

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. License Notice	48

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.

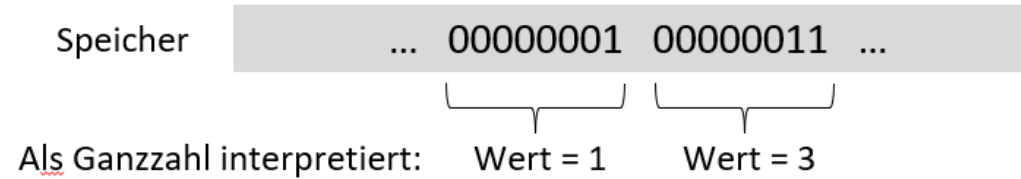


Abbildung 1: Speicher im Computer mit Werten aus dem Programm

? 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) 

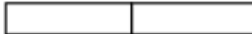
Ganzzahlen:

byte (1 Byte) 


short (2 Byte) 

int (4 Byte) 

long (8 Byte) 

char (2 Byte) 

Gleitkommazahlen:

float (4 Byte) 

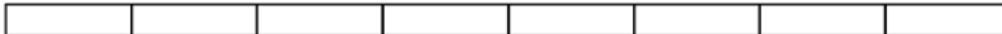
double (8 Byte) 

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	

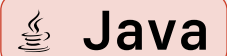
Abbildung 3: Wertebereiche der Datentypen in Java

! 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';
```

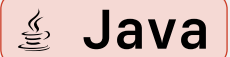


! Merke

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

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

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



- 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?

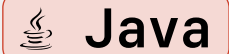
- Sie können den Wert als **float** Literal schreiben:

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



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

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

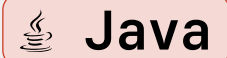


2.9 Konstanten

2. Einfache Datentypen

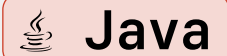
- Wir haben gerade bereits das Beispiel der Kreiszahl π gehabt.
- In Java gibt es das Schlüsselwort `final`, um Konstanten zu definieren.
- Diese können dann nicht mehr verändert werden.

```
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);
```



Java

☰ 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;



3.3 Bezeichner

3. Kommentare und Bezeichner

```
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 (ergibt 3)</code>	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.

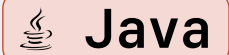
Operator	Art	Wert des Ausdrucks	Änderung von a
<code>++a</code>	Präfix	<code>a + 1</code>	<code>a = a + 1</code>
<code>a++</code>	<u>Postfix</u>	<code>a</code>	<code>a = a + 1</code>
<code>--a</code>	Präfix	<code>a - 1</code>	<code>a = a - 1</code>
<code>a--</code>	<u>Postfix</u>	<code>a</code>	<code>a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

- Es gibt auch die gleichen logischen Operatoren wie in C!

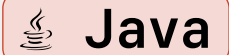
Operator	Bezeichnung	Priorität
<	kleiner	5
<=	kleiner oder gleich	5
>	größer	5
>=	größer oder gleich	5
==	gleich	6
!=	ungleich	6

Abbildung 3: Logische Operatoren 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);
```



5. License Notice

5.1 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.