

Versnellers Blok C PHP Les 2

Leerdoelen

- Constructors
- Class Inheritance
- Static Methods
- Namespaces

Wat was een class ook alweer?

- Blauwdruk van een object
- Bevat properties en functions
- Maken we een instantie (instance) van
- Herbruikbaar, uitbreidbaar en logisch

```
<?php
```

```
class Car {  
    public $brand;  
    public $type;  
    //etc.  
  
    public function showBrand(){  
        echo $this->brand;  
    }  
}
```

```
$myCar = new Car();  
$myCar->brand = "Toyota";  
$myCar->showBrand();
```

Wat was een class ook alweer?

- Class namen zijn altijd PascalCase
- Properties en functions zijn altijd camelCase.

```
<?php
```

```
class Car {  
    public $brand;  
    public $type;  
    //etc.  
  
    public function showBrand(){  
        echo $this->brand;  
    }  
}
```

```
$myCar = new Car();  
$myCar->brand = "Toyota";  
$myCar->showBrand();
```

Wat was een class ook alweer?

- Public = mag binnen de class, subclass en erbuiten gebruikt worden.
- Protected = mag binnen de class en subclass gebruikt worden.
- Private = mag alleen binnen de class gebruikt worden.

```
<?php

class Car {
    public $brand;
    public $type;
    //etc.

    private function getBrand() {
        return $this->brand;
    }

    public function showBrand(){
        echo $this->getBrand();
    }
}

$myCar = new Car();
$myCar->brand = "Toyota";
$myCar->showBrand();
```

Wat is een constructor

- Wordt uitgevoerd zodra er een instantie gemaakt wordt van de class.
- Wordt gebruikt om bijv. automatisch alvast wat properties in te stellen.
- Functienaam is altijd `__construct()`

```
<?php

class Car {
    public $brand;

    public function __construct($brandName) {
        $this->brand = $brandName;
    }
}

$myCar = new Car("Toyota");
```

Wat is een constructor

- We stellen de properties in door de class met parameters aan te roepen. Dit kan er meer dan een zijn.
- Je kunt de properties van een class en de parameters van een constructor ook voorzien van stricte types.

```
<?php

class Car {
    public string $brand;
    public string $type;
    public int $doors;
    public int $wheels;

    public function __construct(string $brand, string $type, int $doors, int $wheels) {
        $this->brand = $brand;
        $this->type = $type;
        $this->doors = $doors;
        $this->wheels = $wheels;
    }
}

$myCar = new Car("Toyota", "Corolla", 5, 4);

echo $myCar->doors; // 5
```

Opdracht!

1. Maak een class met de naam Person.
2. Voeg een constructor toe die twee eigenschappen initialiseert: firstName en lastName.
3. Voeg een methode toe genaamd sayHello() die een begroeting retourneert, bijvoorbeeld:
"Hallo, ik ben [voornaam] [achternaam]!".
4. Maak een instantie van de class Person aan met jouw eigen voor- en achternaam.
5. Roep de methode zegHallo() aan en toon het resultaat op het scherm met echo.

Class Inheritance

Een inherited class neemt alle eigenschappen van een andere class over, en voegt daaraan nieuwe eigenschappen toe.

We doen dit door "extends Class" toe te voegen aan de definitie.

```
<?php

class Person {
    public string $firstName;
    public string $lastName;

    public function __construct(string $firstName, string $lastName)
    {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
    }

    public function sayHello() {
        return "Hallo, ik ben $this->firstName $this->lastName!";
    }
}

class Teacher extends Person {
    public string $subject;

    public function __construct(string $firstName, string $lastName, string $subject)
    {
        parent::__construct($firstName, $lastName);
        $this->subject = $subject;
    }

    public function showSubject(){
        return "Ik geef les in $this->subject.";
    }
}

$teacher = new Teacher("Steven", "van Rosendaal", "WEB");
echo $teacher->sayHello();
echo "<br>";
echo $teacher->showSubject();
```

Class Inheritance

1. We maken een class aan met properties, een constructor en een function.
2. We maken een tweede class aan die de eerste class extend.
3. We voegen een nieuwe property toe die niet in de eerste class aanwezig was.
4. We maken een nieuwe constructor die de constructor van de originele class pakt, en de nieuwe property bijvoegt.
5. We geven de tweede class zijn eigen function.
6. We maken een instance van de tweede class. We kunnen nu de properties van beide classes toewijzen via de constructor.
7. Nu we een instantie hebben van Teacher, kunnen we van zowel de Person class als de Teacher class functies aanroepen omdat Teacher de Person class extend.

```
<?php

class Person {
    public string $firstName;
    public string $lastName;

    public function __construct(string $firstName, string $lastName)
    {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
    }

    public function sayHello() {
        return "Hallo, ik ben $this->firstName $this->lastName!";
    }
}

class Teacher extends Person {
    public string $subject;

    public function __construct(string $firstName, string $lastName, string $subject)
    {
        parent::__construct($firstName, $lastName);
        $this->subject = $subject;
    }

    public function showSubject(){
        return "Ik geef les in $this->subject.";
    }
}

$teacher = new Teacher("Steven", "van Rosendaal", "WEB");
echo $teacher->sayHello();
echo "<br>";
echo $teacher->showSubject();
```

Nog een opdracht!

1. Maak een class Device met de volgende eigenschappen:
 - name (bijv. "printer", "telefoon")
 - brand (bijv. "HP", "Apple")
2. Een constructor die deze twee eigenschappen instelt.
3. Een methode info() die bijvoorbeeld deze tekst retourneert:
 - "Dit is een printer van [brand]."
4. Maak een subclass Laptop die erft van Apparaat:
5. Voeg een extra eigenschap toe: operatingSystem (bijv. "Windows", "macOS").
6. De constructor moet alle drie de eigenschappen aannemen en via parent::__construct() de eerste twee instellen.
7. Voeg een methode toonDetails() toe die de info van het apparaat laat zien en extra vermeldt wat het besturingssysteem is, zoals:
 - "Dit is een laptop van [brand]. Het besturingssysteem is [operatingSystem]."
8. Maak een object van Laptop en roep toonDetails() aan.

Gratis tip:

Je kunt zelfs functies van de extended class compleet overschrijven door dezelfde functie opnieuw aan te maken.

```
<?php

class Person {
    public string $firstName;
    public string $lastName;

    public function __construct(string $firstName, string $lastName)
    {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
    }

    public function sayHello() {
        return "Hallo, ik ben $this->firstName $this->lastName!";
    }
}

class Teacher extends Person {
    public string $subject;

    public function __construct(string $firstName, string $lastName, string $subject)
    {
        parent::__construct($firstName, $lastName);
        $this->subject = $subject;
    }

    public function sayHello() {
        return "Hallo, ik ben $this->firstName $this->lastName en ik ben een $this->subject docent!";
    }
}
```

Static Functions

Static functions zijn functions die we kunnen aanroepen zonder een instantie te maken van een class.

Deze worden vooral gebruikt als een methode geen properties nodig heeft, of als we een "helper" of "utility" willen maken.

```
<?php

class Calculator {
    public static function add($a, $b) {
        return $a + $b;
    }

    public static function multiply($a, $b) {
        return $a * $b;
    }
}

// Static function oproepen zonder een object te maken:
echo Calculator::add(5, 3);           // 8
echo "<br>";
echo Calculator::multiply(4, 6);     // 24
```

Static Functions

De functies beginnen altijd met "public static function" en werkt verder zoals een normale function. We kunnen alleen geen properties gebruiken, en de waardes moeten altijd met parameters aangeleverd worden.

We kunnen vervolgens zonder "\$var = new Class" de class aanroepen. We roepen vervolgens een static function aan door "::" erachter te zetten, gevolgd door de functie die we willen uitvoeren.

```
<?php

class Calculator {
    public static function add($a, $b) {
        return $a + $b;
    }

    public static function multiply($a, $b) {
        return $a * $b;
    }
}

// Static function oproepen zonder een object te maken:
echo Calculator::add(5, 3);           // 8
echo "<br>";
echo Calculator::multiply(4, 6);     // 24
```

We doen een worp op nog een opdracht

1. Maak een class Dice.
2. Voeg een static method toe genaamd roll().
3. Deze methode moet een willekeurig getal tussen 1 en 6 retourneren (zoals een echte dobbelsteen). Zoek zelf even op internet op welke functie je daarvoor nodig hebt.
4. Roep de methode gooi() meerdere keren aan zonder een object te maken, en toon de resultaten.

Namespaces

Vergelijk het met mappen op je computer.

Stel je hebt twee bestanden met dezelfde naam, bijvoorbeeld config.txt. Dat kan zolang ze in verschillende mappen staan. Zo werkt het ook in PHP met namespaces: je kunt classes, functies of constanten met dezelfde naam hebben, zolang ze in verschillende namespaces zitten.

Namespaces

```
<?php
namespace Math;

class Calculation {
    public static function add($a, $b) {
        return $a + $b;
    }
}
```

```
<?php
namespace Financial;

class Calculation {
    public static function add($a, $b) {
        return ($a + $b) * 1.21; // Inclusief btw
    }
}
```

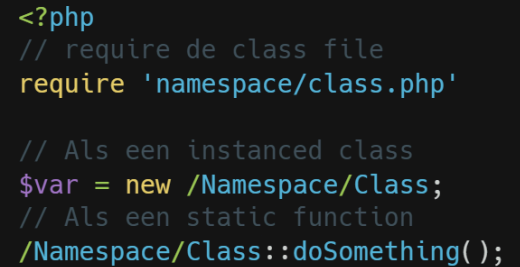
```
<?php
require 'math.php';
require 'financial.php';

echo \Math\Calculation::add(5, 10);    // 15
echo "<br>";
echo \Financial\Calculation::add(5, 10); // 18.15
```

Namespaces

Namespaces hebben de volgende regels:

1. Gebruik namespace ALTIJD bovenaan een php file. Er zijn NOOIT meerdere namespaces in een bestand.
2. Er zijn eigenlijk liever ook niet meerdere classes in een namespaced bestand. Maak voor iedere class met dezelfde namespace een aparte .php file aan.
3. Roep de classes nu aan met "/Namespace/Classnaam"



```
<?php
// require de class file
require 'namespace/class.php'

// Als een instanced class
$var = new /Namespace/Class;
// Als een static function
/Namespace/Class::doSomething();
```

Namespaces

Aangezien we het liefste maar een class per file willen hebben, maar classes elkaar wel moeten kunnen extenden, kunnen we "use-statements" gebruiken om andere classes in te laden. We hoeven helemaal geen include of require te gebruiken in dit geval.

/Technology/Vehicle.php

```
<?php
namespace Technology;

class Vehicle {
    // Class inhoud
}
```

/Technology/Car.php

```
<?php
namespace Technology;

use Technology\Vehicle;

class Car extends Vehicle {
    // Class inhoud
}
```

/index.php

```
<?php

use Technology\Car;

$myCar = new Car;
```

Eindopdracht!

<https://github.com/curio-lesmateriaal/php-les2>