PGdP Tutorium: Dritte Stunde

Benedikt Werner

München, 08. November 2018

Hausaufgaben

- Probleme oder Fragen zu den Hausaufgaben?

- Zentralübung: Donnerstag 20 Uhr, MI HS 1
- Hausaufgabenbesprechung: Samstag 12-16 Uhr, MI HS 3

- Stück für Stück vorgehen
- Teilprobleme lösen

- Lösungen anschauen und verstehen
- Nachfragen!

Expressions

- An *expression* is a construct made up of
  - variables,
  - operators,
  - and method invocations,

- which are constructed according to the syntax of the language, that evaluates to a single value.

Expressions

```
42
```

```
7 * 191
```

```
3.5 + 1 / 12
```

```
"Hello World"
```

```
true
```

```
1 > 2
```

```
x
```

```
abc == def
```

```
author.getName()
```

```
Math.sqrt(16)
```

```
x = 3
```

```
2 * 3.5 + "abc"
```

Expressions

- Jeder Ausdruck bzw. jede Expression hat einen
  - eindeutigen Typ
  - und einen eindeutigen Wert.

Ausdruck	Typ	Wert
`3 + 5`	`int`	`7`
`"Hello World"`	`String`	`"Hello World"`
`3 > 2`	`boolean`	`false`
`x = 3`	`int`	`3`
`0.5 * (6 + 3) + "a"`	`String`	`"4.5a"`

Statements

- Statements are roughly equivalent to sentences in natural languages.
- A statement forms a complete unit of execution.

- Statements können alleine in einer Zeile stehen kann
- Alles was mit einem ; endet oder { } enthält

Statements

```
int x;

x = 3;

String test = "Hello World";

System.out.println(test);

if (x > 3) {
  x = 3;
}
```

Einfache Datentypen

Datentyp	Bedeutung
byte	Bytes, Ganzzahl 0-255
short	Kleine Ganzzahl
int	Ganzzahl
char	Buchstabe, gleichzeitig Ganzzahl
long	Sehr große Ganzzahl
float	Kommazahl
double	Präzisere Kommazahl
String	Text
boolean	Wahrheitswert

Einfache Operatoren

Operator	Bedeutung	
*	Multiplikation	
/	Division	
+, -	Addition, Subtraktion	
<, <=, >, >=	Größenvergleiche	
==, !=	Gleichheit, Ungleichheit	
~	Bitweises Komplement	
&	Bitweises Und	AND
^	Bitweises Entweder-Oder	XOR
|	Bitweises Oder	OR
<<, >>, >>>	Bitshift	

Boolean Operatoren

Operator	Bedeutung	
!	Nicht	NOT
&&	Und	AND
||	Oder	OR

!true	Nicht Wahr	false
!false	Nicht Falsch	true
3 < 5 && 4 == 4	3 < 5 und 4 == 4	true
3 < 5 && 4 == 6	3 < 5 und 4 == 6	false
3 < 5 || 3 < 6	3 < 5 oder 3 < 6	true
3 < 2 || 3 < 7	3 < 2 oder 3 < 7	true
3 == 4 || 3 == 5	3 == 4 oder 3 == 5	false

Basistypen vs Referenztypen

- Basistypen werden direkt als Wert gespeichert
  - int, char, double, boolean, etc.
  - Nicht String!
  - Im Prinzip alles was klein geschrieben wird

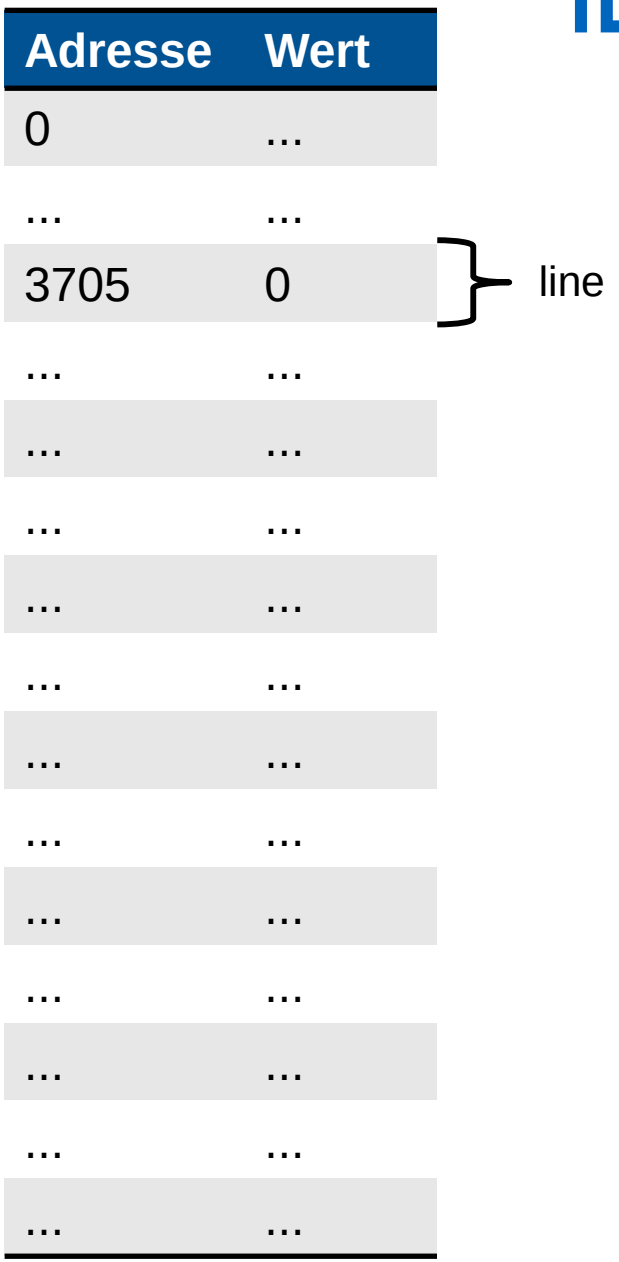
- Bei Referenztypen wird nur eine Referenz gespeichert
  - Die Referenz ist die Speicheradresse des Objekts

Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line;
```

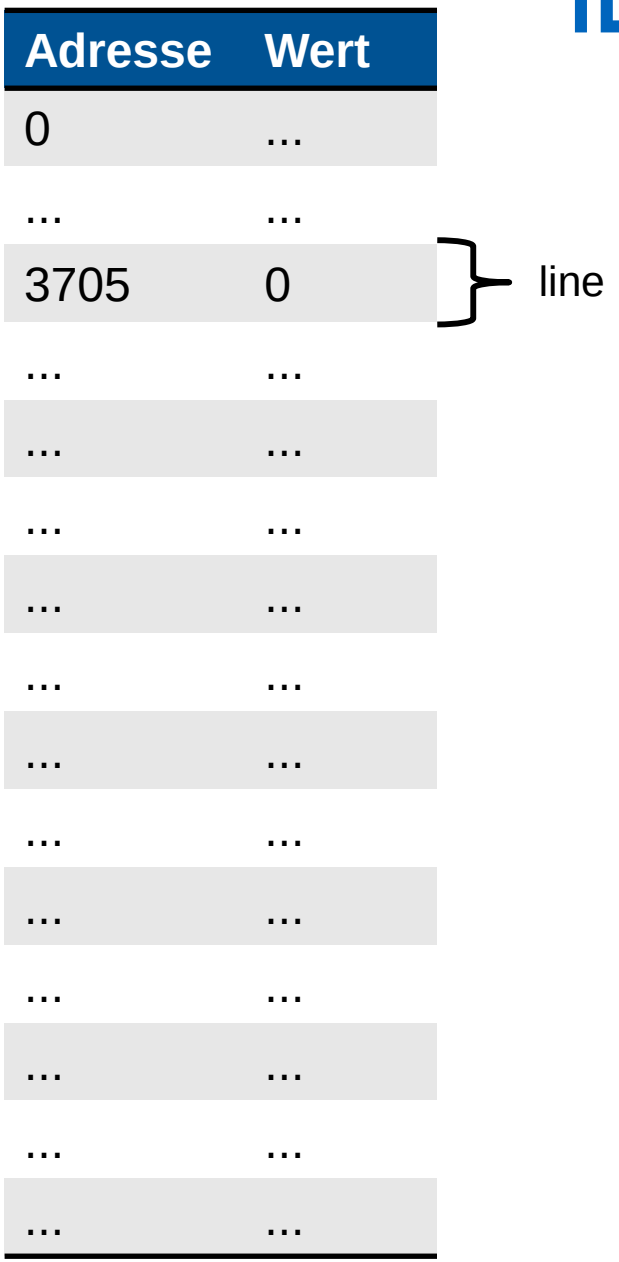


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
```

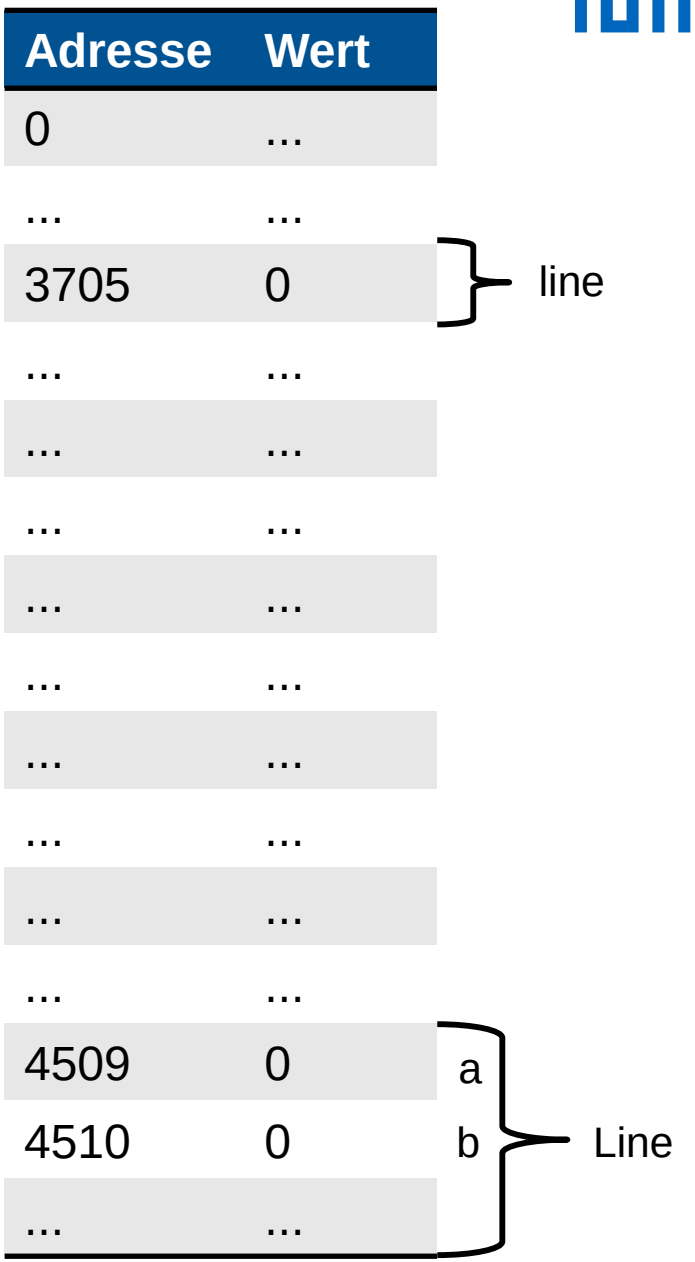


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
```

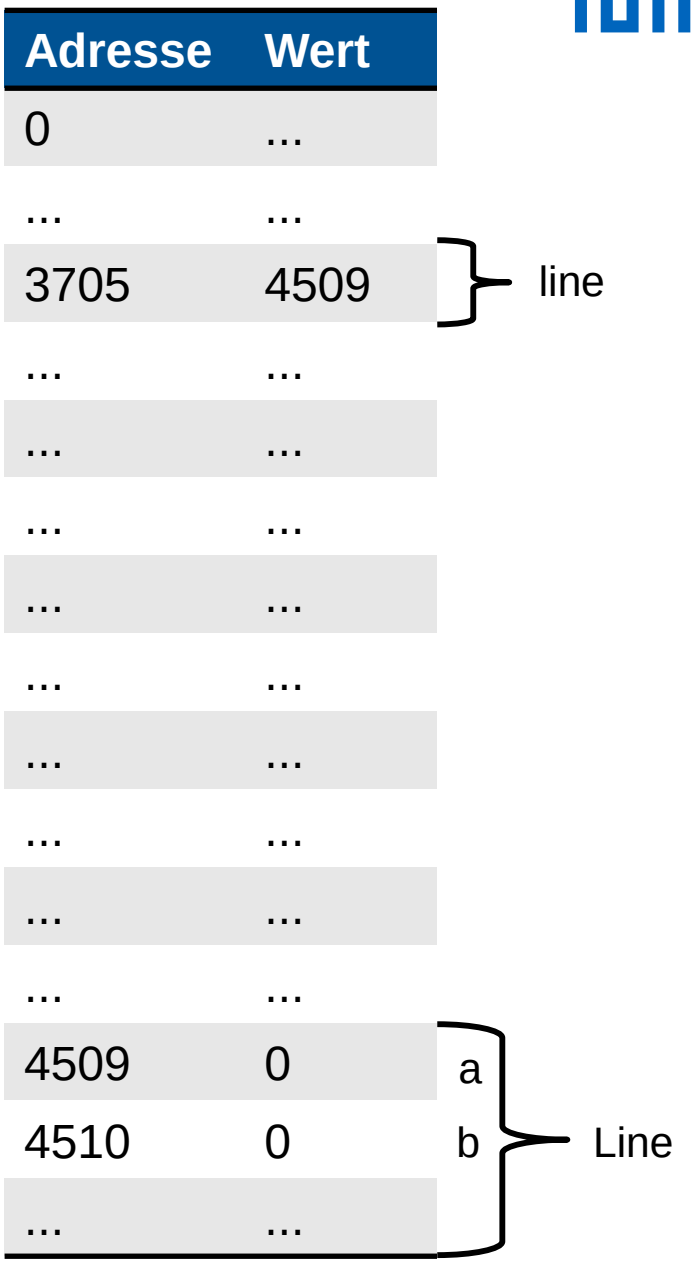


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
```

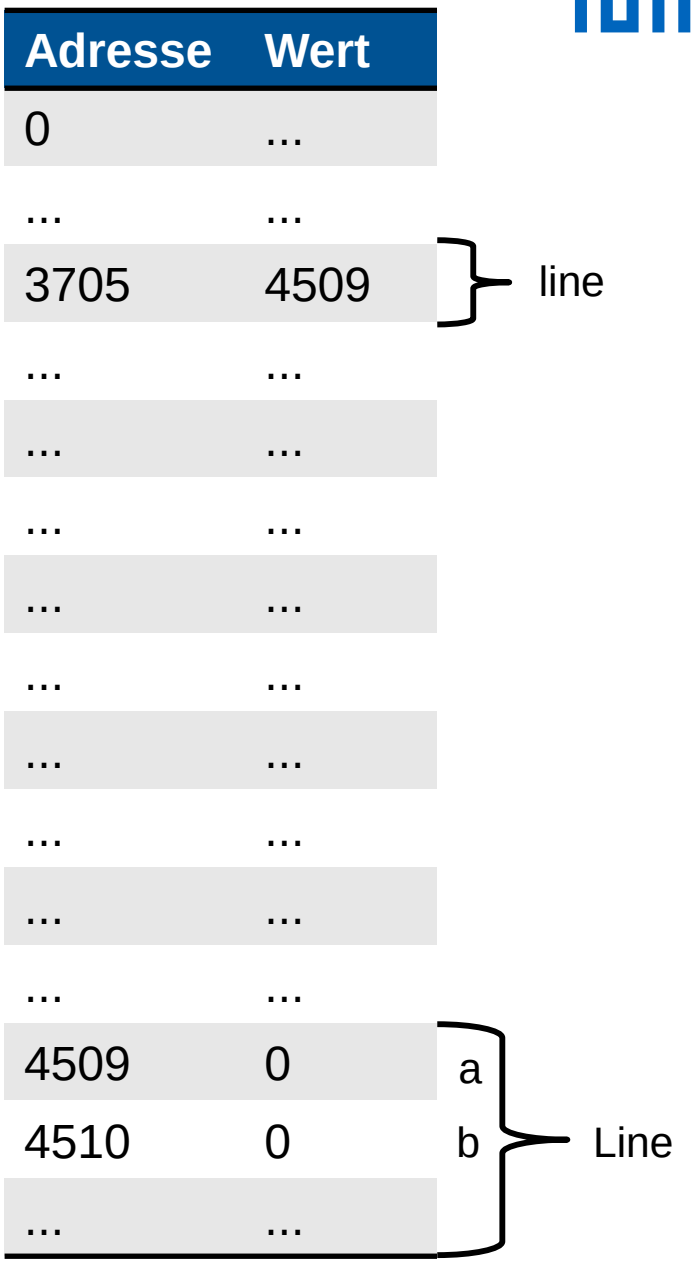


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
line.a = new Point(3, 5);
```

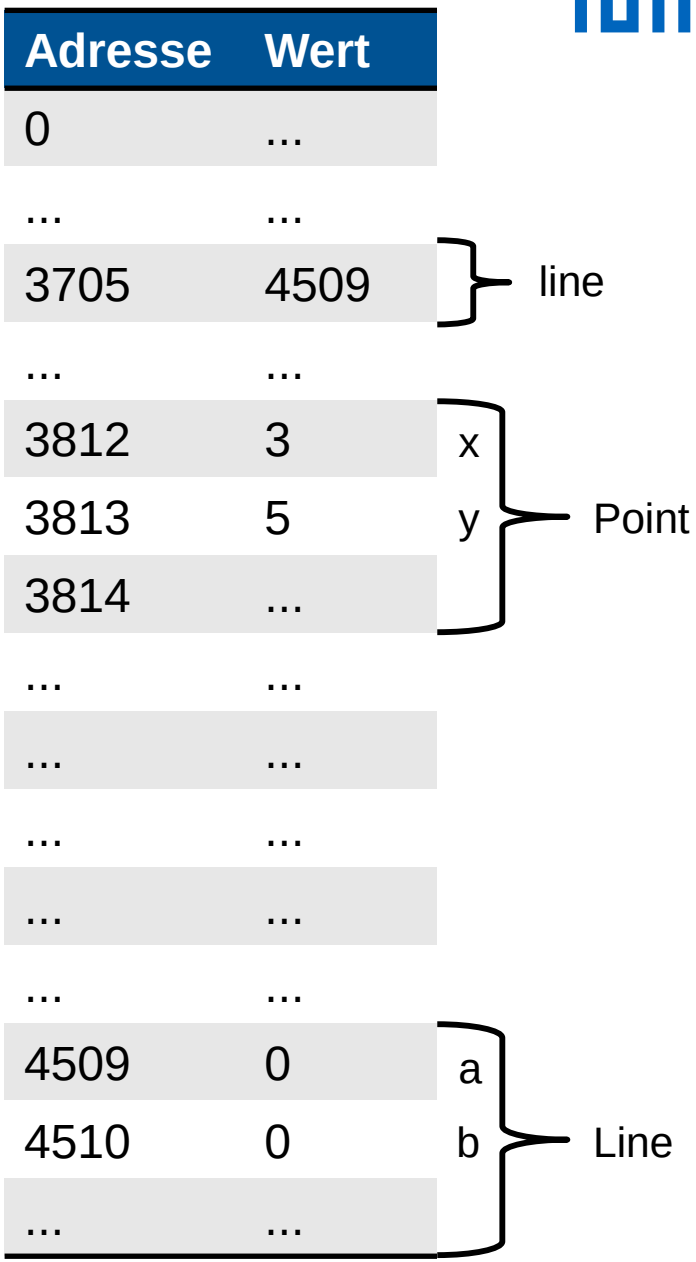


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
line.a = new Point(3, 5);
```

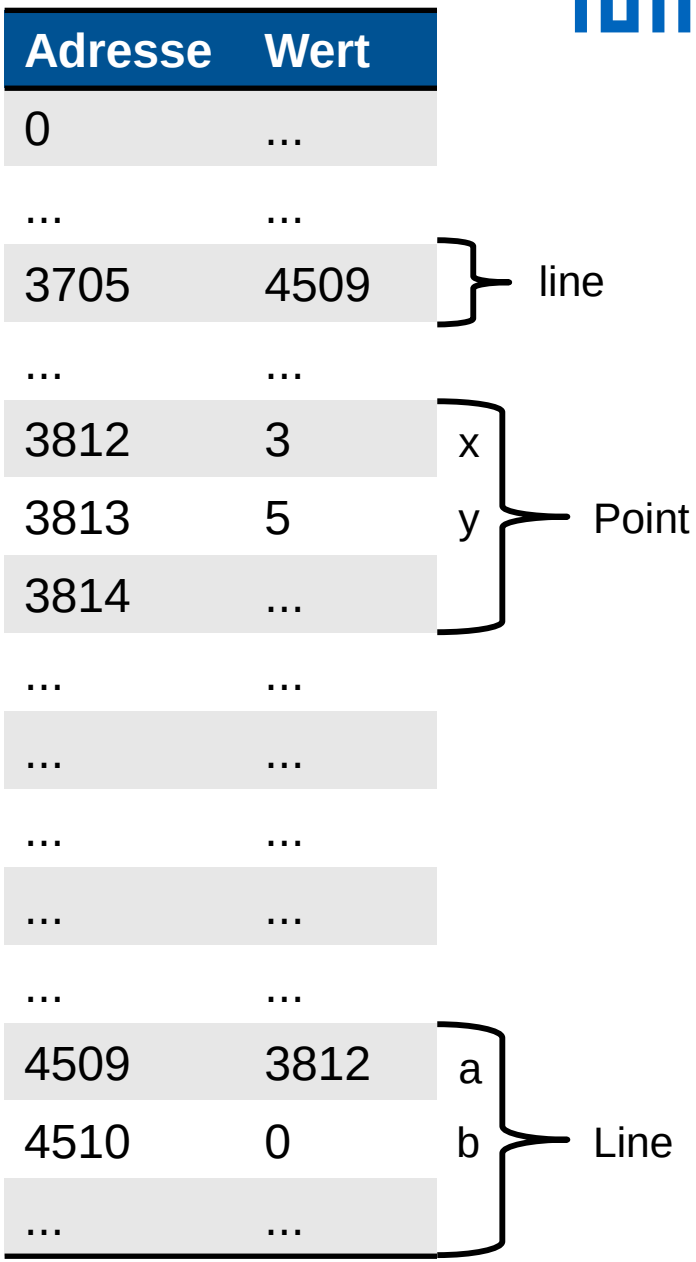


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
line.a = new Point(3, 5);
```

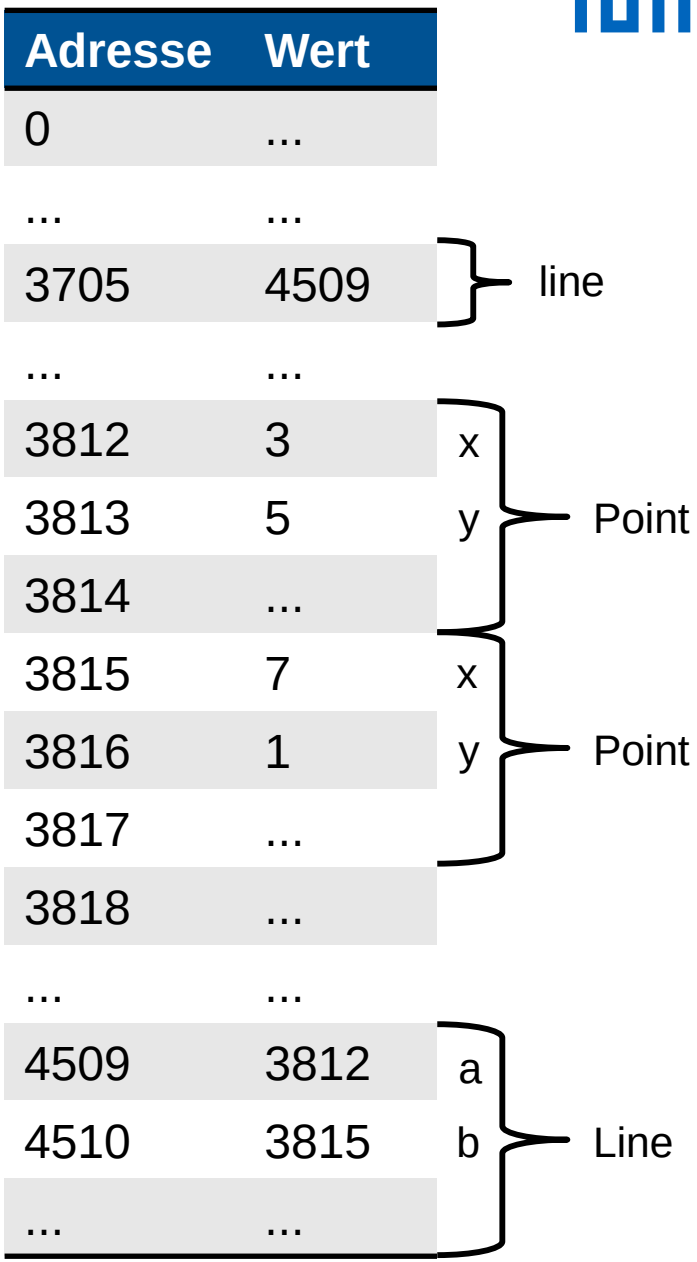


Referenztypen

```
public class Point {
  int x, y;
}

public class Line {
  Point a, b;
}

Line line = new Line();
line.a = new Point(3, 5);
Line.b = new Point(7, 1);
```



Getter und Setter

```
public class Person {
  private int age;

  public int getAge() {
    return age;
  }

  public void setAge(int newAge) {
    if (newAge >= 0)
      age = newAge;
  }
}
```

- Attribute `private`
- Getter und Setter `public`
- Wenn readonly nur Getter

If-Statement

```
if (condition) {
    // do something
}
```

```
if (condition) {
    // do something
}
else {
    // !condition
}
```

```
if (condition1) {
    // do something
}
else if (condition2) {
    // …
}
…
else {
    // nothing is true
}
```

Unit Testing

- Testmethoden, die eine andere Methode testen
- Jeden möglichen „Ausführungsweg“ der Methode testen
- Rand- und Sonderfälle prüfen

```
public int method(int x) {
    if (x > 42) {
        if (x == 1337)
            return x + 2000;
        return (int) Math.sqrt(x);
    }
    return x * 10;
}
```

Strings und Characters

```
String myString = "Hello World!";
char myChar = 'c';
```

Characters sind im Prinzip Zahlen:

```
char c = 'A';
char c = 65;

int number = c;          number = 'B';

c += 3;                  'B' < 'C'
```

ASCII Tabelle

Scan-code	ASCII hex	ASCII dez	Zeichen
	00	0	NUL ^@
	01	1	SOH ^A
	02	2	STX ^B
	03	3	ETX ^C
	04	4	EOT ^D
	05	5	ENQ ^E
	06	6	ACK ^F
	07	7	BEL ^G
0E	08	8	BS ^H
0F	09	9	TAB ^I
	0A	10	LF ^J
	0B	11	VT ^K
	0C	12	FF ^L
1C	0D	13	CR ^M
	0E	14	SO ^N
	0F	15	SI ^O
	10	16	DLE ^P
	11	17	DC1 ^Q
	12	18	DC2 ^R
	13	19	DC3 ^S
	14	20	DC4 ^T
	15	21	NAK ^U
	16	22	SYN ^V
	17	23	ETB ^W
	18	24	CAN ^X
	19	25	EM ^Y
	1A	26	SUB ^Z
01	1B	27	Esc ^[
	1C	28	FS ^\
	1D	29	GS ^]
	1E	30	RS ^^
	1F	31	US ^_

Scan-code	ASCII hex	ASCII dez	Zch.
	20	32	SP
02	21	33	!
03	22	34	"
29	23	35	#
05	24	36	$
06	25	37	%
07	26	38	&
0D	27	39	'
09	28	40	(
0A	29	41	)
1B	2A	42	*
1B	2B	43	+
33	2C	44	,
35	2D	45	-
34	2E	46	.
08	2F	47	/
0B	30	48	0
02	31	49	1
03	32	50	2
04	33	51	3
05	34	52	4
06	35	53	5
07	36	54	6
08	37	55	7
09	38	56	8
0A	39	57	9
34	3A	58	:
33	3B	59	;
2B	3C	60	<
0B	3D	61	=
2B	3E	62	>
0C	3F	63	?

Scan-code	ASCII hex	ASCII dez	Zch.
	40	64	@
1E	41	65	A
30	42	66	B
2E	43	67	C
20	44	68	D
12	45	69	E
21	46	70	F
22	47	71	G
23	48	72	H
17	49	73	I
24	4A	74	J
25	4B	75	K
26	4C	76	L
32	4D	77	M
31	4E	78	N
18	4F	79	O
19	50	80	P
10	51	81	Q
13	52	82	R
1F	53	83	S
14	54	84	T
16	55	85	U
2F	56	86	V
11	57	87	W
2D	58	88	X
2C	59	89	Y
15	5A	90	Z
	5B	91	[
	5C	92	\
	5D	93	]
29	5E	94	^
35	5F	95	_

Scan-code	ASCII hex	ASCII dez	Zch.
0D	60	96	`
1E	61	97	a
30	62	98	b
2E	63	99	c
20	64	100	d
12	65	101	e
21	66	102	f
22	67	103	g
23	68	104	h
17	69	105	i
24	6A	106	j
25	6B	107	k
26	6C	108	l
32	6D	109	m
31	6E	110	n
18	6F	111	o
19	70	112	p
10	71	113	q
13	72	114	r
1F	73	115	s
14	74	116	t
16	75	117	u
2F	76	118	v
11	77	119	w
2D	78	120	x
2C	79	121	y
15	7A	122	z
	7B	123	{
	7C	124	|
	7D	125	}
	7E	126	~
53	7F	127	DEL

String-Methoden

```
char charAt(int i):
char c = "Hello World!".charAt(1); // c = 'e'


int length():
int len = "Hello World!".length(); // len = 12
```

While-Schleife

```
while (condition) {
    // do something
    // repeat until condition is not true anymore
}
```

```
int x = 2;
while (x < 100) {
    x = x * 2;
    System.out.println(x);
}
```

For-Schleifen

```
for (int i = 0;     i < 5;     i++) {
    // do something 5 times
}
```

```
int i = 0;

while (i < 5) {
    // do something

    i++;
}
```

For-Schleifen

```
for (int i = 0;     i < text.length();     i++) {

    char c = text.charAt(i);

    // do something with c
}

for (char c = 'a';      c <= 'z';      c++) {
    // do something for every letter a-z
}
```

break

```
for (int i = 0; i < 2000; i++) {
  if (i * i == 1787569) {
      System.out.println(i+"*"+i + "= 1.787.569");
      break;
  }
}
```