Geschwister-Scholl-Gymnasium
Löbau
Fachbereich Informatik

# Komplexe Leistung

## Objektorientierte Programmierung im Informatikunterricht


**Eingereicht von:** Flogex


**Fach:** Informatik (Jahrgangsstufe 11)

**Eingereicht am:** 04. April 2016

# Inhalt

# Hauptteil

## 1 Einleitung

Objektorientierte Programmiersprachen erfreuen sich einer sehr großen Beliebtheit. Allein 4 von 5 der beliebtesten Programmiersprachen des TIOBE-Index 2016[1] unterstützen das objektorientierte Programmierparadigma. Dieser Erfolg kommt nicht zuletzt durch die Orientierung an der menschlichen Denkweise zustande, durch welche das objektorientierte Programmieren für einen Entwickler natürlich wirkt, denn er kann Erfahrungen aus seinem Leben im Code umsetzen.

In der Praxis der Softwarearchitektur und -entwicklung wird dieses Konzept sehr oft verwendet. Von objektorientierter Analyse über das objektorientierte Design bis zur objektorientierten Programmierung - überall werden Kenntnisse in der Objektorientierung verlangt.

### 1.1 Zielstellung

Ziel dieser Komplexen Leistung ist es, die grundlegenden Bausteine und Konzepte der Objektorientierung zu erklären, sodass ein Leser einen Überblick über diese erhält. Im zweiten Teil soll die objektorientierte Programmierung in der Programmiersprache *JavaScript* angewendet werden. Dabei soll aufgezeigt werden, wie trotz der scheinbaren Einschränkungen dieser Sprache hinsichtlich der Objektorientierung das Konzept umgesetzt werden kann.

### 1.2 Begriffserklärungen

Die Implementierung ist die konkrete Umsetzung einer in der Spezifikation abstrakt beschriebenen Logik. Eine Methode stellt beispielsweise die Implementierung einer Operation dar.

Die Spezifikation beschreibt auf einer abstrakten Ebene das Verhalten und die Anforderungen des zu spezifizierenden Objekts. Die Spezifikation einer Klasse definiert die Menge aller Operationen sowie Kontrakte für Methoden. Die Spezifikation einer

[1] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Methode - die Operation - beschreibt die Funktionalität, den Kontrakt und die Signatur des Aufrufs.

"Ein Datentyp ist eine strukturierte Menge von Werten. Damit ist gemeint, dass auf den Elementen des Typs eine Anzahl von Operationen (mit bestimmten zugesicherten Eigenschaften) definiert sind, mit deren Hilfe man Objekte (Daten) dieses Typs manipulieren kann."[2]

Die Signatur einer Methode besteht aus ihrem Namen und der Anzahl sowie den Datentypen der Parameter.

Im Folgenden werden die Gesamtheit von Attributen und Daten sowie Operationen und Methoden eines Objektes oder Teile davon als *Elemente des Objekts* bezeichnet.

# 2 Objektorientierte Betrachtungsweise

## 2.1 Objekte im Alltag

Das Konzept der Objekte entspricht der Sichtweise der Menschen. Man sieht die Objekte *Auto* oder *Haus* als zusammenhängende Gegenstände, denen Eigenschaften direkt zugeordnet sind und welche über Funktionalitäten beruhend auf diesen Eigenschaften verfügen. Durch die objektorientierte Programmierung versucht man, die reale Welt abzubilden. Objekte in der Programmierung müssen aber nicht realen Gegenständen entsprechen, sondern können auch Datenstrukturen wie eine Adresse sein.

Menschen nehmen ebenfalls eine Klassifizierung von Objekte in der realen Welt vor. So gehören die Objekte *Dreieck*, *Rechteck* und *Kreis* selbstverständlich zu der Gruppe der *geometrischen Formen*. Diese Denkweise wird in vielen objektorientierten Programmiersprachen als Klassenkonzept umgesetzt.

## 2.2 Definition der Objektorientierung

[2] http://cvpr.uni-muenster.de/teaching/ss09/info2SS09/script/Kapitel11-ADT-1.pdf

Die Objektorientierung ist ein Konzept der Softwareentwicklung, in welchem ein System aus Objekten besteht, die untereinander Nachrichten austauschen.[3] Ein Objekt verfügt über Eigenschaften, Methoden und eine Identität. In vielen Programmiersprachen, wie beispielsweise Java, lassen sich ähnliche Objekte zu Klassen zusammenfassen.

Den Begriff *Objektorientierung* prägte Alan Kay, einer der Entwickler der Programmiersprache *Smalltalk*, welche als eine der ersten objektorientierten Programmiersprachen angesehen wird. Ebenfalls definierte er die drei Grundelemente der objektorientierten Programmierung: Datenkapselung, Vererbung und Polymorphie, auf welche in den folgenden Kapiteln eingegangen werden soll.

## 2.3 Objekte, Botschaften, Eigenschaften, Methoden

### 2.3.1 Das Objekt

Namensgebend für die Objektorientierung sind die Objekte. Diese werden durch ihre Attribute mit entsprechenden Werten (Zustand) und ihre Funktionalitäten (vom Objekt unterstützte Operationen) charakterisiert. Logik und Daten liegen also - im Gegensatz zu dem Vorgänger der objektorientierten Programmierung, der strukturellen Programmierung - als eine Einheit im Objekt vor und werden nicht getrennt. Durch die Verwaltung zusammengehöriger Daten und der dazugehörigen Funktionalitäten durch ein Objekt kann eine Anwendung modularisiert werden: Sie wird aus Bausteinen zusammengesetzt, welche bei Bedarf auch ausgetauscht oder in einer anderen Anwendung wiederverwendet werden können - "vergleichbar mit der Entwicklung und dem Zusammenbau eines Motors, bei dem im Wesentlichen genormte Maschinenteile (Schrauben, Bolzen etc.) zum Einsatz kommen."[4]

Ebenfalls verfügt ein Objekt über eine eindeutige Identität, durch welche es sich von einem anderen Objekt unterscheiden lässt, auch wenn beide über den gleichen Zustand und die gleichen Funktionalitäten verfügen.

---

[3] vgl. https://de.wikipedia.org/wiki/Objektorientierung

[4] http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_03_001.html#dodtpf766d18e-5ded-46aa-8e8e-625d86cb0d78

Beispiel: Ein Auto wird durch die Automarke, seine Farbe und seine Maße charakterisiert. Von einem anderen Fahrzeug des gleichen Typs, mit der gleichen Farbe und den gleichen Maßen lässt sich das Auto aber durch seine Fahrzeugidentifikationsnummer unterscheiden.

Ein weiteres wichtiges Merkmal von Objekten ist die Fähigkeit, Botschaften mit anderen Objekten auszutauschen.

### 2.3.2 Botschaften

Durch den Austausch von Botschaften/Nachrichten können Objekte miteinander interagieren. Dabei sendet das erste Objekt (Sender) einem zweiten Objekt (Empfänger) die Aufforderung, eine Operation auszuführen. Diese Botschaft besteht aus dem Namen der aufzurufenden Operation sowie einer Liste der übergebenen Parameter. Das zweite Objekt reagiert und sendet wiederum eine Botschaft mit dem Ergebnis der ausgeführten Operation an das erste Objekt. Ein Beispiel ist das Abbremsen eines PKW: Tritt der Fahrer auf das Bremspedal, sendet er die Aufforderung, das Fahrzeug abzubremsen. Der PKW reagiert mit dem starten des Bremsvorgangs.

### 2.3.3 Eigenschaften

Objekte haben Attribute (Eigenschaften) mit entsprechenden Daten/Werten. Dabei gibt das Attribut die Bedeutung an, zum Beispiel *Seitenanzahl*, während die Daten den inneren Zustand des Objekt repräsentieren und damit die konkreten Werte, zum Beispiel *356*.

Attribute können direkt auf Daten basieren oder aus anderen Attributen berechnet werden. Beispielsweise wird das Volumen eines Quaders aus Höhe, Breite und Tiefe errechnet und muss nicht direkt gespeichert werden. Diese Attribute bezeichnet man als *abgeleitete Attribute*. Zwischen Attributen können folglich Abhängigkeiten bestehen.[5]

### 2.3.4 Operationen und Methoden

---

[5] vgl. Lahres, Bernhard & Raýman, Gregor & Strich, Stefan: Objektorientierte Programmierung - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn, 2016; S. 70

Neben Attributen besitzen Objekte auch ein Verhalten beziehungsweise eine Funktionalität.

*Operationen* beschreiben das Verhalten eines Objekts abstrakt und definieren die Syntax des Aufrufs, so zum Beispiel die Parameter und deren Typen oder den Typ des Rückgabewertes. Operationen sind von außen sichtbar. Die Gesamtheit aller Operationen wird *Schnittstelle des Objekts* genannt. Über ihre Schnittstellen können Objekte untereinander kommunizieren.[6]

*"Methoden* [...] sind die konkrete Umsetzung [Implementierung] von Operationen."[7] Sie sind nach außen hin nicht sichtbar, weil sie nur für das jeweilige Objekt relevant sind.

Weil ein Aufrufer aber die Interna des aufgerufenen Objektes nicht kennt und die (auch fachliche) Korrektheit der Methode nicht selbst überprüfen kann, ist es wichtig, dass sich beide Objekte an eine Vereinbarung (Kontrakt) beim Aufruf einer Operation halten, welche Vor- und Nachbedingungen definiert. Vorbedingungen sind die Voraussetzung zur Durchführung einer aufgerufenen Operation, für deren Einhaltung der Aufrufer sorgen muss. Nachbedingungen sind die definierten Leistungen, die das aufgerufene Objekt zu erbringen hat, wenn eine Operation aufgerufen und die Vorbedingungen erfüllt wurden. Hat ein Benutzer beispielsweise sichergestellt, dass Strom an einem PC anliegt, und drückt daraufhin den Startknopf, muss der PC hochgefahren werden. Neben Vor- und Nachbedingungen gibt es auch Invarianten, welche unveränderliche Bedingungen darstellen.[8] Die im Kontrakt vereinbarten Bedingungen zählen zur Schnittstelle eines Objekts.

Methoden können die übergebenen Parameter, die Daten des Objekts oder Rückgabewerte anderer Operationen für Berechnungen nutzen und Resultate an den Aufrufer zurückgeben, aber auch Daten des aktuellen Objektes verändern.

Methoden ohne Rückgabewert heißen Prozeduren. Geben sie Werte zurück, werden sie Funktionen genannt.

---

[6] vgl. ebenda, S. 77
[7] ebenda, S. 77
[8] vgl. ebenda, S. 92

## 2.4 Klassen

In der objektorientierten Programmierung können ähnliche Objekte, die sich nur durch ihre Identität und ihren Zustand unterscheiden, zu Klassen zusammengefasst werden. Die Einteilung in Klassen ist sinnvoll, weil Objekte eines Typs, beispielsweise verschiedene Autos, die gleichen Operationen anbieten. Ebenfalls kann man für alle Autos die gleichen Attribute angeben. Die Nutzung von Klassen führt zu einem geringeren Programmieraufwand, da die Elemente einer Klasse nicht für jedes Objekt neu erstellt werden müssen.

Objekte, die einer Klasse angehören, werden als Exemplar oder Instanz dieser Klasse bezeichnet. Die Klasse wiederum ist der *Bauplan* oder die *Schablone* für ihre Instanzen, also ein abstraktes Modell[9], welches erst durch die abgeleiteten Objekte konkrete Ausprägungen erhält. Zum Beispiel werden erst in den Instanzen die Werte der Attribute festgelegt. Instanzen werden durch den Aufruf des *Konstruktors* der Klasse erstellt.

"Die Schnittstelle einer Klasse besteht [...] aus allen durch die Klasse definierten Eigenschaften und Operationen."[10] Weil eine Klasse die Schnittstelle aller ihrer Exemplare festlegt, ist sie gleichzeitig der (Daten-)Typ ihrer Instanzen: Klassen sind Datentypen.

Eine Klasse kann auch selbst Methoden und Daten enthalten, die unabhängig von Exemplaren aufgerufen werden können. Bekannt ist beispielsweise die Konstante *Pi*, die in vielen Programmiersprachen direkt in der Klasse *Math* definiert ist und sich durch *Math.Pi* aufrufen lässt.

## 2.5 Zusammenfassung

In der Objektorientierung gibt es Objekte und Klassen. Objekte besitzen Attribute, Methoden und eine Identität. Ebenfalls können sie untereinander Botschaften austauschen. Gemeinsamkeiten von Objekten können in Klassen zusammengefasst werden. Objekte, die von Klassen abgeleitet sind, nennt man Instanzen.

---

[9] http://www.oop-uml.de/klasse.php

[10] Lahres, Bernhard & Raýman, Gregor & Strich, Stefan: Objektorientierte Programmierung - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn, 2016; S. 91

# 3 Merkmale der Objektorientierten Programmierung

## 3.1 Abstrakter Datentyp

Ein *Abstrakter Datentyp* beschreibt, "was ein Datentyp an Funktionalität leisten soll, ohne dabei schon festzulegen, wie die Funktionalität zu implementieren ist."[11] Es wird nur die Semantik und die Signatur einer Operation beschrieben, aber noch nicht die Implementierung. Auch die definierten Attribute enthalten noch keine Daten.

In der Objektorientierung bildet jede Schnittstellenklasse (siehe *Arten von Klassen*) einen Abstrakten Datentyp.

## 3.2 Datenkapselung

Ein weiteres wichtiges Merkmal eines Abstrakten Datentyps ist die Datenkapselung: Ein Objekt hat als einziges das Recht, seine Daten zu lesen oder zu bearbeiten - es schützt diese also vor dem Zugriff von außen. Möchte ein anderes Objekt auf die Daten zugreifen, muss es dies über bereitgestellte Schnittstellen, meistens *Getter*- und *Setter-Methoden*, tun.[12] Der innere Zustand des Objekts ist dabei für den Verwender nicht sichtbar. Dieser sieht nur Informationen über die Schnittstelle.

Die Datenkapselung bietet verschiedene Vorteile:

- Ein ungeregelter Zugriff auf die Daten von außen wird verhindert. Der innere Zustand des Objekts bleibt immer gültig, da vor dem Zugriff auf Attribute eine Überprüfung der Änderungen stattfinden kann. So kann zum Beispiel überprüft werden, ob ein eingegebenes Datum auch existieren kann.
- Eine Eigenschaft, die nur für interne Zwecke verwendet wird, ist nicht sichtbar.
- Bei Anpassungen treten keine Nebeneffekte auf, solange nur die interne Implementierung des Objekts und nicht dessen Schnittstellen geändert werden. Gleichzeitig erhöht sich damit die Testbarkeit einer Anwendung.

---

[11] http://cvpr.uni-muenster.de/teaching/ss09/info2SS09/script/Kapitel11-ADT-1.pdf

[12] https://fachinformatiker-anwendungsentwicklung.net/anwendungsentwickler-podcast-6-haeufige-fragen-im-fachgespraech-kapselung/

- Ein Objekt sollte die Implementierung einer Operation nicht nach außen hin zeigen, sondern nur beschreiben, was es tut (*Geheimnisprinzip*).

Wenn ein Kunde in ein Autohaus geht, um sich dort ein Auto zu kaufen, sollte das Objekt *Autohaus* die herstellerinternen Details, zum Beispiel die Art und Weise, wie das Auto produziert wurde oder wie es von der Fabrik in das Autohaus kam - sozusagen die Implementierung der Methode *autoKaufen()* - verbergen, also kapseln.

Manchmal wird als Datenkapselung auch die Eigenschaft bezeichnet, Daten und Logik gemeinsam (in einer Kapsel) zu speichern.

In vielen objektorientierten Programmiersprachen kann Datenkapselung durch die Verwendung von Zugriffsmodifizierern umgesetzt werden.

Es werden im Allgemeinen folgende Sichtbarkeitsstufen unterschieden (vergleiche Tabelle 1):

- Öffentlich (public): Jeder Benutzer des Objekts kann auf dessen Methoden und Daten zugreifen.
- Privat (private): Ein direkter Zugriff von außen wird verhindert. Nur das Objekt selbst (in einigen Programmiersprachen auch Objekte der gleichen Klasse) kann auf diese Methoden oder Daten zugreifen.
- Geschützt (protected): Das Objekt selbst und alle erbenden Objekte können auf die Methoden und Daten des Objekts zugreifen.
- (je nach Programmiersprache auch package, internal und weitere)

In anderen Programmiersprachen wie JavaScript lässt sich Datenkapselung durch die Definition von Eigenschaften in tieferliegenden Gültigkeitsbereiche umsetzen, sodass in darüber liegenden Gültigkeitsbereichen diese Eigenschaften nicht sichtbar sind.[13]

## 3.3 Vererbung

Durch *Vererbung* können spezifische Klassen von einer allgemeinen abgeleitet werden, letztere vererbt also Attribute und Operationen (oder auch Methoden) an die neue

[13] vgl. Wenz, Christian: JavaScript - Grundlagen, Programmieren, Praxis. Galileo Press: Bonn, 2014; S. 156f.

Klasse. Das heißt, eine *Unterklasse* besitzt alle Elemente der allgemeinen *Oberklasse* und kann diese erweitern und neue Elemente hinzufügen. In der Oberklasse können Gemeinsamkeiten der Unterklassen zusammengefasst werden. Es entsteht eine Hierarchie der Klassen.[14]

Ein Beispiel ist die (hier unvollständig aufgeführte) Einteilung von Fahrzeugen in Abbildung 1 (siehe Anhang).

All diese Fahrzeuge haben gemeinsam, dass sie sich fortbewegen können. Sie können auch durch ihre Antriebsart oder ihre Farbe charakterisiert werden. Diese Methoden und Attribute können in der Oberklasse *Fahrzeug* zusammengefasst werden.

Diese Gemeinsamkeiten werden in jeder weiteren Unterklasse konkretisiert und erweitert. So besitzt ein PKW die Eigenschaften und Funktionalitäten, die in der Klasse *Fahrzeug* definiert sind. Es können aber auch Eigenschaften hinzugefügt oder geändert werden.

Die Oberklasse wird durch die Vererbung und das Hinzufügen von Elementen in ihren Unterklassen nicht beeinflusst. Werden aber Elemente in der Oberklasse hinzugefügt, stehen sie automatisch in allen Unterklassen zur Verfügung.

In den meisten Programmiersprachen kann eine Klasse nur von einer Oberklasse erben (Einfachvererbung). Wenige Programmiersprachen, wie beispielsweise C++, unterstützen die Mehrfachvererbung.

### 3.3.1 Arten der Vererbung und Arten von Klassen

In *Schnittstellenklassen* (*interfaces*) wird nur die Schnittstelle definiert, das heißt, Attribute erhalten keine Werte und für Operationen werden keine Implementierungen vorgenommen: Es sind nur *abstrakte* Methoden vorhanden. "Von Schnittstellenklassen können keine Exemplare erstellt werden [...]"[15], weil in einem Objekt alle Operationen implementiert sein müssen. Wenn von Schnittstellenklassen geerbt wird, spricht man von der *Vererbung der Spezifikation*.

---

[14]vgl. Lahres, Bernhard & Raýman, Gregor & Strich, Stefan: Objektorientierte Programmierung - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn, 2016; S. 158

[15] ebenda; S, 170

*Konkrete Klassen* enthalten Implementierungen für all ihrer Methoden, deshalb können Objekte ihres Typs erstellt werden. Erbt eine Klasse von einer konkreten Klasse, erbt sie in fast allen Programmiersprachen gleichzeitig die Implementierung der Methoden.[16] Man spricht von der *Vererbung der Implementierung*.

*Abstrakte Klassen* sind zwischen Schnittstellenklassen und konkreten Klassen einzuordnen: Sie enthalten die Implementierung von einigen Methoden, nicht aber von allen. Deshalb kann auch von ihnen kein Objekt erzeugt werden.

Bei der Vererbung der Spezifikation wird durch die Unterklasse die Schnittstelle der Oberklasse übernommen, also alle Attribute und Operationen inklusive deren Kontrakte. [17]

Bei der Vererbung der Implementierung erben Unterklassen neben der Schnittstelle auch alle bereits implementierten Methoden der Oberklasse, sofern dies nicht durch den Zugriffsmodifizierer *private* eingeschränkt wird. Es lassen sich dadurch Wiederholungen im Quellcode vermeiden, denn gleiche Methoden müssen nicht in Ober- und Unterklasse definiert werden.[18]

Methoden der Oberklassen können nicht nur übernommen, sondern auch *überschrieben*, das heißt abgeändert oder erweitert werden. Diese Methoden der Oberklasse bezeichnet man dann als *virtuell*. Es ist möglich, dass die Methode *starten()* bei einem PKW den String "starte Motor" zurückgibt, während es bei einem Flugzeug "starte Turbine" und bei der generellen Klasse Fahrzeug einfach nur "starten" ist. Auch bei den Methoden findet also eine Konkretisierung statt. Wichtig dabei ist, dass Namen, Typen der Parameter und gegebenenfalls der Rückgabetyp, also die *Signaturen* der Methoden, übereinstimmen müssen.

Weil auf jeden Fall die Schnittstelle geerbt wird, ist ein Objekt der Unterklasse ebenfalls vom Typ der Oberklasse. Einer Methode, welche ein Fahrzeug als Parameter fordert, kann ein Zug, ein PKW oder ein Fahrrad übergeben werden (siehe Polymorphie).

---

[16] vgl. ebenda; S. 169
[17] vgl. ebenda; S. 674
[18] vgl. ebenda; S. 674

### 3.3.2 Prototypen

In vielen objektorientierten Programmiersprachen werden Objekte durch den Konstruktor einer Klasse erzeugt. Die Methoden und Attribute (deren Bedeutung, nicht deren Daten) von Objekten können dann nicht mehr verändert werden.

In der *prototypenbasierten* oder auch *klassenlosen Programmierung* werden Objekte nicht von einer Klasse abgeleitet, sondern durch das Kopieren bereits vorhandener Objekte, der *Prototypen*, erstellt.[19] Dabei werden die Attribute und Funktionen übernommen; diese können aber überschrieben werden. Ebenfalls können neue Eigenschaften hinzugefügt werden.

## 3.4 Polymorphie

Die *Polymorphie* (*Vielgestaltigkeit*) ist das dritte grundlegendes Konzept der Objektorientierung, durch welches eine Modularisierung einer Anwendung und das Ersetzen von einzelnen Modulen erst möglich wird.

Polymorphie bezeichnet die Eigenschaft, dass einer Variable nicht nur ein Objekt eines einzigen (Daten-)Typs, sondern auch Objekte aller Subtypen zugeordnet werden können. Dies wird möglich, weil die Unterklassen die Schnittstelle der Oberklasse erben. Wenn ein Objekt der Basisklasse als Variable, Parameter oder Rückgabetyp erwartet wird, können immer auch Objekte der Subklassen eingesetzt werden.

Wird eine Operation auf einem Objekt der Basisklasse aufgerufen, wird erst zur Laufzeit ermittelt, welche konkrete Methode ausgeführt wird - die der Basisklasse, wenn das Objekt eine Instanz dieser ist, oder die überschriebene Methode einer Unterklasse. Dies nennt man *dynamisches* oder *spätes Binden*.[20] Das Verhalten kann sich also beim Aufruf der gleichen Operation unterscheiden. Dabei gilt: Die Methode der Basisklasse wird nicht aufgerufen, wenn die Unterklasse die Methode selbst implementiert.

---

[19] vgl. Lahres, Bernhard & Raýman, Gregor & Strich, Stefan: Objektorientierte Programmierung - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn, 2016

[20] vgl. http://www.kroening-online.de/Method/Objektorientierung/m_oo.php#Vererbung

Beispiel: Ein Objekt Computer kann zur Laufzeit durch das Objekt Laptop oder Smartphone ersetzt werden. Genauer: Es werden Laptop und Smartphone in einer Objektvariablen des Typs Computer gespeichert.

Durch dynamisches Binden ist es möglich, dass beim Aufruf der Methode *anrufen()* auf dem Laptop Skype, auf dem Smartphone aber die Telefon-App des Betriebssystems geöffnet wird. Die Methode liefert also je nach konkretem Typ ein unterschiedliches Ergebnis.

Ein passendes Beispiel beschreibt auch das Buch *Objektorientierte Programmierung - Das umfassende Handbuch*: "Glühbirnen gibt es in verschiedenen Formen und Gestalten. Da reicht das Repertoire von der 20-Watt-Normalbirne über 150-Watt-Superleuchtbirnen bis zur Energiesparlampe. Wir können diese verschiedenen Formen aber alle an einer ganz definierten Stelle anbringen: in einer dafür vorgesehenen Fassung."[21] Alle Lampen können also in die gleiche Fassung geschraubt werden, obwohl sie am Ende unterschiedlich leuchten.

So kann auch in einem Programm auf dem Computer ein Modul durch eine andere ersetzt werden, solange sie in die dafür vorgesehene Halterung passt, das heißt die Spezifikation der Operation übereinstimmt.

In statisch typisierten Programmiersprachen ist es möglich, dass Operationen mit dem gleichen Bezeichner vorliegen, die sich aber beispielsweise in Anzahl oder Typen der Parameter unterscheiden. Die Auswahl der korrekten Methode bezeichnet man in diesem Fall als *Überladung* oder *statische Polymorphie*.

## 3.5 Vor- und Nachteile der OOP

Nachdem die grundlegenden Konzepte der Objektorientieren Programmierung nun dargelegt wurden, ist es sinnvoll, einige Vor- und Nachteile der Konzepte sowie der OOP an sich aufzuzeigen.

### 3.5.1 Vorteile

[21] Lahres, Bernhard & Raýman, Gregor & Strich, Stefan: Objektorientierte Programmierung - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn 2016

1. Die Objektorientierung ist an die Sichtweise der Menschen angelehnt. Sie ist dadurch nachvollziehbar und überschaubar.
2. Durch die Einheit von Daten und Funktionalitäten im Objekt sowie die Polymorphie wird eine Anwendung modularisiert. Diese einzelnen Module können bei Bedarf einfach angepasst oder ausgetauscht werden.
3. Die Wiederverwendbarkeit wird durch die Modularisierung erhöht. Es können bereits erstellte Module in einer anderen Anwendung genutzt werden, was zu einer Zeitersparnis führt.
4. Die Suche nach Fehlern wird durch die Möglichkeit, einzelne Module zu testen, vereinfacht.
5. Durch Abstraktion und Vererbung der Implementierung entsteht ein geringerer Programmieraufwand.

### 3.5.2 Nachteile und Probleme

1. Der Anspruch, die reale Welt abzubilden, kann zu Problemen führen. So ist es ein Problem, dass ein Quadrat von einem Rechteck oder ein Kreis von einer Ellipse erben müsste (*Kreis-Ellipsen-Problem*).
2. Das dynamische Binden kann zu intransparentem und schwer lesbaren Code führen.

## 3.6 Zusammenfassung

Die drei grundlegenden Konzepte der objektorientierten Programmierung sind Datenkapselung, Vererbung und Polymorphie. Datenkapselung bedeutet, dass ein Objekt seine Attribute und Methoden vor dem Zugriff von außen schützt und diese sich nur über bereitgestellte Schnittstellen lesen oder bearbeiten lassen. Durch Vererbung können Klassen von einer allgemeinen Klasse abgeleitet werden. Dabei wird zwischen Vererbung der Spezifikation und Vererbung der Implementierung unterschieden. Die Implementierung von geerbten Methoden kann überschrieben werden. Durch Polymorphie können einer Variable Objekte unterschiedlicher Typen zugeordnet werden. Beim Aufruf einer Operation wird erst zur Laufzeit entschieden, welche konkrete Methode aufgerufen wird (dynamisches Binden).

# 4 OOP In JavaScript

## 4.1 Geschichte der OOP in JavaScript

Früher war der Umfang von JavaScript-Code häufig nicht sehr groß. Daher sah man es nicht als sinnvoll an, das Konzept der Objektorientierung zu nutzen[22], welches vor allem bei großen Projekten wirksam ist. Mit zunehmender Bedeutung der Skriptsprachen wurden bald aber auch umfangreiche Projekte mit JavaScript realisiert, sodass OOP immer mehr Verwendung fand und auch auf wiederverwendbaren Code oder Erweiterbarkeit geachtet wurde. Bei der Programmierung von Anwendungen beispielsweise für Windows, für welche auch JavaScript eingesetzt werden kann, kommt man an der Verwendung von Objekten nicht vorbei.

Mit ECMAScript 6, der neuen Standardisierung von JavaScript, soll die Verwendung des objektorientierten Paradigmas weiter gefestigt werden, indem Schlüsselwörter wie class oder extends eingeführt werden.[23] Das heißt aber nicht, dass in JavaScript das Klassenkonzept umgesetzt wird, sondern es sind schlicht Vereinfachungen der Syntax: Im Hintergrund wird der prototypenbasierte Code generiert, ähnlich wie es beispielsweise bei *TypeScript* heute schon der Fall ist.

## 4.2 Erstellen eines Objekts

Ein Objekt in JavaScript ist eine unsortierte Liste von primitiven Datentypen (Number, String, Boolean, Undefined, Null). Eigenschaften und Funktionen werden in Schlüssel-Wert-Paaren beschrieben.[24]

### 4.2.1 Erstellen eines Objekts durch Object Literals

Da JavaScript eine klassenlose objektorientierte Sprache ist, können Objekte ohne Klassenzugehörigkeit erstellt werden. Eine Möglichkeit bieten *Object Literals*. Alle Eigenschaften und Funktionen eines Objekts werden in geschweiften Klammern notiert. Schlüssel und Wert werden dabei durch einen Doppelpunkt voneinander getrennt.

---

[22] vgl. Wenz, Christian: JavaScript - Grundlagen, Programmieren, Praxis. Galileo Press: Bonn, 2014; S. 137
[23] vgl. http://es6-features.org/#ClassDefinition
[24] vgl. http://javascriptissexy.com/javascript-objects-in-detail/

```
var objectName = {
      property1 : "data",
      property2 : "data",
      functionName : function() {
          return "Result";
      }
};
```

Der Zugriff auf das Objekt erfolgt durch den Aufruf *objectName.property*:

```
var propValue = objectName.property1 //"data"
```

Es können nach der Definition auch weitere Eigenschaften und Methoden hinzugefügt werden:

```
objectName.property3 = 21;
```

Der Zugriff auf die neuen Eigenschaften gibt deren Werte zurück, diese Eigenschaften wurden also neu angelegt.

Es ist möglich, neben *Strings* auch *Numbers* als Schlüssel zu verwenden. Der Zugriff erfolgt dann wie folgt:

```
var temperature = {
      10 : "too cold",
      20 : "okay",
      24 : "perfect",
      35 : "too hot",
};
var myTemperature = temperature["35"];  //"too hot"
```

Über diese Klammer-Notation kann auch auf Schlüssel, welche Strings sind, zugegriffen werden:

```
var value = objectName["key"];
```

Der Aufruf von Funktionen wird ähnlich gestaltet. Der Unterschied besteht darin, dass jetzt noch eine Parameterliste übergeben wird.

```
var result = objectName.functionName();  //"Result"
```

### 4.2.2 Erstellen eines Objekts durch den Object-Konstruktor

In JavaScript ist alles ein *Object*. Deshalb können Objekte auch durch den Aufruf eines Konstruktors erzeugt werden. Danach werden neue Eigenschaften und Funktionen hinzugefügt.

```
var objectName = new Object();
```

```
objectName.property1 = true;
```

Diese Arten der Objektdefinition sind nur geeignet, wenn man wenige Objekte eines Typs erstellen möchte. Werden allerdings viele Objekte des gleichen Typs benötigt, müsste man für jedes Objekt den gleichen Code nochmal verwenden. Dies ist aufwändig, vor allem, wenn Änderungen nötig sind.

## 4.3 Definition einer Klasse

Indem man in JavaScript Klassen definiert, kann dieses Problem behoben werden. Dabei wird eine Konstruktorfunktion erstellt.

```
functionClassName(param1) {
    this.property1 = param1;
    this.functionName = function() {
        return "Result";
    }
}
```

Bevor Attribute oder Methoden der Klasse angesprochen werden können, müssen Instanzen dieser Klasse erstellt werden. Danach können die Aufrufe erfolgen.

Es können nun mehrere Objekte gleichen Typs instanziiert werden.

```
var objectName1 = new ClassName ("param1", "param2");
var objectName2 = new ClassName ("param1", "param3");
```

Durch this erfolgt der Zugriff auf das aktuelle Objekt. In einem Konstruktor ist es ein Platzhalter für das zu erzeugende Objekt. Nach der Erstellung erhält das neue Objekt die Werte von this.

Es können weitere Attribute und Methoden auch durch den Aufruf der *prototype*-Eigenschaft hinzugefügt werden:

```
ClassName.prototype.property3 = "newAddedProperty3";
```

## 4.4 Datenkapselung

In JavaScript gibt es keine Zugriffsmodifizierer, welche die Sichtbarkeit von Elementen steuern können. Datenkapselung kann aber trotzdem umgesetzt werden, da auf Variablen in einem Objekt, welche nicht mit this, sondern mit var markiert wurden, von außen nicht zugegriffen werden kann. Auf diese privaten Felder kann nur innerhalb der Konstruktorfunktion zugegriffen werden.

```
function ClassName(param1) {
    var property1 = param1; //private Eigenschaft
    this.getProperty1 = function() {
      return property1;
    }
}

var myObject = new ClassName("parameter");
myObject.property1; // undefined
myObject.getProperty1(); //parameter
```

Die Funktion *getProperty1* liegt im gleichen Gültigkeitsbereich wie *property1* und kann deshalb auf dieses Attribut zugreifen. Von außerhalb ist es nicht sichtbar. Es beschreibt eine Funktion, die bei ihrem Aufruf auf die in ihrem Gültigkeitsbereich liegenden Elemente zugreifen.

Methoden, die über die *prototype*-Eigenschaft definiert wurden, haben keinen Zugriff auf private Variablen.

```
ClassName.prototype.publicGetProperty1 = function() {
    return property1;
};

new ClassName().publicGetProperty1();  //"property1" ist
                                       undefiniert
```

## 4.5 Vererbung

Die Vererbung wird in JavaScript durch Prototypen realisiert, da es keine Klassen gibt.

### 4.5.1 Vererbung bei Objekten

Erzeugt man ein neues Objekt aus einem bereits vorhandenen Objekt (dem Prototypen), dann erstellt das neue Objekt eine Referenz auf alle geerbten Attribute und Funktionen des Prototypen.

```
var object1= {a:10, b:20, c:30, d:40};
var object2 = Object.create(object1);
var a = object2.a; //10
```

*object2* erbt das Attribut *a* (und alle anderen) von *object1*.

Wenn beim Aufruf eines Attributs oder einer Methode diese nicht im Objekt selbst vorhanden sind, wird der Aufruf an den darüberliegenden Prototypen weitergereicht.

```
object2.hasOwnProperty("a"); //false
```

### 4.5.2 Vererbung bei Klassen

Soll eine Klasse von einer anderen erben, muss jene ein Objekt der Oberklasse referenzieren. Dazu wird die Prototype-Eigenschaft der Unterklasse aufgerufen.

```
function SuperClass() {
    //...
}
function SubClass() {
    //...
}
SubClass.prototype = new SuperClass();
```

Alle Attribute und Methoden der *SuperClass* stehen auch für Objekte der *SubClass* bereit.

Wenn bei Instanzierungen von Objekten Parameter übergeben werden, wäre es schlecht, in jeder erbenden Klasse alle Parameter in einer eigenen Variable zu speichern, obwohl die Basisklasse auch einige Parameter übernimmt.

```
function SuperClass(a,b) {
  this.a = a;
  this.b = b;
};
function SubClass(a,b,c) {
  BaseClass.call(this, a, b);
  this.c = c;
};
```

Mit der *call*-Funktion kann der Aufruf an die Basisklasse weitergeleitet werden.

## 4.6 Polymorphie

Bei einer dynamisch typisierten Programmiersprache wie JavaScript sind die Auswirkungen der Polymorphie schlechter sichtbar als bei statisch typisierten, da eine Variable jeden beliebigen Typ annehmen kann.

In JavaScript ist eine statische Polymorphie aufgrund des dynamischen Typsystems nicht umgesetzt.

### 4.6.1 Überschreiben von Methoden

In JavaScript lassen sich Methoden überschreiben, indem eine Methode mit dem gleichen Namen in einer Unterklasse erstellt wird. Dabei müssen noch nicht einmal die Anzahl der Parameter übereinstimmen, da die nicht übergebenen Parameter dann einfach *undefined* sind.

```
function Basis() {
    this.myOperation = function() {
        return "Funktion der Basisklasse";
    };
}

function SubA() {
    this.myOperation = function() { //Überschreibe myOperation
        return "Funktion der Subklasse";
    };
}
SubA.prototype = new Basis();

function SubB() {} //myOperation wird nicht überschrieben
SubB.prototype = new Basis();
```

Ruft man nun die Funktion *myOperation* auf Objekten aller drei Klassen auf, werden die Objekte von Basis und SubB " Funktion der Basisklasse" zurückgeben, die von SubA aber "Funktion der Subklasse". Da die Klasse SubB die Funktion nicht überschrieben hat, wird der Aufruf an den Prototypen, die Klasse Basis weitergeleitet.

```
var a = new SubA();
var b = new SubB();
a.hasOwnProperty("myOperation"); //true
b.hasOwnProperty("myOperation"); //false
```

Nur die Klasse *SubA* hat eine eigene Funktion "*myOperation*".

Es ist weiterhin möglich, dass in einer Überschriebenen Methode die Methode des Prototypen aufgerufen wird.

```
this.overrideOperation = function() {
    return "Override: " + Class.prototype.operation();
};
```

### 4.6.2 Virtuelle Methoden

Alle Methoden (und Eigenschaften) einer JavaScript-Klasse können sowohl geerbt als auch überschrieben werden und sind polymorph, sodass man diese als virtuelle Methoden bezeichnen kann.

# 5 ProgrammingWiki

Im ProgrammingWiki werden weitere Beispiele zur Objektorientierung in JavaScript gezeigt. Die Beschreibung des objektorientierten Konzepts findet dort auf einem weniger abstrakten Niveau statt und es werden konkrete Beispiele gezeigt sowie Aufgaben gestellt. Die Gliederung der einzelnen Kapitel im ProgrammingWiki ist ähnlich der Gliederung dieser Komplexen Leistung: Nach einer allgemeinen Einführung werden die Begriffe *Objekt* und *Klasse* definiert. Daraufhin wird auf die grundlegenden Konzepte der OOP, Datenkapselung, Vererbung und Polymorphie, eingegangen. Als Abschlussprojekt soll eine vorgegebene elektrische Schaltung als objektorientierte Anwendung in JavaScript umgesetzt werden.

Das ProgrammingWiki soll sowohl zur Arbeit im Unterricht als auch zum Selbststudium eingesetzt werden können, um die objektorientierte Programmierung zu veranschaulichen. Es kann natürlich noch um einige Punkte ergänzt werden, wie die Prinzipien des objektorientierten Entwurfs (SOLID-Prinzipien), die Beziehungen zwischen Objekten (Aggregation / Assoziation) oder aber auch um einen Überblick über objektorientierte Programmierung in anderen Programmiersprachen.

Die Arbeit im ProgrammingWiki ist unter der URL
http://programmingwiki.de/Objektorientierung_in_JavaScript zu finden.

# 6 Zusammenfassung

In dieser Komplexen Leistung wurde ein Überblick über das Konzept der Objektorientierung, dessen Grundbausteine, die Vor- und Nachteile sowie die Anwendung des Konzepts in JavaScript gegeben.

Die Objektorientierung ist das Paradigma, welches fast nur noch in der Anwendungsentwicklung zu finden ist. Ich denke, dass jeder, der in der Softwareentwicklung tätig ist, Kenntnisse über dieses Konzept hat, denn egal, ob in Programmiersprachen wie Java, C#, C++ oder eben JavaScript - es wird meist nur noch die Objektorientierung verwendet. Deshalb kommt man meist schon beim Erlernen der ersten Programmiersprache nicht daran vorbei.

Vor allem die Modularisierung einer Anwendung bringt meiner Meinung nach erhebliche Vorteile mit sich: Man muss vieles nicht mehr selbst programmieren, sondern kann auf die Klassen anderer Entwickler zurückgreifen. Ein von mir oft verwendetes Paket ist der *HtmlAgilityPack*, welcher mir das Kopfzerbrechen über das "DOM-Parsen" erspart.

# Anhang

## Grafiken und Tabellen

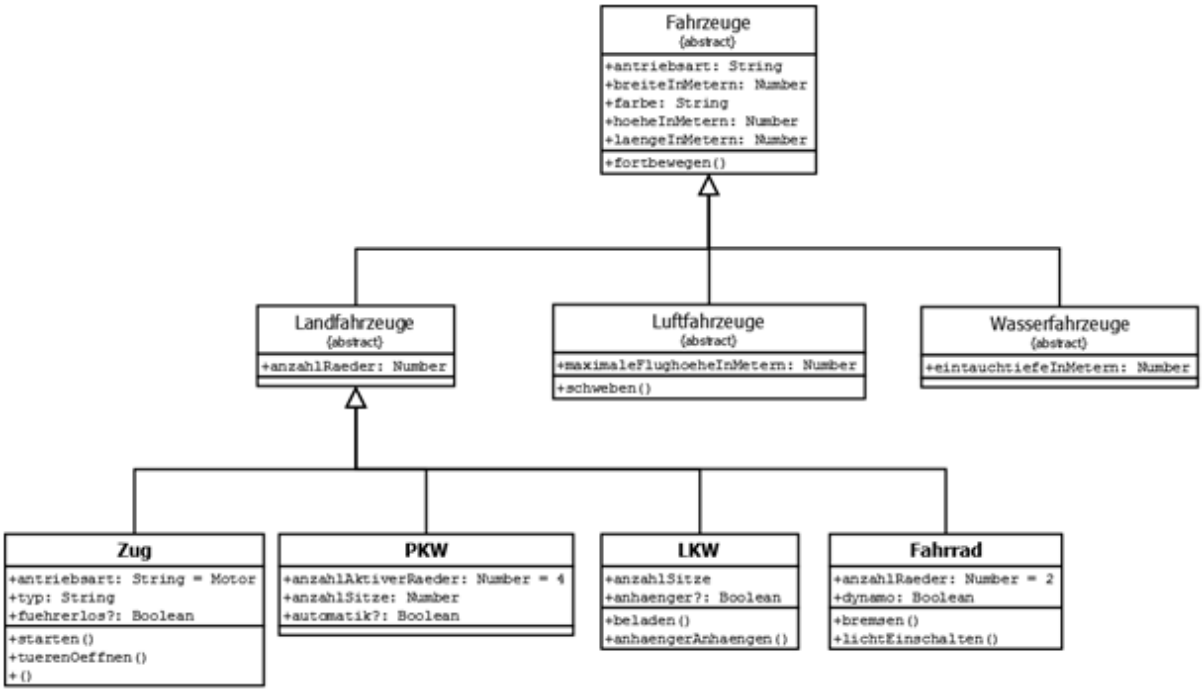


Abbildung 1 - Klassifizierung von Fahrzeugen (UML)

| | Aktuelles Objekt (*this*) | Erbende Objekte | Außenstehende Objekte |
|---|---|---|---|
| private | x | | |
| protected | x | x | |
| public | x | x | x |

Tabelle 1 - Zugriff auf Elemente eines Objekts

# Literaturverzeichnis


1. AL-zami: what is polymorphism in Javascript. In: http://stackoverflow.com/questions/27642239/what-is-polymorphism-in-javascript, zugegriffen am 27.03.2016
2. Bovell, Richard: JavaScript Objects in Detail. In: http://javascriptissexy.com/javascript-objects-in-detail/, zugegriffen am 05.03.2016
3. Bovell, Richard: JavaScript Prototype in Plain Language. In: http://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/, zugegriffen am 04.03.2016
4. Krönig, Rainer: Einstieg in die Objektorientierung. In: http://www.kroening-online.de/Method/Objektorientierung/m_oo.php
5. Kücükyilmaz, Hakan & Haas, Thomas M. & Merz, Alexander: Einsteigen und durchstarten mit PHP5 - Grundlagen, Objektorientierung und PEAR. dpunkt.verlag: 2005
6. Kühnel, Andreas: Visual C# 2012 - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn, 2013
7. Lahres, Bernhard & Raýman, Gregor & Strich, Stefan: Ibjektorientierte Programmierung - Das umfassende Handbuch. Rheinwerk Verlag GmbH: Bonn, 2016
8. Müller, Dietmar: Warum "Objektorientierte Programmierung"?. In: http://www.dietmar-mueller.de/mediapool/18/188364/data/oop_warum.pdf
9. Petri, Björn & Petria, Britta: Objektorientierung. In: http://java-tutorial.org/objektorientierung.html
10. Stefan Macke: Anwendungsentwickler-Podcast #2: Häufige Fragen im Fachgespräch – Objektorientierung. In: https://fachinformatiker-anwendungsentwicklung.net/anwendungsentwickler-podcast-2-haeufige-fragen-im-fachgespraech-objektorientierung/
11. (inforrmatikZentrale) Unbekannt: OOP - Geheimnisprinzip, Kapselung. In: http://www.informatikzentrale.de/oop-geheimnisprinzip-kapselung.html
12. (oop-uml.de) Unbekannt: Objektorientierte Programmierung und Unified Modeling Language - Klasse. In: http://www.oop-uml.de/klasse.php
13. (w3schools.com) Unbekannt: JavaScript Objects. In: http://www.w3schools.com/js/js_object_definition.asp.
14. (w3schools.com) Unbekannt: JavaScript Object prototypes. In: http://www.w3schools.com/js/js_object_prototypes.asp
15. (wikipedia.org) Unbekannt: Objektorientierung. In: https://de.wikipedia.org/w/index.php?title=Objektorientierung&oldid=138591884
16. Wenz, Christian: JavaScript - Grundlagen, Programmieren, Praxis. Galileo Press: Bonn, 2014