

Überblick

Zunächst wird programmiersprachenunabhängig in die Objektorientierung eingeführt. Dann wird gezeigt, wie Modellelemente wie Klassen und Objekte in der UML (Unified Modelling Language) dargestellt werden können.

Hier wird nur ein Überblick ohne Beispiele gegeben. Alle gezeigten Konzepte werden später ausführlich diskutiert.

Inhalt

1. Die Idee der Objektorientierung	3
1.1. Objektorientierung als Klassifizieren von Problemen.....	3
1.2. Probleme der Prozeduralen Programmierung.....	5
2. Konzepte der Objektorientierung	6
2.1. Klasse	6
2.2. Schnittstellenklasse.....	7
2.3. Abstrakte Klasse	7
2.4. Attribut.....	8

2.5. Operation, Methode.....	10
2.6. Schnittstelle	11
2.7. Merkmal	12
2.8. Notiz	13
2.9. Vererbung.....	14
2.10. Objekt	15
3. UML	16
3.1. Klassen	18
3.2. Abstrakte Klasse	19
3.3. Attribut.....	20
3.4. Operation, Methode.....	21
3.5. Notiz	23
3.6. Vererbung.....	24
3.7. Objekt.....	25

1. Die Idee der Objektorientierung

Das Konzept der Objektorientierung wird aus folgenden Blickwinkeln betrachtet:

- Modellierung von Aufgabenstellungen aus der realen Welt,
- Probleme beim Entwurf von Softwaresystemen mit prozeduraler Programmierung.

1.1. Objektorientierung als Klassifizieren von Problemen

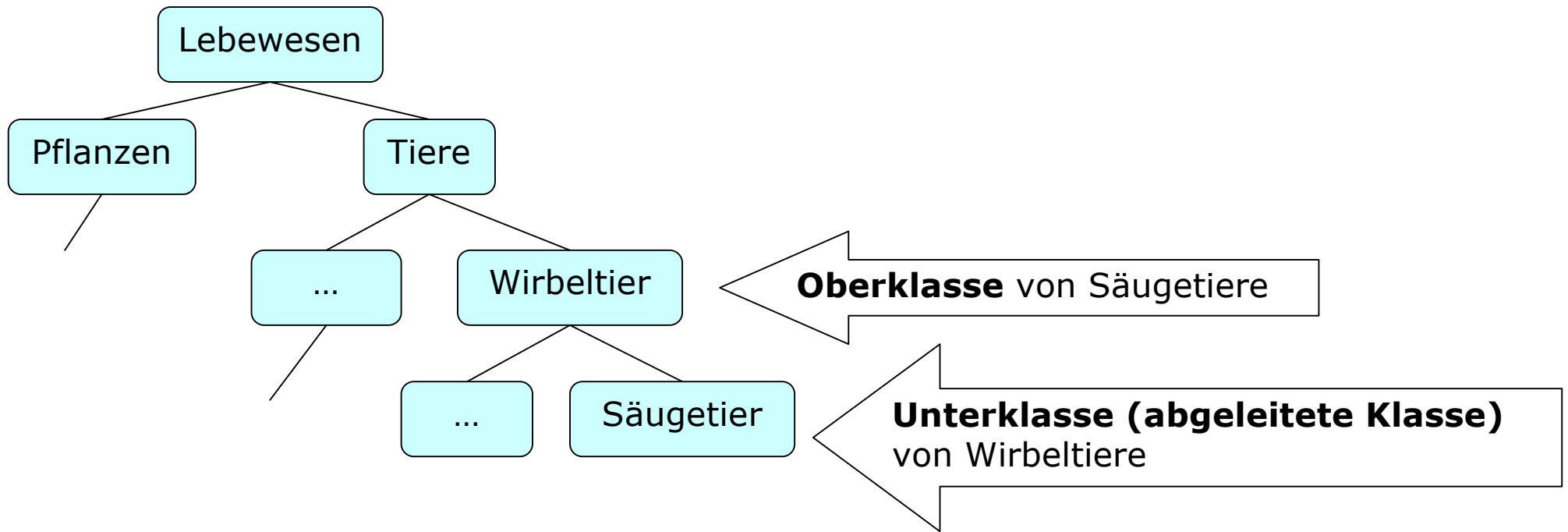
Die **Grundidee** der Objektorientierten Programmierung (kurz OOP) ist es, einen Ausschnitt der **realen Welt** in einem Programm **abzubilden**.

In allen wissenschaftlichen Disziplinen hat sich das **hierarchische Ordnungsprinzip** bewährt: **Einteilung** der **Objekte** der realen Welt in **Klassen**.

Beispiel Biologie: Einteilung der Lebewesen in die Klassen Tiere, Pflanzen, Wirbeltiere, ...

Dabei sind Gruppen von **Objekten** mit **gleichen Eigenschaften** zu einer **Klasse** zusammengefasst: gehört ein Objekt zu einer Klasse, dann hat es auch deren Eigenschaften.

Die so gebildeten Klassen sind geordnet, so sind alle Säugetiere auch Wirbeltiere.



OOP ist ein **Klassifizieren von Problemen**, weniger eine Beschreibung der Aktionen (dies ist Hauptgegenstand der Prozeduralen Programmierung).

1.2. Probleme der Prozeduralen Programmierung

Eines der Probleme der Prozeduralen Programmierung ist die Wiederverwendung und Anpassung von Code.

Software wird häufig um **neue Funktionen** erweitert, indem ein „**ähnliches**“ **Modul kopiert** und angepasst wird. Dann wird es in das System integriert.

Diese Vorgehensweise hat zwei große **Nachteile**:

- Das kopierte und modifizierte Modul muss **vollständig getestet** werden, also auch bereits getestete Teilfunktionalität (des Originals).
- Durch die Redundanzen müssen **Fehler**, die im redundanten Teil auftreten in **allen Kopien bereinigt** werden. Dies ist eine weitere Fehlerquelle und sehr aufwendig.

Die **Redundanzen** im Quellcode **erschweren** die Änderungs- und **Wartbarkeit** des so entstandenen Systems – schlimmstenfalls ist das System nicht mehr wartbar.

Durch OOP soll dies verhindert werden, da **Gemeinsamkeiten in Oberklassen** zusammengefasst werden und die Erweiterungen in abgeleiteten Klassen behandelt werden.

2. Konzepte der Objektorientierung

Zunächst werden die wichtigsten Begriffe der OOP erklärt.

2.1. Klasse

Verwandte Begriffe: Class, Typ

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. Zusätzlich zu Eigenschaften und Fähigkeiten kann eine Klasse auch Definitionen von [Zusicherungen](#), und [Merkmalen](#) enthalten.

2.2. 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.

2.3. 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.

2.4. 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. 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.

2.5.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. 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.

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 dass 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.

2.6.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.

2.7.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

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.

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

2.10. 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.

3. UML

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.

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.

3.1. Klassen

Klassen werden durch Rechtecke dargestellt, die den Namen der Klasse und/oder die Attribute und Operationen der Klasse enthalten. Klassename, Attribute und Operationen werden durch eine horizontale Linie getrennt. Der Klassename steht im Singular und beginnt mit einem Großbuchstaben. Attribute können näher beschrieben werden, z.B. durch ihren Typ, einen Initialwert. 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.

Beispiel:

Klasse1
attribut1: Typ1 = Initialwert1 attribut2: Typ2 = Initialwert2
operation1() operation2()

3.2. Abstrakte Klasse

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.

Beispiel:



3.3.Attribut

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

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

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

Beispiel:

Konto
<u>-saldo: float</u>

3.4. Operation, Methode

Die Signatur einer Operation sieht wie folgt aus:

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

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 Sichtbarkeitskennzeichen.

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

Innerhalb des Klassenrechteckes werden Operationen im unteren Teil aufgeführt.

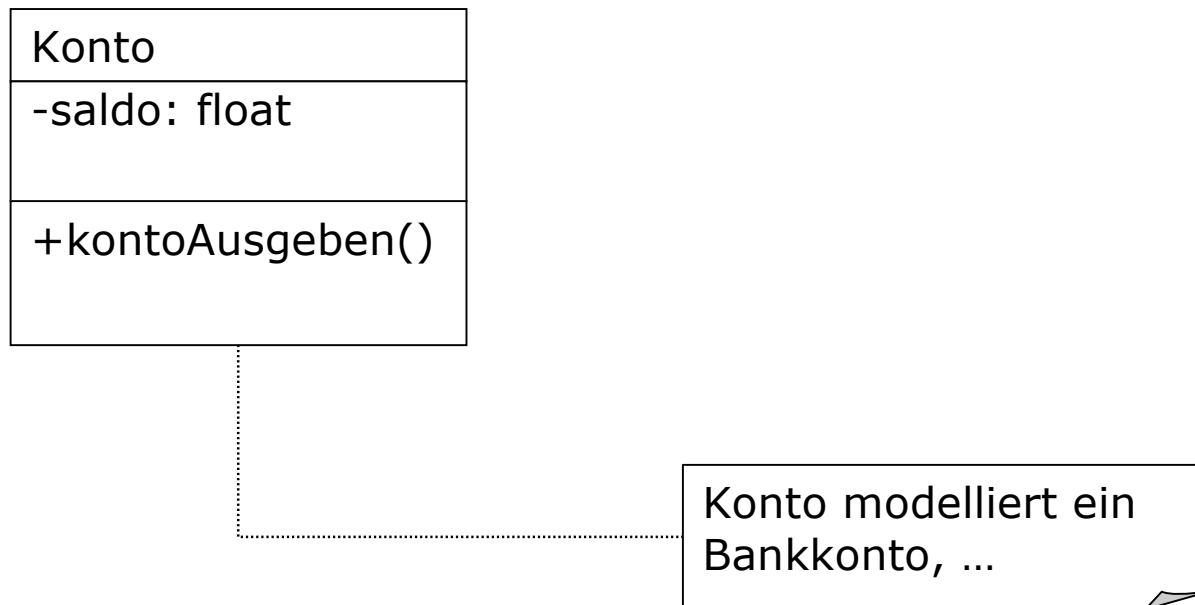
Beispiel:

Konto
-saldo: float
+kontoAusgeben()

3.5. Notiz

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.

Beispiel:



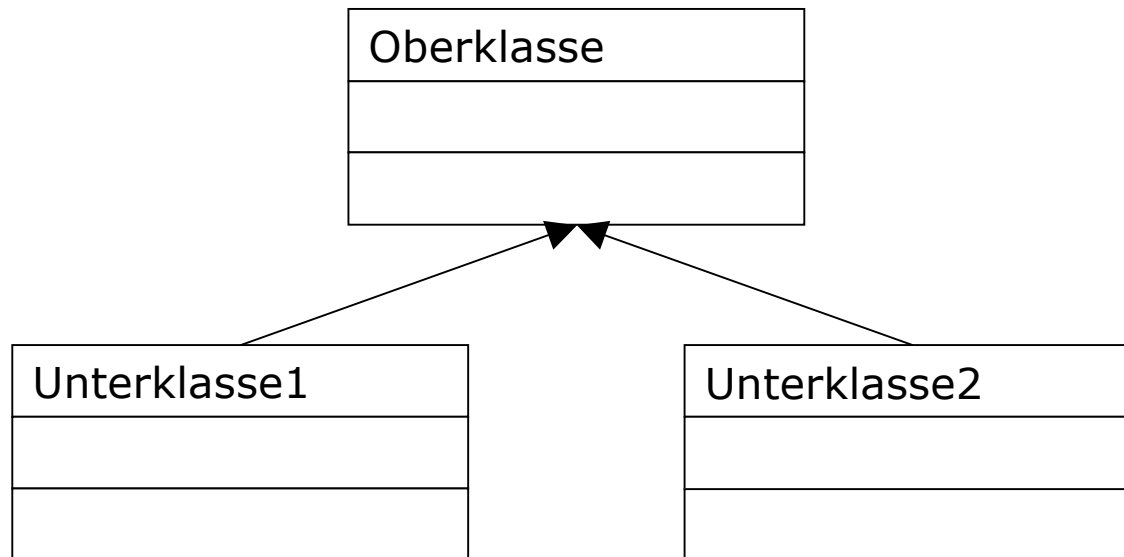
3.6. Vererbung

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.

Bemerkung:

Wegen der eingeschränkten Möglichkeit von Word werden hier ausgefüllte Pfeile verwendet!

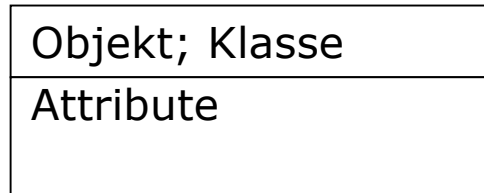
Beispiel:



3.7. Objekt

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.

Beispiel:



Hörsaalübung:

Entwickeln Sie ein UML Klassendiagramm für Bank-Konten:

- Ein Konto hat eine Kontonummer, eine Währung und einen Saldo.
- Auf Konten können Einzahlungen und Auszahlungen durchgeführt werden, der aktuelle Kontostand kann abgefragt werden.
- Ein Festgeldkonto ist ein Konto, bei dem ein Geldbetrag eine feste Zeit gebunden ist und verzinst wird. Das Festgeldkonto kann abgerechnet werden, d.h. der Zinsbetrag wird dem Konto gutgeschrieben.
- Ein Girokonto ist ein Konto ohne Verzinsung aber mit einer Überziehungslinie. Die Überziehungslinie kann angezeigt und verändert werden.