Vorlesung

TECHNISCHE HOCHSCHULE LÜBECK

Programmieren I und II

Unit 2

Grundlagen imperativer Programmierung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
1

1

Disclaimer

TECHNISCHE HOCHSCHULE LÜBECK

Zur rechtlichen Lage an Hochschulen:

Dieses Handout und seine Inhalte sind durch den Autor selbst erstellt. Aus Gründen der Praktikabilität für Studierende lehnen sich die Inhalte stellenweise im Rahmen des Zitatrechts an Lehrwerken an.

Diese Lehrwerke sind explizit angegeben.

Abbildungen sind entweder selber erstellt, als Zitate kenntlich gemacht oder unterliegen einer Lizenz, die nicht die explizite Nennung vorsieht. Sollten Abbildungen in Einzelfällen aus Gründen der Praktikabilität nicht zweifelsfrei als Zitat kenntlich sein, so ergibt sich die Herkunft immer aus ihrem Kontext: „Zum Nachlesen ...“.

Creative Commons:

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
2

2

TECHNISCHE HOCHSCHULE LÜBECK

Prof. Dr. rer. nat. Nane Kratzke

Praktische Informatik und betriebliche Informationssysteme

- **Raum: 17-0.10**
- **Tel.:** 0451 300 5549
- **Email:** nane.kratzke@th-luebeck.de

@NaneKratzke

Updates der Handouts auch über Twitter #prog_inf und #prog_itd

3

Units

TECHNISCHE HOCHSCHULE LÜBECK

1. Semester	Unit 1 Einleitung und Grundbegriffe	Unit 2 Grundlagen imperativer Programmierung	Unit 3 Selbstdefinierbare Datentypen und Collections	Unit 4 Einfache I/O Programmierung
	Unit 5 Rekursive Programmierung, rekursive Datenstrukturen, Lambdas	Unit 6 Objektorientierte Programmierung und UML	Unit 7 Konzepte objektorientierter Programmiersprachen, Klassen vs. Objekte, Pakete und Exceptions	Unit 8 Testen (objektorientierter) Programme
2. Semester	Unit 9 Generische Datentypen	Unit 10 Objektorientierter Entwurf und objektorientierte Designprinzipien	Unit 11 Graphical User Interfaces	Unit 12 Multithread Programmierung

4

Abgedeckte Ziele dieser UNIT

TECHNISCHE HOCHSCHULE LÜBECK

Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmus-begriffs, Implementieren einfacher Algorithmen	Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)	Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Mulithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
5

5

Themen dieser Unit

TECHNISCHE HOCHSCHULE LÜBECK

Datentypen	Operatoren	Kontrollstrukturen	Routinen
• Werte • Variablen • Wertetypen	• Ausdrücke • Arithmetisch • Relational • Logisch • Bedingte Auswertung • Zuweisung • Type Cast	• Anweisungs-folgen wiederholen • Bedingte Ausführung von Anweisungs-folgen • Mehrfach-Verzweigungen • Schleifen	• Parametrisier-barer Code • Aufrufen wieder-verwendbarer Funktionalität

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
6

6

Zum Nachlesen ...

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 4

Grundlagen der Programmierung in JAVA

Abschnitt 4.3

Einfache Datentypen

Abschnitt 4.4

Der Umgang mit einfachen Datentypen

7

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

- Ganzzahlige Datentypen
- Gleitkommatypen
- Wahrheitstyp
- Zeichen
- Zeichenketten
- Typumwandlungen
- Deklaration und Initialisierung
- Wertezuweisung an Variablen
- Auslesen von Werten aus Variablen

8

Variablen

TECHNISCHE HOCHSCHULE LÜBECK

- dienen in Programmiersprachen dazu
- Werte zu speichern
- und mittels eines Namens (symbolische Adresse) ansprechen zu können.

Arbeitsspeicher

symbolische Adresse	*Adresse im Speicher*	*Inhalt der Speicherzelle*	*Typ des Inhalts*
	⋮	⋮	
b	94	107	*ganzzahliger Wert*
	⋮	⋮	

Quelle: Grundkurs Programmieren in Java

9

Primitive Datentypen

TECHNISCHE HOCHSCHULE LÜBECK

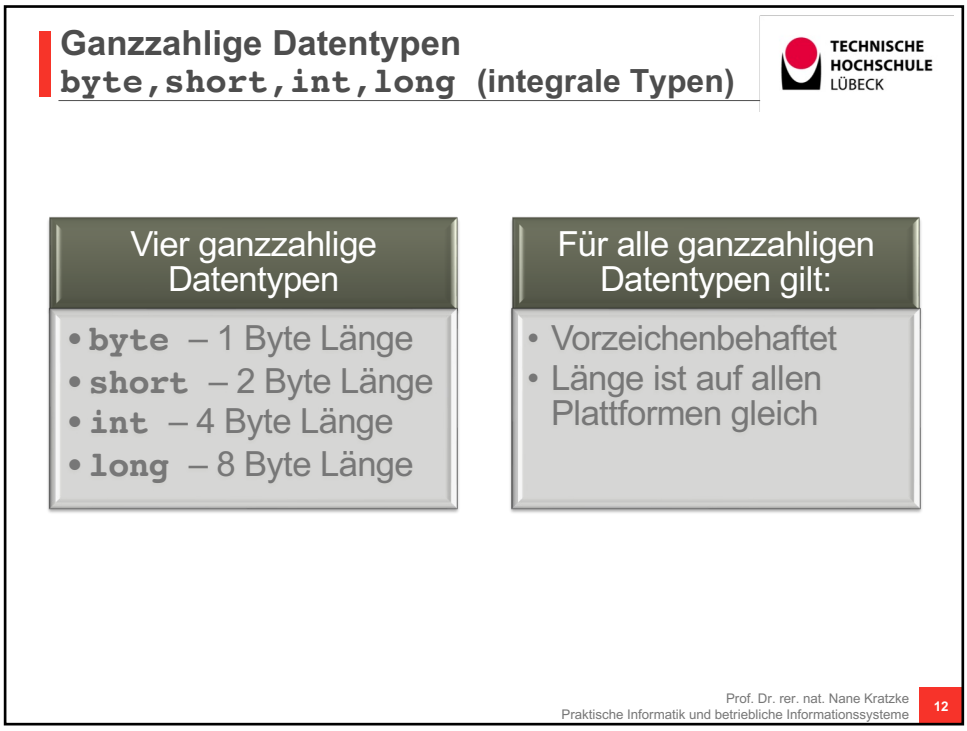
- **JAVA kennt 8 primitive Datentypen.**

Typname	Länge (in Byte)	Wertebereich	Standardwert
`boolean`	1	`true, false`	`false`
`char`	2	Alle Unicode-Zeichen	`\u0000`
`byte`	1	-2^7 ... 2^7-1	0
`short`	2	-2^{15}...$2^{15}-1$	0
`int`	4	-2^{31}...$2^{31}-1$	0
`long`	8	-2^{63}...$2^{63}-1$	0
`float`	4	$\pm 3{,}402823...*10^{38}$	0.0
`double`	8	$\pm 1{,}797693...*10^{308}$	0.0

10


Worum geht es nun?
TECHNISCHE HOCHSCHULE LÜBECK
Ganzzahlige Datentypen
Gleitkommatypen
Wahrheitstyp
Zeichen
Zeichenketten
Typumwandlungen
Deklaration und Initialisierung
Wertezuweisung an Variablen
Auslesen von Werten aus Variablen
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
11

11


Ganzzahlige Datentypen
byte,short,int,long (integrale Typen)
TECHNISCHE HOCHSCHULE LÜBECK
Vier ganzzahlige Datentypen
• byte – 1 Byte Länge
• short – 2 Byte Länge
• int – 4 Byte Länge
• long – 8 Byte Länge
Für alle ganzzahligen Datentypen gilt:
• Vorzeichenbehaftet
• Länge ist auf allen Plattformen gleich
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
12

12

Ganzzahlige Datentypen
`byte, short, int, long` (integrale Typen)

TECHNISCHE HOCHSCHULE LÜBECK

```
byte max = Byte.MAX_VALUE;
byte min = Byte.MIN_VALUE;
System.out.println(min);
System.out.println(max);
```

-128

127

```
short max = Short.MAX_VALUE;
short min = Short.MIN_VALUE;
System.out.println(min);
System.out.println(max);
```

-32768

32767

```
int max = Integer.MAX_VALUE;
int min = Integer.MIN_VALUE;
System.out.println(min);
System.out.println(max);
```

-2147483648

2147483647

```
long max = Long.MAX_VALUE;
long min = Long.MIN_VALUE;
System.out.println(min);
System.out.println(max);
```

-9223372036854775808

9223372036854775807

13

Gleitkommatypen
`float, double`

TECHNISCHE HOCHSCHULE LÜBECK

Zwei Datentypen für Nichtganzzahlen

- **`float`**
  - 4 Byte Länge
  - Einfache Genauigkeit
- **`double`**
  - 8 Byte Länge
  - Doppelte Genauigkeit

Für alle Fießkommadatentypen gilt:

- Dezimalnotation bestehend aus
  - Vorkommateil
  - Dezimalpunkt
  - einem Nachkommaanteil
  - einem Exponenten (optional)
  - einem Suffix (optional)

Vorkomma.*Nachkomma*[e*Exponent*][f|d]

Beispiele:

3.4e3d	=	3.400,0	doppelte Genauigkeit (double)
.6	=	0,6	
1.	=	1,0	
2f	=	2,0	einfache Genauigkeit (float)

14

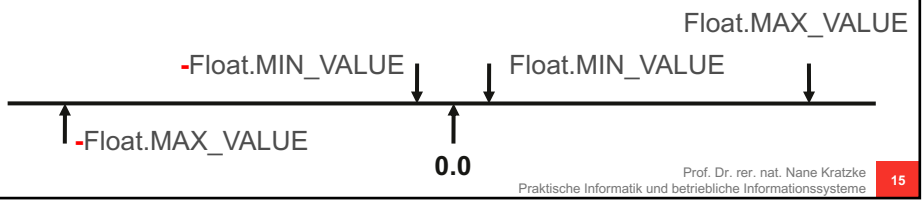
Gleitkommatypen
`float, double`

TECHNISCHE HOCHSCHULE LÜBECK

Beispiel funktioniert für `double` analog

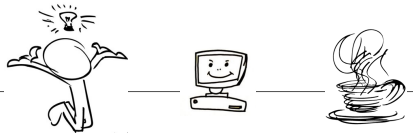
```
float max_float = Float.MAX_VALUE;
float a_float = 3.4e3f;
float min_float = Float.MIN_VALUE;
System.out.println(min_float);
System.out.println(a_float);
System.out.println(max_float);
```

1.401298464324817E-45
3400.0
3.4028234663852886E38



15

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

```
double a = 0.7;
double b = 0.9;
double x = a + 0.1;
double y = b - 0.1;
if (x == y) {
  System.out.println("Ich kann rechnen.");
} else {
  System.out.println("Ich kann nicht rechnen.");
}
```

Ergibt welche Ausgabe?

```
Ich kann nicht rechnen.
```


16

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Sie sollten daher besser so etwas schreiben:

```
double a = 0.7;
double b = 0.9;
double x = a + 0.1;
double y = b - 0.1;
double delta = 1E-25d; // definiert Gleichheit
if (Math.abs(x - y) < delta) {
  System.out.println("Ich kann rechnen.");
} else {
  System.out.println("Ich kann nicht rechnen.");
}
```


17

Wahrheitstyp (I) `boolean`

TECHNISCHE HOCHSCHULE LÜBECK

`boolean` kennt zwei verschiedene Werte

- `true`
- `false`
- Variablen dieses Typs dienen der Verarbeitung von Wahrheitsaussagen

Wahrheitswerte beruhen ausschließlich auf `boolean`

- Andere Programmiersprachen werten oft den Inhalt einer Variable ungleich null aus, was zu Unsauberheiten in der Programmierung führt
- Dies geht in JAVA nicht!

18

Wahrheitstyp (II)
boolean

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = false;

if (a) {
   System.out.println(„a war true“);
} else {
   System.out.println(„a war false“);
}
```

Korrekter Einsatz eines boolschen Variable in JAVA.

```
int a = 0;

if (a != 0) {
   System.out.println(„a war true“);
} else {
   System.out.println(„a war false“);
}
```

entspr.

Falscher Einsatz einer ganzzahligen Variable als boolean Ersatz in JAVA.

Diverse Prog-Sprachen erlauben so etwas.

19

Zeichen
char

TECHNISCHE HOCHSCHULE LÜBECK

Zur Verarbeitung von Zeichen bietet JAVA den Datentyp char an.

char-Literale werden in **einfache Hochkommata** gesetzt.

- ‘a‘, ‘b‘, ‘c‘, ...

```
char zeichen = ‘A‘;
```

Was machen Sie, wenn Sie das Zeichen ‘ ausdrücken wollen?

```
char zeichen = ‘‘‘;
```

Verwirrt den JAVA-Compiler, da er nicht mehr weiß, wo das Zeichen anfängt und aufhört.

```
char zeichen = ‘\‘‘;
```

Ausweg: Nutzung sogenannter ESCAPE-Sequenzen,

20

Zeichentyp `char` – (ESC-Sequenzen)

TECHNISCHE HOCHSCHULE LÜBECK

ESC-Sequenz	Bedeutung
`\b`	Rückschritt (Backspace, dass durch die DEL-Taste erzeugte Zeichen)
`\t`	Horizontaler Tabulator (das durch die TAB-Taste erzeugte Zeichen)
`\n`	Zeilenschaltung (Newline)
`\f`	Seitenumbruch (Formfeed)
`\r`	Wagenrücklauf (Carriage Return – das durch die ENTER Taste erzeugte Zeichen)
`\"`	Doppeltes Anführungszeichen
`\'`	Einfaches Anführungszeichen
`\\`	Backslash \

ESCAPE Zeichen werden zur Darstellung von Sonderzeichen oder nicht darstellbaren Zeichen genutzt.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
21

21

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
22

22

Strings

TECHNISCHE HOCHSCHULE LÜBECK

- In JAVA werden Zeichenketten durch die Klasse `String` repräsentiert.
- Reihung von Elementen des Typs `char`.
- Ein String ist eine indizierte Liste von Zeichen.

D	i	e	s		i	s	t		e	i	n		S	a	t	z	.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
23

23

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Jedes Zeichen eines Strings kann über einen Index angesprochen werden.

Das erste Zeichen hat den Index 0.

```
String str = "Dies ist ein Satz.";
char c = str.charAt(5);
```

```
String substr = str.substring(9, 12);
```

```
String finstr = str.substring(13);
```

0					5					10					15				
D	i	e	s		i	s	t		e	i	n		S	a	t	z	.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
24

24

Methoden der Klasse `String`

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- **Länge**
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Länge eines Strings entspricht der Anzahl an Zeichen eines Strings.

Die Länge eines leeren Strings ist 0.

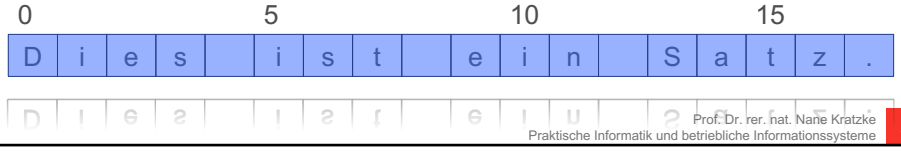
```
String str = "Dies ist ein Satz.";
int length = str.length();
```

18

```
String empty = "";
int length = empty.length();
```

0

```
boolean v = str.isEmpty();   // => false
boolean l = empty.isEmpty(); // => true
```



25

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- **Vergleichen**
- Suchen
- Ersetzen
- Zerlegen

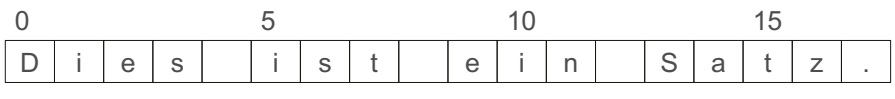
Mit den folgenden **`equals`** Methoden lassen sich Strings inhaltlich auf Gleichheit vergleichen.

```
String h1 = „hallo“;
String h2 = „HALLO“;
boolean gleich = h1.equals(h2);
```

false

```
gleich = h1.equalsIgnoreCase(h2);
```

true



26

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit den folgenden Operatoren lassen sich Strings lexikalisch vergleichen.

```
String name1 = "Müller";
String name2 = "Meier";
int res = name1.compareTo(name2);
```

> 0 – d.h. lexikalisch dahinter einzusortieren

```
res = name2.compareTo(name1);
```

< 0 – d.h. lexikalisch davor einzusortieren

```
res = name1.compareTo(name1);
```

= 0 – d.h. Zeichenketten sind gleich

0					5							10					15			
D	i	e	s		i	s	t		e	i	n		S	a	t	z	.			

27

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit der **`indexOf`** Methode lassen sich Zeichenketten in Strings finden:

```
String str = "Dies ist ein Satz";
int i = str.indexOf("Satz");
```

13

```
i = str.indexOf("existiert nicht");
```

-1 – d.h. Suchstring wurde nicht gefunden

```
i = str.indexOf("i");
```

1

```
i = str.indexOf("i", 3);
```

5

0					5							10					15			
D	i	e	s		i	s	t		e	i	n		S	a	t	z	.			

28

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit den **`startsWith/endsWith`** Methoden lässt sich prüfen, ob eine Zeichenkette Anfang/Ende einer anderen Zeichenkette ist.

```
String str = "Dies ist ein Satz";
boolean b = str.startsWith("Satz");
```

`false`

```
String str = "Dies ist ein Satz";
boolean b = str.endsWith("Satz");
```

`true`

0					5							10					15			
D	i	e	s		i	s	t		e	i	n		S	a	t	z	.			

29

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit der **`contains`** Methode lässt sich prüfen, ob eine Zeichenkette Bestandteil einer anderen Zeichenkette ist.

```
String str = "Dies ist ein Satz";
boolean b = str.contains("Satz");
```

`true`

```
String str = "Dies ist ein Satz";
boolean b = str.contains("kein");
```

`false`

0					5							10					15			
D	i	e	s		i	s	t		e	i	n		S	a	t	z	.			

30

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit den folgenden Methoden lassen sich Ersetzungen in Zeichenketten vornehmen.

```
String str = "Dies ist ein Satz.";
String newstr = str.toLowerCase();
```

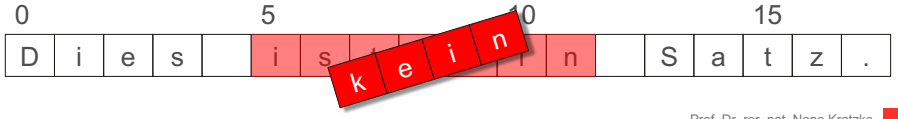
dies ist ein satz.

```
newstr = str.toUpperCase();
```

DIES IST EIN SATZ.

```
newstr = str.replaceAll("ist ein", "kein");
```

Dies kein Satz.



31

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

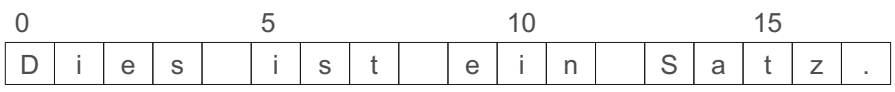
- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit den folgenden Methoden lassen sich Ersetzungen in Zeichenketten vornehmen.

```
String str = "   Dies ist ein Satz.  ";
String newstr = str.trim();
```

Dies ist ein Satz.

(führende und folgende Whitespaces gelöscht)



32

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

- Zeichenextraktion
- Länge
- Vergleichen
- Suchen
- Ersetzen
- Zerlegen

Mit der **`split`** Methode lassen sich Zeichenketten in Teilzeichenketten zerlegen.

```
String str = "Dies ist ein Satz.";
String[] subs = str.split(" ");
```

D	i	e	s

i	s	t

e	i	n

S	a	t	z	.

0					5					10					15				
D	i	e	s		i	s	t		e	i	n		S	a	t	z	.		

33

Methoden der Klasse String

TECHNISCHE HOCHSCHULE LÜBECK

Weitere Methoden der Klasse `String` finden Sie hier:

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Quelle: http://www.google.com

0					5					10					15				
D	i	e	s		i	s	t		e	i	n		T	i	p	p	.		

34

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Schreiben Sie nun ein Programm, das einen beliebigen Satz von der Konsole einliest und bestimmt wie viele Worte dieser Satz hat.

```
Bitte geben Sie einen Satz ein:
> Dies ist nur ein doofes Beispiel
Der Satz ‘Dies ist nur ein doofes Beispiel‘ hat 6 Wörter.
```


35

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

- Ganzzahlige Datentypen
- Gleitkommatypen
- Wahrheitstyp
- Zeichen
- Zeichenketten
- Typumwandlungen

Siehe Operatoren (Typecast)

Haben sie – ohne es zu merken – bereits alles in Unit I gehört.

- Deklaration und Initialisierung
- Wertezuweisung an Variablen
- Auslesen von Werten aus Variablen

36

Variablen

TECHNISCHE HOCHSCHULE LÜBECK

Eine Variable ist ein **Speicherplatz**. Über einen **Variablennamen** (Bezeichner) kann man auf den Inhalt einer Variablen zugreifen.

Einer Variablen kann ein bestimmter Inhalt (Wert) zugewiesen und dieser später wieder ausgelesen werden.

Um Variablen zu verstehen, muss man begreifen wie

(1) man Variablen deklariert,

(2) Variablen Werte zuweist

(3) und Werte aus Variablen ausliest.

37

Deklaration von Variablen

TECHNISCHE HOCHSCHULE LÜBECK

Um eine Variable nutzen zu können, muss man diese einführen. Dies erfolgt durch eine **Deklaration**.

Mittels einer Deklaration kann man eine Variable **benennen** und einer Variablen einen **Datentyp** zuweisen.

```
short ziel; // Deklarationsanweisung zur Erzeugung
            // einer Variablen vom Typ short mit
            // dem Namen ziel
```

```
int zahl = 15;  // Initialisierungsanweisung zu
                // Erzeugung einer Variablen vom Typ
                // int mit dem Namen zahl und dem
                // initialen Wert fünfzehn
```


38

Wertezuweisung an Variablen

TECHNISCHE HOCHSCHULE LÜBECK

Um einer Variable Werte zu zuweisen, benötigt man einen **Zuweisungsoperator** =. Es können mit diesem

- Werte
- Ausdrücke oder
- Routinenrückgaben

einer Variablen zugewiesen werden.

```
double var;

var = 5.0;          // Zuweisung eines Wertes
var = 5.0 + 3;      // Zuweisung des Werts eines
                    // Ausdrucks (hier 8)
var = Math.sqrt(9);// Zuweisung der Rückgabe einer
                    // Routine (hier 3, Wurzel aus 9)
```


39

Lesen aus Variablen

TECHNISCHE HOCHSCHULE LÜBECK

Es gibt üblicherweise **keinen Ausleseoperator** für Variablen in Programmiersprachen. Variablen kommen in Ausdrücken vor und werden im Rahmen der Auswertung dieser Ausdrücke ausgelesen. Einer der einfachsten Ausdrücke ist einfach das Vorkommen einer Variablen. Eine Variable wird also immer dann ausgelesen, wenn sie in einem **Ausdruck** vorkommt.

```
double var = 16.0;

// Auslesen und direktes Ausgeben einer Variablen
System.out.println(var);

// Auslesen, Ausdruck berechnen und Ausgeben einer
// Variablen
System.out.println(var + 8);

// Auslesen, Wert an andere Routine übergeben und Ausgeben
// des Routinenergebnisses
System.out.println(Math.sqrt(var));
```


40

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Sie sollen verschiedene Variablen in einem Programm deklarieren.
Finden Sie passende und möglichst platzsparende Datentypen für eine Variable, die angibt

(1) wie viele Menschen in Deutschland leben,
(2) wie viele Menschen auf der Erde leben,
(3) ob es gerade Tag ist,
(4) wie hoch die Trefferquote eines Stürmers ist,
(5) wie viele Semester sie zu studieren beabsichtigen,
(6) wie viele Studierende sich für einen Studiengang gemeldet haben,
(7) mit welchem Buchstaben ihr Name beginnt.

41

Mini-Übung:



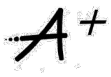
TECHNISCHE HOCHSCHULE LÜBECK

Welche der folgenden expliziten Typkonvertierungen ist unnötig, da Sie im Bedarfsfall implizit durchgeführt wird.

- `(int) 3`
- `(long) 3`
- `(long) 3.1`
- `(short) 3`
- `(short) 31`
- `(double) 31`
- `(int) 'x'`
- `(double) 'x'`

42

Zusammenfassung



TECHNISCHE HOCHSCHULE LÜBECK

- **Einfache Datentypen**
  - Ganzzahlige Datentypen
  - Gleitkommatypen
  - Wahrheitstyp
  - Zeichen
  - Zeichenketten
  - Typumwandlungen

- **Variablen**
  - Deklaration und Initialisierung
  - Wertezuweisung an Variablen
  - Auslesen von Werten aus Variablen

43

Themen dieser Unit

TECHNISCHE HOCHSCHULE LÜBECK

Datentypen	Operatoren	Kontrollstrukturen	Routinen
• Werte • Variablen • Wertetypen	• Ausdrücke • Arithmetisch • Relational • Logisch • Bedingte Auswertung • Zuweisung • Type Cast	• Anweisungs-folgen wiederholen • Bedingte Ausführung von Anweisungs-folgen • Mehrfach-Verzweigungen • Schleifen	• Parametrisier-barer Code • Aufrufen wieder-verwendbarer Funktionalität

44

Zum Nachlesen ...

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 4

Grundlagen der Programmierung in JAVA

Abschnitt 4.4.2

Operatoren und Ausdrücke

Abschnitt 4.4.3

Allgemeine Ausdrücke

45

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

Arithmetische Operatoren	Relationale Operatoren	Logische Operatoren
Bedingte Auswertung	Zuweisungsoperatoren	String-Verkettung
Type Cast Operator	new Operator	Ausdrücke

46

Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

- dienen in Programmiersprachen dazu
- Werte
  - miteinander zu **verrechnen** oder
  - miteinander zu **vergleichen** oder
  - Variablen **zuzuweisen**

Bsp.: Berechnung

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

Sys.out.println(a + b);
```

Bsp.: Vergleich

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

Sys.out.println(a < b);
```

Bsp.: Zuweisung

```
int a = 5; int b = 2;
a = b;
Sys.out.println(b);
```

Bsp.: Zuweisung und Vergleich

```
int a = 5; int b = 2;
boolean c = a < b;
Sys.out.println(c);
```


47

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

48

Arithmetische Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

Erwarten numerische Operanden

Liefern numerische Operanden

Dienen numerischen Berechnungen

49

Liste arithmetischer Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

Operator	Bezeichnung	Bedeutung
+	Positives Vorzeichen	+n = n
-	Negatives Vorzeichen	-n = -1 * n
+	Summe	a + b = Summe von a und b
-	Differenz	a – b = Differenz von a und b
*	Produkt	A * b = Produkt von a und b
/	Quotient	a / b = Quotient von a und b. Bei ganzzahligen Typen handelt es sich um die Division ohne Rest.
%	Restwert (Modulo)	a % b = Rest der ganzzahligen Division von a durch b.
++	Präinkrement	++a ergibt a + 1
++	Postinkrement	a++ ergibt a + 1
--	Prädekrement	--a ergibt a - 1
--	Postdekrement	a-- ergibt a -1

50

Miniübung zu arithmetischen Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

```
int ai = 5; int bi = 2;
float af = 5.0; float bf = 2.0;
```

```
System.out.println(ai + bi);
```
7

```
System.out.println(ai / bi);
```
2

```
System.out.println(af / bi);
```
2.5

```
System.out.println(ai % bi);
```
1

51

Miniübung zu arithmetischen Prä- und Postfix-Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

```
int ai = 5; int bi = 2;
```

```
System.out.println(ai++);
```
5

```
System.out.println(ai);
```
6

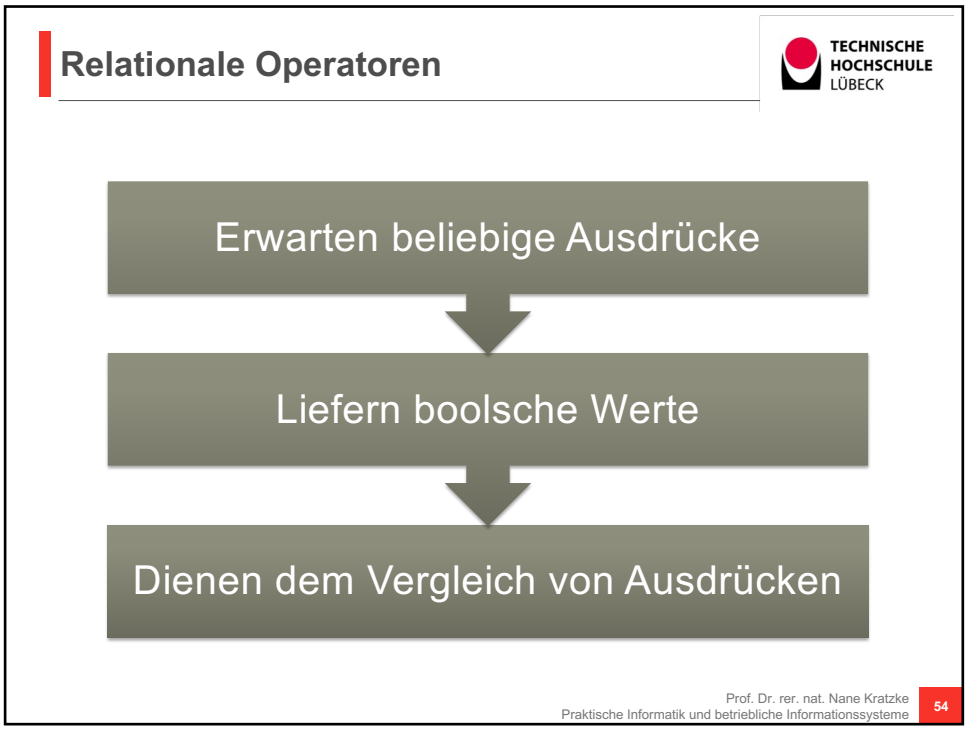
```
System.out.println(--bi);
```
1

```
System.out.println(bi);
```
1

52


Worum geht es nun?
TECHNISCHE HOCHSCHULE LÜBECK
Arithmetische Operatoren
Relationale Operatoren
Logische Operatoren
Bedingte Auswertung
Zuweisungsoperatoren
String-Verkettung
Type Cast Operator
new Operator
Ausdrücke
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
53

53


Relationale Operatoren
TECHNISCHE HOCHSCHULE LÜBECK
Erwarten beliebige Ausdrücke
Liefern boolsche Werte
Dienen dem Vergleich von Ausdrücken
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
54

54

Liste relationaler Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

Operator	Bezeichnung	Bedeutung
==	gleich	a == b ergibt true wenn a und b gleich sind. Sind a und b Referenztypen müssen sie auf dasselbe Objekt zeigen.
!=	ungleich	a != b ergibt true, wenn a und b nicht gleich sind. Sind a und b Referenztypen ergibt a !=b true wenn a und b auf verschiedene Objekte zeigen.
<	kleiner	a < b ergibt true wenn a kleiner als b ist.
>	größer	A > b ergibt true wenn a größer als b ist.
<=	Kleiner gleich	a <= b ergibt true wenn a kleiner als b oder gleich b ist.
>=	größer gleich	a >= b ergibt true wenn a größer als b oder gleich b ist.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
55

55

Miniübung zu relationalen Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

```
int ai = 5; int bi = 2;
```

```
System.out.println(ai == bi);
```
false

```
System.out.println(ai != bi);
```
true

```
System.out.println(ai < bi);
```
false

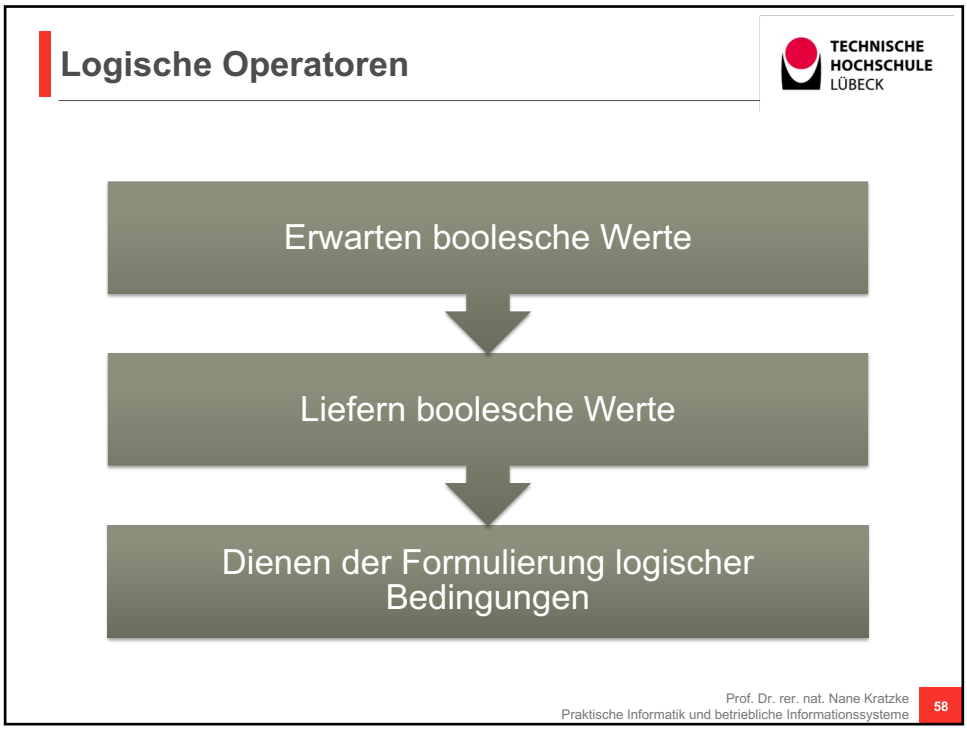
```
System.out.println(ai >= bi);
```
true

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
56

56


Worum geht es nun?
TECHNISCHE HOCHSCHULE LÜBECK
Arithmetische Operatoren
Relationale Operatoren
Logische Operatoren
Bedingte Auswertung
Zuweisungsoperatoren
String-Verkettung
Type Cast Operator
new Operator
Ausdrücke
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
57

57


Logische Operatoren
TECHNISCHE HOCHSCHULE LÜBECK
Erwarten boolesche Werte
Liefern boolesche Werte
Dienen der Formulierung logischer Bedingungen
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
58

58

Liste logischer Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

Operator	Bezeichnung	Bedeutung
!	Logisches NICHT	!a ergibt true wenn a false ist und false, wenn a true ist.
&&	UND (short-circuit)	a && b ergibt true, wenn a und b true sind. Ist a bereits false wird b nicht mehr ausgewertet.
||	ODER (short-circuit)	a || b ergibt true, wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. Ist bereits a wahr wird b nicht mehr ausgewertet. (logisches Oder)
&	UND	a & b ergibt true, wenn a und b true sind.
|	ODER	a | b ergibt true, wenn mindestens einer der beiden Ausdrücke a oder b wahr ist. (logisches Oder)
^	Exklusiv-ODER	a ^ b ergibt true wenn a und b einen unterschiedlichen Wahrheitswert haben (sprachliches Oder)

59

Miniübung zu logischen Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = true;
boolean b = false;
```

```
System.out.println(!a);
```
false

```
System.out.println(!b);
```
true

```
System.out.println(a && b);
```
false

```
System.out.println(b || a);
```
true

```
System.out.println(b ^ a);
```
true

60

Miniübung zu logischen Operatoren (Short Circuit Verhalten)

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = false;
boolean c = true;
int x = 1;
```

```
System.out.println(a && 5 / --x == 0);
```
false

```
System.out.println(a & 5 / --x == 0);
```
Division by Zero!

```
System.out.println(c || 5 / --x == 0);
```
true

```
System.out.println(c | 5 / --x == 0);
```
Division by Zero!

61

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

62

Bedingte Auswertung

TECHNISCHE HOCHSCHULE LÜBECK

- Der Fragezeichen Operator ?: ist der einzige dreiwertige Operator
- Kann häufig eingesetzt werden, um if-Abfragen zu vermeiden.
- a ? b : c
  - Ist a true wird b zurückgeliefert
  - Ist a false wird c zurückgeliefert

63

Miniübung: Bedingte Auswertung

TECHNISCHE HOCHSCHULE LÜBECK

```
int a = 6;
System.out.println(a % 2 == 0 ? “gerade“ : “ungerade“);
```

gerade

```
String satz = “Dies ist nur ein Beispiel“;
String h = “Hello“;
String w = “World“;
System.out.println(satz.contains(“kein“) ? w + h : h + w);
```

HelloWorld

64

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Schreiben Sie nun ein Programm, das einen beliebigen Satz von der Konsole einliest und bestimmt ob der Satz eine gerade Anzahl an Worten hat.

```
Bitte geben Sie einen Satz ein:
> Dies ist nur ein doofes Beispiel.
Der Satz hat eine gerade Anzahl an Worten.
```

```
Bitte geben Sie einen Satz ein:
> Dies ist nur ein Beispiel.
Der Satz hat eine ungerade Anzahl an Worten.
```


65

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

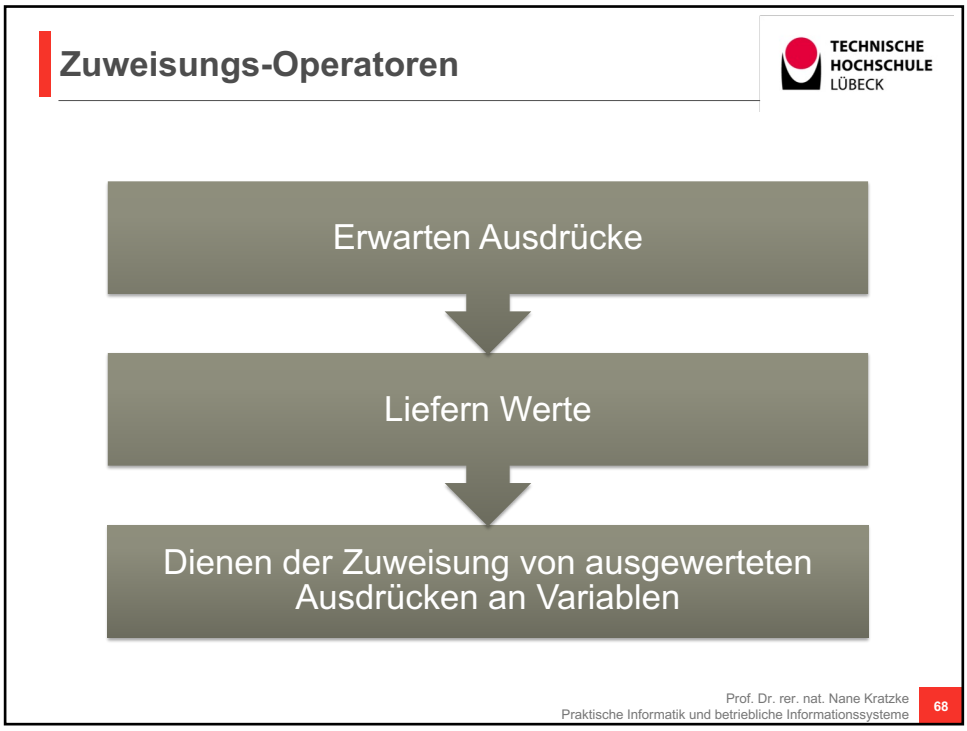
Lösung:

Ganz einfach mit der bedingten Auswertung ...

66


Worum geht es nun?
TECHNISCHE HOCHSCHULE LÜBECK
Arithmetische Operatoren
Relationale Operatoren
Logische Operatoren
Bedingte Auswertung
Zuweisungsoperatoren
String-Verkettung
Type Cast Operator
new Operator
Ausdrücke
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
67

67


Zuweisungs-Operatoren
TECHNISCHE HOCHSCHULE LÜBECK
Erwarten Ausdrücke
Liefern Werte
Dienen der Zuweisung von ausgewerteten Ausdrücken an Variablen
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
68

68

Liste der Zuweisungsoperatoren

TECHNISCHE HOCHSCHULE LÜBECK

Operator	Bezeichnung	Bedeutung
=	Einfache Zuweisung	a = b weist a den Wert von b zu und liefert b als Rückgabe.
+=	Additionszuweisung	a += b weist a den Wert von a + b zu und liefert a + b als Rückgabe.
-=	Subtraktionszuweisung	Analog += Operator mit -
*=	Multiplikationszuweisung	Analog += Operator mit *
/=	Divisionszuweisung	Analog += Operator mit /
%=	Modulozuweisung	Analog += Operator mit %
&=	UND-Zuweisung	Analog += Operator mit logischem &
|=	ODER-Zuweisung	Analog += Operator mit logischem |
^=	XOR-Zuweisung	Analog += Operator mit ^ (XOR)

Wichtig: Eine Zuweisung ist immer auch ein Ausdruck, d.h. sie kann in anderen Ausdrücken auftauchen.

69

Zuweisungen sind auch Ausdrücke

TECHNISCHE HOCHSCHULE LÜBECK

Da Zuweisungen in Java auch immer Ausdrücke sind, kann man dies nutzen, um derartige Zuweisungsfolgen zu formulieren.

```
int a, b, c;
a = b = c = 5;
```

Ein Wert wird dabei mehreren Variablen zugewiesen. Die Semantik der Zuweisung wird deutlicher, wenn man sie klammert.

```
a = (b = (c = 5));
```

Eigentlich erfolgt hier nicht eine Zuweisung, sondern drei Einzelzuweisungen. Dabei wird der Wert eines Zuweisungsausdrucks immer für eine weiter links stehende Zuweisung genutzt.

```
int hc = (c = 5);
int hb = (b = hc);
         a = hb;
```


70

Miniübung zu Zuweisungs-Operatoren

TECHNISCHE HOCHSCHULE LÜBECK

```
int a = 5;
int b, c;
```

```
System.out.println(c = b = a);
```
5 (a, b, c)

```
System.out.println(b /= 1);
```
5

```
System.out.println(c += 3);
```
8

```
System.out.println(a %= 2);
```
1

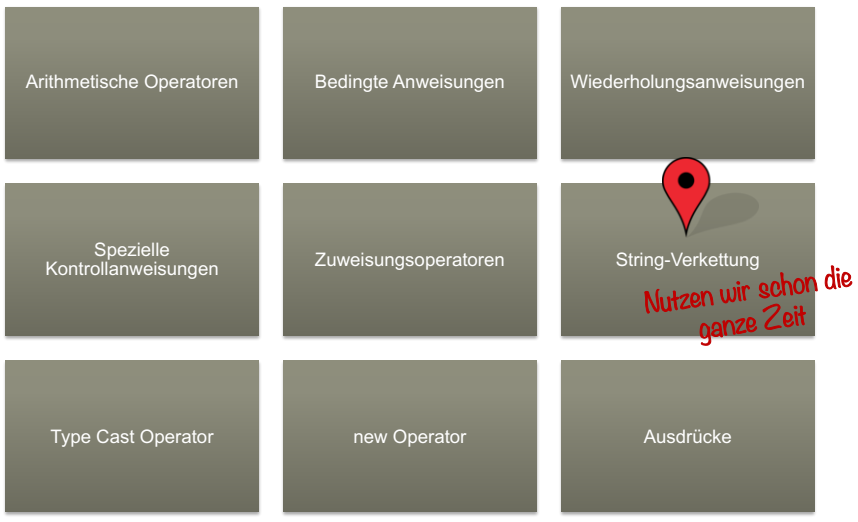
```
c -= b *= a++;
System.out.println(a);
System.out.println(b);
System.out.println(c);
```
2 (a)
5 (b, Postinkrement!)
3 (c)

Wichtig: *Angaben in Klammern werden nicht mit auf der Konsole ausgegeben (diese dienen nur der besseren Zuordnung zu den Variablen).*

71

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK



72

String Verkettung (Konkatenation)

TECHNISCHE HOCHSCHULE LÜBECK

- Der **+ Operator** kann auch auf Strings angewendet werden.
- Er hat dann die Semantik einer Aneinanderreihung der Strings.

```
String a = „Nice “;
int i = 2;
String b = „ meet you“;
System.out.println(a + i + b);
```

Nice 2 meet you

- Bei der Operation wird ggf. ein Nichtstring Operand in einen String gewandelt.
  - Die Stringwandlung wird bei primitiven Typen durch den Compiler vorgenommen.
  - Bei Referenztypen wird hierzu die Methode `toString()` vorher aufgerufen.

73

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Was erzeugt dieser Quelltext für eine Ausgabe?

```
int a = 5;
int b = 2;
System.out.println(a + b + “=“ + a + “+“ + b);
```

```
7=5+2
```

```
int a = 5;
int b = 2;
System.out.println(a + “+“ + b + “=“ + a + b);
```

```
5+2=52
```

```
int a = 5;
int b = 2;
System.out.println(a + “+“ + b + “=“ + (a + b));
```

```
5+2=7
```


74

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

75

Typumwandlung
Type Casting

TECHNISCHE HOCHSCHULE LÜBECK

Typumwandlungen sind immer dann erforderlich, wenn ein Wert einen Datentyp hat, der nicht einem Zieltyp entspricht. Hierbei wird in statisch typisierten Programmiersprachen eine implizite von einer expliziten Typumwandlung unterschieden.

Implizite Typumwandlung:

Ein kleinerer Zahlenbereich (z.B. `byte`) wird in einen größeren Zahlenbereich (z.B. `short`) abgebildet. Diese Fälle kann der Compiler automatisch behandeln, es können keine Datenverluste auftreten.

```
short ziel = 23; // 23 ist vom Typ byte, da < 128
```

Explizite Typumwandlung:

Ein größerer Zahlenbereich (z.B. `short`) wird in einen kleineren Zahlenbereich (z.B. `byte`) abgebildet. Diese Fälle kann der Compiler nicht automatisch behandeln, da Datenverluste auftreten könnten. Der Programmierer muss daher diese Fälle mit einem expliziten Cast „bestätigen“.

```
byte ziel = (byte)512; // 512 ist vom Typ short, da >= 128
```


76

Implizite und explizite Typumwandlung
Implicite and explicte type casting

TECHNISCHE HOCHSCHULE LÜBECK

implizit ↓ / explizit ↑

Typname	größter Wert	kleinster Wert	Länge
byte	127	-128	8 Bits
short	32767	-32768	16 Bits
int	2147483647	-2147483648	32 Bits
long	9223372036854775807	-9223372036854775808	64 Bits

Quelle: Programmieren in Java

implizit ↓ / explizit ↑

Typname	größter positiver Wert	kleinster positiver Wert	Länge
float	≈3.4028234663852886E+038	≈1.401298464324817lE-045	32 Bits
double	≈1.7976931348623157E+308	≈4.9406564584124654E-324	64 Bits

Quelle: Programmieren in Java

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
77

77

Miniübung:



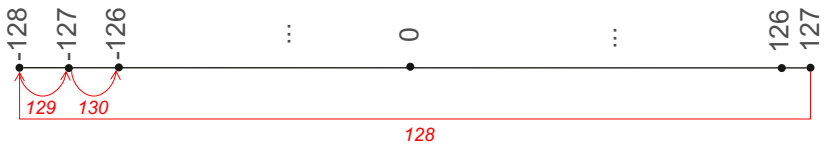
TECHNISCHE HOCHSCHULE LÜBECK

Typname	größter Wert	kleinster Wert	Länge
byte	127	-128	8 Bits
short	32767	-32768	16 Bits
int	2147483647	-2147483648	32 Bits
long	9223372036854775807	-9223372036854775808	64 Bits

```
short a = 130;
byte b = (byte)a;
System.out.println(a + " = " + b);
```

Ergibt welche Ausgabe?

```
130 = -126
```



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
78

78

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

79

Ausdrücke

TECHNISCHE HOCHSCHULE LÜBECK

Ausdrücke setzen sich in Programmiersprachen üblicherweise aus Operatoren und Operanden zusammen und werden für **numerische Berechnungen, Vergleiche** oder **logische Bedingungen** benötigt. Die Operanden selbst können dabei wieder

- **Variablen**
- **Geklammerte Ausdrücke** oder
- **Methodenaufrufe** sein.

Der einfachste Ausdruck ist einfach die Angabe einer Variablen oder eines Wertliterals.

Sind beide Operanden einer Operation wieder Ausdrücke, wird immer erst der linke und dann der rechte Operand berechnet, d.h. die **Auswertung** eines Ausdrucks erfolgt (*üblicherweise*) **von links nach rechts** gem. der Operatorbindungsstärke.

80

Vorrangregeln

TECHNISCHE HOCHSCHULE LÜBECK

Kennen Sie noch die Regel: Punktrechnung vor Strichrechnung?

Java kennt dasselbe mit ein paar mehr zu berücksichtigenden Gruppen.

81

Vorrangregeln Operatorgruppen

TECHNISCHE HOCHSCHULE LÜBECK

Steigende Bindung der Operatoren

Gruppe	Operatoren	Bezeichnung
1	++, --, !, (type)	Inkrement, Dekrement, Nicht, Type Cast
2	*, /, %	Multiplikation, Division, Modulo
3	+, -	Addition, Subtraktion
5	<, <=, >, >=,	Kleiner, Kleiner-Gleich, Größer, Größer-Gleich
6	==, !=	Gleich, Ungleich
7	&	Logisches Und
8	^	XOR
9	|	Logisches Oder
10	&&	Logisches Und (short circuit)
11	||	Logisches Oder (short circuit)
12	?:	Bedingte Auswertung
13	=, +=, -=, *=, ...	Zuweisungen

82

Veranschaulichung

TECHNISCHE HOCHSCHULE LÜBECK

```
int a = 5; int b = 3;
int e = a – b * a - b;
```

```
int e = a – 3 * 5 - b;
```

```
int e = a – 15 - b;
```

```
int e = 5 - 15 - b;
```

```
int e = -10 - b;
```

```
int e = -10 - 3;
```

```
int e = -13;
```


83

Veranschaulichung

TECHNISCHE HOCHSCHULE LÜBECK

```
int a = 5; int b = 3;
int e = (a – b) * (a - b);
```

```
int e = (5 – 3) * (a - b);
```

```
int e = 2 * (a - b);
```

```
int e = 2 * (5 - 3);
```

```
int e = 2 * 2;
```

```
int e = 4;
```


84

Veranschaulichung

TECHNISCHE HOCHSCHULE LÜBECK

```
int a = 5; int b = 3;
int e = Math.pow(a,2) – 2*a*b + Math.pow(b,2);
```

```
int e = Math.pow(a,2) – 2*5*b + Math.pow(b,2);
int e = Math.pow(a,2) – 10*b + Math.pow(b,2);
int e = Math.pow(a,2) – 10*3 + Math.pow(b,2);
int e = Math.pow(a,2) – 30 + Math.pow(b,2);
int e = Math.pow(5,2) – 30 + Math.pow(b,2);
int e = 25 – 30 + Math.pow(b,2);
int e = -5 + Math.pow(b,2);
int e = -5 + Math.pow(3,2);
int e = -5 + 9;
int e = 4;
```


85

Miniübung



TECHNISCHE HOCHSCHULE LÜBECK

Welche Ausgabe erzeugt?

```
int a = 5; int b = 3;
System.out.println(
      (a – b) * (a – b) == a*a – 2*a*b + b*b
);
```

```
true
```

Binomische Formel!

$$(a-b)(a-b) = a^2 - 2ab + b^2$$

86

Miniübung



TECHNISCHE HOCHSCHULE LÜBECK

Wird dieser Ausdruck zu `true` oder `false` ausgewertet?

```
5 * 3 + 4 < 20 && true & false | 1 + 2 * 3 > 2
```

Gruppe 2:
```
15 + 4 < 20 && true & false | 1 + 6 > 2
```

Gruppe 3:
```
19 < 20 && true & false | 7 > 2
```

Gruppe 5:
```
true && true & false | true
```

Gruppe 7:
```
true && false | true
```

Gruppe 9:
```
true && true
```

Gruppe 10:
```
true
```


87

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Gegeben ist folgender Ausdruck:

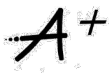
```
b = Math.sqrt(3.5 + x) * 5 / 3 - (x + 10) * (x -4.1) < 0;
```

Zerlegen sie diesen gem. der Operatorprioritäten so in Zwischenergebnisse wie nachfolgend begonnen. Beginnen Sie dabei von links nach rechts:

```
z1 = 3.5 + x;
z2 = Math.sqrt(z1);
```


88

Zusammenfassung



TECHNISCHE HOCHSCHULE LÜBECK

- **Klassische Operatoren**
  - Arithmetisch
  - Relational
  - Logisch
  - Zuweisung
  - bedingte Auswertung (`x ? A : B`)

- **Weitere Operatoren**
  - Stringkonkatenation
  - Type Cast `(type)`
  - `new` Operator

- **Ausdrücke**
  - Vorrang- und Auswerteregeln für Operatoren

89

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 3

(Variablen, Datentypen, Operatoren, Ausdrücke)

- W3.1: Einfache Rechenaufgaben
- W3.3: Blutalkoholkonzentration
- W3.4: Stoffwechselrate
- W3.5: Baumstammvolumen
- W3.6: Körperoberfläche
- W3.7: RGB nach CYMK

Nur eingeschränkt zu empfehlen

- *W3.2 (Processing Framework erforderlich)*

90

Themen dieser Unit

TECHNISCHE HOCHSCHULE LÜBECK

Datentypen	Operatoren	Kontrollstrukturen	Routinen
• Werte • Variablen • Wertetypen	• Ausdrücke • Arithmetisch • Relational • Logisch • Bedingte Auswertung • Zuweisung • Type Cast	• Anweisungs-folgen wiederholen • Bedingte Ausführung von Anweisungs-folgen • Mehrfach-Verzweigungen • Schleifen	• Parametrisier-barer Code • Aufrufen wieder-verwendbarer Funktionalität

91

Zum Nachlesen ...

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 4

Grundlagen der Programmierung in JAVA

Abschnitt 4.5

Anweisungen und Ablaufsteuerung

92

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

- Anweisungen und Blöcke
- Bedingte Anweisungen
- Wiederholungs-anweisungen
- Spezielle Kontrollanweisungen

93

Anweisungen

TECHNISCHE HOCHSCHULE LÜBECK

- Imperative Programme setzen sich primär aus einer oder mehreren Anweisungen zusammen.
- Eine **Anweisung** stellt eine in der Syntax einer Programmiersprache formulierte einzelne Vorschrift dar,
- die **im Rahmen der Abarbeitung** des Programms **auszuführen** ist und für die Ausführung des Programms eine spezifische Bedeutung hat.

- Beispiele für solche Anweisungen können sein:
  - **Deklaration von Variablen**
  - **Zuweisungen**
  - Aber auch **Entscheidungsanweisungen** (if, switch)
  - oder **Wiederholungsanweisungen**
  - Aufruf von **Unterprogrammen** (Methoden)

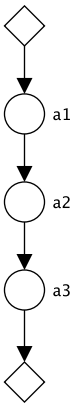
94

Anweisungsfolgen

TECHNISCHE HOCHSCHULE LÜBECK

- Ein Semikolon schließt jeweils eine Anweisung ab.
- Aufeinander folgende Anweisungen werden nacheinander abgearbeitet.

```
a1;
a2;
a3;
```



Beispiel:

```
int x = 6;
int y = 7;
System.out.println(x * y);
```


95

Blöcke { }

TECHNISCHE HOCHSCHULE LÜBECK

- In der Programmiersprache Java bezeichnet ein Block eine **Folge von Anweisungen**, die durch **{** und **}** geklammert sind.
- Solch ein Block kann immer dort, wo eine einzelne Anweisung erlaubt ist, verwendet werden, da ein Block im Prinzip eine zusammengesetzte Anweisung ist.
- Blöcke können geschachtelt werden.
- Blöcke dienen auch der logischen Gliederung von Quelltexten.

96

Blöcke (Beispiel)

TECHNISCHE HOCHSCHULE LÜBECK

```
{                  // Anfang des aeusseren Blocks
  int x = 5;               // Deklarationsanweisung und Zuweisung
  x++;                     // Postfix-Inkrement-Anweisung
  {                        // Anfang des ersten inneren Blocks
    long y;                      // Deklarationsanweisung
    y = x + 123456789;           // Zuweisung
    System.out.println(y);       // Ausgabeanweisung/Methodenaufruf
    ;                            // Leere Anweisung
  }                        // Ende des ersten inneren Blocks
  System.out.println(x);   // Ausgabeanweisung/Methodenaufruf
  {                        // Anfang des zweiten inneren Blocks
    double d;                    // Deklarationsanweisung
    d = x + 1.5;                 // Zuweisung
    System.out.println(d);       // Ausgabeanweisung/Methodenaufruf
  }                        // Ende des zweiten inneren Blocks
}                  // Ende des aeusseren Blocks
```

Quelle: Programmieren in Java

Achtung: Variablen, die in einem Block definiert sind, sind immer nur bis zum Ende des Blocks gültig sind. Man spricht in diesem Zusammenhang auch vom Gültigkeitsbereich der Variablen.

97

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

98

Verzweigungen if-Abfrage

TECHNISCHE HOCHSCHULE LÜBECK

- Verzweigungen dienen dazu bestimmte Programmteile nur beim Eintreten vorgegebener Bedingungen auszuführen.

If Variante

```
if (ausdruck)
  anweisung;
```

If Else Variante

```
if (ausdruck)
  anweisung;
else
  anweisung;
```

If Block Variante

```
if (ausdruck) {
  Block von anweisungen;
}
```

If Else Block Variante

```
if (ausdruck) {
  Block von Anweisungen;
} else {
  Block von Anweisungen;
}
```

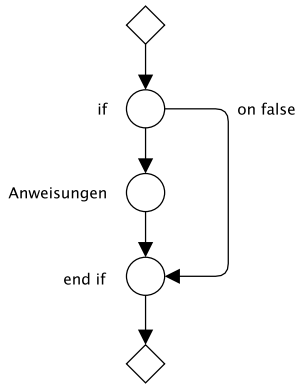

99

Syntax und Kontrollflussgraph der `if` Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

Syntaxregel:

```
if (test) {
  anweisung;
}
```



Beispiel:

```
int i = 5;
if (i < 2) {
  System.out.println("Kleiner 2");
}
```

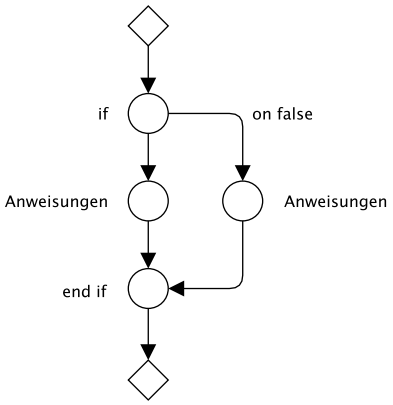

100

Syntax und Kontrollflussgraph der `if else` Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

Syntaxregel:

```
if (test) {
  anweisung;
} else {
  anweisung;
}
```



Beispiel:

```
int i = 5;
if (i < 2) {
  System.out.println("Kleiner 2");
} else {
  System.out.println("Größer oder
    gleich 2");
}
```


101

Geschachtelte ifs und „dangling else“

TECHNISCHE HOCHSCHULE LÜBECK

- Es können mehrere if und if-else Anweisungen geschachtelt werden.

Beispiel:

```
if (bed1)
  if (bed2)
    if (bed3)
      anweisung;
```

Dangling Else:

```
if (bed1)
  if (bed2)
    if (bed3)
      anweisung;
  else
    anweisung;
```

Zu welchem if gehört das else?

```
if (bed1)
  if (bed2)
    if (bed3)
      anweisung;
  else
    anweisung;
```


102

Mini-Übung
if Strukturen Raten (I)

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = true; boolean b = false; boolean c = true;
```

```
if (a)
  System.out.println(„A“);
else
  System.out.println(„B“);
```

A

```
if (b)
  System.out.println(„A“);
else
  System.out.println(„B“);
```

B

103

Mini-Übung
if Strukturen Raten (II)

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = true; boolean b = false; boolean c = true;
```

```
if (a)
  if (!b)
    if (c)
      System.out.println(„A“);
    else
      System.out.println(„B“);
```

A

```
if (a)
  if (c)
    if (b)
      System.out.println(„A“);
    else
      System.out.println(„B“);
```

B

104

Mini-Übung if Strukturen Raten (III)

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = true; boolean b = false; boolean c = true;
```

```
if (b)
  System.out.println(„A“);
else {
  if (!a) System.out.println(„B“);
  if (!c)
    if (b) System.out.println(„C“);
  else
    System.out.println(„D“);
}
```

Keine Ausgabe

Be aware of dangling else

105

Mini-Übung if Strukturen Raten (IV)

TECHNISCHE HOCHSCHULE LÜBECK

```
boolean a = true; boolean b = false; boolean c = true;
```

```
if (b) {
  System.out.println(„A“);
} else {
  if (!a) {
    System.out.println(„B“);
  }
  if (!c) {
    if (b) {
      System.out.println(„C“);
    }
  }
  else {
    System.out.println(„D“);
  }
}
```

D

Sie sollten der Konvention folgen ifs und elses grundsätzlich einen geklammerten Block folgen zu lassen.

106

Verzweigungen
Switch-Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

- Switch Anweisung ist eine Mehrfachverzweigung.
- sie wertet einen im Ergebnis ganzzahligen Ausdruck aus
- und springt einen case Zweig oder den default Zweig an.

Syntax:

```
switch (ausdruck) {
  case Konstante: anweisung;
  ...
  default: anweisung;
}
```


107

Verzweigungen
Switch-Anweisung (Beispiel)

TECHNISCHE HOCHSCHULE LÜBECK

Welche Ausgabe erzeugt dieser Code?

```
int i = 4;

switch (i % 3) {
  case 1: System.out.println(„Rest 1“);
  case 2: System.out.println(„Rest 2“);
  case 3: System.out.println(„Rest 3“);
  default: System.out.println(„Rest 0“);
}
```

Rest 1
Rest 2
Rest 3
Rest 0

Achtung: Nachdem ein case- oder default-Label angesprungen wurde, werden alle dahinter stehenden Anweisungen ausgeführt.

Will man das nicht, muss man das Label mit einer break Anweisung dazu zwingen, am Ende der switch-Anweisung fortzusetzen.

108

Verzweigungen
Switch-Anweisung (Beispiel)

TECHNISCHE HOCHSCHULE LÜBECK

Welche Ausgabe erzeugt dieser Code?

```
int i = 4;

switch (i % 3) {
  case 1: System.out.println(„Rest 1“); break;
  case 2: System.out.println(„Rest 2“); break;
  case 3: System.out.println(„Rest 3“); break;
  default: System.out.println(„Rest 0“);
}
```

Rest 1

Die ergänzende break Anweisung realisiert die Semantik der switch Anweisung, wie man sie intuitiv erwarten würde.

109

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

110

Abweisende Schleife

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
while (ausdruck) {
  anweisung;
}
```

- Prüfen des Ausdrucks.
- Solange dieser True ist, wird der Anweisungsblock oder eine Einzelanweisung ausgeführt.
- Ist der Ausdruck bereits zu Beginn false, wird der Anweisungsblock nicht ausgeführt. Daher **abweisende Schleife**.

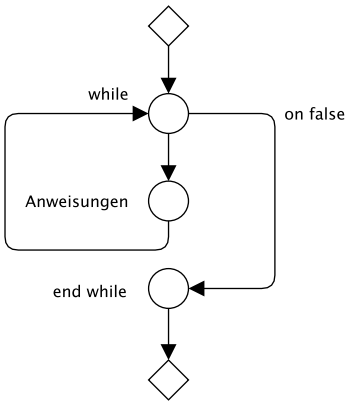
111

Syntax und Kontrollflussgraph der `while` Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

Syntaxregel:

```
while (test) {
  anweisung;
  ...
}
```



Beispiel:

```
int i = 1;
while (i < 5) {
  System.out.println(i);
  i++;
}
```


112

Abweisende Schleife
Mini-Übung: while Schleifen

TECHNISCHE HOCHSCHULE LÜBECK

```
int i = 0;
while (i < 4) System.out.print(i++ + " ");
```

0 1 2 3

```
int i = 0;
while (i < 0) System.out.println(i++);
```

Keine Ausgabe

```
int i = 0;
while (i <= 0)
  System.out.println(--i);
```

-1
-2
-3
... endet nie!

Nicht abweisende Schleife

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
do {
  anweisung;
} while (ausdruck);
```

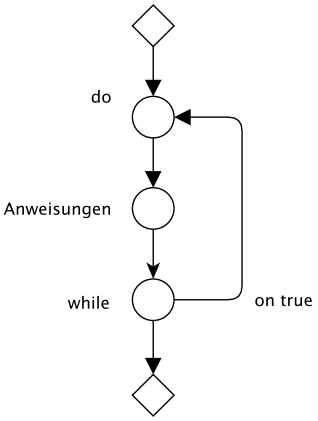
- Der Anweisungsblock wird mindestens einmal ausgeführt, daher **nicht abweisende Schleife**.
- Prüfen des Ausdrucks
  - Ist dieser True, wird der Anweisungsblock oder Einzelanweisung wieder ausgeführt
  - Ist dieser false wird die Programmausführung hinter dem while Ausdruck fortgesetzt.

Syntax und Kontrollflussgraph der do while Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

Syntaxregel:

```
do {
  anweisung;
  ...
} while (test);
```



Beispiel:

```
int i = 1;
do {
  System.out.println(i);
  i++;
} while (i < 5);
```


115

Nicht abweisende Schleife Mini-Übung: do while Schleifen

TECHNISCHE HOCHSCHULE LÜBECK

```
int i = 0;
do
  System.out.println(i++);
while (i < 4);
```

0
1
2
3

```
int i = 0;
do {
  System.out.println(++i);
  System.out.println(--i);
} while (i < 4);
```

1
0
1
0
... endet nie!

116

Zählschleife for – klassische Variante

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
for (init; test; update) {
  anweisung;
}
```

Init Ausdruck	Test Ausdruck	Update Ausdruck
• Aufruf einmalig **vor Start** der Schleife • Optional • mehrere kommaseparierte Ausdrücke mögl. • Variablendeklaration möglich	• Aufruf **jew. am Anfang** einer Schleife • Optional, wenn nicht angegeben wird true gesetzt • Schleife wird nur ausgeführt, wenn Ausdruck true	• Aufruf **jew. am Ende** der Schleife • Optional • Mehrere kommaseparierte Ausdrücke möglich • Dient dazu den Schleifenzähler zu ändern

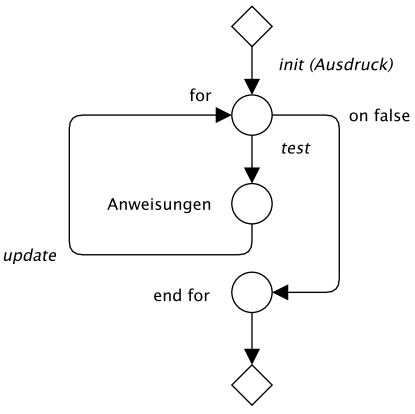
117

Syntax und Kontrollflussgraph der for Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

Syntaxregel:

```
for (init; test; update) {
  anweisung;
}
```



Beispiel:

```
for (int i = 1; i < 5; i++) {
  System.out.println(i);
}
```


118

Zählschleife
for – klassische Variante (Beispiele)

TECHNISCHE HOCHSCHULE LÜBECK

```
for (int i = 1; i <= 3; i++) System.out.println(i);
```

1
2
3

```
int i = 1;
while (true) {
  if (!(i <= 3)) break;
  System.out.println(i);
  i++;
}
```

1
2
3

```
//init-Ausdruck – einmalig

//test-Ausdruck – vor Schleifenlauf

//Update-Ausdruck – am Ende
```

Jede for-Schleife kann nach diesem Muster umformuliert werden.

119

Zählschleife
for – klassische Variante (Beispiele)

TECHNISCHE HOCHSCHULE LÜBECK

```
for (int i = 1; i <= 3; i++) System.out.print(i);
```

123

```
for (int i = 7; i > 0; i -= 2) {
  System.out.println(i);
}
```

7
5
3
1

```
int[] xs = {7, 5, 3, 1};
for (int i = 0; i < xs.length; i++) {
  System.out.println(xs[i]);
}
```

7
5
3
1

120

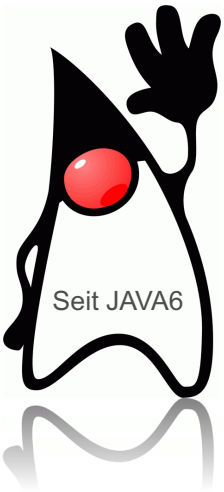
Zählschleife
foreach Variante

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
for (object : collection) {
  anweisung;
}
```

- Object ist ein „Laufobjekt“ oder eine „Zählervariable“
- Collection ist eine Instanz des Typs Iterable oder ein Array
- Zu lesen ist dieser Ausdruck als **for object in collection**, d.h.
  - es werden aus einer Collection alle Elemente elementweise durchlaufen,
  - daher auch **foreach** Variante



121

Foreach Schleife
Mini-Übung: foreach Schleifen

TECHNISCHE HOCHSCHULE LÜBECK

```
int[] is = {7, 5, 3, 1};
for (int i : is)
  System.out.println(i);
```

7
5
3
1

122

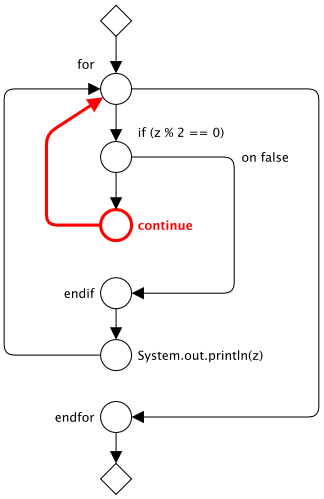
Rest einer Schleife überspringen
`continue`

TECHNISCHE HOCHSCHULE LÜBECK

- Die `continue` Anweisung springt an den Kopf der Schleife in der die `continue` Anweisung steht.
- Alle auf `continue` folgenden Anweisungen werden übersprungen.

```
// Nur ungerade Zahlen
for (int z = 0; z <= 10; z++) {
  if (z % 2 == 0) continue;
  System.out.print(z);
}
```

```
1 3 5 7 9
```



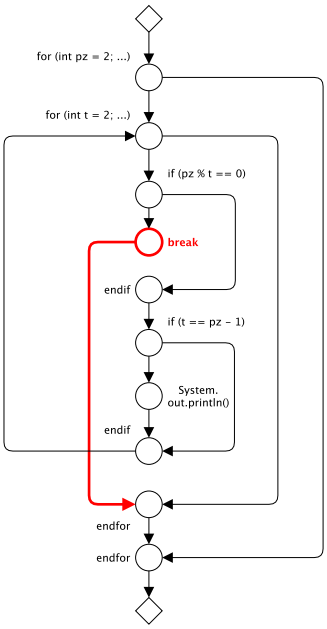
123

Schleife abbrechen
`break`

- Die `break` Anweisung springt an das Ende der Schleife in der die `break` Anweisung steht.
- Durch `break` wird die komplette Schleife abgebrochen.

```
// Primzahlen
for (int pz = 2; pz <= 20; pz++) {
  for (int t = 2; t < pz; t++) {
    if (pz % t == 0) break;
    if (t == pz - 1) {
      System.out.print(pz + " ");
    }
  }
}
```

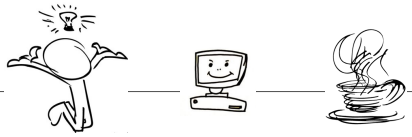
```
3 5 7 11 13 17 19
```



124

125

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Bestimmen Sie jetzt pro Wort eines eingegebenen Satzes, ob dieses eine gerade oder ungerade Länge hat.

Bestimmen sie ergänzend die durchschnittliche Wortlänge.

```
Bitte geben Sie einen Satz ein:
> Dies ist nur ein doofes Beispiel.
Dies: gerade
ist: ungerade
nur: ungerade
ein: ungerade
doofes: gerade
Beispiel.: ungerade
Die durchschnittliche Wortlänge beträgt: 4.666666666666667
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
126

126

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Lösung:

127

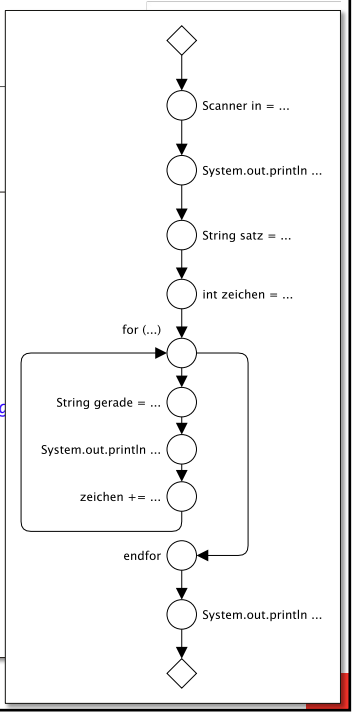
Mini-Übung:


Geben Sie nun für diesen Quelltext den Kontrollflussgraphen an:

```
Scanner in = new Scanner(System.in);
System.out.println("Bitte einen Satz eingeben: ");
String satz = in.nextLine();
String[] worte = satz.split(" +");

int zeichen = 0;
for (String wort : worte) {
  String gerade = wort.length() % 2 == 0 ? "gerade" : "ung
  System.out.println(wort + ": " + gerade);
  zeichen += wort.length();
}

System.out.println(
  "Die durchschnittliche Wortlänge beträgt: " +
  ((double)zeichen / worte.length)
);
```



128

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

- Anweisungen und Blöcke
- Bedingte Anweisungen
- Wiederholungs-anweisungen
- Spezielle Kontrollanweisungen

129

Exceptions

TECHNISCHE HOCHSCHULE LÜBECK

Exceptions sind ein Mechanismus zur strukturierten Behandlung von Fehlern die zur Laufzeit eines Programms auftreten

- Das Auslösen einer Ausnahme wird „throwing“ genannt
- Das Behandeln einer Ausnahme wird „catching“ genannt

130

Grundprinzip des Exception Mechanismus

TECHNISCHE HOCHSCHULE LÜBECK

- Laufzeitfehler oder vom Entwickler gewollte Bedingung löst Exception aus
- Behandlung entweder im Programmteil oder Weitergabe
- Bei Weitergabe hat der Empfänger erneut die Möglichkeit die Exception zu behandeln oder weiterzugeben
- Wird die Ausnahme von keinem Programmteil behandelt, führt sie zum Abbruch der Applikation

131

Behandlung von Exceptions `try-catch` Anweisung

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
try {
  anweisung;
} catch (Ausnahmetyp ex) {
  anweisung;
}
```

- Der **`try`-Block** enthält eine oder mehrere Anweisungen, bei deren Ausführung Exceptions entstehen können.
- In diesem Fall wird die normale Programmausführung unterbrochen und die Anweisungen im **`catch`-Block** zur Behandlung der Exception ausgeführt.

132

Behandlung von Exceptions Beispiel (OutOfBounds)

TECHNISCHE HOCHSCHULE LÜBECK

```
try {
  int[] list = { 42 };
  int i = list[1];
  System.out.println(„Dies wird nicht mehr ausgegeben.“);
} catch (Exception e) {
  System.out.println(„Exception: " + e.getClass());
  System.out.println(“Message: " + e.getMessage());
}
System.out.println(“Weiter - als wäre nichts gewesen.");
```

In diesem Beispiel löst der Zugriff auf das erste Elemente von `list` eine `IndexOutOfBoundException` aus, die in der `catch` Klausel gefangen und behandelt wird.

133

Exception-spezifische `catch` Klauseln

TECHNISCHE HOCHSCHULE LÜBECK

- Ausnahmen werden durch die `Exception` Klasse oder davon abgeleitete Unterklassen repräsentiert.
- Hierdurch ist es möglich mehrere Exceptionarten durch mehrere `catch` Blöcke abzufangen und spezifisch zu behandeln.
- Z.B. Division by Zero anders als IO Exceptions bei Dateioperationen

```
try {
  ...
}
catch (ArrayOutOfBoundException e) { ... }
catch (NumberFormatException e) { ... }
catch (IndexOutOfBoundException e) { ... }
```


134

Die `finally` Klausel

TECHNISCHE HOCHSCHULE LÜBECK

- Mit der optionalen `finally` Klausel kann ein Block definiert werden, der immer aufgerufen wird, wenn der zugehörige `try` Block betreten wurde.
- Der `finally` Block wird aufgerufen, wenn
  - Das normale Ende des `try`-Blocks erreicht wurde
  - Eine Ausnahme aufgetreten ist, die durch eine catch Klausel gefangen wurde
  - Wenn eine Ausnahme aufgetreten ist, die nicht durch eine `catch` Klausel gefangen wurde
  - Der `try`-Block durch ein `break` oder `return` Sprunganweisung verlassen werden soll.

```
try { ... }
catch (Exception e) { ... }
finally { ...
}
```


135

Ausnahmen erzeugen

TECHNISCHE HOCHSCHULE LÜBECK

- Mit Hilfe der `throw`-Anweisung können Exceptions erzeugt werden.
- Methoden in denen dies erfolgen kann, müssen dies in ihrer Signatur mittels `throws` deutlich machen.
- Die Behandlung solcher Ausnahmen folgt den gezeigten Regeln.

```
public boolean isPrim(int n) throws ArithmeticException {
  if (n <= 0) throw new ArithmeticException(„n < 0“);
  if (n == 1) return false;
  for (int i = 2; i <= n/2; i++)
    if (n % i == 0)
      return false;
  return true;
}
```


136

Mini-Übung: Exception Raten

TECHNISCHE HOCHSCHULE LÜBECK

Welche der roten Strings werden ausgegeben?

```
try {
  boolean prim = isPrim(3);
  System.out.println("A");
  prim = isPrim(-1);
  System.out.println("B");
} catch (Exception e) {
  System.out.println("C");
} finally {
  System.out.println("D");
}
```

A
C
D

```
try {
  boolean prim = isPrim(3);
} catch (Exception e) {
  System.out.println("A");
} finally {
  System.out.println("B");
}
```

B

137

Assertions

TECHNISCHE HOCHSCHULE LÜBECK

Assertions	Assertionhandling	Einsatz als
• Dienen dazu Programmzustände sicherzustellen • Hierzu werden Invarianten formuliert. • Eine Invariante ist eine logische Bedingung die immer wahr ist und zu jedem Zeitpunkt einer Programmausführung gelten muss. • Gilt sie nicht ist das Programm nicht korrekt.	• kann über einen Schalter in der JVM • ein oder ausgeschaltet werden • Können so nur in einer Testphase aktiviert werden	• **Preconditions** – Sicherstellung korrekter Eingabewerte • **Postconditions** – Sicherstellung korrekter Ausgabewerte und Zustände • **Schleifeninvarianten** – Bedingungen zu Beginn und Ende von (besonders komplexen) Schleifen

138

Assertions

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
assert bedingung [: "Fehlertext"];
```

- Das Programm überprüft bei eingeschaltetem Assertion-Handling (`-enableassertions`) der JVM die Bedingung.
- Ist sie `true` wird die Programmausführung fortgesetzt, andernfalls wird ein `AssertionError` ausgelöst.

Beispiel: Pre- und Postconditions bei einem Sortieralgorithmus

TECHNISCHE HOCHSCHULE LÜBECK

Precondition Invariante

```
public void bubbleSort(int[] xs) {
  assert xs.length >= 0 : "Negative Array Länge";

  boolean unsorted=true; int temp;

  while (unsorted) {
    unsorted = false;
    for (int i=0; i < xs.length-1; i++) {
      if (!(xs[i] < xs[i+1])) {
        temp = xs[i]; xs[i] = xs[i+1]; xs[i+1] = temp;
        unsorted = true;
      }
    }
  }

  for (int i = 0; i < xs.length - 1; i++)
    assert xs[i] <= xs[i+1] : "Fehlerhafte Sortierung";
}
```

Postcondition Invariante

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Gegeben sei folgender Ausschnitt aus einem Programm:

```
int i = 20;
while (i > 0) {
  System.out.println(i);
  i -= 2;
}
```

Was bewirkt die Schleife? Wie sähe eine for-Schleife mit gleicher Ausgabe aus?

20
18
16
..
2

```
for (int i = 20; i > 0; i -= 2) {
  System.out.println(i);
}
```


141

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Sie wollen ein Schachbrett nummerieren in der Form

Konsole

```
1  2  3  4  5  6  7  8
2  3  4  5  6  7  8  9
3  4  5  6  7  8  9 10
4  5  6  7  8  9 10 11
5  6  7
6  7  8
7  8  9
8  9 10
```

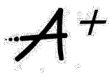
Formuli… …natier-
te Ausg…

```
for (int t = 1; t <= 8; t++) {
  for (int j = t; j < 8 + t; j++) {
    if (j <= 9) System.out.print(" ");
    System.out.print(j + " ");
  }
  System.out.println();
}
```

Quelle: Programmieren in Java

142

Zusammenfassung



TECHNISCHE HOCHSCHULE LÜBECK

- Anweisung und Blöcke
- Entscheidungsanweisung
  - `if`
  - `switch`
- Wiederholungsanweisung
  - `while`
  - `do while`
  - `for`
  - `for(each)`
- Spezielle Kontrollanweisungen
  - Exceptions
  - Assertions (nur zur Information)

143

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 4

(Kontrollstrukturen)

- W4.1: Maximum bestimmen
- W4.2: Summe berechnen
- W4.3: Tippspiel
- W4.4: PIN-Code-Generator
- W4.5: Dominosteine
- W4.11: Schachbrett (bitte abwandeln als einfache Konsolenausgabe)
- W4.13: Zahlenpalindrom

Nur eingeschränkt zu empfehlen

- *W4.6 bis 4.10, 4.12, 4.14*

144

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 7 (Strings)

- W7.1: String-Kompression
- W7.2: Split-Funktion
- W7.3: Geldschein-Blütencheck
- W7.4: Starkes Passwort
- W7.5: E-Mail-Check
- W7.6: Prüfen auf korrekte Klammerung
- W7.7: Sternchenmuster
- W7.8: URL-Encoding
- W7.11: IMDB-Einträge verarbeiten
- W7.12: Geheimsprache
- W7.13: Ähnlich klingende Worte
- W7.14: Textrahmen
- …

Nur eingeschränkt zu empfehlen

- *W7.9, 7.10, 7.15, 7.16*

145

Themen dieser Unit

TECHNISCHE HOCHSCHULE LÜBECK

Datentypen	Operatoren	Kontrollstrukturen	Routinen
• Werte • Variablen • Wertetypen	• Ausdrücke • Arithmetisch • Relational • Logisch • Bedingte Auswertung • Zuweisung • Type Cast	• Anweisungs-folgen wiederholen • Bedingte Ausführung von Anweisungs-folgen • Mehrfach-Verzweigungen • Schleifen	• Parametrisier-barer Code • Aufrufen wieder-verwendbarer Funktionalität

146

Zum Nachlesen ...

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 6

Methoden, Unterprogramme

147

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

Methoden definieren

Methodenaufruf und Methodenrückgabe

Besonderheiten (z.B. Methoden überladen)

148

Was sind Methoden?
Unterprogramme

TECHNISCHE HOCHSCHULE LÜBECK

- Durch Methoden wird ausführbarer Code **(ein Block)** unter einem Namen zusammengefasst.
- Dieser Code kann **parametrisiert**, d.h. mit Platzhaltern versehen, werden.
- Methoden können von anderen Stellen in einem Programm **aufgerufen** werden und kapseln so wieder verwendbare Funktionalität.
- Methoden dienen dazu Programme in sinnvolle Teilabschnitte zu gliedern **(Dekomposition)**.

Methoden

TECHNISCHE HOCHSCHULE LÜBECK

Syntax:

```
{Modifier} Typ Name ([Parameter]) {
   // Anweisungen der
   // Methode
}
```

- Definieren das Verhalten von Objekten
- Pendant zu Funktionen, Prozeduren, Routinen in anderen prozeduralen Programmiersprachen
- Es gibt keine von Klassen unabhängigen Methoden in JAVA
- Methoden haben Zugriff auf Daten des Objekts

Methoden Beispiel

TECHNISCHE HOCHSCHULE LÜBECK

```
public class EchoWords {

  public static String echo(String a, String b, String c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    System.out.println(echo("Hello", " ", "World"));
  }
}
```

(Zugriffs-)modifier | Datentyp der Rückgabe | Methoden-name | Aufruf-parameter

151

Methoden Beispiel

TECHNISCHE HOCHSCHULE LÜBECK

```
public class EchoWords {

  public static String echo(String a, String b, String c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    System.out.println(echo("Hello", " ", "World"));
  }
}
```

(Zugriffs-)modifier | Datentyp der Rückgabe | Methoden-name | Aufruf-parameter

152

Methoden Beispiel

TECHNISCHE HOCHSCHULE LÜBECK

```
public class EchoWords {

  public static String echo(String a, String b, String c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    System.out.println(echo("Hello", " ", "World"));
  }
}
```

(Zugriffs-)modifier | Datentyp der Rückgabe | Methoden-name | Aufruf-parameter

153

Methoden Beispiel

TECHNISCHE HOCHSCHULE LÜBECK

```
public class EchoWords {

  public static String echo(String a, String b, String c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    System.out.println(echo("Hello", " ", "World"));
  }
}
```

(Zugriffs-)modifier | Datentyp der Rückgabe | Methoden-name | Aufruf-parameter

154

Methoden
Beispiel

TECHNISCHE HOCHSCHULE LÜBECK

```
public class EchoWords {

  public static String echo(String a, String b, String c) {
    return a + b + c;
  }

  public static void main(String[] args) {
    System.out.println(echo("Hello", " ", "World"));
  }
}
```

(Zugriffs-)modifier | Datentyp der Rückgabe | Methoden-name | Aufruf-parameter

155

Methoden
Variable Parameterlisten

TECHNISCHE HOCHSCHULE LÜBECK

```
public static String echovar(String... words) {
  String ret = "";
  for(String w : words) {
    ret += w;
  }
  return ret;
}
```

- Der letzte Parameter kann variabel gehalten werden.
- Hier zu muss ein ... an den Parameter gehängt werden.
- Es können beliebig viele Parameter an eine Methode übergeben werden.
- Der Parameter wird wie ein Array (Liste von Werten, siehe Unit 3) des angegebenen Typs behandelt.

156

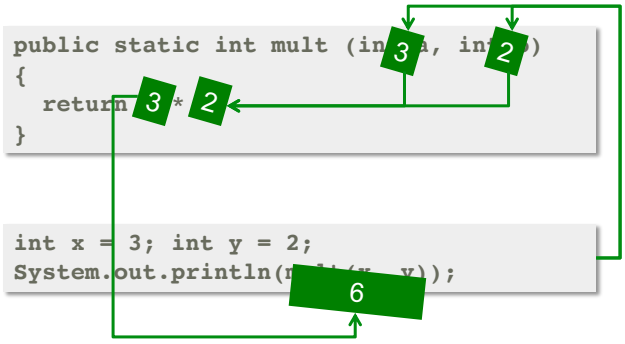
Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

157

Methodenaufruf und -rückgabe

TECHNISCHE HOCHSCHULE LÜBECK



Bei einem Methodenaufruf werden die Parameter der Methode (ebenso wie die lokalen Variablen) für jeden Aufruf neu erzeugt und mit den Aufrufwerten beschrieben.

Die Methode liefert ein Ergebnis zurück (`return`) welches anstelle des Methodenaufrufs im aufrufenden Ausdruck verwendet wird.

158

Methoden

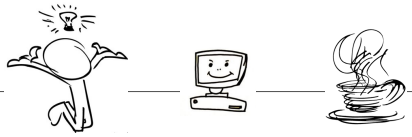
Stellen Sie sich immer DREI FRAGEN!

TECHNISCHE HOCHSCHULE LÜBECK

- **Was geht rein?**
  - Parameterabfolge
  - Datentypen der Parameter
- **Was kommt raus?**
  - Datentyp der Rückgabe
- **Wie heißt die Funktionalität?**
  - Name der Methode

159

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Schreiben Sie nun eine Methode, die die durchschnittliche Wortlänge eines Satzes bestimmt und wie folgt aufgerufen werden kann.

```
System.out.println(
      averageWordLength("Dies ist ein Beispiel.")
);
```

```
4.75
```


160

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Lösung:

161

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Schreiben Sie nun eine Methode, die die durchschnittliche Wortlänge mehrerer Sätze bestimmt und wie folgt aufgerufen werden kann.

```
System.out.println(
  averageWordLength(
    "Dies ist ein Beispiel.",
    "Dies ist noch ein Beispiel.",
    "Und hier ein weiteres Beispiel.",
    "Die Methode soll beliebig viele Sätze",
    "verarbeiten können."
  )
);
```

```
5.409090909090909
```


162

Mini-Übung:



TECHNISCHE HOCHSCHULE LÜBECK

Lösung:

163

Worum geht es nun?

TECHNISCHE HOCHSCHULE LÜBECK

Methoden definieren

Methodenaufruf und Methodenrückgabe

Besonderheiten (Methoden überladen, CBR, CBV)

164

Überladen von Methoden

TECHNISCHE HOCHSCHULE LÜBECK

Jede Methode muss einen Namen haben. Zwei Methoden können jedoch denselben Namen haben, sofern sich ihre Parametrisierung unterscheidet, man nennt dies eine Methode **überladen**.

```
public static int max(int a, int b)
{
  return (a > b) ? a : b;
}
```

```
public static double max(double a, double b)
{
  return (a > b) ? a : b;
}
```

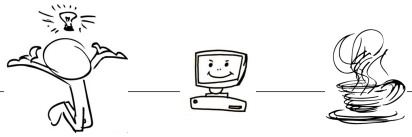
JAVA **unterscheidet** Methoden gleichen Namens

- anhand der **Zahl** der Parameter
- anhand des **Typs** der Parameter
- anhand der **Position** der Parameter

JAVA **unterscheidet** Methoden gleichen Namens **nicht** nach

- **Namen** der Parameter
- dem **Rückgabetyp** der Methode

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Gegeben sei folgende Methodendefinition, welche Überladungen sind dann zulässig?

```
public static int max(int a, int b)
{
  return (a > b) ? a : b;
}
```

`public static int max(int x, int y) { ... }`	Nicht nach Namen
`public static double max(int a, int b) { ... }`	Nicht nach Rückgabe
`public static int max(double a, int b) { ... }`	Typ der Argumente
`public static int max(int a, int b, int c) { ... }`	Anz. der Argumente

Methoden
Call by reference vs. Call by value

TECHNISCHE HOCHSCHULE LÜBECK

- Es gibt in gängigen Programmiersprachen grundsätzlich zwei Arten Parameter an eine Routine zu übergeben

Call by reference	Call by value
• Es wird ein Zeiger auf eine Variable übergeben. • Innerhalb der Routine wird über den Zeiger auf der Variable außerhalb der Routine gearbeitet. • **Der Inhalt der Variable außerhalb der Routine wird verändert.**	• Der Wert einer Variable wird in die Parametervariable kopiert. • Innerhalb der Routine wird auf der Kopie gearbeitet. • **Der Inhalt der Variable außerhalb der Routine wird nicht verändert.**

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
167

167

Methoden
Call by value in JAVA (I)

TECHNISCHE HOCHSCHULE LÜBECK

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
168

168

Methoden
Call by value in JAVA (II)

TECHNISCHE HOCHSCHULE LÜBECK

- In JAVA gibt es zwei Arten von Datentypen (Variablen)

Primitive Datentypen	Referenztypen

• Logische (boolean)
• Zeichentyp (char, String)
• Ganzzahlen (byte, short, int, long)
• Fließkommazahlen (float, double)
• Strings
• Hier wird der Inhalt in der Variable als Wert gespeichert
	• Alle anderen Variablen sind Referenztypen und beinhalten lediglich Verweise auf die eigentlichen Inhalte irgendwo im Hauptspeicher
	• Dies gilt z.B:
	  • Arrays (siehe Unit 3)
	  • alle Klassen (siehe Unit 3)

Methoden
Call by value in JAVA (III)

TECHNISCHE HOCHSCHULE LÜBECK

- Dies bedeutet für JAVA

Call by value mit primitiven Datentypen	Call by value mit Referenztypen
• Logische Typen (bool) • Zeichentypen (char) • Ganzzahlen (int, etc) • Fließkommazahlen (float, etc) • Zeichenketten (Strings) • funktioniert in der **Call by value** Semantik	• Arrays • Klassen • funktioniert dennoch in der **Call by reference** Semantik

Methoden
Call by value Beispiele (I)

TECHNISCHE HOCHSCHULE LÜBECK

Methodenaufruf mit primitiven Datentypen

```
void add_var1(int a, int b) {
  String result = a + " + " + b + " = " + (a+b);
  System.out.println(result);
  a = 0;
  b = 0;
}
```

Der folgende Aufruf erzeugt

```
int a = 5;
int b = 3;
add_var1(a, b);
add_var1(a, b);
```

welche Ausgabe?

```
5 + 3 = 8
5 + 3 = 8
```


171

Methoden
Call by value Beispiele (II)

TECHNISCHE HOCHSCHULE LÜBECK

Methodenaufruf mit Referenztypen

```
void add_var2(int[] xs) {
  String result = xs[0] + " + " + xs[1] + " = " +
                  (xs[0] + xs[1]);
  System.out.println(result);
  xs[0] = 0;
  xs[1] = 0;
}
```

Der folgende Aufruf erzeugt

```
int[] xs = {5, 3};

add_var2(xs);
add_var2(xs);
```

welche Ausgabe?

```
5 + 3 = 8
0 + 0 = 0
```


172

Methoden
Call by value Beispiele (III)

TECHNISCHE HOCHSCHULE LÜBECK

Methodenaufruf mit variabler Anzahl primitiver Datentypen

```
void add_var3(int... xs) {
  String result = xs[0] + " + " + xs[1] + " = " +
                  (xs[0] + xs[1]);
  System.out.println(result);
  xs[0] = 0;
  xs[1] = 0;
}
```

Der folgende Aufruf erzeugt

```
int a = 5;
int b = 3;
add_var3(a, b);
add_var3(a, b);
```

welche Ausgabe?

```
5 + 3 = 8
5 + 3 = 8
```


173

Methoden
Call by value Beispiele (IV)

TECHNISCHE HOCHSCHULE LÜBECK

Methodenaufruf mit Referenztypen und erzwungenem CBV Verhalten:

```
void add_var2(int[] xs) {
  String result = xs[0] + " + " + xs[1] + " = " +
                  (xs[0] + xs[1]);
  System.out.println(result);
  xs[0] = 0;
  xs[1] = 0;
}
```

Der folgende Aufruf erzeugt

```
int[] xs = {5, 3};

add_var2(xs.clone());
add_var2(xs.clone());
```

welche Ausgabe?

```
5 + 3 = 8
5 + 3 = 8
```


174

Methoden
Call by value mittels der clone-Methode

TECHNISCHE HOCHSCHULE LÜBECK

clone()

- erzeugt ein Duplikat eines Objekts im Hauptspeicher
- und kann dazu genutzt werden,
- die Call by value Semantik
- auch bei Referenztypen sicherzustellen.

175

Methoden
Call by value Beispiele (V)

TECHNISCHE HOCHSCHULE LÜBECK

Methodenaufruf mit unterbundenem CBR Verhalten:

```
void add_var2(final int[] xs) {

String result = xs[0] + " + " + xs[1] + " = " +
                  (xs[0] + xs[1]);
  System.out.println(result);

  xs[0] = 0; // Diese beiden Zeilen würden dann
  xs[1] = 0; // gar nicht erst kompiliert werden.

}
```

Das Schlüsselwort **`final`** vor einem Parameter sagt dem Compiler, dass er keine schreibenden Zugriffe auf Parameter gestatten darf.

Bspw. verlangt der SUN Coding Standard, dass alle Parameter einer Methode mit dem Schlüsselwort final zu deklarieren sind.

Noch schöner wäre wenn Java alle Parameter automatisch als `final` behandeln würde und änderbare bspw. mit einem Schlüsselwort `changeable` deklariert werden könnten.

176

Überblick über die Eigenschaften von Datentypen in JAVA

TECHNISCHE HOCHSCHULE LÜBECK

Strings und Arrays sind etwas zwischen primitiven Datentypen und Referenztypen

	Primitiver Datentyp	Referenzdatentyp	Objektcharakter	Wertesemantik	Referenzsemantik
Ganze Zahlen (`byte, short, int, long`)	x			x	
Boolesche Werte (`boolean`)	x			x	
Fließkommazahlen (`float, double`)	x			x	
Zeichenketten (`String`)			x	x	
`Array`			x		x
Klassen (alles von `Object` abgeleitete)		x	x		x

177

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Schreiben Sie eine Methode, die den Tangens einer **`double`**-Zahl, die als Parameter übergeben wird, berechnet. Implementieren Sie den Tangens gemäß der Formel $\tan(x) = \sin(x)/\cos(x)$. Sie dürfen die Methoden `Math.sin` und `Math.cos` zur Berechnung von Sinus und Cosinus verwenden, jedoch innerhalb der Methode keine einzige Variable vereinbaren.

```
public static double tan(double x) {
       return Math.sin(x) / Math.cos(x);
}
```


178

Miniübung:



TECHNISCHE HOCHSCHULE LÜBECK

Schreiben Sie nun eine Methode `reverse`, die einen String umdreht und zurückgibt.

Folgende Zeile:

```
System.out.println(reverse("Hello"));
```

soll dies ausgeben:

```
olleH
```

```
public static String reverse(String s) {
  String ret = "";
  for (char c : s.toCharArray()) {
    ret = c + ret;
  }
  return ret;
}
```


Zusammenfassung

TECHNISCHE HOCHSCHULE LÜBECK

- **Methodendefinition**
  - Zugriffsmodifikator
  - Rückgabetyp
  - Name
  - Parameter (inkl. variabler Anzahl an Parametern)

- **Methodenaufruf und -rückgabe**
- **Besonderheiten bei Methoden**
  - Überladen von Methoden
  - Call by Value/Call by Reference
  - Primitive Datentypen als Parameter
  - Referenzdatentypen als Parameter

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad

TECHNISCHE HOCHSCHULE LÜBECK

Kapitel 5 (Funktionen)

- W5.1: Endliches Produkt
- W5.2: Fakultät
- W5.3: Konfektionsgröße
- W5.4: Schaltjahr Prüfung
- W5.5: Literzahlen umwandeln
- W5.6: LKW-Maut
- W5.7: Analoger Uhrzeiger
- W5.8: Körperoberfläche 2.0
- W5.9: Sportwetten
- W5.10: GPS-Luftlinie
- W5.11: IBAN-Generator
- W5.12: Sanduhr

Nur eingeschränkt zu empfehlen

- *W5.13 bis 5.16*

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
181

181