Inhalt - Theorie der OOP

3.3 Prinzipien der objektorientierten Programmierung

Die objektorientierte Programmierung basiert auf einigen einfachen Ideen. Grundpfeiler der OOP-Philosophie sind vier Konzepte[4] :

3.3.1 Abstraktion

Wenn wir auf dem Computer Daten verarbeiten wollen, müssen wir Objekte der tatsächlichen Welt in einem Computerprogramm nachbilden. Dazu ist es nötig, diese Objekte auf die relevanten Gesichtspunkte zu reduzieren. Die Reduktion auf die wesentlichen Gesichtspunkte nennt man Abstraktion. Wenn wir z.B. in einem Autorennspiel ein Auto programmieren wollen, ist nicht sinnvoll, jeden Aspekt des Autos auf das Programm zu übertragen. Es ist beispielweise egal, ob das Auto bequeme Sportsitze oder eine Klimaanlage hat. Uns interessieren eher Eigenschaften wie die Geschwindigkeit und die Leistung. Eigenschaften sind die Variablen, die für das Objekt gültig sind. Auf der anderen Seite wollen wir auch etwas mit dem Auto tun, beispielsweise lenken, beschleunigen und abbremsen. Dies sind Funktionen, die das Auto erfüllen muss, um ein vollwertiges Auto zu sein. Funktionen heißen in der objektorientierten Programmierung Methoden. Ein Objekt in der OOP definiert sich folglich aus Verbund von Eigenschaften und Methoden.

Ein **Objekt** ist ein Verbund aus Variablen und zugehörigen Methoden.[5]

Für unser Autoobjekt könnten wir z.B. die Eigenschaften (englisch: *property*) Geschwindigkeit und PS-Zahl, sowie die Methoden Beschleunigen und Bremsen definieren.

```
property geschwindigkeit, leistung

on beschleunigen
...
end beschleunigen

on bremsen
...
end bremsen
```

Es ist wichtig zu unterscheiden, dass Objekt in diesem Sinne nicht gleich Programmcode bedeutet. Der Programmcode, der die Eigenschaften und Methoden definiert und beschreibt, heißt Klasse; in Lingo spricht man von Parent-Skript oder übergeordnetem Skript.

Eine **Klasse** bzw. ein **Parent-Skript** enthält die Beschreibung der Eigenschaften und Methoden aller Objekte eines bestimmten Typs.

In Lingo wird durch die Einstellung *Parent Script* bzw. *Übergeordnetes Skript* in den Skripteigenschaften signalisiert, dass es sich nicht um ein Filmskript, sondern die Beschreibung eines Objekttyps handelt. Es ist wichtig, dass ein Parent-Skript einen Namen enthält, damit später auf das Skript Bezug genommen werden kann. In unserem Fall wollen wir es *AutoObjekt* nennen.

Auf die Methoden und Eigenschaften einer Klasse kann in der Regel noch nicht zurückgegriffen werden, es muss zuerst eine sogenannte **Instanz**, also ein Beispiel der Klasse geschaffen werden. Das erzeugen einer Klasse wird auch Konstruktion des Objektes genannt; dabei wird eine spezielle Methode, **Konstruktor** genannt, einer Klasse aufgerufen. Erst dabei wird Speicherplatz für das Objekt und seine Eigenschaften reserviert.

In Lingo heißt der Konstruktor immer new(). Es ist üblich, im new-Handler die Objekteigenschaften zu initialisieren, d.h. auf definierte Anfangswerte zu setzen:

```
on new me
geschwindigkeit = 0
leistung = 120
return me -- jeder new-Handler endet mit dieser Zeile
end new
```

Einsteiger können sich anfangs nur schwer etwas unter dem Begriff Instanz vorstellen; interessanterweise gehen wir jedoch tagtäglich ganz selbstverständlich mit Instanzen um. Wenn wir ein Programm wie den Windows-Texteditor Notepad öffnen, ist uns klar, dass das Programm nicht direkt von der Festplatte läuft, sondern zuerst Speicherplatz für sich reservieren muss. Dabei werden die Eigenschaften des Programms gesetzt, wie Fenstergröße und der zu editierende Text. Wir nehmen unsere Eingabe also an einer Instanz des Programms Notepad vor. Wenn wir das Programm noch einmal öffnen, öffnet sich ein zweites Fenster, also eine weitere Instanz. Sie hat andere Eigenschaften, sie kann also einen anderen Text editieren und eine andere Fenstergröße aufweisen. Alle Funktionen, wie Laden und Speichern, funktionieren jedoch identisch wie in unserer ersten Instanz, sie wirken sich nur auf eine andere Datenmenge aus. Wir erwarten nicht, dass eine Eingabe im einen Fenster den Text im anderen in irgend einer Weise beeinflusst.

Das Erzeugen von Instanzen als ersten Schritt zur Nutzung eines Objektes wird oft von OOP-Einsteigern als lästig und unnötig empfunden - es hat jedoch praktische Aspekte. In unserem Autorennspiel müssen wir nicht für jedes teilnehmende Auto ein neues Skript schreiben, sondern erzeugen einfach Instanzen unseres *AutoObjekt*-Skripts.

```
auto1 = script("AutoObjekt").new()
auto2 = script("AutoObjekt").new()
auto3 = script("AutoObjekt").new()
```

Die Instanzen speichern wir in Variablen ab; diese enthalten eine Referenz auf das erzeugte Objekt. Über die Referenzvariablen *auto1*, *auto2* und *auto3* können wir also auf unsere Auto-Objekte Einfluss nehmen.

Wichtig

Auf Objekte wird immer über eine **Referenzvariable** zugegriffen.

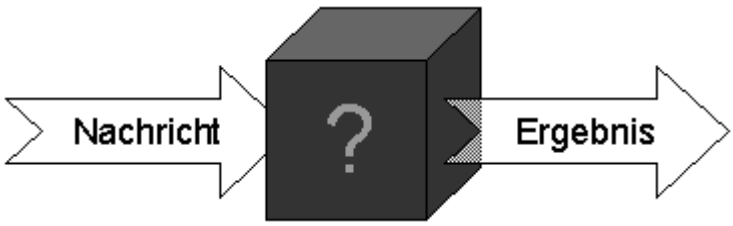
Wie diese Referenz genau aussieht, wird im Kapitel "Erzeugen von Instanzen mit dem new-Handler" genauer beschrieben.

3.3.2 Datenabstraktion

OOP-Einsteiger, die vorher schon Lingo programmiert haben, werden sich vielleicht fragen: "Wenn ein Objekt ein Verbund aus Variablen und Methoden, also Handlern ist, ist dann ein Director-Filmskript nicht auch ein Objekt? Jedes Filmskript definiert doch auch globale Variablen und eigene Handler." Was Variablen und Handler eines Objektes jedoch von denen eines Filmskriptes unterscheidet, ist der Gültigkeitsbereich. Globale Variablen und Handler sind im ganzen Film gültig, Eigenschaften und Methoden gelten nur innerhalb eines Objekts. Daraus folgt, dass es in verschiedenen Objekten gleichlautende Eigenschaften und Methoden geben kann. Man gibt einfach an, welches Objekt gemeint ist. Wenn wir in unserem Autorennenbeispiel wissen wollen, wie schnell *Auto1* fährt, schreiben wir `the speed of auto1`, bzw. `auto1.speed`. Wenn wir *Auto2* abbremsen wollen, bremst der Ausdruck `auto2.bremsen()` wirklich nur dieses Auto ab; die anderen Autos bleiben unbeeinflusst.

In unserem Autorennenbeispiel ist es sicher sinnvoll, die Eigenschaft *speed* nicht direkt zu setzen; in den seltensten Fällen wird sich die Geschwindigkeit eines Autos schlagartig ändern, sondern ist ein Folge des Beschleunigens oder Abbremsens. Aus diesem Grund sollte von diesen Methoden gebrauch gemacht werden. Dieser Vorgang nennt sich Datenabstraktion. Es sollte dem Programmierer unwichtig sein, was im Innern des Objektes vor sich geht, wenn er die *bremsen*-Methode aufruft; die tatsächliche Implementierung bleibt ihm verborgen. Dadurch ist es später einfach, die Logik des Auto-Objekt z.B. durch ein ausgefeilteres physikalisches Modell zu ersetzen. Wenn wir aus dem Autorennen ein Pferderennen machen wollen, ersetzen wir das Auto-Objekt durch ein Pferdobjekt; es muss lediglich sicher gestellt werden, dass die entsprechenden Methoden vorhanden sind, dass es also eine definierte Schnittstelle gibt.

Dieses Prinzip wird auch als Black-Box-Prinzip oder *Encapsulation* bezeichnet. Gemeint ist hier dasselbe: man schickt eine Nachricht an ein Objekt und erhält ein Ergebnis; um das, was genau in der Kiste passiert, muss man sich nicht kümmern.



Das Black-Box-Prinzip

Leider kann man in Lingo nicht verhindern, dass man Eigenschaften direkt manipuliert, die nur das Objekt selbst verändern darf. Umso wichtiger ist es daher, dass man das Auslesen oder Setzen von Eigenschaft gleichsam in eine Methode "einwickelt", und sich konsequent daran hält, die Eigenschaften nur über diese Methoden zu setzen und auszulesen. Dies erleichtert das nachträgliche Verändern des gesamten Programmcodes sehr.

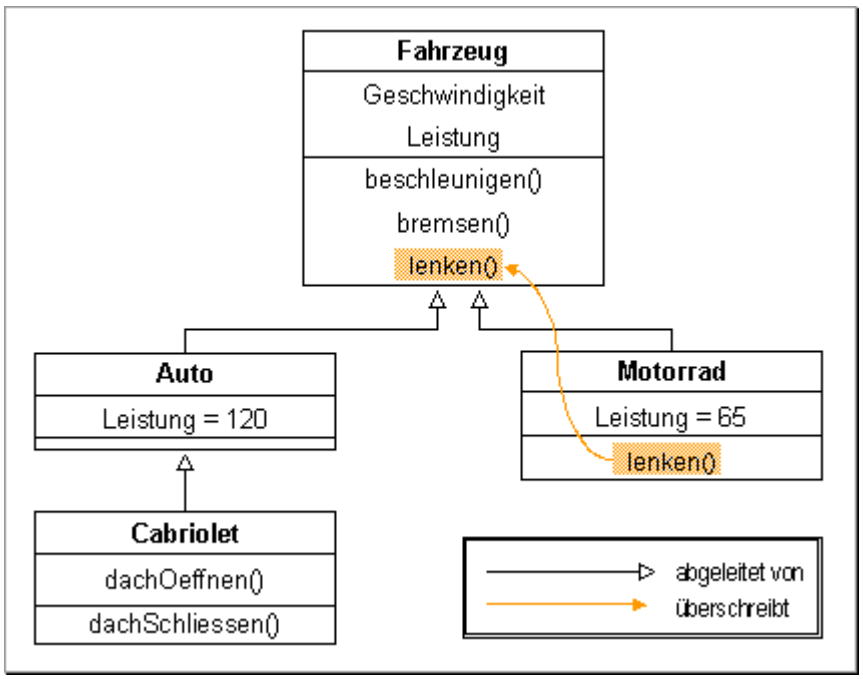
Die objektorientierte Vorgehensweise bedingt also auch ein Umdenken. Ein Programm wird nicht mehr als Aneinanderreihung von Programmzeilen und Werten gesehen, sondern als Objekte, die über Nachrichten miteinander Kommunizieren. Daher spricht man in diesem Zusammenhang oft von einem Paradigmenwechsel.

3.3.3 Vererbung

Vererbung ist ein Prinzip, dass in Director eine eher untergeordnete Bedeutung hat. Es ermöglicht einer Klasse, Unterklasse genannt, Methoden und Eigenschaften einer anderen, der sogenannten Superklasse oder Oberklasse, mitzubenutzen. Dabei stellt die Unterklasse eine Spezialisierung der Superklasse dar. Die Unterklasse erhält dabei alle Methoden und Eigenschaften der Superklasse, man sagt auch, die Unterklasse erbt von der Superklasse.

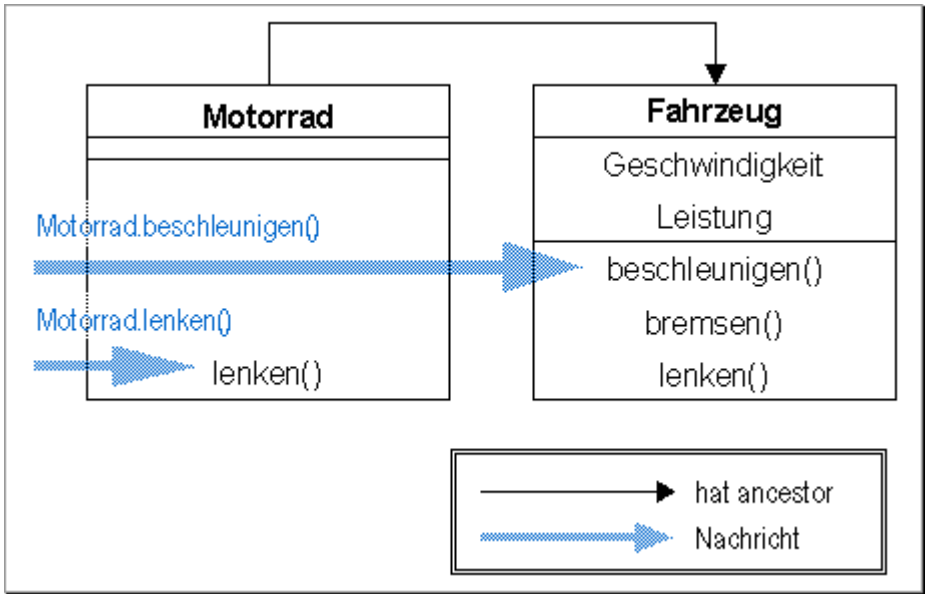
Bei unserem Autorennen könnten wir Beispielsweise von der Klasse *Auto* die speziellere Klasse *Cabriolet* ableiten. Sie hat die gleichen Methoden und Eigenschaften wie ein normales Auto, aber zusätzlich werden die Methode *dachOeffnen()* und *dachSchliessen()* implementiert, die ein allgemeines Auto nicht hat. Um auch andere Fahrzeuge am Rennen teilnehmen zu lassen, ließe sich die allgemeine Klasse *Fahrzeug* einführen, die schon alle Methoden implementiert, die wir im Rennen benötigen, z.B. *beschleunigen()*, *bremsen()* und *lenken()*. Somit ist gesichert, dass alle von Fahrzeug abgeleiteten Klassen diese Methoden tatsächlich implementieren. Dabei ist es nicht nötig, diese Methoden zu übernehmen; sie können auch überschrieben werden; wenn z.B. eine von *Fahrzeug* abgeleitete Motorrad-Klasse ein anderes Lenkverhalten aufweisen muss, kann sie eine eigene *lenken()*-Methode implementieren, welche die allgemeine aus *Fahrzeug* verdeckt. Genauso kann eine Methode die gleichnamige Methode der Superklasse aufrufen und zusätzlich eigene Aspekte implementieren.

Die Vererbung der Teilnehmer unseres Autorennens könnte z.B. so aussehen:



Vererbung erfüllt in Hochsprachen wie Java und C++ vor allem zwei Aufgaben: einerseits hilft sie, das Programm besser zu strukturieren, andererseits verringert sie den Umfang des Programmcodes und erleichtert die Programmpflege.

In Lingo funktioniert Vererbung vollkommen anders. Sie schafft keine klaren Programmstrukturen, sondern baut eher eine Nachrichtenkette auf. Das Schlüsselwort dafür ist die Objekteigenschaft the ancestor. Diese stellt eine Art übergeordnetes Objekt dar; Methoden und Eigenschaften, die in einem Objekt nicht definiert werden, werden automatisch an den ancestor weitergegeben. Wird hier keine Implementierung gefunden, wird, falls vorhanden, diese Nachricht an dessen ancestor weitergegeben u.s.w.



In Java und C++ beschreibt spielt sich Vererbung auf Klassenebene ab, in Lingo auf Objektebene. Lingo wird aus diesem Grund manchmal in Internetdiskussionen als klassenlose objektorientierte Programmiersprache bezeichnet, d.h. die Vererbung dient dazu, bestehende Objekte, Prototypen genannt, durch eigene Methoden und Eigenschaften zu erweitern. Folglich kann auch von einem Nicht-Skriptobjekte geerbt werden, zum Beispiel einem Sprite oder einer Liste.

In Lingo ist es möglich, den ancestor jederzeit zu wechseln. Man könnte also von dynamischer Vererbung reden, im Gegensatz zur statischen Vererbung in Java und C++, bei denen die Klassen in einem statischen, unveränderbaren Verhältnis stehen. Die Metapher mit dem *ancestor*, also dem Vorfahren, hinkt in Lingo jedoch ein wenig, denn wer kann schon seine Vorfahren so ohne weiteres wechseln?

Wie die Vererbung mit Lingo im Detail funktioniert, ist im Kapitel "Vererbung mit the ancestor" nachzulesen.

3.3.4 Polymorphismus

Unter Polymorphismus (Vielgestaltigkeit) versteht man den Umstand, dass erst bei der Ausführung eines Programmes bekannt ist, auf welches Objekt eine Operation ausgeführt wird. Beispielsweise könnte in unserem Autorennen ein Pferd mitlaufen, solange es die Methoden *bremsen()*, *laufen()* und *lenken()* implementiert, auch wenn es nicht von Fahrzeug abgeleitet ist. In einem Zeichenprogramm könnte jedes zeichenbare Objekt die Methode *zeichnen()* implementieren, egal ob es eine geometrische Form, ein Schriftzug oder eine komplexe Figur ist. Das Programm ruft diese Methode von allen Objekten auf, die es zeichnen will. Es stellt sich dann erst im Programmablauf heraus, auf welche Objekttypen sich die Methode bezieht.

3 Vorteile der objektorientierten Programmierung 3.4 Lingo vs. Java