweisung

6.5 switch-Anweisung

6. Kontrollstrukturen

```
1  switch (weekDay) {  
2      case 1:  
3      case 2:  
4      case 3:  
5      case 4:  
6      case 5:  
7          System.out.println("Arbeiten");  
8          break;  
9      case 6:  
10         System.out.println("Einkaufen");  
11         break;
```



6.5 switch-Anweisung

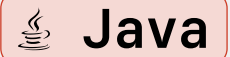
6. Kontrollstrukturen

```
12     case 7:
13         System.out.println("Ausruhen");
14         break;
15     default:
16         System.out.println("Den Tag kenn' ich nicht ...");
17 }
```

6.6 while-Schleife

Mit der `while`-Schleife wird eine Anweisung so lange ausgeführt, wie die Bedingung `true` ist.

```
1 while (Bedingung) {  
2     Anweisungen  
3 }
```



- Falls die Bedingung zu Beginn bereits `false` ist, wird die Anweisung nie ausgeführt.
- Auch **kopfgesteuerte** oder **abweisende** Schleife genannt.

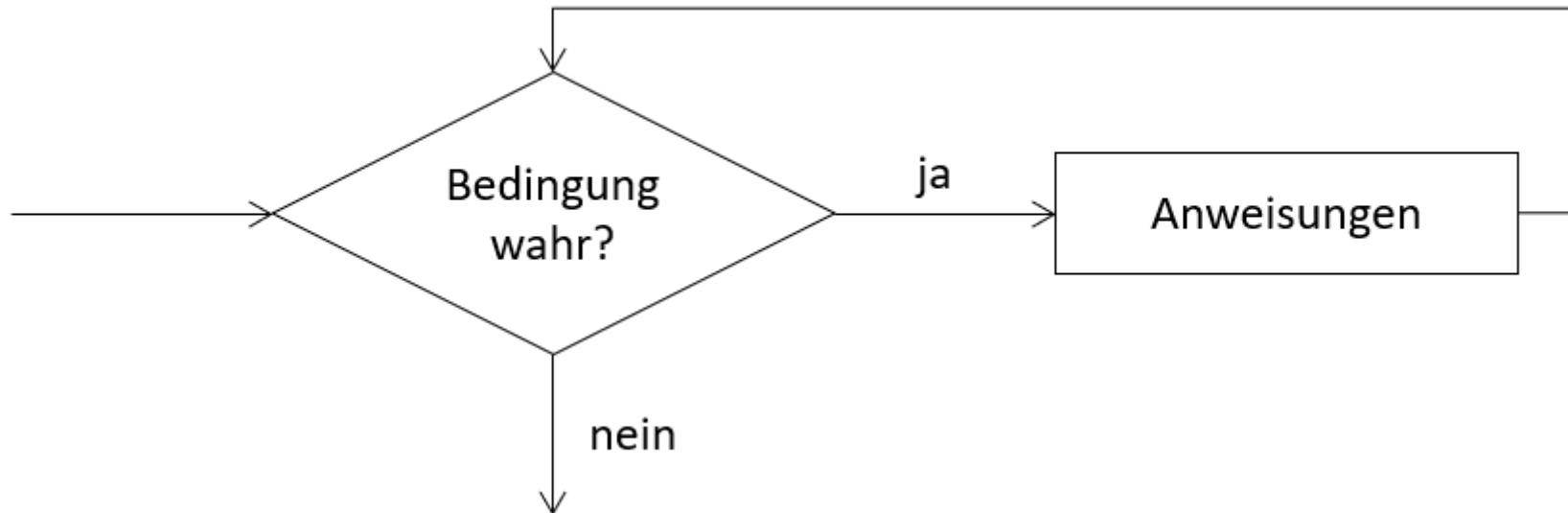


Abbildung 7: while-Schleife in Java

6.7 do-while-Schleife

Mit der `do-while`-Schleife wird eine Anweisung mindestens einmal ausgeführt. Wenn die Bedingung `true` ist, wird die Anweisung wieder ausgeführt.

```
1  do {  
2      Anweisungen  
3  } while (Bedingung);
```



- Auch **fußgesteuerte** oder **nicht abweisende** Schleife genannt.

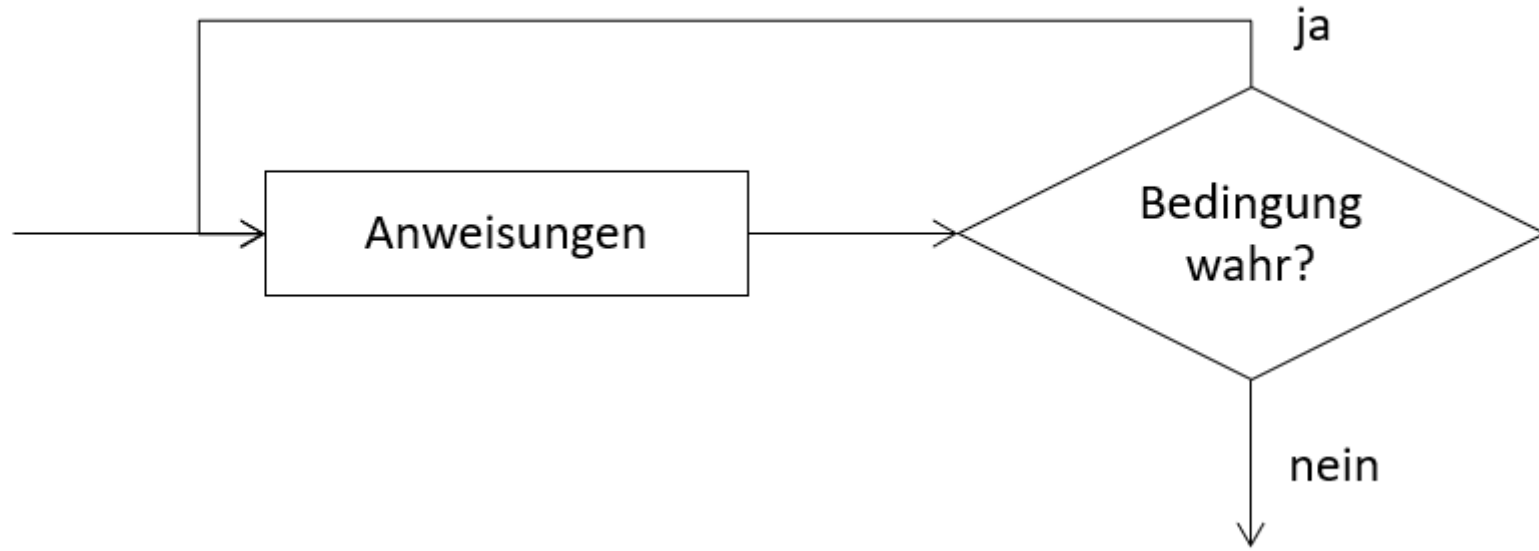


Abbildung 8: do-while-Schleife in Java

Mit der for-Schleife können Sie eine Anweisung eine bestimmte Anzahl von Malen wiederholen.

```
1  for (Init; Bedingung; Update) {  
2      Anweisungen  
3  }
```



- Falls Bedingung `false` ist, wird die Anweisung nie ausgeführt.
- Init wird nur einmal ausgeführt, dafür aber immer.
- Update wird nach jeder Iteration ausgeführt.

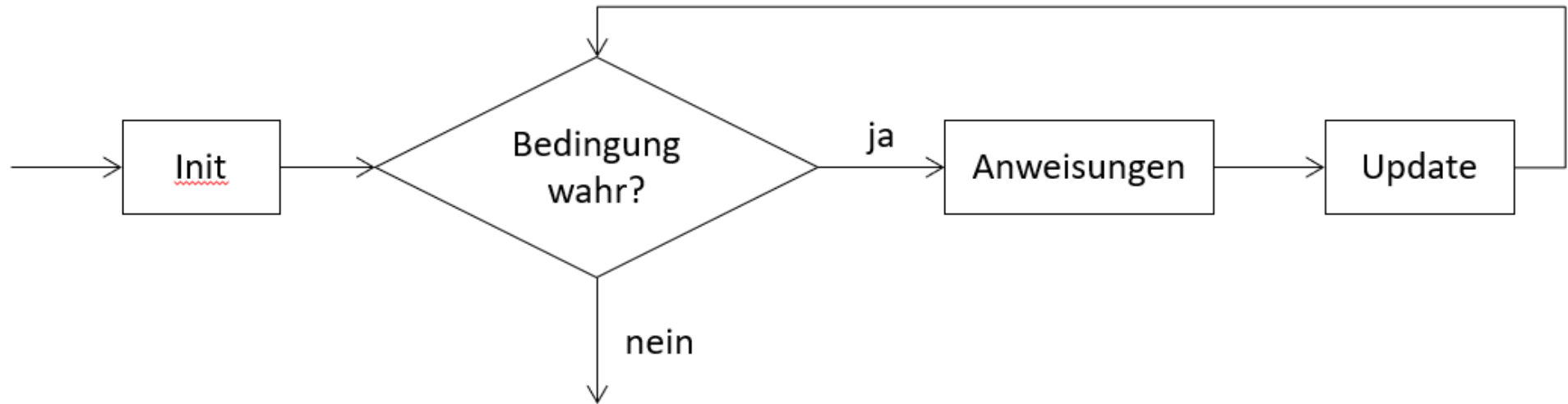


Abbildung 9: for-Schleife in Java

6.9 Sprunganweisungen

Mittels Sprunganweisungen können Sie den Programmfluss steuern. `break` beendet die Schleife und `continue` springt zum nächsten Schleifendurchlauf.

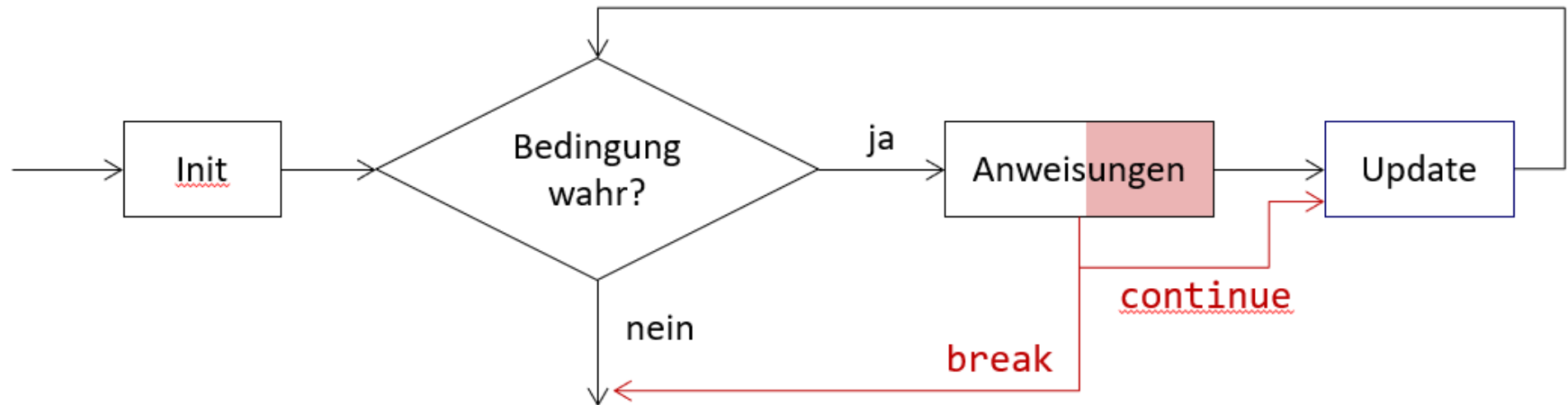
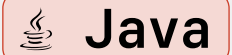


Abbildung 10: Visualisierung von `break` und `continue` in Java

? Frage

Was passiert in dem folgenden Code?

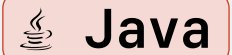
```
1 System.out.println("Break (bei i == 2):");
2 for (int i = 0; i <= 4; i++) {
3     if (i == 2) {
4         break;
5     }
6     System.out.println("  i = " + i);
7 }
```



? Frage

Was passiert in dem folgenden Code?

```
1 System.out.println("\nContinue (bei i == 2):");
2 for (int i = 0; i <= 4; i++) {
3     if (i == 2) {
4         continue;
5     }
6     System.out.println("  i = " + i);
7 }
```

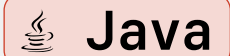


- Wie bereits erwähnt, ist der Coding Style wichtig. Daher gibt es bei den Kontrollstrukturen auch einen Coding Style.
- Öffnende geschweifte Klammern werden in der gleichen Zeile wie die Kontrollstruktur geschrieben (das gilt für alle öffnenden Klammern).
- Nach einer schließenden geschweiften Klammer wird ein Zeilenumbruch gemacht. Bei `else` steht die schließende Klammer in der gleichen Zeile.



Beispiel

```
1  int a = 4, b = 8;  
2      int maximum;  
3  
4  if (a > b) {  
5      maximum = a;  
6  } else {  
7      maximum = b;  
8  }
```



7. License Notice

7.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>
- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.