Matthias Kannengiesser

Know-how ist blau.

3. aktualisierte Auflage

PHP 5 MySQL 5

Berücksichtigt PHP 5.3

> PHP und MySQL verstehen und fehlerfrei anwenden
> Dynamische Webseiten und komplexe Webanwendungen programmieren
> Komplette PHP/MySQL-Umgebung XAMPP, Aptana Studio, MySQL-Referenz und alle Programmierbeispiele auf CD-ROM

FRANZIS

Einleitung

PHP und MySQL sind ein unschlagbares Duo und dieses gilt es bei der Realisierung von dynamischen Webseiten im Griff zu haben. Das vorliegende Buch widmet sich ausführlich beiden beteiligten Protagonisten.

Über dieses Buch

Die überarbeitete und aktualisierte Neuauflage des Bestsellers bietet Ihnen das Know-how aus zwei Büchern. Das Buch setzt sich aus folgenden Teilen zusammen:

- *Teil 1* – Der erste Teil des Buchs widmet sich der wohl populärsten Skriptsprache – PHP. Es soll Ihnen den Weg durch die mannigfaltige Struktur von PHP weisen. Durch die zahlreichen Codesnippets und Anwendungen sollen sowohl Umsteiger als auch fortgeschrittene Entwickler genügend Raum für eigene kreative Lösungsansätze erhalten. Darüber hinaus werden auch die grundlegenden Sprachelemente ausführlich beschrieben, sodass es zu einem unverzichtbaren Nachschlagewerk wird. Auf eines möchte ich jedoch bereits an dieser Stelle hinweisen: Dieser Teil des Buchs erhebt keinen Anspruch auf Vollständigkeit, da PHP in der fünften Generation bereits so umfangreich ist, dass man gut und gerne Dutzende von Büchern zum Thema verfassen könnte. Die zahlreichen PHP-Erweiterungen würden dieses Unterfangen zum Scheitern verurteilen.
- *Teil 2* – Der zweite Teil des Buchs befasst sich mit dem Wesen von Datenbanken. Den Schwerpunkt bildet dabei MySQL. MySQL gehört zu den populärsten Datenbanken und seit der fünften Generation auch zu den umfangreichsten. Ich verspreche Ihnen nicht, dass Sie innerhalb kürzester Zeit und nur nach der Lektüre dieses Buches zum MySQL-Spezialisten werden können. Erst die Arbeit an eigenen Projekten wird Ihnen zeigen, wie sinnvoll das Buch bei Ihrer alltäglichen Arbeit mit MySQL einsetzbar ist. Vorkenntnisse in MySQL sind sicher kein Grund, das Buch nicht zu lesen – im Gegenteil, auch für Fortgeschrittene sind weitergehende Informationen enthalten. Es handelt sich also nicht um eine oberflächliche Betrachtung, sondern vielmehr um eine Vertiefung des Stoffs.
- *Teil 3* – Der letzte Teil des Buchs liefert nützliche Informationen zu Sicherheitsaspekten.

Tipp: Die Buch-CD hält noch einige Überraschungen für Sie bereit. Ich habe keine Mühe gescheut, Ihnen einige nützliche Tools zusammenzutragen.

Up-to-date - PHP 5.3.0 und MySQL 5.1.36

An dieser Stelle würde ich Sie gerne mit einigen Informationen zu den im Buch enthaltenen Skripts versorgen. Sämtliche Skripts sind sowohl unter PHP 5.2.10 als auch

PHP 5.3.0 getestet worden. Die datenbankbezogenen Anwendungen wurden allesamt unter MySQL 5.0.27 und 5.1.36 auf Herz und Nieren geprüft. Sie sollten bei Ihrer Arbeit mit PHP und MySQL möglichst auf aktuelle und stabile Releases zurückgreifen, wenn es darum geht, einen Produktionsserver zu betreiben.

Allein in PHP 5.3.0 wurden vom Entwicklerteam weit über 140 Bugfixes vorgenommen und diverse PHP-Erweiterungen erweitert und verbessert.

Beim Upgrade von PHP 5.x auf 5.3.0 ist bei einigen Erweiterungen auf die Unterschiede zu achten, daher empfiehlt es sich, einen Blick auf die folgende Datei zu werfen: *http://de3.php.net/manual/en/migration53.php.*

Hinweis: Die vollständige PHP 5.3.0 Changelog-Datei kann unter *http://php.net/ChangeLog-5.php* eingesehen werden.

Gute Voraussetzungen

Sie sollten bereits einige HTML-Grundkenntnisse haben und sich mit dem Einsatz eines Webbrowsers auskennen. Grundlegende Programmierkenntnissse wären sicher auch nicht fehl am Platz. Aber besonders wichtig: Haben Sie Spaß beim Lesen, und nehmen Sie sich ausreichend Zeit zum Experimentieren.

Quelle – Website zum Buch

Die Website zum Buch ist *www.atomicscript.de.*

Der Autor

Matthias Kannengiesser ist Dipl.-Informatiker und Projektmanager im IT-Bereich. Er arbeitet seit mehreren Jahren als IT-Consultant für namhafte Unternehmen. In den letzten Jahren hat er vor allem an der Entwicklung von PHP/MySQL-basierten Lösungen gearbeitet. Seit mehr als zehn Jahren hält er Seminare, Workshops und Vorträge zu den Themen ActionScript, Lingo, JavaScript, PHP und Datenbankdevelopment. Er ist bundesweit als Fachdozent für Institute und Unternehmen tätig und Autor für Magazine wie *Internet Intern, Internet World, MX Magazin* und *Internet Professionell.*

Danksagung

Ich will mich von Herzen bei meinen lieben und geschätzten Freunden und Kollegen bedanken. Das sind insbesondere:

Caroline Kannengiesser – Dank an mein Schwesterherz für die Unterstützung

Dr. Peter Schisler, Ingrid Singer, Bianca Lange und Michael Wrobel (L4 Institut)

Team von Apachefriends.org – Jungs, Ihr seid großartig und XAMPP ist unschlagbar!

Team von Tutorials.de – PHP und MySQL Forever

Team von Traum-projekt.com – Was kostet die Welt?

Alex, Bernd, Jens, Frank, Ina, Conni, Ralph, Christopher, Christian, Markus, Mario, Gökhan, Ralf und all diejenigen, die ich hier vergessen habe.

Einen besonderen Dank möchte ich den Herren Franz Graser und Peter Schmid-Meil aussprechen, meinen Lektoren beim Franzis Verlag: Danke für die kompetente Betreuung bei der Umsetzung dieses Buches.

Last but not least: Ganz viele Umarmungen und Küsse gehen an meine großartige und liebevolle Mama!

Feedback

Ich würde mich über Reaktionen und Anregungen freuen. Sie erreichen mich unter folgender Adresse zu erreichen:

matthiask@atomicscript.de

Ihr

Matthias Kannengiesser

Inhaltsverzeichnis

5 Objektorientierte Programmierung mit PHP 5

5.1 Der Umgang mit Klassen und Objekten

Die Zend Engine 2 ist der Versuch, PHP für den Einsatz im Enterprise-Sektor konkurrenzfähig zu machen und einzugliedern. Sicherlich wurde PHP auch schon vorher von Firmen gezielt eingesetzt. Die Anforderungen an eine Programmiersprache hinsichtlich Zuverlässigkeit und Sicherheit wurden jedoch keineswegs erschöpfend erfüllt.

Die wesentliche Änderung von PHP 5 und der von Zend entwickelten Zend Engine stellt die vollständig überarbeitete Unterstützung für die objektorientierte Programmierung in PHP dar. PHP 5 bietet gegenüber PHP 4 wesentlich bessere Möglichkeiten im Umgang mit Klassen und Objekten, wie z. B. Destruktoren, Exceptions (Fehlerbehandlungsroutinen), Interfaces (Schnittstellen), und eine verbesserte Kapselung. Im Gegensatz zu PHP 4 werden bei Zuweisungen und Übergaben Objekte immer als Referenz behandelt. In Zukunft dürfen Sie sich noch auf weitere Features wie Mehrfachvererbung und strikte Typisierung freuen.

Achtung: In PHP 4 werden Objekte bei Übergabe an Funktionen oder bei Zuweisungen immer als Kopie übergeben und nicht als Referenz!

Zunächst ein Blick auf die in PHP 5 hinzugekommenen Schlüsselwörter.

Schlüsselwort	*Beschreibung*
abstract	Deklariert Klassen oder Methoden als abstrakt.
clone	Wird verwendet, um eine Kopie eines Objekts zu erzeugen.
const	Definiert klassenspezifische Konstanten.
final	Deklariert Klassen oder Methoden als endgültig.
implements	Angabe, wenn eine Klasse ein Interface implementiert.
interface	Deklariert eine Objektschnittstelle.
instanceof	Entspricht `is_a()` und sollte stattdessen in PHP 5 verwendet werden. Gibt `TRUE` zurück, wenn das Objekt von einer Klasse abstammt.
private	Private Mitglieder wie Eigenschaften und Methoden sind für Aufrufer der Klasse nicht sichtbar.
protected	Geschützte Mitglieder wie Eigenschaften und Methoden sind für Aufrufer der Klasse nicht sichtbar, können jedoch in direkt abgeleiteten Klassen verwendet werden.

Schlüsselwort	*Beschreibung*
public	Öffentliche Mitglieder wie Eigenschaften und Methoden sind für alle Aufrufer der Klasse sichtbar. Dies ist der Standardwert, das heißt, ohne Angabe des Schlüsselworts sind alle Mitglieder öffentlich.
self	Kann in Klassen verwendet werden, um Bezug zu Methoden innerhalb derselben Klasse zu nehmen.
static	Methoden oder Eigenschaften können ohne Instanz eines Objekts verwendet werden.
try	Leitet einen Block ein, der der Ausnahmebehandlung unterliegt.
catch	Leitet einen Block ein, der eine spezifische Ausnahme behandelt.
throw	Generiert eine Ausnahme (engl. exception).
__autoload	Global verfügbare Funktion, mit deren Hilfe Klassenabhängigkeiten zur Laufzeit erfüllt werden können. Wird aufgerufen, wenn ein Objekt der Klasse erzeugt werden soll, die Klasse aber nicht deklariert ist.
__call	Ruft dynamisch Methoden auf. Die so deklarierte Funktion wird immer dann aufgerufen, wenn in der betreffenden Klasse keine Methode des verlangten Namens vorgefunden wird.
__construct	Reservierter Name für den Konstruktor einer Klasse. Der Konstruktor wird aufgerufen, bevor das Objekt erzeugt wird. Dieser wird vor allem verwendet, um einen definierten Zustand zu erzeugen. Auslöser ist der Aufruf des Schlüsselworts `new`.
__destruct	Reservierter Name für den Destruktor einer Klasse. Der Destruktor wird aufgerufen unmittelbar bevor das Objekt zerstört wird. Dieser wird vor allem verwendet, um mit dem Objekt verbundene Ressourcen zu bereinigen.
_get	Ruft dynamisch Eigenschaften zum Lesen auf. Die so deklarierte Funktion wird immer dann aufgerufen, wenn in der betreffenden Klasse keine Methode des verlangten Namens vorgefunden wird.
__set	Ruft dynamisch Eigenschaften zum Schreiben auf. Die so deklarierte Funktion wird immer dann aufgerufen, wenn in der betreffenden Klasse keine Methode des verlangten Namens vorgefunden wird.
__METHOD__	Unechte Konstante. Wird diese innerhalb einer Klasse verwendet, enthält sie den Namen der Klasse und der Methode. Innerhalb einer Funktion, jedoch außerhalb einer Klasse enthält sie lediglich den Funktionsnamen.

5.1.1 Klassendefinition in PHP 5

Wenn es um die objektorientierte Programmierung geht, dann stellt die Definition einer Klasse den Ausgangspunkt dar. Um in PHP eine solche Klasse deklarieren zu können, wird das Schlüsselwort `class` benötigt, welches auch in diversen anderen Hochsprachen sein Unwesen treibt.

Beispiel – Klasse (samt Mitgliedern, wie Eigenschaften und Methoden)

```
<?php
// Klassen - Definition
class Produkt
{
```

```
    // Klassen - Eigenschaften
    public $name  = "Vaio X2O";
    public $hersteller  = "Sony";
    public $preis = 1999.95;

    // Klassen - Methode
    public function Kaufen($kaeufer)
    {
      echo "Käufer: $kaeufer\n";
       echo "Verkauf eines ";
       echo $this->name . " von ";
       echo $this->hersteller . "\n";
       echo "-------------------\n";
       echo "Preis: " . $this->preis;
    }
}
?>
```

Definiert wurde hier eine Klasse mit dem Namen `Produkt`. Sie enthält die drei Eigenschaften `$name`, `$hersteller` und `$preis`, die mit Standardwerten belegt werden. Das Schlüsselwort `public` vor den Variablennamen deklariert sie als öffentliche Mitglieder. Zusätzlich enthält die Klasse noch eine Methode `Kaufen`, welche ebenfalls als öffentliches Mitglied deklariert wurde.

Bisher liegt lediglich die Klassendefinition vor, als Nächstes folgt die Erzeugung eines Objekts, um auf die Klassenbestandteile (Mitglieder bzw. Elemente) zugreifen zu können.

5.1.2 Objekte erzeugen und verwenden

Objekte sind in der Lage, Aktionen auszuführen. Dazu muss jedoch erst ein solches Objekt mithilfe des Schlüsselworts `new` erzeugt werden.

Beispiel – Objekt erzeugen und auf Methode zugreifen

```
// Objekt - Erzeugung
$einkauf = new Produkt();

// Methoden - Aufruf
echo "Onlineshop: \n";
$einkauf->Kaufen("Caroline");
```

Ausgabe

```
Onlineshop:
Käufer: Caroline
Verkauf eines Vaio X2O von Sony
-------------------
Preis: 1999.95
```

Das erzeugte Objekt `$einkauf` wird in einer Objektvariablen gespeichert. Anschließend kann das Objekt verwendet werden. Auf Eigenschaften und Methoden wird über die Verweissyntax `$objetname->...` zugegriffen.

5.1.3 Konstruktoren und Destruktoren

Der bereits aus PHP 4 bekannte Konstruktor ist Teil eines Phasenmodells, wobei das Objekt nach dem Initialisierungszustand (Konstruktor) zur Benutzung freisteht und am Ende beseitigt wird (Destruktor). Ein typisches Szenario, welches wir auch bei Aktionen im Alltag wieder finden, wenn wir beispielsweise etwas auf- und abbauen.

Funktionsweise von Konstruktoren und Destruktoren

Ein Konstruktor ist nichts anderes als eine Funktion, die immer dann aufgerufen wird, wenn das Objekt der Klasse instanziert wird. Der Konstruktor wird ab sofort mit `__construct()` deklariert, obwohl er auch dann weiterhin als Konstruktor erkannt wird, wenn er den Namen der Klasse trägt. Konstruktoren und Destruktoren besitzen im Gegensatz zu gewöhnlichen Funktionen keinen Returntyp. Sämtliche Aufräumarbeiten, die zum sorgfältigen Beseitigen eines Objekts beitragen, übernimmt der parameterlose Destruktor `__destructor()`. Dazu gehört neben der zuletzt aufgerufenen Referenz die Freigabe jeglicher Speicherressourcen für das Objekt.

Ein Objekt existiert, wie alle anderen Variablen auch, bis zum Ende des Skripts oder bis es gezielt zerstört, also auf `NULL` gesetzt oder mit `unset()` vernichtet wird. Normalerweise ergeben sich daraus keine Konsequenzen, PHP sorgt automatisch für die Freigabe des belegten Speichers. Es gibt jedoch Anwendungsfälle, in denen externe Programme an der Kommunikation beteiligt sind, wie z. B. Datenbanken. In solchen Fällen wäre es fatal, wenn ein Objekt eine Verbindung zur Datenbank herstellt und dann zerstört wird, während die Verbindung bestehen bleibt. Es entstehen verwaiste Verbindungen. Verfügt eine Datenbank lediglich über rudimentäre Kontrollmechanismen oder eine begrenzte Anzahl von erlaubten Verbindungen, führt dies früher oder später zu Fehlern. Um das Verhalten am Ende der Existenz eines Objekts zu kontrollieren, sind somit Destruktoren genau das richtige Mittel. Sie werden unmittelbar vor der endgültigen Zerstörung aufgerufen.

Beispiel – Konstruktor und Destruktor

```
<?php

class MeineKlasse {
   public function __construct() {
       echo "Befinden uns im Konstruktor.<br>";
       $this->name = "MeineKlasse";
   }

   public function __destruct() {
       echo "Zerstöre die Klasse: " . $this->name . ".<br>";
   }
}

$objekt = new MeineKlasse();

?>
```

Ausgabe

```
Befinden uns im Konstruktor.
Zerstöre die Klasse: MeineKlasse.
```

Beispiel – ohne public-Schlüsselwort

```
<?php

class Lebewesen
{
   function __construct()
   {
        echo "Konstruktor";
   }
   function __destruct()
   {
        echo "Destruktor";
   }
}

$objekt = new Lebewesen();
print_r($objekt);

?>
```

Ausgabe

```
Konstruktor Lebewesen Objekt
(
)
Destruktor
```

Achtung: Bei abgeleiteten Klassen werden Konstruktoren und Destruktoren der Elternklasse nicht automatisch ausgeführt und müssen explizit mit `parent::__construct()` beziehungsweise mit `parent::__destruct()` aufgerufen werden. Das Schlüsselwort `parent` verweist dabei auf die Klasse, von der mit `extends` geerbt wurde. Der Konstruktor bzw. Destruktor wird über seinen Namen aufgerufen. Nach `parent` darf lediglich mit dem statischen Verweisoperator `::` gearbeitet werden, da zum Zeitpunkt des Konstruktoraufrufs das Objekt noch nicht existiert und deshalb die Definition direkt benutzt wird.

5.1.4 Zugriffsbeschränkung (Datenkapselung)

Eigenschaften und Methoden von Klassen lassen sich mit den Schlüsselwörtern `private` und `protected` vor unerlaubtem Zugriff schützen. Damit ist es möglich, die Sichtbarkeit von Eigenschaften und Methoden einzuschränken. Zugriff auf als `private` deklarierte Eigenschaften und Methoden besteht nur innerhalb der Klasse selbst, Eigenschaften oder Methoden, die als `protected` deklariert wurden, stehen dagegen zusätzlich auch in abgeleiteten Klassen zur Verfügung. Das Schlüsselwort `public` schränkt den Zugriff in keiner Weise ein und entspricht dem Klassenmodell in PHP 4. Mit `public`, `private` oder `protected` kann in PHP 5 der Zugriff auf Eigenschaften, Methoden und Klassen genau festgelegt und eine gute Kapselung erreicht werden.

Zusammengefasst kann man sagen: Es handelt sich bei `public`, `private` oder `protected` um Schlüsselwörter, die sich zum Verstecken und Kapseln von Daten innerhalb von Klassen eignen und mit deren Hilfe man in der Lage ist, die Daten nur noch den in der

Klasse definierten Methoden zugänglich zu machen und somit den unbefugten Zugriff von außen strikt zu unterbinden.

Einsatzmöglichkeiten und Funktionsweise

Mit der Datenkapselung wird die Trennung von Nutzungs- und Implementierungsschicht, von Realisierung und Nutzung verfolgt: Ein Entwickler, der eine Klasse eines anderen Entwicklers nutzen möchte, braucht die internen Abläufe der Klasse nicht zu kennen, er verwendet nur die vereinbarte Schnittstelle.

Zu diesem Zweck werden den Eigenschaften und Methoden einer Klasse bei ihrer Deklaration eine von drei möglichen Sichtbarkeiten zugewiesen:

- `public` – Standardwert. Objekte sämtlicher Klassen können die Eigenschaften oder die Methoden sehen und verwenden. Aus Gründen der Abwärtskompatibilität ist die Angabe optional und kann entfallen.
- `private` – Nur Objekte derselben Klasse können die Eigenschaften oder die Methoden sehen und verwenden. Sie sollten beachten, dass abgeleitete Klassen oder Aufrufe von Objekten nicht darauf zugreifen.
- `protected` – Verhält sich wie `private`, jedoch dürfen Objekte von Subklassen auf `protected`-deklarierte Eigenschaften und Methoden ihrer Superklasse zugreifen.

Hinweis: `public` stellt im Grunde nichts anderes dar als ein Alias von `var`, welches in PHP 4 eingesetzt wird, um Klassenattribute festzulegen. Die Deklaration von Eigenschaften mit `var` wird weiterhin unterstützt, sollte aber in PHP 5 nicht verwendet werden, es sei denn, die Skripts sollen PHP 4-kompatibel sein.

5.1.5 OOP – Rundgang

In älteren PHP-Generationen wurden Objekte wie simple Datentypen behandelt, dies hat sich nun geändert. Objekte verweisen nun nicht mehr auf ihren Wert (*by value*), sondern auf ihr Handle (*by reference*), ein Handle stellt dabei einen Verweis auf einen Objektbezeichner dar.

Um die Neuerungen aus PHP 5 im Überblick darzustellen, sollten wir uns einem Beispiel zuwenden.

Beispiel

```
<?php

//--- class ---/
//
// Aufruf zum erstellen einer Klasse!
// Alles, was sich innerhalb der geschweiften
// Klammern des Klassenaufrufs befindet,
// unterliegt der strengen Klassenhierachie
// von PHP 5

class MeineKlasse {
```

```
//
//--- var ---
//
// Die Variablendefinitionen, die innerhalb
// der Klasse existent sind! Optional können
// hier Parameter vordefiniert werden!
// (String,Array,Boolean,Integer)
//

var $zeit;
var $eingabe;
var $meldung = "Ausgabe beendet!";

//
//--- private ---
//
// Mit Private definiert man Parameter, die nur
// innerhalb dieser Klasse aufgerufen werden
// können und somit NICHT vererbbar sind und von
// sogenannten Subklassen (Unterklassen) nicht
// abgerufen werden können!
//

private $MathePI;

//
//--- __construct ---
//
// Diese Funktion wird beim Aufruf der Klasse
// Automatisch aufgerufen! Dem Konstruktor kann
// man optionale Parameter  beim Aufruf der Klasse
// übergegeben. Dieser dient dazu, die Variablen
// eines Objekts zu initialisieren. Er kann und
// darf NIE Funktionsergebnis liefern!
//

function __construct($eingabe, $zeit) {
  // Nehme den Parameter entgegen und weise ihm die
  // vordefinierten Variable eingabe und zeit zu.
  $this->eingabe = $eingabe;
  $this->zeit = $zeit;
}

//
//--- protected ---
//
// Mit protected werden innerhalb der Klasse
// Funktionen definiert, die nur in der Klasse
// selbst von anderen Funktionen aufgerufen
// werden können! Sie sind daher außerhalb
// der Klasse NICHT ausführbar.
//
// Hinweis: print & echo Befehle haben hier
```

```
// nichts zu suchen!
//

protected function saved_funk() {
  // Nehme Eigenschaft input und rufe die private
 // Funktion auf!
  return floor(($this->eingabe)*($this->private_funk()));
}

//
//--- public ---
//
// Mit public hingegen definiert man eine
// Funktion, die innerhalb als auch außerhalb
// der Klasse angesprochen werden darf! Dies
// gilt übrigens auch für Variablen!
//

public function ausgabe_funk() {
    // Rufe die geschützte Funktion auf!
    return $this->saved_funk();
}

//
//--- private ---
//
// Der Parameter private ist mit der protected-
// Funktion vergleichbar, wobei er noch strikter
// vorgeht. Er verhindert eine Vererbung an
// weitere Klassen. Privat ist nur in der
// definierten Superklasse erreichbar und sonst
// nirgends!
//

private function private_funk() {
  // Zuweisung des Werts PI an die Eigenschaft
 // MathePi
  return $this->MathePI = M_PI;
}

//
//--- __destruct ---
//
// Ist eine Callback-Funktion, die bei jedem
// Aufruf der Klasse zurückgegeben wird! Daher
// auch NICHT Explizit aufgerufen werden muss!
// Sie ist dann sinnvoll, wenn man einen bestimmten
// Wert fest zurückgeben möchte! Es werden nur
// echo- und print-Befehle wiedergegeben!
//
// Ausnahme ist hier das Schreiben in Sessions
// oder Cookies, die zuvor definiert wurden!
//
```

```
  function __destruct() {
    $inhalt  = "Ausgabe " . $this->zeit . "<br>";
    $inhalt .= "Von <b>" . __FUNCTION__ . "</b> aus " . $this;
    $inhalt .= " der Superklasse <b>" . __CLASS__ . "</b><br>";
    $inhalt .= $this->meldung . "<br>";
    echo $inhalt;

  }
}
?>
```

Nachdem Sie sich mithilfe der Beispielklasse einen Überblick über die Struktur einer Klasse verschafft haben, gehen wir nun dazu über, ein Objekt mithilfe der Klasse zu erzeugen.

```
<?php

echo "<h3>MeineKlasse</h3>";

$zeitStempel = date("h:i:s ", mktime());

// Objekt erzeugen
$meinObjekt = new MeineKlasse(16, $zeitStempel);

// Ansprechen der public-Funktion
$ausgabe = $meinObjekt->ausgabe_funk();
echo "Ergebnis: " . $ausgabe . "<br>";

?>
```

Ausgabe

```
MeineKlasse
Ergebnis: 50
Ausgabe 11:45:14
Von __destruct aus Object id #1 der Superklasse MeineKlasse
Ausgabe beendet!
```

Des Weiteren ist es möglich, ein Objekt zu klonen und ihm neue Parameter zu übergeben.

```
<?php

// Klon von meinObjekt
$neuesObjekt = clone $meinObjekt;

// Abruf der Funktion über den Klon
$klonausgabe = $neuesObjekt->ausgabe_funk();
echo "Ergebnis: " . $klonausgabe . "<br>";

?>
```

5.1.6 Objekte klonen

Das durch die Zend Engine 2 veränderte Objektreferenzverhalten führt dazu, dass man stets eine Referenz auf eine Ursprungsklasse erzeugen kann und niemals eine Kopie eines Objekts samt Eigenschaften erhält. Wenn man komplexe Datenstrukturen hat und eine Klasse A erzeugt, die im weiteren Verlauf noch benötigt wird, es aber zusätzlich

noch eine Klasse B gibt, die eine Kopie benötigt, um damit einzelne Teile der Objektstruktur zu bearbeiten, ohne dabei die Ursprungsinstanz der Klasse A zu verändern, spielt das neu eingeführte Klonen von Objekten eine bedeutsame Rolle. Hierdurch erhält man sämtliche Eigenschaften des Ursprungsobjekts, inklusive aller Referenzabhängigkeiten.

Wird nun eine Kopie eines Objekts benötigt, kann diese Kopie mithilfe von `clone` erzeugt werden. Ein Objekt wird nach dem Aufruf von `clone $objekt` geklont.

Beispiel

```
<?php

class Fahrzeug
{
   function __construct($typ)
   {
        $this->name  = $typ;
   }
   function ProduziereFahrzeug($obj, $name)
   {
        $obj->name = $name;
   }
}

$mobil = new Fahrzeug("PKW");
$neues_mobil = clone $mobil;

$neues_mobil->ProduziereFahrzeug($neues_mobil, "Fahrrad");

// Ausgabe - PKW
echo $mobil->name;
// Ausgabe - Fahrrad
echo $neues_mobil->name;

echo "Original Objekt:\n";
print_r($mobil);
echo "\n\n";
echo "Geklontes Objekt:\n";
print_r($neues_mobil);

?>
```

Ausgabe

```
PKW
Fahrrad

Original Objekt:
Fahrzeug Object
(
    [name] => PKW
)

Geklontes Objekt:
Fahrzeug Object
(
    [name] => Fahrrad
)
```

Sollen sämtliche Objekteigenschaften geklont werden, dann wird zu allererst überprüft, ob Sie die `__clone()`-Methode selbstständig definiert haben, um das Klonen selbst zu übernehmen. In diesem Fall sind Sie dafür verantwortlich, die notwendigen Eigenschaften im erzeugten Objekt zu deklarieren. Sollte dies nicht der Fall sein, wird die interne Methode `__clone()`, über die sämtliche Klassen verfügen, für dieses Objekt ausgeführt, die sämtliche Objekteigenschaften dupliziert. Bei einer abgeleiteten Klasse kann die `__clone()`-Methode der Elternklasse mit `parent::__clone()` verwendet werden.

Beispiel

```
<?php

class Lebewesen
{
   public $name = "";

   function __construct()
   {
        $this->name  = "Zelle";
   }
   function __clone()
   {
        $this->name = "Klon-Zelle";
        echo "Ich wurde geklont: ";
   }
}

class Menschen extends Lebewesen
{
   function __construct()
   {
        parent::__clone();
   }
}

$objekt = new Menschen();
print_r($objekt);

?>
```

Ausgabe

```
Ich wurde geklont: Menschen Object
(
    [name] => Klon-Zelle
)
```

Beispiel – Vertiefung

```
<?php

class Form {
   public $form_farbe;
   public $form_breite;
   public $form_hoehe;

   function setze_farbe($farbe) {
        $this->form_farbe = $farbe;
```

```
   }

   function __clone() {
        $this->form_farbe = $this->form_farbe;
        $this->form_breite = 100;
        $this->form_hoehe = $this->form_hoehe;
   }
}

$form_objekt = new Form();
$form_objekt->setze_farbe("blau");
$form_objekt->form_breite = 300;
$form_objekt->form_hoehe = 300;

$clone_objekt = clone $form_objekt;
echo $clone_objekt->form_farbe ."<br>";
echo $clone_objekt->form_breite ."<br>";
echo $clone_objekt->form_hoehe ."<br>";

?>
```

Ausgabe

```
blau
100
300
```

Beim Beispiel handelt es sich um eine fiktive Form, wobei sich inmitten der Klasse die `__clone()`-Methode befindet. Dadurch besteht die Möglichkeit, Objekteigenschaften mit neuen Werten zu belegen. Nach der Instanz des Form-Objekts werden die Klassenvariablen mit einem Wert initialisiert und die Objekteigenschaften werden weiterführend durch den Aufruf von `clone $form_objekt` geklont. Bei den nachfolgenden Ausgaben des geklonten Objekts ist festzustellen, dass die Breite innerhalb der `__clone()`-Methode modifiziert wurde und daher einen veränderten Wert ausgibt. Der Farb- und Höhenwert wurde nicht verändert und entspricht dem der zuvor initialisierten Klassenvariablen.

Beispiel – abschließende Anwendung

```
<?php

class Adresse {
   static $id = 0;

   function Adresse() {
       $this->id = self::$id++;
   }

   function __clone() {
          $this->id = self::$id++;
          $this->vorname = $this->vorname;
       $this->nachname = $this->nachname;
       $this->ort = "New York";
   }
}

$obj = new Adresse();
$obj->vorname = "Matthias";
```

```
$obj->nachname = "Kanengiesser";
$obj->ort = "Berlin";

print $obj->id . "<br>";

$clone_obj = clone $obj;
print $clone_obj->id . "<br>";
print $clone_obj->vorname . "<br>";
print $clone_obj->nachname . "<br>";
print $clone_obj->ort . "<br>";

?>
```

Ausgabe

```
0
1
Matthias
Kanengiesser
New York
```

5.1.7 Klassenvererbung in PHP 5

An der Klassenvererbung wurden kaum Änderungen vorgenommen. Achten Sie bitte darauf, dass die als `private` deklarierten Methoden nicht an Subklassen oder vererbte Klassen weitergegeben werden, sie sind nicht vererblich.

```
<?php
//
// Superklasse erweitern
//

class ErweiterteKlasse extends MeineKlasse {

  var $neueingabe;

  function __construct($in) {
        $this->neueingabe = $in;
  }

  public function nehmeInfo() {
     $daten  = "Rufe mit ErweiterteKlasse Funktionen in MeineKlasse auf
:";
     $daten .= $this->ausgabe_funk() + $this->neueingabe . "<br>";
     $daten .= "Wie Sie sehen wird die private Methode nicht";
    $daten .= " zurückgegeben! ;) <br>";
     return $daten;
  }

}
$erweitertesObjekt = new ErweiterteKlasse(5, $zeitStempel);
echo $erweitertesObjekt->nehmeInfo();

?>
```

Erweiterung von Subklassen

Natürlich sind Sie nicht nur auf eine Vererbungsebene beschränkt. Sie können eigentlich auch eine Subklasse oder eine Subklasse einer Subklasse erweitern. Zum Beispiel könnten Sie eine Klasse `Mercedes` schreiben, die die Klasse `Auto`, oder eine Klasse `Lear`, die die Klasse `Plane` erweitert. Der Ablauf ist immer genau so, als würden Sie eine ganz normale Klasse erweitern. Es ist vollkommen unwichtig, ob die Superklasse eine Subklasse einer anderen Klasse ist oder nicht.

5.1.8 Finale Klassen und Methoden

Sie haben bereits erfahren, dass sich Klassen mithilfe des Schlüsselworts `extends` vererben lassen. In manchen Fällen soll dies aber nicht so sein, entweder für eine Klasse als solche oder auch nur für einzelne Methoden. Denn manche Methoden sind für die Funktion der Objekte von elementarer Bedeutung. Gelingt der Schutz mit `private` nicht, da der Zugriff von außen benötigt wird, muss das Überschreiben durch das Schlüsselwort `final` verhindert werden. Von einer so gekennzeichneten Klasse kann nicht geerbt werden, und bei als `final` gekennzeichneten Methoden ist das Überschreiben verboten.

Beispiel – Syntax

```
// Finale Klasse
final class Produkt
{
   ...
   // Finale Methode
   final public function Kaufen($kaeufer)
   {
        ...
   }
}
```

Hinweis: Für Methoden kann `final` mit `private` und `protected` kombiniert werden; Eigenschaften können nicht `final` sein.

5.1.9 Abstraktion von Klassen und Methoden

Werden Klassen oder Methoden als `abstract` gekennzeichnet, wird der Benutzer explizit dazu aufgefordert, hier eigenen Code zu schreiben. Somit ist `abstract` das genaue Gegenteil von `final` – statt des ausdrücklichen Verbots folgt nun das ausdrückliche Gebot. Der Entwickler der Klassen gibt damit Struktur, Namen und Aufbau vor, nicht jedoch die konkrete Implementierung, da dies möglicherweise von der Anwendung abhängt.

Eine Ableitung von Objekten von abstrakten Klassen ist nicht möglich. Es muss deshalb immer eine Implementierung erfolgen. Dies gilt auch für abstrakte Methoden. Es ist jedoch möglich, eine Klasse als abstrakt zu definieren und einige der Methoden bereits voll auszuformulieren.

```
<?php

// Durch abstract wird die Klasse für direkte
// Aufrufe blockiert
abstract class SuperKlasse {
  // Die Variable $wert mit protected verriegeln
  protected $wert = 5;

  // Funktionsname zur Vererbung freigeben
  abstract function ausgabe();

  function multiplitzieren($eingabe) {
     return $eingabe*$this->wert;
  }
}

// Unterklasse (Subklasse)
class ErweiterteKlasse extends SuperKlasse {
  // RICHTIG
  function ausgabe() {
    // Die Funktion ist der Abstract-Klasse bekannt und kann
    // somit auf sie zugreifen!
    return $this->multiplitzieren(10);
  }

  // FALSCH
  function ausgeben() {
    // Dies kann nicht funktionieren, da die Funktion
    // ausgeben() der Abstract-KLASSE nicht bekannt ist!
    return $this->multiplitzieren(10);
  }

}

// Aufruf der extends-Klasse
$testObjekt = new ErweiterteKlasse();

// Nicht vergessen! Die Funktion ausgabe() kann nur
// bei der extends(vererbten)-Klasse aufgerufen werden,
// nicht bei der Superklasse, die mit abstract
// verriegelt wurde!
echo  $testObjekt->ausgabe();

?>
```

Ausgabe

```
50
```

Beispiel

```
<?php
// mit abstract die Klasse für direkte Aufrufe blocken
abstract class Fahrzeug {
  // Variable $tueren mit protected verriegeln
  protected $tueren = 4;

  // Gebe Methodenname zur Vererbung Frei
  abstract function ausgabe();
```

```
  function starten($wert) {
     return  $wert . " mit " . $this->tueren . " Türen wurde gestartet!";
  }
}

// Erstelle Subklasse und nehme mit extens die abstract-OberKlasse
class PKW extends Fahrzeug {
  // RICHTIG
  function ausgabe() {
    // Die Funktion ist der abstrakten Klasse bekannt und
   // kann somit auf sie zugreifen!
    return $this->starten("PKW");
  }

  // FALSCH
  function abfahren() {
    // Das kann nicht funktionieren, da die Funktion
    // abfahren() der abstrakten Klasse nicht bekannt
   // ist!
    return $this->starten("PKW");
  }
}

// Aufruf der Klasse
$meinpkw = new PKW();

// Achtung: Die Funktion ausgabe() kann lediglich bei der
// extends (vererbten) Klasse aufgerufen werden, nicht bei
// der Oberklasse, die mit abstract verriegelt ist!
echo  $meinpkw->ausgabe();

?>
```

Ausgabe

```
PKW mit 4 Türen wurde gestartet!
```

Hinweis: Der Sinn einer abstrakten Klasse liegt darin, dass wenn man eine Vererbung auf mehrere Oberklassen vornimmt, es zu keinen Variablenverletzungen kommen soll. Sie ist also bei mehrfacher Vererbung sehr zu empfehlen!

Abschließend noch einige Besonderheiten, die es beim Einsatz von `abstract` zu beachten gilt:

- Von abstrakten Klassen kann kein Objekt instanzliiert werden.
- Von einer abstrakten Klasse kann nur abgeleitet werden.
- Methoden abstrakter Klassen, die selbst als `abstract` definiert sind, müssen bei einer Ableitung implementiert werden.
- Eine abstrakte Klasse kann Methoden enthalten, die nicht als `abstract` definiert sind. Sobald jedoch eine Methode als `abstract` definiert ist, muss auch die Klasse insgesamt `abstract` sein.

5.1.10 Interface – Objektschnittstellen

Vor allem bei umfangreichen PHP-Anwendungen kommt es äußerst selten vor, dass lediglich ein einzelner Entwickler daran arbeitet. Angenommen, Sie gehören zu einem Team von Entwicklern, bei dem jeder einzelne Entwickler an einem separaten Teil – d. h. einer anderen Klasse – einer umfangreicheren PHP-Anwendung arbeitet. Die meisten dieser Klassen stehen miteinander nicht in Beziehung. Dennoch müssen die verschiedenen Klassen miteinander kommunizieren können. Sie müssen also eine Schnittstelle oder ein Kommunikationsprotokoll definieren, das alle Klassen befolgen.

Eine Möglichkeit wäre, dass Sie eine Kommunikationsklasse erstellen, welche sämtliche Methoden definiert, und dann jede einzelne Klasse dieser übergeordneten Klasse erweitern oder von ihr erben lassen. Da die Anwendung jedoch aus unverwandten Klassen besteht, ist es nicht sinnvoll, sämtliche Klassen in eine gemeinsame Klassenhierarchie zu pressen. Die bessere Lösung ist das Erstellen einer Schnittstelle, in der die Methoden deklariert werden, die diese Klassen zur Kommunikation verwenden. Anschließend können Sie jede Klasse diese Methoden implementieren lassen, d. h. ihre jeweils eigenen Definitionen zur Verfügung stellen. Für eine erfolgreiche Programmierung sind in der Regel keine Schnittstellen erforderlich. Werden Schnittstellen sinnvoll eingesetzt, kann das Design Ihrer Anwendung effektiver, skalierbarer und leichter zu pflegen sein.

Schnittstellen – Definition

In der objektorientierten Programmierung sind Schnittstellen (engl. interfaces) mit Klassen vergleichbar, deren Methoden deklariert wurden, die aber sonst nichts anderes »tun«. Eine Schnittstelle setzt sich somit aus »leeren« Methoden zusammen. Eine andere Klasse kann die von der Schnittstelle deklarierten Methoden implementieren. Objektschnittstellen können auch als Sonderfälle von abstrakten Klassen gelten. Schnittstellen werden mit dem Schlüsselwort `interface`, gefolgt von einem Namen, deklariert und enthalten per Definition nur abstrakte Methoden. Auf die explizite Angabe von `abstract` bei Methoden kann verzichtet werden. Im Unterschied zu abstrakten Klassen werden Schnittstellen mit dem Schlüsselwort `implements` von einer Klasse implementiert.

Schnittstellen erstellen

Schnittstellen werden auf dieselbe Art und Weise erstellt wie Klassen. Schnittstellen deklarieren Sie mit dem Schlüsselwort `interface`. Darauf folgen der Name der Schnittstelle und dann die geschweiften Klammern, die den Körper der Schnittstelle definieren.

Innerhalb von Schnittstellen dürfen keine Eigenschaften enthalten sein, und von sämtlichen Methoden darf nur der »Kopf« geschrieben werden, direkt abgeschlossen mit einem Semikolon, statt der geschweiften Klammern.

Bei der Implementierung wird wie bei der Klassenvererbung vorgegangen, anstatt `extends` kommt jedoch das Schlüsselwort `implements` zum Einsatz.

```
<?php

interface einInterface {
   public function machWas();
```

```
}

interface anderesInterface {
   public function machWasAnderes();
}

class MeineKlasse implements einInterface, anderesInterface {
   public function machWas() {
        // ...
   }

   public function machWasAnderes() {
        // ...
   }
}

?>
```

Wie Sie sehen, kann eine Klasse eine beliebige Anzahl an Schnittstellen über das Schlüsselwort `implements` implementieren.

Da die Klasse `MeineKlasse` die Schnittstellen `einInterface` und `anderesInterface` implementiert, können Objekte dieser Klasse beispielsweise an Methoden übergeben werden, die als Parameter ein Objekt vom Typ `MeineKlasse`, `einInterface` oder `anderesInterface` erwarten, sämtliche dieser Typanforderungen kann ein solches Objekt erfüllen.

Abschließend habe ich noch ein vertiefendes praktisches Beispiel für Sie.

Beispiel

```
<?php

interface Warenkorb
{
   function ArtikelPlatzieren($artikel);
   function ArtikelEntfernen($artikel);
}

class Onlineshop implements Warenkorb
{
   private $bestellung = array();
   private $auftrag;

   function ArtikelPlatzieren($artikel)
   {
        array_push($this->bestellung, $artikel);
   }

   function ArtikelEntfernen($artikel)
   {
        if (in_array($artikel, $this->bestellung)) {
                $raus = array_search($artikel,$this->bestellung);

                unset($this->bestellung[$raus]);
        }
   }
```

```
   function Bestellen()
   {
        foreach ($this->bestellung as $key)
        {
                $auftrag .= $key . "\n";
        }
        return $auftrag;
   }
}

$kunde = new Onlineshop();
$kunde->ArtikelPlatzieren("Sony TV X100");
$kunde->ArtikelPlatzieren("Panasonic DVR");
$kunde->ArtikelPlatzieren("ActionScript Praxisbuch");
$kunde->ArtikelPlatzieren("5 Kilo Hanteln");

echo "Im Warenkob (nach Artikelplatzierung):\n" . $kunde->Bestellen() .
"\n";

$kunde->ArtikelEntfernen("Panasonic DVR");

echo "Im Warenkob (nach ArtikelEntfernen):\n" . $kunde->Bestellen() .
"\n";

?>
```

Ausgabe

```
Im Warenkob (nach Artikelplatzierung):
Sony TV X100
Panasonic DVR
ActionScript Praxisbuch
5 Kilo Hanteln

Im Warenkob (nach ArtikelEntfernen):
Sony TV X100
ActionScript Praxisbuch
5 Kilo Hanteln
```

5.1.11 Statische Eigenschaften und Methoden

Statische Eigenschaften und Methoden werden direkt von der Klasse aus angesprochen, und nicht über das Objekt. Damit teilen sich sämtliche Objekte einer Klasse diese Mitglieder, ganz gleich, ob es sich dabei um Eigenschaften oder Methoden handelt. Anstatt ein Objekt der Klasse zu erzeugen, um dann von diesem aus auf die Attribute und Methoden zugreifen zu können, kann immer direkt auf die Eigenschaften und Methoden der Klasse zugegriffen werden. Sollten Sie eine statische Eigenschaft oder Methode erzeugen wollen, verwenden Sie beim Deklarieren das Schlüsselwort `static`.

Einsatzfälle für statische Variablen und Methoden:

- Nutzen Sie statische Mitglieder für Aufgaben, die keinen Bezug zu den spezifischen Daten eines Objekts haben, beispielsweise Umrechnungen.
- Statische Mitglieder ermöglichen die Implementierung von Verweiszählern, wie beispielsweise Eigenschaften, die allen Objekten gleich sind.

- Statische Mitglieder ermöglichen den direkten Aufruf ohne vorherige Instanziierung. Dies ist sinnvoll, wenn man ohnehin nur ein Objekt benötigt.

```
<?php

class Klasse {
   static $static_var = 5;
   public $mein_prop = 'Hallo';

   public static function ausgabe() {
       return "Ein Text...";
   }

}

// Abfrage der statischen Variablen
echo Klasse::$static_var;

// Abfrage der statischen Funktion
echo Klasse::ausgabe();

$objekt = new Klasse;

// Abfrage der public-Variablen durch das Objekt
echo $objekt->mein_prop;

?>
```

Beispiel – Anzahl der erzeugten Objekte bzw. Entwickler

```
<?php
class Entwickler
{
   static $zaehler;

   public function __construct()
   {
        Entwickler::$zaehler++;
   }

   public function GetEntwicklerAnzahl()
   {
        return Entwickler::$zaehler;
   }
}
$entwickler1 = new Entwickler();
$entwickler1 = new Entwickler();
$entwickler1 = new Entwickler();

echo "Es existieren " . $entwickler1->GetEntwicklerAnzahl() . "
Entwickler.";
?>
```

Ausgabe

```
Es existieren 3 Entwickler.
```

Um auf einfache Weise eine Klasse zu schaffen, die in der Lage ist, die Anzahl ihrer erzeugten Objekte zu erfassen, ist das Schlüsselwort `static` Gold wert. Durch Einsatz von `static` ist die Variable `$counter` der Klasse `Entwickler` in den Klassenkontext

überführt worden. Diese Variable existiert für sämtliche Objekte lediglich einmal. Sobald ein neues Objekt erzeugt wird, wird der Wert im Konstruktor um 1 erhöht. Um dabei die Variablen `$counter` verarbeiten zu können, erfolgt der Zugriff über einen Klassenverweis, die Pseudovariable `$this` kann nicht verwendet werden. Dabei wird der Zugriff durch den statischen Verweisoperator `::` und den vollständigen Namen der Variablen, inklusive `$`-Zeichen, ermöglicht. Dies gilt sowohl für Zugriffe innerhalb der Klasse, wie im vorliegenden Beispiel, als auch für externe.

Hinweis: Statische Variablen und Methoden können darüber hinaus als `public`, `private` oder `protected` gekennzeichnet werden.

Beispiel – statische Methoden

```
<?php
class Datenbank
{
   // Variablendeklaration
   protected $db;

   // Statische Methode
   public static function verbindeDB($host, $user, $password)
   {
        @$db = new mysqli($host, $user, $password);

        if (mysqli_connect_errno()) {
               printf("Verbindungsfehler: %s\n", mysqli_connect_error());
               exit();
        }
        return $db;
   }
   public function oeffneDB()
   {
        $this->db = $this->verbindeDB("localhost", "root", "");
        return $this->db;
   }
}
// Methodenaufruf ohne Objekterzeugung
// Lediglich bei static-Methoden möglich!
$privatzugang_a = Datenbank::verbindeDB("localhost", "root", "");
echo $privatzugang_a;

// Aus einem Objekt heraus
$zugang = new Datenbank();
$privatzugang_b = $zugang->oeffneDB();
echo $privatzugang_b;
?>
```

5.1.12 Verweisoperator/Gültigkeitsbereichsoperator (::)

Der Gültigkeitsbereichsoperator, auch Verweisoperator genannt, ist ein Kürzel, das Zugriff auf statische, konstante und überschriebene Mitglieder einer Klasse ermöglicht.

Hinweis: Wenn solche Elemente außerhalb der Klassendefinition angesprochen werden sollen, muss der Name der Klasse verwendet werden.

Beispiel – :: außerhalb der Klassendefinition

```
<?php

class Flugzeug
{
    const MAX_FLUGHOEHE = 15000;
}

echo Flugzeug::MAX_FLUGHOEHE . " Fuss";

?>
```

Ausgabe

```
15000 Fuss
```

Schlüsselwörter self und parent

Die zwei speziellen Schlüsselwörter `self` und `parent` werden verwendet, um auf Mitglieder wie Eigenschaften und Methoden innerhalb einer Klassendefinition zuzugreifen. Die beiden Schlüsselwörter weisen folgende Besonderheiten auf:

- `self` – Kann in Klassen verwendet werden, um Bezug auf Methoden innerhalb derselben Klase zu nehmen.
- `parent` – Verweist auf die Klasse, von der mit `extends` geerbt wurde.

Beispiel – :: innerhalb der Klassendefinition

```
<?php

class Flugzeug
{
    const MAX_FLUGHOEHE = 15000;
}

class A320 extends Flugzeug
{
    public static $hoehen_toleranz = 3000;

    public static function fliegen() {
        echo "Flughöhe (max): " . parent::MAX_FLUGHOEHE . " Fuss\n";
        echo "Flughöhen (Toleranz): " . self::$hoehen_toleranz . "
Fuss\n";
    }
}

A320::fliegen();
?>
```

Ausgabe

```
Flughöhe (max): 15000 Fuss
Flughöhen (Toleranz): 3000 Fuss
```

Wenn eine abgeleitete Klasse die Definition der Methode einer Mutter überschreibt, wird PHP die Methode ihrer Mutter nicht aufrufen. Es obliegt der abgeleiteten Klasse, ob die Methode der Mutterklasse aufgerufen wird oder nicht. Dies gilt ebenfalls für Konstruktoren und Destruktoren, Überladung und magische Methodendefinitionen.

Beispiel – Methode einer übergeordneten Klasse aufrufen

```
<?php
class Lebewesen
{
    protected function erzeugen() {
        echo "Lebewesen::erzeugen()\n";
    }
}

class Menschen extends Lebewesen
{
    // Definition der übergeordneten Klasse überschreiben
    public function erzeugen()
    {
        // Trotzdem die Funktion der übergeordneten Klasse aufrufen
        parent::erzeugen();
        echo "Menschen::erzeugen()\n";
    }
}

$mensch = new Menschen();
$mensch->erzeugen();
?>
```

Ausgabe

```
Lebewesen::erzeugen()
Menschen::erzeugen()
```

5.1.13 Klassenkonstanten

Vor PHP 5 waren Konstanten immer global, was den Einsatz etwas problematisch machte, da Namenskonflikte fast schon vorprogrammiert waren. Da sich Konstanten von Objekt zu Objekt nicht ändern, verhalten sie sich wie statische Mitglieder und werden auch genau wie diese verwendet. Der einzige Unterschied besteht darin, dass sich der Inhalt nicht verändern lässt – außer während der Definition. Diese Definition erfolgt innerhalb der Klasse.

Im Gegensatz zu den globalen Konstanten, die mit der `define()`-Funktion erzeugt werden, erfolgt die Deklaration in Klassen mit dem Schlüsselwort `const`.

Der Zugriff auf eine solche Konstante erfolgt über *Klassenname::Konstante*. Soll der Zugriff auf die Konstante aus einer Methode derselben Klasse erfolgen, so kann auch *self::Konstante* verwendet werden.

Beispiel – const und static

```
<?php
class MeineKlasse {
```

```
   const Konstante = 16;
   static $statisch = 1;

   function erhoehen() {
        return self::$statisch++;
   }
}

$a = new MeineKlasse;
$b = new MeineKlasse;

echo "MeineKlasse::Konstante = " . MeineKlasse::Konstante . "<br>";
echo "\$a->erhoehen() = " . $a->erhoehen() . "<br>";
echo "\$b->erhoehen() = " . $b->erhoehen() . "<br>";

?>
```

Ausgabe

```
MeineKlasse::Konstante = 16
$a->erhoehen() = 1
$b->erhoehen() = 2
```

Beispiel – Physik

```
<?php

class Physik
{
   const ATOM_GEWICHT = 1.00895;
}

echo "Wert: " . Physik::ATOM_GEWICHT;

?>
```

Ausgabe

```
Wert: 1.00895
```

5.1.14 Objekte – Referenzen und Kopien

Das Sprachverhalten der Zend Engine sah es bisher vor, beim Anlegen oder bei der Übergabe von Objektinstanzen nicht das eigentliche Objekt zu übergeben, sondern lediglich eine Kopie davon, so wie man es von Variablen kennt. Dieses Verhalten wird als *copy by value* bezeichnet und sorgte oft für Verwirrung. Die eigentliche Problematik liegt darin, dass etwaige Änderungen der Eigenschaften des Objekts sich nicht auf das Objekt selbst auswirken würden, sondern auf die Kopie des Objekts. Um tatsächlich nur eine Referenz auf das Objekt anzuwenden, konnte man sich nur mit der expliziten Deklarierung des `&new`-Operators behelfen.

Die Verhaltensweise und Umgangsweise von Objekten orientiert sich strikt an Hochsprachen wie Java. Dort wird eine Objektinstanz mit einem *object handle* referenziert (*by reference*) und dieses Handle bezieht sich in allen Aktionen auf das Ursprungsobjekt. Der große Vorteil dabei ist, dass PHP 5 dies implizit erledigt, was nicht nur eleganter und flexibler ist, sondern auch spürbar die Performance steigert. Der konsequente Wechsel zum *object by reference*-Paradigma bietet aber auch Vorteile beim

Umgang in puncto Usability und Semantik, wodurch Features wie Destruktoren und Dereferenzierung profitieren.

```
<?php

class Fahrzeug
{
   function __construct($typ)
   {
        $this->name  = $typ;
   }
   function ProduziereFahrzeug($obj, $name)
   {
        $obj->name = $name;
   }
}

$mobil = new Fahrzeug("PKW");
$mobil->ProduziereFahrzeug($mobil, "LKW");
// Ausgabe - LKW
echo $mobil->name;

?>
```

5.1.15 Magische Methoden (Interzeptormethoden)

Die Interzeptormethoden oder auch magischen Methoden von PHP 5 werden automatisch beim Zugriff auf nicht bekannte Eigenschaften und Methoden eines Objekts, beim Versuch, ein Objekt einer nicht deklarierten Klasse zu erzeugen, sowie bei der Typumwandlung eines Objekts in einen String aufgerufen.

- `__autoload($className)` wird aufgerufen, wenn ein Objekt der Klasse `$className` erzeugt werden soll, die Klasse aber nicht deklariert ist.
- `__call($methodName, $parameters)` wird aufgerufen, wenn eine nicht deklarierte Methode `$methodName` mit einem Objekt aufgerufen wird. Der zweite Parameter `$parameters` enthält die Parameter des Methodenaufrufs.
- `__get($memberName)` wird aufgerufen, wenn lesend auf das Attribut `$memberName` eines Objekts zugegriffen wird, das Attribut aber nicht gesetzt ist.
- `__set($memberName, $value)` wird aufgerufen, wenn schreibend auf das Attribut `$memberName` eines Objekts zugegriffen wird und das Attribut vorher nicht gesetzt war. Der zweite Parameter `$value` enthält den Wert, mit dem das Attribut belegt werden soll.
- `__toString()` wird aufgerufen, wenn eine Typumwandlung eines Objekts in einen String durchgeführt werden soll.

Einsatz von __autoload()

Die global verfügbare `__autoload()`-Funktion kann verwendet werden, um eigentlich nicht definierte Klassen nachzuladen. Wird auf eine nicht definierte Klasse zugegriffen, wird, falls vorhanden, `__autoload()` aufgerufen und ausgeführt. Ist `__autoload()` ent-

sprechend implementiert, können Klassen bequem nachgeladen werden. Damit ist es möglich, Klassen zur Laufzeit erst einzubinden, wenn sie tatsächlich benötigt werden.

Das folgende Beispiel zeigt den einfachsten Fall der Verwendung von `__autoload()` und geht davon aus, dass alle Klassen in einer Datei deklariert sind, deren Name sich aus dem Namen der Klasse und der Dateiendung *.php* zusammensetzt.

Beispiel – einfacher Klassenlader mithilfe von _autoload()

```
<?php

function __autoload($klassenname)
{
   include_once("$klassenname.php");
}
$daten = new Kontakte($host, $nutzer, $passwort);
?>
```

In diesem Beispiel steht die Klasse `Kontakte` eigentlich nicht zur Verfügung, da sie weder definiert noch eingebunden ist. Beim Versuch, `Kontakte` zu instanziieren, wird `__autoload()` aufgerufen und die Klasse mit `include_once()` nachgeladen.

Hinweis: Ist die geforderte Klasse nach Ausführung der `__autoload()`-Funktion weiterhin unbekannt, so wird eine Fehlermeldung ausgegeben.

Einsatz von __call()

Die `__call()`-Funktion wird aufgerufen, wenn versucht wird, eine nicht deklarierte Methode mit einem Objekt aufzurufen. Mit der `__call()`-Methode ist es möglich, dem Funktionsaufruf Parameter an die `__call()`-Funktion zu übergeben. Der `__call()`-Funktion stehen dabei zwei Parameter zur Verfügung:

- Im ersten steht der Name der aufrufenden Methode zur Verfügung.
- Im zweiten Parameter stehen die übergebenen Werte als numerisches Array zur Verfügung.

Hinweis: Die `__call()`-Methode wird zum Überladen von Methoden verwendet. Klassenmethoden können überladen werden, um eigenen in Ihrer Klasse definierten Code auszuführen, indem man diese speziell benannte Methode definiert.

Beispiel

```
<?php

class Handies
{
   public $anzahl = 0;

   function __call($funktionsname, $parameter)
   {
        $this->anzahl = count($parameter);
        echo "Aufruf von $funktionsname mit $this->anzahl Parameter \n";
        if ($this->anzahl > 0) print_r($parameter);
   }
```

```
}

$test = new Handies();
$test->SetzeHersteller("Nokia","Siemens");
$test->SetzePreise(99.95, 199.99, 50);

?>
```

Ausgabe

```
Aufruf von SetzeHersteller mit 2 Parameter
Array
(
    [0] => Nokia
    [1] => Siemens
)
Aufruf von SetzePreise mit 3 Parameter
Array
(
    [0] => 99.95
    [1] => 199.99
    [2] => 50
)
```

Im folgenden Beispiel wird der Datentyp geprüft und anschließend an die passende Funktion übergeben!

```
<?php

// Datentyp prüfen via __call()
class Auswertung {

  function __call($eingabe,$inhalt) {

    if($eingabe=='pruefen') {

      if(is_integer($inhalt[0]))
           $this->ausgabe_integer($inhalt[0]);

      if(is_string($inhalt[0]))
           $this->ausgabe_string($inhalt[0]);

      if(is_array($inhalt[0]))
           $this->ausgabe_array($inhalt[0]);

    }

  }

  private function ausgabe_integer($daten) {
    echo("Der Wert " . $daten . " ist ein Integer!<br>");
  }

  private function ausgabe_string($daten) {
    echo("Der Wert " . $daten . " ist ein String!<br>");
  }

  private function ausgabe_array($daten) {
    echo("Die Werte " . implode(",", $daten) . " sind in einem
Array!<br>");
  }
```

```
}

// Klassenaufruf
$test = new Auswertung();

$test->pruefen(3);
$test->pruefen("3");

$array = array(10,20,30);
$test->pruefen($array);
?>
```

Ausgabe

```
Der Wert 3 ist ein Integer!
Der Wert 3 ist ein String!
Die Werte 10,20,30 sind in einem Array!
```

Einsatz von __set() und __get()

Eine weitere Variante der `__call()`-Methode sind `__set()` und `__get()`. Mit ihnen kann man direkt beim Aufruf die Werte beeinflussen. Die beiden Methoden sind spezielle Methoden, auf die von außerhalb der Klasse als Attribute zugegriffen werden kann, welche jedoch in der Klasse selbst als Methoden definiert vorliegen. Einer der wichtigsten Vorteile der Methoden ist, dass sie Eigenschaften erzeugen können, die von außerhalb wie Attribute erscheinen, die intern jedoch mit komplexen Abläufen arbeiten.

Funktionsweise von __set() und __get()

Wird auf Eigenschaften eines Objekts zugegriffen, die nicht explizit definiert sind, wird die `__set()`-Methode aufgerufen, um einen Wert zu definieren. Soll dieser Wert abgefragt werden, wird die `__get()`-Methode aufgerufen. Sind weder `__set()` noch `__get()` implementiert, kommt es bei einem Zugriff auf nicht definierte Eigenschaften zu Fehlern.

Eine Besonderheit stellt folgendes Verhalten der beiden Methoden dar:

- Wenn man mit `__set()` eine Eigenschaft definiert und mit `__get()` nicht, erhält man eine »Nur-Schreib«-Eigenschaft.
- Wenn man mit `__get()` eine Eigenschaft definiert und mit `__set()` nicht, erhält man eine »Nur-Lese«-Eigenschaft.

Hinweis: Die `__get()`/`__set()`-Methoden werden zum Überladen von Mitgliedern verwendet. Klassenmitglieder können überladen werden, um eigenen in ihrer Klasse definierten Code auszuführen, indem man diese speziell benannten Methoden definiert.

Beispiel

```
<?php

class Handies
{
   private $mobils = array(
   "Nokia" => 10,
   "Siemens" => 20
```

```
   );

   public function __get($varname)
   {
        return $this->mobils[$varname];
   }

   public function __set($varname, $wert)
   {
        $this->mobils[$varname] = $wert;
   }
}

$handy = new Handies();

$handy->Nokia++;
$handy->Siemens++;

echo "Wert von Nokia: " . $handy->Nokia . "\n";
echo "Wert von Siemens: " . $handy->Siemens . "\n";

?>
```

Ausgabe

```
Wert von Nokia: 11
Wert von Siemens: 21
```

Einsatz von __toString()

Die neue `__toString()` Methode ermöglicht es Ihnen, eine Typumwandlung vorzunehmen und ein Objekt in einen String umzuwandeln.

Verfügt eine Klasse in PHP 5 über eine `__toString()`-Methode, so wird sie aufgerufen, wenn ein Objekt der Klasse in einen String umgewandelt werden soll. So gibt das folgende Beispiel »Der Kontostand beträgt 9999.95 Euro.« aus anstatt `Object id #1`. Letzteres wäre der Fall, wenn die Klasse `Bankkonto` über keine `__toString()`-Methodendeklaration verfügen würde.

```
<?php

class Bankkonto {
   private $guthaben = 9999.95;

   public function __toString() {
        return sprintf('Der Kontostand beträgt %01.2f Euro.',$this-
>guthaben);
   }
}

$meinKonto = new Bankkonto;
print $meinKonto;

?>
```

Ausgabe

```
Der Kontostand beträgt 9999.95 Euro.
```

Stichwortverzeichnis

A

E

F

G

L

M

P

T

U

Matthias Kannengiesser

PHP 5
MySQL 5

PHP und MySQL – das ist das bewährte Erfolgsduo für dynamische Internetanwendungen aller Art. Erfolgsautor Matthias Kannengiesser vermittelt in dieser aktualisierten Version seines Standardwerks das Profiwissen, das Sie für die anspruchsvolle Webprogrammierung brauchen. Ob objektorientierte Programmierung mit PHP, das Arbeiten mit Datenbanktabellen und Verzeichnissen oder komplexe Anwendungen wie Webshops oder Foren – Matthias Kannengiesser kennt die Lösung.

Alle PHP-Neuheiten

PHP 5.3 ist die umfangreichste Aktualisierung der Sprache seit der Einführung von PHP 5. Die neu eingeführten Befehle unterstützen vor allem die Entwicklung komplexer Webanwendungen. Eine besonders wichtige Neuerung sind die Namensräume (Namespaces), die dabei helfen, auch umfangreiche Programme einfach und klar zu strukturieren. Die neu eingeführten Lambda-Funktionen und Closures unterstützen darüber hinaus die Entwicklung von sicherem und zuverlässigem Code und erleichtern das Finden von Fehlern erheblich. Matthias Kannengiesser zeigt, wie Sie diese neu eingeführten Features sinnvoll und produktiv einsetzen.

Echtes PHP-Profiwissen

Der Grund für den Erfolg der Skriptsprache PHP ist ihre Einfachheit. Der Autor demonstriert in diesem Buch, wie Sie eine PHP-Umgebung auf Ihrem PC einrichten und mit einfachen Praxisbeispielen schnelle Erfolgserlebnisse erzielen. Darauf aufbauend, vermittelt er das Profiwissen, das Entwickler für komplexe Webanwendungen brauchen. So nehmen zum Beispiel die objektorientierten Features von PHP sowie die Fehlerbehandlung und -vermeidung breiten Raum ein.

Mit Datenbanken umgehen

Die Grundlage jeder dynamischen Webanwendung ist eine Datenbank, MySQL ist hier der Standard. Dieses Buch zeigt, wie Sie mit MySQL Datenbanklösungen programmieren, die zu Ihren Anforderungen passen. Sie erfahren, wie Sie Datenmodelle und Datenbanktabellen erstellen, Daten einfügen, auslesen, aktualisieren und wieder löschen können. Matthias Kannengiesser zeigt außerdem, wie Sie Ihre Datenbank von PHP aus ansprechen und die gespeicherten Informationen auf Ihrer Website jederzeit korrekt darstellen.

Aus dem Inhalt:

Über den Autor:

Matthias Kannengiesser ist Diplominformatiker und arbeitet als IT-Projektmanager, -Consultant und -Fachdozent für namhafte Unternehmen im In- und Ausland. Seit mehr als zehn Jahren hält er Seminare, Workshops und Vorträge zu den Themen PHP und Datenbankprogrammierung, ActionScript, Flash, Lingo und JavaScript und schreibt erfolgreiche Bücher und Magazinbeiträge zu diesen Themen. Er lebt in San Jose im US-Bundesstaat Kalifornien.

Auf CD-ROM:

- PHP/MySQL-Umgebung XAMPP 1.7.1 für Windows, Linux und Mac OS X
- PHP-Entwicklungsplattform Aptana Studio 2.0 für Windows, Linux und Mac OS X
- MySQL Workbench für Windows, Linux und Mac OS X
- Zahlreiche Bonuskapitel, unter anderem eine umfangreiche MySQL-Referenz
- Sämtliche Programmierbeispiele

9 783645 600101

30,– EUR [D]

ISBN 978-3-645-60010-1

Besuchen Sie unsere Website
www.franzis.de