

**Vorlesung**

**Programmieren I und II**

Unit 2  
Grundlagen imperativer Programmierung

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

**Prof. Dr. rer. nat.  
Nane Kratzke**

*Praktische Informatik und betriebliche Informationssysteme*

- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: [kratzke@fh-luebeck.de](mailto:kratzke@fh-luebeck.de)

@NaneKratzke

Updates der Handouts auch über Twitter #prog\_inf und #prog\_itd

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

**Units**

1. Semester	Unit 1 Einführung und Grundbegriffe	Unit 2 Grundlagen imperativer Programmierung	Unit 3 Selbstdefinierbare Basistypen 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 Erbschaften	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

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

**Abgedeckte Ziele dieser UNIT**

Kennen existierender Programmierparadigmen und Laufzeitmodelle	<b>Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)</b>	<b>Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)</b>	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmusbegriffs, 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 Multithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

**Themen dieser Unit**

<p><b>Datentypen</b></p> <ul style="list-style-type: none"> <li>Werte</li> <li>Variablen</li> <li>Werttypen</li> </ul>	<p><b>Operatoren</b></p> <ul style="list-style-type: none"> <li>Ausdrücke</li> <li>Arithmetisch</li> <li>Relational</li> <li>Logisch</li> <li>Bedingte Auswertung</li> <li>Zuweisung</li> <li>Type Cast</li> </ul>	<p><b>Kontrollstrukturen</b></p> <ul style="list-style-type: none"> <li>Anweisungsfolgen wiederholen</li> <li>Bedingte Ausführung von Anweisungsfolgen</li> <li>Mehrfach-Verzweigungen</li> <li>Schleifen</li> </ul>	<p><b>Routinen</b></p> <ul style="list-style-type: none"> <li>Parametrisierbarer Code</li> <li>Aufrufen wieder-verwendbarer Funktionalität</li> </ul>
--	--	--	---

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

**Zum Nachlesen ...**

**Kapitel 4**

Grundlagen der Programmierung in JAVA

**Abschnitt 4.3**

Einfache Datentypen

**Abschnitt 4.4**

Der Umgang mit einfachen Datentypen

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

### Worum geht es nun?



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

### Variablen



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

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

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

### Primitive Datentypen




- 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 <sup>7</sup> ... 2 <sup>7</sup> -1	0
short	2	-2 <sup>15</sup> ... 2 <sup>15</sup> -1	0
int	4	-2 <sup>31</sup> ... 2 <sup>31</sup> -1	0
long	8	-2 <sup>63</sup> ... 2 <sup>63</sup> -1	0
float	4	±3,402823... * 10 <sup>38</sup>	0.0
double	8	±1,797693... * 10 <sup>308</sup>	0.0

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

### Worum geht es nun?



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

### Ganzzahlige Datentypen

**byte, short, int, long (integrale Typen)**



<p>Vier ganzzahlige Datentypen</p> <ul style="list-style-type: none"> <li><b>byte</b> – 1 Byte Länge</li> <li><b>short</b> – 2 Byte Länge</li> <li><b>int</b> – 4 Byte Länge</li> <li><b>long</b> – 8 Byte Länge</li> </ul>	<p>Für alle ganzzahligen Datentypen gilt:</p> <ul style="list-style-type: none"> <li>Vorzeichenbehaftet</li> <li>Länge ist auf allen Plattformen gleich</li> </ul>
---	--

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

### Ganzzahlige Datentypen

**byte, short, int, long (integrale Typen)**



<pre>byte max = Byte.MAX_VALUE; byte min = Byte.MIN_VALUE; System.out.println(min); System.out.println(max);</pre>	<pre>short max = Short.MAX_VALUE; short min = Short.MIN_VALUE; System.out.println(min); System.out.println(max);</pre>
-128	-32768
127	32767
<pre>int max = Integer.MAX_VALUE; int min = Integer.MIN_VALUE; System.out.println(min); System.out.println(max);</pre>	<pre>long max = Long.MAX_VALUE; long min = Long.MIN_VALUE; System.out.println(min); System.out.println(max);</pre>
-2147483648	-9223372036854775808
2147483647	9223372036854775807

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

## Gleitkommatypen float, double

Zwei Datentypen für Nichtganzzahlen

- float**
  - 4 Byte Länge
  - Einfache Genauigkeit
- Double**
  - 8 Byte Länge
  - Doppelte Genauigkeit

Für alle Fließkommatypen gilt:

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

**Vorkomma.Nachkomma[eExponent][f[d]**

**Beispiele:**

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 13

## Gleitkommatypen float, double

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 14

## Miniübung:

```
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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 15

## Miniübung:

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.");
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 16

## Wahrheitstyp (I) boolean

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!

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 17

## Wahrheitstyp (II) boolean

```
boolean a = false;

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

Korrektter Einsatz eines booleschen Variable in JAVA.

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

Falscher Einsatz einer ganzzahligen Variable als boolean Ersatz in JAVA.  
Diverse Prog-Sprachen erlauben so etwas.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 18

### Zeichen

#### char

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,

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 19

### Zeichentyp

#### char – (ESC-Sequenzen)

ESC-Sequenz	Bedeutung
\b	Rückschritt (Backspace, dass durch die DEL-Taste erzeugte Zeichen)
\t	Horizontaler Tabulator (das durch die TAB-Taste erzeugte Zeichen)
\n	<b>Zeilenschaltung (Newline)</b>
\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 20

### Worum geht es nun?

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 21

### Strings

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

Dies ist ein Satz.

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

### Methoden der Klasse String

**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);
```

051015

Dies ist ein Satz.

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

### Methoden der Klasse String

**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
```

```
String empty2 = “ ”;
length = empty2.length();
1
```

051015

Dies ist ein Satz.

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

## Methoden der Klasse String

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

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

gleich = hallo.equalsIgnoreCase(HALLO);
true
```

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

## Methoden der Klasse String

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 .

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

## Methoden der Klasse String

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 .

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

## Methoden der Klasse String

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.replace(„ist ein“, „kein“);
Dies kein Satz.
```

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

## Methoden der Klasse String

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

```
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 .

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

## Methoden der Klasse String

Weitere Methoden der Klasse `String` finden Sie hier:  
<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>



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

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

## Worum geht es nun?

Ganzzahlige Datentypen

Gleitkommatypen

Wahrheitstyp

Zeichen

Zeichenketten

Typumwandlungen

Deklaration und Initialisierung

Wertezuweisung an Variablen

Auslesen von Werten aus Variablen

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

## Typumwandlung Type Casting

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. int) 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
```

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

## Implizite und explizite Typumwandlung Implicite and explicit type casting

↑ implizit	Typname	größter Wert	kleinster Wert	Länge	↓ explizit
	<b>byte</b>	127	-128	8 Bits	
	<b>short</b>	32767	-32768	16 Bits	
	<b>int</b>	2147483647	-2147483648	32 Bits	
	<b>long</b>	9223372036854775807	-9223372036854775808	64 Bits	

  

↑ implizit	Typname	größter positiver Wert	kleinster positiver Wert	Länge	↓ explizit
	<b>float</b>	≈ 3.4028234663852886E+038	≈ 1.4012984643248173E-045	32 Bits	
	<b>double</b>	≈ 1.7976931348623157E+308	≈ 4.9406564584124654E-324	64 Bits	

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

## Miniübung:

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

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

Ergibt welche Ausgabe?

130 = -126

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

## Worum geht es nun?

Ganzzahlige Datentypen

Gleitkommatypen

Wahrheitstyp

Zeichen

Zeichenketten

Typumwandlungen

Deklaration und Initialisierung

Wertezuweisung an Variablen

Auslesen von Werten aus Variablen

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

## Variablen

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.

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

## Deklaration von Variablen



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
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 37

## Wertezuweisung an Variablen



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)
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 38

## Lesen aus Variablen



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 Routineergebnisses
System.out.println(Math.sqrt(var));
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 39

## Mini-Übung:



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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 40

## Mini-Übung:



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'

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 41

## Zusammenfassung



### Einfache Datentypen

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



### Variablen

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



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 42

## Themen dieser Unit



Datentypen	Operatoren	Kontrollstrukturen	Routinen
<ul style="list-style-type: none"> <li>• Werte</li> <li>• Variablen</li> <li>• Wertetypen</li> </ul>	<ul style="list-style-type: none"> <li>• Ausdrücke</li> <li>• Arithmetisch</li> <li>• Relational</li> <li>• Logisch</li> <li>• Bedingte Auswertung</li> <li>• Zuweisung</li> <li>• Type Cast</li> </ul>	<ul style="list-style-type: none"> <li>• Anweisungsfolgen wiederholen</li> <li>• Bedingte Ausführung von Anweisungsfolgen</li> <li>• Mehrfach-Verzweigungen</li> <li>• Schleifen</li> </ul>	<ul style="list-style-type: none"> <li>• Parametrisierbarer Code</li> <li>• Aufrufen wiederverwendbarer Funktionalität</li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 43

## Zum Nachlesen ...




**Kapitel 4**  
Grundlagen der Programmierung in JAVA

**Abschnitt 4.4.2**  
Operatoren und Ausdrücke

**Abschnitt 4.4.3**  
Allgemeine Ausdrücke

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 44

## Worum geht es nun?



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 45

## Operatoren



- 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.: Zuweisung**

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

**Bsp.: Vergleich**

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

**Bsp.: Zuweisung und Vergleich**

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 46

## Worum geht es nun?



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 47

## Arithmetische Operatoren



Erwarten numerische Operanden

↓

Liefern numerische Operanden

↓

Dienen numerischen Berechnungen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 48

### Liste arithmetischer Operatoren

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
--	Prädecrement	--a ergibt a - 1
--	Postdecrement	a-- ergibt a - 1

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

### Miniübung zu arithmetischen Operatoren

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

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

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

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

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

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

### Miniübung zu arithmetischen Prä- und Postfix-Operatoren

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

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

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

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

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

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

### Worum geht es nun?

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

### Relationale Operatoren

Erwarten beliebige Ausdrücke

↓

Liefern boolesche Werte

↓

Dienen dem Vergleich von Ausdrücken

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

### Liste relationaler Operatoren

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

### Miniübung zu relationalen Operatoren

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```
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 55

### Miniübung zu relationalen Operatoren (Referenztypen)

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```
List v = new LinkedList(); v.add(5);
List w = v;
List x = new LinkedList(); x.add(5);
```

v: [5]  
w: [5]  
x: [5]

System.out.println(v == w); **true**

System.out.println(v != x); **true**

System.out.println(w == x); **false**

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

### Worum geht es nun?

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

### Logische Operatoren

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```

Erwarten boolesche Werte
↓
Liefen boolesche Werte
↓
Dienen der Formulierung logischer Bedingungen
    
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 58

### Liste logischer Operatoren

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 59

### Miniübung zu logischen Operatoren

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```
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**

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 60

### Miniübung zu logischen Operatoren (Short Circuit Verhalten)

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```
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!
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 41

### Worum geht es nun?

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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 42

### Bedingte Auswertung

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

- 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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 43

### Miniübung: Bedingte Auswertung

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```
int a = 6;
int b = 2;
System.out.println(a % 2 == 0 ? „Hello“ : „World“);
```

Hello

```
String hw = „Hello World“;
String h = „Hello“;
System.out.println(hw.endsWith(h) ? h : hw);
```

Hello World

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 44

### Worum geht es nun?

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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 45

### Zuweisungs-Operatoren

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```

    graph TD
    A[Erwarten Ausdrücke] --> B[Liefern Werte]
    B --> C[Dienen der Zuweisung von ausgewerteten Ausdrücken an Variablen]
    
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 46

## Liste der Zuweisungsoperatoren

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.**

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 67

## Zuweisungen sind auch Ausdrücke

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;
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 68

## Miniübung zu Zuweisungsoperatoren

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

System.out.println(b = a);      5 (Ausdrucksrückg.)

System.out.println(b);          5 (Wert von b)

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

System.out.println(a);          1 (Wert von a)

a = b \*= ++a;                      a == 10; b == 10  
   nicht a == 2!

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 69

## Worum geht es nun?

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 70

## String Verkettung (Konkatenation)

- 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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 71

## String Konkatenation bei Referenztypen

Gegeben sind diese beiden Klassen:

```
class A {
    public String toString() {
        return „Hello“;
    }
}
class B {
    public String toString() {
        return „World“;
    }
}
```

Wie lautet die Ausgabe dieser Statements?

```
A a = new A();
B b = new B();
System.out.println(a + " " + b);
```

Hello World

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 72

### Worum geht es nun?

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 73

### Type Cast Operator

- Typumwandlung
- Der Ausdruck
  - `(type) a`
  - wandelt die Variable `a` in den Datentyp `type`
- Wird z.B. bei der Arbeit mit Collections benötigt
  - Diese basieren alle auf dem primitivsten Element `Object`
  - Müssen spezifische Methoden der Elemente genutzt werden, so kann explizit auf den erforderlichen Type „gecastet“ werden.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 74

### Type Cast Operator Beispiel

```
Stack s = new Stack();
Auto auto1 = new Auto();
s.push(auto1);

Auto auto2;
auto2 = (Auto)s.pop()
```

*/\* Hier ist ein Type Cast erforderlich, da der Compiler davon ausgeht, es wird ein Element des Typs Object zurückgeliefert, da Stack nur auf Object definiert ist.\*/*

Collections und selbst definierte Referenzdatentypen werden noch in Unit 3 behandelt.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 75

### Worum geht es nun?

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 76

### new Operator

In JAVA werden Objekte und Arrays mit dem **new Operator** erzeugt.

Das Erzeugen sind Ausdrücke, die eine Referenz auf das gerade erzeugte Array oder Objekt zurückliefern.

Der Unterschied zwischen Referenzdatentypen (Erzeugung mit dem new Operator) und primitiven Datentypen wird in Unit 3 erläutert.

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

### Worum geht es nun?

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 78

## Ausdrücke

Ausdrücke setzen sich in Programmiersprachen üblicherweise aus Operatoren und Operanden zusammen und dienen dem Ausdruck **komplexerer Berechnungen** oder **logischer Bedingungen**. Die Operanden selbst können dabei wieder

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

Der einfachste Ausdruck ist einfach die Angabe einer Variablen.

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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 79

## Vorrangregeln

Kennen Sie noch die Regel:  
**Punktrechnung vor Strichrechnung?**

Java kennt dasselbe mit ein paar mehr zu berücksichtigenden Gruppen.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 80

## Vorrangregeln Operatorgruppen

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

Steigende Bindung der Operatoren

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 81

## Veranschaulichung

```
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;
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 82

## Veranschaulichung

```
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;
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 83

## Veranschaulichung

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

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

```
int e = 25 - 2*a*b + Math.pow(b, 2);
```

```
int e = 25 - 2*5*b + Math.pow(b, 2);
```

```
int e = 25 - 10*b + Math.pow(b, 2);
```

```
int e = 25 - 10*3 + 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;
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 84

**Miniübung**    

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!  

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **85**

**Miniübung**    

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`

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **86**

**Miniübung:**    

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);
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **87**

**Zusammenfassung**  

- Klassische Operatoren**
  - Arithmetisch
  - Relational
  - Logisch
  - Zuweisung
  - bedingte Auswertung ( $x ? A : B$ )
- Weitere Operatoren**
  - Stringkonkatenation
  - Type Cast (`t.type`)
  - new Operator
- Ausdrücke**
  - Vorrang- und Auswerteregeln für Operatoren




Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **88**

**Themen dieser Unit** 

Datentypen	Operatoren	Kontrollstrukturen	Routinen
<ul style="list-style-type: none"> <li>Werte</li> <li>Variablen</li> <li>Werttypen</li> </ul>	<ul style="list-style-type: none"> <li>Ausdrücke</li> <li>Arithmetisch</li> <li>Relational</li> <li>Logisch</li> <li>Bedingte Auswertung</li> <li>Zuweisung</li> <li>Type Cast</li> </ul>	<ul style="list-style-type: none"> <li>Anweisungsfolgen wiederholen</li> <li>Bedingte Ausführung von Anweisungsfolgen</li> <li>Mehrfach-Verzweigungen</li> <li>Schleifen</li> </ul>	<ul style="list-style-type: none"> <li>Parametrisierbarer Code</li> <li>Aufrufen wiederverwendbarer Funktionalität</li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **89**

**Zum Nachlesen ...** 



**Kapitel 4**  
 Grundlagen der Programmierung in JAVA  
**Abschnitt 4.5**  
 Anweisungen und Ablaufsteuerung

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **90**

### Worum geht es nun?

Anweisungen und Blöcke

Bedingte Anweisungen

Wiederholungsanweisungen

Spezielle Kontrollanweisungen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 91

### Anweisungen

- 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)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 92

### Blöcke { }

- 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 der Strukturierung von Quelltexten.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 93

### Blöcke (Beispiel)

```

int x = 5;           // Anfang des äusseren Blocks
x++;                // Deklarationsanweisung und Zuweisung
{                   // Postfix-Inkrement-Anweisung
    long y;         // Anfang des ersten inneren Blocks
    y = x + 123456789; // Deklarationsanweisung
    System.out.println(y); // Ausgabeanweisung/Methodenaufruf
    ;               // Leer 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 äusseren Blocks
    
```

**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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 94

### Worum geht es nun?

Anweisungen und Blöcke

Bedingte Anweisungen

Wiederholungsanweisungen

Spezielle Kontrollanweisungen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 95

### Verzweigungen if-Abfrage

- 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;
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 96

### Syntax und Flowchart der if Anweisung

```

    graph TD
      START([START]) --> test{test}
      test -- true --> anweisung[anweisung(en);]
      anweisung --> STOP([STOP])
      test -- false --> STOP
  
```

**Syntaxregel:**

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

**Beispiel:**

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 97

### Syntax und Flowchart der if else Anweisung

```

    graph TD
      START([START]) --> test{test}
      test -- true --> anweisung1[anweisung(en);]
      anweisung1 --> STOP([STOP])
      test -- false --> anweisung2[anweisung(en);]
      anweisung2 --> STOP
  
```

**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");
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 98

### Geschachtelte ifs und „dangling else“

- 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;
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 99

### Mini-Übung if Strukturen Raten (I)

```
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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 100

### Mini-Übung if Strukturen Raten (II)

```
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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 101

### Mini-Übung if Strukturen Raten (III)

```
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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 102

### Mini-Übung if Strukturen Raten (IV)

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

```
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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 103

### Verzweigungen Switch-Anweisung

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

- 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;
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 104

### Verzweigungen Switch-Anweisung (Beispiel)

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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

Rest 2

Rest 3      Will man das nicht, muss man das Label mit einer break Anweisung dazu zwingen, am Ende der switch-Anweisung fortzusetzen.

Rest 0

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 105

### Verzweigungen Switch-Anweisung (Beispiel)

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 106

### Worum geht es nun?

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

Anweisungen und Blöcke

Bedingte Anweisungen

Wiederholungsanweisungen

Spezielle Kontrollanweisungen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 107

### Abweisende Schleife

FACH HOCHSCHULE LÜBECK  
University of Applied Sciences

**Syntax:**

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

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 108

### Syntax und Flowchart der while Anweisung

**Syntaxregel:**

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

**Beispiel:**

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 109

### Abweisende Schleife Mini-Übung: while Schleifen

```
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!

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 110

### Nicht abweisende Schleife

**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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 111

### Syntax und Flowchart der do while Anweisung

**Syntaxregel:**

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

**Beispiel:**

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 112

### Nicht abweisende Schleife Mini-Übung: do while Schleifen

```
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);
```

0  
0  
0  
... endet nie!

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 113

### Zählschleife for – klassische Variante

**Syntax:**

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

Init Ausdruck	Test Ausdruck	Update Ausdruck
<ul style="list-style-type: none"> <li>Aufruf einmalig <b>vor Start</b> der Schleife</li> <li>Optional</li> <li>mehrere kommaseparierte Ausdrücke mögl.</li> <li>Variablendeklaration möglich</li> </ul>	<ul style="list-style-type: none"> <li>Aufruf <b>jew. am Anfang</b> einer Schleife</li> <li>Optional, wenn nicht angegeben wird true gesetzt</li> <li>Schleife wird nur ausgeführt, wenn Ausdruck true</li> </ul>	<ul style="list-style-type: none"> <li>Aufruf <b>jew. am Ende</b> der Schleife</li> <li>Optional</li> <li>Mehrere kommaseparierte Ausdrücke möglich</li> <li>Dient dazu den Schleifenzähler zu ändern</li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 114

### Syntax und Flowchart der for Anweisung

**Syntaxregel:**

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

**Beispiel:**

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

```

graph TD
    START([START]) --> init[init]
    init --> test{test}
    test -- true --> anweisung[anweisung(en)]
    anweisung --> update[update]
    update --> test
    test -- false --> STOP([STOP])
    
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 115

### Zählschleife for – klassische Variante (Beispiele)

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

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

1 //init-Ausdruck – einmalig  
2 //test-Ausdruck – vor Schleifenlauf  
3 //Update-Ausdruck – am Ende

Jede for-Schleife kann nach diesem Muster umformuliert werden.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 116

### Zählschleife for – klassische Variante (Beispiele)

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

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

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 117

### Zählschleife foreach Variante

**Syntax:**

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

- Object ist ein „Laufobjekt“ oder eine „Zählvariable“
- 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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 118

### Foreach Schleife Mini-Übung: foreach Schleifen

```
List l = Arrays.asList(7, 5, 3, 1);
for (Object o : l)
    System.out.println(o);
```

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 119

### Worum geht es nun?

Anweisungen und Blöcke

Bedingte Anweisungen

Wiederholungsanweisungen

Spezielle Kontrollanweisungen

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 120

## Exceptions

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 121

## Grundprinzip des Exception Mechanismus

- Laufzeitfehler oder vom Entwickler gewollte Bedingung löst Exception aus
- Behandlung in ein einem 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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 122

## Behandlung von Exceptions try-catch Anweisung

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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 123

## Behandlung von Exceptions Beispiel (OutOfBounds)

```
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 `IndexOutOfBoundsException` aus, die in der catch Klausel gefangen und behandelt wird.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 124

## Exception-spezifische catch Klauseln

- 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 (ArrayOutOfBoundsException e) { ... }  
catch (NumberFormatException e) { ... }  
catch (IndexOutOfBoundsException e) { ... }
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 125

## Die finally Klausel

- 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 { ... }
```

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

## Ausnahmen erzeugen

FACH HOCHSCHULE LÜBECK  
 University of Applied Sciences

- 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;
}
    
```

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

## Mini-Übung: Exception Raten

FACH HOCHSCHULE LÜBECK  
 University of Applied Sciences

Welche der roten Strings werden ausgegeben?

```

try {
    boolean prim = isPrim(3);
    System.out.println("isPrim 3 passed");
    prim = isPrim(-1);
    System.out.println("isPrim -1 passed");
} catch (Exception e) {
    System.out.println("catch passed.");
} finally {
    System.out.println("Finally passed.");
}
    
```

isPrim 3 passed  
Catch passed.  
Finally passed.

```

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

Finally passed.

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

## Assertions

FACH HOCHSCHULE LÜBECK  
 University of Applied Sciences

Assertions	Assertionhandling	Einsatz als
<ul style="list-style-type: none"> <li>Dienen dazu Programmzustände sicherzustellen</li> <li>Hierzu werden Invarianten formuliert.</li> <li>Eine Invariante ist eine logische Bedingung die immer wahr ist und zu jedem Zeitpunkt einer Programmausführung gelten muss.</li> <li>Gilt sie nicht ist das Programm nicht korrekt.</li> </ul>	<ul style="list-style-type: none"> <li>kann über einen Schalter in der JVM ein oder ausgeschaltet werden</li> <li>Können so nur in einer Testphase aktiviert werden</li> </ul>	<ul style="list-style-type: none"> <li><b>Preconditions</b> – Sicherstellung korrekter Eingabewerte</li> <li><b>Postconditions</b> – Sicherstellung korrekter Ausgabewerte und Zustände</li> <li><b>Schleifeninvarianten</b> – Bedingungen zu Beginn und Ende von (besonders komplexen) Schleifen</li> </ul>

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

## Assertions

FACH HOCHSCHULE LÜBECK  
 University of Applied Sciences

Syntax:

```

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

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

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

## Beispiel: Pre- und Postconditions bei einem Sortieralgorithmus

FACH HOCHSCHULE LÜBECK  
 University of Applied Sciences

```

public void bubbleSort(int[] xs) {
    Precondition Invariante
    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;
            }
        }
    }

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

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

## Miniübung:

FACH HOCHSCHULE LÜBECK  
 University of Applied Sciences

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?

```

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

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

**Miniübung:**






Sie wollen ein Schachbrett nummerieren in der Form

```

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
    
```

Konsole

```

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();
}
    
```

Formuliere die Ausgabe

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 133

**Zusammenfassung**




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




Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 134

**Themen dieser Unit**



Datentypen	Operatoren	Kontrollstrukturen	Routinen
<ul style="list-style-type: none"> <li>• Werte</li> <li>• Variablen</li> <li>• Wertetypen</li> </ul>	<ul style="list-style-type: none"> <li>• Ausdrücke</li> <li>• Arithmetisch</li> <li>• Relational</li> <li>• Logisch</li> <li>• Bedingte Auswertung</li> <li>• Zuweisung</li> <li>• Type Cast</li> </ul>	<ul style="list-style-type: none"> <li>• Anweisungsfolgen wiederholen</li> <li>• Bedingte Ausführung von Anweisungsfolgen</li> <li>• Mehrfach-Verzweigungen</li> <li>• Schleifen</li> </ul>	<ul style="list-style-type: none"> <li>• Parametrisierbarer Code</li> <li>• Aufrufen wiederverwendbarer Funktionalität</li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 135

**Zum Nachlesen ...**




**Kapitel 6**  
Methoden, Unterprogramme

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 136

**Worum geht es nun?**



Methoden definieren

Methodenaufruf und Methodenrückgabe

Besonderheiten (z.B. Methoden überladen)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 137

**Was sind Methoden?  
Unterprogramme**



- Durch Methoden wird ausführbarer Code 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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 138

### Methoden

**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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 139

### Methoden Beispiel

```
public class EchoWords {  
    public static String echo(String word1,  
                             String word2,  
                             String word3)  
    {  
        return word1 + word2 + word3;  
    }  
    public static void main(String[] args)  
    {  
        System.out.println(echo("Hello", " ", "World"));  
    }  
}
```

(Zugriffs-)modifier    Datentyp der Rückgabe    Methodenname    Aufrufparameter

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 140

### Methoden Beispiel

```
public class EchoWords {  
    public static String echo(String word1,  
                             String word2,  
                             String word3)  
    {  
        return word1 + word2 + word3;  
    }  
    public static void main(String[] args)  
    {  
        System.out.println(echo("Hello", " ", "World"));  
    }  
}
```

Zugriffsmodifier    Datentyp der Rückgabe    Methodenname    Aufrufparameter

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 141

### Methoden Beispiel

```
public class EchoWords {  
    public static String echo(String word1,  
                             String word2,  
                             String word3)  
    {  
        return word1 + word2 + word3;  
    }  
    public static void main(String[] args)  
    {  
        System.out.println(echo("Hello", " ", "World"));  
    }  
}
```

(Zugriffs-)modifier    Datentyp der Rückgabe    Methodenname    Aufrufparameter

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 142

### Methoden Beispiel

```
public class EchoWords {  
    public static String echo(String word1,  
                             String word2,  
                             String word3)  
    {  
        return word1 + word2 + word3;  
    }  
    public static void main(String[] args)  
    {  
        System.out.println(echo("Hello", " ", "World"));  
    }  
}
```

(Zugriffs-)modifier    Datentyp der Rückgabe    Methodenname    Aufrufparameter

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 143

### Methoden Beispiel

```
public class EchoWords {  
    public static String echo(String word1,  
                             String word2,  
                             String word3)  
    {  
        return word1 + word2 + word3;  
    }  
    public static void main(String[] args)  
    {  
        System.out.println(echo("Hello", " ", "World"));  
    }  
}
```

(Zugriffs-)modifier    Datentyp der Rückgabe    Methodenname    Aufrufparameter

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 144

### Methoden

#### Variable Parameterlisten

```

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 145

### Worum geht es nun?

Methoden definieren

Methodenauf  
und  
Methodenrückgabe

Besonderheiten  
(z.B. Methoden  
überladen)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 146

### Methodenaufwurf und -rückgabe

```

public static int mult (int a, int b)
{
    return a * b;
}

int x = 3; int y = 2;
System.out.println(mult(x, y));
    
```

Bei einem Methodenaufwurf 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 Methodenaufwurfs im aufrufenden Ausdruck verwendet wird.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 147

### Worum geht es nun?

Methoden definieren

Methodenaufwurf  
und  
Methodenrückgabe

Besonderheiten  
(Methoden  
überladen, CBR,  
CBV)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 148

### Überladen von Methoden

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 **Rückgabtyp** der Methode

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 149

### Miniübung:

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 150

### Methoden Call by reference vs. Call by value

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

Call by reference	Call by value
<ul style="list-style-type: none"> <li>Es wird ein Zeiger auf eine Variable übergeben.</li> <li>Innerhalb der Routine wird über den Zeiger auf der Variable außerhalb der Routine gearbeitet.</li> <li>Der <b>Inhalt der Variable</b> außerhalb der Routine <b>wird verändert</b>.</li> </ul>	<ul style="list-style-type: none"> <li>Der Wert einer Variable wird in die Parametervariable kopiert.</li> <li>Innerhalb der Routine wird auf der Kopie gearbeitet.</li> <li>Der <b>Inhalt der Variable</b> außerhalb der Routine <b>wird nicht verändert</b>.</li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 151

### Methoden Call by value in JAVA (I)

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 152

### Methoden Call by value in JAVA (II)

In JAVA gibt es zwei Arten von Datentypen (Variablen)

Primitive Datentypen	Referenztypen
<ul style="list-style-type: none"> <li>Logische (boolean)</li> <li>Zeichentyp (char, String)</li> <li>Ganzzahlen (byte, short, int, long)</li> <li>Fließkommazahlen (float, double)</li> <li>Strings</li> <li>Hier wird der Inhalt in der Variable als Wert gespeichert</li> </ul>	<ul style="list-style-type: none"> <li>Alle anderen Variablen sind Referenztypen und beinhalten lediglich Verweise auf die eigentlichen Inhalte irgendwo im Hauptspeicher</li> <li>Dies gilt z.B.:                             <ul style="list-style-type: none"> <li>Arrays (siehe Unit 3)</li> <li>alle Klassen (siehe Unit 3)</li> </ul> </li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 153

### Methoden Call by value in JAVA (III)

Dies bedeutet für JAVA

Call by value mit primitiven Datentypen	Call by value mit Referenztypen
<ul style="list-style-type: none"> <li>Logische Typen (bool)</li> <li>Zeichentypen (char)</li> <li>Ganzzahlen (int, etc)</li> <li>Fließkommazahlen (float, etc)</li> <li>Zeichenketten (Strings)</li> <li>funktioniert in der <b>Call by value</b> Semantik</li> </ul>	<ul style="list-style-type: none"> <li>Arrays</li> <li>Klassen</li> <li>funktioniert dennoch in der <b>Call by reference</b> Semantik</li> </ul>

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 154

### Methoden Call by value Beispiele (I)

**Methodenaufwurf 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
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 155

### Methoden Call by value Beispiele (II)

**Methodenaufwurf 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
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 156

### Methoden Call by value Beispiele (III)

  
University of Applied Sciences

**Methodenaufwurf 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
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **151**

### Methoden Call by value Beispiele (IV)

  
University of Applied Sciences

**Methodenaufwurf 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
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **158**

### Methoden Call by value mittels der clone-Methode

  
University of Applied Sciences

**clone()**

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



Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **159**

### Methoden Call by value Beispiele (V)

  
University of Applied Sciences

**Methodenaufwurf 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.

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **160**

### Überblick über die Eigenschaften von Datentypen in JAVA

  
University of Applied Sciences

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

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **161**

### Miniübung:






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);
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme **162**

**Miniübung:**    

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;  
}
```

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 163

**Zusammenfassung**  

- **Methodendefinition**
  - Zugriffsmodifikator
  - Rückgabetyt
  - Name
  - Parameter (inkl. variabler Anzahl an Parametern)
- **Methodenaufwurf und -rückgabe**
- **Besonderheiten bei Methoden**
  - Überladen von Methoden
  - Call by Value/Call by Reference
  - Primitive Datentypen als Parameter
  - Referenzdatentypen als Parameter

Prof. Dr. rer. nat. Nane Kratzke  
Praktische Informatik und betriebliche Informationssysteme 164