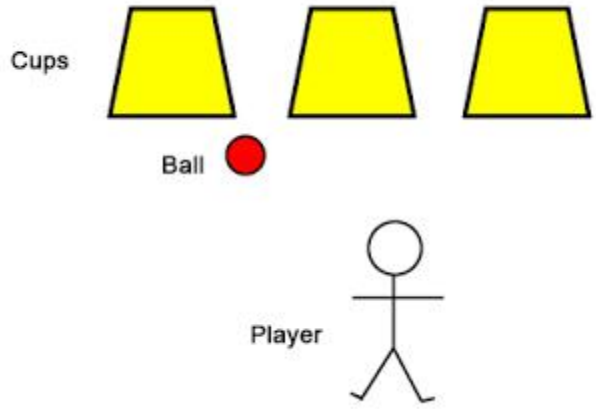
Shell Game (OOP)

Het volgende project gaat over het "Shell Game" (Balletje Balletje) in OOP stijl.



Deel 1

1. Hoeveel objecten telt dit spel? (volgens bovenstaand schema)
2. En hoeveel classes?
3. Maak een map **shell-game** met daarin het bestand **index.php.**
4. Maak een class **Ball** in een apart bestand genaamd **class-ball.php**.
    - Deze heeft alleen de property **color** (string).
    - Include dit bestand in **index.php**
5. Maak in **index.php** een object aan van **Ball** en zet deze in een variabele.
    - Zet de kleur van dit object op "**red**".
6. Maak een class **Player** in een apart bestand genaamd **class-player.php.**
    - Deze heeft de properties **name** (string) en **amount** (integer)
    - Include dit bestand in **index.php**
7. Maak in **index.php** een object aan van **Player** en zet deze in een variabele.
    - Geef deze je eigen naam en geef jezelf **100** als startkapitaal (amount).
8. Maak een class **Cup** in een apart bestand genaamd **class-cup.php.**
    - Deze heeft de properties **color** (string)**, type** (string) en de methods **liftUp**() en **putDown**(). De methoden zelf mag je nog leeg laten.
    - Include dit bestand in **index.php**
9. Maak in **index.php** de objecten aan van **Cup** en zet deze in aparte variabelen.
    - Geef ze allemaal de kleur **yellow** en het type **plastic**.

Alle properties en methods van de classes in deze opdracht worden gedefinieerd met **public** (zoals in onderstaand voorbeeld).

Denk wederom aan het schrijven van **nette, gestructureerde code** en **duidelijke comments**!

Voorbeeld Class & Object

Creëer de class Car:

```
class Car {

    // property
    public $brand = '';

    // method
    public function drive() {
        // here should come the code for driving
    }

}
```

Maak een object (instance) van Car:

```
// create a new object
$someCar = new Car();
$someCar->brand = 'Ford';
$someCar->drive();
```

Deel 2

Verder met het volgende deel van de Shell Game.

View

1. Maak een **view.php** bestand met daarin een html pagina
2. Zet hierin de volgende styling:

```
	/* CSS styling */
	body { text-align:center; font-family: arial; }
	.cups { margin:20px auto 70px; width:520px; height:183px; border-bottom:3px solid
#000; }
	.cup { width:132px; height:113px; background-image:url(cup.png); float:left; margin:0
20px; }
	.ball { position:absolute; margin-top:72px; margin-left:44px; z-index:-1; width:43px;
height:41px; background:url(ball.png); }
	.putdown { margin-top:70px; }
	.putdown .ball { margin-top:72px; }
	.liftup { margin-top:0px; }
	.liftup .ball { margin-top:142px; }
	.player { margin:0 auto; width:92px; height:243px; background:transparent url(player.png)
no-repeat 0 60px; }
	.clear { clear:both; }
	.red { background-color:red; }
	.yellow { background-color:yellow; }
	.blue { background-color:blue; }
```

3. Download de images uit deze post en plaats deze in dezelfde map als de view
4. Include de view in de **index.php**
5. Zet de volgende code in de <body> van je html:

```
<div class="cups">
   <div class="cup yellow liftup">
      <div class="ball red"></div>
   </div>
   <div class="cup yellow putdown"></div>
   <div class="cup yellow"></div>

   <div class="clear"></div>
</div>

<div class="player">
   <strong>*name*: *amount*</strong>
</div>
```

Player

1. Maak in de class Player de method **show()**.
2. Zorg dat deze method de html code voor de speler toont:

```
<div class="player">
   <strong>*name*: *amount*</strong>
</div>
```

Vervang hierbij *name* en *amount* door de waarde van de properties.

3. Vervang in de view de html code van de speler door deze method aan te roepen.
4. Speel hiermee door de **name** en **amount** aan te passen en kijk of het werkt.

Ball

1. Maak in de class Ball de method **show()**.
2. Zorg dat deze method de html code van de bal toont:

```
<div class="ball red"></div>
```

Hierbij kan **red** ook ook een andere kleur bevatten, namelijk **blue** of **yellow**.

3. Vervang in de view de html code van de bal door deze method aan te roepen.
4. Verander de property **color** en kijk of het werkt.

Cup

1. Maak in de class Cup de method **show()**.
2. Zorg dat deze method de html code van de beker toont, bijv:

```
<div class="cup yellow"></div>
```

Hierbij kan **yellow** ook ook een andere kleur bevatten, namelijk **blue** of **red**.

3. Vervang in de view de html code van de cups door deze method aan te roepen.
4. Verander de property **color** en kijkt of het werkt (het balletje zie je nu even niet meer, deze ligt onder de beker)
5. Voeg de property **positionUp** (boolean) toe.
6. Schrijf nu de code voor de methods **liftUp**() en **putDown**() waarbij je de waarde van **positionUp** aanpast.
7. Pas nu de method **show**() aan door als de positie omhoog is **liftup** toe te voegen in de html en in het andere geval **putdown**:

```
<div class="cup yellow liftup"></div>
```

8. Vervang in de view de html code van de bekers door deze method aan te roepen.
9. Verander de property **color** en kijkt of het werkt.
10. Roep voor bepaalde cups **liftUp**() aan.
11. Kijk ook wat er gebeurt als je vervolgens ook **putDown**() aanroept.
12. Verwijder de property **containsBall**.
13. Voeg property **ball** toe en zet deze default op **null**.
14. Vul deze property voor één van de cups (objecten) met het bal object.
15. Pas in Cup de **show**() method aan. Als de property **ball** een object van de class **Ball** bevat, dan moet de html van de bal worden getoond, bijv:

```
<div class="cup yellow liftup">
<div class="ball red"></div>
</div>
```

Spel

1. Herschrijf in index.php door de objecten van cups in een array te plaatsen
   - Pas dit ook aan in de view
2. Schrijf een functie waarmee je het spel start en het volgende doet:
   - Zet alle bekers naar beneden
   - Zorg dat de bal random onder één van de bekers wordt geplaatst
   - Sla in een sessie op waar de bal ligt
3. Zorg dat je met behulp van een GET een gok kan doen onder welke beker de bal zit, bijv:

```
index.php?show_cup=1
```

4. Test dit door in de URL de GET mee te geven
5. Voeg nu de property **id** toe aan class **Cup**
6. Pas vervolgens de in **show**() method de html aan door ** ** te vervangen door **<a>** en voeg daaraan een **href** toe, bijv:

```
<a href="?show_cup=..." class=".....">
```

   Achter **show_cup** komt de **id** te staan

7. Geef nu alle bekers een **id.**
8. Maak in de view een link waarmee je het spel opnieuw kan beginnen.
9. Kijk hoe je het spel een logische flow kunt geven en dat je als gebruiker de juiste meldingen te zien ("kies een beker", "goed / fout gegokt")

Game

1. Maak nu in een apart php bestand de class **Game** aan met de property **amountPerGame** (float).

2. Maak hiervan een object aan en geef deze een **amountPerGame** van 20.
3. Probeer nu of het lukt om de **amount** van de speler kunt aanpassen bij win en bij verlies.
    - Sla hiervoor de **amount** ook op in een cookie
    - Zorg dat de cookie waarde de volgende keer weer wordt uitgelezen en aan de speler wordt toegekend

Deel 3

Game

1. Zet functie **start**() in class Game
2. Splits de code op door aparte methods te maken voor:
    - Het neerzetten van alle bekers
    - Bal onder een beker zetten
3. Maak voor elke class een constructor. De constructor bevat params waarmee alle properties kunnen worden gezet. Voorbeeld player:

```
public function __construct($name = '', $amount = 0) {
   $this->name = $name;
   $this->amount = $amount;
}
$player = new Player('Victor', 100);
```

Intermezzo: magic methods

Magic Ball

Dit is niet onderdeel van het spel, maar bedoeld om het e.e.a. te demonstreren.

1. Maak een apart bestand aan **class-magic-ball.php**
2. Maak hierin een child class van Ball met de naam **MagicBall** met een private property **size** en een protected property **type**.
3. Maak een method **show**() die alleen de naam van de class en de kleur op het scherm toont.
4. Maak een bestand **test-magic.php** en include **class-ball.php** en **class-magic-ball.php**.
5. Creer in dit bestand een object van **MagicBall** en roep **show**() aan.
6. Zet voor het object de waarde van **size** op 20. Wat zie je?
7. Zet voor het object de waarde van **type** op “soft”. Wat zie je?
8. Zet nu in de class beide properties op public.
9. Probeer stap 6 en 7 nogmaals. Wat zie je nu?
10. Bekijk de bijgevoegde bestanden met daarin uitleg en tests van de magic methods **__get**, **__set**, **__call** en **__destruct**.

Deel 4

(1) __toString()

1. Maak voor de class **Player** een method **__toString**(). Deze method geeft de html terug als return. Het lijkt op de show() method, maar dan zonder echo en met return waarde. Voorbeeld:

```
    public function __toString() {
        $html = '';
        $html .= '<div class="player">';
        $html .= '</div>';
        return $html;
    }
```

2. Pas nu in de **view.php** deze code aan:

```
$player->show();
```

   naar:

```
echo $player;
```

3. Wat doet deze code?
4. Pas nu in de class **Player** de method **show**() aan, zodat deze weer DRY wordt.
5. Doe stap 1 t/m 4 ook voor de class **Cup.**
6. En ook voor de class **Ball**.

(voor de uitwerking zie Deel 4-1.zip)

(2) Constanten

1. Maak in de class **Cup** twee constanten voor **type** met de waarde 'plastic' en 'glass'.
2. Vervang bij het creëren van de bekers in de **index.php** de waarde voor **type** door een constante.
3. Vervang in de constructor de default waarde van type in één van de constanten:

```
public function __construct($id, $color = '', $type = '', $positionUp
= false, $ball = null) {
```

4. Pas de **__toString**() method aan dat er geen kleur wordt meegegeven als het een beker van glas is.
5. Test dit door het spel met bekers van glas te spelen.

(3) Interfaces

1. Maak een interface voor de class **Ball** met daarin de method **show**() en **__toString**().
2. Implementeer de interface in de class **Ball**.
3. Pas de code zo dat in de class **Cup** de bal alleen maar gezet kan worden met de method **setBall**().
4. Maak nu een interface voor **Cup** met de methods **liftUp**(), **putDown**(), **setBall**(), **show**() en **__toString**().

(4) Child classes and abstract

1. Maak twee child classes voor **Player** genaamd **HumanPlayer** en **ComputerPlayer**.
2. Verplaatst de **__toString** method van **Player** naar **HumanPlayer**.
3. Maak voor **ComputerPlayer** een **__toString** method, die een lege string retourneert.
4. Definieer in de class **Player** de method **__toString** als abstract.
5. Definieer ook de class als abstract.
6. Definieer de class **Ball** als final.
7. Voor nu nog eens de test uit van MagicBall. Wat zie je?

Deel 5

Voor degene, die nog een extra uitdaging willen met de Shell Game.

(1) Schaalbaarheid

1. Maak het aantal bekers in het spel schaalbaar, zodat je het spel ook met bijvoorbeeld 6 of 10 bekers kunt spelen.
2. Maak het spel zodat meerdere bekers een bal kunnen bevatten (waarbij tenminste 1 beker leeg is, anders win je natuurlijk altijd).
3. Iets lastiger, maar kun je ook het aantal spelers schaalbaar maken?

(2) Opties voor de speler

1. Zorg dat een speler zelf kan kiezen welke kleur de bekers worden.
2. Zorg dat een speler zelf kan kiezen welke kleur de bal krijgt.

(3) Static

1. Kun je voor alle classes een static property maken, die de standaard format geeft voor de html output. Bijvoorbeeld:

```
static public $html = '<div class="ball %s"></div>';
```

2. Pas de __toString methods aan, zodat deze gebruik maakt van de static property. Tip gebruik de PHP functie sprintf(), waarbij je parameters vervangt. Bijvoorbeeld:

```
public function __toString() {
    return sprintf(self::$html, $this->color);
}
```

Nu kun je de html format van de classes (en objecten) aanpassen. Bijvoorbeeld:

```
Ball::$html = '<div class="ball %s"><span
class="yellow">BAL</span></div>';
```

3. Properties zouden in principe door middel van getters en setters moeten worden benaderd. Maak de property private en maak hiervoor een public getter en setter.