



## Schulzentrum Ybbs an der Donau

In Kooperation mit dem HERDT-Verlag stellen wir Ihnen eine PDF inkl. Zusatzmedien für Ihre persönliche Weiterbildung zur Verfügung. In Verbindung mit dem Programm HERDT|Campus ALL YOU CAN READ stehen diese PDFs nur Lehrkräften und Schüler\*innen der oben genannten Lehranstalt zur Verfügung. Eine Nutzung oder Weitergabe für andere Zwecke ist ausdrücklich verboten und unterliegt dem Urheberrecht. Jeglicher Verstoß kann zivil- und strafrechtliche Konsequenzen nach sich ziehen.

---

### Objektorientierte Programmierung mit PHP 7

---

Ralph Steyer

1. Ausgabe, April 2019

ISBN: 978-3-86249-888-8

OOP-PHP7



# Impressum

Matchcode: OOP-PHP7

Autor: Ralph Steyer

Produziert im HERDT-Digitaldruck

1. Ausgabe, April 2019

HERDT-Verlag für Bildungsmedien GmbH  
Am Kümmerling 21-25  
55294 Bodenheim  
Internet: [www.herdt.com](http://www.herdt.com)  
E-Mail: [info@herdt.com](mailto:info@herdt.com)

© HERDT-Verlag für Bildungsmedien GmbH, Bodenheim

Alle Rechte vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder einem anderen Verfahren) ohne schriftliche Genehmigung des Verlags reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Dieses Buch wurde mit großer Sorgfalt erstellt und geprüft. Trotzdem können Fehler nicht vollkommen ausgeschlossen werden. Verlag, Herausgeber und Autoren können für fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.

Wenn nicht explizit an anderer Stelle des Werkes aufgeführt, liegen die Copyrights an allen Screenshots beim HERDT-Verlag. Sollte es trotz intensiver Recherche nicht gelungen sein, alle weiteren Rechteinhaber der verwendeten Quellen und Abbildungen zu finden, bitten wir um kurze Nachricht an die Redaktion.

Die in diesem Buch und in den abgebildeten bzw. zum Download angebotenen Dateien genannten Personen und Organisationen, Adress- und Telekommunikationsangaben, Bankverbindungen etc. sind frei erfunden. Eventuelle Übereinstimmungen oder Ähnlichkeiten sind unbeabsichtigt und rein zufällig.

Die Bildungsmedien des HERDT-Verlags enthalten Verweise auf Webseiten Dritter. Diese Webseiten unterliegen der Haftung der jeweiligen Betreiber, wir haben keinerlei Einfluss auf die Gestaltung und die Inhalte dieser Webseiten. Bei der Bucherstellung haben wir die fremden Inhalte daraufhin überprüft, ob etwaige Rechtsverstöße bestehen. Zu diesem Zeitpunkt waren keine Rechtsverstöße ersichtlich. Wir werden bei Kenntnis von Rechtsverstößen jedoch umgehend die entsprechenden Internetadressen aus dem Buch entfernen.

Die in den Bildungsmedien des HERDT-Verlags vorhandenen Internetadressen, Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen waren zum Zeitpunkt der Erstellung der jeweiligen Produkte aktuell und gültig. Sollten Sie die Webseiten nicht mehr unter den angegebenen Adressen finden, sind diese eventuell inzwischen komplett aus dem Internet genommen worden oder unter einer neuen Adresse zu finden. Sollten im vorliegenden Produkt vorhandene Screenshots, Bezeichnungen bzw. Beschreibungen und Funktionen nicht mehr der beschriebenen Software entsprechen, hat der Hersteller der jeweiligen Software nach Drucklegung Änderungen vorgenommen oder vorhandene Funktionen geändert oder entfernt.

Schulversion

<b>1 Informationen zu diesem Buch</b>	<b>4</b>	<b>5 Vererbung</b>	<b>55</b>
1.1 Voraussetzungen und Ziele	4	5.1 Worum geht es bei Vererbung?	55
1.2 Bevor Sie beginnen ...	7	5.2 Die konkrete Umsetzung der Vererbung in PHP	59
		5.3 Vererbung im Praxisprojekt	61
		5.4 Der Sichtbarkeitsmodifizierer <code>protected</code>	69
<b>2 Das Umfeld von PHP</b>	<b>8</b>	<b>5.5 Überschreiben und verdecken</b>	<b>73</b>
2.1 Webseiten und Webanwendungen	8	5.6 Übungen	76
2.2 PHP	10		
2.3 Übungen	12		
<b>3 Die Idee der objektorientierten Programmierung</b>	<b>15</b>	<b>6 Abstrakte Klassen und Schnittstellen</b>	<b>78</b>
3.1 PHP und OOP	15	6.1 Abstrakte Klassen	78
3.2 Was ist objektorientierte Programmierung?	16	6.2 Schnittstellen	84
3.3 Die Kernkonzepte der Objektorientierung	18	6.3 Abstrakte Techniken im Praxisprojekt	87
		6.4 Übungen	94
<b>4 Klassen, Objekte und Konstruktoren</b>	<b>20</b>	<b>7 Eine objektorientierte Webseite</b>	<b>95</b>
4.1 Eine Klasse – „Bauplan“ für Objekte	20	7.1 Die Weiterentwicklung des Praxisprojekts	95
4.2 Klassen in PHP deklarieren	21	7.2 Die Programmdatei	98
4.3 Eigenschaften	23	7.3 Übung	100
4.4 Methoden	24		
4.5 Der Zugriff auf Instanzelemente	28	<b>8 Objektorientierter Datenbankzugriff</b>	<b>101</b>
4.6 Datenkapselung, Getter- und Setter-Methoden	29	8.1 PHP und Datenbanken	101
4.7 Objekte erzeugen und der Konstruktor	31	8.2 Die moderne Art des Datenbankzugriffs – PDO	103
4.8 Ein erster OO-Aufbau einer Webseite	36	8.3 Abfragen senden	107
4.9 Objekte löschen – Garbage Collector und Destruktor	38	8.4 Prepared Statements	114
4.10 Objekte klonen	43	8.5 Das Praxisprojekt mit Datenbanken	117
4.11 Anonyme Klassen	47	8.6 Fazit und Abschlussbemerkungen	119
4.12 Klassenmember und der Gültigkeitsbereichsoperator	47	8.7 Übung	120
4.13 Autoloading	52		
4.14 Übungen	54	<b>A Anhang: Installationen und Quellangaben</b>	<b>122</b>
		A.1 Programme und Tools	122
		A.2 Programmieren und Debuggen mit Xdebug und PDT	131
		A.3 Quellangaben im Internet	140
		<b>Stichwortverzeichnis</b>	<b>141</b>

# 1

## Informationen zu diesem Buch

### 1.1 Voraussetzungen und Ziele

#### Zielgruppe

- ✓ Studenten und Auszubildende in IT-Berufen
- ✓ Schüler mit IT-Schwerpunkt
- ✓ Fortgeschrittene PHP-Einsteiger und Web-Programmierer mit PHP-Kenntnissen
- ✓ Erfahrenere PHP-Entwickler älterer PHP-Versionen (ohne OO)
- ✓ Fortgeschrittene Entwickler anderer Programmiersprachen

#### Empfohlene allgemeine Vorkenntnisse

- ✓ Grundkenntnisse im Umgang mit Windows, Linux oder MacOS
- ✓ Grundkenntnisse in der Bedienung von Anwendungsprogrammen
- ✓ Grundkenntnisse im Umgang mit dem Internet
- ✓ Allgemeines Grundwissen in Programmierung
- ✓ Grundlagen-Kenntnisse in PHP

#### Erforderliche PHP-Grundkenntnisse

- ✓ Grundlegende Sprachelemente
- ✓ Variablen und Operatoren; Kontrollstrukturen
- ✓ Felder (Arrays); Funktionen
- ✓ Kenntnisse von Super-Globals (standardmäßig in PHP vorhandene Variablen)
- ✓ Interaktive Webseiten (Auswertung von Formularen mithilfe von PHP)
- ✓ Arbeit mit externen Dateien; Umgang mit Sessions
- ✓ Zeichenkettenfunktionen
- ✓ Klassische Anwendungen von PHP wie der Umgang mit Datums- und Zeitfunktionen

## Lernziele

Lernziele der Unterlage sind das Erarbeiten von objektorientierter Programmierung mit PHP und deren Anwendung in einigen Praxis-Situationen. Folgende Themen werden behandelt:

- ✓ Sie verstehen den Ansatz der objektorientierten Denkweise und der objektorientierten Programmierung.
- ✓ Sie erlernen die einzelnen Techniken und Terminologien der objektorientierten Programmierung.
- ✓ Sie wissen, wie man mit PHP die objektorientierte Programmierung umsetzen kann.
- ✓ Sie können einschätzen, wann ein objektorientierter Ansatz bei PHP sinnvoll ist und wann man hybrid oder gänzlich funktional bzw. prozedural vorgehen sollte.
- ✓ Sie lernen verschiedene objektorientierte PHP-Anwendungen wie etwa den Datenbankzugriff mit PDO (PHP Data Objects) und die Umsetzungen einer kompletten Webseite mit OOP kennen.

## Hinweise zur Hard- und Software

Als Hardwarebasis wird ein normaler PC vorausgesetzt. Im vorliegenden Buch werden eine Windows-10-Umgebung sowie lokale Installationen der folgenden kostenfreien Software verwendet:

- ✓ XAMPP (Webserver, PHP, MySQL/MariaDB; <http://www.apachefriends.org>)
- ✓ Notepad++ (Editor; <http://notepad-plus.sourceforge.net>)
- ✓ FileZilla (FTP-Client; <http://filezilla-project.org>)
- ✓ Verschiedene Standardbrowser (Chrome, Firefox, Edge bzw. Internet Explorer)

Eine Kurzübersicht sowie eine Installations- und Konfigurationsanleitung für die genannte Software finden Sie im Anhang. Alternativ können Sie die verwendeten PHP-Skripte auch mit funktionsähnlicher alternativer Software erstellen und testen. Neben dem im Buch als Vorgabe verwendeten Betriebssystem werden Linux und MacOS X bei Bedarf ebenso berücksichtigt. Bei einigen Ausblicken werden ebenso bei Bedarf verschiedene weitere Programme und Tools (insbesondere Entwicklungstools bzw. IDEs) verwendet oder erwähnt.

## Anmerkungen zur PHP-Version

Die Basis der Unterlage ist PHP in der Version 7 (7.2.x). Bemerkenswert an der Version 7 ist, dass damit verschiedene Elemente der vorherigen Versionen nicht nur als deprecated – veraltet – erklärt, sondern explizit abgeschafft wurden. Üblicherweise enthalten neue Sprachversionen auf der einen Seite gewisse Erweiterungen. Auf der anderen Seite erklären neue Versionen einer Programmiersprache fast immer gewisse vorhandene Elemente als „deprecated“. Das bedeutet, dass diese zwar nicht mehr eingesetzt werden sollen, aber prinzipiell noch funktionieren.

Wenn man hingegen in einer neuen Sprachversion Elemente explizit **entfernt**, ist solch ein Update nicht mehr zu 100 % abwärtskompatibel. Das bedeutet, dass Quellcode, der mit Sprachelementen einer älteren Version erstellt wurde, unter der neuen Version u. U. nicht mehr funktioniert. Da solch ein Schritt massive Auswirkungen auf bestehende Projekte hat, wird so eine Maßnahme nur selten durchgeführt. Um PHP mit der Version 7 auf einen sicheren, stabilen und professionellen Stand zu heben, wurde dieser radikale Schritt jedoch notwendig. Vorherige Updates hatten diesen Schritt bereits angedeutet bzw. vorbereitet.

Zwei der wichtigsten Veränderungen bei PHP von der vorherigen Version 5.6 auf die Version 7 waren die Abschaffung der MySQL-Schnittstelle (ersetzt durch MySQLi und vor allen Dingen PDO) sowie mehrerer Funktionen im Umgang mit regulären Ausdrücken.

## Typografische Konventionen

### Hervorhebungen im Text

Im Text erkennen Sie bestimmte Programmelemente an der Formatierung. So werden z. B. Bezeichnungen für Programmelemente wie Register immer *kursiv* geschrieben und wichtige Begriffe **fett** hervorgehoben.

<i>Kursivschrift</i>	kennzeichnet alle von Programmen vorgegebenen Bezeichnungen für Schaltflächen, Dialogfenster, Symbolleisten, Menüs bzw. Menüpunkte (z. B. <i>Datei - Schließen</i> ) sowie alle vom Anwender zugewiesenen Namen wie Dateinamen, Ordnernamen, eigene Symbolleisten, Hyperlinks und Pfadnamen.
Courier New	kennzeichnet Programmtext, Programmnamen, Funktionsnamen, Variablennamen, Datentypen, Operatoren etc.
<i>Courier New kursiv</i>	kennzeichnet Zeichenfolgen, die vom Anwendungsprogramm ausgegeben oder in das Programm eingegeben werden.
[ ]	Bei Darstellungen der Syntax einer Programmiersprache kennzeichnen eckige Klammern optionale Angaben.
	Bei Darstellungen der Syntax einer Programmiersprache werden alternative Elemente durch einen senkrechten Strich voneinander getrennt.

## 1.2 Bevor Sie beginnen ...

### **HERDT** BuchPlus – unser Konzept:

Problemlos einsteigen – Effizient lernen – Zielgerichtet nachschlagen

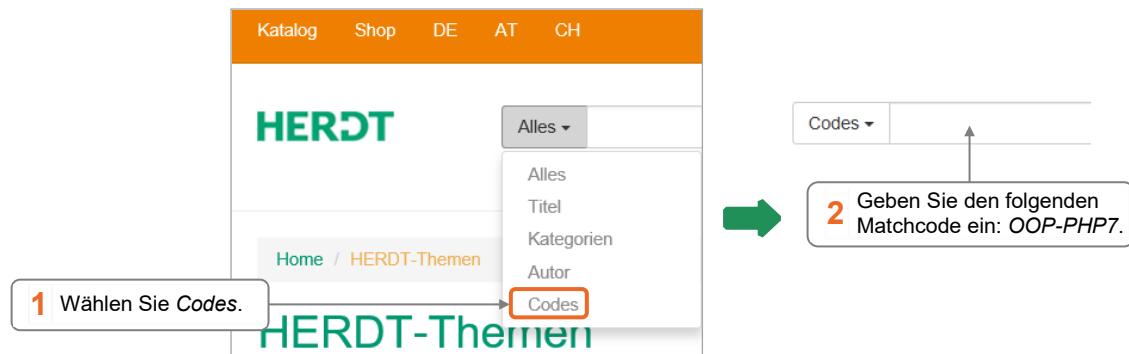
(weitere Infos unter [www.herdt.com/BuchPlus](http://www.herdt.com/BuchPlus))

Nutzen Sie dabei unsere maßgeschneiderten, im Internet frei verfügbaren Medien:



So können Sie schnell auf die BuchPlus-Medien zugreifen:

- Rufen Sie im Browser die Internetadresse [www.herdt.com](http://www.herdt.com) auf.



# 2

## Das Umfeld von PHP

### 2.1 Webseiten und Webanwendungen

PHP findet seine hauptsächliche Anwendung im Umfeld des WWW (World Wide Web) und bei der Erstellung von Inhalten für dieses WWW. Es ist aus Webseiten und Webanwendungen aufgebaut.

#### Statische und dynamische Webseiten

Eine Webseite ist im Grunde nur ein Klartextdokument im WWW, das mit einem Browser (Client) unter Angabe eines Uniform Resource Locators (URL) von einem Webserver angefordert werden kann. Der Webserver und der Browser „wissen“ jedoch auf Grund des verwendeten **MIME-Typs** (Multipurpose Internet Mail Extensions), dass so ein Textdokument eine spezielle Bedeutung hat. Der MIME-Type oder auch Internet Media Type klassifiziert bei der Übertragung von Daten per http diese im Rumpf der Datenpakete.

Der MIME-Type besteht aus zwei Teilen:

- ✓ der Angabe eines allgemeinen Medientyp
- ✓ der Angabe eines Subtyps.

Beide Angaben werden durch einen Schrägstrich voneinander getrennt.

Im Fall einer Webseite sieht das so aus:

text/html

Der innere Rahmen des Dokuments besteht bei einer Webseite aus HTML (Hyper Text Markup Language), weshalb in diesem Zusammenhang auch von einer HTML-Seite oder einem HTML-Dokument gesprochen wird.

Bei klassischen oder **statischen Webseiten** werden unter Verwendung von HTML-Befehlen (Tags) fest vorgegebene Texte, Grafiken, Hyperlinks etc. in der Webseite verbunden. So eine statische Webseite gestattet dann meist nur das Betrachten dieser Inhalte (unter Umständen durch Multimedia erweitert) und die Verzweigung über ebenfalls statische Hyperlinks. Eine eventuelle Kommunikation mit dem Webserver erfolgt in der Regel über HTML-Formulare und die CGI-Schnittstellenvereinbarung (Common Gateway Interface).

Im Gegensatz zu statischen Webseiten, die bereits auf dem Webserver als fertige HTML-Datei vorliegen, werden **dynamische Webseiten** häufig erst im Moment der Anforderung durch den Client auf dem Webserver erzeugt (etwa mittels PHP). Das so zunächst dynamisch generierte HTML-Dokument wird dann vom Webserver an den Browser übertragen und dort angezeigt, wobei der Benutzer in der Regel dieses jedoch nicht mehr von einem statischen HTML-Dokument unterscheiden kann.

Der Begriff der „Dynamik“ bezieht sich also in diesem Zusammenhang auf die Erstellung von Inhalten über Programmierung auf dem Webserver, nicht auf das Verhalten der Webseite im Client.

## Webanwendungen

**Webanwendungen**, häufig auch Webapplikationen genannt, sind Anwendungen beziehungsweise Programme, die zu einem gewissen Teil oder auch ausschließlich auf einem Webserver ausgeführt werden. Die Darstellung der Informationen und Interaktion mit dem Anwender erfolgt jedoch auch hier über den Browser. Bei modernen Webanwendungen wird aber meist ebenfalls im Browser (also im Client) programmiert (in der Regel mit JavaScript). Obwohl es prinzipiell möglich ist, dass man bei Webanwendungen sogar ausschließlich im Client programmiert, ist auch das eher selten. Bei modernen Webapplikationen ist die Programmierung fast immer auf Server und Client verteilt.

Logik, die auf den Server gehört, betrifft kritische Schritte oder nur zentralisiert verwaltbare Dinge, wie:

- ✓ Verifizierung von Benutzerdaten
- ✓ Datenbankabfragen
- ✓ Verwaltung von Benutzereingaben
- ✓ Verwaltung von Sessions

Die clientseitige Programmierung betrifft im Wesentlichen die Interaktion mit dem Anwender, wie:

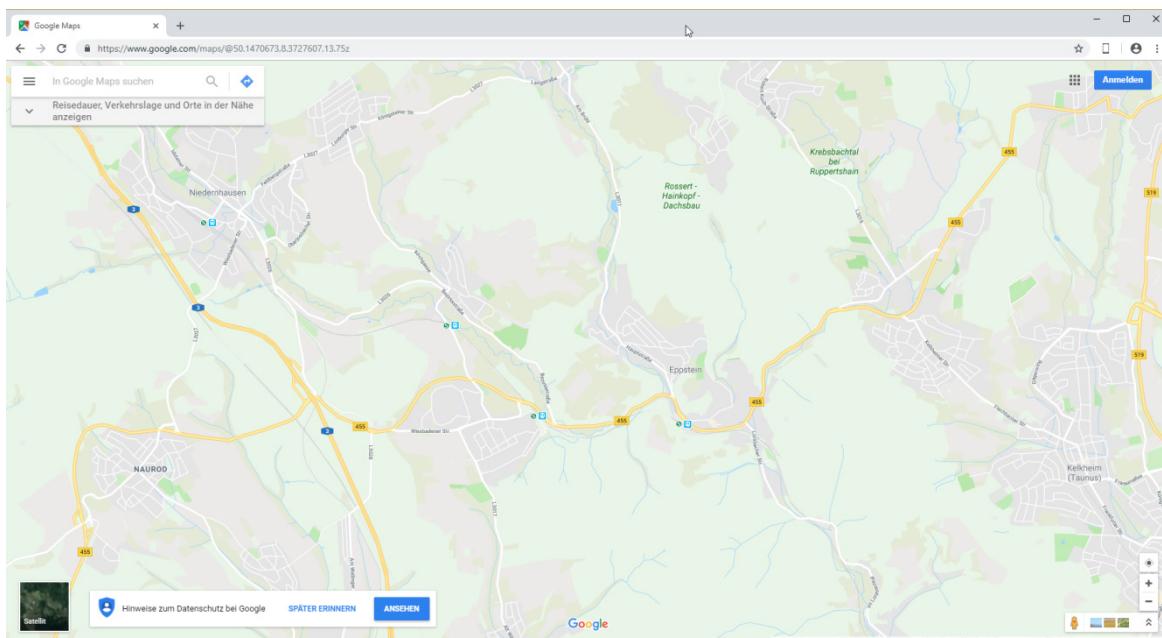
- ✓ Anzeigen von Fehlern bei der Benutzereingabe und allgemeine Benutzerführung
- ✓ Animationseffekte wie das harmonische Aufblenden von Menüs
- ✓ Übermitteln von Daten im Client an den Server (Bildschirmauflösung, verwendetes Gerät, Standort des Benutzers, etc.) – oft auch unbemerkt vom Anwender im Hintergrund

Die Datenübertragung zwischen dem Client und dem Server findet aber auch bei Webanwendung wie im WWW üblich auf der Transportebene auf Basis von TCP/IP (Transmission Control Protocol/Internet Protocol) und auf Anwendungsebene durch das Protokoll HTTP (Hypertext Transfer Protocol) oder dessen sicherer, verschlüsselter Version HTTPS (Hypertext Transfer Protocol Secure) statt. Die räumliche Entfernung zwischen dem Client und dem Webserver ist für eine Webanwendung ohne Bedeutung, denn beide sind über das Internet oder ein lokales Netzwerk miteinander verbunden.

In umfangreichen, professionellen Umgebungen werden die serverseitigen Programmierschritte der Webanwendungen oft nicht mehr direkt auf dem Webserver ausgeführt, sondern auf zusätzlichen **Applikationsservern**. Die Anwendungsdaten werden in der Regel über **Datenbankmanagementsysteme** zur Verfügung gestellt.

## Rich Internet Application (RIA)

In den letzten Jahren hat sich eine Gruppe an besonderen Webanwendungen etabliert, die in ihren Interaktionsmöglichkeiten und oft auch im gesamten Look & Feel wie Desktop-Anwendungen erscheinen. Diese werden **Rich Internet Applications** (RIAs) genannt. Sie bieten innerhalb des Browsers Interaktionsmöglichkeiten mit dem Anwender wie beispielsweise Drag & Drop oder komplexe Menüs. Oder sie sind grafisch besonders anspruchsvoll, sodass man sie von der Optik kaum von klassischen Desktop-Anwendungen unterscheiden kann. Sehr oft tauschen RIAs auch im Hintergrund Daten mit dem Webserver aus, wobei hier sehr oft eine Technologie namens **Ajax** (Asynchronous JavaScript And XML) zum Einsatz kommt.



Google Maps ist eine typische RIA

## 2.2 PHP

PHP stand ursprünglich für „Personal Homepage Tools“, steht aber mittlerweile für „PHP: Hypertext Preprocessor“. Das ist ein sogenanntes rekursives Akronym, da der Bezeichner „PHP“ in der ausgeschriebenen Version wiederholt wird. PHP wird überwiegend bei serverseitiger Web-Programmierung eingesetzt, ist aber darauf nicht beschränkt.

Die Geschichte von PHP geht bis in das Jahr 1995 und den von Rasmus Lerdorf entwickelten und zunächst nur auf Perl basierenden PHP/FI (Personal Home Page/Forms Interpreter) zurück. Derzeit ist die Version PHP 7 aktuell, die eine deutlich bessere Performance im Vergleich zu den Vorversionen aufweist.

Für die Popularität von PHP gibt es mehrere Gründe, u. a. diese:

- ✓ Der Einstieg in PHP ist einfach.
- ✓ PHP ist leicht zu benutzen, denn PHP-Befehle können einfach in umgebende HTML-Strukturen eingebettet werden. Diese Vorgehensweise wird manchmal SSI (Server Side Includes) genannt, wobei der Begriff im engen Sinn eine eigenständige Technologie beschreibt, die aber als Namensgeber gerne verwendet wird.
- ✓ PHP ist Open-Source und nicht zuletzt deshalb stellen nahezu alle Webhoster Angebote mit PHP-Unterstützung zur Verfügung.
- ✓ PHP kann auf allen gängigen Betriebssystemen verwendet werden.
- ✓ PHP unterstützt viele verschiedene Datenbanksysteme und hat auch für ziemlich jede denkbare Anwendung eine riesige Standardbibliothek.
- ✓ Es gibt eine große und aktive PHP-Community.
- ✓ Zu vielen Aufgabenstellungen gibt es bereits fertige PHP-Skripte.
- ✓ PHP hat sich auf die Programmierung von dynamischen Webseiten und Webanwendungen spezialisiert und bietet hierzu sehr viele leistungsfähige Features.
- ✓ Mit PHP kann man sowohl prozedural als auch objektorientiert programmieren.

## Funktionsweise von PHP

Obwohl Grundlagen in PHP vorausgesetzt werden, soll im Folgenden kurz behandelt werden, wie PHP-Dateien bei einer Verwendung im Web grundsätzlich ausgeführt werden. Der Vorgang ist so gut wie immer gleich:

- ✓ Im ersten Schritt sendet der Client (Browser) per HTTP(S) eine Anfrage nach einer PHP-Datei an den Webserver (der sogenannte Request).
- ✓ Der Webserver identifiziert anhand der Dateinamenerweiterung die Anfrage als PHP-Request und sucht die angeforderte PHP-Datei im Dateisystem.
- ✓ Die PHP-Datei wird zur weiteren Verarbeitung an den PHP-Interpreter übergeben, der für den Server konfiguriert ist.
- ✓ Der PHP-Interpreter führt im nächsten Schritt zunächst eine Syntaxanalyse durch. Der Interpreter zerlegt dabei den Quelltext der Datei und bringt ihn in eine für die Ausführung geeignete Form. Anschließend wird der PHP-Code ausgeführt. Ggf. werden zusätzliche Programme an der Verarbeitung beteiligt (etwa ein Datenbankmanagementsystem – DBMS).
- ✓ Das Ergebnis liefert der Interpreter zurück an den Webserver.
- ✓ Der Webserver sendet die so dynamisch generierte Antwort (Response) an den Client zurück. Das Ergebnis wird meist ein dynamisch erzeugtes HTML-Dokument sein, das zwar die Dateierweiterung `.php`, aber den MIME-Type `text/html` hat. Es kann sich aber auch um ein PDF-Dokument, eine Grafik oder eine Datei eines anderen Formats handeln.



Der PHP-Interpreter auf einem Windows-System

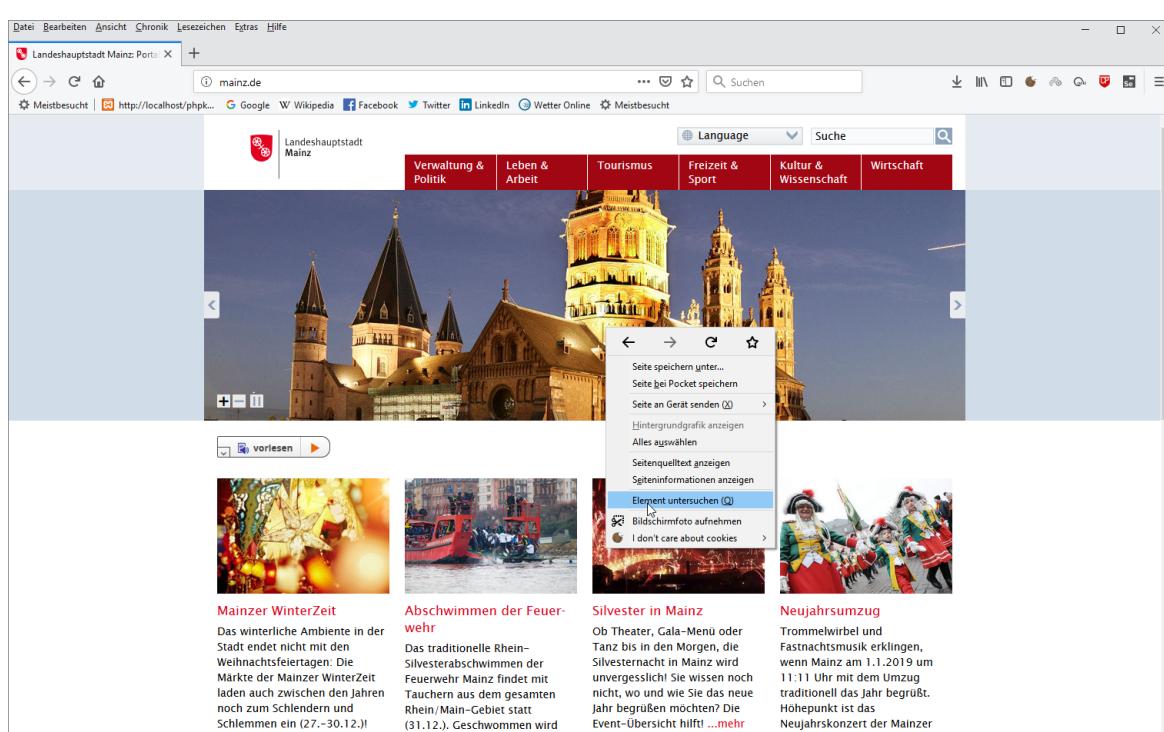
## 2.3 Übungen

**Basisdateien:** --

**Ergebnisdateien:** --

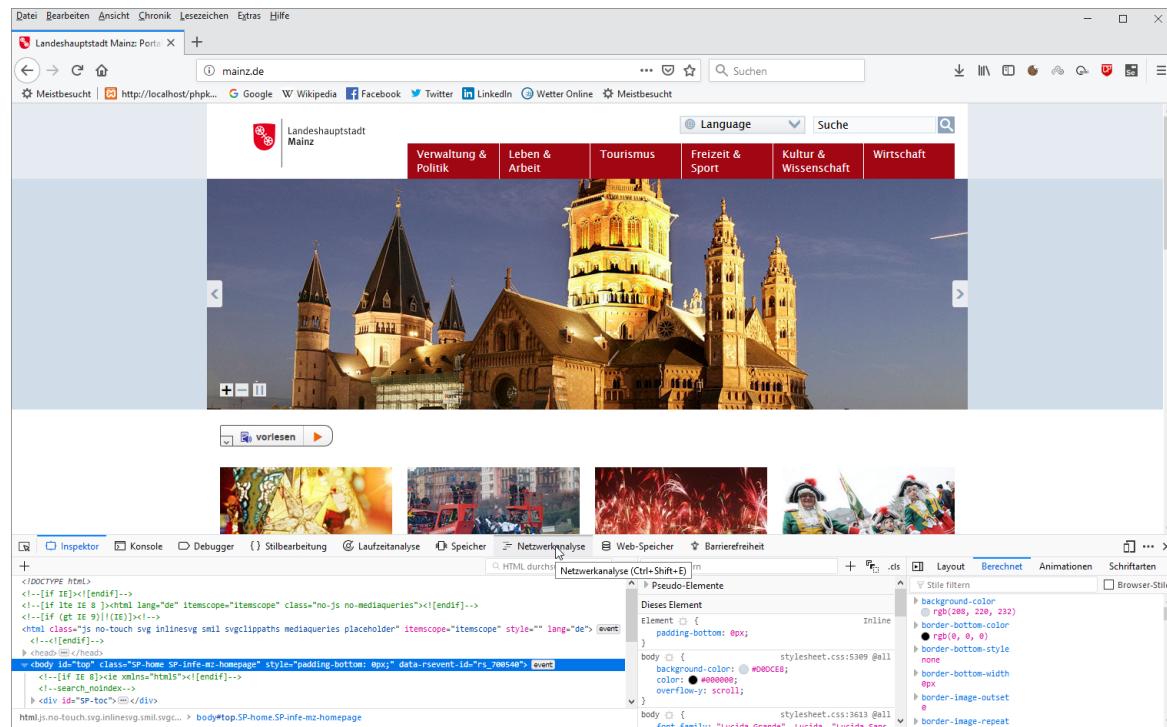
1. Öffnen Sie einen modernen Browser.
2. Laden Sie eine populäre Webseite, z. B. die einer Landeshauptstadt oder <http://www.herdt.com>.
3. Klicken Sie mit der rechten Maustaste in die Webseite. Im Kontextmenü erkennen Sie einen Befehl zum Untersuchen der Webseite. Der Befehl steht mittlerweile in allen modernen Browsern zur Verfügung, wird aber unterschiedlich benannt. In Firefox oder Edge nennt er sich „Element untersuchen“, in Chrome „Untersuchen“.

Alternativ können Sie aber auch die Taste **F12** verwenden.



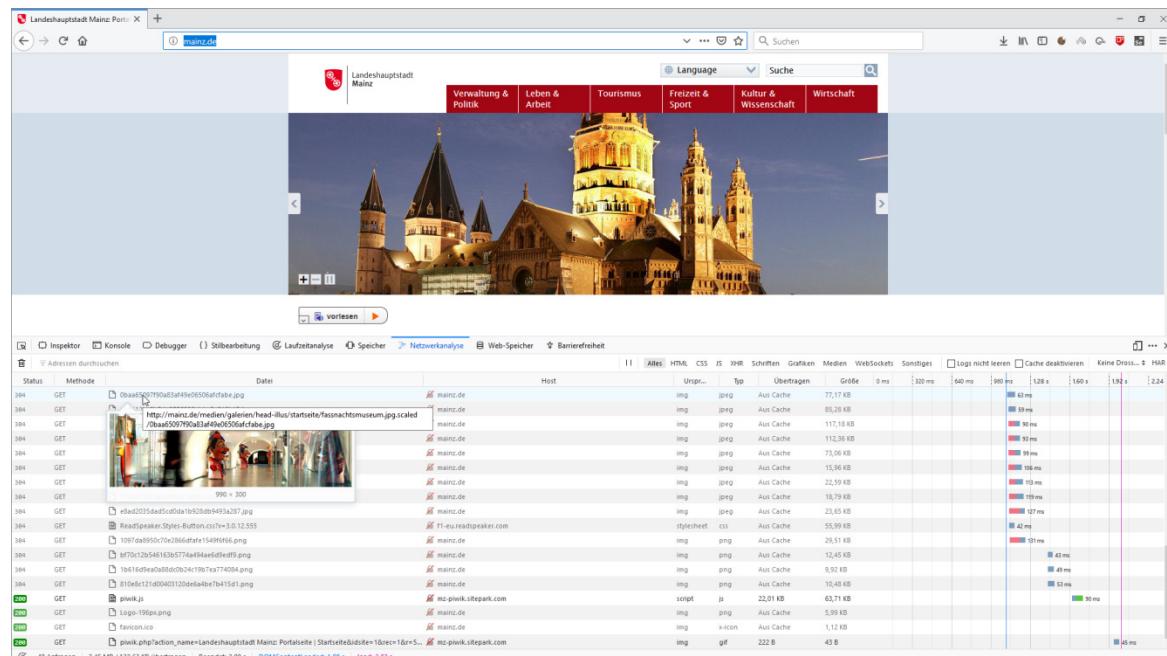
*In jedem modernen Browser gibt es die Möglichkeit, eine Seite zu untersuchen*

4. Innerhalb des Browserfensters werden nun verschiedene Bereiche angezeigt, mit denen man verschiedenste Facetten einer Webseite untersuchen kann. Etwa den tatsächlich verwendeten Quellcode, Fehler, die Style Sheets oder auch die übertragenen Daten.
5. Betrachten Sie die verschiedenen Bereiche.



### Die Hintergründe einer Webseite

6. Wählen Sie die Netzwerkanalyse (Network) aus. Das ist ein sogenannter Sniffer, mit dem Sie die Kommunikation zwischen dem Browser und dem Webserver analysieren können.
7. Laden Sie mit **F5** die Webseite neu. Sie sehen dann alle Dateien, die von der Webseite geladen und für den Aufbau des Angezeigten wirklich verwendet werden.



### Auswahl des integrierten Sniffers und Neuladen der Seite

8. Klicken Sie eine der Dateien an. Sie werden dann die genauen Informationen zu dieser Datei bekommen – etwa unter dem Content-Type des in diesem Kapitel vorgestellten MIME-Type.

The screenshot shows the Network tab of a browser's developer tools. At the top, there are several tabs: Verwaltung & Politik, Leben & Arbeit, Tourismus, Freizeit & Sport, Kultur & Wissenschaft, and Wirtschaft. Below these tabs, there is a search bar and a language selection dropdown set to English (en). The main content area displays a complex image, likely a historical manuscript or map, with various text snippets and illustrations. Below the image, there are four small thumbnail images representing different sections of the site.

The Network tab lists 48 requests:

Status	Methode	Daten	Host	Umspr...	Typ	Übertragen	Größe	0 ms	100 ms	540 ms	990 ms	128 s	180 s	190 s	2,24 s	
304	GET	Oban505790a3b3a49fe0650fa...	mainz.de		img	jpg	Aus Cache	77,17 kB								
304	GET	59b1a12bc1b2e32e3560babab...	mainz.de		img	jpg	Aus Cache	85,28 kB								
304	GET	569a66d64ca7c7a5d6e0e1e3f3...	mainz.de		img	jpg	Aus Cache	117,15 kB								
304	GET	4a3a439362699f28fb30e3a460...	mainz.de		img	jpg	Aus Cache	112,36 kB								
304	GET	94944a4a48136641b1ba4a371...	mainz.de		img	png	Aus Cache	73,06 kB								
304	GET	2660025265d9f1c07b50a8e792...	mainz.de		img	jpeg	Aus Cache	15,96 kB								
304	GET	deaf512f204a48796107a61914...	mainz.de		img	jpeg	Aus Cache	22,59 kB								
304	GET	174439110e02629a119d51a2...	mainz.de		img	jpeg	Aus Cache	16,79 kB								
304	GET	e1e02021aa5e050da1926d954...	mainz.de		img	jpeg	Aus Cache	23,65 kB								
304	GET	ReadSpeaker.Style.Button.css...	11-eu.readspeaker.com		stylesheet	css	Aus Cache	55,99 kB								
304	GET	1097ca3a50c7e2666d9afe149...	mainz.de		img	png	Aus Cache	29,51 kB								
304	GET	5f7f212b54513b3774e49aee4...	mainz.de		img	png	Aus Cache	12,45 kB								
304	GET	18616fd9a0a80d0b2419b7e...	mainz.de		img	png	Aus Cache	9,92 kB								
304	GET	611a6d121a040120a4eab7e...	mainz.de		img	png	Aus Cache	10,40 kB								
200	GET	pixik.js	mc.pixik.sitespark.com	xscript	js	js	22,81 kB	63,71 kB								
200	GET	Logo-196px.png	mainz.de		img	png	Aus Cache	5,99 kB								
200	GET	favicon.ico	mainz.de		img	x-icon	Aus Cache	1,12 kB								
200	GET	pixik.php?action=name=Land...	mc.pixik.sitespark.com	img	gif	222 B	43 B									

Details for the selected request (pixik.js):

- Anfragekopfzeilen (433 B):
  - Accept: \*/\*
  - Accept-Encoding: gzip, deflate
  - Accept-Language: de,en-US;q=0.7,en;q=0.3
  - Cache-Control: no-cache
  - Connection: keep-alive
- Anfragegekodete Zeichen (433 B):
  - Content-Type: application/javascript
  - Date: Fri, 28 Dec 2018 07:54:21 GMT
  - Elap: 7e4d56ed01049e307-gzip
  - Keep-Alive: timeout=5, max=100
  - Last-Modified: Fri, 07 Jun 2018 15:14:30 GMT
  - Server: Apache
  - Vary: Accept-Encoding

Untersuchung einer Datei

# 3

## Die Idee der objektorientierten Programmierung

### 3.1 PHP und OOP

Noch heute programmiert man in PHP in der Regel prozedural beziehungsweise strukturiert oder zumindest nicht streng objektorientiert. Oftmals werden OOP-Techniken und klassische Techniken aus der prozeduralen beziehungsweise strukturierten Welt vermischt. Das ist sicher vor allen Dingen historisch bedingt, denn PHP wurde am Anfang sehr einfach und rein prozedural konzipiert. Der Fokus lag auf der Spezialisierung von Webapplikationen, wie etwa auf der möglichst einfachen Zusammenarbeit mit clientseitigen Formularen aus einem Browser und serverseitigen Datenbanken.

Im Laufe der Zeit wurde PHP aber sukzessive um die Möglichkeit der objektorientierten Programmierung erweitert – erst nur mit wenigen Teilespekten, nach und nach kamen so gut wie alle wichtigen Elemente der OOP hinzu. Modernes PHP unterstützt nun mehrere Programmierparadigmen.

Bereits mit der Version 4 haben einzelne objektorientierte Merkmale in PHP Einzug gehalten und seit PHP 5 sind fast alle modernen Konzepte der objektorientierten Entwicklung möglich. So kann man heutzutage objektorientierte Programmierung in PHP nutzen, muss es aber nicht. In PHP 7 können jedoch einige Dinge nur noch objektorientiert optimal umgesetzt werden.

Aber auch heute noch wird in PHP, wie erwähnt, oft entweder rein prozedural beziehungsweise strukturiert programmiert oder aber es werden lediglich Objekte und einzelne Techniken aus der OOP eingesetzt; ein Projekt wird aber nicht wirklich objektorientiert umgesetzt.

Dafür gibt es Gründe, die durchaus ihre Berechtigung haben:

- ✓ Das Gerüst einer Webseite oder Webapplikation ist reines HTML und PHP wird lediglich an einigen Stellen eingebaut. Damit ist der äußere Rahmen schon explizit nicht objektorientiert.
- ✓ OOP ist nicht trivial. Das Wissen um die Konzepte und Techniken der OOP ist oft bei PHP-Programmierern nicht vorhanden. Und da es meist auch andere Möglichkeiten gibt, ist der Druck nicht vorhanden, sich in die OOP einzuarbeiten.

- ✓ Leider nutzt PHP bei der Notation der objektorientierten Programmierung im Vergleich zu den meisten anderen OO-Sprachen eine oft abweichende Syntax. Das erschwert Umsteigern den Einstieg, wobei sich erfahrene OO-Programmierer bereits nach kurzer Zeit zurechtfinden werden.
- ✓ Viele Projekte existieren schon länger und verwenden Code, der mit PHP-Versionen erstellt wurde, die noch nicht oder nicht vollständig OOP unterstützt haben.
- ✓ Der Aufwand für eine vollständige objektorientierte Umsetzung einer Aufgabenstellung ist meist höher als bei einem klassischen Ansatz.
- ✓ Viele Aufgaben eines PHP-Skripts bestehen aus der Abarbeitung einer Folge von Anweisungen, die einfach sequenziell ablaufen sollen, z. B. die Entgegennahme von Daten eines Nutzers, die darauf folgende Abfrage in der Datenbank und letztendlich das Senden einer aufbereiteten Antwort. Das Erzeugen von Objekten ist unnötiger Overhead, der möglicherweise sogar die Performance negativ beeinflusst oder den Quellcode unnötig aufbläht.

Trotzdem gibt es unumstößliche Vorteile der OOP, denn sonst hätte sich diese nicht durchgesetzt. Und das gilt auch für PHP-Projekte. Um diese Vorteile zu sehen, muss erst einmal geklärt werden, was OOP ist.

## 3.2 Was ist objektorientierte Programmierung?

### Hintergründe der OOP

Objektorientierte Programmierung ist kein wirklich neuer Ansatz. Im Gegenteil – die Idee der objektorientierten Programmierung reicht bis zum Anfang der 70er Jahre und teilweise noch weiter zurück. Sehr frühe Vertreter von Programmiersprachen mit objektorientiertem Denkansatz sind Lisp oder Logo. In den 70er Jahren entstand als wichtigster Vertreter erster objektorientierter Sprachen „Smalltalk“.

Aber erst mit Java in der Mitte der 90er Jahre hat sich die OOP wirklich durchgesetzt. Dabei ist OOP eigentlich mit dem Anspruch angetreten, die Programmierung am Denken von Menschen auszurichten und nicht Menschen zu einer computergerechten Denkweise zu zwingen. Denn die reale Welt besteht aus Objekten, derer sich Menschen bedienen. Auf diese Art soll auch die OOP funktionieren.

### Strenge Objektorientierung versus hybride Sprachen

Mit Smalltalk als Ideengeber entstanden über die Zeit eine Reihe moderner, eigenständiger Programmiersprachen, die sich ausdrücklich als rein objektorientierte Programmiersprachen verstanden und mit prozeduralen Erblasten (globale Variablen, Trennung von Anweisungen und Daten etc.) vollkommen gebrochen haben. Java oder C# zählt man dazu. Diese Programmiersprachen erzwingen einen streng objektorientierten Ansatz und unterstützen im Allgemeinen die wichtigsten theoretischen Konzepte der objektorientierten Programmierung – allerdings in unterschiedlichem Maße.

Andere Sprachen, die in dieser Zeit oder auch später entstanden sind, verfolgen einen hybriden Ansatz und erzwingen keine strenge objektorientierte Programmierung. Dazu zählen neben PHP auch Python, JavaScript oder F# und auch C/C++. Diese Sprachen erlauben auch Dinge wie globale Variablen oder Funktionen, die in dem strengen objektorientierten Ansatz ausdrücklich verboten sind. Viele Experten sind durchaus geteilter Meinung, ob man den sehr sicheren und klaren OO-Ansatz aufweichen und die freie Wahl des Programmierparadigmas sowie deren Vermischung wirklich als den Königsweg darstellen soll. Aber die Tendenz geht in der letzten Zeit in diese Richtung und auch die strengen objektorientierten Sprachen wie Java und C# öffnen sich mit besonderen Syntaxerweiterungen diesen Denkweisen.

## Ziele der OOP – Wiederverwendbarkeit und bessere Softwarequalität

Wie bereits erwähnt, nutzen viele Programmierer auch heute noch nicht die OOP in PHP. Dabei gibt es für den Einsatz von OOP in PHP zahlreiche Vorteile. Je größer und komplexer ein PHP-Projekt ist, desto mehr spricht für einen objektorientierten Ansatz. Das entscheidende Argument für die Verwendung der objektorientierten Programmierung und der Grund dafür, dass sie sich durchgesetzt hat, ist die **Wiederverwendbarkeit**.

In der althergebrachten Programmierung programmiert man prozedural beziehungsweise funktional. Dieses bedeutet die Umsetzung eines Problems in ein Programm durch eine Folge von Anweisungen, die in einer festgelegten Reihenfolge auszuführen sind. Einzelne zusammengehörende Anweisungen werden dabei maximal in kleinere Einheiten von Befehlsschritten zusammengefasst (sogenannte Unterprogramme – Funktionen oder Prozeduren).

Insbesondere gilt, dass die Datenebene und die Anweisungsebene aufgetrennt werden können. Das Problem dieser Tatsache ist, dass Änderungen in der Datenebene Auswirkungen auf die unterschiedlichsten Programmsegmente haben können und damit die Wiederverwendbarkeit von Code allgemein oder auch von Unterprogrammen sehr eingeschränkt ist, da oft Abhängigkeiten zu anderen Bestandteilen eines Programms existieren (beispielsweise zu globalen Variablen).

Objektorientierte Programmierung lässt sich im Vergleich zu diesem Paradigma der Programmierung darüber definieren, dass zusammengehörende Anweisungen und Daten eine zusammengehörende, abgeschlossene und eigenständige Einheit bilden – nämlich die Objekte. Und diese lassen sich ohne Abhängigkeiten oder Kenntnisse des inneren Aufbaus verwenden und ohne Randwirkungen intern ändern.

Das entspricht vollständig der Anwendung von Objekten im realen Leben. Ein Autofahrer schaltet beispielsweise bei Regen den Scheibenwischer seines Autos an (Verwendung eines Objekts) und ist sich vermutlich nicht im Klaren, wie dieser intern funktioniert. Wird der Scheibenwischer durch einen neuen ersetzt, kann er auch diesen Scheibenwischer wie gewohnt verwenden.

Objektorientierte Programmierung hebt die Trennung von Daten- und Anweisungsebene auf.

Das zweite zentrale Ziel der objektorientierten Programmierung ist es, eine möglichst hohe **Softwarequalität** zu erreichen. Allgemein betrachtet man bei Software sowohl die innere als auch die äußere Softwarequalität.

- ✓ Die innere Softwarequalität definiert sich aus der Sicht des Entwicklers. Die objektorientierte Programmierung bietet durch die Möglichkeiten der Abstraktion, Hierarchiebildung, Kapselung von Interna, Wiederverwendbarkeit, Schnittstellenbildung und weiterer Techniken hervorragende Ansätze zur Gewährleistung einer hohen inneren Softwarequalität.
- ✓ Die äußere Softwarequalität spiegelt die Sicht des Kunden wieder. Dieser erwartet Dinge wie Korrektheit, Stabilität, Anwendungsfreundlichkeit oder Erweiterbarkeit einer Software.

Ein Paradigma der objektorientierten Programmierung ist, dass eine hohe innere Softwarequalität zu einer hohen äußeren Softwarequalität führt.

Man kann also in PHP durchaus stichhaltige Gründe für die Verwendung von OOP finden, sowohl für eine vollständig objektorientierte Umsetzung eines Projekts als auch für eine teilweise.

### 3.3 Die Kernkonzepte der Objektorientierung

Wie angedeutet ist die Idee der objektorientierten Programmierung schon recht alt und die objektorientierte Denkweise samt vieler Fachbegriffe, die man in der OOP nutzt, gehen sogar auf die griechischen Philosophen der Antike zurück. Schon in der Philosophie von Aristoteles oder Platon findet man Begriffe wie Klassen, Metaklassen oder Instanzen und deren Deutung passt auch heute noch auf die technische Auslegung in der OOP.

In diesem Abschnitt werden die Kernkonzepte und Terminologien in der objektorientierten Software-Entwicklung kurz erläutert, die in diesem Buch mit PHP umgesetzt werden, soweit PHP dies unterstützt. Es sind folgende:

- ✓ **Objekte** dienen als Abstraktion eines realen Gegenstands oder Konzepts mit einem spezifischen Zustand und einem spezifischen Verhalten. Sie nennt man **Instanzen** einer Klasse.
- ✓ Alle Bestandteile, die direkt zu einem Objekt gehören, nennt man **Instanzelemente** oder **Instanzmember**. Der Name resultiert daraus, dass „Instanz“ oft als Synonym für „Objekt“ genommen wird. Man bezeichnet diese Bestandteile oft auch als „nicht-statisch“ oder, da sie für jede Instanz einer Klasse dynamisch erstellt werden, als „dynamisch“.
- ✓ **Klassen** sind eine Art Baupläne für Objekte. In der umgekehrten Sichtweise klassifizieren sie alle Gemeinsamkeiten, die verwandte Objekte auszeichnen. Man nennt sie auch die „Objekttypen“ oder kurz „Typen“.
- ✓ Alle Bestandteile, die nicht direkt zu einem spezifischen Objekt, sondern zu einer Klasse gehören, nennt man **Klassenelemente** oder auch statische Elemente oder **Klassenmember**.
- ✓ **Attribute** sind eine Beschreibung objekt- und klassenbezogener Daten. Oft wird auch der Bezeichner **Eigenschaft** für „Attribut“ verwendet, wobei auch hier einige Sprachen Besonderheiten aufweisen. Ebenso ist **Feld** ein oft genutztes Analogon.
- ✓ **Methoden** sind die Beschreibung objekt- und klassenbezogener Funktionalität. Sie sind das, was in der klassischen Programmierung Unterprogramme, Funktionen oder Prozeduren darstellen.

- ✓ **Assoziationen und Aggregationen** beziehungsweise **Kompositionen** sind Bezeichnungen von Mechanismen zum Ausdruck von Objektbeziehungen.
- ✓ **Vererbung** ist eine besondere Form der Assoziation und als Mechanismus zum **Generalisieren** und **Spezialisieren** von Objekttypen besonders wichtig in der objektorientierten Programmierung.
- ✓ Der Begriff **Polymorphie** ist leider nicht ganz eindeutig und wird in verschiedenen Quellen unterschiedlich streng ausgelegt. Generell bedeutet er Vielgestaltigkeit. Die Polymorphie dient hauptsächlich zur flexiblen Auswahl geeigneter Methoden identischen Namens anhand des Objekttyps und der Argumentenliste. Aber der Begriff kann auch weiter gefasst werden und dann weitere Mechanismen zum vielgestaltigen Verhalten umfassen. Auch der Begriff des späten oder dynamischen Bindens wird dafür verwendet. Allgemein bezeichnet man damit eine Auswahl einer Operation, die nicht nur vom Bezeichner der Operation, sondern auch vom Kontext abhängt: bei Methoden etwa von der Anzahl und dem Typ der Parameter, wenn diese eine Sprache berücksichtigen und bei Operatoren von den Typen der Operanden. PHP setzt die Polymorphie etwas eingeschränkt um.
- ✓ **Schnittstellen** (Interfaces) implementieren einen Mechanismus zur Strukturierung von Klassenbeziehungen.
- ✓ **Abstrakte Klassen** implementieren einen Mechanismus zum Erzwingen bestimmter Operationen in spezialisierten Klassen. In der Regel kann man damit teilweise vorgeben, dass etwas zu tun ist, aber noch nicht festlegen, wie es zu tun ist.
- ✓ **Generische Klassen** (Generics) dienen zur Darstellung von Klassenfamilien, wobei das in PHP nicht unterstützt und in diesem Buch nicht behandelt wird.

# 4

## Klassen, Objekte und Konstruktoren

### 4.1 Eine Klasse – „Bauplan“ für Objekte

Es gibt zwei Sichtweisen auf das, was man als **Klasse** bezeichnet.

- ✓ Eine Klasse fasst gewisse Dinge zusammen, die einige gewünschte Merkmale gemeinsam haben. Wenn man beispielsweise eine Gruppe von Tieren beobachtet und diejenigen zusammenfasst, die gewisse relevante Merkmale haben, dann **klassifiziert** man diese. Eine Klasse ist also in dieser Sichtweise die Zusammenfassung aller relevanten Merkmale von mehreren vorhandenen Dingen (Instanzen oder Objekte).
- ✓ Die andere Sichtweise versteht Klassen als **Baupläne** oder Schablonen, um Objekte mit gleichen relevanten Merkmalen zu erstellen.

### Abstraktion

Mit einer Klasse werden alle Eigenschaften und Verhaltensweisen eines Objekts oder mehrerer verwandten Objekte beschrieben, soweit diese **relevant** sind. „Relevanz“ bedeutet, dass man nur die Merkmale betrachtet, die in einer gewissen Situation von Bedeutung sind (**Abstraktion**). Da „Abstraktion“ eher kompliziert klingt, kann man es auch „Vereinfachung“ nennen.

### Beispiel

Für Kunden einer Bank sind die folgenden Merkmale interessant:

- ✓ Adressdaten
- ✓ Name
- ✓ Alter
- ✓ Geschlecht

Nicht relevant sind in dem Fall die Hobbies, das Gewicht oder die Haarfarbe eines Menschen, obwohl solche Merkmale bei realen Personen existieren. In der konkreten Situation sind diese irrelevant und werden in einer zur Aufgabenstellung passenden Klasse nicht abgebildet. Man lässt also Dinge weg, obwohl diese in der realen Welt durchaus vorhanden und spezifizierbar sind und das ist eine Vereinfachung (Abstraktion) der Situation.

In einer anderen Situation könnten genau die gleichen realen Dinge beziehungsweise Objekte ganz anders klassifiziert werden. Denken Sie etwa an eine Partnervermittlung. In dem Fall sind Merkmale wie die Haarfarbe, die Größe, das Gewicht oder die Hobbies durchaus interessant, während die Adresse erst einmal irrelevant sein kann.

## Instanzen übernehmen Merkmale, die in Klassen definiert sind

Eine Klasse versucht, diese in einer konkreten Situation benötigten Aspekte der realen Welt, die betrachtet werden sollen, in der Programmierung umzusetzen. Von dieser Schablone können dann in der Regel beliebig viele Objekte (Instanzen) angelegt werden. Jedes Objekt einer Klasse übernimmt automatisch die definierten Eigenschaften und Fähigkeiten, die in der Klasse deklariert wurden, sofern sie dafür vorgesehen werden.

## 4.2 Klassen in PHP deklarieren

Die Deklaration einer Klasse wird in PHP wie folgt vorgenommen:

- ✓ Zuerst notiert man das Schlüsselwort `class`. Im Gegensatz zu den meisten anderen OO-Sprachen notiert man in PHP **keine** vorangestellten Modifizierer bezüglich der sogenannten Sichtbarkeit. Diese tauchen aber bei den Bestandteilen der Klasse auf und werden da erläutert.
- ✓ Dem Schlüsselwort `class` folgt ein möglichst aussagekräftiger Name der Klasse. Die Bezeichnung folgt den üblichen Regeln in PHP. Sie beginnt mit einem Unterstrich oder einem Buchstaben, gefolgt von einer beliebigen Anzahl von Buchstaben, Ziffern oder Unterstrichen. Nach Konvention werden Klassennamen immer in der Singularform angegeben und der erste Buchstabe immer großgeschrieben.
- ✓ In geschweiften Klammern wird der eigentliche Quellcode der Klasse (die Implementierung) notiert. Dieser besteht aus internen, von außen verborgenen, PHP-Anweisungen und den Dingen, die von außen zugänglich sein sollen (Eigenschaften und Methoden oder verallgemeinert **Member** beziehungsweise **Mitglieder** einer Klasse genannt).

Die Struktur einer Klasse in PHP sieht also rein formal folgendermaßen aus:

```
class Klassename {
    // Eigenschaften
    ...
    // Methoden
    ...
}
```

*Die formale Klassendeklaration in PHP*

Eine leere, aber formal bereits vollständige PHP-Klasse sieht etwa so aus:

```
class Kunde { }
```

*Eine leere, aber nichtsdestotrotz vollständige Klassendeklaration in PHP*

Der Name einer Klasse wird im Singular notiert. Das ist zwar nicht verbindlich, wird aber in der professionellen OO-Programmierung konsequent so umgesetzt. Eine Klasse ist die Beschreibung für genau ein Objekt. Wenn man mehrere Objekte braucht, erzeugt man trotzdem in jedem Schritt immer nur ein Objekt – nur mehrfach hintereinander. Denken Sie dabei an die vereinheitlichte Großschreibung!

## Eine Klasse, eine PHP-Datei

Es ist zwar nicht notwendig, aber sehr sinnvoll, dass jede Klasse in einer eigenen PHP-Datei gespeichert wird. Der Name der Quelltextdatei ergibt sich nach Konvention (auch nicht zwingend, aber ebenso sehr sinnvoll) aus dem Namen der Klasse und einer Kennzeichnung dafür, dass es sich bei der Datei um eine Klassendeklaration handelt.

Eine Klasse `Kunde` würde nach dieser Strategie in einer PHP-Datei mit dem Namen `Kunde.class.php` gespeichert werden.

**!** Beachten Sie, dass bei manchen Betriebssystemen bei den Datei- und Verzeichnisnamen nicht zwischen Groß- und Kleinschreibung unterschieden wird, etwa bei Windows. Unix-basierte Systeme unterscheiden diese jedoch und da die überwiegende Anzahl der Webserver in der Praxis auf einem Unix-/Linux-System läuft, sollte man auch in seiner lokalen Testumgebung unbedingt die Groß- und Kleinschreibung von PHP-Dateien beachten und eine konsequente Strategie verfolgen. Im Buch wird die Regel konsequent eingehalten – alle Klassendateien für den vorderen Teil des Namens werden genauso wie die Klassen geschrieben. Sie beginnen also in jedem Fall mit einem Großbuchstaben. Das ist später beim Laden dieser Dateien sehr wichtig.

Das Einhalten der Strategie ist nicht zwingend erforderlich, deckt sich jedoch mit Strategien, wie man in Java oder C# Klassen und den Bezug zu den deklarierenden Quellcodedateien strukturiert und somit viel Arbeit sowie potentielle Probleme vermeidet. Wenn Sie die Klassen dann benötigen, können diese Dateien nach einer klaren und einfachen Vorgabe mit `include-` oder `require-`Befehlen eingebunden werden.

Am Ende des Kapitels wird die damit assoziierte Technik „Autoloading“ beschrieben. Diese bündelt und erleichtert das automatische Laden von Klassendateien, insbesondere, wenn Sie die Konventionen einhalten.

Wenn Bedarf an mehreren Klassen besteht, ist es durchaus sinnvoll, eine weitere Datei zu nutzen. In diese werden zunächst alle benötigten Klassendateien eingeschlossen und sie dann in die endgültige PHP-Datei inkludiert.

Dieses Verfahren ist Teil dessen, was mit „Trennung von Code und Design“ oder „Modularisierung“ beschrieben wird und dem hauptsächlichen Ziel der OOP zugutekommen soll – wiederverwendbaren Code zu erstellen.

## 4.3 Eigenschaften

Eine Form eines Members einer Klasse nennt sich **Eigenschaft**. Eigenschaften in der klassischen Theorie der OOP sind erst einmal nur Variablen, die innerhalb einer Klasse definiert wurden und von außen zugänglich sind. Über die verschiedenen Werte der Eigenschaften (nicht die Anzahl oder Art der Eigenschaften) unterscheiden sich Objekte gleichen Typs. Kunden einer Bank unterscheiden sich beispielsweise im Namen (und natürlich auch den Werten der anderen Eigenschaften). Alle Kunden haben einen Namen, aber die Werte sind in der Regel unterschiedlich. Aber natürlich können auch Werte von Eigenschaften gleich sein – etwa von der Eigenschaft Geschlecht. Damit sich Objekte gleichen Typs jedoch überhaupt unterscheiden lassen, sollten sie sich in mindestens dem Wert einer Eigenschaft unterscheiden.

Die als Eigenschaften in der Klasse deklarierten Variablen werden bei der Erstellung eines Objekts als Kopie der Instanz zugeordnet. Diese Eigenschaften gehören somit zum Objekt. Einfacher ausgedrückt: Wird aus einer Klasse ein neues Objekt erzeugt, erhält es eine Kopie aller dort definierten Eigenschaften. Jedes neue Objekt startet deshalb erst einmal mit dem identischen Satz an Eigenschaften („alles, was ein Objekt ausmacht“). In einigen, aber nicht in allen Programmiersprachen können Objekte auch nach der Erzeugung individuell um Eigenschaften erweitert werden.

Die so deklarierten Eigenschaften sind **Instanzeigenschaften** (auch **Instanzvariablen** genannt). Wenn sich die Werte nach der Erstellung einer Instanz in dieser ändern, wird sich das nicht auf die eventuell existierenden anderen Instanzen und auch nicht auf die Klasse auswirken. Die Eigenschaften „gehören“ damit explizit zu jeder einzelnen Instanz. Wenn man nichts anderes vereinbart, meint man mit „Eigenschaften“ immer Instanzeigenschaften. Wobei diese Erklärung impliziert, dass es auch andere Eigenschaften geben kann. Diese gehören zur Klasse und man nennt diese dann Klasseneigenschaften oder auch statische Eigenschaften.

### Die Sichtbarkeit von Eigenschaften

Nun muss man bei der Deklaration von Eigenschaften beziehungsweise allgemein Variablen (als auch Methoden – also allgemein Membern) in Klassen in PHP noch Angaben zur sogenannten Sichtbarkeit machen. Seit PHP 5 werden Eigenschaften mit ihrer vorangestellten Sichtbarkeitsstufe beziehungsweise ihrem Gültigkeitsbereich deklariert. Dazu gibt es **Sichtbarkeitsmodifizierer**, oft kurz **Modifizierer** genannt. Es gibt jedoch auch Modifizierer, die nicht die Sichtbarkeit, sondern andere Dinge regeln (etwa `abstract`).

Sichtbarkeitsmodifizierer	Erklärung
<code>private</code>	Sichtbarkeitsstufe <b>Privat</b> <ul style="list-style-type: none"> <li>✓ Da die Variable von außen nicht zugänglich ist, steht sie nur innerhalb der Klasse zur Verfügung und ist damit im eigentlichen Sinn keine Eigenschaft. Obwohl man in einigen Quellen auch von einer „privaten Eigenschaft“ spricht.</li> </ul>
<code>protected</code>	Sichtbarkeitsstufe <b>Geschützt</b> <ul style="list-style-type: none"> <li>✓ wie <code>private</code>, aber es ist auch Zugriff aus erbenden (abgeleiteten) Klassen möglich (Vererbung).</li> </ul>
<code>public</code>	Sichtbarkeitsstufe <b>Öffentlich</b> <ul style="list-style-type: none"> <li>✓ überall sichtbar und verwendbar</li> </ul>

Beachten Sie, dass es in anderen OO-Sprachen noch weitere Sichtbarkeitsmodifizierer gibt und sich die Bedeutungen im Detail unterscheiden können. So gut wie immer sind jedoch `public` und `private` vorhanden und von der Bedeutung gleich.

Im weiteren Inhalt dieses Buches wird eine Art „Projekt“ vorangetrieben, das sukzessive die Erweiterung von Klassen um immer neue Techniken zeigt und gegen Ende zu einer vollständig objektorientiert erstellen Webseite führt. Dazu werden die verschiedenen Entwicklungsstände in unterschiedlichen Verzeichnissen gespeichert, die eine Art „Versionierung“ darstellen sollen und deshalb kapitelweise jeweils mit Zahlen am Ende durchnummert werden. In jedem neuen Kapitel wird mit dem Endstand des vorherigen Kapitels weitergearbeitet. Beachten Sie, dass dabei vorherige Versionen umgearbeitet und verändert werden, um eine höhere „Softwarequalität“ zu erreichen und dem Ziel näherzukommen.

Eine PHP-Klasse, welche einen Bankkunden mit gewissen öffentlichen Eigenschaften abstrahieren soll, sieht beispielsweise so aus (Verzeichnis v1):

```
<?php
class Kunde
{
    public $vorname = "";
    public $nachname = "";
    public $alter = 0;
    public $geschlecht = "";
    public $adresse = "";
}
?>
```

*Eine Klassendeklaration mit öffentlichen Eigenschaften*

Eine weitere Klasse könnte etwa ein Konto repräsentieren (Verzeichnis v1):

```
<?php
class Konto
{
    public $kontostand = 0;
    public $kontotyp = "";
}
?>
```

*Eine weitere Klassendeklaration mit öffentlichen Eigenschaften, die ein Konto repräsentiert*

## 4.4 Methoden

Methoden sind die zweite Variante eines Members einer Klasse und in PHP erst einmal nur innerhalb einer Klasse definierte Funktionen. Damit wird festgelegt, welche Aktionen Objekte durchführen können.

Die Deklaration einer Methode beginnt wie bei Funktionen mit dem Schlüsselwort `function`. Aber auch hier können die Sichtbarkeitsstufen mithilfe der Schlüsselwörter `public`, `protected` und `private` vorangestellt werden.

Wird kein Schlüsselwort angegeben, wird automatisch die öffentliche Sichtbarkeitsstufe `public` angenommen. Das ist die gängigste Einstellung in OOP für Methoden, aber damit leider in PHP inkonsistent beziehungsweise inkonsistent, wo zur Deklaration von Eigenschaften **zwingend** ein Sichtbarkeitsmodifizierer vorangestellt werden muss.

Die nachfolgende Deklaration erweitert die bisherige Klasse `Konto` um zwei Methoden zum Einzahlen und Auszahlen eines Geldbetrags auf das Konto. Dazu wird irgendwann der Kontostand verändert werden müssen, aber für die rein formale Deklaration ist das noch nicht notwendig (Verzeichnis v2).

```
<?php

class Konto
{
    public $kontostand = 0;
    public $kontotyp = "";
    public function einzahlen(float $betrag) {
    }
    function auszahlen(float $betrag) {
    }
}
?>
```

*Die Klassendeklaration wurde um zwei Methoden erweitert*

Beachten Sie, dass aus didaktischen Gründen die Methode `einzahlen()` **explizit** als `public` deklariert wurde, während die Methode `auszahlen()` durch das Weglassen eines Sichtbarkeitsmodifizierers im derzeitigen Stand des Projekts **implizit** als öffentlich deklariert wurde. Das wird in der Folge noch vereinheitlicht und soll einen Teil der angestrebten sukzessiven Verbesserung der „Softwarequalität“ darstellen.

Bei der Deklaration von Methoden sollten Sie sich auf eine der beiden Strategien festlegen. Wenn Sie sich also vom Programmierstil an vergleichbaren objektorientierten Sprachen orientieren und vor allen Dingen konsequent bei den Vorgaben zu den Eigenschaften in PHP bleiben wollen, dann notieren Sie den Modifizierer jedes Mal. Wenn Sie den Code kompakter schreiben wollen, können Sie den Modifizierer weglassen – aber dann auch jedes Mal.

Da Methoden im Grunde „normale“ Funktionen darstellen, die nur innerhalb von Klassen deklariert werden, gelten alle üblichen Regeln für Funktionen – sowohl für die Bezeichner als auch für die Übergabe- und Rückgabewerte.

Insbesondere kann man seit PHP 7.0 bei der Angabe von Parametern **skalare Datentypen** angeben (im Beispiel wird `float` genommen). Das macht die Verwendung von Funktionen und damit auch Methoden erheblich sicherer und stabiler. Aber Sie werden zur Angabe von solchen Datentypen nicht gezwungen. Sollten Sie noch mit älteren PHP-Versionen arbeiten, müssen Sie diese Vereinbarungen der Datentypen sogar weglassen. Allerdings ist grundsätzlich ein Einsatz von PHP-Versionen vor 7.x zum Zeitpunkt der Erstellung der Unterlagen nicht mehr zu empfehlen, da der Support älterer Versionen eingestellt wurde. Deshalb wird die Angabe von Datentypen in absehbarer Zeit grundsätzlich keine Probleme mehr machen.

Die so deklarierten Methoden sind Instanzmethoden. Sie „gehören“ explizit zu jeder einzelnen Instanz und können damit auch auf deren Instanzeigenschaften direkt zugreifen.

## Zugriff auf Instanzeigenschaften aus Methoden und das Schlüsselwort `$this`

Methoden sind für die aktiven Schritte in einem Objekt verantwortlich und in sehr vielen Fällen greifen Sie dabei auf Eigenschaften in der Klasse beziehungsweise der Instanz zurück.

Vollführen wir zur Erklärung einen kurzen Rücksprung in die prozedurale Welt. Dort ist eine globale Variable in PHP ja erst einmal nicht in einer Funktion verfügbar, sondern muss explizit mit dem Schlüsselwort `global` in der Funktion bekanntgegeben werden.

Wie ist das nun in der objektorientierten Welt mit Eigenschaften?

Eigentlich dürfte man gar nicht mit dieser Frage konfrontiert werden, denn globale Variablen sind in dem objektorientierten Ansatz streng verboten. Viele Experten sind sich einig, dass der Verzicht auf globale Variablen einer der Hauptfortschritte der OOP war.

Unabhängig von den globalen Variablen – Instanzvariablen werden ja auch außerhalb von Methoden deklariert. Sind sie damit nicht auch globale Variablen? Nein – sie werden ja innerhalb der Klassendeklaration notiert und nicht wie globale Variablen vollkommen im „leeren Raum“. Man könnte sie maximal als „klassenglobal“ (keine offizielle Definition) bezeichnen.

Diese Instanzvariablen stehen auch in den Instanzmethoden direkt zur Verfügung und müssen dort nicht erst durch ein Schlüsselwort oder auf andere Weise explizit bekanntgegeben werden. So etwas ist also möglich (Verzeichnis v3):

```
<?php

class Konto
{
    public $kontostand = 0;
    public $kontotyp = "";

    public function einzahlen(float $betrag) {
        $kontostand += $betrag;
    }

    public function auszahlen(float $betrag) {
        $kontostand -= $betrag;
    }
}
```

*Die Deklaration der zwei Methoden wurde modifiziert*

In den Deklarationen der Methoden zum Einzahlen und Auszahlen eines Betrags wird in dieser Version des Beispiels der Wert der Eigenschaft `$kontostand` verändert, indem einfach direkt auf diese Instanzeigenschaft `$kontostand` in den Instanzmethoden zugegriffen wird. Der Parameter `$betrag` fungiert wie üblich als lokale Variable in der Methode – das ist vollkommen identisch mit „normalen“ Funktionen.



Beachten Sie, dass ab sofort alle Methoden in der Weiterentwicklung des „Kapitelprojekts“ einheitlich als `public` deklariert sind.

Doch was wäre, wenn der Name des Parameters einer Instanzmethode identisch zum Namen einer Instanzeigenschaft der Klasse gewählt würde? Etwa so:

```
<?php

class Konto
{
    public $kontostand = 0;
public $kontotyp = "";

    public function einzahlen(float $betrag) {
        $kontostand += $betrag;
    }

    public function auszahlen(float $betrag) {
        $kontostand -= $betrag;
    }

    public function setKontotyp(string $kontotyp) {
        $kontotyp = $kontotyp;
    }
}
```

*Der Kontotyp soll gesetzt werden, aber das geht so nicht*

Es gibt also eine Instanzeigenschaft `$kontotyp` und einen Parameter in der Methode `setKontotyp()` gleichen Namens und dieser Parameter wird in der Methode verwendet.

Rein von der Syntax ist das problemlos möglich. Nur kann man bei der Zuweisung des Werts von dem Parameter `$kontotyp` in der Methode `setKontotyp()` auf der linken Seite der Zuweisung nicht erkennen, dass damit die Eigenschaft außerhalb der Methode gemeint ist. Der als lokale Variable verfügbare Parameter bekommt damit einfach selbst wieder den Wert zugewiesen. Das ist zwar falsch, aber sinnlos.

Eine lokale Variable in der Methode **verdeckt** die Instanzvariable bei gleichem Bezeichner.

Um auch auf verdeckte Instanzvariablen zugreifen zu können, gibt es das Schüsselwort `$this`. Es wird innerhalb von Methoden verwendet und stellt eine Referenz auf das aufrufende beziehungsweise aktuelle Objekt dar. Wenn Sie innerhalb einer Klasse auf die eigene Objektvariable zugreifen möchten, können Sie dazu die Referenz `$this` als Platzhalter für das aktuelle Objekt verwenden. Die Instanzeigenschaft oder Instanzmethode wird mit einem `->` nachgestellt.

Das Beispiel würde also so erweitert (Verzeichnis v4):

```
<?php
class Konto
{
    public $kontostand = 0;
public $kontotyp = "";
    public function einzahlen(float $betrag) {
        $kontostand += $betrag;
    }
```

```

    }
    public function auszahlen(float $betrag) {
        $kontostand -= $betrag;
    }
    public function setKontotyp(string $kontotyp) {
        $this -> kontotyp = $kontotyp;
    }
}
?>
```

*Der Kontotyp wird gesetzt*

Nun ist die Syntax eindeutig. Mit `$this -> kontotyp` meint man also explizit die Instanzvariable und ohne das vorangestellte `$this` ist es nur dann die Instanzvariable, wenn diese nicht von einer lokalen Variable verdeckt wird. Ansonsten ist es der Bezeichner der verdeckenden lokalen Variable. Damit erhalten Sie eine große Flexibilität.

Diese Art der identischen Benennung von Parametern und Instanzvariablen ist keine Ausnahme oder unglücklich gewählt, sondern bei konsequenter Anwendung von OOP und Datenkapselung sogar Regelfall. Manche Programmierer sagen sogar, sie sei unabdingbar, da nur durch den gleichen Namen von Parameter und Eigenschaft immer klar ist, welche Eigenschaft gemeint ist. Darauf wird bei „Settern“ und „Gettern“ genauer eingegangen.

## 4.5 Der Zugriff auf Instanzelemente

Bereits bei der Erklärung von `$this` wurde die Pfeilnotation mit `->` verwendet. In PHP kann jede Eigenschaft oder Methode über ein vorangestelltes Objekt (auch das aktuelle Objekt über `$this` oder ein anonymes Objekt) angesprochen werden, wenn man danach `->` verwendet und dann die Eigenschaft oder Methode notiert.

Das nennt man eine **Botschaft** an das Objekt (Message).

Formal sieht diese Pfeilnotation so aus:

```
$variable -> eigenschaft;
$variable -> methode();
```

*Formaler Zugriff auf Eigenschaften und Methoden über ein vorangestelltes Objekt*

**!** Beachten Sie bei der Schreibweise, dass das Zeichen `$` nur vor der Objektbezeichnung, aber **nicht** vor der Eigenschaft hinter dem Pfeil steht. Das ist eine tückische Fehlerquelle (gerade für Umsteiger).

Beachten Sie ferner, dass die meisten anderen objektorientierten Sprachen bei Botschaften an ein Objekt das Objekt und die Eigenschaften oder Methoden durch einen Punkt abtrennen. Man spricht hier von der **Punktnotation** oder **Dot-Notation**. Da der Punkt in PHP aber für die Stringverkettung verwendet wird, muss hier eine andere Syntax zum Einsatz kommen – eben die Pfeilnotation über `->`. Auch das irritiert Umsteiger immer wieder.

## 4.6 Datenkapselung, Getter- und Setter-Methoden

Bei der konsequenten objektorientierten Programmierung ist ganz wesentlich, dass ein Objekt seine innere Struktur so gut wie irgend möglich vor anderen Objekten oder am besten überhaupt vor einem direkten Zugriff von außen verstecken kann. Man nennt dies **Information Hiding**, **Datenkapselung** oder kurz **Kapselung**.

Um das zu gewährleisten, stellen echte objektorientierte Sprachen die Zugriffsmodifizierer beziehungsweise Zugriffsmodifier für eine sehr feingliedrige Steuerung des Zugriffs auf die Objektinterna zur Verfügung. Aber auch die Art der Programmierung selbst kann massiven Einfluss auf die Umsetzung einer optimalen Datenkapselung haben. Mit zunehmender Erfahrung in der OOP hat man gewisse Arten an Zugriffsmethoden auf Eigenschaften sehr zu schätzen gelernt:

- ✓ **Setter-Methoden** oder kurz **Setter** sind Methoden zum indirekten Schreibzugriff auf Eigenschaften eines Objekts.
- ✓ Analog sind **Getter-Methoden** oder kurz **Getter** Methoden zum indirekten Lesezugriff auf die Eigenschaften.

Sie können damit den Zugriff auf eine Eigenschaft mit einer gewissen Logik versehen und zudem steuern, ob ein Zugriff nur lesend (kein Setter – readonly-Eigenschaft), nur schreibend (kein Getter – writeonly-Eigenschaft) oder beides sein darf.



Wichtig ist, dass bei der Umsetzung **konsequenter** Datenkapselung **alle (!)** Eigenschaften als „privat“ gekennzeichnet und damit nicht mehr nach außen gegeben werden. Strenggenommen stellen sie damit keine Eigenschaften mehr dar, sondern nur noch lokale Felder. Es ist jedoch nicht immer notwendig oder manchmal auch nicht möglich, so eine Konsequenz radikal umzusetzen. Wie fast immer ist die Einhaltung von theoretischen Konzepten kein reiner Selbstzweck, sondern man versucht eine möglichst ideale Lösung eines Problems umzusetzen. Und dafür bringt oft auch eine teilweise Datenkapselung Vorteile. Und wenn ein direkter Zugriff effektiver ist, sollte man ihn nicht rein der Theorie wegen verwerfen.

Diese Steuerung der Art des Zugriffs auf Objekteigenschaften ist in mächtigen OO-Sprachen sehr populär und wurde etwa von Microsoft im Rahmen des .NET-Frameworks noch weiterentwickelt. Hier nutzt man besagte Getter- und Setter-Methoden nicht mehr, sondern geht einen Schritt weiter. Es gibt eine noch viel strengere, zusätzliche und hinsichtlich der OO-Theorie proprietäre Syntaxerweiterung, in der Attribute und Eigenschaften (in der klassischen OO synonym zu verwenden) nicht mehr dasselbe sind. Im .NET-Framework sind Eigenschaften ganz spezielle syntaktische Zugriffstechniken (Properties genannt), die nur noch einen indirekten Zugriff auf die Attribute gestatten. Diese sind nur noch private Felder.

### Information Hiding first

Auch in anderen Sprachen der OOP sollten Sie den direkten Zugriff auf Eigenschaften vollkommen vermeiden. Das ist keine Verpflichtung, aber die

- |                       |               |                 |
|-----------------------|---------------|-----------------|
| ✓ Wartbarkeit,        | ✓ Sicherheit, | ✓ Änderbarkeit, |
| ✓ Zuverlässigkeit und | ✓ Stabilität  |                 |

von Code wird massiv verbessert, wenn der Zugriff auf Eigenschaften ausschließlich indirekt mithilfe von Methoden möglich ist.

Ein Entwickler stellt nur die Methoden zur Verfügung, die er für sinnvoll erachtet und begrenzt damit u. a. den Zugriff auf Eigenschaften. Darüber können auch unsinnige Wertveränderungen von Eigenschaften und nicht gewünschtes Auslesen von Informationen gefiltert werden.

- ✓ Es hat sich etabliert, dass Methoden, die direkt den Wert einer Eigenschaft ändern, den Bezeichner `setEigenschaft()` bekommen. Man beginnt also mit dem Token `set` und notiert danach mit Camelnotation (also einem Großbuchstaben) den Namen der zu ändernden Eigenschaft. Der Bezeichner des Parameters soll zwingend den Namen der zu ändernden Eigenschaft bekommen, was bereits vorgestellt wurde.
- ✓ Methoden, die den Wert einer Eigenschaft zurückliefern, sollten Sie entsprechend `getEigenschaft()` nennen (Beginn mit `get`). Nur bei booleschen Lesemethoden nimmt man oft die Notation `isEigenschaft()` (Beginn mit dem Token `is`).
- ✓ Methoden, die eine Eigenschaft nicht einfach setzen oder modifizieren, sondern mit einer komplexeren Logik darauf zugreifen, können bei Bedarf auch anders benannt werden.

Wenn jenseits der reinen Syntax diese Art des Aufbaus von Klassen und OO-Code betrachtet wird, spricht man vom **Design** des Codes, in dem Fall genauer vom **objektorientierten Design** (OOD). Um es noch einmal zu betonen – man wird zu dieser Art der Programmierung nicht von einer Sprache wie PHP, C# oder Java und deren reiner Syntax gezwungen, aber sie hat sich als sehr vorteilhaft etabliert und vermeidet zahlreiche Probleme, die bei einer Missachtung unweigerlich auftreten.

Die folgende Modifizierung des Konto-Beispiels zeigt die vorgeschlagene Methodendeklaration und -anwendung für die Klasse Konto (Verzeichnis v5):

```
<?php

class Konto
{
    private $kontostand = 0;
    private $kontotyp = "";

    public function einzahlen(float $betrag) {
        $this -> kontostand += $betrag;
    }
    public function auszahlen(float $betrag) {
        $this -> kontostand -= $betrag;
    }
    public function setKontotyp(string $kontotyp) {
        $this -> kontotyp = $kontotyp;
    }
    public function getKontotyp(): string{
        return $this -> kontotyp;
    }
    public function getKontostand(): float{
        return $this -> kontostand;
    }
}
?>
```

*Das Konto befolgt nun die Regeln der Datenkapselung*

- ✓ Alle ehemals als `public` deklarierten Eigenschaften sind nun `private`.
- ✓ Alle Methoden sind ausdrücklich als `public` deklariert.
- ✓ Der Zugriff auf den Kontotyp erfolgt ausschließlich über Getter und Setter.
- ✓ Den schreibenden Zugriff auf den Kontostand könnte man auch über Getter und Setter implementieren. Aber sowohl aus didaktischen Gründen als auch aufgrund der Tatsache, dass ein Setter nicht einfach den neuen Wert der Variable setzt, sondern um den Wert des Parameters verändert, und es zudem einen logischen Unterschied zwischen einer Einzahlung und einer Auszahlung gibt, wird hier zum Setzen des Werts mit zwei expliziten Methoden gearbeitet. Zur Abfrage gibt es dann einen Standard-Getter.

Der Code der vorherigen Version des Projekts wurde in dem letzten Schritt umstrukturiert, ohne die Funktionalität (wesentlich) zu ändern. Man nennt so etwas Refaktorisierung oder engl. Refactoring. Refaktorisierung ist eine der wichtigsten Maßnahmen, um einen erstmal grundsätzlich fehlerfreien und funktional sinnvollen Code auf ein Level zu heben, dass dessen Softwarequalität den notwendigen Ansprüchen genügt. Dabei wird die Wartbarkeit, Robustheit, Stabilität, Sicherheit, Performance etc. verbessert, aber das Programm macht an sich das Gleiche wie vor der Maßnahme.

Das objektorientierte Design der Klasse `Konto` ist nun hinsichtlich der Datenkapselung und Konsistenz schon in einem recht guten Zustand. Oder anders ausgedrückt – die innere Software-Qualität ist gut, was nach der OO-Theorie zu einer guten äußeren Software-Qualität führen soll oder sie zumindest unterstützt.

Für den Kunden muss man diese Schritte zur Datenkapselung noch durchführen, was am Ende des Kapitels als Aufgabe konzipiert ist.

## 4.7 Objekte erzeugen und der Konstruktor

### Der Konstruktor

Wie Klassen aufgebaut sind haben wir in wichtigen Details gesehen. Aber bisher kann noch kein Objekt daraus erzeugt werden. Denn dafür gibt es den Konstruktor. Dabei handelt es sich um eine spezielle Methode, die automatisch für jedes neu zu erzeugende Objekt aufgerufen wird. So ein Objekt wird erzeugt, indem man mit dem Schlüsselwort `new` den Namen der Klasse, gefolgt von einem runden Klammernpaar, notiert. Das ruft dann implizit die Konstruktormethode auf.

Die Syntax sieht formal so aus:

```
new Klassename()
```

*Das formale Instanziieren einer Klassendeklaration in PHP*

Jede Klasse besitzt immer einen Defaultkonstruktor, der keine Parameter besitzt.

Ein Objekt der Klasse `Konto` könnte so instanziert werden:

```
new Konto()
```

*Das konkrete Instanziieren einer Klassendeklaration in PHP*

Bei Bedarf kann man in die Klammern auch Argumente (Variablen oder Literale) notieren:

```
new Klassename ($p1, $p2, 1, true, ...)
```

*Das formale Instanziieren einer Klasse mit der Übergabe von Parametern*

Das Angeben von Parametern ist fast immer richtig, denn damit kann man einen „Anfangszustand“ eines Objekts festlegen. Ein nicht sauber initialisiertes Objekt ist selten sinnvoll.

Für ein Konto wäre es folgerichtig, wenn dieses bereits bei der Erzeugung einen Kontotyp und einen Kontostand hätte. Das kann man so machen:

```
new Konto(1000, "Girokonto")
```

*Das konkrete Instanziieren einer Klassendeklaration in PHP*

Wenn man beim Instanziieren Argumente übergibt, sollte man diese beim Erzeugen von der Instanz auch verwenden. Vor allen Dingen gibt es erst einmal keine passende Methode mit diesen Parametern.

Da kommt dann der konkrete Konstruktor ins Spiel.

Seit PHP 5 hat der Konstruktor (die Methode) selbst den festgelegten Namen `__construct()`. Auch mit dieser Syntax weicht PHP von der Philosophie der meisten anderen objektorientierten Sprachen ab, was Umsteiger beachten müssen.

**!** Der Methodename des Konstruktors beginnt mit zwei Unterstrichen `__`. Methoden, die in PHP mit diesen beiden Unterstrichen beginnen, werden als **magische Methoden** bezeichnet. Das bedeutet, dass diese Methoden in den passenden Konstellationen (in diesem Fall dem Erzeugen eines Objekts der Klasse) automatisch ausgeführt werden.

Ein Konstruktor wird jedoch in der Regel dazu eingesetzt, einem neu erzeugten Objekt eine Reihe von individuellen Werten für die definierten Eigenschaften zuzuweisen. Aber das ist nicht zwingend. Wenn jedoch an einen Konstruktor Werte übergeben werden, spricht man von einem **parametrisierten Konstruktor**. Dieser muss explizit für jede Klassendeklaration erstellt (redefiniert) werden.

**!** Sie müssen einen Konstruktor einer Klasse nicht redefinieren, denn er steht – wie schon erwähnt – ohne Parameter (leerer oder parameterloser Konstruktor) immer automatisch zur Verfügung, wenn Sie eine Klassendeklaration erstellt haben (Defaultkonstruktor). Wenn Sie jedoch einen Konstruktor redefinieren, steht der Defaultkonstruktor danach **nicht mehr** zur Verfügung.

In dem Beispiel von oben ist folgender Ansatz sinnvoll (Verzeichnis v6):

```
<?php
class Konto {
    private $kontostand = 0;
    private $kontotyp = "";
    public function __construct(float $kontostand, string
        $kontotyp) {
```

```

    $this -> kontostand = $kontostand;
    $this -> setKontotyp($kontotyp);
}

public function einzahlen(float $betrag) {
    $this -> kontostand += $betrag;
}

public function auszahlen(float $betrag) {
    $this -> kontostand -= $betrag;
}

public function setKontotyp(string $kontotyp) {
    $this -> kontotyp = $kontotyp;
}

public function getKontotyp(): string{
    return $this -> kontotyp;
}

public function getKontostand(): float{
    return $this -> kontostand;
}

?>

```

#### *Die Notation eines parametrisierten Konstruktors*

Bei der Initialisierung der Klasse erhalten die Eigenschaften über die Methode `__construct()` einen expliziten Anfangszustand, da diese automatisch beim Erzeugen eines Objekts ausgeführt wird.

Die Konstruktormethode erwartet in dem Beispiel zwei Argumente, die dann im Inneren den Eigenschaften des aufrufenden Objekts zugewiesen werden.

Beachten Sie, dass zum Zuweisen des Kontostands die Notation der Eigenschaft selbst über das vorangestellte `$this` zum Einsatz kommt, für den Kontotyp hingegen der Setter verwendet wird.

Diese Art des Designs des Konstruktors ist kein Zufall, sondern folgt etablierten Konventionen. Indem man im Konstruktor konkret Setter (oder manchmal auch Getter) verwendet (sofern vorhanden, was im Beispiel nur für den Kontotyp der Fall ist), kann man bereits sämtliche Logik verwenden, die man ggf. dort implementiert hat. Allerdings ist es von der konkreten Situation abhängig, ob so ein Vorgehen sinnvoll ist. Etwa kann es sinnvoll sein, dass ein Konto beim Erzeugen keinen negativen Kontostand aufweisen kann. Allerdings kann es möglich sein, dass für das Konto ein Dispokredit vereinbart ist und das Konto während der späteren „Lebenszeit“ durchaus einen negativen Stand aufweisen kann. Der Setter würde also einen negativen Wert des Kontostands gestatten müssen und könnte beim Erzeugen nicht so einfach den positiven Stand gewährleisten. Der Setter müsste entweder nur für den Fall des Erzeugens eines Objekts komplexer aufgebaut werden oder aber man verlagert die Logik doch besser in den Konstruktor und verzichtet dort auf den Aufruf des Setters. Es hängt also wie immer an der Abstraktion der Realität, wie der Code modelliert wird. Aber in vielen Fällen möchte man die Logik eines Setters bereits im Konstruktor nutzen.

Um die Instanziierung des Kontos nun zu sehen, soll ein Kunde ein Konto besitzen. Dazu wird die Klasse Kunde um die entsprechende Eigenschaft erweitert und dort im Konstruktor auch das Konto erzeugt (Verzeichnis v6).

```
<?php
require_once("Konto.class.php");

class Kunde {
    public $vorname = "";
    public $nachname = "";
    public $alter = 0;
    public $geschlecht = "";
    public $adresse = "";
    public $konto = null;

    public function __construct(string $vorname, string $nachname, int
        $alter, string $geschlecht, string $adresse, Konto $konto) {
        $this -> vorname = $vorname;
        $this -> nachname = $nachname;
        $this -> alter = $alter;
        $this -> geschlecht = $geschlecht;
        $this -> adresse = $adresse;
        $this -> konto = $konto;
    }
}

?>
```

Die erweiterte Klasse „Konto“

- ✓ Als erstes muss in der PHP-Datei der Klasse Kunde die PHP-Datei mit der Deklaration der Klasse Konto eingebunden werden, denn diese wird hier als Typ benötigt (`require_once ("Konto.class.php");`).
- ✓ Da auch die Klasse Kunde einen parametrisierten Konstruktor haben soll, muss bei der Erzeugung eines neuen Objekts die im Konstruktor vorgegebene Anzahl an Argumenten übergeben werden.
- ✓ Die Eigenschaft `$konto` ist neu. Sie ist vom Typ Konto. Letzteres ist ein Referenztyp – also ein Objekt vom Typ der Klasse Konto. Beachten Sie, dass das nicht mit „Call-by-Reference“ bei der Übergabe von Parametern (Auszeichnung mit vorangestelltem &) verwechselt werden darf, obwohl hier schon Ähnlichkeiten bestehen.

## Objektidentität

Wenn Sie aus einer Klasse mehrere Instanzen erzeugen, sind diese erst einmal „baugleich“. Aber auch sonst kann es vorkommen, dass Instanzen der gleichen Klasse sich zur Laufzeit in keiner ihrer Eigenschaften unterscheiden. Für solche (scheinbar) identischen Objekte muss jedoch ein eindeutiges Unterscheidungskriterium vorhanden sein. Dies nennt man eine eigene **Objektidentität**. Die Objektidentität wird über einen eindeutigen Objektbezeichner ausgedrückt.

Dieser Bezeichner wird bei der Erstellung des Objekts vergeben. Ihm wird die Referenz auf das Objekt zugewiesen. Der Bezeichner identifiziert das Objekt so lange eindeutig, bis es nicht mehr existiert.

Diese Objektidentität ist eindeutig und unabhängig von den anderen Eigenschaften eines Objekts.

Die Objektidentität und der Objektbezeichner lassen sich mit dem Bezeichner einer Variablen vergleichen. Wie bei anderen Variablenbezeichnern innerhalb eines Gültigkeitsbereichs muss dieser Bezeichner immer eindeutig sein und auch dann, wenn es zufällig zwei Objekte gleichen Typs mit identischen Attributwerten gibt, ist eine eindeutige Unterscheidung möglich.

## Assoziationen und UML

Wenn man in der OOP in einer Klasse eine Eigenschaft vom Typ einer anderen Klasse verwendet, wird eine sogenannte **Assoziation** (Beziehung) zwischen dem aufrufenden und dem verwendeten Objekt bzw. den Klassen aufgebaut. Diese sind in einer Sprache mit dem Namen **UML** (Unified Modeling Language) genauer beschrieben.

Diese Sprache ist keine Programmiersprache im eigentlichen Sinn, sondern eine formale Metasprache, um Personen mit unterschiedlichen Kenntnissen eine gemeinsame Basis zu bieten. Beim Erstellen einer größeren Applikation sind ja nicht nur Programmierer beteiligt. Es gibt Leute, die Fachvorgaben machen, andere analysieren ein Problem (im Fall der Objektorientierung spricht man von der objektorientierten Analyse – OOA). Und dann gibt es Fachleute, die ein objektorientiertes Modell schaffen, das später konkret programmiert werden soll. Das nennt man das objektorientierte Design (OOD).

Das Problem ist, dass diese verschiedenen Gruppen, die allesamt wichtig für ein Programmierprojekt sind, meist sehr unterschiedliche Kenntnisse haben und oft auch über keine gemeinsame (Fach-)Sprache verfügen. Um eine gemeinsame Kommunikation zu gestatten, greift man deshalb in vielen professionellen Projekten auf eine formale Metasprache wie UML mit graphischen Modellen und Diagrammen sowie einer symbolischen Syntax zurück, die die Strukturen und Abläufe in einer Applikation verdeutlichen soll.

Eine Assoziation in UML beschreibt im häufigsten Fall eine Verbindung zwischen genau zwei Klassen. Es kann auch die Beziehung zwischen mehreren Klassen beschrieben werden oder aber zwischen anderen Strukturen in der OOP wie Schnittstellen, Objekten, Fallbeschreibungen etc.

Wenn nun ein Kunde ein Konto hat, wie es im obigen Beispiel der Fall ist, ist die Beziehung zwischen dem Kunden und dem Konto eine sogenannte **Aggregation** oder **Komposition** (die exakte Unterscheidung ist nicht ganz einfach, da sie nicht standardisiert ist). Man kann sich das so vorstellen, dass ein Objekt ein Teil eines anderen Objekts ist. Das Konto ist nach der Erzeugung eines Objekts vom Kunden Teil des Kundenobjekts.

## 4.8 Ein erster OO-Aufbau einer Webseite

Im Laufe der Unterlage soll eine Webseite mit einem objektorientierten Design entwickelt werden. Das bedeutet, dass sich die gesamte Webseite aus einem Hauptobjekt entwickeln soll. Später wird daraus das gesamte Grundgerüst samt einer Verknüpfung zu externen Ressourcen (Style Sheets) erzeugt. Zunächst soll hier nur eine erste Logik zu sehen sein, um nicht den Fokus zu verlieren.

### Eine Hauptklasse und eine `main()`-Methode

Nun soll das Projekt um eine weitere Klasse erweitert werden, die dann ein Objekt vom Typ der Klasse `Kunde` erstellt und auch aktiv Informationen in einer generierten Webseite ausgibt. Dabei verwenden wir eine zentrale Klasse mit Namen `Program` (und der Datei `Program.class.php`), die den vollständigen „Programmablauf“ beinhalten soll und dazu nur eine Hauptmethode bereitstellt.

Wenn Sie wollen, dass diese Datei automatisch vom Browser angezeigt wird, wenn ein Anwender nur den DNS-Namen Ihrer Webseite (also ohne explizite Datei) angibt, wird diese Datei in der Praxis meist `index.php` genannt. In dieser Unterlage soll darauf verzichtet werden, um die Einheitlichkeit der Konventionen zu gewährleisten. Es gibt natürlich auch noch andere Mittel als das Benennen mit `index.php`, um ein automatisches Laden zu garantieren. Sie können auf dem Webserver eine zusätzliche Datei `index.php` bereitstellen, die mit einer `header()`-Anweisung zu dieser Datei dann weiterleitet (Stichwort Landingpage).

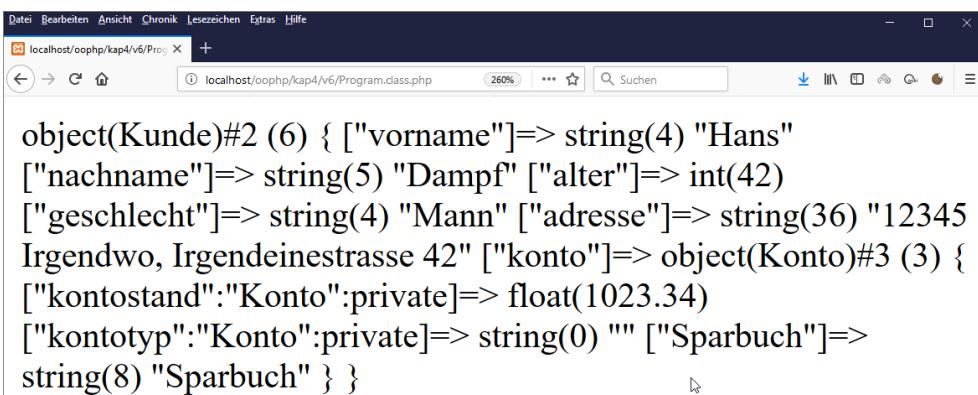
Dieser Ansatz wird in vielen objektorientierten Sprachen wie Java oder C# sogar erzwungen und damit wird auch ein PHP-Projekt „wirklich“ objektorientiert, sofern man nicht durch Funktionen, globale Variablen etc. dieses Konzept wieder torpediert (Verzeichnis v6).

```
<?php
require_once ("Kunde.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    private function main() {
        $kunde = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42",
            new Konto(floatval("1023.34"), "Sparbuch"));
        var_dump($kunde);
    }
}
$main = new Program();

?>
```

Eine Klasse, die den vollständigen „Programmablauf“ beinhaltet

- ✓ Als erstes muss in der PHP-Datei der Klasse `Program` die PHP-Datei mit der Deklaration der Klasse `Kunde` eingebunden werden, denn diese Klasse wird hier instanziert und dann in Form einer Assoziation verwendet (`require_once ("Kunde.class.php");`). Die Klasse `Konto` steht ja bereits durch die Inklusion in der Datei für die Klasse `Kunde` zur Verfügung.
- ✓ Da die Klasse `Program` keinen parametrisierten Konstruktor haben soll, müsste man diesen eigentlich nicht redefinieren. Da aber bei der Erzeugung eines Objekts der Klasse automatisch die `main()`-Methode gestartet werden soll, ist es sinnvoll einen parameterlosen Konstruktor zu redefinieren und dann diesen aufzurufen (`$this -> main();`). Das ist elegant und objektorientiert gedacht. Natürlich könnten Sie auch in dem Konstruktor selbst alle Logik unterbringen und auf die explizite Notation der `main()`-Methode verzichten.
- ✓ Die einzige „prozedurale Verschmutzung“ dieses Ansatzes besteht darin, dass irgendwo das zentrale Objekt, welches das Programm repräsentiert, erzeugt werden muss. Und das macht die Anweisung `$main = new Program();`. Der Rest kann auch in PHP vollkommen objektorientiert durchgeführt werden. Die Zuweisung einer Referenz auf das Objekt, welches das Programm selbst repräsentiert, zu einer Variablen wie `$main` ist im Grunde nicht notwendig und wird in den folgenden Versionen dieses Beispielprojekts auch weggelassen. In dem Fall erzeugt man das Objekt anonym, was im nächsten Schritt erläutert wird.
- ✓ Was noch beachtet werden muss – die Methode `main()` wird hier `private` erklärt. In vielen anderen OO-Sprachen, deren Ansatz hier als Vorbild dient, wird diese Methode als `public` deklariert. Das könnten wir auch machen, aber mit der Privatisierung wird der Ansatz des Information Hidings verfolgt.
- ✓ In der `main()`-Methode wird der Konstruktor der Klasse `Kunde` mit den geforderten Parametern aufgerufen. Beachten Sie, dass das Konto auch erst an dieser Stelle erzeugt wird – und zwar ohne einer Variablen eine Referenz darauf zuzuweisen. Das nennt man dann ein anonymes Objekt und diese Art der Instanziierung eine anonyme Instanziierung. Das wird in der Praxis immer dann gemacht, wenn man nach der Instanziierung keine Referenz auf das Objekt mehr benötigt oder diese auf anderem Weg (wie in dem Beispiel über ein Objekt vom Typ `Kunde`) bereitsteht.
- ✓ Mit `var_dump()` werden erst einmal alle Informationen zu dem als Parameter ausgegebenen Objekt ausgegeben.



The screenshot shows a browser window with the URL `localhost/oophp/kap4/v6/Program.class.php`. The page content displays the output of the `var_dump` function for a variable named `kunde`. The output is as follows:

```
object(Kunde)#2 (6) { ["vorname"]=> string(4) "Hans"
["nachname"]=> string(5) "Dampf" ["alter"]=> int(42)
["geschlecht"]=> string(4) "Mann" ["adresse"]=> string(36) "12345
Irgendwo, Irgendeinestrasse 42" ["konto"]=> object(Konto)#3 (3) {
["kontostand":"Konto":private]=> float(1023.34)
["kontotyp":"Konto":private]=> string(0) "" ["Sparbuch"]=>
string(8) "Sparbuch" } }
```

*Das Objekt vom Typ `Kunde` wurde erzeugt und die enthaltenen Informationen ausgegeben.*

In PHP kann der Konstruktor auch explizit von einem bestehenden Objekt aufgerufen werden. Dieses Vorgehen ist bei etlichen anderen OOP-Sprachen nicht möglich. Der direkte Aufruf des Konstruktors führt zu einer Reinitialisierung der im Konstruktor verwendeten Eigenschaften. Für das vorige Beispiel wäre z. B. folgende Codezeile für ein bestehendes Objekt \$kunde denkbar:

```
$kunde ->__construct ("Otto", "Dort", 23, "Mann",
"12346 Irgendwas, Irgendeineanderestrasse 4711",
new Konto(floatval("4023.34"), "Sparbuch"));
```

Da die Ausnutzung von Besonderheiten einer Sprache hinsichtlich der Klarheit und Lesbarkeit für Umsteiger negativ ist und es keine wesentlichen Vorteile der Syntax gibt, sollte man von dieser Konstruktion in der Regel Abstand nehmen. Aus didaktischen Gründen wird das Verfahren aber in der Folge bei dem Vorantreiben des „Kapitelprojekts“ immer wieder vorgestellt.

## 4.9 Objekte löschen – Garbage Collector und Destruktor

Objekte müssen nach der Erzeugung irgendwann auch wieder beseitigt werden. Das passiert spätestens dann, wenn das Skript zu Ende ist. Aber auch während das Skript noch abgearbeitet wird, können Objekte gelöscht werden. In diesem Fall handelt es sich dann aber meist um sehr umfangreiche Skripte.

Dieses Beseitigen von Objekten geschieht in PHP in der Regel automatisch durch einen Mechanismus, den man **Garbage Collector** oder **Garbage Collection** nennt. Das Laufzeitsystem räumt schon während der Abarbeitung des Skripts im Hintergrund alle Objekte auf, auf die es keine Referenzen mehr gibt. Oder aber wenn das Ende eines Skripts erreicht wird. Das läuft wie gesagt automatisch im Hintergrund ab und normalerweise kümmert sich ein PHP-Programmierer nicht selbst um die Speicherbereinigung, obwohl es für Sonderfälle einige Funktionen wie `gc_collect_cycles()` gibt. Man kann also beim Löschen von Objekten gewisse Maßnahmen durchführen – das wird hier aber nicht weiter vertieft.

In PHP können Sie mit dem Destruktor das Gegenstück zum Konstruktor definieren. Die Destruktormethode trägt den festgelegten Namen `__destruct()`, die ebenfalls mit zwei Unterstrichen `__` beginnt.

Es handelt sich hierbei um eine meist nicht so wichtige Methode, die aber in einigen Situationen genutzt werden kann, um automatisch „Aufräumarbeiten“ auszuführen, wenn ein Objekt explizit gelöscht wird oder das Ende des Skripts erreicht wird. In der Regel gehören zu den Tätigkeiten der Methode das Freigeben von Ressourcen, das Trennen offener Verbindungen oder das Speichern von Informationen.

Es darf aber nicht falsch verstanden werden – der Destruktor löscht nicht das Objekt. Das macht der Garbage Collector. Der Destruktor wird lediglich beim Löschen des Objekts aufgerufen.

- ! In PHP kann der Destruktor auch explizit von einem bestehenden Objekt aufgerufen werden (ähnlich wie der Konstruktor). Der direkte Aufruf des Destruktors erfolgt dann in Form von `$objekt ->__destruct()`. Allerdings ist diese Notation in der Regel nutzlos und nicht zu empfehlen, da zwar die in der Methode definierten Aktionen ausgeführt werden, das Objekt jedoch nicht gelöscht wird.

Das erste Beispiel zeigt die Definition eines Beispiel-Destruktors und seine Funktionalität bei Erreichen des Skriptendes. Zuerst wird ein Destruktor in der Programmdatei angelegt (Verzeichnis v7).

```
<?php
require_once("Kunde.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    public function __destruct() {
        echo "<br />Objekt wird gelöscht...";
    }
    public function main(){
        $kunde = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42",
            new Konto(floatval("1023.34"), "Sparbuch"));
        var_dump($kunde);
    }
}
new Program();
?>
```

Die Klasse, die den vollständigen „Programmablauf“ beinhaltet, redefiniert einen Destruktor

- ✓ Die Destruktormethode wird deklariert. Sie enthält nur die Anweisung einer Ausgabe, dass Objekte gelöscht werden. Dabei handelt es sich konkret um die Meldung, die beim Löschen des „Programmobjekts“ ausgelöst wird. Beachten Sie, dass Sie die Destruktormethode nicht zwingend als `public` deklarieren müssen, da dies Vorgabe in PHP ist. Aber um einen einheitlichen Stil durchzuhalten, sollte man es tun, wenn man auch sonst Methoden explizit als `public` auszeichnet.
- ✓ Wie die Ausgabe des Beispiels am Bildschirm zeigt, wird bei Skriptende der Destruktor aufgerufen, denn das Objekt, was das Programm repräsentiert, wird gelöscht.
- ✓ Die Erzeugung des Objekts, welches das Programm selbst repräsentiert, erfolgt nun anonym.

```
F:\xampp\htdocs\oophp\kap4\v7\Program.class.php:17
object(Kunde) [2]
    public 'vorname' => string 'Hans' (length=4)
    public 'nachname' => string 'Dampf' (length=5)
    public 'alter' => int 42
    public 'geschlecht' => string 'Mann' (length=4)
    public 'adresse' => string '12345 Irgendwo, Irgendeinestrasse 42' (length=36)
    public 'konto' =>
        object(Konto) [3]
            private 'kontostand' => float 1023.34
            private 'kontotyp' => string '' (length=0)
            public 'Sparbuch' => string 'Sparbuch' (length=8)

Objekt wird gelöscht...
```

Der Destruktor wurde aufgerufen.

Nun kann ein Destruktor auch in den anderen Klassen des Projekts deklariert werden, beispielsweise für das Konto. Das Listing soll die modifizierte Kontoklasse sein:

```
<?php

class Konto {
    private $kontostand = 0;
    private $kontotyp = "";
    public function __construct(float $kontostand, string $kontotyp) {
        $this -> kontostand = $kontostand;
        $this -> setKontotyp($kontotyp);
    }
    public function __destruct() {
        echo "<br />Objekt mit dem Kontostand " . $this -> kontostand .
            " wird gelöscht...";
    }
    public function einzahlen(float $betrag) {
        $this -> kontostand += $betrag;
    }
    public function auszahlen(float $betrag) {
        $this -> kontostand -= $betrag;
    }
    public function setKontotyp(string $kontotyp) {
        $this -> kontotyp = $kontotyp;
    }
    public function getKontotyp(): string{
        return $this -> kontotyp;
    }
    public function getKontostand(): float{
        return $this -> kontostand;
    }
}
?>
```

*Die Klasse für das Konto redefiniert nun auch einen Destruktor*

Und das ist die modifizierte Klasse für den Kunden, wo auch der Destruktor redefiniert werden soll:

```
<?php
require_once("Konto.class.php");
class Kunde {
    public $vorname = "";
    public $nachname = "";
    public $alter = 0;
    public $geschlecht = "";
    public $adresse = "";
    public $konto = null;

    public function __construct(string $vorname, string $nachname,
        int $alter, string $geschlecht, string $adresse, Konto $konto)
```

```

{
    $this -> vorname = $vorname;
    $this -> nachname = $nachname;
    $this -> alter = $alter;
    $this -> geschlecht = $geschlecht;
    $this -> adresse = $adresse;
    $this -> konto = $konto;
}

public function __destruct() {
    echo "<br />Objekt mit dem Namen " . $this -> vorname . " ".
        $this -> nachname ." wird gel&ouml;scht...";
}

}
?>

```

Und auch die Klasse für das Konto redefiniert einen Destruktor

Beachten Sie, dass in beiden neuen Destruktoren über `$this` auf das aktuelle Objekt zugegriffen wird, um damit etwa spezifische Eigenschaften von dem gerade gelöschten Objekt zu nutzen. In dem einfachen Beispiel werden nur Werte von Eigenschaften ausgegeben, aber das kann man natürlich viel umfangreicher nutzen.

The screenshot shows a browser window with the following details:

- Title Bar:** Datei, Bearbeiten, Ansicht, Chronik, Lesezeichen, Extras, Hilfe.
- Address Bar:** localhost/oophp/kap4/v7/Prog
- Toolbar:** Back, Forward, Home, Search (Suchen), Links, Rights, Refresh, Stop.
- Links:** Meistbesucht, http://localhost/phpk..., Google, Wikipedia, Facebook, Twitter, LinkedIn.
- Content Area:**
  - Text: F:\xampp\htdocs\oophp\kap4\v7\Program.class.php:17:
  - Object (Kunde) [2]
    - public 'vorname' => string 'Hans' (length=4)
    - public 'nachname' => string 'Dampf' (length=5)
    - public 'alter' => int 42
    - public 'geschlecht' => string 'Mann' (length=4)
    - public 'adresse' => string '12345 Irgendwo, Irgendeinestrasse 42' (length=36)
    - public 'konto' =>
      - object (Konto) [3]
        - private 'kontostand' => float 1023.34
        - private 'kontotyp' => string '' (length=0)
        - public 'Sparbuch' => string 'Sparbuch' (length=8)
  - Text: Objekt mit dem Namen Hans Dampf wird gelöscht...
  - Text: Objekt mit dem Kontostand 1023.34 wird gelöscht...
  - Text: Objekt wird gelöscht...

Mehrere Objekte wurden aufgerufen und jedes Mal der dazu spezifische Destruktor.

Beachten Sie ebenfalls die Reihenfolge, in der die Objekte gelöscht werden. Wobei diese **nicht** deterministisch ist und Sie sich niemals darauf verlassen dürfen, dass Objekte in einer bestimmten Reihenfolge gelöscht werden.

## Ein Objekt mit **unset()** löschen

Man kann Objekte auch gezielt löschen (also nicht einfach durch das Erreichen des Skriptendes) und auch dann wird implizit der Destruktor des jeweiligen Objekts aufgerufen. Das zeigt die Abwandlung des bisherigen Codes im Verzeichnis v8, in dem ein Objekt gezielt mit `unset()` gelöscht wird. Wobei man etwas genauer sein sollte – mit `unset()` wird die Referenz auf das Objekt beseitigt und dann kann der Garbage Collector das Objekt löschen:

```
<?php
require_once("Kunde.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    public function __destruct() {
        echo "<br />Objekt wird gelöscht...";
    }
    public function main() {
        $kunde1 = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42",
            new Konto(floatval("1023.34"), "Sparbuch"));
        $kunde2 = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42",
            new Konto(floatval("1023.34"), "Sparbuch"));
        $kunde1 ->__construct ("Otto", "Dort", 23, "Mann",
            "12346 Irgendwas, Irgendeineanderestrasse 4711",
            new Konto(floatval("4023.34"), "Sparbuch"));
        var_dump($kunde1);
        var_dump($kunde2);
        unset($kunde1);
    }
}
$main = new Program();
?>
```

Mit `unset()` wird ein Objekt beseitigt und das ruft implizit den Destruktor auf.

The screenshot shows a web browser window with the URL `localhost/oophp/kap4/v8/Prog`. The page content displays the output of a PHP script. It starts with a message about deleting an object with a balance of 1023.34. Below this, two objects of type `Kunde` are shown, each with a nested `Konto` object. The first object's `Konto` has a balance of 4023.34. The second object's `Konto` has a balance of 1023.34. Following this, several deletion messages are listed: "Object with name Otto Dort deleted...", "Object with balance 4023.34 deleted...", "Object with name Hans Dampf deleted...", "Object with balance 1023.34 deleted...", and "Object deleted...".

```

Objekt mit dem Kontostand 1023.34 wird gelöscht...
F:\xampp\htdocs\oophp\kap4\v8\Program.class.php:20:
object(Kunde) [2]
    public 'vorname' => string 'Otto' (length=4)
    public 'nachname' => string 'Dort' (length=4)
    public 'alter' => int 23
    public 'geschlecht' => string 'Mann' (length=4)
    public 'adresse' => string '12346 Irgendwas, Irgendeineanderestrasse 4711' (length=45)
    public 'konto' =>
        object(Konto) [6]
            private 'kontostand' => float 4023.34
            private 'kontotyp' => string '' (length=0)
            public 'Sparbuch' => string 'Sparbuch' (length=8)

F:\xampp\htdocs\oophp\kap4\v8\Program.class.php:21:
object(Kunde) [4]
    public 'vorname' => string 'Hans' (length=4)
    public 'nachname' => string 'Dampf' (length=5)
    public 'alter' => int 42
    public 'geschlecht' => string 'Mann' (length=4)
    public 'adresse' => string '12345 Irgendwo, Irgendeinestrasse 42' (length=36)
    public 'konto' =>
        object(Konto) [5]
            private 'kontostand' => float 1023.34
            private 'kontotyp' => string '' (length=0)
            public 'Sparbuch' => string 'Sparbuch' (length=8)

Objekt mit dem Namen Otto Dort wird gelöscht...
Objekt mit dem Kontostand 4023.34 wird gelöscht...
Objekt mit dem Namen Hans Dampf wird gelöscht...
Objekt mit dem Kontostand 1023.34 wird gelöscht...
Objekt wird gelöscht...

```

*Mehrere Objekte wurden aufgerufen und jedes Mal dazu der spezifische Destruktor.*

- ✓ In dem Beispiel werden zwei Objekte mit identischen Werten erzeugt.
- ✓ Dann wird eines der Objekte durch den impliziten Konstruktorauftrag in den Werten verändert.
- ✓ Die folgende Ausgabe zeigt die Werte der beiden Objekte.
- ✓ Danach wird ein Objekt mit `unset()` zum Löschen freigegeben, was zum Einsatz des Garbage Collectors und damit auch zum Aufruf des Destruktors führt. Beachten Sie aber auch hier die Reihenfolge, in der Meldungen auf der Webseite erscheinen und wie und wann die Objekte gelöscht werden. Das Beispiel macht ganz deutlich, dass diese nicht deterministisch ist. Die explizite Verwendung des Destruktors durch eine Redefinition erfolgt – wie schon angedeutet – auch deshalb eher selten. Auch das gezielte Löschen von Objekten bzw. das Freigeben der Referenzen auf das Objekt erfolgt ebenso eher selten, sondern man überlässt die Beseitigung meist vollständig dem Garbage Collector.

## 4.10 Objekte klonen

Wenn Sie aus einer Klasse ein neues Objekt erstellen, erhält es eine Kopie aller Eigenschaften der Klasse mit den vorgesehenen Vorgabewerten. Manchmal brauchen Sie aber auch ein neues Objekt, dessen Werte davon abweichen und das in einer veränderten Form bereits vorliegt. Sie wollen also eine Kopie eines bestehenden Objektes mit den zu diesem Zeitpunkt gültigen Werten aller Eigenschaften – einen sogenannten Klon – erzeugen. Obwohl dieser Vorgang nur selten notwendig ist, soll er der Vollständigkeit halber behandelt werden.

## Klon versus Referenz

Wenn Sie ein bestehendes Objekt einer Variablen zuweisen, erzeugen Sie nur eine zusätzliche Referenz auf dasselbe Objekt und es wird kein weiteres Objekt angelegt. Etwa so:

```
$kopie = $original
```

Beide Variablen referenzieren nun einfach auf das gleiche Objekt.

In PHP gibt es das Schlüsselwort `clone`, mit dem ein neues Objekt erzeugt werden kann, dessen Eigenschaften samt deren Werten exakt dem geklonten Objekt entsprechen. Das erfolgt in der Form:

```
$kopie = clone $original
```

Leider weicht PHP aber auch beim Klonen von Objekten mit dieser Syntax massiv von der sonst üblichen Syntax beim Klonen ab, bei der jedes Objekt bereits eine implizite Methode zum Klonen besitzt, die einfach über das Objekt aufgerufen wird (es ist also dort kein zusätzliches, „externes“ Schlüsselwort notwendig).

Wobei es auch in PHP so eine integrierte Klonmethode gibt, die Sie dann redefinieren müssen, wenn Sie keine reine Kopie des Ausgangsobjektes erstellen, sondern einige Werte anpassen wollen.

Dann redefinieren Sie in Ihrer Klasse die vorgegebene Standardmethode `__clone()`. In dieser Methode nehmen Sie die gewünschten Änderungen an den Werten des Ausgangsobjektes vor, wobei die Methode keine Argumente akzeptiert und von einem Objekt nicht direkt aufgerufen werden kann (was eben sonst beim Klonen in anderen Sprachen die typische Vorgehensweise ist). Man spricht hier von benutzerdefiniertem Klonen.

Wird die Methode `__clone()` nicht redefiniert, erhält das geklonte Objekt automatisch eine vollständige Kopie der Werte des Ausgangsobjektes. Dies wird als vordefiniertes Klonen bezeichnet.

Im Verzeichnis `v9` finden Sie die Modifizierungen des Kapitelprojekts, bei der Objekte geklont werden. Dabei soll die Klasse `Konto` eine explizite `__clone()`-Methode bereitstellen.

```
<?php
class Konto {
    private $kontostand = 0;
    private $kontotyp = "";
    public function __construct(float $kontostand, string $kontotyp) {
        $this -> kontostand = $kontostand;
        $this -> setKontotyp($kontotyp);
    }
    public function __destruct() {
        echo "<br />Objekt mit dem Kontostand " . $this -> kontostand .
        " wird gelöscht...";
    }
    public function __clone() {
        $this -> kontostand = $this -> kontostand + 100;
    }
    public function einzahlen(float $betrag) {
```

```

    $this -> kontostand += $betrag;
}
public function auszahlen(float $betrag) {
    $this -> kontostand -= $betrag;
}
public function setKontotyp(string $kontotyp) {
    $this -> kontotyp = $kontotyp;
}
public function getKontotyp(): string{
    return $this -> kontotyp;
}
public function getKontostand(): float{
    return $this -> kontostand;
}
}
?>

```

*Die Klasse für das Konto redefiniert nun auch einen Destruktor*

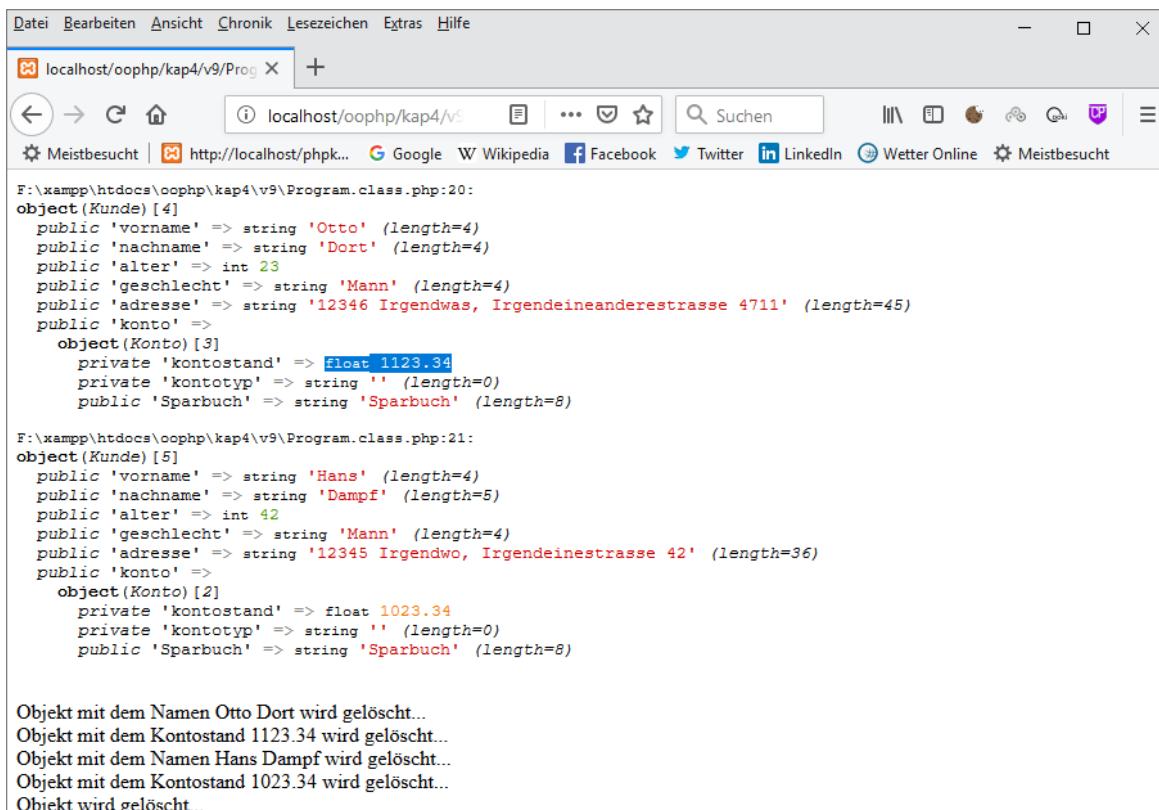
Die Methode `__clone()` sorgt dafür, dass das geklonte Objekt nicht alle Eigenschaften des Ausgangsobjekts unverändert erhält, sondern einen anderen Wert für die Eigenschaft `$kontostand`. Dieser wird um 100 erhöht. Der Wert für den Kontotyp wird unverändert geklont. Beachten Sie, dass Sie die Methode nicht zwingend als `public` deklarieren müssen, da dies Vorgabe in PHP ist. Wie bei allen anderen Methoden (auch den besonderen) gilt, dass man es jedoch tun sollte, wenn man auch sonst Methoden explizit als `public` auszeichnet.

```

<?php
require_once("Kunde.class.php");
class Program {
    public function __construct(){
        $this -> main();
    }
    public function __destruct() {
        echo "<br />Objekt wird gelöscht...";
    }
    public function main() {
        $kontol = new Konto(floatval("1023.34"), "Sparbuch");
        $kontonew = clone $kontol;
        $kunde1 = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42", $kontol);
        $kunde2 = clone $kunde1;
        $kunde1 ->__construct ("Otto", "Dort", 23, "Mann",
            "12346 Irgendwas, Irgendeineanderestrasse 4711", $kontonew);
        var_dump($kunde1);
        var_dump($kunde2);
    }
    new Program();
?>

```

- ✓ In der Programmklasse wird erst einmal ein Objekt vom Typ Konto erzeugt.
- ✓ Das Objekt wird geklont und da diese Klasse Konto eine explizit `__clone()`-Methode besitzt, wird diese aufgerufen und der Kontostand um den Wert 100 beim Klonen erhöht.
- ✓ Beim Erzeugen eines Objekts von Typ Kunde wird das erste Kontoobjekt (das Original) verwendet.
- ✓ Dann wird der Kunde geklont und da die Klasse für den Kunden keine explizite `__clone()`-Methode besitzt, wird eine identische Kopie von dem Objekt angelegt.
- ✓ Der erste Kunde wird dann in den Werten durch den Aufruf des Konstruktors über das Objekt verändert und dabei wird das geklonte Konto verwendet.
- ✓ Die folgenden Ausgaben zeigen insbesondere den höheren Wert des Kontostands für das zweite Objekt vom Typ Konto.



The screenshot shows a web browser window with the URL `localhost/oophp/kap4/v9/Prog`. The page content displays the output of a PHP script. The script defines two classes, `Kunde` and `Konto`, and creates instances of both. It then performs cloning operations and prints messages indicating which objects are being deleted.

```

F:\xampp\htdocs\oophp\kap4\v9\Program.class.php:20:
object(Kunde) [4]
public 'vorname' => string 'Otto' (length=4)
public 'nachname' => string 'Dort' (length=4)
public 'alter' => int 23
public 'geschlecht' => string 'Mann' (length=4)
public 'adresse' => string '12346 Irgendwas, Irgendeineandererstrasse 4711' (length=45)
public 'konto' =>
    object(Konto) [3]
        private 'kontostand' => float 1123.34
        private 'kontotyp' => string '' (length=0)
        public 'Sparbuch' => string 'Sparbuch' (length=8)

F:\xampp\htdocs\oophp\kap4\v9\Program.class.php:21:
object(Kunde) [5]
object(Konto) [2]
public 'vorname' => string 'Hans' (length=4)
public 'nachname' => string 'Dampf' (length=5)
public 'alter' => int 42
public 'geschlecht' => string 'Mann' (length=4)
public 'adresse' => string '12345 Irgendwo, Irgendeinestrasse 42' (length=36)
public 'konto' =>
    object(Konto) [2]
        private 'kontostand' => float 1023.34
        private 'kontotyp' => string '' (length=0)
        public 'Sparbuch' => string 'Sparbuch' (length=8)

Objekt mit dem Namen Otto Dort wird gelöscht...
Objekt mit dem Kontostand 1123.34 wird gelöscht...
Objekt mit dem Namen Hans Dampf wird gelöscht...
Objekt mit dem Kontostand 1023.34 wird gelöscht...
Objekt wird gelöscht...

```

*Benutzerdefiniertes Klonen und vordefiniertes Klonen*

## 4.11 Anonyme Klassen

In PHP 7 wurde die Unterstützung sogenannter **anonymer Klassen** hinzugefügt. Das Konzept kennt man in einigen OO-Sprachen wie Java schon länger. Dabei wird eine Klassendeklaration ohne Namen innerhalb eines Methodenparameters notiert. Damit man diese allerdings verwenden kann, muss sie unmittelbar instanziert werden.

So könnte das aussehen:

```
...
$obj->meineMethode(new class {
    public function ausgabe($msg)
    {
        echo $msg;
    }
});
```

*Eine anonyme Klassendeklaration, die direkt instanziert wird*

## 4.12 Klassenmember und der Gültigkeitsbereichsoperator

Normalerweise greift man auf Eigenschaften und Methoden über ein vorangestelltes Objekt zu. Es gibt aber auch die Möglichkeit, dass man über eine Klasse auf Methoden und Attribute zugreift, ohne ein Objekt der Klasse zu instanzieren. Man spricht in dem Fall von den sogenannten **statischen Methoden** – auch **Klassenmethoden** genannt – beziehungsweise **statischen Attributen** – auch **Klassenattribute** genannt. Oder allgemein von **Klassenmembers** oder statischen Elementen einer Klasse.

Diese werden im Quellcode innerhalb der Klasse mit dem Schlüsselwort **static** deklariert, was die Bezeichnung als „statisches“ Element erklärt.

Beispiel (Verzeichnis **Klassenmember**):

```
<?php
class Person {
    public static $firma = "Schaffe + Schaffe";
}
```

Die Eigenschaft **\$firma** ist als Klassenmember deklariert und kann in der Folge in der „Programmkasse“ direkt über die vorangestellte Klasse **Person** verwendet werden.

```
<?php
require_once("Person.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    public function main(){
        echo Person :: $firma;
    }
}
new Program();
?>
```

Der Zugriff auf Klassenmember erfolgt jedoch nicht über die Pfeilnotation mit `->`, sondern über den **Gültigkeitsbereichsoperator `::` (Paamayim Nekudotayim)**.

**!** Auch hier weicht PHP leider von der sonst üblichen Syntax ab, denn fast alle anderen OO-Sprachen verwenden für den Zugriff auf Klassenmember den Punkt. Damit sind die Zugriffe über Objekte und Klassen auf Member vor allen Dingen einheitlich. Und eine weitere Inkonsistenz in PHP gilt es zu beklagen, denn im Gegensatz zum Zugriff auf Instanzvariablen, wo beim Namen des Attributs das `$`-Zeichen unterbleiben muss, muss es beim Zugriff auf Klassenvariablen hinter dem Operator zwingend notiert werden.

Natürlich kann man auch Methoden mit `static` als Klassenmethoden auszeichnen. Beachten Sie die Erweiterungen des Codes:

```
<?php
class Person {
    public static $firma = "Schaffe + Schaffe";

    public static function ausgeben(int $wert) {
        echo "<hr />Der Wert ist $wert";
    }
}
?>
```

Die Methode `ausgeben()` ist jetzt auch als Klassenmember deklariert und kann in der Folge in der „Programmkasse“ direkt über die vorangestellte Klasse `Person` verwendet werden.

```
<?php
require_once("Person.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    public function main(){
        echo Person :: $firma;
        Person :: ausgeben(42);
    }
}
```

```

    }
}

new Program();
?>

```

*Die Programmklasse mit dem Zugriff auf Klassenmember von Person*

Sie können in PHP auf statische Elemente nicht nur über die Klasse, sondern auch über eine Instanz der Klasse zugreifen. In einigen Sprachen ist das möglich, aber es wird grundsätzlich nicht empfohlen. Auf Klassenelemente greift man immer nur über die Klasse zu. Wenn Sie das in PHP jedoch tun wollen, dürfen Sie nicht die Pfeilnotation, sondern müssen den Gültigkeitsbereichsoperator verwenden, dem in diesem Fall das Objekt vorangestellt wird. Etwa so:

```
$p :: ausgeben(2);
```

Eine Klasse kann natürlich neben statischen Elementen auch Instanzmember beinhalten. Etwa so wie in dieser kleinen Erweiterung:

```

<?php
class Person {
    public static $firma = "Schaffe + Schaffe";
    public $name = "Hans";
    public static function ausgeben(int $wert) {
        echo "<hr />Der Wert ist $wert";
    }
}
?>

```

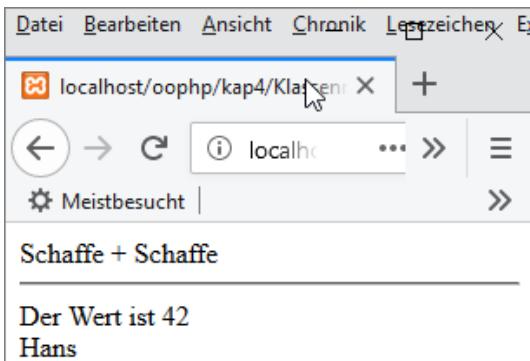
*Instanz- und Klassenelemente gemeinsam*

```

<?php
require_once("Person.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    public function main(){
        echo Person :: $firma;
        Person :: ausgeben(42);
        $p = new Person();
        echo "<br />" . $p -> name;
    }
}
new Program();
?>

```

*Zugriff erst über eine Klasse und dann über die Instanz*



*Mal Klasse, mal Instanz*

Der Einsatz von Klassenelementen sollte allgemein mit großer Vorsicht erfolgen. Da die Werte in allen Instanzen verfügbar sind, werden sich Änderungen auch global auswirken und das kann zu unangenehmen Randeffekten führen. Man setzt Klassenelemente eigentlich nur dann ein, wenn

- ✓ Informationen eben in allen Instanzen identisch bereitstehen sollen,
- ✓ das Instanziieren von verschiedenen Objekten keinen Sinn macht, weil sich die Objekte in ihren Eigenschaften und Methoden nicht unterscheiden (etwa weil sie nur konstante Werte beinhalten) und/oder
- ✓ das Instanziieren einer Klasse unnötig aufwändig ist.

Sie können aus einer Klassenmethode nicht auf Instanzelemente zugreifen und auch `$this` steht dort nicht zur Verfügung. So etwas ginge also **nicht**, wenn `$name` eine Instanzvariable wäre:

```
public static function ausgeben(int $wert) {
    echo "<hr />Der Wert ist $wert" . $this->name;
}
```

Wenn man etwas darüber nachdenkt, ist dies offensichtlich. Denn es ist ja noch gar kein Objekt vorhanden, auf das man zugreifen könnte. Klassenelemente existieren explizit, ohne dass ein Objekt der Klasse erzeugt sein muss. Das ist der Sinn der Klassenelemente. Und Instanzelemente existieren erst, wenn eine Instanz erzeugt wurde.

## Konstanten in der OOP und Klassenkonstanten

Konstanten gibt es in PHP schon lange und natürlich auch unabhängig vom objektorientierten Ansatz. Es ist jedoch auch möglich, dass Sie in einer Klasse konstante Werte definieren. Sie stellen damit eine spezielle Form von Eigenschaften dar.

Wie bei PHP-Konstanten üblich, wird beim Bezeichner auf das vorangestellte `$` verzichtet. Konstantenbezeichner werden zudem per Konvention immer vollständig in Großbuchstaben notiert.

Allgemein kann der Wert von Klassenkonstanten ein Literal und seit PHP 5.6.0 auch ein konstanter Ausdruck (etwa eine Addition von zwei Zahlen) sein.

Die standardmäßige Sichtbarkeit einer Klassenkonstante ist `public`, weshalb man das nicht zwingend auszeichnen muss. Genaugenommen sind sogar erst seit PHP 7.1.0 Sichtbarkeitsmodifizierer für Klassenkonstanten erlaubt. Aber auch hier gilt wieder, dass man einen einheitlichen Stil einhalten sollte. Entweder man verzichtet konsequent darauf, optionale Sichtbarkeitsmodifizierer zu notieren oder man notiert sie immer.

Betrachten wir eine kleine Erweiterung des bisherigen Quellcodes:

```
<?php
class Person {
    public static $firma = "Schaffe + Schaffe";
    public $name = "Hans";
public const ORT = "Irgendwo";
    public static function ausgeben(int $wert) {
        echo "<hr />Der Wert ist $wert";
    }
}
?>
```

```
<?php
require_once("Person.class.php");
class Program {
    public function __construct(){
        $this -> main();
    }
    public function main(){
        echo Person :: $firma . ", " . Person :: ORT;
        Person :: ausgeben(42);
        $p = new Person();
        echo "<br />" . $p -> name;
    }
}
new Program();
?>
```

Sie sehen, dass die Konstante `ORT` aus der Klasse `Person` in der Programmklasse über die vorgestellte Klasse `Person` und den Gültigkeitsbereichsoperator verwendet wird. Damit ist das eine Klassenkonstante bzw. ein statisches Element.

Obwohl die Konstante in der Klasse explizit statisch ist, darf man in PHP **nicht** den Modifizierer `static` davor notieren.

Da Sie in PHP im Prinzip auf statische Elemente auch über ein vorangestelltes Objekt und den Gültigkeitsbereichsoperator zugreifen können, können Sie auch auf Klassenkonstanten zugreifen. Etwa so:

```
$p = new Person();
echo $p :: ORT;
```

Wie schon erwähnt, ist von dieser Vorgehensweise aber abzuraten.

## 4.13 Autoloading

Wenn Sie in einer Datei versuchen, eine Klasse zu instanziieren, die nicht definiert ist, bricht PHP die Ausführung mit einem Fehler ab. Das ist eigentlich trivial, denn was nicht deklariert ist, kann nicht verwendet werden – unabhängig von Klassen oder der OOP.

Nur gilt bei der OOP in PHP, dass man jede Klasse in einer eigenen Datei deklarieren sollte. Und es kann sein, dass diese Datei bei der Verwendung der Klassen nicht zur Verfügung steht, weil sie nicht geladen wurde.

Um das zu verhindern, lädt man üblicherweise in einer Seite vorab mittels `require`, `require_once` oder – bei optionalem Einsatz – `include`-Anweisungen alle Klassen, die eventuell in ihr verwendet werden könnten (häufig unabhängig davon, ob sie nun verwendet werden oder nicht). Das führt allerdings dazu, dass die Verarbeitungszeit des Skripts wächst, da der Parser alle Klassen verarbeiten muss, auch die, die ggf. nicht verwendet werden. Eine Lösung ist, fehlende Klassen nach Bedarf automatisch zu laden – mit dem sogenannten **Autoloading**. In dem Zusammenhang sind zwei Funktionen von besonderem Interesse.

### Die Funktion `__autoload()`

Die Funktion `__autoload()`, die – was an den zwei Unterstrichen zu sehen ist – zu den magischen Methoden gehört, bietet eine Möglichkeit zum automatischen Laden erforderlicher Klassen. Dazu wird einfach die Funktion redefiniert – wie ein individueller Konstruktor oder Destruktor – und der Name der zu ladenden Klasse als Parameter notiert. Die Funktion `__autoload()` wird automatisch aufgerufen, wenn ein Objekt der Klasse instanziert wird. Im Inneren führt man dann ein `require_once` aus.

Das sieht formal so aus:

```
function __autoload($className)
{
    require_once $className . '.class.php';
}
```

*Das formale Redefinieren der `__autoload`-Funktion*

Der Parameter `$className` enthält nach dem Aufruf den Namen der benötigten Klasse. Im Inneren wird die Konvention für die Dateinamenserweiterung angewendet und die Erweiterung `.class.php` an den Klassennamen gehängt.

Man sieht an der Stelle erneut, wie wichtig die konsequente Einhaltung von Konventionen und Regeln ist, denn das Ganze funktioniert nur, wenn Sie diese für die Bezeichnungen und auch für den Ablageort der Klassendateien einhalten.

## Die Funktion `spl_autoload_register()`

Die Funktion `__autoload()` ist sehr praktisch, aber unflexibel, denn es darf nur eine Methode mit diesem Namen geben und im Fall von einer komplexeren Logik kann diese nur über entsprechende Kontrollstrukturen im Inneren der Funktion umgesetzt werden. Einfacher wäre es, für jeden Fall eine eigene Funktion für das Autoloading zur Verfügung stellen zu können.

Die Möglichkeit besteht und ist auch die in der PHP-Dokumentation empfohlene Vorgehensweise beim Autoloading. Sie deklarieren dazu eine beliebige Funktion, die die Klassendatei lädt, und registrieren diese individuelle Autoloading-Funktion mit der Funktion `spl_autoload_register()`. Als Parameter erwartet die Funktion den Bezeichner der Autoload-Funktion als String.

**SPL** steht für Standard PHP Library und bezeichnet eine Sammlung von Klassen und Schnittstellen (Interfaces) zur Lösung von Standard-Programmierproblemen. Die Bibliothek gehört seit PHP 5.0 zum Umfang der Standardinstallation.

```
spl_autoload_register("meineAutofunktion");
```

*Registrieren einer eigenen Autoloadfunktion*

Die so registrierte Funktion wird jetzt jedes Mal ausgeführt, wenn ein Objekt einer Klasse instanziert wird, die noch nicht zur Verfügung steht.

Um Fehler beim Laden von Klassendateien abzufangen, kann man mit einer Ausnahmebehandlung arbeiten oder zumindest mit der Funktion `file_exists()` prüfen, ob es die einzubindende Datei überhaupt gibt. Darauf aufbauend können Sie eine vernünftige Logik implementieren und etwa das spätere Instanziieren unterbinden, eine alternative Klassendatei laden, etc.

## 4.14 Übungen

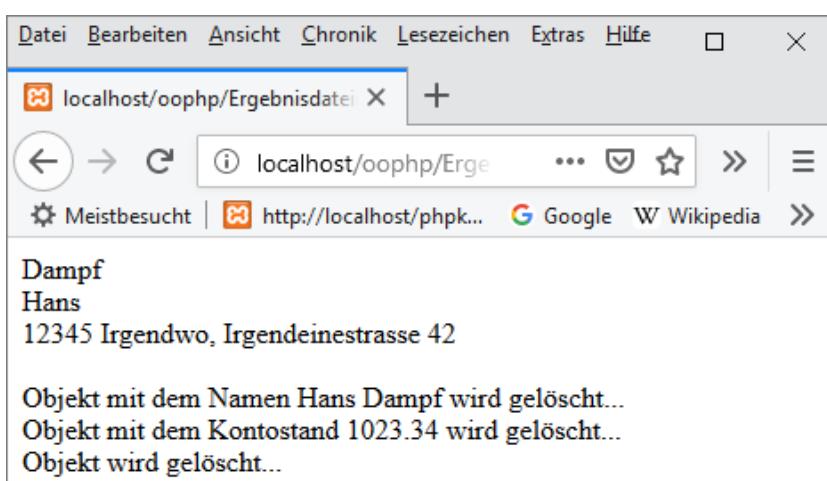
**Übungsdateien:** Verzeichnis v9

**Ergebnisdateien:** Konto.class.php,

Kunde.class.php,

Program.class.php

1. Modifizieren Sie die Klasse Kunde so, dass sämtliche Eigenschaften nur noch über Getter und Setter zugänglich sind.
2. Sämtliche Eigenschaften sollen sowohl zu schreiben als auch zu lesen sein. Einzige Ausnahme ist die Eigenschaft `$konto`. Die Art eines Kontos soll nach der Erzeugung eines Kontoobjekts nicht mehr geändert werden dürfen. Ein einfacher Weg das zu gewährleisten ist, dass hier nur ein Getter implementiert wird. Der Setter muss zwingend wegfallen.
3. Modifizieren Sie den Konstruktor der Klasse Kunde so, dass beim Setzen der Werte der Eigenschaften ausschließlich die Settermethoden verwendet werden. Ausnahme ist wieder die Eigenschaft `$konto`. Da hier kein Setter vorhanden ist, muss diese Eigenschaft direkt gesetzt werden.
4. Erzeugen Sie in der Programmklasse in der `main()`-Methode zuerst ein Objekt vom Typ Konto.
5. Verwenden Sie dieses Objekt beim Erzeugen eines Objekts vom Typ Kunde in dem Konstruktor der Klasse.
6. Lassen Sie sich nacheinander mit `echo`-Anweisungen den Nachnamen, den Vornamen und die Adresse des Kunden ausgeben. Dazu stehen die entsprechenden Getter zur Verfügung.



So ungefähr soll die Ausgabe aussehen.

# 5

## Vererbung

### 5.1 Worum geht es bei Vererbung?

In der OOP werden ähnliche Objekte zu Gruppierungen (Klassen) zusammengefasst und die dort notierten Eigenschaften und Methoden für eine spätere Erzeugung von realen Objekten verwendet. Zentrale Bedeutung hat dabei eine hierarchische Struktur der Gruppierungen, die in der OOP in der Regel zum Einsatz kommt – von allgemein bis fein. Klassen bilden dabei einen sogenannten **Klassenbaum**, der im Folgenden noch genauer erläutert wird.

Diese Strukturierung soll vor allem einen Effekt haben – die Wiederverwendung von Quellcodepassagen, die man bereits hat.

Nähern wir uns dieser Denkweise über ein Beispiel aus der realen Welt.

#### Ein Szenario in der realen Welt

Ein Bankkunde (in der OO-Notation Objekt vom Typ `Kunde`) ist natürlich ein Mensch. Damit ist er in der OO-Notation auch ein Objekt vom Typ `Mensch`. Ein Mensch ist als Typ eine **Verallgemeinerung** eines Kunden. Ein Kunde hingegen eine **Spezialisierung** eines Menschen.

Nun ist jeder Mensch aber selbst wieder ein Lebewesen. Damit ist ein Objekt vom Typ `Mensch` ebenso ein Objekt vom Typ `Lebewesen`. Ein Lebewesen ist also wieder eine Verallgemeinerung von einem Menschen. Und ein Tier wäre beispielsweise auch ein Lebewesen. Das Lebewesen wäre also sowohl von einem Menschen als auch einem Tier eine Verallgemeinerung.

Das Modell solch einer Typbeziehung können Sie beliebig fortsetzen. Aus Sicht des allgemeinen Lebewesens in Richtung einer weiteren Verallgemeinerung (etwa die Elemente in einer bestimmten Region oder in einem Land), aber auch vom Kunden aus in Richtung einer weiteren Spezialisierung (etwa Privatkunde oder Geschäftskunde).

Der konkrete Nutzen der Vererbung liegt darin, dass man bestimmte Eigenschaften und Methoden (Fähigkeiten) für einen allgemeinen Typ beschreiben (definieren) kann und dann werden diese Eigenschaften und Methoden auch für spezialisierte Typen gelten. Sie müssen sie damit dort nicht mehr neu beschreiben (definieren). Es ist offensichtlich, dass Sie sich sehr viel Arbeit ersparen und die Wartbarkeit eines Systems (eines Beschreibungskonzepts) erhöhen können, wenn Sie Typisierungen intelligent vornehmen und verschachteln. In der OOP ist die schon angedeutete Wiederverwendbarkeit das zentrale Argument.

## Superklasse und Subklasse

Wie genau Sie dieses Verfahren zur Beschreibung von Typbeziehungen ausarbeiten ist Ihnen überlassen. Denken Sie daran, dass Objekte in unserer Wahrnehmung, aber auch in der OOP selbst, Abstraktionen der Wirklichkeit darstellen und diese Verallgemeinerungen bzw. Spezialisierungen erst recht. Bezuglich des Modells gibt es aber in der objektorientierten Philosophie gewisse Regeln bzw. Empfehlungen, die das Verfahren erst effektiv machen.

Gemeinsame Erscheinungsbilder sollten in einer möglichst „hohen“, allgemein formulierten Klasse zusammengefasst werden. Erst wenn Unterscheidungen möglich beziehungsweise notwendig sind, die nicht für alle Mitglieder einer Beschreibung gelten können, werden Untergruppierungen – untergeordnete Klassen – gebildet.

**Vererbung** (engl. Inheritance) bezeichnet nun eine Verbindung bzw. Beziehung (Assoziation) zwischen einer Klasse und einer oder mehrerer anderer Klassen, in der die abgeleitete Klasse das Verhalten und den Aufbau der Oberklasse übernimmt, ggf. neues Verhalten und einen erweiterten Aufbau besitzt und eventuell übernommenes Verhalten modifiziert.

Die vererbende Klasse nennt man Basisklasse, Oberklasse, Elternklasse oder **Superklasse**. Die erbende Klasse nennt man abgeleitete Klasse, Unterklassen, Kindklasse oder **Subklasse**. Am üblichsten sind die Bezeichner Superklasse bzw. deren englische Form Superclass sowie Subklasse bzw. Subclass. Wir werden in der Folge hauptsächlich „Superklasse“ und „Subklasse“ als Begriffe verwenden.

Die über Vererbung verknüpften Klassen bilden – wie schon erwähnt – einen so genannten **Klassenbaum**. Dieser kann im Prinzip beliebig tief werden. Ebenso tief, wie es notwendig ist, um eine Problemstellung detailliert zu beschreiben. Vererbung ist über eine beliebige Anzahl von Ebenen hinweg im Klassenbaum möglich.

- ✓ Die oberste Klasse des Baums heißt **Wurzelklasse** (root class).
- ✓ Die Klassen am Ende eines Vererbungsbaumes heißen **Blattklassen** (leaf classes).
- ✓ Klassen, von denen aus keine Unterklassen mehr gebildet werden dürfen, heißen **finale Klassen** (final classes). Finale Klassen können nicht mehr weiter spezialisiert werden.

Kommen wir noch einmal zu Generalisierung und Spezialisierung zurück. Vererbung bezeichnet konkret den Mechanismus, Attribute und Methoden mit Hilfe einer Klassenbeziehung wieder zu verwenden. Bei der Beschreibung der Beziehungen in einem Klassenbaum redet man von Generalisierung bzw. Verallgemeinerung und Spezialisierung. Diese stellen gegensätzliche Sichtweisen auf die gleiche Beziehung dar – nur halt aus der Sicht der Superklasse oder aus der Sicht der Subklassen.

Generalisierung leitet sich als Begriff aus der Tatsache ab, dass die Superklasse ihre Unterklassen verallgemeinert und nur alle Gemeinsamkeiten ihrer Subklassen darstellt.

Eine Subklasse ist immer ein Spezialfall ihrer Superklasse und sollte in jeder Hinsicht mit ihr kompatibel sein. Ein Kunde ist immer auch ein Mensch, also ein Spezialfall eines Menschen.

Eine Vererbungsbeziehung wird in der OOP bzw. UML auch als Ist-ein-Relation (Is-A-Relation) oder Art-von- beziehungsweise Kind-of-Relation bezeichnet.

Der Begriff Spezialisierung leitet sich von der Tatsache ab, dass die Subklassen die Superklasse verfeinern (sie spezialisieren), indem sie neue Klassenelemente hinzufügen beziehungsweise Methoden anders implementieren. Durch die Spezialisierung ist eine Instanz einer Klasse zugleich Instanz aller ihrer Superklassen und jedes öffentlichen Elements, das für eine der Superklassen definiert wurde. Eine Instanz einer Subklasse kann immer dann verwendet werden, wenn eine Instanz einer der Superklassen gefordert wird.

Wenn Sie etwa in unserem theoretischen Modell ein Objekt vom Typ `Mensch` brauchen, können Sie auch ein Objekt `Kunde` nehmen.

## Top-down und Bottom-up

Wenn Sie bei einer Aufgabe versuchen, Klassen zu entwickeln, die in einer Vererbungsbeziehung stehen, können Sie die Analyse und Beschreibung nach verschiedenen standardisierten Methoden vornehmen.

Da gibt es einmal die **Top-down-Methode**. Ausgangspunkt ist dabei die Gesamtaufgabe, die in Teilaufgaben zerlegt wird. Bei komplexen Aufgabenstellungen können die Teilaufgaben in weitere Teilaufgaben aufgeteilt werden. Bei der Entwicklung eines einzelnen Programms ist diese Methode oft sinnvoll.

Die Alternative ist die **Bottom-up-Methode**. Diese Methode ist für Vorgehensweisen interessant, bei denen eine komplette Aufgabenstellung nicht exakt beschrieben ist bzw. noch Änderungen erwartet werden. Einzelne Module werden später zu einem großen Modul zusammengesetzt. Die Bottom-up-Methode gilt meist beim Entwurf großer Programmsysteme als sinnvoll.

Eine Art Mischvariante nennt sich **Up-down-Methode** oder auch Middle-Out bzw. Gegenstromverfahren. Bei dieser Strukturierungsmethode wird die Gesamtaufgabe durch Top-down-Methoden verfeinert und Teilaufgaben werden bottom-up abstrahiert.

## Die technische Umsetzung einer Vererbung

Bei einer Vererbung in der OOP übernimmt die Subklasse alle Attribute und Methoden der Superklasse, soweit diese offengelegt werden. Die Beziehung der ursprünglichen Klasse, der Superklasse zur abgeleiteten, der Subklasse, ist immer streng hierarchisch. Jede Subklasse erbt alle offengelegten Eigenschaften und Methoden ihrer Superklasse.

Sie beinhaltet also immer mindestens die gleichen Eigenschaften und Methoden wie die Superklasse, wobei durch geeignete Zugriffsregeln in der Superklasse Strukturen gegenüber den Subklassen verborgen werden können. Zusätzlich kann (und sollte in der Regel) die Subklasse neue Attribute festlegen oder zumindest die Werte von Attributen verändern und/oder neue Methoden hinzufügen beziehungsweise bestehende Methoden modifizieren. Die abgeleitete Klasse verwendet bei Bedarf die Methoden oder Eigenschaften der Superklasse. Diesen Mechanismus kann man sich analog zur Verwendung von Bibliotheken und dort implementierten Funktionalitäten vorstellen.

## Erben ist nicht kopieren

Abgeleitete Klassen übernehmen beim Erben die offengelegten Eigenschaften und Methoden aller übergeordneten Klassen, wobei Übernehmen nicht heißt, dass eine Subklasse die Befehle und Eigenschaften der Superklasse in ihre eigene Deklaration kopiert. Stattdessen gibt es nur eine formale Verknüpfung zwischen den Klassen.

## Die Auswahl von passenden Elementen im Klassenbaum

Die Methoden- beziehungsweise Eigenschaftenauswahl in einem Klassenbaum muss natürlich geregelt sein. Sie erfolgt nach einer einfachen Regel.

- ✓ Ist der Nachrichtenselektor (der Methoden- oder Attributname einer Botschaft) in der Objektklasse des Empfängers nicht vorhanden, so wird die gewünschte Methode oder Eigenschaft in der nächsthöheren Superklasse des Nachrichtenselektors gesucht.
- ✓ Ist sie dort nicht vorhanden, erfolgt die Suche in der nächsthöheren Klasse, bis die Wurzelklasse des Klassenbaums erreicht ist.

Die Ausführung der Methode beziehungsweise der Zugriff auf die Eigenschaft erfolgt also in der ersten Klasse, in der sie gefunden wird (von der aktuellen Klasse im Klassenbaum aufwärts gesehen).

Gibt es im Klassenbaum keine Methode beziehungsweise Eigenschaft des spezifizierten Namens, so kommt es zu einer Fehlermeldung. Klassen, die sich auf derselben Ebene wie die aktuelle Klasse oder in anderen Zweigen des Klassenbaums befinden, werden nicht durchsucht.

## Mehrfachvererbung versus Einfachvererbung

In der Theorie der Objektorientierung gibt es Einfachvererbung sowie Mehrfachvererbung. In der Einfachvererbung (engl.: Single Inheritance) gilt für die Klassenhierarchie in einer baumartigen Struktur die Voraussetzung, dass eine Subklasse immer nur genau eine direkte Superklasse hat.

Eine Superklasse kann jedoch eine beliebige Anzahl an direkten Subklassen haben.

Wenn jedoch die Möglichkeit besteht, eine einzige Klasse direkt mit beliebig vielen Superklassen durch die Vererbung zu verknüpfen, nennt man dies Mehrfachvererbung (engl.: Multiple Inheritance). Objekte der Subklasse erben in dem Fall direkt Eigenschaften aus verschiedenen Superklassen.

Es gibt unbestritten einige gute Gründe für die Mehrfachvererbung. Der entscheidende Vorteil der mehrfachen Vererbung liegt in der Möglichkeit, die Probleme der realen Welt einfacher beschreiben zu können, denn auch dort hat ein Objekt Eigenschaften und Fähigkeiten aus verschiedenen übergeordneten logischen Bereichen. Ein Auto kann in der Denkweise der Mehrfachvererbung beispielsweise direkt von verschiedenen Superklassen erben. Die fiktive Klasse *Auto* erbt zum Beispiel direkt von den Klassen *Motor*, *Karosserie*, *Fahrgestell*, *Bremsen* und *Innenraum*.

Mit Mehrfachvererbung lässt sich das Beziehungsgeflecht des realen Objekts durch die Sammlung der Superklassen (relativ) vollständig und einfach beschreiben. Ein weiterer Vorteil ist ebenso, dass man leicht Dinge ergänzen kann, die man bei der ersten Realisierung einer Klasse vergessen hat.

Wenn bestimmte Eigenschaften und Methoden in einer Subklasse vergessen wurden, nimmt man einfach eine weitere Superklasse hinzu, die die fehlenden Elemente vererben kann.

Dieser Vorteil einer relativ vollständigen und einfachen Abbildung der Natur steht jedoch in keinem Verhältnis zu den damit verbundenen Nachteilen. Die schlimmsten Nachteile sind sicher die kaum nachvollziehbaren Beziehungsgeflechte in komplexeren Programmen bis hin zu Ringbeziehungen, in denen Klassen über mehrere Ebenen hinweg Superklassen von sich selbst sein können.

PHP nutzt nur Einfachvererbung.

## Vererbung zum Aufbau eines APIs

Vererbung nutzt man vor allen Dingen dann, wenn man objektorientiert ein API (Application Programming Interface) aufbauen möchte. Dies ist eine standardisierte Programmierschnittstelle, mit der vorhandener Quelltext oder allgemein Softwarebestandteile ohne Anpassungen der Schnittstelle für verschiedene Systeme verwendet werden können.

Und besonders da unterstützen die Vererbung und Datenkapselung die Wiederverwendbarkeit, was ausdrücklich im Sinn eines APIs ist. Objektorientierte Programmierschnittstellen sind damit deutlich flexibler als funktionsorientierte.

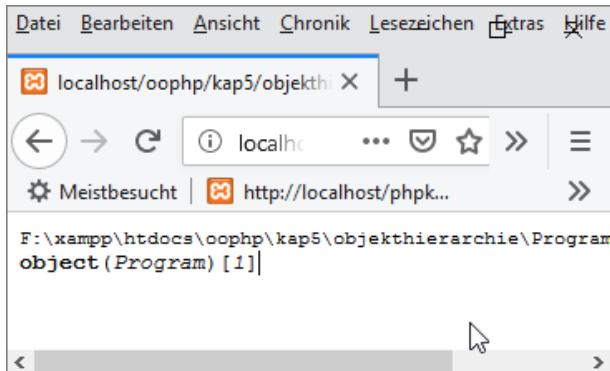
## 5.2 Die konkrete Umsetzung der Vererbung in PHP

Was Vererbung im Allgemeinen in der OOP darstellt und wie sie in der Theorie umgesetzt wird, haben wir gesehen. Schauen wir uns jetzt an, wie PHP dieses Konzept genau realisiert. Zuerst muss eine wichtige Tatsache festgehalten werden. Alle Klassen gehen in PHP auf einen Typen `object` zurück. Das bezeichnet in PHP ein verallgemeinertes Objekt ohne spezifische Member. Wobei das nicht so konsequent umgesetzt ist wie in wirklich von Anfang an objektorientiert konzipierten Sprachen wie Java oder C#, wo es ein elementares Basisobjekt gibt.

Betrachten Sie einmal das folgende Listing (Ordner `objekthierarchie`):

```
<?php
class Program {
    public function __construct() {
        $this -> main();
    }
    private function main() {
        var_dump($this);
    }
}
new Program();
?>
```

Mit `var_dump()` werden die Informationen von `$this` ausgegeben



Das aktuelle Objekt ist vom Typ object.

In dem Quellcode wird eine Klasse nicht explizit von einer Superklasse abgeleitet und damit gibt es hier noch keine vererbten Member mit Ausnahme der Tatsache, dass `$this` vorhanden und ein Objekt ist. Und `var_dump($this)` gibt aus, dass das aktuelle Objekt vom Typ `object` ist, also ein allgemeines Objekt.

## Erweiterung einer Klasse mit `extends`

Wenn man in PHP eine Klasse explizit als Subklasse einer Klasse angeben will, wird das mit dem Schlüsselwort `extends` erfolgen, das hinter dem Bezeichner der Klasse notiert wird und dem die Klasse folgt, die als Superklasse dienen soll. Das sieht formal so aus:

```
class Klassenname extends NameDerSuperKlasse {
    // Eigenschaften
    ...
    // Methoden
    ...
}
```

Die formale Klassendeklaration mit Vererbung in PHP

Damit spezifizieren Sie die Klasse, auf die Ihre neue Klasse aufbaut (die Superklasse). Durch die Erweiterung einer Superklasse machen Sie aus Ihrer Klasse (die Subklasse) zuerst eine Art virtuelle Kopie dieser Superklasse und ermöglichen gleichzeitig Veränderungen an dieser neuen Kopie.

Das könnte man also mit realen Klassen machen, wenn man erst einmal nur die Klassenschablonen betrachtet:

```
// Superklasse
class Person {
}
// Klasse Mitarbeiter
class Mitarbeiter extends Person {
}
// Klasse Kunde
class Kunde extends Person {
```

Eine Superklasse mit zwei Subklassen

In dem Fall sind die Klassen `Mitarbeiter` und `Kunde` Subklassen einer gemeinsamen Superklasse `Person`. Und die Konstruktoren der Subklassen rufen bei der Instanziierung den Konstruktor der Superklasse auf, um auch die vererbten Member zur Verfügung zu haben.

Es gibt in PHP eine ganze Reihe an nützlichen Funktionen zum Umgang mit Klassen und Objekten. Die Funktion `get_class_methods()` liefert in einem Array die Namen aller Methoden einer Klasse. Als Parameter können Sie sowohl den Bezeichner der Klasse als auch ein Objekt dieser Klasse benutzen. Ähnlich funktioniert die Funktion `get_class_vars()` zur Ermittlung der öffentlichen Attribute. Als Parameter muss der Bezeichner der Klasse angegeben werden. Ein Objekt wie bei der Funktion `get_class_methods()` ist hier nicht möglich. Die Funktion `get_parent_class()` gibt den Namen der Elternklasse eines Objektes zurück. Die Funktion `get_class()` liefert hingegen den Klassennamen eines Objekts. Ähnlich wie die Funktion `file_exists()` bei Dateien testet die Funktion `class_exists()` bei Bedarf das Vorhandensein einer Klasse. Die Funktion `get_declared_classes()` liefert ein Array mit allen deklarierten Klassen in einem PHP-Kontext.

## 5.3 Vererbung im Praxisprojekt

Ziehen wir zur Verdeutlichung der ganzen Ausführungen rund um die Vererbung ein konkretes Beispiel heran. Dazu greifen wir auf das „Praxisprojekt“ zurück, das wir seit Kapitel 4 bearbeiten. Dieses Projekt wird jetzt um mehrere Versionen weiterentwickelt und dabei refaktorisiert. Wir wollen vor allen Dingen ein API mit Klassen aufbauen, die in Vererbungsbeziehung zueinander stehen.

In diesem Kapitel starten wir wieder mit dem Verzeichnis `v1`, das die erste Version des Projekts in diesem Kapitel widerspiegeln und erst einmal nur den Stand des gesamten Quellcodes aus den Ergebnisdateien zu Kapitel 4 enthalten soll.

In diesem Projekt wird erst einmal in einer eigenen PHP-Datei eine neue Klasse `Mitarbeiter` angelegt, die folgenden Quellcode haben soll:

```
<?php

class Mitarbeiter {
    private $vorname = "";
    private $nachname = "";
    private $salter = 0;
    private $geschlecht = "";
    private $adresse = "";
    private $personalnr = 0;

    public function __construct(string $vorname, string $nachname,
        int $salter, string $geschlecht, string $adresse, int $personalnr)
    {
        $this -> setVorname($vorname);
        $this -> setNachname($nachname);
        $this -> setAlter($salter);
        $this -> setGeschlecht($geschlecht);
    }
}
```

```
$this -> setAdresse($adresse);
$this -> setPersonalnr($personalnr);
}

public function setVorname(string $vorname) {
    $this -> vorname = $vorname;
}
public function getVorname(): string{
    return $this -> vorname;
}
public function setNachname(string $nachname){
    $this -> nachname = $nachname;
}
public function getNachname(): string{
    return $this -> nachname;
}
public function setAlter(int $alter){
    $this -> alter = $alter;
}
public function getAlter(): int{
    return $this -> alter;
}
public function setGeschlecht(string $geschlecht){
    $this -> geschlecht = $geschlecht;
}
public function getGeschlecht(): string{
    return $this -> geschlecht;
}
public function setAdresse(string $adresse) {
    $this -> adresse = $adresse;
}
public function getAdresse(): string{
    return $this -> adresse;
}
public function setPersonalnr(string $personalnr){
    $this -> personalnr = $personalnr;
}
public function getPersonalnr(): string{
    return $this -> personalnr;
}
public function __destruct() {
    echo "<br />Objekt mit dem Namen " . $this -> vorname . " " .
        $this -> nachname ." wird gel&ouml;scht...";
}
}
?>
```

Die neue Klasse in der Datei „Mitarbeiter.class.php“

Nun ziehen wir zum Vergleich die Datei *Kunde.class.php* heran, wie sie nach der Übung aus Kapitel 4 aussehen sollte:

```
<?php
require_once("Konto.class.php");
class Kunde {
    private $vorname = "";
    private $nachname = "";
    private $alter = 0;
    private $geschlecht = "";
    private $adresse = "";
    private $konto = null;

    public function __construct(string $vorname, string $nachname,
        int $alter, string $geschlecht, string $adresse, Konto $konto) {
        $this -> setVorname($vorname);
        $this -> setNachname($nachname);
        $this -> setAlter($alter);
        $this -> setGeschlecht($geschlecht);
        $this -> setAdresse($adresse);
        $this -> konto = $konto;
    }
    public function setVorname(string $vorname) {
        $this -> vorname = $vorname;
    }
    public function getVorname(): string{
        return $this -> vorname;
    }
    public function setNachname(string $nachname) {
        $this -> nachname = $nachname;
    }
    public function getNachname(): string{
        return $this -> nachname;
    }
    public function setAlter(int $alter) {
        $this -> alter = $alter;
    }
    public function getAlter(): int{
        return $this -> alter;
    }
    public function setGeschlecht(string $geschlecht) {
        $this -> geschlecht = $geschlecht;
    }
    public function getGeschlecht(): string{
        return $this -> geschlecht;
    }
    public function setAdresse(string $adresse) {
        $this -> adresse = $adresse;
    }
}
```

```

    }
    public function getAdresse(): string{
        return $this -> adresse;
    }
    public function getKonto(): Konto{
        return $this -> konto;
    }
    public function __destruct() {
        echo "<br />Objekt mit dem Namen " . $this -> vorname . " " .
            $this -> nachname ." wird gel&ouml;scht...";
    }
}
?>

```

Die Klasse „Kunde“, wie sie nach der Übung aus Kapitel 4 aussehen sollte

Man erkennt auf den ersten Blick eine hohe Übereinstimmung von Quellcodepassagen bei beiden Klassen. Und da macht es ganz viel Sinn, diese in einer gemeinsamen Superklasse zusammenzufassen und nur die Codezeilen in diesen beiden Klassen zu belassen, in denen sich Unterschiede zeigen. Diese Art der Vorgehensweise ist die Bottom-Up-Methode und führt dann zu der oben schon angedeuteten Struktur, die sich in dem Verzeichnis v2 widerspiegeln soll.

## Die Superklasse

Der Aufbau der neuen Datei *Person.class.php* ist aufgrund der bisherigen Überlegungen eigentlich zwangsläufig und führt zur Superklasse Person:

```

<?php
class Person {
    private $vorname = "";
    private $nachname = "";
    private $salter = 0;
    private $geschlecht = "";
    private $adresse = "";

    public function __construct(string $vorname, string $nachname,
        int $salter, string $geschlecht, string $adresse) {
        $this -> setVorname($vorname);
        $this -> setNachname($nachname);
        $this -> setAlter($salter);
        $this -> setGeschlecht($geschlecht);
        $this -> setAdresse($adresse);
    }

    public function setVorname(string $vorname) {
        $this -> vorname = $vorname;
    }

    public function getVorname(): string{
        return $this -> vorname;
    }
}

```

```
public function setNachname(string $nachname) {
    $this -> nachname = $nachname;
}
public function getNachname(): string{
    return $this -> nachname;
}
    public function setAlter(int $alter){
        $this -> alter = $alter;
    }
public function getAlter(): int{
    return $this -> alter;
}
    public function setGeschlecht(string $geschlecht){
        $this -> geschlecht = $geschlecht;
    }
public function getGeschlecht(): string{
    return $this -> geschlecht;
}
    public function setAdresse(string $adresse){
        $this -> adresse = $adresse;
    }
public function getAdresse(): string{
    return $this -> adresse;
}

public function __destruct() {
    echo "<br />Objekt mit dem Namen " . $this -> vorname . " " .
    $this -> nachname ." wird gel&ouml;scht...";
}

}
?>
```

Die neue Klasse „Person“

Hier befinden sich alle Gemeinsamkeiten von einem Kunden und einem Mitarbeiter – und nur diese.

## Zugriff auf den Konstruktor der Superklasse und das Schlüsselwort **parent**

Die beiden Klassen für den Mitarbeiter und den Kunden sollten jetzt nur die Unterschiede deklarieren. Aber ganz so einfach ist das nicht, denn es gibt dort „Probleme“ mit dem Konstruktor. Diese treten interessanterweise aber nur auf, weil das Design des Quellcodes schon eine sehr hohe Qualität hat. Würden wir mit einer „schlechteren“ Qualität arbeiten, würden die Probleme nicht auftreten.

Genau genommen hängen die „Probleme“ daran, dass wir mit parametrisierten Konstruktoren arbeiten, was ja sehr sinnvoll ist, wenn jedes erzeugte Objekt einen definierten Anfangszustand haben soll. Ohne die Verwendung parametrisierter Konstruktoren hätten wir die nachfolgenden Probleme nicht, deren Lösung führt dann aber sogar zu einer noch viel besseren Softwarequalität.

Doch was bedeutet das? In der Klasse `Person` gibt es einen parametrisierten Konstruktor, der aber nur die Eigenschaften initialisiert, die auch in dieser Superklasse vorhanden sind. Das ist offensichtlich.

Redefinieren Sie in einer Klasse keinen expliziten Konstruktor, wird in deren Defaultkonstruktor automatisch der Defaultkonstruktor einer eventuellen Superklasse aufgerufen oder – wenn keine Superklasse da ist – das Objekt einfach erzeugt.

Redefinieren Sie in einer Klasse einen Konstruktor, wird aber auch dieser explizit redefinierte Konstruktor in einer Subklasse immer erst einmal den Defaultkonstruktor einer eventuellen Superklasse aufrufen – also den ohne Parameter. Und den gibt es in der Klasse `Person`, wie gerade ausgeführt, nicht.

Nun müssen Sie sowohl für den Kunden als auch den Mitarbeiter einen eigenen Konstruktor definieren, denn beide Typen definieren mehr Eigenschaften als die Superklasse und diese zusätzlichen Eigenschaften müssen damit jeweils über eigene Konstruktoren der Subklassen initialisiert werden.

Nur erst einmal kommen wir nicht weiter, da der Defaultkonstruktor von der Superklasse `Person` nicht mehr da ist (was ja – wie mehrfach erwähnt – ein Qualitätsmerkmal von gut designter OO-Software darstellt). Deshalb findet die Subklasse diesen notwendigen Konstruktor ihrer Superklasse nicht.

Es gibt in der Superklasse `Person` jedoch einen parametrisierten Konstruktor und diesen müssen Sie im Konstruktor der abgeleiteten Klasse gezielt als erste Anweisung aufrufen. Dafür gibt es das Schlüsselwort `parent`, was ein Stellvertreterbegriff für die jeweilige Superklasse ist. Dieser Token ergibt sich daher, dass man Klassen in einem Klassenbaum auch so sieht, dass sie in einer Generationenbeziehung stehen. Deshalb nennt man eine Superklasse auch gelegentlich Elternklasse.

Einen bestimmten Konstruktor der Superklasse rufen Sie also im Konstruktor der Subklasse (als erste Anweisung!) formal so auf:

```
parent :: __construct(Parameter)
```

*Der formale Aufruf des Konstruktors der Superklasse*

Es kommt der Gültigkeitsbereichsoperator zum Einsatz, weil `parent` ja für eine **Klasse** der Stellvertreterbegriff ist.

Und damit können wir die Klassen für den Kunden und den Mitarbeiter nun vollständig erstellen.

## Die Subklassen

Der Aufbau der neuen Datei *Mitarbeiter.class.php* ist aufgrund der bisherigen Überlegungen im Grunde wieder fast zwangsläufig, wobei die Schritte im Konstruktor doch Erklärungen bedürfen:

```
<?php
require_once("Person.class.php");
class Mitarbeiter extends Person {
    private $personalnr = 0;

    public function __construct(string $vorname, string $nachname,
        int $alter, string $geschlecht, string $adresse, int $personalnr)
    {
        parent::__construct($vorname, $nachname, $alter, $geschlecht,
            $adresse)
        $this -> setPersonalnr($personalnr);
    }
    public function setPersonalnr(string $personalnr) {
        $this -> personalnr = $personalnr;
    }
    public function getPersonalnr(): string{
        return $this -> personalnr;
    }
    public function __destruct() {
        echo "<br />Objekt mit der Superklasse " . get_parent_class($this) .
            " wird gelöscht. Es ist vom Typ " . get_class($this) . ".";
    }
}
?>
```

*Die neue Version der Klasse Mitarbeiter*

Zuerst sehen Sie das Einbinden der Quelltextdatei der Superklasse Person und dann die Erweiterung der Klassendeklaration der Subklasse mit `extends`.

In der Subklasse `Mitarbeiter` verbleiben nur noch die Methoden zum Zugriff auf die Personalnummer und ein Destruktor, der eine individuelle Meldung ausgibt.

Im redefinierten Konstruktor finden Sie als ersten Aufruf den Zugriff auf den parametrisierten Konstruktor der Superklasse. Die Parameter sind die Werte, die an den Konstruktor der Subklasse `Mitarbeiter` übergeben, aber in der Superklasse `Person` initialisiert werden. Sie werden also an die Superklasse weitergereicht. Nur die individuelle Eigenschaft wird hier noch explizit im abgeleiteten Konstruktor gesetzt.

Von Interesse ist auch der Destruktor. Denn in diesem werden die Superklasse und die aktuelle Klasse des Objekts ausgegeben, wenn das Objekt beseitigt wird. Beachten Sie, dass auch in der Superklasse `Person` der Destruktor eine Meldung ausgibt. Interessant wird sein zu erörtern, was man aus der angezeigten Meldung schließen kann: Denn es ist nicht die Meldung des Konstruktors der Superklasse, sondern der Subklasse. Es gibt also zur Laufzeit dann kein Objekt vom Typ der Superklasse, sondern nur Objekte vom Typ der Subklassen, die aber alle vererbten Dinge der Superklasse bereitstellen.

Den gleichen Aufbau der Klasse findet man dann auch in der anderen Subklasse Kunde und dieser zeigt den wesentlichen Vorteil der Vererbung: Man muss gemeinsame Anweisungen nur an einer Stelle notieren und nicht zweimal den identischen Code. Nur die Unterschiede werden in Subklassen notiert.

```
<?php
require_once("Konto.class.php");
require_once("Person.class.php");
class Kunde extends Person {
    private $konto = null;

    public function __construct(string $vorname, string $nachname,
        int $alter, string $geschlecht, string $adresse, Konto $konto) {
        parent :: __construct($vorname, $nachname, $alter, $geschlecht,
            $adresse)
        $this -> konto = $konto;
    }

    public function getKonto(): Konto{
        return $this -> konto;
    }
    public function __destruct() {
        echo "<br />Objekt mit der Superklasse " . get_parent_class($this) .
            " wird gel&ouml;scht. Es ist vom Typ " . get_class($this) . ".";
    }
}
?>
```

*Die neue Klasse Kunde*

So kann man jetzt die beiden Subklassen zur Instanziierung von Objekten verwenden:

```
<?php
require_once("Kunde.class.php");
require_once("Mitarbeiter.class.php");

class Program {
    public function __construct(){
        $this -> main();
    }

    function __destruct() {
        echo "<br />Objekt wird gel&ouml;scht...";
    }

    private function main(){
        $kontol = new Konto(floatval("1023.34"), "Sparbuch");
        $kundel = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42", $kontol);
        $mitarbeiter1 = new Mitarbeiter("Fred", "Feuerstein", 41, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42", $kontol);
    }
}
```

```

        "666 Steinhausen, Steinstrasse 42", 4711);
    var_dump($kunde1);
    var_dump($mitarbeiter1);
}
}

new Program();
?>

```

Die Programmklasse

```

F:\xampp\htdocs\oophp\kap5\v2\Program.class.php:20:
object(Kunde) [3]
private 'konto' =>
object(Konto) [2]
private 'kontostand' => float 1023.34
private 'kontotyp' => string '' (length=0)
public 'Sparbuch' => string 'Sparbuch' (length=8)
private 'vorname' (Person) => string 'Hans' (length=4)
private 'nachname' (Person) => string 'Dampf' (length=5)
private 'alter' (Person) => int 42
private 'geschlecht' (Person) => string 'Mann' (length=4)
private 'adresse' (Person) => string '12345 Irgendwo, Irgendeinestrasse 42' (length=36)

F:\xampp\htdocs\oophp\kap5\v2\Program.class.php:21:
object(Mitarbeiter) [4]
private 'personalnr' => string '4711' (length=4)
private 'vorname' (Person) => string 'Fred' (length=4)
private 'nachname' (Person) => string 'Feuerstein' (length=10)
private 'alter' (Person) => int 41
private 'geschlecht' (Person) => string 'Mann' (length=4)
private 'adresse' (Person) => string '666 Steinhausen, Steinstrasse 42' (length=32)

Objekt mit der Superklasse Person wird gelöscht. Es ist vom Typ Kunde.
Objekt mit dem Kontostand 1023.34 wird gelöscht...
Objekt mit der Superklasse Person wird gelöscht. Es ist vom Typ Mitarbeiter.
Objekt wird gelöscht...

```

Das Programm verwendet nun Vererbung.

## 5.4 Der Sichtbarkeitsmodifizierer `protected`

Bisher kamen bei den Sichtbarkeitsmodifizierern nur `private` und `public` zum Einsatz. Aber es gibt noch den Sichtbarkeitsmodifizierer `protected` (geschützt). Auf so definierte Attribute und Methoden kann aus der eigenen Klasse oder davon abgeleiteten Klassen zugegriffen werden.

Und das bedeutet im Umkehrschluss, dass dieser Modifizierer vor allen Dingen im Zusammenhang mit Vererbung wichtig ist. Man deklariert alle oder bestimmte Member als `protected`, wenn man Subklassen explizit Zugang zu diesen Membern geben will, aber **keinen anderen Klassen**.

Das soll nun implementiert werden, indem in der Klasse `Person` eine Methode deklariert wird, die in den Subklassen `Kunde` und `Mitarbeiter` zur Verfügung stehen soll – aber sonst nirgends.

## Aufräumen des Praxisprojekts

Parallel zur Implementierung neuer Funktionalitäten wird das Kapitelprojekt um Ausgaben und Methoden reduziert, die bisher aus didaktischen Gründen notwendig oder sinnvoll waren. Es entfallen ab der neuen Projektversion v3 alle expliziten Destruktoren und die Klonmethoden sowie die `var_dump()`-Ausgaben im Programmskript, damit das Projekt übersichtlicher wird und der Fokus deutlicher auf den neuen Themen liegt. Denn das Ziel ist ja letztendlich die Erstellung einer Webseite: Metainformationen, die zu einem gewissen Zeitpunkt didaktisch sinnvoll waren, sind nicht mehr gewünscht. Beachten Sie, dass diese Veränderungen damit explizit keine Refaktorisierung, sondern eine funktionale Weiterentwicklung darstellen.

Das ist der Quellcode der neuen Version von der Klasse Person:

```
<?php

class Person {
    private $vorname = "";
    private $nachname = "";
    private $salter = 0;
    private $geschlecht = "";
    private $adresse = "";

    public function __construct(string $vorname, string $nachname, int
        $salter, string $geschlecht, string $adresse) {
        $this -> setVorname($vorname);
        $this -> setNachname($nachname);
        $this -> setAlter($salter);
        $this -> setGeschlecht($geschlecht);
        $this -> setAdresse($adresse);
    }

    public function setVorname(string $vorname) {
        $this -> vorname = $vorname;
    }
    public function getVorname(): string{
        return $this -> vorname;
    }

    public function setNachname(string $nachname) {
        $this -> nachname = $nachname;
    }
    public function getNachname(): string{
        return $this -> nachname;
    }
    public function setAlter(int $salter){
        $this -> alter = $salter;
    }
    public function getAlter(): int{
        return $this -> alter;
    }
}
```

```

    }
    public function setGeschlecht(string $geschlecht) {
        $this -> geschlecht = $geschlecht;
    }
    public function getGeschlecht(): string{
        return $this -> geschlecht;
    }
    public function setAdresse(string $adresse) {
        $this -> adresse = $adresse;
    }
    public function getAdresse(): string{
        return $this -> adresse;
    }

    protected function kommunizieren($wie){
        echo "<hr />$wie<hr />";
    }
}
?>
```

*Die Superklasse aller Personen im Projekt*

Die neue Methode `kommunizieren()` soll in den Subklassen bereitstehen und wird deshalb als `protected` deklariert.

Das ist die erste Subklasse, in der die Methode verwendet wird:

```

<?php
require_once("Person.class.php");
require_once("Konto.class.php");

class Kunde extends Person {
    private $konto = null;

    public function __construct(string $vorname, string $nachname,
        int $alter, string $geschlecht, string $adresse, Konto $konto) {
        parent :: __construct($vorname, $nachname, $alter, $geschlecht,
            $adresse);
        $this -> konto = $konto;
    }

    public function getKonto(): Konto{
        return $this -> konto;
    }
    public function geschaeftsvorfall(){
        echo "Bisheriger Kontostand: " .
        $this -> getKonto() -> getKontostand() . " EUR.<br />";
        $this -> kommunizieren("Kontakt zwischen Kunde und Mitarbeiter.");
        $this -> konto ->auszahlen(100);
    }
}
```

```

        echo "Neuer Kontostand: " .
        $this -> konto ->getKontostand() . " EUR.<br />";
    }
}
?>

```

*Die Klasse für den Kunden*

In der neuen Methode `geschaeftsvorfall()` gibt es zwei interessante Stellen:

- ✓ Es gibt eine **verkettete Pfeilnotation**. Über `$this -> getKonto()` wird zuerst ein Objekt vom Typ `Konto` zurückgegeben und dieses dann direkt (anonym) verwendet, um dessen Methode `getKontostand()` aufzurufen und den Kontostand zurückzugeben.
- ✓ Mit `$this -> kommunizieren("Kontakt zwischen Kunde und Mitarbeiter.")` wird die vererbte Methode aus der Superklasse dann in der Subklasse verwendet. Diese steht explizit durch den Modifizierer in der Subklasse zur Verfügung.

Auch die zweite Subklasse soll die geschützte Methode verwenden:

```

<?php
require_once("Person.class.php");
class Mitarbeiter extends Person {
    private $personalnr = 0;

    public function __construct(string $vorname, string $nachname,
        int $alter, string $geschlecht, string $adresse, int $personalnr) {
        parent :: __construct($vorname, $nachname, $alter, $geschlecht,
            $adresse);
        $this -> setPersonalnr($personalnr);
    }

    public function setPersonalnr(string $personalnr) {
        $this -> personalnr = $personalnr;
    }
    public function getPersonalnr(): string{
        return $this -> personalnr;
    }
    public function kontaktwunsch(){
        echo "<hr />Notwendiger Kontakt zum Kunden.<br />";
        $this -> kommunizieren("Kontakt zwischen Mitarbeiter und Kunde.");
    }
}
?>

```

*Die Klasse für den Mitarbeiter*

So kann man jetzt die beiden Subklassen zur Instanziierung von Objekten verwenden sowie die Methoden von einem Kunden und Mitarbeiter, die dann selbst wiederum die geschützte Methode nutzen. Dabei ist zu beachten, dass die Methoden, die dann im Programmksript selbst aufgerufen werden, (implizit) `public` sein müssen. Denn diese Programmklasse ist keine Subklasse von `Person` oder deren Subklassen.

```

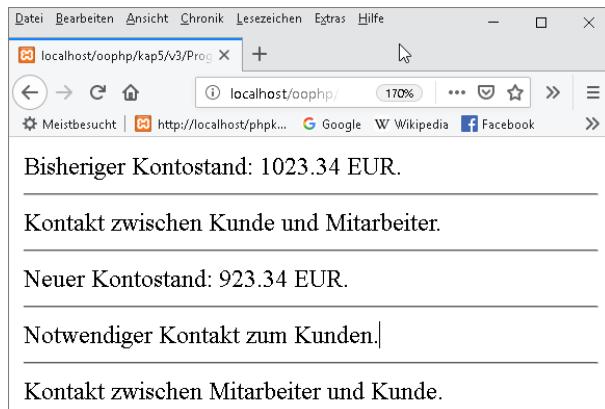
<?php
require_once("Kunde.class.php");
require_once("Mitarbeiter.class.php");

class Program {
    public function __construct() {
        $this -> main();
    }

    private function main(){
        $kontol = new Konto(floatval("1023.34"), "Sparbuch");
        $kundel = new Kunde("Hans", "Dampf", 42, "Mann",
            "12345 Irgendwo, Irgendeinestrasse 42", $kontol);
        $kundel -> geschaeftsvorfall();
        $mitarbeiter1 = new Mitarbeiter("Fred", "Feuerstein", 41, "Mann",
            "666 Steinhausen, Steinstrasse 42", 4711);
        $mitarbeiter1 -> kontaktwunsch();
    }
}
new Program();
?>

```

*Die neue Programmklasse*



*Die Subklassen verwenden die geschützte Methode.*

## 5.5 Überschreiben und verdecken

Wenn Sie Member aus einer Superklasse vererbt bekommen, können Sie das „Erbe“ auch verändern. Das erfolgt durch eine „Redefinition“ von vererbten Membern, wobei dies genauer untersucht werden muss.

Wenn etwa Methoden in der Superklasse und der Subklasse identisch benannt werden, verdecken die Methoden der abgeleiteten Klassen gleichnamige Methoden der Superklassen. Methoden der Superklassen werden von gleichnamigen Methoden der abgeleiteten Klassen überschrieben (**override**).

Die deutsche Übersetzung von „Overriding“ ist etwas unglücklich. Übersetzt heißt „to override“ so viel wie „außer Kraft setzen“, „verdecken“ oder „überdefinieren“. Der deutsche Begriff für die Technik hieße also besser „Überdefinieren“ oder „Verdecken“, „Überschreiben“ hat sich jedoch allgemein etabliert.

Die Superklasse soll für eine Verdeutlichung so aussehen (Ordner *override*):

```
<?php
class SuperK {
    public $z = 41;
    public function ausgabe() {
        echo "<hr />Methode der Superklasse<hr />";
    }
}
?>
```

*Die Superklasse*

Das ist die Subklasse, die sowohl eine Eigenschaft als auch eine Methode gleichen Namens besitzt:

```
<?php
require_once("SuperK.class.php");
class SubK extends SuperK {
    public $z = 42;
    public function ausgabe() {
        echo "<hr />Methode der Subklasse<hr />";
    }
}
?>
```

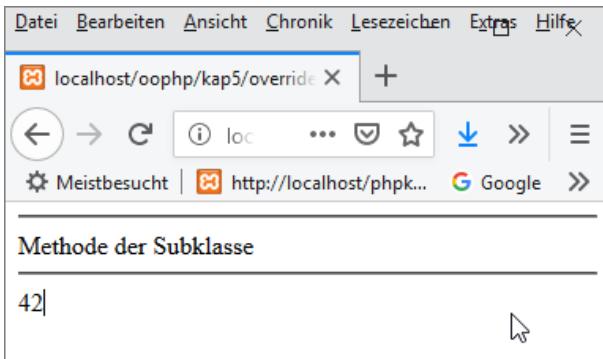
*Die Subklasse*

Das ist dann die Programmklasse, die ein Objekt vom Typ der Subklasse erstellt:

```
<?php
require_once("SubK.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }
    private function main() {
        $obj = new SubK();
        $obj -> ausgabe();
        echo $obj -> z;
    }
}
new Program();
?>
```

*Die neue Programmklasse*

An der Ausgabe sehen Sie, dass sowohl die Methode als auch die Eigenschaft der Superklasse überschrieben wurden und statt der vererbten Member die Member aus der Subklasse verwendet werden.



*Die überschriebene Methode und Eigenschaft*

## Wichtige Besonderheiten beim Überschreiben

Es gibt einige wichtige Besonderheiten und Regeln beim Überschreiben, die zu beachten sind:

- ✓ Die überschreibende Methode muss bzw. sollte hinsichtlich der Parameter die gleiche Signatur wie die überschriebene Methode haben. Das bedeutet, dass die Parameter gleich sein sollten. Wenn das nicht der Fall ist, wird die Methode der Superklasse dennoch überschrieben, aber es wird eine Warnung generiert.
- ✓ Die Sichtbarkeit der überschreibenden Member muss zwingend mindestens so hoch sein wie die der überschriebenen Member. Das bedeutet, dass ein geschütztes Member (`protected`) in der Superklasse von einem geschützten (`protected`) und einem öffentlichen (`public`) Member in der Subklasse überschrieben werden kann, ein öffentliches Member in der Superklasse jedoch nur von einem öffentlichen Member in der Subklasse.
- ✓ Private Member werden nicht vererbt und damit ist die Redefinition kein Überschreiben.
- ✓ Auf verdeckte Member der Superklasse können Sie in der Subklasse über `parent` und den Gültigkeitsoperator zugreifen. Etwa so: `parent :: ausgabe () ;`.

## Überschreiben von Methoden verbieten – `final`

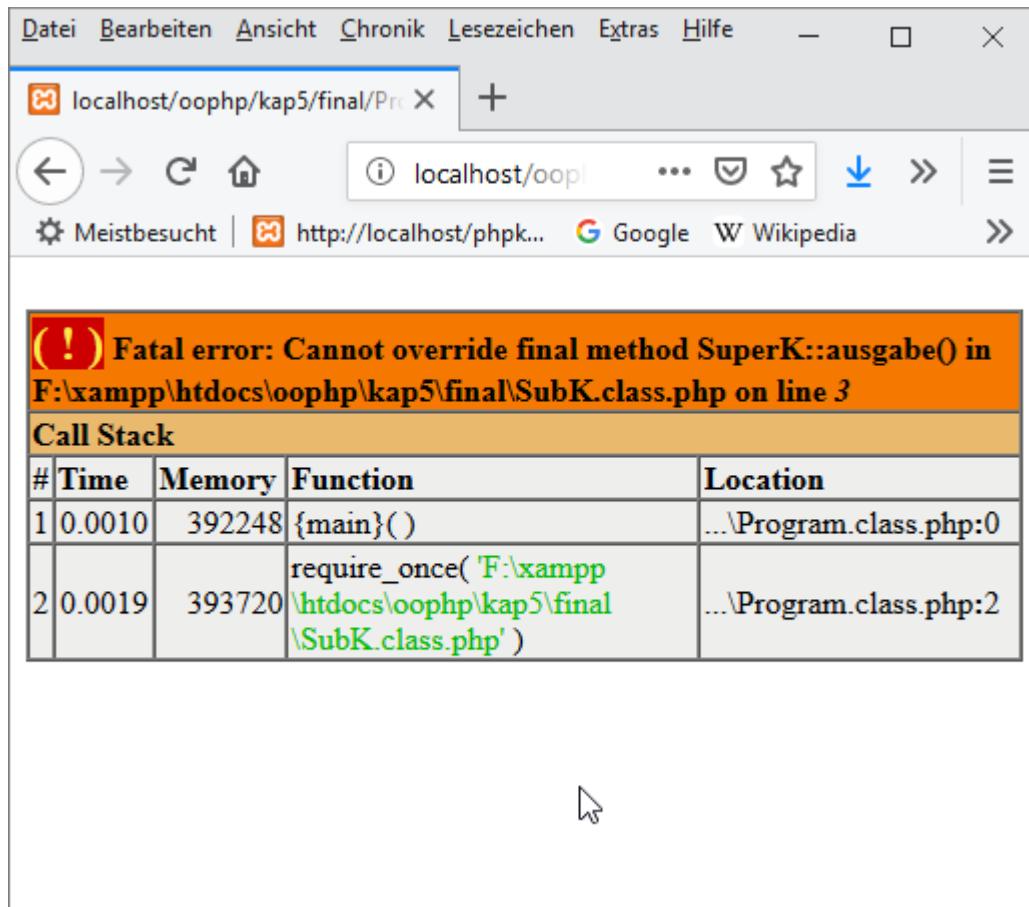
So einfach es auch ist, in Subklassen Bestandteile einer Superklasse zu überschreiben – manchmal muss man das jedoch verhindern. PHP stellt dazu für Methoden das Schlüsselwort `final` zur Verfügung.

Das soll mit einem Beispiel demonstriert werden, das so nicht funktionieren kann (Verzeichnis `final`). Die Superklasse soll für eine Verdeutlichung so aussehen:

```
<?php
class SuperK {
    public $z = 41;
    public final function ausgabe() {
        echo "<hr />Methode der Superklasse<hr />";
    }
}
?>
```

*Die Superklasse*

Wenn Sie die Methode in der Subklasse überschreiben wollen, erhalten Sie eine Fehlermeldung.



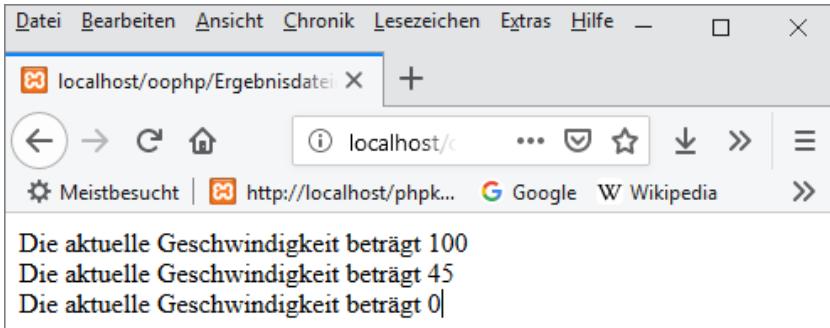
*Die finale Methode darf nicht überschrieben werden.*

## 5.6 Übungen

**Übungsdateien:** --

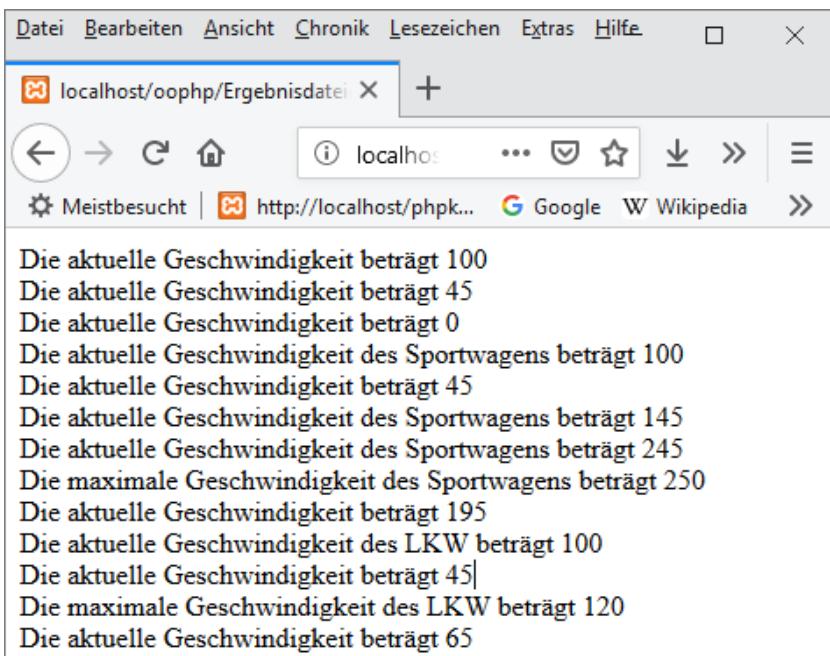
**Ergebnisdateien:** Auto.class.php,  
Sportwagen.class.php, Lastkraftwagen.class.php,  
Program.class.php, Program2.class.php

1. Erstellen Sie eine Klasse Auto.
2. Diese Klasse stellt eine private Eigenschaft \$geschwindigkeit und die öffentlichen Methoden bremsen() und beschleunigen() bereit.
3. Beide Methoden bekommen das Maß der Geschwindigkeitsänderung als Parameter übergeben.
4. Nach einer Änderung der Geschwindigkeit soll diese ausgegeben werden.
5. Beim Bremsen darf die Geschwindigkeit nicht negativ werden. Die Beschleunigung ist unbegrenzt möglich.
6. Erstellen Sie eine Programmklasse und erzeugen Sie dort von der Klasse eine Instanz.
7. Beschleunigen und bremsen Sie die Instanzen mehrfach.



*So ungefähr könnte die Ausgabe aussehen.*

8. Ergänzen Sie in der Klasse Auto einen geschützten Getter und Setter auf die Geschwindigkeit.
  9. Erzeugen Sie zwei Subklassen der Klasse Auto mit Namen Sportwagen und Lastkraftwagen.
  10. Fügen Sie in beide eine private Eigenschaft \$hoechstgeschwindigkeit hinzu. Der Wert für einen Sportwagen soll auf 250 und für einen Lastkraftwagen auf 120 initialisiert werden.
  11. Überschreiben Sie die die Methode beschleunigen() in beiden Klassen so, dass die Höchstgeschwindigkeit nicht überschritten werden kann. Wenn die Geschwindigkeitsänderung die Höchstgeschwindigkeit überschreiten würde, setzen Sie die aktuelle Geschwindigkeit auf die Höchstgeschwindigkeit. Geben Sie individuelle Meldungen aus. Die Methode bremsen() soll unverändert in der geerbten Version verwendet werden.
  12. Erstellen Sie eine Programmklasse und erzeugen Sie dort von allen drei Klassen eine Instanz.
  13. Beschleunigen und bremsen Sie die drei Instanzen mehrfach.



*So ungefähr könnte die Ausgabe mit individuellen Meldungen der Subklassen aussehen.*

# 6

## Abstrakte Klassen und Schnittstellen

### 6.1 Abstrakte Klassen

Es gibt in der OOP eine besondere Form einer Klasse, die **abstrakte Klasse** genannt wird. In dem Kapitel sehen Sie, was das ist und was Ihnen solche abstrakten Klassen eigentlich bringen. Im Wesentlichen bezeichnet man damit (potenziell) unvollständigen Code. Dabei können sowohl Methoden einer Klasse unvollständig sein als auch die Klasse selbst.

Damit dieser unvollständige Code sinnvoll genutzt werden kann, erzwingt das gewisse Verhaltensweisen, die hauptsächlich der Vervollständigung dieses Codes vor einer eventuellen konkreten Instanziierung dienen.

#### Grundsätzliches zu abstrakten Klassen und Methoden in PHP

Abstrakte Klassen als auch Methoden müssen in PHP besonders gekennzeichnet werden. Sie werden beide mit dem vorangestellten Schlüsselwort `abstract` deklariert.

Eine abstrakte Klasse ist damit im Prinzip eine gewöhnliche Klasse, der aber der besagte Modifizierer `abstract` vorangestellt wird und dies hat Folgen.

Die signifikante Eigenschaft von abstrakten Klassen ist, dass sie Methoden enthalten dürfen, die noch nicht vollständig sind. Zur Kennzeichnung haben diese ein Semikolon statt eines Methodenkörpers hinter der Methodensignatur stehen.

Damit ist es aber nicht mehr sinnvoll, dass von einer abstrakten Klasse eine Instanz gebildet wird. So eine Instanz könnte dann diesen unfertigen Code bereitstellen und wenn dieser aufgerufen wird, kann offensichtlich keine sinnvolle Aktion stattfinden.

Eine abstrakte Klasse muss nicht zwingend „unfertigen“ Code enthalten – sie kann auch ganz normalen Quellcode enthalten.

Solch eine abstrakte Struktur ist aber nicht nutzlos – ganz im Gegenteil. Der hauptsächliche Nutzen von abstrakten Klassen liegt in den Möglichkeiten zur Strukturierung eines Klassenbaums und das ist immens wichtig für ein professionelles OO-Design, was eine möglichst optimale Qualität bietet.

Man kann in PHP zwei wichtige Möglichkeiten für die Verwendung abstrakter Klassen unterscheiden:

- ✓ Sie verwenden die abstrakte Klasse ohne Instanziierung. Dann verwendet man deren bereits vollständige Klassenelemente (statische Elemente), denn das ist grundsätzlich möglich.
- ✓ Sie vervollständigen die abstrakte Klasse. Das erfolgt in einer Subklasse der abstrakten Klasse durch Überschreiben. Anschließend instanzieren Sie die Subklasse. Das ist der Regelfall der Verwendung von abstrakten Klassen.

Um die Idee der abstrakten Strukturen deutlicher zu machen, ziehen wir unser bisheriges Beispiel heran und entwickeln es weiter. Aber vorher soll erst einmal eine abstrakte Klasse erstellt und daran beobachtet werden, was bei verschiedenen Aktionen passiert. Um dieses Verhalten von PHP in gewisser Konstellation jedoch zu verstehen, ist die Kenntnis von einem besonderen Konzept hilfreich.

## Vertragsbasierte Programmierung

In der OOP werden in vielen Situationen so genannte **Verträge** beziehungsweise **Kontrakte** zwischen dem Ersteller einer gewissen Codestruktur (etwa einer abstrakten Klasse) und dessen Verwender (jemand, der diese abstrakte Klasse verwenden möchte) geschlossen. Entweder in Form von eigens dafür vorgesehenen Sprachkonstrukten oder durch geeignete Programmierung.

PHP unterstützt – im Gegensatz zu etwa Java oder C# – vertragsbasierte Programmierung nur in Teilen. Durch die lose Typisierung sind insbesondere Kontrakte zu Datentypen und Rückgabewerten von Methoden nicht vorhanden.

Vertragsbasierte Programmierung oder Kontrakt-Programmierung (contract based programming) legt zwischen den aufrufenden und den aufgerufenen Stellen fest, welche Bedingungen vor und nach der Verwendung eines Codesegments gelten müssen. Bedingungen beziehen sich etwa auf den Zustand des aufgerufenen Objekts, die übergebenen Parameter und die zwingenden Ergebnisse. Ein Kontrakt kann insbesondere festlegen, welche Eingangs- und Ausgangsbedingungen bei einem Methodenaufruf erfüllt sein müssen.

Bei einem Kontrakt werden vom theoretischen Konzept einzelne Bestandteile unterschieden:

- ✓ **Preconditions** legen die Parameter und den Anfangszustand des aufgerufenen Objekts oder der Methode fest.
- ✓ **Postconditions** legen die Ergebnisse und den Zustand des aufgerufenen Objekts oder der Methode nach Ausführung des Aufrufs fest.
- ✓ **Invariants** überprüfen die Konsistenz des Zustands von dem aufgerufenen Objekt oder der Methode.

Der Vorteil von Kontrakten – soweit ein System diese unterstützt – ist, dass sowohl der Ersteller von Code als auch der Verwender zu 100% sicher sein können, dass der Code genauso genutzt wird (Ersteller) bzw. werden kann (Verwender), wie geplant (Ersteller) und zugesagt (Verwender) wird.

## Abstrakte Techniken in der Praxis testen

Gegeben sei nun das folgende Listing, das nur eine abstrakte Klasse ohne konkrete Implementierungen zeigt (Verzeichnis *abstrakt1*):

```
<?php
abstract class A1 {
}
?>
```

*Eine abstrakte Klasse*

Nun konnte man in allen bisherigen Beispielen Klassen einfach instanziieren. Betrachten Sie das folgende Listing:

```
<?php
require_once("A1.class.php");
class Program {
    public function __construct() {
        $this -> main();
    }

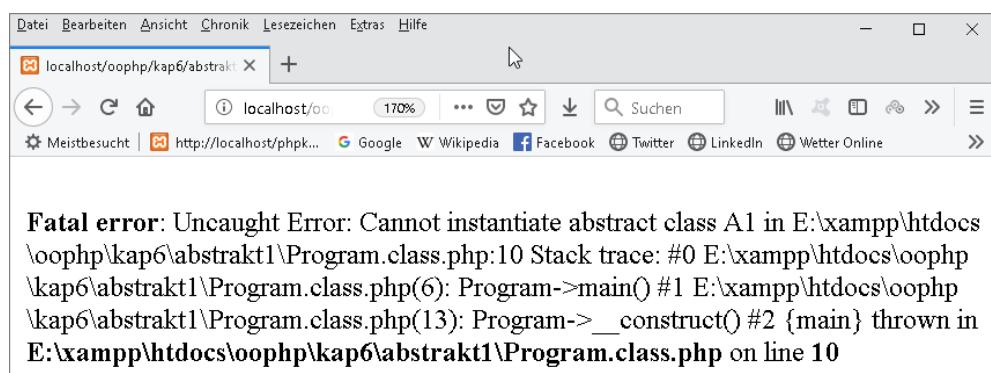
    public function main() {
        new A1();
    }
}

new Program();
?>
```

*Der Versuch, die abstrakte Klasse zu instanziieren*

Als erste Anweisung muss die Quellcodedatei importiert werden, in der die abstrakte Klasse deklariert ist. Wenn man das PHP-Skript mit dem eigentlichen Programmcode ausführt, wird eine Ausnahme geworfen und das Skript beendet. Doch muss man erneut betonen, dass dies kein „Problem“ ist, sondern eine erwünschte Reaktion. Man deklariert Klassen ja nicht „einfach so“ oder „aus Spaß“ als abstrakt. Sondern man verfolgt damit Ziele und eines davon ist, dass diese Klasse unter keinen Umständen mehr zum Instanziieren von einem Objekt verwendet werden darf.

Der Ersteller der Klasse hat mit der Deklaration als abstrakt einen Kontrakt mit Bedingungen formuliert, die ein Verwender zwingend einhalten muss. Und der Kontrakt sagt, dass die Klasse nicht instanziert werden darf.



*Das Instanziieren führt zum erhofften Fehler.*

Doch warum tut man das? Bisher ist die Situation noch nicht so, dass ein Instanziieren der Klasse ein Problem wäre.

Aber das würde beim folgenden Code vorliegen (*abstrakt2*):

```
<?php
abstract class A1 {
    public function einzahlen(float $betrag) {
        echo "Sie haben $betrag EUR eingezahlt.";
    }

    public abstract function auszahlen(float $betrag);

}
?>
```

*Eine abstrakte Klasse mit einer abstrakten Methode*

Sie sehen, dass es in der Klasse einmal eine neue Methode `einzahlen()` gibt, die vollständig ist und über eine Instanz aufgerufen werden könnte.

Und es gibt eine abstrakte Methode `auszahlen()`, die keinen Körper hat, sondern einfach mit einem Semikolon endet.

Was sollte denn passieren, wenn diese Methode über eine Instanz aufgerufen wird? Es kann also nicht sein, dass es eine Instanz geben darf, die solche „unfertigen“ Methoden bereitstellt und deshalb wird die Instanziierung unterbunden.

Diese Fehlerreaktionen sind Teil dessen, was man in der OOP eben vertragsbasierte Programmierung nennt. Obwohl diese Vereinbarung nicht auf abstrakte Klassen und Methoden beschränkt ist (selbst in PHP), ist sie vor allen Dingen da sinnvoll, wie hier zu erkennen ist.

Um die Konsequenzen noch deutlicher zu machen, testen die nächsten zwei (fehlerhaften) Listings die vertragsbasierte Programmierung weiter aus.

In dem Code im Verzeichnis *abstrakt3* wird bei der unvollständigen Methode `auszahlen()` auf den vorangestellten Modifizierer `abstract` verzichtet.

```
<?php
abstract class A1 {
    public function einzahlen(float $betrag) {
        echo "Sie haben $betrag EUR eingezahlt.";
    }

    public function auszahlen(float $betrag);

}
?>
```

*Eine unvollständige Methode ohne Kennzeichnung als abstrakt*

Das führt schon vor der Instanziierung zu einem fatalen Fehler (nicht nur einer Ausnahme zur Laufzeit).

```
Fatal error: Non-abstract method
A1::auszahlen() must contain body in E:\xampp
\htdocs\oophp\kap6\abstrakt3\A1.class.php on
line 7
```

*Die Syntax der abstrakten Methode ist falsch.*

Sie werden bei dem Test schon an der Meldung sehen, dass die Methode einen Körper haben muss, da Sie nicht als abstrakt gekennzeichnet wurde. Auch hier wird streng darauf geachtet, dass der Kontrakt eingehalten wird und der sagt, dass eine Methode oder Körper als abstrakt gekennzeichnet werden muss.

Nun soll noch ausgetestet werden was passiert, wenn eine abstrakte Methode in einer „normalen“ Klasse notiert wird – also in einer Klasse, die selbst nicht als abstrakt gekennzeichnet wird (*abstrakt4*).

```
<?php
class A1 {
    public function einzahlen(float $betrag) {
        echo "Sie haben $betrag EUR eingezahlt.";
    }
    public abstract function auszahlen(float $betrag);
}
?>
```

*Eine nichtabstrakte Klasse mit einer abstrakten Methode*

```
Fatal error: Class A1 contains 1 abstract method and
must therefore be declared abstract or implement the
remaining methods (A1::auszahlen) in E:\xampp
\htdocs\oophp\kap6\abstrakt4\A1.class.php on line 9
```

*Die Syntax der abstrakten Methode ist falsch.*

Auch hier wird mit einem fatalen Fehler abgebrochen und die Meldung angezeigt, dass eine abstrakte Methode nur in einer abstrakten Klasse deklariert werden darf.

Um es noch einmal deutlich zu machen – diese Fehler und Ausnahmen sind kein Teil eines Problems, sondern deren sichere und elegante Lösung! Sie erlauben es auf Grund von zwingenden Kontrakten mit „gefährlichem“ Code umzugehen und dabei sicherzustellen, dass trotzdem nichts passieren kann. Das ist die Kernidee der vertragsbasierten Programmierung.

## Eine abstrakte Klasse direkt verwenden

Abstrakte Klassen können zwar – wie wir gesehen haben – nicht instanziert werden. Allgemein kann in einer abstrakten Klasse aber jeder bereits vollständig definierte Code unmittelbar (als Klassenelement) verwendet werden.

Etwa so, wie im folgenden kleinen Code zu sehen ist (*abstrakt5*):

```
<?php
abstract class Person {
    public static function alarm() {
        echo "Feuer<br />";
    }
    const VERSION = 1.1;
}
?>
```

*Eine abstrakte Klasse mit statischen Klassenmembern*

Diese beiden Member können Sie verwenden, ohne die Klasse Person instanziieren zu müssen (was ja auch gar nicht geht).

```
<?php
include_once ("Person.class.php");

class Program {
    public function __construct() {
        Person::alarm();
        echo Person::VERSION;
    }
}
new Program();
?>
```

Testen Sie das Beispiel in einem Browser. Sie sehen, dass Sie hier direkt über den Gültigkeitsbereichsoperator :: die Klassenmember verwenden können.

## 6.2 Schnittstellen

Nun bleibt bei den abstrakten Techniken noch die Frage offen, was es mit Schnittstellen (Interfaces) auf sich hat? Eine Schnittstelle ist etwas Ähnliches wie eine abstrakte Klasse, aber nicht ganz identisch.

Eine Schnittstelle beinhaltet **ausschließlich** unvollständigen (abstrakten) Code oder Konstanten. Auch syntaktisch gibt es einen deutlichen Unterschied – statt mit dem Schlüsselwort `class` wird eine Schnittstelle mit dem Schlüsselwort `interface` begonnen.

Etwa so:

```
interface IInfos {  
    ...  
}
```

*Eine Schnittstelle*

Es gibt in einigen Sprachen Namenskonventionen für Schnittstellen. Oft beginnen sie mit einem *I* (für Interface) oder enden mit *able*. In PHP wird das jedoch selten eingehalten. Oft wird es so sein, dass jede Schnittstelle wie bei Klassen in einer extra Datei notiert wird, die Dateien für Schnittstellen jedoch statt mit `class` mit `interface` gekennzeichnet werden.

### Schnittstellen implementieren

Ein weiterer entscheidender Unterschied zu Klassen jeder Art ist die mögliche Verwendung von Schnittstellen. Eine Klasse kann in PHP genau eine andere Klasse erweitern (Einfachvererbung), aber eine Klasse kann eine beliebige Anzahl an Schnittstellen über das Schlüsselwort `implements` implementieren. Diese werden dann bei der Angabe mit Kommata getrennt.

Etwa so:

```
class MeineKlasse implements Interface1, Interface2, Interface3 {  
    ...  
}
```

*Mehrere Schnittstellen implementieren*

Implementiert eine Klasse mehrere Schnittstellen, so entspricht dies einer eingeschränkten Form der Mehrfachvererbung, die es ja in PHP nicht gibt. Da im Gegensatz zur echten Mehrfachvererbung, wie sie beispielsweise C++ bietet, keine Funktionalität (sprich: kein Code) vererbt wird, kann es nicht zu Problemen bei der Vererbung kommen, für die die Mehrfachvererbung berücksichtigt ist.

Mithilfe von Schnittstellen können bestimmte Verhaltensweisen von Objekten modelliert und aus unterschiedlichen Dateien zusammengefügt werden. Weil die Methoden in Schnittstellen abstrakt sind, müssen sie irgendwann in der Vererbungshierarchie überschrieben und damit vervollständigt werden.

Beachten Sie jedoch, dass die Methoden in Schnittstellen explizit nicht mit dem Modifizierer `abstract` gekennzeichnet werden dürfen. Die mehrfach in PHP vorzufindende Philosophie ist hier wohl, dass man selbstverständliche Angaben weglassen soll bzw. muss.

Weiter sollte festgehalten werden, dass die Methoden in Schnittstellen immer öffentlich sind, was beim Überschreiben zu beachten ist. Die Sichtbarkeit kann nicht reduziert werden und deshalb muss man beim Überschreiben immer `public` notieren.

Alle Member in einer Schnittstelle können als Klassenelemente verwendet werden, entweder indirekt über die implementierende Klasse oder über die Schnittstelle selbst.

Schauen wir ein Beispiel mit Schnittstellen an (Ordner *interface*).

## Schnittstellen in der Praxis

Hier ist zuerst das Listing für die folgende Schnittstelle Metainfos in der Datei *Metainfos.interface.php*:

```
<?php
interface Metainfos {
    const VORNAME = "Ralph";
    const NACHNAME = "Steyer";
}
?>
```

*Die erste Schnittstelle*

In der ersten Schnittstelle werden nur zwei Konstanten definiert.

Dieses ist die zweite Schnittstelle in der Datei *Verhaltensweisen.interface.php*:

```
<?php
interface Verhaltensweisen {
    public function telefonieren();
    public function chatten();
}
?>
```

*Die zweite Schnittstelle*

In der zweiten Schnittstelle werden zwei Methoden deklariert. Diese sind abstrakt. Beachten Sie das Fehlen des Modifizierers `abstract` in der Schnittstelle.

Es ist selbstverständlich möglich, Konstanten und die Deklaration von abstrakten Methoden zusammen in einer Schnittstelle zu deklarieren.

Hier ist der Code der Klasse Person, welche die beiden Schnittstellen implementiert:

```
<?php
require_once ("Metainfos.interface.php");
require_once ("Verhaltensweisen.interface.php");
class Person implements Metainfos,
    Verhaltensweisen {
    public function telefonieren() {
        echo "Mit Kunden telefonieren.<br />";
    }
    public function chatten() {
        echo "Mit Kunden chatten.<br />";
    }
}
?>
```

*Die Klasse implementiert zwei Schnittstellen*

In dem Code werden zuerst die PHP-Dateien mit den Schnittstellen hinzugebunden und dann diese in die Klasse implementiert.

Wenn man in der Klasse die Implementierung nicht machen wollte, müsste die Klasse abstrakt deklariert werden. Nur dann kann man verhindern, dass die implementierten abstrakten Methoden dort nicht überschrieben werden müssen. Allerdings müssen diese Methoden dann eben irgendwo im Klassenbaum überschrieben werden und in unserem Beispiel wird das hier bereits gemacht.

In dem folgenden Programm verwenden wir nun die überschriebene Methode, aber auch die Konstanten aus der anderen Schnittstelle.

```
<?php

include_once ("Person.class.php");

class Program {
    public function __construct() {
        echo Person::VORNAME."<hr />";
        echo Metainfos::NACHNAME."<hr />";
        $mensch = new Person();
        $mensch -> chatten();
        $mensch -> telefonieren();
    }
}
new Program();
?>
```

*Verwenden von Methoden und Konstanten*

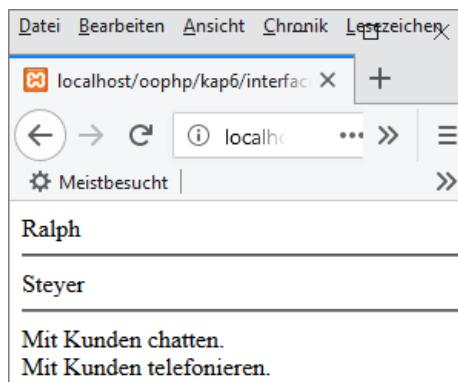
Im Konstruktor wird ein Objekt vom Typ Person erzeugt und die Methoden werden verwendet.

Beachten Sie die Zeilen:

```
echo Person::VORNAME."<hr />";
echo Metainfos::NACHNAME."<hr />";
```

Am Beginn der Klasse werden die Konstanten genutzt, die in der Schnittstelle Metainfos deklariert werden; aber auf zwei unterschiedliche Weisen:

- ✓ Beim ersten Zugriff stellen wir die Klasse voran, die die Schnittstelle implementiert.
- ✓ Beim zweiten Zugriff stellen wir die Schnittstelle selbst voran.



*Die Schnittstellen werden verwendet.*

Doch trotz der Ausführungen zu den Effekten und syntaktischen Regeln bei abstrakten Methoden und Klassen sowie Schnittstellen wurde bisher möglicherweise noch nicht deutlich, wo genau der Nutzen liegt. Das wird im Folgenden erläutert.

## 6.3 Abstrakte Techniken im Praxisprojekt

Wir haben ja in dem Projekt, das in der Unterlage generell und in diesem Kapitel insbesondere in mehreren Versionen vorangetrieben wird, eine Klasse zur Beschreibung eines Kunden. Nun kann man sich vorstellen, dass Kunden in Privatkunden und Geschäftskunden unterteilt werden und diese beim Überziehen eines Disporedits unterschiedliche Zinsen zahlen müssen und auch einen abweichenden Kreditrahmen haben.

Jeder Kunde muss also Zinsen zahlen, wenn der Kontostand bei einer Auszahlung negativ wird. Das wollen wir in dem API abbilden. Aber ein Privatkunde bekommt wie gesagt andere Konditionen als ein Geschäftskunde.

Aufgrund der bisherigen Struktur des Projekts soll in der Methode `auszahlen()` eine Methode zur Prüfung des Kontostands eines Kunden implementiert werden. Offensichtlich ist es aufgrund der abweichenden Rahmenbedingungen sinnvoll, für Privatkunden und Geschäftskunden unterschiedliche Methoden bereitzustellen, die diesen Vorgang beschreiben. Aber da die Methoden für beide Objekttypen im Grunde das Gleiche machen, könnte man die Methode identisch benennen und in der Klasse `Konto` beim Aufruf der Methode `auszahlen()` eine solche Methode aufrufen. Nur muss man erkennen, ob in der Situation das Konto eines Privatkunden oder das eines Geschäftskunden vorliegt und dann in Abhängigkeit davon innerhalb der Methode die weitere Vorgehensweise anpassen.

Sie können auch eine Defaultaktion vorgeben, die für eine der beiden Spezialisierungen dann zu überschreiben ist oder aber noch gar nichts Sinnvolles macht.

Das wirft ein paar Fragen auf:

- ✓ Warum sollte man für einen Fall eine Defaultaktion überhaupt deklarieren? Und wenn ja, welcher ist „wichtiger“, um ihn für die Vorgabe auszuwählen?
- ✓ Was passiert, wenn Sie weitere Subklassen mit anderen Unterscheidungen eines Kunden benötigen?
- ✓ Macht es überhaupt Sinn, bei einem nicht genau spezifizierten Kunden schon Informationen einzeln aufzuführen, die eigentlich erst bei einer Spezialisierung festzulegen sind?

Diese Fragen leiten zur Bedeutung von abstrakten Methoden und Klassen über.

Abstrakte Methoden werden wie normale Methoden vererbt. Das bedeutet, dass alle Subklassen einer abstrakten Klasse die abstrakte Methode als Erbe vorliegen haben. Wenn dieses abstrakte Erbe nicht behandelt ist, wird das bei einer Instanziierung der Subklasse zu einem Fehler führen. Denn durch die Vererbung gibt es in Subklassen dann auch unfertigen Code und damit müssen diese Subklassen abstrakt deklariert werden.

Oder aber die Subklasse kann die Methoden **überschreiben** (override)! Das ist letztendlich auch das Ziel. Denn damit wird fertiger Quellcode bereitgestellt und die Subklassen können instanziert werden. Und durch das Konzept noch mehr erreicht – die Superklasse kann nicht mehr instanziert werden und deshalb kann keine unlogische Konstellation auftreten. Die vertragsbasierte Programmierung stellt ebenso sicher, dass ein „Vergessen“ oder „Missachten der Vorschriften“ aus einer Superklasse nicht mehr möglich ist.

## Das Projekt v1

Die Weiterentwicklung des Praxisprojekts basiert auf dem letzten Stand des vorherigen Kapitels. Die folgenden Dateien bleiben dabei weitgehend unverändert. Nur werden die enthaltenen echo-Anweisungen durch `return` ersetzt:

- ✓ `Person.class.php`
- ✓ `Mitarbeiter.class.php`

Das ist der letzte Schritt dahin, die Qualität des APIs zu optimieren.

Methoden in Klassen eines APIs sollten niemals eine Ausgabe vornehmen. Das Ergebnis sollte in jedem Fall mit `return` geliefert werden. Nur dann kann der Aufrufer der Methoden selbst entscheiden, wie und ob die Meldung auszugeben ist.

Alle anderen Dateien werden modifiziert und es kommen neue Dateien hinzu.

## Die neue Schnittstelle *IAuszahlungen*

Die abstrakte Methode `auszahlen()` soll in einer eigenen Schnittstelle deklariert werden.

```
<?php
interface IAuszahlungen {
    public function auszahlen(float $betrag);
}

?>
```

Die Schnittstelle definiert die Methode `auszahlen()`

Diese Schnittstelle soll in der abstrakten Klasse `Konto` eingebunden werden, die dann die Methodendeklaration nicht mehr benötigt. Der Rest der Klasse bleibt unverändert.

```
<?php
require_once ("IAuszahlungen.interface.php");

abstract class Konto implements IAuszahlungen {

    protected $kontostand = 0;
    private $kontotyp = "";

    public function __construct(float $kontostand, string $kontotyp) {
        $this->kontostand = $kontostand;
        $this->setKontotyp($kontotyp);
    }

    public function einzahlen(float $betrag) {
        $this->kontostand += $betrag;
    }

    public function setKontotyp(string $kontotyp) {
        $this->kontotyp = $kontotyp;
    }

    public function getKontotyp(): string {
        return $this->kontotyp;
    }

    public function getKontostand(): float {
        return $this->kontostand;
    }
}
```

Die Schnittstelle wird implementiert

Zwei weitere Schnittstellen kommen in dem Projekt hinzu und sollen individuelle Konstanten für zwei verschiedene Typen an Konten spezifizieren.

```
<?php
interface IGeschaeftsKonditionen {
    const DISPOZINS = 4.6;
    const KREDITLIMIT = 5000;
    const ZINSSATZ = 0.5;
}
?>
```

*Die Schnittstelle definiert Konstanten für ein Geschäftskonto*

```
<?php
interface IPrivatKonditionen {
    const DISPOZINS = 5.6;
    const KREDITLIMIT = 1000;
}
?>
```

*Die Schnittstelle definiert Konstanten für ein Privatkonto*

Diese Schnittstellen werden jeweils passend in zwei neue Subklassen von Konto eingebunden, in denen dann die Methode auszahlen() mit einer individuellen Logik überschrieben wird. In diesen Klassen kommen dann auch die Konstanten aus den Schnittstellen zum Einsatz.

Das ist die Klasse Privatkonto:

```
<?php
require_once ("Konto.class.php");
require_once ("IPrivatKonditionen.interface.php");
class Privatkonto extends Konto implements IPrivatKonditionen {

    public function __construct(float $kontostand, string $kontotyp) {
        parent::__construct($kontostand, $kontotyp);
    }

    public function auszahlen(float $betrag) {
        if($this->kontostand - $betrag >= 0) {
            $this->kontostand -= $betrag;
            return "Der gewünschte Betrag $betrag EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt " . $this -> getKontostand() .
                " EUR.<br />";
        }
        else if($this->kontostand - $betrag >=
            -IPrivatKonditionen::KREDITLIMIT) {
            $this->kontostand -= $betrag;
            return "Der gewünschte Betrag $betrag EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt " . $this -> getKontostand() . " EUR. " .
                "Sie nutzen für den negativen Betrag einen Dispokredit von " .
                . IPrivatKonditionen::DISPOZINS . "%<br />";
        }
        else if($this->kontostand - $betrag < -
            IPrivatKonditionen::KREDITLIMIT) {

```

```

        return "Ihr Disporahmen ist überschritten. Der gewünschte Betrag
kann nicht ausgezahlt werden.<hr />";
    }
}
?
?>

```

*Die Klasse für das Privatkonto*

Beachten Sie ebenso den Aufruf des Konstruktors der Superklasse. An diesen werden die übergebenen Parameter einfach unverändert weitergereicht.

Das ist die Klasse Geschäftskonto, die nahezu gleich aufgebaut ist, nur eine individuelle Logik beim Auszahlen bereitstellt:

```

<?php
require_once("Konto.class.php");
require_once ("IGeschaeftsKonditionen.interface.php");
class Geschaeftskonto extends Konto implements IGeschaeftsKonditionen
{

    public function __construct(float $kontostand, string $kontotyp) {
        parent::__construct($kontostand, $kontotyp);
    }

    public function auszahlen(float $betrag) {
        if($this->kontostand - $betrag >= 0) {
            $this->kontostand -= $betrag;
            return "Der gewünschte Betrag $betrag EUR wurde
ausgezahlt. Ihr neuer Kontostand beträgt "
            . $this -> getKontostand() . " EUR. " .
            "Für den positiven Einlagebetrag erhalten Sie Zinsen in Höhe von
" .IGeschaeftsKonditionen::ZINSSATZ. "%<br />";
        }
        else if($this->kontostand - $betrag >= -
IGeschaeftsKonditionen::KREDITLIMIT) {
            $this->kontostand -= $betrag;
            return "Der gewünschte Betrag $betrag EUR wurde
ausgezahlt. Ihr neuer Kontostand beträgt " . $this ->
getKontostand() . " EUR. " .
            "Sie nutzen für den negativen Betrag einen Dispokredit von " .
            IGeschaeftsKonditionen::DISPOZINS . "%<br />";
        }
        else if($this->kontostand - $betrag <
-IGeschaeftsKonditionen::KREDITLIMIT) {
            return "Ihr Disporahmen ist überschritten. Der gewünschte
Betrag kann nicht ausgezahlt werden.<hr />";
        }
    }
?
?>

```

*Die Klasse für das Geschäftskonto*

In der Klasse Kunde müssen bei dem gewählten Design nun diese beiden Dateien statt der Datei für das „normale“ Konto eingebunden werden. Der Rest bleibt hier unverändert:

```
<?php
require_once("Person.class.php");
require_once("Privatkonto.class.php");
require_once("Geschaeftskonto.class.php");

class Kunde extends Person {
...
?>
```

*Nur die Dateien müssen hier eingebunden werden.*

Und das ist das Listing der neuen Programmdatei:

```
<?php
require_once("Kunde.class.php");
require_once("Mitarbeiter.class.php");

class Program {
    public function __construct() {
        $this -> main();
    }

    private function main() {
        $kontol = new Privatkonto(floatval("1023.34"), "Girokonto");
        $kunde1 = new Kunde("Hans", "Dampf", 42, "Mann", "12345 Irgendwo,
Irgendeinestrasse 42", $kontol);
        echo "Allgemeiner Geschäftsvorfall von " . $kunde1->getVorname() .
" " . $kunde1->getNachname() . "<br />";
        echo $kunde1 -> geschaeftsvorfall();
        echo "Auszahlungsvorfall von " . $kunde1->getVorname() . " " .
$kunde1->getNachname() . "<br />";
        echo $kontol-> auszahlen(1000);
        echo "Auszahlungsvorfall von " . $kunde1->getVorname() . " " .
$kunde1->getNachname() . "<br />";
        echo $kontol-> auszahlen(1000);

        $kontol2 = new Geschaeftskonto(floatval("1023.34"), "Girokonto");
        $kunde2 = new Kunde("Otto", "Schmidt", 56, "Mann", "12345 Irgendwo,
Irgendeinestrasse 2", $kontol2);
        echo "Auszahlungsvorfall von " . $kunde2->getVorname() . " " .
$kunde2->getNachname() . "<br />";

        echo $kontol2-> auszahlen(1000);
        echo "Auszahlungsvorfall von " . $kunde2->getVorname() . " " .
$kunde2->getNachname() . "<br />";
        echo $kontol2-> auszahlen(1000);
        echo "Auszahlungsvorfall von " . $kunde2->getVorname() . " " .
$kunde2->getNachname() . "<br />";
```

```

        echo $konto2-> auszahlen(1000);
        echo "Auszahlungsvorfall von " . $kunde2->getVorname() . " " .
        $kunde2->getNachname() . "<br />";
        echo $konto2-> auszahlen(4000);
    }
}
new Program();
?>

```

### Die neue Programmklasse

The screenshot shows a web browser window with the following content:

- Header:** Datei, Bearbeiten, Ansicht, Chronik, Lesezeichen, Extras, Hilfe.
- Title Bar:** localhost/oophp/kap6/v1/Prog X | +
- Address Bar:** localhost/oophp/kap6/v1/Program.class.php
- Toolbar:** Back, Forward, Stop, Refresh, Search, etc.
- Links:** Meistbesucht, http://localhost/phpk..., Google, Wikipedia, Facebook, Twitter, LinkedIn, Wetter Online, Meistbesucht.
- Main Content:**
  - Allgemeiner Geschäftsvorfall von Hans Dampf  
Bisheriger Kontostand: 1023.34 EUR.
  - Kontakt zwischen Kunde und Mitarbeiter.  
Der gewünschte Betrag 100 EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt 923.34 EUR.  
Neuer Kontostand: 923.34 EUR.
  - Auszahlungsvorfall von Hans Dampf  
Der gewünschte Betrag 1000 EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt -76.66 EUR. Sie nutzen für den negativen Betrag einen Dispokredit von 5.6%  
Auszahlungsvorfall von Hans Dampf  
Ihr Disporahmen ist überschritten. Der gewünschte Betrag kann nicht ausgezahlt werden.
  - Auszahlungsvorfall von Otto Schmidt  
Der gewünschte Betrag 1000 EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt 23.34 EUR. Für den positiven Einlagebetrag erhalten Sie Zinsen in Höhe von 0.5%  
Auszahlungsvorfall von Otto Schmidt  
Der gewünschte Betrag 1000 EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt -976.66 EUR. Sie nutzen für den negativen Betrag einen Dispokredit von 4.6%  
Auszahlungsvorfall von Otto Schmidt  
Der gewünschte Betrag 1000 EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt -1976.66 EUR. Sie nutzen für den negativen Betrag einen Dispokredit von 4.6%  
Auszahlungsvorfall von Otto Schmidt  
Ihr Disporahmen ist überschritten. Der gewünschte Betrag kann nicht ausgezahlt werden.

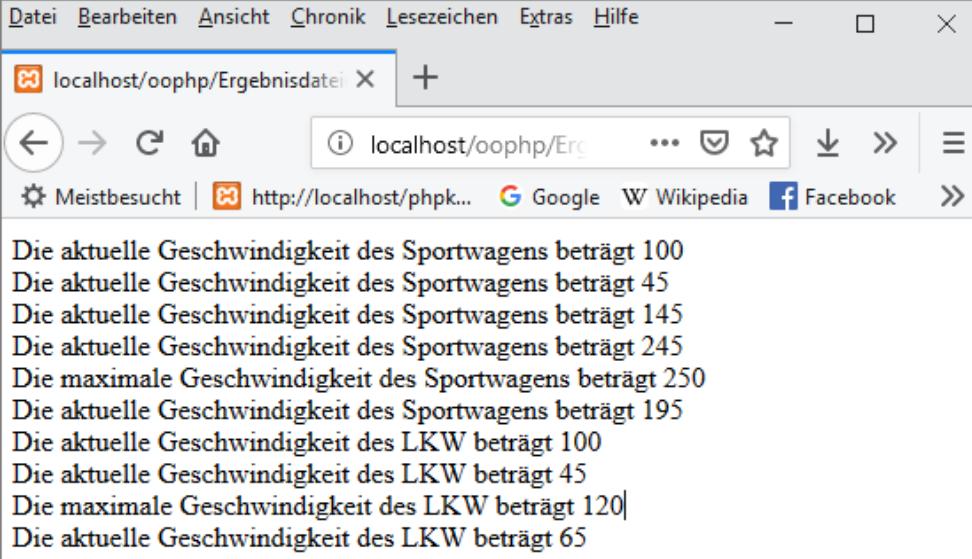
### Die Ausgaben der Programmklassen

## 6.4 Übungen

**Übungsdateien:** --

**Ergebnisdateien:** *Auto.class.php*,  
*Sportwagen.class.php*,  
*Lastkraftwagen.class.php*,  
*Program.class.php*

1. In der Superklasse Auto sollen die beiden Methoden zum Bremsen und Beschleunigen als abstract erklärt werden.
2. Überschreiben Sie die Methoden in den beiden Subklassen Sportwagen und Lastkraftwagen entsprechend der Regeln, die in den Übungen in Kapitel 5 vorgegeben sind. Geben Sie individuelle Meldungen aus – auch für die Methode bremsen () .
3. Erstellen Sie eine Programmklasse und erzeugen Sie dort von beiden Subklassen eine Instanz.
4. Beschleunigen und bremsen Sie die Instanzen mehrfach.



The screenshot shows a web browser window with the following content:

```
Die aktuelle Geschwindigkeit des Sportwagens beträgt 100
Die aktuelle Geschwindigkeit des Sportwagens beträgt 45
Die aktuelle Geschwindigkeit des Sportwagens beträgt 145
Die aktuelle Geschwindigkeit des Sportwagens beträgt 245
Die maximale Geschwindigkeit des Sportwagens beträgt 250
Die aktuelle Geschwindigkeit des Sportwagens beträgt 195
Die aktuelle Geschwindigkeit des LKW beträgt 100
Die aktuelle Geschwindigkeit des LKW beträgt 45
Die maximale Geschwindigkeit des LKW beträgt 120
Die aktuelle Geschwindigkeit des LKW beträgt 65
```

So ungefähr könnte die Ausgabe aussehen.

# 7

## Eine objektorientierte Webseite

### 7.1 Die Weiterentwicklung des Praxisprojekts

Unser Praxisprojekt soll nun mit Benutzereingaben interagieren. Als Anfangssituation soll vorausgesetzt werden, dass es bereits ein Privatkonto mit einem gewissen Kontostand gibt und der Besucher über ein Webinterface einen bestimmten Betrag von seinem Konto abbuchen kann.

Das Webinterface beinhaltet dazu ein kleines Formular und zeigt die relevanten Informationen in der Webseite an. Dazu wird diese mit Style Sheets (CSS) geringfügig gestaltet.

Da eine Abbuchung den Kontostand verändert, muss dem bei einer erneuten Abbuchung Rechnung getragen werden. Dauerhaft kann eine solche Information nur in einer Datenbank gespeichert werden (was im nächsten Kapitel erklärt wird), aber zumindest temporär kann man den Kontostand auch in einer Session speichern. Das wird im folgenden Beispiel demonstriert.

The screenshot shows a web browser window with the following details:

- Header:** Datei, Bearbeiten, Ansicht, Chronik, Lesezeichen, Extras, Hilfe.
- Title Bar:** Auszahlung Konto.
- Address Bar:** localhost/oophp/kap7/v1/Program.class.php
- Toolbar:** Back, Forward, Stop, Refresh, Search, etc.
- Links:** Meistbesucht, http://localhost/php..., Google, Wikipedia, Facebook, Twitter, LinkedIn, Wetter Online, Meistbesucht.
- Content Area:**
  - A large input field labeled "Gewünschter Auszahlungsbetrag".
  - An input field labeled "Betrag" with a placeholder value.
  - A button labeled "Daten absenden".
  - A message box containing the text "Der Kontostand beträgt aktuell 1023.34 EUR."

Das Webinterface, über das ein Kunde sich einen Betrag auszahlen lassen kann

## Die Projektstruktur

In der Weiterentwicklung des Projekts sollen externe Style Sheets und Textbausteine mit HTML-Fragmenten verwendet werden. Diese werden in Unterverzeichnissen organisiert:

- ✓ Die Style Sheet-Datei soll in einem Verzeichnis *lib/css* stehen.
- ✓ Die Textbausteine befinden sich in einem Verzeichnis *lib/res*.

So eine Aufteilung ist in der Praxis so gut wie immer zu finden. Auch weitere externe Ressourcen wie JavaScripts, Grafiken, Videos, Audios etc. werden in solchen Unterverzeichnisse abgelegt.

## Die CSS-Datei *Konto.css*

Das ist der Aufbau der CSS-Datei, wobei die Details irrelevant sind. Es werden einige HTML-Tags und ein ID in geringem Maße gestaltet.

```
body{  
    background:lightgray;  
    color:blue;  
}  
h1, #info{  
    background:white;  
    box-shadow: 5px 5px #111;  
    border-radius: 5px;  
    text-align:center;  
    margin-left:20px;  
    margin-right:20px;  
    padding:10px;  
}  
label{  
    display:inline-block;  
    margin-left:20px;  
    padding:5px;  
}  
input{  
    padding:5px;  
    box-shadow: 5px 5px #111;  
    border-radius: 5px;  
    margin:10px;  
    padding:5px;  
}
```

Diese Datei muss dann im Header der Webseite eingebunden werden. Dieser wird samt dem restlichen Beginn des Grundgerüsts der Webseite in eine Textdatei ausgelagert.

## Die Text-Datei *kopf.txt*

In der Textdatei befindet sich der Beginn des Grundgerüsts der Webseite.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="uft-8">
<title>Auszahlung Konto</title>
<link href="lib/css/konto.css" type="text/css" rel="stylesheet">
</head>
<body>
```

Die wichtigste Stelle ist die Verbindung zur Style Sheet-Datei.

## Die Text-Datei *formular.txt*

Nun soll in der Webseite ein Formular angezeigt werden und auch das soll in eine externe Textdatei ausgelagert werden:

```
<h1>Gewünschter Auszahlungsbetrag</h1>
<form action=<?=$_SERVER['PHP_SELF']?>" method="post">
<label for="">Betrag</label><input name="betrag"/><br />
<input type="submit"/><br />
</form>
```

Sie sehen den Quellcode für ein einfaches Formular mit einem Eingabefeld und einer Schaltfläche zum Absenden des Formulars. Wichtig ist, dass die Daten wieder an die aktuelle Datei selbst verschickt werden. Dazu wird die übliche Super Global S\_SERVER verwendet.

## Die Text-Datei *ende.txt*

Die dritte Textdatei enthält nur das Ende des Grundgerüsts. Man kann sich aber vorstellen, dass hier auch andere Informationen untergebracht werden können.

```
</body>
</html>
```

Wenn diese Ressourcen zusammengesetzt werden, entsteht die gewünschte Webseite.

The screenshot shows a Windows-style application window titled "Auszahlung Konto". The address bar indicates the URL is "localhost/oophp/kap7/v1/Program.class.php". The main content area contains a form with a title "Gewünschter Auszahlungsbetrag". It has a text input field labeled "Betrag" and a button labeled "Daten absenden". Below the form, a message box displays the text: "Der gewünschte Betrag 567 EUR wurde ausgezahlt. Ihr neuer Kontostand beträgt 456.34 EUR."

*Es wurde ein gewisser Betrag ausgezahlt und der neue Kontostand wird in der Session vermerkt*

Das Formular wird nicht plausibilisiert und/oder gegen Fehleingaben geschützt. Das muss in der Praxis natürlich gemacht werden, führt hier aber zu weit. Es wird vorausgesetzt, dass ein Anwender das Eingabefeld korrekt ausfüllt.

## 7.2 Die Programmdatei

Die Fragmente der Webseite sind nun vorhanden und durch die Wahl des bisherigen OO-Designs können sämtliche Klassen des APIs unverändert bleiben – zumindest für die Zielsetzung, die hier gezeigt werden soll. Damit wird ganz deutlich, welche Vorteile ein sorgfältig geplantes und so gut wie möglich gekapseltes Design bietet.

Es muss jetzt nur noch die „Businesslogik“ erstellt werden. Und das erfolgt in der aktualisierten Programmklasse:

```
<?php
session_start();
require_once("Kunde.class.php");
require_once("Mitarbeiter.class.php");

class Program {
    public function __construct(){
        $this -> main();
    }

    private function main(){
        require_once("lib/res/kopf.txt");
        require_once("lib/res/formular.txt");
        if( isset($_POST['betrag']) ) {
            $kontol = new Privatkonto(
                floatval($_SESSION["kontostand"]), "Girokonto");
            $kunde1 = new Kunde("Hans", "Dampf", 42, "Mann",
                "12345 Irgendwo, Irgendeinestrasse 42", $kontol);
            $meldung = $kontol-> auszahlen(floatval($_POST['betrag']));
        }
    }
}
```

```

echo "<div id='info'>$meldung </div>"; }
else {
    $kontol = new Privatkonto(floatval("1023.34"), "Girokonto");
    $kunde1 = new Kunde("Hans", "Dampf", 42, "Mann", "12345 Irgendwo,
                        Irgendeinestrasse 42", $kontol);
    $meldung = "Der Kontostand beträgt aktuell "
               . $kontol->getKontostand() . " EUR.";
    echo "<div id='info'>$meldung </div>";
}
$_SESSION["kontostand"]=$kontol->getKontostand();
require_once("lib/res/ende.txt");
}
}
new Program();
?>

```

Die Index-Seite

Zuerst wird eine Session erstellt. Darüber soll der Kontostand weitergegeben werden.



Ab sofort müssen Kreditzinsen gezahlt werden und der neue Kontostand wurde in die Session geschrieben

In der `main()`-Methode werden die einzelnen Textdateien in der vorgesehenen Reihenfolge eingebunden. Nun muss unterschieden werden, ob die Datei mit oder ohne Datenübergabe per POST aufgerufen wird.

- ✓ Wenn das Formular abgeschickt wird, steht der auszuzahlende Betrag in dem Super Global `$_POST`. Deshalb wird unterschieden, ob `$_POST['betrag']` gesetzt wurde (es wurde ein Betrag zum Abbuchen im Formular abgeschickt) oder nicht (erster Aufruf der Seite).
- ✓ Wurde die Seite zum ersten Mal aufgerufen, wird ein Konto mit einem Vorgabebetrag erstellt. Dazu wird eine Standardmeldung in der Webseite angezeigt, indem ein `div`-Container mit entsprechendem Inhalt generiert wird. Der Kontostand wird in die Session geschrieben.
- ✓ Wird ein Betrag zur Abbuchung mit dem Formular verschickt, wird ein Kontoobjekt mit dem Betrag erstellt, der in der Session steht. Ebenso wird die Methode `auszahlen()` aufgerufen. Auch dann wird der neue Kontostand in die Session geschrieben.

Das ist die gesamte Logik. Alle bisher schon implementierten Regeln hinsichtlich des Disporahmens etc. stehen unverändert und ohne Anpassung zur Verfügung.

The screenshot shows a web browser window with the title "Auszahlung Konto". The address bar displays "localhost/oophp/kap7/v1/Program.class.php". The main content area has a header "Gewünschter Auszahlungsbetrag". Below it is a form field labeled "Betrag" with an empty input box. A button labeled "Daten absenden" is visible. A message box at the bottom contains the text "Ihr Disporahmen ist überschritten. Der gewünschte Betrag kann nicht ausgezahlt werden." (Your withdrawal limit has been exceeded. The requested amount cannot be withdrawn.)

*Der Disporahmen wurde überschritten*

## 7.3 Übung

**Übungsdateien:** Verzeichnis v1

**Ergebnisdateien:** --

1. Modifizieren Sie die CSS-Formatierungen nach Ihrem Geschmack.
2. Passen Sie bei Bedarf die angezeigten Inhalte mit weiteren Informationen an.

# 8

## Objektorientierter Datenbankzugriff

### 8.1 PHP und Datenbanken

Kaum eine etwas anspruchsvollere Webanwendung kommt heutzutage noch ohne die Anbindung an eine Datenbank aus. Eines der wichtigsten Features von PHP war von Anfang an die sehr gute Unterstützung für Datenbanken. PHP bietet bereits mit diversen integrierten Sprachmerkmalen beste Voraussetzungen zur Kommunikation mit einer Datenbank bzw. einem Datenbankmanagementsystem. PHP kann dabei mit verschiedenen Datenbanksystemen umgehen. PHP bietet Schnittstellen zu über zwanzig verschiedenen Datenbanksystemen – in der Praxis hat sich aber besonders die Kombination aus PHP und MySQL etabliert. Im Rahmen des Themas dieser Unterlage soll der objektorientierte Datenbankzugriff eine wichtige Umsetzung der OO-Theorie der vorherigen Kapitel als praktische Anwendung zeigen.

#### Datenbank versus DBMS (Datenbankmanagementsystem)

Wenn man mit Datenbanken umgeht, muss man eine „einfache“ Datenbank von einem erheblich leistungsfähigeren und flexibleren Managementsystem für Datenbanken (DBMS) differenzieren. Eine Datenbank ist erst einmal nur eine Sammlung von Informationen, die so geordnet sind, dass sie auf einfache Weise abgerufen, verwaltet und aktualisiert werden können. Datenbanken können daher auch einzelne Dateien oder aber eine Gruppe zusammengehörender Dateien sein, die als einheitliche „Datenbasis“ zu verstehen sind. Diese Datenbasis kann direkt verwendet werden, wenn man deren Struktur kennt, etwa eine Access-Datei.

Das DBMS ist eine Verwaltungssoftware für solch eine Datenbasis. Es muss schnell, sicher, flexibel und vor allen Dingen stabil sein. So ein Datenbankmanagementsystem kann dazu

- ✓ mehrere Datenbanken verwalten,
- ✓ verschiedene Benutzer zulassen,
- ✓ Benutzern unterschiedliche Rechte einräumen,
- ✓ Lasten verteilen und Ressourcen effektiv nutzen und
- ✓ die gesamten Zugriffe auf die konkreten Dateien abstrahieren und damit sowohl die konkreten Dateien als auch die Speicherungsorte vor dem Anwender verbergen.

Ein DBMS agiert damit als Serversystem, über das sämtliche Anfragen auf eine konkrete Datenbank bzw. Datenbasis laufen müssen.

## Das MySQL-RDBMS

Bei MySQL handelt es sich um ein relationales Datenbankmanagementsystem (RDBMS), was eine spezielle Form eines DBMS charakterisiert. Allerdings hat sich in der Praxis dennoch nur DBMS als Abkürzung etabliert.

Neben einigen kommerziellen Lizenzen gibt es MySQL in einer Community Edition als Open-Source-Software. Deshalb ist MySQL sehr weit verbreitet und in vielen Hosting-Angeboten bei den gängigen Providern inkludiert. Wobei aufgrund des Lizenzierungsmodells des „Eigentümers“ von MySQL (Oracle) mit MariaDB eine Abspaltung – ein sogenannter Fork – des MySQL-Projekts zunehmend an Akzeptanz und Bedeutung gewinnt. MariaDB entspricht in weiten Teilen MySQL und verwendet zum Beispiel auch dieselben Schnittstellen zu PHP.

In einem RDBMS wie MySQL werden die Daten in Tabellen (Relationen) organisiert, die in einer bestimmten Beziehung zu anderen Tabellen derselben Datenbank stehen können.

## Grundsätzlicher Ablauf eines Datenbankzugriffs aus PHP

Der Ablauf einer Zusammenarbeit von PHP und MySQL (oder auch einem anderen DBMS) ist immer ähnlich und besteht aus den folgenden Schritten:

1. Verbindung zum MySQL-Server herstellen, Datenbank auswählen und Anmeldeinformationen senden.
2. Abfrage(n) senden.
3. Ergebnis(se) verarbeiten, aufbereiten und ausgeben.
4. Verbindung schließen.

## Die Datenbankschnittstellen MySQL und MySQLi

Um aus PHP auf das MySQL-RDBMS zuzugreifen, gab es nahezu von Anfang an eine Datenbankschnittstelle (auch Erweiterung genannt) mit dem Namen MySQL, dem später eine etwas modernere Variante mit Namen MySQLi folgte.

Die Namen der Funktionen aus dem PHP-API, welche die jeweilige Datenbankschnittstelle nutzen, beginnen in der Regel entweder mit *mysql* oder *mysqli* und sind in der Anwendung meist sehr ähnlich.

Die Funktionen aus der MySQL-Schnittstelle sind rein prozedural, während die Funktionen aus MySQLi sowohl prozedurale Programmierung als auch objektorientierte Programmierung unterstützen. Beide sind jedoch veraltet, teils ziemlich umständlich, aufwändig sowie inkonsistent und mit PHP 7 wurde die Unterstützung der ganz alten MySQL-Schnittstelle komplett eingestellt.

## 8.2 Die moderne Art des Datenbankzugsriffs – PDO

Außer für die Pflege von altem Code oder einer veralteten PHP-Version ohne PDO-Unterstützung als Basis gibt es kaum einen Grund dafür, nicht die moderne Art des Datenbankzugsriffs zu nutzen. Und diese nennt sich PDO (PHP Data Objects).

PDO gibt es seit PHP 5. Der Einsatz von PDO bringt eine ganze Reihe an Vorteilen:

- ✓ Ein sehr positiver Aspekt ist, dass Sie damit einheitlich auf verschiedene Datenbanksysteme zugreifen können.
- ✓ PDO ist rein objektorientiert und unterstützt alle relevanten Techniken und Denkweisen der OOP.
- ✓ Die PDO-Methoden liefern in der Regel entweder einen booleschen Wert oder ein Objekt bzw. PHP-Array als Ergebnis. Das bedeutet, dass man mit klassischen OO- und/oder PHP-Techniken auf die Ergebnisse einer Abfrage zugreifen kann und nicht wie früher spezielle zusätzliche Datenbankfunktionen oder -methoden zur weiteren Auswertung der Ergebnisse nutzen muss.
- ✓ PDO unterstützt noch weiter als die alten Schnittstellen den Umgang mit „Prepared Statements“.

Wie auch bei Zugriffen über MySQL oder MySQLi nutzt man bei PDO zur Formulierung der Abfragen die einheitliche Programmiersprache SQL (Structured Query Language).

### Übersicht SQL

SQL ist eine neutrale Abfragesprache, mit der ein Datenbankclient und Datenbankserver kommunizieren können. Die eigentliche Anwendung der SQL-Anweisungen muss in einem Clientprogramm oder einer Programmiersprache – zum Beispiel in PHP – erfolgen.

SQL ist standardisiert und wird von sehr vielen RDBMS grundsätzlich unterstützt. Allerdings hält sich kaum eines der Systeme exakt an den SQL-Standard, sondern bietet im Detail individuelle Lösungen an. Das gilt auch und besonders für MySQL.

Man teilt SQL-Befehle, die **nicht** Case-sensitiv sind, nach deren Verwendungszweck in drei wesentliche Gruppen auf:

SQL-Befehl	Verwendungszweck
Datendefinitionsbefehle (DDL – Data Definition Language)	Die DDL umfasst Befehle, um ein Datenmodell physisch umzusetzen. Damit werden also die Strukturen erstellt. Dazu gehören beispielsweise das Anlegen, Bearbeiten und Löschen von Datenbanken und Tabellen. Zentraler Befehl ist hier <code>create</code> .
Datenmanipulationsbefehle (DML – Data Manipulation Language)	Die DML umfasst Befehle zum Arbeiten mit den Daten selbst. Dazu zählen beispielsweise das Hinzufügen ( <code>insert</code> ), Aktualisieren ( <code>update</code> ) und Löschen ( <code>delete</code> ), aber auch das Abfragen ( <code>select</code> ) von Daten.
Datenbankkontrollbefehle (DCL – Data Control Language)	Die DCL stellt administrative Befehle zur Verfügung, etwa <code>grant</code> . Darüber kann man z. B. Zugriffsrechte von Benutzern vergeben oder auch entziehen.

In der Praxis werden Sie sich selten Gedanken darüber machen, zu welcher der drei Gruppen ein Befehl konkret gehört. Die wichtigste Gruppe aus einem PHP-Skript wird aber in der Regel die DML sein, denn damit manipulieren Sie Daten oder fragen diese ab.

## Die PDO-Klassen

Da das PDO-Konzept rein objektorientiert ist, bedeutet das, dass es Klassen gibt, welche die Techniken für die Datenbankzugriffe bereitstellen. Die drei wichtigsten Klassen sollten folgende sein:

- ✓ Die Klasse `PDO` ist hauptsächlich dafür verantwortlich, eine Verbindung zum DBMS und zur Datenbank aufzubauen, aber auch SQL-Kommandos an die Datenbank zu schicken und die Verbindung wieder zu schließen. Außerdem stellt sie über einige Eigenschaften beziehungsweise Methoden verschiedene Verbindungsinformationen bereit.
- ✓ Über die Klasse `PDOSatement` greifen Sie auf ein Ergebnis einer Abfrage zu. Das ist ein spezielles Objekt mit allen dafür relevanten Daten und Methoden.
- ✓ Wenn bei Datenbankaktionen Ausnahmen geworfen werden, können Sie die Klasse `PDOException` verwenden.

Auf Basis dieser drei Klassen kann man bereits die meisten Datenbankoperationen durchführen.

## Mit PDO Verbindungen aufbauen und Datenbanken auswählen

Um mit PDO eine Verbindung zum DBMS und einer konkreten Datenbank herzustellen, instanziiieren Sie zuerst ein Objekt der Klasse `PDO`. Die PHP-Dateien zu diesem Abschnitt finden Sie im Ordner `pdo`.

- ✓ Dem Konstruktor der Klasse können Sie in Form eines einzigen Strings optional Parameter für den Verbindungsaufbau übergeben.
- ✓ Sofern Sie den leeren Konstruktor nutzen, geben Sie anschließend über die Methode `connect()` des `PDO`-Objekts die notwendigen Verbindungsinformationen an.
- ✓ Weitere Parameter des Konstruktors regeln den konkreten Zugriff.

Die Angaben für einen Verbindungsaufbau zu einer Datenbank über das MySQL-DBMS im ersten Parameter des Konstruktors sind folgende:

- ✓ Der Name des Datenbankhosts (etwa `localhost`). Das ist eine notwendige Angabe.
- ✓ Die Angaben zur Datenquelle. Auch diese Information muss zwingend bereitstehen.

Um dann den Zugriff zu steuern, werden in der Regel noch genauere Informationen benötigt. Etwa diese, welche als weitere Parameter an den Konstruktor übergeben oder mit geeigneten Methoden im Nachhinein angegeben werden:

- ✓ Benutzername
- ✓ Passwort dieses Benutzers
- ✓ Name der Datenbank
- ✓ Optionale weitere Parameter wie einen Port, wenn das erforderlich ist

Die formale Syntax des Konstruktors der Klasse PDO lautet wie folgt:

```
public PDO::__construct ( string $dsn [, string $username [, string $password [, array $options ]]] )
```

*Das Schema des PDO-Konstruktors*

Für eine Datenbank oophp mit einem User root mit leerem Passwort könnte der Aufruf des Konstruktors zum Beispiel folgendermaßen aussehen:

```
<?php
    $pdo = new PDO('mysql:dbname=oophp;host=localhost', 'root', '');
?>
```

*Verbindungsaufbau mit PDO*

Obwohl der Konstruktor für die Datenquelle nur einen String übergeben bekommt, hat dieser eine „innere“ Struktur, die zwei „innere“ Parameter beinhaltet. Beide „inneren“ Parameter sind innerhalb des Strings durch Semikolon voneinander getrennt. Beachten Sie, dass damit der User und das Passwort eine zusätzliche Struktur (weitere Parameter des Konstruktors) sind und nicht zu dieser „inneren“ Logik zählen.

Schauen wir uns den ersten Parameter des Konstruktors einmal etwas näher an. Es wurde zuvor bereits erwähnt, dass es sich hierbei um Angaben zur Datenquelle handelt. Für die Angaben zur Datenquelle wird sehr häufig die englische Bezeichnung **Data Source Name** oder kurz **DSN** verwendet. Der DSN wird als Zeichenkette angegeben.

- ✓ Zunächst steht darin der zu verwendende PDO-Datenbanktreiber gefolgt von einem Doppelpunkt. Da wir hier mit MySQL beziehungsweise MariaDB arbeiten, ist das mysql:.
- ✓ Danach folgen treiberspezifische Parameter, in unserem Fall sind das die folgenden zwei:
  - ✓ Mit dbname wird der Datenbankname angegeben.
  - ✓ Mit host wird der Datenbankhost spezifiziert.

## Ausnahmen abfangen

In der Praxis wird in der Regel bei der Instanziierung von PDO direkt überprüft, ob der Verbindungsaufbau erfolgreich war. Ist der Verbindungsaufbau gescheitert, wird entsprechend reagiert. Wenn beim Verbindungsaufbau etwas schief läuft, dann löst das eine Ausnahme vom Typ **PDOException** aus. Entsprechend baut man um die Erzeugung einer PDO-Instanz eine Ausnahmebehandlung auf.

```
<?php
try {
    $pdo = new PDO('mysql:dbname=oophp;host=localhost', 'root', '');
} catch (PDOException $e) {
    die("Es ist ein Fehler aufgetreten!");
}
```

*Absichern der Instanziierung*

In einem `try`-Block wird versucht, das PDO-Objekt zu instanziieren. Wenn das nicht klappt, wird eine `PDOException` ausgelöst und diese im `catch`-Block abgefangen.

In unserem Beispiel verwenden wir die Funktion `die()`, um eine Meldung auszugeben und das Skript dann zu beenden. Alternativ können Sie auch die Funktion `exit()` benutzen oder natürlich auch eine andere Reaktion bereitstellen.

Wenn genaue Informationen zum Fehler ausgegeben werden sollen, können Sie das Objekt vom Typ `PDOexception` nutzen. Die Methode `getMessage()` enthält zum Beispiel die konkrete Fehlermeldung und die Eigenschaft `connect_error` zeigt im Fehlerfall als Wert eine Beschreibung des letzten Verbindungsfehlers an. Im Erfolgsfall ist der Wert der Eigenschaft `null`.

## Zeichensatz der Verbindung

Beim Zugriff auf Datenbanken kann es sein, dass sprachspezifische Sonderzeichen nicht richtig dargestellt werden. Das betrifft die Umlaute genauso wie das ß, auch wenn in der Datenbank und auf der Webseite bzw. in der PHP-Datei die entsprechenden Einstellungen gemacht wurden. Selbst eine Metaangabe mit der Kodierung wie mit `<meta charset="utf-8" />` genügt nicht unbedingt.

Wenn Sie einen konkreten Zeichensatz wie UTF-8 in Ihren PHP-Skripten benutzen, sollten Sie das konsequent für **alle Ebenen** umsetzen. Das betrifft:

- ✓ alle Dateien,
- ✓ alle Daten in der Datenbank,
- ✓ alle Verbindungen zur Datenbank und
- ✓ die Anzeige durch den Browser.

Bei der Verbindung zur Datenbank kann man beim Erzeugen des PDO-Objekts dem Data Source Namen einen Parameter `charset` hinzufügen und dort den Zeichensatz (etwa den Wert `utf8`) angeben. Damit wird für den inneren Aufbau der DSN-Struktur ein weiterer „innerer“ Parameter notiert, der im String wieder durch Semikolon getrennt wird. Das sieht dann folgendermaßen aus:

```
<?php
try {
    $pdo = new PDO('mysql:dbname=oophp;host=
localhost; charset=utf8',
    'root', '');
} catch(PDOException $e) {
    die("Es ist ein Fehler aufgetreten!");
}

?>
```

*Zeichensatz festlegen*

Sie müssen das Setzen des Zeichensatzes übrigens nicht extra prüfen. Wenn dabei etwas nicht richtig sein sollte, wird eine Ausnahme geworfen.

## 8.3 Abfragen senden

Nachdem das PDO-Objekt die Verbindung erfolgreich aufgebaut hat und die Datenbank ausgewählt ist, wird man im nächsten Schritt Abfragen auf der Datenbank ausführen. Dabei umfasst dies nicht nur das Abfragen, sondern auch das Einfügen, Löschen oder Aktualisieren von Daten. Dieser Schritt erfolgt einheitlich. SQL-Statements können Sie in PHP mit der Methode `query()` der Klasse PDO an das RDBMS schicken. Die Logik wird allein von SQL bestimmt. Nur der Umgang mit dem Ergebnis divergiert.

- ✓ Bei einer Abfrage von Daten werden Sie diese weiter verarbeiten.
- ✓ Beim Erstellen, Einfügen, Löschen oder Aktualisieren von Daten oder Strukturen genügt es zu wissen, ob die Aktion erfolgreich war oder nicht.

Die formale Syntax von `query()` lautet in der einfachsten Form:

```
public PDOStatement PDO::query(string $statement) query (string
$query)
```

*Die formale Logik von `query()`*

Das SQL-Kommando wird vom Typ `String` als Parameter übergeben. Der Rückgabewert von `query()` hängt von der Art des SQL-Befehls und vom Erfolg der Ausführung ab.

- ✓ Schlägt die Ausführung des Kommandos fehl, liefert `query()` immer `false` zurück.
- ✓ Ist das SQL-Kommando ein `SELECT`, `SHOW`, `DESCRIBE` oder `EXPLAIN`, liefert die Methode `query()` ein Objekt vom Typ `PDOStatement` zurück. Damit kann man dann mit verschiedenen Methoden das Ergebnis weiterarbeiten.
- ✓ Bei allen anderen SQL-Kommandos (zum Beispiel `CREATE`, `DROP`, `INSERT`, `UPDATE` etc.) liefert `query()` nach erfolgreicher Ausführung `true` zurück.

Diese Systematik ist wesentlich einfacher und logischer als es früher bei der MySQLi- oder gar MySQL-Schnittstelle war.

### Ergebnisse einer Abfrage verarbeiten, aufbereiten und ausgeben

Zuerst wird der Fall betrachtet, dass mit einem `select`-Befehl Daten aus einer Entität (Tabelle) ausgewählt werden. Als Beispiel sollen zunächst alle Datensätze einer Tabelle `kunde` selektiert werden.

Deren Struktur kann man mit `phpMyAdmin` anlegen. In Hinblick auf eine Synchronisierung mit den Klassen des Praxisprojekts ist folgende Struktur sinnvoll, die das nachfolgende SQL-Statement beschreibt:

```
--  
-- Datenbank: `oophp`  
--  
-- -----  
--  
-- Tabellenstruktur für Tabelle `kunde`  
--
```

```

CREATE TABLE `kunde` (
  `id` int(11) NOT NULL,
  `vorname` varchar(30) NOT NULL,
  `nachname` varchar(30) NOT NULL,
  `_alter` int(11) NOT NULL,
  `geschlecht` varchar(10) NOT NULL,
  `adresse` text NOT NULL,
  `kontoid` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- 
-- Daten für Tabelle `kunde`
-- 

INSERT INTO `kunde` (`id`, `vorname`, `nachname`, `_alter`,
`geschlecht`, `adresse`, `kontoid`) VALUES
(1, 'Hans', 'Dampf', 42, 'Mann', '12345 Irgendwo,  
Irgendeinestrasse 42', 1),
(2, 'Otto', 'Schmitt', 56, 'Mann', '12345 Irgendwo,  
Irgendeinestrasse 2', 2),
(3, 'Wilma', 'Feuerstein', 32, 'Frau', '44565 Steinzeit, Steinweg  
1', 3);

```

Der SQL-Dump zeigt die Struktur und den Inhalt einer Relation kunde

Sie erkennen an der Struktur der Relation die Eigenschaften eines Objekts vom Typ Kunde wieder, wie diese in der Klasse Kunde in den vorherigen Kapiteln festgelegt wurde. Dazu gibt es noch eine Id, um die Datensätze zu unterscheiden (Primary Key). Das Feld kundenid soll ein Fremdschlüssel auf eine andere Entität mit dem Namen konto darstellen, ohne dass dies mit einem Constraint explizit markiert wird. Diese Beziehung wurde so auch im Design der Klassen realisiert.

#	Name	Typ	Kollation	Attribute	Null	Standard	Kommentare	Extra	Aktion
1	<b>id</b>	int(11)			Nein	kein(e)		AUTO_INCREMENT	Bearbeiten  Löschen  Mehr
2	<b>vorname</b>	varchar(30)	latin1_swedish_ci		Nein	kein(e)			Bearbeiten  Löschen  Mehr
3	<b>nachname</b>	varchar(30)	latin1_swedish_ci		Nein	kein(e)			Bearbeiten  Löschen  Mehr
4	<b>_alter</b>	int(11)			Nein	kein(e)			Bearbeiten  Löschen  Mehr
5	<b>geschlecht</b>	varchar(10)	latin1_swedish_ci		Nein	kein(e)			Bearbeiten  Löschen  Mehr
6	<b>adresse</b>	text	latin1_swedish_ci		Nein	kein(e)			Bearbeiten  Löschen  Mehr
7	<b>kontoid</b>	int(11)			Nein	kein(e)			Bearbeiten  Löschen  Mehr

Aktion	Schlüsselname	Typ	Unique	Gepackt	Spalte	Kardinalität	Kollation	Null	Kommentar
Bearbeiten  Löschen	<b>PRIMARY</b>	BTREE	Ja	Nein	id	3	A	Nein	

Die Struktur der Datenbank in „phpMyAdmin“

	id	vorname	nachname	alter	geschlecht	adresse	kontoid
<input type="checkbox"/> Bearbeiten	1	Hans	Dampf	42	Mann	12345 Irgendwo, Igendeinestrasse 42	1
<input type="checkbox"/> Bearbeiten	2	Otto	Schmitt	56	Mann	12345 Irgendwo, Igendeinestrasse 2	2
<input type="checkbox"/> Bearbeiten	3	Wilma	Feuerstein	32	Frau	44565 Steinzeit, Steinweg 1	3

Der Inhalt der Tabelle kunde in „phpMyAdmin“

## Die Abfragemethoden

Für den Zugriff auf die Ergebnisse einer Abfrage nutzt man in der Regel die Methoden `fetch()` oder `fetchAll()`. Die formale Syntax lautet so:

```
public array PDOStatement::fetchAll([ int $fetch_style [, mixed
$fetch_argument [, array $ctor_args = array() ]] ] )

public mixed PDOStatement::fetch([ int $fetch_style [, int
$cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset =
0 ]] ] )
```

Die `fetch`-Methoden von PDO

Die Methode `fetch()` liefert genau einen Datensatz und der interne Datensatzzeiger wird dabei um den Wert 1 inkrementiert. Bei `fetchAll()` erhält man dagegen alle Datensätze. Meist ist deshalb der Einsatz von `fetchAll()` bequemer, weil keine mehrfachen Aufrufe erforderlich sind, wenn man mehrere Datensätze benötigt.

Über den Parameter `$fetch_style` steuern Sie, in welcher Form beziehungsweise von welchem Typ die Methoden das Ergebnis zurückgeben. Die Methode erwartet hier eine der PDO-Klassenkonstanten, die mit `PDO::FETCH_` beginnen und in der Dokumentation nachzulesen sind. In vielen Fällen kann man aber beide Methoden in den Vorgabeeinstellungen verwenden.

Mit diesem kurzen Code kann man die Entität dann per PDO auslesen.

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '' );
} catch(PDOException $e) {
    die("Es ist ein Fehler aufgetreten!");
```

```

}
if ($stmt = $pdo->query( "SELECT * FROM kunde" )) {
    $data = $stmt->fetchAll();
    echo "<pre>", print_r( $data ),"</pre>";
}
?>

```

### Auslesen der Entität

```

Array
(
    [0] => Array
        (
            [id] => 1
            [0] => 1
            [vorname] => Hans
            [1] => Hans
            [nachname] => Dampf
            [2] => Dampf
            [_alter] => 42
            [3] => 42
            [geschlecht] => Mann
            [4] => Mann
            [adresse] => 12345 Irgendwo, Irgendeinestrasse 42
            [5] => 12345 Irgendwo, Irgendeinestrasse 42
            [kontoid] => 1
            [6] => 1
        )

    [1] => Array
        (
            [id] => 2
            [0] => 2
            [vorname] => Otto
            [1] => Otto
            [nachname] => Schmitt
            [2] => Schmitt
            [_alter] => 56
            [3] => 56
            [geschlecht] => Mann
            [4] => Mann
            [adresse] => 12345 Irgendwo, Irgendeinestrasse 2
            [5] => 12345 Irgendwo, Irgendeinestrasse 2
            [kontoid] => 2
            [6] => 2
        )

    [2] => Array
        (
            [id] => 3
            [0] => 3
            [vorname] => Wilma
            [1] => Wilma
            [nachname] => Feuerstein
            [2] => Feuerstein
            [_alter] => 32
            [3] => 32
            [geschlecht] => Frau
            [4] => Frau
            [adresse] => 44565 Steinzeit, Steinweg 1
            [5] => 44565 Steinzeit, Steinweg 1
            [kontoid] => 3
            [6] => 3
        )
    1
)

```

Die vollständige Struktur des Ergebnisobjekts der SQL-Abfrage

- ✓ Nach dem erfolgreichen Verbindungsaufbau wird mit der Methode `query()` das SQL-Kommando an das RDBMS gesendet.
- ✓ Die Methode `query()` liefert beim SELECT-Kommando entweder ein Objekt vom Typ `PDOStatement` (Erfolgsfall) oder im Fehlerfall `false`.
- ✓ Das Ergebnis wird in jedem Fall einer Variablen `$stmt` zugewiesen.
- ✓ Hat `$stmt` den Wert `false`, wird keine Auswertung vorgenommen.
- ✓ Im Erfolgsfall ist `$stmt` eine Instanz der Klasse `PDOStatement` und deren Methode `fetchAll()` ist sehr interessant und mächtig. Wie man bei der Ausgabe im Browser erkennen kann, liefert sie das Ergebnis der Abfrage als Array – und zwar in der Grundeinstellung jeweils doppelt. Besser gesagt, es ist ein zweidimensionales Array, denn jeder Datensatz wird in einem eigenen Array gespeichert. Es gibt also sowohl ein numerisches und als auch ein assoziatives Array, was eine sehr flexible Auswertung gestattet, denn man kann die Felder jedes Datensatzes über einen numerischen Index und über einen Namen (der der Spaltenbezeichnung in der Ergebnismenge entspricht) ansprechen.

Man kann `fetchAll()` mit Parametern auch so konfigurieren, dass nur ein assoziatives oder numerisches Array geliefert wird. Es gibt noch einige weitere Klassenkonstanten in PDO für die Verwendung in den Methoden `fetch()` und `fetchAll()`.

Eine Übersicht über alle Klassenkonstanten finden Sie im PHP-Handbuch unter <http://php.net/manual/de/pdo.constants.php>.

## Verbindung schließen

Die Verbindung wird durch das Objekt beendet, wenn dieses entweder am Skript-Ende oder mit der Funktion `unset()` zerstört wird. Auch das ist eine Vereinfachung gegenüber den alten MySQL- und MySQLi-Schnittstellen.

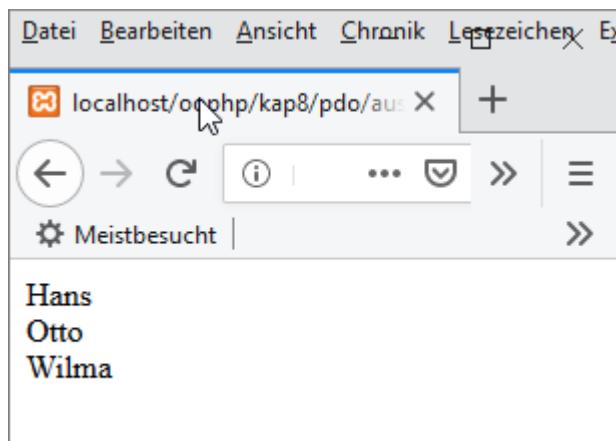
## Das Abfrageergebnis im Quelltext

Normalerweise werden Sie das Ergebnis nicht einfach mit `print_r()` ausgeben, sondern Sie werden es in angemessener Form präsentieren oder weiterverarbeiten: eine Ausgabe mehrerer Datensätze in einer Tabelle, einzelne Datensätze in einem Formular oder nur eine oder mehrere Spalten wie im folgenden Beispiel:

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', ''
    ) catch ( PDOException $e ) {
        die( "Es ist ein Fehler aufgetreten!" );
    }
if ($stmt = $pdo->query ( "SELECT * FROM kunde" )) {
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < sizeof($data); $i++) {
    echo $data[$i]['vorname'] . "<br />";
}
?>
```

*Gezielt Felder auswerten*

Das Ergebnisarray wird in der folgenden Schleife durchlaufen, und darin wird gezielt der Vorname aus jedem Datensatz angezeigt.



Anzeige der Vornamen aus der Entität

Nun sollte aber auffallen, dass die Anzahl der ermittelten Datensätze über eine „Funktion“ `sizeof()` bestimmt wird und das widerspricht einem objektorientierten Ansatz massiv. Man kann diskutieren, ob man solch eine geringfügige „Verschmutzung“ des objektorientierten Ansatzes akzeptieren kann oder ob man wirklich rein objektorientiert arbeiten muss. In dem Fall liefert PDO aber genügend Möglichkeiten, den Rückgriff auf Funktionen zu vermeiden.

## Informationen über das Ergebnis

Ein Objekt vom Typ `PDOStatement` stellt über Methoden und Eigenschaften eine Reihe interessanter Informationen zur Verfügung.

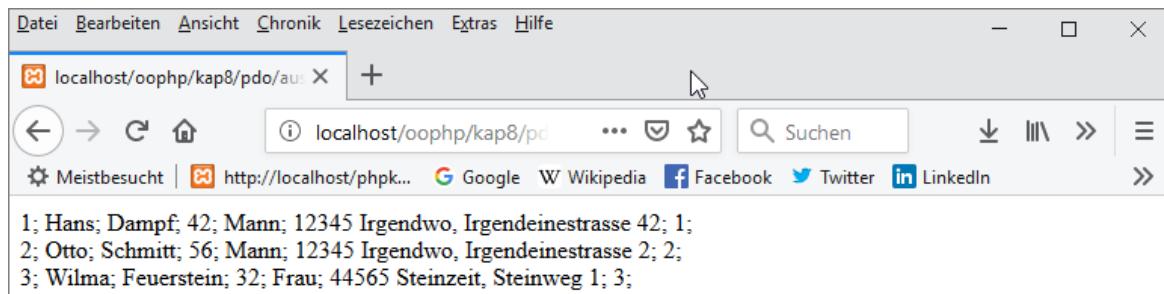
- ✓ Die Methode `rowCount()` liefert als Integer die Anzahl der durch die letzte Abfrage betroffenen Datensätze. Für SELECT-Anweisungen reagiert `rowCount()` allerdings datenbankabhängig. Nicht jedes RDMBS liefert die Anzahl der betroffenen Datensätze auf diese Weise zurück.
- ✓ Die Methode `columnCount()` liefert die Anzahl der Spalten in der Ergebnismenge.
- ✓ Die Methoden `fetch_field()` oder `fetch_fields()` liefern genauere Informationen zu Spalten. Die Methode `fetch_field()` liefert Ihnen nacheinander Informationen zu einer Spalte. Beim ersten Aufruf ist das die Spalte ganz links, beim zweiten die Spalte unmittelbar rechts daneben und so weiter, bis alle Spalten durchlaufen sind. Die Methode `fetch_field()` liefert ein Objekt mit der Spaltendefinition zurück oder `false`, wenn keine Informationen verfügbar sind.
- ✓ Das Attribut `field_count` enthält als Wert die Anzahl aller Spalten der Ergebnismenge.

Normalerweise werden Sie ein Ergebnis einer Abfrage in angemessener Form präsentieren oder weiterverarbeiten. So kann man etwa mehrere Datensätze in einer Tabelle oder einzelne Datensätze in einem Formular ausgeben. Sie können aber auch nur eine oder mehrere Spalten ausgeben, wie im folgenden Beispiel:

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '');
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
if ($stmt = $pdo->query ( "SELECT * FROM kunde" )) {
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++) {
    for($j = 0; $j < $stmt->columnCount(); $j++) {
        echo $data[$i][$j] . "; ";
    }
    echo "<br />";
}
?>
```

*Alle Felder anzeigen*

Das Ergebnisarray wird in einer verschachtelten Schleife durchlaufen und darin wird jeder Datensatz vollständig angezeigt.



*Anzeige aller Daten aus der Entität*

- ✓ Die äußere Schleife läuft über alle ermittelten Datensätze.
- ✓ In der inneren Schleife beginnt man mit der ersten Spalte und durchläuft die innere Schleife bis einschließlich der letzten Spalte.

## Daten in der Datenbank verändern

Das Ändern der Datenbank ist noch einfacher als das Abfragen. Sie brauchen bloß das SQL-Statement fehlerfrei zu formulieren und dann ggf. auf den Erfolgsfall oder den Fehler zu reagieren. Der Aufwand für die Auswertung eines Ergebnisses ist damit deutlich reduziert.

Das Ändern der Datenbank erfolgt etwa so wie im Beispiel, das in die Entität `konto` einen neuen Datensatz einfügt. Die Struktur entspricht der Klasse `Praxisprojekt` und enthält nur zusätzlich einen Primary Key, der aber automatisch hochgezählt wird.

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '');
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
if ($stmt = $pdo->query ( "INSERT INTO konto (kontostand,
kontotyp) VALUES ('4567', 'Girokonto')" )) {
    echo "Alles ok";
}
?>
```

*Einen Datensatz einfügen*

## 8.4 Prepared Statements

Wenn möglich, sollte man bei Datenbankzugriffen aus PHP mit sogenannten Prepared Statements arbeiten. Aber warum und was ist das?

### Nutzen von Prepared Statements

Ein Prepared Statement ist eine vorbereitete Anweisung für ein Datenbanksystem, in der es jedoch – im Gegensatz zu gewöhnlichen Statements – noch keine Parameterwerte gibt. Stattdessen werden Platzhalter angegeben, die zur Ausführungszeit gefüllt werden. Solche vorbereiteten Anweisungen haben einige gravierende Vorteile.

- ✓ Die Geschwindigkeit der Ausführung ist oft besser, denn bei wiederholtem Aufruf eines Prepared Statement kann das RDBMS diese puffern, nachdem die Abfrage geprüft wurde. Eine weitere Prüfung der Anweisung selbst kann bei erneutem Aufruf entfallen und es müssen nur noch die variablen Bestandteile (die Werte der Variablen für die Platzhalter) müssen an das RDBMS übertragen werden.
- ✓ Noch wichtiger ist die Sicherheit. Eine große Gefahr geht in diesem Zusammenhang von den sogenannten **SQL-Injections** aus. Diese beschreiben das Injizieren eines schädlichen SQL-Kommandos in die Datenbank. Prepared Statements schützen jedoch vor schädlichen SQL-Anweisungen, da hier bereits das geprüfte Template gepuffert vorliegt und nur ganz bestimmte variable Bestandteile zulässt. Sollte also ein SQL-Kommando eingeschleust werden, das nicht dem Prepared Statement entspricht, fällt das sofort auf und es wird nicht ausgeführt.

### Prepared Statements in PDO

Die Basis zum Umgang mit Prepared Statements in PDO bilden die folgenden Methoden:

- ✓ Die Methode `prepare()` aus der Klasse `PDO`. Damit bereitet man die SQL-Anweisung vor.
- ✓ Die Methode `bindParam()` aus der Klasse `PDOStatement` bindet die variablen Parameter an die konkreten Werte.
- ✓ Die Methode `execute()` aus der Klasse `PDOStatement` führt die Anfrage aus.

Wir sehen uns die Verwendung von vorbereiteten Anweisungen jetzt an einem ersten Beispiel an. Dieses ist nur eine kleine Modifizierung des Beispiels mit der Ausgabe der Daten aus der Entität kunde.

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '');
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
if ( $stmt = $pdo->prepare ( "SELECT * FROM kunde" ) ) {
    $stmt->execute ();
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++) {
    for($j = 0; $j < $stmt->columnCount(); $j++) {
        echo $data[$i][$j] . "; ";
    }
    echo "<br />";
}
?>
```

*Alle Felder mit Prepared Statements anzeigen*

Bei dieser Anwendung von Prepared Statements wird aber noch nicht deutlich, was der Vorteil ist. Erweitert man jedoch die Auswahlanweisung um eine **where**-Bedingung, wird die Datenbindung sichtbar. Dazu soll ein Zwischenschritt gemacht werden.

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '');
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
if ( $stmt = $pdo->prepare ( "SELECT * FROM kunde WHERE id=2" ) ) {
    $stmt->execute ();
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++) {
    for($j = 0; $j < $stmt->columnCount(); $j++) {
        echo $data[$i][$j] . "; ";
    }
    echo "<br />";
}
?>
```

*Nur einen Datensatz anzeigen*

Bei dem Beispiel wird nur der Datensatz angezeigt, dessen Primary Key den Wert 2 hat.

Aber auch jetzt nutzen wir noch keine Datenbindung. Das geht so:

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '');
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
$id = 2;
if ( $stmt = $pdo->prepare ( "SELECT * FROM kunde WHERE id=:id" ) )
{
    $stmt->bindParam ( ':id', $id );
    $stmt->execute ();
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++) {
    for($j = 0; $j < $stmt->columnCount(); $j++) {
        echo $data[$i][$j] . "; ";
    }
    echo "<br />";
}
?>
```

*Nur einen Datensatz anzeigen – mit Datenbindung*

- ✓ In dem Beispiel wird eine Variable `$id` mit dem Wert 2 initialisiert. Das ist der Wert des gewünschten Primary Keys, nach dem wir in diesem Beispiel selektieren. Der Wert könnte selbstverständlich auch aus einem Formular kommen.
- ✓ In der WHERE-Klausel wird anstelle eines konkreten Wertes ein sogenannter **benannter Platzhalter** `:id` eingesetzt. Der Doppelpunkt kennzeichnet den Platzhalter als solchen. Den Namen können Sie frei wählen. Allerdings sollten Sie einen möglichst sprechenden Namen benutzen. In unserem Beispiel entspricht der Name des Platzhalters der Bezeichnung des korrespondierenden Feldes in der Datenbanktabelle. Dadurch wird der Zusammenhang sofort deutlich.
- ✓ Der Platzhalter wird später durch den Wert einer Variablen ersetzt. Welche Variable das ist, legen wir in diesem Beispiel mit der Methode `bindParam()` fest. Man spricht in diesem Zusammenhang von **Binden** oder **Parameterbindung**.

Wollen Sie mehrere Platzhalter verwenden, notieren Sie diese im SQL-Statement mit vorangestelltem Doppelpunkt und notieren dann jeweils eine `bindParam()`-Methode dafür.

Neben den benannten Platzhaltern gibt es auch **unbenannte** (positionsabhängige) Platzhalter.

- ✓ Repräsentiert werden diese durch das Fragezeichen im SQL-Statement.
- ✓ Als ersten Parameter der Methode `bindParam()` müssen Sie die Position des Platzhalters im SQL-Statement angeben, wobei diese nicht (!) nullindiziert ist. Die erste Position eines Platzhalters hat also den Wert 1.

```
<?php
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '');
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
$id = 2;
if ( $stmt = $pdo->prepare ( "SELECT * FROM kunde WHERE id=?") )
{
    $stmt->bindParam ( 1, $id );
    $stmt->execute ();
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++) {
    for($j = 0; $j < $stmt->columnCount(); $j++) {
        echo $data[$i][$j] . "; ";
    }
    echo "<br />";
}
?>
```

*Nur einen Datensatz anzeigen – mit unbenannten Parametern*

## 8.5 Das Praxisprojekt mit Datenbanken

Aufbauend auf dem letzten Stand des Praxisprojekts soll jetzt der Kontostand nach einer Auszahlung zu einer Änderung in der Datenbank führen und auch der aktuelle Kontostand und der Kontotyp sollen aus der Datenbank ausgelesen werden. Bisher wird der aktuelle Kontostand noch über die Session verwaltet, was ja explizit nicht dauerhaft ist. Ebenso die Daten des Kunden aus der Datenbank eingelesen werden.

Es sollen in der Modifizierung für den Datenbankzugriff Prepared Statements verwendet werden. Dabei muss nur die Programmdatei angepasst werden. Das API bleibt unverändert und es werden sogar diverse Schritte vereinfacht, wenn man den Kontostand in einer Datenbank speichern kann.

Hier ist zuerst das vollständige Listing (v1), das danach im Detail durchgegangen werden soll:

```
<?php
session_start();
require_once("Kunde.class.php");
require_once("Mitarbeiter.class.php");

class Program {
    public function __construct() {
        $this -> main();
    }

    private function main() {
        require_once("lib/res/kopf.txt");
        require_once("lib/res/formular.txt");
```

```

// Auswahl Kunde
$kode = array();
try {
    $pdo = new PDO (
        'mysql:dbname=oophp;host=localhost;charset=utf8',
        'root', '' );
} catch ( PDOException $e ) {
    die ( "Es ist ein Fehler aufgetreten!" );
}
$id = 2; // Kunde mit der Id 2 soll genommen werden
if ($stmt = $pdo->prepare ( "SELECT * FROM kunde WHERE id=:id" )) {
    $stmt->bindParam ( ':id', $id );
    $stmt->execute ();
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++){
    for($j = 0; $j < $stmt->columnCount(); $j++) {
        $kode[$j] = $data[$i][$j];
    }
}
// Abfrage Kontostand
// Der Foreign Key in der Entität kunde - das zugeordnete Konto
$id= $kode[6];
if ($stmt = $pdo->prepare ( "SELECT * FROM Konto WHERE id=:id" )) {
    $stmt->bindParam ( ':id', $id );
    $stmt->execute ();
    $data = $stmt->fetchAll ();
}
for($i = 0; $i < $stmt->rowCount(); $i++){
    $kontostand = $data[$i]['kontostand'];
    $kontotyp = $data[$i]['kontotyp'];
}
$kontol = new Privatkonto(floatval($kontostand), $kontotyp);
$kundel = new Kunde($kode[1], $kode[2], intval($kode[3]),
$kode[4], $kode[5], $kontol);

if( isset($_POST['betrag']) ) {
    $meldung = $kontol-> auszahlen(floatval($_POST['betrag']));
    // Den Kontostand aktualisieren
    $kontostand = $kontol->getKontostand();
    $sql = "UPDATE konto SET kontostand = :kontostand WHERE id =
:id";
    if ($stmt = $pdo->prepare ($sql)) {
        $stmt->bindParam ( ':id', $id );
        $stmt->bindParam ( ':kontostand', $kontostand );
        $stmt->execute ();
    }
} else {
    $meldung = "Der Kontostand beträgt aktuell ".
    $kontol->getKontostand() . " EUR.";
}
echo "<div id='info'>$meldung </div>";
require_once("lib/res/ende.txt");
}
new Program();
?>

```

Den Kontostand und Kunden in der Datenbank verwalten

Bei dem Listing gibt es ein paar bemerkenswerte Stellen:

- ✓ Mit `$kunde = array();` wird ein Array angelegt, das nach dem Auslesen aus der Datenbank die Eigenschaften eines Kunden beinhalten soll.
- ✓ Mit der Variablen Deklaration `$id = 2;` wird der Kunde mit der Id 2 als Kandidat ausgewählt.
- ✓ Konkret aus der Datenbank wird der Kunde mit dem Prepared Statement ermittelt, das mit der `where`-Bedingung diese Id auswählt (`if ($stmt = $pdo->prepare ("SELECT * FROM kunde WHERE id=:id" ))`).
- ✓ Mit `$data = $stmt->fetchAll();` und der nachfolgenden verschachtelten Schleife werden im Grunde alle Datensätze selektiert, die von der SQL-Anweisung ermittelt werden. Aber da die Angabe der Id in Verbindung mit der Struktur der Entität (Primary Key) dafür sorgt, dass nur **genau ein** Datensatz als Resultat ermittelt wird, kann man dennoch diese im Grunde „oversized“ angelegte verschachtelte Schleife unverändert verwenden.
- ✓ Im Inneren der inneren Schleife wird das Kundenarray mit `$kunde[$j] = $data[$i][$j];` aufgebaut. Danach sind alle Eigenschaften des Kunden so gefüllt, wie es für den Konstruktor der Klasse Kunde notwendig ist.
- ✓ Mit der Anweisung `$id= $kunde[6];` erhält man aus dem Datensatz des Kunden auch als Foreign Key auf die Entität `konto` das zugeordnete Konto.
- ✓ Mit diesem Foreign Key kann man eine `select`-Anweisung erstellen, die das passende Konto aus der Datenbank holt (`if ($stmt = $pdo->prepare ("SELECT * FROM Konto WHERE id=:id" ))`).
- ✓ Da auch hier **dann genau ein** Datensatz ermittelt wird, kann man in der folgenden Schleife mit `$kontostand = $data[$i]['kontostand'];` und `$kontotyp = $data[$i]['kontotyp'];` den aktuellen Kontostand als auch den Kontotyp ermitteln.
- ✓ Mit diesen Daten können nun wie bisher die Objekte vom Typ `Privatkonto` und `Kunde` erstellt werden.
- ✓ Sofern nun eine Auszahlung erfolgt, muss danach der Kontostand in der Datenbank noch aktualisiert werden. Das erfolgt mit einer `Update`-Anweisung, die in einer Variablen gespeichert wird (`$sql = "UPDATE konto SET kontostand = :kontostand WHERE id = :id";`).
- ✓ Die beiden benannten Parameter werden dabei mit `$stmt->bindParam (':id', $id);` und `$stmt->bindParam (':kontostand', $kontostand);` gebunden.

## 8.6 Fazit und Abschlussbemerkungen

Sie haben nun einen umfangreichen Einstieg in die objektorientierte Programmierung mit PHP erhalten, der mit einem Projekt zum objektorientierten Datenbankzugriff seinen Abschluss gefunden hat. Obwohl PHP ursprünglich nicht objektorientiert konzipiert wurde, bieten die neuen Versionen mittlerweile alle relevanten OO-Techniken an und Sie können damit Ihre Web-Projekte komplett objektorientiert gestalten. Aber Sie können durchaus auch den objektorientierten Ansatz mit der klassischen, prozeduralen Vorgehensweise mischen. PHP lässt Ihnen die Wahl.

## 8.7 Übung

**Übungsdateien:** Verzeichnis v1

**Ergebnisdatei:** Anlegen.class.php

1. Für das letzte Beispiel soll eine neue Klasse Anlegen erstellt werden, die eine vollständige Webseite objektorientiert erstellt und dabei ein Webformular bereitstellt, über das alle relevanten Daten für einen Kunden und ein Konto eingegeben werden können.
2. Das Formular wird nicht plausibilisiert und/oder gegen Fehleingaben geschützt. Es wird vorausgesetzt, dass ein Anwender alle Felder korrekt ausfüllt.
3. Die Struktur der Klasse soll der Struktur der Programmklasse entsprechen. Es soll die gleiche CSS-Datei verwendet werden und der Kopfbereich und das Ende können als einzubindende Textdateien übernommen werden.
4. Das Formular soll ebenso als externe Textressource erstellt und wie das entsprechende Formular in der bisherigen Programmklasse eingebunden werden.

The screenshot shows a web browser window with the following details:

- Header:** Datei, Bearbeiten, Ansicht, Chronik, Lesezeichen, Extras, Hilfe.
- Title Bar:** Auszahlung Konto.
- Address Bar:** localhost/ooph/Ergebnisda
- Toolbar:** Back, Forward, Stop, Refresh, Search, Home, etc.
- Links:** Meistbesucht, http://localhost/phpk..., Google, Wikipedia, Facebook, Twitter, LinkedIn, Wetter Online, Meistbesucht.
- Content Area:**

**Kunde und Konto anlegen**

Vorname	Willi
Nachname	Wüst
Alter	31
Geschlecht	Mann
Adresse	44567 Nowhere, Hippstergasse 77
Kontotyp	Sparbuch
Kontostand	5678.78

Daten absenden

Anlegen neuer Daten.

So ungefähr soll die Webseite mit dem Formular aussehen.

5. Die eingegebenen Daten sollen mit Prepared Statements und benannten Parametern in die Datenbank geschrieben werden.

6. Die eingegebenen Daten werden zwar in einem Formular erfasst, verteilen sich aber auf zwei Entitäten. Es sind also zwei SQL-Anweisungen notwendig, die nacheinander ausgeführt werden.
7. Da in dem Datensatz eines Kunden ein Foreign Key des verknüpften Kontos gespeichert werden muss, muss dieser vorher bestimmt werden. Dazu kann man die Tatsache nutzen, dass die Primary Keys der Entitäten automatisch hochgezählt werden. Damit muss die Id des neu angelegten Kontos um den Wert 1 höher sein als die des letzten vorhandenen Kontos (sortiert nach der Id, was automatisch der Fall ist). Wenn man ein select-Statement über die Entität `konto` ausführt, erhält man den Wert der letzten Id und muss den Wert nur um den Wert 1 erhöhen und erhält damit den zu verwendenden Wert des Foreign Keys für die Entität `kunde`.
8. Geben Sie nach dem Einfügen der Daten eine entsprechende Meldung aus.
9. Kontrollieren Sie den Erfolg mit `phpMyAdmin` oder einem anderen MySQL-Clienten.

The screenshot shows a web browser window with the title "Auszahlung Konto". The address bar displays "localhost/oophp/Ergebnisda". The main content is a form titled "Kunde und Konto anlegen". The form fields are:

- Vorname (input field)
- Nachname (input field)
- Alter (input field)
- Geschlecht (input field)
- Adresse (input field)
- Kontotyp (input field)
- Kontostand (input field)

Below the form is a button labeled "Daten absenden". A message at the bottom of the form area says "Neue Daten wurden angelegt." A cursor arrow is visible at the bottom right of the form area.

*Neue Daten wurden in die Datenbank eingefügt.*

# A

## Anhang: Installationen und Quellangaben

### A.1 Programme und Tools

Für die Arbeit mit PHP werden in der Unterlage und zur Unterstützung der Programmierung mehrere Programme und Tools verwendet. In diesem Anhang finden Sie dazu Informationen zum Bezug und zur Installation.

#### Download und Installation von XAMPP

Um PHP-Skripte testen zu können, benötigen Sie einen PHP-Interpreter, der möglichst mit einem Webserver verknüpft ist. XAMPP ist ein Projekt von Apache Friends. Ziel des Projekts ist die zu einem Software-Bundle mit Namen XAMPP verknüpften Programme Apache-Webserver, MySQL bzw. MariaDB, PHP und Perl sowie einige weiteren Dienste wie FileZilla (FTP-Server), Tomcat (Open-Source-Webserver und Webcontainer, der die Ausführung von Java Servlets und JavaServer Pages – JSP – ermöglicht) implementiert und Mecury (Mailserver) möglichst einfach und schnell lokal zu installieren und zu konfigurieren. Das X steht für ein beliebiges Betriebssystem. Zur Verwaltung des MySQL/MariaDB-Systems wird auch *phpMyAdmin* installiert.

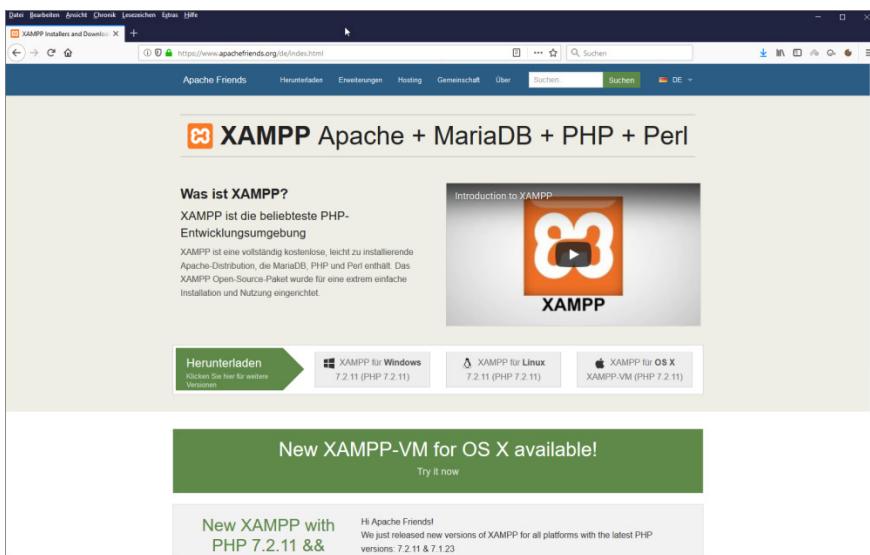
XAMPP existiert in verschiedenen Varianten, beispielsweise für folgende Betriebssysteme:

- ✓ Windows,
- ✓ Linux-Systeme und
- ✓ Mac OS X.

In dieser Unterlage wird zum Testen der PHP-Skripte XAMPP in einer Windows-Umgebung verwendet. Daher wird auch für die Installation und Konfiguration Windows beispielhaft herangezogen, wobei diese Schritte für Linux und Mac OS X ähnlich laufen.

## Download von XAMPP

- XAMPP 1.8.1 (die konkrete Basis dieser Unterlage) oder eine andere Version können Sie sich kostenlos von der Website des Projekts *Apache Friends* (<http://www.apachefriends.org>) herunterladen. Hier findet man auch die wichtigsten Informationen zu Apache, PHP und MySQL/MariaDB sowie andere notwendige oder ergänzende Ressourcen. Über den Menüpunkt *Herunterladen* erhalten Sie die aktuellsten, aber auch frühere Versionen für verschiedene Betriebssysteme. Laden Sie sich am besten immer die neueste Version von XAMPP für Ihr Betriebssystem herunter.

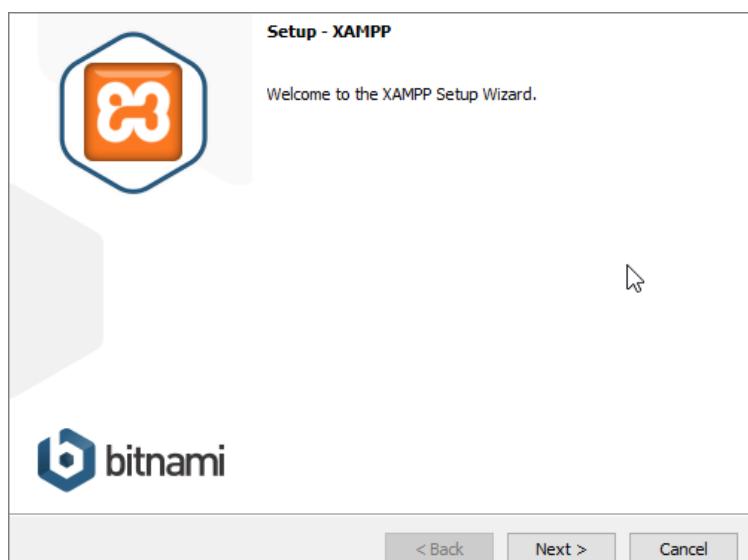


### Downloadseite für XAMPP

- Um XAMPP zu installieren, folgen Sie danach den Schritten des heruntergeladenen Installationsassistenten. Durch den Installer bzw. Wizard ist diese Installation in der Regel recht einfach.

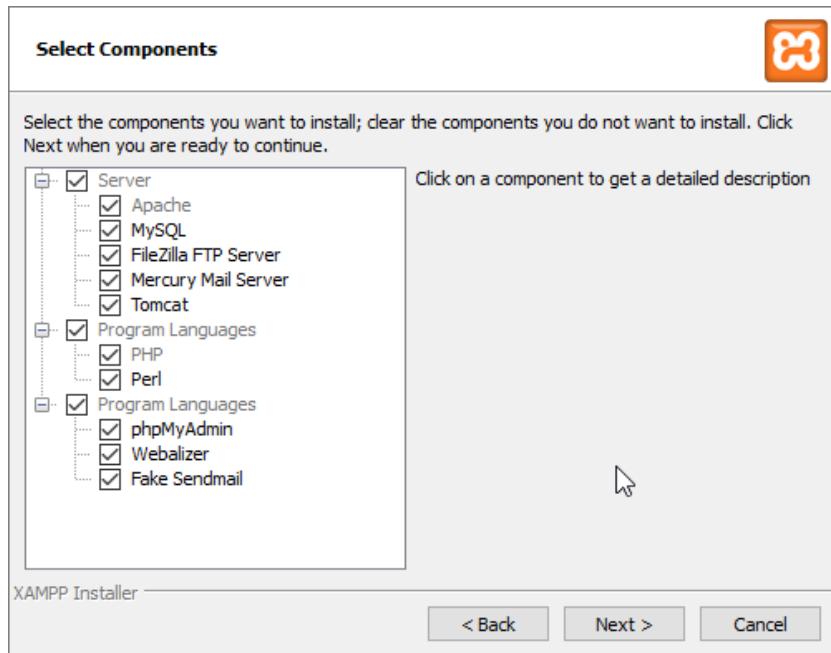
## Die Installation unter Windows

Unter Windows starten Sie einfach den Installer, den Sie auf Ihren PC geladen haben.



### Der Start der Installation

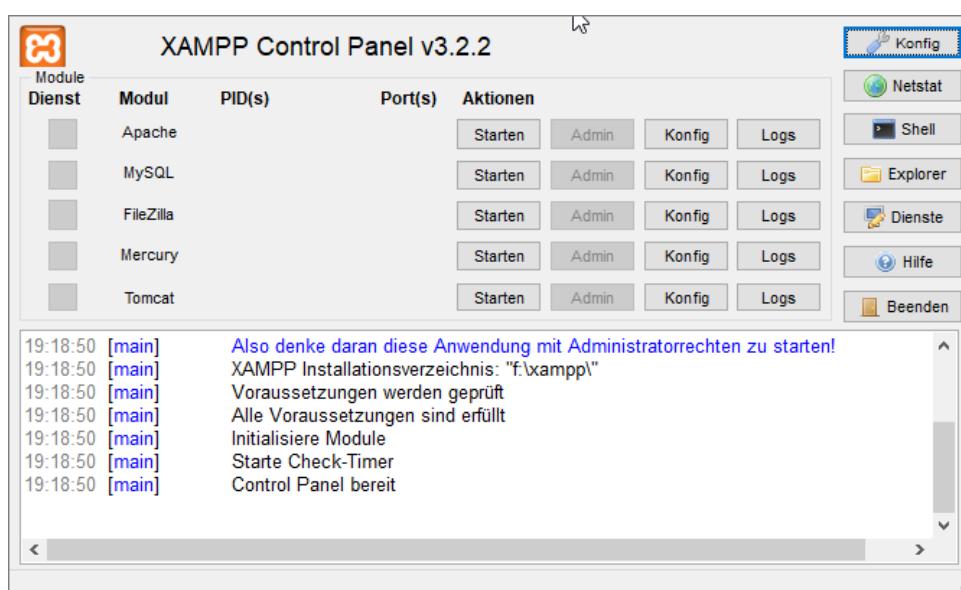
Dieser Installer öffnet wiederum einen Auswahldialog. Wenn Sie wollen, passen Sie von da aus die Pfade und Einstellungen sowie zu installierende Bestandteile an. Die Einstellungen können aber meist in der Voreinstellung bleiben.



*Bei Bedarf die Installation anpassen*

Nach dem Beenden der Installation steht Ihnen ein Control Panel zur Verfügung, das es auch bei den anderen Plattformen gibt. Darüber können Sie die einzelnen Serverdienste (also Apache, MySQL/MariaDB, FileZilla, Mecury und Tomcat) manuell starten und beenden oder auch konfigurieren und bei Bedarf automatisch starten lassen.

Für die Arbeit mit diesem Buch werden nur Apache und MySQL/MariaDB benötigt und das manuelle Starten der Server über das Control Panel genügt.



*Das XAMPP Control Panel*

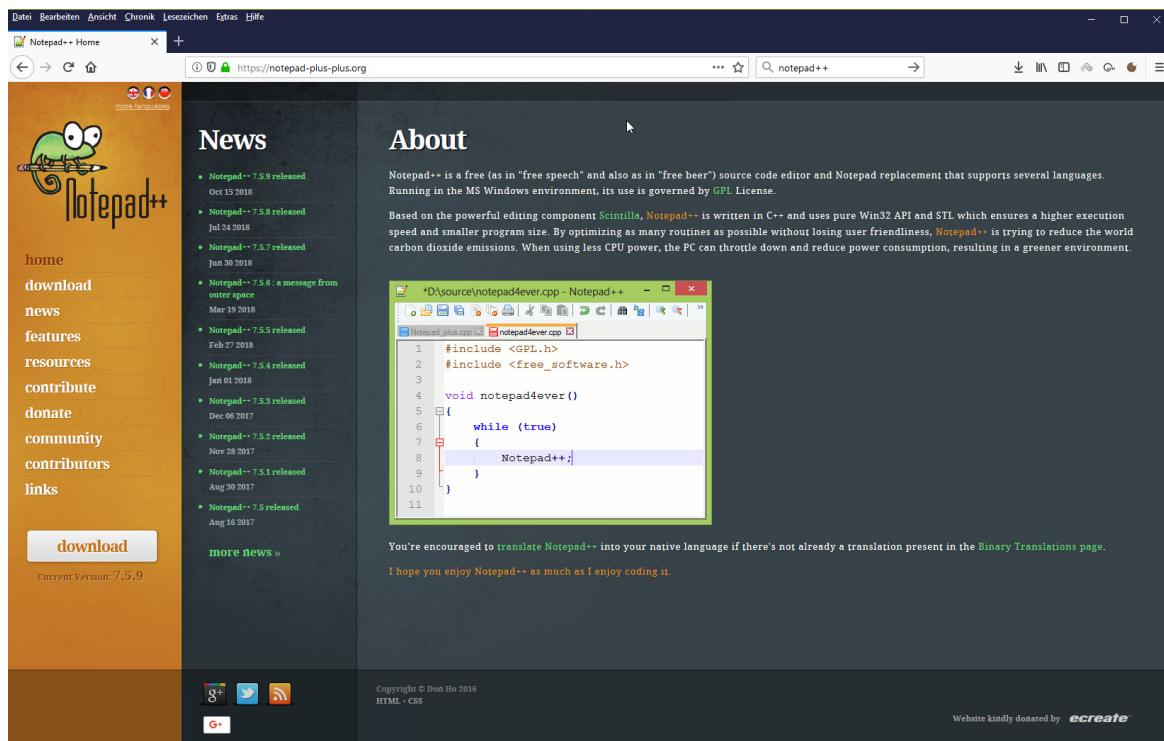
Wenn der Apache-Webserver gestartet wurde, steht dieser in einem Browser auf Ihrem Computer über die Adresse <http://localhost> oder alternativ über die IP-Adresse <http://127.0.0.1> (die IP-Adresse von localhost) zur Verfügung.

Bei erfolgreicher Installation und gestartetem Webserver sehen Sie die XAMPP-Startseite. Beim ersten Start der Seite erscheint eine Abfrage, welche Sprache in Zukunft verwendet werden soll. Diese Sprachauswahl erscheint nur beim ersten Aufruf von XAMPP. Bei weiteren Aufrufen werden Sie automatisch auf die eigentliche Startseite geleitet, von wo aus zu den verschiedenen Möglichkeiten von XAMPP navigiert werden kann.

## Download und Installation von Notepad++

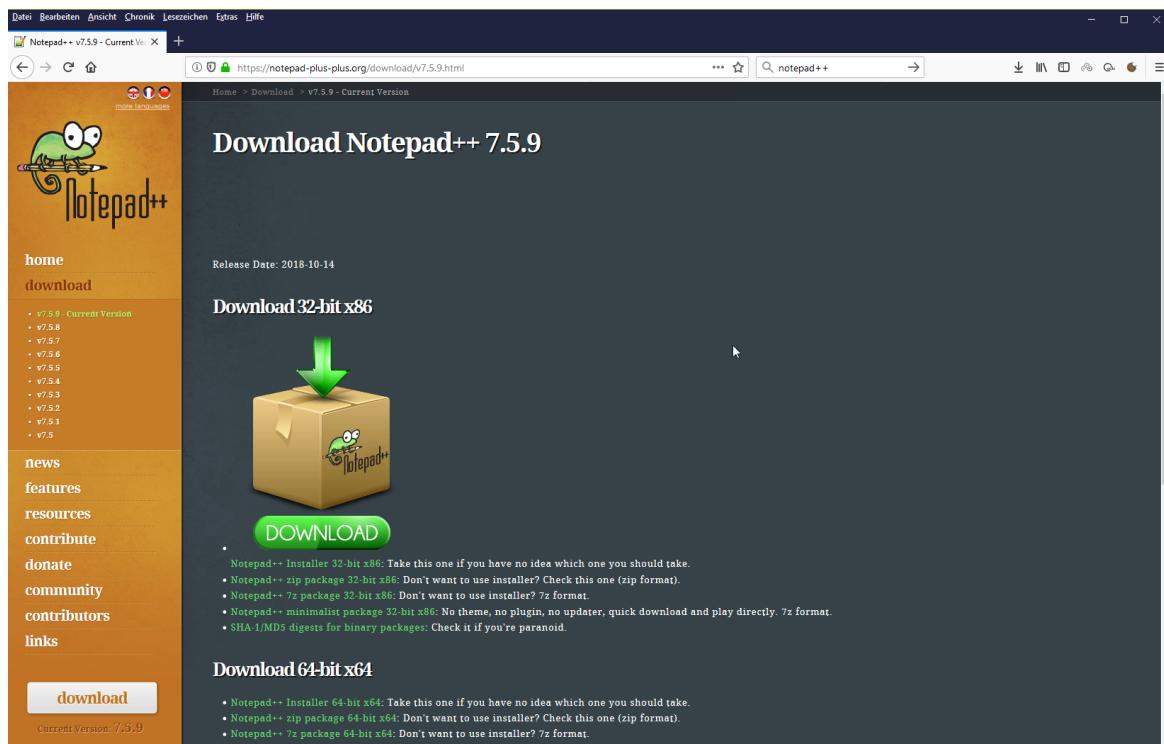
Wenn man PHP-Skripte erstellen möchte, braucht man nur einen reinen Editor. Aber das Programmieren damit ist unbequem und fehlerträchtig. In der Praxis verwenden deshalb fast alle professionellen Programmierer mittlerweile Programmierwerkzeuge, die sie bei der Erstellung und Analyse des Quelltextes unterstützen. Solche Programme kennen einige Bestandteile einer Programmier- oder Beschreibungssprache und unterstützen einfache und teilweise auch komplexere Standardvorgänge: z. B. das Maskieren (die kodierte Darstellung) von Sonderzeichen, das Einfügen von Quellcodeschablonen oder die Hilfestellung für die Übersichtlichkeit durch farbliche Kennzeichnung von bekannten Befehlen. Einige Editoren bieten einem Programmierer die Befehle einer verwendeten Sprache auch direkt an – etwa durch Menüs oder Symbolleisten, in denen der Anwender diese auswählen kann (auch mit der Maus).

Der Windows-Editor Notepad++ (<http://notepad-plus.sourceforge.net/de/site.htm>) bietet zum Beispiel so eine Unterstützung für verschiedene Sprachen und kann sogar um viele Technologien erweitert werden. Auf der Webseite des Projekts finden Sie dazu die wichtigsten Informationen.



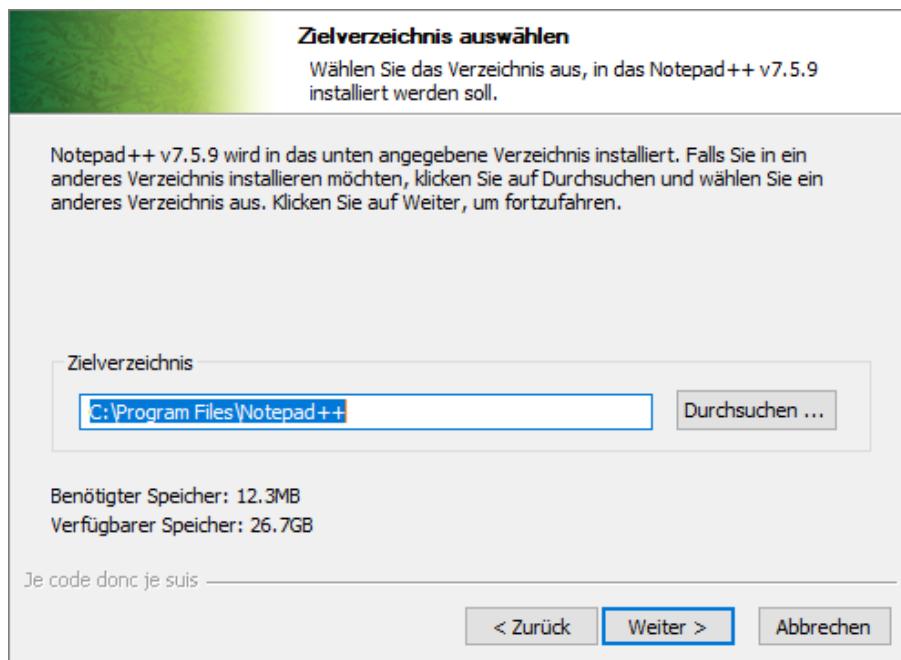
Die Webseite des Notepad++-Projekt

Über den *download*-Link kommen Sie zur Auswahl der gewünschten Version.



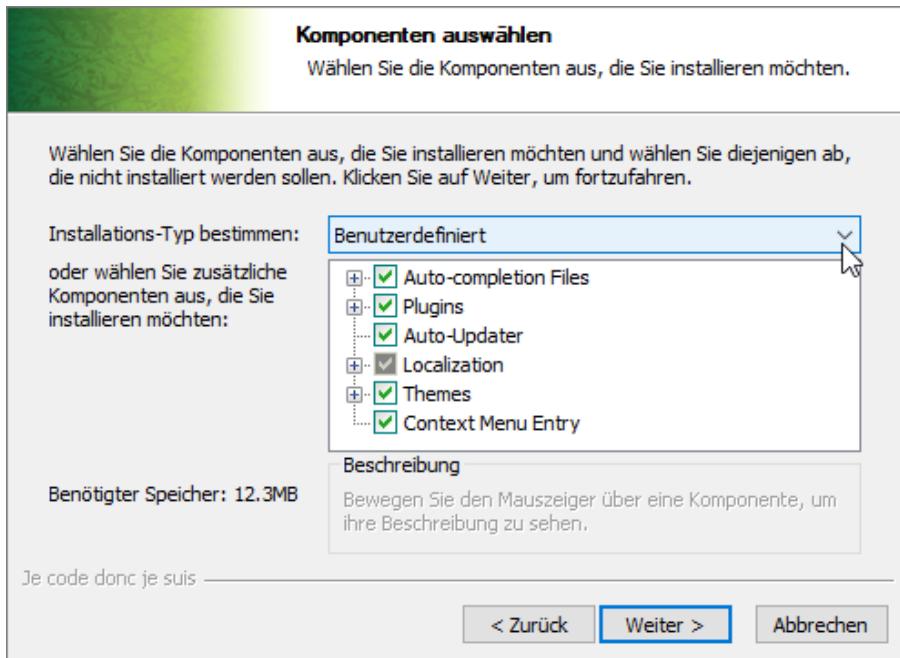
*Auswahl der gewünschten Version*

Die Installation ist weitgehend selbsterklärend und erfordert im Wesentlichen nur die Angabe des Verzeichnisses, wohin das Programm installiert werden soll sowie welche Erweiterungen zu installieren sind.



*Angabe des Zielverzeichnisses*

Bei der Auswahl der Komponenten bleiben Sie am besten bei den Grundeinstellungen.



#### Auswahl der Komponenten

Für Windows, aber auch Linux und MacOS, ist Brackets (<http://brackets.io/>) als Alternative eine sehr gute Wahl.

## Download und Installation von FileZilla

Wenn man von FileZilla spricht, muss man den FileZilla FTP-Client und den FTP-Server unterscheiden. Der FTP-Server wird – wenn Sie das wünschen – mit XAMPP bereits installiert. Aber den FTP-Client müssen Sie ggf. selbst installieren.

The screenshot shows the main page of the FileZilla project website. On the left is a sidebar with links for Home, FileZilla (selected), Features, Screenshots, Download, Documentation, FileZilla Pro, FileZilla Server, General, Development, and Other projects. The main content area has a header "Promotion: FileZilla Pro S3, Google Drive - Cloud, OneDrive, Azure, Dropbox, WebDAV GET IT NOW". Below it is the "Overview" section, which includes a welcome message, a note about the GNU General Public License, and a link to the "FileZilla Pro" page. It also features a "Quick download links" section with two large buttons: "Download FileZilla Client All platforms" and "Download FileZilla Server Windows only". A note below the buttons says: "Pick the client if you want to transfer files. Get the server if you want to make files available for others." There is also a "News" section with links to recent releases (e.g., 2018-10-27 - FileZilla Client 3.38.1 released) and a "New features" section.

### Die Webseite von FileZilla

Auf der Webseite des FileZilla-Projekts (<https://filezilla-project.org/>) erhalten Sie neben dem FTP-Server auch den FTP-Client für verschiedene Betriebssysteme.

This screenshot shows the same FileZilla homepage as above, but with a different URL in the address bar: "Download FileZilla Client for Windows (64bit)". The main content area now features a large green button labeled "Download FileZilla Client" with a "Windows (64bit)" icon. To the right of the button is a screenshot of the FileZilla client interface. Below the button, there is a note about bundled offers and a "More download options" section with links for other platforms like Mac and Linux. At the bottom of the page, a URL "https://download.filezilla-project.org/client/FileZilla\_3.38.1\_win64-setup\_bundled.exe" is provided.

### Download vom FileZilla-FTP-Client

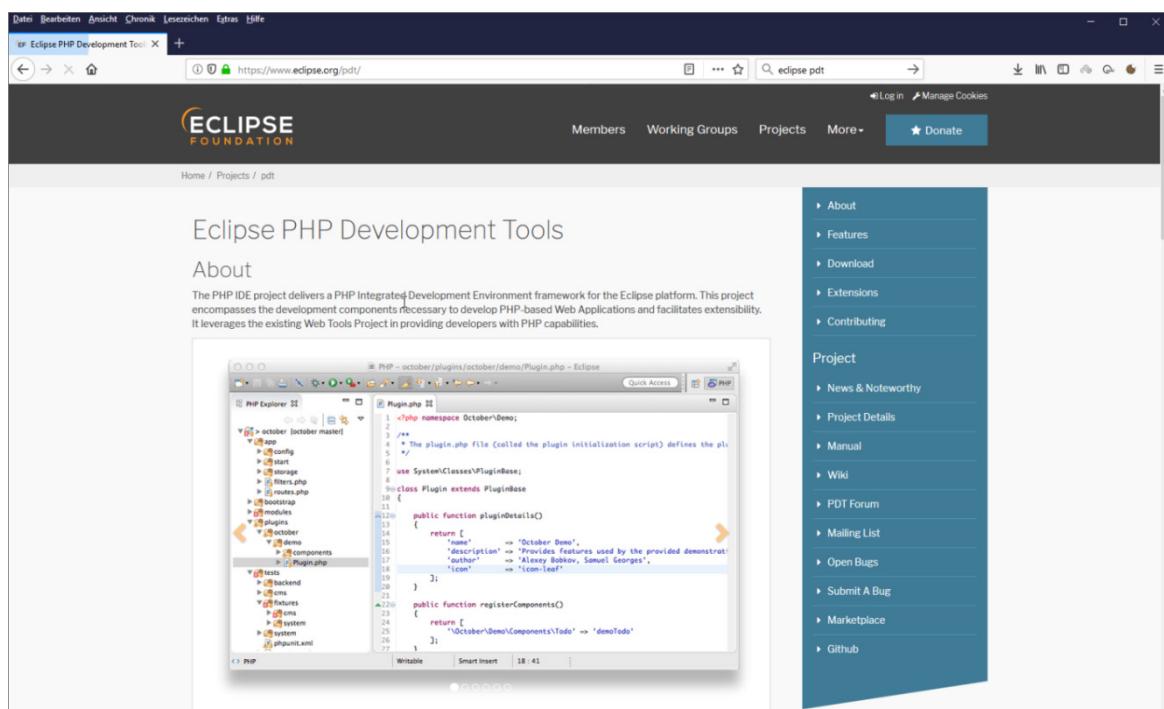
Die Installation erfolgt wieder über einen einfachen Assistenten.

## Download und Installation von Eclipse mit den PDT

Obwohl man PHP-Skripte mit einem einfachen Editor programmieren kann, ist ein unterstützender Editor wie NotePad++ sinnvoll und hilfreich. Es gibt jedoch auch Tools, die noch weit über die Leistungen eines unterstützenden Editors wie Notepad++ hinausgehen. Bei solchen Tools handelt es sich dann meist um so genannte IDEs (Integrated Development Environment), die seit einigen Jahren auch in der Web-Programmierung zur Verfügung stehen. Diese gestatten die Programmierung inkl. Ausführung der Skripte und Programme sowie diverse weitere Vorgänge rund um die Programmierung aus einer integrierten, gemeinsamen Oberfläche.

Eine kostenlose und dennoch sehr mächtige IDE nennt sich Eclipse (<http://www.eclipse.org>). Dieses Programm wurde ursprünglich als Java-IDE konzipiert. Mittlerweile kann die IDE aber über Plugins für eine Vielzahl an Sprachen und Technologien erweitert werden.

Für PHP gibt es unter anderem ein Plugin, das auch die neuesten Versionen von PHP unterstützt – die PHP Development Tools oder kurz PDT (<https://eclipse.org/pdt/>). Das PDT-Projekt kann sowohl als zusätzliches Plugin in eine bestehende Eclipse integriert oder bereits als vorgefertigtes Bundle zusammen mit Eclipse geladen werden. In beiden Fällen liefert ein PHP Integrated Development Environment Framework für die Eclipse-Plattform alle Entwicklungskomponenten, die notwendig sind, um PHP-basierte Web-Applikationen zu entwickeln, aus der integrierten Umgebung auszuführen, zu untersuchen und die Erweiterbarkeit zu erleichtern.



*Download der PDT als Bundle mit einem Eclipse*

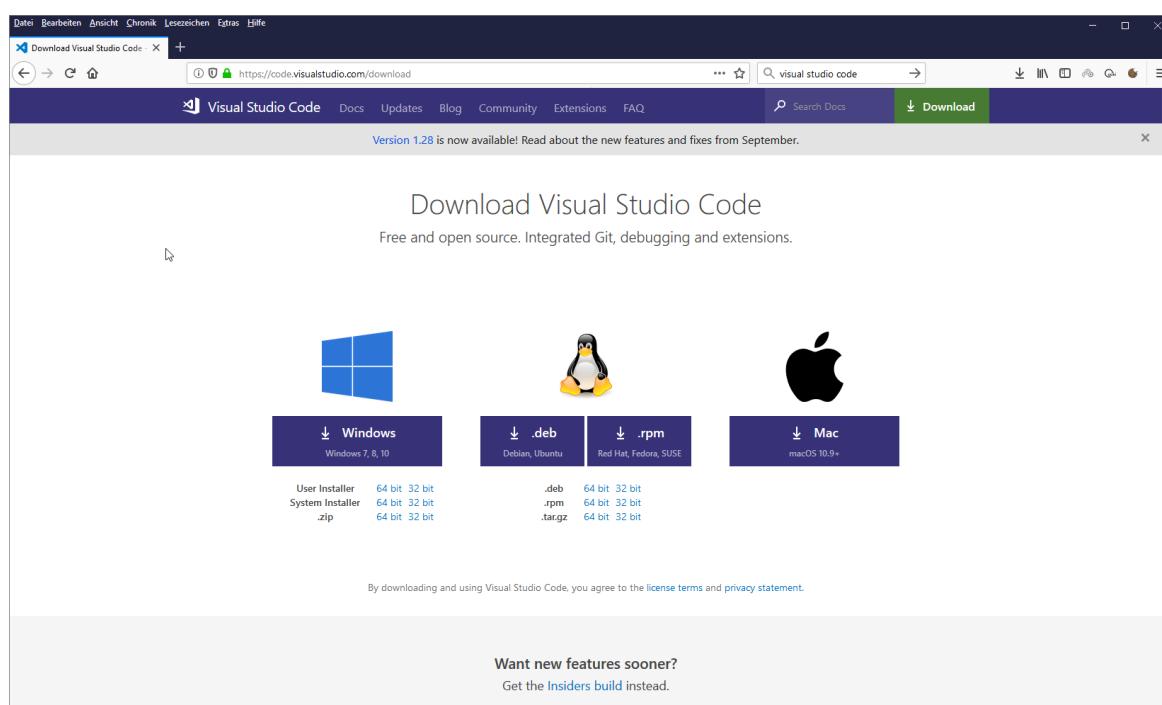
Die eigentliche Installation läuft mit einem Assistenten wie üblich sehr einfach ab.

## Download und Installation von Visual Studio und Visual Studio Code

Neben der vollständigen IDE Visual Studio (<https://visualstudio.microsoft.com/de/>) gibt es von Microsoft auch den schlankerem Editor Visual Studio Code (<https://code.visualstudio.com/>). Beide Entwicklungstools eignen sich hervorragend zur Programmierung. Gerade wenn Sie aus dem Bereich von .NET kommen.

Das Visual Studio, das es auch in einer kostenlosen Community-Version gibt, stellt bereits in der Grundinstallation diverse Editoren und Vorlagen zur Verfügung und die hervorragende Codeunterstützung greift auch bei Web-Technologien. Mit sogenannten Extensions wie den Web Essentials kann man Visual Studio bei Bedarf auch erweitern.

Allerdings gehen die Möglichkeiten von Visual Studio oft weit über das hinaus, was man zur Erstellung von PHP benötigt. Der viel schlankere Code-Editor verzichtet auf viel Overhead und spezialisiert sich auf Entwicklung von modernen, plattformübergreifenden Web- und Cloud-Anwendungen. Unterstützt werden HTML, CSS, JSON, LESS, SASS, PHP oder JavaScript samt Syntax-Highlighting, Auto-Vervollständigung, Bracket Matching und anpassbare Keyboard-Shortcuts.



Hier gibt es Visual Studio Code

Auch diese beiden Tools bieten Standardinstallationsassistenten, die keine Probleme darstellen sollten.

## Verschiedene Browser

Zwar hat man in einigen IDEs einen Vorschaumodus für Web-Technologien zur Verfügung, aber in der Regel wird dabei indirekt auf einen Browser zurückgegriffen. Sie sollten als PHP-Programmierer auf jeden Fall mehrere Browser zur Verfügung haben. Denn leider gilt, dass man Web-Applikationen wegen der teils hohen Abweichungen in der Darstellung und Funktionalität unbedingt in allen relevanten Browsern testen muss. Denn in der Regel wissen Sie ja nicht, welche Browser Ihre Besucher benutzen. Deshalb sollten Sie zu einem effektiven Test einer Web-Applikation immer mehrere Browser heranziehen. Unabdingbar sind Firefox oder einer seiner Verwandten wie der Netscape Navigator oder Mozilla und Chrome beziehungsweise Safari (macOS). Und natürlich sollten Sie unter Windows den Internet Explorer und Edge berücksichtigen.

## Browser als Zentrale der Web-Entwicklung

Moderne Browser sind nicht nur primitive Anzeigeprogramme für Webseiten, sondern mit vielfältigen Features erweitert worden, die auch die Web-Entwicklung betreffen. Es gibt in allen Browsern Entwicklertools, die Sie meist mit der Taste **F12** oder einem entsprechenden Menübefehl aufrufen können.

## A.2 Programmieren und Debuggen mit Xdebug und PDT

So mächtig Eclipse mit den PDT auch ist – das Tool ist nicht ganz selbsterklärend und hinsichtlich seiner Leistungsfähigkeit recht unübersichtlich. Im folgenden Abschnitt finden Sie einige Erklärungen.

### Projekte als Basis

Bei Eclipse bearbeiten Sie in der Regel keine einzelnen Dateien, sondern Eclipse arbeitet grundsätzlich mit sogenannten Projekten. Auch wenn Sie mit PHP arbeiten, werden Sie fast immer erst in Eclipse ein Projekt anlegen. Das ist ein von Eclipse verwaltetes Verzeichnis, in dem sich alle Ressourcen (also etwa die PHP-Dateien und andere Web-Dokumente wie HTML-, CSS- oder JavaScript-Dateien oder Bilder) befinden, aber auch Konfigurationen und Verknüpfungen gespeichert werden.

Projekte wiederum werden in Eclipse gemeinsam in einer übergeordneten Struktur mit Namen „Workspace“ zusammengefasst. Auch das ist ein Verzeichnis mit Metainformationen.

Die Ansichten Ihrer Ressourcen eines Projekts werden in „Perspektiven“ zusammengefasst, die aus sogenannten Views (Sichten) bestehen. Das sind einzelne Fenster innerhalb der IDE, die bestimmte Informationen Ihres Projekts anzeigen. Die Views können aber auch einzeln angezeigt und weggeblendet werden.

Für die Arbeit mit PHP sind verschiedene dieser Views in einer PHP-Perspektive zusammengefasst.

## Die Menüs

Das Hauptmenü von Eclipse hält sich natürlich an die üblichen Standards, die Sie auch sonst in modernen grafischen Programmen finden:

- ✓ Sie werden in jedem datenverarbeitenden Programm, das sich an diese Standards hält, ganz links im Hauptmenü ein **Dateimenü** vorfinden. Über das Dateimenü stehen Ihnen die üblichen Befehle zur Dateiverwaltung zur Verfügung. Sie erzeugen damit neue Dateien und speziell bei Eclipse die Projekte, schließen eine einzelne oder alle offenen Dateien, speichern Dateien einzeln oder alle offenen Dateien in einem Schritt, drucken eine Datei und beenden Eclipse. Ebenso bekommen Sie die zuletzt geöffneten Dateien am unteren Ende vom Menü angezeigt.

Erstmal ist das nicht ungewöhnlich, aber dazu kommen Besonderheiten. Über das **NEW-Untermenü** erzeugen Sie etwa grundsätzlich eine neue Datei. Es gibt jedoch mehrere Arten von Dateien, die Sie in Eclipse erstellen können. Diese können Sie dann in Untermenüs auswählen. Sie greifen in allen Fällen zum Erzeugen eines Ordners oder einer Datei über FILE -> NEW nicht direkt auf die Verzeichnisstruktur des Rechners zu, sondern verwenden als Wurzel das Workspace-Verzeichnis von Eclipse. Dementsprechend müssen alle Dateien und Ordner einem Projektordner untergeordnet werden. Nur das Projekt selbst können Sie bei Bedarf an einen beliebigen Ort im Verzeichnissystem positionieren. Interessant sind auch die Anweisungen zum Import von Ressourcen zur Workbench und dem Export aus Eclipse heraus. Mit den beiden entsprechenden Menübefehlen können Sie die Schritte über Assistenten erledigen.

- ✓ Neben dem Dateimenü befindet sich das **Bearbeitenmenü** mit den üblichen Befehlen.
- ✓ Für das Ausführen von PHP-Skripten aus der IDE heraus ist das **RUN-Menü** sehr wichtig. Man kann dort spezielle Varianten der Ausführung auswählen sowie die Konfiguration spezieller Ablaufbedingungen einstellen. Eclipse wird bei der ersten Ausführung von einem Programm eine neue Konfiguration mit Ablaufbedingungen für ein Programm erzeugen, die mit der Tastenkombination **Strg+F11** aufgerufen werden kann. Damit starten Sie das zuletzt ausgeführte Programm neu.

Auch in der Symbolleiste von Eclipse finden Sie eine Schaltfläche, mit der Sie ein Programm starten können – den RUN-Button. Diesem ist immer die zuletzt ausgeführte Run-Konfiguration zugeordnet, sodass für eine erneute Programmausführung auch ein Klick auf die Schaltfläche genügt. Zusätzlich finden Sie rechts neben der Schaltfläche einen Listenpfeil. Damit können Sie alle erzeugten Run-Konfigurationen wiederfinden und auswählen. Weitere Anmerkungen dazu sollen an späterer Stelle erfolgen (beim Debuggen).

- ✓ Ganz rechts steht das **Hilfemenü**.
- ✓ Wenn ein Programm mit mehreren Fenstern arbeitet, werden Sie unmittelbar links vom Hilfemenü ein **Fenstermenü** finden, mit dem die verschiedenen Fenster und alles, was damit zu tun hat, verwaltet werden können. Das Menü ist die zentrale Navigationsinstanz in Eclipse, wenn Sie irgendwelche Fenster öffnen, schließen oder anordnen wollen.

Wichtig sind die bereits angesprochenen Views und Perspektiven. Eclipse hat einige dieser Perspektiven als sinnvolle Vorschläge integriert. Sie können aber auch eigene Perspektiven erstellen und speichern. Über WINDOW -> OPEN PERSPECTIVE gelangen Sie an ein Untermenü, über das Sie die verschiedenen Perspektiven von Eclipse anzeigen können. Ggf. wählen Sie OTHERS... und Sie sehen in einem Dialog alle Standardperspektiven in Ihrem Eclipse.

Nun bestehen Perspektiven wie gesagt aus einzelnen Ansichten (Views), die immer eine spezialisierte Sicht auf eine bestimmte Situation gestatten. Und diese Views können Sie auch bei Bedarf gezielt öffnen oder schließen – unabhängig von der Perspektive, die nur eine gewisse Vorauswahl an Views bereitstellt. Über WINDOW -> SHOW VIEW gelangen Sie an ein Untermenü, über das Sie losgelöst von einer Perspektive die verschiedenen Ansichten von Eclipse anzeigen können. Dieses Untermenü zeigt Ihnen eine Vorauswahl mit Ansichten, die von der ausgewählten Perspektive abhängt. In allen diesen Untermenüs finden Sie jedoch auch hier den Eintrag OTHER.... Darüber kommen Sie zu einem Dialog, in dem Sie alle verfügbaren Ansichten auswählen können.

Über WINDOW -> PREFERENCES können Sie die zentralen Einstellungen von Eclipse vornehmen.

- ✓ Weitere Menüs sind immer abhängig davon, was ein Programm spezifisch leistet. Auch der genaue Inhalt einzelner Menüs umfasst neben den üblichen Standardbefehlen an der jeweiligen Stelle individuelle, programmtypische Befehle. Und diese orientieren sich in Eclipse natürlich an den Bedürfnissen der Programmierung.

Befehle, die in einer spezifischen Phase nicht zu aktivieren sind, werden von Eclipse wie üblich grau dargestellt. Ebenso unterstützt Eclipse in der Grundeinstellung die gewohnten Tastaturlbefehle für übliche Befehle (etwa **Alt****F4** zum Schließen einer Datei oder **Strg****F** zum Suchen im Text).

## PHP ausführen & debuggen

Das Ausführen von PHP-Skripten aus der IDE und das Debuggen sind sehr verwandt. Im Grunde unterscheiden sich beide Techniken nur darin, dass beim Debuggen die Programmausführung zwischenzeitlich unterbrochen und die Situation untersucht wird. Der zentrale Startpunkt ist aber in beiden Fällen das Run-Menü. Dort starten Sie entweder das „normale“ Ausführen des Programms oder eben das Debuggen.

Das Debuggen von PHP ist jedoch immer noch ein nicht ganz einfaches Thema. Oder genauer – man muss erst einmal einen Debugger für PHP finden, denn der ist in den PDT nicht automatisch integriert: Und dann muss man diesen auch erst einmal zum Laufen bringen. Denn besonders, wenn man objektorientiert programmiert, ist es oft mit einfachen echo-Ausgaben oder dem Auskommentieren von Code-Passagen nicht getan.

Zwar helfen unter PHP einige Testfunktionen wie `var_dump()`, um einen Objektzustand zu verifizieren, aber auch das kommt nicht an die Möglichkeiten von einem Debugger heran. Es gibt einige Debugger für PHP, die im Grunde auch weitgehend gleich sind bzw. aufeinander basieren. Zwei populäre Vertreter sind:

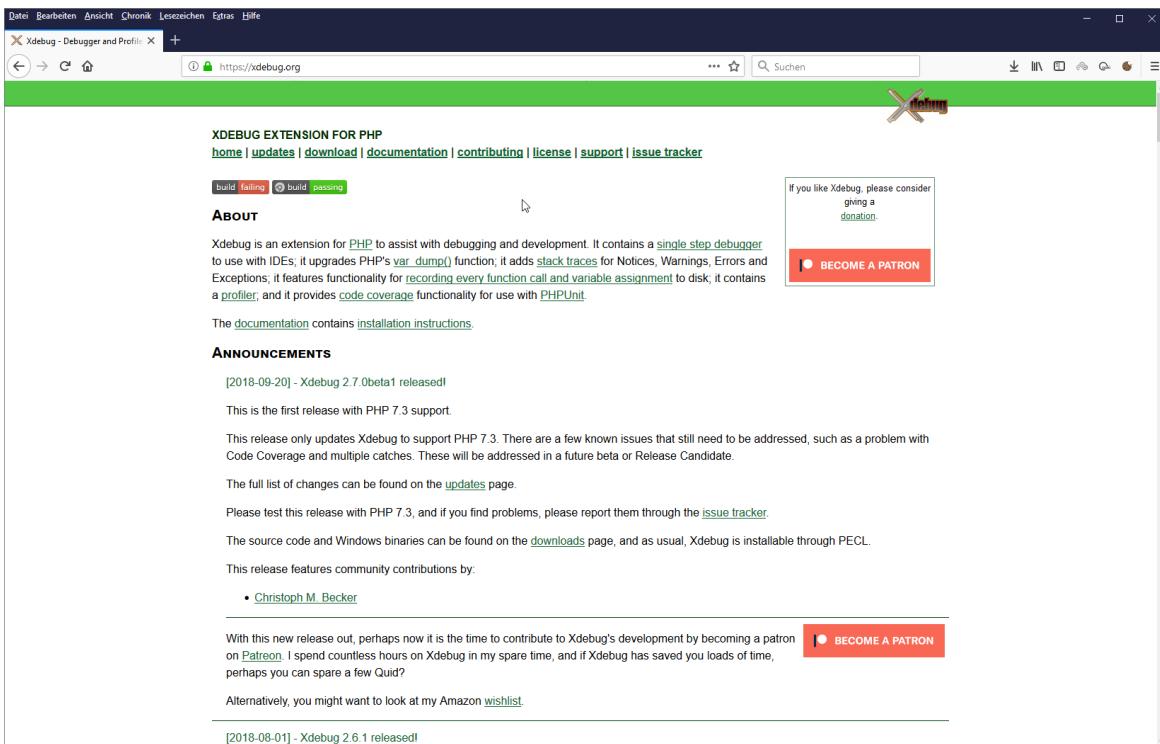
- ✓ Zend Debugger und
- ✓ Xdebug (<https://xdebug.org/>)

Grundlage bei beiden ist der Zend-Debugger. Dazu kommen noch einige kommerzielle Lösungen, wie die von PHPStorm, bei der aber oft auch auf den Zend-Debugger zurückgegriffen wird.

Im folgenden Abschnitt wird nun die Installation und Konfiguration von Xdebug sowie und die Arbeit damit im Zusammenspiel mit den PDT gezeigt.

## Download und Installation von Xdebug

Auf der Webseite des Xdebug-Projekts finden Sie einen Download-Bereich, über den Sie den Debugger als lauffähigen Code für verschiedene Betriebssysteme laden können. Das ist erst einmal notwendig. Für Windows gibt es beispielsweise eine DLL, die dann aus einer IDE wie Eclipse mit den PDT oder auch Visual Studio Code aufgerufen werden kann. Man muss Xdebug also nicht im eigentlichen Sinn „installieren“, sondern die ausführbare Datei verfügbar machen.



Die Webseite von Xdebug enthält einen Download-Bereich

## Konfiguration und Verknüpfung mit dem PHP-System

Richtig konfiguriert arbeitet Xdebug vor allen mit Eclipse und den PDT hervorragend zusammen. Aber es ist nicht ganz so einfach, wenn man das kostenlose Tool Xdebug bzw. unter Windows die DLL nach dem Download einsetzen will. Dabei ist es zunächst einmal wichtig, dass in der Konfigurationsdatei von PHP selbst (der Datei `php.ini`) nach der „Installation“ gewisse Konfigurationen für Xdebug vorgenommen werden, denn Xdebug nutzt explizit den „normalen“ PHP-Interpreter.

Auf den offiziellen Webseiten von Xdebug findet man für den Einsatz unter Windows diese Anleitung, wie man zur Einrichtung vorgehen muss und was in der Datei `php.ini` einzutragen ist – wobei die Pfade und Versionsangaben natürlich anzupassen sind:

- ✓ Download der Datei `php_xdebug-2.6.1-7.1-vc14.dll`.
- ✓ Kopieren der geladenen Datei nach `F:\xampp\php\ext`. Das ist das allgemeine Verzeichnis für Erweiterungen der PHP-Umgebung.
- ✓ Dann ist das Update der Datei `php.ini` notwendig. Sie finden diese im PHP-Verzeichnis von XAMPP (etwa da `F:\xampp\php\php.ini`). Und dabei ist die folgende Zeile einzutragen:  
`zend_extension = F:\xampp\php\ext\php_xdebug-2.6.1-7.1-vc14.dll`

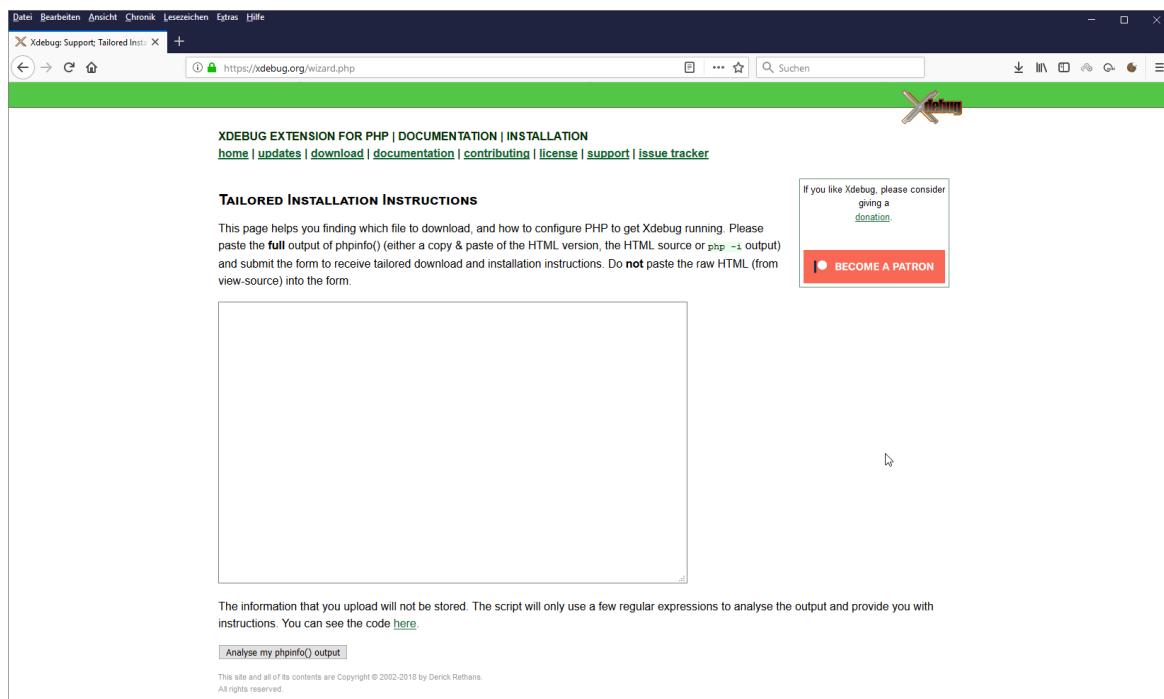
- ✓ Danach ist der Webserver neu zu starten.

Des Weiteren wird oft empfohlen, dass die folgenden Zeilen in der Datei `php.ini` stehen, wobei auch hier die Versionsangaben und Pfade anzupassen sind:

```
[XDebug]
xdebug.remote_enable=true
xdebug.remote_host=127.0.0.1
xdebug.remote_port=9000
xdebug.remote_handler=dbgp
xdebug.profiler_enable=0
xdebug.profiler_output_dir="F:\xampp\php\ext\tmp"
```

Bitte dabei die Angabe `127.0.0.1` statt des synonymen `localhost` beachten. Damit sollen sich einige Probleme vermeiden lassen, wenn man die Diskussionen im Internet beachtet.

Sollte Xdebug mit dieser Konfiguration nicht funktionieren, bietet das Xdebug einen Wizard mit einem Formular an, in das man den Inhalt der Ausgabe der PHP-Funktion `phpinfo()` kopieren muss. Nach der Analyse werden ggf. Korrekturvorschläge unterbreitet und dann sollte der Debugger funktionieren.

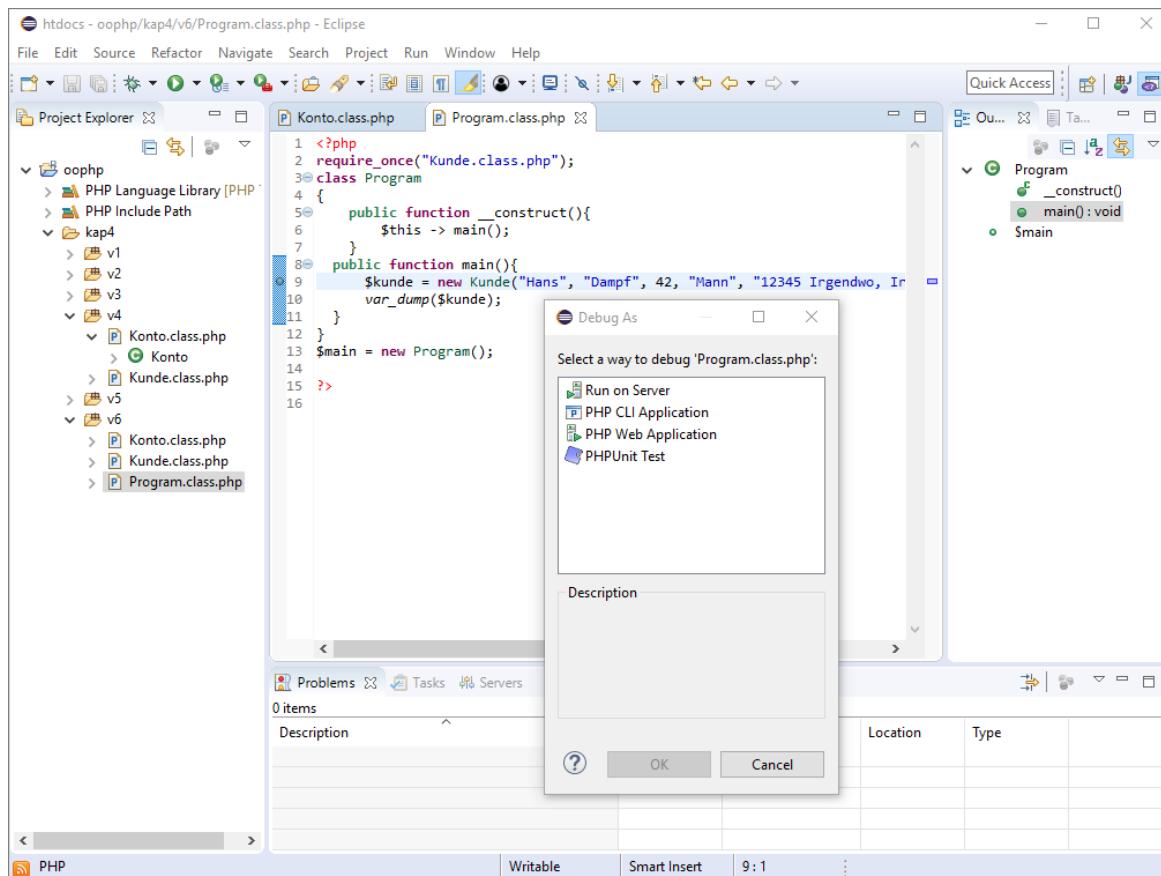


#### *Der Wizard zur Analyse der PHP-Konfiguration*

Die Verwendung von Xdebug geschieht etwa aus Eclipse mit den PDT heraus. Wenn Sie dort ein PHP-Projekt angelegt haben, gibt es unter dem Run-Menü einen Befehl zum Debuggen einer PHP-Datei. Wenn Sie diesen jedoch auswählen wollen, müssen Sie zuerst angeben, auf welche Weise Sie die PHP-Datei debuggen wollen. Denn es gibt ein paar Auswahlmöglichkeiten.

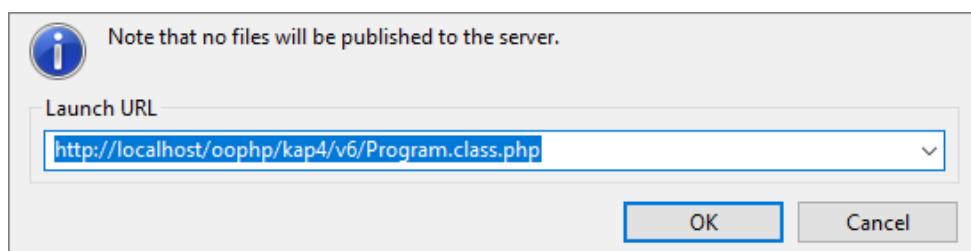
## Debuggen als PHP Web Application

Das Debuggen als klassische PHP Web Application sollte der übliche Weg und in den meisten Fällen sinnvoll sein. Sie rufen bei dem Vorgehen eine PHP-Datei aus einem Browser auf und führen also das PHP-Programm genauso aus, wie es in der Praxis auf dem Webserver ausgeführt wird – nur aus der IDE mittels des Run-Menüs und dem Debug-Befehl. Dies geschieht aber auch, wenn Sie mit dem Run-Menü den „normalen“ Run-Befehl ausführen. Alle Ausgaben werden wieder in der IDE vorgenommen, die auch einen integrierten Browser bereitstellt.



*Auswahl, welche Art von Debugging Sie wünschen*

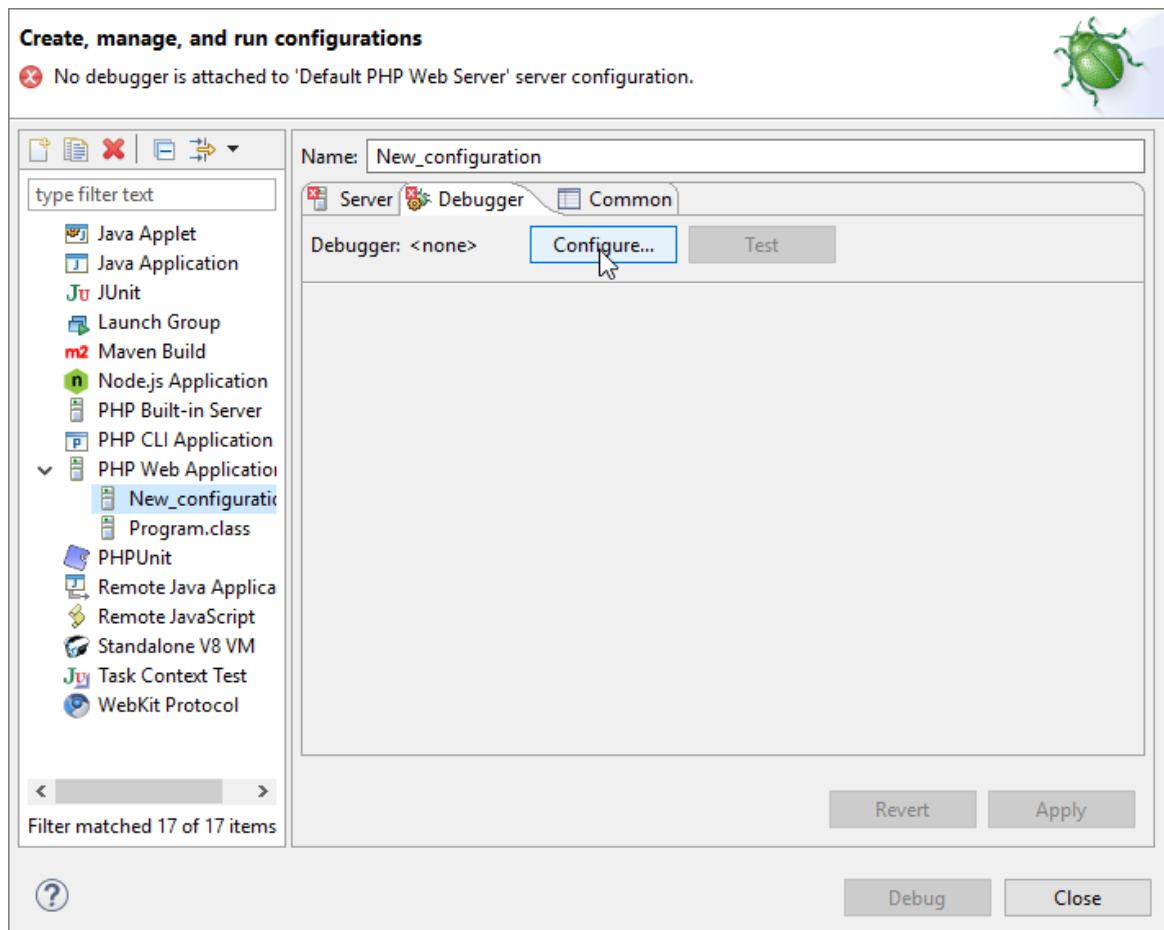
Wenn das Debuggen als PHP Web Application ausgewählt wird, erhalten Sie im folgenden Schritt einen Hinweis auf die URL, unter der die PHP-Datei dazu publiziert wird. Das wird in der Regel ein Unterverzeichnis auf dem lokalen Webserver sein.



*Die URL, die zum Debuggen verwendet wird*

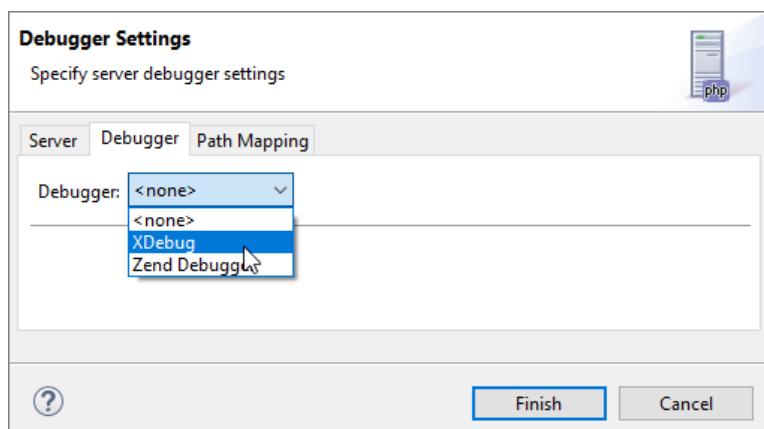
## Einrichten von Eclipse und den PDT

Selbst wenn Xdebug korrekt konfiguriert ist, müssen Sie vor dem ersten Debuggen mit Eclipse und den PDT das gesamte System noch einrichten und vor allen Dingen in Eclipse einstellen, welchen Debugger Sie verwenden. Das macht man unter dem Run-Menü in der Debugg-Konfiguration bzw. der Konfiguration der PHP Web Application und dann dem Register Debugger.



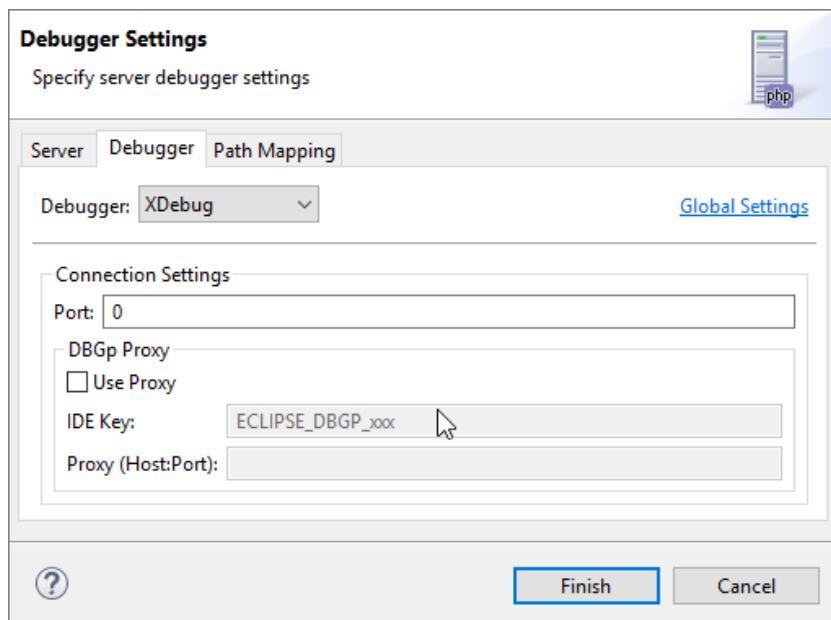
*Vor dem Debuggen muss eine Debugg-Konfiguration eingerichtet werden*

Bei den Debugger Settings wählen Sie XDebug aus:



*Solange kein Debugger ausgewählt wurde, funktioniert das Debuggen aus den PDT heraus nicht*

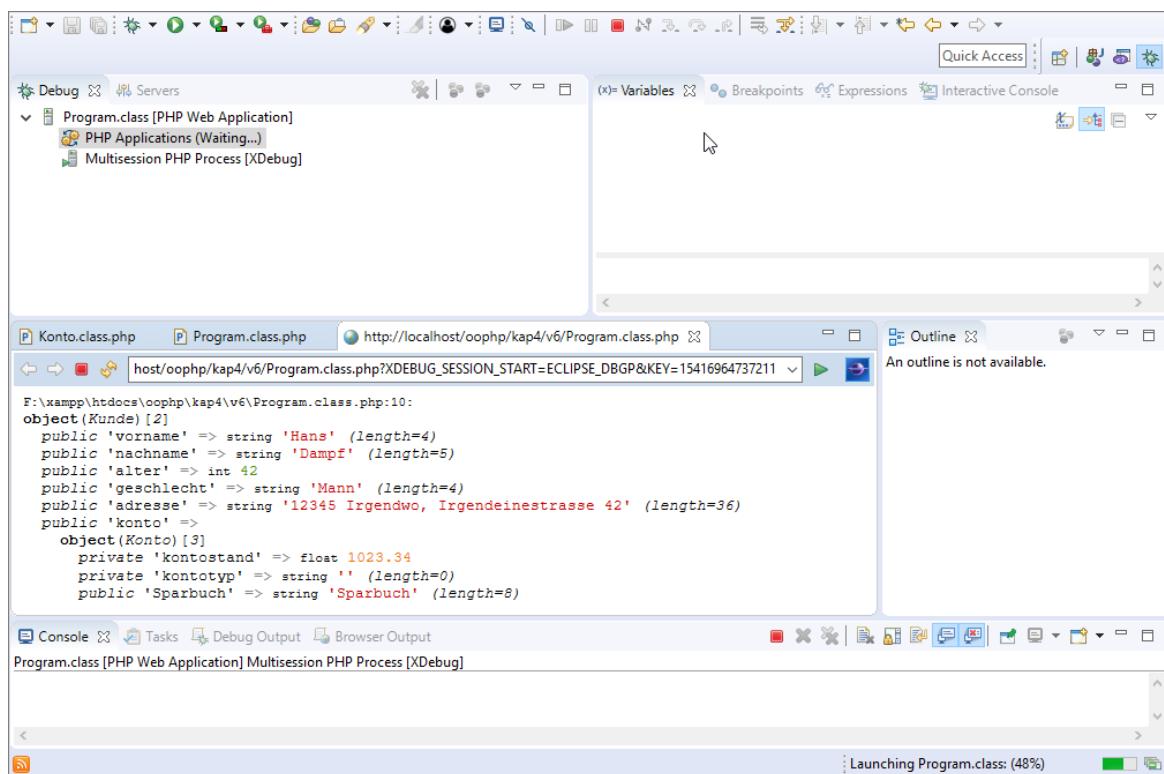
Sofern Xdebug richtig konfiguriert ist, werden damit alle notwendigen Angaben Eclipse und PDT automatisch zur Verfügung gestellt.



*XDebug wird explizit als Debugger ausgewählt und alle Angaben stehen bereit*

Danach sollte das Debuggen mit allen üblichen Schritten funktionieren wie mit

- ✓ dem Unterbrechen des Programmlaufs,
- ✓ dem schrittweisen Fortsetzen oder
- ✓ der genauen Analyse von Variablen, Stack, etc.

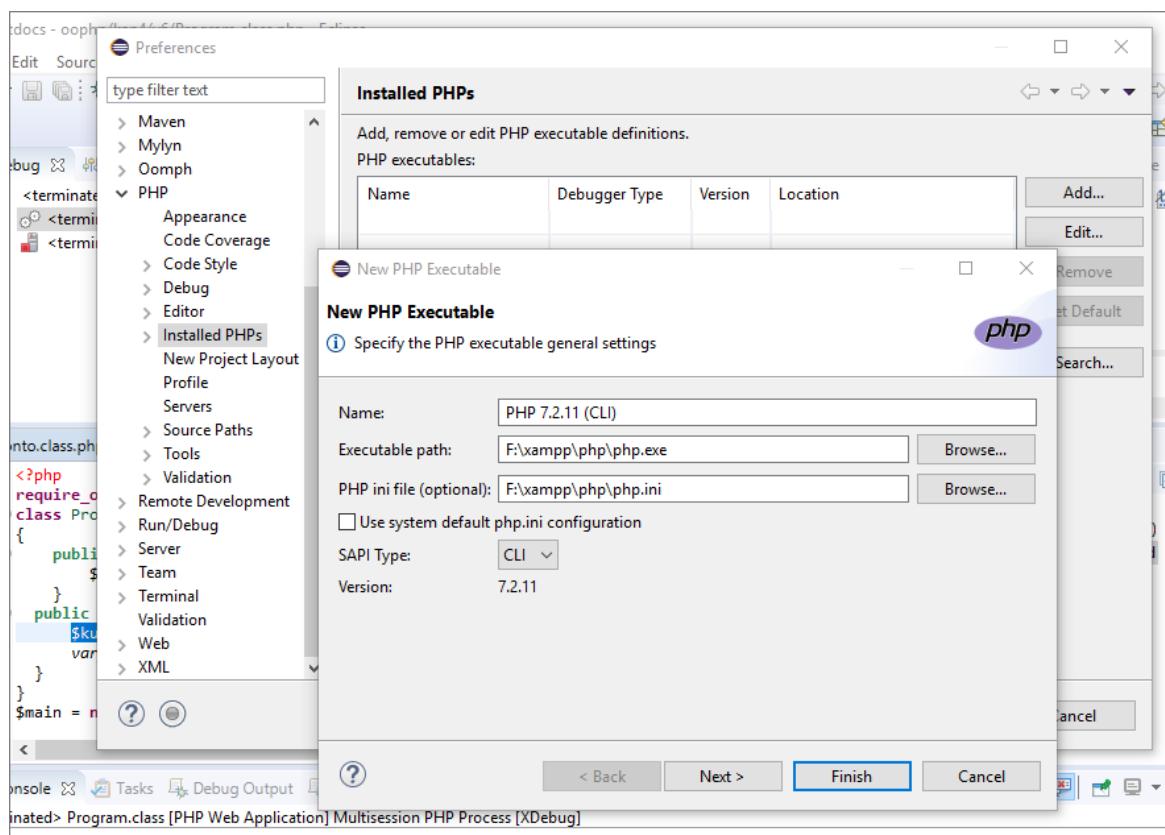


*Der Debug-Modus zeigt diverse Views an, die sich auch anpassen lassen*

## Debuggen als PHP CLI Application

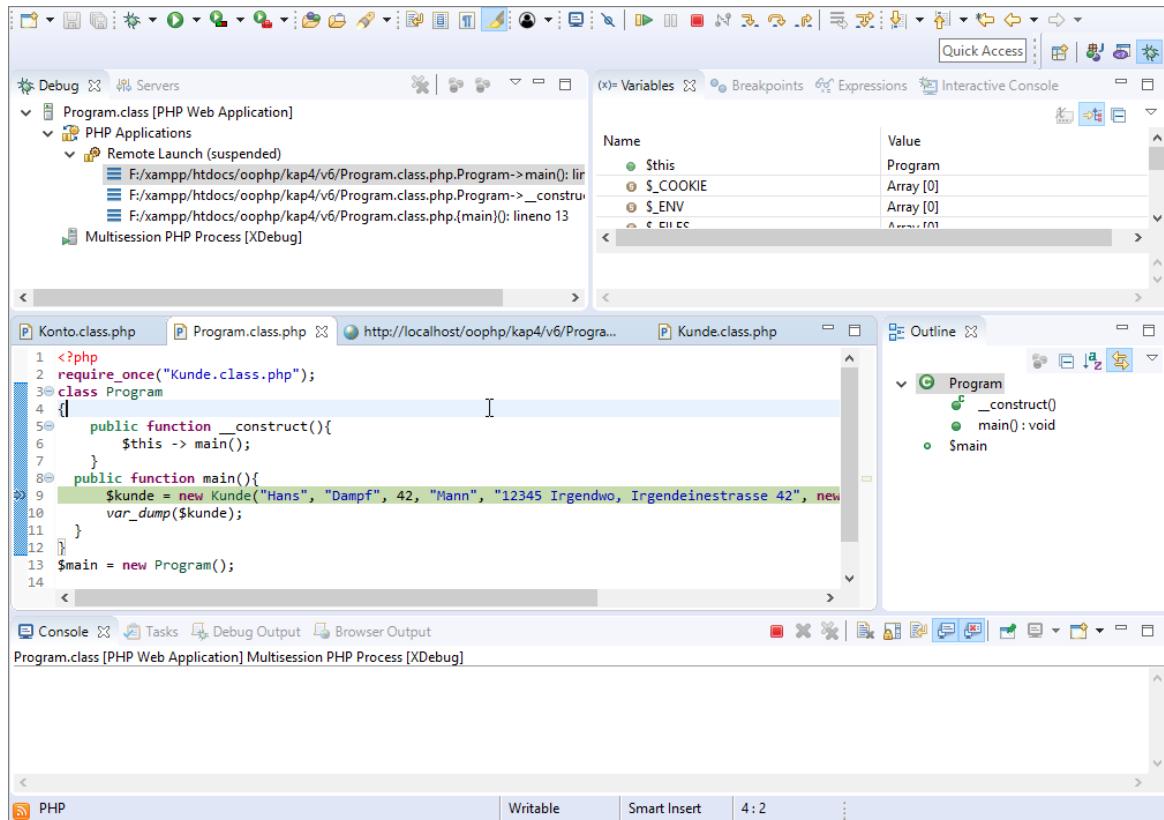
Wenn Sie eine PHP-Datei als PHP CLI Application debuggen oder ausführen, wird das PHP-Skript nicht von einem Browser über http von einem Webserver angefordert und die Ausgabe nicht über einen Webserver an einen Browser geschickt, sondern das Programm läuft vollständig in einer Konsole (CLI – command-line interface – Kommandozeile).

Um so vorgehen zu können, muss Eclipse den Pfad zum PHP-Interpreter (unter Windows php.exe) kennen. Im Fall einer Web-Applikation braucht Eclipse diesen nicht zu kennen, denn er ist ja dem Webserver bekannt und über ihn erfolgt die Anfrage. Den Pfad zum PHP-Interpreter richten Sie in der Konfiguration von Eclipse im Bereich PHP ein.



Zum Debuggen als CLI-Applikation muss ein PHP Executable eingerichtet werden

Danach kann man auch auf diesem Weg mit Xdebug debuggen oder aber auch einfach die PHP-Datei ohne die Randwirkungen von Webserver, http und Browser ausführen.



Wenn der Debugger läuft, können Sie den Quellcode zeilenweise durchlaufen und auch tiefgehend analysieren

### A.3 Quellangaben im Internet

Das deutsche PHP-Forum	<a href="https://www.php.de/">https://www.php.de/</a>
NotePad++	<a href="http://notepad-plus.sourceforge.net">http://notepad-plus.sourceforge.net</a>
PDT	<a href="https://eclipse.org/pdt/">https://eclipse.org/pdt/</a>
FileZilla	<a href="https://filezilla-project.org/">https://filezilla-project.org/</a>
Eclipse	<a href="http://www.eclipse.org">http://www.eclipse.org</a>
Java und JDK	<a href="http://www.oracle.com/technetwork/java/javase/downloads/index.html">http://www.oracle.com/technetwork/java/javase/downloads/index.html</a>
Visual Studio	<a href="https://visualstudio.microsoft.com/de/">https://visualstudio.microsoft.com/de/</a>
Visual Studio Code	<a href="https://code.visualstudio.com/">https://code.visualstudio.com/</a>
W3C	<a href="http://www.w3.org">http://www.w3.org</a>
XDebug	<a href="https://xdebug.org/">https://xdebug.org/</a>
PHP Net	<a href="http://php.net/">http://php.net/</a>

<b>\$</b>		Bottom-up-Methode	57	Eigenschaften	18, 23
<code>\$_SERVER['PHP_SELF']</code>	97	Brackets	127	Einfachvererbung	58, 84
<code>\$this</code>	26	Browser	131	Elemente, statisch	18
		Browser, Entwicklertools	131	Elternklasse	56
				Entwicklertools, Browser	131
<b>:</b>				execute(), PDO	114
<code>::</code>	48			exit()	106
				extends	60
<b>-</b>					
<code>__autoload()</code>	52	<b>C</b>			
<code>__clone()</code>	44	CGI	8	<b>F</b>	
<code>__construct()</code>	32	charset, PDO	106	Feld	18
<code>__destruct()</code>	38	class_exists()	61	fetch(), PDO	109
		CLI Application	139	fetch_all()	111
<b>&gt;</b>		clone	44	fetch_field(), PDO	112
<code>-&gt;</code>	28	columnCount(), PDO	112	fetch_fields(), PDO	112
<b>1</b>		Common Gateway Interface	8	fetchAll(), PDO	109
127.0.0.1	125	connect(), PDO	104	field_count, PDO	112
		connect_error,		FileZilla	122, 128
		PDOException	106	final	75
<b>A</b>		Content-Type	14	Finale Klassen (final classes)	56
Abgeleitete Klasse	56	Contract based programming	79	FTP-Client	128
<code>abstract</code>	78	Control Panel, XAMPP	124	FTP-Server	122, 128
Abstrakte Klassen	19, 78	create, SQL	103	Funktionen	17
Abstraktion	20				
Aggregationen	19, 35	<b>D</b>			
Ajax	10	Data Control Language	103	<b>G</b>	
Akronym, rekursiv	10	Data Definition Language	103	Garbage Collector	38
Aktuelles Objekt	27	Data Manipulation Language	103	gc_collect_cycles()	38
Anonyme Instanziierung	37	Data Source Name	105	Gegenstromverfahren	57
Anonyme Klassen	47	Dateimenü, Eclipse	132	Generalisieren	19
Anonymes Objekt	37	Datenbanken	101, 117	Generics	19
Apache	122	Datenbankhost, PDO	104	Geschützter Zugriff	69
Apache Friends	122	Datenbankmanagement-		get_class()	61
API	59	system	11, 101	get_class_methods()	61
Application Programming		Datenbankschnittstelle	102	get_class_vars()	61
Interface	59	Datenbanktreiber, PDO	105	get_declared_classes()	61
Applikationsserver	9	Datenbasis, Datenbank	101	get_parent_class()	61
Art-von-Relation	56	Datenkapselung	29	getMessage(),	
Assoziationen	19, 35	Datenquelle, PDO	104	PDOException	106
Asynchronous JavaScript		Datentypen, Parameter	25	Getter	29
And XML	10	DBMS	11, 101	global	26
Attribute	18	DCL	103	grant, SQL	103
Attribute, statische	47	DDL	103	Gültigkeitsbereichsoperator	47, 48
Ausführen, PHP	133	Debuggen	133		
Autoloading	52	Debuggen, PHP	131	<b>H</b>	
		Debugger	133	Hardware	5
<b>B</b>		Defaultkonstruktor	31, 32	HTML	8
Basisklasse	56	delete, SQL	103	HTML-Seite	8
<code>bindParam(), PDO</code>	114	Deprecated	5	HTTP	9
Blattklassen	56	Destruktor	38	HTTPS	9
Botschaft, Objekt	28	die()	106	Hyper Text Markup Language	8
		DML	103	Hypertext Transfer Protocol	9
		Dot-Notation	28	Hypertext Transfer	
		DSN	105	Protocol Secure	9
		Eclipse, PDT	129		

<b>I</b>	<b>L</b>	<b>OOA</b>	35
IDE	Landingpage	36	
implements	Leaf class	56	OOD
<i>index.php</i>	Leerer Konstruktor	32	30, 35
Information Hiding	Lerdorf, Rasmus	10	OOP, Klassen
Inheritance	Lernziele	5	OOP, protected
<i>insert, SQL</i>	Lisp	16	OOP, Sichtbarkeit
Instanzeigenschaften	<i>localhost</i>	125	OOP, Sichtbarkeitsstufe
Instanzelemente	Logo	16	Override
Instanzen	Löschen von Objekten	38	73, 88
Instanziierung, anonyme			
Instanzmember			
Instanzmethoden	<b>M</b>		
Instanzvariable verdecken	Magische Methoden	32, 52	
Instanzvariablen	main()	36	
Integrated Development Environment	MariaDB	122	
interface	Mecury	122	
Interfaces	Mehrfachvererbung	58	Datenbanktreiber
Internet Media Type	Member	21	Datenquelle
Internet Protocol	Message	28	Datenquelle, Zeichensatz
Interpreter, PHP	Methoden	18, 24	PDOException
Invariants	Methoden, magische	32, 52	104, 105
IP	Methoden, statisch/statische	47	PDOStatement
Is-A-Relation	Middle-Out-Methode	57	PDT
Ist-ein-Relation	MIME-Typ	8	Personal Homepage Tools Siehe PHP
<b>J</b>	MIME-Type	14	Pfeilnotation, Objekte
Java	Mitglieder	21	PHP, Bedeutung
Java Servlets	Modifizierer	23	PHP, Debuggen
JavaServer Pages	Modularisierung	22	PHP, Funktionsweise
JSP	Multiple Inheritance	58	PHP, Interpreter
<b>K</b>	Multipurpose Internet Mail Extensions	8	11, 122
Kapselung	MySQL	122	PHP, Version
Kindklasse	MySQLi	102	PHP CLI Application
Kind-of-Relation			139
Klassen			PHP Data Objects
Klassen, abstrakte			5, 103
Klassen, anonyme			PHP Development Tools
Klassen, PDO			129
Klassenattribute			PHP Executable
Klassenbaum			PHP Web Application
Klassenelemente			<i>php.exe</i>
Klassenkonstanten			139
Klassenmember	<b>N</b>		PHP/FI
Klassenmethoden	new	31	PHP: Hypertext Preprocessor
Klonen, Objekte	Notepad++	125	phpinfo()
Kompositionen			phpMyAdmin
Konstruktor	<b>O</b>		PHP-Perspektive
Konstruktor, parametrisierter	Oberklasse	56	PHPStorm
Kontrakt-Programmierung	object	59	Platzhalter, benannt
	Objekt	17, 18	116
	Objekt, aktuelles	27	Platzhalter, unbenannt
	Objekt, anonym	37	116
	Objekt klonen	43	Polymorphie
	Objekt löschen	38, 42	Postconditions
	Objektbeziehungen	19	Preconditions
	Objektidentität	34	19
	Objektorientierte Analyse	35	Prepare Statements
	Objektorientiertes Design	30, 35	114
	Objektorientierung, Kernkonzepte	18	Programmierung, vertragsbasiert
	Objekttypen	18	79
			Projekt, Praxis
			Projekte, Eclipse
			Properties
			protected
			23, 69
			Prozeduren
			17
			Punktnotation

<b>Q</b>	SPL	53	<b>V</b>	
query(), PDO	spl_autoload_register()	53	var_dump()	37, 133
	SQL	103	Verallgemeinerung	55
	SQL-Injektionen	114	Vererbung	19, 55
	SSI	11	Vertragsbasierte Programmierung	79
	Standard PHP Library	53	Vielgestaltigkeit	19
	static	47	Views, Eclipse	131
	Statische Attribute	47	Visual Studio	130
	Statische Elemente	18	Visual Studio Code	130
	Statische Methoden	47	Vorkenntnisse, empfohlene	4
	Structured Query Language	103		
	Subklasse	56		
	Superklasse	56		
<b>R</b>	<b>T</b>		<b>W</b>	
RDBMS	Tags	8	Webanwendungen	9
readonly-Eigenschaft	TCP	9	Webapplikationen	9
Refaktorisierung	text/html	8	Webseiten	8
Referenztyp	Tomcat	122	Wiederverwendbarkeit	17, 55
Rekursives Akronym	Top-down-Methode	57	Workspace, Eclipse	131
Relationales Datenbank-	Transmission Control Protocol	9	Writeonly-Eigenschaft	29
managementsystem	Trennung von Code und Design	22	Wurzelklasse	56
Request	Typen	18		
Response	<b>U</b>			
RIA	Überschreiben	73	XAMPP	122
Rich Internet Application	UML	35	XAMPP, Control Panel	124
Root class	Unified Modeling Language	35	XAMPP, Download	123
rowCount(), PDO	Uniform Resource Locator	8	Xdebug	131
Run, Eclipse	unset()	42		
	Unterklasse	56		
	Unterprogramme	17		
	update, SQL	103		
	Up-down-Methode	57		
<b>S</b>	URL	8		
Schnittstellen				
select, SQL				
Server side Includes				
Setter				
Sicherheit, Datenbank				
Sichtbarkeit				
Sichtbarkeitmodifizierer				
Single Inheritance				
Skalare Datentypen				
Smalltalk				
Sniffer				
Software, verwendete				
Softwarequalität				
Spezialisieren				
Spezialisierung				