	UML – Klassendiagramm		AnPr
	Name	Klasse	Datum

1 Allgemeines

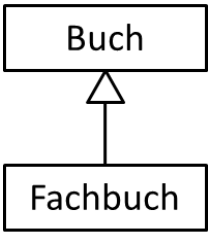
Das in der Praxis wohl am meisten genutzte UML Diagramm ist das Klassendiagramm. Hier werden die Beziehungen einzelner Klassen (und ggf. Methoden und Eigenschaften) übersichtlich dargestellt. Mitunter werden in Entwicklungsprozessen auch mit Hilfe von Tools Code für die Klassen generiert.

2 Beziehungen

Grundsätzlich unterscheiden wir zwei Grundtypen von Beziehungen zwischen Klassen. Einmal die bereits bekannte Vererbung von Eigenschaften und Methoden. Daneben gibt es noch die sogenannte Assoziation, welche angibt, ob ein Objekt eine Referenz zu einem anderen Objekt hat (oder haben kann).

2.1 Vererbung

Bei einer Vererbung wird angezeigt, dass eine (Kind-) Klasse die Eigenschaften, Methoden und auch Assoziazionen einer (Eltern-) Klasse erbt. Der Pfeil zeigt hierbei von der Kindklasse zur Elternklasse. Man spricht hier auch von „Erweiterung", die Kindklasse „erweitert" hierbei die Elternklasse, was den Programmierbegriff „extends" auch erklärt. Ein weiterer Begriff ist „Spezialisierung" – die Kindklasse spezialisiert somit die Elternklasse. Andersherum generalisiert die Elternklasse die Kindklasse, was als „Generalisierung" bezeichnet wird. Die Generalisierung ist somit die Umkehrung der Spezialisierung.



Hinweis: Üblicherweise stellt man die Elternklasse im Diagramm über der Kindklasse dar.

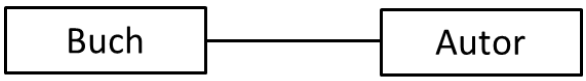
Der Javacode zu dieser Vererbung würde wie folgt lauten:

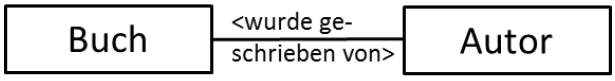
```
class Fachbuch extends Buch {
}
```

Von Vererbung spricht man auch, wenn von einer „abstrakten Klasse" geerbt wird. Abstrakt bedeutet, dass die Klasse nicht instanziiert werden kann – sie also lediglich eine Sammlung von Eigenschaften, Methoden und Assoziationen darstellt, welche spezialisiert werden müssen. Abstrakte Klassen (und Methoden) werden in UML üblicherweise *kursiv* dargestellt.
Weiterhin gibt es Programmiersprachen, welche „Mehrfachvererbungen" erlauben, bei denen eine Kindklasse also mehrere Elternklassen haben kann. Dies wird entsprechend durch mehrere Pfeile symbolisiert.

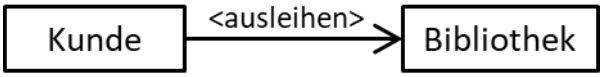
2.2 Assoziation

Mit Hilfe von Assoziationen zeigen wir an, ob zwischen den Elementen eine referenzielle Beziehung besteht. Im Beispiel sehen wir, dass der Autor und das Buch in einer Beziehung stehen. Dies erfolgt durch eine durchgezogene Linie zwischen den Klassen. Wir können also davon ausgehen, dass ein Buch eine Referenz auf einen Autor hat, die Klasse „Buch" also eine Variable beinhaltet, welche der Klasse „Autor" angehört. Um dies noch deutlicher zu machen, kann auf den Assoziationsline noch ein erklärender Begriff ergänzt werden:





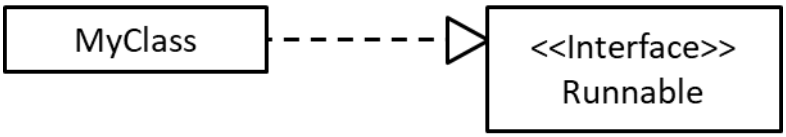
Alternativ kann eine Assoziation auch einen Aufruf modellieren. Hierfür wird in der Regel eine „gerichtete Assoziation" dargestellt, was ein Pfeil mit einer offenen Spitze ist. Nun ist aber der Aufruf nur möglich, wenn auch eine Referenz existiert. Der entscheidende Unterschied zwischen einer reinen Assoziation und einer gerichteten ist also, dass eine Richtung vorgegeben wird, welche bei einem Aufruf ja ebenfalls gegeben wird (Klasse 1 ruft Methode in Klasse 2 auf).



Um die Beziehungen nun noch näher zu spezifizieren, können zusätzliche Informationen hinterlegt werden. Dies ist die Information über die Kardinalität und die Frage, inwieweit das eine Element zwingend ein Teil des anderen Elements ist.

2.2.1 Schnittstellenimplementierung

Eine Schnittstelle (Interface) kann von einer Klasse implementiert werden dies modelliert man ebenfalls mit einem Pfeil:



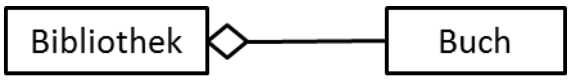
2.3 Kardinalität (Multiplizität)

Um die Kardinalität festzulegen, werden an jedem Ende der Assoziazion die Kennungen hierfür eingetragen:

	Genau eine Beziehung, das Objekt der Klasse A muss exakt einmal vorkommen -> Muss Beziehung
	Viele Beziehungen, das Objekt der Klasse A kann beliebig oft vorkommen -> Kann Beziehung
	0 oder eine Beziehung, das Objekt der Klasse A kann einmal vorkommen -> Kann Beziehung
	Eine oder mehrere Beziehungen, das Objekt der Klasse A muss mindestens einmal vorkommen -> Muss Beziehung

2.4 Aggregation

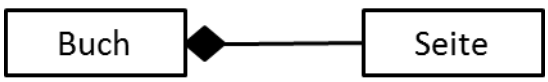
Mit Hilfe einer Aggregation wird dargestellt, dass ein Element Teil eines Ganzen ist, wobei das Ganze ohne dem Teilelement sinnvoll existieren kann. Die Aggregation ist ein Sonderfall der Assoziation.



Merke: Immer dann, wenn nach der Instanziierung des Ganzen nachträglich noch Teilelemente sinnvoll hinzugefügt werden können, handelt es sich um eine Aggregation. Das Ganze aggregiert somit die Einzelelemente. Die Lebensdauer der einzelnen Objekte (der Bücher) und dem Aggregat (der Bibliothek) sind voneinander unabhängig.
Oder anders ausgedrückt – wenn der Sinn der Bibliothek die Sammlung von Büchern ist, welche auch ohne die Bibliothek existieren können, handelt es sich um eine Aggregation.

2.5 Komposition

Die Komposition ist wiederum ein Sonderfall der Aggregation. Es wird dargestellt, dass ein Element ein zwingender Teil eines Ganzen ist; das Ganze also ohne dem Teilelement nicht sinnvoll existieren kann.

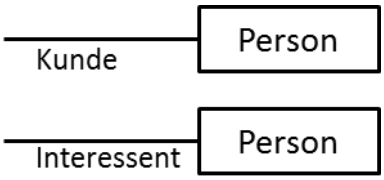


Merke: Immer dann, wenn nach der Instanziierung des Ganzen nachträglich kein Teilelement mehr sinnvoll hinzugefügt werden kann, handelt es sich um eine Komposition. Das Ganze wird also aus den Einzelelementen komponiert. Die Lebensdauer der Objekte (Seiten) hängt immer vom Kompositum (Buch) ab, sie ist in der Regel gleich oder kleiner. Wenn das Buch zerstört wird, sind die Seiten ebenfalls weg.

Oder anders ausgedrückt – wenn das Buch aus den Seiten besteht und durch die Existenz der Seiten bestimmt ist, handelt es sich um eine Komposition.

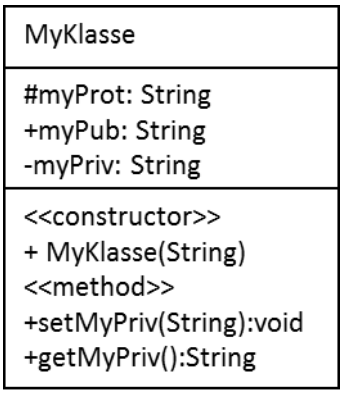
2.6 Rollen

Um spezielle Ausprägungen einer Klasse näher zu bestimmen, können auch Rollen angegeben werden.



3 Klassenaufbau

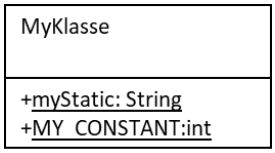
Die Klassenstruktur in Bezug auf Attribute und Methoden kann im Diagramm genauer spezifiziert werden. Hierbei werden die wesentlichen Elemente abgelegt.
Oben steht der (großgeschriebene) Klassenname. Optional kann vor dem Klassennamen das Package, gefolgt von zwei Doppelpunkten eingetragen werden (also: mypackage::MyKlasse).
Danach folgen die Attribute inklusive Sichtbarkeit, gefolgt vom Datentyp. Im Unteren Block stehen die Methoden – im Regelfall zuerst die Konstruktoren und danach die einzelnen Methoden. Jede Methode gibt die notwendigen Parameter und den Rückgabewert an (wobei dieser nicht immer in allen Diagrammen angegeben wird).



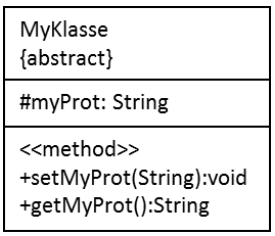
Bei der Sichtbarkeit wird wie folgt unterschieden:

public	+
protected	#
private	-
package	~

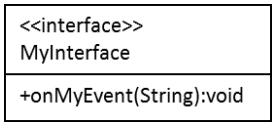
Statische Variablen werden im Regelfall unterstrichen gekennzeichnet. Konstanten (welche sinnvollerweise auch statisch sind) entsprechend der üblichen Namenskonventionen in Großbuchstaben.



Anmerkung1: Weiterhin gibt uns UML die Möglichkeit, abstrakte Klassen zu kennzeichnen, indem in geschweiften Klammern „abstract" hinter oder unter dem Klassennamen geschrieben wird. In gedruckten UML Diagrammen werden abstrakte Klassen und Methoden kursiv dargestellt.



Anmerkung2: Interfaces werden wie Klassen dokumentiert (natürlich ohne Attribute und Konstruktoren), wobei hier <<interface>> im Kopf eingetragen wird:



4 Lizenz