PHP OOP, MySQL Advanced & MVC

SAE Wien
Alexander Hofbauer
hofbauer.alexander+sae@gmail.com

PHP OOP

+ Objektorientierte Programmierung
+ Klassen (z.b. Mensch)
    + Eigenschaften/Properties
    + Methoden/Methods
    + magische Methoden
+ Instanzierte Klassen: Objekte (z.b. Kerstin, Diana, Thomas)
+ Vererbung, Sichtbarkeit, Klassenkonstanten, `Static`, Interfaces, Abstrakte Klassen, Traits, `Final`, Type Hints, Autoloading
+ Namespaces

https://www.php.net/manual/de/language.oop5.php

Klassen

```php
<?php // Human.php

class Human {

    public $name = 'Adam or Eve';

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

    public function getName () {
        echo 'Name: ' . $this->name;
    }

}
```

```php
<?php // something.php

$jeannine = new Human(
    'Jeannine'
);

$jeannine->getName();
// Name: Jeannine
```

Magische Methoden

```
__construct(); // wenn ein neues Objekt aus einer Klasse instanziert wird
__destruct(); // wenn es keine Referenzen mehr auf ein Objekt gibt
__call(); // wenn eine geschützte Methode aufgerufen wird
__callStatic(); // wenn eine geschützte Methode statisch aufgerufen wird
__get(); // wenn geschützte Eigenschaften gelesen werden sollen
__set(); // wenn geschützte Eigenschaften geschrieben werden sollen
__isset(); // wenn geschützte Eigenschaften mit isset() oder empty() geprüft werden
__unset(); // wenn geschützte Eigenschaften mit unset() gelöscht werden
__sleep(); // bevor ein Objekt serialisiert wird
__wakeup(); // nachdem eine Objekt deserialisiert wurde
__serialize(); // wenn ein Objekt serialisiert werden soll
__unserialize(); // wenn ein Objekt deserialisiert wird
__toString(); // wenn ein Objekt in einen String konvertiert wird (bspw. echo)
__invoke(); // wenn ein Objekt als Funktion aufgerufen wird
__set_state(); // wenn ein Objekt mit var_export() verarbeitet wird
__clone(); // wenn ein Objekt geklont wird
__debugInfo(); // wenn ein Objekt mit var_dump() untersucht wird
```

Constructor Property Promotion

```php
<?php // PHP 7

class Point {
  public float $x;
  public float $y;
  public float $z;

  public function __construct(
    float $x = 0.0,
    float $y = 0.0,
    float $z = 0.0
  ) {
    $this->x = $x;
    $this->y = $y;
    $this->z = $z;
  }
}
```

```php
<?php // PHP 8

class Point {
  public function __construct(
    public float $x = 0.0,
    public float $y = 0.0,
    public float $z = 0.0,
  ) {}
}
```

Vererbung

```php
<?php // Human.php

class Human {

    public $name = 'Adam or Eve';
    public $gender = null;

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

    public function getName () {
        echo 'Name: ' . $this->name;
    }

}
```

```php
<?php // Man.php

class Man extends Human {

    public $gender = 'male';

    public function __construct($name) {
        parent::__construct($name);
    }

    public function growBeard () { /* ... */ }

}
```

```php
<?php // something.php

$thomas = new Man('Thomas');
$thomas->growBeard();
$thomas->getName(); // Name: Thomas
```

Sichtbarkeit

```php
<?php

class Foo {

    public $public = "public";
    protected $protected = "protected";
    private $private = "private";

    public function f1 () { /* ... */ }
    protected function f2 () { /* ... */ }
    private function f3 () { /* ... */ }

}
```

```php
<?php

$bar = new Foo();

echo $bar->public; // "public"
echo $bar->protected; // ERROR
echo $bar->f3(); // ERROR
```

Nullsafe Operator

```php
<?php // PHP 7

$country = null;

if ($session !== null) {
  $user = $session->user;

  if ($user !== null) {
    $address = $user->getAddress();

    if ($address !== null) {
      $country = $address->country;
    }
  }
}
```

```php
<?php // PHP 8

$country =
$session?->user?->getAddress()?->country;

?>
```

Klassenkonstanten & `static`

```php
<?php

class Foo {

    const FOO = "BAR";

    public $public = "regular p";
    public static $staticP = "static p";

    public function f1 () { /* ... */ }
    public static function f2 () { /* ... */ }

}
```

```php
<?php

$bar = new Foo();

echo Foo::FOO; // "BAR"

echo $bar->public; // "regular p"
echo Foo::$staticP; // "static p"

echo $bar->f1();
echo Foo::f2();
```

Interfaces

```php
<?php

interface Foo {

    public function f1 ($param1);
    public static function f2();

}

class Foobar implements Foo {

    public function f1 ($param1) { /* ... */ }
    public static function f2 () { /* ... */ }

}
```

+ Mit Interfaces kann definiert werden, welche Funktionen eine Klasse implementieren muss
+ Nützlich, wenn eine “Vorlage” für andere DeveloperInnen erstellt werden soll

Class Abstraction

```php
<?php

abstract class Foo {

    /**
     * Erweiternde Klasse zwingen, diese Methoden zu
     * implementieren
     */
    abstract public function f1 ();
    abstract public static function f2 ();

    // Common method
    public function something () { /* ... */ }

}
```

+ Abstrakte Klassen können nicht instanziert werden
+ Bieten die Möglichkeit zu definieren, welche Funktionen eine Kind-Klasse implementieren muss
+ Methoden, die `abstract` definiert sind, dürfen keinen Funktions-Körper haben

Traits

```php
<?php

trait Foo {

    public function hello () {
        return "Hello";
    }

}

class Bar {
    use Foo;

    public function barfoo () {
        echo $this->hello() . ' World!';
    }

}
```

+ Gruppieren von Funktionen, um sie einfach wiederverwenden zu können
+ Löst Probleme der einfachen Vererbung

```php
<?php

$something = new Bar();
$something->barfoo(); // "Hello World!"
```

Final

```php
<?php

class Foo {

   final public function foobar () { /* ... */ }

}

class Bar extends Foo {

   public function foobar () { /* ... */ }

}

// Fatal error: Cannot override final method Foo::foobar()
```

+ verhindert, dass eine Methode von einer Kind-Klasse überschrieben wird

Type Hints

```php
<?php

function foobar ( string $name, array $hobbies ): object {
    /* ... */
}


// mit Klassennamen

function something ( Human $human ): string  { /* ... */ }
```

- Konvertierung von Parametern und Rückgabewerten
- Strict Mode
  - verlangt die manuelle Konvertierung
  - pro Datei: `declare(strict_types=1);`
- Typen
  - Class/Interface name & self
  - array
  - callable
  - bool
  - float
  - int
  - string
  - iterable
  - object

Autoloading

```php
<?php

spl_autoload_register(function ($class_name) {
    include $class_name . '.php';
});

$foo = new Foo();
$bar = new Bar();
```

Namespaces

```php
<?php

namespace App\Controllers;

class Foo {

    public $public = "public";
    protected $protected = "protected";
    private $private = "private";

    public function f1 () {}
    protected function f2 () {}
    private function f3 () {}

}
```

+ Klassen mit gleichen Namen sind im selben Skriptaufruf möglich, wenn sie einen anderen Namespace haben
+ Ähnlich des Ordnersystems eines Betriebssystems

```php
<?php

$something = new App\Controllers\Foo();
```

MySQL Advanced

+ SQL Injections
+ Prepared Statements
+ JOINs

MySQL Injections

```php
<?php

$id = $_GET['id']; // ' OR 1=1

$query = "SELECT * FROM `users` WHERE `id` = '$id'";
// "SELECT * FROM `users` WHERE `id` = '' OR 1=1"

$result = mysqli_query($link, $query);
```

+ MySQL Injections können verwendet werden um Daten unerlaubt auszulesen oder auch zu schreiben
+ Queries mit dynamischen Parametern MÜSSEN IMMER als Prepared Statements abgeschickt werden!!

Prepared Statements

```php
<?php

$stmt = $link->prepare("SELECT * FROM users WHERE id = ?");

$stmt->bind_param('i', $id);
$stmt->execute();

$result = $stmt->get_result();

if ($result->num_rows === 1) {
    $result = $result->fetch_all(MYSQLI_ASSOC)[0];

    $user = new User($result);

    return $user;

} // ...
```

JOINs

```
SELECT * FROM tabelle1
 LEFT JOIN tabelle2
    ON tabelle1.Spaltenname = tabelle2.Spaltenname
 LEFT JOIN tabelle3
    ON tabelle1.Spaltenname = tabelle3.Spaltenname
WHERE ...
```

+ LEFT JOIN: Alle Datensätze aus der FROM-Tabelle werden mit möglichen Datensätzen aus den JOIN-Tabellen kombiniert. Wird in den JOIN-Tabellen kein passender Datensatz gefunden, so werden die Felder als `null` angegeben
+ INNER JOIN: In allen Tabellen müssen passende Datensätze gefunden werden
+ RIGHT JOIN: kann als LEFT JOIN umgeschrieben werden und ist daher relativ sinnlos

MVC

+ Model View Controller
+ Design Pattern
+ Model: Datenbankabstraktion (inkl. Business logic)
+ Controller: Steuerung/Logik
    + CRUD
+ View: Ausgabe/Darstellung
+ Laravel, Symfony, Ruby on Rails, Django, …

Copyright

Referenzen: http://php.net

Autor: Alexander Hofbauer <hofbauer.alexander+sae@gmail.com>