

# Unified Modelling Language

Hier soll ein kurzer Einblick in die UML gegeben werden. Wir werden UML Diagramme im Verlauf der Veranstaltung verwenden, um Klassenabhängigkeiten darstellen zu können.

Die Ausarbeitung basiert auf [Arbeiten von Herrn Dumke](#).

## Inhalt

1. Überblick.....	4
2. Klassendiagramm .....	6
2.1. Klasse.....	6
2.1.1. Metaklasse .....	8
2.1.2. Parametrisierbare Klasse .....	8
2.1.3. Schnittstellenklasse .....	9
2.1.4. Abstrakte Klasse .....	10
2.2. Attribut.....	12
2.3. Operation, Methode .....	14
2.4. Schnittstelle .....	16

2.5. Zusicherung .....	18
2.6. Merkmal.....	19
2.7. Stereotyp.....	21
2.8. Notiz .....	22
2.9. Paket.....	24
2.10. Vererbung.....	26
2.10.1. Mehrfachvererbung .....	29
2.11. Delegation.....	30
2.12. Assoziation.....	30
2.13. Assoziationsattribute .....	32
2.13.1. Assoziationszusicherung .....	33
2.13.2. Qualifizierende Assoziation.....	33
2.13.3. Abgeleitete Assoziation.....	34
2.13.4. Mehrgliedrige Assoziationen .....	34
2.13.5. Gerichtete Assoziation .....	34
2.14. Navigationsausdruck.....	35

2.15.	Aggregation.....	37
2.16.	Komposition .....	38
2.17.	Abhängigkeit .....	40
2.18.	Verfeinerung.....	41
3.	Kollaborationsdiagramm .....	44
3.1.	Objekt .....	48
4.	Sequenzdiagramm.....	50
5.	Zustandsdiagramm.....	52
5.1.	Zustand .....	53
5.2.	Unterzustand .....	56
5.3.	Ereignis, Transition .....	57
6.	Das Komponentendiagramm .....	59
6.1.	Komponente .....	60
7.	Einsatzdiagramm.....	62
8.	Begriffe.....	63

# 1. Überblick

Die **Unified Modelling Language** (kurz UML) ist eine **Sprache** zur

- Spezifikation,
- Visualisierung,
- Konstruktion und
- Dokumentation

von Modellen für

- Softwaresysteme,
- Geschäftsmodelle und
- andere Nicht-Softwaresysteme.

Sie bietet den Entwicklern die Möglichkeit, den Entwurf und die Entwicklung von Softwaremodellen auf einheitlicher Basis zu diskutieren. Die UML wird seit 1998 als Standard angesehen. Sie lag und liegt weiterhin bei der [Object Management Group \(OMG\)](#) zur Standardisierung vor.

**Entwickelt** wurde die UML von Grady Boch, Ivar Jacobsen und Jim Rumbaugh von [RATIONAL ROSE SOFTWARE](#). Sie kombinierten die besten Ideen **objektorientierter Entwicklungsmethoden** und schufen daraus die UML.

Viele führende Unternehmen der Computerbranche (Microsoft, Oracle, Hewlett-Packard,...) wirkten aktiv an der Entwicklung mit und unterstützen die UML.

Die **UML** ist **keine Methode**. Sie ist lediglich ein **Satz von Notationen** zur Formung einer allgemeinen Sprache zur Softwareentwicklung.

Die Modellelemente der UML werden nach **Diagrammtypen** gegliedert:

- Anwendungsfalldiagramm
- **Klassendiagramm**,
- Aktivitätsdiagramm,
- Kollaborationsdiagramm,
- Sequenzdiagramm,
- Zustandsdiagramm,
- Komponentendiagramm und
- Einsatzdiagramm

Wir werden hier hauptsächlich Klassendiagramme verwenden, um die Klassenstruktur von den Beispielen zur veranschaulichen.

## 2. Klassendiagramm

### 2.1. Klasse

Verwandte Begriffe: Class, Typ, Objektfabrik

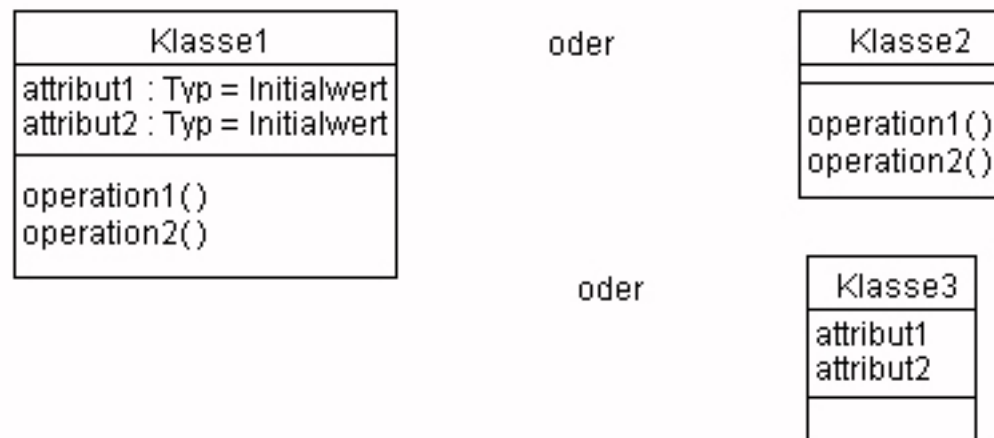
Definition:

Eine Klasse ist eine Menge von [Objekten](#), in der die [Eigenschaften](#) (Attribute), [Operationen](#) und die Semantik der Objekte definiert werden. Alle Objekte einer Klasse entsprechen dieser Festlegung.

Eine Klasse ist eine Zusammenfassung gleichartiger Objekte. Objekte sind die agierenden Grundelemente einer Anwendung. Die Gleichartigkeit bezieht sich auf Eigenschaften (Attribute) und/oder auf Fähigkeiten (Operationen/Methoden) der Objekte einer Klasse. Eine Klasse enthält gewissermaßen die Konstruktionsbeschreibung für Objekte die mit ihr erzeugt werden. Das Verhalten der Objekte wird durch die Möglichkeit eines Objektes, [Nachrichten](#) zu empfangen und zu verstehen beschrieben. Dazu benötigt das Objekt bestimmte Operationen. Die Begriffe Operation und Nachricht sollten nicht synonym verwendet werden. Zusätzlich zu Eigenschaften und Fähigkeiten kann eine Klasse auch Definitionen von [Zusicherungen](#), [Merkmalen](#) und [Stereotypen](#) enthalten.

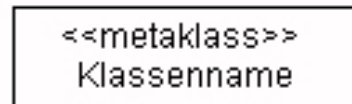
## Notation:

Klassen werden durch Rechtecke dargestellt, die den Namen der Klasse und/oder die Attribute und Operationen der Klasse enthalten. Klassenname, Attribute und Operationen werden durch eine horizontale Linie getrennt. Der Klassenname steht im Singular und beginnt mit einem Großbuchstaben. Attribute können näher beschrieben werden, z.B. durch ihren Typ, einen Initialwert und Zusicherungen. Sie werden aber mindestens mit ihrem Namen aufgeführt. Operationen können ebenfalls durch Parameter, Initialwerte, Zusicherungen usw. beschrieben werden. Auch Sie werden mindestens mit ihrem Namen aufgeführt.



### 2.1.1. Metaklasse

In einigen Programmiersprachen, z.B. in Smalltalk können auch an Klassen Nachrichten gesendet werden und sie können Klassenattribute besitzen. In C++ werden solche Klassenattribute und Klassenoperationen als *static* deklariert. Ein Beispiel für eine Klassenoperation wäre der Klassenoperator *new*, der ein Objekt einer Klasse erzeugt. Die Klassen für die Klassenobjekte werden Metaklassen genannt. Ihre Notation erfolgt ähnlich wie die einer normalen Klasse, sie werden jedoch um den Stereotyp "*metaclass*" erweitert.

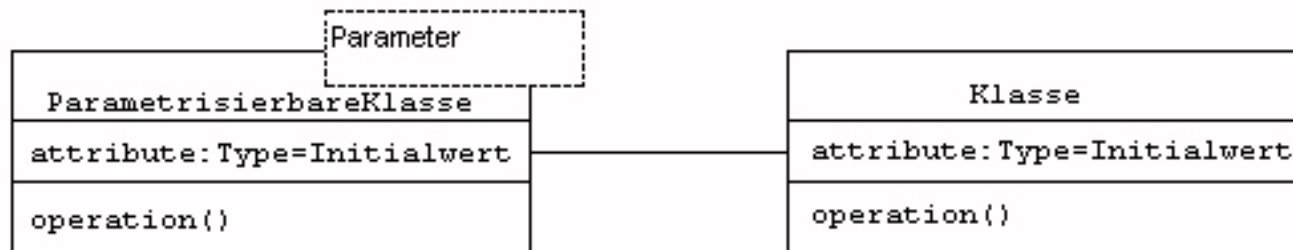


### 2.1.2. Parametrisierbare Klasse

Bei einer parametrisierbaren Klasse handelt es sich um eine Schablone zur Erzeugung von Klassen. Dabei wird eine Makrotechnik verwendet, die meistens nur mit Textersetzung arbeitet. In statisch typisierten Programmiersprachen, wie C++ oder Eiffel sind parametrisierbare Klassen ein wichtiges Hilfsmittel zur Erzeugung von wiederverwendbarem Code. Ein Anwendungsfall dafür sind Klassen, in denen Klassenobjekte abgelegt werden können (sogenannte Behälterklassen).

## Notation:

Parametrisierbare Klassen werden wie Klassen dargestellt, sie erhalten aber zusätzlich in der rechten oberen Ecke in einem gestrichelten Rechteck die notwendigen Parameter. Eine Klasse die durch eine parametrisierbare Klasse erzeugt wird muss mit einer Verfeinerungsbeziehung, die den Stereotyp *"bind"* erhält gekennzeichnet werden.



Durch Angabe konkreter Werte für die Parameter 1 und 2 kann aus der parametrisierbaren Klasse eine konkrete Klasse erzeugt werden.

### 2.1.3. Schnittstellenklasse

Schnittstellenklassen spezifizieren das externe Verhalten von Klassen und enthalten in abstrakter Form Signaturen und Beschreibungen von Operationen. Sie sind abstrakte Klassen mit dem Stereotyp *"interface"*. Klassen, die alle von einer Schnittstellenklasse geforderten Operationen bereitstellen können sind eine Umsetzung dieser Schnittstelle. Eine Klasse kann mehrere Schnittstellen anbieten.

## 2.1.4. Abstrakte Klasse

Verwandte Begriffe: Abstract class, virtuelle Klasse

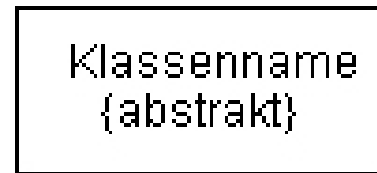
Definition:

Eine abstrakte Klasse bildet die Grundlage für weitere Unterklassen. Sie wurde bewusst unvollständig gehalten. Von ihr können **keine** konkreten [Objekt](#)-Exemplare erzeugt werden.

Eine abstrakte Klasse stellt häufig einen Allgemeinbegriff dar, einen Oberbegriff, von denen konkrete Begriffe abgeleitet werden. Als Beispiel kann der Oberbegriff geometrische Figur dienen. Von ihm kann man die konkreten Begriffe Kreis, Rechteck, Dreieck, usw. ableiten. Von jedem dieser konkreten Begriffe können Exemplare erzeugt werden, z.B. ein Kreis1 mit dem Durchmesser von 12 cm. Eine abstrakte Klasse ist also eine Oberklasse für Unterklassen.

## Notation:

Eine abstrakte Klasse wird wie eine normale Klasse dargestellt. Unter dem Klassennamen steht das Merkmal *abstrakt*. Sie kann wie eine normale Klasse [Attribute](#), [Operationen](#) und [Zusicherungen](#) enthalten.



## 2.2.Attribut

Verwandte Begriffe: Datenelement, Instanzvariable, Member

Definition:

Ein Attribut ist ein Datenelement, das in jedem [Objekt](#) einer [Klasse](#) gleichermaßen enthalten ist und von jedem Objekt mit einem individuellen Wert repräsentiert wird.

Attribute sind Informationen bzw. Daten die ein Element einer Klasse näher beschreiben. Sie werden mindestens durch ihren Namen beschrieben, können aber zusätzlich einen Initialwert und [Zusicherungen](#) besitzen. Durch Zusicherungen kann der Wertebereich bzw. die Wertemenge eines Attributes eingeschränkt werden. Mit Hilfe von [Merkmalen](#) können weitere besondere Eigenschaften von Attributen beschrieben werden, z.B. daß ein Attribut nur gelesen werden darf.

Neben normalen Attributen existieren noch so genannte abgeleitete Attribute. Diese werden durch eine Berechnungsvorschrift automatisch berechnet. Sie sind innerhalb eines Objektes nicht durch einen physischen Wert repräsentiert und sie benötigen keinen Initialwert.

Eine weitere Ausprägung von Attributen sind Klassenattribute. Sie gehören nicht zu einem einzelnen Objekt, sondern zu einer Klasse. Alle Objekte dieser Klasse können ein solches Klassenattribut benutzen.

Es gibt, je nach Programmiersprache, die Möglichkeit die Sichtbarkeit der Attribut nach außen hin einzuschränken. Dies geschieht mit Hilfe der **Sichtbarkeitskennzeichen**:

- *public*: verfügbar für alle Klassen des Systems

- *protected*: verfügbar für Objekte der eigenen und aller abgeleiteten Klassen
- *private*: verfügbar nur für Objekte dieser Klasse
- In C++ existiert zusätzlich der Friend-Mechanismus, mit dem eine Klasse ausgewählten anderen Klassen Zugriffsrechte gewähren kann.

## Notation:

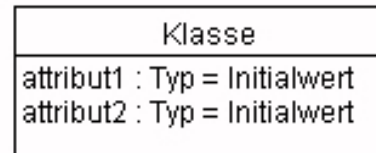
Attributnamen beginnen mit einem Kleinbuchstaben, Klassennamen mit einem Großbuchstaben, Merkmale und Zusicherungen stehen in geschweiften Klammern.

```
attribut : Klasse = Initialwert {Merkmal} {Zusicherung}
```

Abgeleitete Attribute werden mit einem vorangestellten Schrägstrich markiert. Klassenattribute werden unterstrichen und die Sichtbarkeitsangaben mit *public*, *protected* und *private* mit "+", "#" und "-" gekennzeichnet.

- /abgeleitetes Attribut
- klassenattribut
- +publicAttribut
- #protectedAttribut
- -privateAttribut

Innerhalb des Klassenrechteckes im Klassendiagramm werden die Attribute vom Klassennamen durch eine horizontale Linie getrennt und stehen somit in der zweiten Kategorie.



## 2.3.Operation, Methode

Verwandte Begriffe: Service, Prozedur, Routine, Funktion, Botschaft, Nachricht, Message

Definition:

Durch Nachrichten können [Objekte](#) miteinander kommunizieren und Operationen aufrufen. Operationen sind Dienstleistungen, die von einem Objekt aufgerufen werden können. Sie werden durch ihre Signatur (Operationsname, Parameter, Rückgabewert) beschrieben. Eine Methode ist die Implementation einer Operation. Sie besteht aus einer Folge von Anweisungen.

Eine Nachricht besteht aus dem Selektor (ein Name) und einer Liste von Parametern. Sie wird an genau einen Empfänger gesendet. Eine Operation setzt sich zusammen aus dem Namen der Operation, den Parametern (falls vorhanden) und einem evt. Rückgabewert. Die Parameter einer Operation entsprechen in ihrer Definition den [Attributen](#). Eine Operation ist innerhalb einer [Klassendefinition](#) eindeutig identifizierbar. Operationen können mit [Zusicherungen](#) ver-

sehen werden, um bestimmte Bedingungen einer Operation beim Aufruf zu sichern. Mit Hilfe von [Merkmalen](#) können bestimmte Eigenschaften einer Operation beschrieben werden. Das Merkmal *abstrakt* verdeutlicht z.B., daß es sich um eine abstrakte Operation handelt. Das Merkmal *veraltet* drückt aus, daß es sich um eine veraltete Operation handelt, die nur noch zur Kompatibilität mit früheren Versionen dient.

Abstrakte Operationen bestehen nur aus ihrer Signatur, ihre Implementation erfolgt in einer Unterklasse. Sie kommen nur in [abstrakten Klassen](#) vor. Eine abstrakte Operation die nicht in einer Unterklasse implementiert ist, ist sinnlos. Es sei denn, es soll lediglich sichergestellt werden das ein Objekt die Nachricht empfangen kann, ohne das etwas geschehen soll.

Die Begriffe Operation und Nachricht werden oft synonym verwendet, was allerdings nicht richtig ist. Objekte kommunizieren untereinander mit Hilfe von Nachrichten. Ein Objekt kann aber nur eine solche Nachricht verstehen zu der es eine entsprechende Operation gibt.

Notation:

Die Signatur einer Operation sieht wie folgt aus:

```
name (argument:Argumenttyp=Standardwert,...) : Rückgabety {Merkmal} {Zusicherung}
```

Der Name einer Operation beginnt mit einem Kleinbuchstaben. Der Name des Argumentes beginnt ebenfalls mit einem Kleinbuchstaben. Das Argument wird durch Nennung seines Typs näher beschrieben, außerdem kann ein Initialwert angegeben werden. Argumentname und Argumenttyp werden durch einen Doppelpunkt getrennt. Innerhalb des Rumpfes einer Operation erfolgt die programmiersprachenabhängige Implementation. Merkmale und Zusicherungen stehen in geschweiften Klammern. Ein

abstrakte Operation kann man durch das Merkmal `abstrakt` kennzeichnen oder aber kursiv schreiben.

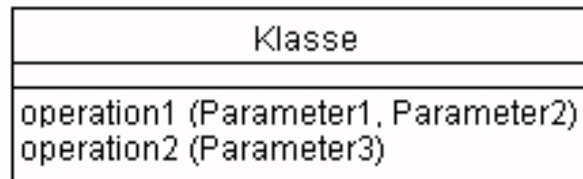
*operation()*

`operation() {abstrakt}`

Klassenoperationen werden durch Unterstreichung gekennzeichnet und die äußere Sichtbarkeit von Operationen durch voranstellen des Sichtbarkeitskennzeichens.

- `klassenoperation()`
- `+publicOperation()`
- `#protectedOperation()`
- `-privatOperation()`

Innerhalb des Klassenrechteckes werden Operationen im unteren Teil aufgeführt.



## 2.4.Schnittstelle

Verwandte Begriffe: Interface, Schnittstellenklasse

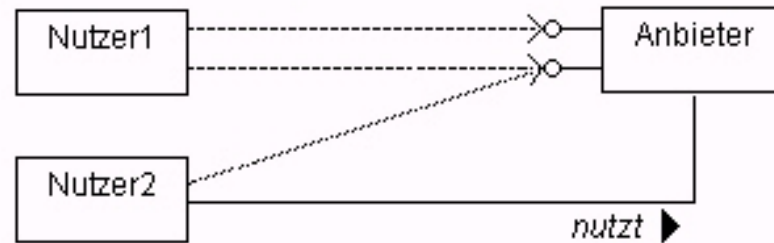
## Definition:

Schnittstellen spezifizieren einen bestimmten Teil des äußerlich sichtbaren Verhaltens von Modellelementen oder einer Menge davon.

Schnittstellen beschreiben das externe Verhalten von [Klassen](#), also speziell ausgewählte Eigenschaften. Sie beinhalten eine Menge von Signaturen für [Operationen](#). Klassen die eine Schnittstelle bereitstellen wollen, müssen solche Signaturen implementieren. Die Nutzung einer Schnittstelle setzt voraus, daß der Nutzer den Schnittstellenanbieter kennt. Die Nutzung basiert üblicherweise auf einer [Assoziationsbeziehung](#). Eine Klasse kann beliebig viele Schnittstellen und andere Eigenschaften bereitstellen.

## Notation:

Schnittstellen werden symbolisiert durch einen kleinen nicht ausgefüllten Kreis, der durch eine Linie mit der Klasse verbunden ist, die die Schnittstelle anbietet. Der Name der Schnittstelle entspricht dem Namen der Schnittstellenklasse. Er wird neben das Schnittstellensymbol gesetzt. Die Nutzung einer Schnittstelle durch andere Klassen wird durch eine Abhängigkeitsbeziehung (gestrichelter Pfeil) notiert.



## 2.5.Zusicherung

Verwandte Begriffe: Constraint, Einschränkung, Integritätsregel, Bedingung

Definition:

Eine Zusicherung ist ein Ausdruck, der mögliche Inhalte, [Zustände](#) oder die Semantik eines Modellelements einschränkt. Bei dem Ausdruck kann es sich um einen [Stereotyp](#) oder ein [Merkmal](#) handeln, aber auch um eine freie Formulierung oder um eine [Abhängigkeitsbeziehung](#).

Eine Zusicherung beschreibt eine Bedingung oder Integritätsregel. So kann sie z.B. die zulässige Wertemenge eines [Attributes](#) einschränken, strukturelle Eigenschaften zusichern, zeitliche Bedingungen stellen und andere, ähnliche Bedingungen setzen. Sie fordert oder verbietet spezielle Eigenschaften. Zusicherungen können an beliebige Modellelemente angefügt werden, z.B. an Attribute, [Operationen](#), [Klassen](#), [Assoziationen](#), usw. Sie repräsentieren zusätzliche

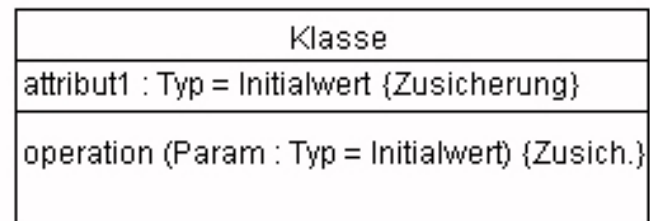
semantische Informationen zu einem Modellelement. Frei formulierte Zusicherungen dienen den Softwaredesignern als Wissensspeicher. Sie bleiben uninterpretiert, d.h. ihr Inhalt wird nicht in Code transformiert.

Notation:

Zusicherungen stehen zwischen geschweiften Klammern.

```
{ Zusicherung }
```

Zusicherungen, die eine direkte Abhängigkeit zweier Elemente definieren, werden durch eine gestrichelte Linie zwischen den entsprechenden Elementen notiert. Ist dabei ein Element vom anderen abhängig, so wird anstatt der Linie ein Pfeil in Richtung des abhängigen Elements notiert. Falls eine Zusicherung mehrere Modellelemente betrifft, oder sie nicht direkt bzw. sinnvoll zugeordnet werden kann ist es möglich, die Zusicherung innerhalb einer [Notiz](#) zu beschreiben. Von dieser führt dann eine gestrichelte Linie zu den beteiligten Modellelementen.



## 2.6.Merkmal

Verwandte Begriffe: Property String, Tagged Value, Eigenschaft, Charakteristikum

## Definition:

Merkmale erweitern die Semantik von Modellelementen um spezielle charakteristische Eigenschaften.

Merkmale fügen vorhandenen, beliebigen Modellelementen bestimmte weitere Eigenschaften hinzu. Sie detaillieren deren Semantik und beeinflussen in vielen Fällen die Code-Generierung. Es gibt sogar eigens zur Code-Generierung geschaffene Merkmale. [Abstrakte Klassen](#) und [Operationen](#) müssen auch im Code als *abstrakt* oder *virtuell* deklariert werden. Private Operationen und [Attribute](#) ebenso.

Beispiele für Merkmale sind:

- *abstrakt* - für abstrakte Klassen und Operationen
- *readonly* - für Attribute die nur gelesen werden dürfen
- *privat* - weist darauf hin, daß das Element nicht benutzt werden darf
- *veraltet* - das Element existiert nur noch zur Kompatibilität mit älteren Versionen

Es können aber auch Angaben zum Autor oder zur Versionsnummer einer Klasse als Merkmal angegeben werden. Merkmale sind somit ein mächtiges und bequemes Mittel zur Definition von semantischen Details.

## Notation:

Merkmale bestehen aus einem so genannten Schlüsselwort-Wert-Paar. Das Schlüsselwort mit dem zugehörigen Wert steht in geschweiften Klammern. Merkmale wer-

den etikettenartig an jedes Modellelement angefügt. Ist der Wert ein Boolean der true ist, kann er weggelassen werden, d.h. `{transient=true}` ist identisch mit `{transient}`.

Beispiele:

- `{abstrakt}`
- `{privat}`
- `{veraltet}`
- `{Version=3.11}`

## 2.7.Stereotyp

Verwandte Begriffe: Verwendungskontext, Zusicherung

Definition: Stereotypen erweitern die vorhandenen Modellelemente des UML-Metamodells projekt-, unternehmens- oder methodenspezifisch. Das entsprechende Modellierungselement wird direkt durch die definierte Semantik beeinflusst.

Auch [Attribute](#), [Operationen](#) und [Assoziationen](#) können mit Stereotypen klassifiziert werden. Attribute und Operationen werden dadurch innerhalb der Klasse in entsprechende Gruppen gegliedert.

Stereotypen unterscheiden die möglichen Verwendungen eines Modellelementes. Dabei werden, ähnlich wie bei der [Mehrfachvererbung](#), mehreren [Klassen](#) bestimmte gemeinsame Ei-

enschaften zugeschrieben. Der Unterschied zur Mehrfachvererbung besteht allerdings darin, daß Stereotypen keine Klassen oder Typen sind. Sie besitzen keine Typsemantik. Stereotypen ermöglichen eine mentale ggf. visuelle Unterscheidung und geben Hinweise auf die Art der Verwendung und auf den Bezug zur vorhandenen Anwendungsarchitektur. Ein Modellierungselement kann mit beliebig vielen Stereotypen klassifiziert werden. Die Semantik des Elementes wird durch die Zuordnung von Stereotypen beeinflußt. Stereotypen sollten projekt-, unternehmens- und methodenspezifisch definiert und vergeben werden.

Notation:

Ein Stereotyp wird in doppelte Winkelklammern (<< >>) geschlossen und beginnt mit einem Kleinbuchstaben. Er wird jeweils vor bzw. über dem Elementnamen platziert. Alternativ können spezielle Symbole verwendet werden. Außerdem steht es einigen Entwicklungswerkzeugen frei, farbliche oder andere visuelle Unterscheidungen zu benutzen.



## 2.8. Notiz

Verwandte Begriffe: Annotation, Kommentar

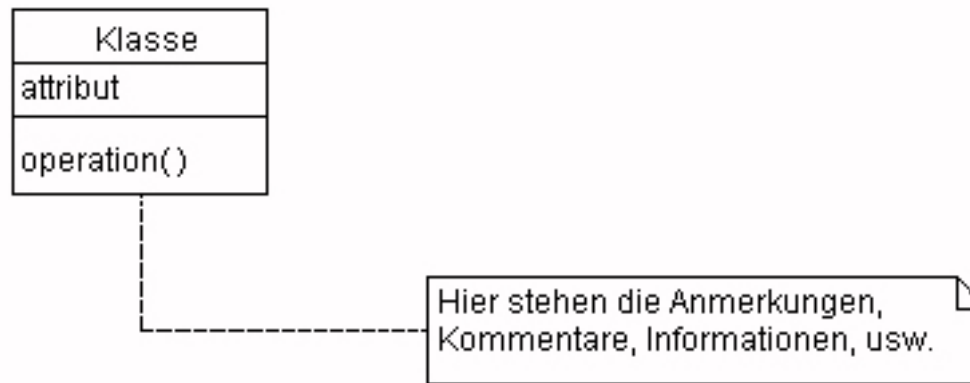
## Definition:

Notizen sind Kommentare zu einem Diagramm oder zu einem beliebigen Element in einem Diagramm, ohne semantische Bedeutung.

Notizen sind Anmerkungen zu [Klassen](#), [Attributen](#), [Operationen](#) und anderen Elementen. Einige Analyse- und Designwerkzeuge unterstützen die Möglichkeit, Notizen mit benutzerdefinierten Strukturen und Namen anzulegen. In einer Notiz können z.B. Informationen über den Entwicklungsstand, verantwortliche Entwickler eines Modellelementes, Versionsnummer einer Klasse u.ä. stehen. Möglich ist auch Daten des Projektmanagements in einer Notiz zu speichern, beispielsweise der bisherige Aufwand und der geschätzte Restaufwand für ein Projekt.

## Notation:

Notizen werden als Rechtecke mit einem Eselsohr dargestellt. Sie enthalten die zu notierenden Informationen und besitzen wahlweise eine Linie bzw. Abhängigkeitsbeziehung, die vom Rechteck zum zugehörigen Modellelement führt.



## 2.9. Paket

Verwandte Begriffe: Package, Klassenkategorie, Subsystem

Definition:

Pakete sind Ansammlungen von Modellelementen mit denen das Gesamtmodell in kleine überschaubare Einheiten untergliedert wird. Sie definieren keine Modellsemantik. Jedes Modellelement gehört zu genau einem Paket.

Pakete können verschiedene Modellelemente, z.B. [Klassen](#) und Anwendungsfälle enthalten. Sie können hierarchisch gegliedert werden, also ihrerseits wieder Pakete enthalten. Pakete werden aufgrund physischer oder logischer Zusammenhänge gebildet. Vorhandene Bibliotheken, Untersysteme und [Schnittstellen](#) bilden jeweils eigene Pakete. Wird das eigentli-

che Modell zu groß, kann es ebenfalls nach logischen Gesichtspunkten in Pakete gegliedert werden. Pakete dienen zur Modellorganisation, definieren aber keine Modellsemantik.

Eine Klasse besitzt immer ihr Stammpaket, sie kann jedoch auch in verschiedenen anderen Klassen vorkommen. Dann wird sie allerdings nur in der Schreibweise

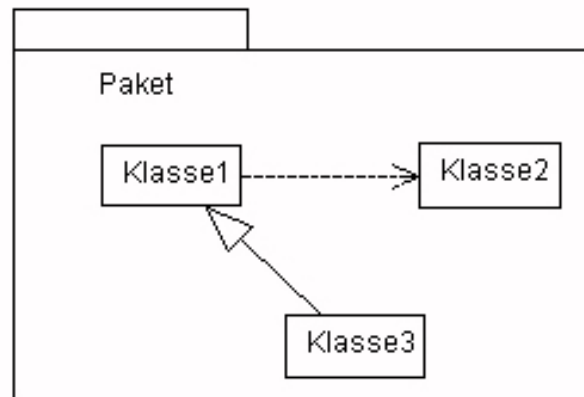
```
Paketname::Klassenname
```

zitiert. Dadurch entstehen [Abhängigkeiten](#) zwischen den Paketen, beispielsweise nutzt eine Klasse Klassen anderer Pakete. Modelliert man eine gute Architektur des Gesamtsystems, führt das zu wenigen Abhängigkeiten zwischen den Paketen.

Mit Hilfe von Paketen können während der Modellierungsphase auch sogenannte Anwendungsbausteine gebildet werden. Das sind Pakete die einen fachlichen Ausschnitt des Gesamtmodells beinhalten. Innerhalb dieses Ausschnittes sind aber alle durch die Anwendungsarchitektur vorgegebenen Bestandteile enthalten.

Notation:

Pakete werden als Aktenregister dargestellt, welches den Namen des Paketes enthält. Oberhalb des Paketnamens können Stereotypen notiert werden. Werden innerhalb des Symbols Modellelemente angezeigt, steht der Name auf der Registerlasche, andernfalls innerhalb des Rechteckes.



## 2.10. Vererbung

Verwandte Begriffe: Inheritance, Generalisierung, Spezialisierung

Definition:

Vererbung ist ein Programmiersprachenkonzept, d.h. ein Umsetzungsmechanismus für die Relation zwischen Oberklasse und Unterklasse, wodurch [Attribute](#) und [Operationen](#) der Oberklasse auch den Unterklassen zugänglich werden. Generalisierung und Spezialisierung sind Abstraktionsprinzipien zur hierarchischen Gliederung der Semantik eines Modells.

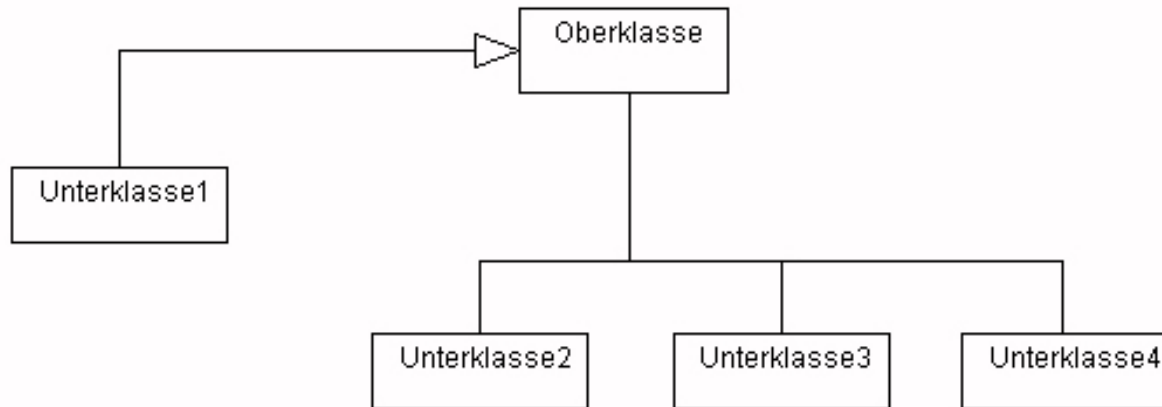
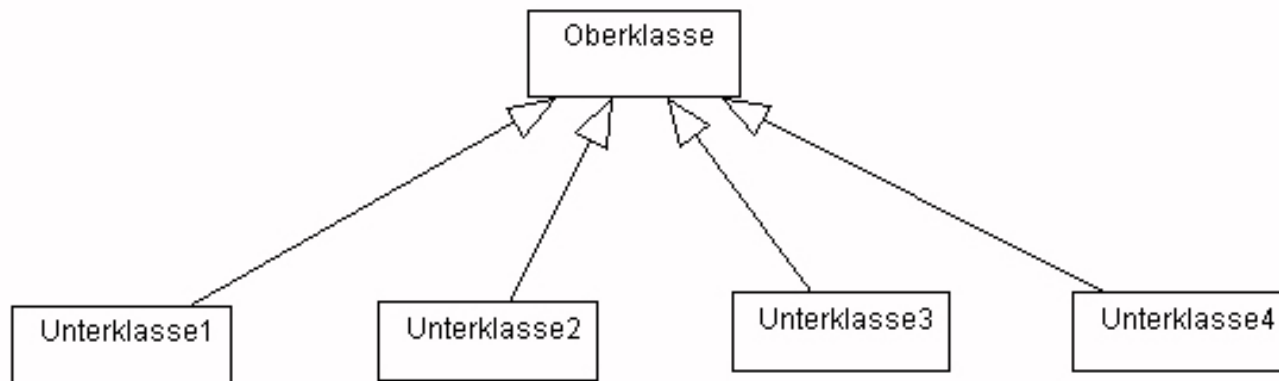
Mit Hilfe der Vererbung können [Klassen](#) hierarchisch strukturiert werden. Dabei werden Eigenschaften einer Oberklasse an die zugehörige Unterklasse weitergegeben. Bei der Generalisierung bzw. Spezialisierung werden Eigenschaften hierarchisch gegliedert, d.h. Eigenschaften mit allgemeinerer Bedeutung werden Oberklassen zugeordnet und speziellere Eigenschaften werden Unterklassen zugeordnet. Eigenschaften der Oberklasse werden dabei an die zugehö-

rige Unterklasse weitergegeben. Eine Unterklasse verfügt folglich über ihre speziellen Eigenschaften und über die Eigenschaften ihrer Oberklasse(n).

Die Unterscheidung in Ober- und Unterklasse erfolgt m.H. eines Unterscheidungsmerkmals, dem so genannten **Diskriminator**. Dieser definiert den für die Strukturierung maßgeblichen Aspekt. Er ist nicht von selbst gegeben, sondern das Ergebnis der Modellierungsentscheidung. Beispielsweise könnte man Fahrzeuge aufgrund des Diskriminators Antriebsart (Benzinmotor, Dieselmotor, Elektromotor) gliedern oder aber aufgrund des Fortbewegungsmediums (Luft, Wasser, Straße, Schiene).

Notation:

Die Vererbung wird mit einem nicht ausgefüllten Pfeil, der von der Unterklasse zur Oberklasse zeigt dargestellt. Die Pfeile von den Unterklassen können zu einer gemeinsamen Linie zusammengefasst werden oder direkt zur Oberklasse gezogen werden. Zusammengefasste Pfeile stellen Gemeinsamkeiten der Unterklasse, nämlich dass sie Generalisierungen einer Oberklasse mit gemeinsamen Diskriminator sind, stärker dar. Werden Generalisierungen mit gemeinsamen Diskriminator mit direktem Pfeil dargestellt, so sind die Pfeile mit einer gestrichelten Linie, die den Namen des Diskriminators enthält zu verbinden. Alternativ kann auch jeder einzelne Pfeil mit dem Namen des Diskriminators versehen werden. Wird die Angabe des Diskriminators weggelassen, so ist nicht mehr ersichtlich, ob es sich bei den Unterklassen um eigenständige Spezialisierungen handelt oder ob sie durch einen gemeinsamen Diskriminator entstanden sind.



Der Diskriminator ist ein virtuelles Attribut, der in der Relation zwischen Ober- und Unterklasse implizit enthalten ist. Attributwerte des impliziten Diskriminatorattributes wären die Namen der durch die Diskriminierung entstandenen Unterklassen. Die möglichen Werte des Diskriminators können durch eine Klasse mit dem [Stereotyp](#) *powerTyp*, unabhängig von der Generalisierungsbeziehung definiert werden.

## Beispiel:

Generalisierungsbeziehungen können mit folgenden vordefinierten Zusicherungen versehen werden:

- overlapping  
Eine nachfolgende Klasse kann Unterklasse von mehreren der mit overlapping gekennzeichneten Unterklassen sein.
- disjoint  
Eine nachfolgende Klasse ist stets nur Unterklasse einer der nachfolgenden Unterklassen.
- complete  
Alle Unterklassen sind spezifiziert, weitere Unterklassen werden nicht erwartet. Unabhängig davon müssen nicht immer alle Unterklassen in einem Diagramm aufgeführt werden.
- incomplete  
Weitere Unterklassen werden erwartet, sind aber noch nicht modelliert.

### **2.10.1. Mehrfachvererbung**

Bei der Mehrfachvererbung kann eine Unterklasse mehr als eine Oberklasse besitzen. Das ist durchaus kritisch zu betrachten:

- Was passiert, wenn verschiedene Oberklassen gleichnamige Eigenschaften beinhalten, die sich natürlich unterschiedlich verhalten können?

- Von welcher Oberklasse soll die Unterklasse die Eigenschaften übernehmen?
- Dieser Konflikt kann beispielsweise dadurch gelöst werden, dass die Eigenschaft vollständig adressiert, also einschließlich der Oberklassenbezeichnung angesprochen wird.

## 2.11. Delegation

Die Delegation ist ein Mechanismus, bei dem ein [Objekt](#) eine [Nachricht](#) nicht vollständig selbst interpretiert, sondern an ein anderes Objekt weiterleitet. Sie erlaubt es, vorhandene Eigenschaften anderer Klassen mitzubeneutzen bzw. zusätzlich bereitzustellen. Eine Klasse kann m.H. der Delegation ihre Eigenschaften erweitern. Effekte der Vererbung lassen sich mit dem Mittel der [Aggregation](#) nachstellen.

## 2.12. Assoziation

Verwandte Begriffe: Aggregation, Komposition, Link, Objektverbindung, Relation

Definition:

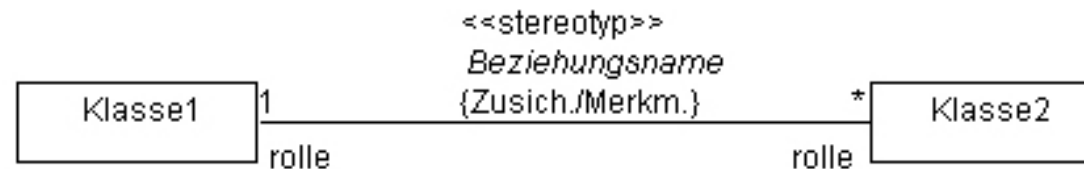
Eine Assoziation beschreibt als Relation zwischen [Klassen](#) die gemeinsame Semantik und Struktur einer Menge von Objektverbindungen. Varianten der Assoziation sind die [Aggregation](#) und die [Komposition](#). Gewöhnlich existiert eine Assoziation über den [Geschäftsvorfall](#) hinaus. Alternativ kann jedoch auch eine Assoziation modelliert werden, die nur für begrenzte Zeit gültig ist. Die Multiplizität einer Assoziation gibt an, mit wieviel Objekten der gegenüberliegenden Klasse ein Objekt assoziiert sein kann. Ist diese Zahl variabel, wird die Bandbreite angegeben, also Minimum und Maximum. Ist das Minimum gleich Null, so ist die Beziehung optional. Auf jeder Seite der Assoziation können Rollennamen vergeben werden. Diese beschreiben, welche Rolle die

jeweiligen Objekte in der Beziehung einnehmen. Außerdem können [Zusicherungen](#) verwendet werden, um die Beziehung einzuschränken.

Assoziationen beschreiben Verbindungen zwischen Klassen. Sie sind notwendig damit die [Objekte](#) miteinander kommunizieren können. Die konkrete Beziehung zwischen zwei Objekten wird Objektverbindung (engl. Link) genannt. Sie ist damit eine Instanz einer Assoziation.

Notation:

Assoziationen werden durch eine Linie zwischen den beteiligten Klassen dargestellt. Die Linie wird mit einem Namen versehen (kursiv), der beschreibt, worin und warum diese Beziehung besteht. Neben dem Beziehungsnamen kann ein kleines ausgefülltes Dreieck, dessen Spitze in Leserichtung zeigt, gezeichnet werden.



Eine Assoziation kann durch Zusicherungen, [Merkmale](#) und [Stereotypen](#) genauer beschrieben werden. Stereotypen werden vor oder über dem Beziehungsnamen notiert, Zusicherungen und Merkmale hinter oder unter dem Namen. An den Enden der Verbindungslinie kann die Multiplizität der Beziehung angegeben werden. Sie wird als einzelne Zahl oder als Wertebereich auf jeder Seite der Assoziation notiert. Der Wertebereich wird wie folgt notiert:

- Angabe des Minimums und des Maximums, getrennt durch zwei Punkte.

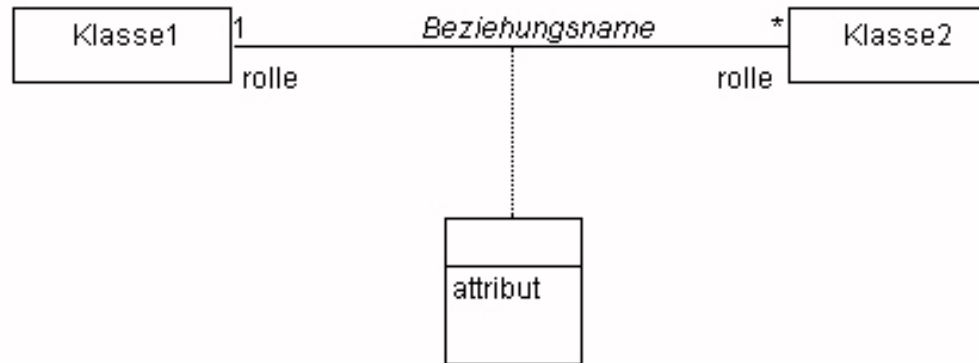
- Mit einem \* wird der Joker beschrieben, also "viele".
- Mit einem Komma können unterschiedliche Möglichkeiten der Multiplizität aufgezählt werden.

Beispiele für Multiplizitätsangaben:

- 1 genau eins
- 0, 1 null oder eins
- 0..4 zwischen null und vier
- 0..\* größer oder gleich null

### **2.13. Assoziationsattribute**

Es existiert auch eine Form in der die Assoziation selbst über [Attribute](#) verfügt. Diese Assoziationsattribute sind dann existenzabhängig von der Assoziation. Man spricht von sogenannten attributierten Assoziationen bzw. von degenerierten Assoziationsklassen, da die Klasse keine eigenständigen Objekte beschreibt.



### 2.13.1. Assoziationszusicherung

Falls eine Assoziation bestimmte Bedingungen erfüllen muß, kann sie mit einer Zusicherung, die in geschweiften Klammern neben der Assoziationslinie steht, versehen werden. Die Zusicherung kann beliebige Inhalte haben, die sowohl formal notiert als auch frei formuliert werden können.

### 2.13.2. Qualifizierende Assoziation

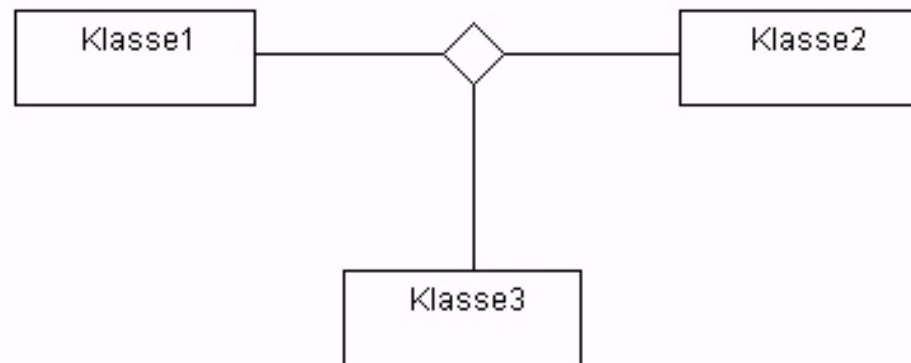
Beziehungen, bei denen ein Objekt viele (\*) Objekte der gegenüberliegenden Seite assoziieren kann, werden durch Behälterobjekte implementiert. Beispielsweise wird ein Dictionary definiert, bei dem der Zugriff jeweils durch Angabe eines Schlüssels erfolgt. Diese Schlüssel sollten schon beim Design als qualifizierende Attribute angegeben werden. Diese werden dann in der Notation als Rechteck an der Seite der Klasse dargestellt, die über diesen Schlüssel auf das Zielobjekt zugreift. Der Qualifizierer ist Bestandteil der Assoziation. Es wird bei dieser Beziehung ausschließlich über den Schlüssel navigiert.

### 2.13.3. Abgeleitete Assoziation

Eine abgeleitete Assoziation wird nicht gespeichert, sondern kann bei Bedarf berechnet bzw. abgeleitet werden.

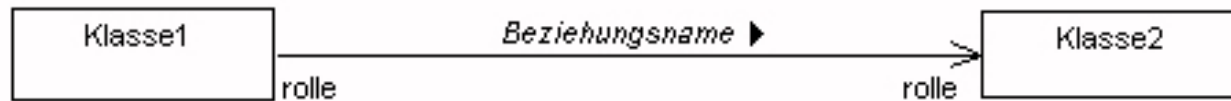
### 2.13.4. Mehrgliedrige Assoziationen

Eine weitere Form der Assoziation ist, neben der Zweierbeziehung und der attributierten Assoziation, die mehrgliedrige Assoziation. An ihr können drei oder mehr Klassen beteiligt sein. Sie sollten wegen ihrer komplexen Strukturen explizit als Klasse modelliert werden.



### 2.13.5. Gerichtete Assoziation

Gerichtete Assoziationen sind Beziehungen, die nur in eine Richtung navigierbar sind. Dargestellt wird die Navigation durch eine offene Pfeilspitze, die die zugelassene Navigationsrichtung angibt.



## 2.14. Navigationsausdruck

Definition:

Navigation ist die Betrachtung von Zugriffsmöglichkeiten auf [Objekte](#) innerhalb eines Netzes. Navigationsangaben sind Beschreibungen von Zugriffspfaden und Zugriffseinschränkungen und der daraus resultierenden Zugriffsergebnisse.

Navigationsausdrücke definieren, wie von einem Objekt zum anderen gelangt werden kann. Im Klassendiagramm definieren sie den Weg von einer [Klasse](#) zur anderen. Angaben über die Navigation können als unverbindliche [Notiz](#), als [Zusicherung](#) und als gerichtete Assoziation erfolgen.

- Eine Notiz wird dann gewählt, wenn die Navigationsangabe hauptsächlich zum besseren Verständnis oder zur Dokumentation beitragen soll.
- Eine Zusicherung wird notiert, wenn die Werte verschiedener [Attribute](#) voneinander abhängig sind und deswegen in spezieller Weise eingeschränkt werden müssen.
- Als gerichtete Assoziation wird die Navigation notiert, wenn nur eine Seite der Beziehung auf die andere zugreifen kann, aber nicht umgekehrt.

## Notation:

Klassennamen, Rollennamen oder Beziehungsnamen werden durch einen Punkt "." getrennt verkettet. Falls über eine Menge (z.B. 1:n - Beziehung) hinweg navigiert wird, werden eckige Klammern um den Ausdruck gelegt, der ein Element der Menge beschreibt.

- ".selektor  
Selektor ist der Name eines Attributes oder der Rollename der gegenüberliegenden Klasse. Das Resultat ist der Attributwert oder die Menge der gegenüberliegenden Objekte.
- "~selektor  
Liefert die Objekte der eigenen Klasse, die zu Objekten der gegenüberliegenden Klasse eine Beziehung für die durch den Selektor benannte Rolle haben (inverse Beziehung).
- "["Boolescher Ausdruck"]"  
Der Ausdruck enthält Objekte, die an dieser Stelle zugreifbar sind und verknüpft diese zu einem logischen Ausdruck. Das Resultat ist die Untermenge von Objekten, für die der Ausdruck das Ergebnis *true* liefert.
- ".selektor["Qualifizierungswert"]"  
Der Zugriff erfolgt hier über den Qualifizierer in einer qualifizierten Assoziation. Das Resultat ist das Objekt, welches durch den angegebenen Qualifizierungswert selektiert wird.

Beispiele:

- vertrag.kunde
- vertraege[summe > 1000]
- arbeitsverhaeltnis[von < today < bis]

## 2.15. Aggregation

Verwandte Begriffe: Ganzes-Teile-Beziehung, Assoziation

Definition:

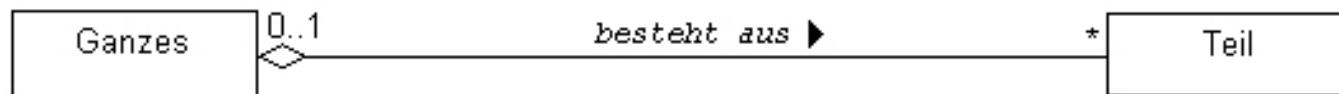
Eine Aggregation ist eine [Assoziation](#), deren beteiligte [Klassen](#) eine Ganzes-Teile-Hierarchie darstellen.

Eine Aggregation ist die Zusammensetzung eines [Objektes](#) aus einer Menge von Einzelteilen. In der Aggregation nimmt das Ganze stellvertretend für seine Teile Aufgaben wahr. Die Aggregatklasse kann [Operationen](#) enthalten, die keine unmittelbare Wirkung im Aggregat selbst erzeugen, sondern die entsprechenden Nachrichten an seine Teile weiterleiten. Die beteiligten Klassen führen keine gleichberechtigte Beziehung. Statt dessen bekommt die Aggregatklasse eine verantwortliche und führende Rolle. In einer Aggregationsbeziehung muß an einem Ende das Aggregat stehen und am anderen Ende die zugehörigen Teile. Würde auf keiner Seite ein Aggregat stehen, wäre es eine Assoziation. Steht auf beiden Seiten ein Aggregat, wäre das ein Widerspruch zur oben gemachten Definition. Ein Teil kann zu mehreren Aggregationen gehören. Es gibt auch Fälle in denen die Teile existenzabhängig vom Aggregat sind. Wenn also das Aggregat nicht mehr existiert, dann werden auch die Teile gelöscht. Wird aber ein Ein-

zerteil gelöscht, so bleibt das Aggregat erhalten. Diese Form der Aggregation nennt man [Komposition](#).

Notation:

Die Aggregation wird als Linie zwischen zwei Klassen dargestellt, und zusätzlich mit einer Raute versehen. Die Raute steht auf der Seite des Aggregats (des Ganzen) und symbolisiert das Behälterobjekt, in dem die Teile gesammelt sind. Die Kardinalitätsangabe auf der Seite des Aggregats ist häufig 1, so dass ein Fehlen der Angabe standardmäßig als 1 interpretiert wird.



## 2.16. Komposition

Verwandte Begriffe: Aggregation, Assoziation

Definition:

Eine Komposition ist eine strenge Form der [Aggregation](#), bei der die Teile vom Ganzen existenzabhängig sind.

Die Komposition ist wie die Aggregation eine Zusammensetzung eines [Objektes](#) aus einer Menge von Einzelteilen. Das Ganze nimmt stellvertretend für seine Teile alle Aufgaben wahr.

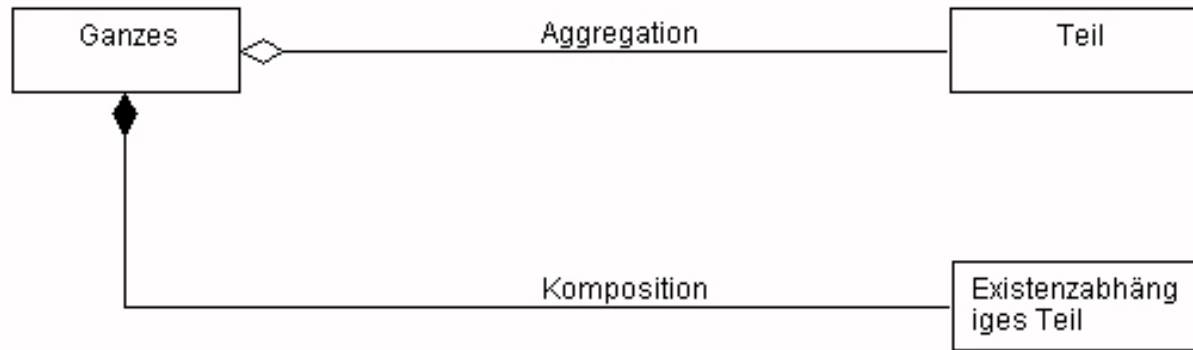
Folgende Unterschiede zur Aggregation existieren:

- Die Kardinalität auf der Aggregatseite kann nur 1 sein.
- Jedes Teil ist von nur genau einem Kompositionsobjekt abhängig.
- Die Lebenszeit der Einzelteile ist der des Ganzen untergeordnet. Sie werden also zusammen mit dem Aggregat oder später erzeugt, und sie werden vorher oder zusammen mit dem Aggregat zerstört.

Falls eine variable Multiplizität für die Teile angegeben ist können diese Teile auch nach der Entstehung des Aggregats erzeugt werden. Von diesem Erzeugungszeitpunkt an sind sie aber abhängig vom Ganzen. Sie können andersherum jederzeit vernichtet werden, jedoch spätestens mit der Vernichtung des Aggregats.

Notation:

Die Komposition wird als Linie zwischen zwei [Klassen](#) gezeichnet. Die Linie wird auf der Seite des Aggregates mit einer ausgefüllten Raute versehen. Kompositionsbeziehungen können mit einer Multiplizitätsangabe, mit einem Beziehungsnamen und mit Rollennamen notiert werden. Mehrere Kompositionsbeziehungen zu einem Ganzen können baumartig zusammengefaßt werden.



## 2.17. Abhängigkeit

Verwandte Begriffe: Dependency

Definition:

Eine Abhängigkeit ist eine Beziehung zwischen zwei Modellelementen, die zeigt, daß eine Änderung in dem einen (unabhängigen) Element eine Änderung in dem anderen (abhängigen) Element bewirkt. Die Abhängigkeit bezieht sich dabei auf die Modellelemente selbst und nicht auf eventuelle Instanzen dieser Elemente.

Beispiele:

- Eine [Klasse](#) benutzt eine [Schnittstelle](#) einer anderen Klasse. Wenn in der schnittstellen-anbietenden Klasse Schnittstelleneigenschaften geändert werden, sind ebenfalls Änderungen in der schnittstellen-nutzenden Klasse erforderlich.

- Bei voneinander abhängigen Klassen erfordern Änderungen in der unabhängigen Klasse Änderungen in der abhängigen Klasse.

Die Abhängigkeit bezieht sich direkt auf die Modellelemente, nicht etwa auf den Zeitpunkt der Programmausführung oder auf [Objekte](#). So ergibt sich auch beispielsweise eine notwendige Compilierungsreihenfolge.

Notation:

Eine Abhängigkeit wird durch einen gestrichelten Pfeil, der auf das unabhängige Element zeigt, dargestellt.



## 2.18. Verfeinerung

Verwandte Begriffe: Refinement

Definition:

Verfeinerungen sind Beziehungen zwischen gleichartigen Elementen unterschiedlichen Detaillierungsgrades.

Folgende Modellierungssachverhalte können mit Verfeinerungsbeziehungen ausgedrückt werden:

- Eine Beziehung zwischen der Analyse- und der Designversion.
- Eine Beziehung zwischen einer sauberen Implementierung und einer optimierten (evt. diffizilen) Variante.
- Eine Beziehung zwischen zwei unterschiedlich granulierten Elementen.
- Eine Beziehung zwischen einer Schnittstellenklasse und einer [Klasse](#), die diese [Schnittstelle](#) umsetzt.

Spezielle Entwurfsentscheidungen können m.H. von Verfeinerungen besser dokumentiert werden. Verfeinerungsbeziehungen können auch Abhängigkeiten zwischen einem Analysemodell und dem Designmodell dokumentieren, sofern zwischen ihnen unterschieden wird. Man sollte aber nur ganz gezielt Verfeinerungsbeziehungen dokumentieren (in Fällen, in denen das Wissen über die Verfeinerung auch wirklich wichtig erscheint), da das Projektbudget und die -dauer üblicherweise begrenzt sind.

Notation:

Die Verfeinerung wird als gestrichelter Generalisierungs-Pfeil dargestellt. Er zeigt auf die Originalvariante, also in Richtung der gröberen Einheit.



### 3. Kollaborationsdiagramm

verwandte Begriffe: collaboration diagram, Zusammenarbeit, Interaktionsdiagramm

#### Definition

Die verschiedenen Modellelemente eines Programmes agieren innerhalb des Programmablaufes miteinander. Um diese Interaktionen für einen bestimmten begrenzten Kontext, unter besonderer Beachtung der Beziehungen unter den einzelnen [Objekten](#) und ihrer Topographie, darzustellen, verwenden wir das Kollaborationsdiagramm.

Das Kollaborationsdiagramm visualisiert die einzelnen Objekte und ihre Zusammenarbeit untereinander. Dabei steht, im Vergleich zum [Sequenzdiagramm](#), der zeitliche Ablauf dieser Interaktionen im Hintergrund, vielmehr werden die für den Programmablauf und das Verständnis des selbigen wichtigen kommunikativen Aspekte zwischen den einzelnen Objekten ereignisbezogen dargestellt. Der zeitliche Verlauf der Interaktionen wird lediglich durch eine Nummerierung der Nachrichten symbolisiert. Die einzelnen Objekte können Nachrichten austauschen, ein Objekt kann sich jedoch auch stets selbst Nachrichten zusenden, ohne da eine [Assoziation](#) vorhanden sein müsste. Das Kollaborationsdiagramm kann für die Darstellung von Entwurfs-Sachverhalten benutzt werden und, in etwas detaillierter Form, von Realisierungssachverhalten. Es beinhaltet stets kontextbezogene begrenzte Projektionen des Gesamtmodells.

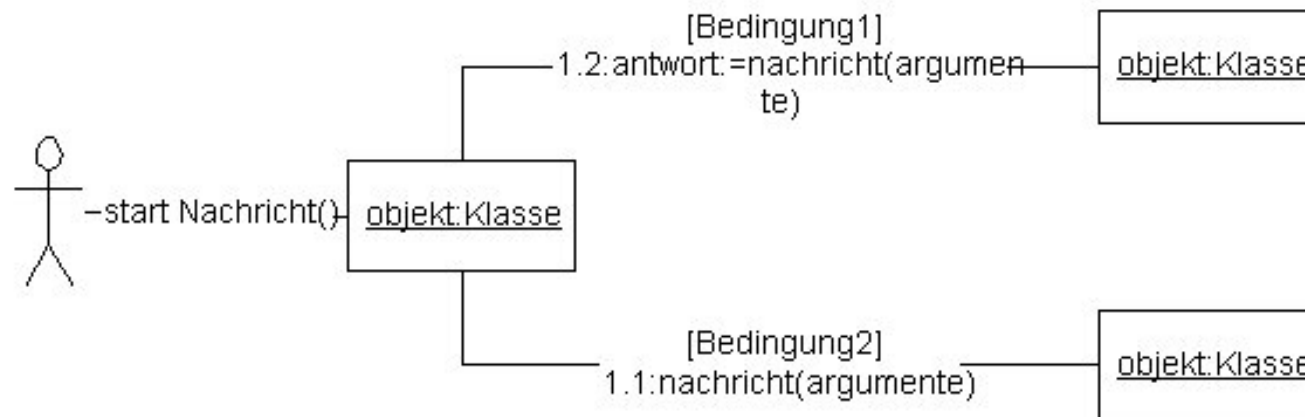
#### Notation

Die einzelnen Objekte werden als Rechtecke dargestellt. Diese werden durch Assoziationslinien mit einander verbunden, auf denen dann die Nachrichten, bestehend aus

einer durchgehenden zeitlichen Numerierung, dem Namen der Nachrichten und Antworten und ihren möglichen Argumenten.

Ein Pfeil zeigt die Richtung der Nachricht vom Sender zum Empfänger. Antworten werden in der Form *antwort:=nachricht()* symbolisiert.

Die zeitliche Numerierung erfolgt mittels Durchnummerierung der einzelnen Nachrichten angefangen bei 1, die interaktionsauslösende Startnachricht erhält noch keine Numerierung.



Somit visualisiert man eine Nachricht in folgender Form:

*[Vorgängerbedingung] Numerierung [Antwort:=] Nachrichtennamen (Argumentenliste)*

Die Vorgängerbedingung beinhaltet eine Aufzählung aller Numerierungen der Nachrichten, die bereits verschickt sein müssen, bevor die neue Nachricht versendet wird.

Die Nummern werden durch Kommas getrennt aufgeführt und mit einem "/" wird die Aufzählung abgeschlossen. Es muss nicht in jedem Fall eine Vorgängerbedingung aufgeführt werden, sie dient jedoch der Synchronisation der Nachrichtenabfolge. Die Numerierung der Nachrichten erfolgt in aufsteigender Reihenfolge.

Werden innerhalb einer [Operation](#), die eine empfangene Nachricht interpretiert, neue Nachrichten verschickt werden, so erhalten diese neuen Nachrichten eine Unternummerierung durch einen Punkt von der Hauptnumerierung getrennt (z.B. 1.1:, 1.2.1:, 1.3.3.4:). Somit kann die Tiefe der Verschachtelung einer Nachricht innerhalb anderer Nachrichten nachvollzogen werden. Die Numerierung wird durch einen Doppelpunkt abgeschlossen.

Sollen Nachrichten mehrfach versendet werden, so wird das durch ein Sternchen "\*" gekennzeichnet. Diese Iteration einer Nachricht kann optional durch Argumente in eckigen Klammern nach dem Sternchen näher beschrieben werden (z.B. 1.4.2.\* [i:=1..n]:) Iterationen von Nachrichten werden im Normalfall sequenziell vollzogen, sollen diese parallel ausgeführt werden, so wird dies nach dem Sternchen durch zwei senkrechte Linien dargestellt (z.B. 1.4.2.\* || [i:=1..n]:).

Als Iterationsargument kann auch eine Bedingung angegeben werden, bei deren Erfüllung dann die entsprechende Nachricht gesendet wird. Somit können auch weitreichendere Interaktionsstrukturen realisiert werden. (z.B. 1.4.2.\* [x<9]:)

Die Antwort auf eine Nachricht kann explizit durch einen Namen gekennzeichnet sein. Dieser kann dann in anderen Nachrichten als Argument auftauchen. Der Gültigkeitsbereich verhält sich analog zu lokalen Variablen innerhalb der sendenden Nachricht und kann auch eine solche sein.

Der Name der Nachricht ist in der Regel gleichlautend zu der, durch die Nachricht interpretierten Operation des Objektes.

Innerhalb des, in einem Kollaborationsdiagramm dargestellten Kontextes können Objekte erzeugt und zerstört werden. Objekte die erzeugt wurden, werden mit "new", Objekte die im Kontext zerstört werden, sind durch "destroyed" und Objekte die erzeugt und dann innerhalb den Kontextes auch wieder vernichtet werden sind mit "transient" zu kennzeichnen.

Die Nachrichtenabfolge zwischen den Objekten kann durch Synchronisationsbedingungen ge-

steuert werden. Diese werden durch unterschiedliche Pfeilformen dargestellt. Man unterscheidet folgende mögliche Synchronisationsbedingungen:

- *sequentiell* - das sendende Objekt kann erst dann fortfahren, wenn das empfangende Objekt die Nachricht aufgenommen und vollständig verarbeitet hat
- *synchron* - das sendende Objekt wartet nur bis das empfangende Objekt die Nachricht angenommen hat
- *eingeschränkt* - die Übermittlung einer eingeschränkten Nachricht wird abgebrochen, sollte das empfangende Objekt die Nachricht nicht sofort annehmen
- *zeitabhängig* - die Übermittlung einer zeitabhängigen Nachricht wird abgebrochen, sollte das empfangende Objekt die Nachricht nicht bis zum Ablauf einer bestimmten Zeit annehmen
- *asynchron* - die asynchrone Nachricht gerät in die Warteschlange des empfangenden Objektes, das sendende Objekt fährt fort.

Um die Beziehungen zwischen den einzelnen Objekten noch detaillierter betrachten zu können, besteht die Möglichkeit, den Verbindungslinien zwischen den Objekten zuzuordnen. Diese werden am empfangenden Objekt auf der Verbindungslinie dargestellt und können folgende Notation besitzen:

- *association* - beschreibt eine [Assoziation](#), [Aggregation](#) oder [Komposition](#) zwischen den Objekten und kann, da dies als Standardfall anzusehen ist auch weggelassen werden

- *global* - das empfangende Objekt ist global
- *local* - das empfangende Objekt ist lokal in der sendenden Operation - also "new" oder "transient"
- *parameter* - das empfangende Objekt ist ein Parameter in der sendenden Operation
- *self* - das empfangende Objekt ist das sendende Objekt

### 3.1. Objekt

verwandte Begriffe: Exemplar, Instanz

Definition

Ein Objekt ist ein aktives, konkret vorhandenes Modellelement in einem laufendem System. Es ist ein Exemplar einer [Klasse](#), gekennzeichnet durch eine Anzahl von [Attributen](#), deren Struktur durch die Klasse, der das Objekt entstammt, definiert ist, deren Werte jedoch bei jedem Objekt individuell sein können. Ein Objekt verfügt über eine endliche Anzahl von durch seine Klasse vorgegebenen [Operationen](#) und kann somit auf empfangene Nachrichten reagieren.

Eine Klasse beschreibt die abstrakte Struktur eine Menge von Objekten, deren Attribute und ausführbare Operationen. Ein Objekt kann gleichzeitig eine Instanz, ein Exemplar mehrerer Klassen sein. Man nennt dies multiple Klassifikation. Ebenfalls kann ein Objekt nacheinander mehreren Klassen angehören (dynamische Klassifikation) In der Regel ist ein Objekt jedoch einer Klasse eindeutig zugeordnet.

## Notation

Objekte werden durch Rechtecke visualisiert. Diese beinhalten den Namen des Objektes und eventuell zusätzlich den Namen der Klasse des Objektes. Teilt man das Rechteck in zwei durch eine horizontale Linie getrennte Rechtecke, können im unteren Teil auch noch die Attribute des Objektes mit aufgeführt werden.



## 4. Sequenzdiagramm

verwandte Begriffe: sequence diagram, Interaktionsdiagramm, Ereignisfadendiagramm, Szenario, Nachrichtendiagramm

### Definition

Das Sequenzdiagramm beschreibt die zeitliche Abfolge von Interaktionen zwischen einer Menge von [Objekten](#) innerhalb eines zeitlich begrenzten Kontextes.

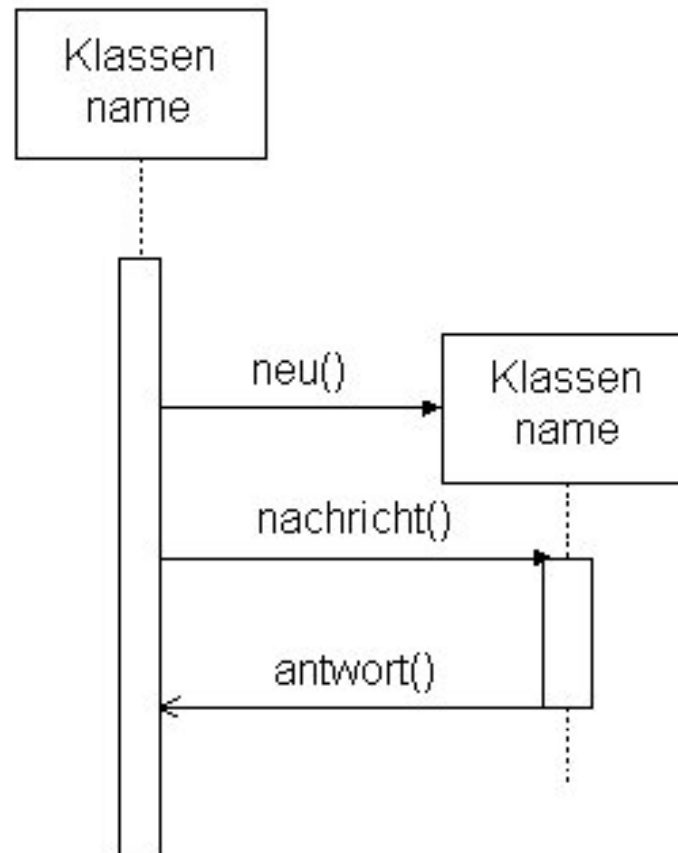
Mittels des Sequenzdiagrammes beschreibt man die Interaktionen zwischen den Modellelementen ähnlich, wie bei einem [Kollaborationsdiagramm](#), jedoch steht beim Sequenzdiagramm der zeitliche Verlauf des Nachrichtenaustausches im Vordergrund. Die Zeitlinie verläuft senkrecht von oben nach unten, die Objekte werden durch senkrechte Lebenslinien beschrieben und die gesendeten Nachrichten waagerecht entsprechend ihres zeitlichen Auftretens eingetragen.

### Notation

Die Objekte werden durch Rechtecke visualisiert. Von Ihnen aus gehen die senkrechten Lebenslinien, dargestellt durch gestrichelte Linien, ab. Die Nachrichten werden durch waagerechte Pfeile zwischen den Objektlebenslinien beschrieben. Auf diesen Pfeilen werden die Nachrichtennamen in der Form: *nachricht(argumente)* notiert. Nachrichten, die als Antworten deklariert sind erhalten die Form: *antwort:=nachricht()*

Nachrichten können Bedingungen der Form: *[bedingung] nachricht()* zugewiesen bekommen. Iterationen von Nachrichten werden durch ein Sternchen "\*" vor dem

Nachrichtennamen beschrieben. Objekte, die gerade aktiv an Interaktionen beteiligt sind, werden durch einen Balken auf ihrer Lebenslinie zu kennzeichnen. Objekte können während des zeitlichen Ablaufes des begrenzten Kontextes erzeugt und gelöscht werden. Ein Objekt wird erzeugt indem ein Pfeil mit der Aufschrift *neu()* auf ein neues Objektsymbol trifft und zerstört indem seine Lebenslinie in einem Kreuz endet.



## 5. Zustandsdiagramm

verwandte Begriffe: State Diagram, State machine, Zustandübergangsdigramm, Endlicher Automat

### Definition

Ein [Objekt](#) kann in seinem Leben verschiedenartige [Zustände](#) annehmen. Mit Hilfe des Zustandsdiagrammes visualisiert man diese, sowie Funktionen, die zu Zustandsänderungen des Objektes führen.

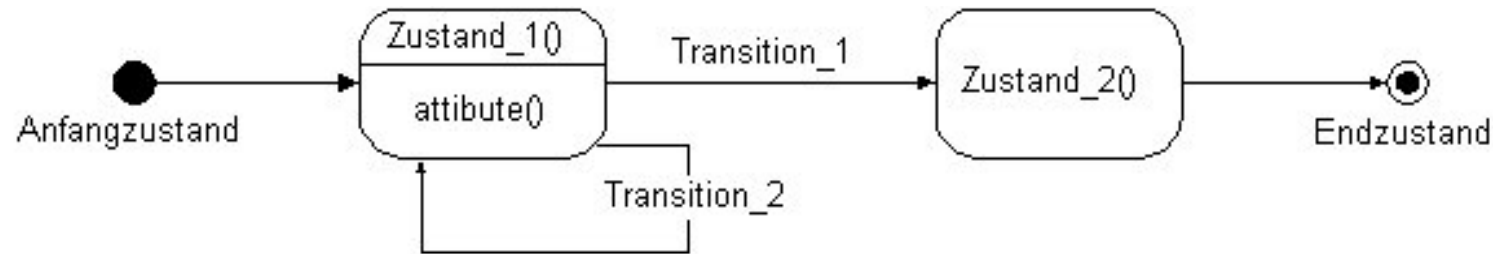
Ein Zustandsdiagramm beschreibt eine hypothetische Maschine, die sich zu jedem Zeitpunkt in einer Menge endlicher Zustände befindet. Sie besteht aus:

- einem Anfangszustand
- einer endlichen Menge von Zuständen
- einer endlichen Menge von [Ereignissen](#)
- einer endlichen Anzahl von [Transitionen](#), die den Übergang des Objektes von einem zum nächsten Zustand beschreiben
- einem oder mehreren Endzuständen

### Notation

Im Zustandsdiagramm werden die Zustände als abgerundete Rechtecke verbunden durch Pfeile, auf denen die Transitionen stehen, visualisiert. Startzustand ist ein ge-

gefüllter Kreis, die Endzustände sind leere Kreise mit einem kleineren gefüllten in der Mitte.



## 5.1.Zustand

verwandte Begriffe: State, Startzustand, Endzustand

Definition

Ein Zustand ist die endliche nicht-leere Menge von möglichen [Attribut](#)werten, die die [Objekte](#) einer [Klasse](#) annehmen können. Er gehört genau zu einer Klasse. Die Zusammenhaenge zwischen den Zuständen eines Objektes werden in einem [Zustandsdiagramm](#) visualisiert.

Ein Zustand wird als Zeitspanne zwischen zwei Ereignissen angesehen. Eine Änderung von Attributwerten eines Objektes, die das Verhalten des Objektes maßgeblich verändern heißt Zustandsänderung.

Zustände sind durch einen für sie eindeutigen Namen definiert. Gleichnamige Zustände innerhalb eines Zustandsdiagrammes beschreiben den selben Zustand eines Objektes. Zusätzlich können anonyme Zustände definiert werden, zwei namenlose Zustände in einem Diagramm beschreiben grundsätzlich verschiedene Zustände eines Objektes.

Start- und Endzustand eines Objektes sind als besondere Zustandstypen anzusehen, da zu einem Startzustand kein Übergang stattfinden und dem Endzustand eines Objektes keine Zustandsänderung folgen kann.

Ein Zustand ist durch eine endliche Menge von Zustandsvariablen gekennzeichnet. Diese ist eine Untermenge der Attribute der Klasse des Objektes, zu dem der Zustand gehört. In einem Zustand werden nur die, zur Beschreibung und eindeutigen Identifikation des Zustandes notwendigen Zustandsvariablen, aufgeführt.

Eine Klasse muss nicht notwendigerweise Zustände beeinhaltend, sie muß über entsprechendes signifikantes Verhalten verfügen. Die Modellierung von Zuständen erübrigt sich, sobald alle [Operationen](#) eines Objektes einer Klasse unabhängig von seinem inneren Zustand in beliebiger Reihenfolge durchgeführt werden können.

Ein Ereignis löst den Übergang von einem Zustand eines Objektes in den nächsten Zustand - die [Transition](#) aus. Es ist definiert durch einen Namen und einer Liste möglicher Ereignisargumente.

Ein Zustand kann Bedingungen mit dem Ereignis verbinden, die erfüllt sein müssen, um den den Folgezustand zu erreichen, bzw. um zu entscheiden, welchen Folgezustand das Objekt einnimmt.

Durch Ereignisse können Aktionen innerhalb eines Zustandes eines Objektes ausgelöst werden. Auslöser solcher Aktionen sind:

- *entry* - löst eine Aktion automatisch bei Erreichen eines Zustandes aus
- *exit* - löst eine Aktion automatisch bei Verlassen eines Zustandes aus
- *do* - löst eine Aktion automatisch aus, solange der Zustand aktiv ist, d.h. kein anderer Zustand durch das Objekt erreicht wird

## Notation

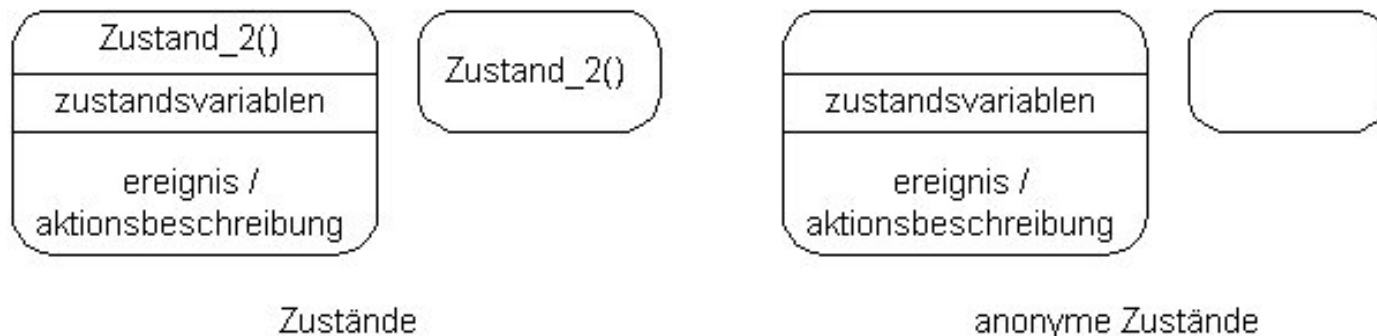
Zustände werden durch Rechtecke mit abgerundeten Ecken visualisiert. Sie können in bis zu drei Bereiche geteilt werden. Wobei der obere Bereich den Namen des Zustandes beinhalten kann. Der mittlere Bereich enthält die Zustandsvariablen. Da sie Attribute der Klasse sind, werden sie folgendermaßen notiert:

*variable : Klasse = Initialwert { Merkmal } { Zusicherung }*

Im unteren Bereich werden mögliche innere Ereignisse, Bedingungen und daraus resultierende Operationen definiert. Für sie gilt folgendes Format:

*Ereignis / Aktionsbeschreibung*

Die Aktionsbeschreibung kann ein Operationsname sein oder frei formuliert werden. Sie kann Zustandsvariablen, Attribute einer Klasse oder Parameter der eingehenden Transition enthalten.  $\ddot{y}$



Ein Startzustand wird als gefüllter Kreis visualisiert, während man einen Endzustand als nicht gefüllten Kreis, in dem sich ein kleinerer voller Kreis befindet, darstellt.



## 5.2. Unterzustand

### Definition

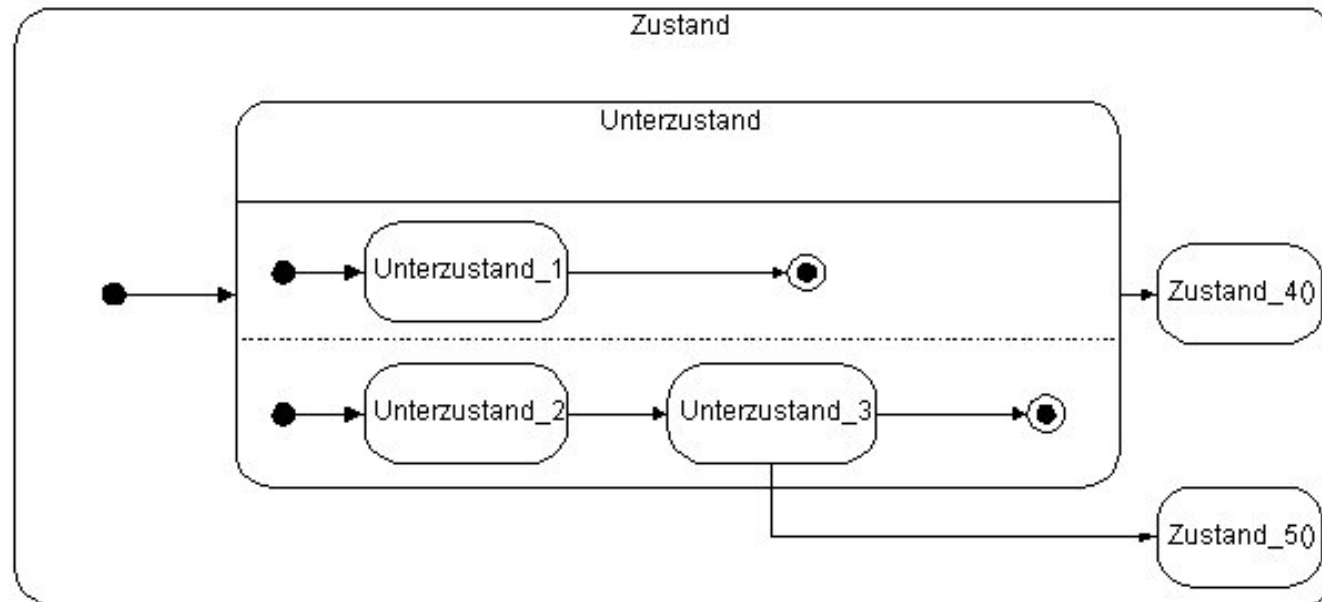
Zustände können in verschiedene sequentielle oder parallele Unterzustände aufgesplittet werden.

Ist es nötig, Ereignisse innerhalb eines Zustandes eines Objektes näher zu untersuchen, lässt sich dieser durch die Definition von Unterzuständen aufgliedern.

### Notation

Unterzustände werden in die eigentlichen Zuständen eingebettet. Die Notation von Unterzuständen ist gleich denen von Zuständen.

Parallele, konkurrierende Unterzustände werden durch gestrichelte Linien in verschiedene Abschnitte unterteilt.



### 5.3. Ereignis, Transition

verwandte Begriffe: Event, Zustandsübergang

Definition

Eine Transition ist ein Zustandsübergang eines [Objektes](#), der durch ein wesentliches Vorkommnis - ein Ereignis ausgelöst wird.

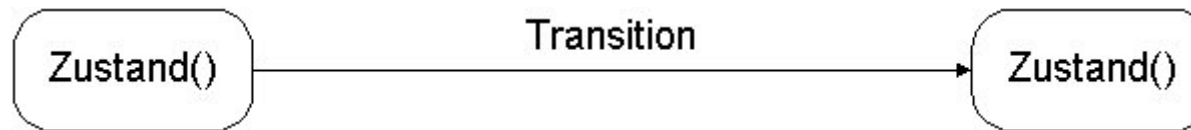
Ein Ereignis kann eintreten wenn, eine oder mehrere für eine Transition notwendige Bedingung erfüllt ist oder ein Objekt eine ereignisauslösende Nachricht erhalten hat.

Ein Objekt kann verschiedene Zustände annehmen. Um die Ursachen (Ereignisse) für Zustandsänderungen (Transitionen) zu verdeutlichen kann ein [Zustandsdiagramm](#) verwendet

werden. Hier wird dargestellt, wann ein Objekt Ereignisse erhalten darf und welche [Operationen](#) diese auslösen, bzw. welche Zustandsänderungen diese bewirken. Transitionen werden in der Regel durch, zum Teil an Bedingungen geknüpfte, Ereignisse ausgelöst. Die Bedingungen müssen dann auf jeden Fall erfüllt sein, bevor der Zustandsübergang stattfinden kann.

## Notation

Ein Zustandsübergang als Folge eines Ereignisses wird als Pfeil zwischen zwei Zuständen symbolisiert. Der Pfeil kann auch zum Ausgangszustand zurückführen. Die Transitionsbeschreibung wird in folgender Form auf dem Pfeil notiert:  
*ereignis(argumente) [bedingung] / operation(argumente)^zielobjekte.gesendetesEreignis(argumente)*



## 6. Das Komponentendiagramm

verwandte Begriffe: component diagram

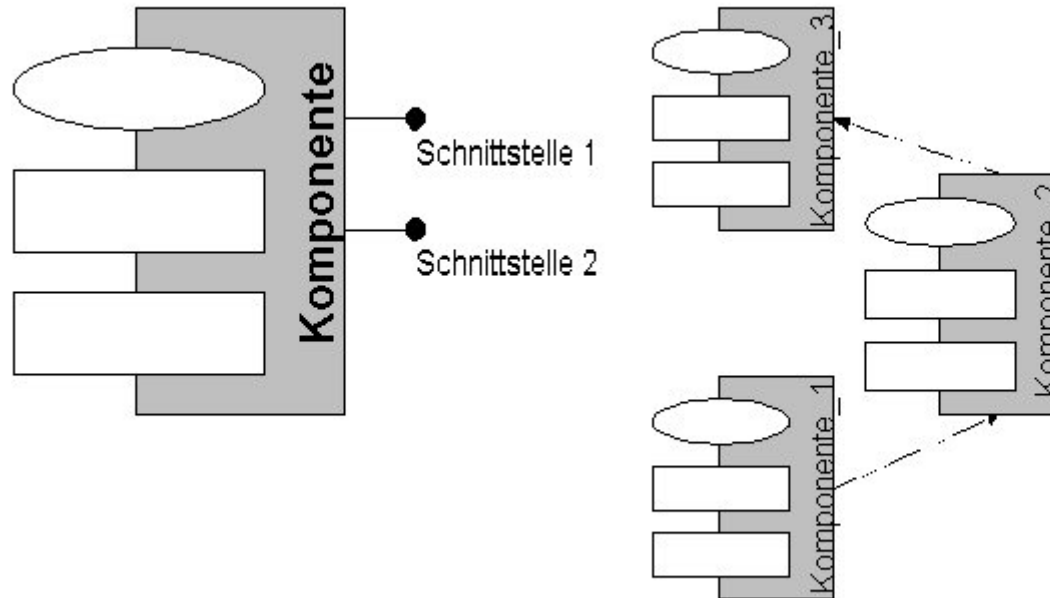
### Definition

Damit bei späterer Implementierung der Softwarelösung Compiler- und Laufzeitabhängigkeiten klar sind, werden die Zusammenhänge der einzelnen [Komponenten](#) der späteren Softwarelösung in einem Komponentendiagramm dargestellt.

### Notation

Die einzelnen Komponenten werden als Rechtecke dargestellt, die den Namen und den Typ der jeweiligen Komponente enthalten. An deren linken Rand tragen sie eine Elipse sowie 2 Rechtecke. Eine Komponente kann wiederum weitere Elemente, wie [Objekte](#), Komponenten oder Knoten enthalten und [Schnittstellen](#) besitzen.

Die Abhängigkeiten zwischen den einzelnen Komponenten werden durch gestrichelte Pfeile symbolisiert. Die in dieser Art dargestellten Abhängigkeiten zeigen die spätere Compilerreihenfolge auf.



## 6.1. Komponente

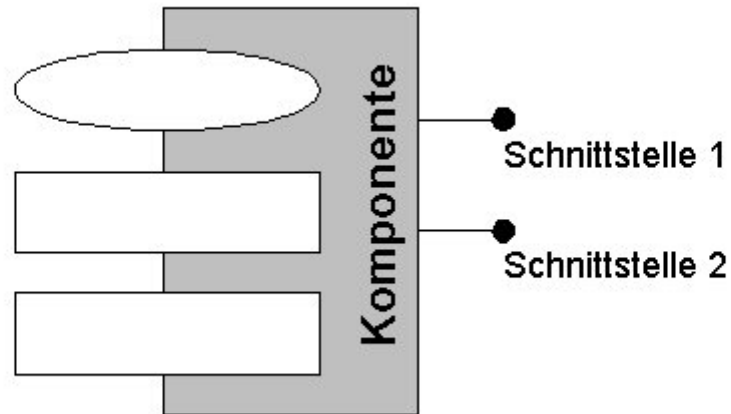
verwandte Begriffe: component, component diagram

### Definition

Eine Komponente stellt ein physisches Stück Programmcode dar, welches entweder ein Stück Quellcode, Binärcode oder ein ausführbares Teilprogramm der gesamten Softwarelösung sein kann.

### Notation

Eine Komponente wird als Rechteck dargestellt, welches den Namen und den Typ der jeweiligen Komponente enthält. An dessen linkem Rand trägt es eine Elipse sowie 2 Rechtecke. Eine Komponente kann wiederum weitere Elemente, wie [Objekte](#), Komponenten oder Knoten enthalten und [Schnittstellen](#) besitzen.



## 7. Einsatzdiagramm

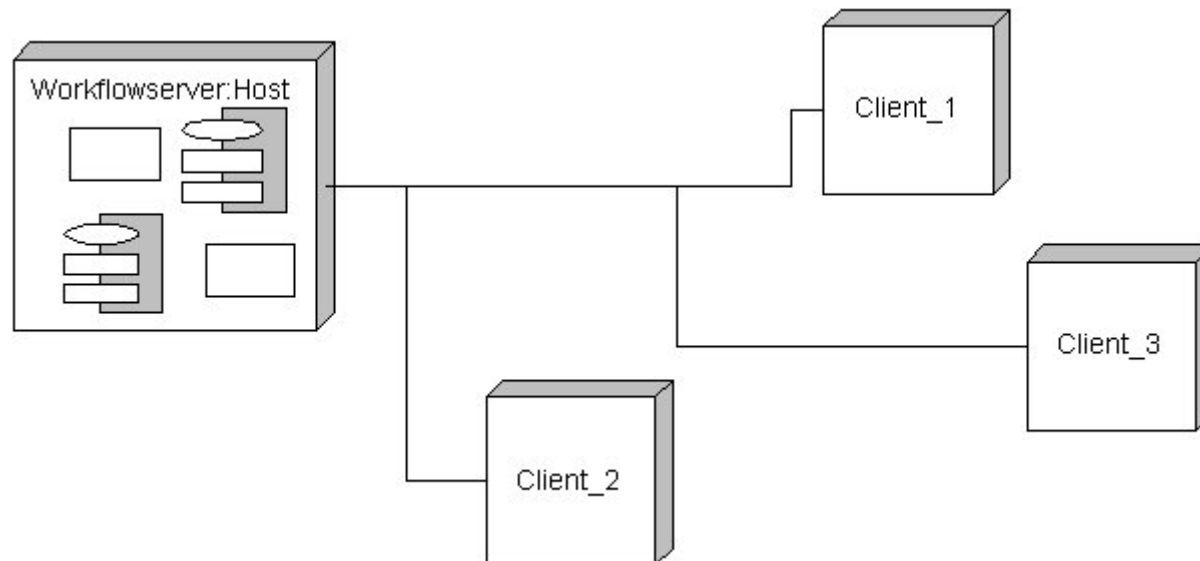
verwandte Begriffe: Knoten Diagramm, deployment diagram, Knoten

Definition

Ein Einsatzdiagramm beschreibt, welche [Komponenten \(Objekte\)](#) auf welchen Knoten ablaufen, d.h. wie diese konfiguriert sind und welche Abhängigkeiten bestehen.

Notation

Knoten werden im Einsatzdiagramm als Quader visualisiert. In den Knoten können die dort ablaufenden Komponenten(Objekte) dargestellt werden, wobei auch [Schnittstellen](#) und Abhängigkeitsbeziehungen zwischen den Elementen erlaubt sind. Knoten die miteinander kommunizieren werden durch Linien miteinander verbunden.



## 8. Begriffe

### Anwendungsfall:

Ein Anwendungsfall beschreibt die Art und Weise wie ein Akteur auf das System wirkt. Er ist eine Beschreibung für das äußerlich sichtbare Systemverhalten. Alle Anwendungsfälle zusammen ergeben demnach das systembeschreibende Modell. Anwendungsfälle sind ein Hilfsmittel zur Anforderungsanalyse, d.h. sie sollen verdeutlichen WAS das System leisten soll, aber nicht wie das System etwas leisten soll. Anwendungsfälle helfen, die Kommunikation zwischen den zukünftigen Systemnutzern und den Systementwicklern zu unterstützen bzw. zu verbessern. Ein Anwendungsfall ist eine textliche Beschreibung der Akteure, der Ereignisse und der Interaktionen zwischen Akteuren und Ereignissen (auch untereinander).

### Akteur:

Ein Akteur ist eine Person, Organisation oder wiederum ein System, das mit dem Anwendungsfallsystem kommuniziert und es beeinflusst. Der Akteur befindet sich außerhalb der Systemgrenze, d.h. er ist nicht Teil des Anwendungsfallsystems.

### Geschäftsvorfall:

Ein Geschäftsvorfall ist zum Beispiel die Vermietung eines PKW. Ein Geschäftsprozeß ("Pkw-Vermietung") beschreibt die gesamte Abfolge von Arbeitsgängen (Bestellung aufnehmen, vorhandene PKW ermitteln, ...), um diesen Prozeß fertigzustellen. Dabei können auch Teilprozesse enthalten sein, die nicht durch die zu entwickelnde Software unterstützt werden (z.B. Autoreinigung).

## Nachricht:

Eine Nachricht dient zur Kommunikation zwischen Objekten. Mit ihr werden Informationen versendet..

## Oberklasse/Unterklasse:

Die Klasse, die Eigenschaften weitergibt wird Oberklasse (bzw. Superklasse) genannt.

Die Klasse die etwas erbt heißt Unterklasse (bzw. Subklasse).

## Behälterklasse:

Behälter sind Objekte, die eine Menge anderer Objekte referenzieren und die Operationen bereitstellen, um auf diese Objekte zuzugreifen.