OOP in PHP

Key concecpts

private, public, protected, encapsulation, final, const, static, self, __construct, __destruct, __toString, type hinting, abstract, interface, clone, __clone, immutable, namespaces, autoloading

Alternatieve bronnen

https://phpro.org/tutorials/Object-Oriented-Programming-with-PHP.html
http://php.net/manual/en/language.oop5.php
https://www.sitepoint.com/php-53-namespaces-basics/
https://www.pluralsight.com/courses/object-oriented-php-essential-constructs
https://www.pluralsight.com/courses/object-oriented-php-classical-inheritance-model

1. Getting started

Benodigde Software:

- PHPStorm IDE

  https://www.jetbrains.com/phpstorm/
  https://www.jetbrains.com/help/phpstorm/2016.3/quick-start-guide.html

- php (CLI)

  http://php.net/downloads.php

- Composer
  https://getcomposer.org/download/

Maak een project in PHPStorm.

Configureer het project zodanig dat de command line PHP-interpreter gebruikt wordt.

https://www.jetbrains.com/help/phpstorm/2016.3/configuring-local-php-interpreters.html

Maak het bestand app.php (File, New PHP File) en voer uit.

app.php

```
<?php
phpinfo();
```

2. Een eerste klasse

Maak de klasse Point in Point.php

http://php.net/manual/en/language.oop5.basic.php
http://php.net/manual/en/language.oop5.visibility.php
http://php.net/manual/en/language.oop5.decon.php#object.construct
http://php.net/manual/en/language.oop5.magic.php#object.tostring

Point.php

```
<?php

final class Point
{
    private $x;
    private $y;

    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    function __toString()
    {
        return "($this->x , $this->y)";
    }

    public function calculateDistance(Point $point)
    {
        return sqrt( ($this->x-$point->x)*($this->x-$point->x)+
                    ($this->y-$point->y)*($this->y-$point->y) );
    }

}
```

app.php

```
<?php
require_once 'Point.php';
$point1=new Point(1,2);
print($point1);
$point2=new Point(5,11);
print($point2);
$distance=$point1->calculateDistance($point2);
```

Maak getters en setters en gebruik deze in app.php (Code, Generate, ... of Alt-Insert).
Maak een constante MAXIMUM_XY, stel deze gelijk aan 100. Deze constante wordt in de klasse Point.php gebruikt als self::MAXIMUM_XY. Waarden voor x en y die binnenkomen moeten altijd liggen tussen 0 en MAXIMUM_XY. Pas setX, setY aan. Roep vanuit de constructor setX en setY aan.

http://php.net/manual/en/language.oop5.constants.php

3. Static variables

Tel hoeveel keer de constructor aangeroepen wordt. Maak hiervoor de static variable countInitialisations.

```
private static $countInitialisations=0;
```

Deze variabele wordt gebruikt in de klasse Point als self::$countInitialisations. Verhoog deze waarde in de constructor. Voorzie ook een getter voor countInitialisations.

http://php.net/manual/en/language.oop5.static.php

app.php

```
<?php
require_once 'Point.php';
$point1=new Point(1,2);
$point2=new Point(5,11);
print(Point::getCountInitialisations());
$point2=null;
print(Point::getCountInitialisations());
```

Probeer app.php uit.

Voorzie een destructor __destruct waar je de waarde van countInitialisations verlaagt met 1. Test opnieuw uit.

http://php.net/manual/en/language.oop5.decon.php

Probeer code completion om de destructor te genereren: __destr<ctrl-space>

4. Abstract class en inheritance

Maak de klasse Shape. Van een Shape weten we concreet dat er een waarde voor een Punt gespecifieerd moet worden, we kennen de uitwerking van de functie __toString en we weten dat er een omtrek berekend moet kunnen worden via de functie calculatePerimiter maar we weten niet hoe de omtrek voor een algemene Shape berekend kan worden. Shape is een abstracte klasse. Er kunnen geen objecten gemaakt worden van deze klasse. De klasse dient als superklasse van concrete klasse zoals Rectangle. In deze klasse moet de abstracte functie wel uitgewerkt worden.

Shape.php

```
<?php

abstract class Shape
{
    private $point;

    public function __construct(Point $point)
    {
        $this->point=$point;
    }

    function __toString()
    {
        return $this->point->__toString();
    }

    public abstract function calculatePerimiter();

}
```

Rectangle.php

```
<?php

final class Rectangle extends Shape
{
    private $width, $height;
    public function __construct(Point $point, $width, $height)
    {
        parent::__construct($point);
        $this->width=$width;
        $this->height=$height;
    }

    public function calculatePerimiter()
    {
        return 2*$this->width+2*$this->height;
    }

    public function __toString()
    {
        return "Rectangle, Point= ". parent::__toString() ." width=
                $this->width height= $this->height";
    }
}
```

app.php

```
<?php
require_once 'Point.php';
require_once 'Shape.php';
require_once 'Rectangle.php';
$point=new Point(1,2);
$rectangle=new Rectangle($point,12,2);
print($rectangle);
```

Maak de klasse Circle afgeleid van Shape. Circle heeft als bijkomende eigenschap radius. Test uit.

5. Interfaces

Binnen een interface worden abstracte methodes en constantes gedefinieerd. Een concrete klasse die de interface implementeert moet de abstracte methodes uitwerken.

http://php.net/manual/en/language.oop5.interfaces.php

Binnen de interface Drawable wordt de methode draw gedefinieerd maar niet uitgewerkt. De klasse Shape implementeert de interface en de klasse Rectangle is verplicht om de methode draw uit te werken.

Drawable.php

```
<?php

interface Drawable
{
    public function draw();
}
```

Shape.php

```
<?php

abstract class Shape implements Drawable
{
    private $point;

    public function __construct(Point $point)
    {
        $this->point=$point;
    }

    function __toString()
    {
        return $this->point->__toString();
    }

    public abstract function calculatePerimiter();

}
```

Rectangle.php

```
<?php

final class Rectangle extends Shape
{
    private $width, $height;
    public function __construct(Point $point, $width, $height)
    {
        parent::__construct($point);
        $this->width=$width;
        $this->height=$height;
    }

    public function calculatePerimiter()
    {
        return 2*$this->width+2*$this->height;
    }

    public function __toString()
    {
        return "Rectangle, Point= ". parent::__toString() .
               " width= $this->width height= $this->height";
    }

    public function draw()
    {
        print($this->__toString());
    }
}
```

app.php

```
<?php
require_once 'Point.php';
require_once 'Drawable.php';
require_once 'Shape.php';
require_once 'Rectangle.php';
$point=new Point(1,2);
$rectangle=new Rectangle($point,12,2);
$rectangle->draw();
```

Pas de code voor Circle aan.

Interfaces worden dikwijls aan de rand van een domein gedefinieerd. Ze vormen een contract met de buitenwereld. Bijvoorbeeld bij een forech structuur hoort de interface Iterable. De klasse IterableString implementeert de interface dus een object van de klasse IterableString kan gebruikt worden in een foreach.

IterableString.php

```
<?php

final class IterableString implements Iterator {
    private $index;
    private $contents;

    public function __construct($contents)
    {
        $this->contents = $contents;
        $this->index = 0;
    }

    public function current()
    {
        return $this->contents[$this->index];
    }

    public function key()
    {
        return $this->index;
    }

    public function next()
    {
        $this->index++;
    }

    public function rewind()
    {
        $this->index=0;
    }

    public function valid()
    {
        return strlen($this->contents) > $this->index;
    }
}
```

app.php

```
<?php
require_once('IterableStirng.php');
$text=new IterableString("abcd");
foreach ($text as $character => $index){
    print("$index $character\n");
}
```

Probeer deze code uit.

6. Clone

Vertrekkend van onderstaande code. Verklaar de uitvoer van app.php.

Point.php

```
<?php

final class Point
{
    private $x;
    private $y;

    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    public function getX()
    {
        return $this->x;
    }

    public function setX($x)
    {
        $this->x = $x;
    }

    public function getY()
    {
        return $this->y;
    }

    public function setY($y)
    {
        $this->y = $y;
    }

    function __toString()
    {
        return "($this->x , $this->y)";
    }

}
```

Line.php

```
<?php

final class Line
{
    private $startPoint;
    private $endPoint;

    function __construct(Point $startPoint,Point $endPoint)
    {
        $this->startPoint=$startPoint;
        $this->endPoint=$endPoint;
    }

    public function __toString()
    {
        return "Line: $this->startPoint $this->endPoint";
    }

}
```

app.php

```
<?php
require_once 'Point.php';
require_once 'Line.php';
$startPoint=new Point(1,2);
$endPoint=new Point(3,4);
$line=new Line($startPoint,$endPoint);
print("$line\n");
$startPoint->setX(22);
print($line);
```

Uitvoer:

```
Line: (1 , 2) (3 , 4)
Line: (22 , 2) (3 , 4)
Process finished with exit code 0
```

Voeg het keyword clone toe aan de constructor van Line en probeer opnieuw. Via clone wordt er een copy gemaakt van het Point-object. Point heeft enkel primitieve waarden. Deze worden door het commando clone correct gekopieerd.

Line.php

```
<?php

final class Line
{
    private $startPoint;
    private $endPoint;

    function __construct(Point $startPoint,Point $endPoint)
    {
        $this->startPoint=clone $startPoint;
        $this->endPoint=clone $endPoint;
    }

    public function __toString()
    {
        return "Line: $this->startPoint $this->endPoint";
    }

    public function setStartPointX($x)
    {
        $this->startPoint->setX($x);
    }

    public function setStartPointY($y)
    {
        $this->startPoint->setY($y);
    }

    public function setEndPointX($x)
    {
        $this->endPoint->setX($x);
    }

    public function setEndPointY($y)
    {
        $this->endPoint->setY($y);
    }
}
```

Voer app.php opnieuw uit en bekijk de uitvoer.

In de huidige vorm is Line zelf nog niet correct clone-baar.

app.php

```
<?php
require_once 'Point.php';
require_once 'Line.php';
$startPoint=new Point(1,2);
$endPoint=new Point(3,4);
$line=new Line($startPoint,$endPoint);
$line2=clone $line;
print("$line2\n");
$line->setStartPointX(22);
print($line2);
```

Heeft als uitvoer

```
Line: (1 , 2) (3 , 4)
Line: (22 , 2) (3 , 4)
Process finished with exit code 0
```

De methode __clone wordt uitgevoerd wanneer het commando clone aangeroepen wordt. Deze methode kan toegevoegd worden in de klasse Line. Bij het aanroepen van clone op een Line-object wordt nu ook een clone gemaakt van beide Point-objecten.

Line.php

```
<?php

final class Line
{
    private $startPoint;
    private $endPoint;

    function __construct(Point $startPoint,Point $endPoint)
    {
        $this->startPoint=clone $startPoint;
        $this->endPoint=clone $endPoint;
    }

    function __clone()
    {
        $this->startPoint = clone $this->startPoint;
        $this->endPoint = clone $this->endPoint;
    }

    public function __toString()
    {
        return "Line: $this->startPoint $this->endPoint";
    }

    public function setStartPointX($x)
    {
        $this->startPoint->setX($x);
    }

    public function setStartPointY($y)
    {
        $this->startPoint->setY($y);
    }

    public function setEndPointX($x)
    {
        $this->endPoint->setX($x);
    }

    public function setEndPointY($y)
    {
        $this->endPoint->setY($y);
    }

}
```

Pas Line.php aan en voer app.php opnieuw uit.

7. 'Immutable' objects

Een immutable object is een object waarvan de toestand niet gewijzigd kan worden na creatie. Bij elke wijziging van de toestand wordt een nieuw object aangemaakt.

Om de klasse Point immutable te maken worden de setters verwijderd. Verder worden de methodes changeX en changeY aangemaakt. Deze geven een nieuw Point-object met aangepaste waarde voor x of y terug.

Point.php

```
<?php

class Point
{
    private $x;
    private $y;

    public function __construct($x, $y)
    {
        $this->x = $x;
        $this->y = $y;
    }

    public function changeX($x){
        return new self($x,$this->y);
    }

    public function changeY($y){
        return new self($this->x,$y);
    }

    public function getX()
    {
        return $this->x;
    }

    public function getY()
    {
        return $this->y;
    }

    function __toString()
    {
        return "($this->x , $this->y)";
    }

}
```

app.php

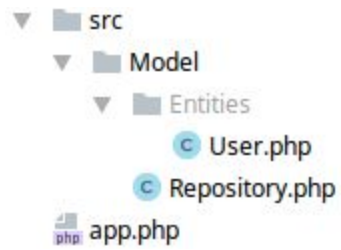
```
<?php
require_once 'Point.php';
$point=new Point(1,2);
$point2=$point;
$point=$point->changeX(12);
print($point);
print($point2);
```

7. Namespaces & autoloading

Namespaces en sub-namespaces worden gedefinieerd aan de hand van het keyword namespace.

Een van de PSR (PHP Standards Recommendations) is om één namespaces per bestand met een klasse of interface te gebruiken en om de namespaces te laten overeenkomen met de directory-structuur.

Als voorbeeld worden de klassen Repository en User in de bestanden src/Model/Repository.php en src/Model/Entities/User.php geplaatst.



src/Model/Repository.php

```
<?php namespace Model;

class Repository
{
    ...
}
```

src/Model/Entities/User.php

```
<?php namespace Model\Entities;

class User
{
    ...
}
```

In het bestand app.php wordt geen namespace gedefinieerd. Alle code in dit bestand hoort in de default-namespace (\ ).

app.php

```
<?php
require_once 'src/Model/Repository.php';
require_once 'src/Model/Entities/User.php';
$user = new Model\Entities\User();
$repository = new Model\Repository();
```

De klassen kunnen in app.php gebruikt worden in relatieve notatie (t.o.v. namespace \ )

```
$user = new Model\Entities\User();
$repository = new Model\Repository();
```

maar ook in absolute notatie:

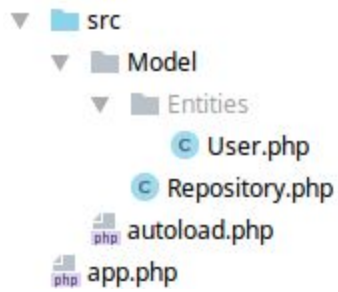
```
$user = new \Model\Entities\User();
$repository = new \Model\Repository();
```

Verder kunnen via het keyword use klassen en interfaces geïmporteerd worden zodat het oproepen van de klasse of interface eenvoudiger verloopt:

app.php

```
<?php
require_once 'src/Model/Repository.php';
require_once 'src/Model/Entities/User.php';
use Model\Repository;
use Model\Entities\User;
$user = new User();
$repository = new Repository();
```

Een autoloader (srr/autoload.php) vereenvoudigt het ophalen van code:



app.php

```
<?php
require_once 'src/autoload.php';
use Model\Repository;
use Model\Entities\User;
$user = new User();
$repository = new Repository();
```

src/autoload.php

```
<?php
if (!function_exists('classAutoLoader')) {
    function classAutoLoader($className)
    {
        $fileName = 'src/'.
                    str_replace('\\', '/', $className).
                    '.php';
        require_once $fileName;
    }
}
spl_autoload_register('classAutoLoader');
```

Voor elke niet gekende klasse wordt de functie classAutoLoader aangeroepen. In deze functie wordt

Model\Repository	omgezet naar	src/Model/Repository.php
Model\Entities	omgezet naar	src/Model/Entities/User.php

(voor Windows moet waarschijnlijk een andere directory seperator gebruikt worden: src\Model\Repository.php ipv src/Model/Repository)

http://php.net/manual/en/language.namespaces.rationale.php
https://mattstauffer.co/blog/a-brief-introduction-to-php-namespacing

Het genereren van een autoloader kan heel gemakkelijk via Composer. Composer is een dependency manager voor PHP-code.

composer.json

```
{
    "autoload": {
        "psr-4": {
            "Model\\": "src/Model/",
            "Model\\Entities\\": "src/Model/Entities"
        }
    }
}
```

Het commando

composer dump-autoload -o

genereert de map vendor en bestand vendor/autoload.php

app.php

```
<?php
require_once 'vendor/autoload.php';
use Model\Repository;
use Model\Entities\User;
$user = new User();
$repository = new Repository();
```

https://getcomposer.org/doc/01-basic-usage.md#autoloading