Barry Linnert
Institut für Informatik

Freie Universität Berlin

Programmieren

Barry Linnert

Sommersemester 2020

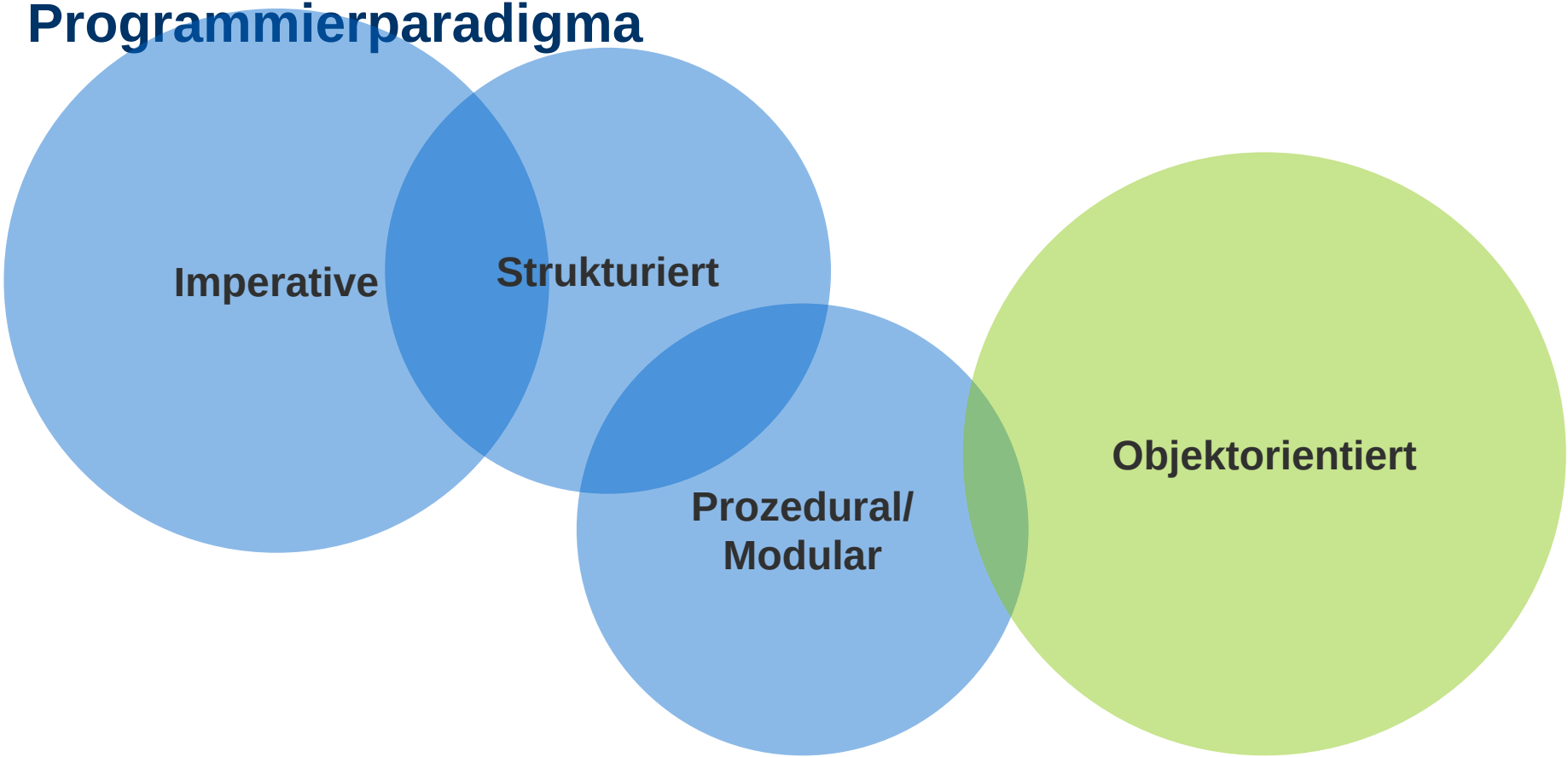
Gliederung der heutigen Vorlesung

- Kurze Wiederholung
- Imperative Bestandteile
  - Variablen, insbesondere Datentypen und Typsystem
  - Methoden, insbesondere Operatoren und Anweisungen zur Ablaufsteuerung
- Zusammenfassung

Einführung in die Objektorientierte Programmierung

WIEDERHOLUNG

Programmierparadigma



Ebenen der Objektorientierten Denkweise

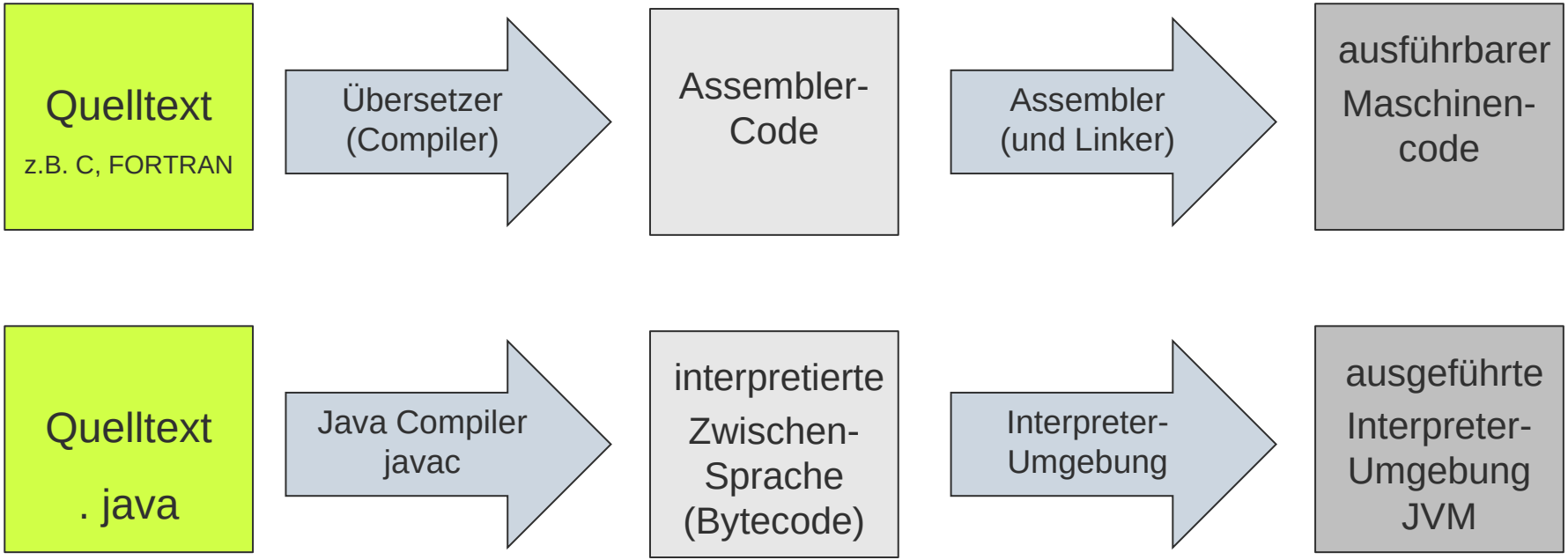
Höhere Ebene

“Arbeiten” mit Objekten

Niedrigere Ebene

Definition der benötigten Objekte (Klassen) mit imperativen Elementen

Ablauf der Programmausführung



Beispiel für ein erstes Java-Programm

```
/* Ein einfaches aber vollstaendiges Java-Programm */
public class MyFirstProgram {

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

}// end of class MyFirstProgram
```

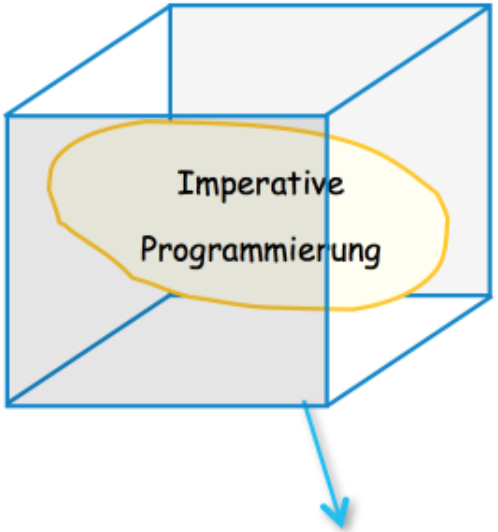
Einführung in die Objektorientierte Programmierung (Teil 2)

IMPERATIVE BESTANDTEILE

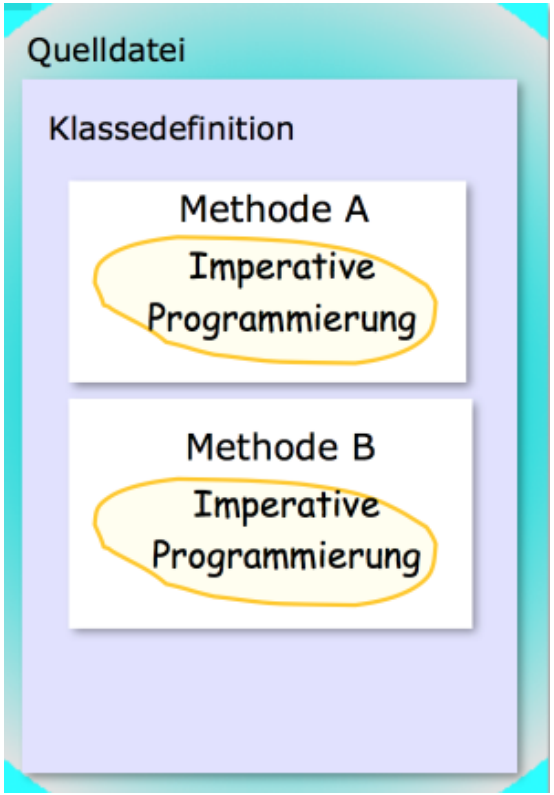
Erlernen von Objektorientierten Konzepten

- Eigentlich ist egal welche Sprache verwendet wird.
- Die wichtigsten Konzepte und Algorithmen sind sprachunabhängig.
- Java ist keine perfekte Sprache und sicher auch nicht die Beste.
- Es kommen mit Sicherheit noch bessere Programmiersprachen.

Imperative Grundbestandteile von Java



Der imperative Bestandteil eines Java-Programms befindet sich innerhalb der Methoden.



Klasse-Definition

Attribute:

- *Eigenschaft$_1$*
- *Eigenschaft$_2$*

. . . .

Verhalten:

- *Methode$_1$*
- *Methode$_2$*
- *Methode$_3$*

. . . .

Beispiel: Katze-Klasse

Attribute :

- Name
- Besitzer
- Farbe
- hungrig

. . . . .

Verhalten:

- isst
- läuft
- kratzt
- schläft

. . .

Modellierung von Rechtecken

Eigenschaften

Operationen

Berechne Fläche
Berechne Umfang
Verkleinern
Vergrößern
Verschieben
Kopieren

Modellierung von Rechteckobjekten

```
public class Rectangle {

    // Eigenschaften
        ...
        ...

    // Operationen
        ...
        ...

}
```

Dateiname:
Rectangle.java

```
public class Rectangle {

    // Attribute
    int x;
    int y;
    int width;
    int height;


    // Methoden
    public int perimeter() {
    return 2*(width + height);
    }

    public int area() {
    return (width * height);
    }

} // end of class Rectangle
```

Variablen-deklaration

Eigenschaften

Operationen

Berechne Fläche
Berechne Umfang

Einführung in die Objektorientierte Programmierung

VARIABLEN

Variablen

Imperative Programmiersprachen

- Variablen sind Stellen im Speicher, in denen Werte abgelegt werden können. Variablen speichern Zustände.

Variablen müssen normalerweise vor der erstmaligen Benutzung deklariert werden.

Funktionale Programmiersprachen

- Eine Variable stellt nur den symbolischen Namen von einem Wert oder einem Ausdruck dar, der zu einem Wert ausgewertet werden kann.

Der Wert einer Variablen kann nicht verändert werden.

Teile einer Variablen

Variablen haben:

- Typ
- Namen
- Wert
- Adresse

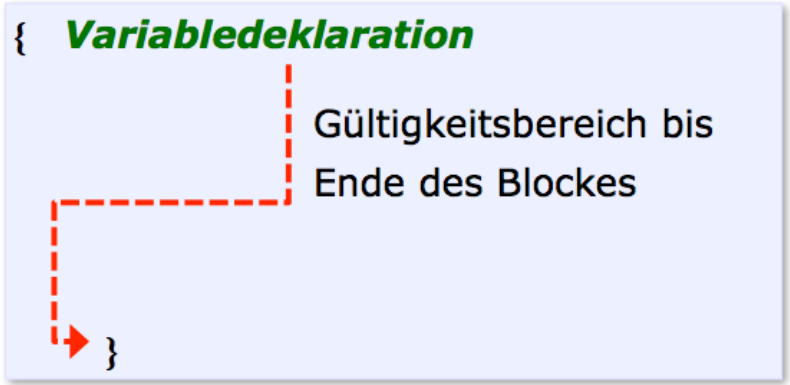
Typ, Namen, Wert: Im Programm sichtbar

Adresse: Stelle im Speicher, an welcher der Wert der Variablen während der Ausführung des Programms abgelegt ist bzw. wird.

Variablen in Java

- ***Variablen*** können überall deklariert werden, aber nicht außerhalb von Klassendefinitionen.

- Ihr Gültigkeitsbereich erstreckt sich von der Stelle ihrer Deklaration bis zum Ende des *Blocks*, in dem sie deklariert wurden.



Variablennamen

- Namen von Variablen haben in Java einige Einschränkungen
  - Namen beginnen mit einem Buchstaben — area, a321
  - oder einem ’_’ — _time, _100
  - enthalten keine Sonderzeichen — +, =, & , ...
  - enthalten kein reserviertes Java-Wort — int, double, main

- und folgen der Konvention
  - Variablennamen fangen klein an — kreis
  - mit Großbuchstaben dazwischen — kreisUmfang
  - oder mit Unterstrichen — kreis_volumen

Einfache Zuweisungen in Java

- Die einfachsten Befehle in jeder Programmiersprache sind die Zuweisungen.

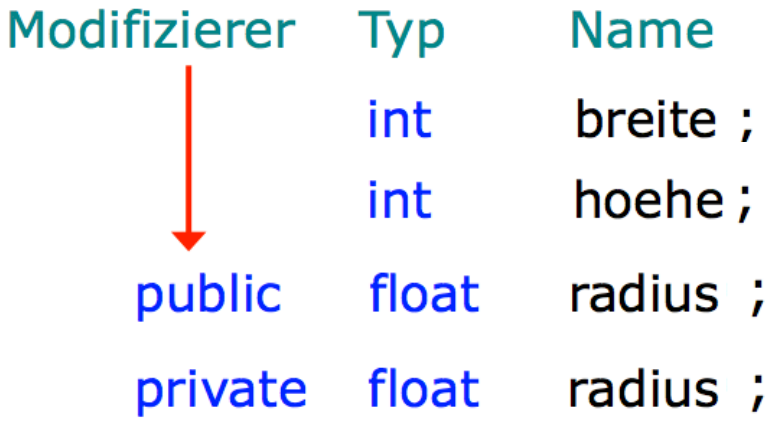
Beispiele:

```
b = 3 * b ;
c = a / b ;
```

Semikolon

Deklaration von Variablen

- In Java müssen alle Variablen vor der ersten Anwendung deklariert werden.



- Der Modifizierer bestimmt die Zugriffsrechte, die andere Objekte auf eine Variable (Attribut) haben.

Deklaration von Variablen (2)

- Bestimmte Variablen sollten für alle Instanzen der Klasse immer den gleichen Wert haben, z.B.

```
float pi = 3.14;
```

- Festlegung durch das vorangestellten Attribut *static*, z.B.

```
static float pi = 3.14;
```

Die Java Math-Klasse

```
static double  sin (double a)
static double  cos (double a)
static double  tan (double a)
```

```
static double  log (double a)
static double  pow (double a, double b)
static double  exp (double a)
static double  sqrt (double a)
```

```
static double  random ()
static long    round (double a)
static double  ceil (double a)
static double  floor (double a)
```

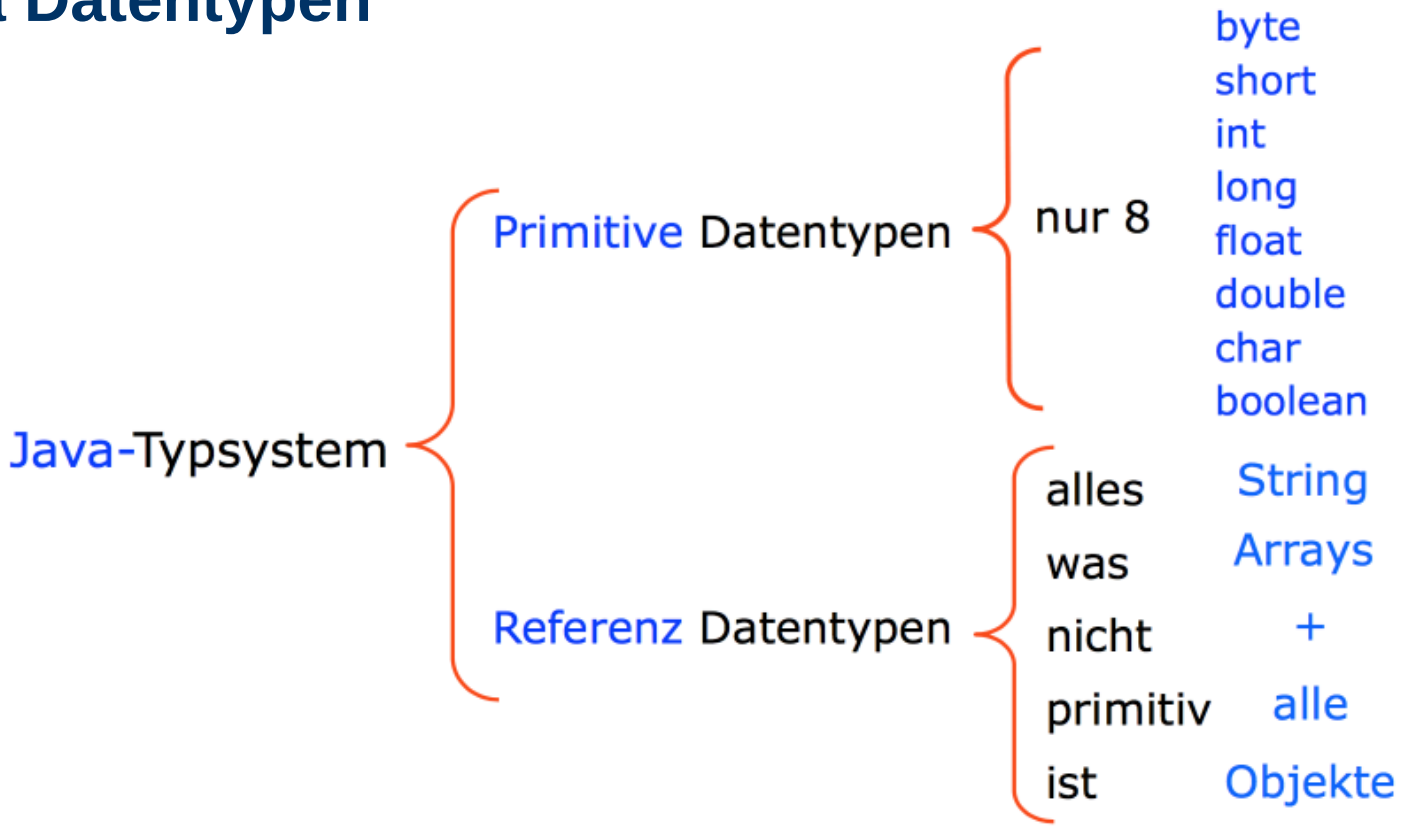
Anwendungsbeispiel:

```
double x,y;
  . . . ..
double zufallszahl = Math.random();
  . . . .
x = Math.sin (y);
  . . . . .
```

Einführung in die Objektorientierte Programmierung (Teil 2)

DATENTYPEN UND TYPSYSTEM

Java Datentypen



Primitive Datentypen in Java

- Java hat **8** elementare Datentypen und legt für jeden primitiven Datentyp eine **feste Speichergröße** und damit einen **festen Zahlenbereich** fest.

Typ	Bits	Zahlenbereich	Standardwert
byte	8	-128 . . . 127	0
short	16	-32.768 . . 32.767	0
int	32	-2^{31} . . . $2^{31}-1$	0
long	64	-2^{63} . . . $2^{63}-1$	0
float	32	ca. $3.402*10^{38}$ . . $1.402*10^{-45}$	0.0
double	64	ca. $1.798*10^{308}$ . . $4.94*10^{-324}$	0.0
boolean	1	true, false	false
char	16	Unicode	\u0000

Java-Typsystem

- Variablentyp muss vor der ersten Benutzung festgelegt werden.
- Der Typ kann sich nicht ändern, solange die Variable gültig ist.
- Der Compiler muss in der Lage sein, den Typ jedes Ausdrucks zu überprüfen, insbesondere welche Variable an welcher Stelle im Programm welchen Typ hat.
- Bei jeder Methodendefinition muss zusätzlich festgelegt werden, welche Typen die Parameter haben müssen und von welchem Typ das Ergebnis sein soll.

```
public class Rectangle {

    // Attribute
    int x;
    int y;
    int width;
    int height;

    // Methoden
    public int perimeter() {
    return 2*(width + height);
    }

    public int area() {
    return (width * height);
    }

} // end of class Rectangle
```

Variablen-deklaration

Methoden

Eigenschaften

Operationen

Berechne Fläche
Berechne Umfang

Freie Universität Berlin

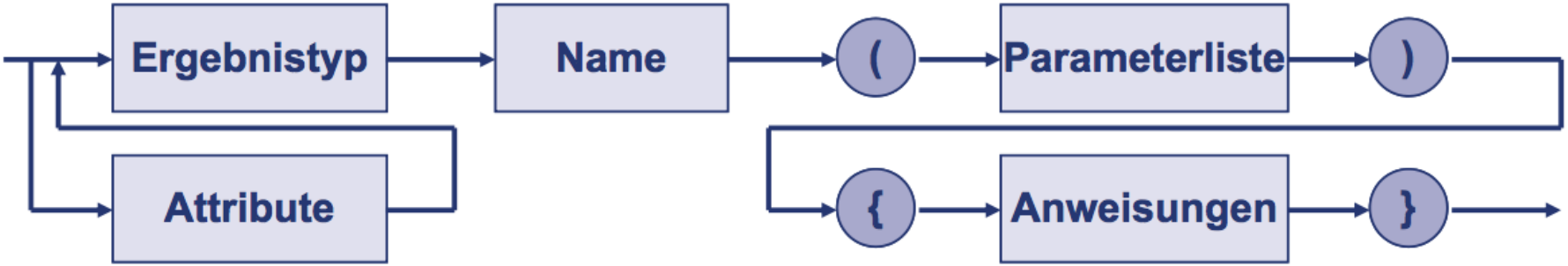
Einführung in die Objektorientierte Programmierung (Teil 2)

METHODEN

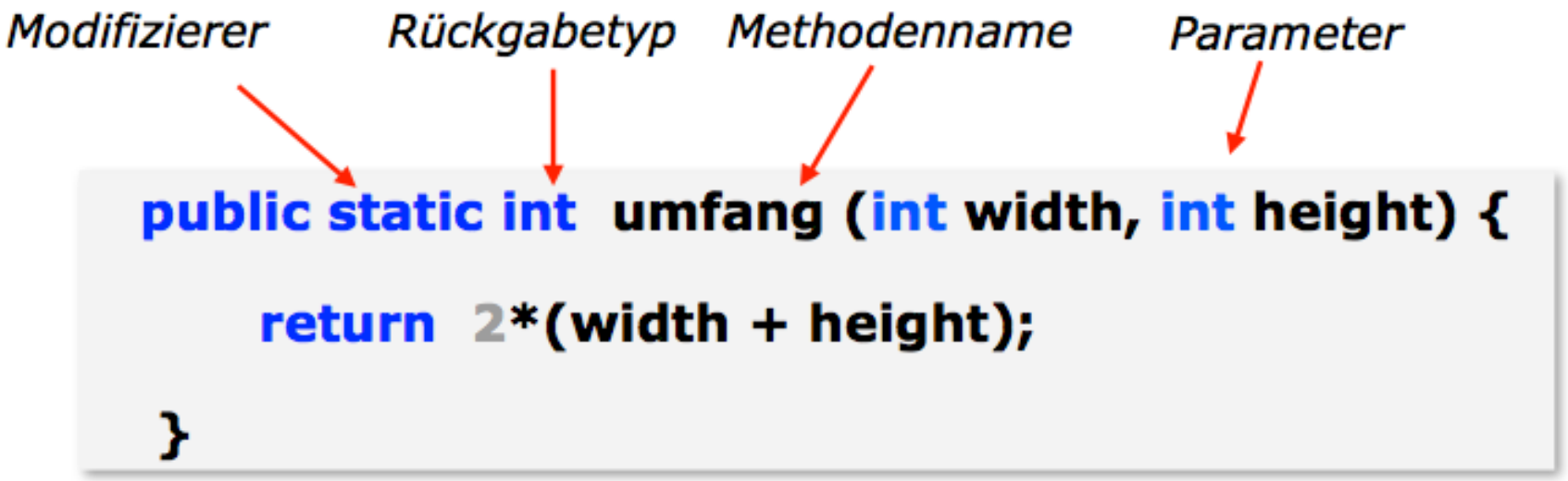
Einführung Klassenmethoden

- **Methoden** enthalten den ablauffähigen Programmcode. Es gibt keine **Methoden** außerhalb von Klassen. **Methoden** dürfen nicht geschachtelt werden.

- Die Syntax von Methoden



Beispiel


Modifizierer
Rückgabetyp
Methodenname
Parameter
public static int umfang (int width, int height) {
return 2*(width + height);
}

Zugriffskontrolle auf Methoden

- Der Zugriff auf Methoden kann genauso wie bei Variablen durch ***Modifizierer*** gesteuert werden:

  - **public**: überall zugänglich.
  - **private**: nur innerhalb der eigenen Klasse zugänglich.
  - **protected**: in anderen Klassen desselben Packages und in Unterklassen zugänglich.

- Methoden ohne **Modifiziere**r sind nur im selben ***Paket*** (package) zugänglich.
  (Eine Gruppe von Klassen mit starken inhaltlichen Bezügen kann zu einem Paket (*package*) zusammengefasst werden. Wir schauen uns das in der nächsten Veranstaltung an.)

Return-Anweisung

- Die return-Anweisung beendet frühzeitig die Ausführung einer Methode.
- In einer Methode kann es mehrere return-Anweisungen geben.
- Methoden, die als Funktionen definiert sind, d.h. ein Ergebnis liefern, müssen durch eine return-Anweisung dieses Ergebnis zurückgeben.
- Wenn eine Methode kein Ergebnis zurückgibt, wird das Schlüsselwort **void** als Rückgabetyp verwendet, und eine return- Anweisung ist nicht erforderlich.

Imperative Programmierung in Methoden

Operatoren

- Arithmetische Operatoren
- Vergleichsoperatoren
- Zuweisungsoperatoren
- Logische Operatoren

Anweisungen

- Einfache Anweisung
- Alternativanweisung
- Switch-Anweisung
- Schleifen
- For-Anweisung

Einführung in die Objektorientierte Programmierung

OPERATOREN

Arithmetische Operatoren

unär

Operator	Benutzung	Beschreibung
-	- x	Negatives Vorzeichen

binär

Operator	Benutzung	Beschreibung
*	a * b	Multiplikation
/	3 / 5	Division
%	6 % 5	Divisionsrest
+	10 + 3	Addition
-	10 - 3	Subtraktion

Vergleichsoperatoren

binär

Operator	Benutzung	Beschreibung
<	op1 < op2	Kleiner
>	op1 > op2	Größer
<=	op1 <= op2	Kleiner oder gleich
>=	op1 >= op2	Größer oder gleich
==	op1 == op2	Gleichheit
!=	op1 != op2	Ungleichheit

Häufige Fehlerquelle

- Der Vergleichsoperator ist „==“ und nicht etwa „=“

- Beispiele:

```
if(x=0){ ... }
```

Exception

vs.

```
if(x=true){ ... }
```

Zuweisungsoperatoren

- Zuweisungsoperatoren sind alle von der Form op=, wobei op ein Operator ist:

```
+= -= *= /= &= |= ^= %= < <= > >= > > >=
```

- Beispiel

```
x = y = x+5;
```

Inkrement- und Dekrement-Operatoren

Operator	Benutzung	Beschreibung
++	++ op	Inkrementiert op um 1
++	op ++	Inkrementiert op um 1
--	-- op	dekrememtiert op um 1
--	op --	dekrememtiert op um 1

Beispiele

Nutze zuerst den Wert von b innerhalb der Anweisung und dann erhöhe um 1

```
x = a-- + b++ - ++c;
```

Nutze zuerst den Wert von a innerhalb der Anweisung und dann reduziere um 1

Erhöhe zuerst den Wert von c um 1 und dann nutze c in der Anweisung

Beispiele

```
// Beispiel 1
int i=10, j=0;
j= ++i;
j= i++;

//Beispiel 2
int a=5;
int b=4;
int c=0;
int d=0;
c = --a + b++;
c = c + ++a;
d = c++ - ++a + b--;
```

Wert der Variablen	
j: 11	i: 11
j: 11	i: 12

Wert der Variablen			
a: 4	b: 5	c: 8	
a: 5	b: 5	c: 13	
a: 6	b: 4	c: 14	d: 12

Logische Operatoren

Operator	Benutzung	Beschreibung
!	!a	logische Negation
&	a & b	boolesches logisches AND
|	a | b	boolesches logisches OR
&&	a && b	logisches AND (verkürzt)
||	a || b	logisches OR (verkürzt)

Verkürzte Auswertung von Ausdrücken

- **AND**
  - & -> wertet beide Operanden aus
  - && -> beendet die Auswertung, wenn der erste Operand FALSE wiedergibt und das Ergebnis ist dann FALSE
  - Beispiel

```
(x != 0) & (1/x > 1)
```

Exception in thread "main" java.lang.ArithmeticException: / by zero

```
(x != 0) && (1/x > 1)
```

- **OR**
  - exprA | exprB <-- bewerte exprA, dann bewerte exprB und dann führe | aus
  - exprA || exprB <-- bewerte exprA und nur wenn es FALSE wiedergibt, bewerte exprB und führe danach || aus

Bedingte Ausdrücke

- Werden mit dem Operator „ **?:** “gebildet, dem einzigen Operator mit drei Operanden:

```
<op1> ? <op2> : <op3>
```

- *Operand 1:* Ergebnis vom Typ *boolean*
- *Operand 2:* Wenn Operand 1 *TRUE* ist, dann ist das Ergebnis des bedingten Ausdruckes gleich dem Wert des zweiten Operanden.
- *Operand 3:* Wenn Operand 1 FALSE ist, dann ist das Ergebnis des bedingten Ausdruckes gleich dem Wert des dritten Operanden.

Einführung in die Objektorientierte Programmierung

ANWEISUNGEN ZUR ABLAUFSTEUERUNG

Anweisungen zur Ablaufsteuerung

- Einzelne Anweisungen (*statements*) werden mit einem Semikolon abgeschlossen

```
a = b * c;
```

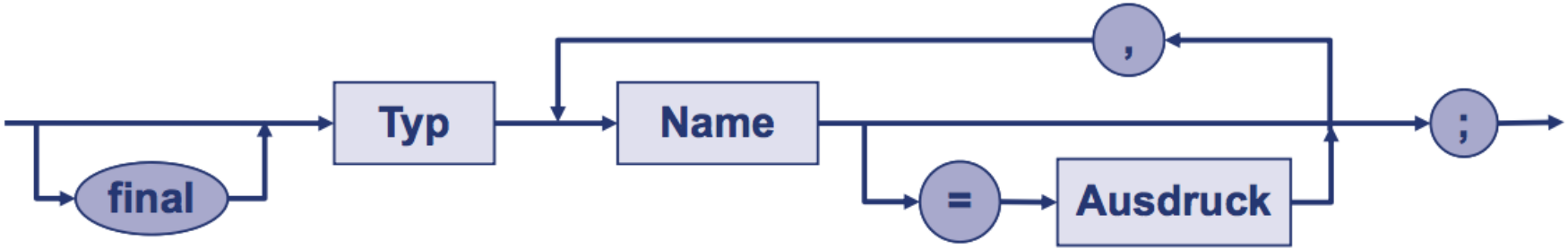
- Anweisungen werden durch geschweifte Klammern zu *Blöcken* zusammengefasst

```
{
  a = b * r;
  x = a + z/a;
}
```

- Die Abarbeitungsreihenfolge der Anweisungen verläuft normalerweise von oben nach unten und von links nach rechts.

Einfache Anweisung: Variablendeklaration

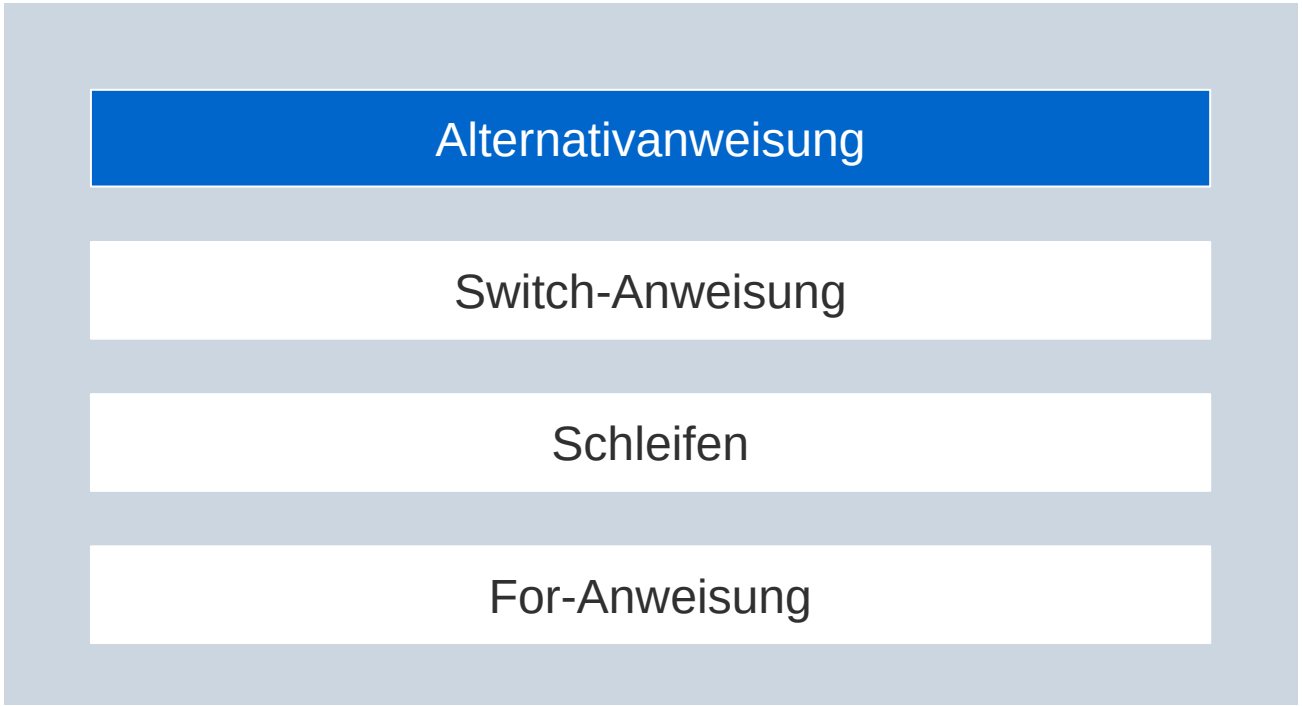
- Syntax der Variablendeklaration



Anweisungen zur Ablaufsteuerung

- Alternativanweisung
- Switch-Anweisung
- Schleifen
- For-Anweisung

Anweisungen zur Ablaufsteuerung



Alternativ-Anweisung

- Formen:

```
if (Bedingung)
        { <Anweisung> }
```

```
if (Bedingung)
        { <Anweisung> }
else
        { Anweisungen }
```

- Syntax:

Beispiel if-else-Anweisung

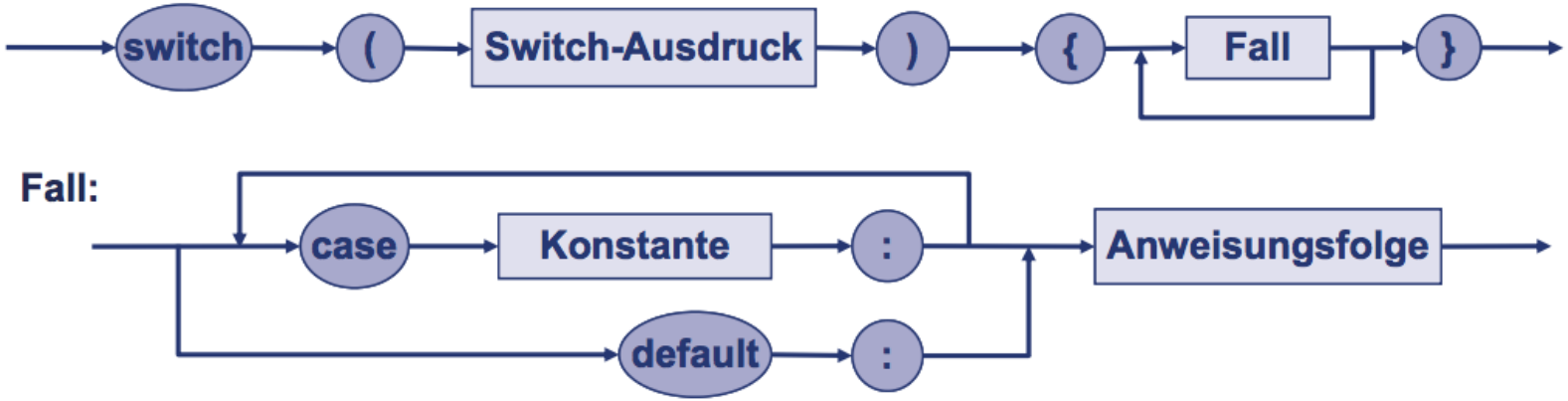
```
...
if( punkte >= 90 ){
        note = 1;}
else if ( punkte >= 80 ){
        note = 2;}
else if ( punkte >= 70 ){
        note = 3;}
else if ( punkte >= 60 ){
        note = 4;}
else {
        note = 5;}
...
```

Anweisungen zur Ablaufsteuerung

- Alternativanweisung
- **Switch-Anweisung**
- Schleifen
- For-Anweisung

Switch-Anweisung

- Syntax



- Form

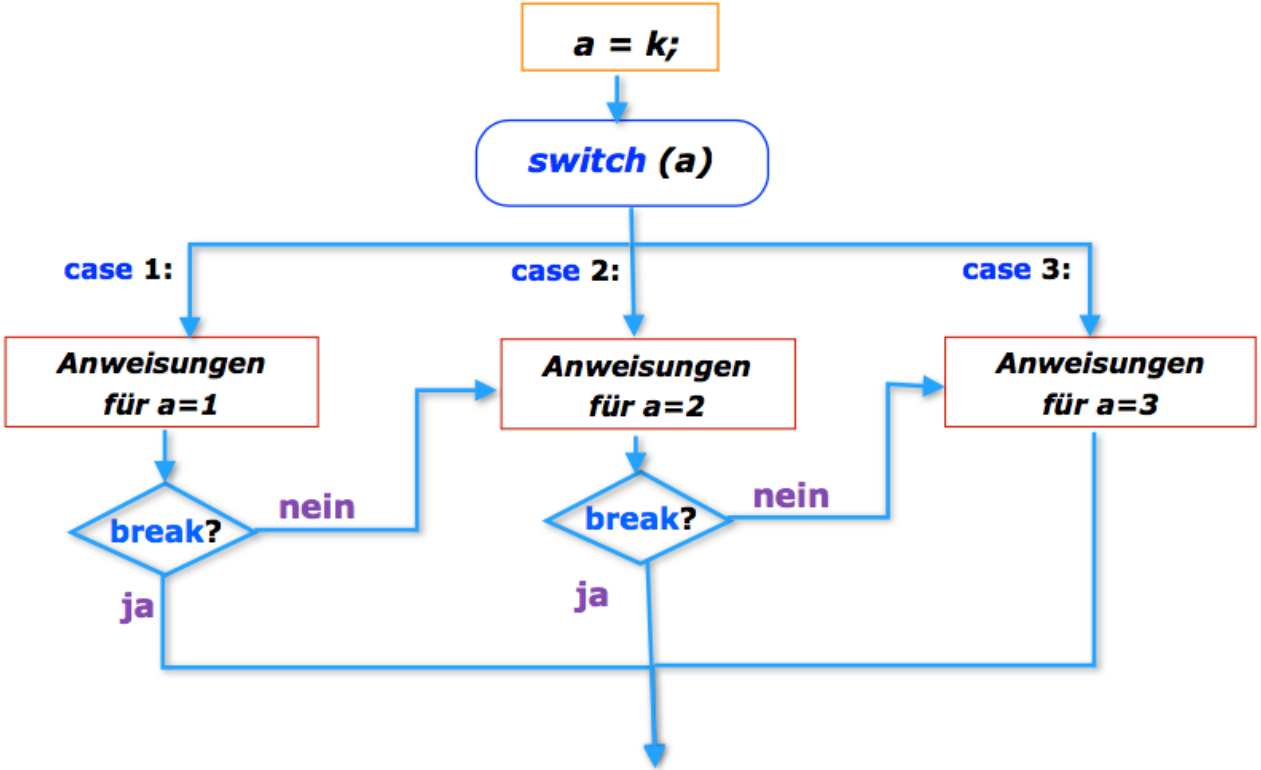
```
switch ( Ausdruck ) {
case Konstante1: Anweisungen;
case Konstante2: Anweisungen;
. . . . . usw.
default : Anweisungen }
```

switch-Anweisung

```
switch (Ausdruck) {
        case Konstante1: {
                Anweisung1
                Anweisung2
                …
                }
                break;
        case Konstante2: {
                Anweisung3
                }
                break;
        . . . . . usw.
        default : Anweisungen
}
```

u.a. short int long char

Kontrollfluss der switch-Anweisung



Beispiel: Erweiterung der Klasse Datum

```
static int tageProMonat(int j, String m){
  switch (m) {
   case "Januar": case "März": case "Mai": case " Juli ”:
   case "August" : case "Oktober" : case "Dezember" : return 31;
   case "Februar": return (schaltjahr(j))? 29 : 28;
   default : return 30;
   }
}
```

Anweisungen zur Ablaufsteuerung

- Alternativanweisung
- Switch-Anweisung
- **Schleifen**
- For-Anweisung

while-Anweisung

- Schleifenbedingung wird vor jeder Iteration gestestet.

- Syntax

- Form

```
while (Bedingung) {
    <Anweisungen>
}
```

Do-while-Anweisung

- Schleifenbedingung wird nach jeder Iteration getestet.

- Syntax

- Form

```
do {
    <Anweisungen>
   }
while (Bedingung);
```

Beispiel

```
import javax.swing.JOptionPane;
public class HelloWorld {

   public static void main(String[] args) {
      String input ;
      do
         input = JOptionPane.showInputDialog("Nenne mir den
                 Geheimcode bitte!");
      while ( ! input.equals ( "Geheim" ) ) ;
      System.out.println("Willkommen, Meister!");
   }
}
```

Anweisungen zur Ablaufsteuerung

- Alternativanweisung
- Switch-Anweisung
- Schleifen
- **For-Anweisung**

For-Anweisung

- **for**-Schleifen verwenden wir, wenn die Einschränkungen der Schleife (*Initialisierung, Bedingung und Inkrementierung*) bekannt sind.

  1. Die ***Initialisierung*** wird einmal zu Beginn ausgeführt.
  2. Der Schleifenrumpf wird ausgeführt, solange die ***Bedingung*** erfüllt ist.
  3. Nach jedem Durchlauf des Rumpfes wird die Anweisung im ***Inkrement*** einmal ausgeführt und die ***Bedingung*** erneut geprüft.

- Form

```
for ( Initialisierung ; Bedingung ; Inkrement ) {
        Anweisungen
}
```

For-Anweisung

Die Initialisierung wird einmal zu Beginn ausgeführt.

Vor jedem Durchlauf des Rumpfes wird die Bedingung geprüft.

Nach jedem Durchlauf des Rumpfes wird die Anweisung im Inkrement einmal ausgeführt.

```
...
for (int i=0; i<=100; i++ ) {
     System.out.println( i*i );
}
...
```

For-Anweisung

Die Initialisierung kann mehrere Initialisierungen von Variablen beinhalten

Nach jedem Durchlauf können mehrere Variablen verändert werden.

```
...
for ( i=0,j=20,k=5; i<=10; i++,j--,k+=5 )
{
    System.out.println( (i+j)*k );
}
...
```

Arrays

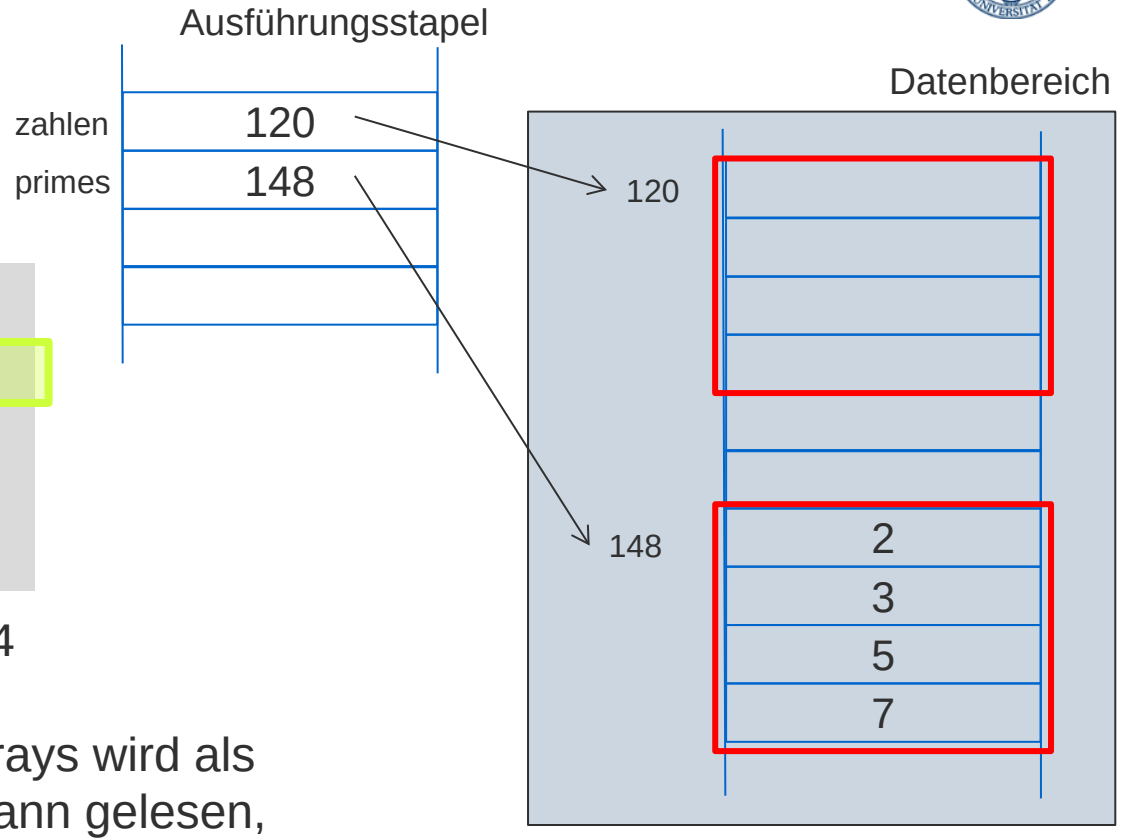
- In Java kann man eine Folge gleichartiger Objekte unter einem Namen zusammenfassen (Reihungen).

Array-Elemente:	17	-5	42	47	99	-33	42	19	-42	191
Indizes:	0	1	2	3	4	5	6	7	8	9

- Erzeugung (eindimensionaler Felder)

```
int[] int1Bsp = { 17, -5, 42, 47, 99, -33, 42, 19, -42, 191};
char[] char1Bsp = {'A', 'a', '%', '\t', '\\', '\"', '\u03a9'};
double[] double1Bsp = { 3.14, 1.42, 234.0, 1e-9d};
```

Arrays



Skript

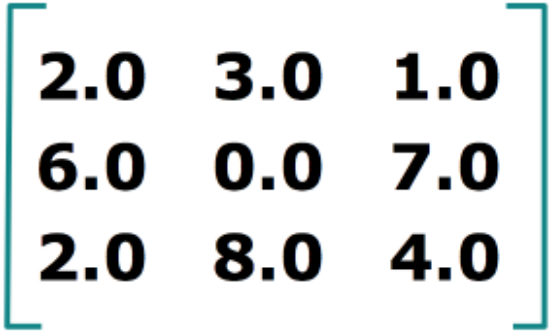
```
int[] zahlen;
zahlen = new int[4];
int[] primes = {2,3,5,7};
```

`primes.length` → `4`

Das length-Attribut des Arrays wird als Konstante generiert und kann gelesen, aber nicht verändert werden.

Mehrdimensionales Array: Array von Arrays

```
...
double[][] matrix = new double[3][3];
...
```




```
...
matrix[0][0] =  1;
...
matrix[ ][ ] =   ;
x = matrix[1][2];
...
```

Variante der for-Schleife

- **for**-Schleifen besuchen oft jede Position eines Arrays oder einer Datenstruktur, und aus diesem Grund haben die Java-Entwickler ab Java 1.5 eine abgekürzte Form der **for**-Schleife eingeführt.

```
for ( Typ Element : Datenstrukturname )
     { Anweisungen }
```

- Jedes Element der Datenstruktur wird der Variablen "Element" zugewiesen und innerhalb der Anweisungsblöcke benutzt.

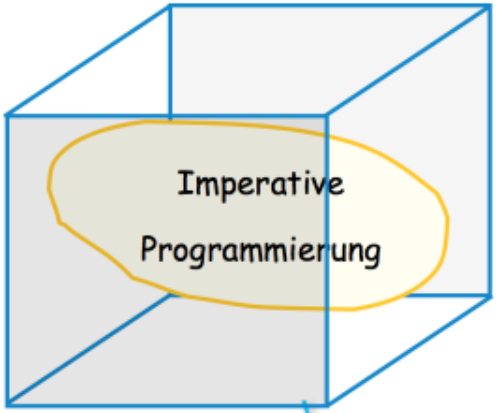
Beispiel

```
...
double[] nums = { 3.4, 5.6, 1.2, 2.3, 5.6, 7.8 };
double prod = 1;
for ( double d : nums ) {
    prod = prod * d;
}
...
```

Einführung in die Objektorientierte Programmierung

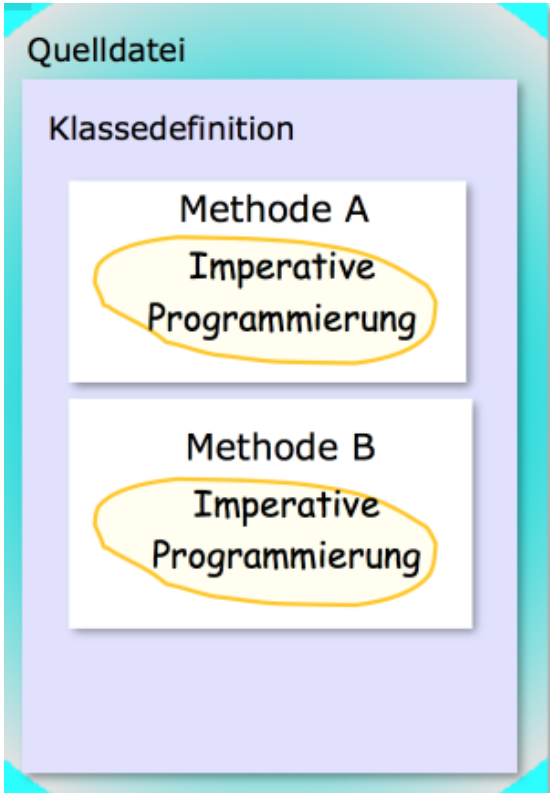
ZUSAMMENFASSUNG

Imperative Grundbestandteile von Java

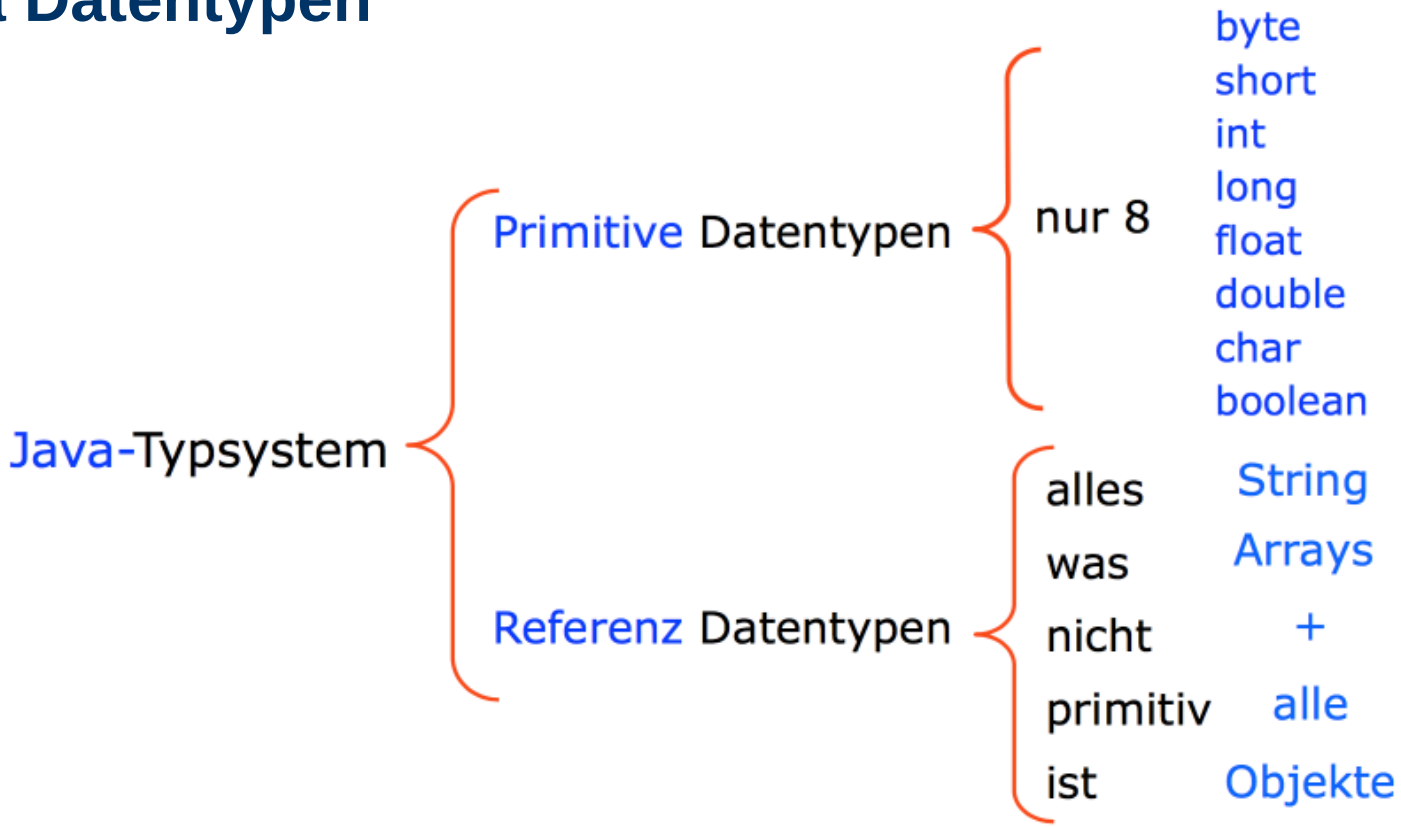


objektorientierte Verpackung

Der imperative Bestandteil eines Java-Programms befindet sich innerhalb der Methoden.



Java Datentypen



Imperative Programmierung in Methoden

Operatoren

- Arithmetische Operatoren
- Vergleichsoperatoren
- Zuweisungsoperatoren
- Logische Operatoren

Anweisungen

- Einfache Anweisung
- Alternativanweisung
- Switch-Anweisung
- Schleifen
- For-Anweisung