INTERNATIONALE HOCHSCHULE

DSPOOPI01

Sommersemester 2023

Objektorientierte Programmierung I mit Java

Sebastian Bichler

LE-BA-INFO-WiSe-22-GTW

LE-BA-WIN-WiSe-22GTW

Java – Overview

References, Tutorials, ...

- https://dev.java/
- https://dev.java/learn/getting-started/
- https://dev.java/learn/language-basics/

https://dev.java/learn/

- https://docs.oracle.com/javase/tutorial/
- https://docs.oracle.com/en/java/javase/19/docs/api/index.html
- https://docs.oracle.com/en/java/javase/19/index.html

- https://www.baeldung.com/java-tutorial
- http://tutego.de/javabuch/aufgaben/

- https://programming.guide/java/

UML

- OMG Ref: http://www.omg.org/cgi-bin/doc?formal/05-07-04
- https://www.oose.de/wp-content/uploads/2012/05/UML-Notations%C3%BCbersicht-2.5.pdf

- diagrams.net: https://app.diagrams.net/
- Dia Diagram Editor: http://dia-installer.de/
- yEd Graph Editor: https://www.yworks.com/downloads#yEd
- MS Visio
- Miro: https://miro.com

- siehe am Ende: UML

Desktop, Web, Middleware, ...

- JavaFX
  - https://openjfx.io/
  - https://www.javatpoint.com/first-javafx-application
- Java Servlets, JSP
  - https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners#servlets-jsps
- Jakarta EE
  - https://www.oracle.com/java/technologies/java-ee-glance.html
  - https://jakarta.ee/
- Vaadin
  - https://vaadin.com/
- ..

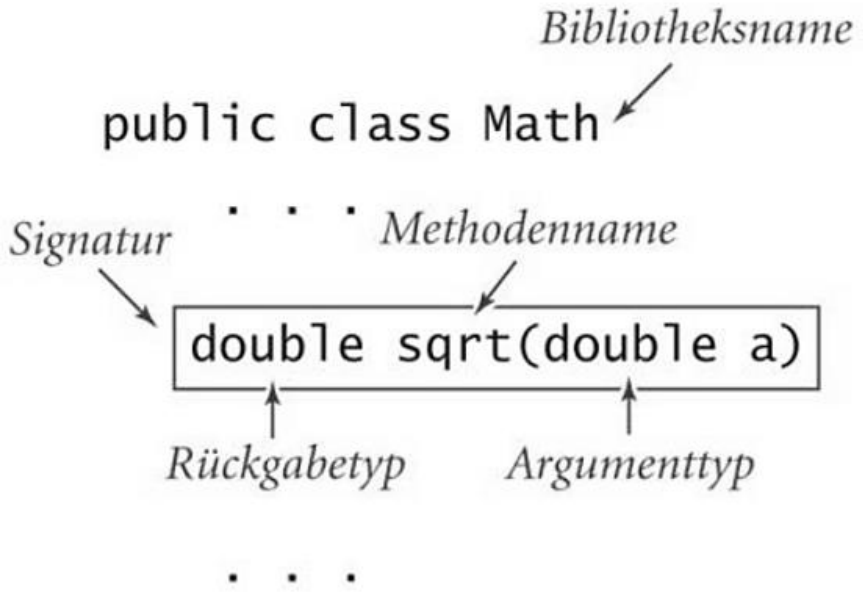
Java-Syntax

- Klassenname (Schlüsselwort `class`)
  - keine Leer- oder Sonderzeichen, beginnend mit Großbuchstaben
  - Upper-Camel-Case-Notation üblich (jedes neue Wort Großbuchstabe), z. B. `FirstViewController`
  - Dateiname muss mit Klassennamen übereinstimmen
- Parametername
  - keine Leer- oder Sonderzeichen, üblicherweise beginnend mit Kleinbuchstaben
- Abstände und Einrückungen optional
  - Empfehlung: bei jeder geöffneten {-Klammer eine Ebene einrücken
- Java-Anweisung endet mit Strichpunkt
- Java-Anweisungen dürfen über mehrere Zeilen reichen

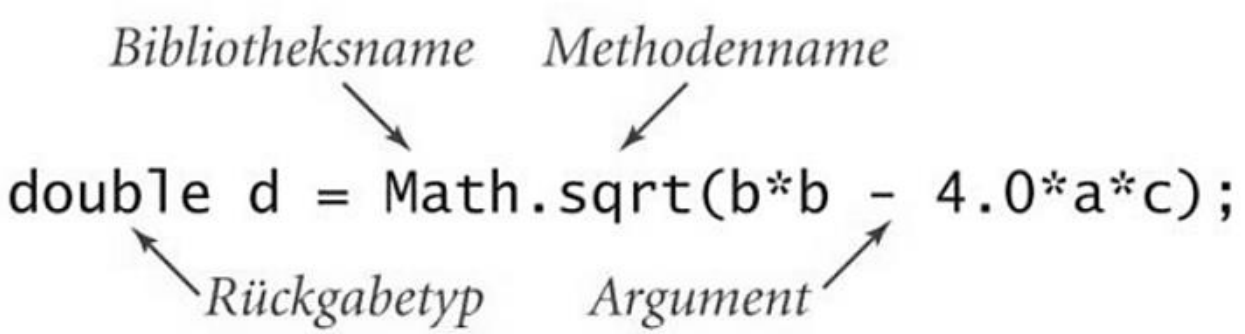
Java-Syntax

- Zeichenketten dürfen aber nicht über mehrere Zeilen reichen (ggf. mit + verbinden)
- Groß- und Kleinschreibung wird unterschieden
- Konstantenname
  - ausschließlich Großbuchstaben
- Java-Klassenbibliothek
  - in Paketen sind Klassen vordefiniert
  - `java.lang` steht automatisch zur Verfügung
  - bei anderen Import erforderlich (sonst muss immer Paket mit angegeben werden),
    - z. B. `import java.time.LocalDate;`

Signatur



Bibliotheksmethode verwenden



- Quelle Abbildung: Sedgewick & Wayne, 2011.

DateTime

```
import java.time.LocalDate;
Import java.time.format.DateTimeFormatter;
class Programm {
  public static void main(String[] args) {
      LocalDatejetzt = LocalDate.now() ;
      DateTimeFormatter meinFormat=DateTimeFormatter.ofPattern(
      "EEEE, d. MMMM yyyy" ) ;
      System.out.println("Heute ist " + meinFormat.format(jetzt) + "!") ;
  }
}
```

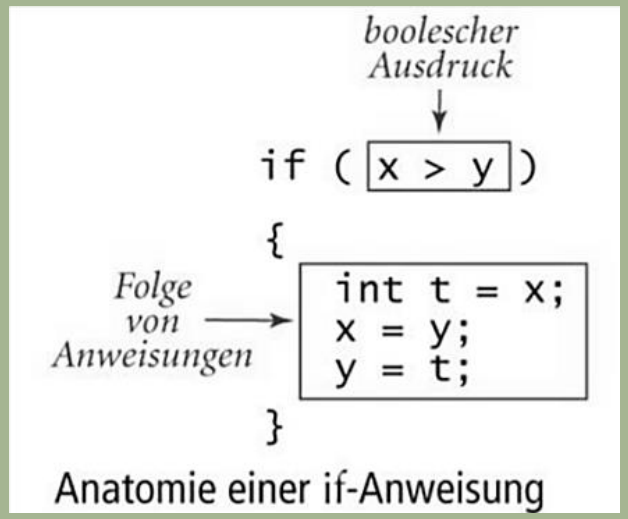
Verzweigungen

ALLGEMEIN

- Treffen von Entscheidungen
- `if`
- `switch`
- ternärer Operator

IF-VERZWEIGUNG

- Klammern und Einrücken entscheidend für sauberen Code
- `else` und `else if`



Anatomie einer if-Anweisung

SWITCH-VERZWEIGUNG

- Alternative zu verschachtelten if-Verzweigungen

```
switch (ausdruck) {
      case   constante1:
       anweisungen;
             break;
      case   constante2:
       anweisungen;
             break;
      default:
       anweisungen;
  }
```

The new „switch“

INTERNATIONALE HOCHSCHULE

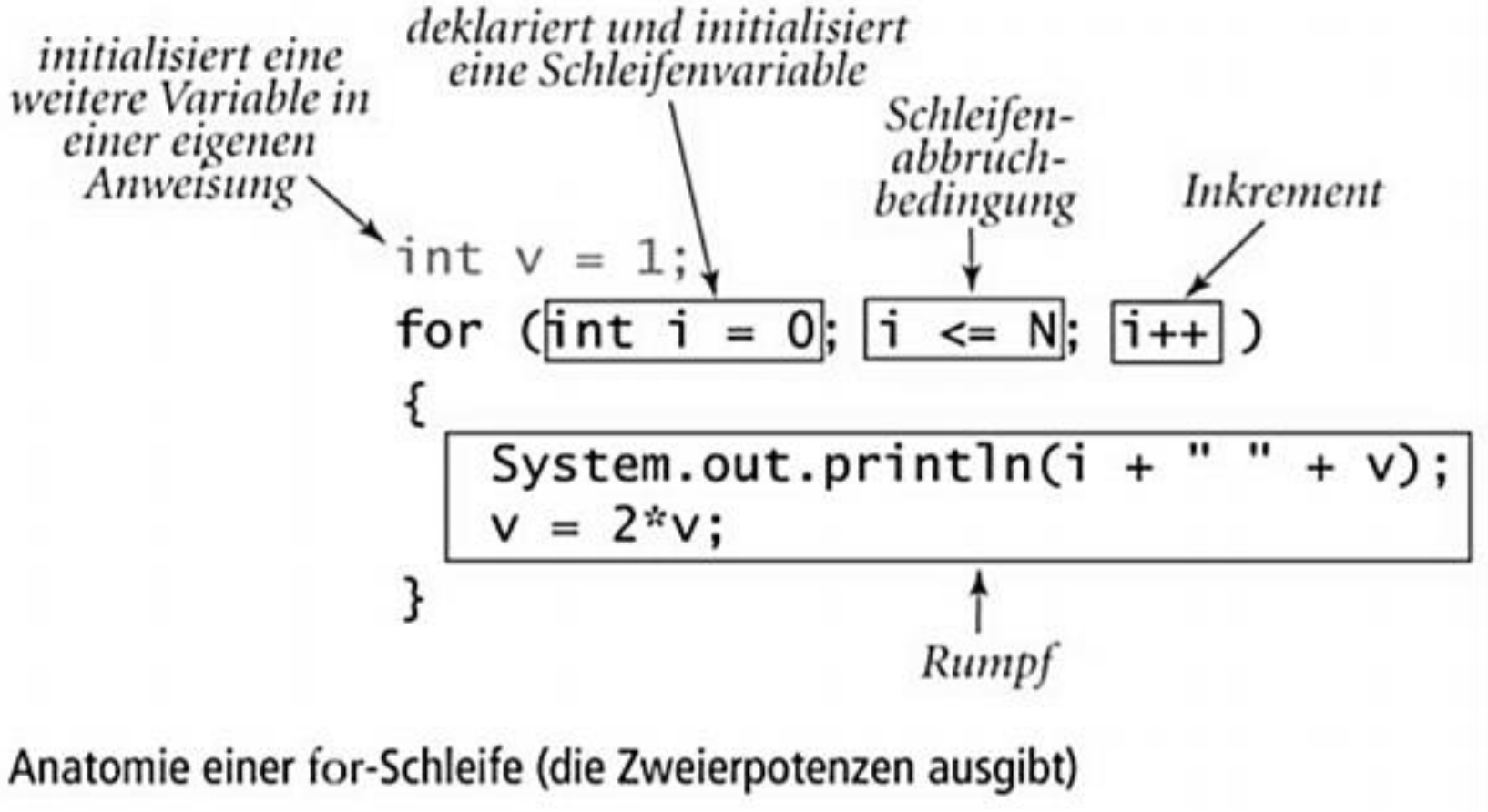
```java
int value = switch (greeding) {
case "hi" -> {
    System.out.println("I am not just yielding!");
    yield 1;
}
case "hello" -> {
    System.out.println("Me too.");
    yield 2;
}
default -> {
    System.out.println("OK");
    yield -1;
}
```

```java
String month = "März";
int days =
        switch(month) {
            case "Januar", "März", "Mai", "Juli", "August",
                    "Oktober", "Dezember" -> 31;
            case "April", "Juni", "September", "November" -> 30;
            case "Februar" -> 28; // oder 29, wenn Schaltjahr
            default -> 0;
        };

if(days != 0)
    System.out.println("Der " + month + " hat " + days + " Tage.");
```

For-loop

Simple

```
for (initialization; Boolean-expression; step)
  statement;
```



Anatomie einer for-Schleife (die Zweierpotenzen ausgibt)

Quelle Abbildung: Sedgewick & Wayne, 2011., https://www.baeldung.com/java-for-loop

For-loop

```
int i;
for(i=0; i<10; i++); {
    System.out.println(i);
}
```

Beachten Sie, dass die beiden folgenden Codes *nicht* gleichwertig sind:

```
// explizite Deklaration
int i;
for(i=0; i<10; i++) {
    doSomething(i);
}
```

```
// Deklaration von 'i' in for
for(int i=0; i<10; i++) {
    doSomething(i);
}
```

For-loop

Labled (break, continue)

- break: Verlassen des aktuellen Loops
- continue: Abbruch der aktuellen Iteration des Loops
- Labels für Wechsel der Iterationsebenen

```
aa: for (int i = 1; i <= 3; i++) {
    if (i == 1)
      continue;
    bb: for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            break aa;
        }
        System.out.println(i + " " + j);
    }
}
```

https://www.baeldung.com/java-for-loop , https://javabeginners.de/Schleifen_und_Verzweigungen/break_und_continue.php

For-loop

Enhanced

```
for(Type item : items)
   statement;
```

- Vor- und Nachteile ...

```
int[] intArr = { 0,1,2,3,4 };
for (int num : intArr) {
    System.out.println("Enhanced for-each loop: i = " + num);
}
```

List<String>

```
for (String item : list) {
    System.out.println(item);
}
```

Map<String,Integer>

```
for (Entry<String, Integer> entry : map.entrySet()) {
    System.out.println(
      "Key: " + entry.getKey() +
      " - " +
      "Value: " + entry.getValue());
}
```

https://www.baeldung.com/java-for-loop

For-loop

Iterable.forEach()

- Vor- und Nachteile ...

```
List<String> names = new ArrayList<>();
names.add("Larry");
names.add("Steve");
names.add("James");
names.add("Conan");
names.add("Ellen");

names.forEach(name -> System.out.println(name));
```

https://www.baeldung.com/java-for-loop

For-loop

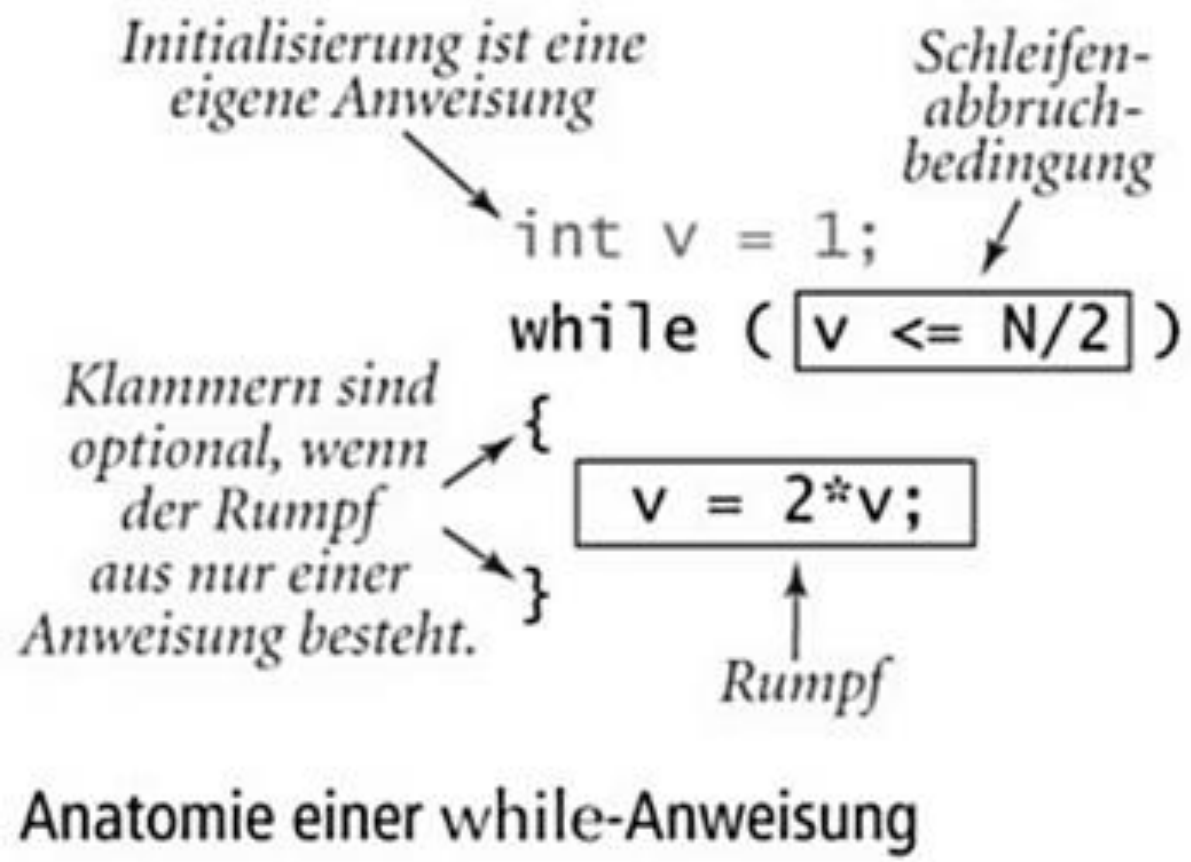
Iterator

```java
// Make a collection
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
cars.add("Mazda");

// Get the iterator
Iterator<String> it = cars.iterator();

// Print the first item
System.out.println(it.next());
```

- Vor- und Nachteile ...
- Iterator vs forEach()
- Modify a collection → iterator

https://www.w3schools.com/java/java_iterator.asp

While-loop



Anatomie einer while-Anweisung

Quelle Abbildung: Sedgewick & Wayne, 2011.

Kommentare

- einzeilig / /
- mehrzeilig von / * bis * /
- Javadoc-Kommentare von / * * bis * /
- klare Kommentare sind essenziell!

INTERNATIONALE HOCHSCHULE

Primitive Datentypen

- müssen deklariert werden:
  `datentyp varnmane;`
- müssen initialisiert werden:
  `varname = wert;`
- auch zusammen möglich:
  `datentyp varname = wert;`

DATENTYP	DETAILS
`byte`	Ganzzahl 1 Byte
`short`	Ganzzahl 2 Bytes
`int`	Ganzzahl 4 Bytes
`long`	Ganzzahl 8 Bytes
`boolean`	true or false
`char`	ein Unicode Zeichen
`float`	Fließkommazahl 4 Bytes
`double`	Fließkommazahl 8 Bytes

Datentyp `String`

- Zeichenfolgen
- im engeren Sinn kein elementarer/primitiver Datentyp, sondern Klasse
- kann zusammen mit primitiven Datentypen als integrierter Datentyp angesehen werden

String-Operationen

```
String a = "now is ";
String b = "the time ";
String c = "to"
```

Aufruf	*Wert*
a.length()	7
a.charAt(4)	i
a.substring(2, 5)	"w i"
b.startsWith("the")	true
a.indexOf("is")	4
a.concat(c)	"now is to"
b.replace('t','T')	"The Time "
a.split(" ")[0]	"now"
a.split(" ")[1]	"is"
b.equals(c)	false

Beispiele für Stringoperationen

Un-/Boxing
(Wrapper-Klassen)

- In Java gibt es primitive Datentypen, weil das Vorteile hat.
- Es gibt zu jedem primitiven Datentyp ein entsprechendes Objekt. Dabei wird um die primitiven Datentypen eine Objektstruktur gelegt – in Form von *Wrapper*-Klassen.
- Somit ist der direkte Zugriff theoretisch nicht möglich. Ab Java 5.0 wurde aber der Umgang dennoch vereinfacht.
  - Un-/Boxing bzw. Autoboxing

```
int i = 55;
Integer j = new Integer(i);
Integer k = new Integer(33);
j = new Integer(j.intValue() + 1);
```

nun möglich

```
int i = 55;
Integer j = i;
Integer k = 33;
j++;
```

INTERNATIONALE HOCHSCHULE

Deklaration und Erzeugung

- zur effizienten Verarbeitung gleichartiger Daten
- werden automatisch initialisiert
- Deklaration:
  - `int[] myIntArray;`
  - `double[] myDoubleArray;`
  - `String[] myStringArray;`
- Erzeugung:
  - `myIntArray = new int[5];`
- Deklaration und Erzeugung in einem:
  - `double[] myDoubleArray = new double[5];`
- Abkürzung bei direkter Belegung mit Werten:
  - `String[] myStringArray = {"rot","grün","blau"};`

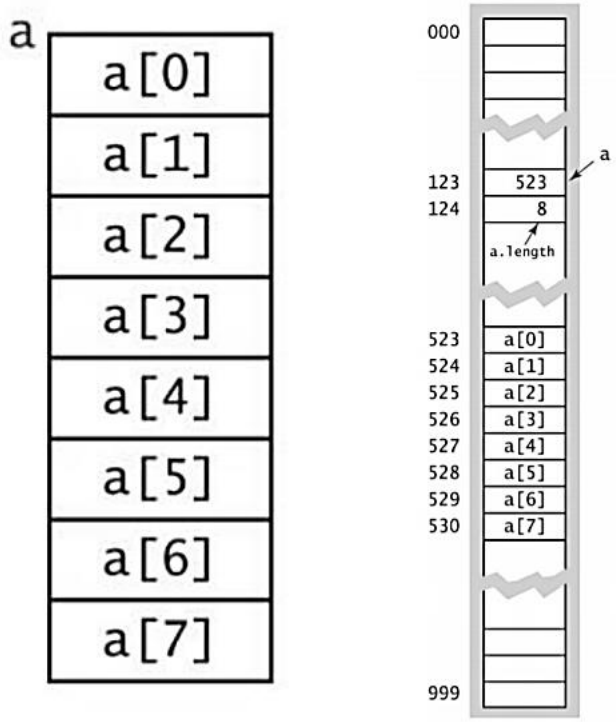
INTERNATIONALE HOCHSCHULE

Zugriff, Lesen, Sortierung

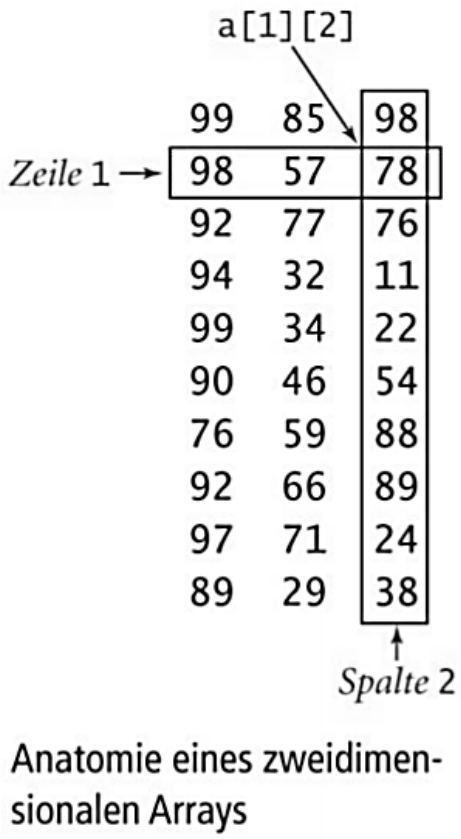
- mehrdimensionale Arrays:
  - `int[][] myMehrdimArray;`
  - `myMehrdimArray = new int[8][8];`
  - `int[][] myMehrdimArray = {{1,2,4},{6,7,8}};`
- Zugriff auf Arrayelemente:
  - `myIntArray[0] = 5;`
  - `System.out.println(myMehrdimArray[1][2]);`
- zum Lesen (und nur Lesen!) in Schleifen:
  - `for (int element: myIntArray)`
  - `System.out.println(element);`
- sortieren und vergleichen von Arrays:
  - `Arrays.sort(myArray);`
  - `Arrays.equals(myArray1, myArray2);`

INTERNATIONALE HOCHSCHULE

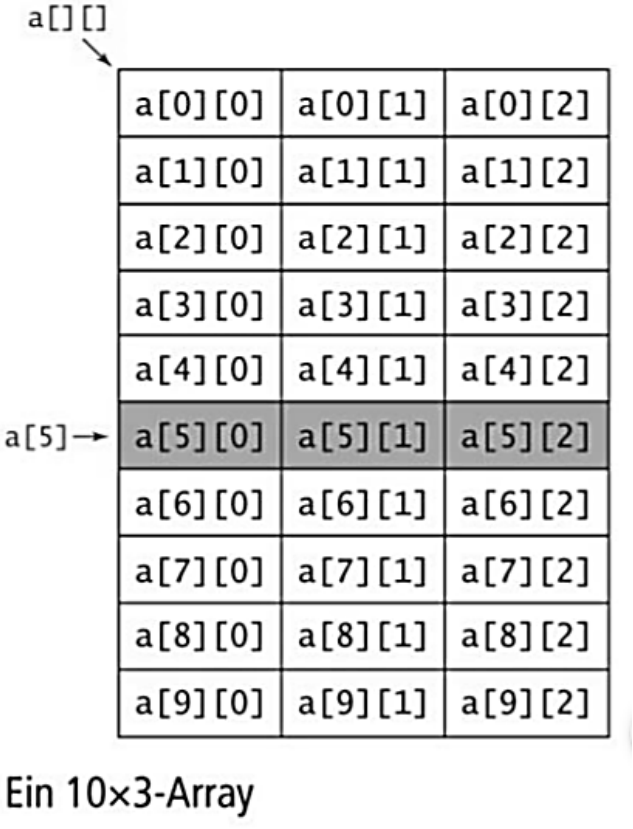
Speicherdarstellung



Mehrdimensionale Arrays



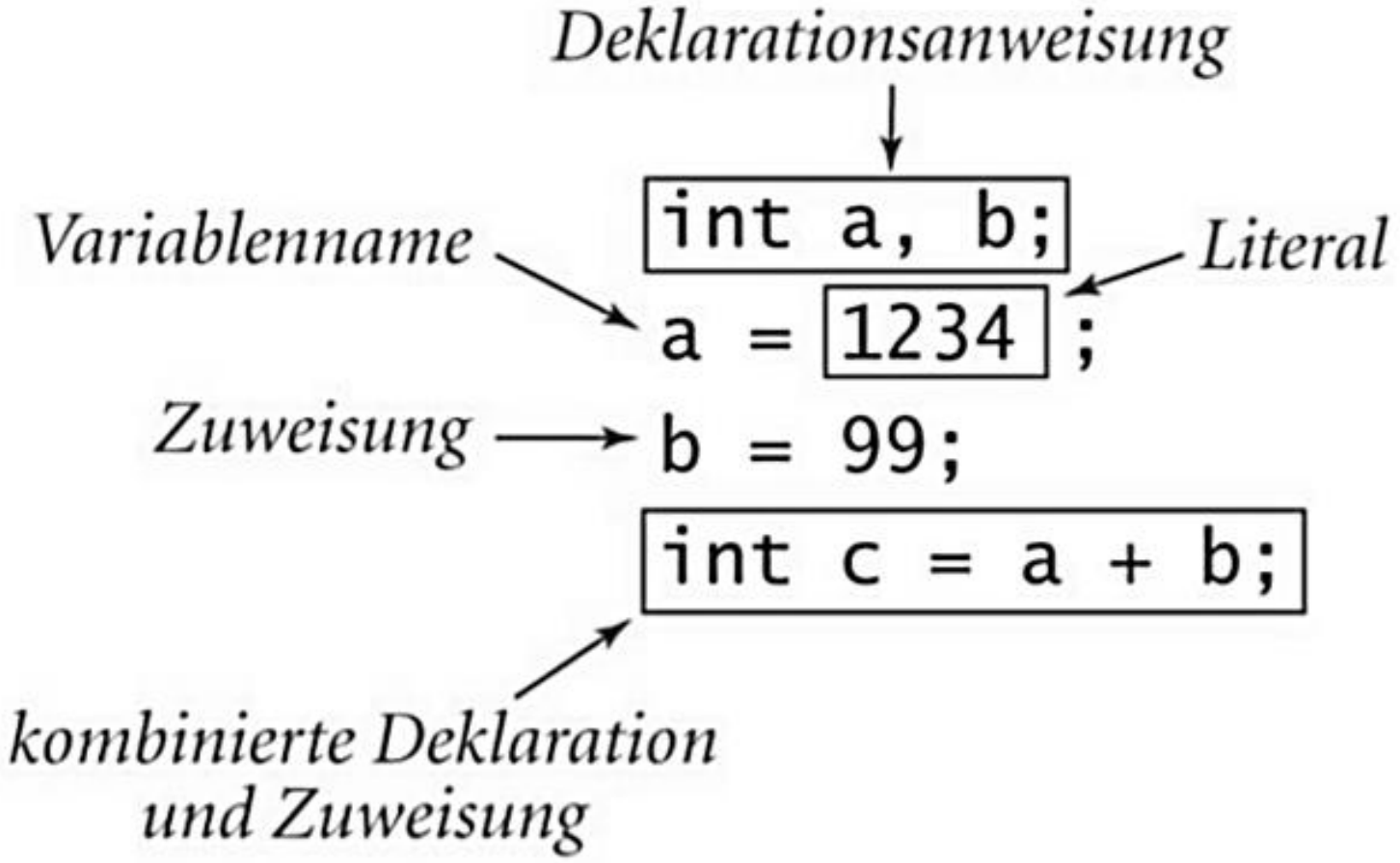
Anatomie eines zweidimensionalen Arrays



Ein 10×3-Array

Quelle Abbildung: Sedgewick & Wayne, 2011.

Verwendung primitiver Datentypen



Quelle Abbildung: Sedgewick & Wayne, 2011.

Variablendeklaration ohne Typangabe

var

- Seit Java 10 ist die Verwendung von **var** erlaubt
- Verfahren heißt: *Local-Variable Type Inference*

```
var i = 10;                 // int
var d = 18.3;               // double
var s = "Java";             // String (Zeichenkette)
```

```
// herkömmlicher Code bis Java 9
String[] sar           = new String[3];
ArrayList<String> list = new ArrayList<>();
Stream<String> stream  = list.stream();
```

In solchen Fällen erlaubt `var` einen viel klareren Code ohne Redundanz:

```
// moderner Code ab Java 10
var sar    = new String[3];
var list   = new ArrayList<String>();
```

Kofler, M. (2022). *Java: Der Grundkurs* (4. Aufl.). Rheinwerk Verlag. https://www.rheinwerk-verlag.de/java-der-grundkurs/

Operatoren

PRIORITÄT	OPERATOR	BEDEUTUNG
1 →	()	Methodenaufruf
	[]	Zugriff auf Felder
	.	Zugriff auf Objekte, Methoden etc.
2 →	++--	Inkrement/Dekrement (Postfix, z.B. a++)
3 ←	++--	Inkrement/Dekrement (Präfix, z.B. ++a)
	+-	Vorzeichen
	!~	logisches Nicht, binäres Nicht
	new	Objekte erzeugen
	(typ)	explizite Typumwandlung (Casting)
4 →	*/ %	Multiplikation, Division, Restwert
5 →	+-	Addition, Substraktion
	+	Verbindung von Zeichenketten

PRIORITÄT	OPERATOR	BEDEUTUNG
7 →	> >=	Vergleich größer, größer-gleich
	< <=	Vergleich kleiner, kleiner-gleich
8 →	== !=	Vergleich gleich, ungleich
9 →	&	logisches Und
10 →	^	logisches Exklusiv-Oder
11 →	|	logisches Oder
12 →	&&	logisches Und (Short-circuit Evaluation)
13 →	||	logisches Oder (Short-circuit Evaluation)
15 ←	=	Zuweisung
	+=-=	Grundrechenarten und Zuweisung

Quelle Text: Kofler, M. (2022). *Java: Der Grundkurs* (4. Aufl.). Rheinwerk Verlag. https://www.rheinwerk-verlag.de/java-der-grundkurs/

INTERNATIONALE HOCHSCHULE

Scanner

- über `java.util.Scanner`
- erzeugen:
  - `java.util.Scanner scan = new java.util.Scanner (System.in) ;`
- lesen/scannen, z. B.:
  - `int a = scan.nextInt () ;`
  - `double b = scan.nextDouble () ;`
  - `String s = scan.nextLine ();`
- schließen:
  - `scan.close () ;`

Allgemein

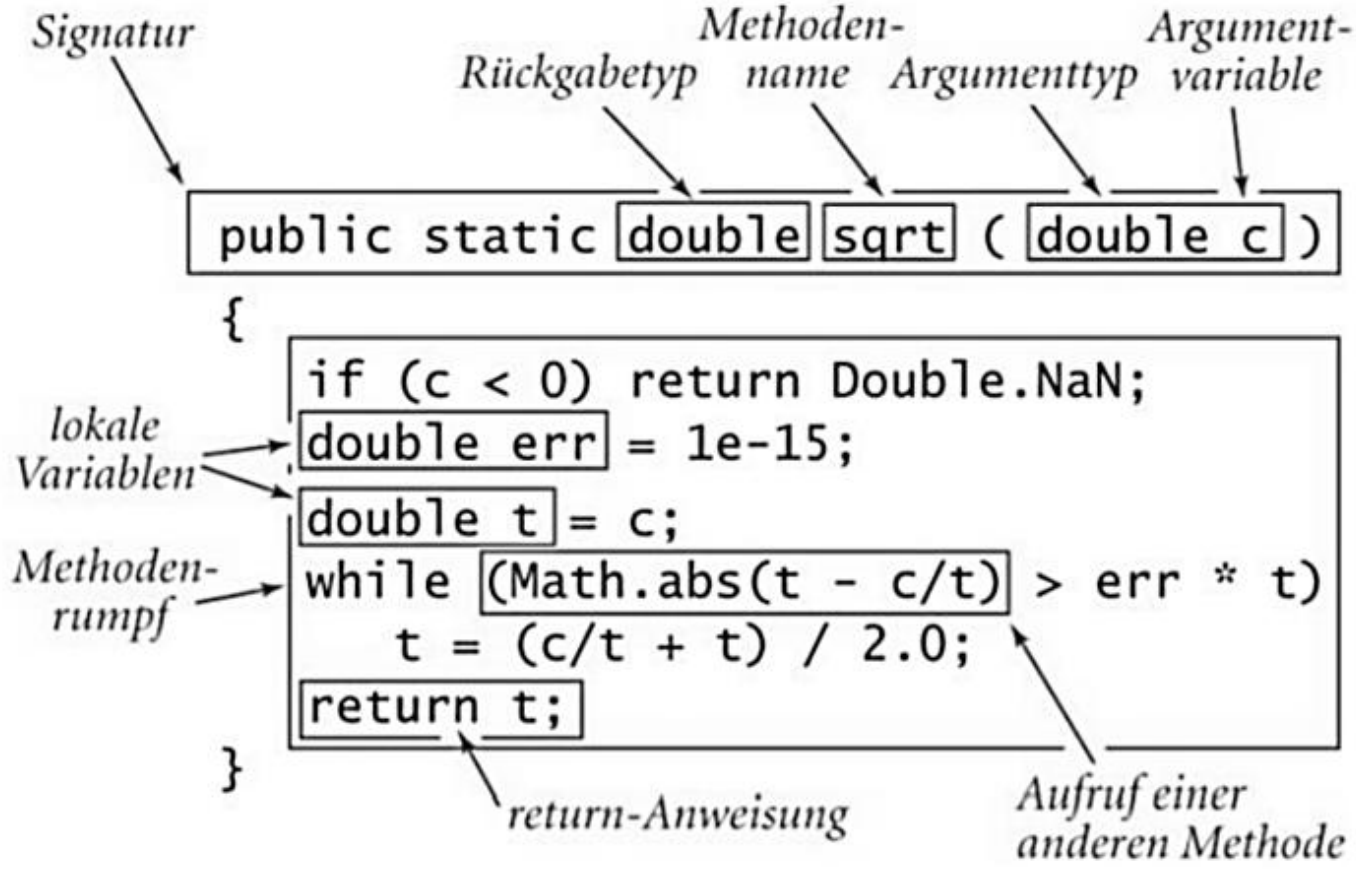
- es geht um die Organisation von Programmcode
- bisher `main`-Methode verwendet
  - nicht optimal
  - „Spaghetti-Code“: undurchsichtig, schwer zu warten
- es gibt statische und nichtstatische Methoden
  - statische: benötigen kein Objekt der Klasse (Klassenmethoden)
  - nichtstatische: Objekt der Klasse erforderlich (Instanzmethoden)
- Bezeichnung mit Verb und beginnendem Kleinbuchstaben
- optionale Modifizierer

Kombinierte Datentypen

class, record

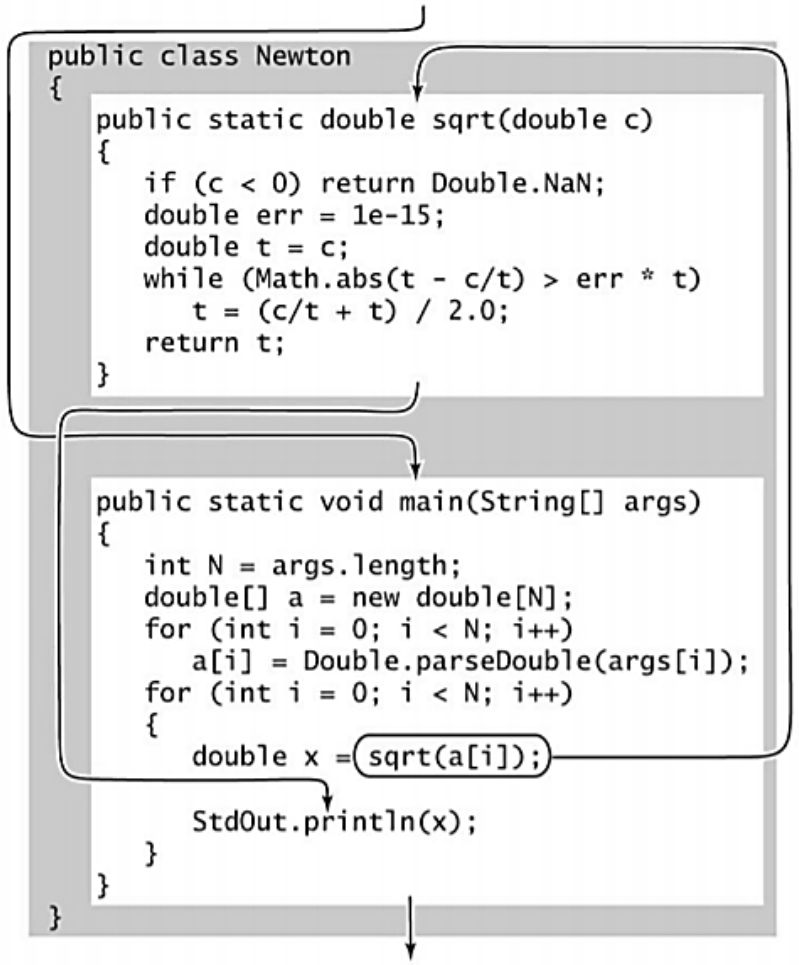
- Ab Java 14 wurden (wieder) records eingefügt. Sie ähneln dem Typ „class“.
- Ohne Records musste man sich besonders bei Datenklassen und dem funktionalen Paradigma mit Bibliotheken wie *Lombok* behelfen.
- Weitere Infos dazu folgen … solange:
  - https://entwickler.de/ddd/record-type-value-objects-werden-endlich-java-native-003/

Anatomie einer Klassenmethode



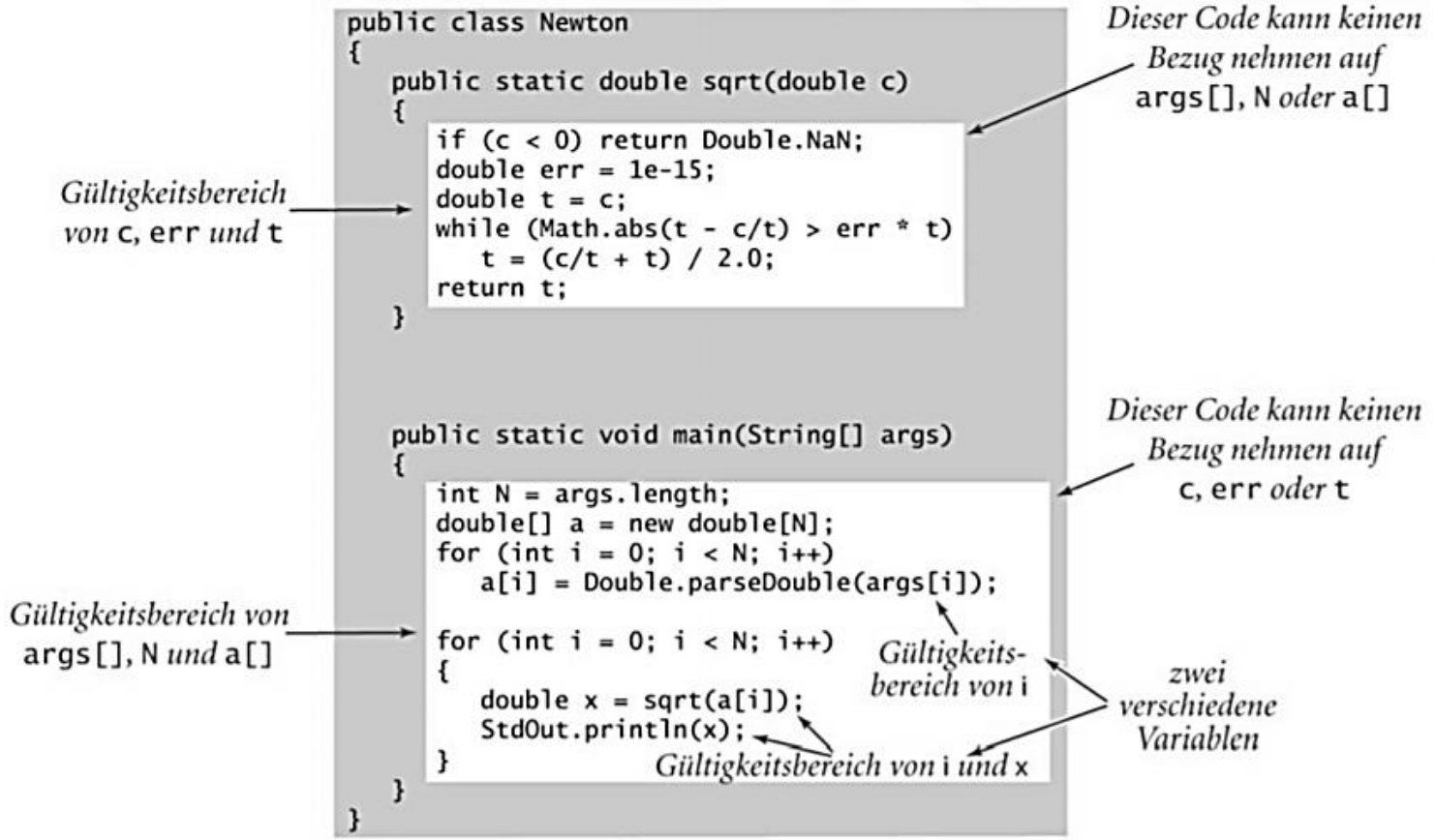
- Quelle Abbildung: Sedgewick & Wayne, 2011.

Programmfluss bei Aufruf einer Klassenmethode



Quelle Abbildung: Sedgewick & Wayne, 2011.

Gültigkeitsbereiche von Variablen



- Quelle Abbildung: Sedgewick & Wayne, 2011.

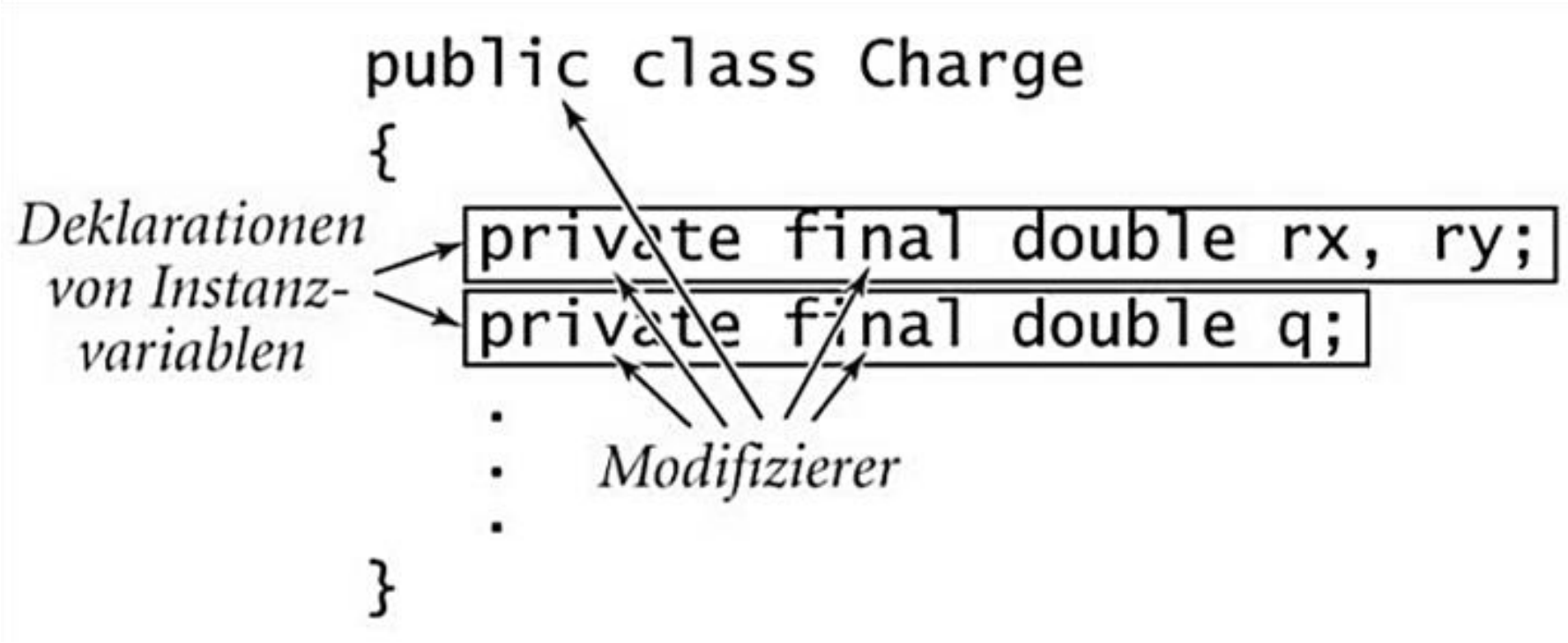
INTERNATIONALE HOCHSCHULE

Top-Level-Klassen

- auf höchster Ebene in einer Java-Datei definiert, nicht innerhalb Klammernebene
- Top-Level-Klassen nicht `private` oder `protected`
- pro Java-Datei nur eine öffentliche Top-Level-Klasse
- bezeichnet mit Substantiv und beginnendem Großbuchstaben
- innerhalb der Klasse können beliebig viele Variablen (Fields) und Methoden definiert werden

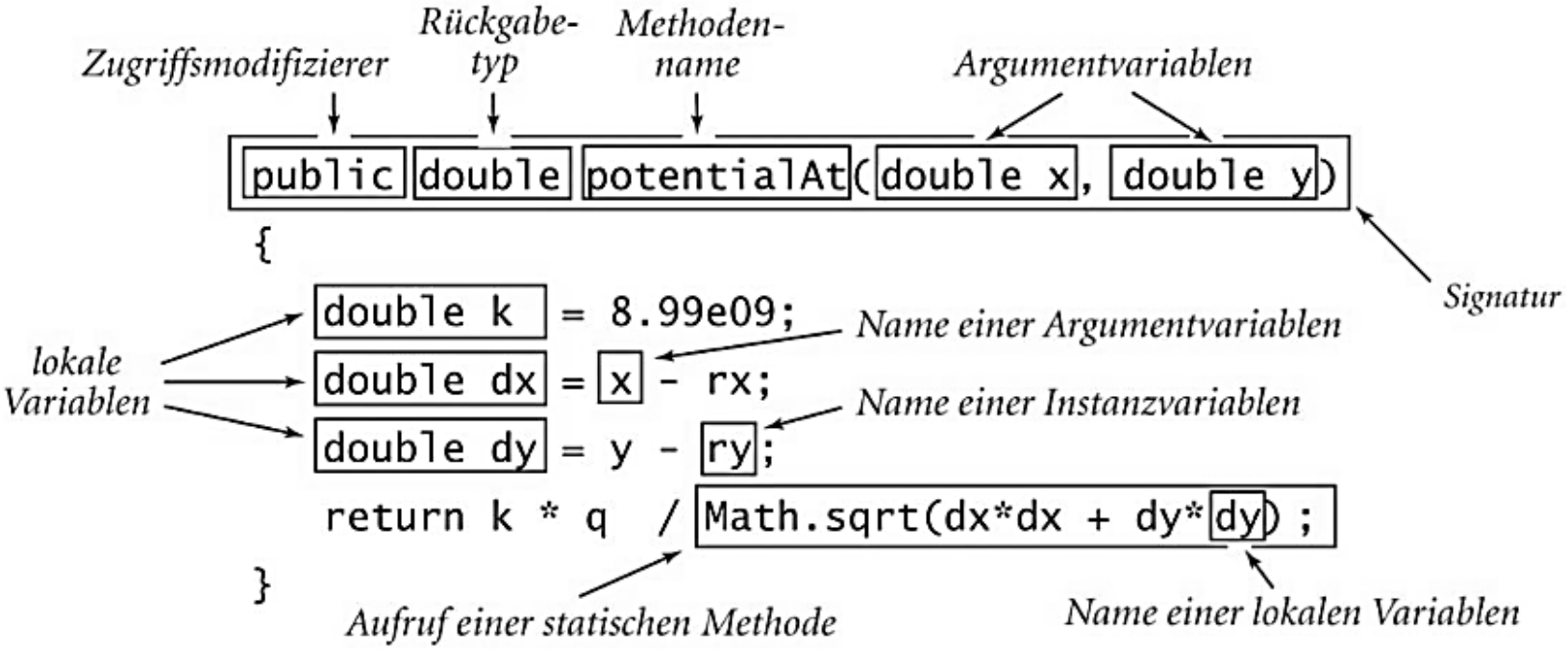
Quelle Text: Kofler, M. (2022). *Java: Der Grundkurs* (4. Aufl.). Rheinwerk Verlag. https://www.rheinwerk-verlag.de/java-der-grundkurs/

Instanzvariablen



Quelle Abbildung: Sedgewick & Wayne, 2011.

Instanzmethoden



Quelle Abbildung: Sedgewick & Wayne, 2011.

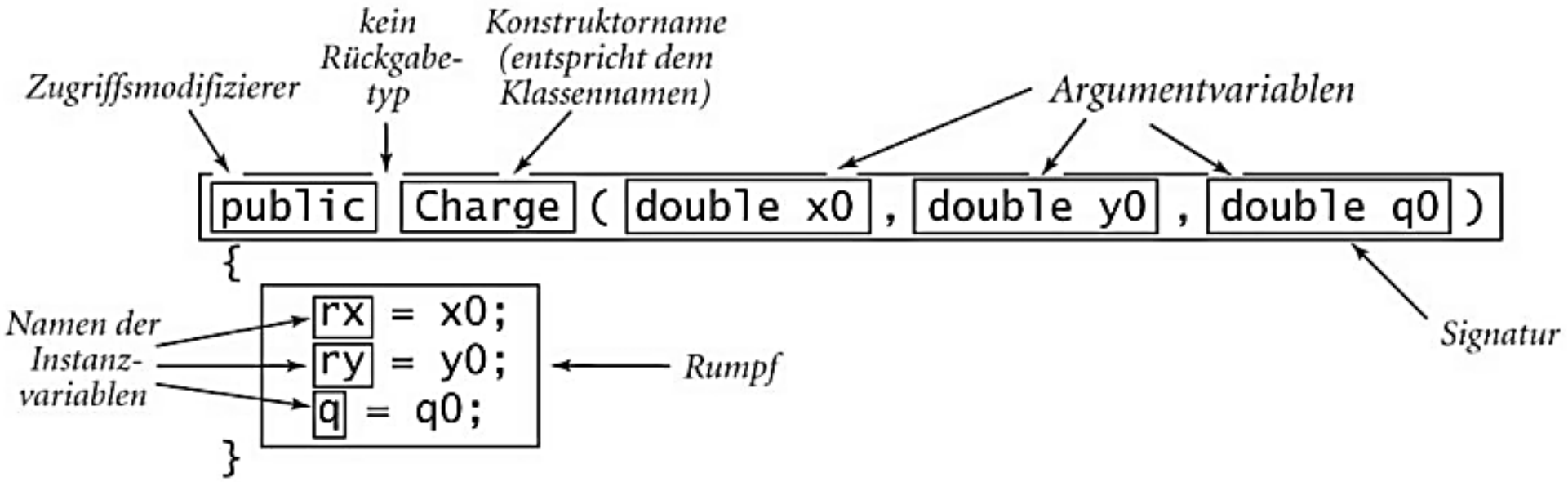
INTERNATIONALE HOCHSCHULE

Allgemeines

- bei Erzeugen eines Objektes mit `new`
- Speicherplatz wird bereitgestellt
- Variablen initialisiert
- Regeln bei Programmierung
- Name Konstruktor = Name Klasse
- Konstruktor liefert kein Ergebnis zurück, kein Rückgabetyp angegeben
- Konstruktoren sind keine Methoden
- Schlüsselwort `this`
- bei Doppeldeutigkeiten `this.name` kennzeichnet dann Instanzvariable
- `this()` ruft Konstruktor auf

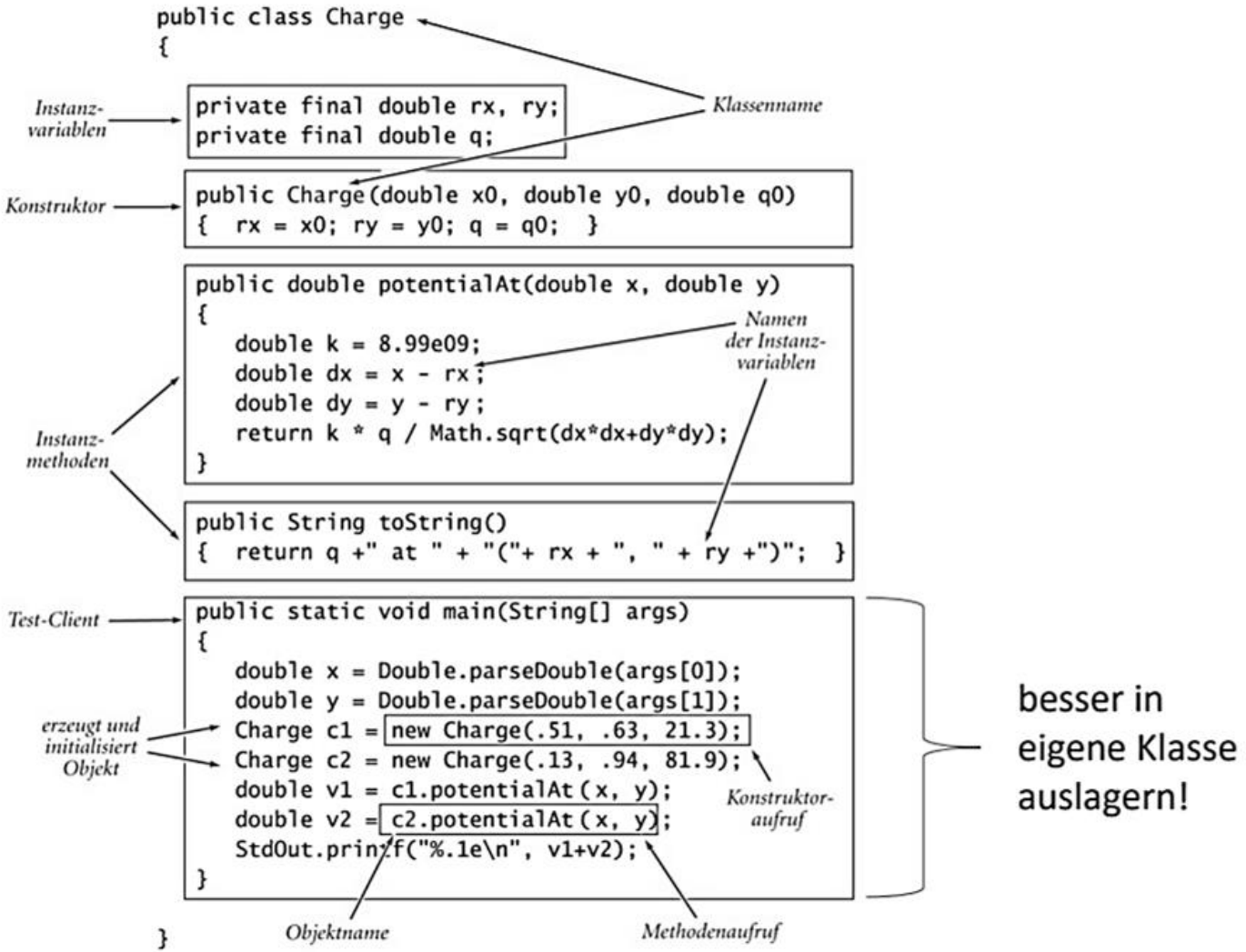
Quelle Abbildung: Sedgewick & Wayne, 2011.

Anatomie eines Konstruktors



Quelle Abbildung: Sedgewick & Wayne, 2011.

Klassen-Überblick



Quelle Abbildung: Sedgewick & Wayne, 2011.

INTERNATIONALE HOCHSCHULE

Getter und Setter

- Instanzvariablen sollen vor direktem Zugriff geschützt sein
- Modifizierer `private`
- Auslesen der Werte mit `getVarname()`
- Verändern der Werte mit `setVarname(neuerWert)`

Quelle Text: Kofler, 2018.

INTERNATIONALE HOCHSCHULE

Modifizierer

- bei Deklaration von
  - Variablen
  - Methoden
  - weiteren Elementen
- geben an auf welcher Gültigkeitsebene Elemente (von außen) genutzt werden können
- innerhalb Klasse immer zugänglich
- vorerst zwei Fälle: `public` und `private`

Quelle Text: Kofler, 2018.

Vererbung

Bedeutung und Zielsetzung

- Redundanz im Code soll vermieden werden
- vorhandener Code soll universell wiederverwendbar sein
- Vererbung bedeutet, auf vorhandene Klasse aufzubauen

Verwendung im Code

- `class Neu extends Alt {`

  `//...`

  `}`
- `Alt`: Basisklasse
- `Neu`: abgeleitete oder erweiterte Klasse, Unterklasse, Kindklasse, Subklasse
- In Java nur eine Basisklasse für eine abgeleitete Klasse möglich
- Modifizierer `protected`, wenn Zugriff aus abgeleiteten Klassen möglich sein soll (anstatt `private`)

Quelle Text: Kofler, 2018.

Methoden und Variablen

- Methoden überschreiben
  - Sichtbarkeit darf nicht eingeschränkt werden
  - optionale Annotation `@Override` veranlasst Compiler zur Prüfung (z. B. Parameterliste)
- Variablen der Basisklasse werden versteckt (nicht überschrieben!), wenn Variablen in abgeleiteter Klasse neu definiert werden
- Schlüsselwort `final` bedeutet:
  - bei Klasse → kann nicht vererbt werden
  - bei Methode → kann nicht überschrieben werden

Quelle Text: Kofler, 2018.

Konstruktoren

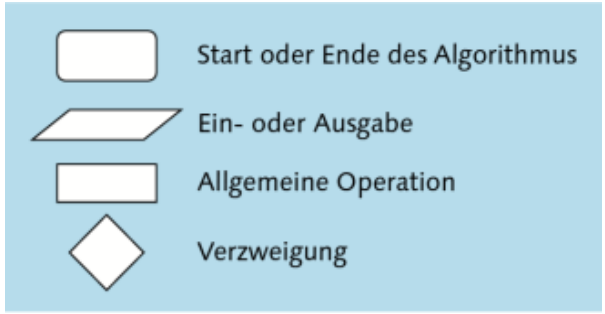
- Konstruktoren werden nicht vererbt
- als **erste Anweisung** kann Konstruktor der Basisklasse aufgerufen werden `super(...)`
- Compiler setzt `super()` ein, …
  - wenn `super(...)` nicht vorhanden und kein anderer Konstruktor der abgeleiteten Klasse aufgerufen wird
  - oder überhaupt kein expliziter Konstruktor vorhanden ist → Fehler, falls Basisklasse keinen parameterlosen Konstruktor hat

Quelle Text: Kofler, 2018.

Offenes

- Generics
- Interfaces

Flussdiagramm



Problem:

Berechne den Quotienten zweier natürlicher Zahlen!

Anfangsdaten:

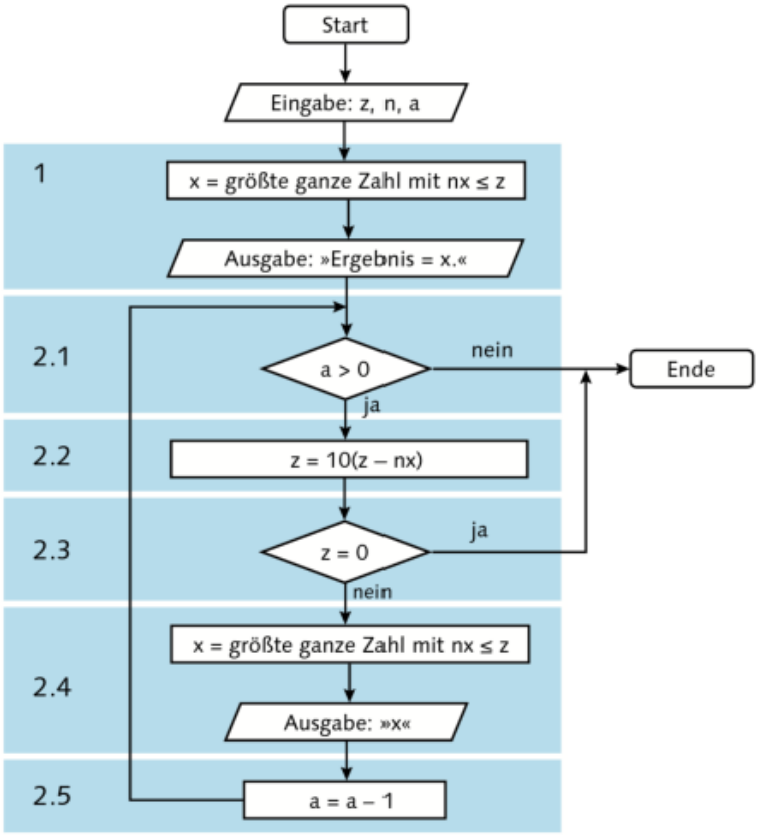
z = Zähler ($z \geq 0$)

n = Nenner ($n > 0$)

a = Anzahl der zu berechnenden Nachkommastellen[3]

Anweisungen:

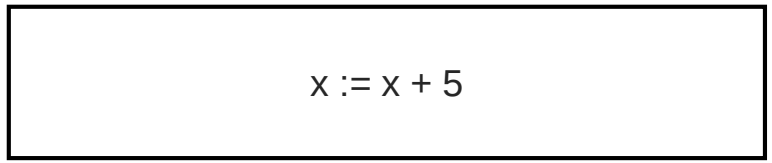
1. Bestimme die größte ganze Zahl x mit $nx \leq z$! Dies ist der Vorkomma-Anteil der gesuchten Zahl.
2. Zur Bestimmung der Nachkommastellen fahre wie folgt fort:
   - 2.1 Sind noch Nachkommastellen zu berechnen (d. h. $a > 0$)? Wenn nein, dann beende das Verfahren!
   - 2.2 Setze $z = 10(z-nx)$!
   - 2.3 Ist $z = 0$, beende das Verfahren!
   - 2.4 Bestimme die größte ganze Zahl x mit $nx \leq z$! Dies ist die nächste Ziffer.
   - 2.5 Jetzt ist eine Ziffer weniger zu bestimmen. Vermindere also den Wert von a um 1, und fahre anschließend bei 2.1 fort!

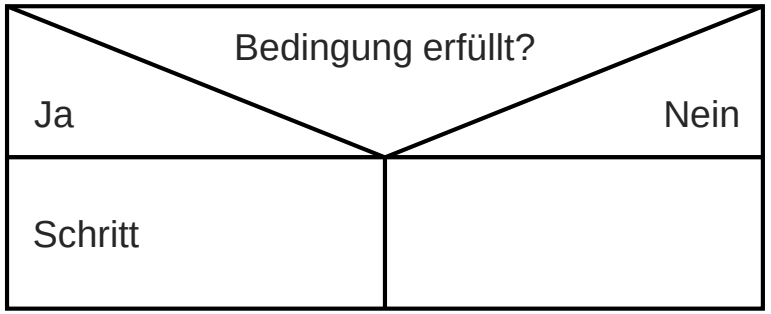


Kaiser, U., & Guddat, M. (2014). *C/C++: Das umfassende Handbuch* (5., aktualisierte und überarb. Aufl). Galileo Press. S. 25ff.

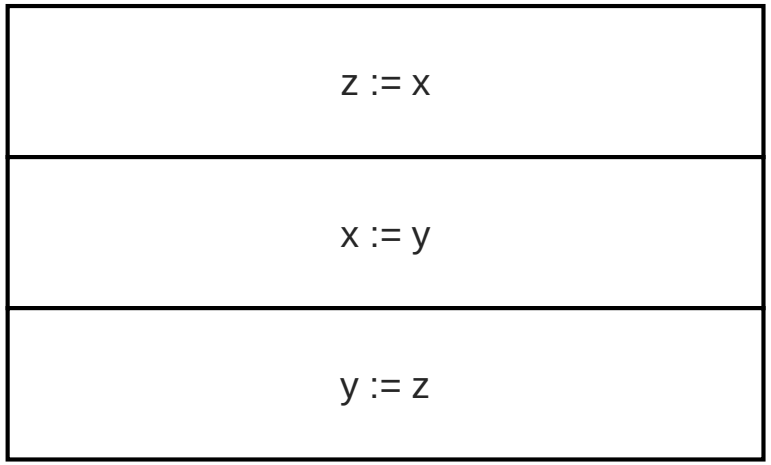
Stuktogramm (Nassi-Shneidermann)

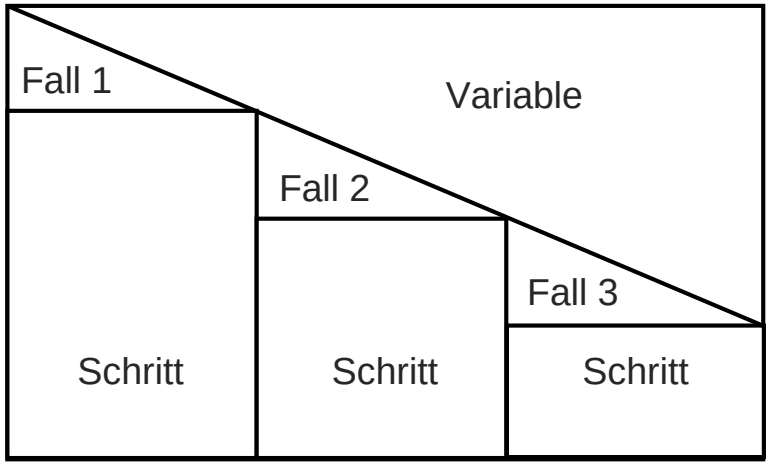
- Wertzuweisung





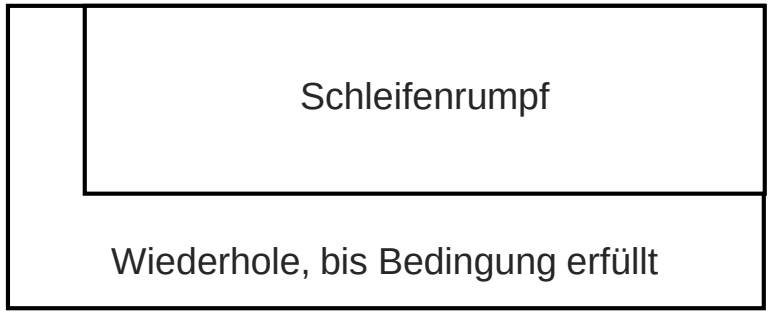
- Folge (Sequenz)



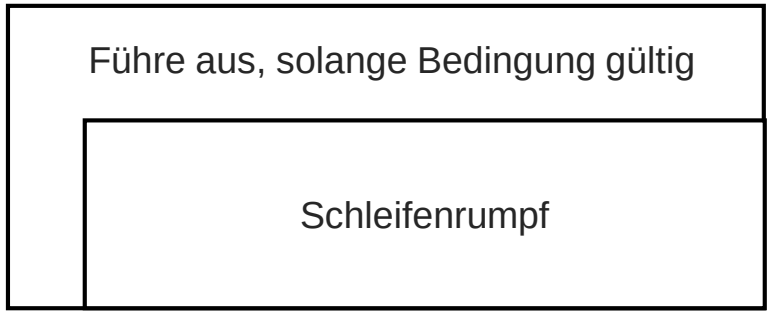


Stuktogramm (Nassi-Shneidermann)

- Wiederholung (Iteration) mit Test der Laufbedingung am Ende

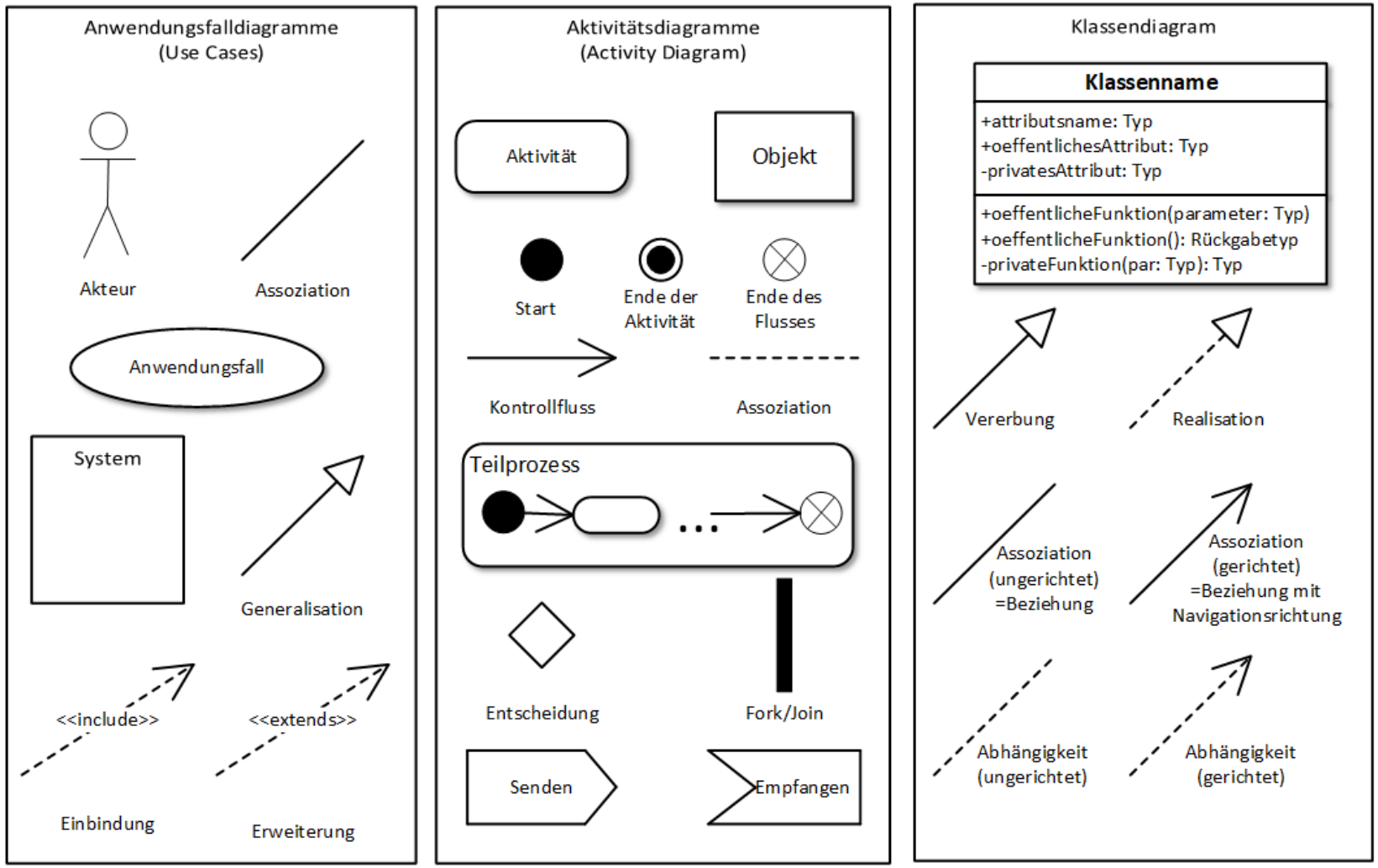


- mit Test der Laufbedingung am Anfang

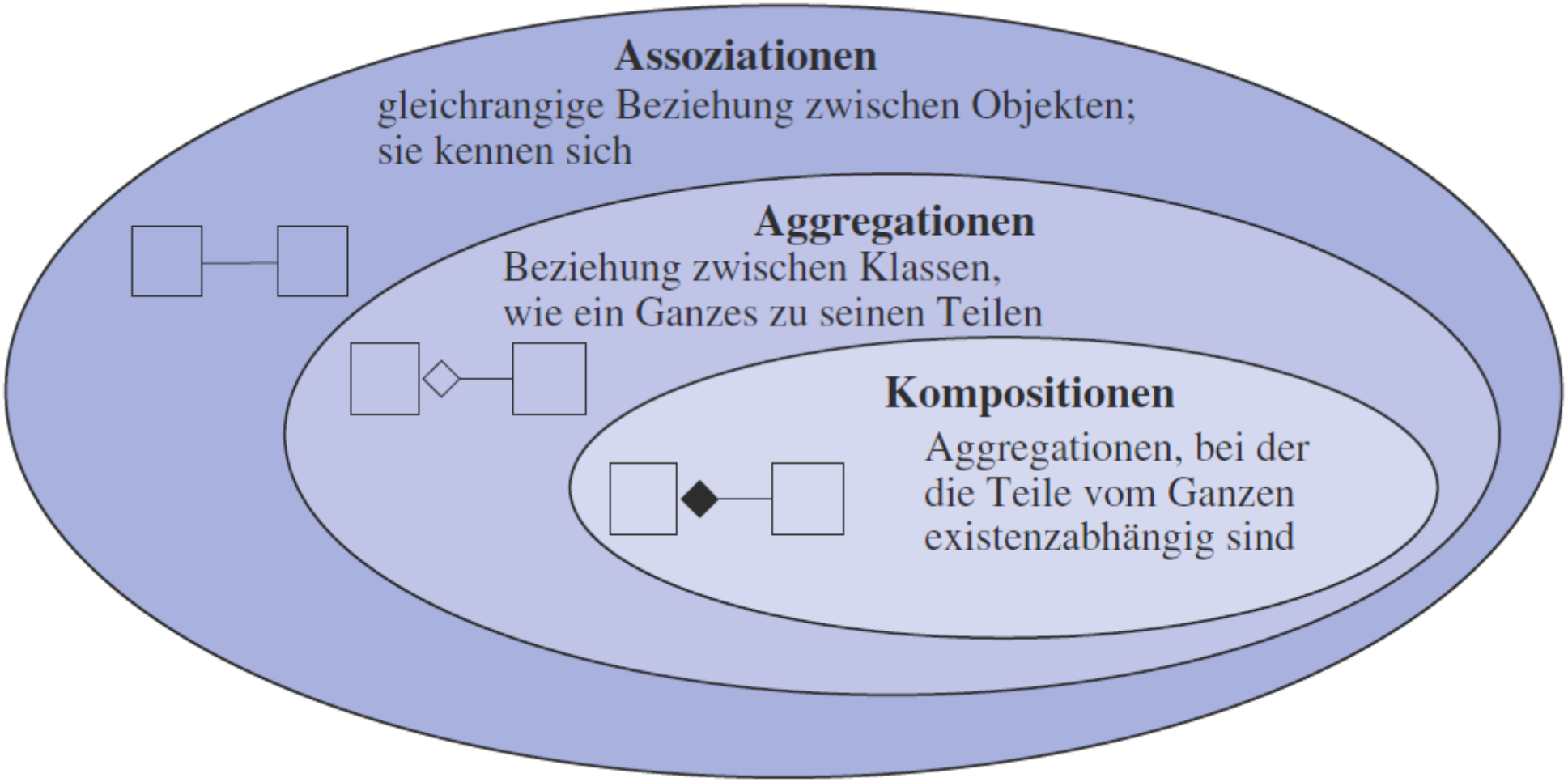


UML

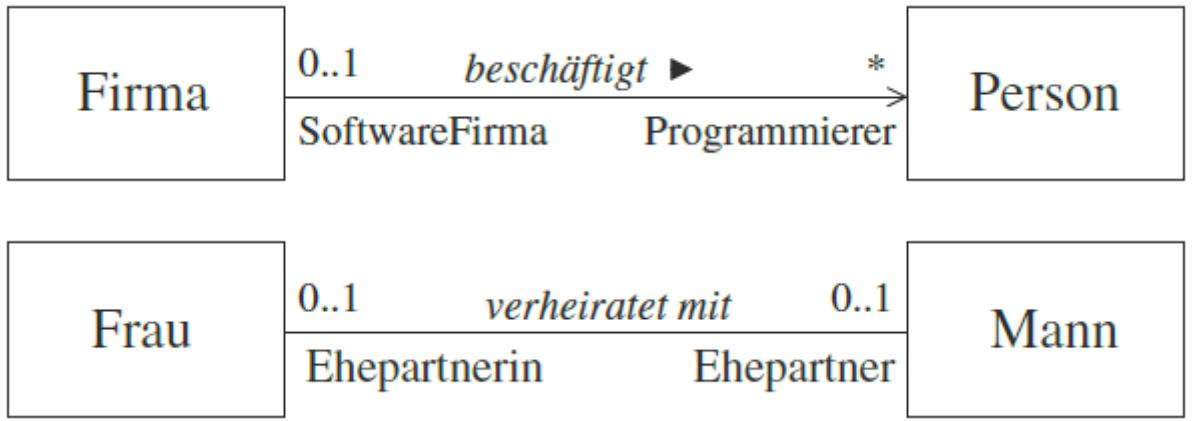
Übersicht



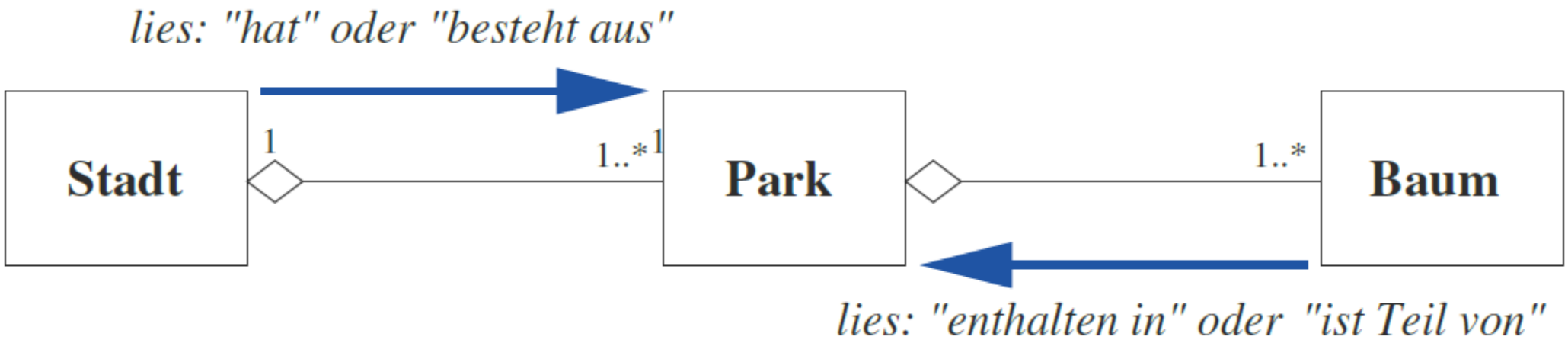
UML (statisch)



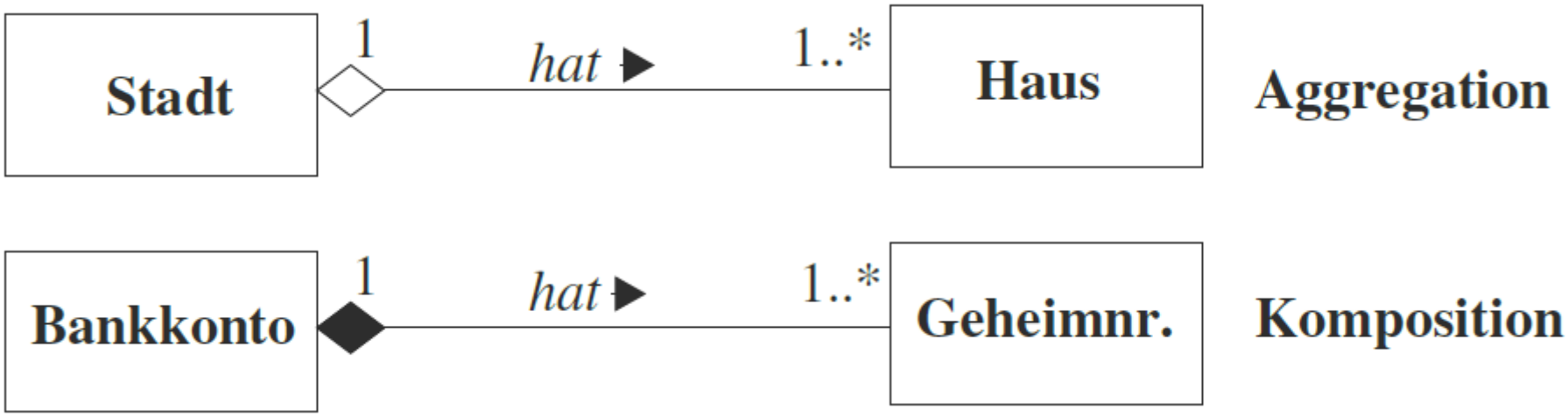
UML (statisch)



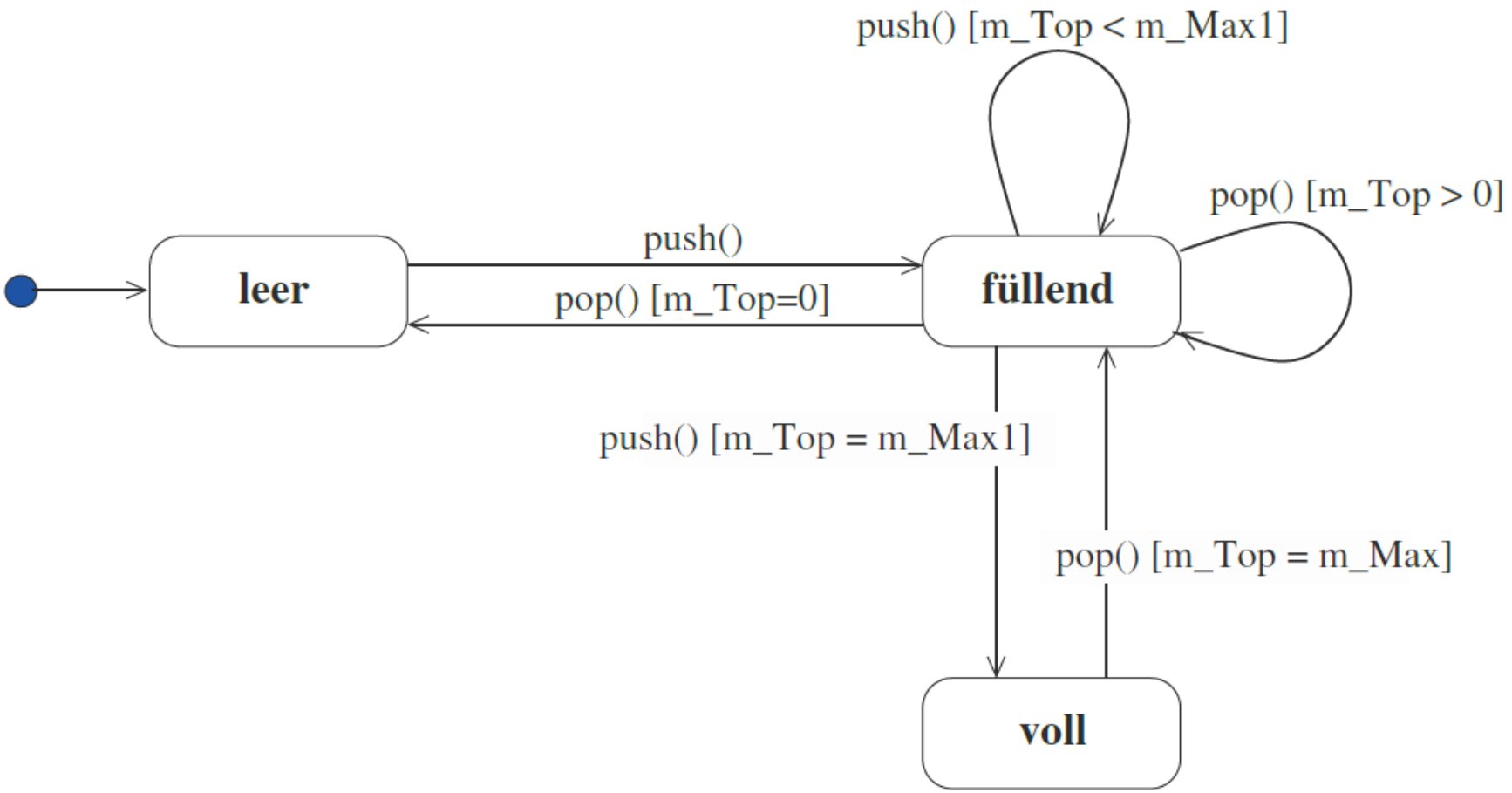
Aggregation (Hat-Beziehung)



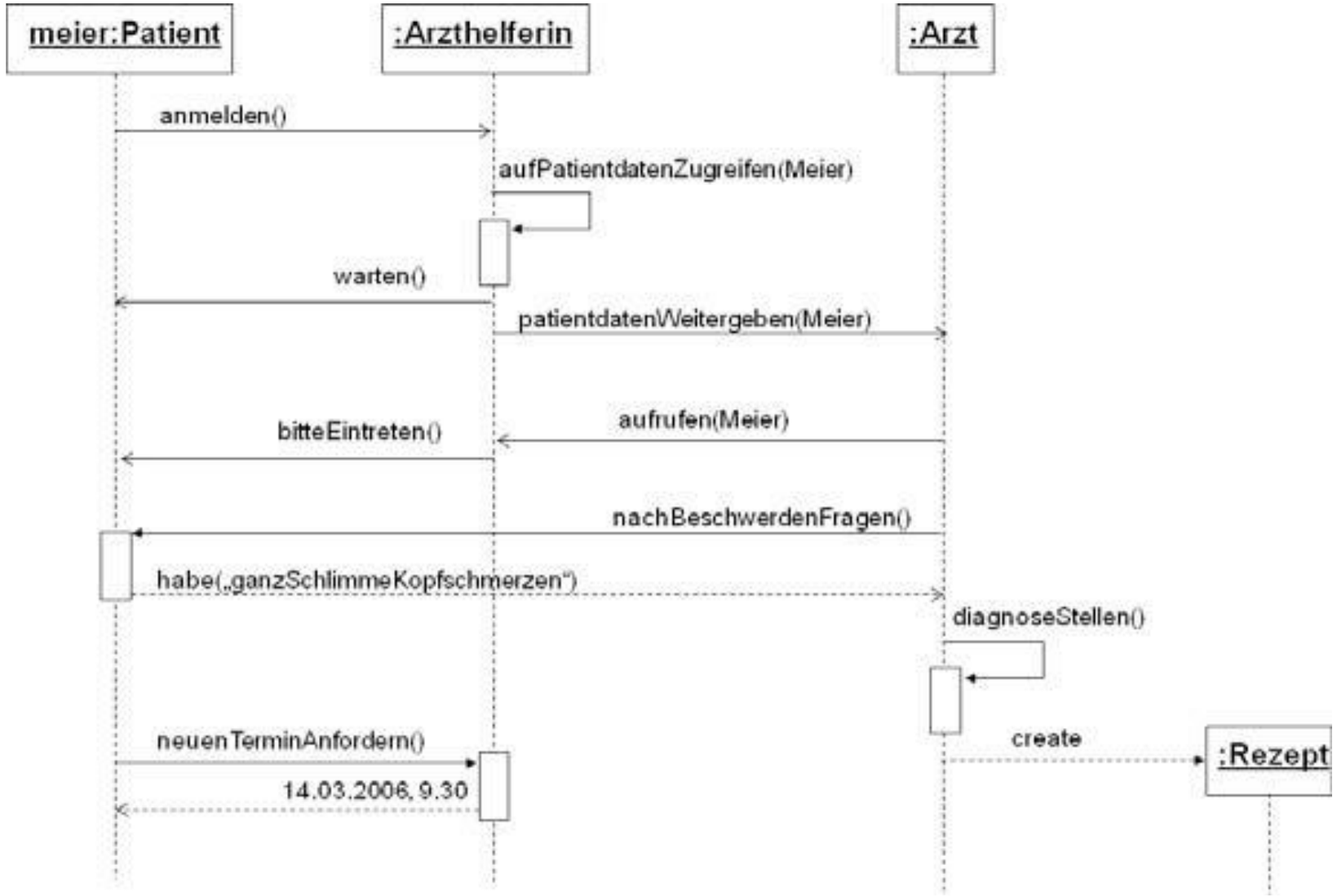
Aggration/Sonderform:Komposition



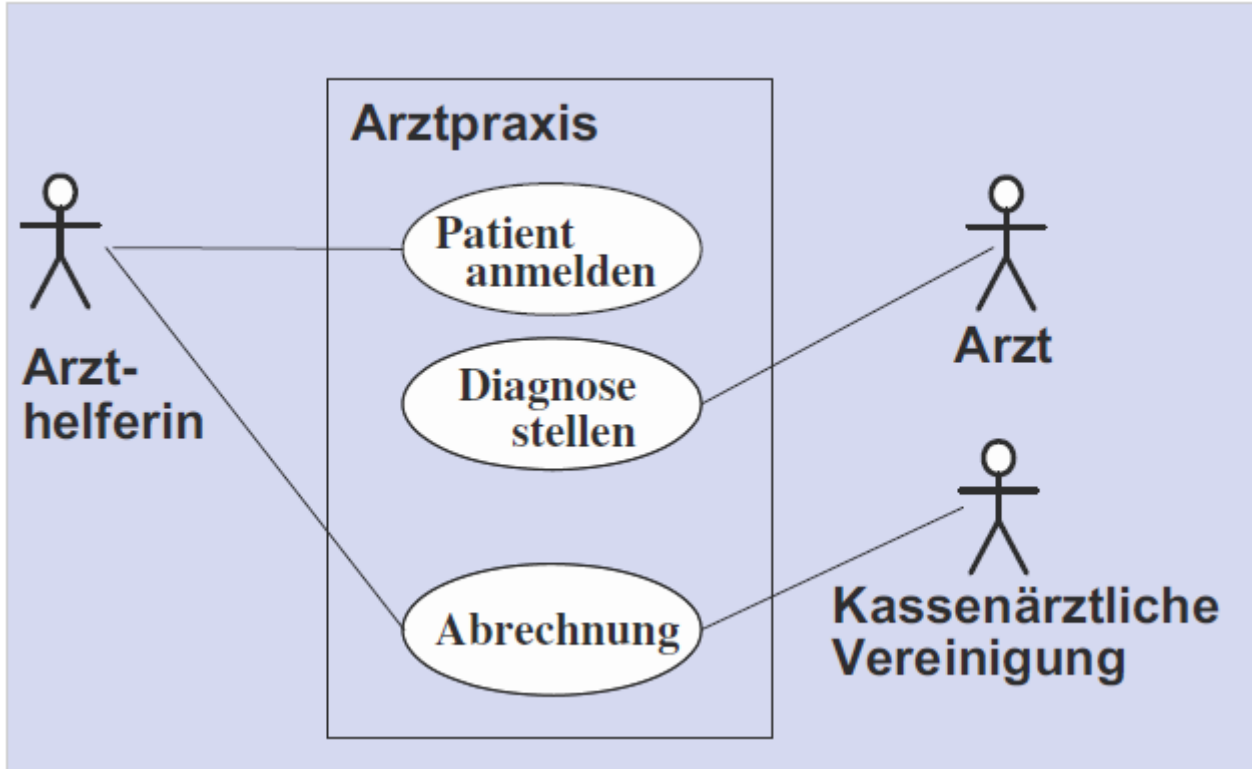
UML (dynamisch)



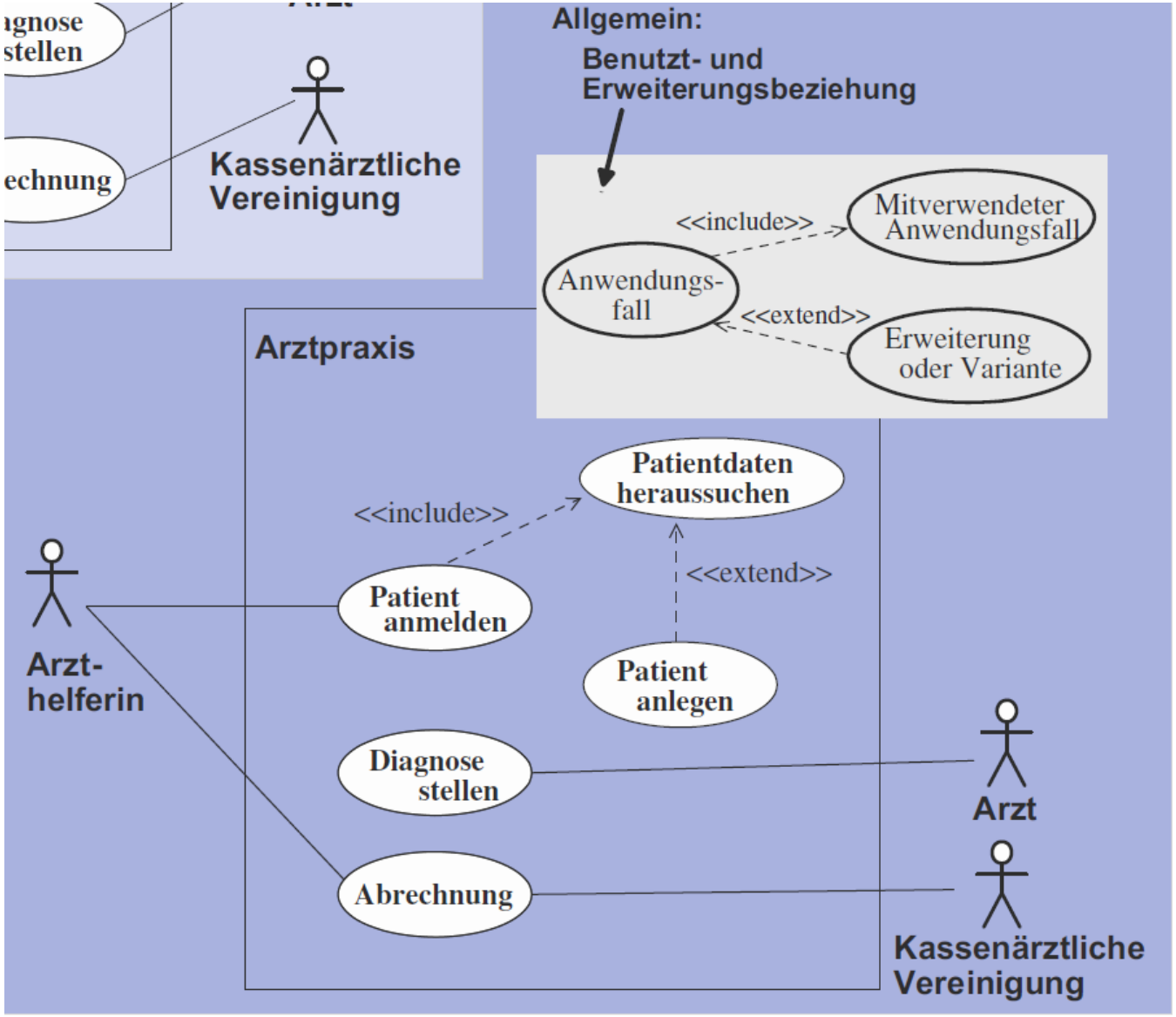
Sequenzdiagramm



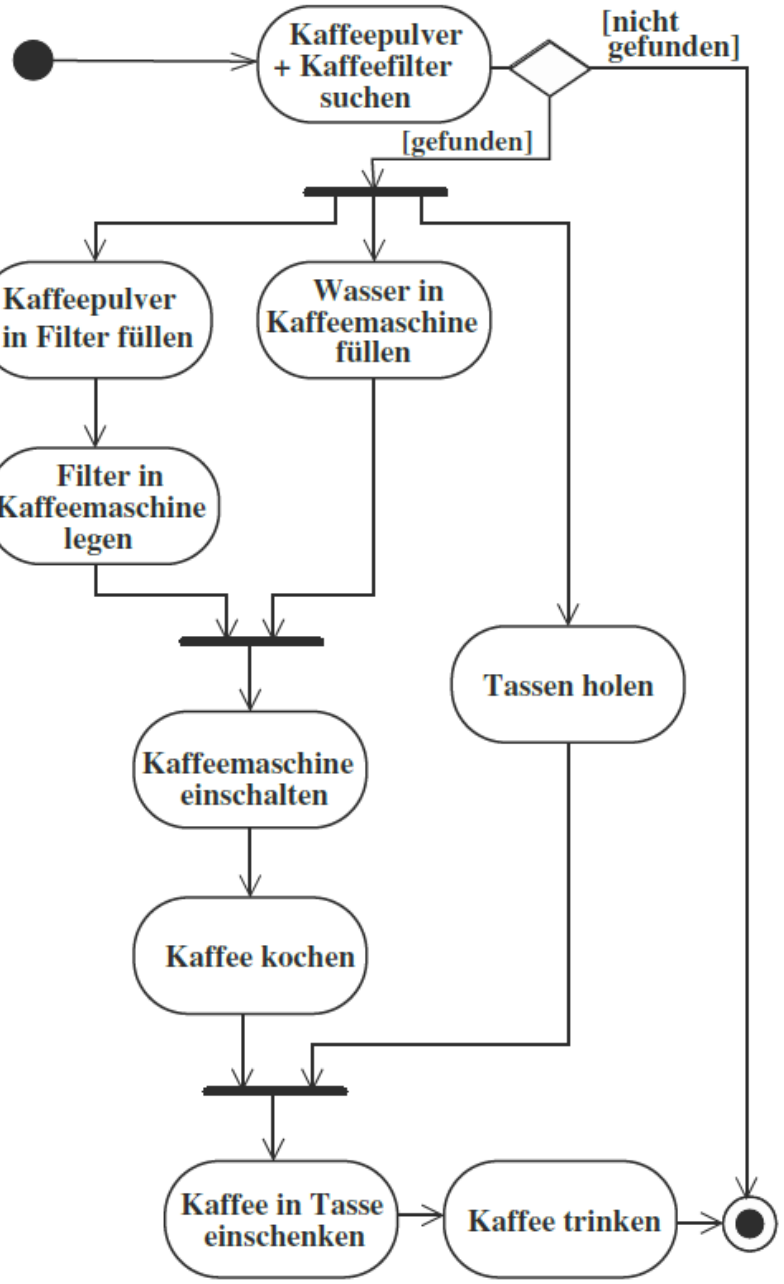
Anwendungsfall/Use Case Diagramm



Anwendungsfall/Use Case Diagramm



Aktivitätsdiagramm



Aktivitätsdiagramm