# Lesdoelen

* Student snapt waarom objecten worden gebruikt
* Student kan zelf een class schrijven en gebruiken binnen een script
* Student kan snap de verschillende elementen en kan deze aanwijzen en benoemen
* Student snap het verschillen tussen prive en publiek methoden binnen een class

Inhoudsopgave

Lesdoelen 1

‘OOP ‘ in php Inleiding 1

OOP en php 1

Wat bedoelen we met Object Oriented Programming? 1

Voordelen van OOP ten opzichte van spaghetti code. 1

Procedureel en functies (spaghetti) 1

OOP: eigenschappen, methodes 3

Een class of blauwdruk van object 4

Binnen php een object initialiseren doormiddel van een class. 5

# OOP in php

## Inleiding OOP met php

We gaan met deze 3 lessen een aanzet geven tot eenvoudige gebruik van OOP in php. De basis komt aanbod en zeker niet alle aspecten die PHP ondersteunt van OOP. De basis kan in latere stadia nog wel eens van pas komen. Voordelen OOP in plaats van procedureel in php is dat scripts die je schrijft overzichtelijker zijn dan eenvoudigweg gebruik maken functies.

Oop en php is een rare combinatie. Dit heeft te maken met dat php een scriptaal is geen programmeertaal. PHP kan alleen maar wat met Objecten doen op het moment dat er wat wordt verwacht van de server. Objecten zijn niet ‘in leven’ of continue actief zoals bij heuze programma’s.

## Wat bedoelen we met Object Oriented Programming?

Alle voorbeelden te over de beste omschrijving voor OOP is functionaliteit (niet functies) en de afbakening daarvan. Alles wat binnen een programma zou kunnen vallen onder een bepaalde type functionaliteit is te beschrijven als een object. Je kan objecten vergelijken met mensen in het algemeen. Eigenlijk zijn wel allemaal “mensen” met dezelfde inherente blauwdruk (class) . We hebben allemaal globaal gezien twee handen, twee voeten etc. etc. (oké dit is wat strikt).

We functioneren allemaal ongeveer hetzelfde. Als je een mens wat vraagt, geeft ie antwoord (als hij niet doof is natuurlijke). Antwoord geven doen we meestal met onze mond en optioneel met gebaren. (methodes) De verschillen tussen alle mensen zou je kunnen zien eigenschappen (properties).

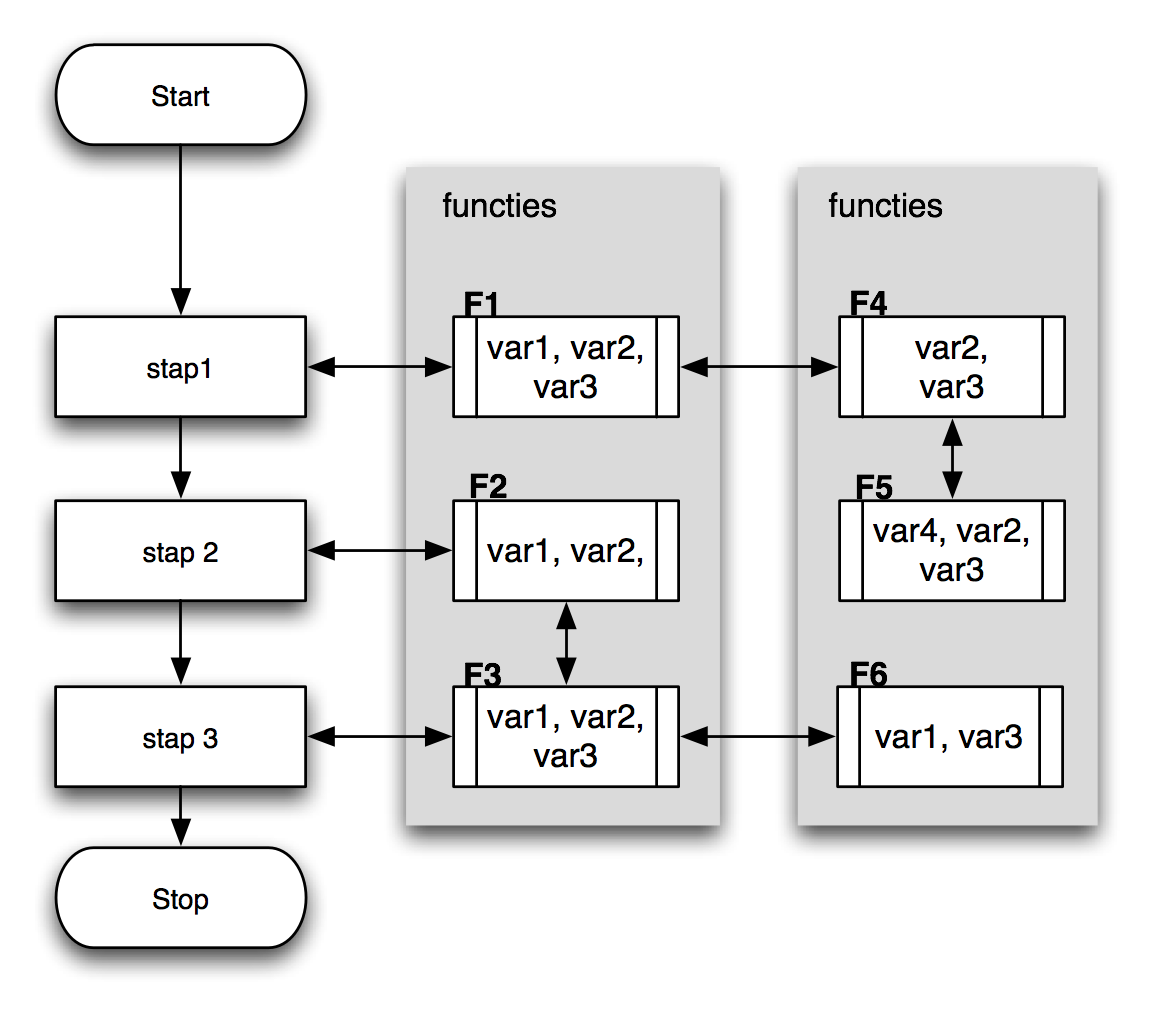
## Voordelen van OOP ten opzichte van spaghetti code.

### Procedureel en functies (spaghetti)

Bij procedureel programmeren ga je er van uit dat er een begin en een einde is aan een script. Als een script wordt gestart dan wordt, stap voor stap en eventueel met functies of sub-routines, een script uitgevoerd. Functies of sub routines wil je vaker gebruiken, anders hebben ze geen nut. Vandaar dat binnen functies gebruik maakt van argumenten. Afhankelijk van de waarde van de argumenten wordt een functie uitgevoerd. Wat is nu het probleem zou je zeggen?

Probleem ontstaat voornamelijk in de logica en het bijhouden van de waardes van variabelen. Elke keer als je een functie opvraagt zal je hoe dan ook waardes voor de argumenten moeten hebben. Dit levert zogenaamde spaghetti code op. Op een gegeven moment weet je niet hoe en waar welke variabele wordt gezet en wat een functie vervolgens mee doet.

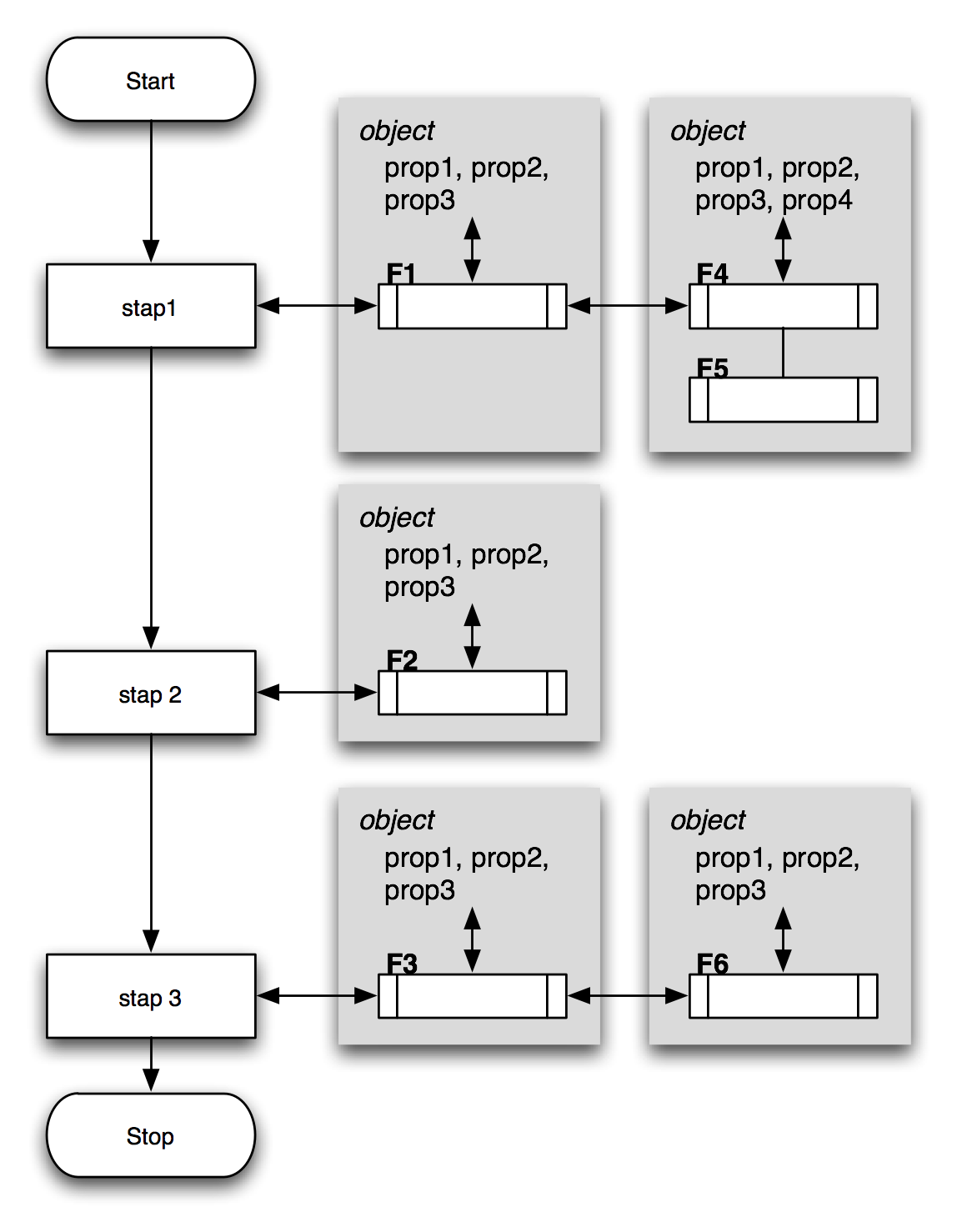
Eenvoudig gebruik van meerdere functies en functies die funtcies gebruiken



Daarnaast is het probleem dat het proces lineair is (stap voor stap). Functies hebben nu eenmaal afhankelijkheden (argumenten) en een volgorde verandering kan een andere uitkomst betekenen. Dat wil je niet want dan wordt deze multi-interpretabel ten koste van de logica.

Zoals je hiernaast ziet worden functies hiernaast in willekeurige volgorde en zo nodig doorelkaar gebruikt. Dit heeft tot gevolg dat functies moeilijk te wijzigen zijn (wanneer wordt hij gebruikt) het niet eenvoudig te volgen wanneer een functie gebruikt wordt.

### OOP: eigenschappen, methodes

 Het grote verschil wat gemaakt wordt binnen OOP in php is dat zolang een object “leeft” daarin varibelen (properties) kunnen worden opgeslagen en naar wens worden gelezen en geschreven, van binnen en buiten het object. Een eigenschap kan dus een bepaalde waarde krijgen , gezet door functies (methodes) binnen het object, en daardoor andere uitkomsten bieden buiten het object. Zeg maar status bepalend.

Open je een nieuw object dan kun je dit weer vanaf heb begin beginnen. De twee objecten hebben eigenlijk niets met elkaar te maken maar vertonen wel hetzelfde gedrag. Als je dus wat vergeet te veranderen heeft dat niet direct gevolg! Daarnaast bestaat er de mogelijkheden om properties en methodes privé of publiek te maken. Privé methodes kunnen alleen door het object gelezen worden (zet een password check) maar niet daarbuiten. Publiek is wel van buiten toegankelijk. De scope van variabelen wordt dus meer afgebakend, maar daardoor handzamer.

## Een class of blauwdruk van object

Als je in php een Object wilt aanmaken zul in het script een class (blauwdruk) moeten aanmaken. Stel nu dat je heb bovenstaande verhaal van mens als gebruiker zou moeten programmeren. Dan kan dat er als volgt in php uitzien.

Voorbeeld user class

class user {

//PROPERTIES: All properties of a user a stated here

private $id;

public $firstname;

public $lastname;

private $username;

private $secondname;

private $password;

private $logged;

//CONSTRUCT: here defaults are set or done

public function \_\_construct($username="anonymous", $password=""){

$this->username ($username);

$this->password ($password);

$this->logged (false);

$this->do\_user\_info();

}

//METHODS: general user functions

private function username ($username){

$this->username = $username;

}

private function password ($password){

$this->password = $password;

}

private function firstname($firstname){

$this->firstname = $firstname;

}

private function lastname ($lastname){

$this-> lastname = $ lastname;

}

public function get\_lastname($lastname){

return $this->lastname;

}

public function get\_firstname ($firstname){

return $this->firstname;

}

public get\_fullname (){

return $this->get\_firstname()." ".$this->get\_lastname();

}

private function logged ($status){

$this->logged = $status;

}

public function get\_logged (){

return $this->logged;

}

//METHODS: Doers

public function do\_login (){

// here we do some login features

// If we can we are logged in or not

// this function sets logged to true or

// leaves this at false (does nothing)

if ($this->get\_password() == “somepassword”

&& $this->get\_username() == “someusername”){

$this->logged(true);

}else{

$this->firstname("John");

$this->lastname("Doe");

}

$this->do\_user\_info ();

}

private function do\_user\_info (){

// depending on username set we will set name properties

if ($this->get\_username() != "anonymous"){

// get user data and do something different

}else{

$this->firstname("John");

$this->lastname("Doe");

}

}

}

Je ziet dat binnen de class de properties worden aangeroepen met “$this->”;

Deze “this” verwijst naar de properties binnen het opgeroepen object. Als je met dezelfde class meerder object aanmaakt wijzen deze naar binnen de verschillende objecten.

Binnen elke class ( object ) heb je te maken met getters en setters (eenvoudig schrijven en lezen van een propertie. Ik maak ze altijd zo dat de setter dezelfde naam heeft als de variabele. Het verschil zit hem dan in de haakjes. De getters geef ik de variabele naam met “get\_” ervoor. Je mag natuurlijk altijd alle namen kiezen die je wilt als je het maar consequent doet.

Binnen het boven genoemde voorbeeld zie een aantal dingen:

* Een user wordt beschreven door een aantal properties die private en public kunnen zijn.
* Een aantal functies zijn publiek andere privé.

Het onderscheid waar op wordt gekozen voor privé of publiek heeft te maken of een variabele direct wel of niet gezet moet kunnen worden (overerving – komt later), en het wel en niet kunnen opvragen van functies of methodes buiten het object.

Binnen dit object is het niet de bedoeling dat je van buitenaf aan kan geven of een gebruiker ingelogd is vandaar dat deze private gezet moet worden. (je kan voor inloggen ook natuurlijk Sessions gebruiken). Dit geldt ook voor het user en password zetten en getten.

Als je het object vraag in te loggen kunnen deze aangemaakt worden bij het opstarten van het object. Niet daarna dit uit veiligheidsoverwegingen. Het hele user en login verhaal is eigenlijk wat complexer betref veiligheid etc. maar voor een voorbeeld is dit goed.

## Binnen php een object initialiseren doormiddel van een class.

Een user aanmaken met php gebeurt als volgt:

$student = new user;

Op het moment van aanroepen wordt het object aangemaakt en de username welke anonymous heet. Helaas is er verder weinig te beleven voor deze user want ik kan niet inloggen. We kunnen de fullname op te vragen binnen dit object:

echo $student->get\_fullname();

// dit geeft aangezien in de \_\_construct een do\_user\_info ()

// zit en er geen user is gespecificeerd het volgende terug:

// “John Doe”

Stel nu dat je meer weet van de persoon dan kan je uitgebreider gebruik maken van deze class. Doormiddel van een aantal functies kun je een login script schrijven welke redelijk compact is.

$student = new user (“0123456”);

$student->do\_login ();

if ( $student->get\_logged() ){

echo “hello ”.$student->get\_fullname();

}else{

if ($student->get\_username() == “anonymous” &&

$student->get\_fulname () == “John Doe”){

echo “Sorry we do not know you!”;

}else{

echo “There is a mismatch in username and password”;

}

}

Aangezien er geen password meegegeven wordt kan de persoon niet ingelogd worden. Deze login zal dan ook . Als het object wordt geopend terwijl er zowel een goede username als password wordt verzonden zal een kan hij geeft de bovenstaande functie een oke.

## --optioneel--: Super/parent classes en overerving

Eigenlijk heeft inloggen weinig te maken met de user (Class) als zodanig. Als je begint is dat wellicht wel handig om te doen. Stel je hebt als student gebruiker deze class geschreven om gebruikers info af te handelen.

Nu zijn object zo gemaakt dat een class of object een afgeleide van een ander kan zijn. Zeg maar een class kan ouders hebben waarvan hij of zij is afgeleid. Voordeel hiervan is dat alle properties en functionaliteit overerft behalve de privé items. Probleem is wel dat php toelaat om privé items te openen van de superclass/extends.

Privé en Publiek toegang voor binnen en buiten objecten Zoals het zou moeten.

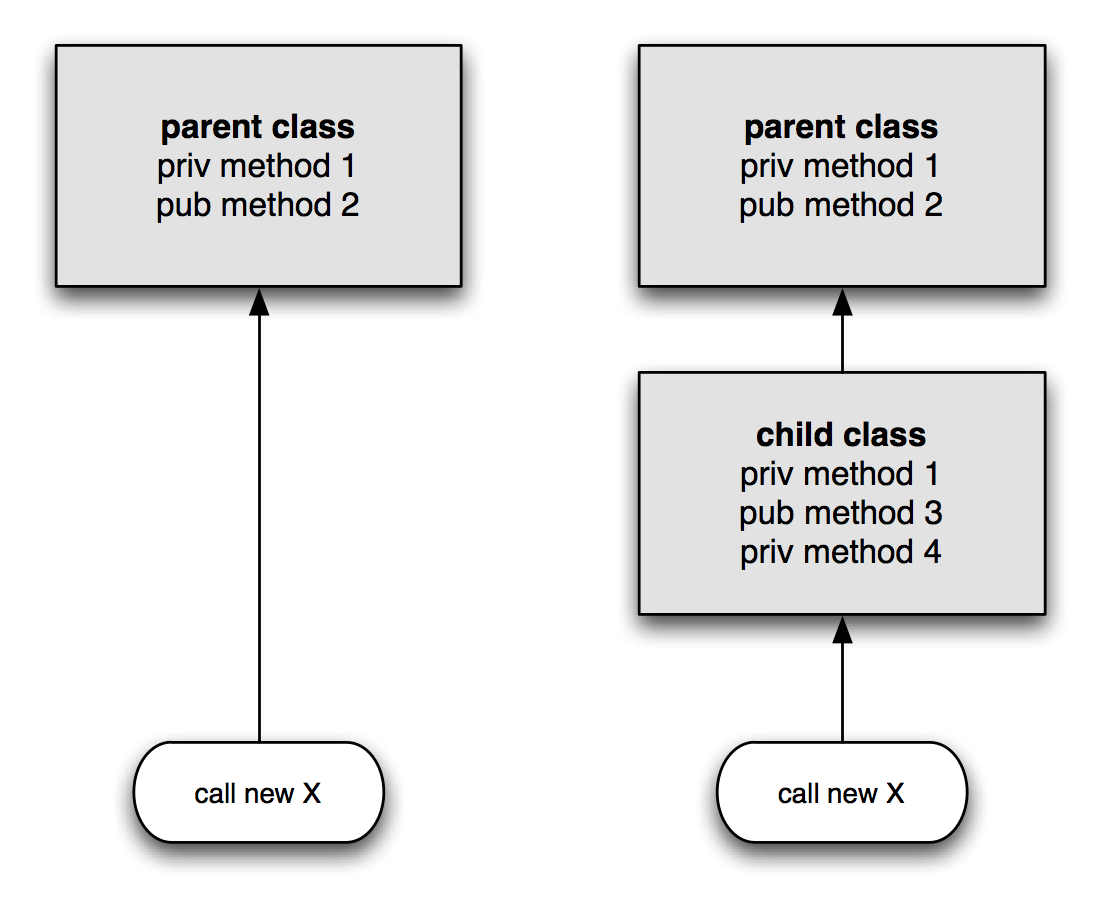
|  |  |  |
| --- | --- | --- |
|  | Public Method | Private Method |
| **Parent** | Yes | Yes |
| **Child / extended** | Yes | No |
| **Outside** | Yes | No |

Privé en Publiek toegang voor binnen en buiten objecten zoals PHP het doet.

|  |  |  |
| --- | --- | --- |
|  | Public Method | Private Method |
| **Parent** | Yes | Yes |
| **Child / extended** | Yes | **Yes!!!** |
| **Outside** | Yes | No |

Je moet dit eigenlijk zijn als dat de het kind een uitbreiding is van de ouder. Voordeel is dat als je eerst functionaliteit zou hebben geschreven voor de object dezelfde functionaliteit te en scripts te open zijn door het kind aan te roepen. Functies met dezelfde naam kunnen en worden dan mits anders aangegeven door het kind uitgevoerd. Dit extra oproepen kan dan in de locale functie met parent::functienaam().

In het voorbeeld hier onder kun je het volgende doen

X = new parent;

X->method2 ();

//Maar ook;

X = new child;

X->method2 ();

// Method2 is van parent

X->method3 ();

//En niet!!

X->method1 ();

X->method4 ();

Met andere woorden de wat meer standaard methodes staan dus in de parent. Je zou ook kunnen zeggen als je functionaliteit door meerdere classes gebruikt kan je een parent class maken waarin je deze verzamelt.

Met betrekking op het user voorbeeld van eerder zou je kunnen stellen dat user functionaliteit algemener is dan de login je kan wellicht beter onderscheid maken:

class user {

private $id;

public $firstname;

public $lastname;

private $secondname;

public function \_\_construct($username="anonymous"){

$this->do\_user\_info();

}

private function firstname($firstname){

$this->firstname = $firstname;

}

private function lastname ($lastname){

$this->lastname = $lastname;

}

public function get\_lastname(){

return $this->lastname;

}

public function get\_firstname (){

return $this->firstname;

}

public function get\_fullname (){

return $this->get\_firstname()." ".$this->get\_lastname();

}

public function do\_user\_info (){

// depending on username set we will set name properties

if ($this->get\_username() == "anonymous"){

$this->firstname("John");

$this->lastname("Doe");

}else{

// get user data and do something different

}

}

}

class login extends user {

private $username;

private $password;

private $logged;

public function \_\_construct($username="anonymous", $password=""){

parent::\_\_construct($username);

$this->username ($username);

$this->password ($password);

$this->logged (false);

}

private function logged ($status){

$this->logged = $status;

}

public function get\_logged (){

return $this->logged;

}

//METHODS: general user functions

private function username ($username){

$this->username = $username;

}

public function get\_username (){

return $this->username;

}

private function password ($password){

$this->password = $password;

}

//METHODS: Doers

public function do\_login (){

$this->logged(true);

$this->do\_user\_info ();

}

}

$student = new login (“0123456”);

$student->do\_login ();

if ( $student->get\_logged() ){

echo "hello ".$student->get\_fullname();

}else{

if ($student->get\_username() == "anonymous"

&& $student->get\_fulname() == "John Doe"){

echo "Sorry we do not know you!";

}else{

echo "Here is a mismatch in username and password";

}

}

# Opdrachten

## opdracht 1: Query-class.

Maak een class waarmee voor jou belangrijke db functies kunnen worden uitgevoerd. Van belang is dat voornamelijk de verwerking van de query’s wat beter wordt geautomatiseerd. Denk dan aan eenvoudiger output: Select een associative array retourneert deze als zodanig. Eventueel is het als uitbreiding mogelijk om daarnaast een library-super “class” de maken die connecties naar de database afhandelt.

## Opdracht 2: Database opdracht:

Maak een database-tabel aan voor een blog.. Een blog tabel bevat blog-posts met de volgende data: id, samenvatting, post, datum, tijd. Vul een aantal records in zodat we wat dummy data hebben. Je mag ook een gelijkende andere database maken, zolang er maar minimaal dit soort velden in terug komt. Denk aan Een lijst met gedane vakken van het afgelopen jaar en de behaalde cijfers. Laat dit weten aan de docent en vraag om toestemming.