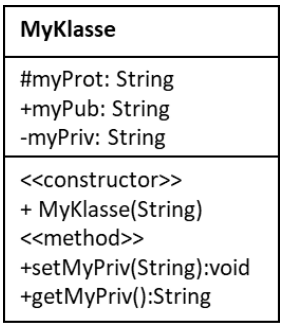
	UML – Klassendiagramm		AnPr	V 1.2
	Name	Klasse	Datum	

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 Klassenaufbau

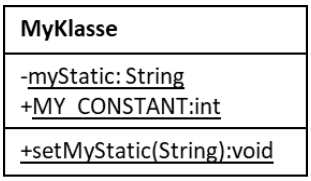
Die **Klasse** wird als Rechteck notiert. Oben steht der (großgeschriebene) Klassenname. Optional kann vor dem Klassennamen das Package, gefolgt von zwei Doppelpunkten eingetragen werden (also: `mypackage::MyKlasse`).
Danach folgen die **Attribute** inklusive Sichtbarkeit (**Zugriffsspezifizierer**), 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). Die Stereotypangabe `<<constructor>>` und `<<method>>` sind optional, da sie aus dem Kontext ersichtlich sind (sprich Konstruktoren heißen wie die Klasse).



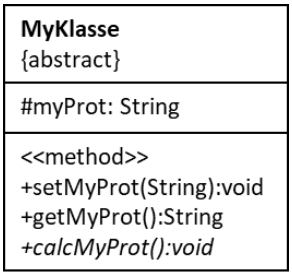
Folgende **Zugriffsspezifizierer** werden unterschieden, wobei package einen Spezialfall darstellt und nicht von allen Prachen unterstützt wird:

Name:	Symbol:	Bedeutung
public	+	Von außen sichtbar („alle“)
protected	#	Vererbungshierarchie nach unten („Erben“)
private	-	Nur Klasse selbst („Ich“)
package	~	Alle innerhalb des Packages („Firma“)

Statische Variablen und Methodenwerden unterstrichen gekennzeichnet. Konstanten (welche sinnvollerweise auch statisch sind) entsprechend der üblichen Namenskonventionen in Großbuchstaben.

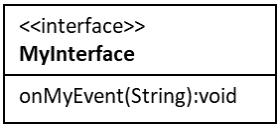


Weiterhin gibt uns UML die Möglichkeit, **abstrakte** Klassen zu kennzeichnen, indem in geschweiften Klammern „abstract“ hinter oder unter dem Klassennamen geschrieben wird. Abstrakte Methoden werden *kursiv* geschrieben (oder auch mit der Info „{abstract}“ versehen.



3 Interface

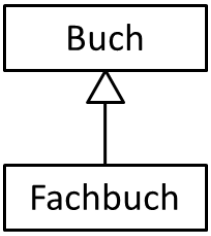
Interfaces werden wie Klassen notiert, allerdings mit der Stereotypangabe <<interface>>. Die Methoden im Interface werden ohne Zugriffsspezifikatoren angegeben, da die Methoden in der implementierenden Klasse ohnehin public sind.



4 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).

4.1 Vererbung



Bei einer **Vererbung** wird angezeigt, dass eine (Kind-) Klasse die Eigenschaften, Methoden und auch Assoziationen einer (Eltern-) Klasse erbt. Der Pfeil zeigt hierbei von der Kindklasse zur Elternklasse. Man spricht hier auch von „**Erweiterung**“, die Kindklasse „erweitert“ also die Elternklasse, was den Programmierbegriff „`extends`“ auch erklärt. Ein weiterer Begriff ist „**Spezialisierung**“ – die Kindklasse spezialisiert die Elternklasse. Andersherum generalisiert die Elternklasse die Kindklasse, was als „**Generalisierung**“ bezeichnet wird. Die Generalisierung ist somit das Gleiche wie die Spezialisierung – man ändert begrifflich nur den Standpunkt.

Hinweis: Üblicherweise stellt man die Elternklasse im Diagramm über der Kindklasse dar, es darf aber abgewichen werden.

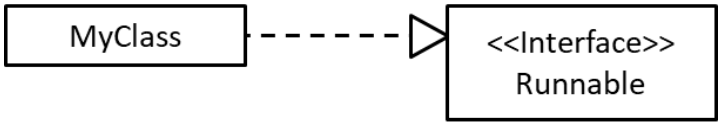
Der Javacode zu dieser Vererbung würde wie folgt lauten:

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

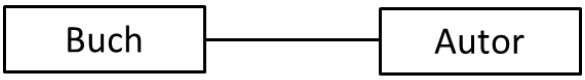
Weiterhin gibt es Programmiersprachen, welche „Mehrfachvererbungen“ erlauben, bei denen eine Kindklasse also mehrere Elternklassen haben kann. Dies wird entsprechend durch mehrere Pfeile symbolisiert.

4.2 Schnittstellenimplementierung

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



4.3 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.

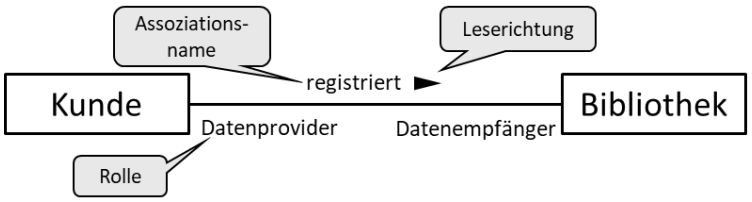
Der Javacode zu dieser Beziehung könnte wie folgt lauten:

```
public class Buch {
  private Autor myAutor;
}
```

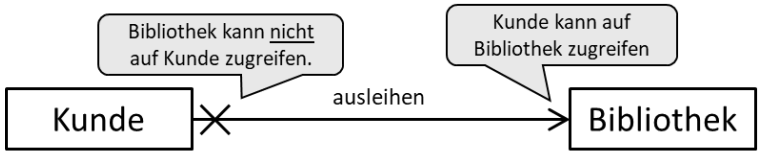
```
public class Autor {
  private Buch[] myBuch;
}
```

Die Buchklasse hat also eine Referenz auf den Autor und der Autor eine Referenz auf Buch – oder wie hier den Büchern. Welche von den beiden jetzt in welcher Form existiert, spezifiziert die einfache Assoziation nicht.

Um dies nun deutlicher zu machen, kann auf den Assoziationsline noch ein erklärender Begriff, der **Assoziationsnamen**, optional mit der Angabe der **Leserichtung** ergänzt werden. Hierbei können weiterhin noch die **Rollen** der Objekte im Rahmen der Assoziation mit spezifiziert werden.

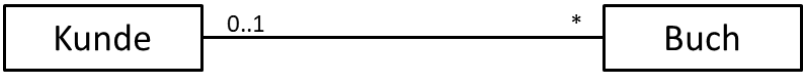


Alternativ kann eine Assoziation auch einen Aufruf modellieren. Hierfür wird in der Regel eine „**gerichtete Assoziation**“ dargestellt, was ein Pfeil mit einer offenen Spitze ist:



Diese gerichtete Assoziation zeigt die „**Navigierbarkeit**“ an. Hiermit will man ausdrücken, dass der Kunde einen Zugriff auf die Bibliothek hat (offener Pfeil). Die Bibliothek hat aufgrund des Kreuzsymbols keinen Zugriff auf den Kunden. Fehlt dieses Kreuzsymbol, so macht das Diagramm über diese Zugriffsrichtung keine klare Aussage.

Eine weitere Angabe ist die Kardinalität oder „**Multiplizität**“:



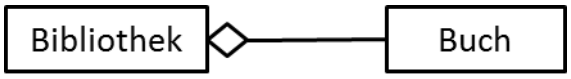
Hier hat der Kunde auf 0 bis unendlich viele Bücher Zugriff, ein Buch wiederum zu keinem, oder einem Kunden. Es gibt verschiedene Multiplizitäten:

Symbol	Bedeutung	Beispiel
1	Genau 1	Ein Buch wurde von genau einem Autor geschrieben
m	Genau m	Ein Buch wurde von genau 4 Autoren geschrieben
*	0 bis unendlich (*)	Ein Kunde kann null bis unendlich viele Bücher leihen
1..*	1 bis unendlich	Ein Kunde muss mindestens ein bis maximal unendlich Bücher lesen
m..*	m bis unendlich	Ein Kunde muss mindestens 10 bis maximal unendlich Bücher lesen
0..1	0 oder 1 (*)	Ein Kunde hat keine oder eine Treuerabattkarte
n..m	n bis m	Ein Buch hat mindestens 10 bis maximal 1000 Seiten
k,l,m..n	k oder l oder n bis m	Ein Team hat 2 oder drei oder 11 bis 15 Mitglieder

(*) optionale Assoziation – sprich es darf auch 0 sein

4.3.1 Aggregation

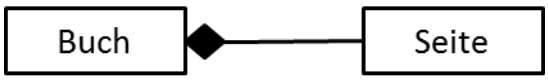
Mit Hilfe einer Aggregation wird dargestellt, dass ein Element Teil eines Ganzen ist, wobei das Ganze ohne dem Teilelement sinnvoll existieren kann. Die Aggregation ist ein Sonderfall der Assoziation.



Merke: Immer dann, wenn das Ganze zerstört wird, kann das Teil immer noch existieren.

4.3.2 Komposition

Die Komposition ist auch ein Sonderfall der Assoziation. Es wird dargestellt, dass ein Element ein zwingender Teil eines Ganzen ist – das Ganze also ohne dem Teilelement nicht sinnvoll existieren kann („Existenzabhängigkeit“).



Merke: Immer dann, wenn das Ganze zerstört wird, wird damit auch das Teil zerstört

Man spricht auch von „Live with it, die with it“.

4.4 Nutzung von Interfaces

Wenn Klassen Interfaces implementieren, so weisen die Nutzer dieser Klassen oft nur eine Variable vom Interfacetypen auf, nicht von der Klasse. Dies wird mit einem eigenen Symbol für die Nutzung realisiert – ein offener gestrichelter Pfeil.