Objektorientierung

- Fachlich zusammengehörige Elemente (Daten und Code) werden in einer *Klasse* zusammengefasst.
- Durch *Kapselung* können auch umfangreiche Programme überblickt werden.
- Code mit sauberen Schnittstellen kann einfach wiederverwendet werden.

UML

Die folgenden Definitionen werden von UML-Darstellungen begleitet.

- Die *Unified Modeling Language* (UML) ist ein verbreiteter grafischer Werkzeugkasten zum Entwurf und zur Darstellung von objektorientierter Software.
- Es gibt Werkzeuge, die Code aus UML-Diagrammen erzeugen können.
- Auch der umgekehrte Weg ist möglich.

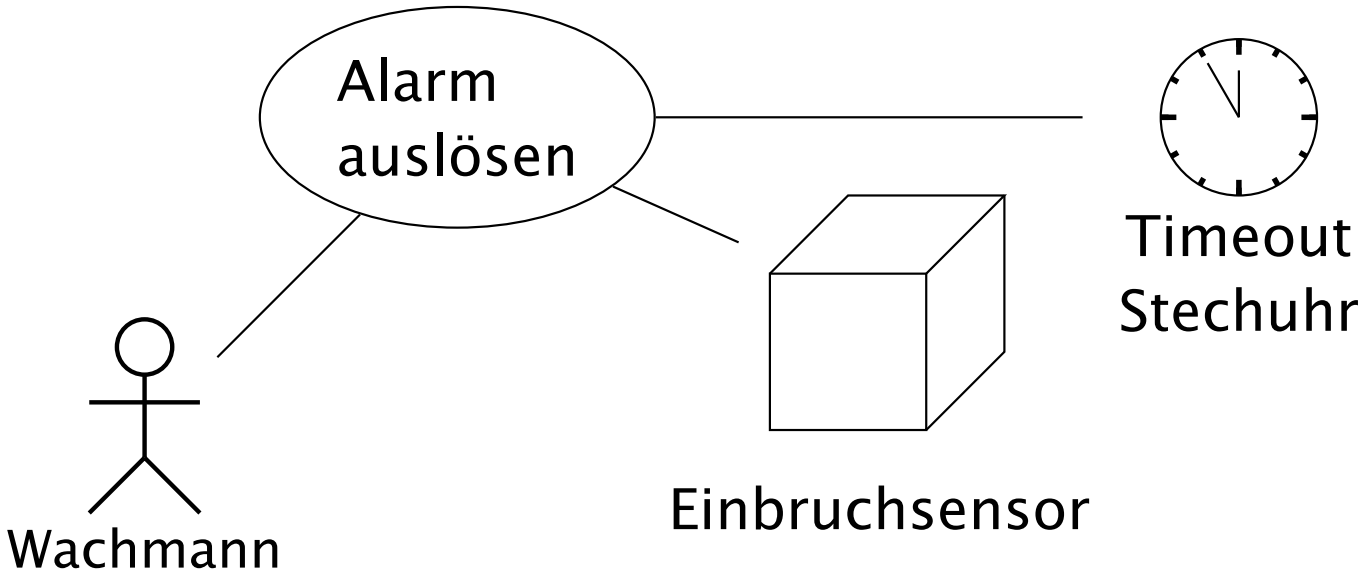
Anwendungsfall

- Am Anfang der Softwareentwicklung steht die Herausarbeitung von Anwendungsfällen (*Use Cases*).
- Anwendungsfalldiagramme bieten einen ersten Überblick darüber, was umgesetzt werden muss.
- Beschreibungen der Anwendungsfälle in Textform, mit EPK oder BPMN sind der nächste Schritt.

Elemente

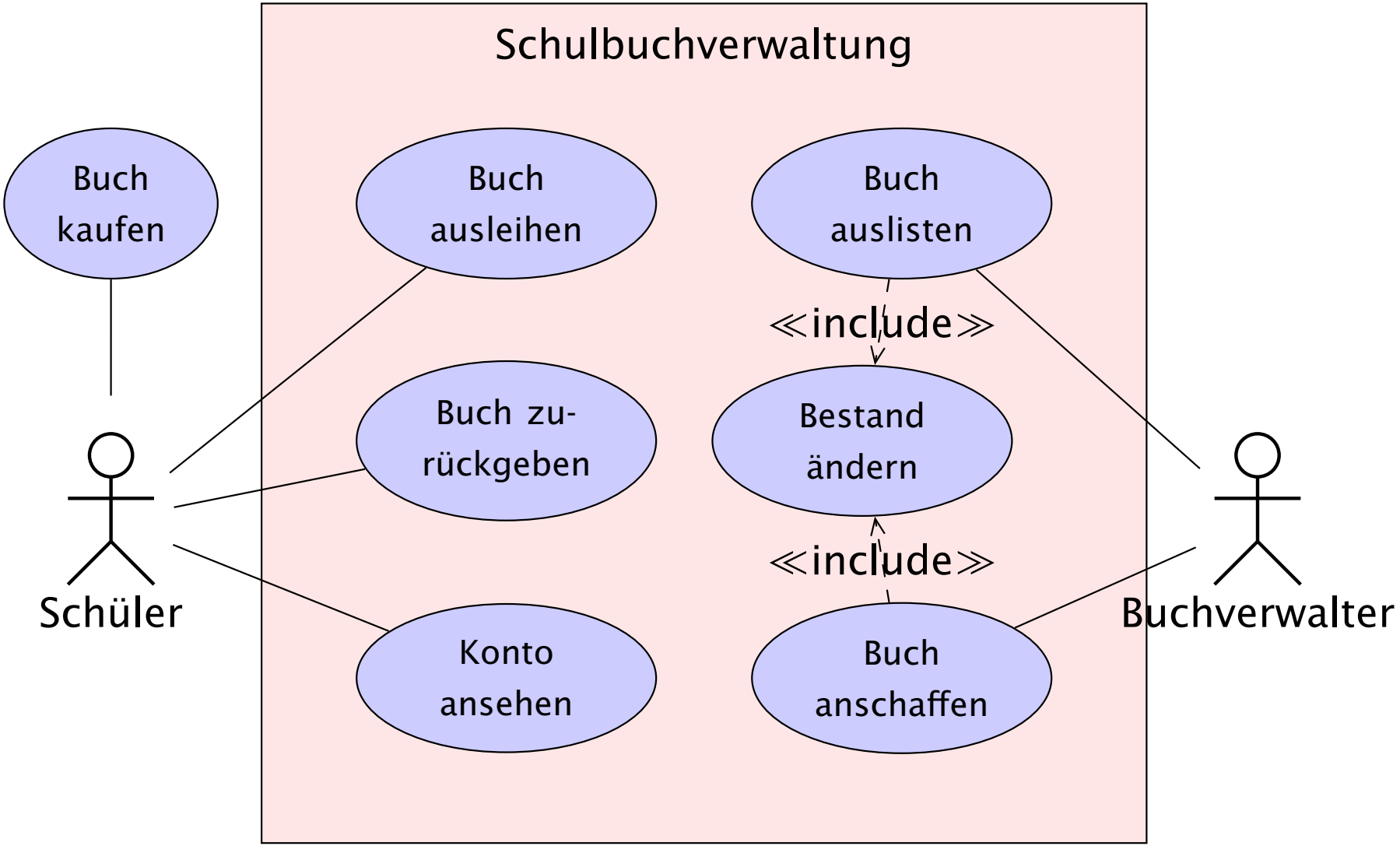
- Ein Akteur (*actor*) löst einen Anwendungsfall aus oder ist daran beteiligt. Akteure können sein:
  - Menschen
  - Technische Systeme
  - Zeit-Ereignisse
- Ein Anwendungsfall (*Use Case*) ist ein zusammenhängender Arbeitsablauf. Er wird durch einen Akteur angestoßen und liefert ein Ergebnis.
  Besonderheiten:
  - Ein *Systemanwendungsfall* wird mit einem Softwaresystem umgesetzt.
  - Ein *abstrakter* Anwendungsfall fasst mehrere ähnliche Anwendungsfälle zusammen. Abstrakte Anwendungsfälle sind *kursiv* beschriftet.

Beispiel



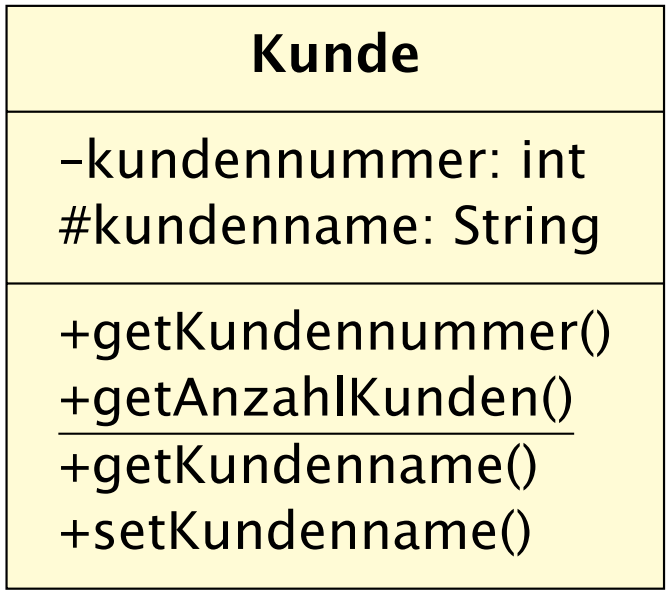
Ein Alarm kann vom Wachmann, vom Einbruchsensor oder durch die Stechuhr ausgelöst werden. Es wirkt seltsam, wenn alle Akteure als Strichmännchen dargestellt werden.

Beispiel


Schulbuchverwaltung
Buch kaufen
Buch ausleihen
Buch auslisten
≪include≫
Buch zu-rückgeben
Bestand ändern
≪include≫
Konto ansehen
Buch anschaffen
Schüler
Buchverwalter

Klasse

- Eine *Klasse* ist ein Bauplan für Objekte.
  → Klassen sind abstrakt, Objekte sind konkret.
- Eine Klasse definiert *Attribute* (Eigenschaften) und *Methoden* (Verhalten).
- Sichtbarkeitsregeln ermöglichen Zugriffsbeschränkungen auf Interna.



Attribut

Attribute sind in den meisten Programmiersprachen streng typisiert.

Zugriffsrechte

private (–) nur für die Klasse selbst

protected (#) für die Klasse selbst und abgeleitete Klassen

public (+) freier Zugriff.

Für den Zugriff auf geschützte Attribute werden `get`- und `set`-Methoden angeboten.
Klassenattribute (*static*) werden von allen Objekten einer Klasse gemeinsam genutzt. Sie werden in Diagrammen unterstrichen.

Methode

- Methoden beschreiben das Verhalten eines Objektes.
- Auch für Methoden werden Zugriffsrechte vergeben.
- Methoden werden nach Namen und Signatur (Art, Anzahl und Reihenfolge der Argumente) unterschieden:
  Mehrere Methoden einer Klasse können gleich heißen, solange sie unterschiedliche Signaturen haben.
- Methoden können genau einen Wert zurückliefern.
- Statische Methoden sind von Objekten unabhängig und können auch ohne Objekt benutzt werden. Sie sind im Diagramm unterstrichen.

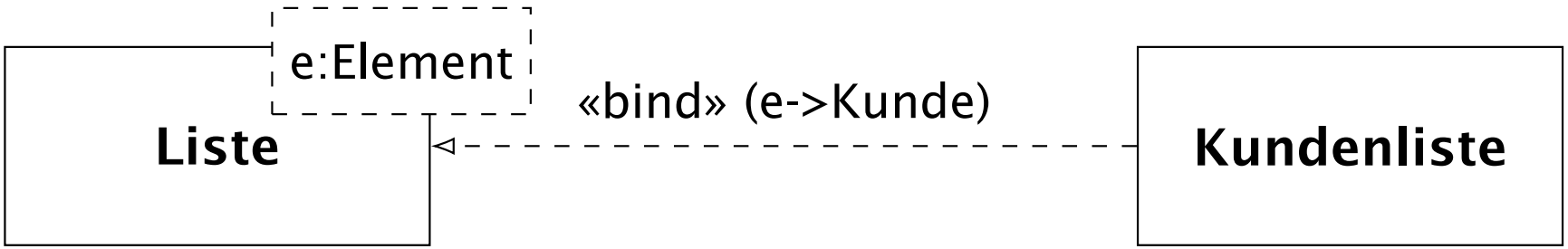
Methoden-Signatur

```
+getAlter(datum: Date): double {datum > geburtsdatum}
```

- Am Anfang kann die Sichtbarkeit angegeben werden (+ heißt *public*).
- In den runden Klammern werden Parameter angeben, jeweils zuerst der Name, ein Doppelpunkt, dann der Datentyp. Mehrere Parameter werden mit Kommas getrennt.
- Nach den runden Klammern wird nach einem Doppelpunkt der Rückgabedatentyp angegeben.
- Zusicherungen folgen in geschweiften Klammern.
- Sichtbarkeit, Name der Funktion und runde Klammern müssen vorhanden sein, alles andere ist optional.

Parametrisierbare Klasse

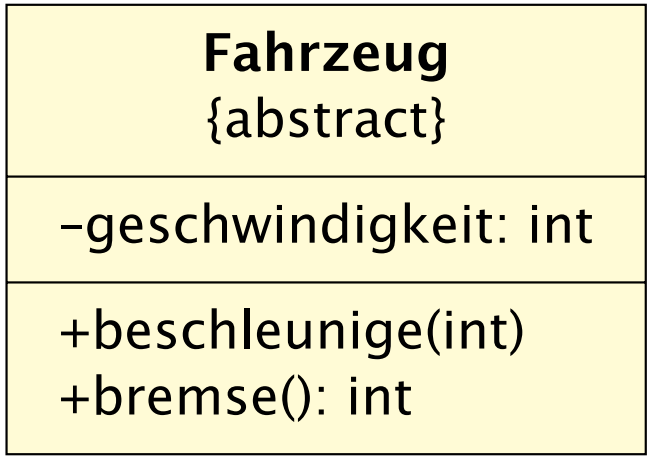
- Parametrisierbare Klassen (in C++ *Templates*) stellen einen Bauplan für Klassen dar.
- Parametrisierbare Klassen sind oft Sammelklassen (z.B. Listen).
- Sie werden durch Angabe eines Datentyps zu konkreten Klassen.

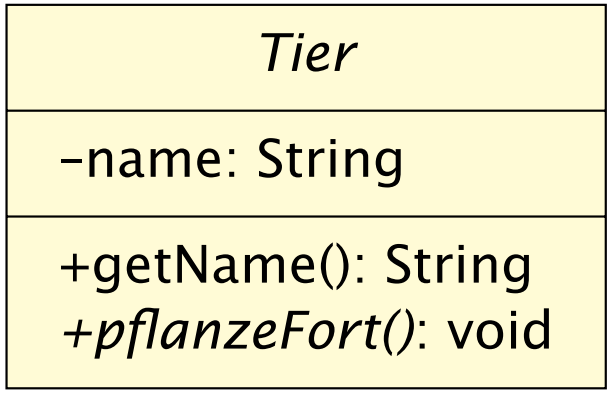



```
ArrayList<Kunde> kundenliste = new ArrayList<>();
```

Abstrakte Klasse

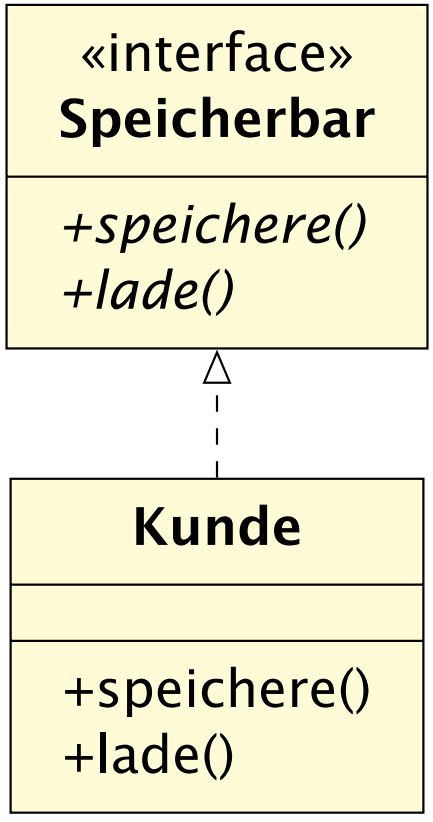
- Abstrakte Klassen können nicht instantiiert werden.
- Sie geben einen Rahmen vor, den Unterklassen ausfüllen müssen.
- Der Klassenname wird kursiv geschrieben oder es wird {abstract} unter dem Klassennamen ergänzt.
- Abstrakte Klassen können konkrete und abstrakte Methoden haben.
- Abstrakte Methoden werden ebenfalls kursiv geschrieben.

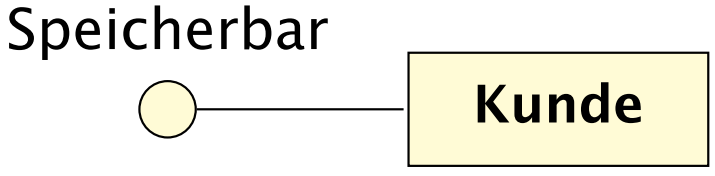




Interface

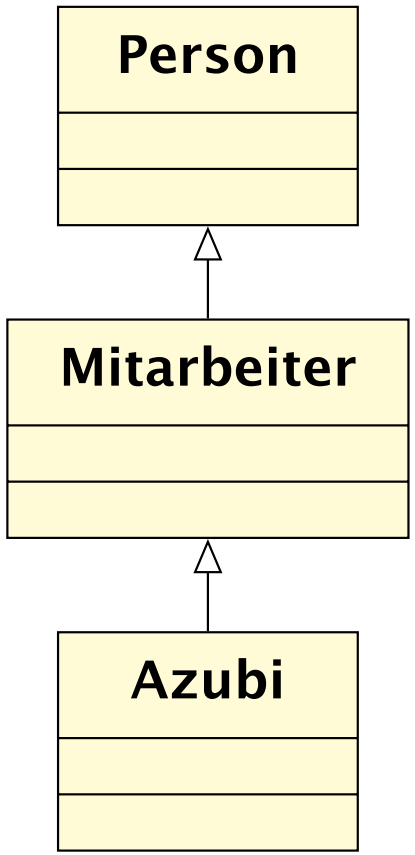
- Ein *Interface* ist eine abstrakte Klasse, die meist nur abstrakte Methoden enthält.
- Interfaces normieren Funktionen, die von mehreren Klassen umgesetzt werden.
- Klassen erben nicht von einem Interface, sie *implementieren* es.
- Der Stereotyp «interface» wird oberhalb des Namens angegeben.
- Die Lollipop-Darstellung (unten) ist alternativ möglich.



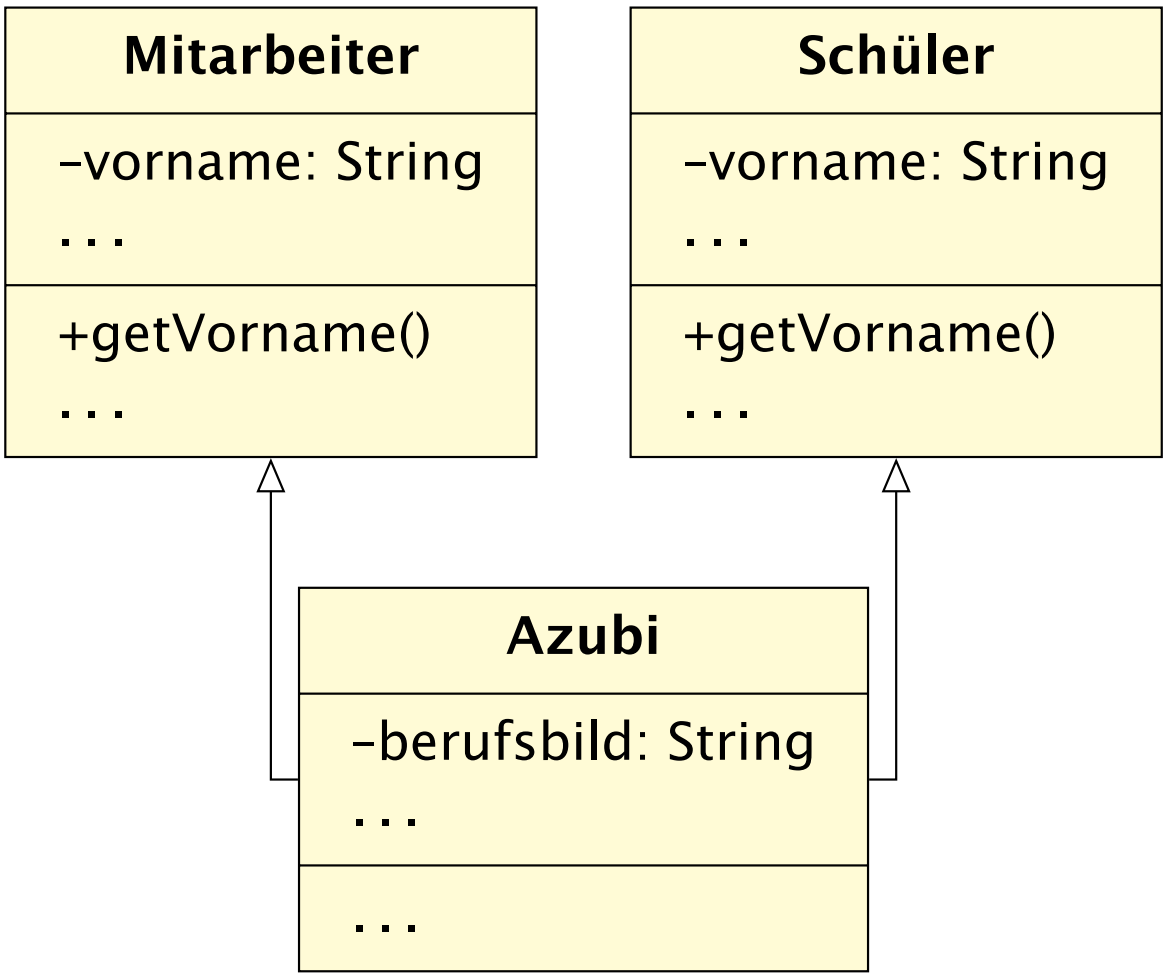


Vererbung: Unterklassen

- Eine Unterklasse (*abgeleitete* Klasse) spezialisiert die Oberklasse.
- Unterklassen enthalten *alle* Attribute und Methoden der Oberklasse, dazu kommen nach Bedarf weitere Methoden und Attribute.
- Unterklassen können Methoden der Oberklasse ersetzen.
- Im Programm kann eine Unterklasse eingesetzt werden, wo die Oberklasse erwartet wird.
- Ein Mitarbeiter hat z.B. zusätzlich zu den Eigenschaften einer Person eine Mitarbeiternummer und ein Gehalt.



Mehrfachvererbung

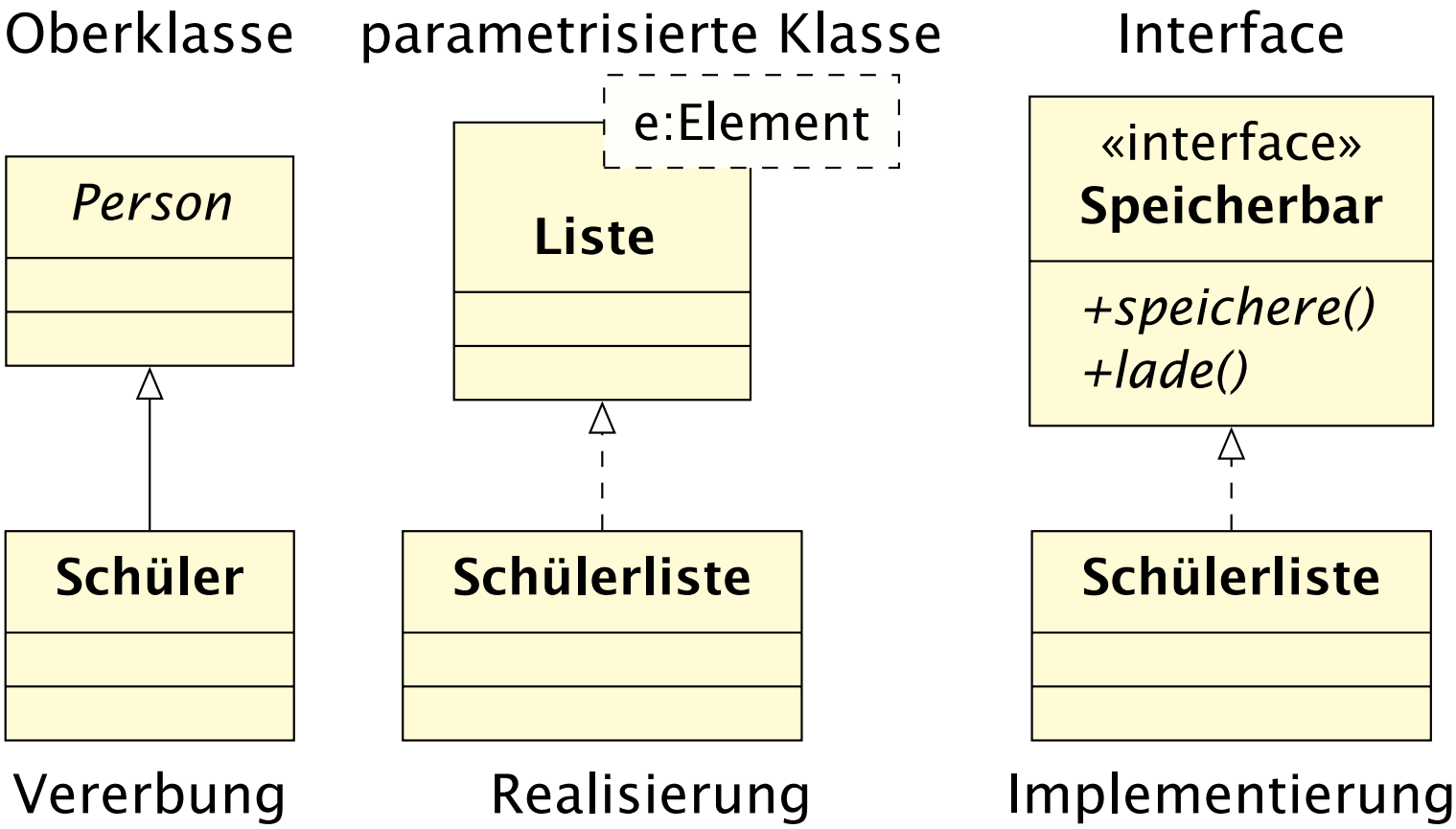


Mehrfachvererbung wirft Probleme auf:

- Azubis sind sowohl Mitarbeiter als auch Schüler.
- Erbt *Azubi* den Vornamen aus *Mitarbeiter* oder aus *Schüler*?

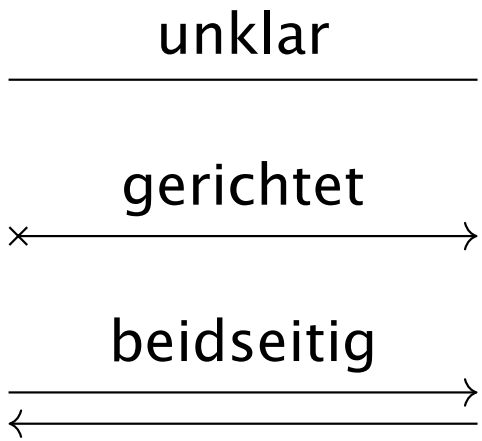
Nicht in jeder Sprache ist Mehrfachvererbung zulässig.

Vererbung, Implementation und Realisierung



Assoziationen

- Beziehungen zwischen Klassen sind normalerweise gerichtet.
- Wenn zwei Klassen in beiden Richtungen miteinander verbunden werden sollen, muss man das gezielt angeben. (Unterschied zu Beziehungen in einer Datenbank!)
- In der Programmierung werden Assoziationen durch Objekt-Variablen realisiert.

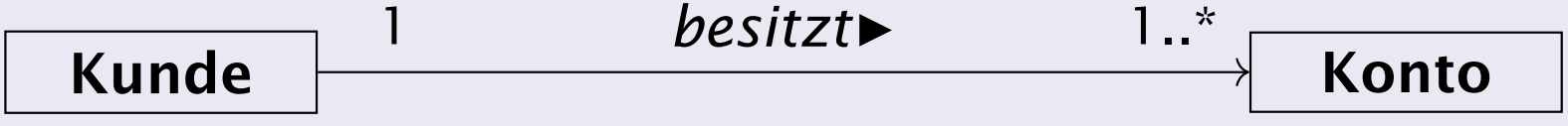


Assoziationen

- Multiplizität entspricht ungefähr der Kardinalität bei relationalen Datenbanken.
- Mengen werden mit Zahlen angegeben, für unbestimmte Mengen steht ein * (kein *n*).
- Assoziationen können beschriftet werden, ein schwarzes Dreieck ▶ gibt die Leserichtung vor.

Lesebeispiel

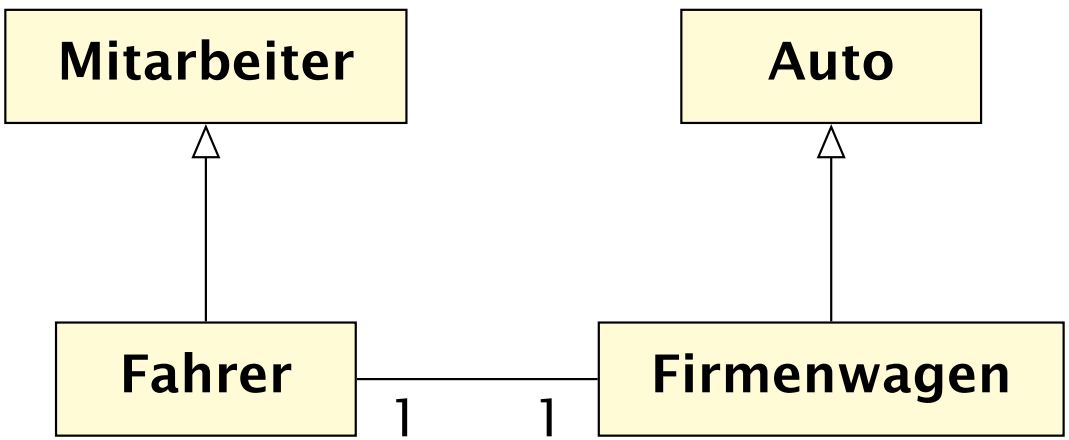
Ein Kunde besitzt mindestens ein Konto. Ein Konto gehört genau einem Kunden.



Hat-Ein und Ist-Ein

Vererbung

- Ein Fahrer *ist ein* Mitarbeiter.
- Ein Firmenwagen *ist ein* Auto.

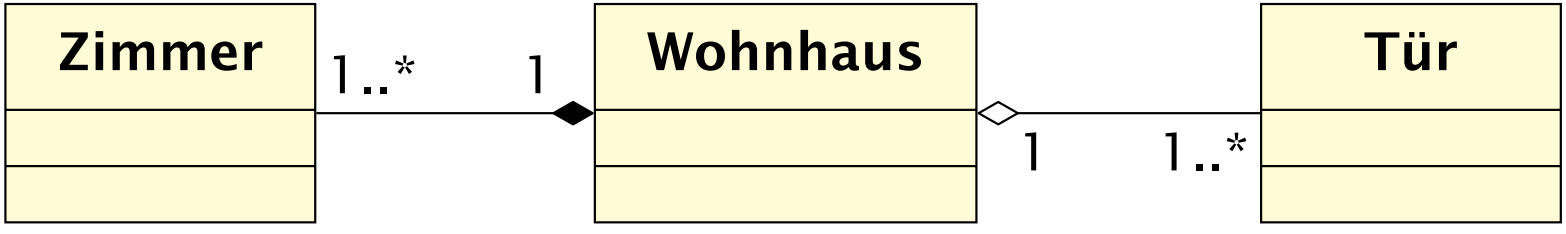


Assoziation

- Ein Firmenwagen *hat einen* Fahrer.
- Ein Fahrer *hat einen* Firmenwagen.

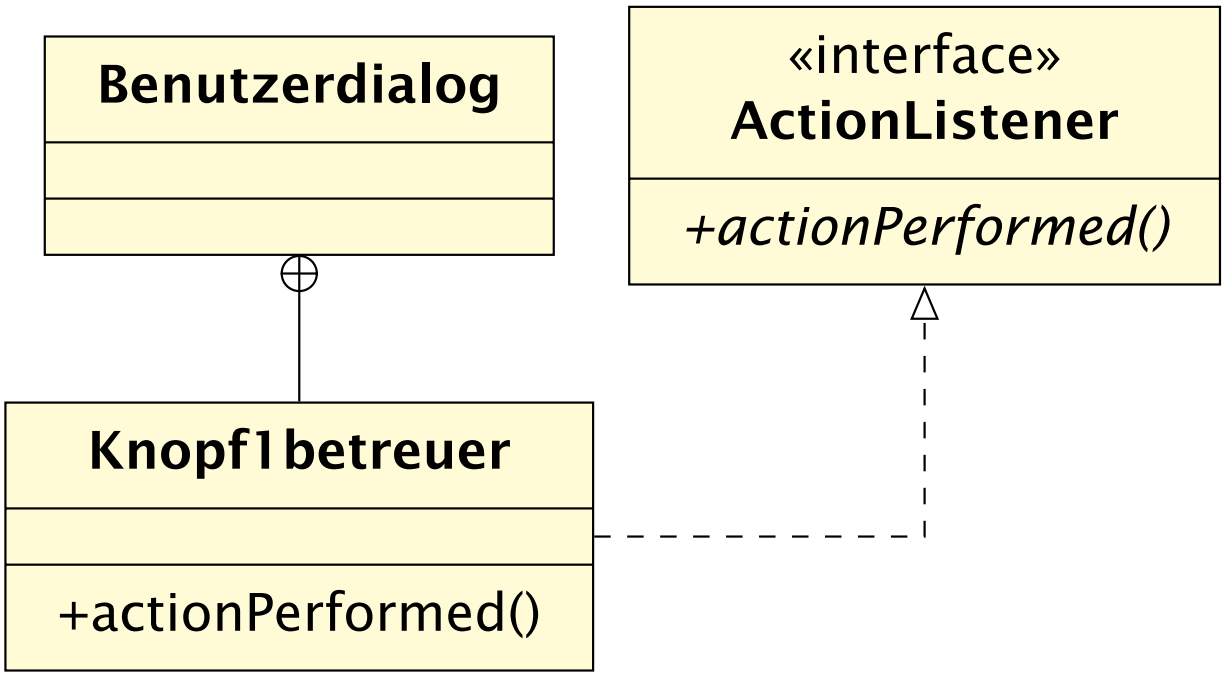
Zusammensetzungen

- Hat-Ein-Beziehungen gibt es als *Aggregation* oder *Komposition*.
- Bei einer *Aggregation* können die Teile auch ohne das Ganze existieren: *Ein Wohnhaus hat mindestens eine Tür; Türen gibt es auch ohne Haus.* Die Verbindungsline zur Containerklasse endet in einer offenen Raute ◊.
- Die Teile einer *Komposition* existieren nicht für sich alleine: *Ein Wohnhaus hat mindestens ein Zimmer, ein Zimmer ohne Haus gibt es nicht.* Die Verbindungsline zur Containerklasse endet in einer schwarzen Raute ♦.



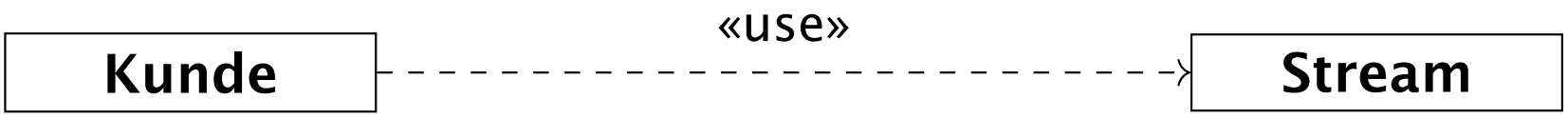
Verschachtelungen

- *Innere* Klassen können speziell gekennzeichnet werden (müssen aber nicht).
- Die Verbindungslinie endet in einem Plus im Kreis auf der Seite der äußeren Klasse.

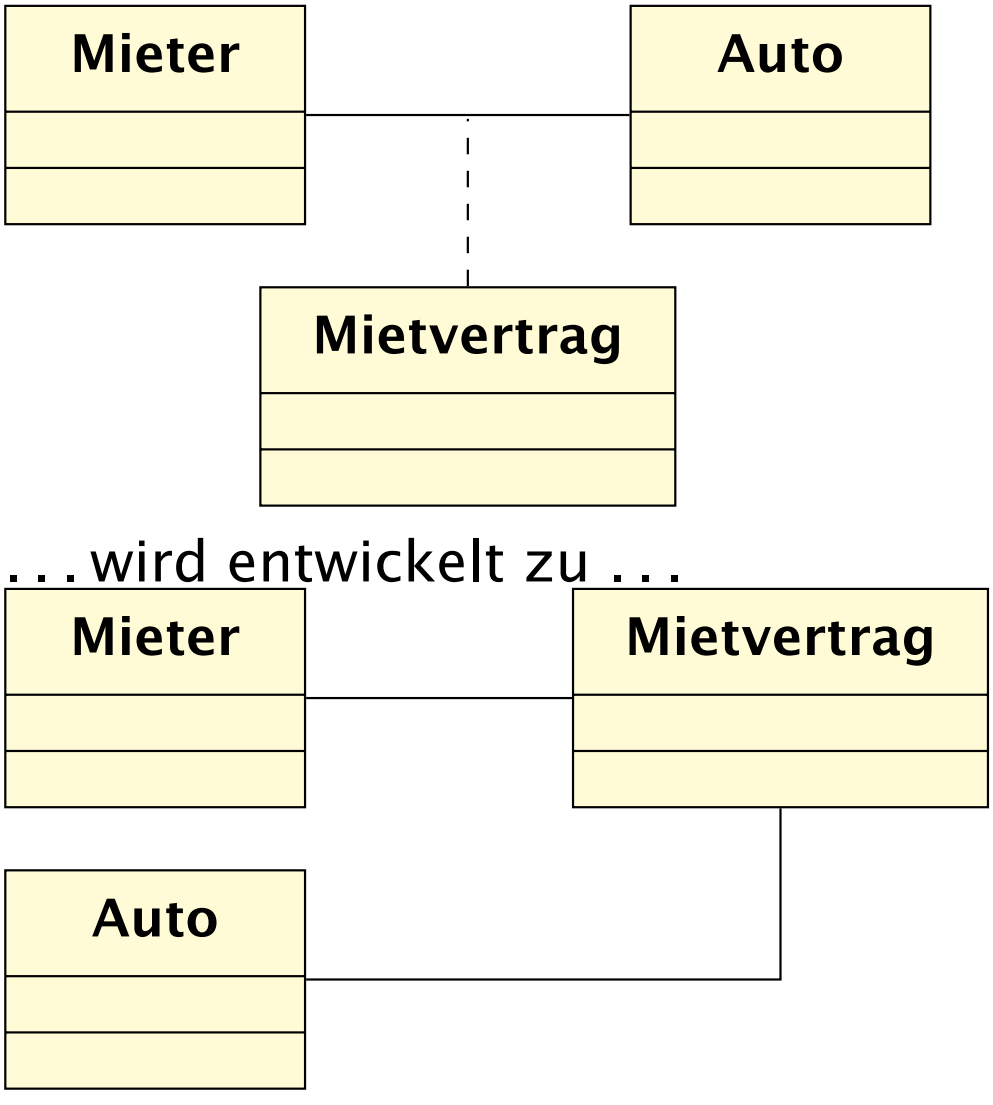


Abhängigkeiten

- Fachliche Abhängigkeiten (keine Vererbung, Implementation, Aggregation oder Komposition) werden mit gestrichelten Linen und offenen Pfeilspitzen > gekennzeichnet.
- Beispiel: Die Klasse *Kunde* benötigt die Klasse *Stream* zum Speichern und Laden.



Attribuierte Assoziationen



- Assoziationsklassen (attribuierte Assoziationen) ähneln $m : n$-Beziehungen in relationalen Datenbankentwürfen.
- Sie sind Ausdruck eines nicht zu Ende gedachten Designs.
- Es ist besser, den Entwurf vollständig auszuführen.

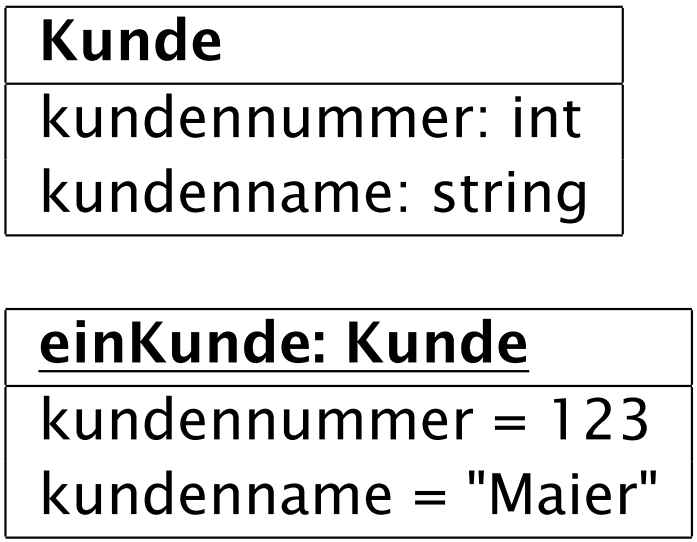
Enumeration

- Eine Enumeration (Enum) ist eine vollständige Liste fester Werte.
- Sie können ähnlich wie Integer-Werte verwendet werden.
- Die Darstellung entspricht der einer Klasse ohne Methoden.

```
public enum Farbe
 {KREUZ, PIK, HERZ, KARO}
Farbe farbe = Farben.KREUZ;
switch(farbe) {
case KREUZ:
...
```

Objekte und Objektdiagramme

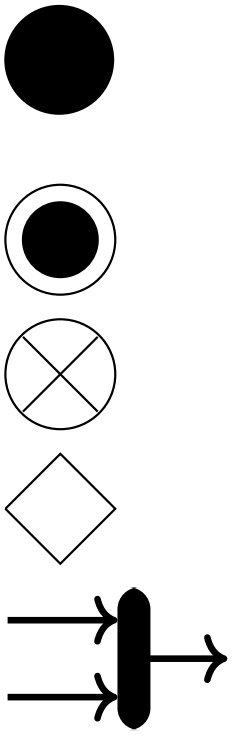
- Objektdiagramme werden vor allem als anschauliches Beispiel für Kunden erstellt, nicht für Entwickler.
- Objekte und Klassen gehören nicht in das selbe Diagramm.
- Ein Objekt ist eine *Instanz* einer Klasse. Es enthält konkrete Attribute.
- Objekte werden unterstrichen dargestellt. Der Klassennamen folgt nach einem Doppelpunkt.



Aktivitätsdiagramm

- Aktivitätsdiagramme beschreiben Abläufe.
- Sie ähneln BPMN-Diagrammen.
- Aktivitätsdiagramme werden nicht zur Code-Generierung verwendet.

Elemente



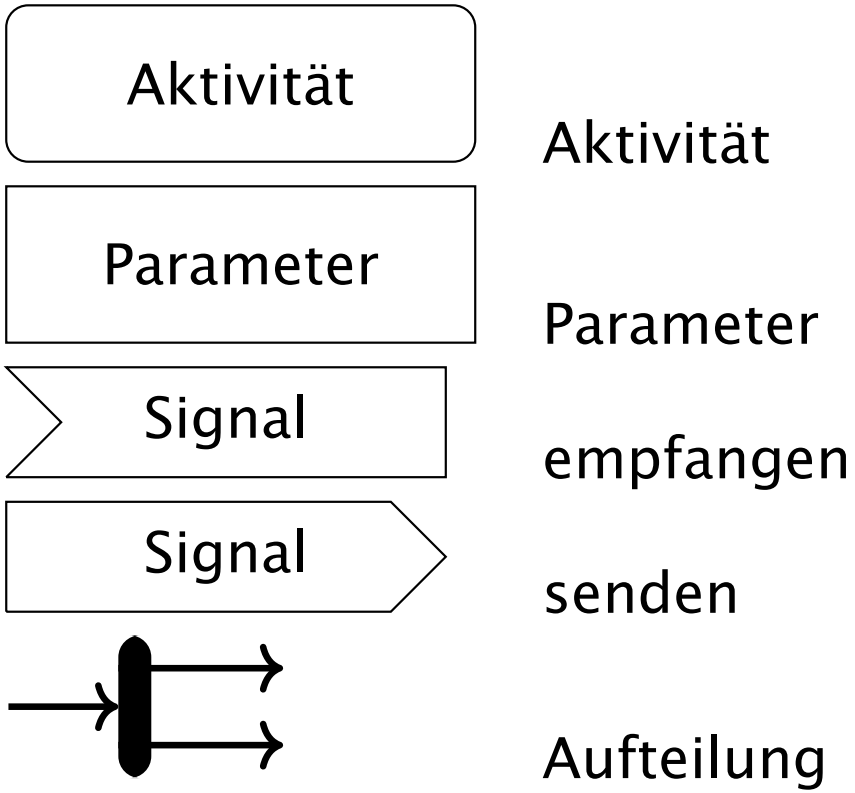
Anfang

Schluss

Abbruch

Entscheidung

Synchronisation



Aktivität

Parameter

empfangen

senden

Aufteilung

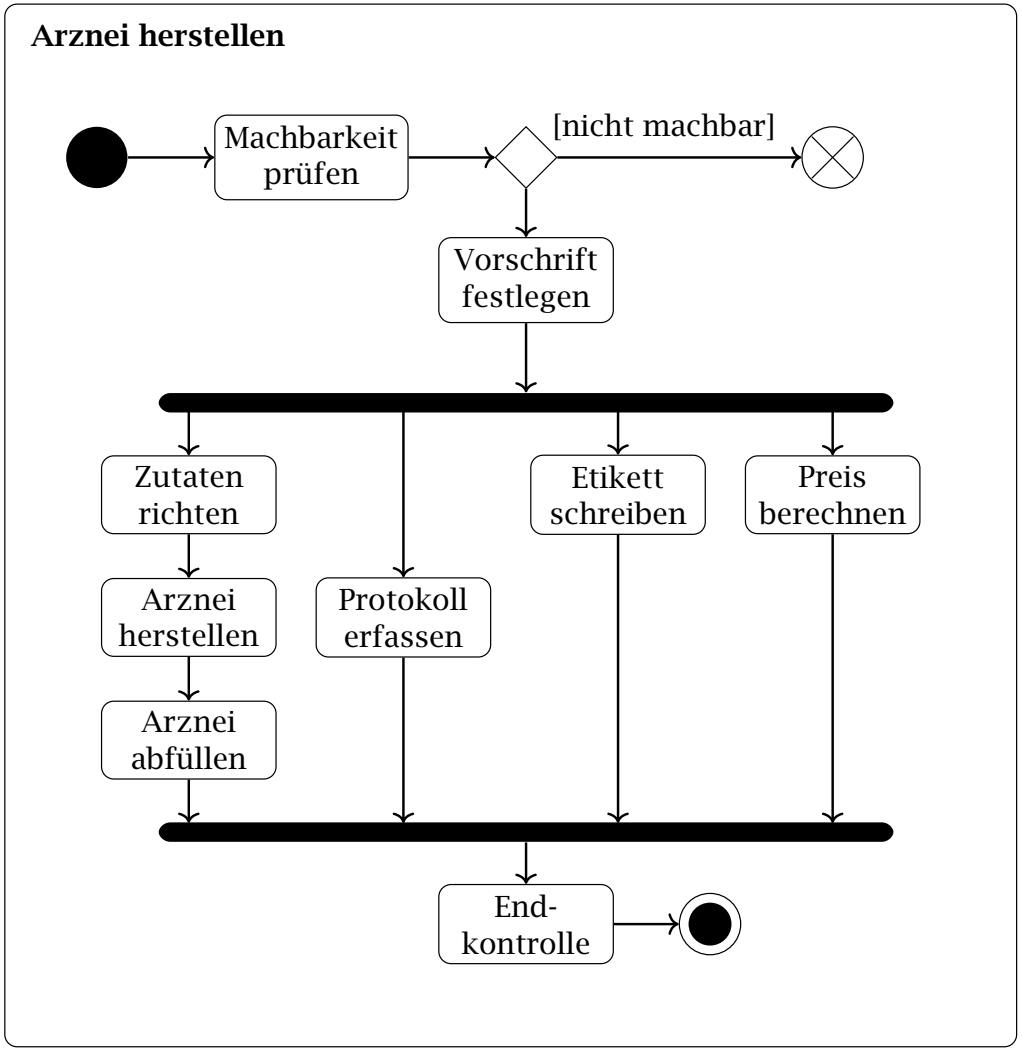
Beispiel

Wenn in einer Apotheke ein Arzneimittel individuell für einen Patienten angefertigt wird, fallen folgende Arbeitsschritte an:

1. Machbarkeitsprüfung
2. Erstellen einer Herstellungsvorschrift
3. Bereitstellen der Zutaten und Geräte
4. Herstellen
5. Protokoll schreiben
6. Etikett schreiben
7. Preis ausrechnen
8. Abfüllen
9. Endkontrolle

Ein Teil der Arbeiten kann parallel ausgeführt werden.

Beispiel

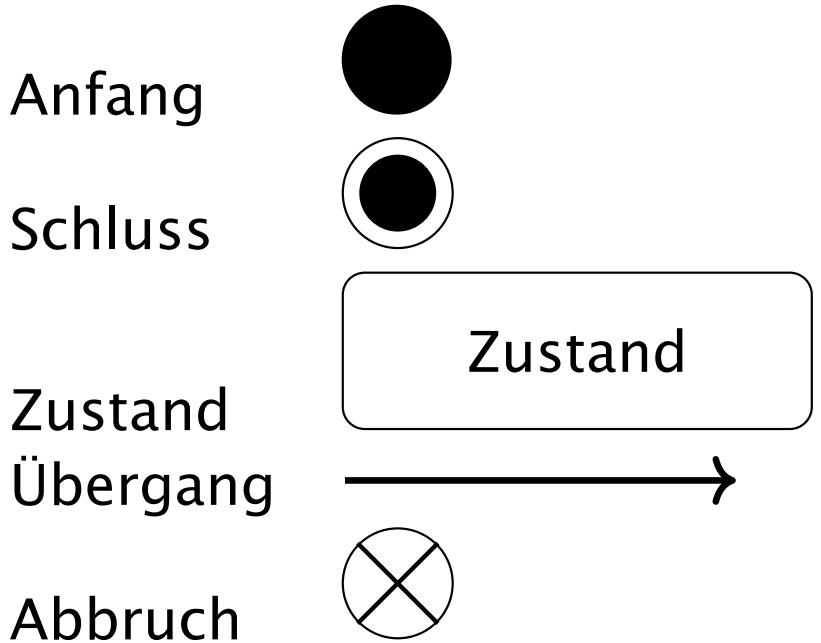


- Die Abbildung zeigt die Aktivität *Arznei herstellen*.
- Mögliche Parallelverarbeitung ist berücksichtigt.
- Die Endkontrolle kann dazu führen, dass Arbeitsvorgänge wiederholt werden müssen (nicht abgebildet).

Zustandsdiagramm

Ein System (z.B. ein Programm) wird als *endlicher Automat* betrachtet. Der Automat besteht aus

- einem Anfangszustand,
- Ereignissen,
- Zwischenzuständen,
- Zustandsübergängen und
- einem oder mehreren Endzuständen.



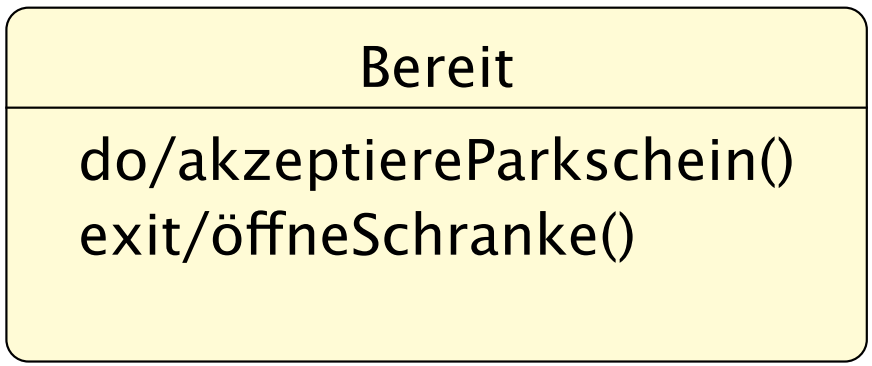
Ereignisse werden in Textform an Übergängen notiert.

Zustand

Interne Aktionen

Ein Zustand kann interne Aktionen enthalten:

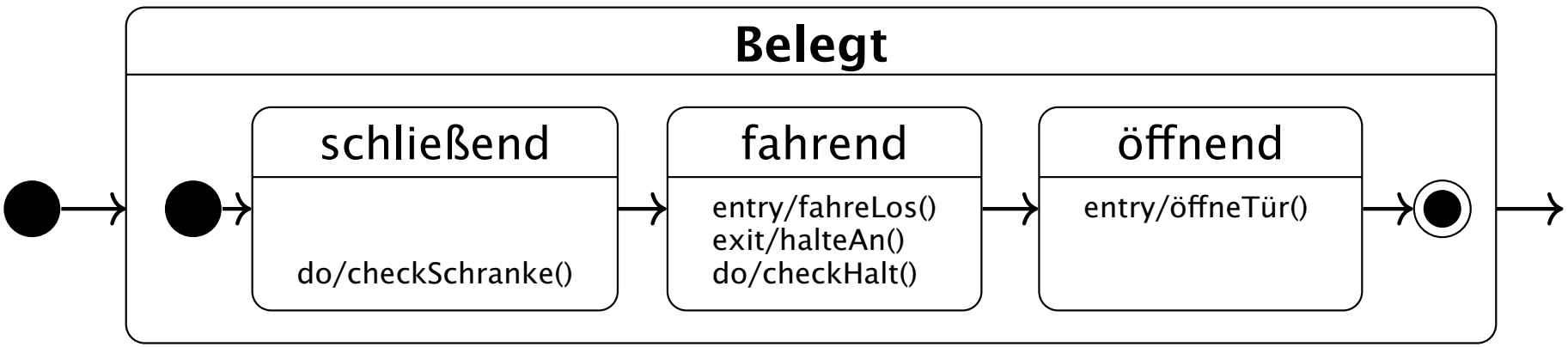
- **entry** wird beim Eintreten ausgelöst,
- **exit** wird beim Verlassen ausgelöst,
- **do** wird laufend ausgelöst, solange der Zustand besteht.



Beispiel: Eine Schranke am Parkhausausgang wartet auf einen Parkschein. Liegt er vor, geht sie auf.

Unterzustand

- Mehrere (Unter-)Zustände können zu einem Zustand zusammengefasst werden.
- Parallele Unterzustände werden durch eine gestrichelte Linie abgetrennt (nicht im Bild).



Ein Fahrstuhl: checkSchranke() steht für die Überprüfung der Lichtschranke und checkHalt() für die Prüfung, ob angehalten werden soll (Ziel erreicht oder Nothalt gedrückt). Nicht gezeigt ist der Zustand *Frei*.

Vergleich Zustands- und Aktivitätsdiagramm

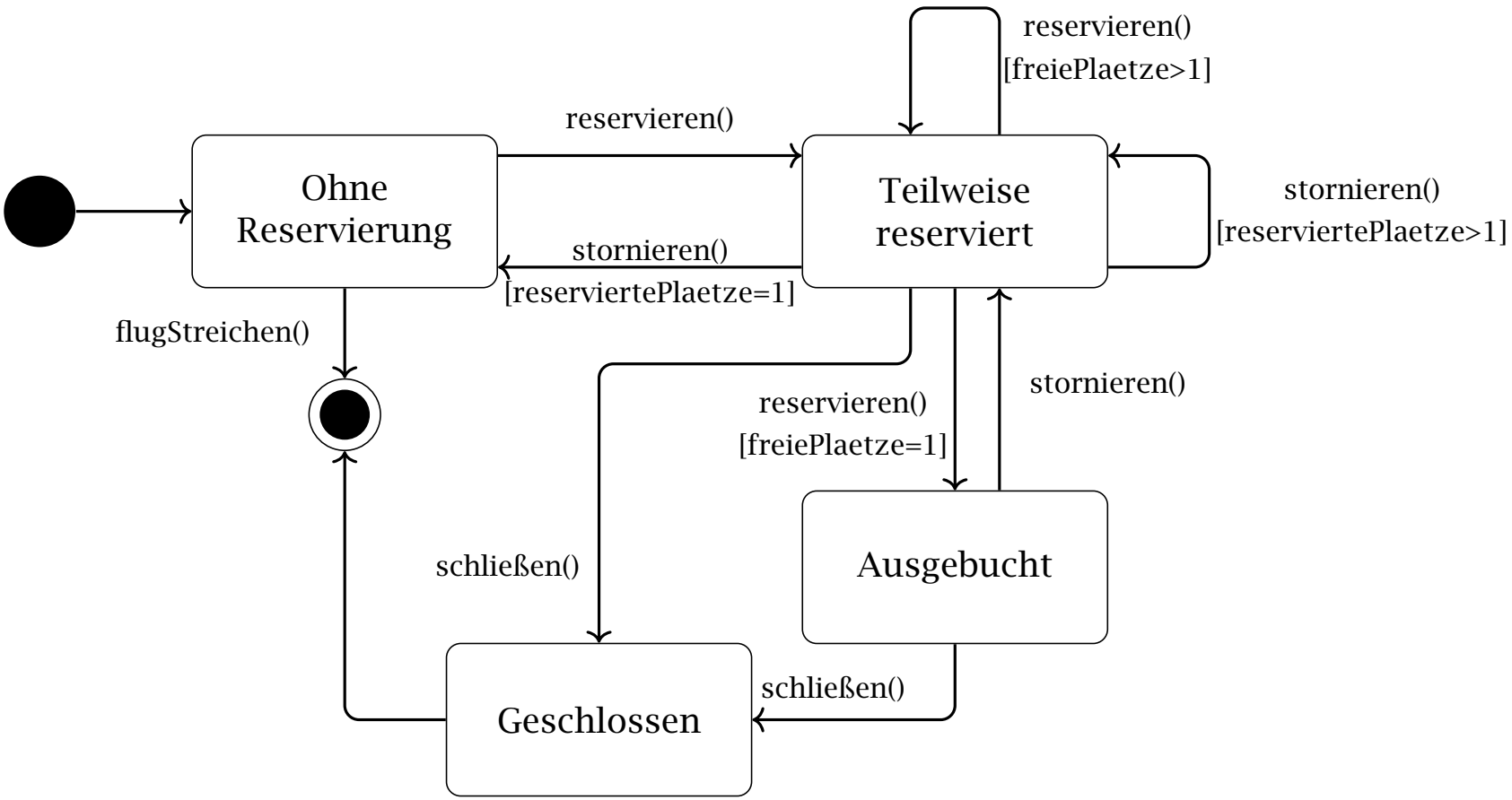
Aktivitätsdiagramm

- beschreibt geschäftliche Abläufe

Zustandsdiagramm

- beschreibt Zustände eines Objektes oder Subsystems und deren Übergänge

Flugbuchung



nach Oestereich, Bernd: Analyse und Design mit UML 2

7. Auflage. Oldenbourg, München 2005

UML-Werkzeuge

Es gibt Werkzeuge mit unterschiedlich guter UML-Unterstützung:

1. Zeichnen von UML-Diagrammen: Jedes vernünftige Zeichenprogramm (z.B. LibreOfficeDraw)
2. Erstellen von Programmcode aus dem Diagramm
3. Erstellen eines UML-Diagramms aus Programmcode (z.B. Umbrello, Papyrus)
4. Einlesen von Programmcode, ändern, ohne Verluste zurückschreiben
5. Intelligente Optimierung (Umstrukturierung) eines Klassendiagramms