# Kurzanleitung UML

Dieses Dokument befasst sich mit ausgewählten Diagrammen der UML.

## Inhaltsverzeichnis



## 1 Überblick

„Die Unified Modeling Language (UML) ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme.“[1] Sie ist durch die Object Management Group (siehe [1]) standardisiert. Die Modelle können durch verschiedene Diagrammtypen dargestellt werden. Grundsätzlich werden zwei Hauptdiagrammtypen unterschieden: Strukturdiagramme (Structure Diagram) und Verhaltensdiagramme (Behavior Diagram). Sie fassen jeweils weitere Diagrammformen zusammen. Eine Übersicht ist in Abbildung 1 auf der nächsten Seite zu finden. Die in der Abbildung grün makierten Diagrammarten werden in diesem Dokument näher beschrieben.
Neben den 14 unterschiedlichen Diagrammen beinhaltet die UML die **Object Constraint Language** (OCL). Sie dient dazu Diagrammen genauere Bedeutungen hinzuzufügen, die mit den jeweiligen Diagrammelementen nicht oder nur sehr umständlich ausgedrückt werden könnten. Eine kurze Einführung zur OCL ist in [6][Seite 336 ff.] zu finden.

Für knifflige Fragestellungen und ausführliche Details zu UML und ihren Diagrammtypen ist die UML-Spezifikation der OMG [4] eine gute Adresse. Die OCL-Spezifikation ist in [3] zu finden. Bei Unklarheiten und Abweichungen in Büchern ist die offizielle Referenz immer der erste Anlaufpunkt zur Klärung. Es sollte beachtet werden, dass jeweils die gültige Spezifikation zur Rate gezogen wird.

[1] entnommen aus [5][Seite 12]. Es bezieht sich auf UML 2.0, aktuell ist z. Zt. (Februar 2010) die Version 2.2. der Spezifikation.

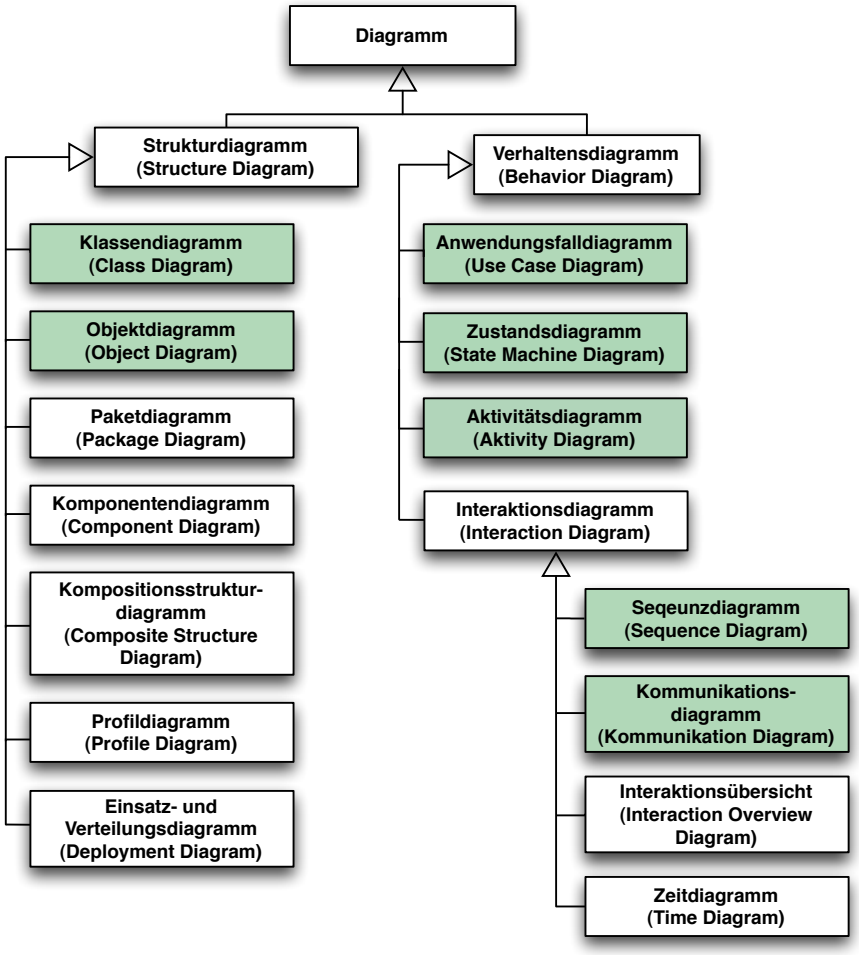


Abbildung 1: Übersicht Diagrammtypen UML

# 2 Strukturdiagramme

## 2.1 Klassendiagramm

Der Diagrammtyp *Klassendiagramm* dient dazu, eine oder mehrere Klassen, unabhängig von der Sprache in der sie implementiert wurden/werden sollen, abzubilden. Es können sowohl die Klasse(n) selbst als auch ihre Beziehungen zu anderen Klassen dargestellt werden. Je nach Menge der Codezeilen kann der Quellcode einer bzw. mehrerer Klassen sehr unübersichtlich sein. Die Idee hinter diesem Diagrammtyp ist, eine übersichtliche Form zu finden, die unabhängig von Kenntnissen über Details der Programmiersprache oder des Programms, zu verstehen ist. Klassendiagramme können sowohl zur Beschreibung eines schon in Code umgesetzten Programms als auch zur Modellierung eines Sachverhalts vor der konkreten Umsetzung in einer Programmiersprache verwendet werden. Klassendiagramme zählen zu den UML-Strukturdiagrammen.

***Beispiel 2.1***

*In Listing 1 ist die Klasse `Person` zu sehen. Sie besitzt zwei private Attribute für Vor- und Nachnamen und eine Klassenvariable, welche ein Trennzeichen zur Ausgabe beinhaltet.*

```
public class Person {
  private String vorname;
  private String nachname;
  private static String trennzeichen = ", ";

  public Person(String vorname, String nachname){
    this.vorname = vorname;
    this.nachname = nachname;
  }

  public String getVorname(){
    return this.vorname;
  }

  public void setVorname(String vorname){
    this.vorname = vorname;
  }

  public String getNachname(){
    return this.nachname;
  }

  public void setNachname(String nachname){
    this.nachname = nachname;
  }

  public static String showTrennzeichen(){
    return trennzeichen;
  }

  @Override
  public String toString(){
    return this.nachname  + trennzeichen + this.vorname;
  }
}
```

Listing 1: Java-Quellcode der Klasse `Person`

Das zu Listing 1 zugehörige Klassendiagramm ist in Abbildung 2 auf der nächsten Seite zu sehen.

Das Diagramm wird von einem Rahmen umschlossen, der in der linken oberen Ecke die Abkürzung CD für Class Diagramm enthält. Dieser Rahmen ist optional und wird häufig weggelassen. Innerhalb des Diagramms ist die Klasse `Person` mit Kommentaren versehen abgebildet.

Eine Klasse ist in drei Bereiche aufgeteilt. Der Kopf enthält den Namen der Klassen, er ist fett geschrieben. Die nächsten beiden Abschnitte beinhalten die Attribute bzw. Methoden. Die Sichtbarkeiten werden durch die Verwendung der Zeichen +, -, # und ~ notiert. Attribute sind ohne Vorzeichen automatisch private und Methoden public. Typen, Parameter und Namen können weggelassen werden, ebenso Attribute und Methoden. Ein Klassendiagramm kann somit auch ausschließlich aus Rechtecken mit Namen bestehen. Diese Abschnitte im Klassendiagramm werden als **Compartments** bezeichnet und können in beliebiger Reihenfolgen und Anzahl verwendet werden. Einzige Einschränkung ist, dass im obersten Abschnitt der Name der Klasse steht und dieser Teil nicht optional ist. Verbreitet ist die Ansicht, zunächst die Parameter und dann die Methoden aufzulisten.

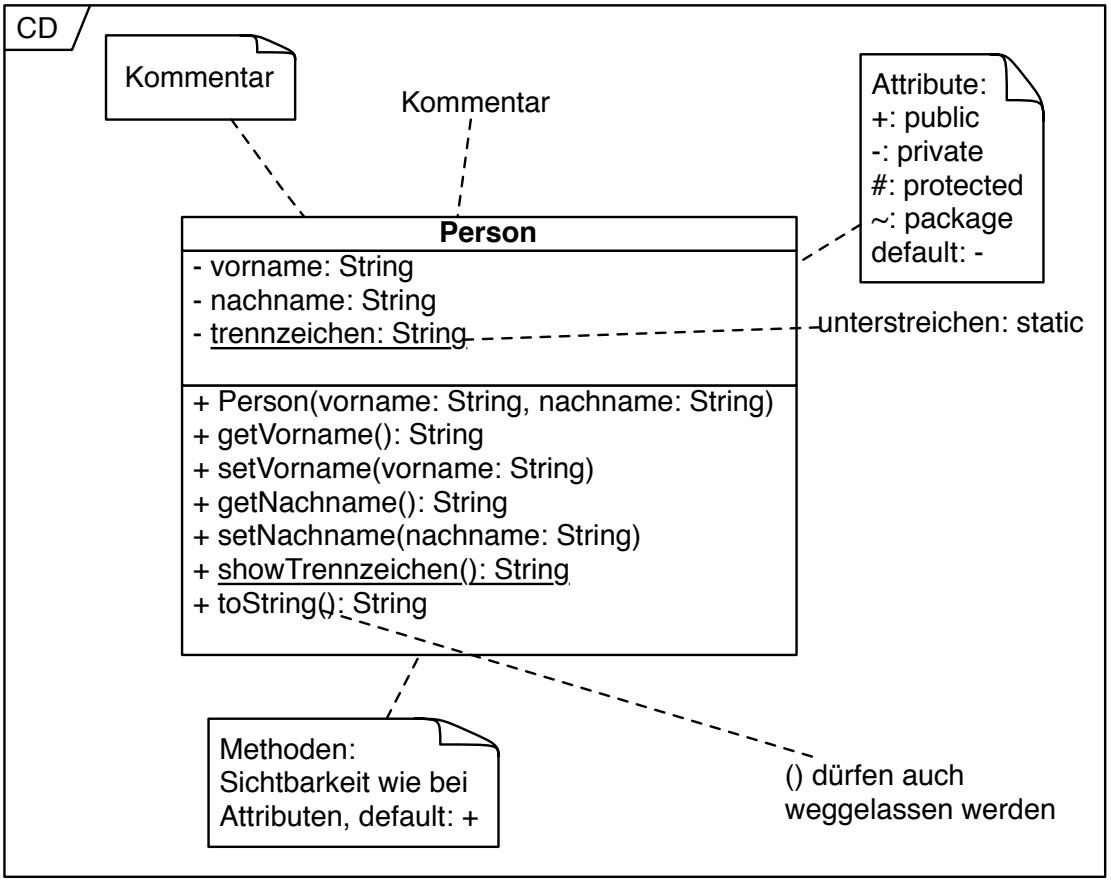


Abbildung 2: Diagramm zur Klasse „Person“

Statische Attribute oder Methoden werden durch Unterstreichen gekennzeichnet. Kommentare können durch Text oder Kommentarrechtecke verbunden mit einer gestrichelten Linie hizugefügt werden. In geschweiften Klammern können zusätzliche **Eigenschaften** angegeben werden (im Bild nicht dargestellt). **Stereotypen** werden durch spitze Klammern gekennzeichnet. Sie dienen dazu den Verwendungskontext zu spezifizieren. Jeder Verwender kann UML durch eigene Eigenschaften und Stereotypen erweitern. Sie sind nach gewissen Regeln frei wählbar, sollten aber nicht willkürlich durch den einzelnen Entwickler, sondern immer projekt- oder unternehmensbezogen definiert werden.

## 2.2 Objektdiagramm

Ein weiterer Typ der UML-Strukturdiagramme ist das Objektdiagramm. Es stellt eine Momentaufnahme der derzeitigen Situation der Objekte dar. Klassendiagramme bilden dagegen die statischen Informationen und Beziehungen zwischen Klassen ab.

***Beispiel 2.2***

*In einen Programm (siehe Listing 2) werden zwei Instanzen der Klasse `Person` angelegt. Ein zugehöriges Objektdiagramm ist in Abbildung 3 zu sehen. Die Definition der Klasse `Person` entspricht der in Listing 1.*

```
public class Person {
...
  public static void main(String args[]) {
    Person p1 = new Person("p", "1");
    Person p2 = new Person("p", "2");
  }
}
```

Listing 2: Beispielhafte Verwendung der Klasse `Person`

*Die Abbildung 3 zeigt zwei Objekte, die jeweils mit dem Namen und dem Typ der Referenz und den Parametern `vorname` und `nachname` versehen sind. Das Attribut `trennzeichen` ist nicht zu finden, da es zur gesamten Klasse und nicht zu den jeweiligen Objekten gehört.*

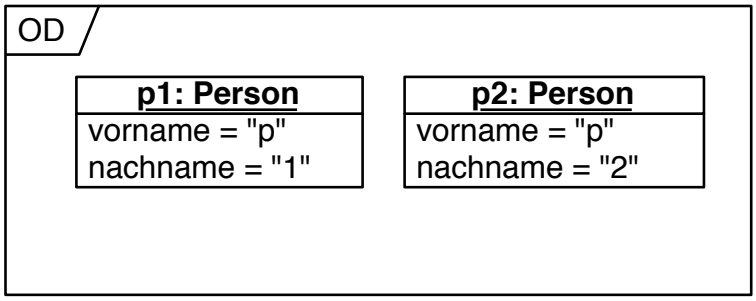


Abbildung 3: Ein mögliches Objektdiagramm zur Klasse `Person`

Ein Objekt innerhalb eines Objektdiagramms wird durch ein Rechteck dargestellt. Es ist in zwei Bereiche aufgeteilt. Im oberen Teil wird der Name und danach der Typ des Objektes notiert. Beides muss unterstrichen werden. Im unteren Bereich werden die Attribute mit den derzeitigen Werten erfasst. Sowohl der Name der Referenz (in diesem Fall z.B. p1) als auch die Attribute und ihre Werte können weggelassen werden. Das Diagramm kann optional durch einen Rahmen begrenzt werden. Verwendet man ihn, wird oben links klein die Abkürzung OD für Object Diagramm notiert.

## 2.3 Klassendiagramme mit Assoziationen

Klassendiagramme können neben Klassen und Kommentaren auch Assoziationen enthalten. Assoziationen bilden die Beziehungen zwischen Klassen ab. Zur genaueren Spezifizierung können sie (müssen aber nicht) mit Multiplizitäten versehen werden. Eine **Multiplizität** gibt den Bereich der erlaubten Kardinalitäten an. Eine **Kardinalität** bezeichnet die Anzahl der zulässigen Objekte. Keine Angabe einer Multiplizität entspricht der Angabe 1. Neben der Multiplizität können sie auch durch einen Namen gekennzeichnet werden. Dies ist sinnvoll, wenn zwischen zwei Objekten zwei unterschiedliche Beziehungen bestehen sollen.

Die ungefüllte Raute an einem Ende wird als Aggregation bezeichnet und drückt aus, dass Objekte der Klasse B zu Objekten der Klasse A gehören, B-Objekte aber auch allein existieren können. Die gefüllte Raute wird als Komposition bezeichnet und drückt aus, dass Objekte der Klasse B nur dann existieren

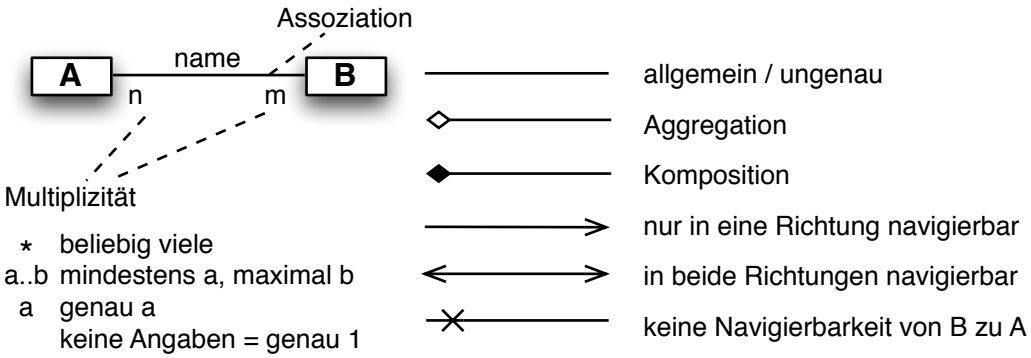


Abbildung 4: Auswahl an möglichen Assoziationen bzw. Mutiplizitäten

können, wenn das/die zugehörigen Objekte der Klasse A existieren. Stirbt das Objekt der Klasse A, müssen auch die zugehörigen Objekte der Klasse B sterben. Umgekehrt, wird ein B-Objekt gelöscht, bleibt das zugehörige A-Objekt davon unberührt.

Die einfache Linie trifft keine Aussagen über die Existenzabhängigkeit der Objekte dieser Klassen. Eine Pfeilspitze an einem oder beiden Enden der Assoziation zeigt an, ob die Objekte der anderen Klasse direkt über eine Referenz erreichbar sind. Befindet sich z.B. an der rechten Seite der Assoziation eine Pfeilspitze, kann von einem Objekt des Typs A ein Objekt des Typs B erreicht werden. Zur Navigierbarkeit von Objekt desTyps B zu Objekt des Typs A wird keine Aussage getroffen. Möchte man ausdrücken, dass A von B aus nicht direkt durch eine Referenz erreicht werden kann, muss dies durch ein Kreuz kenntlich gemacht werden.

In Java ist die Aggregation der *Normalfall*. In C++ wird die Aggregation durch die Verwendung eines Zeigers implementiert. Die Komposition kann in Java nicht direkt abgebildet werden, in C++ ist dies sehr wohl möglich: das ist der Normalfall bei Attributen in Klassen in C++.

Die bisher vorgestellten Assoziationen sind binär, d.h. an einer Beziehung sind zwei Klassen beteiligt. Diese binäre Assoziation stellt eine Spezialform der **n-ären Assoziation** dar. Der Begriff n-äre Assoziation dient als Sammelbezeichnung für alle Assoziationen mit mehr als zwei Enden (n>2). Details siehe [7][Seite 149].

***Beispiel 2.3***

*Es wird eine Klasse `Paar` erstellt. Jedes Objekt der Klasse besteht aus zwei Objekten der Klasse `Person`.*

```
public class Paar {
    private Person[] personen;

    public Paar(Person a, Person b){
        personen = new Person[2];
        personen[0] = a;
        personen[1] = b;
    }
}
```

Listing 3: Klasse `Paar`

*Abbildung 5 zeigt zwei Klassendiagramm-Varianten, die beide zum Quellcode in Listing 3 auf der vorherigen Seite passen.*

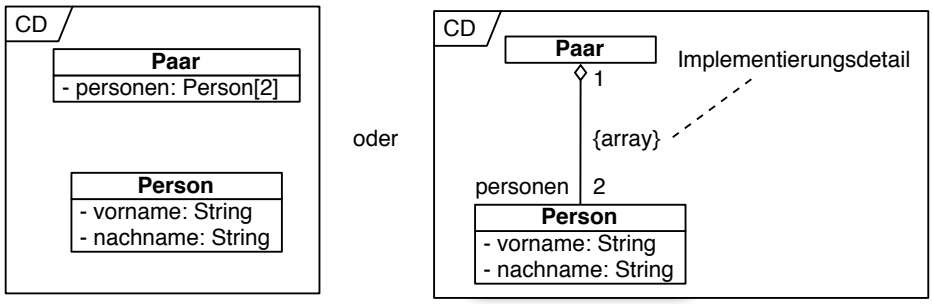


Abbildung 5: Klassendiagramm zur Klasse `Paar`

*Es werden zwei unterschiedliche Darstellungen verwendet. Auf der linken Seite ist die Variante ohne gezeichnete Assoziationen gezeigt. Die Verbindung zwischen den Klassen wird durch das Attribut `personen` in der Klasse `Paar` dargestellt. Der rechte Teil der Abbildung verwendet die Notation der Aggregation. Die Klasse `Person` ist ein Teil der Klasse `Paar`. Die ungefüllte Raute gibt an, dass ein Objekt von `Person` auch ohne die Existenz eines `Paar`-Objektes weiterleben kann.*

*Möchte man inhaltlich ausdrücken, dass ein Objekt von `Person` nur im Zusammenhang mit der Existenz eines `Paar`-Objektes existieren kann, würde man die gefüllte Raute verwenden. Diese Semantik lässt sich mit Java nicht direkt umsetzen. In C++ ist es einfach möglich zu erzwingen, dass die Personen-Objekte ebenfalls sterben, wenn das Paar-Objekt stirbt. Für eine Implementierung im Java-Umfeld sollte in der Regel die ungefüllte Raute verwendet werden.*

*Die Zahlen an den jeweiligen Enden werden als Multiplizitäten bezeichnet. Sie geben an, dass ein `Paar`-Objekt mit zwei `Person`-Objekten verbunden sein muss. Andersherum darf ein `Person`-Objekt nur genau einem `Paar`-Objekt zugeordnet sein. Ziffer 1 an der Raute des Aggregation kann auch weggelassen werden, die Bedeutung würde sich nicht ändern.*

*Um bei der Verwendung einer Assoziation klarzustellen, über welches Attribut diese Assoziation dargestellt wird, kann der Name dieses Attributs an die Assoziation geschrieben werden, und zwar an die Seite, an der sich die Klasse mit dem Typ des Attributs befindet. Im rechten Klassendiagramm in Abbildung 5 steht also der Name `personen` am unteren Ende, also an dem Ende der Assoziation, das mit der Klasse `Person` verbunden ist.*

***Beispiel 2.4***
*Es wird die Klasse `Gruppe` definiert. Eine Gruppe besteht aus mehreren Personen. Eine Person darf mehreren Gruppen angehören. (Listing 4 auf der nächsten Seite)*

```java
public class Gruppe {

    private List<Person> personen;

    public Gruppe(){
        personen = new Vector<Person>();
    }

    public void addPerson(Person person){
        personen.add(person);
    }

    public void removePerson(Person person){
        personen.remove(person);
    }
}
```

Listing 4: Definition und beispielhafte Verwendung der Klasse `Gruppe`

*Abbildung 6 zeigt zwei Varianten eines passenden Klassendiagramms zur Klasse `Gruppe`. Diesmal sind in der rechten Grafik an den Enden der Assoziationen keine Zahlen, sondern $*$ zu finden. Sie stehen für 0 bis n Verbindungen. Denn eine Person darf keiner oder mehreren Gruppen angehören und eine Gruppe besteht aus beliebig vielen (auch keinen) Personen.*

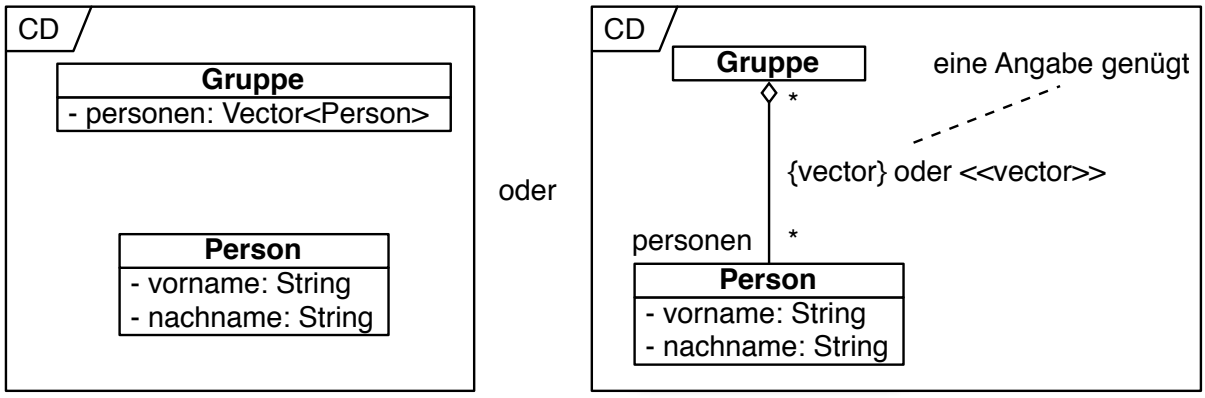


Abbildung 6: Klassendiagramm zur Klasse `Gruppe`

*In beiden Varianten wird festgelegt, wie die Assoziation zwischen den beiden Klassen zu implementieren ist: als `Vector<Person>`. In der linken Variante wird dies durch die Vorgabe des Typs `Vector<Person>` des Attributs `personen` festgelegt, in der rechten Variante durch die Verwendung der Eigenschaft `{vector}` bzw. des Stereotyps `<<vector>>`.*

*Beide Varianten können innerhalb eines Klassendiagramms verwendet werden, aber nicht gleichzeitig für das gleiche Attribut. Es wäre z.B. also falsch, sowohl das Attribut `personen` in die Klasse `Gruppe` zu schreiben als auch die entsprechende Assoziation zwischen den Klassen `Gruppe` und `Person` zu zeichnen.*

In der UML gibt es für dieselben Informationen verschiedene Ausdrucksweisen. Sie ist leider nicht immer eindeutig und sollte ggf. durch ausreichenden Begleittext erläutert werden. Hinzu kommt, dass sich verschiedene offizielle UML-Standards im Umlauf befinden. Im Moment ist die Version 2.2 aktuell.

## 2.4 Klassendiagramme mit Vererbung

Klassendiagramme können auch Vererbungsbeziehungen[2] zwischen Klassen abbilden. Eine Vererbungsbeziehung wird durch eine Linie, die am Ende eine ungefüllte Pfeilspitze besitzt, dargestellt. Diese Art der Verbindung darf nicht mit der gerichteten Assoziation verwechselt werden.

Sowohl Assoziationen als auch Vererbungsbeziehungen stellen Verbindungen zwischen Klassen in Klassendiagrammen dar. Es besteht jedoch ein wichtiger Unterschied: Assoziationen geben an, dass Objekte der einen Klasse mit Objekten der anderen Klasse verbunden sind. Solche Verbindungen sind dann in Form von Links in Objektdiagrammen (s.u.) auch sichtbar. Eine Vererbungsbeziehung spielt sich rein auf der Klassenebene ab und ist dementsprechend im Objektdiagramm nicht sichtbar.

***Beispiel 2.5***
*Die Vererbungshierarchie zwischen der Klasse `GeometrischeFigur` und den Subklassen `Kreis`, `Dreieck` und `Rechteck` ist in Abbildung 7 zu sehen.*

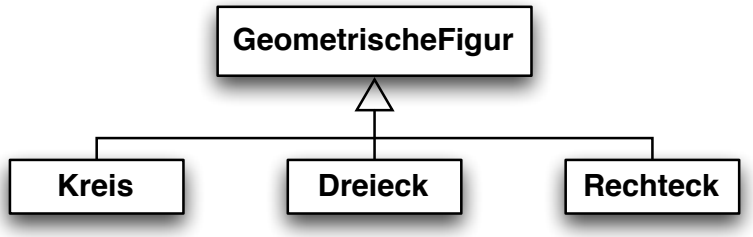


Abbildung 7: Klassendiagramm mit Vererbungsbeziehungen

Listing 5 stellt auszugsweise den Java-Quellcode zu Abbildung 7 dar.

```
public class GeometrischeFigur {...}
public class Kreis extends GeometrischeFigur {...}
public class Dreieck extends GeometrischeFigur {...}
public class Rechteck extends GeometrischeFigur {...}
```

Listing 5: Auszug aus dem Java-Quellcode zu Abbildung 7

***Beispiel 2.6***
*Abbildung 8 auf der nächsten Seite zeigt das Klassendiagramm einer Zoo-Implementierung. Ein Zoo besteht aus mehreren Käfigen. Käfige enthalten Tiere. Tiere werden in Unterklassen genauer spezifiziert. Die Klasse `Tier` ist abstract, von ihr dürfen keine Objekte erzeugt werden.*

[2] In Klassendiagrammen kann auch Generalisierung/Spezialisierung dargestellt werden, wie sie z.B. im erweiterten Entity-Relationship-Diagramm eingesetzt wird. Diese Möglichkeit kann man nicht direkt in Konstrukte einer objektorientierten Programmiersprache wie Java übertragen, deshalb wollen wir sie in dieser Kurzanleitung nicht weiter betrachten.

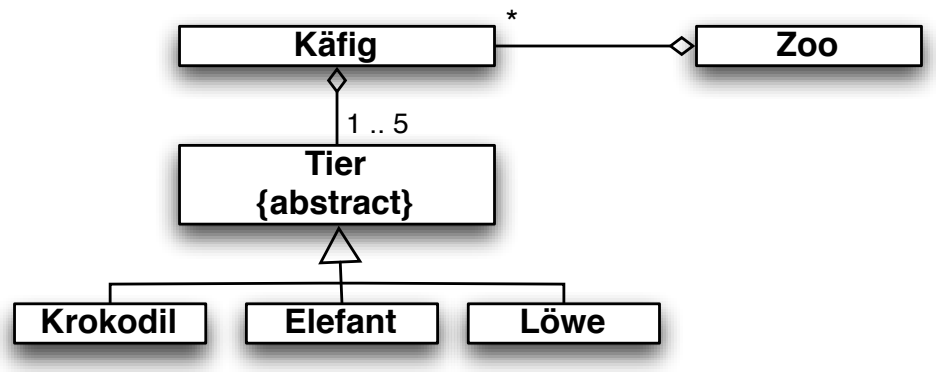


Abbildung 8: Klassendiagramm mit Vererbungsbeziehungen und Assoziationen

Neben der Vererbung lässt sich auch die Interfaceimplementierung mit UML darstellen.

***Beispiel 2.7***
*Abbildung 9 zeigt das Klassendiagramm einer Klassenstruktur, welche die gemeinschaftliche Eigenschaft singend besitzt. Es ist leicht ersichtlich, dass die drei abgebildeten Klassen ansonsten keinerlei Gemeinsamkeiten besitzen und nicht durch eine Vererbungsbeziehung verbunden sein sollten. Das Interface* `Singend` *ist eine Eigenschaft und keine Variante einer Klasse von der Objekte erzeugt werden sollten.*

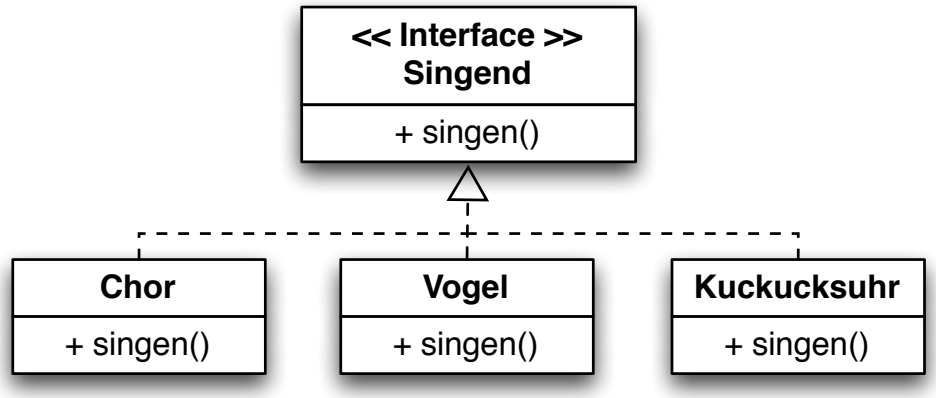


Abbildung 9: Klassendiagramm mit Interface-Implementierung

## 2.5 Objektdiagramme mit Links

Beziehungen zwischen Klassen in einem Klassendiagramm werden als Assoziationen bezeichnet. Die entsprechenden Beziehungen im Objektdiagramm werden als **Links** oder **Objektbeziehungen** bezeichnet.

***Beispiel 2.8***

*Es wird ein Objekt der Klasse `Paar` erstelllt. (Listing 6)*

```
public static void main(String args[]) {
  ...
  Paar paar = new Paar(p1, p2);
  }
```

Listing 6: Beispielhafte Verwendung der Klasse `Paar`

*Die Abbildung 10 zeigt das passende Objektdiagramm. Die Variante auf der rechten Seite verwendet Links (Beziehungen in Objektdiagrammen), um eine Verbindung zwischen den Objekten zu beschreiben. Die ungefüllte Raute am Objekt `Paar` deutet an, dass ein Person-Objekt Teil des Paar-Objekt ist. Wenn das Paar-Objekt gelöscht wird, können die zugeordneten Personen-Objekt weiter existieren.*

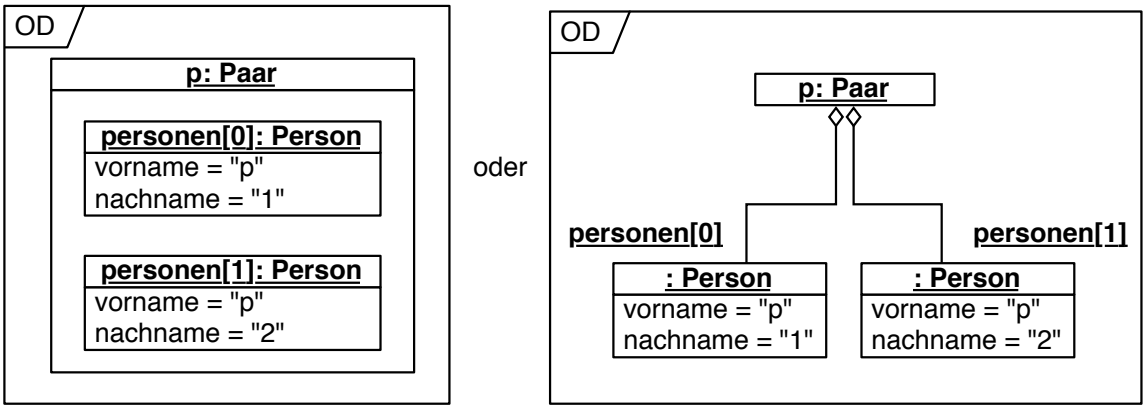


Abbildung 10: Ein mögliches Objektdiagramm zur Klasse `Paar`

*Viele der Angaben in den Diagrammen sind optional. Die Attribute der Klasse `Person` können z.B. entfallen. Abhängig davon, welchen Zweck das Diagramm erfüllen soll, kann die Detailstärke variiert werden. Neben diesen beiden Darstellungen gibt es noch weitere Darstellungsvarianten.*

***Beispiel 2.9***

*In Listing 7 wird die Klasse `Gruppe` verwendet.*

```
public static void main(String args[]) {
  ...
  Gruppe gruppe = new Gruppe();
  gruppe.addPerson(p1);
  gruppe.addPerson(p2);
  gruppe.addPerson(p3);
  gruppe.removePerson(p1);
}
```

Listing 7: Definition und beispielhafte Verwendung der Klasse `Gruppe`

*Das Objektdiagramm in Abbildung 11 zeigt eine Momentaufnahme der Objektsituation nach Hinzufügen der dritten Person in die Gruppe (nach `gruppe.addPerson(p3);`).*

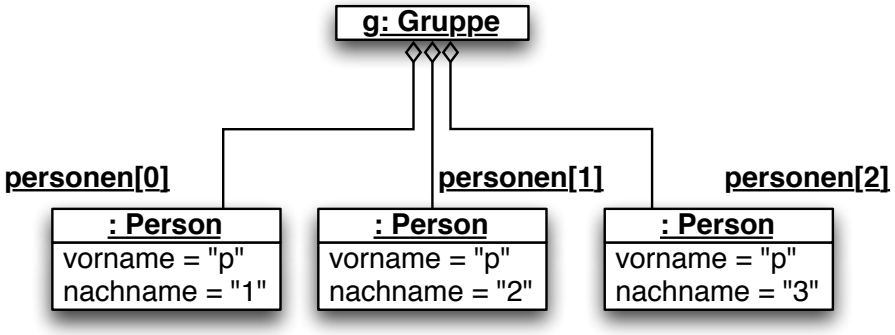


Abbildung 11: Ein mögliches Objektdiagramm zur Klasse `Gruppe`

***Beispiel 2.10***
*Abbildung 12 zeigt ein mögliches Objektdiagramm zu Beispiel 2.6 auf Seite 9.*

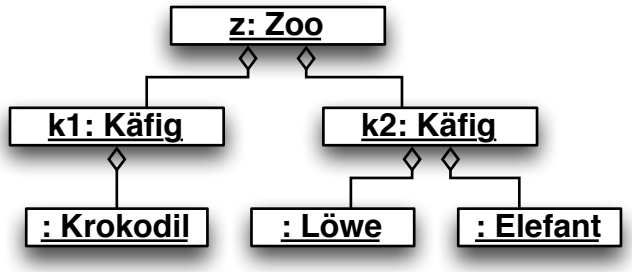


Abbildung 12: Objektdiagramm zu Beispiel 2.6

*Das Diagramm enthält keine Objekte der Klasse `Tier`. Dies ist dadurch begründet, dass in diesem Fall keine konkreten Objekte von Tier erzeugt werden dürfen, da die Klasse `Tier` als abstrakt gekennzeichnet ist. Auffällig ist außerdem, dass sich in einem gemeinsamen Käfig Löwen und Elefanten befinden. Dieser Zustand wird wohl eher nicht der realen Welt entsprechen, aber im Klassendiagramme werden dazu keine Einschränkungen gemacht.*

# 3 Verhaltensdiagramme

Die zweite große Gruppe bilden die Verhaltensdiagramme. Sie bilden dynamische Situationen im Programmablauf ab.

## 3.1 Sequenzdiagramm

Das Sequenzdiagramm ist ein Vertreter der Verhaltensdiagramme und gehört zur Untergruppe der Interaktionsdiagramme. Es ist die *aktive* Form des Objektdiagramms, d.h. es wird nicht nur der momentane Zustand der Objekte (ohne Attribute) abgebildet, sondern ein (ganzer) Ablauf. Im besonderen steht der zeitliche Verlauf der Kommunikation im Vordergrund. Die Zeit läuft von oben nach unten. Sequenzdiagramme dürfen als Beteiligte Objekte und weitere Akteure beinhalten. Die Beteiligten kommunizieren über Nachrichten. Sie werden durch ein Rechteck mit Beschriftung, an dem unten eine vertikale, gestrichelte Linie angebracht ist, dargestellt. Die Gesamtheit aus Rechteck und gestrichelter Linie wird als Lebenslinie bezeichnet.[3]

***Beispiel 3.1***
*Im Listing 8 wird eine Verwendung der Klasse `Gruppe` gezeigt.*

```
public class Prog{
  public static void main(String args[]) {
    Person p1 = new Person("p","1");
    Person p2 = new Person("p","2");
    Gruppe gruppe = new Gruppe();
    gruppe.addPerson(p1);
    gruppe.addPerson(p2);
    gruppe.removePerson(p1);
  }
}
```

Listing 8: Beispielhafte Verwendung der Klasse `Gruppe`

*Abbildung 13 auf der nächsten Seite zeigt ein Sequenzdiagramm, welches diesen Sachverhalt darstellt. Hier wurde auf den Rahmen mit der Kennzeichnung SD verzichtet, er ist wie bei alle Diagrammen optional.*

*Zunächst werden zwei `Person`-Objekte und ein `Gruppe`-Objekt angelegt. Wird der Konstruktor der Klasse `Gruppe` aufgerufen, wird ein neuer `Vector`, der Personen aufnimmt, erzeugt. Durch Anwendung der Methode `addPerson(p1)` wird die Methode `add(p1)` des Vectors aufgerufen und die Person `p1` hinzugefügt. Analog gilt dies für `p2`. Durch `removePerson(p1)` wird die Person wieder aus der Gruppe heraus gelöscht.*

*Wenn auf ein Objekt keine Referenzen mehr verweisen, dann wird dieses Objekt beim nächsten Lauf des Garbage Collectors vermutlich aufgeräumt. Wann das Objekt tatsächlich vernichtet wird, ist ungewiss. Deshalb sollte Vorsicht geboten sein mit der Verwendung von X zur Signalisierung, dass ein Objekt tot ist.*

*Hier im Beispiel wird ein vollständiger Lebenszyklus dargestellt. Beginnend mit der Erzeugung und abschließend mit dem Tod der Objekte. Es ist nicht Pflicht diesen Umfang abzubilden. Häufiger sind Ausschnitte interessanter Stellen zu finden.*

[3] Der unbefangene Betrachter würde vielleicht nur die gestrichelte Linie als Lebenslinie bezeichnen wollen.

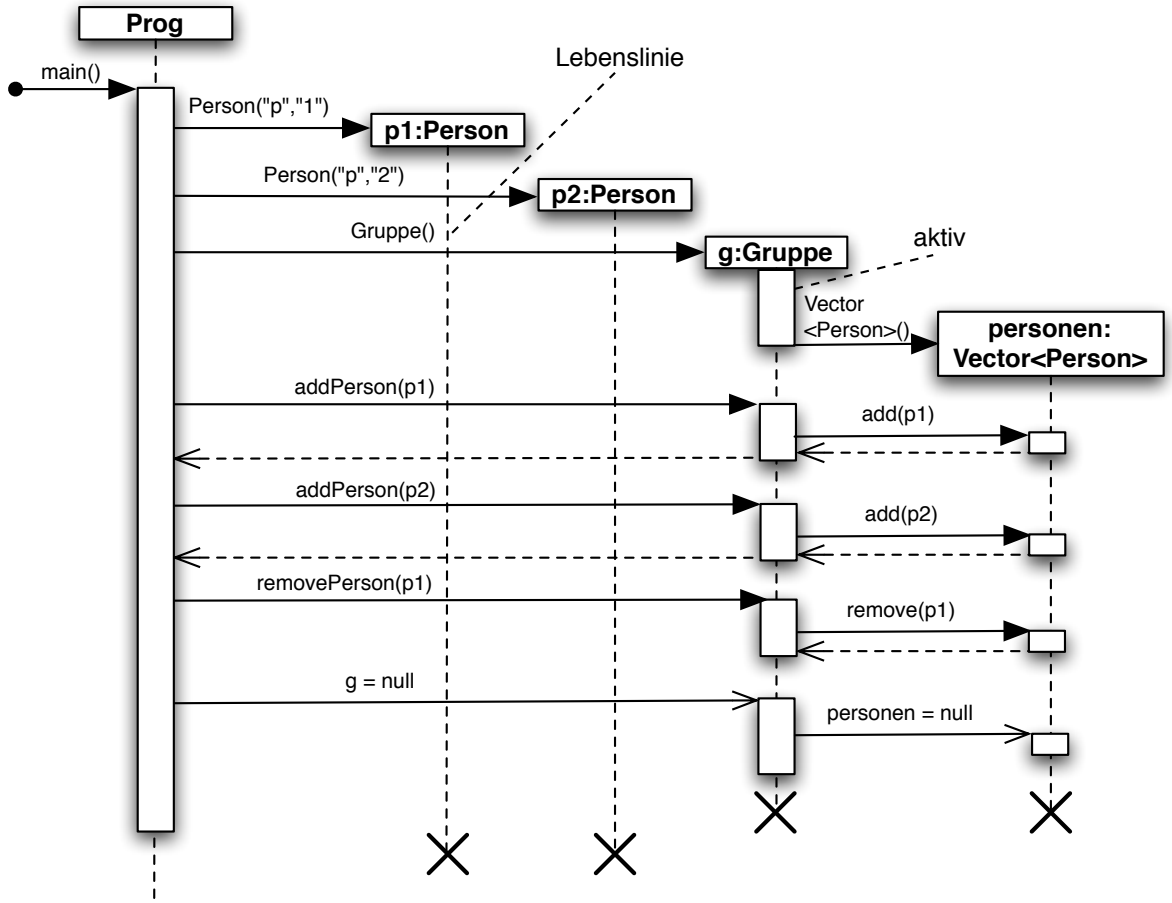


Abbildung 13: Sequenzdiagramm zur Verwendung der Klasse `Gruppe`

Sequenzdiagramme können auf zwei Arten dargestellt werden, entweder als ein bestimmtes Szenario mit konkreten Werten (wie in Beispiel 3.1) oder als Menge mehrerer oder sogar aller möglicher Szenarien. Im Beispiel wurden ausschließlich Objekte betrachtet. Neben Objekten können auch weitere Teilnehmer der Kommunikation dargestellt werden. Sie werden übergreifend als Lebenslinien bezeichnet und bestehen aus Rechtecken zusammen mit gestrichelten, senkrechten Linien, die ihren Ausgangspunkt an den Rechtecken haben. Als Beschriftung enthalten die Rechtecke Namen. Entspricht eine Lebenslinie einem Objekt, sollte zumindest der Typ (= Name der Klasse) und ggf. ein konkreter Name verwendet werden. Anders als im Objektdiagramm werden die Beschriftungen nicht unterstrichen. Kommentare sind durch Wörter an gestrichelten Linien gekennzeichnet.

Rechtecke auf den gestrichelten Linien signalisieren, wann eine Lebenslinie aktiv ist. Die Angabe dieser aktiven Bereiche ist optional. Methodenaufrufe und weitere Kommunikationsmöglichkeiten der Akteure untereinander werden durch Nachrichten dargestellt, die aus einer Beschreibung (z.B. `addPerson(p1)`) und einem waagerechten Pfeil bestehen.

In Abbildung 14 auf der nächsten Seite sind drei mögliche Pfeilarten abgebildet. Eine synchrone Nachricht besitzt eine schwarze, gefüllte Pfeilspitze und wird mit einer Antwort (ungefüllte Pfeilspitze, gestrichelte Linie) quittiert. Synchron bedeutet in diesem Zusammenhang, dass die aufrufende Lebenslinie so lange

blockiert, bis sie eine Antwort bekommt. Die Angabe des Antwortpfeils ist optional. Sie ist aber obligatorisch, wenn die Methode einen Wert zurück liefert.

Eine asynchrone Nachricht wird durch eine ungefüllte Pfeilspitze an einer durchgehenden Linie dargestellt. Asynchron bedeutet, dass der Sender der Nachricht nicht auf eine Antwort wartet und nicht blockiert. Die unteren drei Pfeile sind von ihrer Bedeutung identisch, bis auf dass ihre Herkunft bzw. ihr Ziel unbestimmt ist.

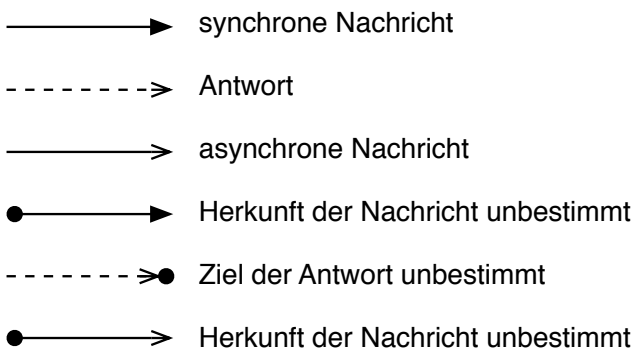


Abbildung 14: Pfeilarten in Sequenzdiagrammen

Abbildung 13 auf der vorherigen Seite zeigt nur einen Ausschnitt der Möglichkeiten. Die Darstellungsmöglichkeiten von Sequenzdiagrammen sind um einiges vielfältiger. Details dazu sind z.B. in [7][Seite 397 ff] zu finden. Sequenzdiagramme können z.B. auch verwendet werden, um Verzweigungen und Schleifen darzustellen. Dadurch werden Sequenzdiagramme eher unübersichtlich. Am besten verwendet man Sequenzdiagramme zur beispielhaften Veranschaulichung von Vorgängen.

## 3.2 Kommunikationsdiagramm

Wie das Sequenzdiagramm ist auch das Kommunikationsdiagramm ein Vertreter der Interaktionsdiagramme. Die Perspektive ist aber eine andere, im Mittelpunkt stehen Objekte und weitere Teilnehmer des Systems und ihre Zusammenarbeit und nicht der zeitliche Verlauf. Grundsätzlich eignen sich Interaktionsdiagramme nicht für die Darstellung von Schleifen und Verzweigungen.

***Beispiel 3.2***
*Abbildung 15 auf der nächsten Seite zeigt ein Kommunikationsdiagramm zum Sachverhalt in Listing 8.*

*Im Diagramm ist das Objekt g der Klasse* `Gruppe` *in der Mitte abgebildet. Es hat zwei Verbindungen zu* `Person`*-Objekten und eine zu einem* `ArrayList`*-Objekt. Die zeitliche Abfolge der Methodenaufrufe wird durch Zahlen an den Nachrichten dargestellt. Beginnend bei (1) wird zunächst ein* `Person`*-Objekt erstellt, dann das zweite (2) und danach ein* `Gruppe`*-Objekt (3). Durch die Nachricht* 3: Gruppe() *wird eine weitere Nachricht* 3.1: new Vector<Person>() *ausgelöst. Dann wird Nachricht (4) verarbeitet und ein* `Person`*-Objekt der Liste hinzugefügt (4.1). Analog dazu läuft (5) und (5.1) ab. Als letzter Schritt wird (6) verarbeitet und ein* `Person`*-Objet aus dem* `Vector`*-Objekt gelöscht (6.1).*

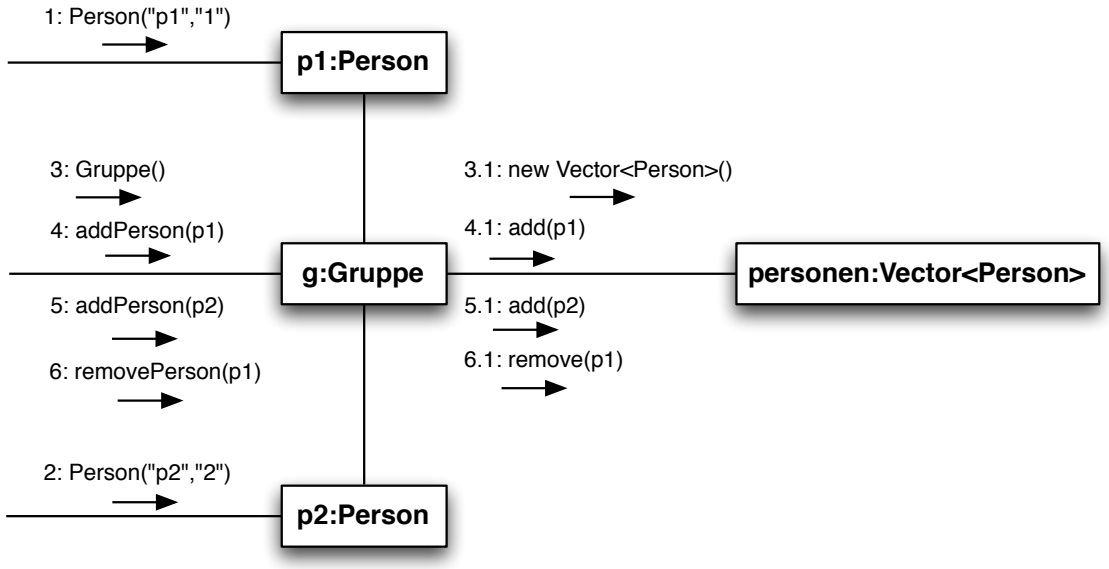


Abbildung 15: Kommunikationsdiagramm zur Verwendung der Klasse `Gruppe`

Im Prinzip enthalten Kommunikationsdiagramme genau die gleiche Art an Beteiligten wie Sequenzdiagramme. Die Beteiligten werden in Kommunikationsdiagrammen jedoch nur als Rechtecke dargestellt. Die gestrichelte, vertikale Linie entfällt. Aber obacht: diese Rechtecke heißen dennoch Lebenslinien. Beziehungen zwischen den Lebenslinien (also zwischen den Rechtecken) werden als einfache Linien gezeichnet. Die Nachrichten, die zwischen den Lebenslinien versendet werden, werden mit einer Richtungsangabe versehen und nach ihrer zeitlichen Abfolge durchnummeriert. Die Pfeilart (gefüllt vs. ungefüllt) hängt von der UML-Version ab. Die gezeigten entsprechen Version 2.1. Nachrichten, die von anderen Nachrichten abhängig sind, werden durch dieselbe Zahl und mit eine durch Punkt getrennte weitere Zahl z.B. (5.1), dargestellt. Die Reihenfolge kann beliebig tiefer geschachtelt werden, z.B. (4), (4.1), (4.2), (4.2.1) und (4.3).

## 3.3 Zustandsdiagramm

Ein Zustandsdiagramm zählt zu den Verhaltensdiagrammen. Es wird auch als State Machine oder endlicher Zustandsautomat bezeichnet und zeigt die Zustände, die ein Objekt im Lauf seines Lebens einnehmen kann. Durch **Ereignisse** (Events) werden **Zustandsübergänge** (Transitions) ausgelöst. Sie werden bevorzugt verwendet, um asynchrone[4] Vorgänge abzubilden. Der abgebildete Lebenszyklus kann über mehrere Anwendungsfälle hinweggehen. Besonders gut lassen sich auch ereignisgesteuerte Vorgänge wie z.B. GUIs damit beschreiben. Das Verhalten beim Eintreten eines Ereignisses hängt immer vom aktuellen Zustand ab.

***Beispiel 3.3***
*Aus der Sicht des Standesamtes kann eine Person in ihrem Leben verschiedene Zustände einnehmen. Dieser*

[4] Asynchron bedeutet, dass auf eine kurze Aktion eine lange Wartezeit folgen kann und kein blockierender Aufruf vorliegt.

*Sachverhalt wird in einem Zustandsdiagramm gezeigt. Abbildung 16 zeigt Klassen im Zusammenhang mit der Abbildung des Familienstandes.*

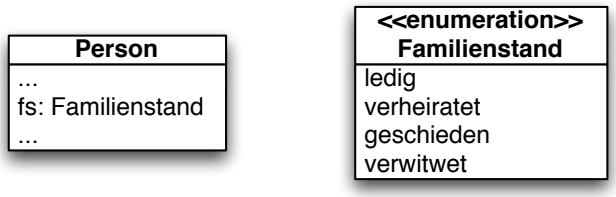


Abbildung 16: Klassen zum Sachverhalt „Familienstände“

*Die Abbildung 17 zeigt das zugehörige Diagramm. Vom Startzustand (schwarzer Kreis) aus erfolgt das Ereignis „Geburt“, wodurch der Zustand „ledig“ erreicht wird. Von hier und jedem anderen Zustand aus kann durch das Ereignis „Tod“ der Endzustand erreicht werden. Vom Zustand „ledig“ aus, wird durch das Ereignis „Heirat“ der Zustand „verheiratet“ erreicht. Tritt nun das Ereignis „Scheidung“ auf, geht das Objekt in den Zustand „geschieden“ über. Von diesem und vom Zustand „verwitwet“ kann durch eine Heirat wieder der Zustand „verheiratet“ erreicht werden. Das Ereignis „Tod des Partners“ bedingt den Zustandübergang nach „verwitwet“.*

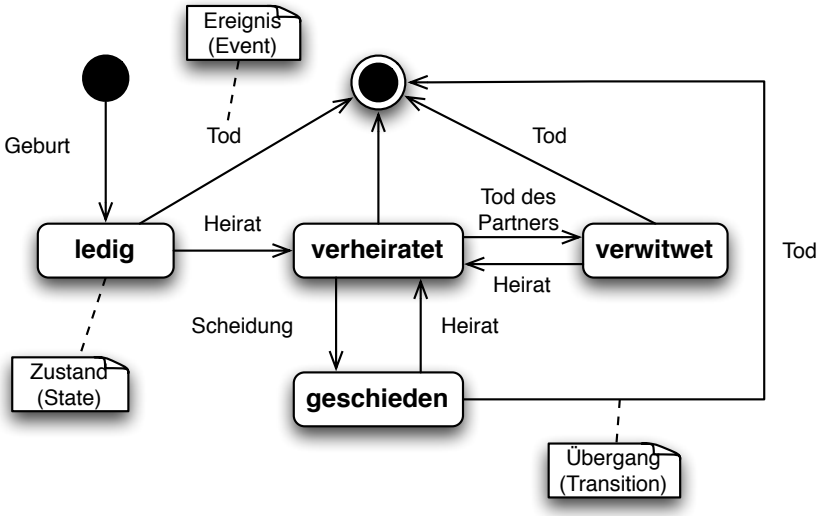


Abbildung 17: Zustandsdiagramm mit den Zuständen von Personen aus der Sicht des Standesamtes

Ein ausgefüllter, schwarzer Kreis bildet den Startzustand. Ein gefüllter Kreis mit einer weißen Umrandung symbolisiert den Endzustand. Er muss nicht in jedem Zustandsdiagramm verwendet werden. Rechtecke mit abgerundeten Ecken sind Zustände. Die allgemeine Form ist in Abbildung 18 auf der nächsten Seite zu sehen. Die Angaben entry, do und exit sind optional.


Abbildung 18: Allgemeine Form eines Zustands im Zustandsdiagramm

Die Pfeile zwischen den Zuständen werden als Zustandsübergänge bezeichnet. Die Beschriftungen entsprechen Ereignissen, die eintreten können. Im Beispiel wurde ausschließlich der Name des Ereignisses notiert. Die weiteren Formen sind in Abbildung 19 zu sehen. Die erste Zeile zeigt die allgemeine Form. Die zweite gibt die Syntax für ein Ereignis an, das nach einer festgelegten Zeit eintritt und in der letzten Zeile ist ein Ereignis dargestellt, welches an einem bestimmten Zeitpunkt eintritt.

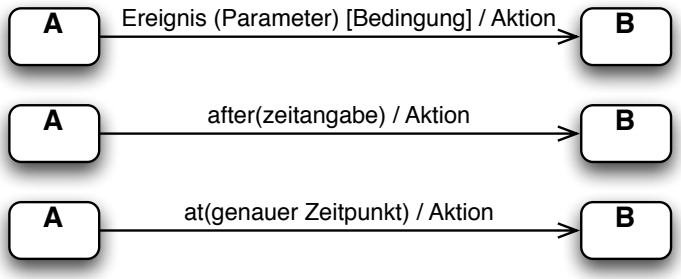


Abbildung 19: Allgemeine Form eines Ereignisses im Zustandsdiagramm

Ein Objekt befindet sich immer in genau einem Zustand. Zustandsdiagramme können auch aus mehreren geschachtelten und/oder parallelen Zuständen bestehen, was durch ein Beispiel erläutert werden soll.

***Beispiel 3.4***
*Das Diagramm aus Beispiel 3.3 wird nun als paralleler Subzustand verwendet.*

*In Abbildung 20 auf der nächsten Seite beinhaltet der äußere Zustand „lebendig“ zwei parallele Zustände. Sie werden durch eine gestrichelte Linie dargestellt. Er besitzt einen Start- und einen Endzustand. Vom Startzustand aus wird im oberen Zustand immer der Zustand „ledig“ erreicht. Es wird keine Einschränkungen getroffen, ob die Person, schlafend oder wach ist und ob sie satt oder hungrig ist. Das Ereignis „Tod“ kann in jedem beliebigen Zustand auftreten und mündet im Endzustand.*

*Der untere der beiden parallelen Zustände ist in sich geschachtelt. Eine Person ist nach der Geburt im Zustand „wach“. Sie ist dann zunächst „hungrig“ und später entweder „satt“ oder „hungrig“. Aus beiden Zuständen kann sie den Zustand „schlafend“ erreichen. Das H im Kreis bedeutet, dass genau der Zustand wieder erreicht wird, der zum Schluss im Zustand „wach“ galt[5], wenn der Zustand „wach“ wieder betreten*

[5] Dies entspricht wohl nicht immer 100% der Realität, es dient in diesem Fall aber als einfaches Beispiel.

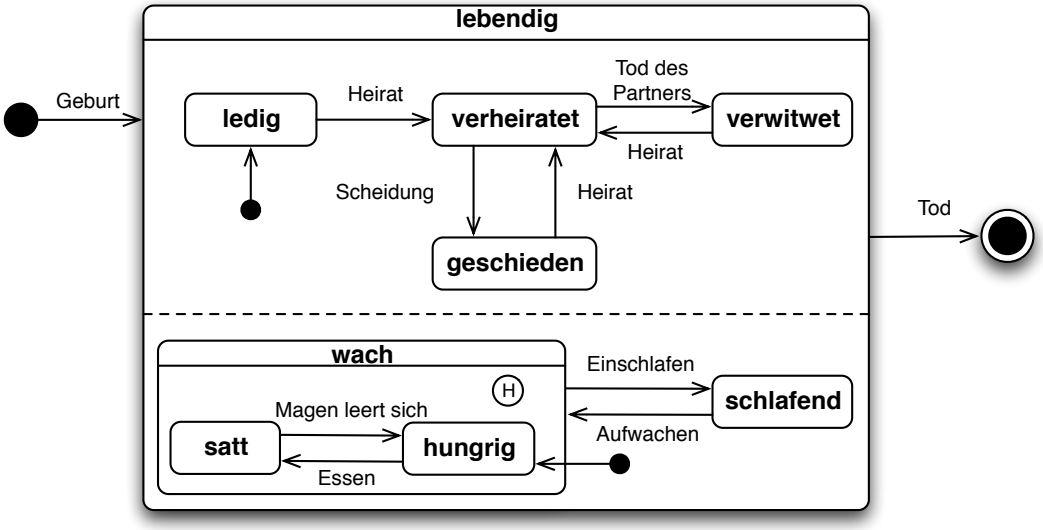


Abbildung 20: Zustandsdiagramm mit parallelen und Unterzuständen

*wird. Er wird als History-Zustand bezeichnet. In parallelen Zustandsdiagrammen befindet sich ein Objekt in jedem Teil des Zustandsdiagramm in genau einem Zustand.*

Mit Historienzuständen kann man für einen zusammengesetzten Zustand oder Unterzustandsautomaten den Zustand, der bei Betreten als Erstes aktiv werden soll, dynamisch bestimmen. Dies bedeutet, dass der erste aktive Zustand davon abhängig ist, aus welchem Zustand der zusammengesetzte Zustand zuvor verlassen wurde. Mit diesem Notationsmittel kann man die Historie der Abarbeitung bei der Auswahl des neuen aktiven Zustands nutzen (entnommen aus [7][Seite 370]).

## 3.4 Aktivitätsdiagramm

Das Aktivitätsdiagramm zeigt einen Ausschnitt eines Programmablaufs. Im Diagramm werden eine oder mehrere Aktivitäten und ihre möglichen Abläufe dargestellt. Aktivitätsdiagramme stellen im Prinzip eine Erweiterung von Flussdiagrammen dar. Eine Erweiterung ist z.B. die Möglichkeit paralleles Verhalten oder Verantwortlichkeiten darzustellen. Will man die Verantwortlichkeit für bestimmte Aktivitäten ausdrücken, kann man Aktivitätsbereiche (Partitionen)[6] einsetzen.

***Beispiel 3.5***
*In Abbildung 21 auf der nächsten Seite wird ein Teil eines Geschäftsprozesses einer Versandfirma dargestellt.*

*Zunächst wird der Auftrag angenommen und im Lager angefragt, ob das Produkt vorrätig ist. Danach folgt ein Kontrollknoten der zwischenden Fällen „vorrätig“ und „nicht vorrätig“ unterscheidet. Je nach Fall wird dann entweder das Produkt hergestellt und eingelagert oder direkt aus dem Lager entnommen*

[6] In Version 1.x der UML wurden sie als Schwimmbahnen (Swim Lanes) bezeichnet.

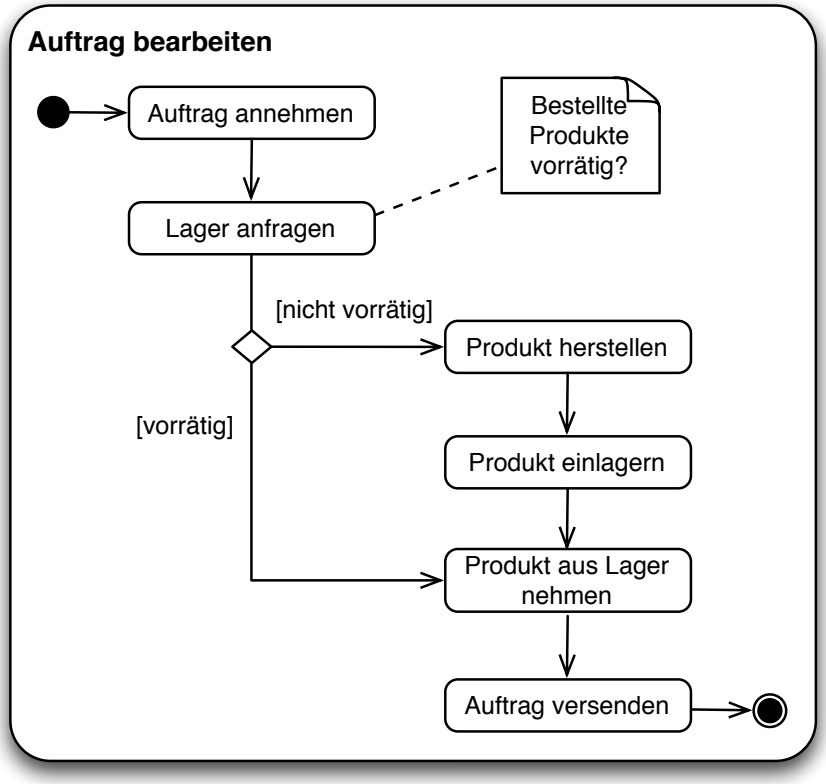


Abbildung 21: Aktivitätsdiagramm einer Versanfirma

*und versendet. In der Aktion „Produkt aus Lager nehmen“ laufen die unterschiedlichen Abläufe wieder zusammen.*

*In Abbildung 22 auf der nächsten Seite wird derselbe Sachverhalt in unterschiedliche Aktivitätsbereiche aufgeteilt dargestellt.*

*Hier wurde eine Aufteilung in drei Aktivitätsbereich (Auftragsbearbeitung, Produktion und Versand) vorgenommen. Die verwendeten Aktitonen und Kontrollknoten sind zur Abbildung 21 identisch. Diese Form der Darstellung bietet sich besonders bei größeren, leicht unübersichtlichen Sachverhalten an.*

Aktivitätsdiagramme eignen sich besser zu Darstellung komplizierterer Abläufe mit Schleifen und Verzweigungen als Sequenz- und Kommunikationsdiagramme.

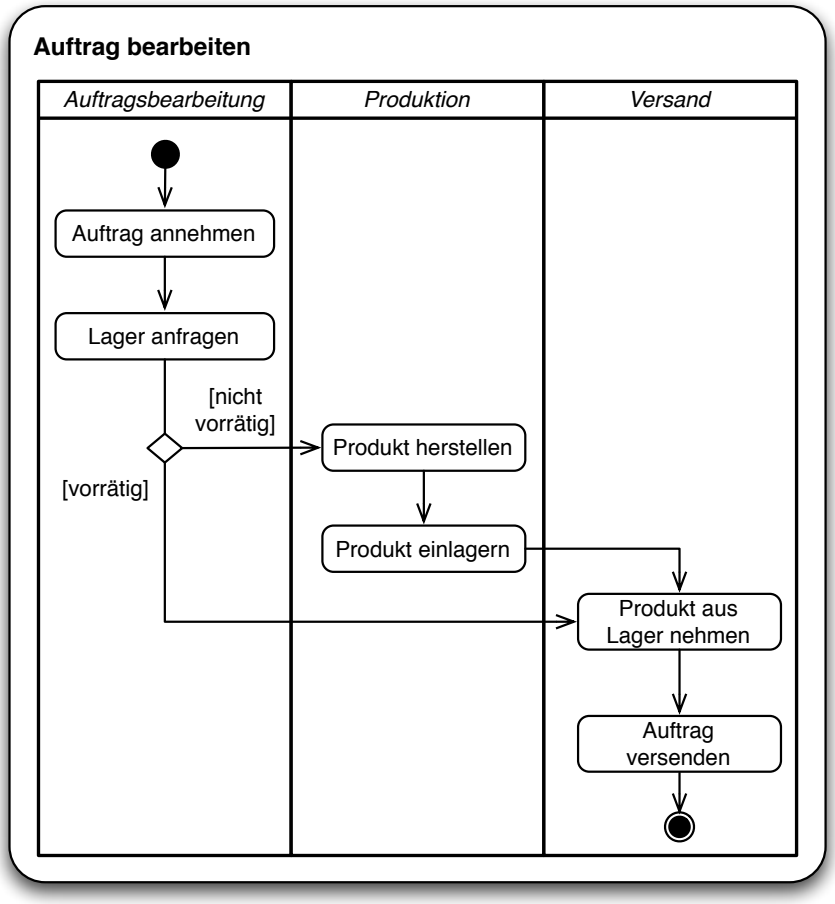


Abbildung 22: Aktivitätsdiagramm mit Aktivitätsbereichen einer Versandfirma

# 4 Literatur

Eine kurze Zusammenfassung des UML-Sprachumfangs und seiner Diagramme ist in [2] oder [6] zu finden. Sehr ausführlich – auch im Hinblick auf Detailfragen – werden die verschiedenen UML-Diagrammarten in [7] behandelt. Anhands eines großen, zusammenhängenden Projekt-Beispiels wird die UML in [8] erläutert.

# Literatur

[1] *Object Management Group*. URL `http://www.omg.org/`.

[2] Martin Fowler. *UML konzentriert*. Addison Wesley, 2004.

[3] Object Management Group. *Object Constraint Language*. Technischer Bericht, Object Management Group, 2006. URL `http://www.omg.org/spec/OCL/2.0/PDF/`.

[4] Object Management Group. *OMG Unified Modeling Language (OMG UML), Superstructure*. Technischer Bericht, Object Management Group, 2009. URL `http://www.omg.org/spec/UML/2.2/Superstructure/PDF/`.

[5] Bernd Oestereich. *Die UML 2.0 Kurzreferenz für die Praxis*. Oldenburg, 2004.

[6] Bernd Oestereich. *Analyse und Design mit UML 2.1*. Oldenbourg, 2006.

[7] Chris Rupp, Stefan Queins & Barbara Zengler. *UML 2 glasklar. Praxiswissen für die UML-Modellierung*. Hanser, 2007.

[8] Harald Störrle. *UML für Studenten*. Pearson Studium, 2005.

Autor: Bodo Igler, Nadja Krümmel, Malte Ried, Burkhardt Renz, Institut für SoftwareArchitektur.