03. INSTANZEN, KLASSEN, PAKETE

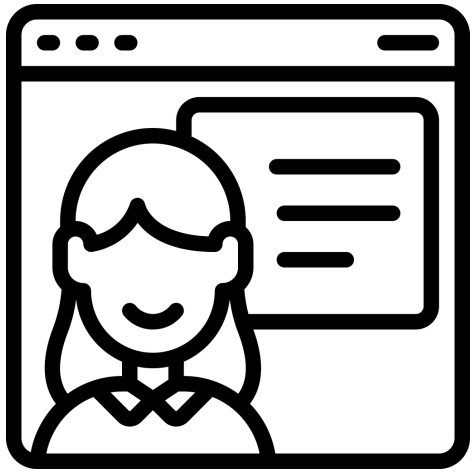
B2.1 - Angewandte Programmierung

WAHL DER SEMESTER-SPRECHER*INNEN

- Zwei Studierende pro Zug
- Wahl für ein Semester
- Aufgaben:
  - Ansprechpartner für Ihre Kommilitonen, um **anonym** Kritik oder konstruktive Anregungen an uns weiterzugeben
  - Multiplikatoren für Professor*innen in das Semester
  - Feedback-Termin am Ende des Semester (mit Kaffee und Kuchen) um Studiengang zu Lehrveranstaltungen zu berichten

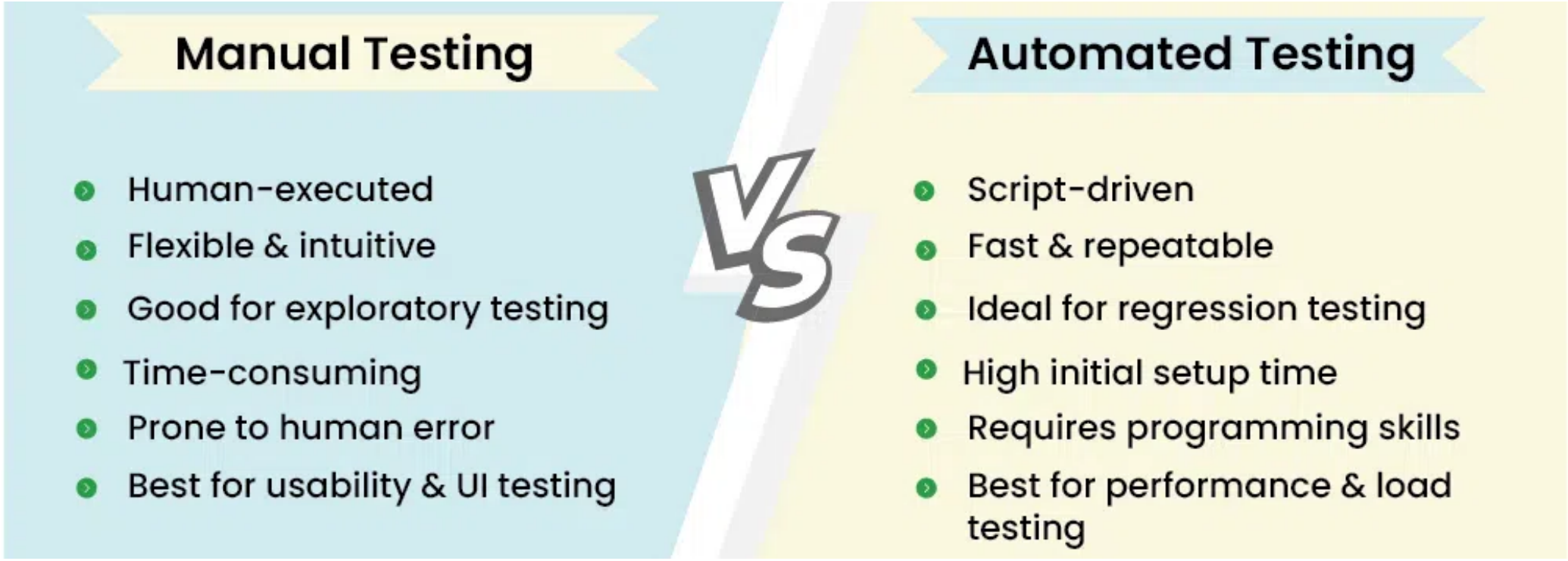
TUTORIEN

- Ihre Tutoren:
  - Jim Feser
  - Nikolaos Pazartziklis
- Tutorium immer dienstags **15:30 – 17:00 Uhr** in Präsenz in Raum A029
- Außerdem über Slack erreichbar, Fragen am besten erstmal für alle sichtbar im `#prog2-general`



WIEDERHOLUNG

- Modelle für Softwareentwicklungs-Projekte
- Phasen und TDD
- Build-Prozess und Build-Tools

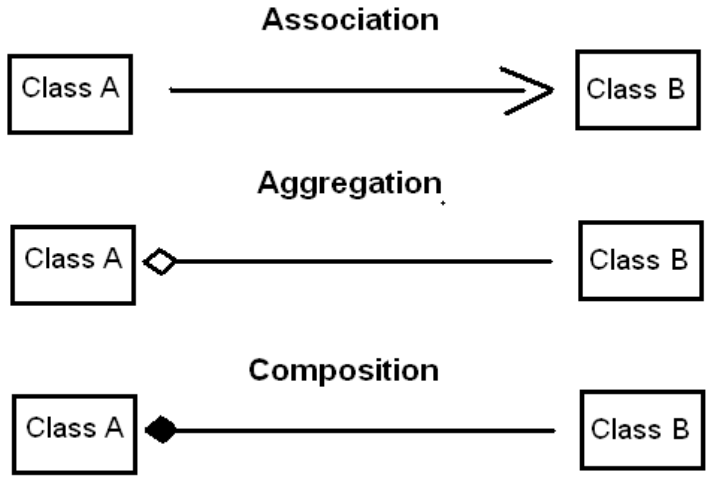


INHALT

- Namensräume (Scope)
- Sichtbarkeiten
- Instanziierung & Laufzeitstruktur
- Objektbeziehungen

		public	private	protected	default
Same Package	Class	YES	YES	YES	YES
	Sub class	YES	NO	YES	YES
	Non sub class	YES	NO	YES	YES
Different Package	Sub class	YES	NO	YES	NO
	Non sub class	YES	NO	NO	NO

https://usemynotes.com/what-are-access-modifiers-in-java/

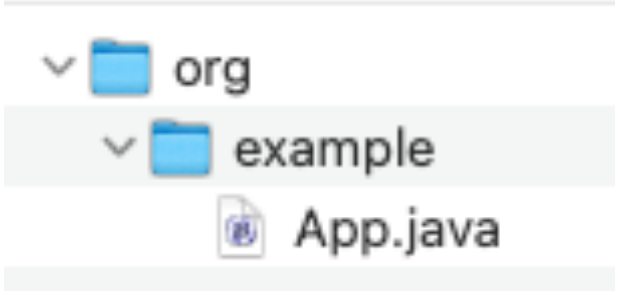


https://softwareengineering.stackexchange.com/questions/235313/uml-class-diagram-notations-differences-between-association-aggregation-and-co

NAMENSRÄUME

- Namensräume in Java sind **Packages**
- Schlüsselwort package gefolgt vom Bezeichner (sog. **Package**)
  - Am Anfang der Quellcode-Datei
  - Legt Namensraum für gesamte Datei fest
  - Package-Bezeichner darf Punkte (`.`), `A-Z`, `a-z` und `_` enthalten
  - Package-Bezeichner wird in Java auf Ordnerstruktur übersetzt

```
package org.example; ->
```



- Bezeichner von Klassen müssen **eindeutig** sein in einem Namensraum

BEISPIEL NAMENSRÄUME

```
01 // Name des Package festlegen für ganze Datei
02 package de.htwberlin;
03
04 // Definition der Klasse System
05 // -> kein Namenskonflikt, da andere Klasse
06 // im Package java.lang existiert
07 public class System {
08
09     public static void main(String[] args) {
10         // Fehler! Eigene Klasse System besitzt
11         // kein Attribut out
12         System.out.println("Hello World!");     !! Syntaxfehler
13
14         // Klasse System über die Notation
15         // mit Package ansprechen um korrekten
16         // Namensraum explizit anzugeben
17         java.lang.System.out.println("Hello World!");
18     }
19
20 }
21
22
23
```

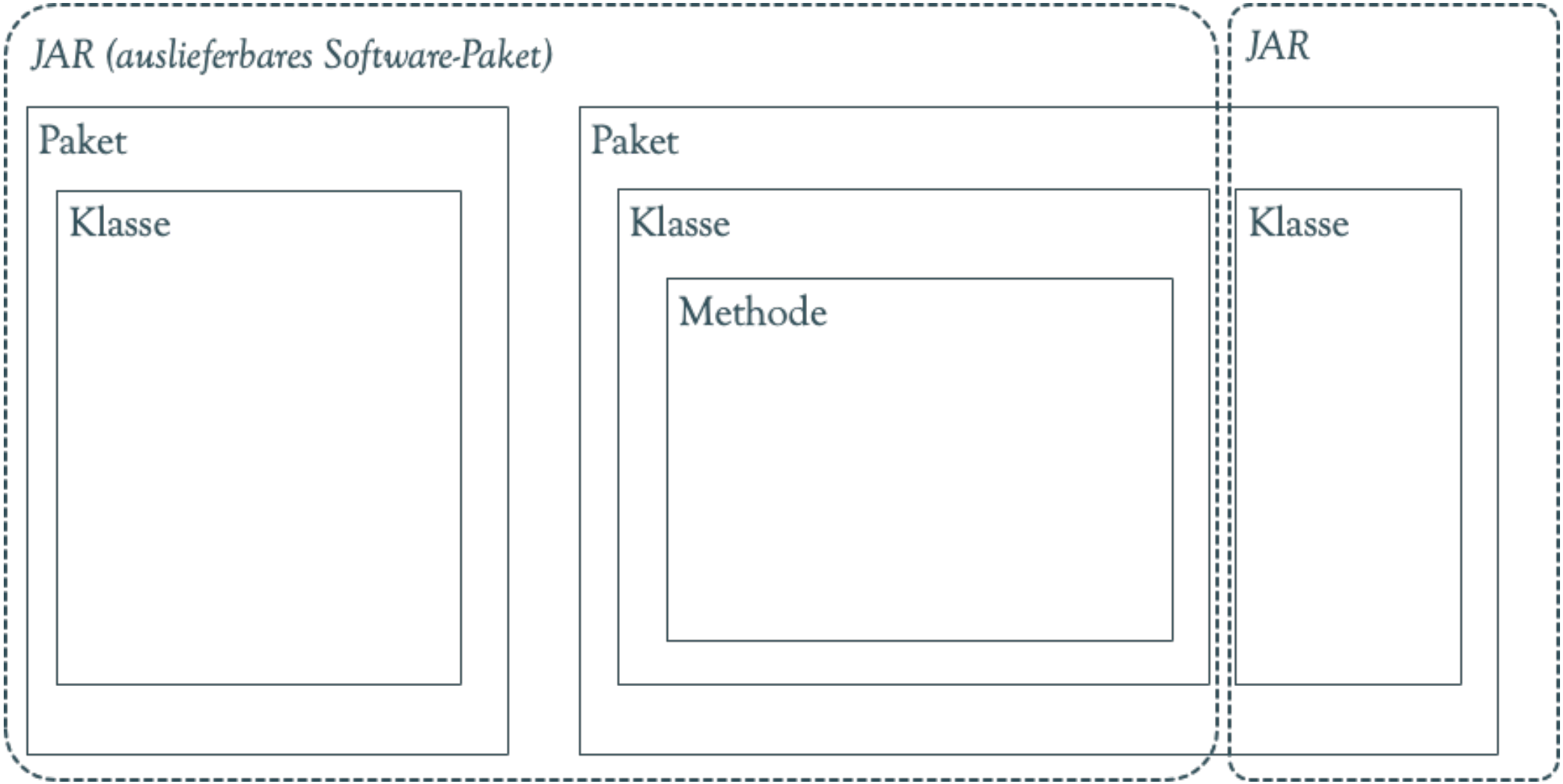
NAMENSRÄUME

- Klassen aus anderen Namensräumen ansprechen mit `.`-Notation
  `<Package-Bezeichner>.<Klassenbezeichner>` z.B.
  `java.lang.System`
- Klasse aus anderem Package kann zur Vereinfachung (und wenn Namenskonflikt besteht) **importiert** werden, um nicht immer vollständig angegeben werden zu müssen:

```
import org.junit.jupiter.api.Test;
```

- Wird kein Package angegeben in einer Datei, so befinden sich die Deklarationen darin im **default**-Package (Package ohne Namen).

NAMENSRÄUME



- Zugriff auf Elemente in Namensraum mit `.`-Operator

METHODEN UND VARIABLEN IN NAMENSRÄUMEN

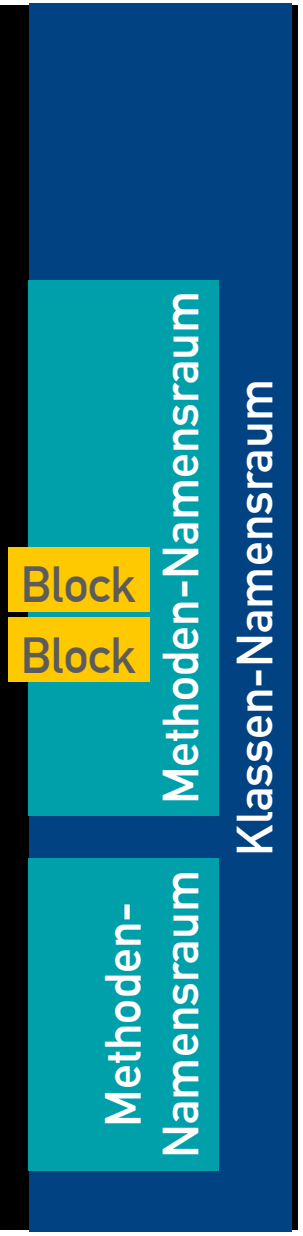
- Methoden befinden sich im Klassen-Namensraum
  - **Signatur** muss **eindeutig** sein (Bezeichner + formale Parameter)

```
void testForHelloWorld() { ... }
void testForHelloWorld(int other) { ... }
```

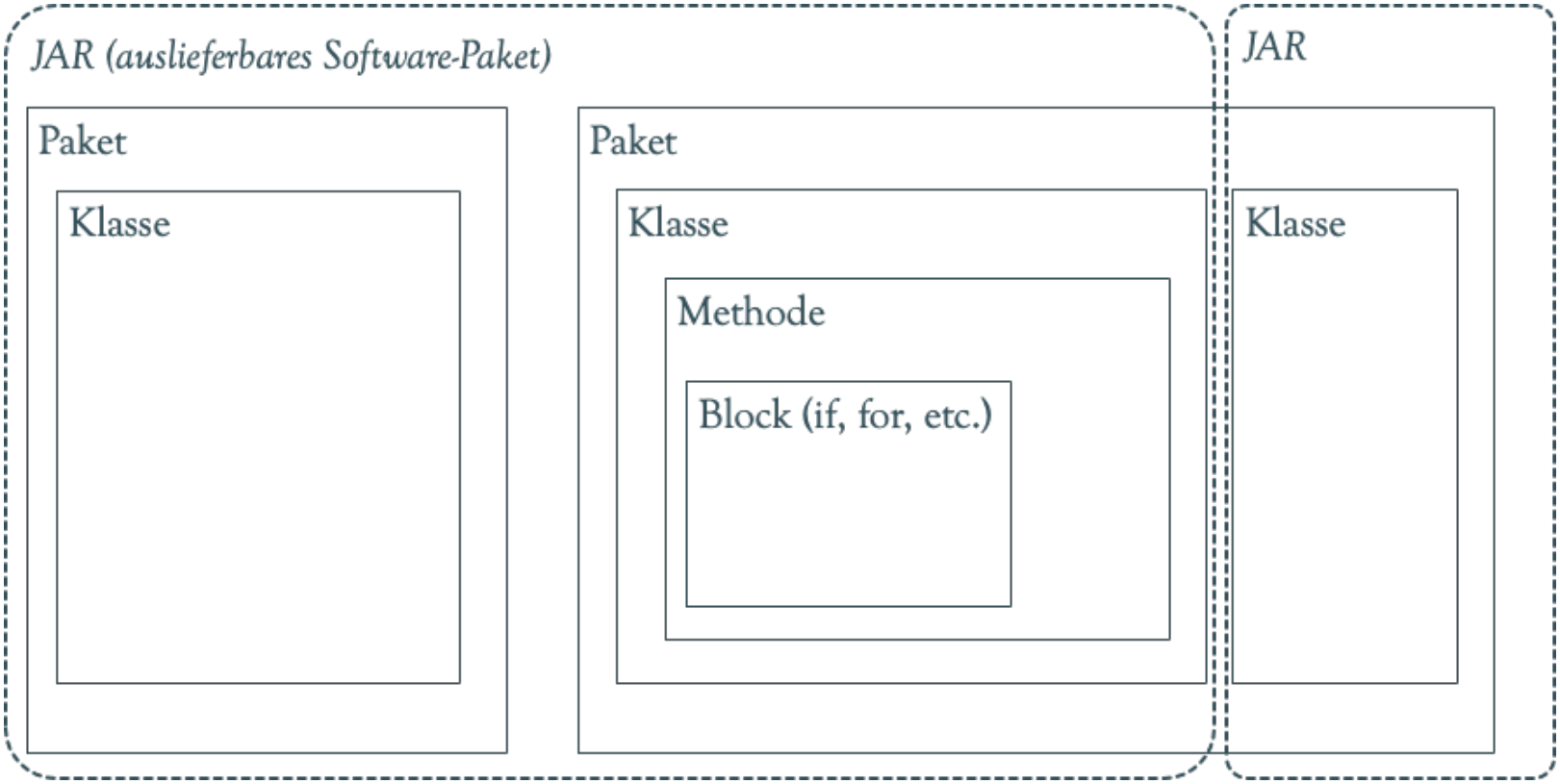
- Variablen befinden sich im
  - Klassen-Namensraum -> Attribute / Klassenvariablen
  - Methoden-Namensraum -> lokale Variablen
  - **Bezeichner** muss im Namensraum **eindeutig** sein

BEISPIEL NAMENSRAUM VS. SCOPE

```java
01 package de.htwberlin;
02
03 public class MyClass {
04     // Deklariert Variable x im Namensraum von MyClass
05     int x = 100;
06
07     int add(int a, int b) {
08         // Deklariert Variable x im Namensraum von add
09         int x = 0;
10
11         // Deklariert Variable i im Namensraum von add
12         for (int i = 0; i < a; i++) { x++; }
13         for (int i = 0; i < b; i++) { x++; }
14
15         return x;
16     }
17
18     static void main(String[] arguments) {
19         // Deklariert eine Variable x Namensraum von main
20         int x = 10;
21         MyClass i = new MyClass();
22         System.out.println("X+X ist: " + i.add(x, x));
23     }
}
```



SCOPES



- Beschreibt Lebenszeit einer Variable -> Variable verlässt Scope wird diese freigegeben
- Variablen im Block- und Methoden-Scope im Methoden-Namensraum

SICHTBARKEITEN

- **Ziel:** Zugriff auf Elemente im Package- und Klassen-Namensraum einschränken
  - Kein Zugriff auf Klasse -> keine Instanzen erzeugen und keine Vererbung
  - Kein Zugriff auf Methode -> kein Aufruf der Methode möglich
  - Kein Zugriff auf Attribut -> kein lesen/schreiben des Attribut möglich
- Unterstützte Sichtbarkeiten:
  - Für Klassen -> `public` und **default**
  - Für Methoden / Attribute -> `public`, `protected`, **default**, `private`

SICHTBARKEIT VON KLASSEN

```java
// Datei MyPublicClass.java
package de.htwberlin;

// Sichtbarkeit public
public class MyPublicClass { ... }

// Datei MyClass.java
package de.htwberlin;

// Sichtbarkeit default
class MyClass { ... }

// Datei Demo.java
package my.other.package;
import de.htwberlin.*;
class Demo {
  static void main(String[] arguments) {
    // Erlaubt, da MyPublicClass die Sichtbarkeit public besitzt
    MyPublicClass i1 = new MyPublicClass();
    // Syntaxfehler, da nicht sichtbar außerhalb von de.htwberlin
    MyClass i2 = new MyClass();
  }
}
```

SICHTBARKEIT VON METHODEN UND ATTRIBUTEN

- Schlüsselwort wird **vor** die Definition der Methode / des Attribut geschrieben `public int add(int a, int b)`
- Ob Zugriff möglich ist abhängig von Situation:
  - Zugriff aus der gleichen Klasse
  - Zugriff aus einer abgeleiteten Klasse im gleichen Package
  - Zugriff aus einer abgeleiteten Klasse in anderem Package
  - Zugriff von allen anderen Stellen im Code

		public	private	protected	default
Same Package	Class	YES	YES	YES	YES
	Sub class	YES	NO	YES	YES
	Non sub class	YES	NO	YES	YES
Different Package	Sub class	YES	NO	YES	NO
	Non sub class	YES	NO	NO	NO

SICHTBARKEITEN BEST PRACTICES

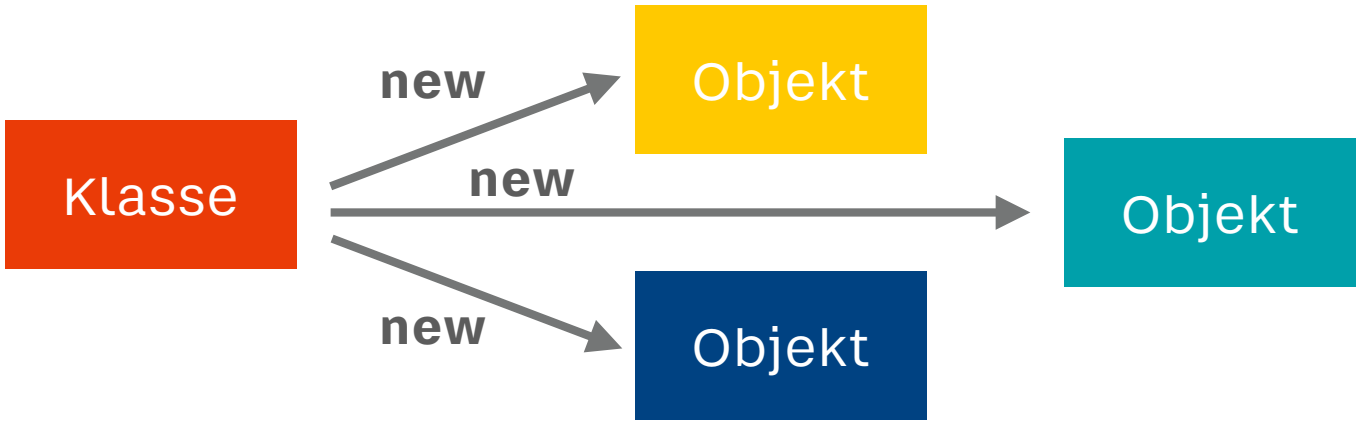
- Attribute immer `protected` oder besser `private`
  - Auslesen von Attributen über Methode (sog. getter)
  - Verändern eines Attributs über Methode (sog. setter)
- `public`-Elemente beschreiben die öffentliche **Schnittstelle** (engl. Interface) Ihrer Anwendung
  - Muss dokumentiert sein
  - Muss getestet sein
  - Soll möglichst stabil sein (keine Veränderungen)

INSTANZIIERUNG

- Beschreibt die Erzeugung eines **Objekts** einer Klasse

- Jedes Objekt besitzt einen **eigenen Zustand**, welcher unabhängig vom Zustand anderer Objekte der gleichen Klasse ist

- **Zustand** beschreibt die Menge aller Attribute eines Objekts

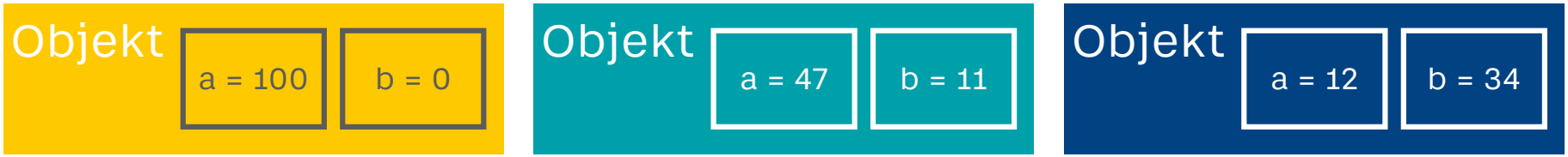


KONSTRUKTOREN

```
01 public class Person {
02     private static List<Person> persons = new ArrayList<>();
03
04     public static void resetNoOfPersons() { persons.clear(); }
05     public static double getAverageAge() {
06         return persons.stream().mapToDouble(Person::getAge).average();
07     }
08
09     public Person(String n, int a) {
10         name = n;
11         age = a;
12         persons.add(this);
13     }
14
15     private String name;
16     private int age;
17
18     public String getName() { return name; }
19     public int getAge() { return age; }
20
21     public void setName(String name) { this.name = name; }
22     public void setAge(int age) { this.age = age; }
23 }
```

INSTANZIIERUNG

- Für jedes Objekt wird eigener **Zustand** im Speicher abgelegt
  Zustand beschreibt die Menge aller Attribute eines Objekts

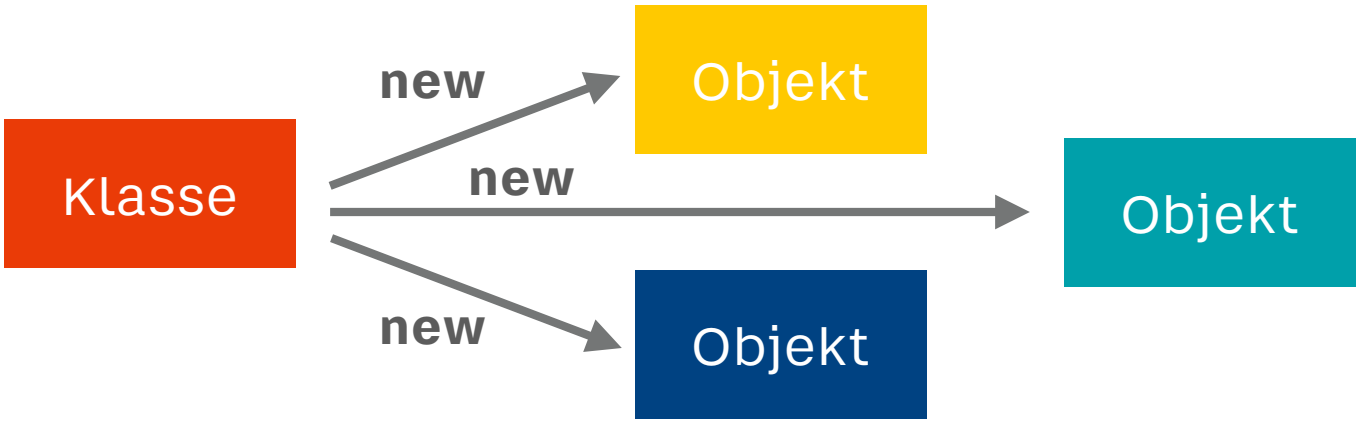


- Die Methoden eines Objekts (**Instanzmethoden**) können auf die Attribute zugreifen, diese verändern und folglich den Zustand verändern.
- Durch den Aufruf der Methode an einem Objekt ist für den Compiler klar, mit welchem Zustand die Methode ausgeführt werden soll.

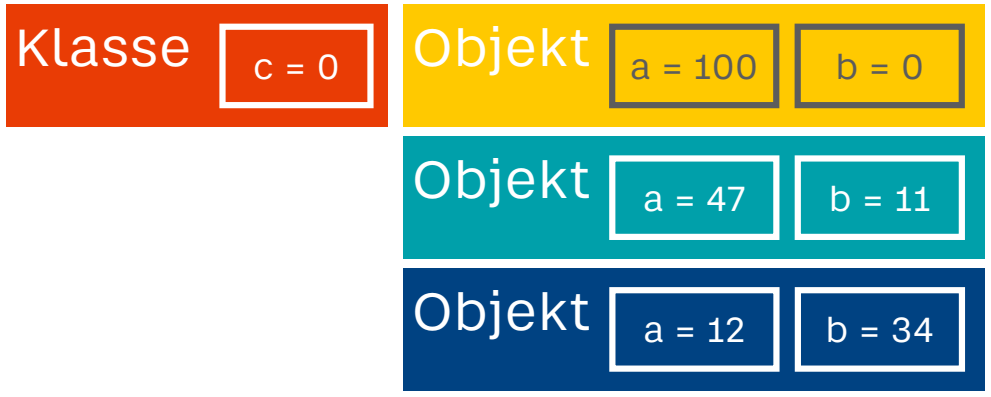
```
MyClass i = new MyClass();
i.add(10, 20);
```

SCHLÜSSELWORT STATIC

- `static` vor Methoden und Variablen macht diese zu
  statischen Methoden bzw. **Klassenmethoden**
  statischen Attributen bzw. **Klassenvariablen / Klassenattributen**




```
public class Klasse {
    int a;
    int b;
    static int c;
}
```

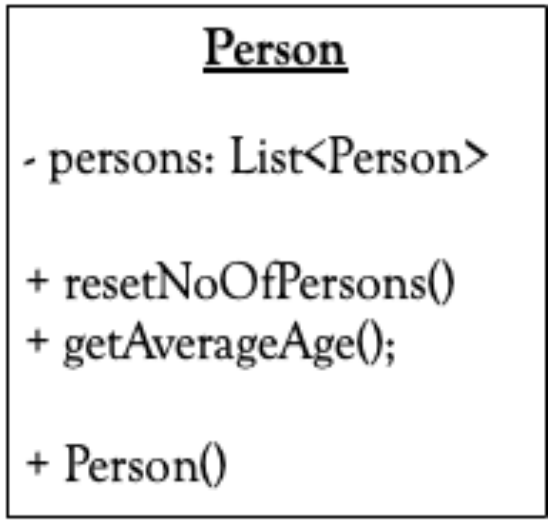


SCHLÜSSELWORT STATIC

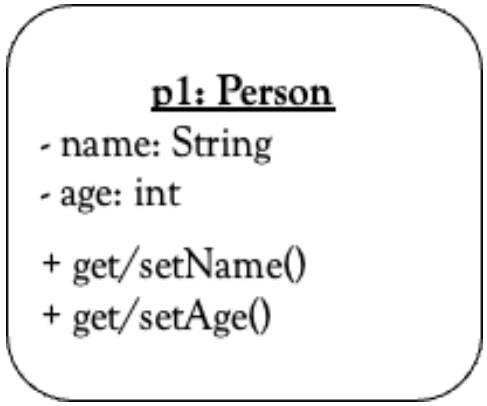
```
01 public class Klasse {
02     int a;
03     int b;
04     static int c;
05 
06     void increaseAll() {
07         a += 1;
08         b += 1;
09         c += 1;
10     }
11 
12     static void increaseC() {
13         c += 1;
14     }
15 
16     public static void main(String[] args) {
17         Klasse objekt = new Klasse();
18         objekt.increaseAll();
19         objekt.increaseC();
20         Klasse.increaseC();
21         System.out.println(objekt.a + " " + objekt.b + " " + objekt.c);
22     }
23 }
```

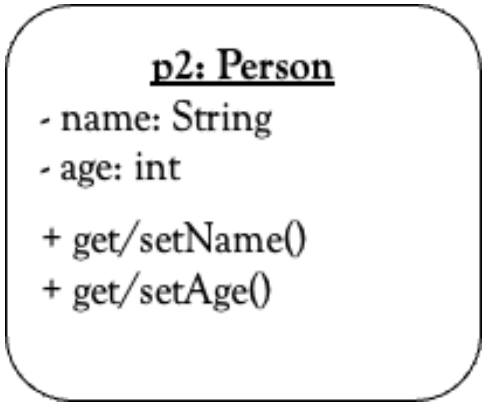
SCHNITTSTELLEN

- Alle `public` Methoden und Attribute zählen zur **Schnittstelle**



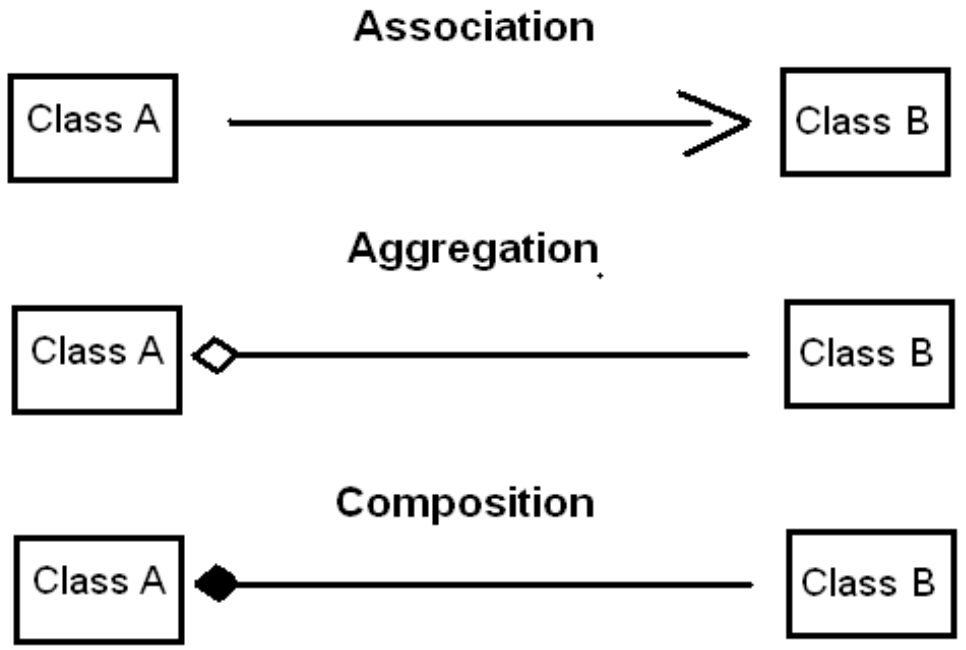

```
var p1 = new Person( n: "Arif",  a: 37);
var p2 = new Person( n: "Doro",  a: 25);
```





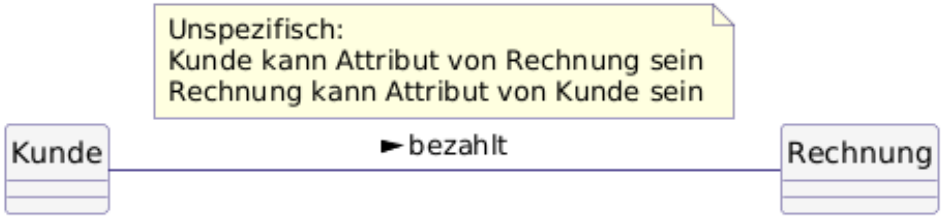
OBJEKTBEZIEHUNGEN

- Klassen und Objekte können in Beziehung zueinander stehen
- Beziehung kann beschrieben und modelliert werden (bspw. mit UML)
- In drei unterschiedlichen Stufen betrachtet
  - Assoziation
  - Aggregation
  - Komposition

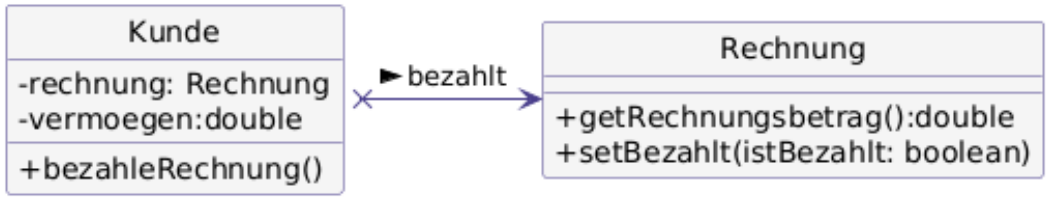


ASSOZIATION

- Zwei Objekte interagieren miteinander ("benutzt ein", "ist zugeordnet zu", "hat eine Beziehung zu")
- Darstellung durch eine Linie mit optionaler Beschreibung

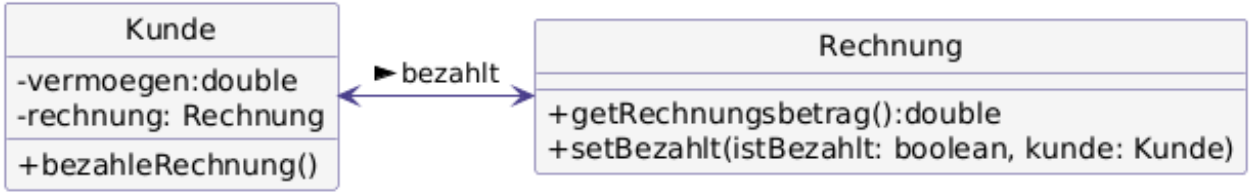


- Richtung der Assoziation (wer greift auf wen zu) kann angegeben werden mittels Pfeilen



Unidirektional

Bidirektional



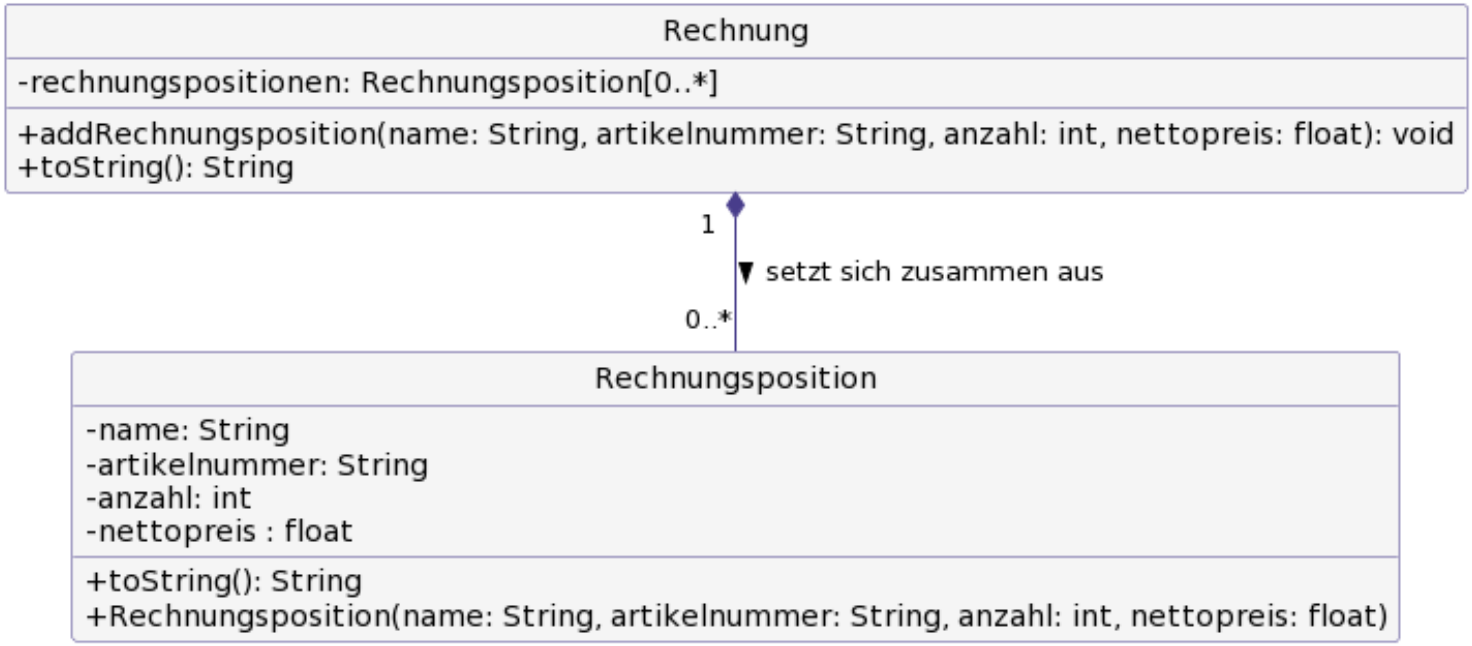
AGGREGATION

- Engere Beziehung als Assoziation -> zeigen eine dauerhafte Teil-Ganzes-Beziehung an ("besteht aus", "hat ein/e", "ist Teil von")
- Instanzen der Teile werden in Attributen des Ganzen verwaltet (per Konstruktor übergeben oder per Setter gesetzt)

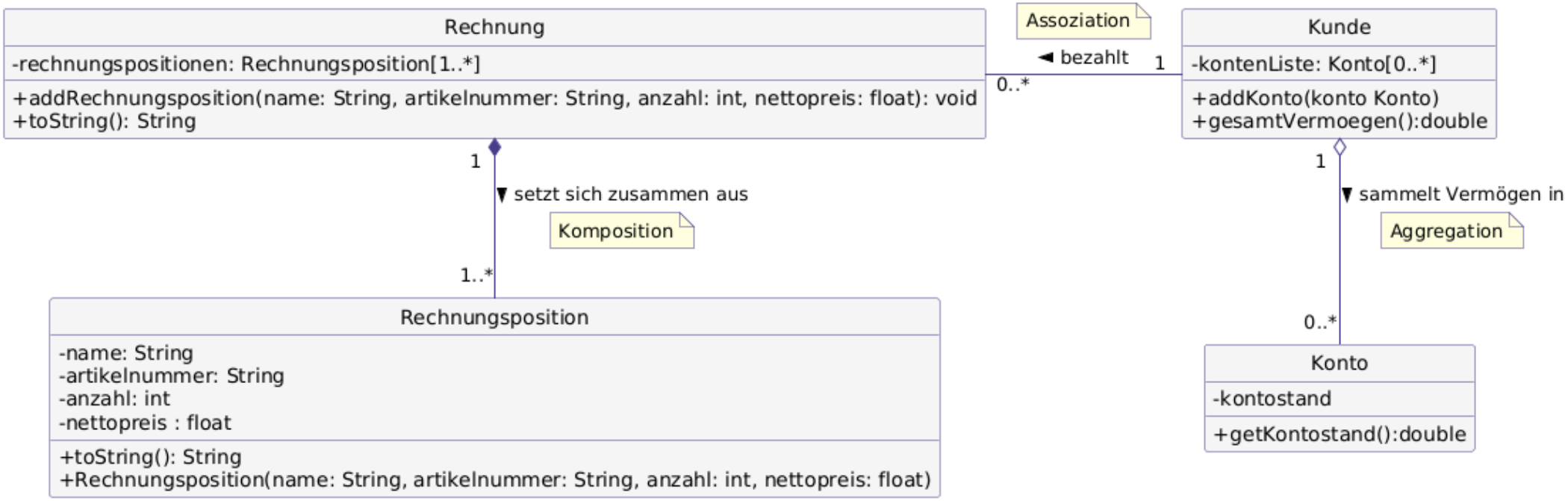
- Notiert als Linie mit Raute an der Seite des "Ganzen"
- Kardinalität kann an der Aggregation notiert werden

KOMPOSITION

- "Starke Form der Aggregation" wird dann eingesetzt, wenn die Teile nicht ohne das Ganze existieren können.
- Notiert als Linie mit ausgefüllter Raute an der Seite des "Ganzen"

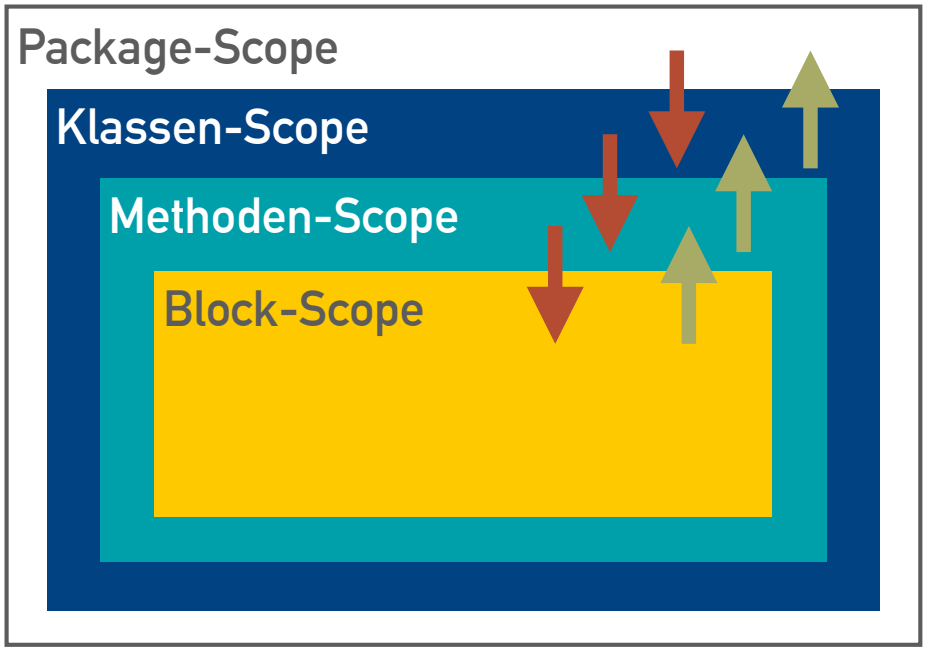


ZUSAMMENFASSUNG ABHÄNGIGKEITEN



ZUSAMMENFASSUNG

- Namensräume
  - Paket
  - Klasse
  - Methode
- Sichtbarkeiten
  - Für Klassen in Paketen
  - Für Methoden und Attribute
- Instanziierung & Laufzeitstruktur
  - Konstruktoren
  - Variablen und Methoden in Klassen und Objekten
- Objektbeziehungen



VIELEN DANK