MEDIEINSTITUTET

Systemutveckling PHP

Föreläsning 08 - PHP OO

Mikael Olsson
mikael.olsson@emmio.se
076-174 90 43

emmio

Dagens ämnen

- Objektorientering

Punkter

- https://github.com/emmio-micke/wies18-php

- Jag hade tänkt att ni skulle använda er av *Classic models*, men vid närmare eftertanke kan ni få använda er av vilken databas ni vill.

- Jag behöver DoD (för de där det är tillämpbart) och godkänna era uppgifter.

Objektorientering

```php
class Student {
    public $name;
    public $age;
    public $teacher;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Teacher {
    public $name;
    public $age;

    function __construct($name = '', $age = 35) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

- Finns det några likheter mellan klasserna?
- Vad gör vi om vi behöver lägga till telefonnr till båda?

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->print();
$teacher->print();
```

- Student och Teacher *ärver* från Person.

- En barnklass som ärver av en föräldraklass har samma klassvariabler och metoder som föräldraklassen.

Övning

- Skapa klassen Vehicle med klassvariablerna color, model, speed och noOfWheels.

    - Skapa metoderna accelerate och decelerate som båda tar en parameter och ökar resp minskar hastigheten med parametern. Kontrollera att parametern inte är orimlig, t ex hastigheten inte blir mindre än 0.

    - Skapa metoden `print()` som skriver ut egenskaperna för fordonet.

- Skapa klasserna Motorcycle och Sportscar som ärver av Vehicle.

    - Skapa objekt av båda klasserna och kör metoden `print()` på dem.

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
    public $teacher;
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->teacher = $teacher;

$student->print();
$teacher->print();
```

- Student har dock ingen klassvariabel för `$teacher` längre.

- Vi kan dock lägga till variabler till vår nya klass, sedan finns den tillgänglig för nya objekt.

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
    public $teacher;

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
        echo 'Teacher: ' . print_r($this->teacher, true) . PHP_EOL;
    }
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->teacher = $teacher;

$student->print();
$teacher->print();
```

- `print()` i Student skriver inte ut läraren.

- Vi kan ladda över (*override*) metoder från föräldrar.

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
    public $teacher;

    function print() {
        parent::print();$this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
        echo 'Teacher: ' . print_r($this->teacher, true) . PHP_EOL;
    }
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->teacher = $teacher;

$student->print();
$teacher->print();
```

- Nu har vi dock redundans igen, vi gör ju redan samma sak i föräldrametoden.
- Vi kan dock anropa föräldrametoden från vår metod.

Scope Resolution Operator (::)

- Scope Resolution Operator, dubbelkolon ger tillgång till statiska, konstanta och överskrivna egenskaper eller metoder i en klass.

- När man refererar till dessa items utanför en klassdefinition använder man namnet på klassen.

```
class MyClass {
    const CONST_VALUE = 'A constant value';
}

echo MyClass::CONST_VALUE;
```

Scope Resolution Operator (::)

- Tre speciella nyckelord, `self`, `parent` och `static`, används för att få tillgång till egenskaper eller metoder inifrån en klass.

```php
class OtherClass extends MyClass
{
    public static $my_static = 'static var';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}

OtherClass::doubleColon();
```

Övning

- `Sportscar` behöver klassvariabeln `noOfDoors`.
    - Anpassa `print()`-metoden för `Sportscar`.
- Skapa klassen `Truck` som ärver från `Vehicle` och har variabeln `loads` (hur många kubikmeter den kan lasta).
    - Hur håller vi reda på hur mycket som är lastat just nu?
    - Skapa metoden `unload()` som tar en parameter och lastar av så mycket från lasten. Kontrollera att man inte försöker lasta av mer än vad som finns.
        - Om parametern är för stor, returnera `false`.
        - Annars, returnera så mycket som har lastats av.
        - Om parametern är tom, lasta av allt och returnera hur mycket som har lastats av.

Traits

- Traits kan användas för att ge tillgång till samma metod i flera klasser.

- Om man använder en trait i en klass har den tillgång till alla variabler och metoder och tvärtom.

```php
trait MyTrait1
{
   function Hello() {
      echo 'Hello';
   }
}

trait MyTrait2
{
   function World() {
      echo 'World';
   }
}

class MyClasse1
{
   use MyTrait1;
   use MyTrait2;

   function __construct() {
      $this->Hello();
      $this->World();
   }
}

$Test = new MyClasse1;
```

Övning

- Skapa två stycken traits.
    - Den ena ska implementera metoden `booking()` som tar ett datum och ett objekt som parameter och skriva ut ett meddelande om att en bokning är skapad för besiktning för fordonet.
    - Den andra ska implementera metoden `paint()` som tar en parameter och byter färg på fordonet.
- Använd trait:sen i `Vehicle`-klassen och anropa dem från dina objekt.

Synlighet

- Synlighet styr när man kan få tillgång till variabler och metoder från utanför klassen.

Synlighet

```php
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private
```

Synlighet

```php
class MyClass
{
    // Declare a public constructor
    public function __construct() { }

    // Declare a public method
    public function MyPublic() { }

    // Declare a protected method
    protected function MyProtected() { }

    // Declare a private method
    private function MyPrivate() { }

    // This is public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Works
$myclass->MyProtected(); // Fatal Error
$myclass->MyPrivate(); // Fatal Error
$myclass->Foo(); // Public, Protected and Private work
```

Synlighet

```
class MyClass2 extends MyClass
{
    // This is public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Fatal Error
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Works
$myclass2->Foo2(); // Public and Protected work, not Private
```

Övning

- Gör variablerna `model` och `color` privata och anpassa koden så allt funkar.

- Gör metoden `print()` protected och gör den tillgänglig.

Hur använder vi OOP?

- Lista produkter, t ex

```
class Product {
    public add($name, $description, $price) {
        // Control parameters and add product to db.
    }

    public getProducts($noOfProducts) {
        // Get the number of products as an array
    }
}

// Page:
$product_obj = new Product();
$products = $product_obj->getProducts(9);

foreach($products) {
    // Print out product.
}
```

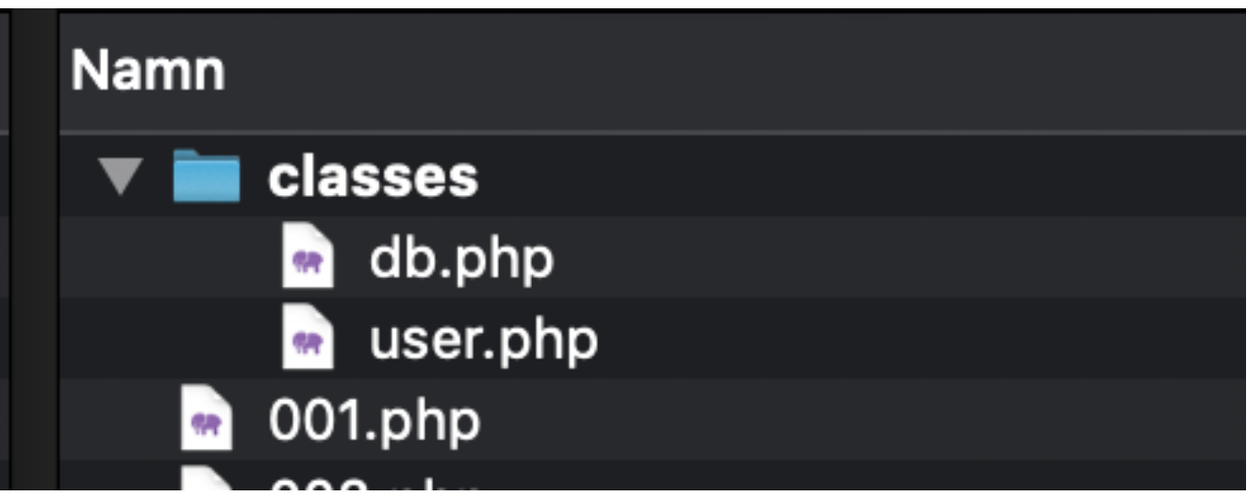
Hur använder vi OOP?

- Lista produkter, t ex

```php
<?php

class User {
    private $firstName;

    public function register($username, $password) {
        // Control parameters and register user.
    }

    public function login($username, $password) {
        // Check if user exists
    }

    public function getName () {
        return $this->firstName;
    }
}

// Page:
if ($userSentForm) {
    $user = new User();

    // Psuedo...
    $user->login(filter_input($username), filter_input($password));

}
```

Hur använder vi OOP?

- Skapa en fil för varje klass och inkludera den när du behöver.




```php
// Page:
include_once "classes/user.php";

if ($userSentForm) {
    $user = new User();
```

Livekodning

- Varukorg
- PDO
- Något annat?

Sätta upp en host

- https://se.000webhost.com/

Sammanfattning

- Objektorientering

Utvärdering

- Prata i grupper om 2-3 personer i två minuter.

- Vad har varit bra idag?

- Vad skulle kunna förbättras?

MEDIEINSTITUTET

Tack för idag!

Vi ses i morgon!

Mikael Olsson
mikael.olsson@emmio.se
076-174 90 43

emmio