Beispiel:

Die in einem Fitnessstudio aufgestellten Trainingsgeräte sollen von einem zentralen Rechner aus gesteuert und der Trainingsbetrieb personalisiert überwacht werden. Dazu gehört die Erfassung des Namens, der Körpergröße, des Gewichts und des BMI (Body Mass Index), der sich wie folgt berechnet:

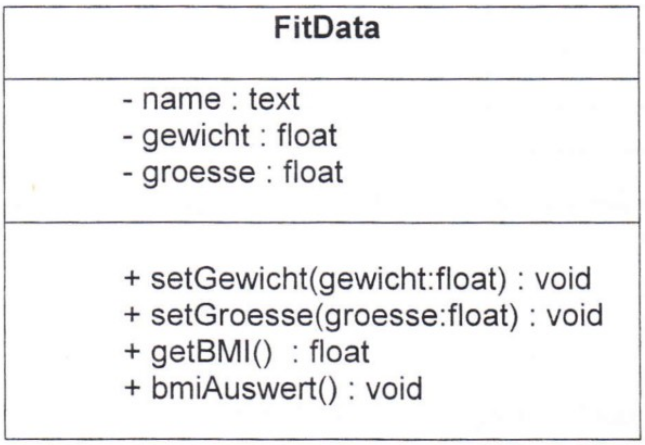
$$BMI = \frac{\text{Körpergewicht}}{\text{Körpergröße}^2} \qquad \left[\frac{kg}{m^2}\right]$$

In der objektorientierten Programmierung sind die Objekte das Wichtigste.
In dem Programm wird jeder Sportler als **Objekt** abgebildet.
Es werden Variablen definiert, in denen die Werte für seine **Eigenschaften** gespeichert werden.
Es werden **Methoden** definiert, mit denen die Werte verändert bzw. die Objekte bearbeitet werden können.

Für diese Objekte muss zunächst ein Bauplan (**Klasse**) → erstellt werden, in dem die Eigenschaften → der Objekte und die

Methoden → für die Objekte definiert werden →

Klassendiagramm



Beschreibung der Methoden:

Methode	Arbeitsweise	Rückgabetyp
setGewicht(gewicht:float)	Setmethode zum Setzen des Attributs Gewicht	void
setGroesse(groesse:float)	Setmethode zum Setzen des Attributs Groesse	void
getBMI()	Methodenaufruf berechnet und liefert den BMI zurück	float
bmiAuswert()	Bewertet einen BMI und gibt Verhaltenshinweise in Textform aus	void

Erst wenn die Klasse (Grundgerüst / Bauplan für die Objekte) fertig ist, werden die Objekte konstruiert (erst dann können sie bearbeitet / programmiert werden):

FitData beate=new FitData();	*FitData karl=new FitData();*	*FitData ernst=new FitData();*
 name=NULL gewicht=NULL groesse=NULL	 name=NULL gewicht=NULL groesse=NULL	 name=NULL gewicht=NULL groesse=NULL
beate.setGroesse(1.67); *beate.setGewicht(67.5);* *beate.setname("Beate");*	*karl.setGroesse(1.77);* *karl.setGewicht(78.5);* *karl.setname("Karl");*	*ernst.setGroesse(1.89);* *ernst.setGewicht(87.3);* *ernst.setname("Ernst");*
 Name='Beate' Gewicht=67.5 Groesse=1.67	 Name='Karl' Gewicht=78.5 Groesse=1.77	 Name='Ernst' Gewicht=87.3 Groesse=1.89

Die Methode *setName()* wurde noch nicht definiert:

```
public void setName(String name)
{
this.name=name;
}
```

Methode setName()
Übergabeparameter: name:String Rückgabewert: --
Klassenvariable name = Übergabeparameter name

2.1 Erklären Sie, wie der Wert des Attributs Gewicht geändert werden kann.

2.1.1 Innerhalb der Klasse FitData und
Innerhalb der Klasse kann dem Attribut Gewicht einfach ein Wert zugewiesen werden, z. B. Gewicht = 76.3

2.1.2 Außerhalb der Klasse FitData
Aus anderen Klassen heraus ist das nicht möglich (Attribut Gewicht hat den Modifizierer private).
Da geht es nur mit der dazugehörigen Set-Methode:
Objekt. setGewicht(76.3) ;

2.1.3 Wie nennt man dieses Prinzip?
Kapselung → die Werte der Eigenschaften sind an das Objekt gekoppelt.
Vorteile:

- die Werte können vor ungewollter Veränderung geschützt werden
- auch bei tausenden von Objekten sind die Werte für *name, gewicht* und *groesse* von z.B. Beate sofort auffindbar

Die Klasse FitData wird um eine weitere Methode erweitert:

+ xy(BMI : float) : float

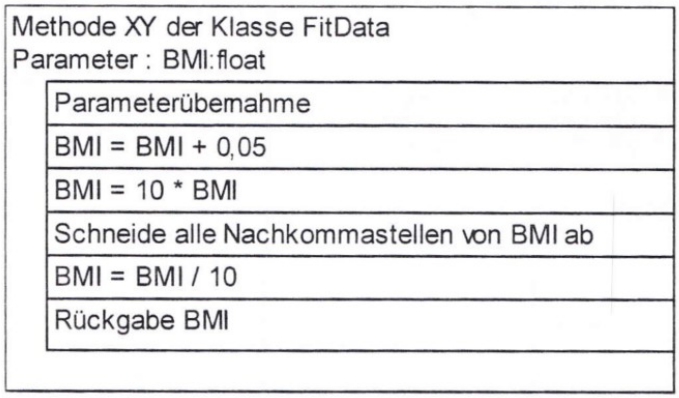
Die Methode XY arbeitet nach folgendem Prinzip (s. neben stehendes Struktogramm):

Ermitteln Sie in einem Schreibtischtest den Wert der Variablen A nach folgendem Methodenaufruf:

A = XY (23.152);

Variable für den Rückgabewert

Übergabeparameter der Methode XY()



2.2 Die Methode *bmiAuswert()* soll nun unter Berücksichtigung folgender Leistungsmerkmale entwickelt werden:

- Gewicht und Größe haben die Einheiten Meter und Kilogramm
- Der BMI soll ausgegeben werden
- Bei einem BMI unter 19 soll eine Textausgabe ("Sie müssen mehr essen!") erfolgen
- ebenso bei einem BMI über 25 ("Sie sollten weniger essen!")
- Bei BMI-Werten dazwischen soll die Ausgabe "Ihr BMI ist in Ordnung" erfolgen

Erstellen Sie dazu ein Struktogramm.

Hinweis: die Methode wird später so aufgerufen: *Objekt.bmiAuswert();*
wird der BMI für Beate untersucht: *beate.bmiAuswert();*

in der Methode *bmiAuswert()* kann der BMI-Wert zuerst über die oben gegebene Formel berechnet werden, oder es kann die Methode *getBMI()* benutzt werden:

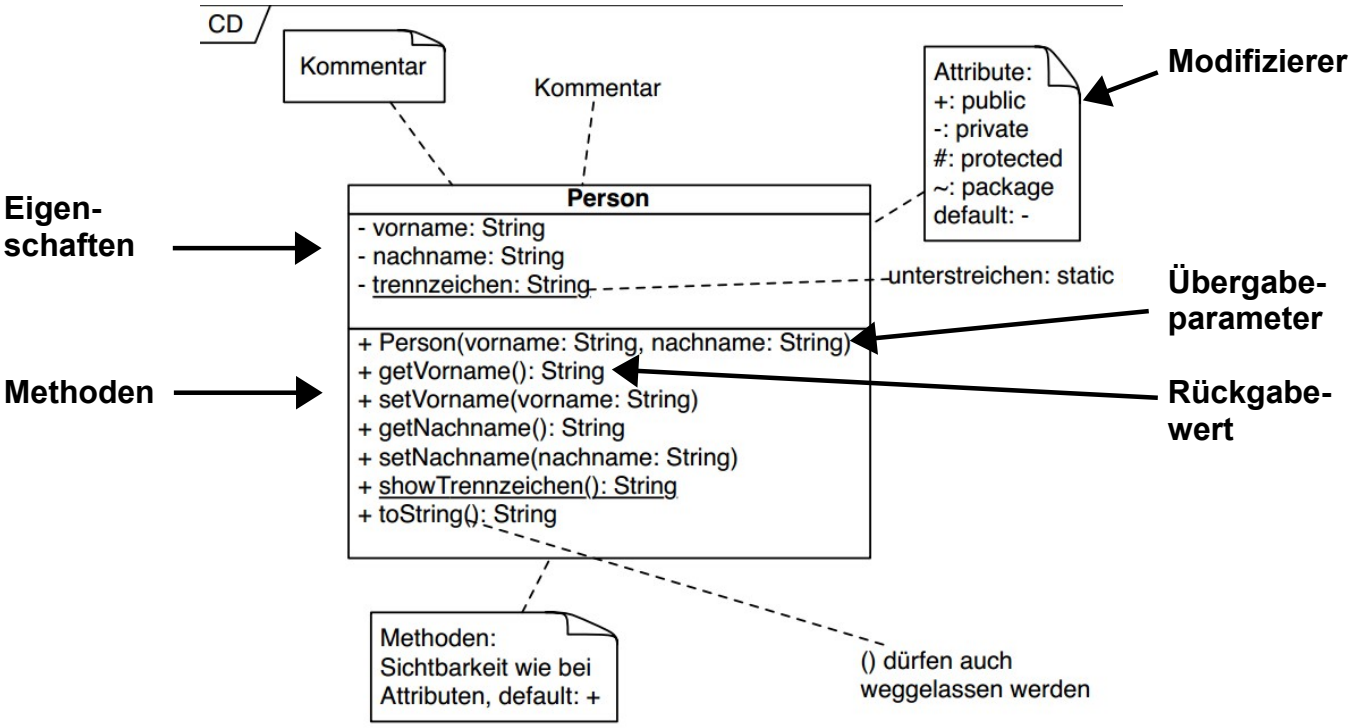
float bmi = beate.getBMI();

→ hier wird der BMI vom Objekt 'beate' berechnet; der Rückgabewert der Methode (der berechnete BMI) wird in der Variablen 'bmi' gespeichert.

BMIAuswert()

Übergabeparameter: -- Rückgabewert: –
GetBMI()
Ausgabe BMI

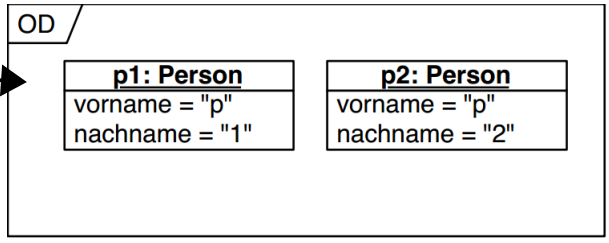
Ein **UML-Klassendiagramm** dient dazu Klassen sprachenunabhängig abzubilden:



Klassen dienen als Grundgerüst/Bauplan für spätere **Objekte** in lauffähigen Programmen
Objektdiagramm stellt eine Momentaufnahme der derzeitigen Objekte dar

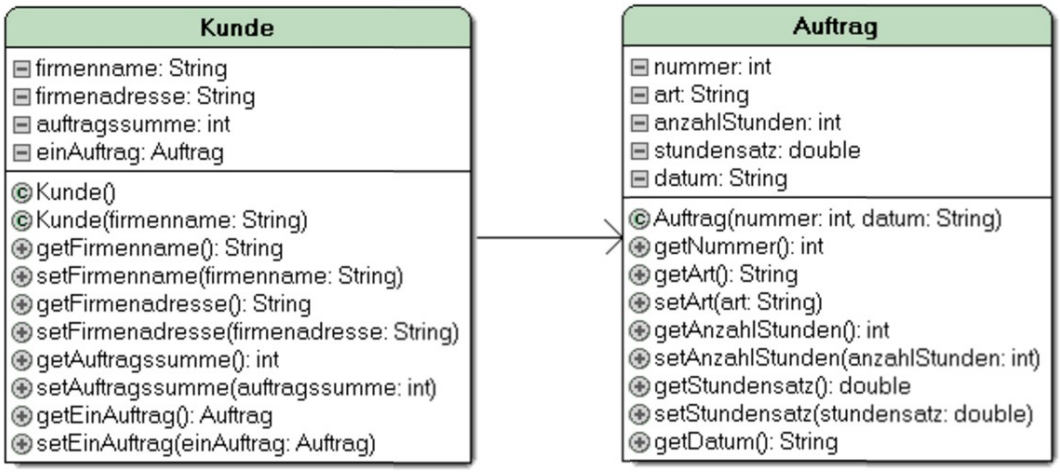
Objektname : Klasse (aus der das Objekt konstruiert wurde)

die aktuellen Werte der Objekteigenschaften

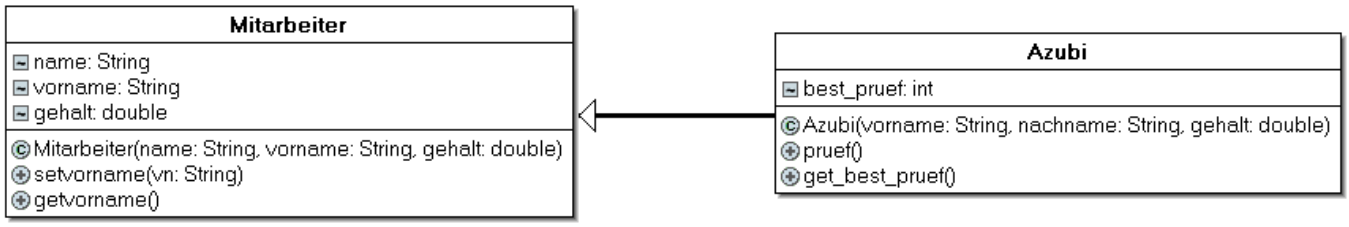


<u>Assoziationen</u> bilden Beziehungen zwischen Klassen ab.

Ein Objekt der Klasse Kunde kennt (s)ein Auftrags-Objekt, umgekehrt kennt ein Objekt Auftrag nicht sein Kunde-Objekt.



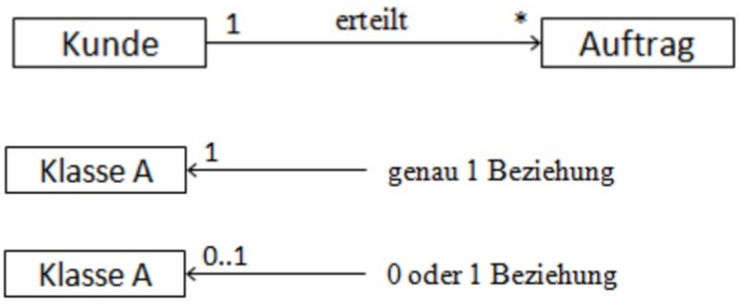
Die wichtigste Beziehung zwischen zwei Klassen: **Vererbung**



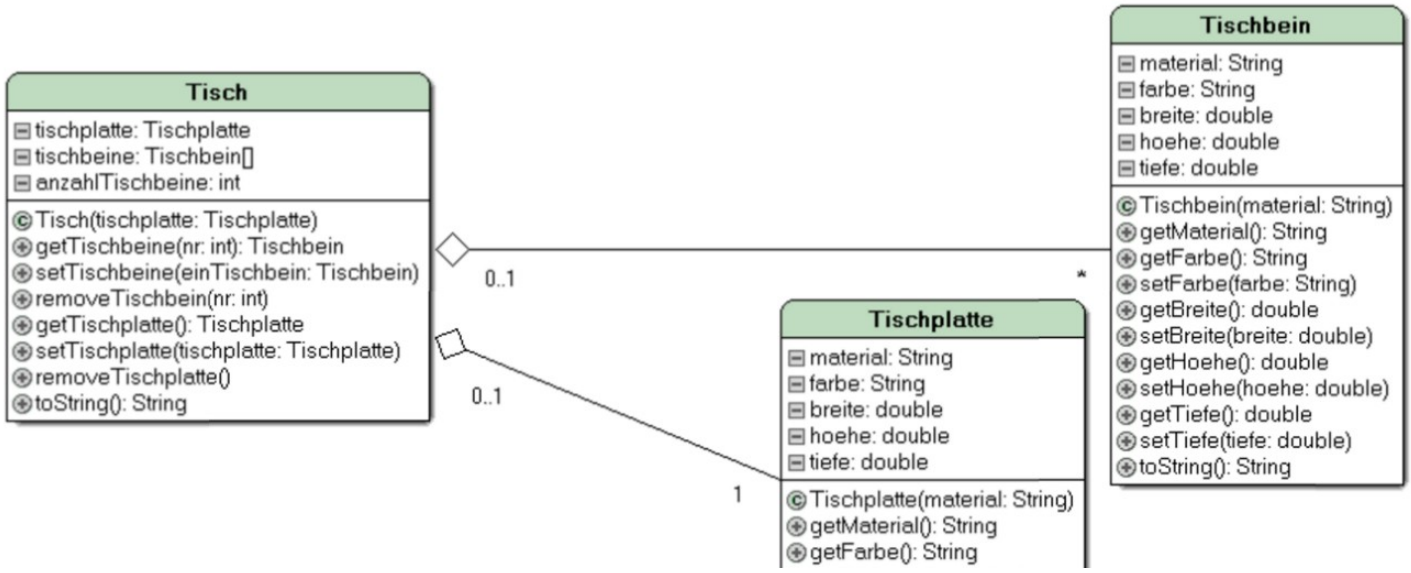
Alles, was in der Klasse Mitarbeiter definiert wurde, steht durch Vererbung auch der Klasse Azubi zur Verfügung. Zusätzliche Eigenschaften und Methoden, die nur für Objekte aus der Klasse 'Azubi' gelten sollen, werden hier ergänzt.

<u>Kardinalität</u> (Multiplzität) bezeichnet die Anzahl der zulässigen Objekte zwischen zwei Klassen.

Ein Kunde kann keinen, einen oder mehrere Aufträge erteilen, ein Auftrag gehört zu genau einem Kunden



<u>Aggregatien</u> (ungefüllte Raute):
Objekte der Klasse Tischbein gehören zu Objekten der Klasse Tisch → Tischbein-Objekte können allein existieren



Ein Tisch kann genau eine Tischplatte und beliebig viele Tischbeine besitzen.
Eine Tischplatte gehört maximal zu einem Tisch, ebenso kann ein Tischbein maximal zu einem Tisch gehören.

Komposition (gefüllte Raute):

Objekte der Klasse Tisch existieren nur, wenn das zugehörige Objekt der Klasse Tischplatte/Tischbein existiert.

Überblick Programmiersprachen

Compiler: Wandelt Quellcode als Ganzes in → Maschinencode → .exe-Datei → Ausführung: Doppelklick

Interpreter: Wandelt Quellcode zeilenweise in → Maschinencode um und führt ihn zeilenweise aus

Ausnahme Java: Compiler → Bytecode → Interpreter (für die entsprechende Plattform) → Ausführung → plattformunabhängig! Mit Abstrichen auch C#

Compiler-sprachen: benötigen keine Umgebung zur Ausführung (nur das passende Betriebssystem) → C++, Visual Basic, Pascal

Interpreter-sprachen: benötigen eine Umgebung (mit passendem Interpreter)
→ JavaScript (Interpreter im Browser) **clientseitig**
→ PHP (Interpreter auf dem Server) **serverseitig**: JSP, ASPX

Datenbanksprache: SQL

Seitenbeschreibungssprache: HTML

Auszeichnungssprache: XML zur Speicherung von Daten in Form von Textdateien.
CSS → Darstellung webbasierter Dokumente (HTML, XML)

Hardwarenahe Sprache: Assembler

Objektorientierte Sprachen: (alle neueren Sprachen) → sie werden in **Klassen** organisiert, in denen **Attribute** (Variablen/Eigenschaften) und **Methoden** definiert werden.