

# Grundlagen - Kapitel 4: Klassen und deren Funktionsweise

In letzter Zeit habe ich hier nicht wirklich viel getan, sodass ich jetzt einmal als eine Kleinigkeit wieder etwas zum Thema ?Auran Game Script? von mir gebe.

Und zwar geht es heute um Klassen. Dabei werde ich die Idee von Klassen, wie man sie aus der Softwareentwicklung her kennt, erklären. Denn: In Trainz haben Klassen genau die gleiche Funktion!

## Die Definition einer Klasse

In fast jedem guten Lehrbuch zu egal welcher Programmiersprache steht ein Satz, der so oder so ähnlich lautet: ?Eine Klasse ist nichts weiter, als ein Objekt.?

Was kann aber der Laie daraus gewinnen? ? Genau: Noch gar nichts.

Wir kennen Klassen bereits von unseren Standard-Scripten:

Jedes Objekt in Trainz gehört einer sog. ?Kind? an (zu Deutsch sinngemäß etwas wie: Objekt-Kategorie). Und da wird es jetzt interessant!

Wir können also nun eine Verbindung zwischen einer Klasse (einem Objekt) und einer Objekt-Kategorie bilden. Auch, wenn ein Softwareentwickler mir für den folgenden Satz den Vogel zeigte, erfüllt er doch den Sinn, daß der Laie die Funktionsweise einer Klasse versteht.

Nämlich: Eine Klasse ist zwar ein Objekt, die Definitionen innerhalb einer Klasse sind aber vielmehr die Definition einer Objekt-Kategorie.

Unabhängig davon, wie eine Klasse in ihrem Inneren definiert ist, können wir jetzt verstehen, daß eine Klasse dazu da ist Objekte zu beschreiben.

Gehen wir einen Schritt weiter:

In unserem Script aus dem [2. Teil \(„Ein Mesh ein- bzw. ausschalten“\)](#) haben wir auch eine Klasse erzeugt und zwar die Klasse ?CTutorialRoadSign?.

Im Grunde genommen haben wir damit schon eine Kategorie geschaffen, auch wenn der Name der Klasse nicht eindeutig als eine solche zu identifizieren ist. Unsere Klasse kann doch jetzt für jedes beliebige Schild mit einem ausschaltbaren Mast fungieren und ihm so die Funktion zum Ausschalten des Mastes geben. Und damit haben wir auch schon erfasst, wie sich eine Klasse definieren lässt.

Unsere Klasse kann nämlich:

- Den Masten ausblenden
- Diese Einstellung laden und speichern und wieder erneut anwenden, nach dem das Objekt neu geladen wurde

Wenn das mal keine ordentliche Klasse ist!

## Das ?Innere? einer Klasse

Wir haben vorhin den allgemeinen Aspekt von Klassen besprochen, ohne darauf genau auf ihr innerstes, also ihrer eigentlichen Definition einzugehen.

Das tun wir nun:

In einem früheren Teil meiner Blog-Einträge habe ich schon einmal von Variablen und Funktionen gesprochen. Da wir uns in Trainz ausschließlich in Klassen bewegen und es keinen globalen Gültigkeitsbereich (also Bereiche außerhalb von Klassen) gibt, sind diese Bezeichnungen nicht ganz richtig.

Die Funktionen innerhalb einer Klasse, also so etwas wie ?GetProperties? werden ?Methoden? genannt, sowie Variablen innerhalb einer Klasse, wie ?m\_bIsPoleVisible?, ?Member? genannt werden. Das tut für die eigentlichen Erläuterungen nicht sonderlich zur Sache, es ist aber ein Muß für mein Programmiererherz, dies zu erwähnen.

Außerdem kann es innerhalb von Klassen auch noch sog. ?Makros? geben. Diese sind zwar nicht sonderlich schwierig, aber genau das ist der Grund, weshalb ich diese einmal hinten an stelle.

Für uns gibt es also bisher nur: Member und Methoden innerhalb einer Klasse.

Was Member (Variablen) und Methoden (Funktionen) sind, sollten wir ja bereits verstanden haben. Wir wissen nun auch, daß eine Klasse ein Objekt beschreibt.

Wenn wir diesen Gedanken nun einmal fort führen, so ist es doch klar: Methoden und Member helfen uns diese Objekte eindeutig beschreiben zu können.

Dazu einmal ein Beispiel:

Stellen wir uns nun einmal vor, wir sollen die Funktionsweise eines Autos mittels einer Klasse bereitstellen und das ohne über die genaue Ausarbeitung nachzudenken. Wir halten das also erstmal ganz einfach.

Was kann ein Auto so machen?

- Fahren und Stehen
- Beschleunigen und Bremsen
- Motor ein- oder ausgeschaltet haben
- Türen öffnen und schließen
- etc.

Wenn wir uns nun überlegen, wie wir das ganze strukturell aufbauen können ergeben sich doch Methoden (Funktionen) wie:

- SetzeMotor
- SetzeBeschleunigung
- ÖffneTüre
- etc.

Und dazu benötigen wir Member (Variablen) wie:

- Geschwindigkeit
- Beschleunigung
- IstMotorEin
- etc.

Man sieht also: Die Leute, die sich die OOP (Objektorientiertes Programmieren, also Programmierung mit Klassen) ausgedacht haben, waren gar nicht so dämlich.

Es erfüllt durchaus seinen Sinn ? ich denke bis hier her, sollte alles klar sein.

### **Thema Vererbung**

Das Thema der Vererbung ist nicht wirklich schwierig, allerdings gibt es immer wieder Verständnisschwierigkeiten, sodaß dieses Thema wirklich ziemlich ausschweifend in einigen Lehrbüchern erklärt wird.

Vererbung kennen wir doch alle aus dem Leben. Vererbung in der Programmierung ist nahe zu das gleiche wie die Vererbung aus der Biologie (=> Fortpflanzung).

Eigenschaften werden von anderen Objekten angenommen. AHA! Wieder das Wort Objekt (Auch wenn es beim Thema Fortpflanzung eher weniger sinnvoll gewählt ist, ein Mensch ist ja so gesehen kein Objekt).

Also: Eine Klasse kann die Eigenschaften anderer Klassen und deren Definitionen für sich übernehmen.

Der Verwendungszweck ist ganz einfach. Wir können für verschiedene Objekte der gleichen Kategorie die gleichen Funktionen bereitstellen. Das klingt vielleicht jetzt ein wenig verwirrend aber ein Beispiel wird sicher ein wenig Licht ins Dunkle bringen:

Bleiben wir beim Auto. Ein Auto ist ja schön und gut, aber wer es Abwechslungsreich haben möchte, hätte sicher gerne noch ein paar Motorräder.

Da haben wir nun ein Problem: Wir können die Klasse ?Auto? nicht 1:1 für ?Motorrad? übernehmen. Es ist auch wenig sinnvoll die gleichen Funktionen mehrfach für mehrere Klassen zu implementieren. Also bauen wir uns doch eine neue Klasse ?Fahrzeug? und packen dort alle allgemeinen Funktionen für Fahrzeuge hinein.

Ein Fahrzeug kann also:

- Beschleunigen und Bremsen
- Fahren und stehen
- Motor ein- und ausschalten
- etc.

Die Methoden brauche ich wohl nicht mehr zu erläutern, wir haben sie ja bereits beim Auto.

Wir löschen also die allgemeinen Fahrzeugfunktionen aus der Klasse ?Auto? raus und sagen nun: Klasse ?Auto? erbt von Klasse ?Fahrzeug? (rein Schematisch). Das heißt: Die Klasse ?Auto? enthält alle Funktionen der Klasse ?Fahrzeug? und kann gleichzeitig eigene Funktionen ergänzen.

Nun können wir auch das Motorrad implementieren und sagen: Klasse ?Motorrad? erbt von Klasse ?Fahrzeug? und auch diese Klasse kann wieder eigene Funktionen ergänzen.

Damit haben wir gleich mehrere Ziele erreicht:

Wir haushalten zum einen ein wenig mit den Ressourcen und zum zweiten gestaltet sich das ganze ein wenig übersichtlicher. Außerdem sind wir so in der Lage die Klasse ?Fahrzeug? immer weiter mit neuen

Klassen zu ergänzen, ohne nochmal die ganzen Methoden zu implementieren.  
Klassen wie ?LKW?, ?Lastwagen?, etc. lassen sich so beliebig ohne großen Aufwand ergänzen.  
Wer es ganz speziell haben will kann ja diese Klassen noch einmal erweitern: Klasse ?Lastwagen? kann zum Beispiel um die Klasse ?Eiswagen? erweitert werden, die dann Funktionen für diese typische Eiswagenmusik, etc. enthalten kann.

Auch das Thema sollte klar sein!

### Umsetzung des ganzen in Trainz

Genau das macht Trainz!

Wir erzeugen jedes Mal eine Klasse in o.g. Beispiel die ?CTutorialRoadSign? und diese erbt von ?MapObject?. Wir wissen also jetzt: isclass ist in Auran Game Script das Schlüsselwörtchen für ?Vererbung?.

Die Hauptklasse eines Objektes in Trainz muß immer von einer Klasse aus dem Spiel erben.  
Das bedeutet, daß es nicht möglich ist ein ganz neues ?Objekt? zu kreieren. Das macht auch keinen Sinn.  
Auran Game Script ist eine sog. API (Application Programming Interface).  
Das heißt, wir müssen uns immer innerhalb dieser Rahmenanwendung bewegen und sind nicht fähig mehr zu machen, als Trainz kann (schade eigentlich!). Auran Game Script ist auch keine wirkliche Programmiersprache sondern eine ziemlich gute fortgeschrittene Script-Sprache der JET-Engine. Diese hat ein großes Potential, läßt es aber nicht zu etwa DLLs oder derartiges zu laden um somit ganz neue Funktionen in Trainz hineinzubringen.  
Klassen sind sogar in der Trainz-Script API ein wenig komplexer. Es fehlen noch Erlätuerungen zu sog. statischen Klassen oder finalen Klassen. Das aber muß noch ein wenig warten.

### Übersicht von möglichen Klassendefinitionen

Aus den Beispielen:

Code

```
class                                     CFahrzeug
{
...
};
```

Code

```
class          CAuto          isclass          CFahrzeug
{
...
};
```

Code

```
class          CMotorrad          isclass          CFahrzeug
{
...
};
```

### In Trainz:

#### Code

```
class          CFahrzeug          isclass          MapObject
{
...
};
```

Man denke dran: Die Hauptklasse eines Objektes in Trainz muß immer von einer Spiel-Klasse erben (Hier: MapObject)!  
Danach kann man (s.u.) weiter vererben.

#### Code

```
class          CAuto          isclass          CFahrzeug
{
...
};
```