# UML-Diagramme - Aktivitätsdia., Zustandsdia., Klassendia., PAP, Struktogramm, Use-Case Dia. 1

### Definition

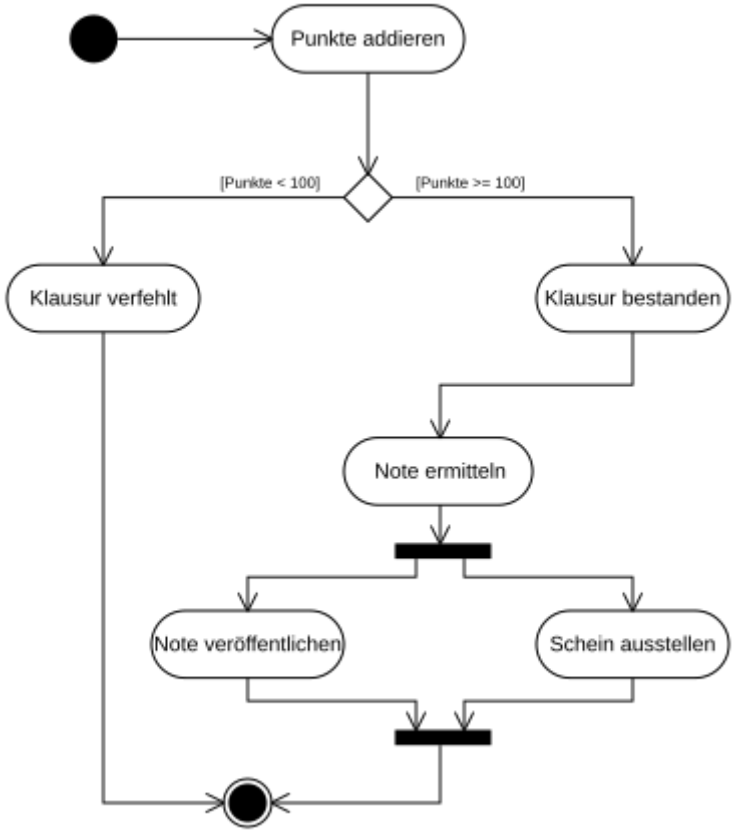
Die **Unified Modeling Language** *(vereinheitlichte Modellierungssprache)*, kurz **UML**, ist eine grafische Modellierungssprache zur Spezifikation, Konstruktion, Dokumentation und Visualisierung von Software-Teilen und anderen Systemen. Sie ist von der ISO 19505 für Version 2.4.1 genormt. Aktuell gibt es 14 verschiedene Diagramme die man mit **UML** zeichnen kann, aufgeteilt in zwei Gruppen. Die Strukturdiagramme und die Verhaltensdiagramme.

## Aktivitätsdiagramm

### Definition

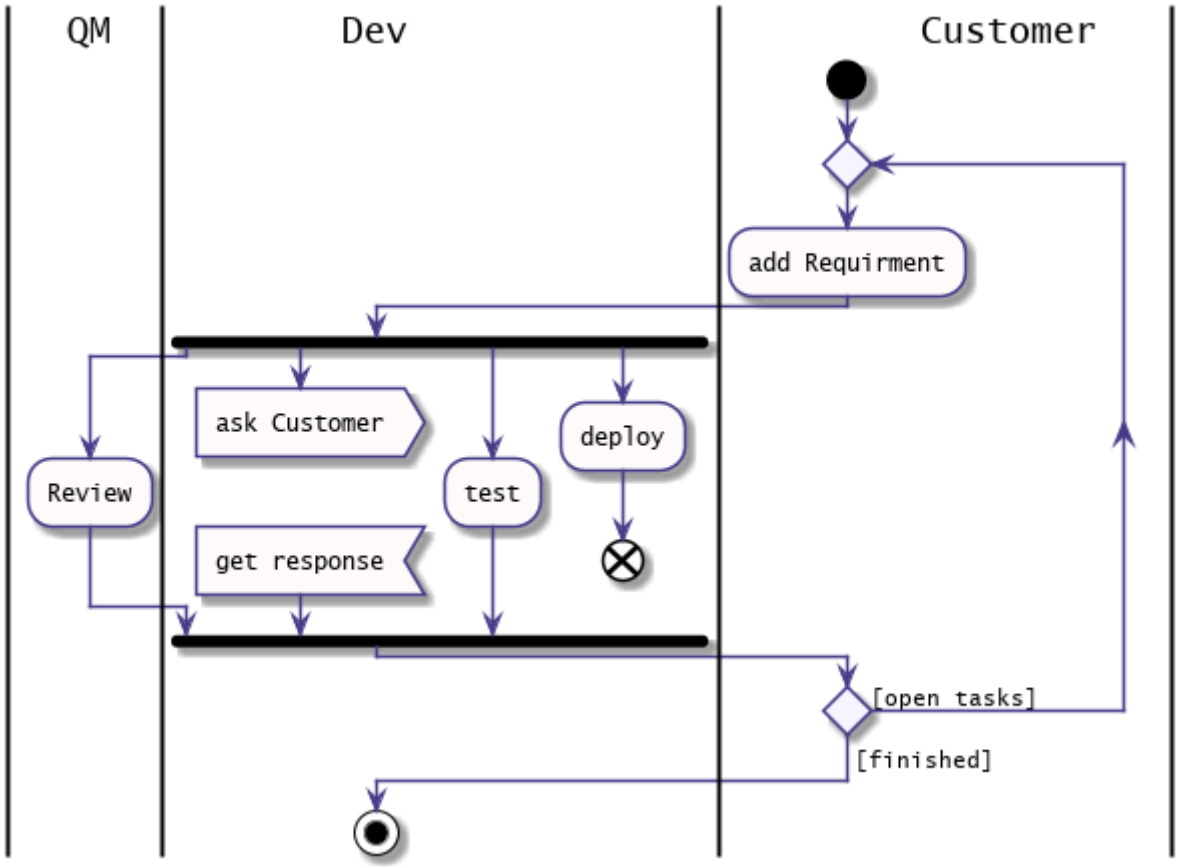
Ein Aktivitätsdiagramm ist ein Verhaltensdiagramm der **UML** , dass Vernetzungen von Aktionen und deren Verbindungen mit Kontroll-und Datenflüssen grafisch darstellen soll.
Mit einem Aktivitätsdiagramm wird meist der Ablauf eines Anwendungsfalls beschrieben, es eignet sich aber zur Modellierung aller Aktivitäten innerhalb eines Systems.
Ein Aktivitätsdiagramm spezifiziert eine Aktivität. Die detaillierten Regeln dafür, wie Token in einer Aktivität fließen, bilden die Grundlage für ein Aktivitätsdiagramm.

**Beispiel 1:**




*Ein übersichtliches Aktivitätsdiagramm*

### Beispiel 2:




*Ein Aktivitätsdiagramm mit Schwimmlinien (QM, Dev & Customer)*

---

## Symbole

### Start- und Endknoten

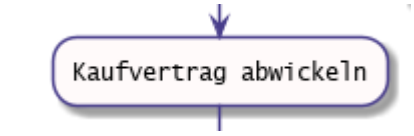
Sind essentiell im Aktivitätsdiagramm! Ein Häufiger Fehler ist es, diese beiden Punkte nicht eingezeichnet zu haben. Sie symbolisieren den Start und das Ende. Im Beispielbild 2 ist das "Ablaufende" im Diagramm zu sehen.


---

### Aktivität

Das **Oval** einer Aktivität ist die Grundlegende Symbolik im Aktivitätsdiagramm. In ihr beschreibt man welcher Teilschritt hier stattfindet. Da es sich um eine Aktivität handelt

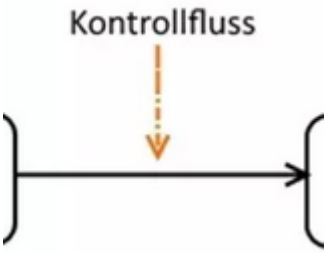
verwendet man **Verben** zum beschreiben!




---

## Kontrollfluss (Pfeil)

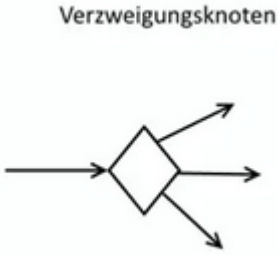
Pfeil der den Ablauf oder die Flussrichtung des Diagramms symbolisiert. Auf dem Pfeil können Bedingungen verzeichnet werden, wie etwa "Punkte >= 100", siehe Verzweigungsknoten!
Alle Pfeile müssen in einem Verzweigungsknoten (Raute), Verbindungsknoten (Raute) oder Parallelisierungsknoten (schwarzer Balken) anfangen oder enden!




---

## Verzweigungsknoten (Entscheidung)

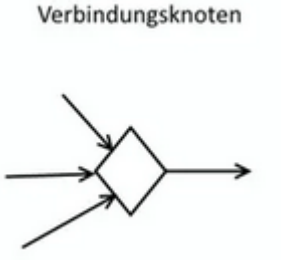
Diese Raute symbolisiert eine Entscheidung, im Code wäre das eine "If-Bedingung". Die Bedingung verzeichnet man auf den Pfeil der von der Raute ab geht.




---

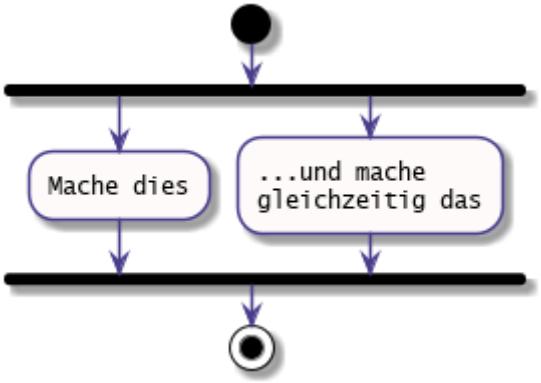
## Verbindungsknoten (Zusammenführung)

Da alle Pfeile im Aktivitätsdiagramm in einer Raute oder schwarzem Balken enden müssen, wird eine Zusammenführung ebenfalls mit einer Raute realisiert.



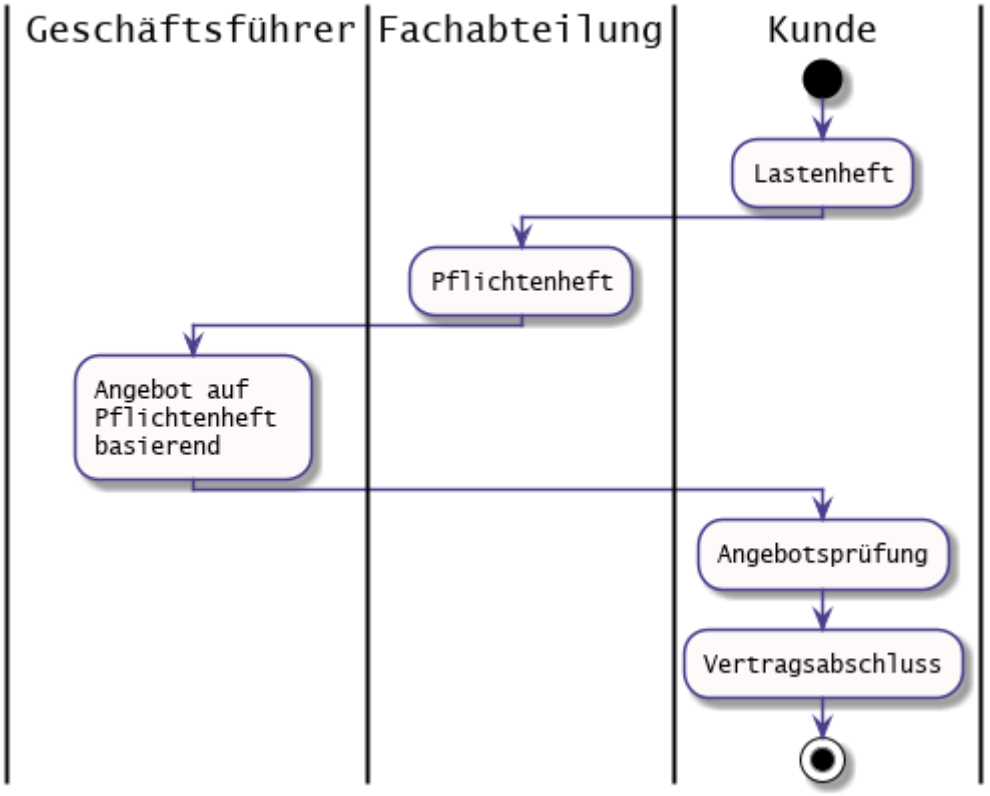
## Parallelisierungsknoten

Der schwarze Balken des Parallelisierungsknoten symbolisiert den gleichzeitigen, parallelen Ablauf einer Aktivität! Erst wenn beide Prozesse im Beispielbild erledigt sind, läuft das Programm weiter und zwar zum Zielpunkt.



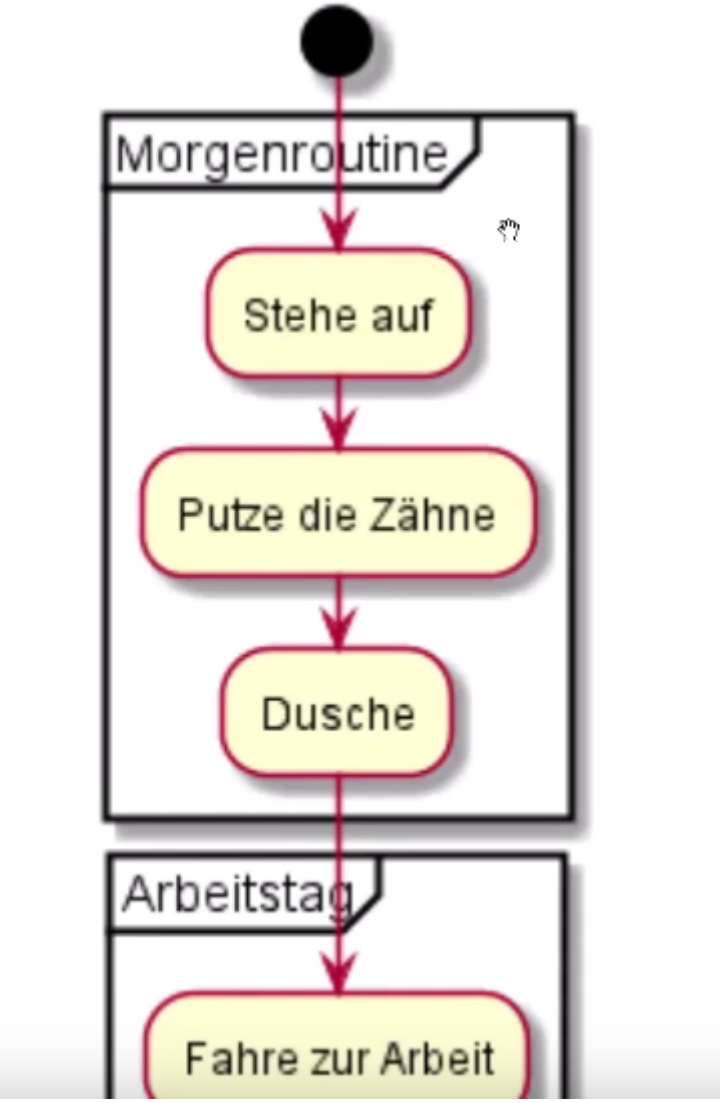

## Schwimmlinien

Schwimmlinien oder engl. Swimlanes sollen die Zuständigkeiten von Abteilungen, Mitarbeiter oder Programm-Threads darstellen. Es ist dabei egal ob die Swimlanes von oben nach unten, oder von links nach rechts gezogen werden. Eine eindeutige Zuweisung ist das wichtigste.



## Gruppierung

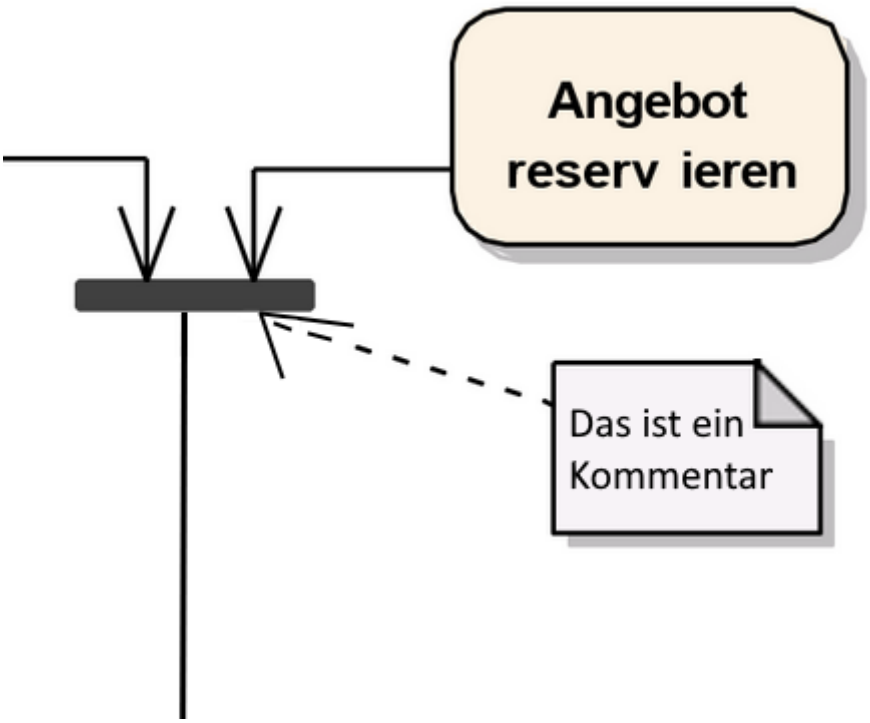
Eine Gruppierung ermöglicht mehrere Teilschritte zu einer Gruppe zusammenzufassen. Im Beispiel unten *Morgenroutine & Arbeitstag*. Dies hat keine Auswirkung auf den Ablauf! Es dient nur der Übersichtlichkeit und der Vereinheitlichung!




## Kommentare

Kommentare im Diagramm sind zugelassen. Eine Zuweisung an die richtige Stelle führt man

mit einem Strichpunktpfeil aus!



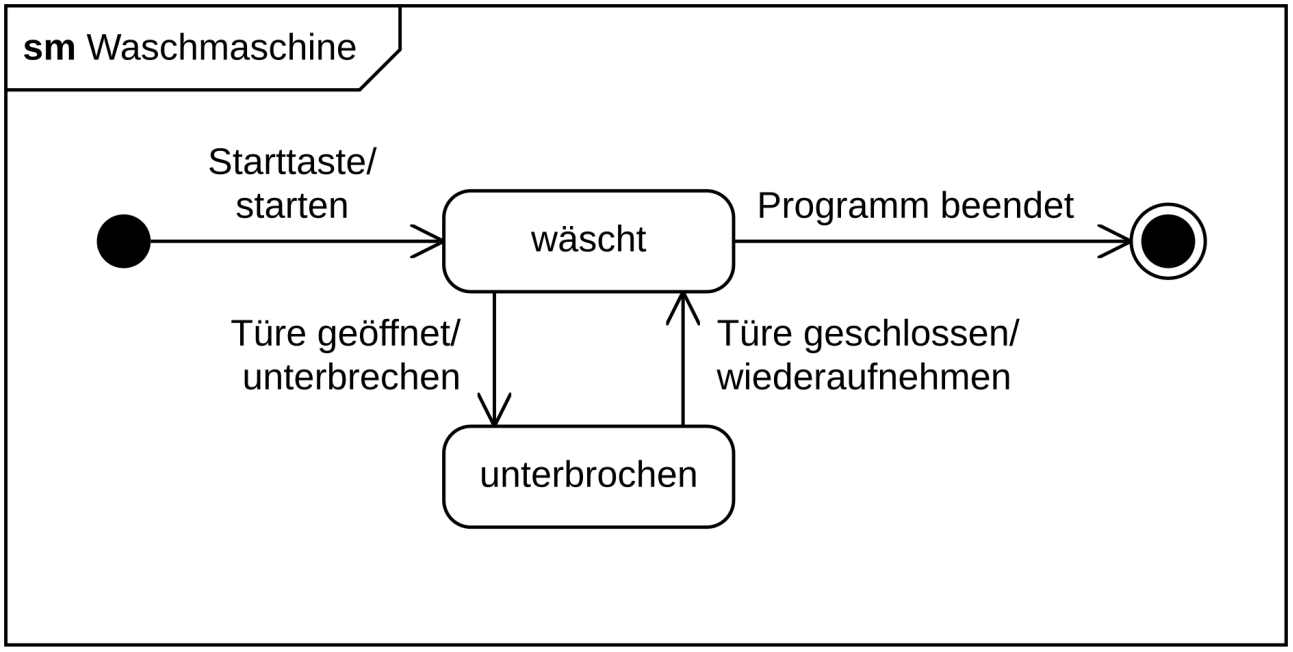

---

# Zustandsdiagramm

### Definition

Darstellung eines endlichen Automaten, um das Verhalten eines Systems zu visualisieren. Dabei werden sowohl Aktionen unterstützt, die vom Zustand des Systems und einem auslösenden Ereignis abhängen, als auch Eintritts- und Austritts-Aktionen, die eher Zustand als Aktionsorientiert sind.

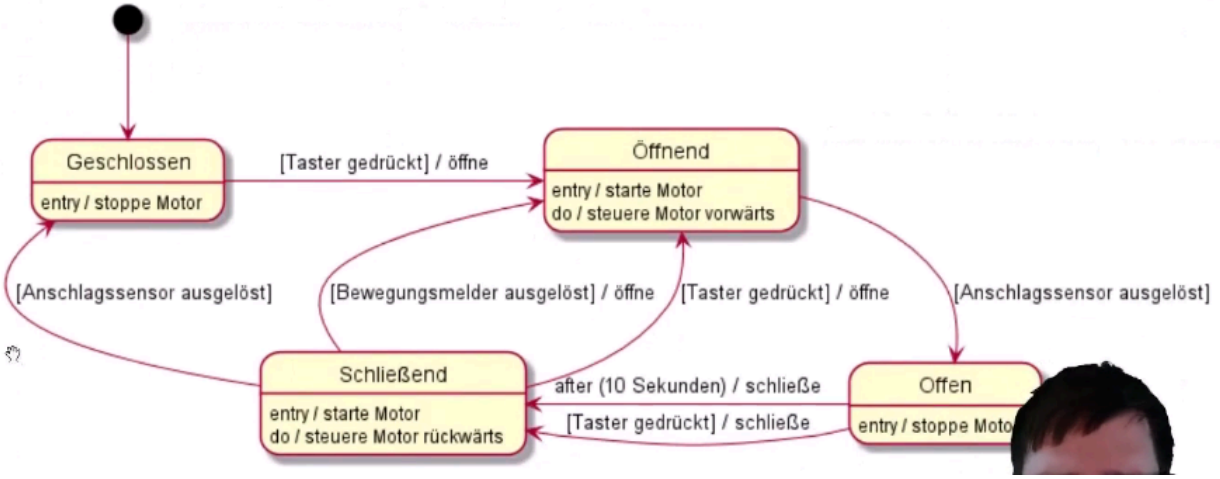
### Beispiel 1:



*Einfaches Zustandsdiagramm, aber Vorsicht: in den Rechtecken fehlt ein Horizontaler Strich!*

---

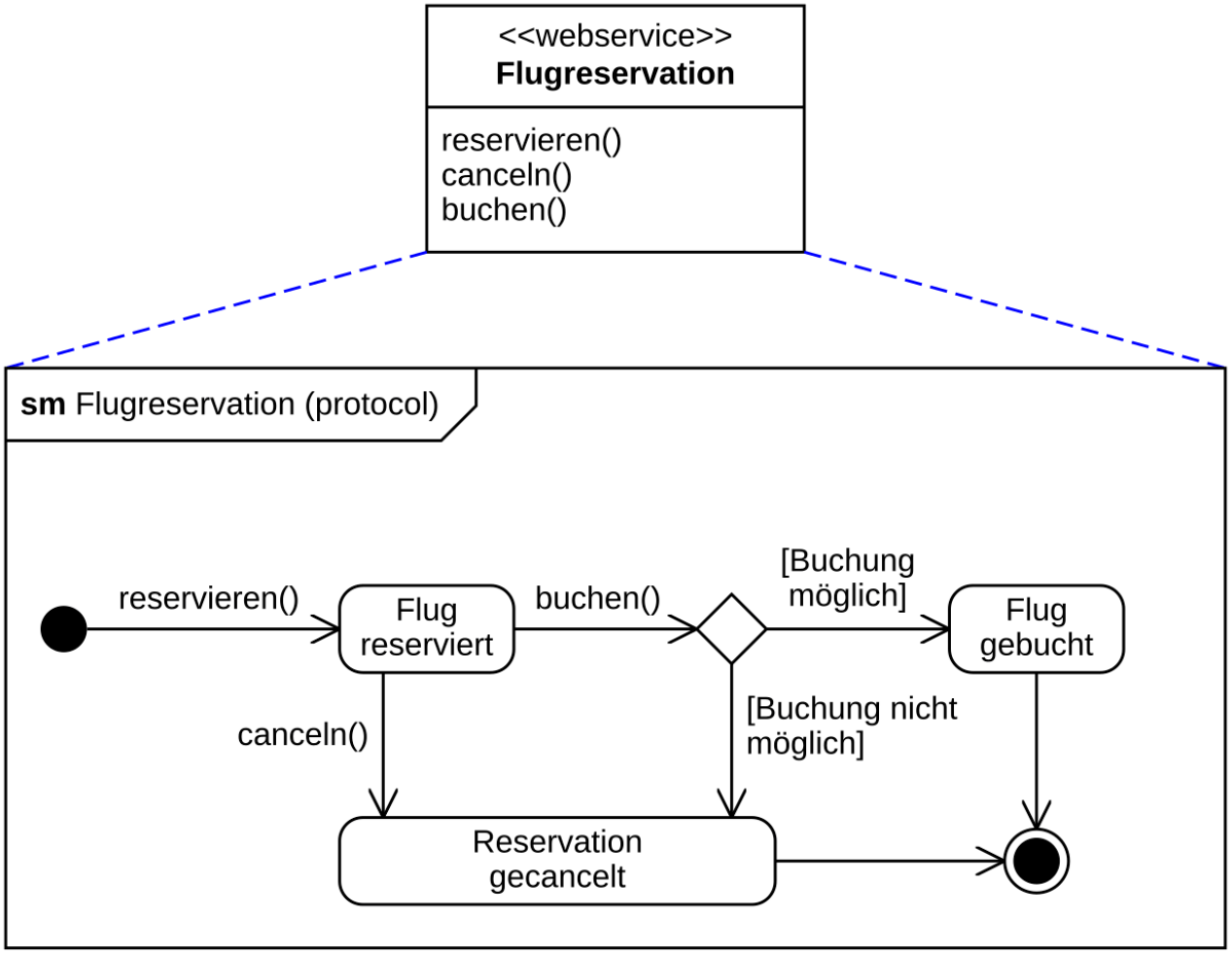
## Beispiel 2:




*Zustandsdiagramm dieses mal mit Horizontalem Strich im Rechteck*

---

## Beispiel 3:



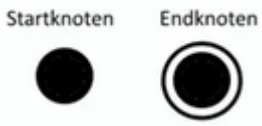
*Ein Zustandsdiagramm das sich aus einem Klassendiagramm "Flugreservation" ergibt um einen genaueren Überblick über den Vorgang "Flugreservation" zu bekommen*

---

## Symbole

### Start/Stopp

Einstieg und Ausstieg in den Ablauf.




---

### Zustände

Die einzelnen Zustände werden mit abgerundeten Rechtecken dargestellt! **KEINE OVALE!** ein horizontaler Strich durch das Rechteck ist **immer** mit einzuzeichnen! (Ja richtig, im Beispielbild 1 & 3 ist dies falsch, siehe oben)




---

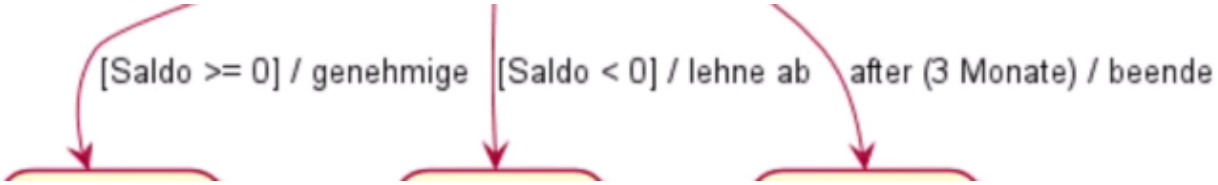
### Zustandsübergänge / Transitionen

Auch Transitionen genannt, hiermit ist der Pfeil im Diagramm gemeint. Wenn sich ein Zustand ändert so durchlebt die Maschine eine Transition, deswegen Transitionen = Pfeil.

---

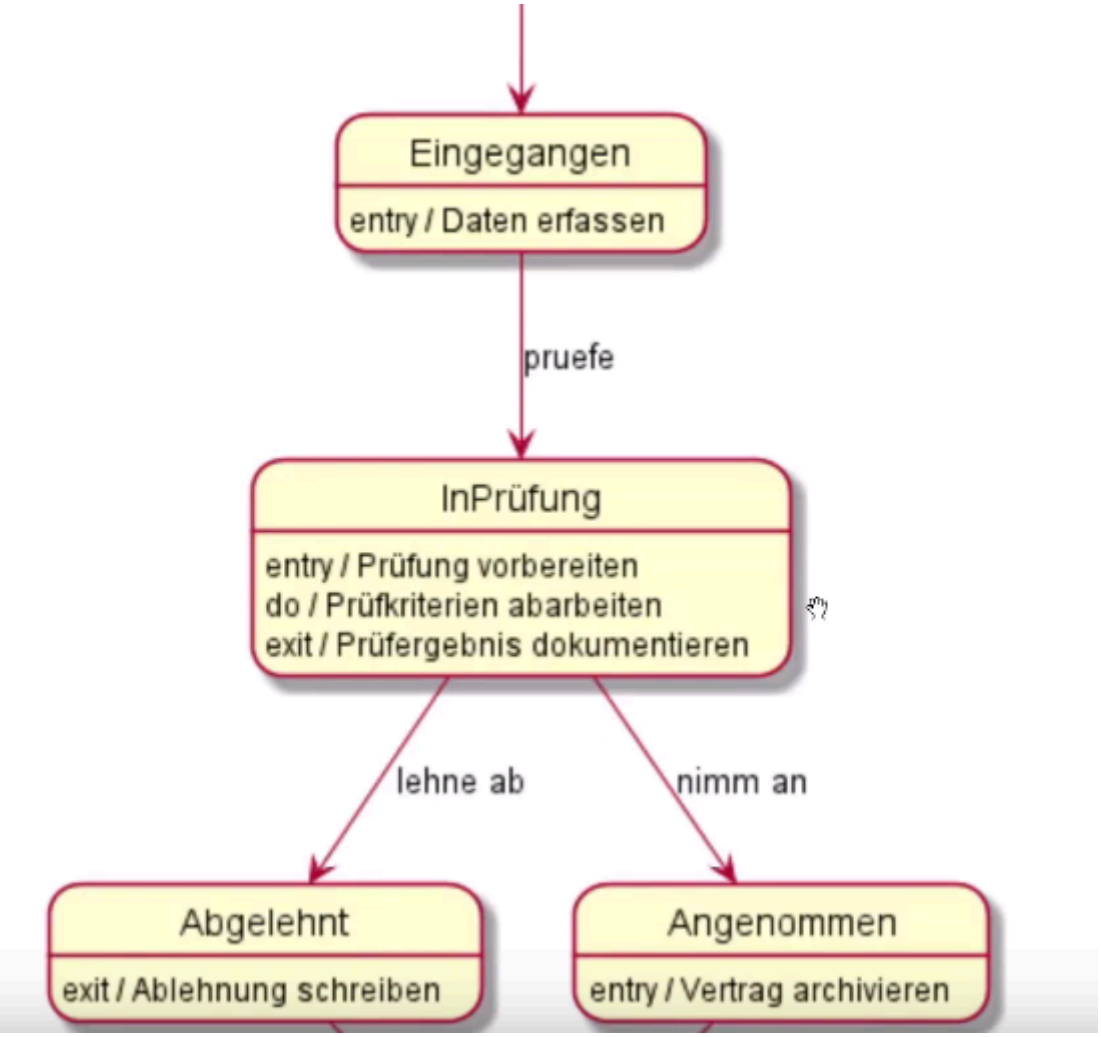
### Effekte & Guards

Transitionen können einen Effekt haben, der beim Durchlaufen auftritt. Sie können mit Guards geschützt werden. Guards geben eine **Bedingung** vor und werden in eckigen Klammern geschrieben. Auch zeitliche Ereignisse können modelliert werden.



### Verhaltensspezifikationen

Zustände können Verhalten beschreiben, das ausgeführt wird, sobald der Zustand eintritt (entry), wenn er verlassen wird (exit) oder während der gesamten Lebenszeit des Zustandes (do).
entry - Wenn sich der Zustand zum markierten Rechteck ändert, wird **immer** der "entry" ausgelöst!
do - Während der Zustand vorherrscht, wird das "do" abgearbeitet!
exit - Beim Verlassen des Zustandes wird die Aktion "exit" ausgelöst!

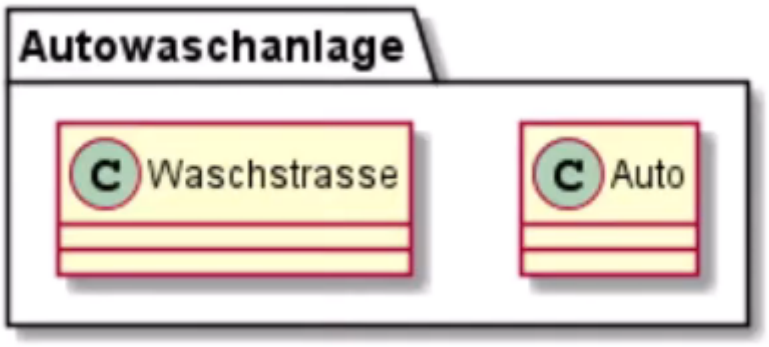



# Klassendiagramm

### Definition

Klassen stellen Baupläne für Objekte dar. In diesen Bauplänen wird *Verhalten* und *Zustand* in einer gemeinsamen Struktur erfasst. Mithilfe dieser Baupläne werden Objekte erstellt als sind identifizierbare konkrete Ausprägungen (Instanzen) dieser Baupläne. Sie verfügen über gespeicherten Zustände (die gleichen Attribute) sowie ein gemeinsames Verhalten (die *selben* Methoden). Objekte einer Klasse unterscheiden sich nur im Zustand (den Attributwerten), nicht im Verhalten.

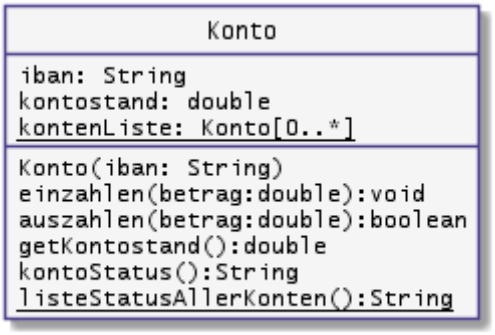
### Beispiel 1:




*Klassen (Waschstraße, Auto) können Paketen (Autowaschanlage) zugeordnet werden*

---

### Beispiel 2:



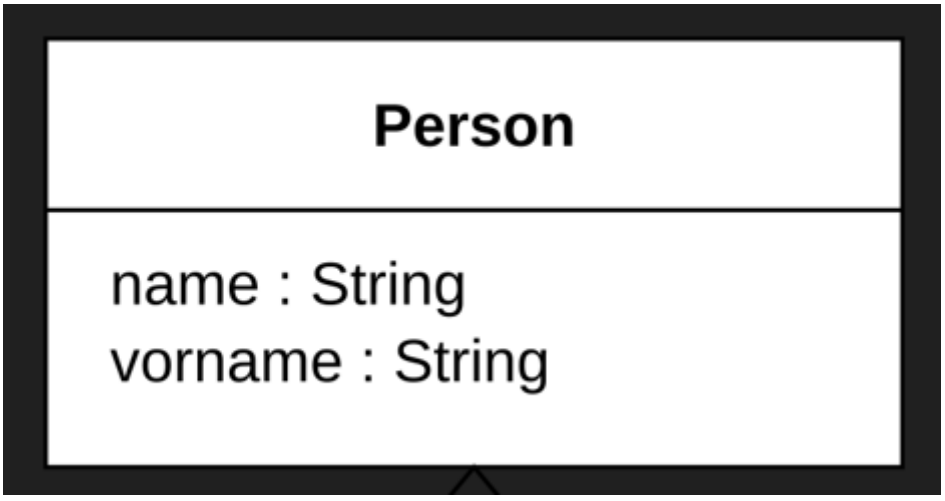

*Die Klasse eines Bankkontos*

---

### Verwendete Inhalte

#### Klassenname

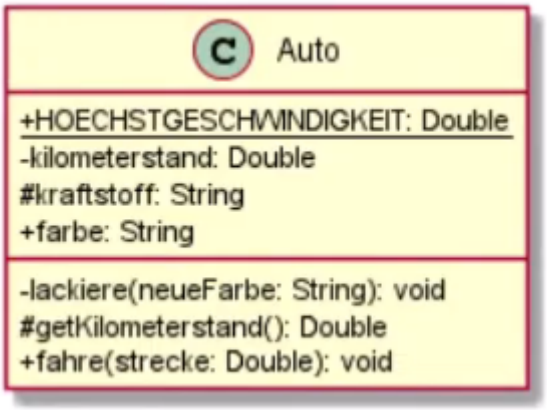
Wird ins erste Drittel des Rechteckes geschrieben. Im Beispiel unten "Person".




---

### Attribute

Werden ins zweite Drittel des Rechteckes geschrieben. Zuerst der Name des Attributes (Kilometerstand z.B.) danach der Datentyp des Attributes (double, int etc.). Richtig, das ist genau **falsch herum** wie in Java Klassen deklariert werden!




---

### Methoden

Methoden werden ins dritte Drittel des Rechteckes geschrieben. Sie stellen Aktionen zur Verfügung was die Klasse tuen kann.
Im Beispiel oben: -lackiere() oder +fahre()

---

### Sichtbarkeiten

Sichtbarkeiten werden mit entsprechenden Vorzeichen dargestellt. Sie können Attribute **sowie** Methoden vorschreiben wie die Sichtbarkeit in der Klasse ausfällt. Es gibt *public, protected, private & package*

- „+“ für *public* – (engl. öffentlich), unbeschränkter Zugriff
- „#“ für *protected* – (engl. geschützt), Zugriff nur von der Klasse sowie von Unterklassen
- „–“ für *private* – (engl. privat), nur die Klasse selbst kann es sehen
- „~“ für *package* – (engl. Paket), innerhalb des Pakets sichtbar

---

### Statische Member

Statische Member werden unterstrichen! Im Englischen *static*.

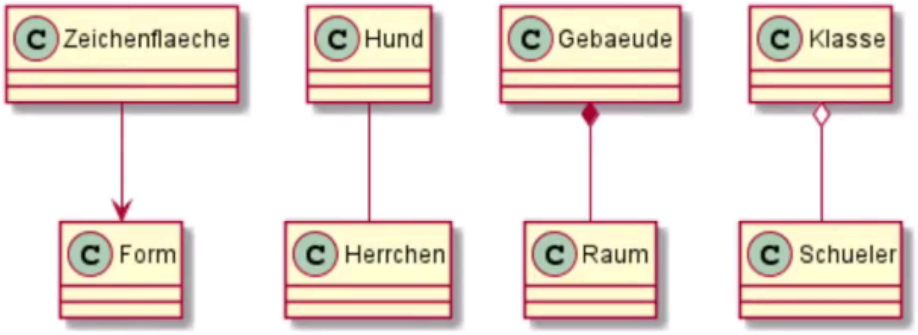
---

### Assoziation, Aggregation, Komposition & Generalisierung

Mithilfe des Klassendiagramms lassen sich Beziehungen sehr leicht abbilden.
Im Beispiel unten sind alle vier Beziehungen beschrieben, diese sind:

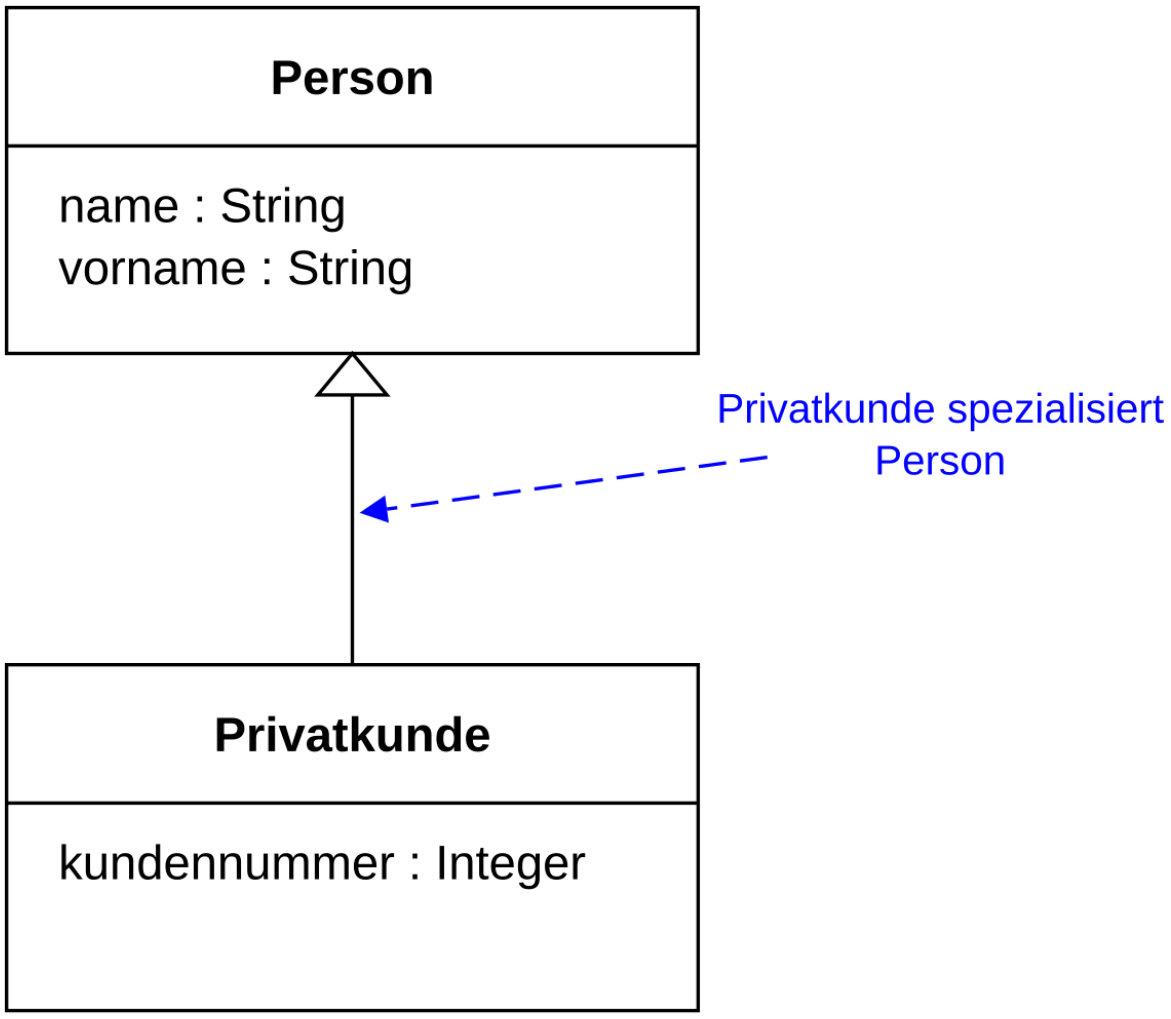
Zeichenfläche -> Form = ist eine Generalisierung
Hund -> Herrchen = ist eine Assoziation
Gebäude -> Raum = ist eine Komposition
Klasse -> Schüler = ist eine Aggregation

- Assoziationen: Klassen können uni- und bidirektionale Assoziationen zueinander haben.
- Aggregation: Eine Teil-Ganzes-Beziehung mit selbstständigem Teil.
- Komposition: Eine Teil-Ganzes-Beziehung mit unselbstständigem Teil.



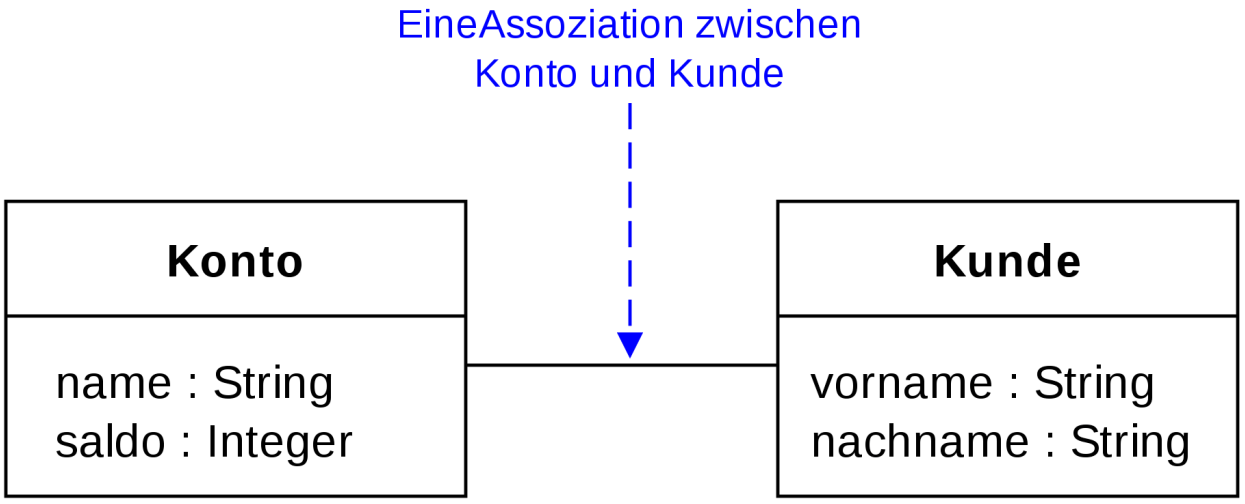

---

**Generalisierung** (Ist mit Pfeil!)
Eine Generalisierung in der UML ist eine gerichtete Beziehung zwischen einer *generelleren* und einer *spezielleren* Klasse. Exemplare der spezielleren Klasse sind damit auch Exemplare der generelleren Klasse. Konkret bedeutet dies, dass die speziellere Klasse implizit über alle Merkmale (Struktur- und Verhaltensmerkmale) der generelleren Klasse verfügt – *implizit* deshalb, weil diese Merkmale in der spezielleren Klasse nicht explizit deklariert werden. Man sagt, dass die speziellere Klasse sie von der generelleren Klasse „erbt“ oder „ableitet“.
Eine Generalisierung wird als durchgezogene Linie zwischen den beiden beteiligten Classifiern dargestellt. Am Ende mit dem generelleren Classifier wird eine geschlossene, nicht ausgefüllte Pfeilspitze gezeichnet.
In gängigen objektorientierten Programmiersprachen entspricht dies dem Konzept der Vererbung, wobei der Pfeil auf die Oberklasse zeigt.




---

**Assoziation** (Ist ohne Pfeil, nur Linie!)
Eine Assoziation beschreibt eine Beziehung zwischen zwei oder mehr Klassen. An den Enden von Assoziationen sind häufig Multiplizitäten vermerkt. Diese drücken aus, wie viele dieser Objekte in Relation zu den anderen Objekten dieser Assoziation stehen.




---

## Komposition & Aggregation

In UML-Klassendiagrammen beschreiben Komposition und Aggregation Beziehungen zwischen Klassen, insbesondere Ganzes-Teil-Beziehungen.

1. Aggregation (leere Raute ◇)

- **Schwache Beziehung** zwischen Ganzem und Teil.
- Das Teilobjekt kann auch ohne das Ganze existieren.
- Beispiel: Ein **Team** besteht aus **Spielern**, aber ein Spieler kann auch ohne ein Team existieren.

2. Komposition (ausgefüllte Raute ◆)

- **Starke Beziehung** zwischen Ganzem und Teil.
- Das Teilobjekt kann **nicht ohne** das Ganze existieren.
- Beispiel: Ein **Auto** hat **Räder**, aber ohne das Auto machen die Räder keinen Sinn.

**Merkregel**:

- **Komposition = starke Abhängigkeit** (Teil stirbt mit dem Ganzen).
- **Aggregation = lose Verbindung** (Teil kann auch eigenständig existieren).



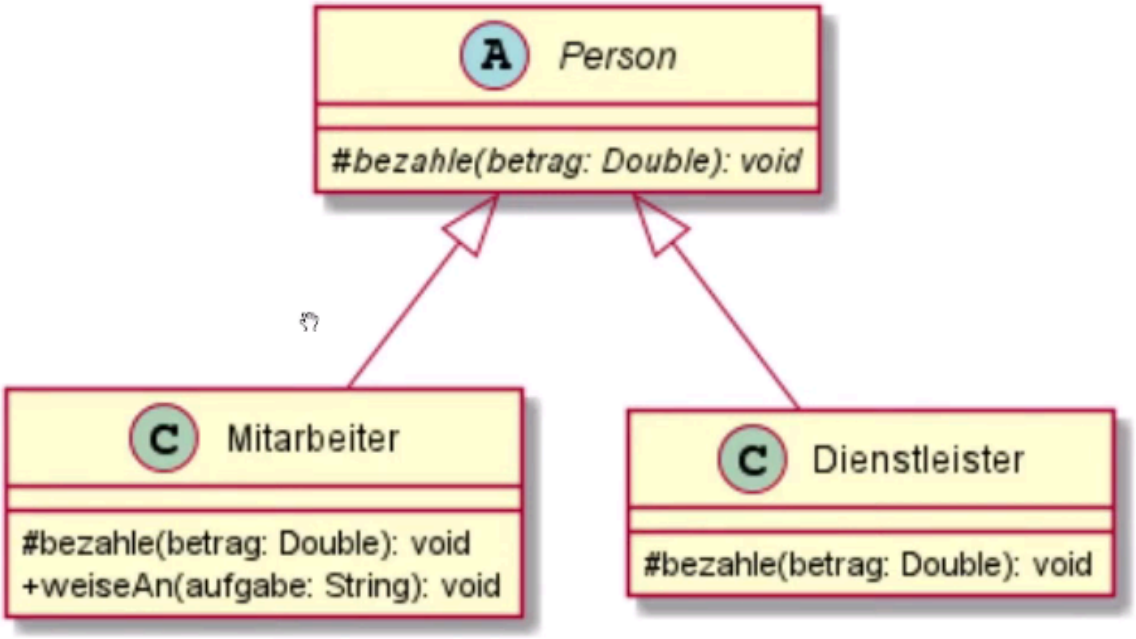
---

## Vererbung und Interfaces

Erst für AP Teil 2!

## Vererbung

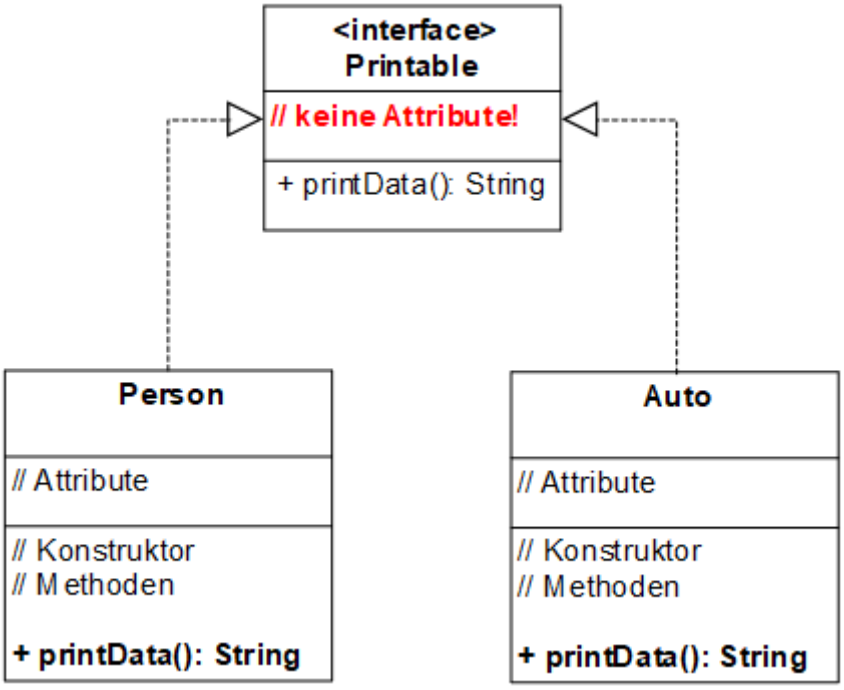
Vererbung ist im unteren Bild zu sehen. Pfeile zeigen auf die "obere" Klasse. Ähnlich wie im Java Code, dort würde stehen "*Mitarbeiter extends Person*". Somit zeigt der Pfeil auf "*Person*". Die Klasse "*Person*" weiß nichts von erbenden Klassen. Im Beispiel auch noch möglich: "*Dienstleister extends Person*".




---

## Interfaces

Wird mit einem Strichpfeil abgebildet. Die Klassen die ein Interface implementieren "zeigen" mit einem Pfeil auf das Interface. Ähnlich wie im Java Code, dort würde in der Klasse "*Person*" stehen "*Person implements Printable*" oder auch "*Auto implements Printable*", die Klasse die ein Interface implementiert "zeigt" auf das Interface.

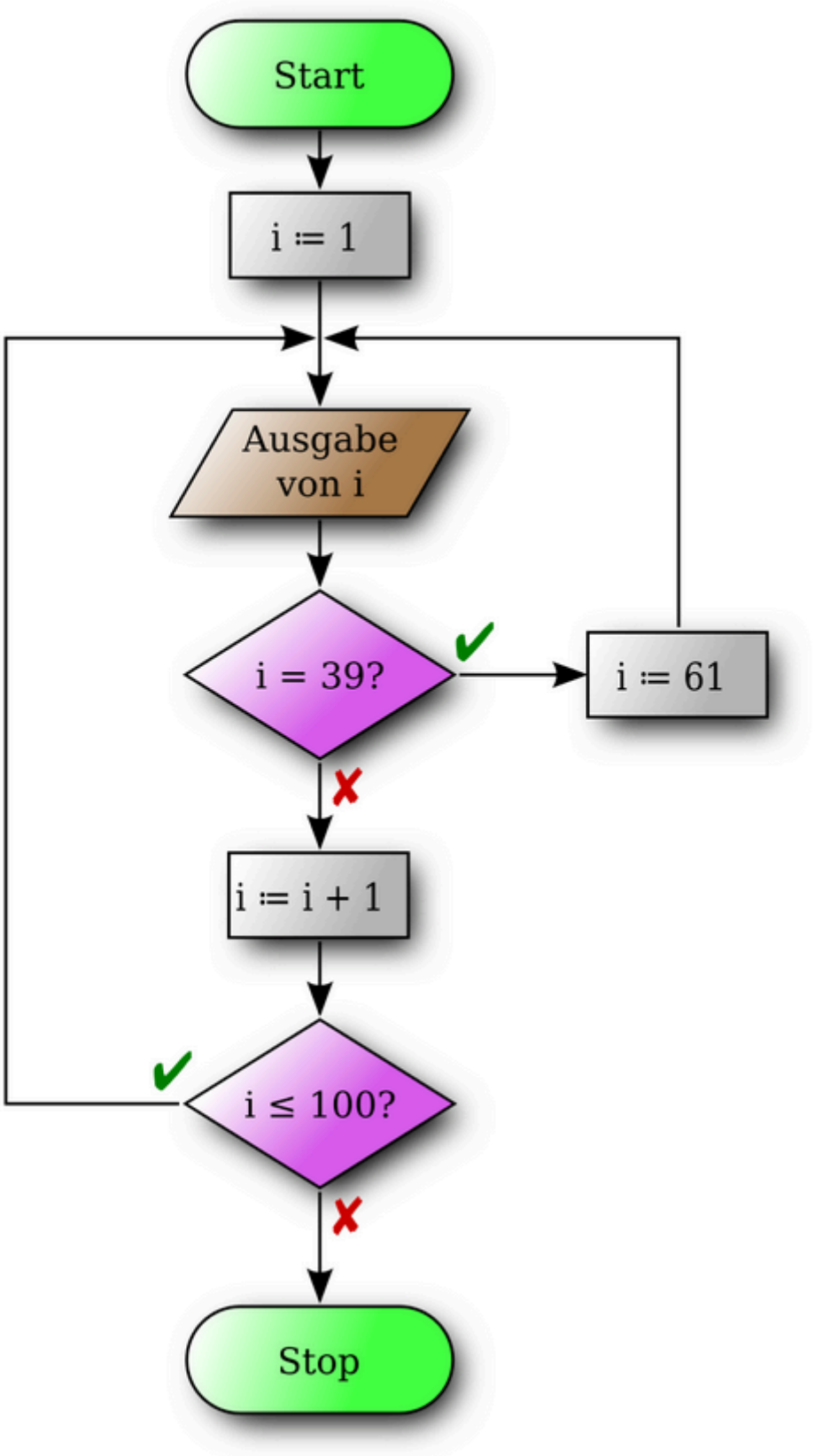


# Programm Ablauf Plan

## Definition

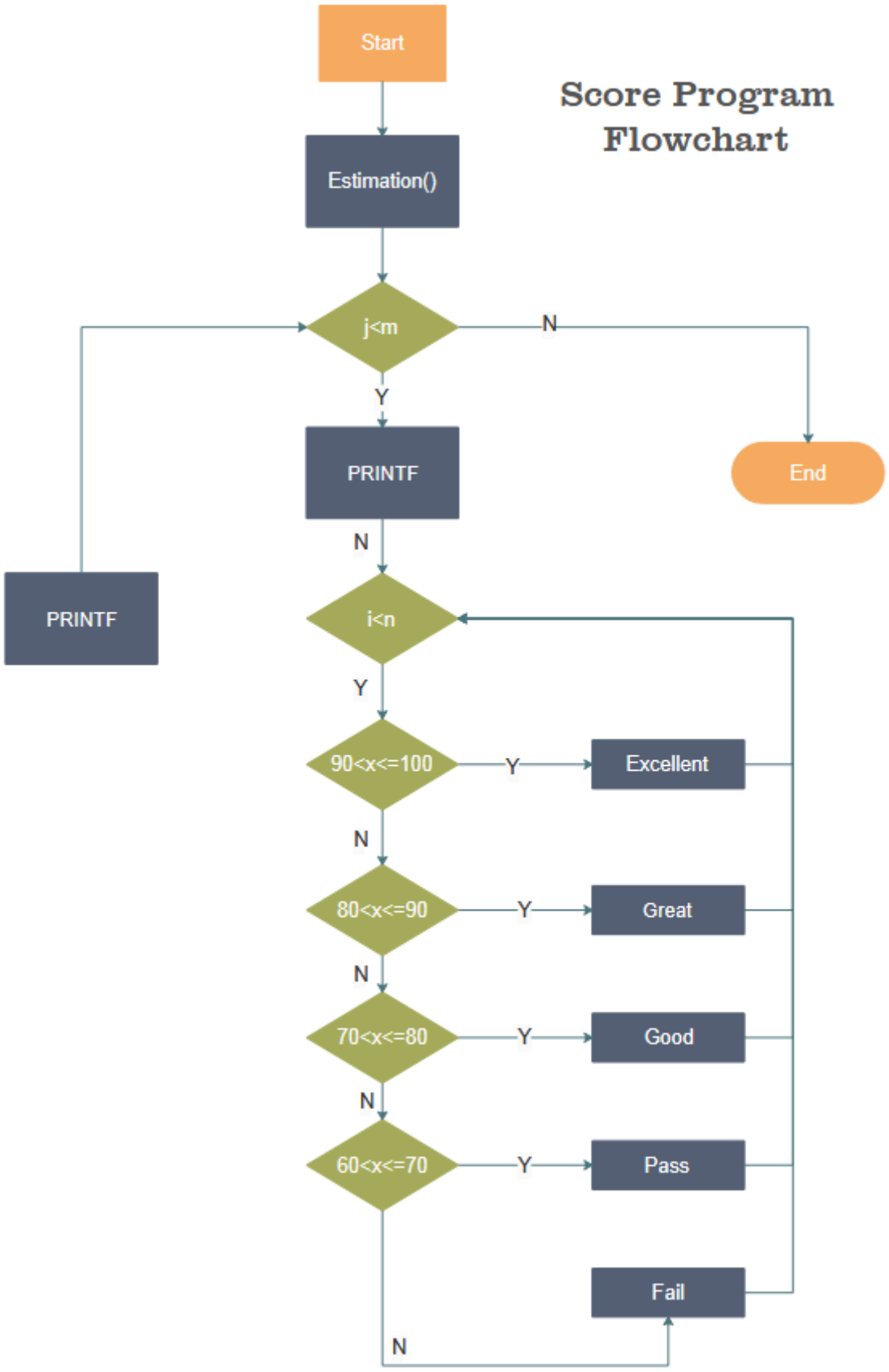
Ein Programmablaufplan (PAP) ist ein Ablaufdiagramm für ein Computerprogramm, das auch als Flussdiagramm (engl. flowchart) oder Programmstrukturplan bezeichnet wird. Es ist eine grafische Darstellung zur Umsetzung eines Algorithmus in einem Programm und beschreibt die Folge von Operationen zur Lösung einer Aufgabe.
Die Symbole für Programmablaufpläne sind nach der DIN 66001 genormt.
Programmablaufpläne werden oft unabhängig von Computerprogrammen auch zur Darstellung von Prozessen und Tätigkeiten eingesetzt. Im Bereich der Softwareerstellung werden sie nur noch selten verwendet.
Das Konzept der Programmablaufpläne stammt, ebenso wie das etwas jüngere Nassi-Shneiderman-Diagramm (Struktogramm), aus der Zeit des imperativen Programmierparadigmas. Bei der Abbildung objektorientierter Programmkonzepte durch UML finden erweiterte Programmablaufpläne (Aktivitätsdiagramme) Anwendung.

**Beispiel 1:**



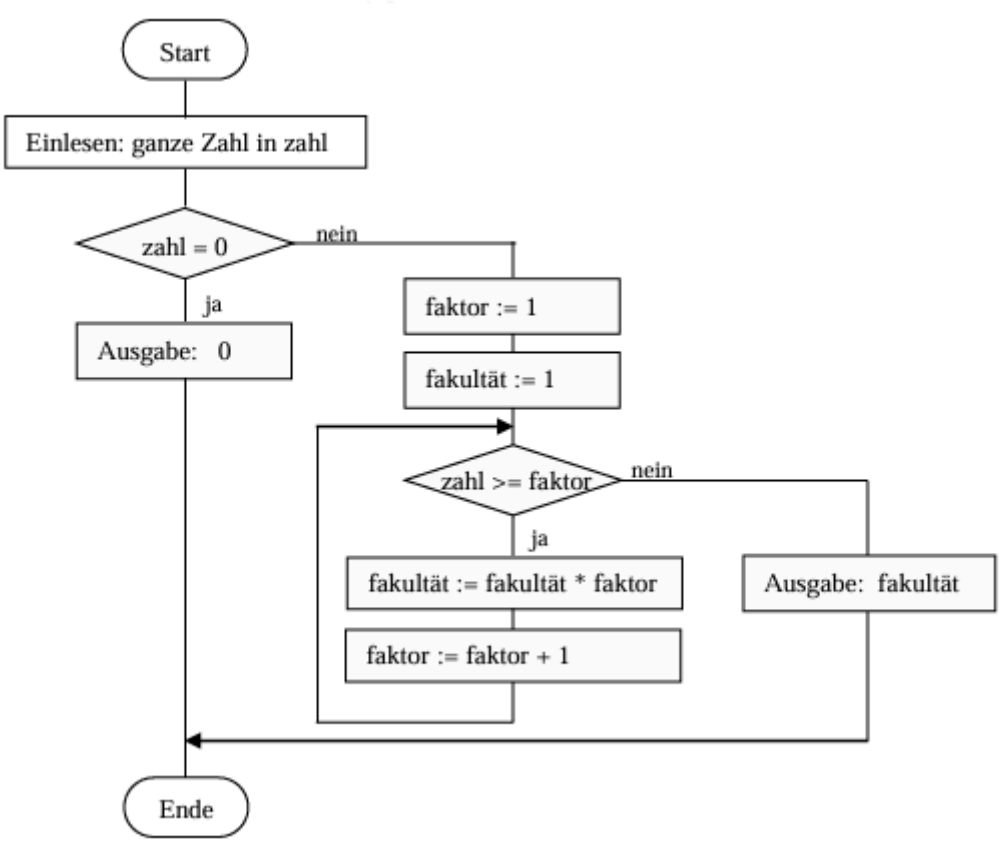

*Beispiel eines Flussdiagramms*

**Beispiel 2:**




*Ein weiteres PAP*

### Beispiel 3:




*Ein dritter Programmablaufplan*

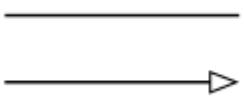
---

## Symbole

### Oval bzw. Rechteck mit runden Ecken: Terminator

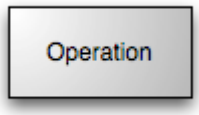

---

### Pfeil, Linie: Verbindung zum nächstfolgenden Element



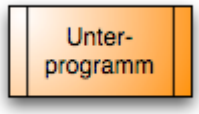
---

### Rechteck: Operation (Tätigkeit)



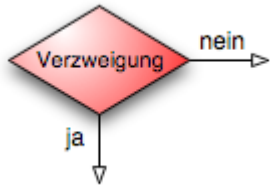

---

### Rechteck mit doppelten, vertikalen Linien: Unterprogramm ausführen



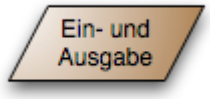

---

### Raute: Verzweigung / Entscheidungen




---

### Parallelogramm: Ein- und Ausgabe (ist in der DIN 66001 von 1982 zwar definiert, soll jedoch nicht für PA verwendet werden)



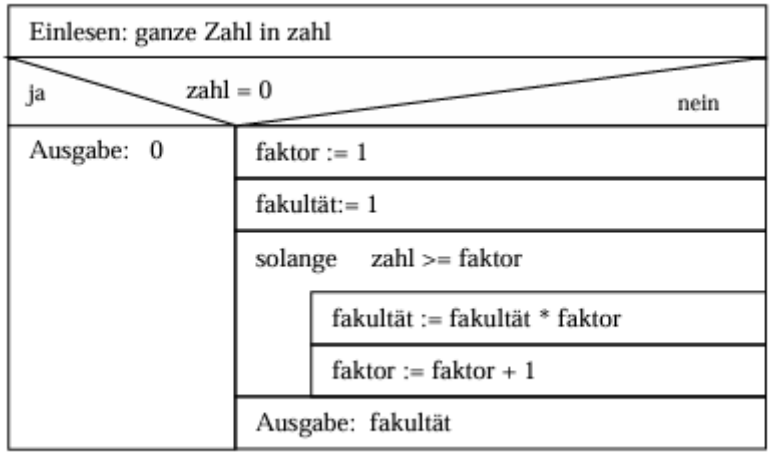

---

# Struktogramm (Nassi-Shneiderman-Diagramm)

### Definition

Ein Nassi-Shneiderman-Diagramm ist ein Diagrammtyp zur Darstellung von Programmentwürfen im Rahmen der Methode der strukturierten Programmierung. Es wurde 1972/1973 von Isaac Nassi und Ben Shneiderman entwickelt und ist in der DIN-Norm 66261 festgelegt.
Da Nassi-Shneiderman-Diagramme Programmstrukturen und Kontrollstrukturen darstellen, werden sie auch als Struktogramme bezeichnet.

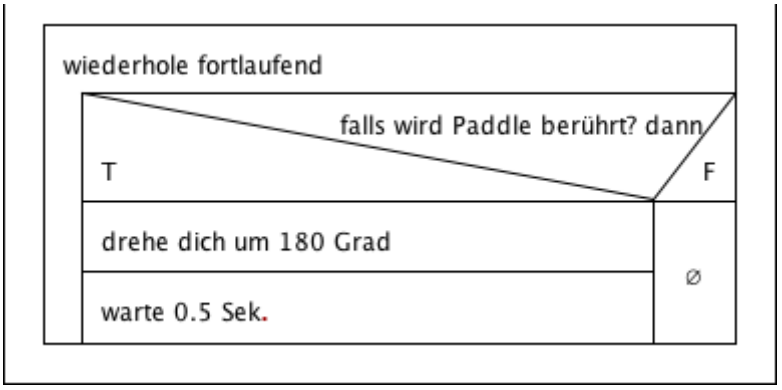
## Beispiel 1:




*Ein übersichtliches Struktogramm*

---

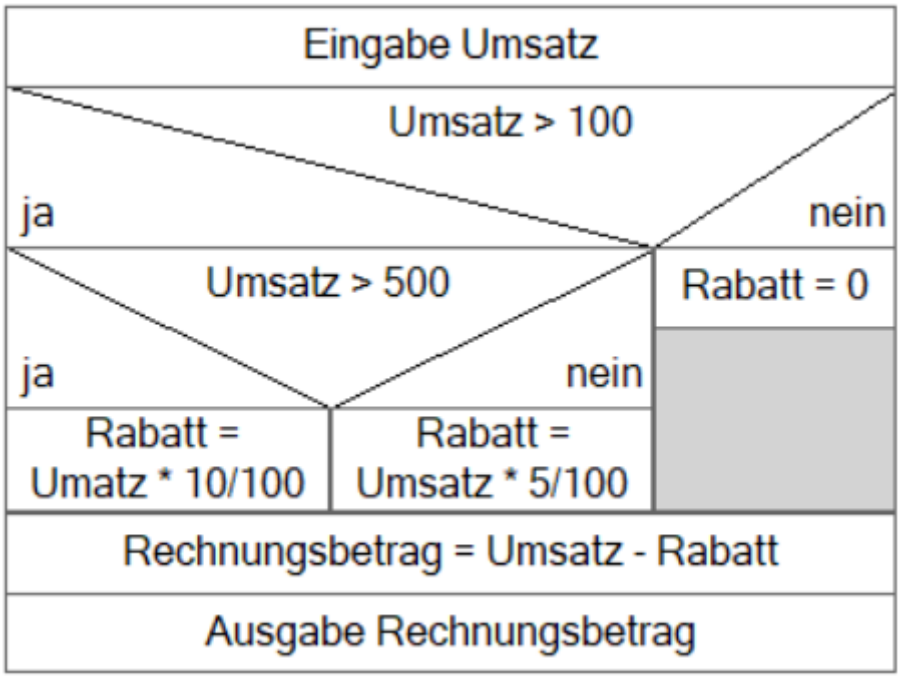
## Beispiel 2:




*Ein Struktogramm ohne Zahlen nur Pseudoanweisungen*

---

### Beispiel 3:

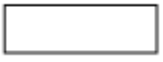



*Struktogramm für die Berechnung eines Rabattes für Kunden je nach Höhe des Umsatzes*

---

## Symbole

### Anweisung

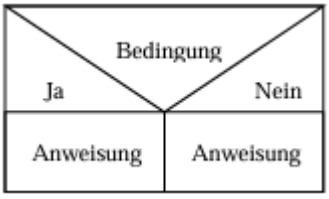


---

### Sequenz (Folgestruktur)

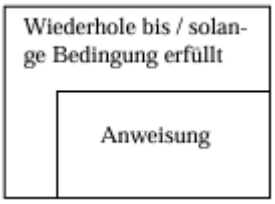

---

### Bedingte Verzweigung (einseitig oder zweiseitig)



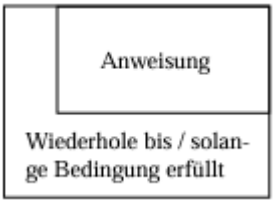

---

## Wiederholung (Schleife) Kopfgesteuert



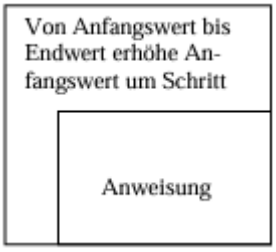

---

## Wiederholung (Schleife) Fußgesteuert



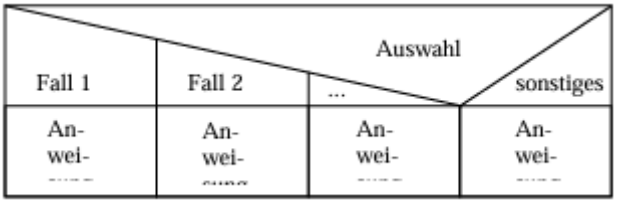

---

## Zählschleife



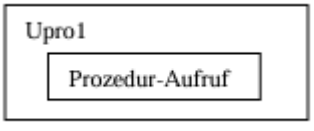

---

## Fallauswahl (Mehrfachverzweigung)




---

## Unterprogramm




---

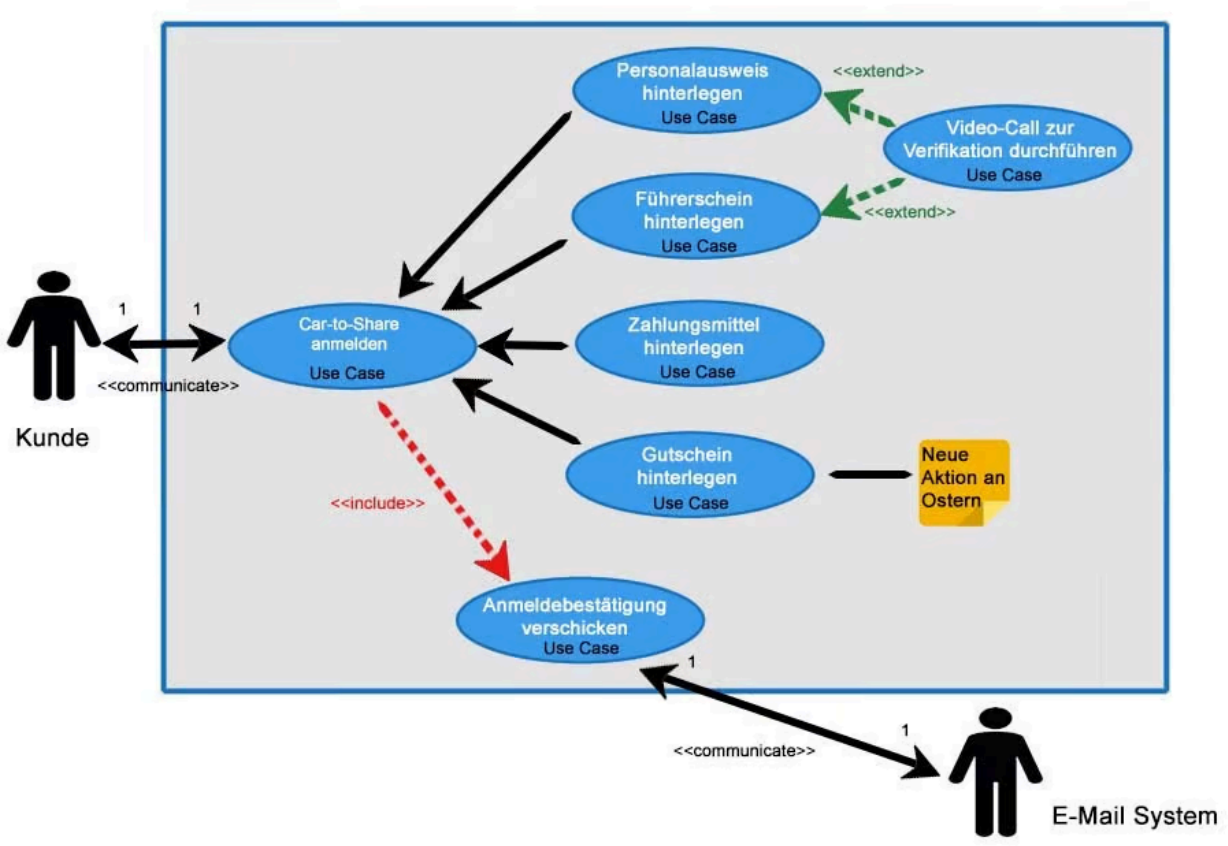
# Use-Case Diagramm

## Definition

Ein Use Case Diagramm ist eine grafische Repräsentation von Anwendungsfällen inklusive deren Beziehungen zur Umwelt und zu anderen Anwendungsfällen. Damit beschreibt es in einer hohen Abstraktion, welche Funktionen und Dienste ein System für einen Anwender bereitstellt.

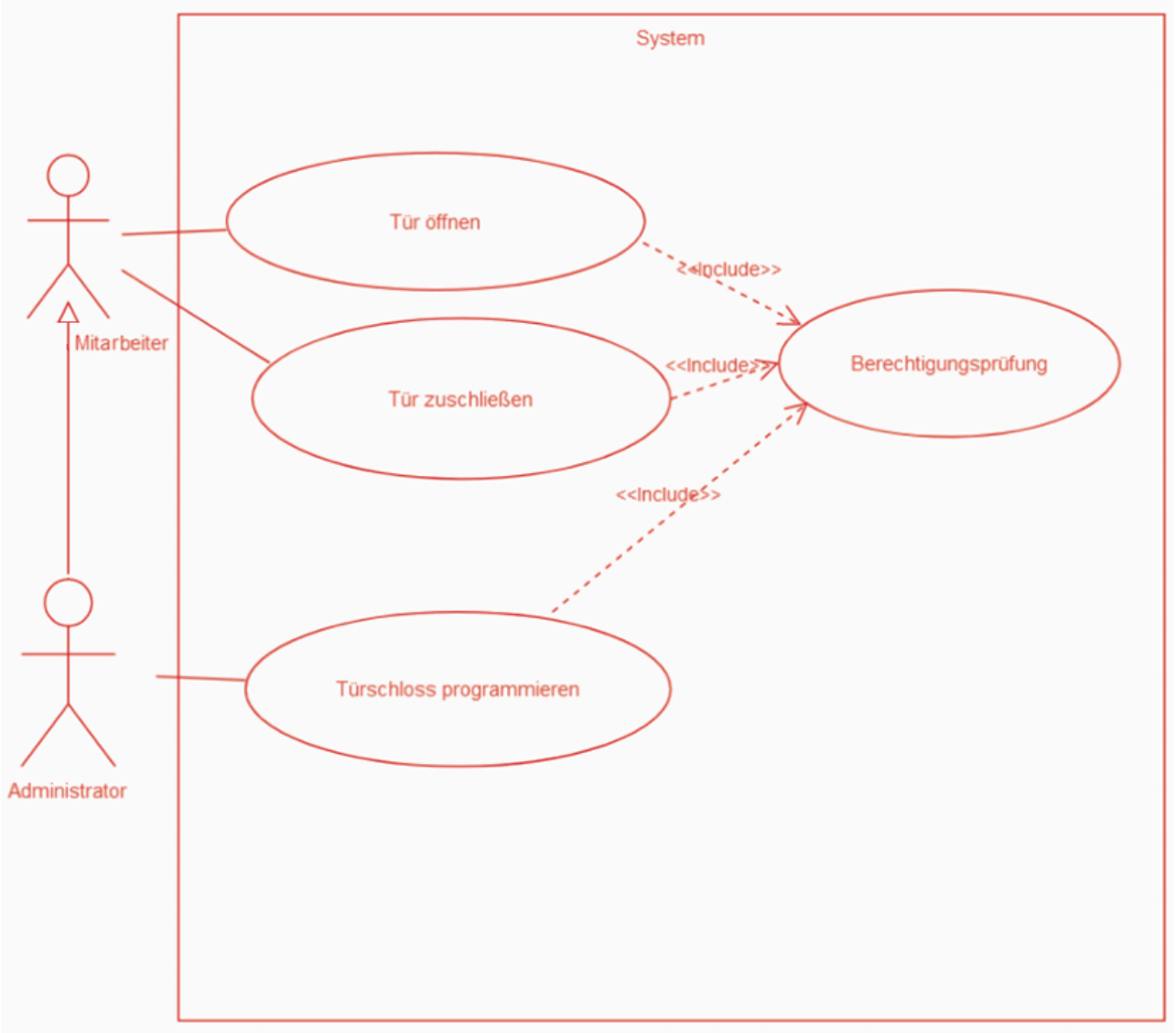
Mit einem Use Case Diagramm – auch Anwendungsfalldiagramm oder Nutzfalldiagramm genannt – werden **weder** die Abläufe des Systems beschrieben, noch die Reihenfolge der Funktionen oder Dienste dargestellt. Ein Anwendungsfalldiagramm visualisiert lediglich die Zusammenhänge zwischen einer Menge von Use Cases und den involvierten Akteuren. Damit eignet es sich sehr gut zur Anforderungsanalyse, also zur Ermittlung oder Verfeinerung von Anforderungen.

## Beispiel 1:



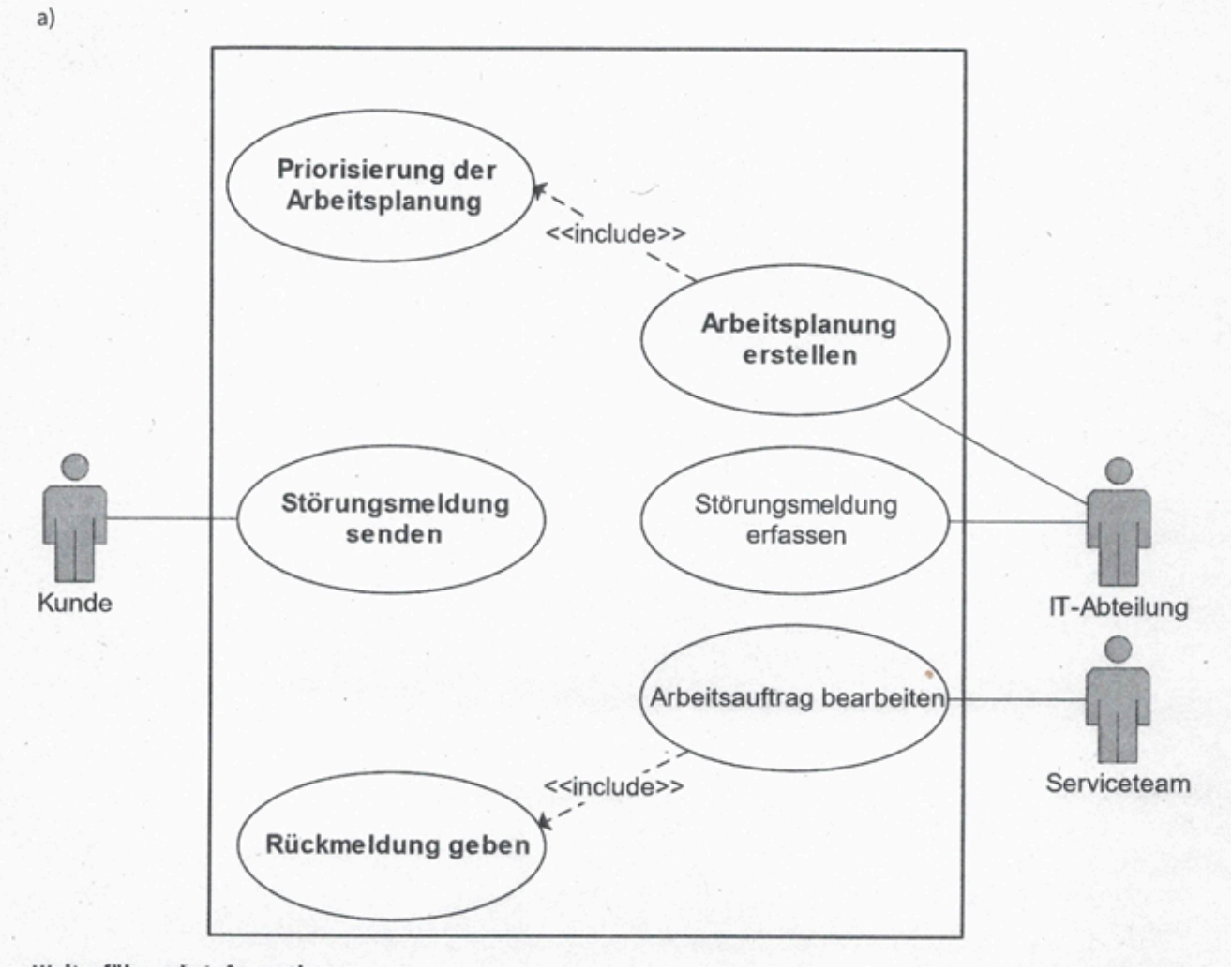

*Einfaches Use-Case Diagramm, Männchen am Rand kann auch ein Service sein (E-Mail System)!*

**Beispiel 2:**




*Use-Case Diagramm aus der AP Teil1 Herbst 2024*

**Beispiel 3:**




*Use-Case Diagramm aus der AP Teil1 Herbst 2023*

---

**Elemente im Use-Case Diagramm:**

- Das **System** ist an sich kein logisches Modellelement, sondern grenzt den Kontext ab. Dieser Systemkontext wird durch einen Rahmen im Diagramm repräsentiert, in dem das System seine Funktionen und Dienste zur Verfügung stellt.
- Der **Akteur** befindet sich außerhalb des Systems. Er wird als Strichmännchen gezeichnet und kann eine konkrete Person aber auch ein abstraktes Element wie bspw. ein Sensor sein. Die Verfeinerung von Akteuren kann per UML Profil erfolgen. Wird ein Akteur definiert, muss dieser auch immer mit mindestens einem Use Case in Verbindung stehen. Sind mehrere Akteure involviert, wird dies durch die Multiplizität ausgedrückt. Ohne weitere Angabe ist diese standardmäßig 1.
- Ein **Anwendungsfall** wird meist als Ellipse visualisiert, wobei sein Name auch außerhalb der Ellipse stehen kann. Ein Diagramm kann mehrere Anwendungsfälle umfassen. Ist ein Use Case nur durch andere Anwendungsfälle ausführbar, wird er als abstrakt bezeichnet. Auch für Use Cases lässt sich die Multiplizität darstellen; sie gibt Auskunft darüber, wie häufig der Anwendungsfall gleichzeitig ausgeführt werden kann.

- Zwischen Akteuren und Anwendungsfällen und zwischen Anwendungsfällen bestehen **Beziehungen**.
- Mit **Notizen** lassen sich Informationen hinzufügen, um das Verständnis zu erhöhen. Sie werden mit einem Rechteck dargestellt, dessen obere rechte Ecke eingeknickt ist. Eine gestichelte Linie verbindet die Notiz mit dem zu erklärenden Element.