

Grundlagen der PHP-Programmierung

PHP 8: Grundlagen der OOP



d-Taube

29.01.2024 bis 19.04.2024 - Oliver Vogt

Objektorientierte Programmierung mit PHP

Agenda Tag 02

Getter- und Setter-Methoden (accessor und mutator)

Arbeiten mit Objekten

- Methoden innerhalb eines Objektes aufrufen
- Objekte als lokale Variablen
- Objekte als Parameter übergeben
- Verhalten von Kopien von Objekten
- Objekte in Attributen ablegen
- Informationen zu Objekten abfragen / Klassentypen und »instanceof«
- Fehlerlevel
- Typdeklarationen (früher type hinting bzw. type hints)

Virtuelle Attribute



Lernziele

Teilgebiet Einführung in die OOP

- Kennenlernen Attribute, Methoden und Pseudovariablen `$this`
- Kennenlernen Arbeiten mit Objekten in PHP
- Kennenlernen des Konzepts der virtuellen Attribute



Namenskonventionen

1. **Klassen** - Klassennamen verwenden den **UpperCamelCase**, beginnen also mit einem Großbuchstaben. Bei zusammengesetzten Wörtern wird jedes Wort großgeschrieben. Klassennamen sind i. R. Nomen und werden immer in Einzahl geschrieben.
2. **Attribute** - Attribute verwenden den **lowerCamelCase**, werden also kleingeschrieben. Bei zusammengesetzten Wörtern wird ab dem 2-ten Wort jedes Wort großgeschrieben. Also analog der Namensgebung von Variablen.
3. **Methoden** - Für Methoden gelten dieselben Regeln wie für Attribute sowie für die Namensgebung von Funktionen, der Name sollte immer mit einem Verb beginnen.

Konzept Getter- und Setter-Methoden (accessor and mutator methods)


Der Sinn von Getter- und Setter-Methoden ist es, die internen Klassenvariablen gegen Zugriff von außen zu schützen.

Getter- und Setter-Methoden sind 'normale' Methoden, durch die man von Außen auf die Attribute lesend oder schreibend zugreifen und diese ggf. dabei noch ändern kann.

Anwendung: Konzept der Kapselung

Das Konzept der Kapselung ist ein grundlegendes Prinzip der objektorientierten Programmierung und spielt eine wichtige Rolle bei der Strukturierung und Organisation von Code. Es hilft, die Komplexität von Code zu reduzieren und die Wartbarkeit und Wiederverwendbarkeit von Code zu verbessern. Es ist wichtig, das Interface einer Klasse stabil zu halten, um die Effizienz der Codeentwicklung zu maximieren.

Als **Datenkapselung** (englisch *encapsulation*, nach David Parnas auch bekannt als **information hiding**) bezeichnet man in der Programmierung das **Verbergen** von *Daten* oder *Informationen* vor dem Zugriff von außen.

Der direkte Zugriff auf die interne Datenstruktur wird unterbunden und erfolgt stattdessen über **definierte Schnittstellen** (*Black-Box-Modell*).  [Wikipedia](#)

Kapselung

- **Datenkapselung** (auch bekannt als *information hiding*) bezeichnet das Verbergen von Daten oder Informationen vor dem Zugriff von außen.
- Der direkte Zugriff auf die interne Datenstruktur wird unterbunden und erfolgt stattdessen über definierte Schnittstellen (*Black-Box-Modell*).
- Ein grundlegendes Prinzip der objektorientierten Programmierung ist die **Kapselung**.
- Kapselung bedeutet, die tatsächliche Funktionalität eines Codeblocks in Funktionen oder in Methoden einer Klasse zu verbergen.
- Bei der sogenannten **Datenkapselung** werden die Attribute nach außen hin versteckt und der Zugriff auf sie über Methoden ermöglicht.
- Man spricht hier auch vom **Interface** einer Klasse, was nichts anderes als die Sicht von außen meint.
- Versuchen Sie immer, das Interface einer Klasse stabil zu halten. Jede Änderung an Methodenaufrufen zieht immer mehr Arbeit nach sich, je größer die Projekte werden.

Getter-Methoden

- Getter-Methoden, die üblicherweise durch das Präfix `get` gefolgt vom Namen des Attributs identifiziert werden, dienen dazu, den Wert eines Klassenattributs abzurufen.
- Dieser Ansatz entspricht den Standardbenennungskonventionen für Methoden.
- Die Verwendung von Getter-Methoden ermöglicht eine zusätzliche Verarbeitung (wie z.B. Änderung der Groß- und Kleinschreibung oder Berechnungen) vor der Ausgabe des Werts.

```
<?php
class User {
    protected $email = 'max@mustermann.de';
    // Verhindert direkten externen Zugriff

    public function getEmail() {
        return $this->email;
    }
}
$user = new User();
echo $user->getEmail();
// Gibt aus: max@mustermann.de
```

Setter-Methoden

- Im Gegensatz dazu sind Setter-Methoden, die typischerweise mit **set** beginnen, gefolgt vom Namen des Attributs, dazu bestimmt, den Wert eines Attributs zu ändern.
- Auch dies folgt den Standardbenennungskonventionen für Methoden.
- In der objektorientierten Programmierung wird empfohlen, Änderungen wann immer möglich intern innerhalb von Klassen vorzunehmen. Dieser Ansatz spart Zeit und reduziert Komplikationen, da er die Notwendigkeit vermeidet, Code zu ändern, der mit diesen Objekten interagiert.

```
<?php
class User
{
    protected $email = '';
    // Initialisiert mit einem leeren String

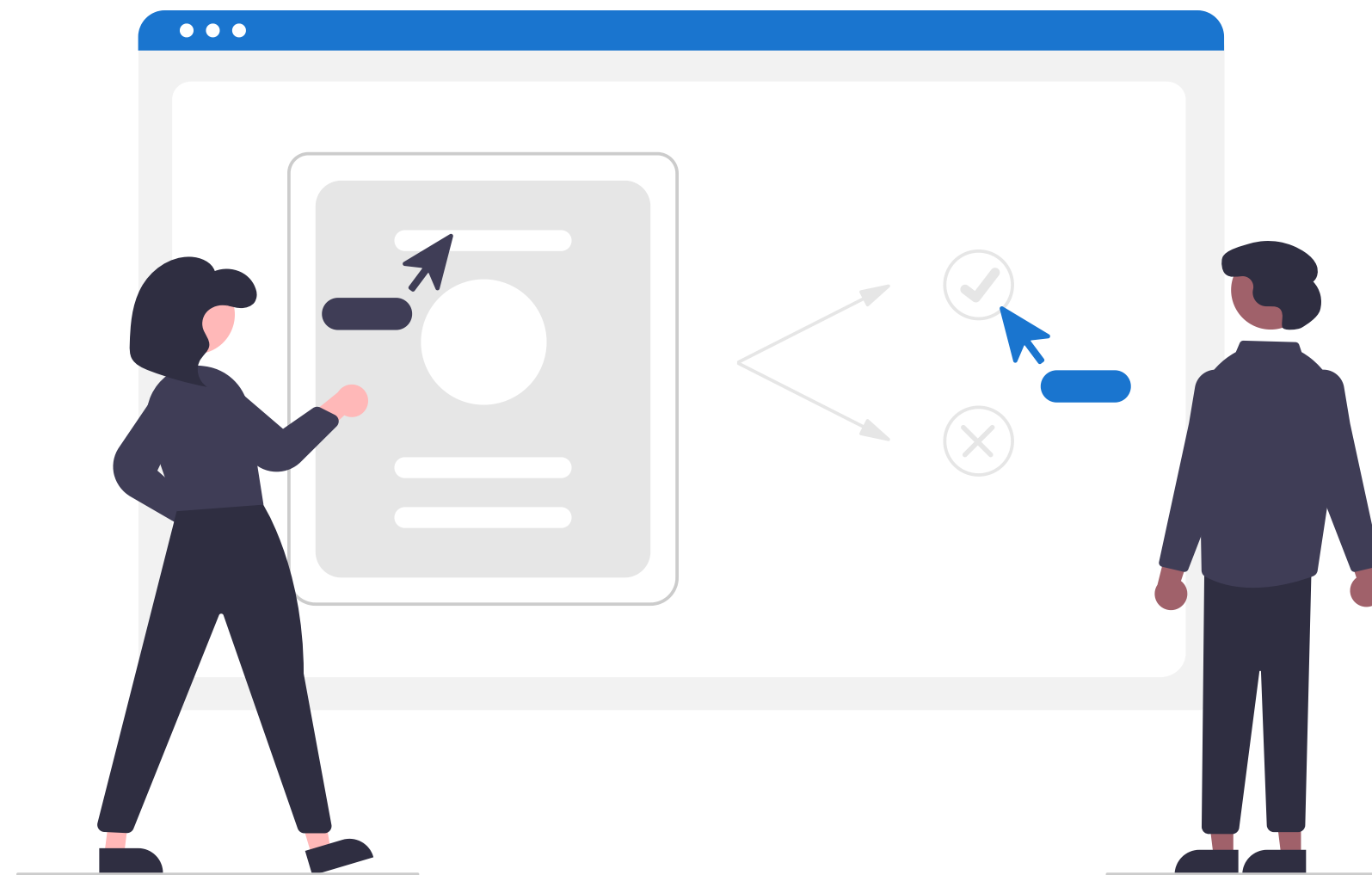
    public function getEmail()
    {
        return $this->email;
    }

    public function setEmail($email)
    {
        $this->email = $email;
        return $this;
    }
}

$user = new User();
$user->setEmail('erika@mustermann.de');
echo $user->getEmail(); // => erika@mustermann.de
```

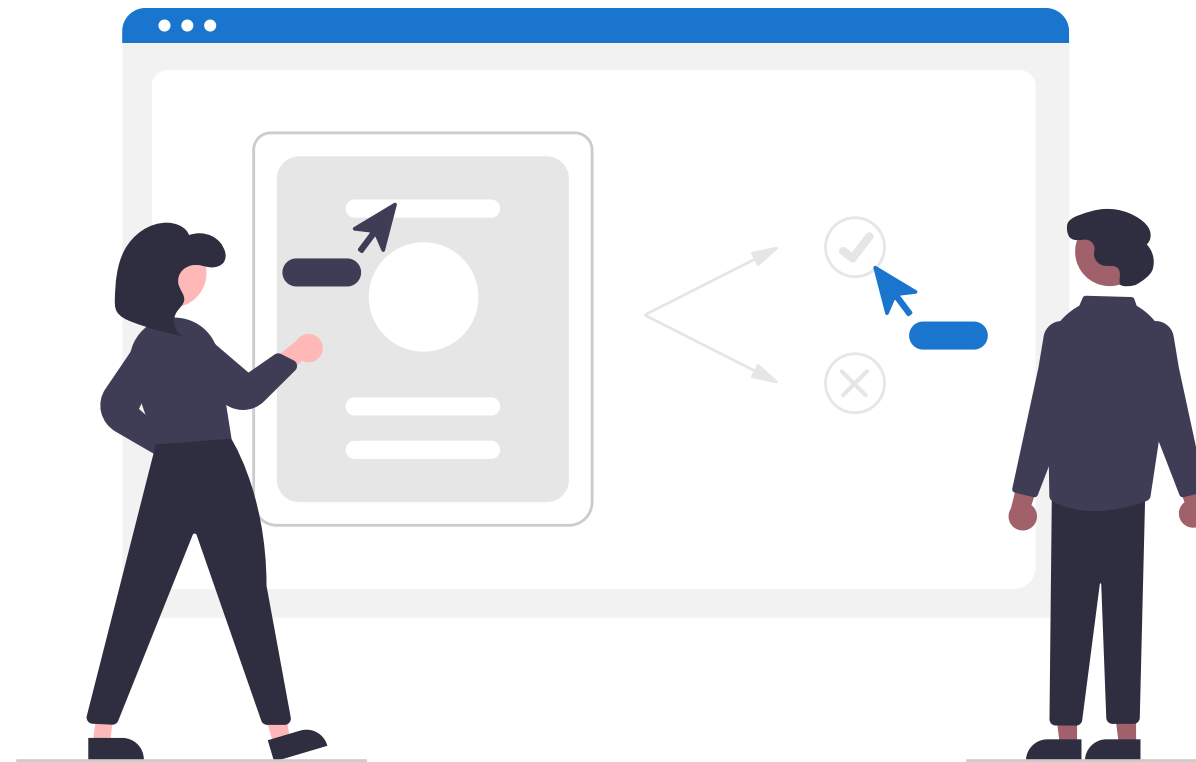

PraxisWorkshop - Getter- und Setter-Methoden

⇒ Beispiel für Getter- und Setter-Methoden in PHP



PraxisWorkshop - Arbeiten mit Objekten

- ⇒ Methoden innerhalb eines Objektes aufrufen
- ⇒ Objekte als lokale Variablen
- ⇒ Objekte als Parameter übergeben
- ⇒ Objekte in Attributen ablegen



Fazit: Arbeiten mit Objekten

Objekte

- lassen sich als lokale Variablen innerhalb von Methoden verwenden
- können als Parameter übergeben werden
- können in Attributen* gespeichert werden

*Objekte in Attributen ablegen

Objekte lassen sich analog den anderen Datentypen auch in den Attributen andere Objekte speichern (entweder durch eine direkte Zuweisung oder mittels eines Setters).

Eine Instanziierung bei der Initialisierung des Attributs geht nicht, da der zu setzende Wert ein konstanter Wert sein muss - d.h. dieser muss zum Kompilierungszeitpunkt ausgewertet werden können und darf nicht von Informationen abhängen, die erst zur Laufzeit zur Verfügung stehen!

Informationen zu Objekten abfragen

[1/3]

1.) Datentyp abfragen

a) mittels `gettype()`

```
gettype ( mixed $var ) : string
```

Liefert den Datentyp der Variablen var. Zur Typprüfung sollten die `is_*` Funktionen verwendet werden.



b) mittels `is_object()`

```
is_object ( mixed $var ) : bool
```

Prüft ob die gegebene Variable ein Objekt ist.



Informationen zu Objekten abfragen

[2/3]

2.) Informationen abfragen

a) mittels `var_dump()`

```
var_dump ( mixed $expression [ mixed $... ] ) : void
```

Die Funktion gibt strukturierte Informationen über einen oder mehrere Ausdrücke aus, darunter auch den entsprechenden Typ und Wert. Arrays und Objekte werden rekursiv durchlaufen und die jeweiligen Werte eingerückt dargestellt, um die Struktur zu verdeutlichen.

Alle öffentlichen (**public**), privaten (**private**) und geschützten (**protected**) **Eigenschaften** eines Objekts werden in der Ausgabe dargestellt, außer wenn das Objekt eine `__debugInfo()` Methode implementiert.



Informationen zu Objekten abfragen

[3/3]

b) mittels `print_r()`

```
print_r ( mixed $expression [, bool $return = FALSE ] ) : mixed
```

`print_r()` zeigt Informationen über eine Variable in menschenlesbarer Form an.

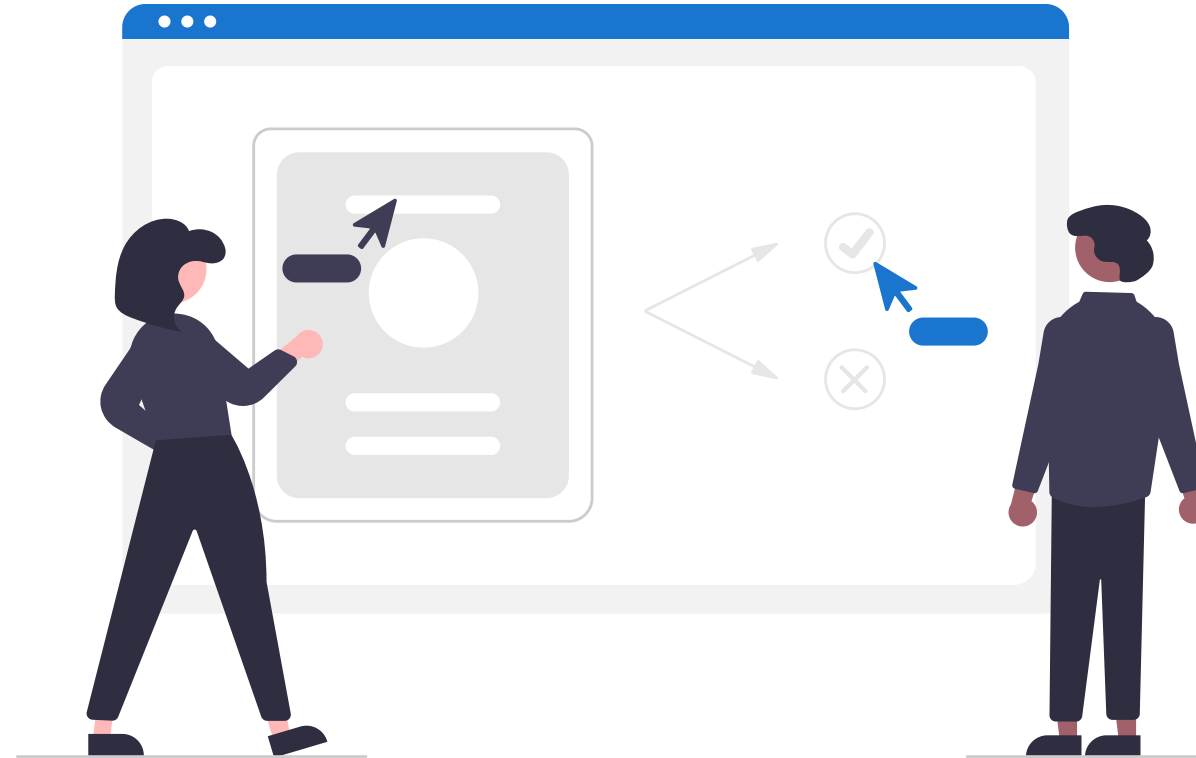


c) weitere Möglichkeiten

- `var_export` — Gibt den Inhalt einer Variablen als parsbaren PHP-Code zurück, d. h. es liefert strukturierte Informationen zum Inhalt der übergebenen Variable. Das Verhalten ist ähnlich dem der `var_dump()`, allerdings ist hier das Ergebnis valider PHP-Code mit dem sich der Inhalt der Variable wieder herstellen lässt.
- `__debugInfo()` - Diese (magische) Methode wird von `var_dump()` aufgerufen, wenn ein Objekt ausgegeben wird, um die Eigenschaften auszulesen die gezeigt werden sollen. Wenn diese Methode in einem Objekt nicht definiert ist so werden alle **Eigenschaften** die **public**, **protected** oder **private** sind angezeigt.

PraxisWorkshop - Informationen zu Objekten abfragen

- ⇒ mittels `gettype()`
- ⇒ mittels `is_object()`
- ⇒ mittels `var_dump()` / `print_r()`



Informationen zur Klasse (Klassenzugehörigkeit) abfragen

a) mittels des **Typ-Operator** `instanceof`

```
(Objekt instanceof Klasse) : bool
```

`instanceof` wird dazu verwendet um festzustellen, ob ein gegebenes Objekt ein Objekt ist, das zu einer bestimmten Klasse gehört.



b) mittels `get_class()`

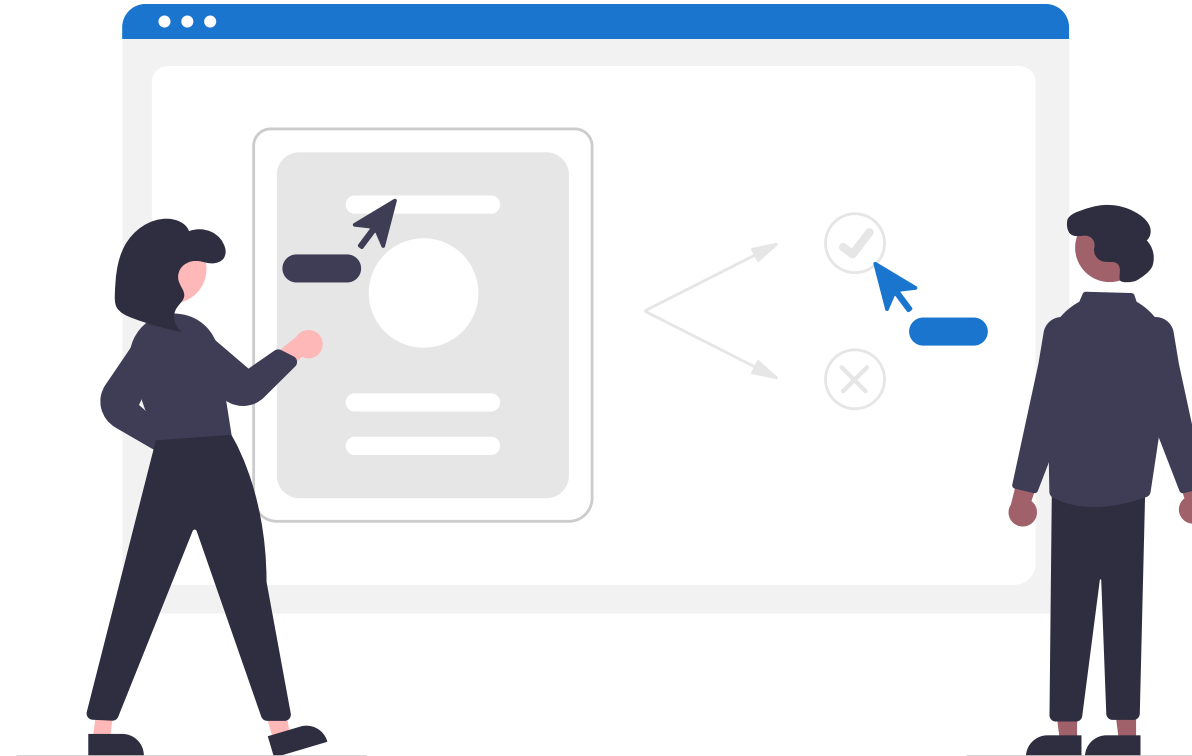
```
get_class ([ object $object ] ) : string
```

Ermittelt den Klassennamen für das übergebene object.



PraxisWorkshop - Informationen zur Klasse (Klassenzugehörigkeit) abfragen

- ⇒ mittels des **Typ-Operator** `instanceof`
- ⇒ mittels `get_class()`
- ⇒ mittels der magischen Konstanten `__CLASS__`



Einstellmöglichkeiten des Fehlerlevels und der Fehlerausgabe

- **Fehlerlevel festlegen:** Das Fehlerlevel bestimmt, welche Arten von Fehlern angezeigt werden sollen. Es gibt verschiedene vordefinierte Konstanten, die wir verwenden können, um das gewünschte Fehlerlevel festzulegen.

```
// Alle Fehler anzeigen  
error_reporting(E_ALL);
```

```
// Nur kritische Fehler anzeigen  
error_reporting(E_ERROR | E_PARSE);
```

```
// Keine Fehler anzeigen  
error_reporting(0);
```



Einstellmöglichkeiten der Fehlerausgabe

- **Fehlerausgabe steuern:** Die Fehlerausgabe bestimmt, wohin die Fehlermeldungen gesendet werden sollen. Es gibt verschiedene Optionen, die wir verwenden können, um die Fehlerausgabe zu steuern.

```
// Fehlermeldungen auf dem Bildschirm anzeigen
ini_set('display_errors', 1);

// Fehlermeldungen in eine Log-Datei schreiben
ini_set('log_errors', 1);
ini_set('error_log', '/pfad/zur/logdatei.log');

// Fehlermeldungen an eine benutzerdefinierte Funktion senden
function customErrorHandler($errno, $errstr, $errfile, $errline) {
    // Benutzerdefinierte Fehlerbehandlung
}
set_error_handler('customErrorHandler');
```

Weiteres zu Fehlerlevel und Fehlerausgabe

Es ist wichtig zu beachten, dass die Einstellungen für das Fehlerlevel und die Fehlerausgabe in der PHP-Konfigurationsdatei (`php.ini`) vorgenommen werden können. Wenn Sie jedoch nur bestimmte Einstellungen für eine bestimmte Anwendung ändern möchten, können Sie die oben genannten Funktionen und Direktiven verwenden, um die Einstellungen programmatisch festzulegen.

Indem wir das Fehlerlevel und die Fehlerausgabe in PHP richtig konfigurieren, können wir die Fehlerbehandlung und -überwachung in unserer Anwendung verbessern und so zu einer stabileren und fehlerfreien Software beitragen.

Laufzeit-Konfiguration

PHP bietet des Weiteren einige Funktionen zur Fehler und Logging-Konfiguration.

 [php.net](https://www.php.net/manual/de/ini.configuration.php) Laufzeit-Konfiguration

Typdeklarationen (Wdh.)

Typdeklarationen erlauben es, dass Funktionsparameter zur Aufrufzeit von einem bestimmten Typ sind. Ist der gegebene Wert von einem falschen Typ dann wird ein Fehler generiert.

Typdeklarationen können zu Funktionsargumenten, Rückgabewerten und, ab PHP 7.4.0, Klasseneigenschaften hinzugefügt werden. Sie stellen sicher, dass der Wert zum Zeitpunkt des Aufrufs vom angegebenen Typ ist, andernfalls wird ein `TypeError` ausgelöst.

Jeder einzelne Typ, den PHP unterstützt, mit Ausnahme von „resource“, kann innerhalb einer User-Land-Typdeklaration verwendet werden. Diese Seite enthält ein Änderungsprotokoll zur Verfügbarkeit der verschiedenen Typen und eine Dokumentation zu deren Verwendung in Typdeklarationen.

Hinweis : Wenn eine Klasse eine Schnittstellenmethode implementiert oder eine Methode erneut implementiert, die bereits von einer übergeordneten Klasse definiert wurde, muss sie mit der oben genannten Definition kompatibel sein. Eine Methode ist kompatibel, wenn sie den [Varianzregeln](#) folgt .



Sichtbarkeit (Wdh.)

Die **Sichtbarkeit** einer **Eigenschaft**, **Methode** oder (von PHP 7.1.0 an) einer **Konstante** kann definiert werden, indem man der Deklaration eines der Schlüsselwörter **public**, **protected** oder **private** voranstellt.

- Auf **public** deklarierte Elemente kann von überall her zugegriffen werden.
- **protected** beschränkt den Zugang auf Elternklassen und abgeleitete Klassen (sowie die Klasse, die das Element definiert).
- **private** grenzt die Sichtbarkeit einzig auf die Klasse ein, die das Element definiert.



⇒ Klasseneigenschaften (**Attribute**) und **Methoden** müssen als **public**, **private** oder **protected** definiert werden, anderenfalls sind sie immer public (gilt insbesondere auch bei Deklarationen mit dem Schlüsselwort **var** (veraltet))!

Typsicherheit in PHP (Wdh.)

Typsicherheit in PHP

- PHP ist schwach typisiert, es ist keine explizite Deklaration für eine verwendete Variable erforderlich.
- Jede Variable hat zu jedem Zeitpunkt einen genau definierten Typ.
- Typdeklaration (*type hints*) ermöglicht typsicheres Programmieren bis zu einem gewissen Grad.
- Limitierung: Implizite Typumwandlung bei skalaren Datentypen (wenn bei nicht strenger Typprüfung).

Empfehlung für höhere Typsicherheit

- Attribute (Objektvariablen) sollten nie direkt gesetzt werden.
- Verwendung von Setter-Methoden für Typprüfung und ggf. Typumwandlung.
- Verwendung von Typdeklarationen und strenger Typprüfung

weitere Möglichkeit \Rightarrow Konzept der virtuellen Attribute

Konzept der virtuellen Attribute

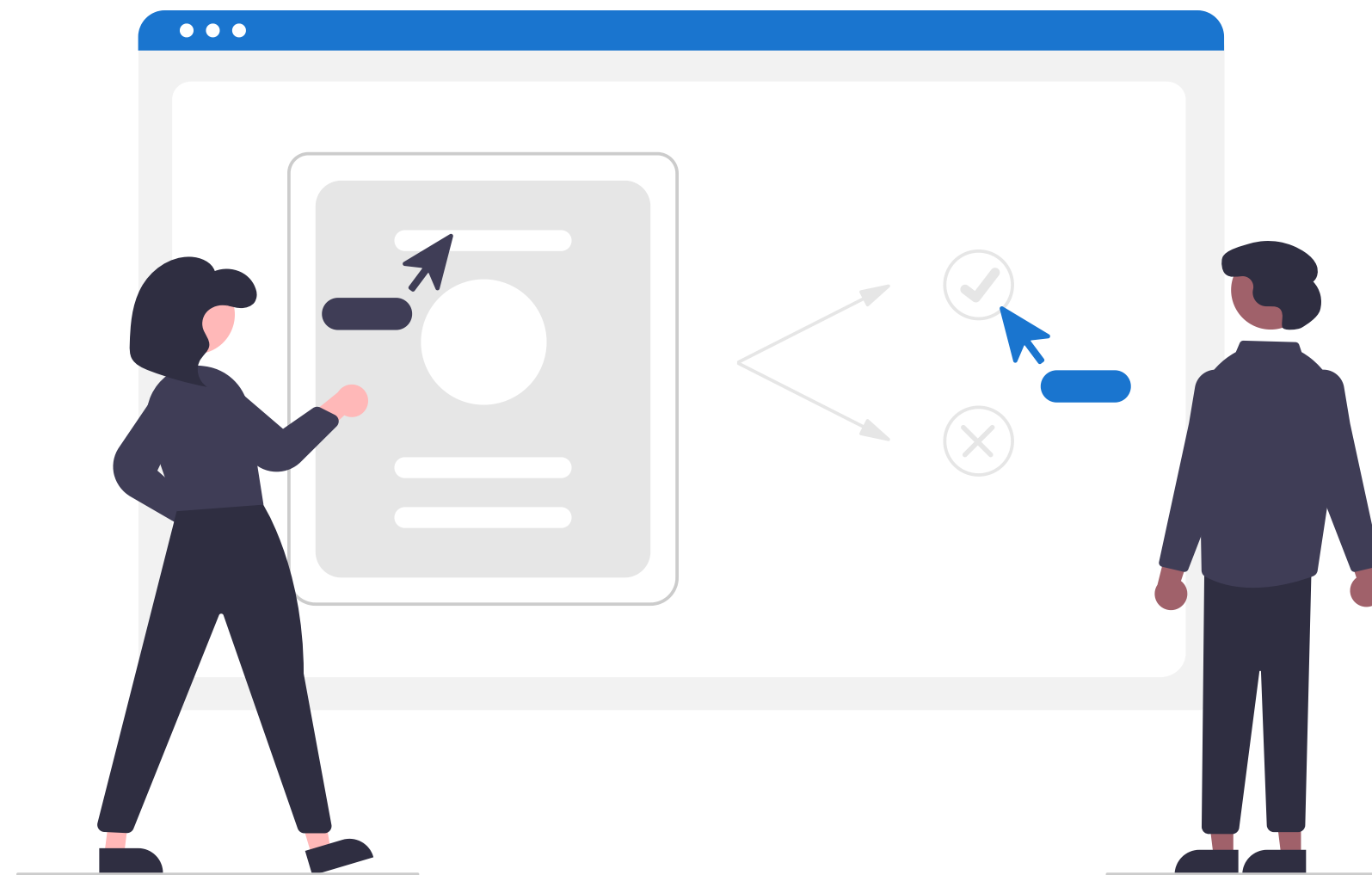
Das Konzept der virtuellen Attribute in PHP bezieht sich in der Regel auf eine Technik in der objektorientierten Programmierung, bei der man scheinbar Attribute in einer Klasse hat, die tatsächlich aber keine physischen Eigenschaften der Klasse sind. Stattdessen werden sie dynamisch über Methoden erstellt und verwaltet.

- **Methoden statt Eigenschaften:** Anstelle von festen Klasseneigenschaften verwendet man Methoden (üblicherweise Getter und Setter), um auf diese "virtuellen" Attribute zuzugreifen. Diese Methoden sind so benannt, dass sie wie der Zugriff auf normale Klasseneigenschaften aussehen.
- **Berechnete Werte:** Oft repräsentieren virtuelle Attribute Werte, die aus anderen, tatsächlichen Attributen der Klasse berechnet werden. Zum Beispiel könnte ein Person-Objekt echte Attribute für Vorname und Nachname haben, aber ein virtuelles Attribut für den vollen Namen, das durch eine Methode generiert wird, die diese beiden kombiniert.
- **Keine Speicherung im Objekt:** Diese virtuellen Attribute werden nicht als Teil des Objektzustands gespeichert. Sie werden bei jedem Aufruf ihrer entsprechenden Methoden neu berechnet oder abgerufen.

Virtuelle Attribute ermöglichen eine flexible und dynamische Art des Umgangs mit Objekteigenschaften, insbesondere in komplexen Anwendungen, wo zusätzliche Berechnungen oder Datenmanipulationen notwendig sind.

PraxisWorkshop - virtuelle Attribute

⇒ Konzept der virtuelle Attribute am Beispiel einer Klasse **Book**



„Erzähle mir und ich vergesse. Zeige mir und ich erinnere. Lass es mich tun und ich verstehe.“ (Konfuzius, 551 bis 479 v. Chr.)



Zu dieser Dokumentation / Lizenzbedingungen

Dieses Dokument ist eine begleitende Dokumentation zu den Workshops und wird fortlaufend kontextbezogen angepasst bzw. ergänzt

→ *Living Document.*

Es hat ausdrücklich nicht den Anspruch eines Lehrbuchs, sondern soll lediglich als Orientierungshilfe für die Durchführung und Wiederholung der Workshops sowie als Einstiegshilfe für eigene weitere Recherchen verstanden werden.

Alle Inhalte dieses Dokuments, sofern nicht selbst erstellt oder nicht anders angegeben, stammen aus Quellen mit frei zugänglichen Inhalte, die unter der Creative Commons Attribution-ShareAlike-Lizenz veröffentlicht oder speziell für die Erstellung von Unterrichts- bzw. Studienmaterial freigegeben sind.

Es sollen ausdrücklich diese Inhalte nicht als Eigene dargestellt und ausgegeben werden!

Sollte trotz Überprüfung eine Urheberrechtsverletzung vorliegen, bitte ich um kurze Information, damit diese umgehend beseitigt werden kann.

Dieses Dokument ist ausschließlich für Schulungszwecke erstellt und teilweise urheberrechtlich geschützt bzw. unterliegt tw. urheberrechtlichen Einschränkungen. Somit sind die Weitergabe sowie die Vervielfältigung dieses Dokuments als Ganzes, aber auch die Verwertung und Mitteilung seines Inhalts nicht erlaubt, soweit nicht ausdrücklich durch die angegebene Lizenz gestattet.

(s. a. ggf. Lizenzangaben im begleitenden Script/Handout)