

# Object-Oriented Programming in Java

## Lecture 2 - Imperative Concepts

Emily Lucia Antosch

HAW Hamburg

30.06.2025

# Contents

1. Introduction .....	2
2. Simple Data Types .....	6
3. Comments and Identifiers .....	29
4. Operators .....	41
5. Type Conversion .....	53
6. Control Structures .....	64
7. License Notice .....	88

# 1. Introduction

---

# 1.1 Where Are We Now?

## 1. Introduction

- In the introduction, I gave you an overview of the topics for the upcoming lecture.
- You have also written your first program in Java!
- Today we'll cover **Imperative Concepts**.

# 1.1 Where Are We Now?

## 1. Introduction

1. Imperative Concepts
2. Classes and Objects
3. Class Library
4. Inheritance
5. Interfaces
6. Graphical User Interfaces
7. Exception Handling
8. Input and Output
9. Multithreading (Parallel Computing)

# 1.2 The Goal of This Chapter

- We will discuss imperative concepts in programming with Java.
- You will understand the simple data types in Java.
- You will control program flow with control structures and loops.
- You will apply the correct coding style.

## 2. Simple Data Types

---

### ? Question

How can a program remember its state?





### ? Question

How can a program remember its state?

- Variables that store the state in the computer's memory.
- The content of the memory on the computer is interpreted based on the **data type**.

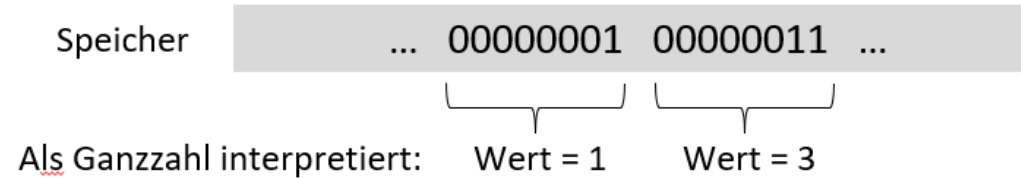


Figure 1: Memory in the computer with values from the program

### ? Question

Which data types do you already know from C?

### ? Question

Which data types do you already know from C?

- int, char, float, double
- struct, enum, union
- void, bool
- Arrays with [] and Pointers with \*


The following data structures are available in Java:

### Wahrheitswert:


*boolean* (1 Bit) 


### Ganzzahlen:

*byte* (1 Byte) 

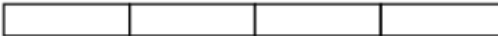
*short* (2 Byte) 

*int* (4 Byte) 

*long* (8 Byte) 

*char* (2 Byte) 

### Gleitkommazahlen:

*float* (4 Byte) 


*double* (8 Byte) 

Figure 2: Data types in Java

- Memory sizes and the corresponding value ranges:

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	

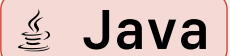
Figure 3: Value ranges of data types in Java

### ! Memorize

Variables must be declared before they can be used.

- A data type is written before the variable name.
- A declaration could look like this:

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



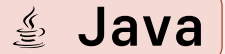


### ! Memorize

After declaration, a value can be assigned. This is called initialization.

- A value is assigned to the variable using the assignment operator =:

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

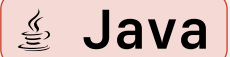


### ! Memorize

Declaration and initialization can also be done in one step.  
This is then called definition.

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

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



- Variables have a scope that is defined by the curly braces.
- Variables can be declared at any point in the code.
- The compiler prevents the use of variables that have not been initialized.

## 2.7 Type Correctness

## 2. Simple Data Types

- Types must be correct to avoid errors.
  - Unlike in C, values must be assigned to the correct data type.
  - The following would not work:

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



Java

Incorrect type

A blue arrow originates from the text 'Incorrect type' and points directly to the variable 'a' in the assignment 'float b = a;'.

### ? Question

What differences do you see between C and Java when it comes to data types?

- No composite data types in Java.
- No unsigned in Java.
- Memory sizes are fixed and guaranteed.
- Zeichen werden mit 2 Byte kodiert.
  - 65,536 characters can be represented instead of 256.

### ! Memorize

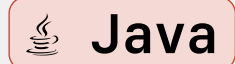
A **literal** is a constant, immutable number or string that appears directly in the code.

- So when you write a specific value directly in code, you use a literal.
- This is then not represented by a variable.

### ? Question

Why do you think the following code doesn't work?

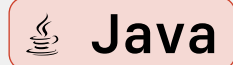
```
1 float point = 3.1416;
```



### ? Question

Why do you think the following code doesn't work?

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



- The number is a fixed floating-point number that is interpreted by Java as **double**.
- Due to type correctness, the value is not stored in a **float** variable. The Java compiler gives an error.



### ? Question

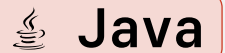
How would you correct the code?

### ? Question

How would you correct the code?

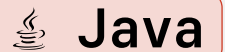
- You can write the value as a **float** literal:

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



- Alternatively, you can store the value in a **double** variable:

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

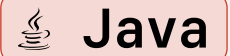


## 2.9 Constants

## 2. Simple Data Types

- We just had the example of the circle number  $\pi$ .
- In Java, there is the keyword to define constants.
- These can then no longer be changed.

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



- After a constant has been declared, it can no longer be changed.  
The following code would therefore generate an error:

```
1  PI = 3;
```

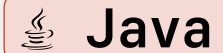


### Task 1

We now want to create a console output:

- Open IntelliJ IDEA and open or create a new executable class.
- Try the following code:

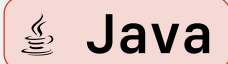
```
1 int age = 24;  
2 System.out.println(24);  
3 System.out.println(age);
```



### Task 2

- Using the “+” operator, you can combine text and variables:

```
1 int age = 24;  
2 System.out.println("My age is " + 24);  
3 System.out.println("My age is " + age);
```





### Tip

- Type `sout` in IntelliJ IDEA and press the Tab key. This saves time when writing `System.out.println()`!

### ? Question

What is a Coding Style? What does the term tell you?





### ? Question

What is a **Coding Style**? What does the term tell you?

- The coding style is a collection of rules that determine how code should be written.
- Uniform code is easier to read and maintain.

### ! Memorize

Compliance with the coding style will be evaluated in the exam!

- All names, and this applies to all identifiers, should be written in English!
- The following naming conventions should be followed:
  - Classes: `CamelCase`
  - Methods and variables: `camelCase`
  - Constants: `UPPER_CASE`
  - Packages: `lowercase`



### Tip

From my experience: Make your variables as meaningful as possible! Then the name can also be longer.

## 3. Comments and Identifiers

---

- As already mentioned, Java uses the Unicode character set.
- This means more characters are possible (65,536 to be exact).
- So you can write your comments in German, English, or Chinese without major restrictions.
- However, I would ask you to write your comments in **German** or **English**.

### ! Memorize

Since your keyboard doesn't have 65,536 characters, you can also copy and paste the characters. Alternatively for 😊:

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



Java

### ? Question

What do you think about the following statement? Why are comments important?

### ” Quote

Make the code readable? Who else is supposed to read it?

— Many Developers



- Comments are important for documenting code and improving maintainability.
- Both users of the code and the developers will need to understand the code. Comments are essential for this.

### ! Memorize

Not the quantity, but the quality of comments is crucial!  
Always comment directly while you are programming!

### ? Question

What is the difference between a **block comment** and a **line comment**?

### ? Question

What is the difference between a **block comment** and a **line comment**?

- **Line comments** start with `//` and end at the end of the line.
- **Block comments** start with `/*` and end with `*/`.

## 3.2 Comments

## 3. Comments and Identifiers

- Example of a line comment:

```
1 // This is a line comment
2 int distance; // Euclidean distance between a and b
```

**Java**

- Example of a block comment:

```
1 /* The calculation of the Euclidean distance
   follows these steps:
2     1. Calculate the difference of coordinates
3     2. Square the difference
4     ... */
```

**Java**

- All things that you name in Java are called **identifiers**. Many things you write need a name!

### ! Memorize

- Follow these rules for identifiers:
  - Letters, numbers, underscores, and dollar signs are allowed.
  - The first character may not be a number.
  - Case sensitivity is observed.
  - No spaces or keywords.
  - Not the literals `true`, `false`, or `null`.

## 3.3 Identifiers

## 3. Comments and Identifiers

- All reserved keywords 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	

### ? Question

Which of the identifiers are **allowed** in your opinion and why?

1     `int length;`

2     `int länge;`

3     `int maxLength;`

4     `int max_length;`

5     `int _max_length;`

6     `int max-length;`

7     `int !maxLength;`



**Java**



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

## 4. Operators

---

- There are the usual arithmetic operators.
- In general, operators are also evaluated from left to right.

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 (ergibt 3)</code>	2
+	Addition	<code>a = b + c</code>	3
-	Subtraktion	<code>a = b - c</code>	3

Figure 1: Arithmetic operators in Java

## 4.2 Increment and Decrement

- There are also the same operators for incrementing and decrementing as in C.

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

Figure 2: Operators for increment and decrement in Java

### ? Question

Think about it: What will appear on the console here?

```
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--);
```

**Java**

## 4.3 Comparison Operators

- 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

Figure 3: Vergleichsoperatoren in Java

### ? Question

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);
```

**Java**



## 4.4 Logical Operators

- 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

Figure 4: Logische Operatoren in Java

### ! Memorize

- 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.

### ? Question

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);
```

**Java**

## 4.5 Assignment Operators

- 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>&lt;op&gt;=</code>	Kombinierte Zuweisung: <code>a &lt;op&gt;= b</code> entspricht <code>a = a &lt;op&gt; b</code>	13

Figure 5: Zuweisungsoperatoren in Java

### ? Question

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);
```



Java

## 5. Type Conversion

---

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



### Warning

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?



### Idea

Sie können einfach schreiben, dass Sie das explizit wollen!

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



Java



### ? Question

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**

### ? Question

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

### ? Question

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.

### ! Memorize

- 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.



### Example

- `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

### ? Question

Zum Mitdenken: Welche der folgenden Zeilen werden kompilieren?

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

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

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



**Java**

## 5.2 Implicit Type Conversion

## 5. Type Conversion

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



**Java**

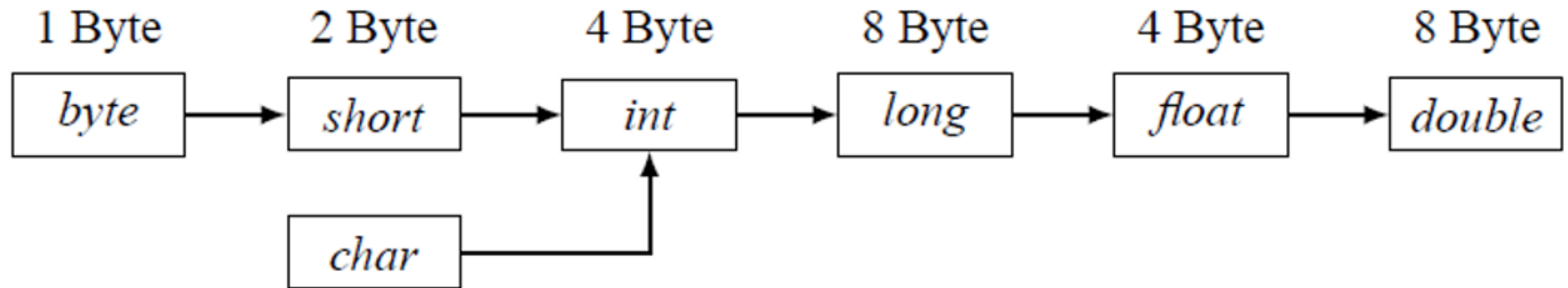


Figure 6: Implizite Typkonvertierung in Java



### ! Memorize

- 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!).

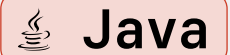
## 6. Control Structures

---

### ! Memorize

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  }
```

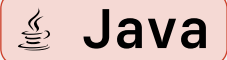


## 6.1 if Statement

## 6. Control Structures

- 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.

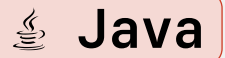
```
1  int a = 4, b = 8;  
2      int maximum = a;  
3  
4  if (b > maximum) {  
5      maximum = b;  
6  }
```



## 6.2 if-else Statement

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

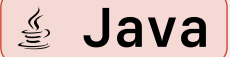
```
1  if (Bedingung) {  
2      Anweisungen 1  
3  } else {  
4      Anweisungen 2  
5  }
```



## 6.2 if-else Statement

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

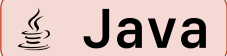
```
1  int a = 4, b = 8;
2  int maximum;
3
4  if (a > b) {
5      maximum = a;
6  } else {
7      maximum = b;
8  }
```



## 6.3 The ? 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;
```



### Task 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"





### Example

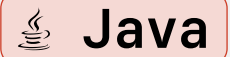
```
1 byte weekDay = 3;  
2  
3 if (weekDay <= 5) {  
4     System.out.println("Arbeiten");  
5 } else if (weekDay == 6) {  
6     System.out.println("Einkaufen");  
7 } else if (weekDay == 7) {  
8     System.out.println("Ausruhen");  
9 }
```



Java

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  }
```



- 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

### Task 4

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

```
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;
```



```
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 Loop

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.

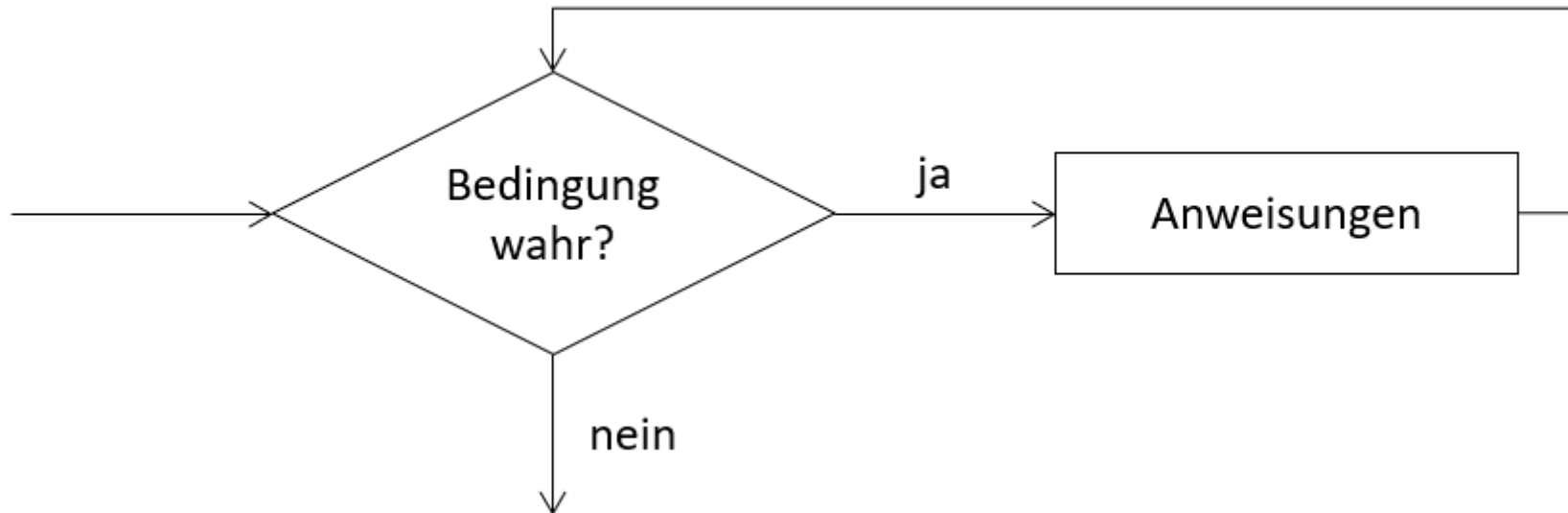


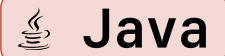
Figure 7: while-Schleife in Java



## 6.7 do-while Loop

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.

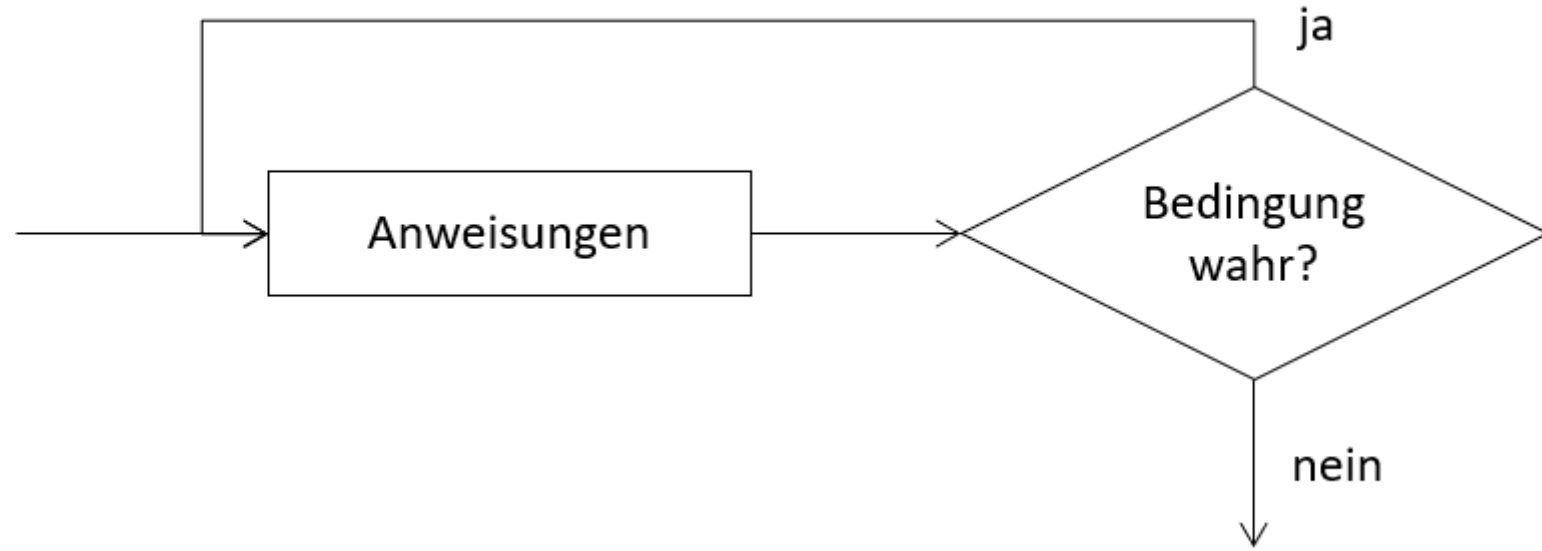
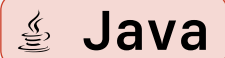


Figure 8: do-while-Schleife in Java

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.

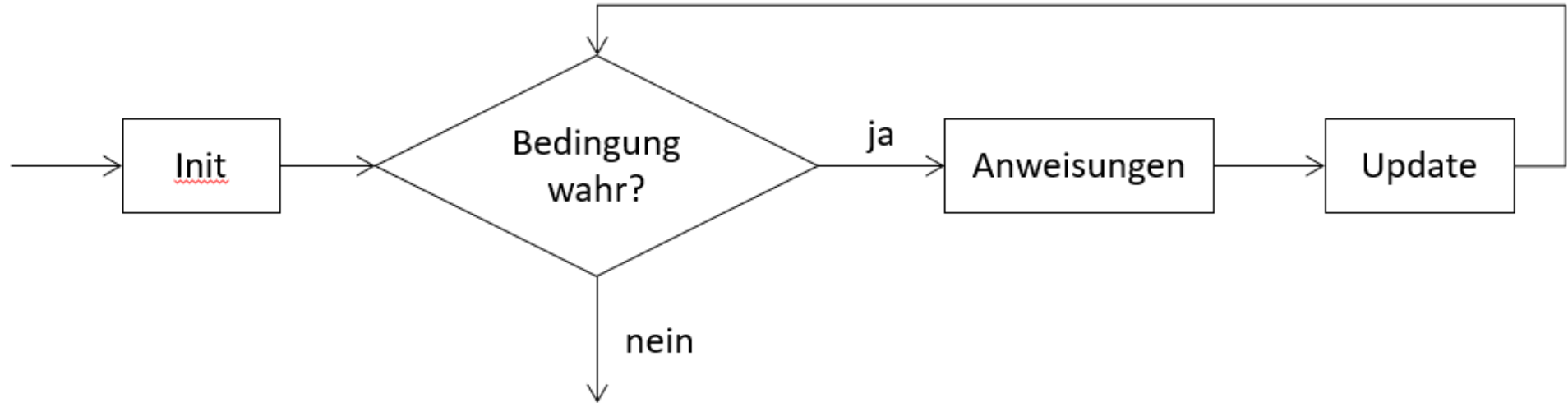


Figure 9: for-Schleife in Java

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

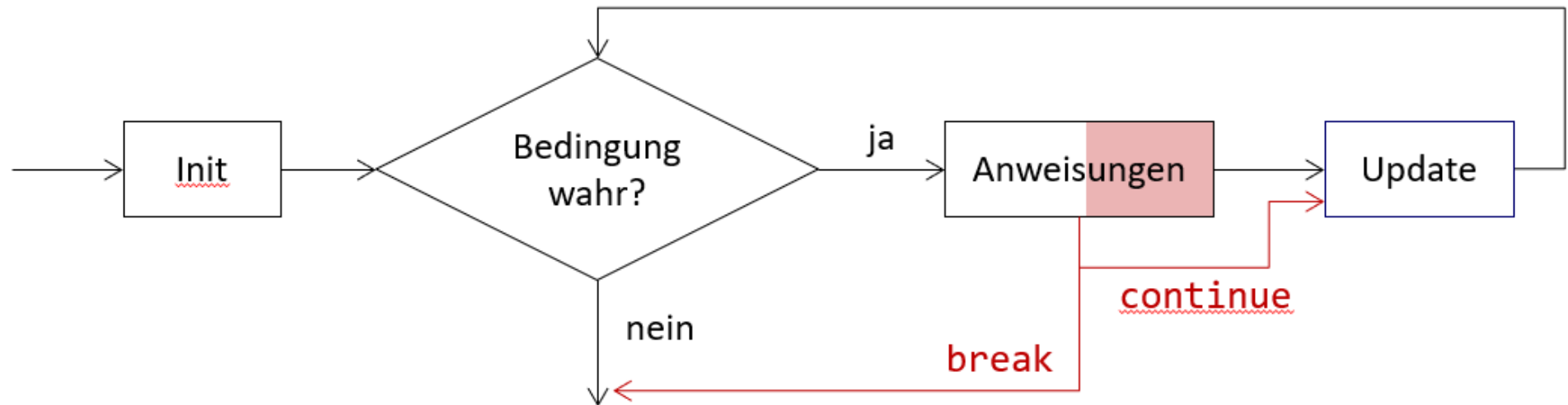


Figure 10: Visualisierung von `break` und `continue` in Java

### ? Question

Was passiert in dem folgenden Code?

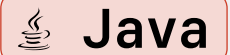
```
1 System.out.println("Break (bei i == 2):");
2 for (int i = 0; i <= 4; i++) {
3     if (i == 2) {
4         break;
5     }
6     System.out.println("  i = " + i);
7 }
```



### ? Question

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 }
```



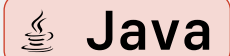
- Wie bereits erwähnt, ist der Coding Style wichtig. Daher gibt es bei den Kontrollstrukturen auch einen Coding Style.
- Öffnende geschweifte Klammern werden in der gleichen Zeile wie die Kontrollstruktur geschrieben (das gilt für alle öffnenden Klammern).
- Nach einer schließenden geschweiften Klammer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





### Example

```
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.