EINFÜHRUNG IN DIE OBJEKTORIENTIERUNG

PHP OBJEKT-ORIENTIERUNG

001

BEGRIFFE DER OBJEKTORIENTIERUNG

OBJEKTORIENTIERUNG

- Klasse - Instanz - Objekt
- Methoden - Interzeptormethoden - statische Methoden
- Konstanten - Eigenschaften - statische Eigenschaften
- Abstrakte Klassen - Interfaces

OBJEKTORIENTIERUNG

- Objektorientierung ist eine Sicht- und Programmierweise für komplexe Systeme, die durch das Zusammenspiel kooperierender Objekte beschrieben werden.

- Objektorientierte Anwendungen werden heute in einem iterativ-inkrementellen Prozess entwickelt.

REALE OBJEKTE

- Objekte sind Gegenstände des realen Lebens, die für die Anwendung mit ihren spezifischen Eigenschaften beschrieben werden.

- Bei einem Onlineshop sind die Objekte zum Beispiel das Produkt oder der Warenkorb.

KLASSENANALYSE: WELCHE KONKRETEN OBJEKT LASSEN SICH IM TEXT IDENTIFIZIEREN?

- „[…] Voraussetzungen um eine **Anzeige aufzugeben** ist dass man **registriert** ist. **User** muss sich **einloggen**. Er kann dann **Anzeigendaten** eingeben und die Anzeige in **Kategorien** einordnen (Fahrrad, Buch, Wohnung, Spülmaschine) Er sollte erfassen können, ob das ein **Angebot** oder eine **Suchanzeige** ist, **Festpreis** oder **Verhandlungsbasis**, **sichtbar** für jeden oder nur die eingeloggten. Er muss einen **Preis** angeben, oder was es kosten darf (**Preisspanne** bei Suchanzeige) Die Anzeige sollte automatisch nach einer voreingestellten **Dauer inaktiv** werden. Der Benutzer soll die Anzeige **zweimal verlängern** können.

- Ein Bild ist gratis, zwei oder mehr Bilder sind kostenpflichtig […] „

OBJEKTEN, EIGENSCHAFTEN, METHODEN

- **Anzeige aufzugeben registriert User einloggen**

Anzeigendaten Kategorien Angebot Suchanzeige Festpreis Verhandlungsbasis sichtbar Preis Preisspanne Dauer inaktiv zweimal verlängern

WELCHE OBJEKTE SIND ENTHALTEN?

- Welche wichtigen Objekte mit Bezug auf die Welt und den Anwendungsfall der Software tauchen im Text auf?
- Der Usecase berichtet von einem **Benutzer**, der sich einloggt, dabei entstehen **Daten**, die evaluiert werden.

PROPERTIES - EIGENSCHAFTEN EINES OBJEKTES

- "Was hat/besitzt das Objekt?"
- Die Eigenschaften des Benutzers sind sein **Benutzername**, seine **E-Mail Adresse** oder das **Passwort**.
- Eigenschaften der Datenobjektes sind **Eingabedaten**, **Erstelldatum**,

METHODS - FUNKTIONEN EINES OBJEKTES

- "Was kann/macht das Objekt?"
- Die Methoden des Benutzers sind **registrieren**, **anmelden**, **Passwort zurücksetzen**, **Profil ändern**, **Profil löschen**.
- Datenmethoden sind **validieren**, **speichern**, **ausgeben**, **löschen**.

OBJEKTE DÜRFEN ABSTRAKT ODER WEICH SEIN

- Ein Objekt muss nicht konkret gegenständlich sein.
- Es muss [aber] nachvollziehbar sein, und der Sicht des zu programmierenden Objektbegriffs sinnvoll entsprechen.

BEGRIFFE DER OO

DER BAUPLAN EINES OBJEKTES WIRD IN EINER KLASSE BESCHRIEBEN.

- **Ein User ist ein Objekt**. Ein User hat einen Namen und eine Emailadresse; er kann seine Daten ändern oder sich anmelden.

KLASSEN

```
// Die Klasse User

class User {
   ...
}
```

Zur Umsetzung der realen Objekte werden Klassen entwickelt, die eine abstrakte Beschreibung der Methoden und Eigenschaften des Objektes zusammenfassen.

OBJEKTE SIND INSTANZEN VON KLASSEN

```
// Programm, das die Klasse User verwendet

// Einbindung der Klasse
require_once 'User.php';

// Erzeugen eines Objektes
$user = new User();

// Testausgabe
var_dump($user); // Ausgabe: object(User)[1]
```

Ein Objekt wird im Programmcode als Instanz einer Klasse definiert.

KLASSEN UND INSTANZEN

```
$user[0] = new User();
$user[1] = new User();
$user[2] = new User();

Alle im Programm hergestellten Userobjekte sind
Instanzen der Klasse User.
```

ATTRIBUTE UND METHODEN

```
class Object {
    // Attributes
    public $a = 0;
    public $b = false;

    public function setA ($value = 0) { ... }
    public function getA () { return $this.a... }
}
```

Schreiben von Attributen und Methoden in einer Klasse.

SICHTBARKEIT VON VARIABLEN UND METHODEN.

PUBLIC, PRIVATE, PROTECTED?

SCOPES ODER DER GÜLTIGKEITSBEREICH VON VARIABLEN

- PHP weist Eigenschaften und Methoden Scopes zu. Die Klasse, die Unterklassen oder auch das Hauptprogramm besitzen jeweils einen eigenen Gültigkeitsbereich.

- Ob eine Klassenvariable in einer Unterklasse oder im Programm sichtbar ist, entscheidet der Sichtbarkeitstyp:

- public, private oder protected

PUBLIC

```
public $a;
public function doSomething () { ... }
```

Die Eigenschaft/Methode kann von überall verwendet werden.
Das Hauptprogramm und alle weiteren Programmteile können auf die Variable zugreifen kann.

PROTECTED

```
protected $a;
protected function doSomething () { ... }
```

Die Eigenschaft ist von außen nicht sichtbar und kann auch von dort nicht verändert werden.
Das Hauptprogramm kann nicht auf die Variable zugreifen. Sie ist nur innerhalb der Klasse gültig. Und in ihren Unterklassen.

PRIVATE

```
private $a;
private doSomething () { ... }

// Zugriff auf eine private Eigenschaft mit einem Setter:
public function setA ($value) {
   if ($value > 0) $this.a=$value;
   else $this.a=0;
}
```

Wie protected, aber ohne Vererbung in Kindklassen.
Die Gültigkeit beschränkt sich auf die Klasse selbst.
Private Attribute werden für Eigenschaften verwendet, sie von außerhalb nicht gesetzt werden dürfen oder zumindest nicht ohne weiteres.

AUFBAU EINER KLASSE

EIGENSCHAFTEN HAT MAN.

```
class User {
// Properties
  public $username;
  public $email;
}
```

Name und Email sind Eigenschaften des Users.
Eigenschaften werden als Variablen zu Beginn der Klasse definiert.

PHP - ES GIBT DATENTYPEN, ABER SIE SIND NICHT FEST

- Die Typisierung von Variablen ist in PHP schwach. Das heisst, die Typen können und dürfen sich während des Programmablaufs ändern.

- Es gibt die einfachen Typen String, Integer, Double, Float, Boolean, aber sie können nicht festgelegt werden.

```
var_dump($user);

object(User)[1]
   public 'username' => null
   public 'email' => null
```

SETZEN VON DEFAULTWERTEN

```
class User {

  public $username = ‘guest‘;
  public $email = ‘john@doe.com‘;

}
```

AUSGABE DER EIGENSCHAFTEN

```
echo $user->username;
echo '<br />';
echo $user->email;
```

DIREKTES BEARBEITEN VON WERTEN

```
$user->email = ‚michael@zenbox.de‘;
```

METHODEN FÜR DAS BEARBEITEN VON WERTEN GETTER UND SETTER

```
class User {

    public function setUsername($arg = 0){
        $this->username += $arg;
        return TRUE;
    }
    public function getUsername(){
        return $this->username;
    }
}
```

Das Ändern von Eigenschaften direkt aus dem Code birgt Risiken und sollte nicht möglich sein. Der Zugriff erfolgt stattdessen über Getter und Setter – Methoden

VORTEIL VON GETTERN UND SETTERN

```
class User {
   // ...
   public function setUsername($arg = ‘‘){
      if (is_string($arg) {
      $this->username = $arg;
      if (sizeof($arg) < 10)
         $this->username = ,‘;
         return FALSE;
      } else {
         $this->username = $arg
         return TRUE;
      }
   }
}
```

Das Ändern oder Lesen von Werten kann kontrolliert werden.

INSTANCEOF

```
if ($user instanceof User) {
  $user->setUsername = $arg;
} else {
  echo 'das ist kein guter Username!';
}
```

Mit instanceof kann geprüft werden, ob eine Instanz aus einer bestimmten Klasse stammt.
Das ist notwendig, wenn eine Methode sicherstellen muss, dass sie im richtigen Kontext angewandt wird.

KONSTANTEN

- Kontanten sind Variablen, deren Wert nur einmal zur Initialisierung gesetzt wird. Während des Programmablaufs ändern Sie sich nicht mehr.

- Der Zugriff auf Konstanten folgt anderen Regeln, wie der auf Variablen. So kann auf eine Klassenkonstante jederzeit zugegriffen werden, auch wenn noch keine Instanz des Objektes aufgerufen wurde.

```
class User {

  private $username;
  private $email;

  const  IS_VIP = true;

}
```

ZUGRIFF AUF KONSTANTEN

```
require_once 'User.php';

if (User::IS_VIP == true) {
  echo ‚Jeder User ist sehr wichtig.';
} else {
  echo ‚Dieser User scheint nicht so wichtig.‘;
}
```

METHODEN

METHODEN KÖNNEN

- Methoden oder Funktionen bilden die Fähigkeiten einer Klasse aus.

- Ein "Produkt hinzufügen" ist eine Methode des Warenkorbes. Ein "neues Produkt erstellen" ist eine Methode der Produktklasse.

```
class User {
  private $username;
  private $gesundheit;
  const  IS_VIP = true;

  public function login() {
     ...
     return TRUE;
  }
}
```

```php
require_once 'User.php';

$user = new User();
echo $user->login();
```

ZUGRIFF AUF EIGENSCHAFTEN UND METHODEN INNERHALB DER KLASSE - SELF UND THIS

SELF - ZUGRIFF AUF KONSTANTEN DER KLASSE IN DER KLASSE

```
if (self::IS_VIP == true) {
    return true;
    } else {
    return false;
    }
}
```

SELF

- Wird immer dann angewendet, wenn man auf Elemente einer Klasse zugreifen will, für die kein Objekt benötigt wird.
- Das sind Konstanten oder auch statische Methoden und Eigenschaften.

THIS

- this bezieht sich auf das aktuelle Objekt als Instanz der Klasse.

THIS

```
public function getUsername() {
  if ( $this->username !== ,') {
      return $this->username;
   }
}
```

```
public function setUsername($arg) {
  if ($arg !== '') {
     $this->username = $arg;
  }
}
```

MAGISCHE UND INTERZEPTOR METHODEN

MAGISCHE METHODEN

- Es gibt einige Methoden, die von PHP in speziellen Fällen aufgerufen werden.
- Jede Klasse besitzt diese automatisch, sie können aber bei der Deklaration auch mit einer neuen Funktionsweise und anderen Parametern überschrieben werden.
- Diese Methoden werden Magic Methods genannt.

MAGISCHE METHODEN

zum Beispiel:

__construct
__destruct
__call und __callStatic
__sleep und __wakeup

__ CONSTRUCT - DER KONSTRUKTOR

- Wenn ein neues Objekt erzeugt, wird ein Konstruktor der Klasse aufgerufen.
- Er wird zum Beispiel dafür verwendet, die Eigenschaften des neuen Objektes festzulegen.
- Der Konstruktor kann nicht überladen werden, er wird nur einmal verwendet.

__ CONSTRUCT - DER KONSTRUKTOR

```
class Session {
  protected $user;

  public function __construct(User $arg) {
    $this->user = $arg;
  }

  public function setUsername($arg) {
    $this->user->username = $arg;
  }
}
```

__DESTRUCT - DER DESTRUKTOR

```
public function __destruct() {
  echo 'Ein '.__CLASS__.' wurde gelöscht!';
}
```

Der Gegenspieler vom Konstruktor ist der Destruktor. Er wird aufgerufen, wenn das Objekt aus dem globalen Speicher gelöscht wird.
Ausgabe: Ein User wurde gelöscht!

__CALL UND __CALLSTATIC

- Methoden müssen noch nicht einmal definiert sein, um aufgerufen zu werden, sie können auch dynamisch zur Laufzeit erzeugt werden.

__CALL UND __CALLSTATIC

```
require_once ‚User.php';
require_once 'Session.php';

$user = new User();
$session = new Session($user);
$session->changeDuration(,‘);
```

```
Fatal error: Call to undefined method Session::changeDuration()
```

Nehmen wir an, wir rufen in unserem Skript die Methode changeDuration() auf.

__CALL

- Die Magische Methode __call in der Klasse Session reagiert auf alle Methodenaufrufe, falls die Methode nicht erreichbar ist.

__CALL

```
public function __call($name, $arguments) {
  if ($name == 'changeDuration') {
    echo ,duration cannot be changed by '. $arguments[0].'!';
  } else {
    throw new Exception('Funktion '
       .$name.' ist nicht definiert in
       ,  Klasse '.__CLASS__);
  }
}
```

SERIALIZE UND UNSERIALIZE

- Objekte können mittels serialize in einen besonderen String umgewandelt werden, z.B. um sie zu speichern.
- Mit der Funktion unserialize werden sie wieder in die Ursprungsform verwandelt.

__SLEEP UND __WAKEUP

- __sleep gibt ein Array zurück, das die Namen und Werte der Eigenschaften des Objekts beinhaltet, die von serialize gespeichert werden sollen.

- Bei Aufruf von unserialize() wird die __wakeup-Methode aufgerufen.

__SLEEP UND __WAKEUP

```
public function __sleep() {
  return array('username');
}

public function __wakeup() {
  $this->duration = 100;
}
```

```php
require_once 'User';

$user = new User();
$stack = serialize($user);
$user = unserialize($stack);

echo $user->getDuration();
```

__CLONE

```
public function __clone() {
  $this->user = clone $this->user;
}
```

Nach dem Klonen wird die Funktion __clone aufgerufen und das Objekt $user durch eine tiefe Kopie erzeugt.

__CLONE

```
require_once ‚User.php';
require_once ‚Session.php';

$user = new User();
$session = new Session($user);
$sessionClone = clone $Session;
$sessionClone->setUsername(‚Berta‘);
echo $user->username;
```

KLASSENEIGENSCHAFTEN UND KLASSENMETHODEN

STATISCHE EIGENSCHAFTEN

- Statische Eigenschaften sind immer schon da.
- Bisher mussten wir ein Objekt erzeugen, um auf dessen Eigenschaften und Methoden zuzugreifen.
- Statischen Eigenschaften und Methoden können bereits aufgerufen werden, wenn die Klasse deklariert wurde.
- Statische Eigenschaften gehören der Klasse, nicht dem Objekt.

SO ZÄHLT MAN DIE ANZAHL DER OBJEKTE, DIE VON EINER KLASSE ERZEUGT WORDEN IST.

```
class User {
  public static $countUser = 0;

   public function __construct() {
      self::$countUser++;
   }
}
```

SO ZÄHLT MAN DIE ANZAHL DER OBJEKTE, DIE VON EINER KLASSE ERZEUGT WORDEN IST.

```
require_once 'User';

echo User::$countUser;

$user[] = new User();
$user[] = new User();
$user[] = new User();

echo User::$countUser;
```

STATISCHE METHODEN

- Statische Methoden sind auch immer schon da.
- Statische Methoden gehören ebenfalls der Klasse, nicht dem Objekt.

STATISCHE METHODEN

```
class User {
  protected static $countUser = 0;
  public function __construct() {
    self::$countUser++;
 }
  public static function getCountUser() {
    return self::$countUser;
  }
}
```

STATISCHE METHODEN

```
require_once 'User';

User::getCountUser();

$user[] = new User();
$user[] = new User();
$user[] = new User();

echo User::$countUser;
```

FORTGESCHRITTEN

ERWEITERUNGEN IN DER OBJEKT ORIENTIERUNG

002

SPEZIALISIEREN UND GENERALISIEREN - VERERBUNG

STEUERUNG VON VERERBUNG UND ÜBERSCHREIBUNG

- Mit Hilfe der Vererbung können Klassen auf anderen Klassen aufbauen. Sie erben alle Eigenschaften und Methoden der Elternklasse.

- Administratoren sind User, Redakteure auch.

- Daher kann eine (abstrakte) Klasse 'User' ihre Eigenschaften an die Klassen 'Admin' oder 'Author' weitergeben.

- Dies ist ein Fall von Spezialisierung.

EIN ADMINISTRATOR IST EIN USER

- Die Klasse Administrator erbt alle Eigenschaften und Methoden der Klasse User.
- User wird Elternklasse genannt,
- Administrator ist die Kindklasse.

EXTENDS

```
class User {
    protected function getUsername () {
        return $this->username
    }
}

class Administrator extends User {
    // ...
}

$Administrator = new Administrator();

echo $Administrator->getUsername();
```

ÜBERSCHREIBEN VON METHODEN UND EIGENSCHAFTEN

```
class Administrator extends User {

  protected $rights = 'limited;

  public function says($message) {
    echo $message;
  }
}
```

```
$author = new Author();

$author->says('Hallo');
```

FORM DER VERERBUNG

PHP kann keine Mehrfachvererbung, d.h., eine Klasse kann immer nur von einer einzigen Klasse abgeleitet werden.

~~class Author extends User, Being {...}~~

VERKETTUNG

```
User => Author => Corrector.

class Corrector extends Author {
   // ...
}
```

ELTERNKLASSE MIT MEHREREN KINDERN

```
class Author extends User {};
class Administrator extends User {};
// ...
```

FINALE KLASSEN UND METHODEN

FINALE KLASSEN UND METHODEN

So wie festgelegt werden kann, dass Klassen und Methoden überschrieben werden müssen, kann man mit dem Schlüsselwort final bestimmen, dass die Klasse oder Methode nicht überschrieben werden darf:

```
class User {
  final public function edit()
  {
    ...
  }
}
```

ABSTRAKTE KLASSEN UND METHODEN

ABSTRAKTE KLASSEN

- Da Administratoren und Autoren User sind, ist das Anlegen einer abstrakten Klasse sinnvoll.
- Aus der abstrakten Klasse kann selbst kein Objekt abgeleitet werden, aber sie trifft allgemeingültige Vorgaben für eine konkrete Klasse.
- Die konkrete Klasse erbt die Methoden und Eigenschaften der abstrakten Klasse.
- Autoren können und haben alles, was User können und haben.

ABSTRAKTE KLASSEN UND METHODEN

- Mit 'abstract' (eigentlich dt. Kurz- oder Zusammenfassung) werden Klassen oder Methoden erstellt, die formal eine Art 'Vorschau' auf die eigentliche Implementierung bilden.

- Methoden, die als 'abstract' definiert werden, müssen in einer abgeleiteten (konkreten) Klasse überschrieben werden, da 'abstract' Methoden keine Funktion implementieren, sondern lediglich die Signatur der Methode darstellen.

ABSTRAKTE KLASSEN UND METHODEN

```
abstract class User{
    public function says();
    public function do();
}
```

ABSTRAKTE KLASSEN UND METHODEN

'Author' übernimmt die Methodensignaturen und überschreibt diese mit seinen eigenen Methoden.

```
class Author extends User {

   public function says() { ... }
   public function do() { ... }
}
```

INTERFACES

INTERFACES

- Ein Interface (Schnittstelle) definiert wie eine abstrakte Klasse, welche Methoden eine konkrete Klasse haben muss.

- Das Interface besitzt aber keine Eigenschaften, sondern enthält nur nur Methodenköpfe. Diese werden von der Klasse überschrieben.

- Daneben enthält ein Interface auch Konstanten.

INTERFACES

- Wenn eine Klasse ein Interface implementiert, muss sie oder die Elternklasse alle Methoden des Interface besitzen. Die Methoden müssen exakt die gleiche Signatur haben, insbesondere die gleichen TypeHints.

- Klassen können mehrere Interfaces implementieren, diese dürfen jedoch keine Methoden mit gleichem Namen haben.

INTERFACES

Ein Interface wird mit dem Schlüsselwort implements implementiert:

```
interface Interface_User {
  public function says();
}
class User implements Interface_User {
  ...
}
```

ABSTRAKTE KLASSEN UND INTERFACES?

- Interfaces und 'abstrakte' Klassen sind nicht kombinierbar.
- Methoden, die im Interface enthalten sind, müssen auch in der Klasse stehen. Eine Klasse aber kann zusätzliche Methoden besitzen.
- Interfaces haben den Vorteil, allgemeine Vorgaben für die Klassenentwicklung zu machen, die - unabhängig von der eigentlichen Umsetzung - in verschiedenen Kontexten eingesetzt werden zu können. Sie fördern die Wiederverwendbarkeit und das Arbeiten in verschiedenen Entwicklerteams.