Westfälische Wilhelms-Universität Münster

Erstellen dynamischer Webseiten mit PHP

WS 2016/2017

wissen.leben
WWU Münster

Dozenten: Patrick Förster, Michael Hasseler

Übersicht der heutigen Inhalte

- Wiederholung Klassen, Attribute, Methoden
- Vererbung
- Instanz-Vergleiche
- Eigene Exceptions

Wiederholung

- Definition einer **Klasse** in PHP

```
class Person {
  …
}
```

- **Attribute**:

```
class Person {
  private|protected|public $NAME = DEFAULT_WERT;
}
```

- **Methoden**:

```
class Person {
  private|protected|public function NAME($PARAM1, …, $PARAM_N) {
    …
  }
}
```

Wiederholung

- Erzeugung einer **Instanz** einer Klasse:

```
$person = new Person();
```

- **new** ruft automatisch die magische Methode **__construct()** (Konstruktor) auf
- Erwartet diese Methode Parameter so werden diese im **new** Aufruf angegeben

```
class Person {
  public $vorname;
  public $nachname;
  public function __construct($vorname, $nachname) {…}
}
```

```
$person = new Person('Max', 'Mustermann');
```

- Der Dereferenzierungs-Operator **->**
  - Zugriff auf ein Instanz-Attribut `$person->vorname`
  - Aufruf einer Instanz-Methode `$person->methode(…)`

Wiederholung

- Die Zuweisung oder das Auswerten von Instanz-Attributen oder der Aufruf einer Instanz-Methode innerhalb einer Instanz erfolgt über **`$this`**

```
$this->nachname = $nachname;
$this->geburtsdatum = $geburtsdatum;
```

- **`$this`** entspricht einer Variablen, die nur innerhalb einer Klasse valide sowie nicht änderbar ist und immer auf die aktuelle Instanz der Klasse verweist

```
public function vollerName() {
  return $this->vorname . ' ' . $this->nachname;
}
```

```
$person1 = new Person('Max', 'Mustermann');
$person2 = new Person('Erika', 'Sonnenschein');
```

```
echo $person1->vollerName();            // ➔ $this ≙ $person1

echo $person2->vollerName();            // ➔ $this ≙ $person2
```

Wiederholung

- Zugriffsmodifikatoren **public** und **private** (**protected** erst mit Vererbung sinnig)

```
class Person {
  public $vorname;
}
```

Zugriff von „**außen**“

```
$person1 = new Person('Max', 'Mustermann');
echo $person1->vorname;
```
➔ **Max**

```
class Person {
  private $vorname;
}
```

```
$person1 = new Person('Max', 'Mustermann');
echo $person1->vorname;
```
➔ **Fatal error**: Cannot access private property

	`class Person { private $vorname;`	`class Person { public $vorname;`
von „**innen**“	`public function vorname() { return $this->vorname; } }`	`public function vorname() { return $this->vorname; } }`

Wiederholung

- **Statische Attribute** gehören zur Klasse nicht zur Instanz

```
class Person {
  public static $INSTANZ_ZAEHLER = 0;
  …
  public function __construct($vorname, $nachname)
    …
    self::$INSTANZ_ZAEHLER++;
  }
}
```

- `::` wird als **Gültigkeitsoperator** bezeichnet
- `self` bezeichnet den *Gültigkeitsbereich* der aktuellen Klasse und ist daher nur innerhalb von Klassen definiert!

- Zugriff auf statische Attribute von außerhalb:

```
echo Person::$INSTANZ_ZAEHLER;
```

Wiederholung

- **Statische Methoden**

```
class Person {
  private static $INSTANZ_ZAEHLER = 0;

  public static function anzahlInstanzen() {
    return self::$INSTANZ_ZAEHLER;
  }
  …
}
```

- Achtung: Statische Methoden gehören zur Klasse nicht zur Instanz
- **`$this`** ist innerhalb von statischen Methoden daher **nicht** definiert!

- Aufruf von innen

```
echo self::anzahlInstanzen();
```

- Aufruf von außen

```
echo Person::anzahlInstanzen();
```

Attribut-Kapselung

- Meist ist eine enge Kapselung von Attribut und Instanz erstrebenswert:
  - Attribute sollten nur durch die zugehörige Instanz geändert werden
- Daher Attribute meist mit Sichtbarkeit **private**
- Der Zugriff von außen erfolgt dann über sognannte **Getter** bzw. **Setter** Methoden:

```
public class Person {
  private $vorname;

  public setVorname($vorname) {$this->vorname = $vorname;}
  public getVorname() {return $this->vorname;}
}
```

- Vorteile:
  - Bessere Kontroller über Lese- und Schreiberechte von außen
  - Schutz vor invaliden Werten
  - Verbergen der internen Repräsentation eines Attributes

Instanz-Iteration (/b01)

- Mit Hilfe der **`foreach`**-Schleife kann über die Attribute einer Instanz iteriert werden:

```
$person1 = new Person('Max', 'Mustermann', '01.02.1976');

foreach ($person1 as $key => $value) {
  echo $key . ": " . $value . NL;
}

➔ // nichts, da alle von Person Attribute private
```

- Allerdings werden bei einer externen Iteration nur die **`public`** Attribute aufgezählt

- Bei einer internen Iteration werden alle Attribute aufgezählt

```
public class Person {
  …
  public function iterierteAugabe() {
    foreach ($this as $key => $value) {
      echo $key . ": " . $value . NL;
    }
  }
}
```

```
$person1->iterierteAugabe();

➔
vorname: Max
nachname: Mustermann
geburtsdatum: 01.02.1976
```

Vererbung

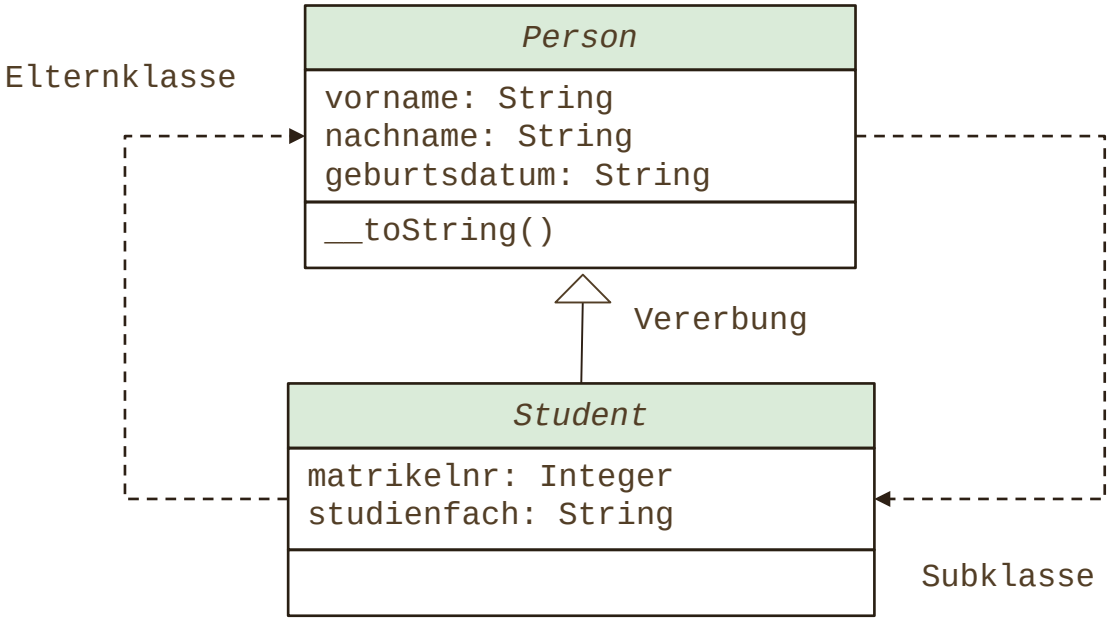
- Als Beziehung zwischen Klassen wurden **Komposition** und **Aggregation** vorgestellt
  - Komposition: Eine Instanz „besteht aus“ anderen Instanzen (abhängige Existenz)
  - Aggregation: Eine Instanz „gehört zu“ einer anderen Instanzen (unabhängige Existenz)

- Daneben gibt es noch die „Ist ein“-Beziehung
  - Ein PKW ist genauso wie ein LKW ein Fahrzeug
  - Ein Dreieck ist genauso wie ein Kreis eine geometrische Figur
  - Ein Angestellter ist genauso wie ein Beamter ein Arbeitnehmer
  - Ein Arbeitnehmer ist genauso wie ein Student ein Mensch
  - Ein Mensch ist genauso wie ein Affe ein Säugetier
  - usw.

- Es wirkt „unnatürlich“ diese Beziehung als Komposition oder Aggregation darzustellen
- Die meisten objektorientierten Programmiersprachen bieten das Konzept der Vererbung an

Klassenbeziehung: Vererbung

- Vererbung repräsentiert eine „ist ein“ Beziehung zwischen Klassen



- Ein Student *ist eine* Person

Vererbung in PHP (/b02)

- Eine Klasse „erbt“ von einer anderen Klasse durch das Schlüsselwort **extends**

```
class Student extends Person {
}
```

- Die erweiternde Klasse erbt dabei alle Attribute sowie Methoden der Elternklasse
- D.h. insbesondere auch die magischen Methoden **__construct** und **__toString**

```
$student1 = new Student('Max', 'Mustermann', '01.02.1976');
echo $student1;

➔ Person: Max Mustermann(Geburtsdatum: 01.02.1976)
```

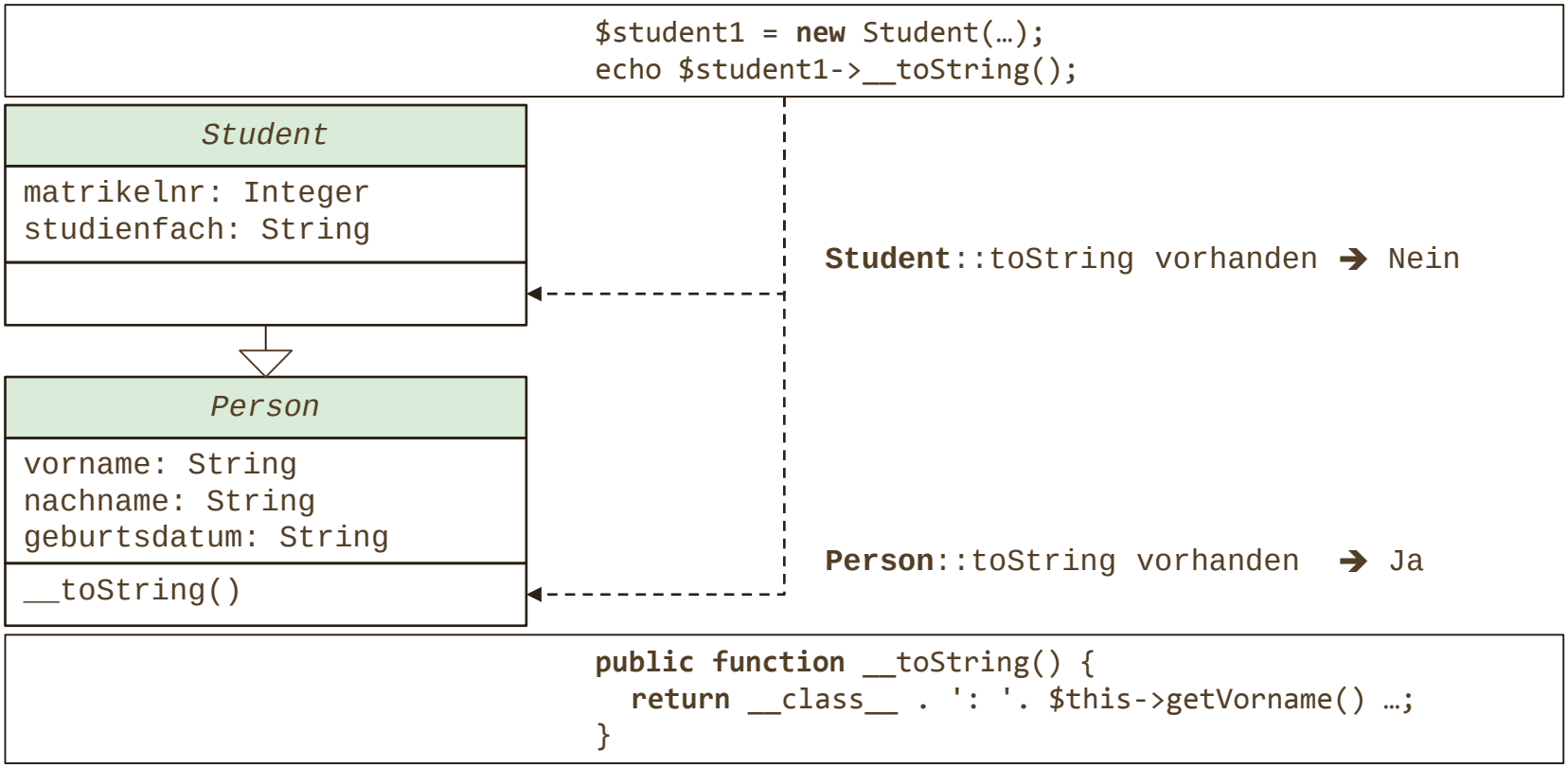
- Und natürlich auch die Getter/Setter

```
$student1->setGeburtsdatum("10.02.1976");
echo $student1->getGeburtsdatum() . NL;

➔ 10.02.1976
```

Vererbung in PHP

- Warum beginnt die Ausgabe von **`$student1->__toString()`** mit **`Person`**?



Vererbung in PHP

- **Vererbungsregel**: ist eine Methode/Eigenschaft in der aufgerufenen Klasse nicht vorhanden wird in der Elternklasse danach gesucht und dann in deren usw. bis zum ersten Auffinden der Methode/Eigenschaft

- Verläuft die Suche erfolglos, so kommt es **Fatal Error**

```
Fatal error: Call to undefined method
```

- Eine erbende Klasse kann Methoden seiner Elternklasse „**überschreiben**“

```
class Student extends Person {
  public function __toString() {
    return __class__;
  }
}
```

```
$student1 = new Student('Max', 'Mustermann', '01.02.1976');
echo $student1->__toString();

➔ Student
```

Gültigkeitsbereich `parent`

- Überschreiben bedeutet im Vererbungskontext nicht, dass die Methode der Elternklasse nicht mehr existiert
- Sie wird allerdings nicht mehr ausgeführt
- Mit dem Gültigkeitsbereich **`parent`** kann explizit auf die Methoden der Elternklasse zugegriffen werden

```
class Student extends Person {

  public function __toString() {
    return parent::__toString() . " ==> " . __class__;
  }

}

➔ Person: Max Mustermann (Geburtsdatum: 01.02.1976) ==> Student
```

Methoden überschreiben

- In PHP basiert Überschreiben einzig und allein auf dem Namen der Methode
- D.h. in einer erbenden Klasse kann eine überschreibende Methode die Parameterliste beliebig neu-definieren
- Angenommen **`Student`** wird um folgende Attribute samt Getter/Setter erweitert:

```
class Student extends Person {
  private $matrikelnr;
  private $studienfach;
  …
}
```

- Analog zu **`Person`** sollten diese im Konstruktor initialisiert werden
- Dazu wird der **`Person`**-Konstruktor überschrieben und um zwei Parameter erweitert:

```
  public function __construct($vorname, $nachname, $geburtsdatum, $matrikelnr, $studienfach) {
    parent::__construct($vorname, $nachname, $geburtsdatum);

    $this->setMatrikelNr($matrikelnr);
    $this->setStudienfach($studienfach);
  }
```

Sichtbarkeitsmodifikator: **protected**

- Warum führt folgender Aufruf zu einem Fehler?

```
class Student extends Person {
  …
  public function __toString() {
    return $this->vorname . " " . $this->nachname . " (Matrikelnr: " . $this->matrikelnr .  ")";
  }
}
➔ Notice: Undefined property: Student::$vorname
```

- Das Attribut **$vorname** wurde in Person als **private** definiert
- D.h. niemand außer der Klasse selbst (bzw. eine konkrete Instanz) darf darauf zugreifen
- Innerhalb von **Student** kann – trotz Vererbung – kein direkter Zugriff erfolgen
  - Zugriff und Manipulation über *Getter/Setter*, falls vorhanden
  - Attribut statt **private** auf **protected** ändern

  ➔ Erbenden Klassen dürfen zugreifen (von „außen“ bleibt der Zugriff verwehrt)
- Gleiches gilt für Methoden, die in der Elternklasse als **private** deklariert wurden
- **Achtung:** Die Sichtbarkeit von Methoden/Attributen ist in der erbenden Klasse **nicht** änderbar!

Der Modifikator: `final`

- Neben den Sichbarkeitsmodifikatoren existiert noch der Modifikator **`final`**

```
class Person {
  …
  public final function getVollerName() {
    return $this->getVorname . ' ' . $this->getNachname;
  }
}
```

- Methoden, die als **`final`** deklariert wurden können in einer erbenden Klasse nicht überschrieben werden:

```
class Student extends {
  public function getVollerName() {…}
}
```

```
Fatal error: Cannot override final method Person::getVollerName()
```

- Man kann auch Klassen als **`final`** deklarieren, so dass von diesen nicht geerbt werden kann

Instanzenvariablen vergleichen (b02)

- Zur Wiederholung
  - **`A == B`**: **`true`**, falls **`A`** und **`B`** unter Konvertierung den gleichen Wert repräsentieren
  - **`A === B`**: **`true`**, falls **`A`** und **`B`** typgleich sind und den gleichen Wert repräsentieren

- Für Instanzvariablen gilt:
  - **`A == B`**: **`true`**, falls **`A`** und **`B`** Instanzen der gleichen Klasse sind und die gleichen Attributwerte (unter **`==`**) haben

```php
class TestObject {
  public $attribut1;

  public function __construct($attribut) {
    $this->attribut1 = $attribut;
  }
}
```

```php
$object1 = new TestObject("1000");
$object2 = new TestObject(1000);
$object3 = new TestObject("Ein String");

vergleiche($object1, $object3);
vergleiche($object2, $object3);
vergleiche($object1, $object2);
```

```
TestObject[1000::string] == TestObject[Ein String::string]: FALSE
TestObject[1000::integer] == TestObject[Ein String::string]: FALSE
TestObject[1000::string] == TestObject[1000::integer]: TRUE
```

Instanzvariablen vergleichen

- Eine Variablen, der per **`new`** eine Instanz zugewiesen wird, repräsentiert nicht die Instanz selbst, sondern ist einen *Verweis* auf die Instanz im Programmspeicher
- Mit dem **`->`** Operator wird dieser Verweis „dereferenziert“ und man erhält Zugriff auf die eigentliche Instanz (den eigentlichen Wert)
- Mit jedem **`new`** wird eine *neue* Instanz erzeugt, also eine neue Referenz zugewiesen
- Für Instanzvariablen gilt:
  - **`A === B`**: **`true`**, falls **`A`** und **`B`** die gleiche Instanz referenzieren

```
$object1 = new TestObject("1000");
$object2 = new TestObject(1000);
$object3 = object1;                    // object3 verweist nun auf die gleiche Stelle im
                                       // Speicher wie object1, also auf TestObject("1000")

vergleiche($object1, $object2);
vergleiche($object2, $object3);
vergleiche($object1, $object3);

TestObject[1000:string] === TestObject[1000:integer]: FALSE
TestObject[1000:integer] === TestObject[1000:string]: FALSE
TestObject[1000:string] === TestObject[1000:string]: TRUE
```

Anwendung Vererbung: Exceptions (/b03)

- **`Exception`**: Möglichkeit Fehler von Rückgabe zu trennen, falls eine sinnvolle Fehlerbehandlung nicht an Ort und Stelle möglich ist

- **`Exception`** ist eine vordefiniert, *nicht* als **`final`** deklarierte Klasse

➔ Man kann von **`Exception`** erben und so das Exception-Handling verfeinern

```
class TypeException extends Exception {

  public function __construct($typ) {
    if (gettype($typ) === 'object') {
      $typ = get_class($typ);
    }

    parent::__construct("Eingabe muss vom Typ <b>$typ</b> sein.");
   }

}
```

Anwendung Vererbung: Exceptions

- TypException wird nun anstelle von Exception bei der Typüberprüfung eingesetzt

```
class Person {
  public function setVorname($name) {
    if (!is_string($name)) throw new Exception("Eingabe muss vom Typ <b>String</b> sein");

    $this->vorname = $name;
  }
  …
}
```

```
class Person {
  public function setVorname($name) {
    if (!is_string($name)) throw new TypException("string");

    $this->vorname = $name;
  }
  …
}
```

Anwendung Vererbung: Exceptions

- Eine **`Exception`** kann anhand ihrer Klasse im **`catch`**-Teil einer **`try`**-Anweisung explizit verarbeitet werden

```
try {
  $person1 = new Person(10, 'Mustermann', '01.02.1976');
  …
}
catch (TypException $e) {
  echo "Typ-Fehler: " . $e->getMessage();
}
catch (Exception $e) {
  echo "Unbekannter Fehler: " . $e->getMessage();
}
```

```
➔ Typ-Fehler: Eingabe muss vom Typ string sein.
```

- Die geworfene **`Exception`** wird von oben nach unten mit den **`catch`**-Zeilen verglichen
- Die erste Typübereinstimmung bekommt den Zuschlag
- Taucht man im obigen Beispiel die **`catch`**-Reihenfolge, erhielte man daher:

```
➔ Unbekannter Fehler: Eingabe muss vom Typ string sein.
```

Zusammenfassung

- Wie ist das Prinzip der Vererbung in PHP umgesetzt
- Was bedeutet „Methodenüberschreibung“ und welche Konsequenzen hat diese
- Was ist beim Vergleichen von Instanzvariablen zu beachten
- Wozu dienen selbstdefinierte Exceptions

Aufgaben

1. Schreiben Sie eine Klasse **`Dozent`**, die von der Klasse **`Person`** erbt.
   a. Überlegen Sie sich sinnvolle Attribute, die **`Dozent`** von **`Person`** unterscheidet
   b. Schützen Sie diese Attribute durch *Getter* bzw. *Setter*

2. Der Klasse **`Vorlesung`** (vgl. Bsp. 3, Vorlesung 9) werden im Konstruktor oder per Setter Instanzen der Klasse **`Person`** zugewiesen.
   a. Ändern Sie diese Klasse so ab, dass nur “Dozenten” akzeptiert werden
   b. Stellen Sie sicher, dass ein **`Dozent`** nicht zweimal hinzugefügt werden kann, indem Sie eine eigene **`Exception`** für diesen Fall definieren und schmeißen.

Anhang: ZIVinteraktiv-Abstimmung

- QR-Code zur anonymen Abstimmung:

- https://www.uni-muenster.de/ZIV/anw/zivinteraktiv/abstimmung.php?code=5804a66a