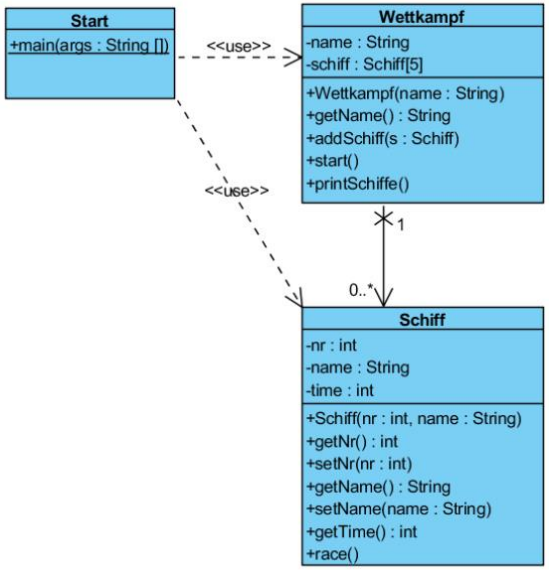
UML (Unified Modeling Language)

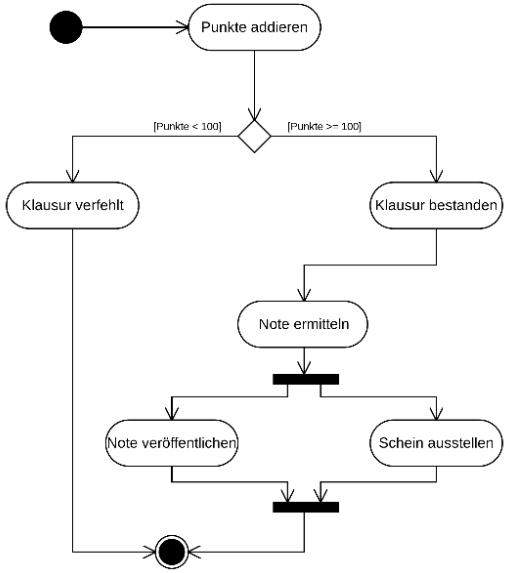
Es handelt sich um eine **grafische** Modellierungssprache in der Softwareentwicklung mit 14 Diagrammen.

Z.B.

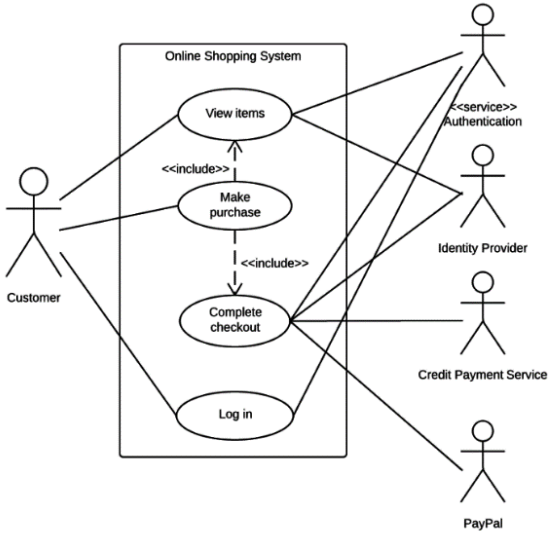
Klassendiagramme:



Aktivitätsdiagramme:

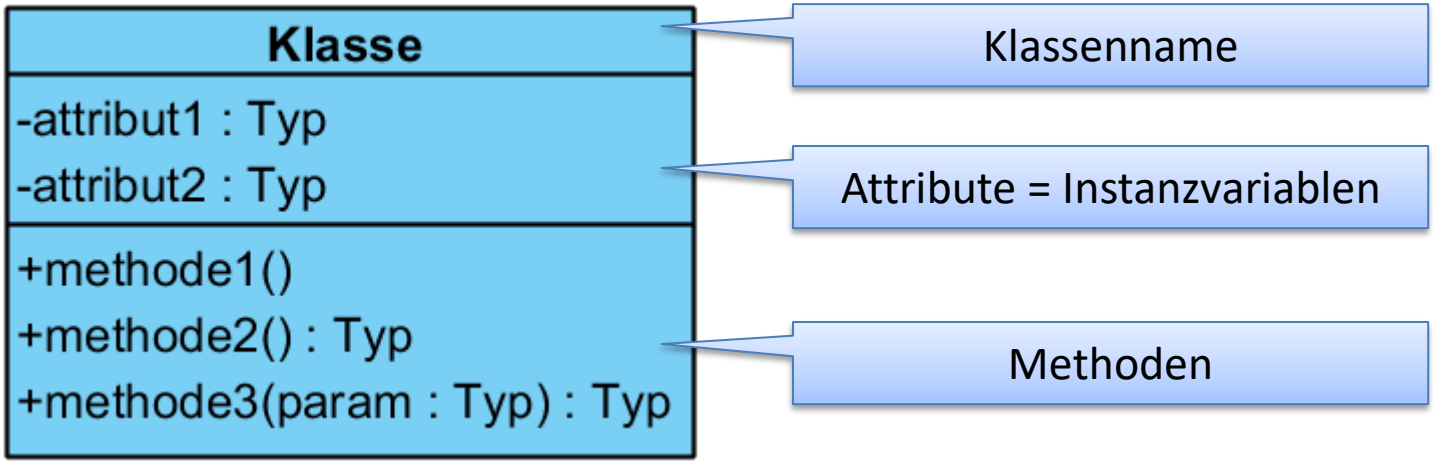


Use Case Diagramme:



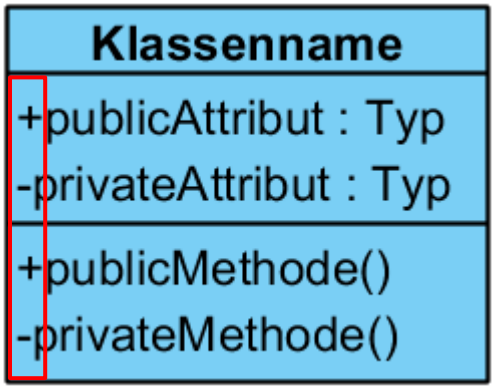
Klassendiagramme

Darstellung einer Klasse



Klassendiagramme

Sichtbarkeiten



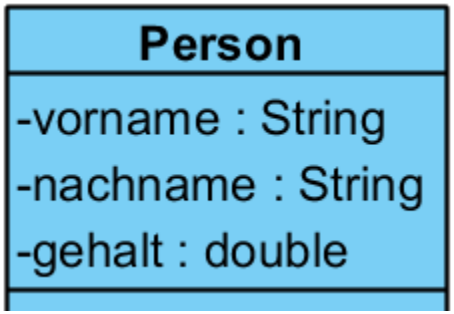
+ public

- private

Klassendiagramme

Attribute

```
public class Person {

        private String vorname;
        private String nachname;
        private double gehalt;
}
```

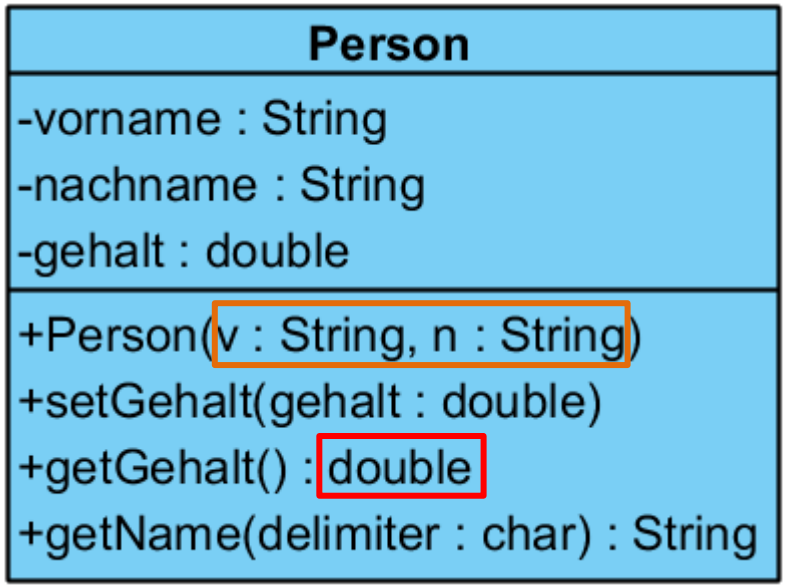


Schema:

Sichtbarkeit Variablenname : Datentyp

Klassendiagramme: Methoden

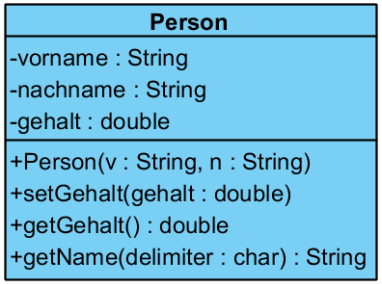

```
public class Person {
    private String vorname;
    private String nachname;
    private double gehalt;

  public Person(String v, String n) {
    vorname = v;
    nachname = n;
  }

  public void setGehalt(double gehalt) {
    this.gehalt = gehalt;
  }

  public double getGehalt() {
    return gehalt;
  }

  public String getName(char delimiter) {
    return vorname + delimiter + nachname;
  }
}
```



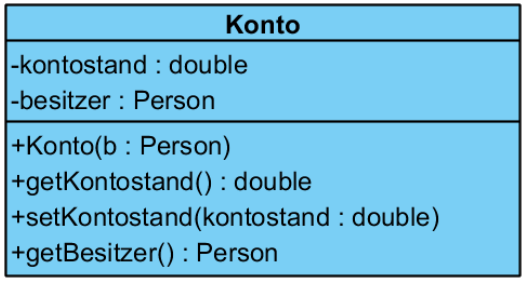
Schema: Sichtbarkeit Methodenname (Parametername : Datentyp) : Rückgabewert

Mehrere Klassen


```java
public class Person {
  private String vorname;
  private String nachname;
  private double gehalt;

  public Person(String v, String n) {
    vorname = v;
    nachname = n;
  }

  public void setGehalt(double gehalt) {
    this.gehalt = gehalt;
  }

  public double getGehalt() {
    return gehalt;
  }

  public String getName(char delimiter) {
    return vorname + delimiter + nachname;
  }
}
```




```java
public class Konto {

  private double kontostand;
  private Person besitzer;

  public Konto(Person b) {
    besitzer = b;
  }

  public double getKontostand() {
    return kontostand;
  }

  public void setKontostand(double kontostand) {
    this.kontostand = kontostand;
  }

  public Person getBesitzer() {
    return besitzer;
  }
}
```



Mehrere Klassen


```
public class Person {

   private String vorname;
   private String nachname;
   private double gehalt;

   ...
}
```

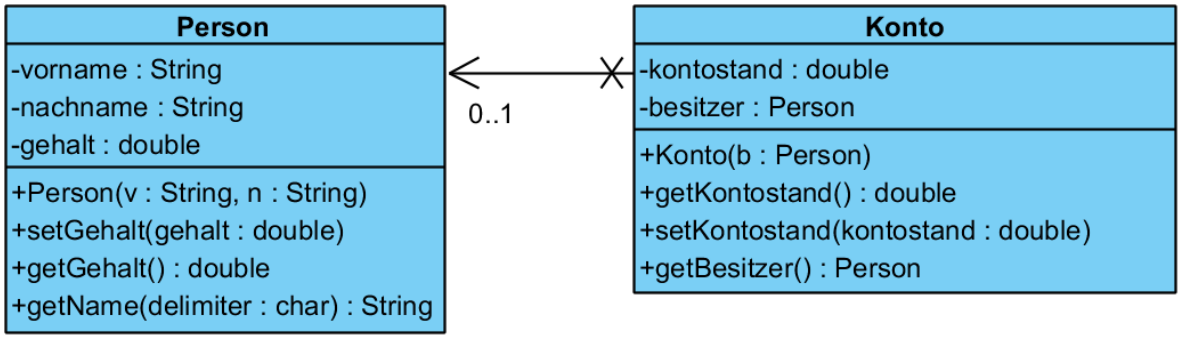
```
public class Konto {

   private double kontostand;
   private Person besitzer;

   ...
}
```

Die beiden Klassen haben untereinander eine Abhängigkeit

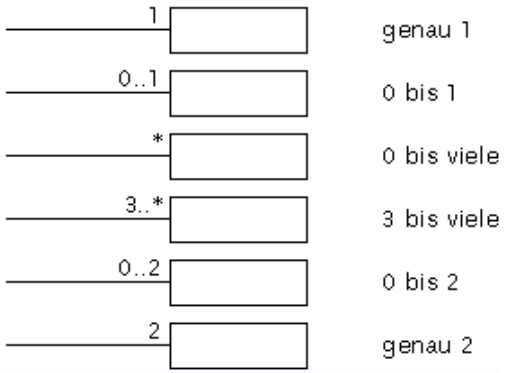
Ein Konto hat einen Besitzer. Der Besitzer wird im Konto als Attribut vom Typ einer Person gespeichert.

Assoziationen (Beziehungen)

- **Assoziationen erkennen Sie, aufgrund der Instanzvariablen einer Klasse!**
- Beziehung zwischen zwei Klassen in Form einer Verbindung
- Am Ende der Verbindung steht die Multiplizität *(multiplicity)*

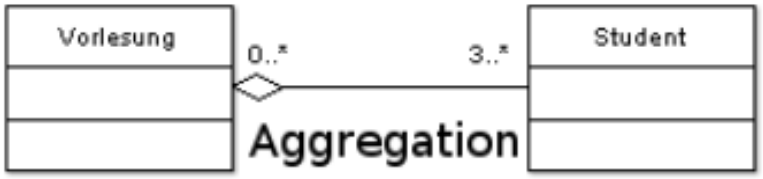


Multiplizitäten:



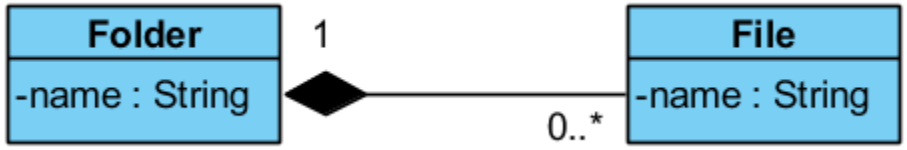
Assoziationen (Beziehungen)

- **Aggregation**
  - Lässt sich durch *ist Teil von* bzw. *besteht aus* beschreiben (Ganzes und Teile)
  - Verstärkung einer einfachen Assoziation



Assoziationen (Beziehungen)

- **Komposition**
  - »starke« Aggregation
  - Multiplizität der Aggregatklasse ist 1 oder 0..1
  - Wird das Ganze gelöscht, so werden automatisch auch seine Teile gelöscht
  - Das Ganze ist verantwortlich für das Erzeugen und Löschen seiner Teile



Klassendiagramme : Tools

Klassendiagramme selbst erstellen

Visual Paradigm Community Edition

https://www.visual-paradigm.com/download/community.jsp

Klassendiagramme aufgrund Code generieren lassen

Visual Paradigm ab Version Standard

Eclipse: Plugin UML Lab https://www.uml-lab.com/de/uml-lab/ (für Studierende ist eine Registrierung notwendig, ansonsten kostenlos)

Weitere Tools:

UMLet: https://www.umlet.com/, http://www.umletino.com/umletino.html
Draw.io: https://www.draw.io/