12 Objektorientiert programmieren in PHP 5

In den vergangenen beiden Kapiteln haben wir erläutert, wie Sie sich Objekte, Klassen, Methoden und Attribute vorstellen können und wie Sie Ihre Klassen und Objekte für Ihre Webanwendung entwickeln.

Wie Sie Ihre Klassen sowie deren Methoden und Attribute in PHP darstellen, erfahren Sie in diesem Kapitel. Ebenfalls erklären wir hier, wie Sie den Zugriff auf Methoden und Attribute unterschiedlich gegen unerwünschten Zugriff schützen und Beziehungen zwischen Objekten in PHP darstellen können.

12.1 Die wichtigsten objektorientierten Sprachelemente

Wir gehen in unserem Buch auf einige PHP-Sprachelemente ein, mit denen Sie objektorientiert programmieren können. Hier wollen wir mit den wichtigsten Möglichkeiten zur Abbildung von Klassen in PHP beginnen.

12.1.1 Klassen definieren

Klassen werden mit class dargestellt

Eine Klasse definieren wir in PHP mit dem Sprachelement `class`. Entsprechend gängiger Kodierungsstandards empfehlen wir, den Klassennamen großzuschreiben, wie beispielsweise bei der folgenden Klasse `Artikel`:

```
class Artikel
{
    // Hier werden Attribute und Methoden der Klasse beschrieben
}
```

Pro Klasse eine Datei

Der Einfachheit und Eindeutigkeit halber sollten Sie jede Klasse in einer separaten Datei abspeichern. Dabei schlagen wir vor, dass Sie der Datei den gleichen Namen wie der programmierten Klasse geben und dabei die Dateinamen großschreiben.

Bei eingebundenen Dateien keine Leerzeilen außerhalb der Skript-Tags

Bei der Speicherung der Klassen müssen Sie darauf achten, dass der Klassencode außerhalb der Skript-Tags `<?php` und `?>` keine Zeilen, auch keine Leerzeilen, enthält. Diese Zeilen können bei Seitenaufrufen (so genannten Header-Aufrufen) zu Fehlern im Programm führen, da sie vom PHP-Interpreter als bereits ausgegebener HTML-Code interpretiert werden.

Einbinden von Klassen mit require_once

Sie haben eine Klasse erstellt und in einer separaten Datei abgespeichert. Der Bauplan für ein Artikelobjekt existiert nun. Erstellen Sie jetzt eine neue PHP-Datei, in der Sie zuerst die `Artikel`-Klasse mit `require_once` bekannt machen müssen.

Mit new ein neues Objekt erzeugen

Sie können jetzt mit Hilfe des `new`-Operators in der neuen Datei ein neues Objekt erzeugen, d. h. ein Objekt nach dem Baumuster oder dem Bauplan der `Artikel`-Klasse. Das neu erzeugte Artikelobjekt wird dabei im System gespeichert und die Referenz auf das neue Objekt in der gleichen Zeile der Variablen `$Artikel` zugewiesen. Über die Variable `$Artikel` haben Sie dann im weiteren Verlauf Zugriff auf das Objekt.

```
<?php
// Bauplan einbinden
require_once 'Artikel.php';

// Objekt Artikel erzeugen
$Artikel = new Artikel();
?>
```

Erzeugen eines Objektes

Wenn Sie mit `new` ein neues `Artikel`-Objekt erzeugen, spricht man auch vom Instanziieren eines Artikels oder im Falle des erzeugten Artikelobjektes von einer Artikelinstanz.

Das Artikelobjekt hat alle Eigenschaften, die in der Klasse `Artikel` definiert wurden. Eine Variable für Objekte wird am besten großgeschrieben – da andere Variablen in der Regel kleingeschrieben werden, erkennen Sie so die großgeschriebene Variable sofort als eine Objektvariable.

Ein Objekt wird somit immer nach folgendem Muster erzeugt (und zwar entsprechend dem Bauplan der angegebenen Klasse):

```
$Variablenname = new Klassenname();
```

Bitte beachten Sie, dass Objekte nur bis zum Aufruf einer neuen Internetseite existieren. In der Beschreibung des Programmierbeispiels auf den Webseiten zum Buch (`http://www.dpunkt.de/php5`) erhalten Sie einige Hinweise, wie Sie Objekte auch über einen Seitenwechsel hinweg retten können.

12.1.2 Attribute verwenden

Attribute werden mit »var« dargestellt

Sie wissen nun, wie Sie Ihre Klassen in PHP darstellen können. Nun wollen wir für eine Klasse `Link` verschiedene Attribute definieren. Attribute werden in PHP-Klassen als Klassenvariablen – Variablen einer Klasse – dargestellt. Dabei wird vor jeder Klassenvariablen das Sprachelement `var` vorangestellt.

```
class Link
{
    // ...
    // Variablendeklaration und -definition
    var $linkId = 33;
    var $objektId = null;
    var $parameter;
    // ...
}
```

Variablendeklaration und -definition sind gleichzeitig möglich

Im obigen Beispiel haben wir drei Attribute für die Klasse `Link` definiert und deklariert. Die Variablen können Sie jeweils mit einem Initialisierungswert definieren – zum Beispiel `33` für `$linkId` und `null` für `$objektId`. Damit haben Sie für die deklarierten Klassenattribute einen Anfangswert – `33` bzw. `null` – definiert. Das Attribut `$parameter` hat einen undefinierten Initialisierungswert.

Wenn Sie ein Objekt erzeugen, dann sind die Attribute bereits mit diesem Initialisierungswert voreingestellt. Die Attribute schreiben Sie am besten immer klein. Setzt sich der Attributname aus mehreren Wörtern zusammen, dann beginnen Sie das Folgewort mit einem Großbuchstaben und hängen es direkt an das erste Wort an.

Zugriff auf Klassenelemente mit »->«

Auf die Attribute einer Klasse können Sie mit dem Klassenoperator `->` zugreifen.

```
<?php
require_once 'Link.php';
// ...
// Objekt Link erzeugen
$Link = new Link();
// ...
// Die Variable linkId des Objektes in $Link ausgeben
echo $Link->linkId;
// ...
?>
```

Nachdem wir die Klasse `Link` mit `require_once` im aktuellen Programm bekannt gemacht haben, können wir das Linkobjekt mit `new` erzeugen. Ab diesem Zeitpunkt können wir auf die Attribute der `Link`-Klasse in der oben dargestellten Art und Weise zugreifen. Unmittelbar hinter der

Variablen `$Link` folgt also das Sprachelement `->` und danach das Attribut der Klasse, auf welches wir zugreifen wollen. Im obigen Beispiel greifen wir also auf das Attribut `linkId` des Linkobjektes, das über die Referenz in `$Link` ansprechbar ist, lesend zu.

Wenn Sie auf ein Attribut des Objektes schreibend zugreifen, dann ändern Sie damit den Objektzustand.

```
$Link->linkId = 3;
```

Vorsicht beim Umgang mit Public-Attributen!

Wenn Sie den Zugriff auf Attribute uneingeschränkt ermöglichen wollen, dann reicht die Deklaration der Klassenvariablen wie oben beschrieben aus. Ein solches Attribut ist dann public (öffentlich). Public-Zugriffe sollten Sie jedoch weitgehend vermeiden. Wenn Sie auf die Attribute nur über klasseneigene Methoden zugreifen, dann haben Sie die gesamten Klassendaten besser gekapselt und somit besser vor unerwünschten Zugriffen geschützt. Wie Sie den Klassenschutz in PHP umsetzen können, erläutern wir später in diesem Kapitel.

12.1.3 Methoden schreiben

Methoden mit »function« definieren

Mit den Methoden können Sie auf die Attribute einer Klasse zugreifen und den Zustand von Objekten ändern. Methoden werden in PHP 5 mit dem Sprachelement `function` beschrieben. Entsprechend den Funktionen beim prozeduralen Programmieren können Sie bei einer Methode auch verschiedene Parameter übergeben. Die Methode einer Klasse kann jedoch ähnlich den Attributen immer nur in Verbindung mit der Klasse bzw. dessen Objekt aufgerufen werden.

Eine Methode wird üblicherweise immer nach den Attributen innerhalb der entsprechenden Klasse platziert. Wir empfehlen, dass Sie den Methodennamen nach Möglichkeit mit einem kleingeschriebenen Verb beginnen. Beispielsweise können Sie die Wörter eines längeren Methodennamens optisch voneinander trennen, indem Sie alle Wörter nach dem ersten Wort mit einem großen Buchstaben beginnen – zwischen den Wörtern ist dabei jedoch kein Leerzeichen. Zum Beispiel:

Listing 12-1 *Auszug aus Mitarbeiter.php*

```
<?php
class Mitarbeiter
{
    // Variablendefinition und -deklaration
    var $personalNummer = 33;
    var $gehalt         = 1000;
    // ...
    function getPersonalNummer()
    {
        return $this->personalNummer;
    }
```

```
    function setPersonalNummer($parameter)
    {
        $this->personalNummer = $parameter;
    }

    // ...

    function getGehalt()
    {
        return $this->gehalt;
    }

    // ...

    function getGehaltPlus($bonus)
    {
        return $this->getGehalt() + $bonus;
    }
    // ...
}
?>
```

Objektzustand mit set- und get-Methoden ändern

Üblicherweise werden für jedes Attribut zwei Methoden geschrieben. Die `set`-Methode benutzen Sie zum Ändern eines Attributinhaltes. Die `get`-Methode können Sie zum Auslesen eines Attributes verwenden.

In den `set`- und `get`-Methoden können Sie die Attributwerte auch zusätzlich modifizieren, bevor das Attribut überschrieben bzw. ausgegeben wird. In der Methode `getGehaltPlus` wird z. B. zu dem Attributwert von `gehalt` vor der Rückgabe des Wertes noch der Inhalt des Methodenparameters `$bonus` hinzuaddiert.

Ähnlich wie bei den Attributen eines in einer Variablen gespeicherten Objektes können Sie auf die `public`-Methoden mit Hilfe des Sprachelements `->` zugreifen. Dabei müssen Sie den Operator `->` direkt nach der Objektvariablen platzieren, unmittelbar gefolgt von der Methode, ggf. mit Angabe von Parametern.

Listing 12-2
Anwendung von Mitarbeiter.php (Auszug)

```
<?php
require_once 'Mitarbeiter.php';
// ...
// Objekt Mitarbeiter erzeugen
$Mitarbeiter = new Mitarbeiter();
// ...
// Die Variable personalNummer des Objektes in $Mitarbeiter ausgeben
echo $Mitarbeiter->getPersonalNummer();
// Zum Gehalt zuerst 2 addieren, dann ausgeben
echo $Mitarbeiter->getGehaltPlus(2);
// ...
?>
```

Nachdem wir die `Mitarbeiter`-Klasse eingebunden und das Mitarbeiterobjekt erzeugt haben (und gleichzeitig über die Referenz in der Variablen `$Mitarbeiter` zugreifbar machen), holen wir über die Methode `getPersonalNummer()` das Attribut `personalNummer` und mit der Methode `getGehaltPlus` das um 2 erhöhte Gehalt aus dem Mitarbeiterobjekt.

Mit »$this« auf objekteigene Attribute und Methoden zugreifen

Wie Sie auf Attribute und Methoden eines Objektes über eine Variable zugreifen können, wissen Sie jetzt. Wenn Sie innerhalb der Methode einer Klasse auf ein Attribut oder eine andere Methode des eigenen Objektes zugreifen wollen, dann benötigen Sie ein weiteres PHP-Sprachkonstrukt. Innerhalb einer Methode können Sie mit dem Sprachelement `$this` auf objekteigene Attribute und Methoden zugreifen.

Jedes Objekt hat seine eigenen Attribute und Methoden, immer gemäß dem in der Klasse definierten Baumuster. Sollten Sie mehrere Objekte erzeugt haben, gilt `$this` immer nur für das jeweils eigene Objekt. Mit `$this->personalNummer` sprechen Sie also das Attribut `$personalNummer` im eigenen Objekt an. Mit `$this->getPersonalNummer()` rufen Sie die entsprechende Methode innerhalb des eigenen Objektes auf.

```
class Mitarbeiter
{
    // ...
    // Variablendefinition und -deklaration
    var $personalNummer = 33;
    // ...
    function getPersonalNummer()
    {
        return $this->personalNummer;
    }
    // ...
}
```

12.1.4 Den Konstruktor eines Objekts anwenden

Sie wollen bereits bei der Erzeugung eines Objektes dieses in einen von Ihnen definierten Zustand bringen? Mit dem Konstruktor – einer speziellen Methode – ist dies möglich.

Listing 12-3 *Auszug aus Link.php*

```
class Link
{
    // ...
    // Variablendefinition und -deklaration
    var $linkId = 33;
    // ...
    // Konstruktor
    function __construct($neuerLink)
```

```
    {
        $this->setLinkId($neuerLink);
    }

    // alle anderen Methoden
    function setLinkId($parameter)
    {
        $this->linkId = $parameter;
        // ggf. weitere Anweisungen, welche bei der
        // Objekterzeugung durchgeführt werden sollen
        // ...
    }

    function getLinkId()
    {
        return $this->linkId;
    }
    // ...
}
```

Der Konstruktor muss immer »__construct()« heißen

Um den Konstruktor im Code zu den einzelnen Klassen immer leicht und schnell zu finden, sollten Sie diesen als allererste Methode vor allen anderen Methoden definieren. Der Konstruktor heißt immer:

```
__construct()
```

Beachten Sie dabei die zwei Unterstriche vor `construct()`.

Konstruktoren werden bei der Objekterzeugung automatisch aufgerufen

Der Konstruktor wird, falls vorhanden, automatisch bei der Erzeugung des Objektes mittels `new` von PHP aufgerufen. Damit können Sie den Objektzustand direkt beim Erzeugen Ihres Objektes definieren. Sie brauchen die »Methode« `__construct()` nicht selbst aufzurufen, dies erledigt PHP beim Erzeugen des Objektes für Sie. Im obigen Beispiel setzt der Konstruktor das Attribut `$linkId` auf den neuen Wert `$neuerLink`. Damit wird der bei der Variablendeklaration definierte Wert `33` überschrieben. Folgender Code verdeutlicht die Funktion des Konstruktors:

Listing 12-4
Anwendung von Mitarbeiter.php (Auszug)

```
<?php
require_once 'Link.php';
// ...
// Objekt Link erzeugen
$Link = new Link(77);      // incl. Aufruf des Konstruktors!
// ...
// Die Variable linkId des Objektes in $Link ausgeben
echo $Link->getLinkId();   // = 77 und nicht 33!
// ...
?>
```

Ohne den Konstruktor würde das PHP-Skript den Wert `33` ausgeben. Mit der Methode `getLinkId()` wird der aktuelle vom Konstruktor bereits überschriebene Attributwert zurückgegeben (also `77`). Wenn Sie

Der Konstruktor kann auch weggelassen werden

keine Zustandsänderung bei der Erzeugung des Objektes benötigen, dann können Sie den Konstruktor auch weglassen – der Code funktioniert auch ohne Konstruktor.

Auch bei Konstruktor mehrere Parameter möglich

Sie können den Konstruktor sowohl ohne Parameter als auch mit Parameter definieren. Bei der Objekterzeugung müssen Sie diese Parameter in der Klammer hinter der Klasse entsprechend mit angeben. Beispielsweise übergeben Sie mit `$Link = new Link(77)` dem Link-Konstruktor die Zahl 77 – in der Klasse `Link` würde der Konstruktor dann `function __construct($parameter)` lauten.

In PHP 4 heißen Konstruktoren anders!

In PHP 4 hatte der Konstruktor den gleichen Namen wie die Klasse. Der Konstruktor hätte in obigem Beispiel also `function Link()` geheißen (da die Klasse auch Link heißt). Auch in PHP 4 wurde der Konstruktor bei der Objekterzeugung automatisch aufgerufen.

12.1.5 Den Destruktor eines Objekts nutzen

Mit dem »Destruktor« Objekte in einen definierten Endzustand versetzen

Früher oder später werden Sie sich eine Methode wünschen, die von PHP – ähnlich dem Konstruktor – beim Löschen eines Objektes automatisch aufgerufen wird. Diese Methode stellt Ihnen PHP mit dem so genannten Destruktor zur Verfügung.

Beim Löschen eines Objektes können Sie mit Hilfe des Destruktors das Objekt in einen definierten Endzustand versetzen. Zum Beispiel können Sie so bestehende Datenbankverbindungen abbrechen oder Objekte speichern. Der Destruktor wird beim Löschen eines Objektes immer automatisch aufgerufen. Sie brauchen sich also um den Aufruf der »Methode« `Destruktor` nicht zu kümmern.

Damit der Destruktor im Code zu den einzelnen Klassen immer leicht und schnell gefunden werden kann, sollte dieser direkt nach dem Konstruktor definiert werden – alternativ nach allen anderen Methoden.

```
class Datenbankzugriff
{
    // ...
    // Variablendefinition und -deklaration
    var $database = null;
    // ...
    // Konstruktor
    function __construct()
    {
        // Datenbankverbindung herstellen
        $this->database =
            DB::connect("mysql://user:passwort@host/dbname);
    }

    // Destruktor
```

```
    function __destruct()
    {
        // Datenbankverbindung schließen
        $this->database->close();
    }

    // weitere Methoden
}
```

Der Destruktor muss immer »__destruct« heißen

Mit dem Konstruktor im obigen Beispiel wird eine Verbindung zum Datenbankserver aufgebaut, sobald ein Objekt gemäß der Klasse `Datenbankzugriff` erzeugt wird. Wenn das Objekt gelöscht werden soll, wird vor dem endgültigen Löschen der Destruktor aufgerufen, in welchem die Datenbankverbindung wieder unterbrochen wird. Der Destruktor muss immer `__destruct` heißen. Vor dem Methodennamen `destruct` stehen zwei Unterstriche.

Vergleich mit PHP 4

In PHP 4 gab es noch keinen Destruktor. Diese Funktionalität wurde erst mit dem Erscheinen der Version 5 von PHP hinzugefügt.

Der Destruktor ist eine Kann-Methode

In PHP 5 kann der Destruktor auch weggelassen werden. Sie müssen den Destruktor nur definieren, wenn Sie beim Löschen des Objektes verschiedene Abläufe sicherstellen wollen.

12.1.6 Beziehungen in Klassen darstellen

Sie haben nun erfahren, wie Sie eine Klasse und deren Attribute und Methoden in PHP 5 darstellen können. Verschiedene Klassen können miteinander in Beziehung stehen. Wie Sie diese Beziehungen zwischen zwei Klassen in PHP 5 abbilden können, zeigen wir Ihnen im Folgenden.

1:n-Beziehungen abbilden

In unserer Beispielanwendung gibt es eine 1:n-Beziehung zwischen den Klassen `Artikel` und `Artikelgruppe`: Sie können einen `Artikel` zu genau einer `Artikelgruppe` zuordnen, einer Artikelgruppe können Sie jedoch mehrere Artikel zuordnen.

Wie wäre es, wenn Sie der Artikelgruppe sagen könnten, dass sie den Preis aller dazugehörenden Artikel um einen bestimmten Betrag erhöhen sollte? Das schaffen Sie nur, wenn die Artikelgruppe weiß, welche Artikel zur Gruppe gehören.

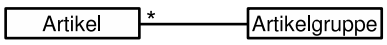


Abbildung 12.1
Direkte Beziehungen zwischen zwei Klassen

Diese 1:n-Beziehung können Sie mit Hilfe einer Liste in PHP in der `Artikel`-Klasse abbilden. Dazu definieren Sie in der Klasse `Artikel` ein-

fach ein Hilfsattribut in der Form einer Liste, in der Sie alle Artikelgruppenobjekte (oder deren Ids) ablegen können. Die Liste können Sie als Array oder in Objektform definieren. Mit einer Methode `linkObject()` können Sie die Beziehungsliste ändern. An folgendem Beispiel erkennen Sie, wie Sie eine Liste als Array umsetzen können.

Listing 12-5 *Beziehung zwischen Artikel und Artikelgruppe*

```
<?php
class Artikel
{
  // ...
  var $listeArtikelgruppe = array();
  // ..
  function linkObject($Artikelgruppe)
  {
     // Artikelgruppenobjekt in Liste einfügen
     $listeArtikelgruppe[] = $Artikelgruppe;
  }
  // ...
}

class Artikelgruppe{}

$Artikel = new Artikel();
$Artikelgruppe = new Artikelgruppe();
$Artikel->linkObject($Artikelgruppe);
?>
```

Bei der Methode `linkObject()` sollten Sie außerdem prüfen, ob die Artikelgruppe bereits in der Liste vorhanden ist.

Wenn Sie eine Beziehung zwischen zwei Objekten löschen wollen, dann brauchen Sie lediglich eine Methode `deleteLink()` zu schreiben. Dieser Methode übergeben Sie das Objekt, zu dem Sie die Beziehung löschen wollen. In der Methode können Sie dann das als Parameter übergebene Objekt in der Beziehungsliste suchen und löschen.

Im Kapitel 11 zur Objektmodellierung ab Seite 161 können Sie Weiteres zur Klassenmodellierung und deren grafischen Darstellung lesen. An dieser Stelle wollen wir noch darauf aufmerksam machen, dass Sie die Beziehungsliste immer auf der n-Seite definieren sollten.

Sie sind damit prinzipiell in der Lage, eine 1:n-Beziehung zwischen Objekten in PHP abzubilden. Eine elegantere Darstellung von Beziehungen im Zusammenhang mit Datenbanken stellen wir Ihnen bei der Beschreibung der Beispielanwendung auf den Webseiten zum Buch vor. Für unsere Beispielanwendung bilden wir die Beziehungen auf einer Datenbank ab. Dabei stellen wir mit Hilfe der `DB_DataObject`-Klasse jede einzelne Tabelle der Datenbank dar. Da sich die Beziehung zwischen zwei Klassen in den unterschiedlichen Tabellen widerspiegelt, wird die-

se Beziehung auch in den entsprechenden Klassen mit DB_DataObject-Funktionalität abgebildet.

Wie Sie eine 1:n-Beziehung einfacher mit Hilfe der DB_DataObject-Klassen aus PEAR bearbeiten können, erfahren Sie bei der Beschreibung der Beispielanwendung auf den Webseiten zum Buch.

n:m-Beziehungen abbilden

Die Beziehung zwischen einer Farbe und einem Artikel in unserer Beispielanwendung ist eine n:m-Beziehung: Ein Artikel ist in mehreren Farben lieferbar und in einer bestimmten Farbe gibt es mehrere unterschiedliche Artikel.

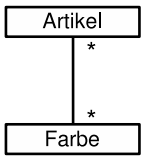


Abbildung 12.2
n:m-Beziehung zwischen zwei Klassen

Rein programmiertechnisch gesehen, können Sie diese n:m-Beziehung wieder mit Hilfe einer Liste (als Array oder in Objektform) darstellen.

Dafür müssen Sie sowohl in der Klasse Artikel als auch in der Klasse Farbe eine Beziehungsliste definieren. In beiden Klassen benötigen Sie wieder die linkObject-Methode, um die Beziehungsliste zu ändern. Wollten Sie jetzt eine n:m-Beziehung hinzufügen, dann müssen Sie über die Methode linkObject() nicht nur das Farbobjekt in die Liste des eigenen Artikelobjektes, sondern auch das eigene Artikelobjekt in die Liste des anderen Farbobjektes einklinken. Folgendes Beispiel mit den beiden miteinander in Beziehung stehenden Klassen Artikel und Farbe verdeutlicht dies:

Listing 12-6
Auszug aus Artikel.php

```
class Artikel
{
  var $Liste = array();
  // ...
  function linkObject($Farbe)
  {
     // Farbobjekt in Liste einfügen
     $Liste[] = $Farbe;

     // aktuelles Artikelobjekt in die Liste der Farben einfügen
     $Farbe->linkObject($this);
  }
  // ...
}
```

Damit können Sie n:m-Beziehungen in PHP abbilden. Wie bei den 1:n-Beziehungen gibt es auch bei den n:m-Beziehungen die Möglichkeit, die Objekte und deren Beziehungen auf einer Datenbank abzulegen und die gesamte Struktur mit Hilfe der `DB_DataObject`-Klassen darzustellen. Für jede Tabelle gibt es dabei wieder ein separates Objekt. Bei der n:m-Beziehung ist dabei jedoch notwendig, dass Sie eine Hilfstabelle definieren. Die Hilfstabelle reduziert die Komplexität der n:m-Beziehung auf zwei einfache 1:n-Beziehungen. Die Abbildung 12.3 zeigt Ihnen die vereinfachte Darstellung der n:m-Beziehung.

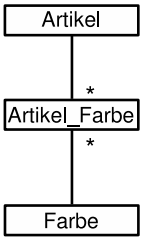


Abbildung 12.3
Aufgelöste n:m-Beziehung zwischen zwei Klassen

Die Umsetzung der n:m-Beziehung können Sie mit Hilfe dieser Vereinfachung mit den bereits bekannten Tipps aus der 1:n-Beziehung in PHP darstellen.

Wenn Sie nun für einen Artikel die lieferbaren Farben herausfinden wollen, dann müssen Sie eine umfangreiche Programmierarbeit leisten und über die Hilfstabelle entsprechende Farben abfragen. Diese Programmierarbeit sollten Sie im Artikelobjekt verstecken. Mit einer Methode `getFarben` können Sie eine Liste aller lieferbaren Farben erstellen und zurückgeben.

Wie Sie eine 1:m-Beziehung einfacher mit Hilfe der PEAR-Klasse `DB_DataObject` bearbeiten können, erfahren Sie in der Beschreibung zum Beispielprogramm auf den Webseiten zum Buch.

Über solche speziellen Methoden können Sie also die n:m-Beziehungen weiter vereinfachen. Die spezielle Methode fragt dabei über die Hilfsklasse alle Objekte ab, die indirekt mit dem eigenen Objekt in Beziehung stehen. Die Frage, in welcher Klasse diese spezielle Methode am besten platziert wird, hängt vom Anwendungsfall ab. Will man z. B. wissen, wie viele Artikel einer Farbe existieren, dann platziert man die spezielle Methode `getFarben` in der `Farb`-Klasse. Will man hingegen eher wissen, in welchen Farben ein Artikel lieferbar ist, dann sollte eine solche Methode in der `Artikel`-Klasse stehen.

12.2 Vererbungsmechanismen

Stellen Sie sich vor, Sie hätten viele Klassen, die zwar vieles gemeinsam haben, jedoch im Detail sehr unterschiedlich sein können. Das riecht nach viel Arbeit und doppelt geschriebenem Code. Das muss nicht sein. Mit der objektorientierten Programmierung können Sie gezielt Klasseneigenschaften von der einen Klasse auf die andere Klasse »vererben«. Wie das geht und wie Sie die Vererbung gezielt beeinflussen können, beschreiben wir hier.

12.2.1 Einfachvererbung

Klasseneigenschaften können vererbt werden

In PHP können Klassen, wie in anderen objektorientierten Sprachen auch, verschiedene Eigenschaften (Attribute und Methoden) von anderen Klassen erben. In PHP ist lediglich die Einfachvererbung möglich. Also das Erben von Vorfahren in einer Linie (z. B. Mutter zu Kind). Das Erben von mehreren Vorfahren gleichzeitig (Mehrfachvererbung, z. B. Mutter zu Kind und gleichzeitig von Vater zu Kind) ist in PHP nicht möglich – was uns nicht daran hindert, die Vorteile der Einfachvererbung zu nutzen.

In diesem Buch haben wir zur Erklärung von Objekten bereits einen Fast-Food-Laden zur Hilfe genommen. Dieser Fast-Food-Laden ist eine Firma genauso wie ein Baumarkt, ein Automobilzulieferer oder eine Bäckerei. Allen gemeinsam sind folgende Punkte:

Gemeinsame Eigenschaften verschiedener Objekte

- Jede Firma gehört einer bestimmten Branche an.
- Alle Firmen haben einen Namen.
- Jede Firma hat ein Geschäftsgebäude, in dem die Arbeit verrichtet wird. Zu jedem Geschäftsgebäude gehört eine eindeutige Adresse.
- Bei jeder Firma arbeiten Mitarbeiter.
- Jede Firma nimmt Geld ein und gibt Geld aus.
- ...

Gemeinsamkeiten sollten nur einmal definiert werden

Würden wir nun für jede Firma eine Klasse in PHP 5 schreiben, so könnten wir in jeder Klasse die oben genannten Gemeinsamkeiten mit entsprechenden Attributen und Methoden angeben. Das kann ziemlich aufwendig sein, da wir gleiche Sachverhalte für jede Firma erneut beschreiben müssten.

Was wäre, wenn man diese Gemeinsamkeiten nur einmal formulieren müsste – also nicht bei jeder Firma separat? Damit würden wir uns viel Arbeit sparen. Auch Änderungen von z. B. Firmengebäuden oder Mitarbeiter könnten leichter durchgeführt werden, da diese nur bei der einmaligen Definition aller Gemeinsamkeiten und nicht bei allen Firmen durchgeführt werden müssten.

Eine Klasse für alle Gemeinsamkeiten aller Firmen

Definieren wir also zuerst eine Klasse mit allen Gemeinsamkeiten und nennen diese Klasse ganz allgemein `Firma`. Der Einfachheit halber definieren wir `set`- und `get`-Methoden nur für die Attribute `$name`, `$branche` und `$einnahmen` sowie den Konstruktor.

Listing 12-7 *Die Klasse Firma*

```
<?php
class Firma
{
    // Variablendefinition und -deklaration
    var $name      = null;
    var $branche   = null;
    var $einnahmen = 0;

    // Konstruktor
    function __construct($branche)
    {
        $this->branche = $branche;
    }

    // alle anderen Methoden
    function setName($name)
    {
        $this->name = $name;
    }

    function getName()
    {
        return $this->name;
    }

    function setEinnahmen($einnahmen)
    {
        $this->einnahmen = $einnahmen;
    }

    function getEinnahmen()
    {
        return $this->einnahmen;
    }
}
?>
```

Durch folgendes Codebeispiel können Sie nun mit Hilfe der Klasse `Firma` den Namen der Firma sowie deren Einnahmen verändern oder abfragen:

Listing 12-8 *Anwendung von Firma.php (Auszug)*

```
<?php
require_once 'Firma.php';
// ...
// Zwei Firmenobjekte erzeugen
```

```
$Baumarkt = new Firma('Baumarkt');       // Firma ist Baumarkt
$FastFood = new Firma('Fast-Food-Laden') // Firma ist Fast-Food-Laden
// ...
// Namen der Firmen definieren
$Baumarkt->setName('Baubiber');
$FastFood->setName('McObject');

// Einnahmen der Firmen definieren
$Baumarkt->setEinnahmen(12544,59);
$FastFood->setEinnahmen(27642,32);

// Ausgeben der Firmendaten
echo 'Die Firma ' . $Baumarkt->getName()
     . ' aus der Branche ' . $Baumarkt->getBranche()
     . ' hatte Einnahmen in Höhe von '
     . $Baumarkt->getEinnahmen() . ' EUR<br />';
echo 'Die Firma ' . $FastFood->getName()
     . ' aus der Branche ' . $FastFood->getBranche()
     . ' hatte Einnahmen in Höhe von '
     . $FastFood->getEinnahmen() . ' EUR <br />';
?>
```

Wir mussten hier für die unterschiedlichsten Firmen nur eine Klasse schreiben. Ob Baumarkt oder Fast-Food-Laden – Daten wie Firmenname, Branche oder Einnahmen können mit Hilfe der Klasse `Firma` gut verwaltet werden.

Spezielle Eigenschaften einer Firma sind in einer speziellen Klasse

Kommen wir nun zu den Besonderheiten. Ein Fast-Food-Laden bietet zum Beispiel eine gewisse Anzahl von Getränken und Speisen an – ein Baumarkt eher nicht. Wir könnten nun alle möglichen Attribute in die Klasse `Firma` packen. Das ist jedoch nicht sinnvoll, da ansonsten die Klasse etwas unübersichtlich wird. Wir schreiben nun für den Fast-Food-Laden eine eigene Klasse, die jedoch alle Eigenschaften (Attribute und Methoden) der Klasse `Firma` erbt.

Listing 12-9
Die Klasse FastFoodLaden

```
<?php
class FastFoodLaden extends Firma
{
    // Variablendefinition und -deklaration
    var $getraenke = 0;
    var $speisen = 0;

    function setGetraenke($getraenke)
    {
        $this->getraenke = $getraenke;
    }

    function getGetraenke()
    {
        return $this->getraenke;
```

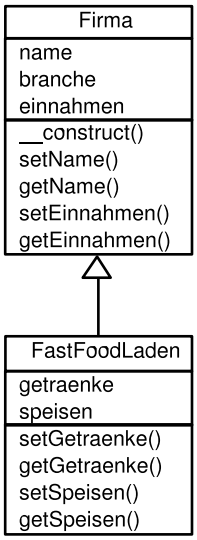
```
    }

    function setSpeisen($speisen)
    {
        $this->speisen = $speisen;
    }

    function getSpeisen()
    {
        return $this->speisen;
    }
}
?>
```

Die spezielle Klasse erbt alle gemeinsamen Eigenschaften

In obigem Beispiel vererbt die Klasse `Firma` der Klasse `FastFoodLaden` alle Eigenschaften. Die Klasse `FastFoodLaden` hat also, ohne dies in der Klasse zu deklarieren, auch alle Attribute und Methoden der Klasse `Firma`. Also auch die Attribute `$name`, `$branche`, `$einnahmen` sowie die entsprechenden Methoden. Dabei ist zu beachten, dass vor dem Klassenbeginn mit `require_once` die Firmenklasse in die Datei der Klasse `FastFoodLaden` eingebunden wird.

Im Klassenmodell der Abbildung 12.4 wird dargestellt, dass die Klasse `FastFoodLaden` Eigenschaften von der Klasse `Firma` erbt.

Abbildung 12.4
Klasse FastFoodLaden erbt von der Klasse Firma



Ein Objekt mit »extends« vom anderen ableiten

Die Klasse `FastFoodLaden` wurde durch das PHP-Sprachelement `extends` von der Mutterklasse `Firma` abgeleitet. Die Klasse `FastFoodLaden` wird in diesem Zusammenhang auch Kind der Mutterklasse `Firma` genannt. Ein neuer Konstruktor oder Destruktor muss nicht definiert

werden – falls vorhanden wird der Konstruktor oder Destruktor der Mutterklasse verwendet.

Mit Hilfe der neuen, von der Klasse `Firma` abgeleiteten Klasse `FastFoodLaden` kann das obige Beispiel nun wie folgt aussehen:

Listing 12-10
Anwendung von FastFoodLaden.php (Auszug)

```
<?php
require_once 'Firma.php';
require_once 'FastFoodLaden.php';
// ...
// Zwei Firmenobjekte erzeugen
$Baumarkt = new Firma('Baumarkt');
$FastFood = new FastFoodLaden('Fast-Food-Laden');
// ...
// Namen der Firmen definieren
$Baumarkt->setName('Baubiber');
$FastFood->setName('McObject');

// Einnahmen der Firmen definieren
$Baumarkt->setEinnahmen(12544,59);
$FastFood->setEinnahmen(27642,32);

// Definition von Speisen und Getränken
$FastFood->setGetraenke(4);
$FastFood->setSpeisen(10);

// Ausgeben der Firmendaten
echo 'Die Firma ' . $Baumarkt->getName()
     . ' aus der Branche ' . $Baumarkt->getBranche()
     . ' hatte Einnahmen in Höhe von '
     . $Baumarkt->getEinnahmen() . ' EUR <br />';
echo 'Die Firma '.$FastFood->getName()
    . ' aus der Branche ' . $FastFood->getBranche()
    . ' hatte Einnahmen in Höhe von '
    . $FastFood->getEinnahmen() . ' EUR mit '
    . $FastFood->getGetraenke() . ' Getraenken und '
    . $FastFood->getSpeisen() . ' Speisen erzielt<br />';
?>
```

Hier wird, im Gegensatz zum ersten Beispiel, mit der Objektvariablen `$FastFood` ein Objekt der Klasse `FastFoodLaden` erzeugt – nicht wie im ersten Beispiel der Klasse `Firma`. Für die zweite Firma werden dann mit den Methoden `setGetraenke()` und `setSpeisen()` die Anzahl von Getränken und Speisen definiert. Bei der Ausgabe der Firmendaten werden die Getränke- und Speisenanzahl mit ausgegeben. Diese Eigenschaften waren in der Mutterklasse noch nicht vorhanden.

Mutterklasse vererbt immer an eine Kindklasse – nicht umgekehrt

Der Aufruf der Methoden `setGetraenke()` und `setSpeisen()` beim Objekt `$Baumarkt` würde einen Fehler verursachen, da die beiden Methoden nur in der Klasse `FastFoodLaden`, jedoch nicht in der Klasse `Firma` existieren. Die Klasse `Firma` vererbt zwar ihre Eigenschaften an die

Klasse `FastFoodLaden` – dies gilt jedoch nicht für den umgekehrten Fall. Die Klasse `FastFoodLaden` besitzt also zu den geerbten Eigenschaften zusätzliche ganz individuelle Eigenschaften, welche in der Klasse `Firma` nirgends zu finden sind.

Ableiten bedeutet vererben und vereinfachen

Stellen Sie sich vor, sie hätten alle Attribute und Methoden der Klasse `Firma` nochmal in der Klasse `FastFoodLaden` definieren müssen. Diesen Aufwand haben Sie sich durch das Ableiten der Klasse `Firma` gespart. Wenn Sie die Klasse `FastFoodLaden` genauer betrachten, ist diese Klasse dadurch auch übersichtlicher geworden (sämtliche Attribute und Methoden der Klasse `Firma` müssen nicht nochmal geschrieben werden).

12.2.2 Attribute und Methoden überschreiben

Wir haben im obigen Beispiel die Klasse `FastFoodLaden` von der Klasse `Firma` mit dem Sprachelement `extends` abgeleitet. In der Kindklasse `FastFoodLaden` sind also auch die Attribute und Methoden der Mutterklasse `Firma` verfügbar, auch wenn diese in der Kindklasse nicht definiert wurden.

Geerbte Eigenschaften können überschrieben werden

Nun gibt es Fälle, in denen zwar eine Methode der Mutterklasse auch in der Kindklasse verfügbar sein sollte, jedoch mit einer leicht veränderten Funktionalität. Damit man die entsprechende Methode nicht über einen neuen Namen ansprechen muss, gibt es die Möglichkeit, Methoden zu überschreiben. Dabei definiert man in der Kindklasse die entsprechende Methode nochmal, obwohl diese in der Mutterklasse bereits existiert. In der Kindklasse wird die überschriebene Methode jedoch mit einer anderen Funktionalität ausgerüstet. Auf diese Weise können Attribute und Methoden der Mutterklasse überschrieben werden.

Zur Verdeutlichung dieser Vorgehensweise, wollen wir noch einmal auf unser Firmenbeispiel zurückgreifen. Nehmen Sie an, dass eine neue Firma ab sofort in einer neuen Währung, den »Alb-Talern«, arbeitet – und nicht mehr in Euro. Dazu leiten wir die `Firma`-Klasse zu einer neuen `Albfirma`-Klasse ab. Um die Firma zu anderen Firmen kompatibel zu halten, speichern wir die Daten nach wie vor in der Währung Euro ab. Lediglich die Änderung der Einnahmen und das Lesen der Einnahmen soll mit der neuen Alb-Taler-Währung (1 Alb-Taler = 42 Euro) durchgeführt werden.

Listing 12-11 *Auszug aus Albfirma.php*

```
<?php
class Albfirma extends Firma
{
    var $wechselkurs = 42; //Euro pro Alb-Taler

    // ...
```

```
    function setWechselkurs($wechselkurs = Null)
    {
        $this->wechselkurs = $wechselkurs;
    }

    function getWechselkurs()
    {
        return $this->wechselkurs;
    }

    function setEinnahmen($einnahmen)
    {
        $this->einnahmen = $einnahmen * $this->getWechselkurs();
    }

    function getEinnahmen()
    {
        return $this->einnahmen / $this->getWechselkurs();
    }
}
?>
```

Bei der Mutterklasse `Firma` wurde noch kein Attribut für den Wechselkurs definiert. In der Kindklasse definieren wir daher den `$wechselkurs` als ein neues Attribut. Ebenso programmieren wir die entsprechenden set- und get-Methoden zu diesem Attribut.

In der Klasse `Firma` gibt es bereits die set- und get-Methoden für das Attribut `$einnahmen`. Wir wollen, dass diese Methoden ab sofort in »Alb-Taler« arbeiten. Dazu müssen wir die beiden Methoden überschreiben. Bei der Methode `setEinnahmen()` multiplizieren wir die übergebenen Einnahmen vor dem Speichern mit dem Wechselkurs.

In der Methode `getEinnahmen()` teilen wir das Einnahmen-Attribut vor der Rückgabe durch den Wechselkurs.

Mit Überschreiben das Erbgut einer Klasse neu definieren

Damit überschreiben wir die gleichnamige Methode der Mutterklasse `Firma` und verändern so das Erbgut in der Kindklasse. Wenn Sie nun eine `Albfirma` erzeugen, dann können Sie den beiden Methoden die Einnahmen in der Währung Alb-Taler übergeben, speichern aber alle Werte nach wie vor in Euro. Folgendes Beispiel verdeutlicht das:

Listing 12-12
Anwendung von Albfirma.php (Auszug)

```
<?php
require_once 'Albfirma.php';

$Albfirma = new Albfirma();
$Albfirma->setEinnahmen(3.450);  // Alb-Taler
echo 'Einnahmen (Alb-Taler): '
     . $Albfirma->getEinnahmen() . '<br />';
echo 'Einnahmen (Euro): ' . $Albfirma->einnahmen;
?>
```

Wenn Sie die Einnahmen der `Albfirma` in Euro handhaben möchten, dann können Sie nach wie vor auf die Methoden der Mutterklasse zurückgreifen. Wie am folgenden Beispiel (siehe dazu 12.2.2) aufgezeigt, müssen Sie dazu nur die entsprechende Methode der Mutterklasse aufrufen.

```
class Albfirma extends Firma
{
    // ...

    function getEinnahmenEuro()
    {
        return Firma::getEinnahmen();
    }
}
```

Da die Methode `getEinnahmen()` der Mutterklasse in Euro arbeitet, kann der Rückgabewert dieser Methode in der Methode `getEinnahmenEuro()` der Kindklasse aufgerufen werden.

Den Aufruf von `getEinnahmen()` mit `::` wollen wir im folgenden Abschnitt zu Objekt- und Klassenaufrufen näher betrachten.

12.2.3 Objekt- und Klassenaufrufe

Statische Methodenaufrufe

Sie können die Methoden eines Objektes auf zwei verschiedene Weisen aufrufen. Bisher haben Sie die Objektaufrufe einer Methode kennen gelernt. Hier wollen wir Ihnen die statischen Methodenaufrufe oder Klassenaufrufe vorstellen. Damit werden Sie in der Lage sein, vorgefertigte Lösungen durch Vererbung noch effektiver zu nutzen.

Objektaufrufe mit »->«

Ein Objekt ist ein Exemplar einer Klasse bzw. ein reales »Exemplar« nach dem Baumuster der entsprechenden Klasse. Zu diesem Objekt gehören sämtliche Werte aller Attribute, welche zusammen einen gewissen Objektstatus ergeben. Jedes Objekt hat also immer einen durch die Attributwerte definierten Zustand. Der Zustand kann mit Hilfe der Methoden geändert werden. Auf die Methoden eines Objekts wird bekanntermaßen mit `->` zugegriffen.

Mit

```
$Baumarkt = new Firma('Baumarkt');
$FastFood = new Firma('Fast-Food-Laden');
```

werden zwei Objekte nach dem Baumuster der Klasse `Firma` erzeugt. Das Objekt in `$Baumarkt` hat z. B. mit dem Attribut `branche` den Status eines Baumarkts. Das Objekt in `$FastFood` hat den Zustand eines Fast-Food-Ladens.

Die Klasse an sich ist lediglich das Baumuster oder der Bauplan für die Objekte. In diesem Bauplan sind keine Attributwerte gespeichert, weshalb eine Klasse an sich auch keinen Zustand haben kann. Im Bauplan ist lediglich eine Anweisung – der Konstruktor – hinterlegt, in welchem Zustand ein Objekt direkt nach dem Erzeugen versetzt werden soll und über welche Methoden in welcher Art und Weise der Zustand des Objektes später geändert werden kann.

Aufrufe von Methoden in Klassen – statische Methodenaufrufe

Bei der objektorientierten Programmierung kann nun der Bauplan einer Klasse real genutzt werden, ohne dass man gleich ein neues Objekt erzeugt. Zum Beispiel kann auf die Methode `connect()` der Klasse `DB` zugegriffen werden, ohne nach dem Baumuster dieser Klasse zuerst ein Objekt zu erzeugen.

```
// funktioniert auch ohne vorheriges "new DB();"
$db = DB::connect($dsn);
```

Klassenaufrufe mit »::«

Durch `DB` stellen wir sicher, dass eine Methode der gleichnamigen Klasse aufgerufen wird. Vor den beiden Doppelpunkten steht also der Klassenname. Die Doppelpunkte haben die gleiche Wirkung wie `->` in Zusammenhang mit `$variable` oder einem Objekt (Ausnahme: statische Methodenaufrufe in Kapitel 12.2.5 ab Seite 207). Hinter den Doppelpunkten wird dann der Methodennamen definiert. Mit `connect()` wird also die entsprechende Methode der vor dem Doppelpunkt angegebenen Klasse aufgerufen.

Wenn Sie in einer abgeleiteten Klasse z. B. den Konstruktor neu programmieren, dann können Sie trotzdem (auch wenn der Konstruktor der Mutterklasse überschrieben ist) auf den Mutterklassenkonstruktor und auf alle Attribute der Mutterklasse zugreifen – falls definiert, auch auf andere überschriebene Methoden:

```
class Mutter
{
    var $name = null;

    function __construct()
    {
        echo $this->name;
    }
}

class Kind extends Mutter
{
    function __construct()
    {
        $this->name = 'Max Muster';
        Mutter::__construct();
    }
}
```

Mit der Klasse `Kind` leiten Sie die Mutterklasse `Mutter` ab. In der Kindklasse überschreiben Sie den Konstruktor. Im überschriebenen Konstruktor der Kindklasse `Kind` definieren Sie das Attribut `$name` und rufen den Konstruktor der Mutterklasse auf. Nur im Konstruktor der Mutter wird das Attribut `$name` am Bildschirm ausgegeben. Würden Sie nun ein Objekt der Kindklasse mit z. B. `$Kind = new Kind();` aufrufen, so könnten Sie auf dem Bildschirm die Ausgabe des Namens erkennen, obwohl diese Ausgabe in der Kindklasse nicht definiert wurde. Da Sie im Konstruktor der Kindklasse den Konstruktor der Mutterklasse aufrufen, kommt die Funktionalität des Mutterklassenkonstruktors trotzdem zum Tragen.

Vergleich: Klassenaufruf mit klassenbezogenem include

Für eingefleischte Skripter könnte man einen solchen Klassenaufruf auch folgendermaßen umschreiben: Man inkludiert an der Stelle des Klassenaufrufes den Code der aufgerufenen Methode. In obigem Beispiel könnte im Konstruktor der Kindklasse auch statt `Mutter::__construct()` der Code `echo $this->name;` stehen – mit dem gleichen Ergebnis.

Mutterklasse mit parent aufrufen.

Wollen Sie die Methode einer Mutterklasse in einer Kindklasse aufrufen, dann können Sie sich auch des Schlüsselwortes `parent` bedienen. Die obige Kindklasse würde dann folgendermaßen aussehen:

```
class Kind extends Mutter
{
    function __construct()
    {
        $this->name = 'Max Muster';
        parent::__construct();
    }
}
```

Der Vorteil dabei ist, dass Sie den eigentlichen Namen der Mutterklasse gar nicht mehr kennen müssen. Sollten Sie die Klassenbezeichnung der Mutterklasse ändern, dann funktioniert der Aufruf `parent::__construct();` immer noch – der Aufruf des Mutterkonstruktors mittels `Mutter::__construct();` dagegen würde dann nicht mehr funktionieren.

Achtung: Konstruktor-Falle

Beachten Sie bitte Folgendes: Wenn Sie in einer Kindklasse einen Konstruktor definieren, dann wird der Konstruktor der Mutterklasse nicht automatisch aufgerufen. Den Mutterklassenkonstruktor müssen Sie im Kindklassenkonstruktor wie oben aufgezeigt selbst aufrufen.

12.2.4 Abstrakte Klassen und Methoden

Es gibt Situationen, in denen wir gemeinsame Funktionalität in einer Mutterklasse zwar definieren, jedoch spezielle Funktionselemente und

Methoden erst in den davon abgeleiteten Klassen programmieren. In solchen Fällen haben Sie die Möglichkeit, eine Klasse zu definieren, jedoch die Erzeugung eines Objektes dieser Klasse nicht zu erlauben – also die Ableitung der Klasse zu erzwingen. Eine so definierte Klasse wird als abstrakt bezeichnet und mit dem Schlüsselwort `abstract` versehen.

Überschreiben von Methoden in der Kindklasse erzwingen

Zu diesem Zweck können Sie mit abstrakten Methoden arbeiten. Dazu definieren Sie eine Methode in der Mutterklasse ohne Funktionalität – eine Methode, in der nichts getan wird. In der Mutterklasse können Sie diese Methode bereits aufrufen. Erst in der Kindklasse aber füllen Sie die abstrakte Methode mit Funktionalität bzw. Code. Mit dem Schlüsselwort `abstract` können Sie den Programmierer zum Überschreiben abstrakter Methoden zwingen.

```
<?php
abstract class Mutter
{
    function __construct()
    {
        $this->doSpecial();
    }

    // Abstrakte Methode
    abstract function doSpecial();
}

class Kind extends Mutter
{
    function doSpecial()
    {
        // Spezielle Funktion der Kindklasse
        // z.B.
        echo 'Alles klar';
    }
}
?>
```

Bereits in der Mutterklasse kann mit einer erst in der Kindklasse definierten abstrakten Methode gearbeitet werden

Im Konstruktor der Mutterklasse `Mutter` wird bereits mit der Methode `doSpecial()` gearbeitet, obwohl in dieser Methode keinerlei Funktion hinterlegt ist. Die abstrakte Methode `doSpecial()` wird erst in der von der Mutterklasse abgeleiteten Kindklasse `Kind` definiert.

Bitte beachten sie, dass Sie die Mutterklasse sowie deren abstrakte Methode als `abstract` definieren. Bei einer abstrakten Methode dürfen Sie keinen Anweisungsteil (auch keine geschweiften Klammern) definieren. In der Kindklasse müssen Sie alle abstrakten Methoden überschreiben.

Vorteilhaft sind abstrakte Methoden dann, wenn Sie in den von einer Mutterklasse abgeleiteten Klassen spezielle Funktionalität unter-

bringen wollen, auf die Sie jedoch für viele Kindklassen gemeinsam in der Mutterklasse zugreifen möchten. Das folgende, etwas einfachere Beispiel macht den Umgang mit abstrakten Methoden nochmal deutlich:

```
<?php
abstract class Firma
{
    var $name    = 'n.n.';
    var $branche = 'n.n.';

    // Konstruktor
    function __construct($branche)
    {
        $this->branche = $branche;
        $this->showFirma();
    }

    // Abstrakte Methode
    abstract function showFirma();

    // alle anderen Methoden
    function setName($name)
    {
        $this->name = $name;
    }

    function getName()
    {
        return $this->name;
    }
}

class FastFoodLaden extends Firma
{
    // Variablendefinition und -deklaration
    var $getraenke = 0;

    function showFirma()
    {
        echo 'Die Firma ' . $this->getName()
             . ' aus der Branche ' . $this->getBranche()
             . ' hatte Einnahmen mit ' . $this->getGetraenke()
             . ' Getränken und ' . $this->getSpeisen()
             . ' Speisen erzielt<br />';
    }

    function setGetraenke($getraenke)
    {
        $this->getraenke = $getraenke;
    }
```

```
    function getGetraenke()
    {
        return $this->getraenke;
    }
}

class SonstigeFirma extends Firma
{
    function showFirma()
    {
        echo 'Die Firma ' . $this->getName()
             . ' ist aus der Branche ' . $this->getBranche()
             . ' <br />';
    }
}
?>
```

Vorteile einer abstrakten show-Methode

Das prinzipielle Beispiel kennen Sie ja bereits aus dem Kapitel 12.2.1 zur Einfachvererbung (Seite 193). Wir haben die Ausgabe der Firmendaten in die `showFirma()`-Methode verlagert. Die gesamten Firmendaten müssen nun also nicht mehr aufwendig jedes Mal neu bei einem Objekt der Klassen programmiert werden – ein Aufruf von `showFirma()` genügt. In der Mutterklasse `Firma` haben wir die `showFirma()`-Methode abstrakt definiert. In den abgeleiteten Klassen `FastFoodLaden` und `SonstigeFirma` wird die abstrakte Klasse überschrieben – dort haben wir die genaue Ausgabe der Firmendaten programmiert. Ganz nebenbei vereinfacht sich dann auch der Code für die Anwendung der Klassen wie folgt:

```
<?php
require_once 'SonstigeFirma.php';
require_once 'FastFoodLaden.php';
// ...
// Zwei Firmen erzeugen
$Baumarkt = new SonstigeFirma('Baumarkt');
$FastFood = new FastFoodLaden('Fast-Food-Laden');
// ...
// Namen der Firmen definieren
$Baumarkt->setName('Baubiber');
$FastFood->setName('McFast');

// Definition von Speisen und Getränken
$FastFood->setGetraenke(4);
$FastFood->setSpeisen(10);

// Ausgeben der Firmendaten
$Baumarkt->showFirma();
$FastFood->showFirma();
?>
```

Das Besondere an diesem Beispiel ist nun, dass sofort beim Erzeugen des Objekts die Methode `showFirma()` im Konstruktor aufgerufen wird. Die Firmendaten werden nun immer auch direkt nach der Objekterzeugung am Bildschirm ausgegeben. Da zum Zeitpunkt der Objekterzeugung lediglich die Branche definiert ist, wird auch nur die Branche richtig ausgegeben. Nach dem Ändern der Objektzustände (Name ändern, Einnahmen ändern, Getränke- und Speisenanzahl ändern bei der Fast-Food-Firma) werden dann die beiden Firmendaten nochmals ausgegeben – nun mit den geänderten Daten.

Das gleiche Ergebnis hätten Sie auch erzielen können, indem Sie in der `showFirma()`-Methode der Mutterklasse `Firma` mit Hilfe eines `switch`-Konstruktes je nach Branche unterschiedliche Ausgaben generiert hätten – dann eben nicht so schön und nicht objektorientiert. Sie hätten dann bereits in der Mutterklasse eigentlich noch nicht bekannte Informationen, wie Getränke- oder Speisenanzahl, verarbeiten müssen.

12.2.5 final und static

Mit dem Schlüsselwort `abstract` können Sie das Überschreiben von Methoden erzwingen. PHP 5 bietet Ihnen darüber hinaus die Möglichkeit, mit dem Schlüsselwort `final` das Überschreiben von Methoden zu unterbinden. Außerdem können Sie in PHP 5 mit der Angabe von `static` Methoden nicht wie gewohnt für jedes erzeugte Objekt, sondern nur einmal für jede Klasse zur Verfügung stellen.

Mit den Schlüsselwörtern `final`, `static` und `abstract` können Sie Ihre Arbeit mit Objekten noch effektiver gestalten.

final

final-Methoden sind nicht überschreibbar

Mit `final` können Sie das Überschreiben einer Methode verhindern.

```
class Mutter
{
    public final function felsenfest()
    {
       // ...
    }
}

class Kind
{
    felsenfest() {}    // Fehler, da Methode final!
}
```

Das Schlüsselwort können Sie dabei zusammen mit `public`, `private` oder `protected` kombinieren.

static

Um auf ein Attribut oder eine Methode sinnvoll zugreifen zu können, mussten Sie ein Objekt der entsprechenden Klasse erzeugen. Attribute und Methoden existieren also nur in erzeugten Objekten – in einer Klasse lediglich als Baumuster.

Statische Klassenelemente funktionieren ohne Objekterzeugung
Statische Elemente existieren je Klasse nur ein Mal

In PHP 5 haben Sie die Möglichkeit, statische Attribute und Methoden zu definieren. Diese Attribute und Methoden können Sie auch aufrufen, wenn Sie noch kein Objekt erzeugt haben (siehe auch Abschnitt 12.2.3 ab Seite 200).

Wenn Sie viele Objekte einer Klasse erzeugen, in der eine statische Methode definiert ist, dann gibt es die Methode nicht wie üblich mehrmals (je Objekt ein Mal), sondern nur (je Klasse) ein Mal.

Statische Attribute und Methoden können Sie dann sinnvoll nutzen, wenn Sie für viele Objekte eine einheitliche objektunabhängige, aber klassenbezogene Arbeit verrichten wollen. Beispielsweise können Sie mit einem statischen Attribut und einer statischen Methode für viele erzeugte Objekte einer Datenbankklasse die Datenbankverbindung nur ein Mal herstellen – und nicht für alle erzeugten Objekte der entsprechenden Klasse.

Listing 12-13
statische Attribute und Methoden

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

    protected static function connectDB($host, $user, $password)
    {
        $db = mysql_connect($host, $user, $password);
        if (!$db) {
           die('Verbindungsfehler: ' . mysql_error());
        } else {
           return $db;
        }
    }

    public function openDB()
    {
        $this->db = $this->connectDB('host', 'user', 'password');
        return $this->db;
    }
}

// Methodenaufruf ohne Objekterzeugung
// ist nur bei static-Methoden möglich!
$myDB   = Test::connectDB();
```

```
// oder aus irgendeinem Objekt heraus
$Test = new Test();
$yourDB = $Test->openDB();
?>
```

Innerhalb statisch definierter Methoden können Sie nicht mit der Objektvariablen `$this` arbeiten.

Wenn Sie statische Methoden überschreiben, dann müssen Sie die überschreibende Methode ebenfalls als `static` definieren.

12.3 Zugriffsmöglichkeiten einschränken

Zugriff auf Objekte unterschiedlich restriktiv ermöglichen

Wir haben schon einige Male auf die Möglichkeit hingewiesen, Objekte unterschiedlich gegenüber dem Zugriff von außen zu schützen. Mit den Sprachelementen `public`, `protected` und `private` können Sie den Zugriff auf Attribute und Methoden für die eigene Klasse, abgeleitete Klassen oder fremde Klassen unterschiedlich restriktiv handhaben.

Egal um welches der drei Schlüsselwörter es sich handelt – `public`, `protected` oder `private` – alle Schlüsselwörter werden im PHP-Code in folgender Weise verwendet:

public, protected und private bei Attributen

Bei Attributen steht das Schlüsselwort direkt vor dem Attribut. Dabei wird das Sprachelement `var` weggelassen.

```
class Beispiel
{
    // ...
    [Schlüsselwort] [$Attributname];
    // ...
}
```

public, protected und private bei Methoden

Entsprechend wir das Schlüsselwort bei Methoden direkt vor dem Sprachelement `function` angegeben.

```
class Beispiel
{
    // ...
    [Schluesselwort] function [Methodennamen]()
    {
        //...
    }
    // ...
}
```

Wenn Sie wie bisher Attribute mit `var` und Methoden nur mit `function` definieren, dann werden diese automatisch als `public` interpretiert.

An folgendem Beispiel können Sie sich einen ersten Überblick zur Funktionsweise der Zugriffsbeschränkungen erarbeiten. Danach gehen wir auf die einzelnen Möglichkeiten detailliert ein.

Listing 12-14
Beispiel zu public, private und protected

```
<?php
class Mutter
{
    public    $public_var    = 'Ich bin public';
    protected $protected_var = 'Ich bin protected';
    private   $private_var   = 'Ich bin private';


    public function getPublic()
    {
        echo $this->public_var    . '<br />';
        echo $this->protected_var . '<br />';
        echo $this->private_var   . '<br />';
        echo '<br />';
    }

    protected function getProtected()
    {
        echo $this->public_var    . '<br />';
        echo $this->protected_var . '<br />';
        echo $this->private_var   . '<br />';
        echo '<br />';
    }

    private function getPrivate()
    {
        echo $this->public_var    . '<br />';
        echo $this->protected_var . '<br />';
        echo $this->private_var   . '<br />';
        echo '<br />';
    }
}

class Kind extends Mutter
{
    public function test()
    {
        echo $this->public_var    . '<br />';  // funktioniert
        echo $this->protected_var . '<br />';  // funktioniert
        echo $this->private_var   . '<br />';  // Fehler

        $this->getPublic();               // funktioniert
        $this->getProtected();            // funktioniert
        $this->getPrivate();              // Fehler
    }
}

$Test = new Kind();

echo $Test->public_var    . '<br />';      // funktioniert
```

```
echo $Test->protected_var . '<br />';      // Fehler
echo $Test->private_var   . '<br />';      // Fehler

echo '<hr />';

$Test->getPublic();                        // funktioniert
$Test->getProtected();                     // Fehler
$Test->getPrivate();                       // Fehler

$Test->test();                             // Methodenaufruf
                                           // funktioniert
?>
```

12.3.1 public

Als public definierte Elemente sind völlig offene Schnittstellen zum Objekt

Sie wollen Objekteigenschaften öffentlich zugreifbar belassen? Attribute ohne den Umweg über Methoden verändern? Dann erfahren Sie hier, wie das geht.

Sowohl Attribute als auch Methoden können `public` definiert werden und stellen damit öffentliche Schnittstellen zur Klasse dar. Über diese Schnittstelle kann jeder uneingeschränkt auf die als `public` definierten Attribute und Methoden zugreifen.

Bei einem Attribut wird das Schlüsselwort `public` direkt vor dem Attribut angegeben und bei einer Methode direkt vor das Sprachelement `function` geschrieben.

Listing 12-15
Auszug aus Bank.php

```
<?php
class Bank
{
    public $money = 0;

    function __construct($money)
    {
        $this->setMoney($money);
    }

    public function setMoney($money)
    {
        $this->money = $money
    }

    public function getMoney()
    {
        return $this->money;
    }
}
?>
```

Bei diesem Beispiel kann jeder von außen auf alle Attribute und alle Methoden zugreifen. Von jeder Stelle innerhalb der eigenen Klasse ist der Zugriff auf die Attribute und Methoden ebenso uneingeschränkt möglich wie aus einer Kindklasse oder aus einer ganz anderen Klassen heraus.

Jeder kann auf Public-Attribute und Public-Methoden zugreifen

Listing 12-16
Anwendung von Bank.php (Auszug)

```
<?php
require_once 'Bank.php';

// Objekt erzeugen
$Bank = new Bank(1000000);

// Attribute direkt oder ueber Methode auslesen
echo $Bank->getMoney() . '</br>';
echo $Bank->money . '</br>';

// Über Methode Safe definieren und auslesen
$Bank->setMoney(2000000);
echo $Bank->getMoney() . '</br>';

// Safe direkt über Attribut definieren
// und auslesen
$Bank->money = 3000000;
echo $Bank->money . '</br>';
?>
```

Attribute sollten jedoch sinnvollerweise nicht `public` definiert werden. Da sämtliche Werte ungeschützt von außen geändert werden können, kann dies für die Lauffähigkeit und Sicherheit des Programms sehr nachteilig sein. Wenn auf Attribute nur über Methoden schreibend oder lesend zugegriffen werden kann, dann sind die Daten im Objekt besser gekapselt – damit kann der Zugriff auf Attribute gezielt gesteuert und definiert werden. Auch werden Programme weniger fehleranfällig und sind leichter pflegbar.

Attribute sollten nach Möglichkeit nicht public definiert werden

12.3.2 private

Wir haben vorher beschrieben, wie Sie Attribute und Methoden öffentlich zugreifbar definieren können. Nun wollen wir den Zugriff auf ein Objekt einschränken.

Als private definierte Klassenelemente sind nur in der eigenen Klasse verfügbar

Attribute wie Methoden können als `private` definiert werden und sind dadurch von außen nicht mehr ansprechbar. Diese Schnittstellen sind nur innerhalb der eigenen Klasse zugänglich.

Auch hier wird wieder bei einem Attribut das Schlüsselwort `private` direkt vor dem Attribut platziert und bei einer Methode direkt vor das Sprachelement `function` geschrieben.

```
<?php
class Bank
{
    private $safe = 0;

    private function setMoneyToSafe($money)
    {
        $this->safe = $money;
    }

    private function getMoneyFromSafe()
    {
        return $this->safe;
    }

    public function setMoney($money)
    {
        $this->setMoneyToSafe($money);
    }

    public function getMoney()
    {
        return $this->getMoneyFromSafe();
    }
}
```

Listing 12-17 *Auszug aus Pfandhaus.php*

```
class Pfandhaus extends Bank
{
    public function getMoreMoney()
    {
        $this->getMoneyFromSafe();
    }

    public function getReallyMoney()
    {
        $this->getMoney();
    }
}
?>
```

Abgeleitete Klassen können nicht auf Private-Elemente der Mutterklasse zugreifen!

Nur Methoden innerhalb der eigenen Klasse `Bank` können auf die Methoden `setMoneyToSafe()` und `getMoneyFromSafe()` zugreifen. Selbst von abgeleiteten Klassen (in obigem Beispiel Klasse `Pfandhaus`) aus kann nicht auf diese Methoden zugegriffen werden. Folgendes Beispiel verdeutlicht dies:

```
<?php
require_once 'Bank.php';
require_once 'Pfandhaus.php';
```

```
// Objekte erzeugen
$Bank = new Bank();
$Pfandhaus = new Pfandhaus();

// Vergebliche Versuche
$Bank->setMoneyToSafe(1000000);
$Pfandhaus->setMoneyToSafe(2000000);
echo $Bank->getMoneyFromSafe();
echo $Pfandhaus->getMoreMoney();

// Zugriff auf Geld funktioniert nur so
$Bank->setMoney(1000000);
$Pfandhaus->setMoney(2000000);
echo $Bank->getMoney();
echo $Pfandhaus->getMoney();
echo $Pfandhaus->getReallyMoney();
?>
```

Guter Objektschutz durch den Einsatz von »private«

Hier wird deutlich, welche Vorteile man durch die `private`-Definition der Attribute und ggf. auch von Methoden hat. Sie können auf diese Weise den Zustand eines Objektes sehr gut schützen und damit auch kapseln, deshalb werden wichtige Attribute oder z. B. auch Hilfsmethoden als `private` definiert.

12.3.3 protected

Als protected definierte Klassenelemente sind für die eigene und davon abgeleitete Klassen verfügbar

Es gibt Anwendungsfälle, bei denen sie zwar von anderen Objekten keinen direkten Zugriff auf objekteigene Attribute und Methoden erlauben wollen, jedoch von erbenden Klassen aus das Nutzen der Objekteigenschaften für sinnvoll finden. Dabei kann Ihnen `protected` weiterhelfen.

Als `protected` definierte Attribute oder Methoden können nur innerhalb der Klasse sowie von erbenden Klassen angesprochen werden.

Das Schlüsselwort `protected` wird direkt vor dem Attribut angegeben und bei einer Methode auch wieder direkt vor das Sprachelement `function` geschrieben.

```
<?php
class Bank
{
    private $safe = 0

    protected function setMoneyToSafe($money)
    {
        $this->safe = $money;
    }
```

```
    // Nur Bank darf abheben
    private function getMoneyFromSafe()
    {
        return $this->safe;
    }

    // Jeder darf einzahlen
    public function setMoney($money)
    {
        $this->setMoneyToSafe($money);
    }

    // Jeder darf abheben
    public function getMoney()
    {
        return $this->getMoneyFromSafe();
    }
}

class Pfandhaus extends Bank
{
    public function setMoreMoney($money)
    {
        $this->setMoneyToSafe($money);
    }

    public function getMoreMoney()
    {
        $this->getMoneyFromSafe();
    }
}
?>
```

Abgeleitete Klassen können auf protected-Elemente der Mutterklasse zurückgreifen

Sowohl Methoden innerhalb der eigenen Klasse `Bank` als auch von dieser Klasse abgeleitete Klassen (z. B. `Pfandhaus`) können auf die Methode `setMoneyToSafe()` zugreifen. Auf die Methode `getMoneyFromSafe()` kann nur die eigene `Bank`-Klasse zugreifen. Von der abgeleiteten Klasse `Pfandhaus` aus kann nur auf `public`- oder `protected`-Elemente zugegriffen werden. Dies gilt gleichermaßen für Attribute und Methoden.

```
<?php
require_once 'Bank.php';
require_once 'Pfandhaus.php';

// Objekte erzeugen
$Bank = new Bank();
$Pfandhaus = new Pfandhaus();

// Vergebliche Versuche
$Bank->setMoneyToSafe(1000000);
$Pfandhaus->setMoneyToSafe(2000000);
```

```
echo $Bank->getMoneyFromSafe();
echo $Pfandhaus->getMoreMoney();

// Zugriff auf Geld funktioniert nur so
$Bank->setMoney(1000000);
$Pfandhaus->setMoney(2000000);
$Pfandhaus->setMoreMoney(3000000);
echo $Bank->getMoney();
echo $Pfandhaus->getMoney();
?>
```

Mit protected bleibt das Erbgut in der Familie

Mit `protected` können Sie den Zugriff auf Attribute und Methoden so einschränken, dass nur die eigene Klasse oder eine von dieser abgeleiteten Klasse die mit `protected` definierten Attribute und Methoden direkt nutzen können.

12.4 Konstanten und Variablen in Objekten

Das Kopieren von Objekten und das Verweisen auf Objekte kann durchaus zu unerwünschten Nebeneffekten führen. Damit Sie mit dem Kopieren und Verweisen in Bezug auf Objekte die PHP-Funktionalität richtig nutzen können, haben wir hier einige Tipps zur Variablendefinition und zum Klonen mit Objekten dargestellt. Außerdem wollen wir in diesem Zusammenhang kurz auf Klassenkonstanten eingehen.

12.4.1 Klassenkonstanten

Klassenkonstanten mit const definieren

In PHP 5 können Sie Klassenkonstanten – also Konstanten innerhalb des Wertebereichs einer Klasse – mit dem Schlüsselwort `const` definieren.

Listing 12-18
Klassenkonstanten

```
<?php
// Skriptkonstante war bisher auch möglich:
define('SCRIPT_KONSTANTE', 'Skriptkonstante');

class Test
{
    // Klassenkonstante ist erst mit PHP 5 möglich:
    const KLASSEN_KONSTANTE = 'Klassenkonstante';

    function zeigeKonstante()
    {
        echo SCRIPT_KONSTANTE;
        echo '<br />';
        echo Test::KLASSEN_KONSTANTE;
    }
}
```