Vorlesung

Webbasierte Informationssysteme

(CS4130)

Objektorientierte Sprachkonstrukte in PHP

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

Objektorientierte Programmierung in PHP

- Vergleichbare Mächtigkeit von OOP-Sprachkonstrukten von PHP zu anderen OO-Programmiersprachen
- Details differieren

```php
abstract class Mammal {
  public function getNumberOfLegs() {
    return $this -> legs; // Achtung: Zugriff auf Variable der Kindsklasse
  }
}
class Dog extends Mammal {
  protected $legs = 4;
}
$waldi = new Dog;
echo $waldi -> getNumberOfLegs() . " legs";
```

OOP: Sichtbarkeit von Variablen und Methoden

- 3 Stufen:
  - `public`
    - Auf diese Variable/Methode kann immer auch von außen zugegriffen und im Fall von Variablen auch verändert werden
  - `private`
    - Auf diese Variable/Methode kann nur innerhalb dieser Klasse zugegriffen werden
  - `protected`
    - Auf diese Variable/Methode kann nur innerhalb dieser sowie deren Unter-/Oberklassen zugegriffen werden

OOP: Referenzierung

- Grundschema:
  - Variable: `$object -> varname`
  - Methode: `$object -> methodname(parameters)`
  - Repräsentation des aktuellen Objektes durch **`$this`**

  $\Rightarrow$ Zugriffe auf Variablen/Methoden innerhalb desselben Objektes mittels **`$this`** `->` …
- Beispiel:

```php
class Person {
  protected $name;
  function setName($var) {
    $this -> name = $var;
  }
}
$doerte = new Person;
$doerte -> setName('Dörte');
```

Achtung:
`$name = $var;`
würde (ohne Fehlermeldung) nur eine neue lokale Variable `$name` setzen!

Analog versucht `name(parameters)` eine Funktion `name` aufzurufen (und nicht eine Objektmethode)!

OOP: Neue Objektvariablen

- Vorsicht bei Tippfehlern:
  Objektvariablen können bei erster Verwendung (ohne Deklaration im Klassenrumpf) eingeführt werden
  - Fehlererkennung versus Flexibilität
- Beispiel:

```
class Friend {
  function setFriend(Friend $friend) {
    $this -> friend = $friend;
  }
}
$doerte = new Friend;
$klaus = new Friend;
$doerte -> setFriend($klaus);
var_dump($doerte->friend);
```

Ausgabe:

```
object(Friend)#2 (0) { }
```

OOP: Vererbung und Schnittstellen

- Deklaration von Vererbung mittels `extends`
  - Nur Einfachvererbung/keine Mehrfachvererbung
- Abmildern der Nachteile fehlender Mehrfachvererbung durch Schnittstellen
  - Schnittstelle legt zu implementierende Methoden (müssen alle `public` sein), aber **nicht** die Variablen einer Klasse fest
  - Eine Klasse kann mehrere Schnittstellen implementieren

OOP: Vererbung und Schnittstellen

- Beispiel:

```php
interface TypeOfSport {
  public function getSport();
}

interface Kicker extends TypeOfSport {
  public function playSoccer();
}

interface TennisPlayer extends TypeOfSport {
  public function playTennis();
}

class Sportsman extends Person implements Kicker, TennisPlayer {
  public function getSport(){ return array('Soccer', 'Tennis'); }
  public function playSoccer(){ … }
  public function playTennis(){ … }
}
```

OOP: Beispiel zur Schnittstelle Iterator

```php
class rowOfTens implements Iterator {
  private $i = 0;
  public function rewind() {
    $this->i = 0;
  }
  public function next() {
    $this->i++;
  }
  public function current() {
    return $this->i * 10;
  }
  public function key() {
    return $this->i;
  }
  public function valid() {
    return $this->i <= 10;
  }
}
$it = new rowOfTens;
foreach($it as $key => $value) {
    echo "$key => $value ";
}
```

Ausgabe:

```
0 => 0
1 => 10
2 => 20
3 => 30
4 => 40
5 => 50
6 => 60
7 => 70
8 => 80
9 => 90
10 => 100
```

OOP: Abstrakte Klassen und Methoden

- Abstrakte Klassen
  - Klassen, von denen es keine Instanz geben darf
- Abstrakte Methoden
  - Methoden, für die keine Implementation in der Klasse angegeben ist
  - müssen in einer nicht abstrakten Unterklasse implementiert werden
  - Eine Klasse mit mind. einer abstrakten Methode muss eine abstrakte Klasse sein

```php
abstract class Mammal {
  public function getNumberOfLegs() {
    return $this -> legs;
  }
  abstract public function nurseWithMilk();
}
```

OOP: Finale Klassen und Methoden

- Ableiten von finalen Klassen nicht möglich

```php
final class bar { … }
class foo extends bar { … } // error (extending final class)!!!
```

- Überschreiben von finalen Methoden nicht möglich

```php
class Person {
  protected $name;
  public final function
           setName($value) {
    $this -> name = $value;
  }
}
```

```php
class Woman extends Person {
  // error: overriding final method:
  public function setName($value){ … }
}
```

- Feature finale Objektvariablen gibt es (leider) **nicht**!

OOP: Objektvariablen überschreiben

- Bitte nur im Ausnahmefall! Besser durch saubere Programmiertechniken lösen!

```php
class Person {
  protected $name = "Person";
  public function setName($value) {
    $this -> name = $value;
  }
// since PHP 7: optional return type
// public function getName():string {
  public function getName() {
    return $this -> name;
  }
}
```

```php
class PersonWithSeveralIdentities
                    extends Person {
  protected $name = array();
  public function setName($value) {
    $this -> name[] = $value;
  }
}
$jenny =
   new PersonWithSeveralIdentities;
$jenny->setName('Jenny');
$jenny->setName('Jennifer');
$jenny->setName('Jakobia');
var_dump($jenny->getName());
```

Ausgabe:

```
array(3) { [0]=> string(5) "Jenny"
           [1]=> string(8) "Jennifer"
           [2]=> string(7) "Jakobia" }
```

Typprüfung

- Ermittlung der Typen von Variablen mittels vieler Funktionen
  - z.B. `is_array($v)`, `is_bool($v)`, `is_float($v)`, `is_int($v)`, `is_null($v)`, `is_numeric($v)`, `is_object($v)`, `is_string($v)`
- Klassennamen eines Objektes ermitteln: `get_class($o)`

```php
abstract class bar {
  public function dump_class() {
    var_dump(get_class($this));
    var_dump(get_class());
  }
}
class foo extends bar {}
$foo = new foo;
$foo -> dump_class();
```

Ausgabe:

```
string(3) "foo"
string(3) "bar"
```

- Überprüfung, ob Objekt `$o` der Klasse `c` oder einer Unterklasse von `c` angehört: `$o instanceof c`

OOP: Typen von Methodenparametern

- Angabe einer Klasse (oder **array**) bei Parametern von Methoden möglich
  - Primitive Datentypen wie **string**, **int** oder **bool** erst seit PHP 7 angebbar
- Beispiel:

```
class Friend {
  protected $friend;
  // declaration of type Friend is optional:
  function setFriend(Friend $friend) {
    $this -> friend = $friend;
  }
}
$doerte = new Friend;
$klaus = new Friend;
$doerte -> setFriend($klaus);
```

OOP: Statische Methoden und Variablen

- Zur Deklarierung von Klassenvariablen und –methoden außerhalb eines Objektkontextes
- Zugriff über `Klassenname::`, **`self`**`::` (für aktuelle Klasse) oder **`parent`**`::` (für Elternklasse)
- Beispiel:

```php
class Friend {
  public static $numberOfFriends = 0;
  public static function incrementFriends() {
    self::$numberOfFriends++;
  }
}
Friend::incrementFriends();
var_dump(Friend::$numberOfFriends);
```

OOP: Konstanten

- Konstanten in Klassen via **const**
- Zugriff ähnlich wie auf statische Variablen (nur ohne $)
- Beispiel:

```
class Tax {
  const VAT = 0.19;
}
class Bill extends Tax {
  public function calculateTax ($sum){
    return $sum * parent::VAT;
  }
}
```

OOP: Aufrufen verdeckter Methoden

- Möglich mittels `::`-Operator
- Beispiel:

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

```php
class Woman extends Person {
  public function setName($value){
    if($this->is_womanName($value)) {
      Person::setName($value);
    }
  }
  public function is_womanName($value){
    return ($value=='Dörte' ||
            $value=='Inga');
  }
}
$doerte = new Woman;
$doerte -> setName('Dörte');
// prints: string(6) "Dörte"
var_dump($doerte->getName());
```

OOP: Magische Methoden

- Alle magischen Methoden beginnen mit einem doppelten Unterstrich
- Automatische Ausführung von magischen Methoden, wenn bestimmte Situationen eintreten
  - Konstruktoren/Destruktoren
  - Klonen
  - Interzeptormethoden
  - Automatisches Abfangen von fehlerhaften Zugriffen auf u.a. Variablen und Methoden einer Klasse

OOP: Magische Methoden – Konstruktoren und Destruktoren

- Aufruf des Konstruktors beim Erzeugen einer neuen Instanz
- Aufruf des Destruktors, falls keine Verweise mehr auf dieses Objekt existieren (und damit dieses Objekt bei der Ausführung des PHP-Skriptes nicht mehr verwendet werden kann)

```
class Database {
  private $name;
  private function
      __construct ($name) {
    $this -> name = $name;
  }
}
$mysql = new MySQL('localhost',
          'user', 'secret');
```

```
class MySQL {
  private $db;
  public function
      __construct ($host, $user, $pw) {
    parent::__construct('MySQL DB');
    $this -> db =
      mysql_connect($host, $user, $pw);
  }
  public function __destruct () {
    mysql_close ($this -> db);
  }
}
```

OOP: Klonen

- Beispiel:

```php
$jenny = new Person;
$jenny->setName('Jenny');
$jennifer = $jenny; // $jenny and $jennifer point to the same object!
$jennifer->setName('Jennifer');
$doerte = clone $jenny; // $doerte contains new object
// $doerte currently has same state as $jenny
$doerte->setName('Doerte');
```

- Nur „seichtes“ Klonen, kein tiefes Klonen
  - Referenzen von Objekten/Arrays innerhalb des zu klonenden Objektes werden übernommen und nicht geklont
  - Tiefes Klonen kann durch magische Methode `__clone` realisiert werden

OOP: Magische Methoden - __clone


```php
class Clone {
  public $name;
  public $clonenumber = 1;
  // will be called after cloning:
  public function __clone() {
    $this -> clonenumber++;
  }
}
$clone1 = new Clone;
$clone1->name = "Jenny";
$clone2 = clone $clone1;
$clone1->name = "Jennifer";
var_dump($clone1);
var_dump($clone2);
```

Ausgabe:
???

OOP: Magische Methoden – `__toString()`

- Zur Steuerung der Ausgabe von **`echo`** und **`print`** (aber nicht von `print_r` und `var_dump`)
- Beispiel:

```php
class Woman {
  protected $name;
  public function __construct($name) {
    $this -> name = $name;
  }
  public function __toString() {
    return get_class($this).' '.$this -> name;
  }
}
$doerte = new Woman('Dörte');
echo $doerte;
```

Ausgabe:

```
Woman Dörte
```

OOP: Magische Methoden – Interzeptormethoden

- Automatisches Abfangen von fehlerhaften Zugriffen (für z.B. Logging von Fehlern)

Interzeptor	Beschreibung
`__call(string $fctname, array $parameter)`	Bei Aufrufversuch einer Methode, auf die man keinen Zugriff hat (z.B. bei falscher Sichtbarkeitsstufe)
`__callStatic(string $fctname, array $parameter)`	…für statische Methoden
`__get(string $varname)`	Aufruf bei einem Lesezugriff auf eine nicht vorhandene oder nicht zugängliche Objektvariable
`__set(string $varname, $value)`	Bei fehlerhaften Schreibversuch auf eine Objektvariable
`__isset(string $varname)`	Bei fehlerhaften Zugriff auf Objektvariablen per `isset`
`__unset(string $varname)`	Bei Anwendung von `unset` auf eine nicht existente oder nicht sichtbare Objektvariable

OOP: Magische Methoden – `__autoload`

- Beim Versuch eine nicht existente Klasse zu instanziieren (via **new**)
- Beispiel
  - Jede Klasse sei in einer eigenen Datei im Unterverzeichnis class abgelegt, Klassenname entspricht Dateiname
  - Folgender Code ladet automatisch fehlende Klassen nach:

```php
function __autoload($class){
  include_once 'class/'.$class.'.php';
}
```

```php
$doerte = new Person('Doerte');
```
führt zunächst
```php
include_once 'class/Person.php';
```
aus (falls keine Person vorher instanziiert wurde)

OOP: Klassen- und Objektfunktionen

Funktion	Beschreibung
`bool class_exists( string $class_name [, bool $autoload=true])`	Prüft, ob eine Klasse verfügbar ist. Anwendung: `function __autoload($class) { include_once 'class/'.$class.'.class.php'; if (!class_exists($class, false)) { die('Class ' . $class . ' not found!'); } }`
`string get_parent_class( [mixed $object])`	Ermittelt die Elternklasse eines Objektes
`array get_class_methods( mixed $class_name)`	Gibt die Methoden einer Klasse zurück
`array get_class_vars( string $class_name)`	Liefert die **vorbelegten** Werte von öffentlichen **Objektvariablen** einer Klasse
`bool method_exists( mixed $obj, string $method)`	Prüft auf Vorhandensein einer Methode in einer Klasse (`$obj` ist Objektinstanz oder Klassenname)
`bool property_exists( mixed $obj, string $prop)`	Existenzüberprüfung von einer Objektvariablen in einer Klasse (`$obj` ist Objektinstanz oder Klassenname)

Ausnahmen

- Methoden der Klasse `Exception`

Methode	Rückgabewert
`__construct($message=null, $code=0)`	`void` (Konstruktor)
`getCode()`	Fehlercode
`getFile()`	Datei mit dem Fehler
`getLine()`	Zeilennummer
`getMessage()`	Fehlermeldung
`getTrace()`	Array der Ablaufverfolgung
`getTraceAsString()`	formatierten String der Ablaufverfolgung

- Benutzerdefinierte Ausnahmen als Unterklassen von `Exception`

Ausnahmebehandlung – Analog zu anderen Programmiersprachen wie Java

- Beispiel:

```php
try{
  try {
    throw new Exception("Hello");
  } catch(Exception $e) {
    echo $e->getMessage()." catch in\n";
    throw $e;
  } finally {
    echo $e->getMessage()." finally \n";
    throw new Exception("Bye");
  }
} catch (Exception $e) {
  echo $e->getMessage()." catch out\n";
}
```

Mehrere catch-Blöcke für unterschiedliche Ausnahmen möglich

Unterschied zu Java:
Funktionsdeklarationen ohne throws-Klausel

Ausgabe:

```
Hello catch in

Hello finally

Bye catch out
```

HTTP-Cookies

- Zur Speicherung von Daten im Browser
  - Daten stehen beim nächsten Besuch der Webseite wieder auf dem Server zur Verfügung
  - Anwendungsszenarien
    - Vermeidung der erneuten Eingabe von Login-Daten bei späteren Besuchen (unsicher!)
    - Tracking von Besuchen der Webseite
    - Spielstände von Online-Spielen
    - Umfragen (Sicherstellen der *einmaligen* Beteiligung)
  - Benutzer muss mit der Speicherung von Cookies einverstanden sein
- Cookies haben einen Zeitpunkt, zu dem sie ungültig (und gelöscht) werden

Setzen eines HTTP-Cookies auf Serverseite

- Bestandteil des HTTP-Headers
  - Vom Server gesendete Cookies müssen vor dem Inhalt der Webseite gesendet werden
    - In PHP Aufrufe von `setcookie(…)` zum Setzen von Daten in Cookies bevor irgendeine Ausgabe erfolgt
    - auch keine vorherige Ausgabe von Leerzeichen-/zeilen
    - ähnliche Restriktion wie bei `header(…)` zum Senden eines HTTP-Headers in Rohform

Beispiel	Beschreibung
`setcookie('TestCookie', 'value');`	Setzen eines Cookies, das nur während dieser Sitzung gilt
`setcookie('TestCookie', 'value', time()+3600);`	Setzen eines Cookies, welches 1 Stunde gültig ist
`setcookie ('TestCookie', '', time() - 3600);`	Löschen eines Cookies durch Angabe eines Verfall-Zeitpunktes in der Vergangenheit

Empfang eines HTTP-Cookies vom Browser

- PHP: Ablage der vom Client gesendeten Cookies automatisch im auto-globalen Array `$_COOKIE`
  - Entsprechende Konfiguration des Webservers vorausgesetzt
  - Cookie kann i.d.R. nur in dem Verzeichnis ausgelesen werden, in dem er auch gesetzt worden ist
    - Verzeichnis/Pfad kann allerdings beim Setzen des Cookies angegeben werden (optionaler Parameter, z.B. `'/'` für ganze Domäne)
  - Zugriff auf den Wert des Cookies durch `$_COOKIE['TestCookie']`

Setzen und Empfangen von Arrays in Cookies

- Beispiel
  - Setzen eines Arrays in einem Cookie:

```
setcookie('c[three]', 'c_three', time() + 3600);
setcookie( 'c[two]',   'c_two',  time() + 3600);
setcookie( 'c[one]',   'c_one',  time() + 3600);
```

  - Speicherung des Arrays erfolgt im Browser in mehreren Cookies
  - Serverseitiger Zugriff auf das Array nach Neuladen der Seite:

```
if (isset($_COOKIE['c'])) {
  foreach ($_COOKIE['c'] as $name => $value) {
    $name = htmlspecialchars($name);
    $value = htmlspecialchars($value);
    echo "$name : $value <br />\n";
  }
}
```

Ausgabe:

```
three : c_three
two : c_two
one : c_one
```

Zusammenfassung

- Ähnliche und genauso mächtige Sprachkonstrukte zur Objektorientiertheit wie bei anderen Programmiersprachen (z.B. Java)
  - interessante Detailabweichungen (durch Skriptsprachen möglich)
- Gute Unterstützung zum einfachen Umgang mit Sessions (vorherige Vorlesungseinheit!) und Cookies zur Speicherung von Daten
  - über einen vorgegebenen unter Umständen längeren Zeitraum (mittels Cookies)