Kap 3: Ausdrücke, Anweisungen + Kontrollstrukturen

Erinnerung:

In Java: 1. Standarttypen (primitive Typen)
2. Referenztypen

Standardtypen	Wertebereich		
boolean	true, false		
char	alle Unicode-Zeichen (der zugehörige Zahlenwert $0, \ldots, 2^{16}-1$)	16 bits	
byte	$-128, \ldots, 127$	8 bits	ganze Zahlen
short	$-2^{15}, \ldots, 2^{15}-1$	16 bits	
int	$-2^{31}, \ldots, 2^{31}-1$	32 bits	
long	$-2^{63}, \ldots, 2^{63}-1$	64 bits	
float		32 bits	Fließkommazahlen
double		64 bits	

Bei arithmetischen Ausdrücken wird der Typ des Ausdrucks immer durch den „breitesten" vorkommenden Typ bestimmt.

z. B. 5 / 11 ergibt 0

5.0 / 11 ergibt 0.55....56

Typen können „überlaufen" (Wertebereich wird verlassen)

```
byte b = 127;
b++;
```
← Compiler fängt das nicht ab

Manchmal möchte man explizite Typumwandlungen machen,
z.B. int-Zahl als double
geht durch "casten" in Java mit (Typ)

Bsp:
```
(double) x / 5
```
↑
stellt sicher, dass der Wert von x als double interpretiert wird. Dadurch reellwertige Division.

später mehr zum casten von Klassen

> Bei Standardtypen spricht man über den Variablennamen immer den Wert der Variablen an.

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

```
a = a + b;
```

Werte werden angesprochen

der Wert von a wird verändert.

Alle anderen Typen sind Referenztypen

z.B.
- Klassentypen
- Arraytypen
- Schnittstellentypen
- Enumerationstypen

zunächst nur Klassentypen

Jede Klasse ist in Java ein Typ.

```
Scanner scan;
```

Klasse, also auch Typ

Variable vom Typ Scanner

```
Character chara;
String str;
```

Die „Werte" eines Klassentyps heißen Objekte.

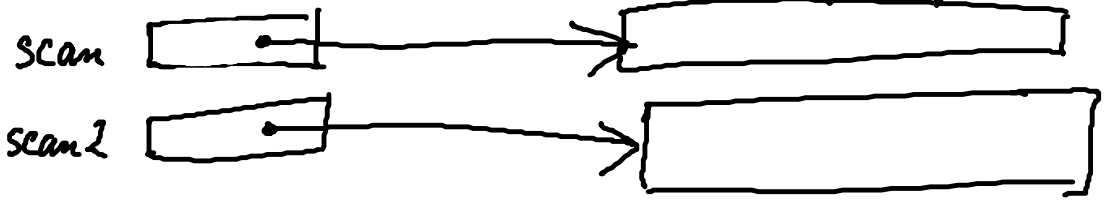
Bei Referenztypen wird unter dem Variablennamen die Speicheradresse (Referenz) für das Objekt angesprochen (also nicht der Wert).

```
Scanner scan2 = new Scanner(System.in);
scan = new Scanner(System.in);
```

Konstruktor, erzeugt Objekt

erzeugt neues Objekt vom Typ Scanner

Speicherbild:



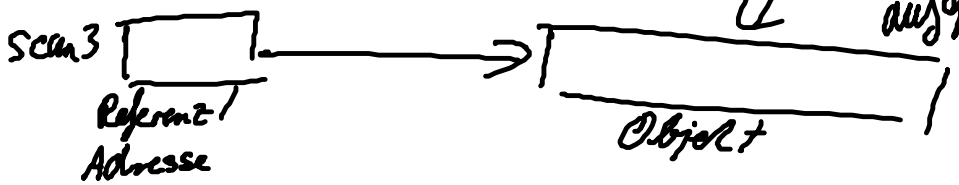
Erzeugung von Objekten mit new + Konstruktor:

```
Scanner scan3 = new Scanner(System.in);
```

Referenz/Adresse

Operator "new" gibt Referenz auf ein neues Objekt

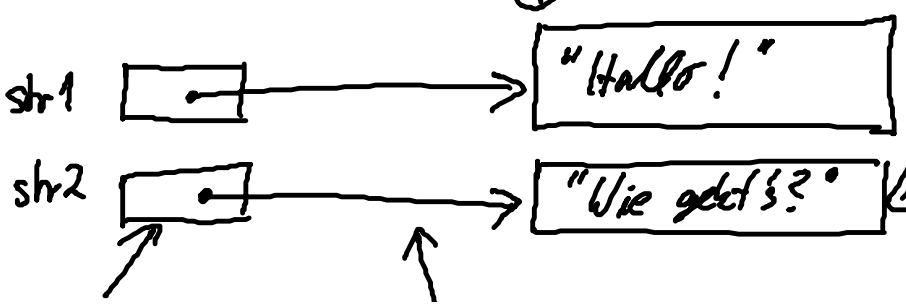
Konstruktor der Klasse Scanner, wird hier mit Parameter System.in aufgerufen




```
String str1 = new String("Hallo!");
String str2 = new String("Wie geht's?");
```

Speicherbild:

new erzeugt Objekt vom Typ String



Speicherplatz mit Adresse

Verweis, Referenz, Zeiger

str1.length() ≙ 6

Über die Referenzen greifen wir auf die Objekte → brauchen Methoden um mit Objekten zu arbeiten.

z.B. bei String { Länge eines Strings berechnen: length() ; Strings vergleichen: equals()

bei Scanner: Kommandozeilen-Eingabe zu lesen: nextInt(), nextDouble()

str1.length()

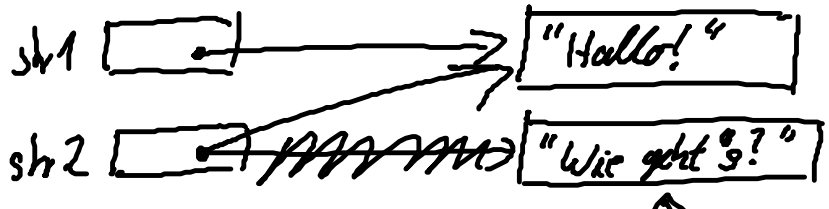
length() - Methode der Klasse String

Punkt-Operator, über ihn wird die Methode auf das Objekt, auf das str1 verweist, angewendet.

n = scan.nextInt();

Methode der Klasse Scanner

wird auf Scanner Objekt, auf das scan zeigt, angewendet.

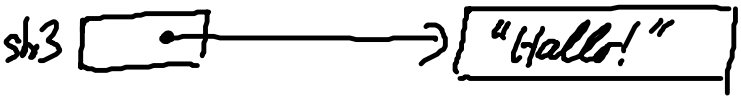


str2 = str1; // hängt neu Referenzen um

Objekt ist nun "verwaist" → Garbage Collector

str1 und str2 sprechen das gleiche Objekt an.

String str3 = new String ("Hallo!");



Test auf Gleichheit:

str1 == str3 ergibt "false", die Adressen verglichen werden

Alternative:

str1.equals(str3) ergibt "true"

- Vergleich mit == zwischen Variablen von Referenztyp vergleicht die Referenzen, nicht die Werte.
- Vergleich der Werte erfolgt mit equals(), sofern die Klasse eine solche Methode implementiert.

Zugeständnis an C-Programmierer:

String str1 = new String ("Hallo!");
String str1 = "Hallo!";

} beide Aussagen sind gleichbedeutend (Ausnahme für Klasse String)

Speicherbilder sind gleich!

Problem: Manchmal möchte man Standardtypen wie Referenztypen behandeln, um z.B. Methoden für sie zur Verfügung zu stellen.

Lösung: Wrapper-Klassen

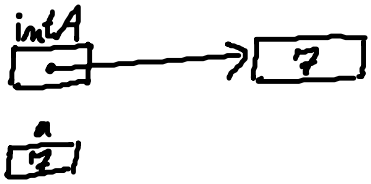
int	Integer
double	Double
long	Long
boolean	Boolean

↑ Standardtyp

↑ Wrapper-Klasse, implementiert auch Methoden, die auf Standardtypen arbeiten

Erzeugung von Wrapper Objekten mit new + Konstruktor:

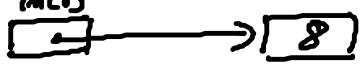
```
Integer intA = new Integer(7);
int a = 7;
```




```
int b = intA.intValue();
```

"get-Methode" der Wrapperklasse Integer
("greift" Wert des Objekts auf das intA zeigt)

```
Integer intB = new Integer(8);
```



Die Werte auf die intA und intB zeigen } zu addieren, war bis Java 1.4 sehr kompliziert.

```
int c = intA.intValue() + intB.intValue(); // c = 7+8
```

(muss so sein, gilt immer noch)

Seit Java Version 1.5 ist Folgendes erlaubt:

```
int c = intA + intB;
```

implizite Typumwandlung (Casting) zwischen Standardtyp und zugehöriger Wrapperklasse

boxing

```
Integer intA = 7;
```

Integer intB = 8;

int c = intA + intB;

↑ unboxing

"auto boxing"