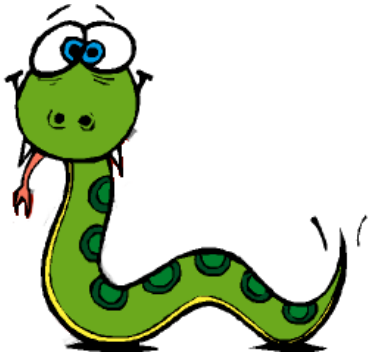
Freie Universität Berlin

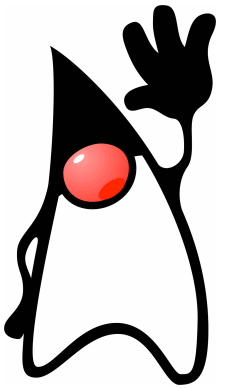
Algorithmen und Programmieren II

Objektorientiertes Programmieren

(Einführung)



{P} S {Q}



SS 2012

Prof. Dr. Margarita Esponda

Warum Objektorientierte Programmierung?

Hauptproblem bei prozeduraler Programmierung

Trennung zwischen Prozeduren (Funktionen) und Daten

- welche Daten sollen lokal und welche global existieren?
- Probleme, einen geeigneten Zugriffsschutz auf die globalen Daten zu finden
- Probleme der realen Welt sind schwer prozedural zu simulieren
- ungeeignet für große Softwaresysteme
- schlechte Wiederverwendbarkeit der Software

Erste Objektorientierte Ideen

Ole-Johan Dahl und **Kristen Nygaard**

- **1965**
- **Simula-I** Programmiersprache
  - erste objektorientierte Programmiersprache
  - speziell für diskrete ereignisorientierte Simulation.
- **Simula-67**
  - Neue Konzepte:
    - Objekte, Klassen, Subklassen, Nebenläufigkeit und *Garbage Collection*.

Erste Objektorientierte Ideen

```
Class Rectangle (Width, Height); Real Width, Height;

Begin
        Real Area, Perimeter;

        Procedure Update;
        Begin
            Area := Width * Height;
            Perimeter := 2*(Width + Height)
        End of Update;

        Boolean Procedure IsSquare;
            IsSquare := Width=Height;


        Update
        OutText("Rectangle created: ");
        OutFix(Width,2,6); OutFix(Height,2,6); OutImage
End of Rectangle;
```

Simula

Erste Objektorientierte Ideen

Ivan Sutherland's **Sketchpad**, 1963

Erstes objektbasiertes Malprogramm

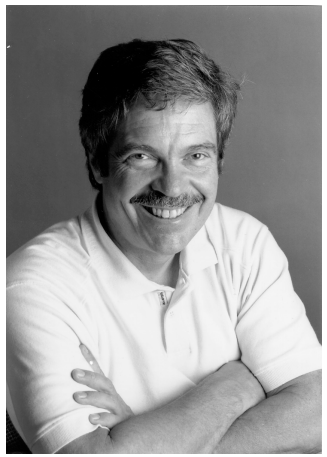
Master- und Instanz-Zeichnungen

draw a car

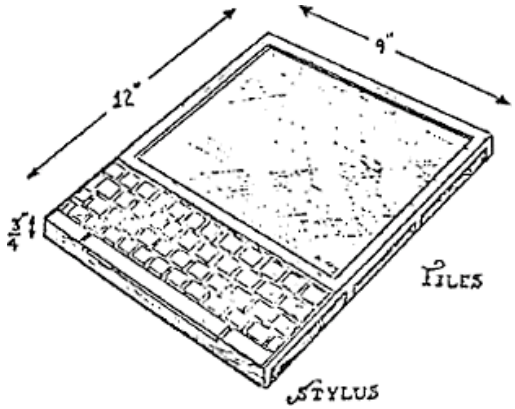
make two instances

Erste Objektorientierte Ideen

Alan Kay



- 1968. Xerox
- erste objektorientierte Konzepte für die Gestaltung von Benutzeroberflächen
- **Dynabook-**Konzept



- **Smalltalk-**Programmiersprache

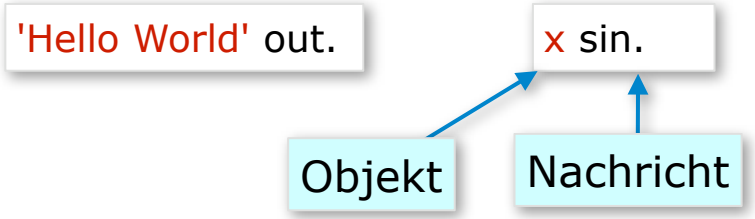
„The best way to predict the future is to invent it."

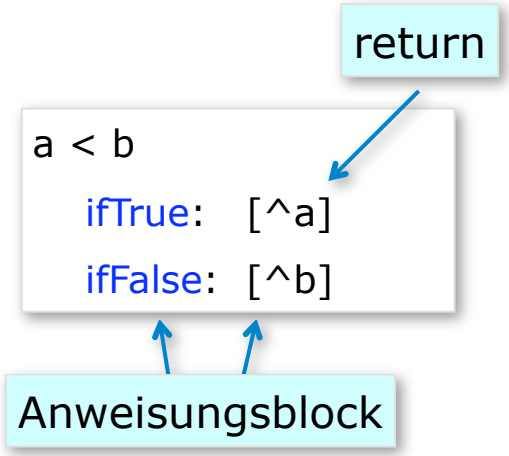
OO-Konzepte können mit anderen Programmiersprachen gelernt werden.

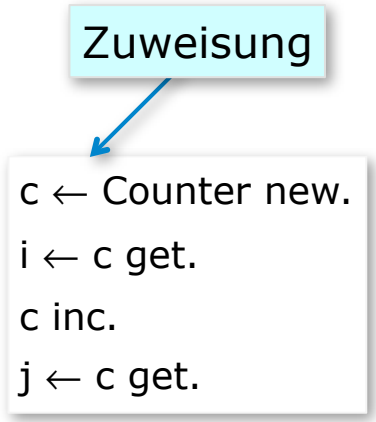
- Eigentlich ist egal welche Sprache.
- Die wichtigsten Konzepte und Algorithmen sind sprachunabhängig.
- Java ist keine perfekte Sprache und auch nicht die beste.
- Es kommen mit Sicherheit noch bessere Programmiersprachen

Smalltalk-Programmiersprache

- erste rein objektorientierte Programmiersprache
- für das Betriebssystem des *Dynabook*
- sehr einfach und elegant
  - "Alles ist ein Objekt"
- die wesentliche objektorientierte Konzepte sind da
- eigenwillige Syntax
- dynamisch typisiert







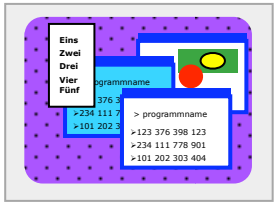
Objektorientierte Programmiersprachen

1962 - Simula I
1967 - **Simula 67**
1971 - Smalltalk -71
1980 - **Smalltalk-80**
1983 - C++
1985 - **Eiffel**
1991 - **Python**
1991 - **Java**
1993 - Ruby
1994 - CLOS
1995 - Delphi (Object Pascal)
1995 - Ada (Objektorientiert)
2001 - **C#**
2002 - COBOL (Objektorientiert)
2003 - Fortran (Objektorientiert)
2004 - PHP (Objektorientiert)
2007 - D
2008 - MATLAB

„I invented the term Object-Oriented, and I can tell you I did not have C++ in mind."
Alan Kay

Warum objektorientiertes Programmieren?

- Simulationsprobleme
  - in der Welt läuft alles parallel
- **Wiederverwendbarkeit** von Software
  - saubere Modularisierung der Software
  - mit klaren Schnittstellen
- Graphische Benutzeroberflächen
  - nicht sequentielle, interaktive Steuerung von Anwendungsprogrammen



- Programmierung verteilter Anwendungen
  - Parallelität und Kommunikation auf natürliche Weise

Was ist ein Objekt?

Ein Objekt vereint Daten und Prozeduren (Funktionen oder Methoden), die auf diesen Daten operieren, in einem Wert (Verbundvariable).

Was ist ein OO-Programm?

Ein OO-Programm ist ein System kooperierender Objekte.

Was ist ein Objekt?

Objekte sind verwandt mit:

C/C++ struct-Datentyp

```
struct  Student {
    int matrikelnummer;
    int alter;
    int semester;
};
```

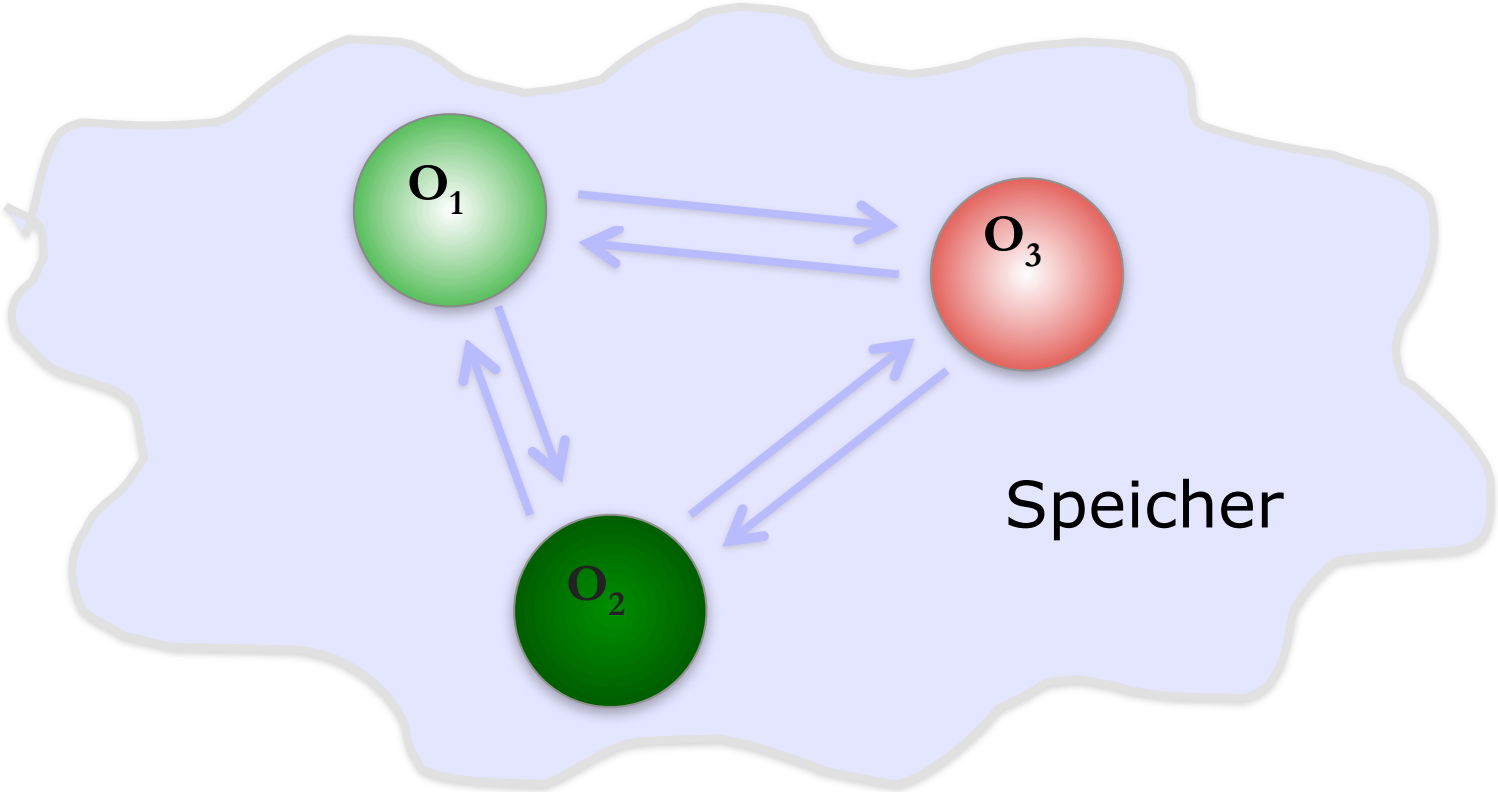
Pascal Records

```
TYPE datum = RECORD
        tag, monat, jahr : INTEGER;
END;
```

Mit dem wesentlichen Unterschied, dass Objekte immer einen zusätzlichen Zeiger auf die Tabelle der Klasse haben, nach deren Vorgabe diesen erzeugt wurden.

OO-Programmausführung

Die OOP betrachtet eine Programmausführung als ein System kooperierender Objekte.



Objektorientiertes Programmieren

Wiederverwendbarkeit

Any fool can write code that a computer can understand.

Good Programmers write code that humans can understand.

Martin Fowler in >>Refactoring<<

Was ist Java?

1991 Patrick Naughton und James Gosling

Programmierumgebung **Oak** bei Sun Microsystems

um Anwendungen für elektronische Geräte und das interaktive Fernsehen leicht zu programmieren.

1992

Green Projekt

Oak Programmierumgebung

keine Erfolg!

Was ist Java?

1996 veröffentlichte *Sun Microsystems* die erste offizielle Entwicklungsumgebung für Java.

2007 Free and Open Source Software. GNU General Public License (GPL)

Sun Microsystems → Oracle

Beispielloser Erfolg:

Noch nie hat eine neue Programmiersprache in so kurzer Zeit so starke Verbreitung gefunden.

Java ist 16 Jahre alt

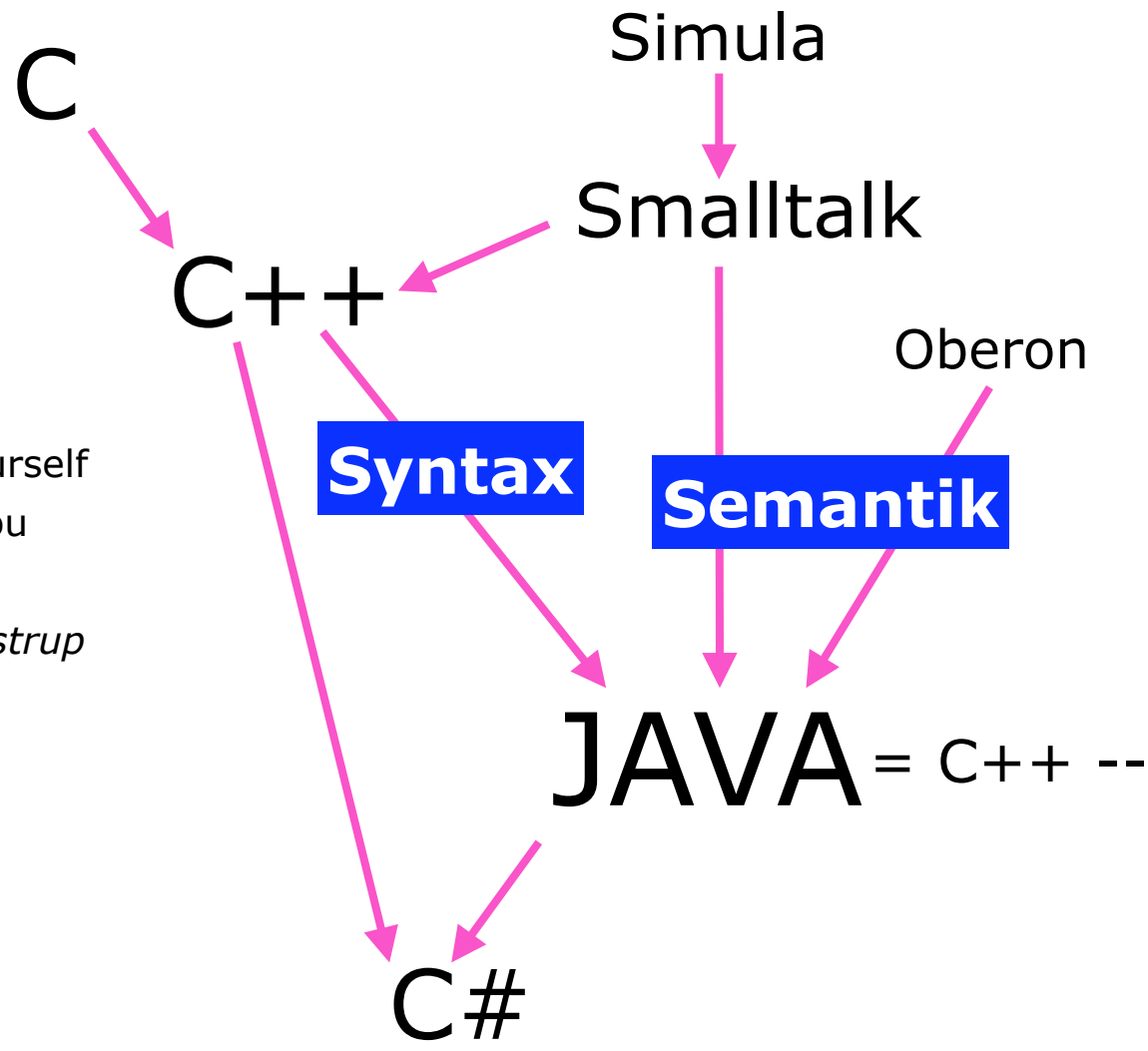
Es gibt 930 Millionen *downloads* der *Java Runtime Environment* pro Jahr

3 Billionen Mobile-Geräte haben eine JVM Java Virtuelle Maschine

Stammbaum von Java

"In C++ it's harder to shoot yourself in the foot, but when you do, you blow off your whole leg."

Bjarne Stroustrup



Objektorientiertes Programmieren

Vorgänge der realen Welt

- inhärent paralleles Ausführungsmodell

Trennung von Auftragserteilung und Auftragsdurchführung

- klar definierte Schnittstellen

Klassifikation und Vererbung

- Anpassbarkeit, Klassifikation und Spezialisierung von Programmteilen

Konzepte Objektorientierter Programmierung

- Objekte
- Klassen
- Nachrichten
- Kapselung
- Vererbung
- Polymorphismus

statisch

Was ist eine Klasse?

Eine Klasse ist ein Bauplan, um Objekte einer bestimmten Sorte zu erzeugen.

Ohne Klassen gibt es keine Objekte in Java!

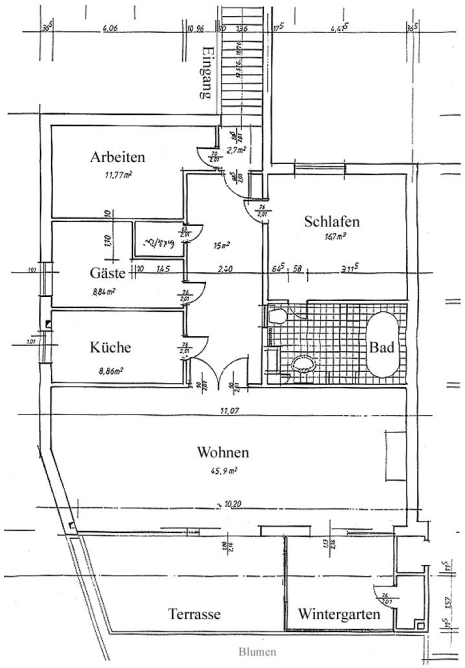
Klassen besitzen den Vorzug der Wiederverwendbarkeit.

statisch

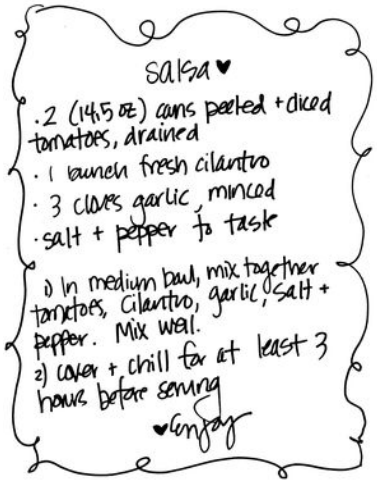
Was ist eine Klasse ?

Stempel

Bauplan



Rezept



dynamisch

Was ist ein Objekt ?

Ein Objekt ist ein Softwarebündel aus **Variablen** und mit diesen Variablen zusammenhängenden **Methoden.**

Ein Objekt ist eine konkrete Ausprägung bzw. eine Instanz einer Klasse.

Jedes Objekt „weiß", zu welcher Klasse es gehört.

dynamisch

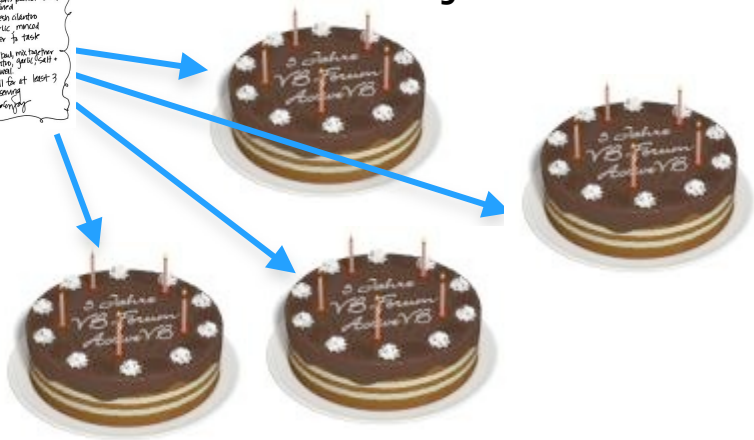
Was sind Objekte?

Klasse

Objekte

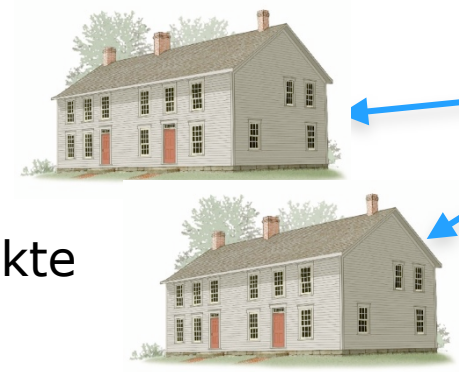
Klasse

Objekte

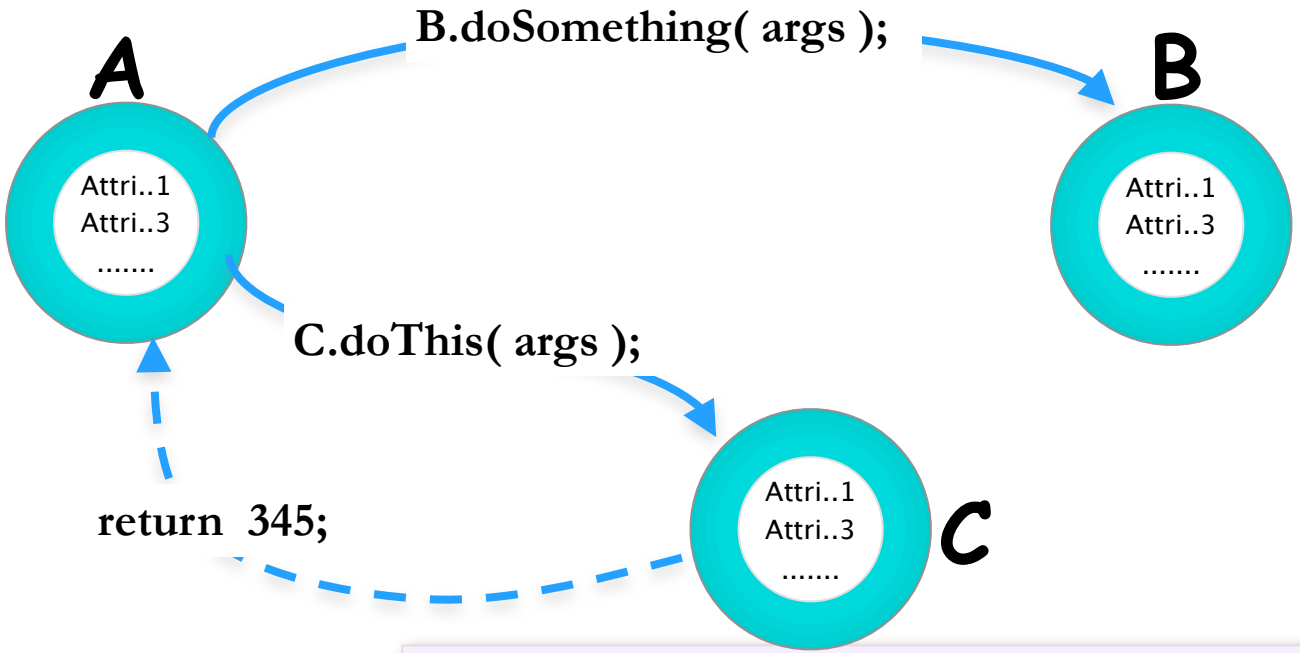


Klasse

Objekte



Was sind Nachrichten ?



Methodenaufrufe

- Empfänger
- Name der auszuführenden Methode
- Parameter

Nachrichten

```
personX.calculateSomething( a, b )
```

Bevor ich eine Nachricht zu einem Objekt schicke, muss dieses Empfängerobjekt im Speicher existieren.

Klasse-Definition

Attribute:

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

. . . .

Verhalten:

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

. . . .

Beispiel: Katze-Klasse

Attribute :

- Name
- Besitzer
- Farbe
- hungrig

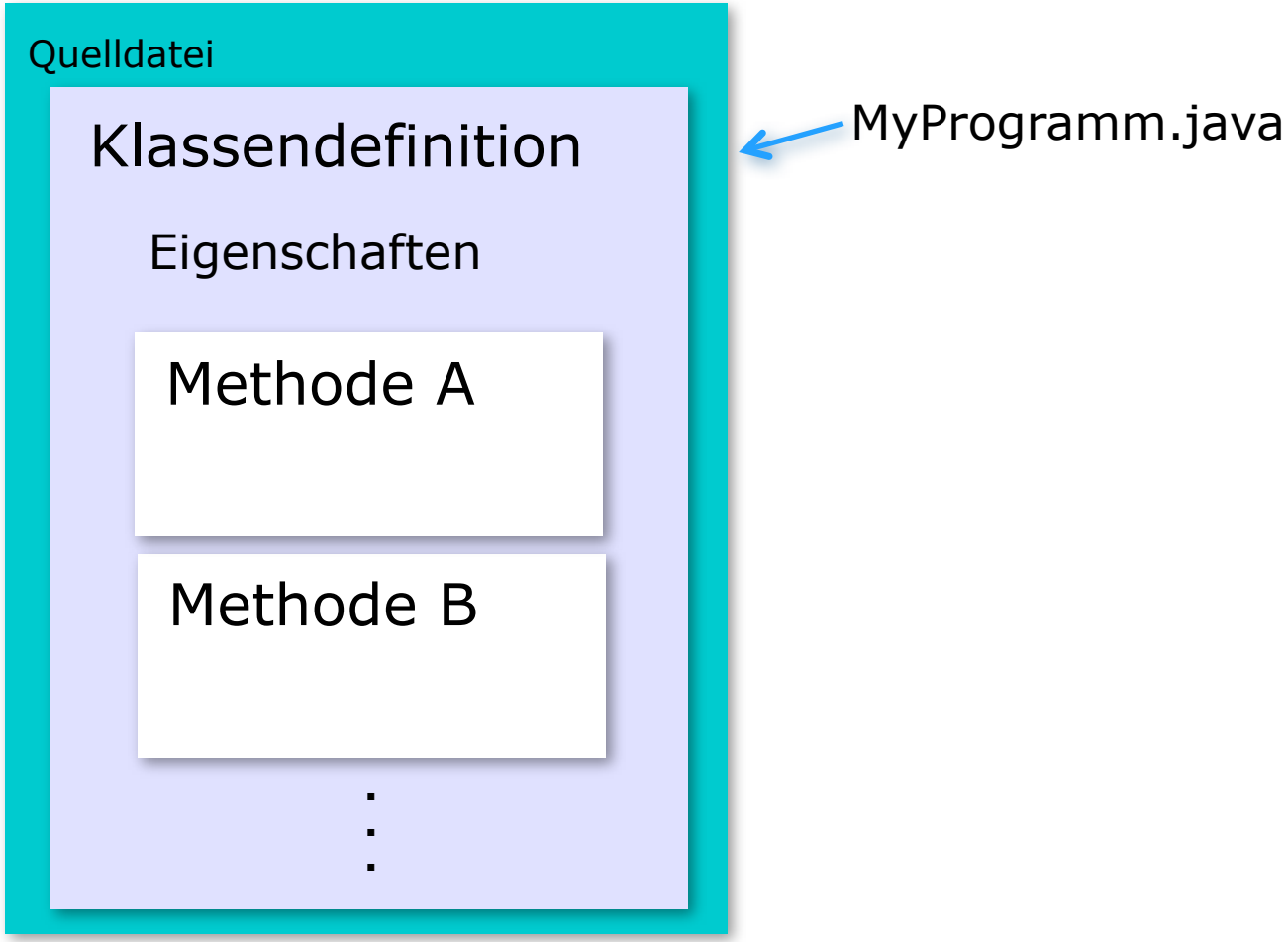
. . . . .

Verhalten:

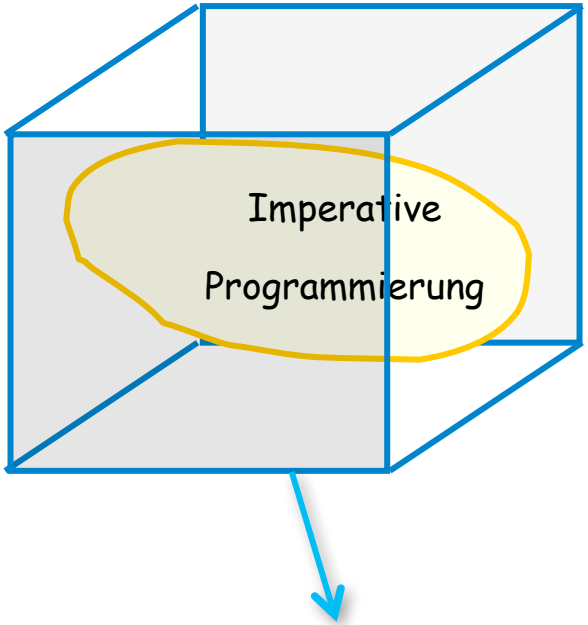
- isst
- läuft
- kratzt
- schläft

. . .

Grundstruktur eines Java-Programms

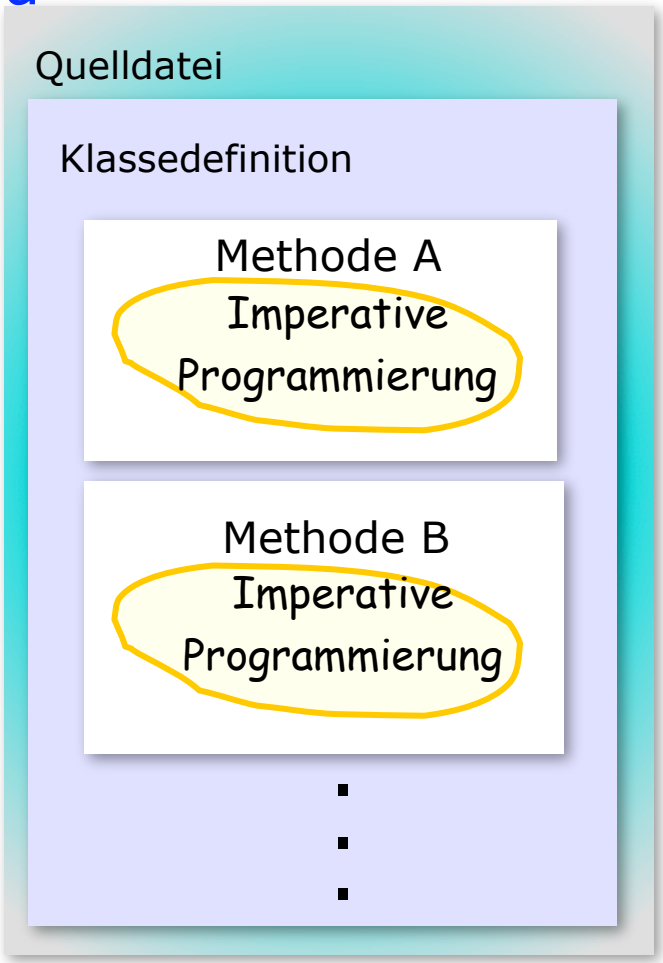


Imperative Grundbestandteile von Java



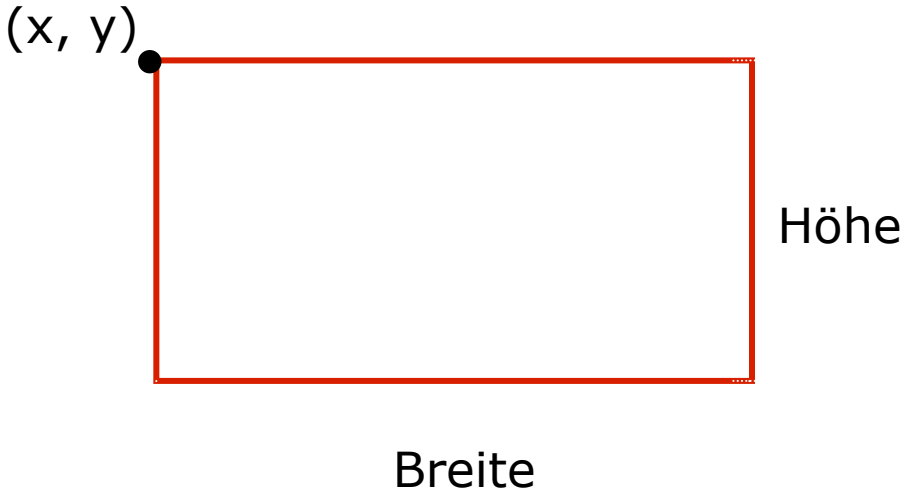
objektorientierte Verpackung

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



Modellierung von Rechteck-Objekten

Eigenschaften

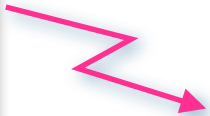


Operationen

- Fläche
- Umfang
- verkleinern
- vergrößern
- verschieben
- klonen

Modellierung von Rechteck-Objekten

```
public class Rectangle {
  // Eigenschaften
      . . .
      . . .
  // Konstruktoren
      . . .
      . . .
  // Operationen
      . . .
      . . .
      . . .
}
```



Dateiname:

Rectangle.java

Java-Anwendung

Rechtecke.java

```
class Rechteck {
  // Attribute
    . . . .
  // Konstruktoren
    . . . .
  // Methoden
    . . . .
    . . . .
   . . . .
}
```

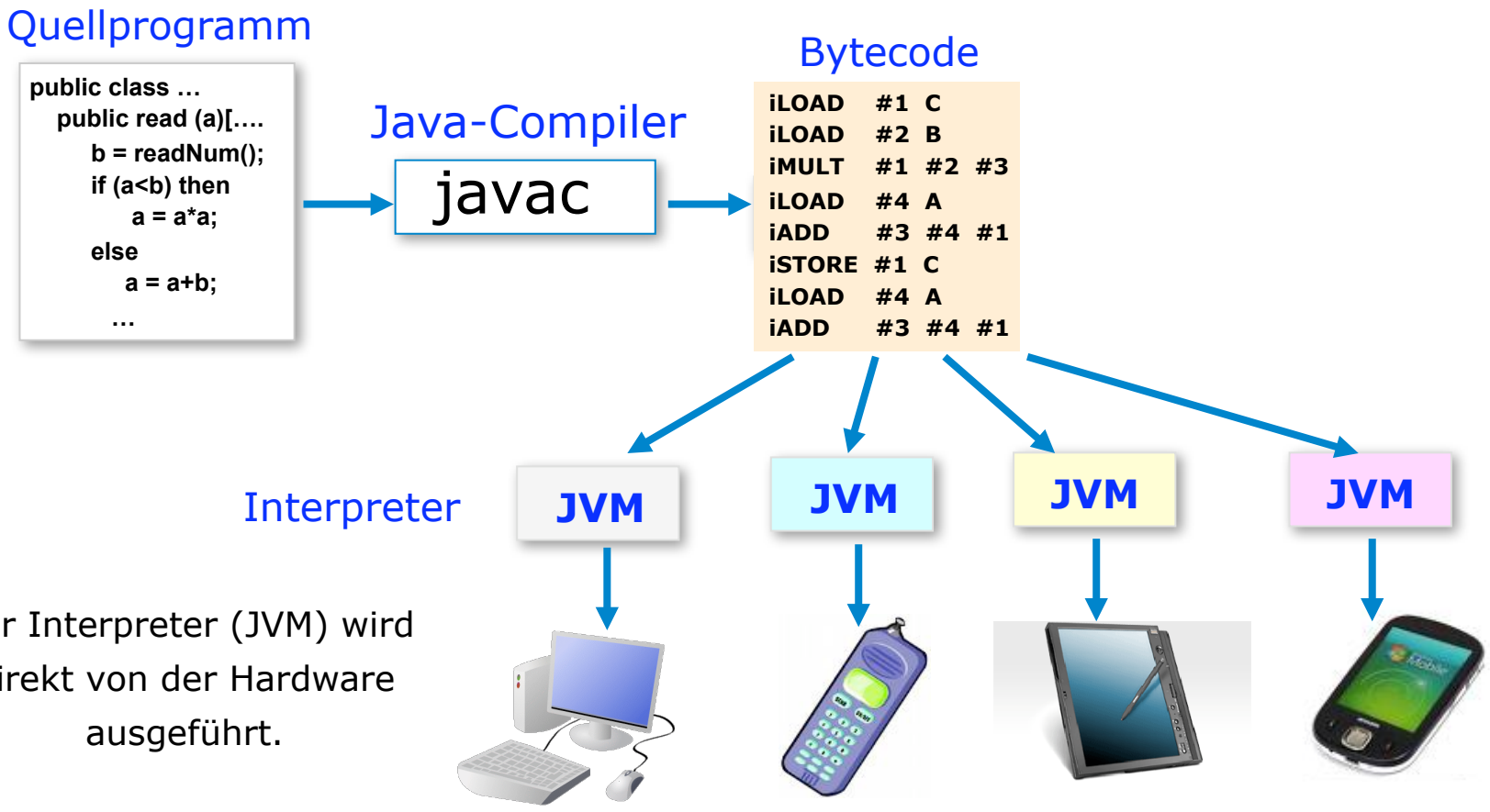
Kreis.java

```
class Kreis {
  // Attribute
    . . . .
  // Konstruktoren
    . . . .
  // Methoden
    . . . .
    . . . .
   . . . .
   . . . .
}
```

Blatt.java

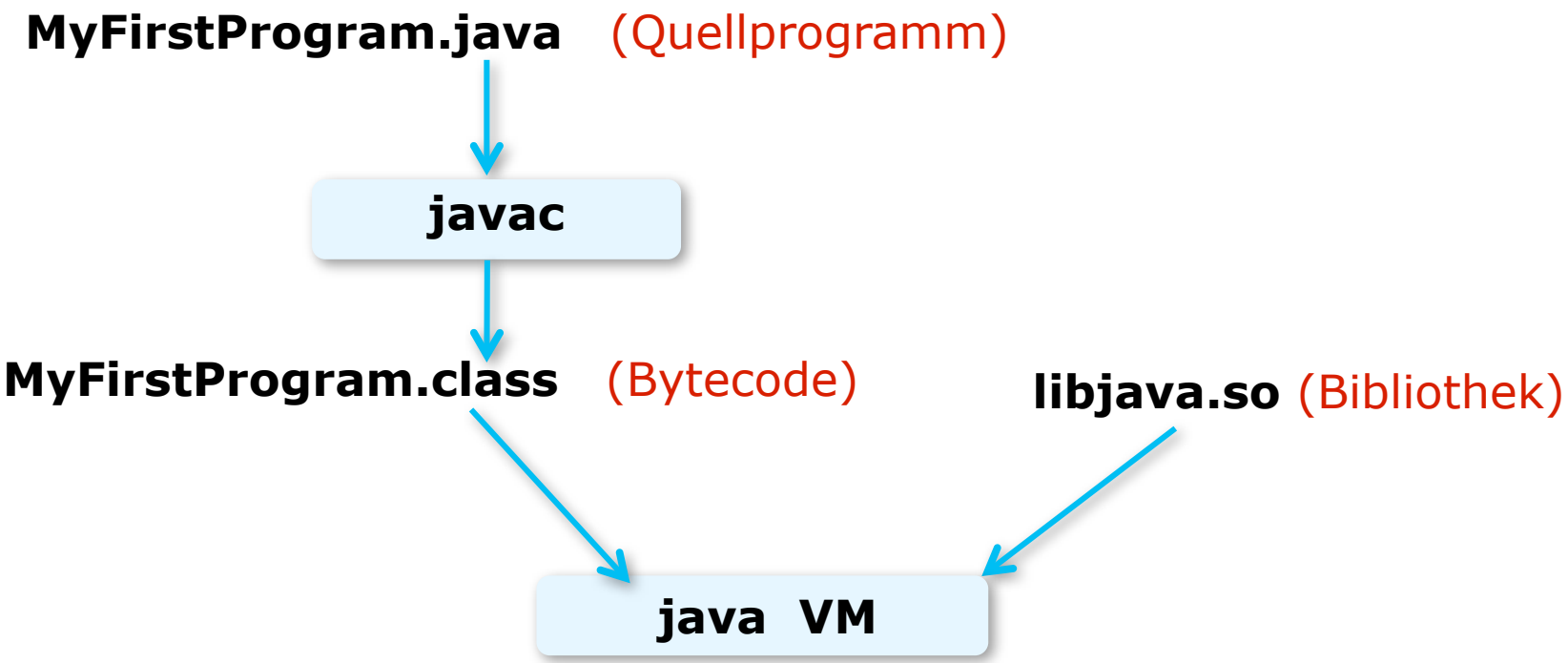
```
class Blatt {
  // Attribute
    . . . .
  // Konstruktoren
    . . . .
  // Methoden
    . . . .
    . . . .
  . . main ( . . . . ) {. .}   <-- start
    . . . .

}
```

Java ist plattformunabhängig



Der Interpreter (JVM) wird direkt von der Hardware ausgeführt.

Übersetzen/Ausführen




```
Übersetzen:     javac  MyFirstProgram.java
Ausführen:      java   MyFirstProgram
```

Java-Programme

- Java-Programme bestehen aus einer oder mehreren *Klassen*

- Eine Klasse ist in der Regel in einer eigenen, gleichnamigen Datei mit der Endung **.java** definiert, z. B.

MyFirstProgram.java

- Die Programmausführung beginnt immer mit der Methode **main** einer der Klassen.

Java-Programm

Beispiel:

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

        public static void main ( String[] args ) {
                System.out.println( "Es läuft ! " );
        }

}// end of class MyFirstProgram
```

Jedes Java-Programm muss mindestens eine Methode namens **main** haben.
Hier fängt die Programmausführung an.

Kommentare in Java

Zeilenend-Kommentare

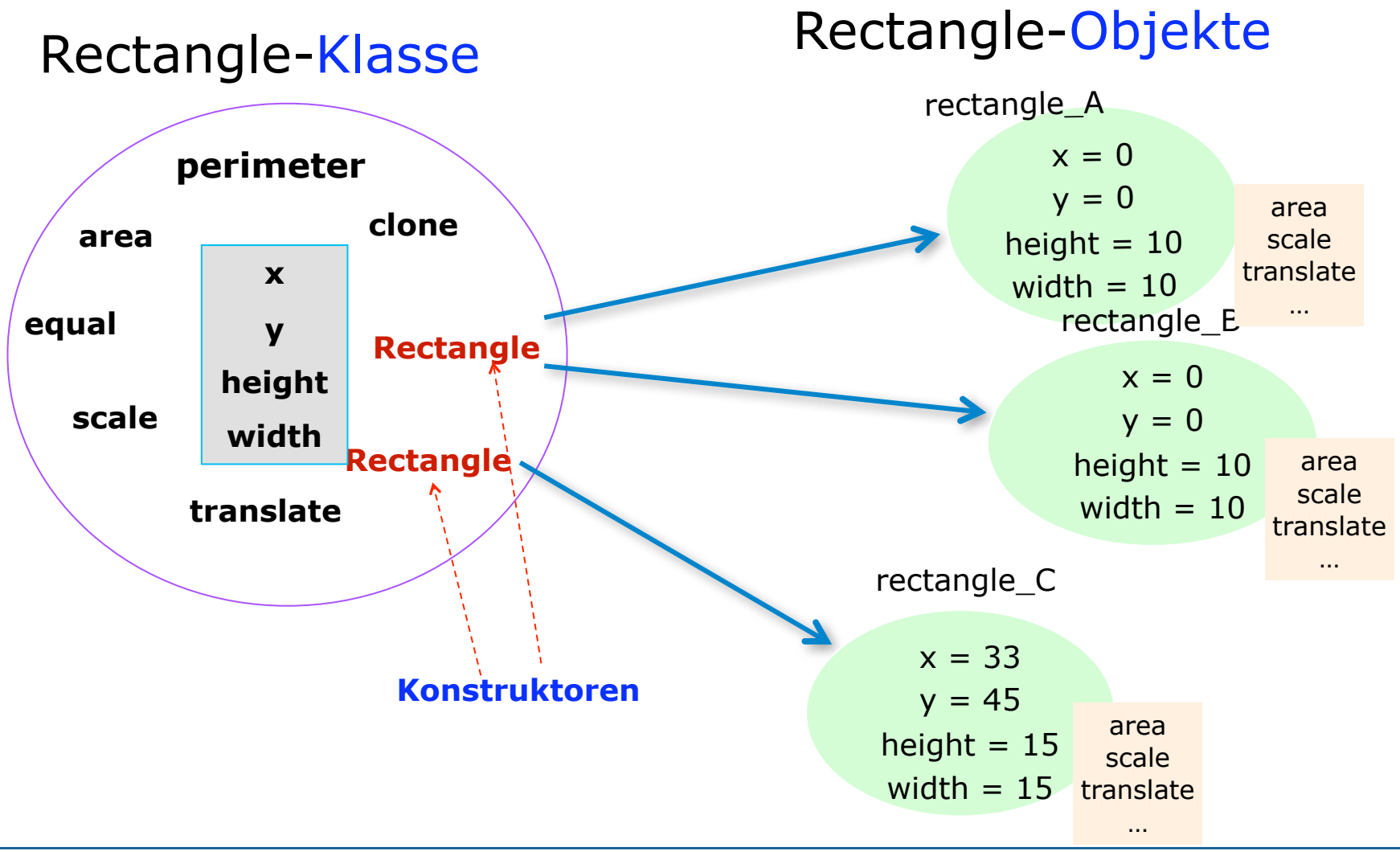
```
// von hier aus bis zum Ende der Zeile wird dieser Text ignoriert
```

Block-Kommentare

```
/*  alle  diese  Zeilen  hier  werden  von
      dem  javac  völlig  ignoriert ........
*/
```

Javadoc-Kommentare

```
/** dieser Text wird von dem javadoc-Programm verwendet,
    um automatische Dokumentation in html-Format zu
    erzeugen
*/
```


Rectangle-Klasse
perimeter
area
clone
equal
x
y
height
width
scale
Rectangle
Rectangle
translate
Konstruktoren
Rectangle-Objekte
rectangle_A
x = 0
y = 0
height = 10
width = 10
area
scale
translate
...
rectangle_B
x = 0
y = 0
height = 10
width = 10
area
scale
translate
...
rectangle_C
x = 33
y = 45
height = 15
width = 15
area
scale
translate
...

Klasse Rectangle

in Java

```
public class TestRectangle{

// main-Methode
   public static void main( String[] args ) {
      Rectangle r1 = new Rectangle();
      Rectangle r2 = new Rectangle();
      int u = r1.area();
      int f = r2.perimeter();
  }
} // end of class TestRectangle
```

```
public class Rectangle{
 // Attribute
  int x;
  int y;
  int width;
  int height;
 // Konstruktoren
  public Rectangle() {
      x = 0;
      y = 0;
      width = 10;
      height = 10;
  }
 // Methoden
  public int perimeter() {
       return 2*(width + height);
  }
  public int area() {
        return (width * height);
  }
} // end of class Rectangle
```

Klasse Rectangle

in Python

```
class Rectangle:
  # Konstruktor
    def __init__(self):
        self.x = 0
        self.y = 0
        self.width = 10
        self.height = 10

  # Methoden
    def perimeter(self):
        return 2*(self.width + self.height)

    def area(self):
        return self.width * self.height
```

Anwendung der Klasse:

```
r1 = Rectangle()
print(r1.area())
print(r1.perimeter())
r1.height = 20
print(r1.area())
```

OOP: Das Grundmodell

- Objekte haben einen lokalen Zustand.

- Objekte empfangen und bearbeiten Nachrichten.

  Ein Objekt kann

  - seinen Zustand ändern,
  - Nachrichten an andere Objekte verschicken,
  - neue Objekte erzeugen oder existierende Objekte löschen.

- Objekte sind grundsätzlich selbständige Ausführungseinheiten, die unabhängig voneinander und parallel arbeiten können.

Variablendeklarationen

Im Unterschied zu Python müssen in Java alle Variablen vor der ersten Anwendung deklariert werden.

Modifizierer	Typ	Name	Wert
	int	breite ;	
	int	hoehe =	10 ;
public	float	radius =	0.0 ;
private	float	radius =	0.0 ;

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

Java-Typsystem

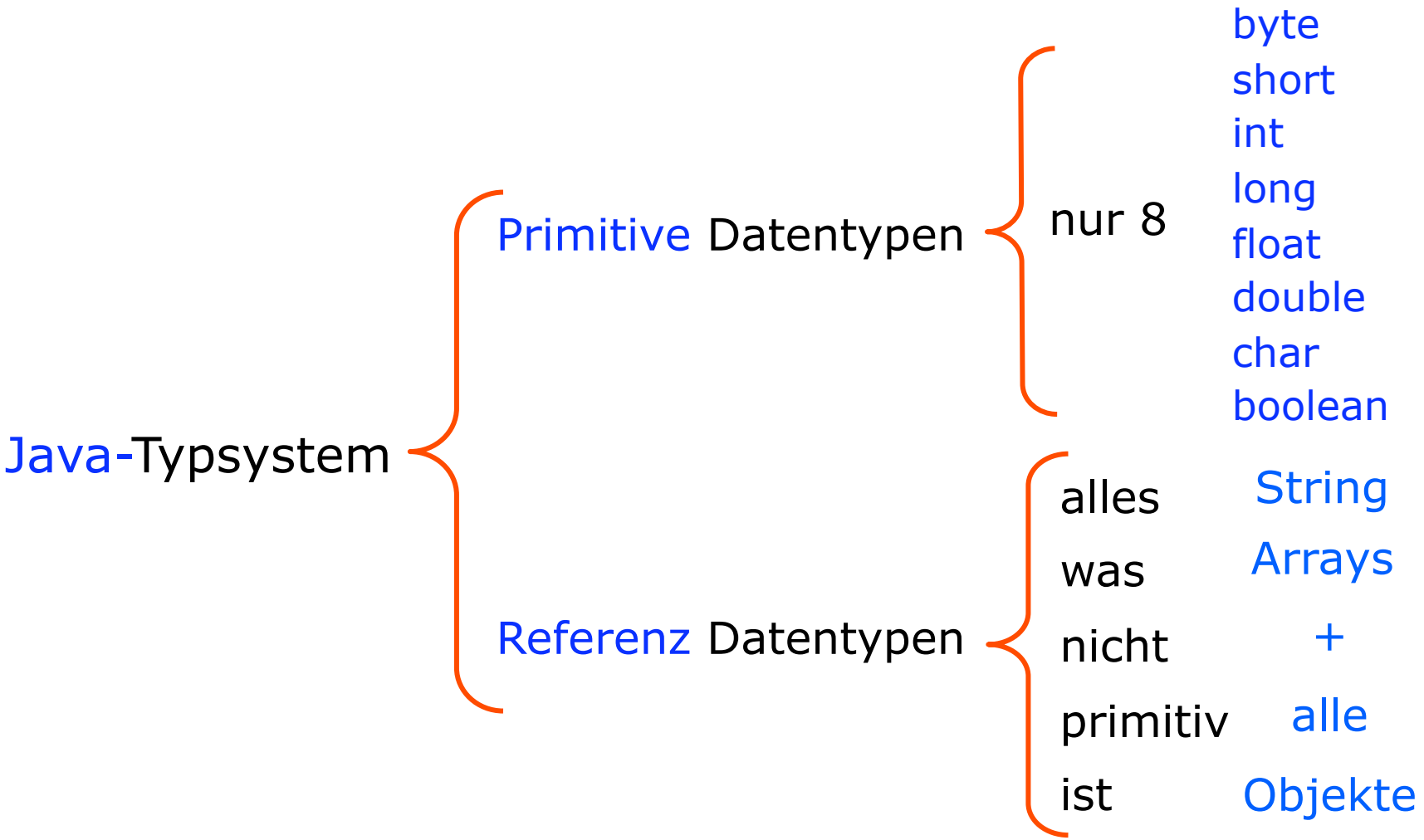
- Java ist ***streng typisiert***, d.h. jeder Ausdruck hat einen wohldefinierten Typ.

- Die Einhaltung aller durch das Typsystem definierten Regeln wird

  - zuerst statisch vom **Übersetzer** überprüft.
  - und dann dynamisch vom **Interpreter**.

- Neue Objekttypen werden durch Klassen definiert.

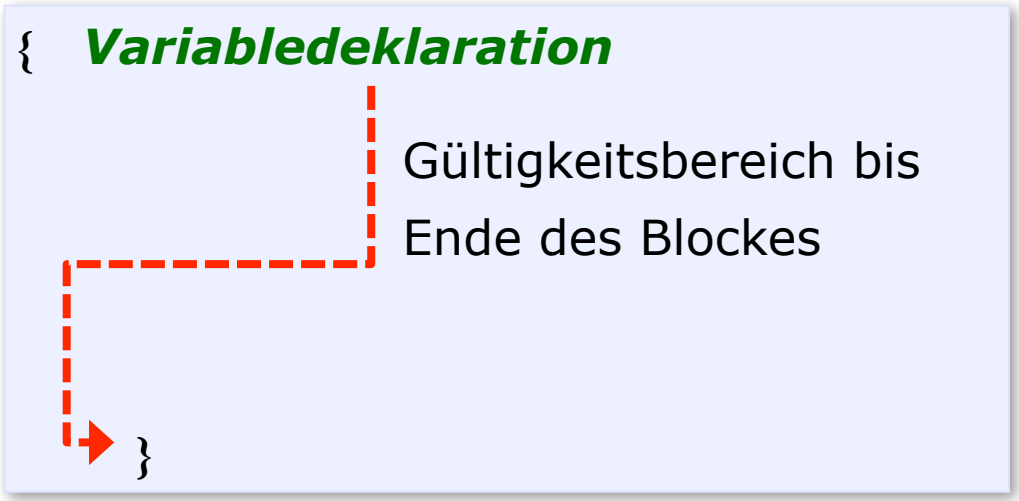
  Variablen solcher Typen sind ***Referenzen****:*


Java-Typsystem
Primitive Datentypen
nur 8
byte
short
int
long
float
double
char
boolean
Referenz Datentypen
alles
was
nicht
primitiv
ist
String
Arrays
+
alle
Objekte

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.



Ausdrücke		Wert	Typ
7 / 2	→	3	int
1 / 2	→	0	int
1 / 2.0	→	0.5	double
4 % 2	→	0	int
7 % 2	→	1	int
1 % 2	→	1	int

Einfache Befehle in Java

```
  …
// Deklaration von Variablen
   int  a, b, c ;
// Zuweisungen
   a  =  7 % 2 ;
   b  =  a * a ;
   c  =  a / b + 1 ;
   …
```

Inkrement- und Dekrement-Operatoren

Operator	Benutzung	Beschreibung	Priorität
++	++ op	Inkrementiert op um 1	13
++	op ++	Inkrementiert op um 1	13
--	-- op	dekrememtiert op um 1	13
--	op --	dekrememtiert op um 1	13

Beispiele:

```
int i = 10, j = 0;
j = ++i;
j = i++;
```

j:	11	i:	11
j:	11	i:	12

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

a: 4	b: 5	c: 8
a: 5	b: 5	c: 13

Was ist der Inhalt der Variablen **a?** **b?** **c?** und **d?**

a?	b?	c?	d?
6	4	14	12

Logische Operatoren

unär

Operator	Zeichen	Rtg.	Beispiel	Priorität
logische Negation	**!**	←	!a	13

binär

UND	**&&**	→	a && b	4
ODER	**||**	→	a || b	3

ternär

bedingter Ausdruck	B **?** A_1 **:** A_2	←	!true ? 4 : 9	2

Anweisungen zur Ablaufsteuerung

if

switch

while

do-while

for

Anweisungen zur Ablaufsteuerung

```
if ( Ausdruck )
   { Anweisungen }
else
   { Anweisungen }
```

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

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

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

Semikolon

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

if-else-Anweisung

Wahrheitswert → boolean

Runde Klammern

```
if (Ausdruck)
   { Anweisungen }
else
   { Anweisungen }
```

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;
...
```

switch-Anweisung

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

switch-Anweisung

short
int
long
char

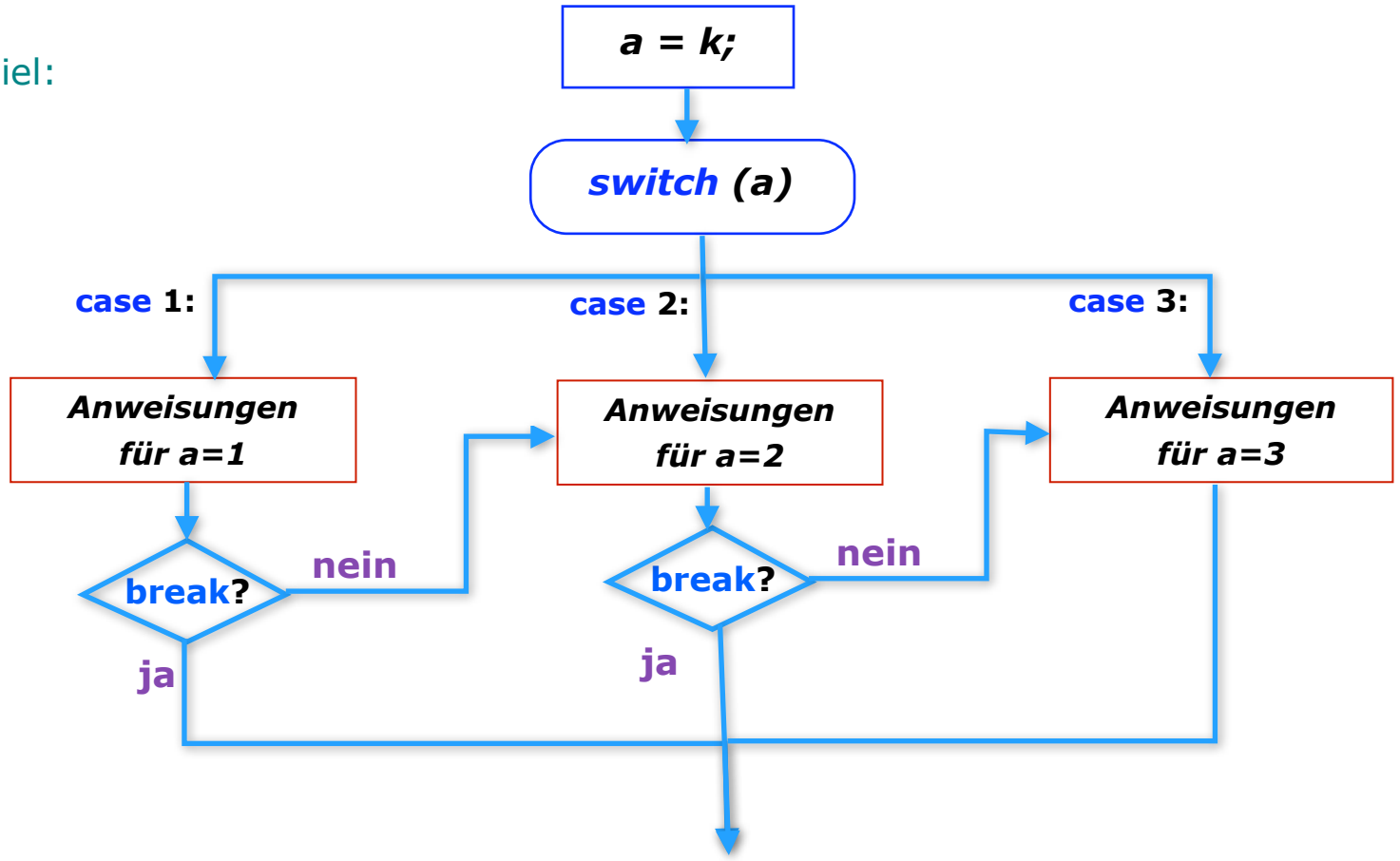
Die switch-Anweisung erlaubt die Verzweigung in Abhängigkeit vom Wert eines *ganzzahligen* Ausdrucks, der mit einer Reihe von Konstanten verglichen wird.

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

switch-Anweisung

Kontrollfluss

Beispiel:

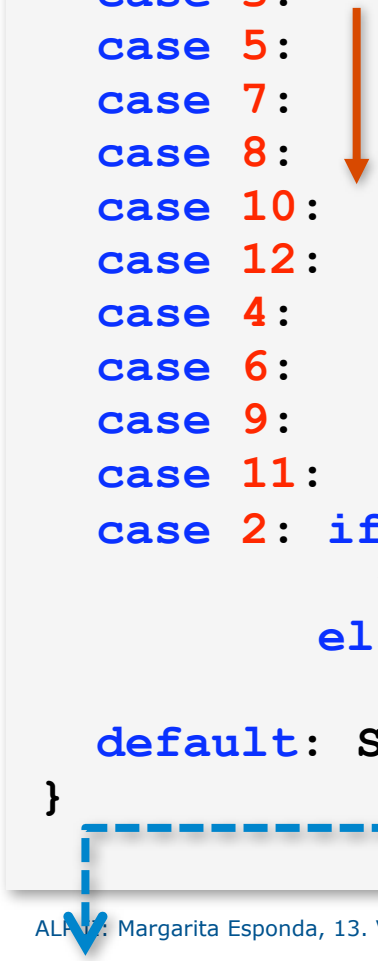


switch-Anweisung

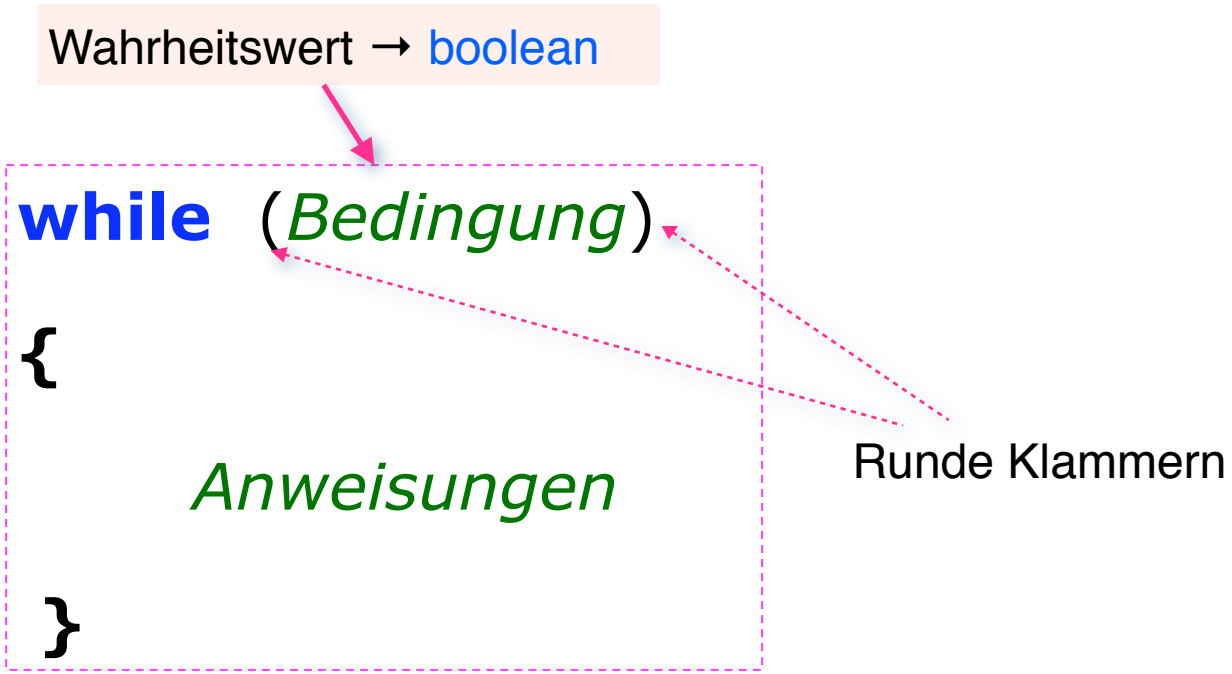
```
int zahl; String word;
...
switch( zahl ) {
   case 1:  word = "eins";     break;
   case 2:  word = "zwei";     break;
   case 3:  word = "drei";     break;
   case 4:  word = "vier";     break;
   case 5:  word = "fünf";     break;
   case 6:  word = "sechs";    break;
   case 7:  word = "sieben";   break;
   case 8:  word = "acht";     break;
   case 9:  word = "neun";     break;
   case 0:  word = "null";     break;
   default: word = "error";
}
```

switch-Anweisung

```
int monat, jahr, tage;
...
switch( monat ) {
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:   tage = 31; break;
  case 4:
  case 6:
  case 9:
  case 11:   tage = 30; break;
  case 2: if ( jahr % 4 == 0 && ( jahr % 100 != 0 || jahr % 400 == 0 ) )
            tage = 29;
          else
            tage = 28; break;
  default: System.out.println("falsche Monatsangabe");
}
```



while-Schleife



Die 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);
   . . . . .
```

do-while-Anweisung

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

Bei der do-while-Anweisung wird zunächst der Schleifen-Rumpf ausgeführt und *anschließend* die Bedingung überprüft.

for-Anweisung

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

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.

for-Anweisung

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

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-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 );
  }
...
```

$$R_n = \sum_{i=1}^{n} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \ldots + \frac{1}{n^2} \approx \frac{\pi^2}{6}$$

```
... double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1/i*i;
}
double  pi = Math.sqrt(sum*6);
...
```

falsch!

2.449489742783178

```
...
double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1/(i*i);
}
double  pi = Math.sqrt(sum*6);
...
```

falsch!

2.449489742783178

ArithmeticException

```
...
double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1.0/(i*i);
}
double  pi = Math.sqrt(sum*6);
...
```

falsch!

3.1414971639472147

Infinity

```
...
double sum = 0;
for(int i=1; i<=n; i++)
{
      sum += 1/(1.0*i*i);
}
double  pi = Math.sqrt(sum*6);
```

richtig!

3.141583104326456

for-Anweisung

Jeder der drei Ausdrücke in der for-Schleife kann auch fehlen; die Semikola müssen aber trotzdem gesetzt werden.

```
for (;;) {
   ...
}
```

=

```
while ( true ) {
   ...
}
```

Endlosschleifen!!