

Morgen 18 Uhr Unterricht Cafe Campus

VL diese Woche: Di, Mi, Fr

VL nächste Woche: Di, Mi

Kap 3: Ausdrücke, Anweisungen, Kontrollstrukturen

↑
wichtige Sprachkonzepte in allen Prog. Sprachen

3.1 Variable und Objekte

Ausdrücke beinhalten Variable

$(x + 7) / y$
↑ ↑
Variable ↓
TextField Input

in Java 2 Arten

① Standardtypen

② Referenztypen

Standardtypen = eingebaute Typen

boolean ← Speicherplatz für Variable von diesem Typ
1 bit

char	←	16 bit	ganze Zahl ohne Vorzeichen
byte	←	8 bit	ganze für unicode Zeichen
short	←		Zahl mit Vorzeichen
int	←	16 bit	ganze Zahl mit Vorzeichen
long	←	32	-----
		64	-----
float	←	32 bit	Fließkomma Zahl
double	←	64 bit	-4-

Einheiten der Darstellung später

Bei Standardtypen wird unter dem Namen der Variablen stets der **WERT** angesprochen

int a = 1; ← weist int Variable a den Wert 1 zu
 int b = 2; ← b 2 zu
 a = a + b; ← Zuweisung an Variable a

↑
 Speicher
 inhalt den a belegt
 wird verändert
 Werte werden angesprochen

Alle anderen Typen in Java sind Referenztypen

- Klassentypen
 - Schnittstellentypen
 - Arraytypen
- } 3 Arten

Hier meint man nur Klassentypen

Jede Klasse stellt einen Typ dar. Man kann also

Variable von diesem Typ definieren. Solche Variable
(genauer: die Werte dieser Variablen) heißen in Java Objekte.

TextField input, output;
↑ ↑ ↑
Klasse, und Variable von diesem Typ
damit ein Typ

Unter dem Namen einer Variable vom Referenztyp wird stets
die **ADRESSE (REFERENZ)** angesprochen, nicht der Wert

Man braucht daher Methoden um

- (i) die Objekte im Speicher zu erzeugen
- (ii) die Werte / Daten einzugeben
- (iii) auf die Werte / Daten zuzugreifen (sie lesen)

in (i) Erzeugung erfolgt mit **new**-Anweisung und
dem Aufruf eines Konstruktors

↑ mit demselben Namen wie
die Klasse

output = new TextField(10);

↑ ↑
Konstruktor erwartet als Argument
eine int Zahl, die die Länge
in # Zeichen angibt (hier 10)

In einer Klasse gibt es i.A. mehrere Konstruktoren

```
output = new TextField("Coma I", 10);
```

↑
Argumente für Konstruktor
sind String und Länge

Jede Klasse hat einen Default Konstruktor
ohne Argumente. Dann wird ein "Standard"-Objekt erzeugt

```
output = new TextField();
```

(ii) Die Eingabe von Daten erfolgt über set-Methoden

```
output.setText("Hallo!");
```

↑
set Methode der Klasse TextField
setzt Text im Objekt output auf Hallo!

(ii) Der Zugriff auf Daten erfolgt über get-Methoden

```
String str = output.getText();
```

↑
get Methode (keine Argumente)

holt String aus Objekt output
und gibt ihn als Wert zurück
dieser Wert wird der String Variable str zugewiesen

(kleinere Ausnahmen bei Strings, später mehr)

in Ausdrücken wird unter
Namen eines Strings der Wert ausgedrückt

```
output.setText ( str + "Wie gehts?" );
```

Ausdruck, der String ergibt

Konstruktor möchte man mit den eingebauten Standardtypen
umgehen wie Klassentypen

Dann existiert für jeden eingebauten Typ ein korrespondierender
Klassentyp ("Wrapper class")

int ↔ Integer

double ↔ Double

boolean ↔ Boolean

↑
Klassen beginnen mit Großbuchstaben
Variable sollten wie mit Großbuchstaben
beginnen

Beispiel für Wrapper Class:

```
int a = 64;
```

```
Integer myInt = new Integer(a);
```

↑ erzeugt Integer Objekt mit Wert
des Variablen a, also 64

~~int b = myInt;~~

↑ Wert wird erwartet
↖ Referenz wird gegeben

int b = myInt.intValue();

↑
get Methode, aus historischen Gründen
so genannt

holt Wert (64) aus Objekt myInt,
weist ihn der int Variablen b zu

Beispiel mit Speicherbildern

TextField input, output;

String str = new String("Hallo!");

im Speicher

input null

null = leere Adresse

output null

str • → Hallo!

↑
Speicher für Adressen
(Referenzen)

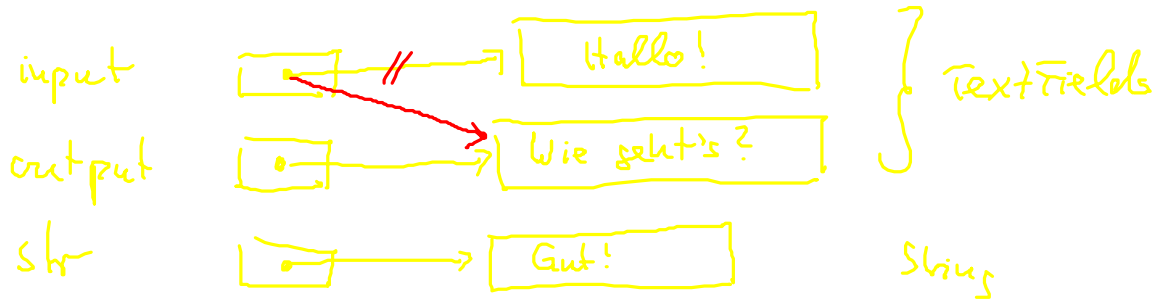
↑
Speicherplatz für Objekt

↑
symbolisiert die Referenz

input = new TextField(str, 10);

```
output = new TextField("Wie geht's?");
```

```
str = "Gut!";
```



```
input = output; Wirkung in Rot
```

input und output sprechen dasselbe Objekt an!

Wie Text aus output in input schreiben?

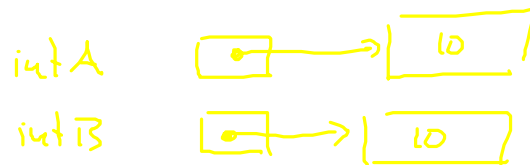
get und set Methoden!

```
input.setText(output.getText());
```

Noch ein Beispiel:

```
Integer intA = new Integer(10);
```

```
Integer intB = new Integer(10);
```



String compare = (intA == intB) + "uu" + intA.equals(intB);

⊗ ergibt String

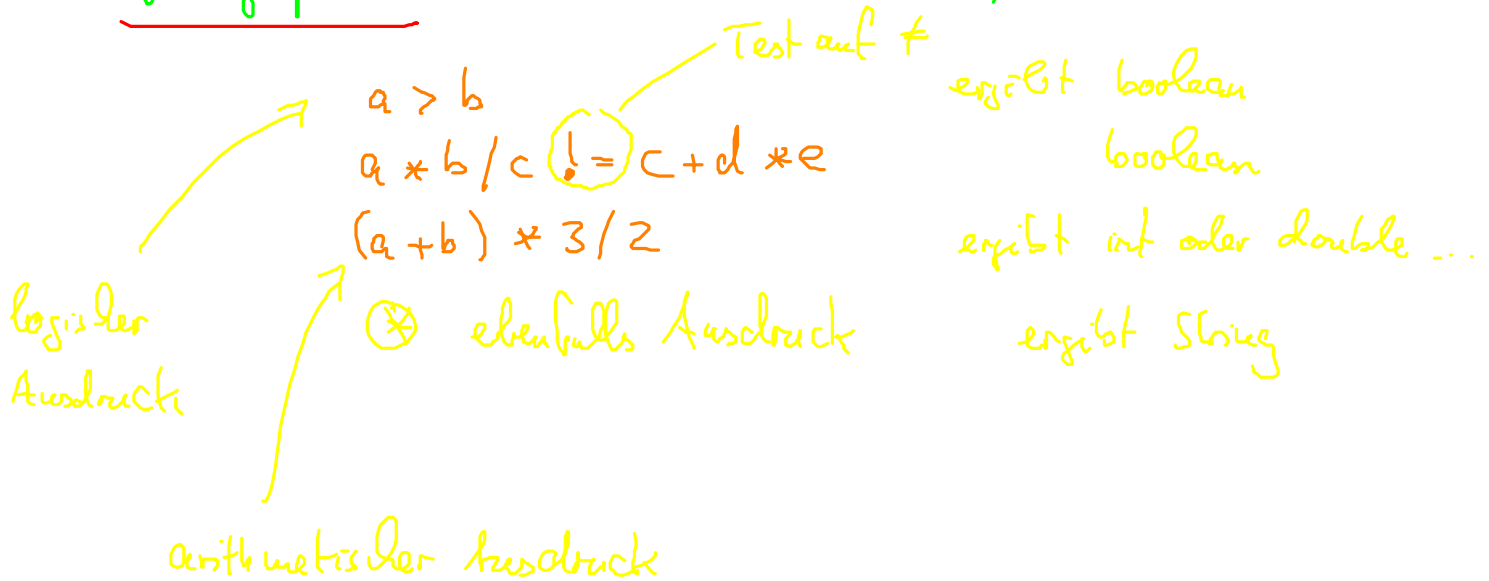
↑
Methode
equals() der
Klasse Integer
vergleicht Wert
von intA mit
intB

compare enthält dann den Text:

false_u_true

3.2 Ausdrücke

grob gesprochen: Ausdruck ist Formel, die einen Wert ergibt



"Verfahren" zwischen Operatoren notfalls mit (...) regeln

$(a * b / c) != (c + d * e)$

genaue Erklärung

wichtig für Feinheiten in Java
(alles gerät aus C)

Java unterscheidet zwischen

$lvalue$ = Ausdruck, der links von Zuweisungszeichen stehen kann, bezeichnet Speicherplatz
 $rvalue$ = Ausdruck allgemein

Ausdruck hat 3 Eigenschaften

- Wert (Rückgabewert) = Wert nach Auswertung
- Typ = Typ des Wertes
- Effekt = Wirkung auf Speicherinhalte
ist diese Wirkung anders als eine Zuweisung an einen $lvalue$, so spricht man von Seiteneffekt

Beispiele:

1. $a = b$ ← Zuweisungsausdruck, weist $lvalue$ Wert des
↑ $rvalue$ zu
 $lvalue$

Wert = Wert, der der linken Seite zugewiesen wird

Typ = Typ der linken Seite

Effekt = Veränderung des Speichers, der durch a bezeichnet wird

2. $c += b = a$

↑
Zuweisungsoperator $c += b \hat{=} a = a + b$

b = a wird zunächst ausgewertet, der Wert wird zu c hinzugefügt und c zugewiesen

a	1	1	
b	5	1	← Seiteneffekt
c	2	3	
	vorher	hierher	

Wert des gesamten Ausdrucks ist 3

Typ ... ist der Typ von c

Effekt: c wird verändert.

b wird verändert ← Seiteneffekt

anpassen

besser schreiben als

$$\left. \begin{array}{l} b = a; \\ c += b; \end{array} \right\} \text{ 2 Effekte, kein Seiteneffekt}$$

3. $\text{if } ((a=b) > c) \text{ } c = a;$

a	2	4	← Seiteneffekt
b	4	4	
c	3	4	
	vorher	nachher	

↑
nach Möglichkeit vermeiden !!

⇔

$\text{if } ((a=b) > c) \{$

} $C = a_i$