OOP

1 Wat is OOP en waarom gebruiken we het?

□□ Leerdoelen

- Je begrijpt het verschil tussen procedureel programmeren en OOP.
- Je weet wat een class, object en method zijn.
- Je kunt herkennen wanneer OOP handig is in grotere projecten.

□□ Uitleg

Wat is OOP?

OOP staat voor **Objectgeoriënteerd programmeren** (object geörienteerd programmeren). In plaats van losse functies en variabelen, werk je met objecten die gedrag (via classes) en gegevens (properties) combineren.

Waarom OOP gebruiken?

- Je kunt code beter organiseren en hergebruiken.
- Je maakt je programma makkelijker uitbreidbaar.
- Je kunt complexe systemen opdelen in logische blokken (objecten).

Voorbeeld: procedureel vs OOP

Procedureel:

```php
<?php
$naam = "Tessa";
$leeftijd = 19;

function begroet($naam) {
    echo "Hallo, ik ben " . $naam;
}

begroet($naam);
?>
```

Objectgeoriënteerd:

```php
<?php
class Persoon {
    public $naam;
    public $leeftijd;

    public function begroet() {
        echo "Hallo, ik ben " . $this->naam;
    }
}

$tessa = new Persoon();
$tessa->naam = "Tessa";
$tessa->leeftijd = 19;
$tessa->begroet();
?>
```

Je ziet hier dat de funvtie en de variabelen in één blok (class) staan.

We gebruiken bij OOP soms vreemde termen:

Procedureel	OOP
variabele	**property** (van een class)
functie	**method**

□□ Opdracht – Van procedureel naar objectgeoriënteerd

Je gaat een simpel PHP-script omzetten naar OOP.

1. Maak een bestand `mijn-procedureel.php` en schrijf de volgende code:

```
<?php
$autoMerk = "Toyota";
$bouwjaar = 2018;

function toonAuto($merk, $jaar) {
    echo "Dit is een $merk uit $jaar";
}

toonAuto($autoMerk, $bouwjaar);
?>
```

2. Maak nu een tweede bestand: `mijn-oop.php` en herschrijf dit script in OOP-stijl:

```
<?php
class Auto {
    ...
    ...
    ...
    ...
}

$auto = new Auto();
$auto->merk = "Toyota";
$auto->bouwjaar = 2018;
$auto->toon();
?>
```

□□ Reflectie

- Wat vond je makkelijker: de procedurele of de OOP-versie?

- Waarom denk je dat OOP belangrijk is bij grotere projecten?

□□ Inleveren

- Lever beide bestanden in: `mijn-procedureel.php` en `mijn-oop.php`
- Maak een kort reflectiebestand: `reflectie-les1-<jouw-naam>.txt`
- Let op: je code moet uitvoerbaar zijn in de browser (via XAMPP of localhost).

2 Classes en objects

□□ Leerdoelen

- Je weet wat een class en een object zijn.
- Je kunt een eenvoudige class maken in PHP.
- Je kunt een object aanmaken op basis van die class en ermee werken.

□□ Uitleg

Wat is een class?

Een class is een soort blauwdruk of bouwtekening. Je beschrijft hiermee hoe een bepaald type object eruitziet en wat het kan doen.

Wat is een object?

Een object is een concreet exemplaar van een class. Je maakt een object aan met `new`. Een class kun je meerdere keren gebruiken om verschillende objecten te maken.

Class, Object, Properties en Methods worden nog een keer duidelijk gemaakt in:

https://slides.com/jamescandan/oop-php-for-beginners-1-2/fullscreen#/5

https://www.youtube.com/embed/tZJcsFp9ttU

Dus een class is een sjabloon van een object en met het commando `new <class_naam>` maken we een nieuw object van de sjabloon. In een class heten de variabelen **properties** en de fucnties heten **methods**.

Voorbeeld:

```
<?php
class Hond {
    public $naam;
    public $leeftijd;

    public function blaf() {
        echo "Woef! Ik ben " . $this->naam;
    }
}

$mijnHond = new Hond();
$mijnHond->naam = "Rex";
$mijnHond->leeftijd = 3;
$mijnHond->blaf();

$jouwHond = new Hond();
$jouwHond->naam = "Sip";
$jouwHond->leeftijd = 5;
$jouwHond->blaf();
?>
```

Je hebt **één class**, maar je hebt **twee objecten** (honden) van deze class: `$mijnHond` en `$jouwHond`

Opdracht – Maak je eigen class

1. Maak een nieuw bestand: `mijn-klas.php`
2. Maak hierin een class `Student` met de eigenschappen `voornaam`, `achternaam` en `opleiding` .
3. Maak een methode `stelVoor()` die zegt: "Hallo, ik ben [voornaam] [achternaam] en ik doe de opleiding [opleiding]."
4. Maak **drie** objecten (`$student01`, `$student02`, en `$student03`) van de class en test het resultaat in je browser.

□□ Reflectie

- Wat viel je op bij het maken van een object?
- Wat gebeurt er als je een properties (voornaam, achternaam en opleiding) **niet** vult voor je `stelVoor()` gebruikt?

□□ Inleveren

- Bestandsnaam: `mijn-klas.php`
- Reflectievragen in een apart txt- of PDF-bestand.

3 Constructors en eigenschappen

□□ Leerdoelen

- Je weet wat een constructor is in een class.
- Je kunt automatisch waarden toekennen bij het maken van een object.
- Je begrijpt het verschil tussen eigenschappen (properties) en methoden.

□□ Uitleg

Wat is een constructor?

Een constructor is een speciale methode die automatisch wordt uitgevoerd als je een object maakt. In PHP heet deze methode `__construct()`.

Waarom handig?

Met een constructor hoef je niet handmatig eigenschappen toe te wijzen nadat je een object hebt aangemaakt.

Voorbeeld zonder constructor:

```php
<?php
class Auto {
    public $merk;
    public $bouwjaar;
}

$auto = new Auto();
$auto->merk = "Toyota";
$auto->bouwjaar = 2018;
?>
```

Voorbeeld met constructor:

```php
<?php
class Auto {
    public $merk;
    public $bouwjaar;

    public function __construct($merk, $bouwjaar) {
        $this->merk = $merk;
        $this->bouwjaar = $bouwjaar;
    }
}

$auto = new Auto("Toyota", 2018);
?>
```

□□ Opdracht – Constructor gebruiken

1. Maak een nieuw bestand: `les3-student.php`
2. Maak opnieuw een class `Student` met eigenschappen: `voornaam`, `achternaam` en `opleiding`.
3. Voeg een constructor toe waarin deze waarden worden ingesteld.
4. Maak een methode `stelVoor()` die de gegevens netjes toont.
5. Maak minstens twee objecten aan en laat de `stelVoor()`-methode zien in de browser.

□□ Reflectie

- Wat is het voordeel van een constructor ten opzichte van handmatige toekenning?
- Wat gebeurt er als je het aantal parameters niet goed doorgeeft?
- Wat betekent `$this->...` in de constructor?

□□ Inleveren

- Bestandsnaam: `les3-student.php`
- Reflectievragen in een apart bestand: `reflectie-les3-<jouw-naam>.txt`

4 Methoden met parameters en returnwaardes

□□ Leerdoelen

- Je begrijpt hoe je een methode maakt die parameters accepteert.
- Je kunt een methode schrijven die een waarde teruggeeft met `return`.
- Je ziet hoe objectmethoden kunnen samenwerken met parameters uit de code.

□□ Uitleg

Wat is een parameter?

Een parameter is een waarde die je doorgeeft aan een methode, zodat deze methode die waarde kan gebruiken in de berekening of actie.

Wat is een returnwaarde?

Met `return` geef je een resultaat terug uit een methode, zodat je dat resultaat ergens anders kunt gebruiken.

Voorbeeld:

```
<?php
class Rekening {
    public $saldo;

    public function __construct() {
        $this-saldo = 0;
    }

    public function stort($bedrag) {
        // vul hier de code aan
    }

    public function toonSaldo() {
        // vul hier de code aan
    }
}

$mijnRekening = new Rekening();
$mijnRekening->stort(50);
echo "Je saldo is: ". $mijnRekening->toonSaldo();
?>
```

□□ Opdracht – Werk met parameters en return

1. Maak een nieuw bestand: `les4-rekening.php`
2. Maak een class `Rekening` met een startsaldo van 0.
3. Voeg een methode `stort($bedrag)` toe die het bedrag bij het saldo optelt.
   □□Tip: vegeet ni `$this->` te gebruiken!
4. Voeg een methode `opname($bedrag)` toe die het bedrag van het saldo afhaalt.
5. Voeg een methode `toonSaldo()` toe die het saldo teruggeeft met `return`.
6. Test de class met een **paar stortingen** en **paar** opnames. Toon aan dat:
   - dat stortingen het saldo verhogen

- dat opnames het saldo verlagen
- dat een te hoge opname het saldo niet verandert

Probeer het zonder code, maar kom je er niet uit dan kun je deze code als template gebruiken .

Template

```
echo "<pre>";

$rekening = new Rekening();

// Test 1: stort 50
// .......
echo "Test 1: verwacht 50 → resultaat: "......

// Test 2: opname van 20
// ......
echo "Test 2: verwacht 30 → resultaat: " ........

// Test 3: opname van 100 (te veel)
.......
echo "Test 3: verwacht nog steeds 30 → resultaat: " ........

echo "</pre>";
```

□□ Reflectie

- Wat gebeurt er als je een opname doet die groter is dan je saldo?
- Waarom heeft de constructor in de class Rekening in het voorbeeld geen parameter (om het saldo op 0 te zetten)?
- Wat zijn de voordelen van werken met `return`?

□□ Inleveren

- Bestandsnaam: `les4-rekening.php`
- Reflectievragen in een apart bestand: `reflectie-les4-<jouw-naam>.txt`

5 Checkpoint: Wat heb je geleerd?

Samenvatting

1. **Classes & Objects**
   - Je kent het verschil tussen een class (blauwdruk) en een object (instantie).
   - Je weet hoe je met `new` een object maakt.
2. **Properties & Methods**
   - Je gebruikt `public` properties om data in een object op te slaan.
   - Je roept methods aan met parameters (`method($param)`) en begrijpt wat een returnwaarde is.
3. **Constructors**
   - Je kunt een `__construct()` schrijven om bij creatie van een object meteen eigenschappen in te stellen.
   - Je weet dat `$this->...` verwijst naar de huidige instantie.
4. **Encapsulation Basics**
   - Je begrijpt waarom je data soms `private` maakt en via getters/setters benadert.

> **Tip:** Ga kort na of je in je eigen voorbeelden bovenstaande concepten herkent en toepast voordat je verdergaat naar encapsulation en private properties in Les 5.

Opdracht

Maak van elk van de vier genoemde onderwerpen een mulitple choice vraag, met 3 foute en één goed antwoord. Als de vragen goed zijn dan zullen ze terug komen in de kennis-check.

Voorbeeld/template

Vraag: Wat is een object?

A: Dat is een variabele in een class.

B: Dat is een instantie van een class.

C: Dat is een functie in een class

D: Dat is een data-structuur in PHP.

Juiste antwoord is B

Vraag:

A:

B:

C:

D:

Juiste antwoord is:

Vraag:

A:

B:

C:

D:

Juiste antwoord is:

Vraag:

A:

B:

C:

D:

Juiste antwoord is:

Vraag:

A:

B:

C:

D:

Juiste antwoord is:

Inleveren

De vier mulitple choice vragen in txt bestand.

6 Encapsulation en private properties

□□ Leerdoelen

- Je weet wat encapsulation is.
- Je begrijpt het verschil tussen `public` en `private`.
- Je kunt `getters` en `setters` gebruiken om toegang tot eigenschappen te regelen.

□□ Uitleg

Encapsulation betekent dat je data (properties van een class) beschermt tegen direct aanpassen van buiten de class. In plaats daarvan gebruik je methods om de data op een gecontroleerde manier te bekijken of aan te passen.

Waarom gebruiken?

Zo voorkom je fouten doordat andere delen van je code zomaar eigenschappen aanpassen zonder controle.

Voorbeeld:

```
<?php
class Bankrekening {
    private $saldo = 0;

    public function stort($bedrag) {
        if ($bedrag > 0) {
            $this->saldo += $bedrag;
        }
    }

    public function getSaldo() {
        return $this->saldo;
    }
```

```
}

$rekening = new Bankrekening();
$rekening->stort(100);
echo "Saldo: ".$rekening->getSaldo();
?>
```

□□ Opdracht – Bescherm je data

1. Maak een nieuw bestand: `les5-bankrekening.php`
2. Maak een class `Bankrekening` met een `private` eigenschap `$saldo`.
3. Maak een `stort()`-methode die alleen positief bedrag accepteert.
4. Maak een `getSaldo()`-methode die het saldo teruggeeft.
5. Test de class met verschillende stortingen en probeer ook een negatieve storting (die moet worden geweigerd).
6. Maak nu een `opnemen($bedrag)` waarmee je geld kan opnemen. Zorg ervoor dat je alleen geld kan opnemen als je voldoende saldo hebt. Heb je dat niet, dan vind er geen opname plaats.
7. Test de method opname($bedrag) uit!

□□ Reflectie

- Wat is het verschil tussen `public` en `private`?
- Waarom zou je eigenschappen `private` maken?
- Wat gebeurt er als je probeert `$saldo` direct aan te passen: `$rekening->saldo = 1000;`
  Wat zie je? En hoe verklaar je dat?

□□ Inleveren

- Bestandsnaam: `les5-bankrekening.php`
- Reflectievragen in pdf.

7 Bibliotheeksysteem met Twee Classes

□□ Leerdoelen

- Je maakt en gebruikt meerdere classes in één project.
- Je begrijpt hoe objecten van verschillende classes met elkaar communiceren.
- Je oefent met private properties, getters/setters en method-parameters.
- Je bouwt een klein functioneel systeem met relaties tussen objecten.

□□ Uitleg

In dit mini-project bouw je een eenvoudig bibliotheeksysteem met slechts twee classes:

1. **Book**: een boek met een titel, auteur en beschikbaarheidsstatus.
2. **Library**: beheert een collectie boeken en wie ze geleend heeft.

□□ Opdracht – Bouw je eigen bibliotheek

Book Class

Book.php bevat de **class Book** waarmee je het object book kan maken.

Een boek heeft een **title,** een **author** en een **available (beschikbaarheid)**. De class bevat **getters** en **setters** om deze properties te lezen en aan te passen.

book.php

```
<?php
class Book {
    private string $title;
    private string $author;
```

```
    private bool $available = true;

    public function __construct(string $title, string $author) {
        $this->title  = $title;
        $this->author = $author;
    }

    public function getTitle(): string {
        return $this->title;
    }

    public function getAuthor(): string {
        return $this->author;
    }

    public function isAvailable(): bool {
        return $this->available;
    }

    public function setAvailable(bool $avail): void {
        $this->available = $avail;
    }

}
?>
```

Library Class

De **class Library** maakt gebruik van de class Book. Deze class kan boeken **toevoegen** aan de library, kan boeken **uitlenen** en boeken weer **terugnemen**. Als laatst is er een method waarmee je alle boeken uit de library kan tonen.

library.php

```
<?php
require_once 'book.php';
```

```
class Library {

    private array $books = [];
    private array $loans = [];

    public function addBook(Book $book): void {
        // Voeg het boek toe onder de sleutel van de titel
        $this->books[$book->getTitle()] = $book;
    }

    public function lendBook(string $title, string $member): string {
        // Bestaat het boek?
        if (!isset($this->books[$title])) {
            return "Boek “{$title}” bestaat niet.\n";
        }
        $book = $this->books[$title];

        // Is het al uitgeleend?
        if (!$book->isAvailable()) {
            return "Boek “{$title}” is al uitgeleend.\n";
        }

        // Leen uit
        $book->setAvailable(false);
        $this->loans[$title] = $member;
        return "{$member} leent “{$title}”.\n";
    }

    public function returnBook(string $title): string {
        // Bestaat het boek?
        if (!isset($this->books[$title])) {
            return "Boek “{$title}” bestaat niet.\n";
        }

        // Was het uitgeleend?
        if (!isset($this->loans[$title])) {
            return "Boek “{$title}” was niet uitgeleend.\n";
        }
```

```
        // Verwerk retour
        $member = $this->loans[$title];
        unset($this->loans[$title]);
        $this->books[$title]->setAvailable(true);
        return "{$member} brengt “{$title}” terug.\n";
    }

    public function listBooks(): string {
        $output = '';
        foreach ($this->books as $book) {
            if ($book->isAvailable()) {
                $status = 'beschikbaar';
            } else {
                $status = 'uitgeleend aan ' . $this->loans[$book->getTitle()];
            }
            $output .= $book->getTitle() . ' – ' . $status . "\n";
        }
        return $output;
    }
}
```

Main code

In de main code wordt de class library gebruikt om boeken in de library te zetten en uit te lenen of terug te nemen.

Met deze code wordt de library- en book class getest.

Kijk goed wat er gebeurt en probeer het te begrijpen.

main.php

```
<?php
require_once 'book.php';
require_once 'library.php';

echo '<pre>';
```

```
$lib = new Library();
$lib->addBook(new Book('1984', 'George Orwell'));
$lib->addBook(new Book('De Avonden', 'Gerard Reve'));

echo "Leen boek 1884 uit aan Alice<br>resultaat: ";
echo $lib->lendBook('1984', 'Alice');
echo "<br>";

echo "Leen boek 1884 uit aan Bob<br>resultaat: ";
echo $lib->lendBook('1984', 'Bob');   // Alice heeft het nog
echo "<br>";

echo "Laat alle Boeken zien<br>resultaat: ";
echo $lib->listBooks();
echo "<br>";

echo "Boek 1984 wordt tgeruggebracht<br>resultaat: ";
echo $lib->returnBook('1984');
echo "<br>";

echo "Leen boek 1884 uit aan Bob<br>resultaat: ";
echo $lib->lendBook('1984', 'Bob');   // nu kan Bob lenen
echo "<br>";

echo "Laat alle Boeken zien<br>resultaat: ";
echo $lib->listBooks();
echo "<br>";

echo '</pre>'
?>
```

Overzicht

Opdracht: draai dit script in je browser en verifieer de uitvoer.

Hier zie je een eenvoudige UML-class-diagram die laat zien hoe de class Library en de class Book met elkaar communiceren

image.png
Image not found or type unknown

- **Library**
  - Bevat een collectie (`books`) van `Book`-objecten.
  - Methoden zoals `addBook()`, `lendBook()`, `returnBook()` en `listBooks()` gebruiken de `Book`-objecten.
- **Book**
  - Heeft private eigenschappen: `title`, `author`, `available` (en eventueel `loansCount`).
  - Biedt publieke getters (`getTitle()`, `isAvailable()`) en een setter (`setAvailable()`) om zijn status te wijzigen.

De pijlen geven aan dat de `Library` klas methoden aanroept op de `Book` objecten, bijvoorbeeld bij het uitlenen (zetten `available` op `false`) en het terugbrengen (zetten `available` op `true`). Zo zie je visueel hoe de objecten samenwerken binnen het systeem.

□□ Opdracht

Van elk boek wi je bijhouden hoevaak ht is uitgeleend.

1. Voeg een property toe aan de class Book en nome die `$aantalKeerUitgeleend`.
2. Verhoog de property als een boek wordt uitgeleend met 1.
3. Voeg een method `getUileenData()` toe in de class Book en laat deze van elk boek, de titel, auteur en het aantal keer dat dit boek is uitgeleend zien.
4. Test dit via de code in mainn.php, 1984 is al twee maal uitgeleend, maak nog een derde boek en leen dat ook één maal uit.

Aan het iend hebben we dus drie boeken: één boek is nooit uitgleend, één boek is 1x uitgeleend en één boek is twee maal uitgleend.

□□ *Reflectie*

1. Wat doet de `foreach loop` in de method `listBooks()` ?
2. Waarom is **`$author`** private in **Book** en kan je de $author van een Book wijzigen?

3. In de key van **$books** uit de class `Library` staat de *titel van het boek*, wat staat er in de *value*?
4. Beschijf wat er in de *key* en de *value* staat van de $loans uit de class Library.

□□ *Inleveren*

1. De aangepast class Book (book.php).
2. De aangepaste main.php
3. Antwoord op de reflectievragen (pdf)

8 Complete mini-project

(ga niet verder voordat je de vorige opdracht goed hebt afgerond)

□□ Leerdoelen

- Je past classes, objects en constructors toe.
- Je gebruikt public en private properties met getters/setters.
- Je schrijft methods met parameters en returnwaardes.
- Je beheert data in een klein OOP-systeem.

□□ Uitleg

Je bouwt een klein “winkelmandje”-systeem in PHP. Je maakt twee classes:

- `Product` met `private $naam`, `private $prijs` en een constructor.
  Voeg getters toe: `getNaam()` en `getPrijs()`.
- `Winkelmandje` met `private $items = []`.
  Voeg methods toe om `voegToe(Product $p, int $aantal)`, `verwijder(Product $p)` en `getTotaal(): float` te implementeren.

□□ Opdracht – Winkelmandje bouwen

1. Maak bestand `les6-product.php` en definieer class `Product`:

```php
<?php
class Product {
    private $naam;
    private $prijs;

    public function __construct(string $naam, float $prijs) {
        // ...
    }

    public function getNaam(): string {
        // ...
    }

    public function getPrijs(): float {
        // ...
    }
}
?>
```

2. Maak bestand `les6-winkelmandje.php` en definieer class `Winkelmandje`:

```php
<?php
class Winkelmandje {
    private $items = []; // array van ['product' => Product, 'aantal' => int]

    public function voegToe(Product $p, int $aantal): void {
        // voeg product toe of verhoog aantal
        // Kijk of het product al in het mandje zit

        // loop door het winkelmandje heen
        foreach ($this->items as &$item) {
            if ($item['product']->getNaam() === $p->getNaam()) { // kijk of we het prodcut al hebben
                // Verhoog het bestaande aantal
                $item['aantal'] += $aantal;
                return; // klaar
            }
        }
```

```
        // Anders (hebben we geen return gehad dus: nieuw item toevoegen
        $this->items[] = [
            'product' => $p,
            'aantal'  => $aantal
        ];
    }

    public function verwijder(Product $p): void {
        // verwijder product volledig uit items
        ......
    }

    public function getTotaal(): float {
        // bereken en return totaalprijs
        ......
    }
}
?>
```

3. Maak een testscript `les6-test.php`:

```
<?php
require 'les6-product.php';
require 'les6-winkelmandje.php';

$p1 = new Product("Boek OOP", 25.50);
$p2 = new Product("USB-stick", 8.99);

$mandje = new Winkelmandje();
$mandje->voegToe($p1, 2);
$mandje->voegToe($p2, 1);
$mandje->verwijder($p2);

echo "Totaal: €" . $mandje->getTotaal();
?>
```

□□ Reflectie

- Hoe zorg je ervoor dat een `Product` niet negatief geprijsd kan zijn?
- Wat gebeurt er als je `voegToe` aanroept met `$aantal = 0`?
- Waarom gebruik je hier `private` properties en geen `public`?
- Hoe kun je de code uitbreiden met een `updateAantal(Product $p, int $nieuwAantal)`?
- In de function `voegToe(Product $p, int $aantal)` moeten we controleren of het prodcut al bestaat. Als dat zo is dan verhogen we het aantal, aders moeten we het product aanmaken. We zouden verwachten dat we met een `if-then-else` controleren of eht product al bestaat. Waarom staat er in de code geen `if-then-else`?

Inleveren

- Bestanden: `les6-product.php`, `les6-winkelmandje.php`, `les6-test.php`
- Reflectie in apart bestand: `reflectie-les6-<jouw-naam>.txt`

9 Test je Kennis

Wat betekent OOP en hoe verschilt het van procedureel programmeren?

OOP staat voor *Objectgeoriënteerd programmeren*. In plaats van functies en variabelen apart te gebruiken, bundel je bij OOP data en gedrag in objecten. Zo kun je code beter organiseren, hergebruiken en opsplitsen in logische blokken .

Wat is een klasse in OOP?

Een klasse is een blauwdruk of sjabloon waarin je beschrijft welke gegevens (*properties*) en functies (*methods*) een object moet hebben .

Wat is een object?

Een object is een concreet exemplaar van een klasse, gemaakt met new. Je kunt meerdere objecten maken van dezelfde klasse, elk met eigen waarden .

Hoe noem je in OOP een variabele en een functie binnen een klasse?

In OOP noem je variabelen **properties**, en functies **methods**

Wat is encapsulation?

Encapsulation betekent dat je de data (properties) van een object beschermt. Je maakt gegevens vaak private en gebruikt methods om ze gecontroleerd te lezen of aanpassen .

Wat is het verschil tussen public en private properties/methods?

- **public**: toegankelijk en aanpasbaar van buiten de class.
- **private**: alleen toegankelijk binnen de class zelf. Dit beschermt de interne gegevens.

Waarom is OOP handig bij grote projecten?

Omdat je code makkelijker kunt organiseren in logische blokken (objecten), hergebruiken, uitbreiden en onderhouden. Daardoor is je programma stabieler en schaalbaarder .

Waarvoor genbruik je '$this' -> in PHP?

`$this->` verwijst naar een property of een object uit **dit** object. Met dit object wordt bedoeld het object waar `$this->` in staat.

7 Inheritance (Overerving)

□□ Leerdoelen

- Je weet wat inheritance (overerving) betekent in OOP.
- Je kunt een class maken die eigenschappen en methodes **erft** van een andere class.
- Je ziet waarom inheritance handig is: minder herhaling, meer overzicht.

□□ Uitleg

Wat is inheritance?

Inheritance betekent dat je een nieuwe class maakt die eigenschappen en methodes **overneemt** van een andere class. Dit heet overerving. De 'ouderclass' noem je ook wel de *superclass*, en de 'kindclass' de *subclass*.

Voorbeeld

```
<?php
class Dier {
    public $naam;

    public function adem() {
        echo $this->naam . " ademt.\n";
    }
}

class Hond extends Dier {
    public function blaf() {
        echo $this->naam . " zegt: Woef!\n";
    }
}

$rex = new Hond();
$rex->naam = "Rex";
$rex->adem(); // komt uit Dier
$rex->blaf(); // komt uit Hond
?>
```

□□ De clas`Hond` **erft** de methode `adem()` van `Dier`, maar voegt ook zijn eigen gedrag toe: `blaf()`.

□□ Opdracht – Maak je eigen dier met overerving

1. **Bestand: dier.php**
   Maak een class `Dier` met:

- een property `$naam`
- een methode `beweeg()` die `"{$this->naam} beweegt."` toont

2. **Bestand: vogel.php**
   Maak een class `Vogel` die `extends Dier`:
   - een methode `vlieg()` die `"{$this->naam} vliegt!"` toont

3. **Bestand: test.php**
   - Maak een object `$mus` van de class `Vogel`
   - Stel zijn naam in op `"Mus"`
   - Roep zowel `beweeg()` als `vlieg()` aan

□□ Reflectie

- Wat gebeurt er als je `vlieg()` oproept op een `Dier`-object?
- Kun je uitleggen waarom `Vogel` de methode `beweeg()` kan gebruiken zonder die zelf te schrijven?

□□ Inleveren

- `dier.php`, `vogel.php`, `test.php`
- Reflectie in `reflectie-les7-<jouw-naam>.txt`

Revision #19
Created 18 June 2025 14:33:53 by Max
Updated 29 June 2025 19:09:35 by Max