Objektorientierte Programmierung (OOP)

Daten sind an **Objekte** gekoppelt, und daher leicht erweiterbar und wieder auffindbar!
Um mit Objekten arbeiten zu können, muss zuvor in einer Klasse definiert werden, welche Daten diese Objekte bekommen sollen → **Eigenschaften**.

> Der **Zustand** eines Objektes wird durch seine **Eigenschaften** beschrieben.
> Das **Verhalten** eines Objektes wird durch **Methoden** definiert.

Die **Klasse** als 'Bauplan' für Objekte:

Definition der Eigenschaften:

anrede, vorname, nachname

im globalen Bereich einer Klasse →
Objektvariablen (global) → sind innerhalb eines Objektes sichtbar.
Eine solche Variable kann in jedem Objekt dieser Klasse einen anderen Wert haben.
Genau genommen wird für jedes neue Objekt eine neue Variable mit dem gleichen Namen erzeugt.

```
class Daten
{
  String anrede;
  String nachname;
  String vorname;
.....
```

Klassennamen sollten mit einem Großbuchstaben beginnen!

Konstruieren eines Objektes:

Objekte werden mit einer Konstruktorfunktion erstellt.

Jede Klasse besitzt dazu einen **Standardkonstruktor,** der immer so heißt wie die Klasse selber.

Syntax:
Klassenname Objektname = new Konstruktorfunktion;

```
class Daten
{
  String anrede;
  String nachname;
  String vorname;

  public static void main(String[ ]args)
  {
 Daten meier=new Daten();
 Daten mueller=new Daten();
 Daten schulze=new Daten();
  }
}
```

Werte zuweisen und ausgeben:

Die Klassenvariablen gibt es nur jeweils einmal, sie können jedem Objekt zugewiesen werden.
Hinter dem Objektnamen kommt ein Punkt.
Hier kann auf die Daten des entsprechenden Objekts zugegriffen werden →
Objektvariablen

Syntax:
Objektname.Objektvariable

```
class Daten
{
  …..
public static void main(String[ ]args)
  {
 ……..
 meier.anrede = "Frau";
 meier.nachname="Meier";
 meier.vorname="Ingrid";
 mueller.anrede = "Herr";
 mueller.nachname="Müller";
 mueller.vorname="Hans";
System.out.println(meier.vorname); }}
```

Objektvariablen erweitern:

Soll die Datenstruktur der Objekte erweitert werden, wird einfach eine neue Klassenvariable definiert.

Die Eigenschaft 'gehalt' wird ergänzt.

```
class Daten
{
  String anrede;
  String nachname;
  String vorname;
  double gehalt;

public static void main(String[ ]args)
  {
 Daten meier=new Daten();
 Daten mueller=new Daten();
 Daten schulze=new Daten();
 meier.gehalt=3900.90;
 mueller.gehalt=4500.00;
System.out.println(mueller.gehalt); }}
```

Praktischer wäre es, einem Objekt schon bei der Konstruktion Werte zu übergeben, die es ohnehin besitzen muss. Dies ist mit dem Standardkonstruktor nicht möglich.

Deshalb wird ein **zusätzlicher Konstruktor** definiert.

```
Daten(String vn, String na, String an)
 {
   vorname = vn;
   nachname = na;
   anrede=an;
 }
 public static void main(String[ ]args)
 {
   Daten meier_2=new Daten("Hans","Meier","Herr);
  }}
```

Die Konstruktorfunktion heißt wie die Klasse. Man kann für beliebige Objektvariablen Werte setzen.

Bei der Konstruktion eines Objektes können die Werte dem Objekt übergeben werden.

Das Objekt 'meier' hat nun 3 Werte für seine Eigenschaften.

Wird eine Funktion mit *new* aufgerufen wird sie zur *Konstruktorfunktion,* sie liefert ein Objekt *(hier. Meier_2)* zurück.

this

Mit 'this' wird kenntlich gemacht, dass es sich bei dieser Variable um ein Attribut **dieser** Klasse handelt **(Objektvariable)**. Dies ist notwendig, falls die lokalen Übergabeparameter den gleichen Variablenname besitzen (zur Sicherheit sollte man this bei jedem Klassenattribut verwenden).

```
Daten(String vorname, String nachname, String anrede)
 {
   this.vorname = vorname;
   this.nachname = nachname;
   this.anrede=anrede;
 }
 public static void main(String[ ]args)
 {
   Daten meier_2=new Daten("Hans","Meier",4555.90);  }}
```

Lokale Variablen *(Übergabeparameter, Variablenname beliebig)*

globale Variablen, Klassenattribute

Basisklasse und lauffähige Klasse

Innerhalb einer Klasse definiert man Eigenschaften und Methoden, über die die Objekte verfügen sollen.
Diese Klasse dient dann als Bauplan für die Objekte im lauffähigen Programm.

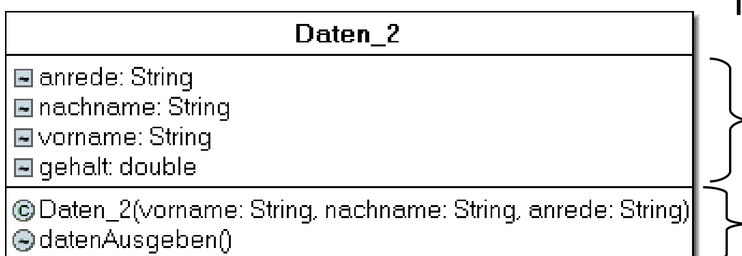
Zur Laufzeit eines Programms werden **Instanzen** (Objekte) erstellt.
In den Objektinstanzen werden dann die Eigenschaften mit konkreten Werten belegt und die in der Klasse definierten Methoden mit dem Objekt ausgeführt.

Klassenmethode erstellen in der Basisklasse

```
import javax.swing.*;
 class Daten_2
{
  String anrede;
  String nachname;
  String vorname;
  double gehalt;

  Daten_2(String vorname, String nachname, String anrede)
  {
    this.vorname = vorname;
    this.nachname = nachname;
    this.anrede=anrede;
  }
  void datenAusgeben()
  {
    JoptionPane.showMessageDialog(null,this.anrede+
                  "\n"+this.vorname+"  "+this.nachname);
  }
}
```

Klassen werden übersichtlich in einem UML (Unified Modeling Language (vereinheitlichte Modellierungssprache)) – Diagramm dargestellt:



Eigenschaften

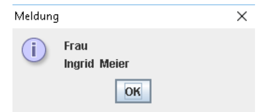
Methoden

lauffähige Klasse / Wertzuweisung, benutzen der Methoden

```
class Anwendung
{
  public static void main(String[ ]args)
  {
     Daten_2 schulze = new Daten_2
("Friedhelm","Schulze","Herr");

    Daten_2 meier = new Daten_2
("Ingrid","Meier","Frau");

    schulze.gehalt=3000.90;
    meier.gehalt=4500.45;
    meier.datenAusgeben();
    schulze.datenAusgeben();
    System.out.println(meier.gehalt);
    System.out.println(schulze.gehalt);
 }}
```




```
4500.45
3000.9
```

In der lauffähigen Anwendung werden Objekte aus dieser Klasse (Bauplan) konstruiert. Ihnen können alle dort definierten Eigenschaften zugewiesen werden und es können alle dort definierten Methoden mit dem Objekt ausgeführt werden.

Syntax:
Objektname.Methode(Werte)

Datenkapselung

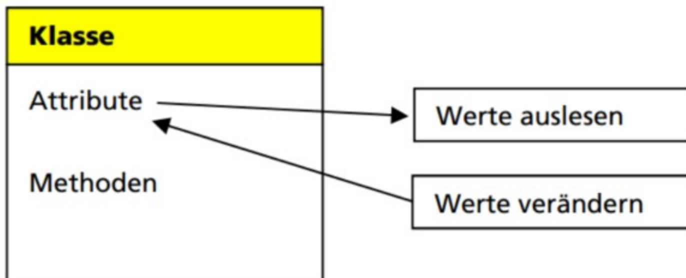
Die Eigenschaft 'gehalt' kann bei jedem Objekt jederzeit gesetzt und verändert werden.
Dies will man in der Regel unterbinden. Deshalb ist es in der OOP möglich Eigenschaften und

Methoden zu schützen → **Kapselung.**
Bei der **Datenkapselung** werden die Attribute nach außen hin versteckt und der Zugriff auf sie über Methoden ermöglicht. Diese können Werte prüfen, bevor sie in den Attributen gespeichert werden.
Über **Modifizierer** kann die Sichtbarkeit und Veränderbarkeit von Variablen und Methoden bestimmt werden.

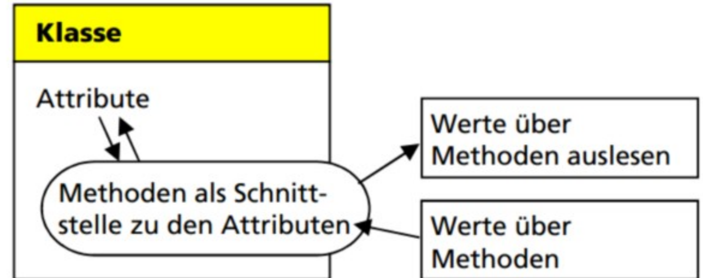
private (Symbol -) Methoden oder Variablen können nur innerhalb der eigenen Klasse verwendet oder aufgerufen werden.

public (Symbol +) Methoden oder Variablen lassen sich generell von allen anderen Klassen und Paketen aus aufrufen.

public:

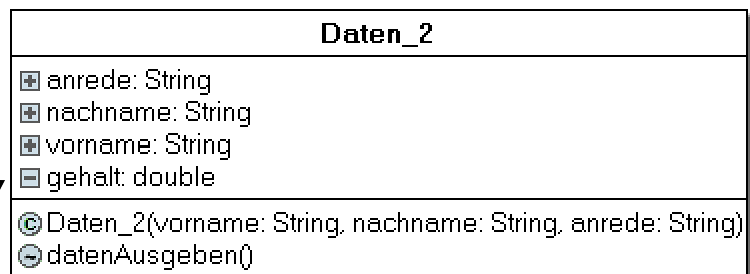


private:



Die Eigenschaft 'gehalt' soll nun durch private geschützt werden.

```
class Daten_2
{
  public String anrede;
  public String nachname;
  public String vorname;
  private double gehalt;
………
```




```
class Anwendung
{
 public static void main(String[ ]args)
 {
  Daten_2 schulze=new Daten_2("Friedhelm","Schulze","Herr");
  Daten_2 meier=new Daten_2("Ingrid","Meier","Frau");
  schulze.gehalt=3000.90;
  meier.gehalt=4500.45;
  System.out.println(meier.gehalt);
  …………...
```

In einer anderen Klasse kann das Attribut *gehalt* so nicht mehr verändert werden

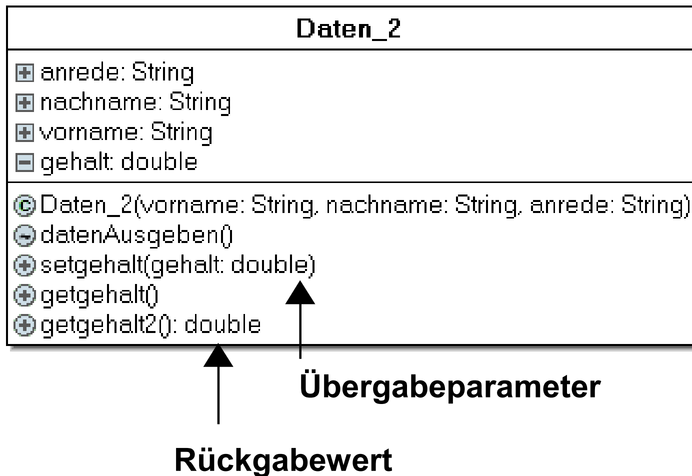
Um die Werte der Objektvariable *gehalt* zu ändern oder auszugeben wird eine eigene Methode benötigt.
Meistens werden alle Eigenschaften bei der Deklaration auf 'private' gesetzt.

Sollen sie in anderen Klassen verändert oder ausgelesen werden, wird für jedes Attribut eine so genannte **set- und get-Methoden** notwendig.
Getter-Methoden geben den Wert einer Eigenschaft zurück.

Durch **Setter-Methoden** besteht die Möglichkeit Werte, die einer Eigenschaft zugewiesen werden sollen auf Gültigkeit zu überprüfen (validieren).

Methoden zum Setzen und Auslesen der Eigenschaft 'gehalt' eines Objektes

UML-Diagramm der Klasse *Daten_2:*




```
class Daten_2
{
  …..
  private double gehalt;
  ……..
  public void setgehalt(double gehalt)
  {
    this.gehalt = gehalt;
  }

  public void getgehalt()
  {
System.out.println(this.gehalt);
  }

public double getgehalt2()
  {
   return this.gehalt }}
```

```
class Anwendung
{
  public static void main(String[ ]args)
  {
    Daten_2 schulze=new Daten_2("Friedhelm","Schulze","Herr");
    Daten_2 meier=new Daten_2("Ingrid","Meier","Frau");
    meier.datenAusgeben();
    schulze.datenAusgeben();

    meier.setgehalt(3333.90);
    double gehalt=meier.getgehalt2();
    System.out.println(gehalt);

  }}
```

Die Objektvariable *gehalt* lässt sich in anderen Klassen nur mit einer set-Methode ändern / setzen und mit einer get-Methode auslesen!

Aufgabe 1: Basisklasse *Daten_2:*

- Definieren Sie eine fünfte Eigenschaft 'abteilung' und setzen Sie sie auf private.
- Erstellen Sie eine zusätzliche Konstruktorfunktion, die einem Objekt Werte für alle 5 Eigenschaften zuweist.
- Erstellen Sie eine set-Methode zum Ändern des Wertes der Eigenschaft *abteilung.*
- Erstellen Sie eine get-Methode zur Rückgabe des Wertes der Eigenschaft *abteilung.*
- Erstellen Sie eine Methode, die alle 5 Eigenschaften eines Objektes auf dem Bildschirm ausgibt.

lauffähiges Programm: Erstellen Sie ein Objekt mit der neuen Konstruktorfunktion und wenden Sie alle in Aufgabe 1 erstellten Methoden an.

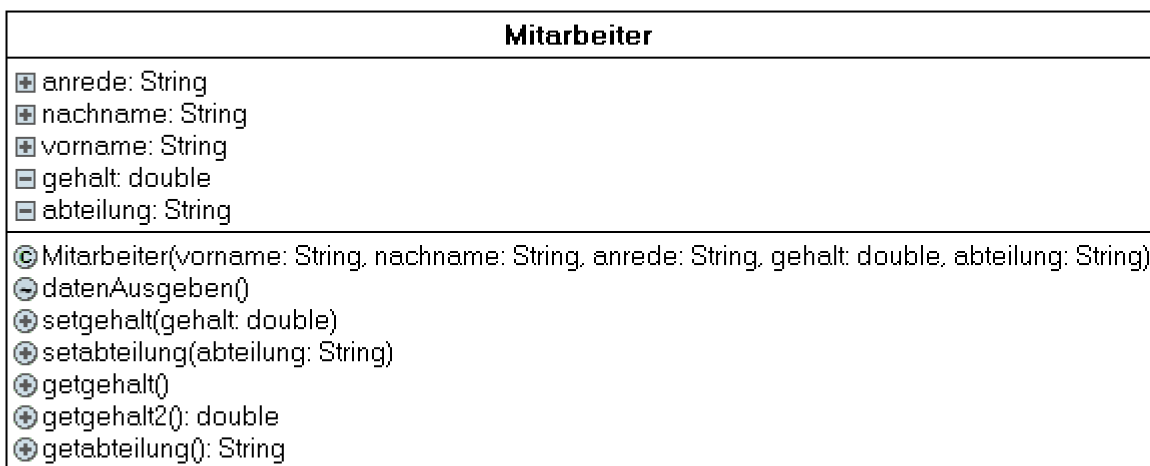
Vererbung

Es soll ein Java-Programm zur Mitarbeiterverwaltung in einer Firma entworfen werden.

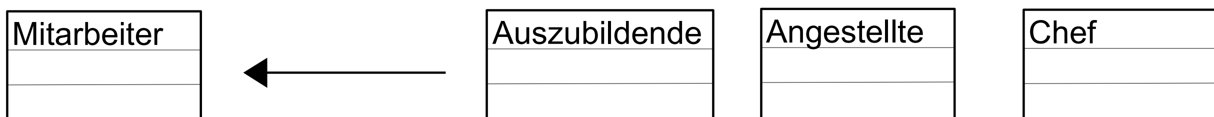
Jeder Mitarbeiter wird in dem Java-Programm als Objekt abgebildet. Dazu wird zunächst eine Klasse 'Mitarbeiter' erstellt, die als Grundgerüst dienen soll. Hier werden grundlegende Eigenschaften (Attribute) festgelegt, die ein Objekt später annehmen kann. Außerdem werden Methoden definiert, die für ein Objekt aus der Klasse Mitarbeiter notwendig sind, bzw. die die Attribute eines Objektes verändern können.

> Das, was alle Mitarbeiter gemeinsam haben, wird durch die Definition einer **Klasse** festgelegt. Der Einzelfall – der individuelle Mitarbeiter – ist hingegen dann eine Instanziierung der Klasse und konkret ein **Objekt (eine Instanz).**
> Bevor das Programmieren beim objektorientierten Ansatz beginnt, wird zuerst eine Basisklasse als Schablone erstellt.

Aufgabe 2:
Benennen Sie dazu die Basisklasse aus Aufgabe 1 um (inklusive Konstruktor). Diese Klasse wird als Grundlage für das Mitarbeiterprogramm dienen (UML s.u.).



Eine Firma hat verschiedene Mitarbeiter, für die individuelle Eigenschaften und Methoden programmiert werden sollen. Dies wird in zusätzlichen Klasse gemacht. Damit diese nicht komplett neu erstellt werden müssen, werden sie von der Klasse Mitarbeiter abgeleitet.



Die Eigenschaften und Methoden der Klasse Mitarbeiter gelten für alle abgeleiteten Klassen. Jede neue Klasse bekommt individuelle Eigenschaften und Methoden dazu.

Die Klasse 'Azubi' soll neben den 5 Eigenschaften der Klasse Mitarbeiter die zusätzliche Eigenschaft *best_pruef* , die bei neuen Objekten den Wert 0 haben soll, bekommen.
Die zusätzlichen Methoden *pruef()* soll bei Aufruf das Attribut *best_pruef* um 1 erhöhen. Die Methode *get_best_pruef()* soll die bestandenen Prüfungen anzeigen.

```
class Azubi extends Mitarbeiter
{

private int best_pruef;
```

Durch das Schlüsselwort extends wird die Klasse Azubi von der Klasse Mitarbeiter abgeleitet → Es stehen hier alle Attribute und Methoden der Klasse Mitarbeiter zur Verfügung (s. UML-Diagramm unten)

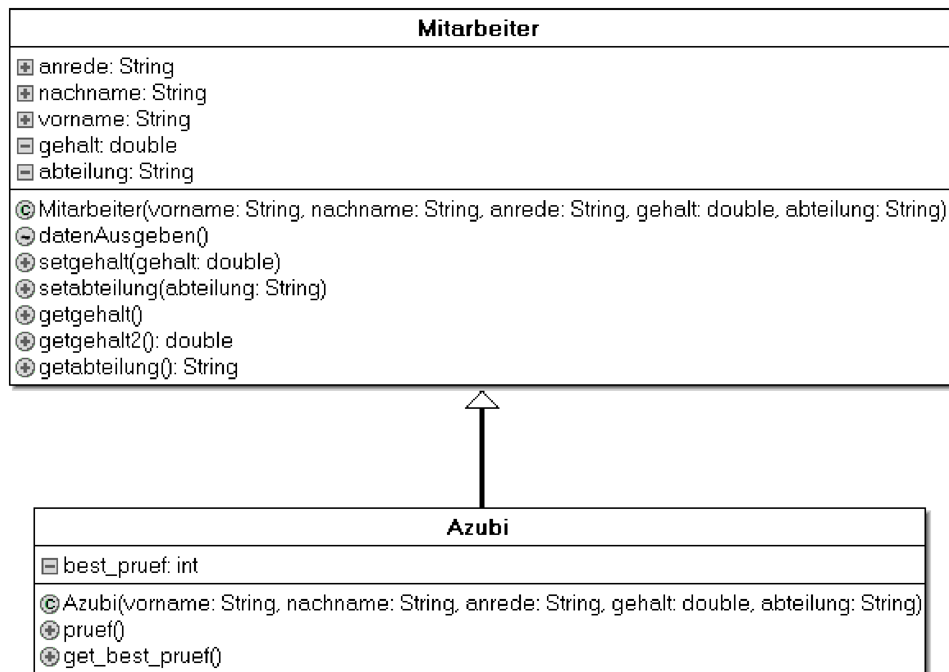
```
 Azubi (String vorname, String nachname, String anrede, double gehalt, String abteilung)
 {
 super(vorname,nachname,anrede,gehalt,abteilung);
 this.best_pruef=0;
 }

 public void pruef()
 {
 this.best_pruef ++;
 }

 public void get_best_pruef()
 {
 System.out.println(this.best_pruef);
 }
}
```

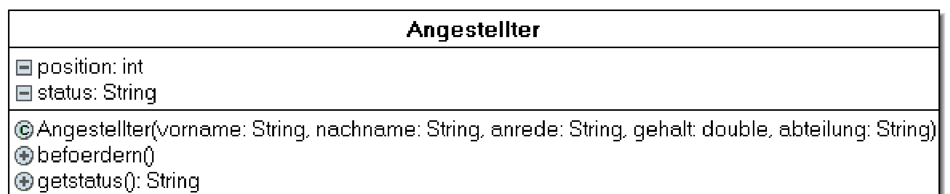
Das Schlüsselwort super steht für die Basisklasse → der Konstruktor der Basisklasse wird aufgerufen.

In der Konstruktorfunktion 'Azubi' wird noch das Attribut best_pruef auf 0 gesetzt → wird ein neues Objekt aus der Klasse 'Azubi' angelegt, hat diese Eigenschaft den Startwert 0



Aufgabe 3:

Erstellen Sie eine Klasse Angestellter (s. UML-Diagramm). Sie soll alle Attribute und Methoden der Klasse Mitarbeiter erben.



- Zwei zusätzliche Attribute sollen erstellt werden
- Konstruktorfunktion:
  bei neuen Objekten soll die Eigenschaft *position* den Wert 0 erhalten und die Eigenschaft *status* den Wert 'Assessor'.

- Methode *getstatus:* der Wert von *status* soll zurück gegeben werden.
- Methode *befoerdern:* die Variable *position* soll nur bis 4 erhöht werden können.

  In Abhängigkeit der Variablen *position* wird der Variablen *status* der Wert Studienrat, Oberstudienrat, Studiendirektor oder Oberstudiendirektor zugewiesen (der Startwert ist Assessor).

> *position* muss der Methode nicht übergeben werden. Innerhalb der Methode kann mit ***this.position*** auf die globale Klassenvariable zugegriffen werden.

Methode befoerdern

Übergabeparameter: - Rückgabewert: -	
ja — position <= 4 ? — nein	
position ++	-
ja — position = 1? — nein	
status = Studienrat	∅
ja — position = 2 ? — nein	
status = Oberstudienrat	∅
ja — position = 3 ? — nein	
status = Studiendirektor	∅
ja — position = 4 ? — nein	
status = Oberstudiendirektor	∅
ja — position > 4 — nein	
Ausgabe: Das Ende der Karriereleiter ist erreicht	∅

Aufgabe 4:

Erstellen Sie eine Klasse Chef.

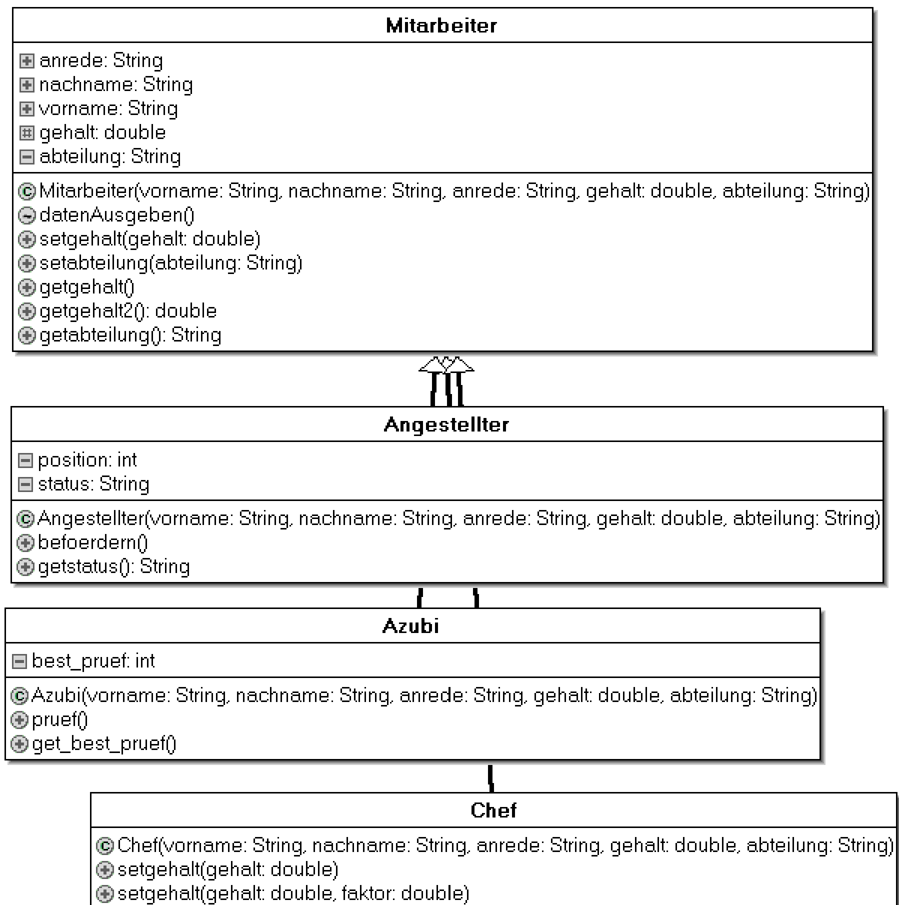
Sie soll alle Attribute und Methoden der Klasse Mitarbeiter erben und um zwei Methoden erweitert werden:

setgehalt(double gehalt)

soll die Eigenschaft ***gehalt*** um das Doppelte des übergebenden Gehalts erhöhen.

setgehalt(double gehalt, double y)

soll die Eigenschaft ***gehalt*** um den Faktor y des übergebenden Gehalts erhöhen.



Die Klassenvariable *gehalt* ist in der Klasse *Mitarbeiter* auf ***private*** gesetzt → diese Eigenschaften werden an vererbte Klassen nicht vererbt →
gehalt muss auf ***protected*** in der Klasse *Mitarbeiter* gesetzt werden, damit vererbte Klassen darauf zugreifen können.

Polymorphie (Vielgestaltigkeit)

Methoden mit identischer Signatur liefern unterschiedliche Ergebnisse.

```
class Chef extends Mitarbeiter
{
 Chef (String vorname, String nachname, String anrede, double gehalt, String abteilung)
 {
 super(vorname,nachname,anrede,gehalt,abteilung);
 }

 public void setgehalt(double gehalt)
 {
  this.gehalt = gehalt * 2;
 }
 public void setgehalt(double gehalt, double faktor)
 {
  this.gehalt = gehalt* faktor;
 }
}
```

Überschreiben von Methoden (dynamische Polymorphie):

gleicher Methodenname und gleiche Übergabeparameter → aber anderer Quellcode als die Methode der Basisklasse
Unterscheidung:
wird das Objekt aus der Klasse Chef erstellt, wird bei Aufruf die Methode aus der Klasse Chef aufgerufen.

Überladen von Methoden (statische Polymorphie):
gleicher Methodenname aber unterschiedliche Übergabeparameter. Unterscheidung: anhand der Übergabeparameter

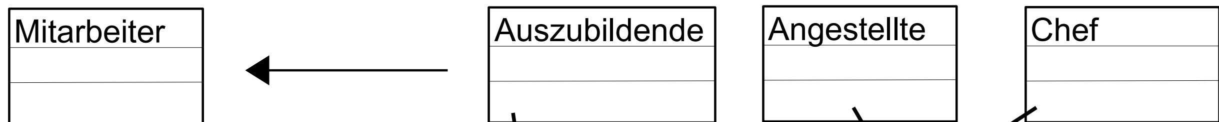
Aufgabe 5

Ergänzen Sie die Klasse '*Mitarbeiter*', die als Basisklasse für die Klassen '*Chef*', '*Azubi*' und '*Angestellter*' dient, wie folgt:

- um eine **weitere** Eigenschaft (*pnr*), in der die Personalnummer eines Mitarbeiters gespeichert werden soll.
- einen zusätzlichen Konstruktor, der einem Objekt Werte für alle 6 Eigenschaften übergibt
- eine Methode (*getalles()*), die alle Werte dieser sechs Eigenschaften eines Objektes anzeigt (in der Standardausgabe)
- eine Methode (*getalles_popup()*), die alle Werte der sechs Eigenschaften eines Objektes in einem Popup-Fenster anzeigt
- Erstellen Sie eine Methode 'setBonus*()*', die das Attribut 'gehalt' eines Objektes um einen Bonus erhöht. Auf das aktuelle Gehalt (Attribut *gehalt*) des Objektes wird in der Methode mit ***this.gehalt*** zugegriffen. Liegt das Gehalt unter 1000.00 beträgt der Bonus 100.00, liegt es über 3000.00 beträgt er 300.00. Bei einem Gehalt zwischen 1000.00 und 3000.00 soll der Bonus 200.00 betragen.
- Übernehmen Sie in den drei Unterklasse '*Chef*', '*Azubi*' und '*Angestellter*' die Konstruktorfunktion aus der Klasse '*Mitarbeiter*', die bei allen Attributen einen Wert übergibt.

Die Basisklasse *Mitarbeiter* und die 3 abgeleiteten Klassen werden eingesetzt:

Objekte, die aus der Klasse ***Auszubildende*** konstruiert werden, können die Eigenschaften annehmen, die in den Klassen ***Mitarbeiter*** und ***Auszubildende*** definiert wurden und es können die Methoden der beiden Klassen angewendet werden.
Objekte, die aus der Klasse ***Chef*** konstruiert werden, können die Eigenschaften annehmen, die in den Klassen ***Mitarbeiter*** und *Chef* definiert wurden und es können die Methoden der beiden Klassen angewendet werden usw.



Lauffähige Anwendung:

Objekte werden aus der passenden Klasse konstruiert. Hier oder in der Oberklasse stehen alle benötigten Eigenschaften und Methoden.

- Konstruieren Sie in dieser Applikation aus jeder der 3 Unterklasse je ein Objekt. Benutzen Sie den Konstruktor, der alle Attribute setzt. Lassen Sie die Eigenschaften der Objekte in einem Popup-Fenster anzeigen.
  Notieren Sie sich die Nachnamen und die Gehälter der Objekte.

- Lassen Sie dem Objekt aus der Klasse 'Azubi' eine Prüfung bestehen
- Das Objekt aus der Klasse 'Angestellter' soll befördert werden
- Bei allen 3 Objekte soll das Gehalt um 100 erhöht werden
- Der Azubi wechselt in die Abteilung 'Forschung', ändern Sie entsprechend das Attribut mit der passenden Methode
- Vom Auszubildenden soll die Anzahl der bestandenen Prüfungen ausgegeben werden

- Alle 3 Objekte sollen einen Bonus von 50 bekommen
- Von allen 3 Objekten soll das Gehalt ausgelesen und in eine Variable gespeichert werden; es soll der Mittelwert der 3 Gehälter auf dem Bildschirm ausgegeben werden

Lassen Sie die Eigenschaften der Objekte wieder in einem Popup-Fenster anzeigen. Notieren Sie sich wieder die Nachnamen und die Gehälter der Objekte. Vergleichen Sie diese Werte mit den zuvor aufgeschriebenen und überprüfen Sie, ob die Applikation richtig programmiert wurde.