4.1 Calculator+ deel 1 (OOP)

Intro

We gaan een calculator maken met een object. Hierbij zullen we ook een begin maken met het MVC-model. Dit model is een standaard voor het ontwikkelen van complexere web sites en wordt ook in Laravel gebruikt. Dit is dus een eerste stapje richting Laravel.

In het eerste deel gaan gebruik maken en oefenen met:

1. het maken van een form met een keuze lijst
2. het aanmaken van een class
3. het instantiëren van een object aan de hand van de class
4. het maken van methods (=functies in een class)
5. het aanroepen van methods
6. het opstellen van een switch-case constructie

Drie files

We gaan drie files maken:

```
form.html
result.php
includes/calc.php
```

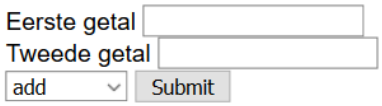
Form.html

We beginnen met de form.html. Maak een HTML-pagina met een form. Het form heeft drie velden plus een submit knop.

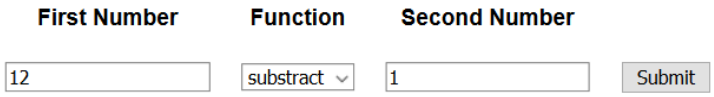
We gaan ook een drop down box maken in het form, lees hier hoe dat werkt:

> https://www.w3schools.com/html/html_form_elements.asp

Het form in eenvoudige vorm ziet er als volgt uit:



We zetten nu het form in een tabel, zodat het er iets fraaier uit ziet. Gebruik hiervoor `cellpadding="10"`

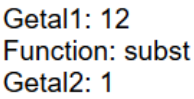


Het form ziet er op deze manier iets fraaier uit, zorg ervoor dat als je de submit knop in drukt dat je dan de pagina result.php laat zien.

Resultaat.php

Zet in resultaat.php code, zodat je de form- variabellen kun zien, dus je ziet het volgende als (bijvoorbeeld als je op submit drukt):

De output ziet er bijvoorbeeld als volgt uit:



We gaan dit niet netjes maken, we doen dit alleen om te controleren of het form goed werkt en of de variabelen goed worden verstuurd. Gebruik je in het form POST of GET? Zet in het form expliciet welke methode je gebruikt, we kunnen het dan later evetueel nog aanpassen.

Zet in de output ook een link terug naar het form: `<a href=.....>Back to form</a>`

Includes/class.php

Nu gaan we onze eerste class maken!

We gaan in dit voorbeeld niet op de meest eenvoudigste manier een programma maken (een calculator in php kun je echt veel eenvoudiger maken), maar we gaan dit op deze manier doen om OOP te leren. Het voorbeeld is eenvoudig, maar de manier waarop we het doen is best ingewikkeld.

Ready, voor je eerste PHP OOP-ervaring?

We gaan naar de file calc.php en maken daar de class Calc

```php
<?php

class Calc {
    private $number1;
    private $number2;

    public function setNumber1($number){
        $this->number1 = $number;
    }

    // maak hier een tweede public method die number2 initialiseerd

}
?>
```

Public function $setNumber2

Maak zelf de tweede public function op de plaats waar nu het commentaar staat. We maken $number1 en $number2 'private', hetgeen betekent dat ze van buiten de class niet beschikbaar zijn. Dit zorgt ervoor dat je als gebruiker van deze class niet precies hoeft te weten hoe alle variabelen heten (dit is een vorm van *encapsulation*).

In de file resultaat.php gaan we nu de class instantiëren; dit betekent dat we een object gaan aanmaken aan de hand van de blauwdruk van de class die we zojuist hebben gemaakt.

We doen dit met: `$myCalcObject = new Calc;`

Include

Om new Calc te kunnen aanroepen moet de de file waarin de class is gedefinieerd wel worden included:

```
include "includes/calc.php";
```

'Include' plaatst als het ware inhoud van de file `calc.php` in `resultaat.php`

In plaats van `include` te gebruiken zou je ook alle code in éé'n grote file kunnen plaatsen. Dit doen we **niet** omdat je dan snel het overzicht verliest. Door de code op te delen en op een logische plaats in de juiste folder te plaatsen houd je het overzicht. In Laravel zul je zien dat je een zeer uitgebreide folder structuur hebt. Alles staat dan op een logische plaats. Onderdeel van het leren van Laravel is weten waar wat staat en hoe de verschillende files met elkaar verband houden. Maar daarover later meer.

Methods aanroepen vanuit resultaat.php

We kunnen nu de methods (functions) vanuit resultaat.php aanroepen die we in de class hebben aangemaakt:

```
$myCalcObject->setNumber1(...);
```

Op de plaats van de puntjes zet je de variabele die uit het form komt en die we zojuist nog hebben afgedrukt.

Op dezelfde manier geven we de object property `$number2` ook een waarde. We maken een tweede method (functie) waarmee we de property (variabele) $number2 de juiste waarde geven: `$myCalcObject->setNumber2(...);`

Het object 'weet' nu welke twee getallen we hebben verstuurd met het formulier. Deze twee getallen staan in de class properties $number1 en $number2

Nieuwe method: executeCalculation

We gaan nu een method maken die het resultaat berekent `echo "Resultaat is: ".$myCalcObject->executeCalculation();`

Laten we weer kleine stapjes maken en eerst doen alsof we alleen kunnen optellen.

We maken dan een method om de twee getallen op te tellen en return-en het resultaat.

```
    public function executeCalculation() {
        // return op de plaats van de puntjes ($number1 pus $number2).
□□//let op dat je elementen van dit object met $this-> aanspreekt.
□□return ..... ;
    }
```

In de code van resultaat.php, voegen we dan een regel toe:

```
echo "Result is: ".$myCalcObject->executeCalculation();
```

Als het goed is dan zie je dit als je het form post:

Getal1: 34
Function: add
Getal2: 12
Result is: 46
back

Stap 2, Switch control stucture

Nu gaan we de *executeCalculaiton* method zodanig aanpassen dat de andere bewerkingen, substitude (aftrekken), multiply (verenigvuldigen), divide (delen) en addition (optellen) ook worden ondersteund. We passen hiervoor eerst de method *executeCalculation* aan, zodat de functie (add, subst, mult of div) wordt meegegeven. We kunnen dit met if-then-else doen, maar het is netter om een switch-case constructie te gebruiken. Deze kun je ook iets eenvoudiger uitbreiden (wat we natuurlijk nog gaan doen).

Lees de documentatie: https://www.php.net/manual/en/control-structures.switch.php

Bekijk ook de voorbeelden bij deze documentatie. In example #2 wordt een switch gemaakt aan de hand van een string ($i), de cases zijn 'apple', 'bar' of 'cake'. Wij maken onze switch aan de hand van de variabelen die we meegeven bij het aanroepen van de method (functie) executeCalculation en onze cases zijn: 'add', 'subst', 'mult' of 'div'.

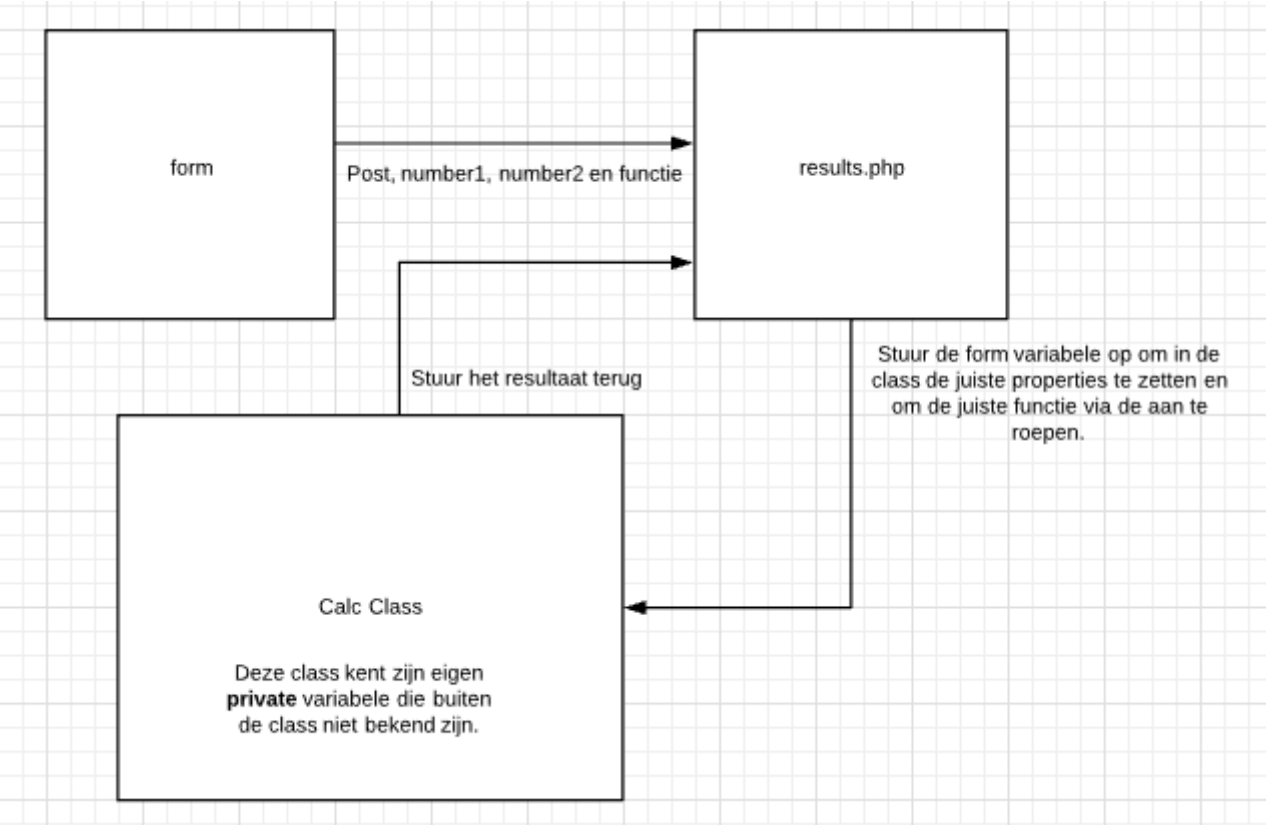
Maak de method *executeCalcultion* af:

1. zorg ervoor dat de functie ('add', 'subst', 'mult' of 'div') wordt meegegeven als parameter.
2. Maak een switch aan de hand van deze parameter en zet bereken bij elke case het juiste resultaat.
3. return het resultaat.

Klaar!

Je hebt nu de eerste milstone bereikt, gefeliciteerd!

In schema ziet de flow van deze mini applicatie er als voglt uit:



Laat het resultaat zien aan je docent voordat je verder gaat met deel 2.

Revision #19
Created 2019-09-10 14:32:24 UTC by Admin
Updated 2020-05-11 08:38:08 UTC by Max