Object oriented programming in PHP

**Inhoud**

[1. The Project 3](#_Toc3239743)

[2. A Class and an Object 4](#_Toc3239744)

[3. Class Methods 7](#_Toc3239745)

[4. Methods that Do work 8](#_Toc3239746)

[5. Multiple Objects 8](#_Toc3239747)

[6. Objects Interact 9](#_Toc3239748)

[7. My Editor is Confused 9](#_Toc3239749)

[8. Using Objects 10](#_Toc3239750)

[9. Private Access 12](#_Toc3239751)

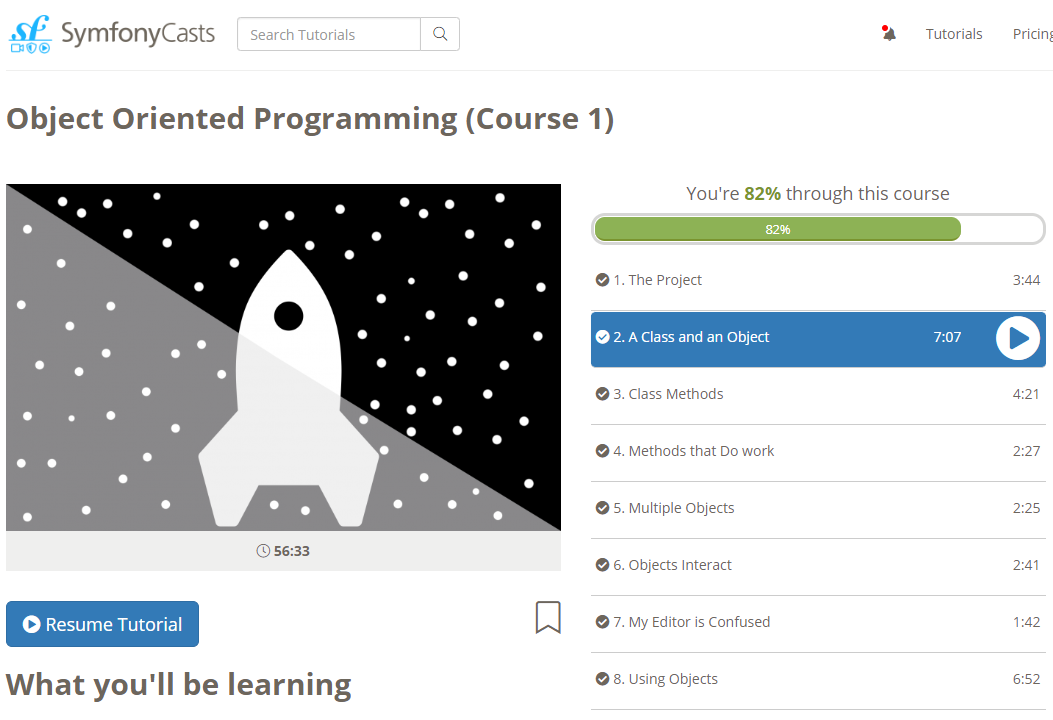
[10. Type Hinting?! 14](#_Toc3239752)

[11. The Constructor 15](#_Toc3239753)

In deze cursus leren we de basisbegrippen van Object Oriented Programming, toegepast in PHP.

We gebruiken een online video tutorial:

<https://symfonycasts.com/screencast/oo>



Elk hoofdstuk in deze cursus is een samenvatting van een online hoofdstuk.

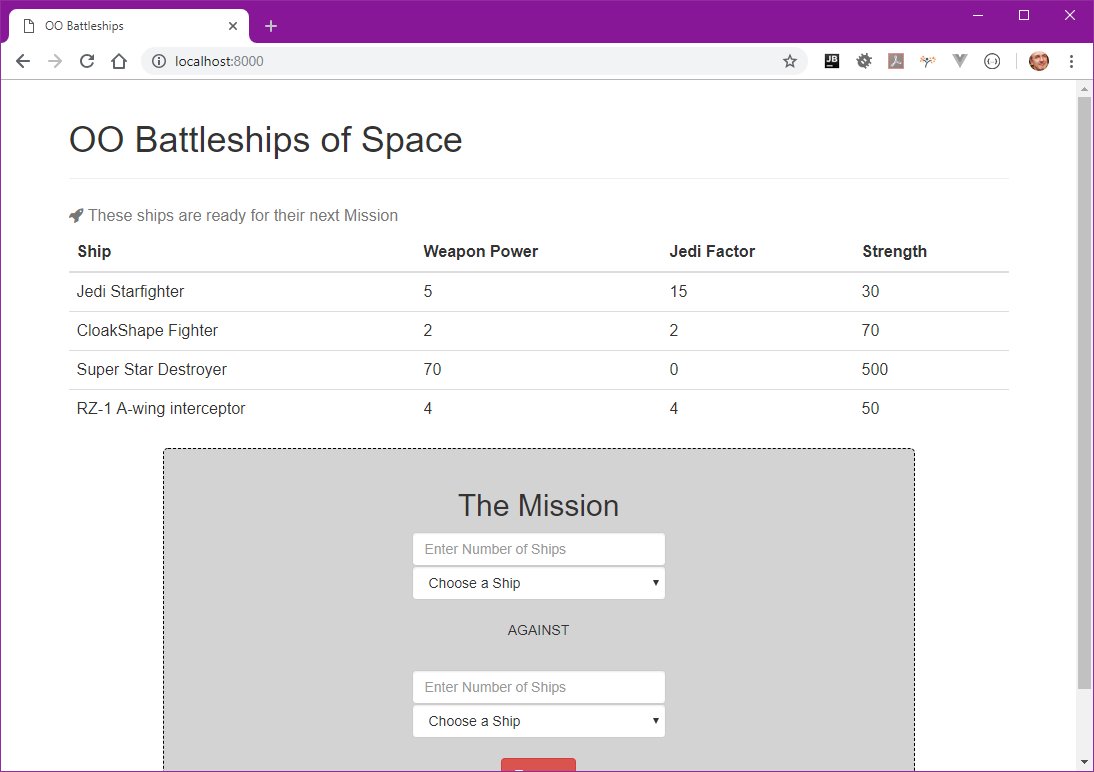
# The Project

* Download de code van de tutorial van Moodle: **code-oo.zip**.
* Pak de code ergens uit (bv. op je Bureaublad)
* Neem de inhoud van de map **start**, en plaats die in de map **htdocs/oo**
* Maak een nieuw project in PHPStorm, **oo** (in de map **www**); PHPStorm stelt voor van de bestaande inhoud van de map te gebruiken; dat is goed.
* Open het **Terminal** venster in PHPStorm en typ:   
    
  **php -S** **localhost:8000**

Op deze manier start je een webserver op poort 8000.

* Open Google Chrome en typ in de adresbalk:

**localhost:8000**



Speel enkele spelletjes, bv. 4 Jedi Starfighters tegen 1 Super Star Destroyer.

We werpen een blik op de code:

* **index.php**
* **functions.php**
* **battle.php**

# A Class and an Object

We maken **play.php**.

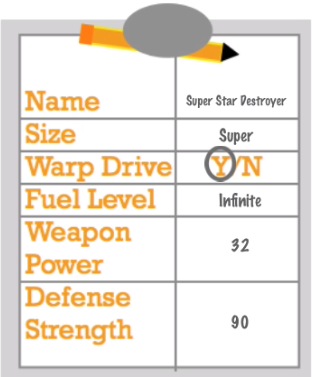
We maken de **klasse** **Ship**.

We maken een **Ship object**.

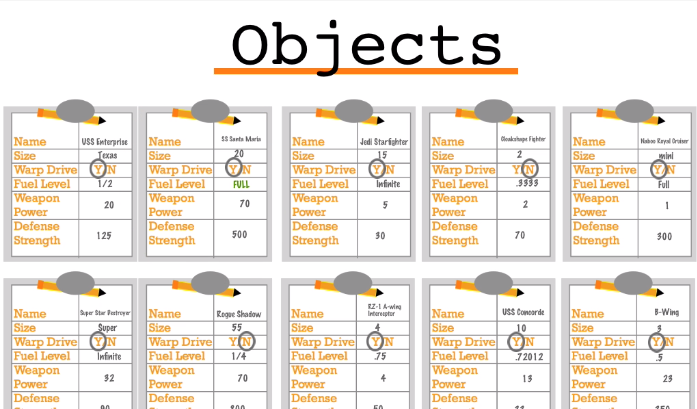
Een **klasse** is als een lege template met mogelijke **eigenschappen (properties)**:



Een **object** is als een **ingevulde template**:



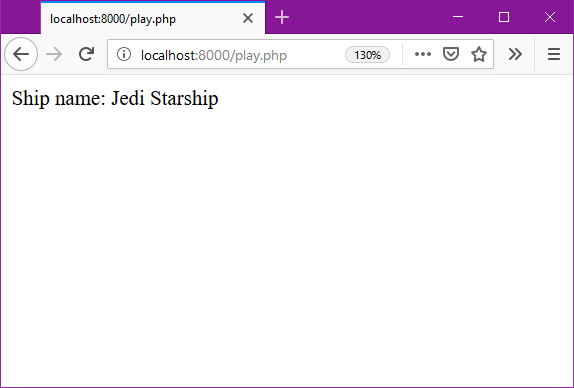
**Een klasse** laat toe om daarvan **vele objecten** te creëren.



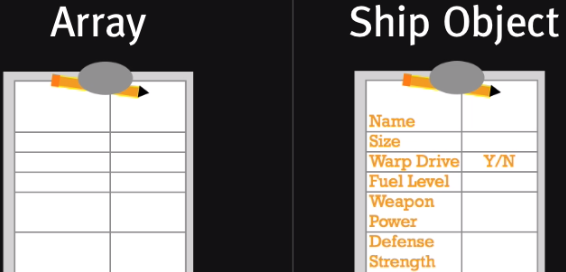
Alle objecten hebben **dezelfde eigenschappen** (bepaald door de eigenschappen van de klasse), maar **niet** noodzakelijk **dezelfde waarden (values)** voor die eigenschappen.

Gebruik het keyword **public** om een nieuwe eigenschap te definiëren.

**class** Ship  
**{  
 public $name**;  
**}  
  
$myShip** = **new** Ship**()**;  
**$myShip**->**name** = **"Jedi Starship"**;  
  
**echo "Ship name: "** . **$myShip**->**name**;



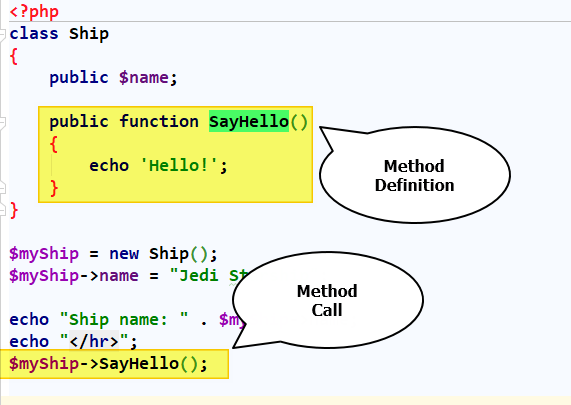
Wat is het belangrijke verschil tussen een array en een klasse?



Een klasse is als **documentatie** voor de programmeur; ze zegt namelijk welke eigenschappen (en methoden) er mogelijk zijn.

# Class Methods

Een **methode** is een functie, gedefinieerd in een klasse.



Met het keyword **$this** kan je – binnen de klasse – de eigenschappen en methoden van de klasse aanspreken/benaderen.

We breiden de klasse **Ship** uit met:

* eigenschappen: **name, weaponPower, jediFactor, strength**
* methoden: **sayHello, getName**

Eigenschappen kunnen een **default value** hebben.

**public $weaponPower** = 0;

We testen alles uit.

# Methods that Do work

We maken de methode **getNameAndSpecs()** die de eigenschappen **name**, **weaponPower**, **jediFactor** en **strength** bondig samenvat:

**Jedi Starship: w:10, j:0, s:0**

We gebruiken de (standaard PHP) **sprintf** functie om alles aan elkaar te plakken.

We voegen een **parameter** toe aan de methode: **$useShortFormat**. Daarmee kunnen we een nog bondigere samenvatting van de eigenschappen vragen:

**Jedi Starship: 10/0/0**

Voer alle wijzigingen door in je eigen code.

# Multiple Objects

We maken de functie **printShipSummary()** voor alle output die we totnogtoe genereerden.

We roepen die functie op.

We creëren nog een **Ship** object, **$otherShip**, met als naam “**Imperial Shuttle**”.

We printen de eigenschappen van beide **Ship** objecten.

# Objects Interact

We maken de methode **doesGivenShipHaveMoreStrength**() en gebruiken die om beide **Ship** objecten te vergelijken.

Deze methode neemt als parameter een **Ship** object **$givenShip**.

We testen deze methode uit. We veranderen de **strength** van beide objecten en bekijken het resultaat.

# My Editor is Confused

De parameter **$someShip** van de functie **printShipSummary** wordt door PHPStorm niet herkend als een **Ship** object. Dat is ook nergens aangegeven.

De methode **sayHello**() wordt evenmin herkend.

Strict genomen hoef je dit niet te veranderen, de code werkt zo ook.

Maar het is handiger als je dat wel doet, als volgt: om de editor (PHPStorm) duidelijk te maken dat **$someShip** een **Ship** object is, schrijven we wat PHP documentatie bovenaan de functie:

*/\*\*  
 \** ***@param*** *Ship $someShip;  
 \*/***function** printShipSummary **($someShip)  
{  
 $someShip**->SayHello**()**;  
**}**

De editor weet nu dat **$someShip** een **Ship** object is, en stelt ook de eigenschappen en methoden daarvan voor als je dit object gebruikt (zodra je **->** hebt getypt).

Tegelijk verdwijnen vervelende waarschuwingen (lijntjes) uit je code.

# Using Objects

Nu we een klasse **Ship** hebben, gaan we die gebruiken in onze app.

We plaatsen de **Ship** class eerst in haar eigen scriptfile **Ship.php** in de map **lib**.

In **play.php** plaatsen we de link naar het klassescript:

**require\_once *\_\_DIR\_\_*** . **"/lib/Ship.php"**;

We testen of het script nog werkt.

We doen hetzelfde nu in **functions.php**. In dit script vervangen we de grote array **$ships** door een nieuwe, waarin **Ship** **objecten** toegevoegd worden (in de plaats van de arrays per ruimtetuig).

Wanneer je **index.php** nu oproept (localhost:8000), geeft die een foutmelding: **Cannot use object of type Ship as array …**

In **index.php** moet je er voor zorgen dat de **array** **syntax** vervangen wordt door de **object syntax**.

Dus niet:

**echo $ship[“name”]; //array syntax**

Maar wel:

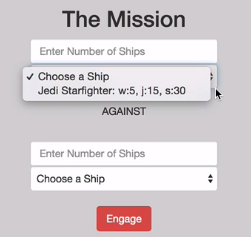
**echo $ship->name; //object syntax**

Dat moeten we overal doen voor de elementen die in de **$ships** array zitten. Dat zijn nu objecten geworden.

Zoek naar de array syntax **‘$ship[‘** in **index.php**: overal waar dit voorkomt moet je de object syntax gebruiken zoals hierboven beschreven.

Als alle array syntax vervangen is door object syntax, moet je **index.php** script weer werken zoals voorheen (met maar één Ship).

* Gebruik in **index.php** de methode **getName**() waar je de naam nodig hebt
* Gebruik de methode **getNameAndSpecs**() daarna in de plaats; maak de parameter **$useShortFormat** optioneel



# Private Access

Er zijn nog heel wat fouten mogelijk in onze applicatie. Een voorbeeld: de eigenschap **strength** van een **Ship** kan ingesteld worden op absurde waarden zoals ‘**Jar Jar Binks**’.

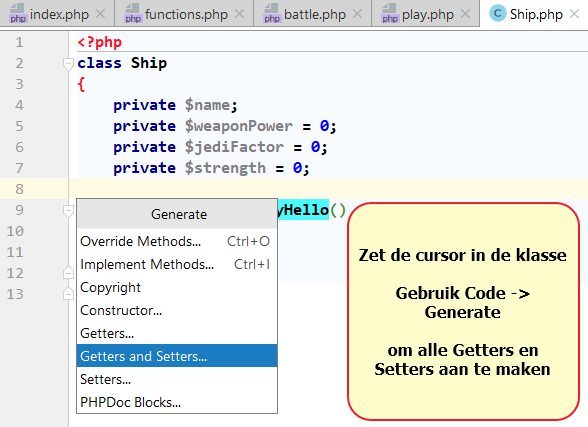
**strength** is een **public** property van een **Ship**. Dat betekent dat we de waarde voor **strength** van buiten de klasse kunnen instellen. Zonder controle.

Dat lossen we zo op:

* maak van **strength** een **private** **property**; daardoor kan ze niet meer van buiten de klasse ingesteld worden
* maak een **public** **methode** **setStrength**()
* in deze methode wordt eerst gecontroleerd of de opgegeven waarde (parameter **$strength**) geldig is (met **is\_numeric**()). Is dat het geval, dan stellen we **strength** in op de opgegeven waarde; zoniet geven we een aangepaste foutmelding.

We maken nu **alle** properties **private**.

In PHPStorm kan je makkelijk alle Getters en Setters genereren met **Code -> Generate**. Zet de cursor wel eerst in de klasse waarvoor je dit wil doen. In ons geval dus de **Ship** class.



In **functions.php** moeten we alle instellingen van de eigenschappen nu vervangen door de aanroep van de **Set methoden**.

Dus niet:

**$ship->weaponPower = 5;**

Maar wel:

**$ship->setWeaponPower(5);**

Idem voor alle andere eigenschappen.

In **index.php** doen we hetzelfde, maar dan met de **Get methoden**.

Dus niet:

**echo $ship->weaponPower;**

Maar wel:

**echo $ship->getWeaponPower();**

Terug naar **functions.php**. Hierin vervolledig je de array **$ships**, zodat er weer 4 ruimtetuigen (of **Ship objecten**) gecreëerd worden.

Het script **battle.php** is nog niet aangepast, dus als je een gevecht start, krijg je een foutmelding. Die lossen we op in het volgende hoofdstuk.

# Type Hinting?!

Het script **battle.php** geeft de nodige foutmeldingen.

De reden daarvoor is dat de functie **battle**() als parameters 1 en 3 arrays verwacht en geen **Ship** objecten.

**function battle(array $ship1, $shipQuantity1, array $ship2, $shipQuantity2)**

Maar de array **$ships** die als argument doorgegeven wordt in het script **functions.php** bevat nu **Ship** **objecten**, en die worden doorgegeven aan de **battle()** functie. Vandaar de eerste fout. Die is vrij onduidelijk.

We kunnen de parameters van de **battle**() functie **onbepaald** laten (door **array** te wissen), of wijzigen in **Ship** objecten.

**function battle(Ship $ship1, $shipQuantity1, Ship $ship2, $shipQuantity2)**

Op die manier:

* krijg je **betere foutmeldingen**
* weet je **editor** van welk type de parameters zijn en kan je beroep doen op **autocomplete** en **waarschuwingen** als het type niet juist is
* is de functie **beter gedocumenteerd** zodat je zelf - of een andere developer - weet welke soort argumenten er verwacht worden

Het **benoemen van het type van parameters** noemt men **Type Hinting**. Type hinting is **optioneel** maar **sterk aan te bevelen**.

We passen **functions.php** aan zodat we overal object syntax gebruiken. In de functie **didJediDestroyShipUsingTheForce**() passen we ook type hinting toe.

Ook in **battle.php** moeten we overal object syntax gebruiken.

De **battle**() functie returnt een array, **$outcome**. In de vorige versie bevatte die nog arrays, maar nu bevat die objecten. Dus ook hier moeten we de ‘oude’ array syntax omzetten in ‘nieuwe’ object syntax. Bijvoorbeeld:

**echo $outcome[‘winning\_ship’]->getName();**

# The Constructor

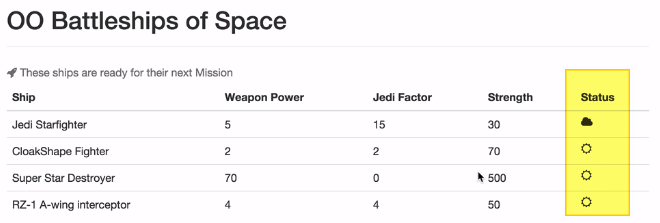
De methode **\_\_construct** wordt de **constructor** genoemd, en wordt meteen uitgevoerd bij het creëren van nieuwe objecten van de klasse waarin hij voorkomt.

**class** Ship  
**{  
 private $name**;  
 **private $weaponPower** = 0;  
 **private $jediFactor** = 0;  
 **private $strength** = 0;  
 **private $underRepair**;  
  
 **//de constructor wordt opgeroepen zodra je een nieuw Ship  
 //maakt met het new keyword: $myShip = new Ship();** **public function** \_\_construct**()  
 {  
 //we geven elk schip 30% kans om in herstelling te zijn  
 $this**->**underRepair** = *rand***(**1**,**100**)** < 30;  
 **}**

…

Voor de nieuwe eigenschap **underRepair** maken we een nieuwe booleaanse methode **isFunctional**(). Die geeft de status van het Ship weer.

Met deze methode geven we in het overzicht een **zonnetje** (functioneel) of een **donkere wolk** (niet functioneel) weer.



We gebruiken dezelfde methode om de inhoud van de keuzelijsten aan te passen, zodat kapotte ruimtetuigen er niet in voorkomen.

De constructor kan ook **parameters** hebben.

We voorzien een parameter **$name** die het mogelijk maakt om op één lijn een nieuw object te creëren en meteen de naam door te geven.

**public function** \_\_construct**($name)  
{  
 $this**->**name** = **$name**;  
 **//we geven elk schip 30% kans om in herstelling te zijn  
 $this**->**underRepair** = *rand***(**1**,**100**)** < 30;  
**}**

…

**$myShip** = **new** Ship();

De parameter **$name** is in bovenstaande code **verplicht** maar kan ook **optioneel** gemaakt worden.