

# Einführung und erste Grundlagen der Modellierung mit UML

Dozent: Prof. Dr. Michael Eichberg  
Kontakt: [michael.eichberg@dhbw.de](mailto:michael.eichberg@dhbw.de), Raum 149B  
Version: 1.1.1

---

Folien: [https://delors.github.io/prog-modellierung\\_mit\\_uml/folien.de.rst.html](https://delors.github.io/prog-modellierung_mit_uml/folien.de.rst.html)  
[https://delors.github.io/prog-modellierung\\_mit\\_uml/folien.de.rst.html.pdf](https://delors.github.io/prog-modellierung_mit_uml/folien.de.rst.html.pdf)  
Fehler melden: <https://github.com/Delors/delors.github.io/issues>

# UML (Unified Modeling Language)

- UML (Unified Modeling Language) ist eine standardisierte Modellierungssprache, die zur Beschreibung von Software-Systemen verwendet wird.
- UML besteht aus verschiedenen Diagrammtypen, die unterschiedliche Aspekte eines Systems beschreiben.
- UML wird in der Softwareentwicklung verwendet, um Konzepte und Entwürfe zu kommunizieren und zu besprechen. Eine detaillierte Modellierung ist nicht (mehr) das Ziel/üblich.
- Die Hoffnungen/Erwartungen, die an UML Anfang der 2000er gestellt wurden, wurden nicht erfüllt. Seit 2017 ist die Version 2.5.1 aktuell.

## Warnung

Wenn Sie UML verwenden, dann verwenden Sie die Notationen spezifikationskonform, da sonst der Sinn der Notation (vollständig) verloren geht oder es sogar zu Missverständnissen kommt.

# 1. Aktivitätsdiagramme

---

# Aktivitätsdiagramme - Einführung

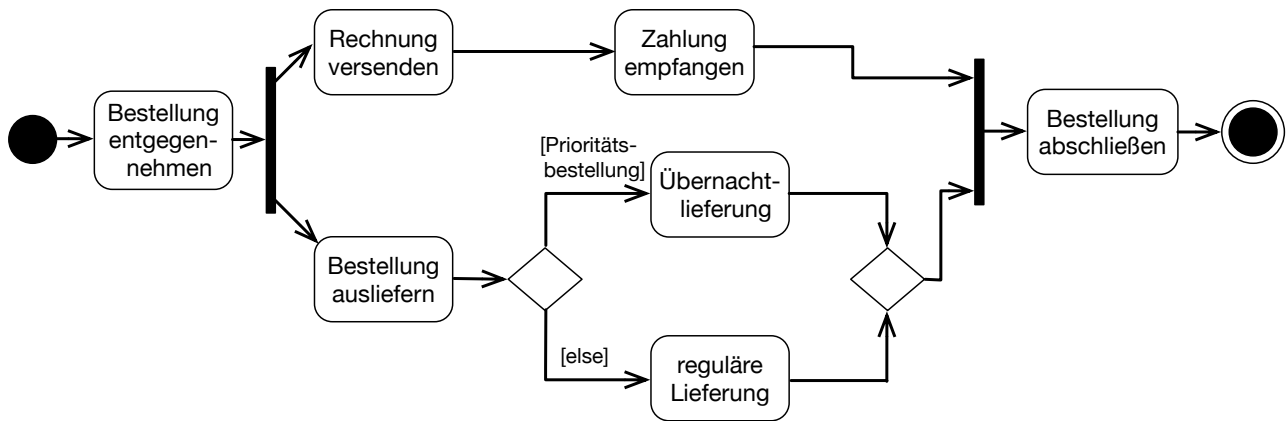
Zentraler Aspekt: Wie realisiert mein System ein bestimmtes Verhalten?

- dienen der Modellierung von Abläufen

(Von dem Ablauf einer einfachen Operation bis hin zu einem kompletten Geschäftsvorfall.)

- können Nebenläufigkeiten und Entscheidungen modellieren

# Aktivitätsdiagramm für die Abwicklung einer Bestellung



## Zentrale Elemente eines Aktivitätsdiagramms:

- die Aktionen (abgerundete Rechtecke) steht für eine Operation, die im Rahmen des Ablaufs ausgeführt wird und in dem Kontext des Diagramms nicht weiter verfeinert wird.
- der Startknoten (gefüllter Kreis); es kann mehrere Startknoten geben.
- der Endknoten (gefüllter Kreis mit einem Rand) beendet die Aktivität
- Kanten (gerichtete Linien) verbinden die Elemente (Aktionen) und beschreiben den Ablauf. Kanten können mit Bedingungen (in eckigen Klammern) versehen werden.
- Entscheidungen (Rauten) beschreiben, dass der Ablauf in Abhängigkeit von einer Bedingung unterschiedlich weitergeht. Dienen auch dazu mehrere alternative Pfade zusammenzuführen.
- Parallelisierungs- und Synchronisierungspunkte ( Fork und Join ) (schwarze Balken) beschreiben, dass der Ablauf an dieser Stelle in mehrere Pfade aufgeteilt wird und später wieder zusammengeführt wird.

# Übung

## 1.1. Modellieren Sie ein Aktivitätsdiagramm für die Berechnung der Fakultät

Nehmen Sie ggf. den Code aus der Musterlösung zur Übung als Grundlage.

# Übung

## 1.2. Bearbeitung einer Bewerbung modellieren

Modellieren Sie die Bearbeitung einer Bewerbung aus Sicht eines Unternehmens mit Hilfe eines Aktivitätsdiagramms.

Bedenken Sie dabei Aspekte, wie die Prüfung der formalen Voraussetzungen, Prüfung durch die Fachabteilung, die Einladung zu einem Vorstellungsgespräch, die Absage oder die Annahme der Bewerbung.

Identifizieren Sie ggf. Möglichkeiten für eine parallele Verarbeitung.

## 2. Klassendiagramme



Curtis Gesetz: [...] Gute Entwürfe erfordern fundierte Anwendungskenntnisse.

—Albert Endres and Dieter Rombach; *A Handbook of Software and Systems Engineering*;  
Addison Wesley 2003

# Klassen und Objekte

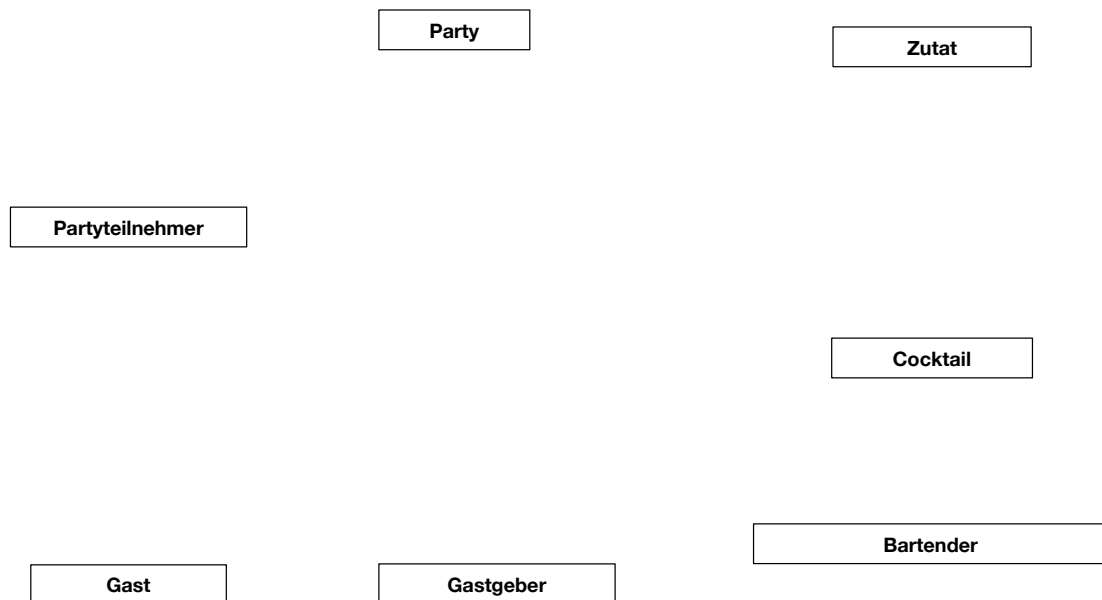
Eine **Klasse** beschreibt *eine Gruppe von* **Objekten** mit:

1. derselben Semantik,
2. denselben Eigenschaften und
3. demselben Verhalten.

D. h. eine Klasse definiert einen Typ.

Konkrete Ausprägungen dieses Typs sind die Objekte.

# Modellierung einer Party - Klassen [1]



---

Wir haben erst einmal nur die Klassen identifiziert/modelliert, die für Parties zentral sind.

Hierbei repräsentieren die Klassen verschiedene „Dinge“:

- Eine Party als virtuelles Konstrukt, das eine bestimmte Anzahl von Partyteilnehmern hat.
- Einen Gast, der an einer Party teilnimmt.
- Ein Cocktail, welcher aus verschiedenen (konkreten) Zutaten besteht.
- Ein Partyteilnehmer welcher eine Abstraktion für Gäste und Gastgeber darstellt.

---

[1] Das Beispiel ist stark angelehnt an Abb. 6.1 aus UML2 Glasklar, Hanser Verlag




# Attribute

- Attribute sind logische Datenwerte eines Objekts und haben immer einen Datentyp.
- Die Attribute in einem Modell sollten vorzugsweise „primitive“ Datentypen sein.
- Häufig macht es Sinn Mengen (x Liter, y Kilogramm, etc.) als Klassen zu modellieren, um Einheiten zuordnen zu können.

Z. B. sollte der Datentyp des Attributs „Betrag“ einer Zahlung die Währung angeben.

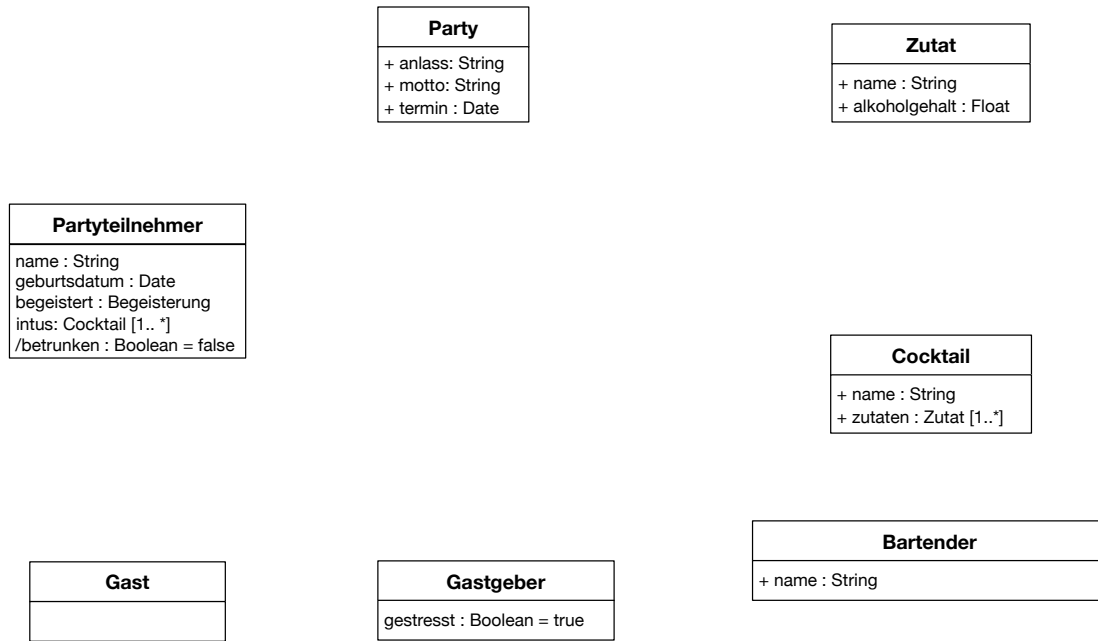
- Attribute können weiterhin:
  - abgeleitet sein und ggf.
  - Defaultwerte haben sowie
  - Sichtbarkeiten haben.

---

Sehr häufig betrachten wir als primitive Datentypen: Boolesche Werte (Boolean), Datumsangaben ( *Date*), Zahlen ( *Number*), Zeichen ( *Character*), Strings, Adressen, Farben, Telefonnummern,...

Bitte bedenken Sie, dass wir auf Modellierungsebene nicht an technische Details gebunden sind.

# Modellierung einer Party - Attribute



---

## Grundlegende Attributdeklarationen:

**Syntax:** [**<Sichtbarkeit>**] [**/**] **<Attributname>** [**:** **<Datentyp>**] [**[** **<Multiplizität>** **]**] [**=** **<Defaultwert>**]

**Sichtbarkeiten:**

- **+** : public; d. h. alle Instanzen dürfen auf das Attribut zugreifen.
- **-** : private; d. h. nur Instanzen der Klasse dürfen auf das Attribut zugreifen.
- **#** : protected; d. h. nur Instanzen der Klasse und von Subklassen dürfen auf das Attribut zugreifen.
- **~** : package; d. h. nur Instanzen der Klasse und von Klassen im selben Package dürfen auf das Attribut zugreifen.
- Ist die Sichtbarkeit nicht explizit angegeben, so ist die typische Annahme **private**.

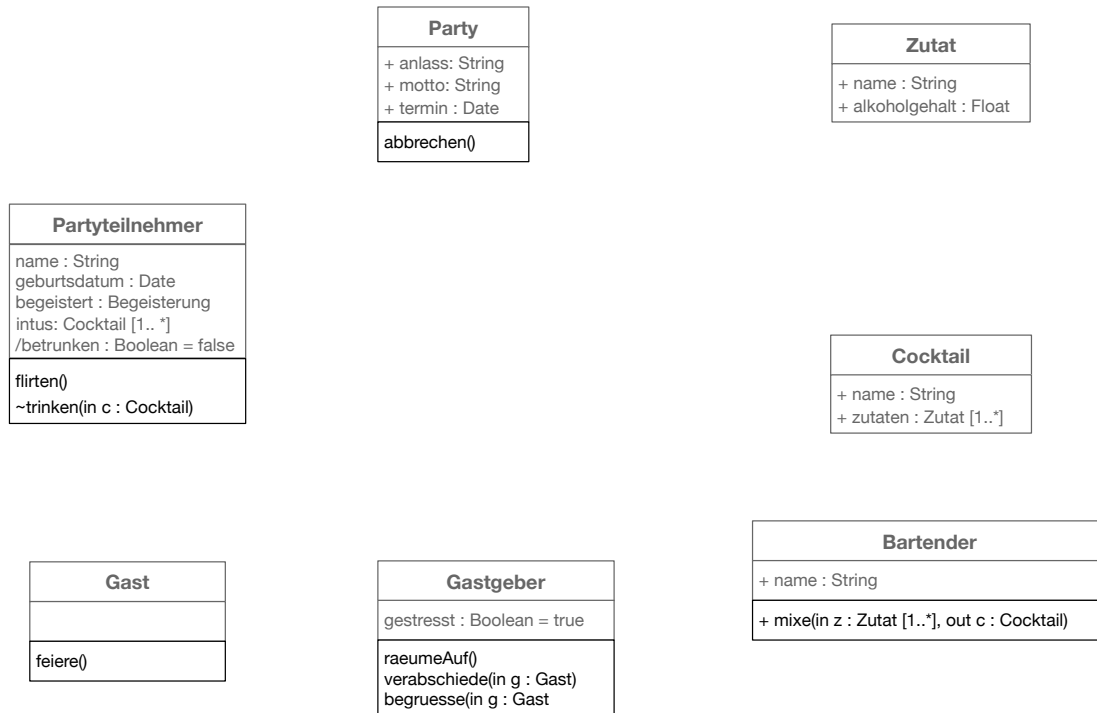
**/:** Bedeutet, dass das Attribut abgeleitet ist. Es kann aus anderen vorliegenden Daten jederzeit berechnet werden.

**Datentyp:** Der Datentyp des Attributs. Es können primitive oder auch komplexe Datentypen sein.

**Multiplizität:** Die Anzahl der Instanzen, die das Attribut haben kann. Übliche Multiplizitäten sind 0..1 (d. h. optional), 1 (d. h. genau einmal), 0..\* (d. h. beliebig oft), 1..\* (d. h. mind. einmal), 2..\*.



# Modellierung einer Party - Operationen/Methoden



Methoden bzw. Operationen sind die Verhaltensbeschreibungen einer Klasse. Sie beschreiben, was ein Objekt einer Klasse tun kann.

Grundlegende Methodendeklarationen:

**Syntax:** [`<Sichtbarkeit>`] `<Methodenname>` [ ( `<Parameterliste>` ) ] [ `<Rückgabetyt>` ]

**Sichtbarkeiten:** (*wie bei Attributen*)

**Parameterliste:** Die Liste der Parameter, die die Methode erwartet.

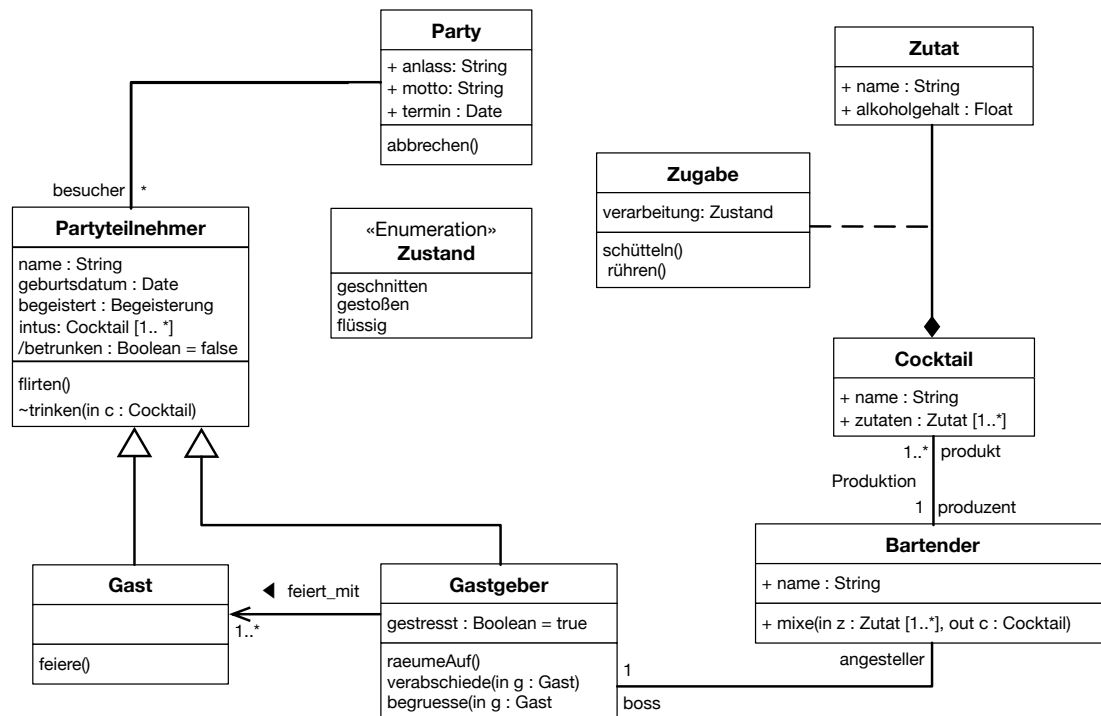
**Syntax:** `<Übergaberichtung>` `<Parameternamen>` : `<Datentyp>` [ [ `<Multiplizität>` ] ] [ `= <Defaultwert>` ]

**Übergaberichtung:** Die Übergaberichtung gibt an, ob der Parameter nur gelesen (**in**), nur beschrieben (**out**) oder sowohl gelesen als auch beschrieben (**inout**) wird. Wird die Übergaberichtung nicht explizit angegeben, so wird **in** angenommen.

**Multiplizität:** (*wie bei Attributen*)

**Rückgabetyt:** Der Datentyp des Rückgabewertes der Methode. Es können primitive oder auch komplexe Datentypen sein.

# Modellierung einer Party - Beziehungen



Um zu beschreiben, wie Instanzen der Klassen miteinander in Verbindung stehen, unterscheiden wir folgende grundlegende Beziehungen:

- **Assoziation:** Eine Assoziation beschreibt eine Beziehung zwischen zwei Klassen. Sie kann eine Richtung haben und eine Multiplizität.

Zwischen zwei Klassen können mehrere Assoziationen bestehen.

Eine Assoziation kann zyklisch sein.

Am Ende einer Assoziation kann ein Name und eine Multiplizität stehen, die die Beziehung aus Sicht der Klasse am anderen Ende der Assoziation beschreiben.

Ein Pfeil gibt die Navigationsrichtung an.

Im Beispiel ist explizit modelliert, dass ein Cocktail immer genau von einem Bartender produziert wird. Ein Bartender kann aber mehrere Cocktails produzieren.

- **Aggregation:** Eine Aggregation (◇ „am Anfang“) ist eine spezielle Form der Assoziation, bei der eine Klasse eine andere Klasse besitzt.

Im Allgemeinen sind die Unterschiede zwischen Aggregation und Assoziation nicht eindeutig bzw. ausreichend definiert und es ist deswegen sinnvoll eine normale Assoziation zu wählen.

Selbst einer der Autoren der UML Spezifikation hat bereits vor langer Zeit festgehalten:

In spite of the few semantics attached to aggregation, everybody thinks it is necessary (for different reasons). Think of it as a modelling placebo.



- **Komposition:** Eine Komposition (◆ „am Anfang“) ist eine spezielle Form der Aggregation, bei der die Lebensdauer des Besitzers die Lebensdauer des Besitzten bestimmt.

Im Beispiel ist modelliert, dass ein Cocktail aus mehreren Zutaten besteht. Weiterhin gilt, dass nach dem Genuss des Cocktails die Zutaten nicht mehr existieren.

- **Generalisierung:** Eine Klasse (Sub) kann von einer anderen Klasse (Sup) *erben* (Sub ▷ Sup). Die abgeleitete Klasse ist eine Spezialisierung der Basisklasse, die alle Attribute und Methoden der Basisklasse übernimmt und ggf. erweitert.

#### ⚠ Warnung

Technisch ist es in den meisten Programmiersprachen möglich bestehendes Verhalten ggf. so zu verändern, dass es nicht mehr kompatibel ist mit dem Verhalten der Basisklasse.

**Dies ist unter allen Umständen zu vermeiden, da es zu schwerwiegenden Fehlern führen kann.**

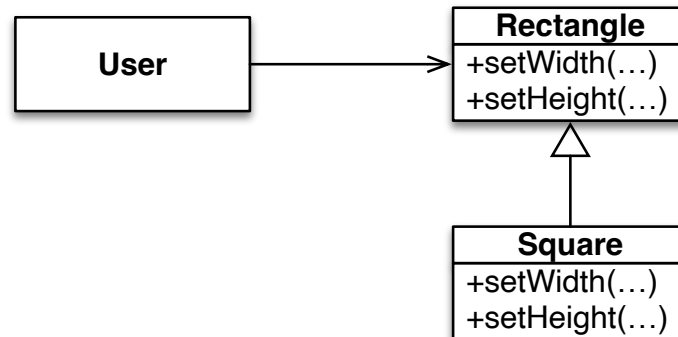
(Beispiele wären Methodenparameter oder Rückgabewerte, die auf einmal einen anderen Wertebereich haben. Oder, wenn andere Seiteneffekte auftreten.)

- **Assoziationsklasse:** Eine Assoziationsklasse (eine Klasse verbunden mit einer Assoziation über eine gestrichelte Linie) beschreibt eine Assoziation zwischen zwei anderen Klassen detaillierter und wird insbesondere dann verwendet, wenn die Attribute und Operationen nicht sinnvoll den beteiligten Klassen zugeordnet werden können. Sie kann Attribute und Methoden haben, die die Beziehung zwischen den beiden Klassen beschreiben.

# Modellierungsfehler

## ⚠ Warnung

Ein falsches Verständnis — insbesondere von der **Generalisierung** — kann zu schweren Fehlern in der Modellierung führen.



Der schwerwiegende Modellierungsfehler, der in diesem Beispiel gemacht wurde, ist einfach. Es wurde bei der Modellierung vergessen, dass es bei der Programmierung ggf. nicht nur um mathematische Konzepte geht, sondern auch das Verhalten zu berücksichtigen ist.

In Hinblick auf das Verhalten ist es falsch, dass ein Quadrat von einem Rechteck erbt. Ein Quadrat ist ein Spezialfall eines Rechtecks, bei dem die Seitenlängen gleich sind. Würden wir in unserem Code glauben, dass wir - zum Beispiel - die Breite eines Rechtecks verändern, da der Datentyp `Rectangle` ist, sich hinter dem `Rectangle` ein Objekt vom Typ `Square` verbergen, dann würde sich auch die Höhe des `Rectangle` verändern. Das ist nicht das Verhalten, das wir als Nutzer einer Instanz der Klasse erwarten würden.

# Klassenraumübung

## 2.1. Modellierung einer Reise

- Eine Reise hat einen Start- und einen Zielort sowie ein Start- und ein Enddatum.
- Eine Reise kann aus mehreren Teilreisen bestehen.
- Eine Teilreise hat einen Start- und einen Zielort sowie ein Start- und ein Enddatum.
- Eine Teilreise kann mit verschiedenen Verkehrsmitteln durchgeführt werden: Flugzeug, Zug, Auto oder Schiff.
- Jedes Verkehrsmittel hat spezifische Eigenschaften (z. B. Flugnummer für Flugzeuge, Kennzeichen für Autos, etc.).
- Eine Reise kann von mehreren Reisenden unternommen werden.

# Übung

## 2.2. Modellierung einer einfachen Bibliotheksverwaltung

Modellieren Sie eine einfache Bibliotheksverwaltung mit Hilfe eines Klassendiagramms.

- Es gibt Bücher, die von Autoren geschrieben werden.
- Ein Buch kann in mehreren Exemplaren vorhanden sein.
- Ein Buch hat einen Titel, einen Autor und eine ISBN-Nummer.
- Ein Autor hat einen Namen und ein Geburtsdatum.
- Ein Exemplar hat eine eindeutige Nummer. Jedes Exemplar ist entweder ausgeliehen oder verfügbar.
- Ein Ausleiher hat einen Namen und eine Adresse.

Modellieren Sie auch explizit die Beziehungen zwischen den Klassen - inkl. Multiplizitäten.

---

Sie können die Modellierung auf Papier machen oder zum Beispiel eine (kostenlose) UML-Modellierungssoftware verwenden: z. B. <https://app.diagrams.net>. Alternativ kann auch **Mermaid** verwendet werden, um die Diagramme deklarativ zu beschreiben.

# Übung

## 2.3. Modellierung eines einfachen Kursmanagementsystem für Universitäten

Erstellen Sie ein passendes UML-Diagramm.

- Eine Vorlesung hat immer einen Namen, eine Nummer und einen Raum.
- Manchmal hat eine Vorlesung auch Teilnehmer.
- Ein Dozent liest ggf. mehrere Vorlesungen.
- Ein Student besucht in der Regel eine oder mehrere Vorlesungen.
- Zu einer Vorlesung gibt es ggf. mehrere optionale Übungen.
- Eine Prüfung kann entweder eine Klausur oder eine Portfolio-Prüfung sein. Letztere besteht aus einer Präsentation zu einem Thema und einer schriftlichen Ausarbeitung. Beide haben eine festgelegte Anzahl an Punkten. Die Endnote ergibt sich aus dem Durchschnitt der beiden Noten.
- Hat die Veranstaltung eine Portfolio-Prüfung, dann ist jeder Studierende für das gesamte Semester einer bestimmten Studiengruppe zugeordnet.

# Übung

## 2.4. Darstellung von Mathematischen Ausdrücken.

Ihre Aufgabe ist es mathematische Ausdrücke in UML zu modellieren. Zum Beispiel einen Ausdruck wie:  $(a + b) \cdot c$  oder auch eine Ausdruck wie  $\frac{a}{b}$  oder auch  $\sqrt{a^2 + b^2}$ .

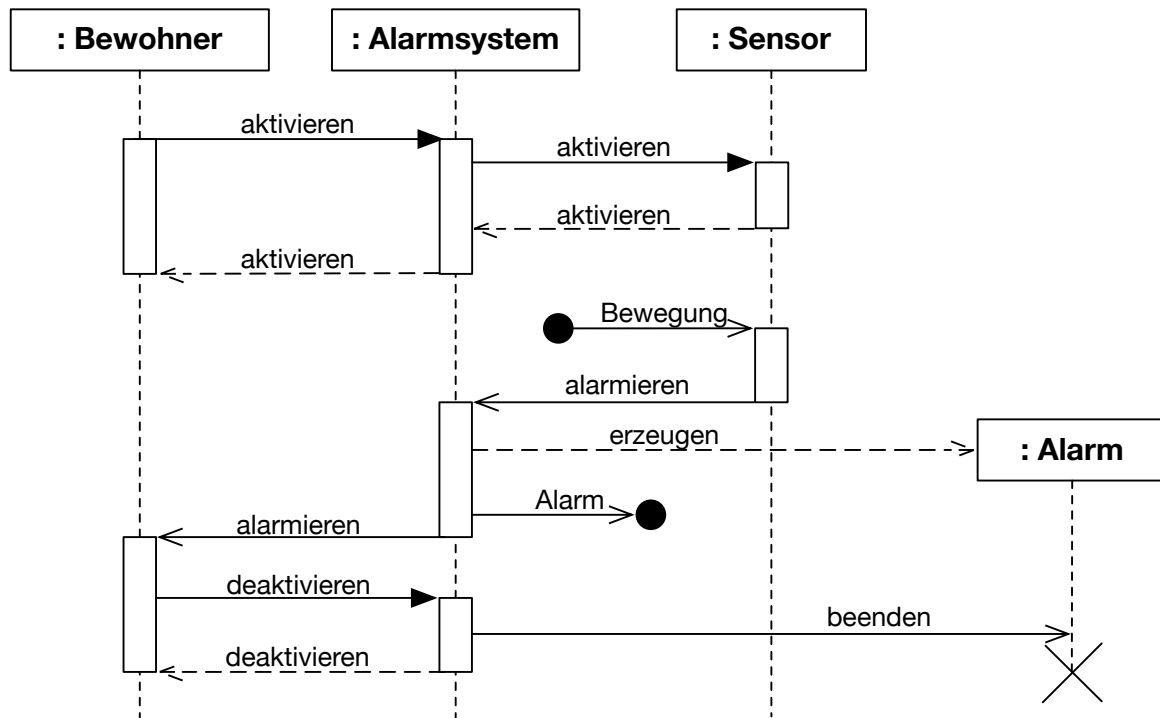
Beachten Sie bei Ihrer Modellierung, dass Sie beliebig komplexe Ausdrücke modellieren können. Die grundlegenden Operationen sollten: Addition, Subtraktion, Multiplikation, Division, Potenzierung, Wurzelziehen und den Logarithmus zur einer beliebigen Basis umfassen ( $\log_b x$ ); das unäre Minus soll auch unterstützt werden ( $-x$ ). Darüber hinaus sollen beliebig viele benannte Variablen ( $x, y, \dots$ ) und Konstanten (Zahlen) in den Ausdrücken verwendet werden können.

Welche Methoden machen auf Ausdrücken immer Sinn und würden Sie dann in Ihrer Modellierung definieren?

## 3. Sequenzdiagramme

---

## Beispiel: Sequenzdiagramm für ein Alarmsystem



In Sequenzdiagrammen wird der zeitliche Ablauf von Interaktionen zwischen Objekten dargestellt.

- Eine Ausführungssequenz wird durch eine vertikales Rechteck über der Lebenslinie dargestellt.
- Bei einem synchronen Nachrichtenaustausch wartet der Sender, bis der Empfänger diese abgearbeitet hat. Er wird durch eine durchgezogene Linie mit einem gefüllten Dreieck (→) dargestellt.
- Bei einem asynchronen Nachrichtenaustausch wartet der Sender nicht auf eine Antwort des Empfängers. Er wird durch eine durchgezogene Linie mit einem offenen Pfeil (→) dargestellt.
- Eine Nachricht, die ein Objekt erzeugt wird mit einer gestrichelten Linie dargestellt.
- Eine Antwortnachricht wird durch eine gestrichelte Linie mit einem offenen Pfeil (←) dargestellt.

### ⚠ Warnung

In vielen Diagrammen wird auf die Feinheiten bzgl. der korrekten Darstellung der Nachrichten wenig Wert gelegt. Sollte sie sich nicht sicher sein, dass der Ersteller bewusst synchrone und asynchrone Nachrichten unterschieden hat, dann sollten Sie davon ausgehen, dass es sich um synchrone Nachrichten handelt.



# Übung

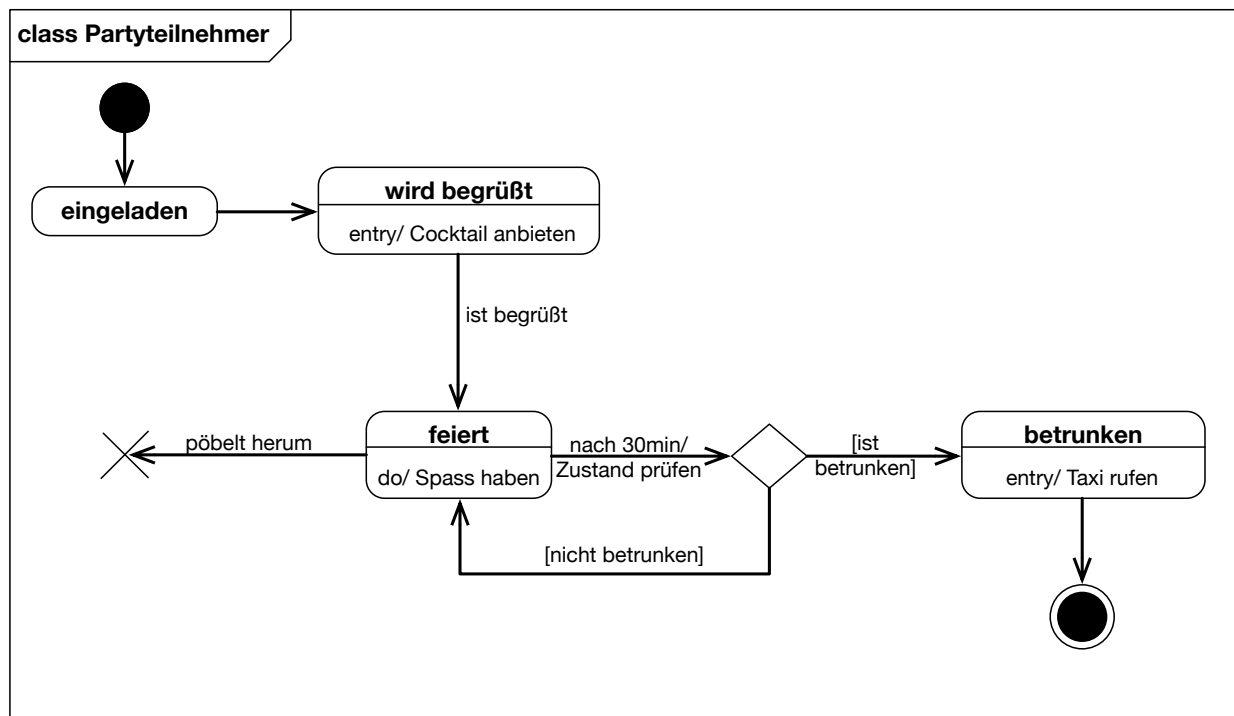
## 3.1. Erstellen Sie ein Sequenzdiagramm für die Bestellung eines Cocktails.

- Ein Gast bestellt einen Cocktail beim Barkeeper.
- Der Barkeeper bereitet dann den Cocktail zu, indem er erst die Zutaten hinzufügt und danach diese fachgerecht mixt. Sobald er fertig ist, überreicht er den Cocktail an den Gast.
- Da der Gast sehr durstig ist, trinkt er den Cocktail in einem Zug aus.

Hinweis: es gibt mehrere Möglichkeiten, wie das obige Szenario modelliert werden kann, da nicht alles explizit vorgegeben ist. Treffen Sie eine bewusste Entscheidung, wie Sie das Szenario modellieren.



# Beispiel: Zustandsautomat für einen Partyteilnehmer



Ein Zustandsautomat beschreibt das Verhalten eines Objekts in Abhängigkeit von seinem Zustand.

Ein Zustandsautomat besteht aus:

- Startknoten:** Startpunkt des Zustandsautomaten. Er hat höchstens eine ausgehende Transition.  
Dargestellt mit einem schwarzen Kreis (●).
- Endzustand:** Endpunkt des Zustandsautomaten. Er hat keine ausgehenden Transitionen. Es kann mehrere Endzustände geben.
- Entscheidung:** In Abhängigkeit vom Ergebnis (Auswertung der Entscheidungsbedingung), der auf dem Weg zur Entscheidung getroffenen Aktionen, wird der Zustandsautomat in unterschiedliche Zustände überführt.  
Es gibt mindestens zwei ausgehende Transitionen.  
Dargestellt mit einer Raute (◇).
- Terminator:** Beendet (auch) einen Zustandsautomaten.  
Beendet die Lebensdauer des Zustandsautomaten. (In diesem Fall könnte man dies so interpretieren, dass der Partyteilnehmer die Party verlässt/rausgeworfen wird und wir uns auch nicht weiter für den Partyteilnehmer interessieren.)  
Dargestellt mit einem großen X.
- Transitionen (Übergänge):** Verbinden Zustände und Entscheidungen.

Syntax: Trigger [Guard] / Verhalten

Der Trigger beschreibt das Ereignis, das den Übergang auslöst. Ein Guard (Wächter) beschreibt die Bedingung, die wahr sein muss. Das Verhalten beschreibt die Aktion, die ausgeführt wird beim Durchlaufen des Übergangs.

Dargestellt mit einem Pfeil ( $\rightarrow$ ).

- Zuständen und Übergängen dazwischen
- Ereignissen, die einen Übergang auslösen,
- Aktionen (`entry`, `exit`, `do`), die ausgeführt werden,
- Start- und Endzuständen.
- Entscheidungsknoten

In diesem Fall modellieren wir die Zustände eines Partyteilnehmers.

- Ein Partyteilnehmer kann in den Zuständen „eingeladen“, „wird begrüßt“, „feiert“ und „ist betrunken“ sein.

# Übung

## 4.1. Modellieren Sie den Zustandsautomaten für einen Zimmerventilator.

- Der Ventilator kann in drei Zuständen sein: „Aus“, „Stufe 1“, „Stufe 2“.
- Der Endzustand ist der Zustand „Aus“.
- Zwischen Stufe 1 und Stufe 2 kann beliebig oft hin und her gewechselt werden.
- In Stufe 1 dreht der Ventilator langsam, in Stufe 2 schnell.