Objektorientierte Programmierung mit PHP

xobe|xduo@zhaw.ch

prozedural zu objektorientiert

- Warum objektorientierte Programmierung?
  - Programme wurden immer komplexer
  - Abbildung der realen Welt
- Bündelung von Attribute und Methoden/ Funktionen
- Wiederverwendung der Programmierer selbst, arbeiten im gewohnten Umfeld
- Übernehmen der Denkweise und Wiederverwendung der Module

Grundelemente der Objektorientierung (1)

- Klasse - Abstrakte Beschreibung der Objekte

- Abstraktion - Ein Objekte ist eine konkrete Ausprägung einer Klasse. Ein Objekt kann als abstraktes Modell eines ”Arbeiters” betrachtet werden.

- Kapselung - Verbergen von Informationen. Der Zugriff erfolgt über definierte Schnittstelle (Methoden) auf Daten, die dadurch gekapselt werden.

- Polymorphie - Vielgestaltigkeit. Verschiedene Objekte reagieren unterschiedlich auf die gleiche Nachricht. Zur Laufzeit -> dynamische Polymorphie.

- Vererbung - Abgeleitete Klasse besitzt die Objekte und Methode der Basisklasse -> erben.

Grundelemente der Objektorientierung (2)

- Assoziation - Zwei unabhängige Objekte arbeiten zusammen um ein gemeinsames Ziel zu erreichen.

- Komposition - Ein Objekt enthält ein anderes wobei das enthaltene Objekte nicht selbstständig existieren kann. Implementiert die Assoziation.

- Lebensdauer - Die Dauer vom Erzeugen bis zum Zerstören eines Objektes. Bei einer Komposition sind diese identisch, nicht bei einer Assoziation

- Besitzer - Das erzeugende Objekt ist der Besitzer. Bei der Assoziation kann die Besitzerrolle wechseln oder überhaupt keinen Einfluss haben, nicht bei der Komposition.

Begriffe (1)

- **Klasse, Vorlage, Template**
  - Vorlage für ein Objekt. Beinhaltet die Eigenschaften des Objektes und Methoden. Klassen können von anderen Klassen Erben.
- **Subklasse, Unterklasse, Kindklasse**
  - Eine Klasse, die sich in der Klassenhierarchie weiter unten befindet als die betrachtete.
- **Superklasse, Oberklasse, Elternklasse**
  - Eine Klasse, die sich in der Klassenhierarchie weiter oben befindet, als die betrachtete.
- **Objekt, Instanz, Abbildung**
  - Eine Instanz einer Klasse. Ermöglicht die Verwendung von Eigenschaften und Methoden.

Begriffe (2)

- **Eigenschaft, Merkmal, Attribut**
  - Anlegen und Auslesen von Daten, die sich auf ein bestimmtes Objekt einer Klasse beziehen.
- **Methode, Fähigkeit, Funktion**
  - Eine Gruppe von Anweisungen in einer Klasse, die definieren, wie sich die Objekte der Klasse verhalten.
- **Klassenvariable**
  - Eine Variable, die ein Attribut einer ganzen Klasse anstelle einer bestimmten Instanz einer Klasse beschreibt.
- **Instanzvariable, Member-Variable**
  - Eine Variable, die ein Attribut einer Instanz einer Klasse beschreibt.

Klassen in PHP (1)

- Werden in PHP mit dem Schlüsselwort `class` eingeleitet
- Das Erzeugen eines Objektes wird als instanzieren bezeichnet

```php
class Klassenname {
 var $a; // Eigenschaft1
 var $b; // Eigenschaft2
// Konstruktor
 function Klassenname() {
  $this->a = wert_a;
 }
}
$klasse = new Klassenname();
```

Klassen in PHP (2)

- Einsatz von `class`
  - Klassendefinition wird mit class eingeleitet
  - Darauf folgt der Klassenname
  - Der gesamte Klassenrumpf muss in geschweiften Klammern stehen
- Einsatz von `$this`
  - Innerhalb einer Klasse auf sich selbst verweisen
  - vor this ein $-Zeichen, danach der Verweis mit dem spez. Operator ->
  - Achtung: `this->$var` oder `$this->$var` ist falsch!
- Einsatz von `var`
  - Die Namen sind lokal in dieser Klasse
  - Ist eine Objekteigenschaft und kann auch in Methoden verwendet werden
- Einsatz von `new`
  - Erstellt eine Instanz der Klasse, wobei die zugewiesene Variable auf diese verweist

Vererbung in PHP

- Existierende Klassen erweitern, anstelle diese komplett neu zu erstellen

```php
// Klasse 1
// Oberklasse
class Klassenname1 {
var $a;
...
}
// (abgeleitet) Erweiterte Klasse 2 – erbt von Klasse 1
// Unterklasse erbt von Oberklasse
class Klassenname2 extends Klassenname1 {
var $b;
...
}
```

Konstruktoren und Destruktoren

- Konstruktoren erstellen einen Initialisierungszustand (Anfangszustand)
  - Eine Methode, die immer beim erzeugen aufgerufen wird
- Um einen Konstruktor zu erstellen, benötigt es kein zusätzliches Schlüsselwort
  - Der Name der Konstruktormethode ist derselbe, wie derjenige der Klasse
  - Bei einer `return` Anweisung geht der Wert beim Instanzieren verloren
    - Der Konstruktor kann aber nach dem instanzieren erneut aufgerufen werden
- Destruktoren gibt es in PHP in einer spez. Form (genauere Betrachtung folgt später)
  - Keine Komplexe Speicherverwaltung, Objekte belegen Speicher solange, wie das Skript läuft
  - Eine Speicherverwaltung erfolgt durch die ZendEngine im Hintergrund

Aufruf einer Klassenfunktion (1)

- In der Unterklasse den Konstruktor der Oberklasse aufrufen
- Optimierung durch `parent::Konstruktorname`
  - Kann nützlich sein bei Änderung der Klassenhierarchie

```
class Chip {
 function Chip() {
  echo "Erstelle einen Chip.<br>";
 }
}
class Chipsaetze extends Chip {
 function Chipsaetze() {
  Chip::Chip();
  echo "Mach einen Chipsatz daraus.<br>";
} }
$chipsatz = new Chipsaetze;
```

Aufruf einer Klassenfunktion (2)

```php
class Chip {
 function Chip() {
  echo "Chip wurde produziert.<br>";
 }
 function produzieren($anzahl) {
  $anzahl++;
  return($anzahl);
} }
class Chipsaetze extends Chip {
 function Chipsaetze() {
  Chip::Chip();
  echo "Chipsatz wurde produziert.<br>";
 }
 function produzieren($anzahl) {
  $anzahl = Chip::produzieren($anzahl);
  $anzahl = $anzahl*$anzahl;
  return($anzahl);
} }
```

Metainformationen über Klassen

- `get_class()`, `get_parent_class()` und `get_class_methods()`
  - Ermitteln von Klassenname, Klassenname der übergeordneten Klasse und ermitteln der Klassenmethode
- `get_declared_classes()`
  - Ermittelt die deklarierten Klassen in einem Skript, möglichst spät im Skript zu verwenden
- `get_object_vars()` und `get_class_vars()`
  - Objekt- resp. Klassenvariabeln anzeigen
- `method_exists()` und `class_exists()`
  - Prüft auf Existenz der Methode, Prüft ob die Klasse deklariert wurde
- `is_a()` und `is_object()`
  - Prüft ob das Objekt von der angegebenen Klasse ist, Prüft ob es sich um ein Objekt handelt

OOP ab PHP5

- Eingeführt in PHP5
- Bietet wesentlich mehr Möglichkeiten in Bezug zur OOP
- Wird durch die Zend Engine 2 möglich
- Weitere Features werden folgen
  - Mehrfachvererbung, striktere Typisierung
- Wichtigste Änderung -> Objektzugriff `by reference`, nicht `by value`

Konstruktoren/Destruktoren (1)

- Konstruktoren werden ab PHP5 über Interzeptormethoden oder auch ”magischen Methoden” definiert
- `__construct()`
  - Kann aber auch der Name der Klasse sein
- `__destruct()`
  - Wird aufgerufen, sobald ein Objekt zerstört wird
    - Bei Skriptende
    - `unset()`
    - `NULL`
- Anwendungen für Desktruktoren
  - Speichern von Objekten
  - Trennen von Datenbankverbindungen
  - Setzen von Variabeln
- Konstruktor einer Superklasse ausführen: `parent::__destruct()`

Konstruktoren/Destruktoren (2)

```php
class Klasse {
 public function __construct() {
  echo "Befinden uns im Konstruktor.<br>";
  $this->name = "MeineKlasse";
 }
 public function __destruct() {
  echo "Zerstöre die Klasse: " . $this->name .
".<br>";
} }
$objekt = new Klasse();
```

Datenkapselung (1)

- `public, private` oder `protected`
  - Schlüsselwörter
  - Können mit Eigenschaften sowie mit Methoden verwendet werden
- Trennung von Nutzungs- und Implementierungsschicht
  - Ein Entwickler der eine Klasse eines anderen Entwickler verwendet braucht die internen Abläufe nicht zu kennen
  - Erstellen von tranparenten Klassen
  - Die Sourcen stehen immer zur Verfügung
- `public`
  - Ist der Standardwert, sollte kein Schlüsselwort verwendet werden
  - Objekte der Klasse können die Eigenschaften/Methoden sehen und verwenden
  - Aus Gründen der Abwärtskompatibilität kann die Angabe entfallen

Datenkapselung (2)

- `private`
  - Nur Objekte derselben Klasse können auf Eigenschaften/Methoden zugreiffen.
  - Abgeleitete Klassen können nicht darauf zugreiffen oder diese ausführen
- `protected`
  - Gleich wie private, jedoch können Subklassen auf die Eigenschaften/Methoden zugreiffen die in der Superklasse als protected deklariert sind

Zugriffsrecht	Zugriff aus gleicher Klasse	Zugriff von einer Subklasse	Zugriff von einer beliebigen Klasse
public	ja	ja	ja
protected	ja	ja	nein
private	ja	nein	nein

Datenkapselung (3)

- Oftmals entstehen die Zugriffsrechte bei der Programmierung selbst
- Verschiedene Strategien
  - Erst alles public, bei Problemen sukzesive ändern zu private oder protected
  - Erst alles private, danach sukzesive ändern zu public oder protected
- Zugriffswege auf Eigenschaften kontrollieren
  - Zugriffswege auf Methoden kontrollieren mit `__call()`
  - Setzen und Gebrauchen von Eigenschaften über eigene Methoden

```php
public function setName() {
 return $this->name;
}
public function getName($name) {
 $this->name = $name;
}
```

Objekte und Referenzen

- Orientierung an anderen Hochsprachen
  - `$name = $object->getName();`
  - Wird gelesen als, gehe zur Objektreferenz zur Variable `$object` und führe dort die Methode `getName()` aus
  - Vorteile in der Semantik, also der Schreibweise von Programmen
  - Performanceverbesserung, da `$object` als Referenz nur wenig speicher braucht

```
class Fahrzeug {
 function __construct($typ) {
  $this->name = $typ;
 }
 function produziereFahrzeug($obj, $name) {
  $obj->name = $name;
} }
$mobil = new Fahrzeug("PKW");
$mobil->ProduziereFahrzeug($mobil, "LKW");
echo $mobil->name;
```

Objekte klonen

- Wird ein Objekt einer anderen Variable zugewiesen, wird nur die Referenz kopiert
- Um Objekte zu kopieren verwendet man `clone`
  - `$new_object = clone $object;`
- Interzeptormethode `__clone()`
  - Vor dem Klonen wird überprüft, ob die Methode __clone() selbstständig definiert wurde
  - Der Programmierer übernimmt das Klonen selbstständig
  - Setzen von Standardwerten beim Klonen von Objekten
- Ist diese Methode nicht definiert, wird eine Kopie erstellt

Objekte klonen Beispiel

```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 = "Kannengiesser";
$obj->ort = "Berlin";
print $obj->id . "<br>";
$clone_obj = clone $obj;
```

Klassenvererbung

- Methoden, die als private deklariert sind, können nicht vererbt werden
  - Es wird unterschieden zwischen Überladen und Überschreiben
- Überladen, overloading
  - In der Unterklasse existiert eine Methode, mit anderer Parameterzahl und gleichem Namen
  - Überladene Methoden definieren den Funktionsumfang in einer anderen Weise
- Überschreiben, overriding
  - In der Unterklasse existiert eine Methode, mit gleicher Parameterzahl und gleichem Namen
  - Weitere Regeln beachten für das Überschreiben
  - Überschriebene Methoden erweitern i.d.R. den Funktionsumfang

Beispiel ”Überladen”

```php
class Klasse {
 public function ausgeben($inhalt) {
  echo "Ausgabe von Klasse (Oberklasse)";
  echo $inhalt . "\n<br />";
 }
}
class UnterKlasse extends MeineKlasse {
 public function ausgeben($inhalt, $autor) {
  echo "Ausgabe von UnterKlasse (Unterklasse)
\n<br />";
  echo $autor . ": ". $inhalt . "\n<br />";
} }
$objekt = new UnterKlasse();
$objekt->ausgeben("Dies ist ein
Satz.","Caroline");
```

Beispiel "Überschreiben"

```
class Klasse {
 public function ausgeben($inhalt, $autor) {
  echo "Ausgabe von Klasse (Oberklasse)\n<br />";
  echo $autor . ": ". $inhalt . "\n<br />";
 }
}
class UnterKlasse extends MeineKlasse {
 public function ausgeben($inhalt, $autor) {
  parent::ausgeben($inhalt, $autor);
  echo "Ausgabe von UnterKlasse (Unterklasse)
\n<br />";
  echo $autor . ": ". $inhalt . "\n<br />";
} }
$objekt = new UnterKlasse();
$objekt->ausgeben("Dies ist ein
Satz.","Caroline");
```

Regeln zum Überschreiben

- Methoden können einfacher überschrieben als überladen werden
- Das Überschreiben erfolgt nur, sollten die folgenden Regeln eingehalten sein:
  - Der Methodenname muss übereinstimmen
  - Die Parameterliste und der Rückgabetyp müssen übereinstimmen.
  - Eine überschreibende Methode darf im Zugriff nicht eingeschränkter sein als die Ursprungsmethode.
  - Eine überschreibende Methode darf keine anderen Ausnahmebedingungen veranlassen als die Originalmethode.

Finale Klassen

- Klasse, die nicht vererbt werden kann
- `private` kann nicht verwendet werden, da die Methoden nicht sichtbar wären
- Es ist auch möglich, nur einzelne Methoden als `final` zu deklarieren
- Wird eine mit final deklarierte Methoden übreschrieben/überladen resultiert ein fatal error
- Wird eine mit `final` deklarierte Klasse vererbt resultiert ein fatal error
  - Es macht Sinn, sämtliche Methoden in einer mit `final` deklarierten Klasse auch mit `final` zu deklarieren

```
final class Klasse {
 final public function __construct() {}
}
class AndereKlasse {
 final public function __construct() {}
}
```

Abstraktion von Klassen und Methoden (1)

- Abstrakte Klassen sind das Gegenteil von finalen Klassen
  - Nachdem ausdrücklichen Verbot, sind Abstrakte Klassen/Methoden ein ausdrückliches Gebot um Methoden zu erstellen
  - Es können nicht einzelne Methoden mit `abstract` deklariert werden, ohne die Klasse als `abstract` zu deklarieren.
  - Abstrakte Klassen können nicht instanziert werden
  - Es können keine zusätzlichen Methoden definiert werden

```php
abstract class AbstractClass {
 abstract protected function getValue();
 abstract protected function prefixValue($prefix);
 public function printOut() {
  print $this->getValue() . "\n";
 }
}
```

Abstraktion von Klassen und Methoden (2)

- Einige Besonderheiten von abstrakten Klassen
  - Von abstrakten Klassen kann kein Objekt instanziiert werden.
  - Von einer abstrakten Klasse kann nur abgeleitet werden.
  - Methoden abstrakter Klassen, die selbst als abstract definiert sind, müssen bei einer Ableitung implementiert werden.
  - 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.

Schnittstellen (1)

- Schnittstellen werden auf dieselbe Art deklariert wie Klassen
  - Werden jdeoch mit dem Schlüsselwort interface eingeleitet
  - Innerhalb der Schnittstelle dürfen keine Eigenschaften enthalten sein
  - Innerhalb der Schnittstelle dürfen nur Methodenköpfe geschrieben werden
    - Diese werden mit einem Semikolon abgeschlossen
- Schnittstellen werden auf Klassen nicht vererbt, sondern implementiert
  - Schnittstellen sind weiche abstrakte Klassen
- Beachten der schwachen Typisierung für Objekte
- Schnittstellen selbst können anderen Schnittstellen vererbt werden
- Das definieren von Schnittstellen macht in grossen Projekten Sinn mit vielen Entwicklern
  - Nur weil PHP5 Schnittstellen anbietet, müssen Sie die nicht gebrauchen

Schnittstellen (2)

```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;
} }
```

Statische Eigenschaften und Methoden (1)

- Statische Eigenschaften/Methoden werden direkt über den Klassennamen angesprochen
  - Es muss also kein Objekt instanziert werden
  - Das Schlüsselwort static wird verwendet um solche Eigenschaften/Methoden zu erzeugen
- Es kann verschiedene Einsatzfälle für statische Variablen/Methoden geben
  - Statische Mitglieder für Aufgaben, die keinen direkten Bezug zu den spezifischen Daten eines Objektes haben wie z.B. Umrechnungen
  - Statische Variablen werden oft für Verweiszähler benötigt, also Variablen, die allen instanzierten Objekten gleich sind
  - Statische Mitglieder ermöglichen den Aufruf ohne Instanzierung. Sollte von dem Objekte keine Instanz benötigt werden, macht dies auch Sinn
- Statische Mitglieder können auch als `public, private` oder `protected` gekennzeichnet werden

Statische Eigenschaften und Methoden (2)

```php
class Datenbank {
 protected $db;
 public static function verbindeDB($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;
} }
$zugang_a = Datenbank::verbindeDB("localhost", "root", "");
$zugang = new Datenbank();
$zugang_b = $zugang->oeffneDB();
```

Interzeptormethoden

- Werden in PHP auch magische Methoden genannt
  - Diese Methoden werden aufgerufen bei verschiedenen Events
    - `__construct()`, `__destruct()`, `__wakeup()`, `__sleep()`
  - Ein anderer Grund für einen Aufruf kann ein nicht deklariertes Mitglied/ Klasse sein
    - `__autoload()`, `__call()`, `__get()`, `__set()`, `__toString()`
  - Weitere Interzeptormethoden folgen später

Interzeptormethoden (1)

- `__autoload($className)` wird aufgerufen, wenn ein Objekt einer Klasse erzeugt werden soll, die Klasse aber nicht deklariert ist.
- `__call($methodName, $parameters)` wird aufgerufen, wenn eine nicht deklarierte Methode auf einem Objekt aufgerufen wird.
- `__callStatic($methodName, $parameters)` wird aufgerufen, wenn eine nicht deklarierte Methode in einem statischen Kontext auf einem Objekt aufgerufen wird.
- `__get($name)` wird aufgerufen, wenn lesend auf ein nicht gesetztes Attribut zugegriffen wird auf einem Objekt.
- `__set($name, $value)` wird aufgerufen, wenn eine nicht deklariertes Attribut gesetzt wird auf einer Klasse.
- `__toString()` wird aufgerufen, wenn eine Typumwandlung eines Objektes in ein String durchgeführt werden soll.

Interzeptormethoden (2)

- `__isset($name)` wird aufgerufen, wenn isset() oder empty() auf ein Attribut einer Klasse zugreift, das nicht deklariert worden ist.
- `__unset($name)` wird aufgerufen, wenn unset() auf ein Attribut einer Klasse zugreift, dass nicht deklariert worden ist.
- `__sleep()` wird aufgerufen, wenn serialize() auf ein Objekt angewendet wird. Der Programmierer muss somit selbst die serialisierung durchführen
- `__wakeup()` wird aufgerufen, wenn unserialize() auf ein Objekt angewendet wird. Der Programmierer muss somit selbst die deseriealisierung durchführen.
- `__invoke()` wird aufgerufen, wenn versucht wird, ein Objekt auszuführen
- `__set_state()` wird als statische Methode aufgerufen, sobald var_export() auf ein Objekt ausgeführt wird
- `__clone()` wird aufgerufen, sobald versucht wird, ein Objekt mittels clone zu kopieren.

__set und __get

- Kommt zum Einsatz, sollte auf eine Eigenschaft zugegriffen werden, die explizit definiert ist
- Sind weder __set() noch __get() deklariert, kommt es beim Zugriff zu Fehlern
- 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.

```php
class Natel {
 private $mobils = array( "Nokia" => 10, "Siemens" => 20 );
 public function __get($varname) {
  return $this->mobils[$varname];
 }
 public function __set($varname, $wert) {
  $this->mobils[$varname] = $wert;
} }
```

Dereferenzierung von Objekten

- Aus einer Methode/Funktion heraus ein Objekt zurückgeben
- Das zurückgegebene Objekt kann wiederum auf alle Klassenmethoden zugreiffen

```
class Person {
 var $Name;
 var $Vater;
 var $Mutter;
 function __construct($wert) {
  $this->Name = $wert;
 }
}
$objOpa = new Person('Armin');
$objVater = new Person('Wendelinus');
$objVater->Vater = $objOpa;
$objMatze = new Person('Matze');
$objMatze->Vater = $objVater;
echo "Matzes Opa heißt " . $objMatze->Vater->Vater-
>Name;
```

Mehrfachvererbung in PHP (1)

- Mehrfachvererbung ist grundsätzlich nicht möglich
- Mit Mehrfachvererbung kann man die Beziehungen von realen Objekten einfacher beschreiben
- Vergessene Methoden beim Design können leichter ergänzt werden
  - Teilweise führt dies zu sehr komplexen Beziehungsgeflechten
  - Wartbarkeit bleibt dadurch meistens auf der Strecke
- In PHP gibt es keine echte Mehrfachvererbung, über Schnittstellen realisiert
  - Bei dieser Art der unechten Mehrfachvererbung kann es zu Kollisionen kommen
  - Es können nur Methodendefinitionen vererbt werden

Mehrfachvererbung in PHP (2)

```php
interface Fahrzeug {
 public function fahren();
}
interface MaschinenFahrzeug {
 public function einschalten();
 public function ausschalten();
}
class Auto implements Fahrzeug, MaschinenFahrzeug {
 public function fahren() {
  echo "Auto fahren.\n<br />";
 }
 public function einschalten() {
  echo "Motor gestartet.\n<br />";
 }
 public function ausschalten() {
 echo "Motor ausgeschaltet.\n<br />";
 }
}
$einAuto = new Auto();
$einAuto->fahren();
$einAuto->einschalten();
$einAuto->ausschalten();
```

Weiterführende Literatur

- http://www.php.net/manual/de/language.oop5.php
- http://www.php.net/manual/de/language.oop5.magic.php#language.oop state
- http://www.php.net/manual/de/language.oop5.abstract.php
- http://www.php.net/manual/de/language.oop5.interfaces.php
- http://www.php.net/manual/de/language.oop5.overloading.php
- http://www.php.net/manual/de/language.oop5.decon.php

http://stackoverflow.com/questions/1279613/what-is-an-orm-and-where-can-i-learn-more-about-it