



**Vlaamse Dienst voor Arbeidsbemiddeling en
Beroepsopleiding**



Deze cursus is eigendom van VDAB ©

Inhoudstafel

Een woordje vooraf.....	5
Over deze cursus	5
Conventies.....	6
Voorkennis.....	6
Ontwikkelomgeving	6
Extra hulp?.....	7
Hoofdstuk 1 Inleiding.....	10
Een eerste blik.....	10
Dynamische inhoud toevoegen	12
Presentatie versus logica	14
Hoofdstuk 2 De code uitgespit	18
Variabelen	18
Expressies	23
Controlestructuren.....	27
Lussen	31
Hoofdstuk 3 Gebruikersinvoer	35
Via de URL met GET.....	35
Via een formulier met GET	38
Via een formulier met POST	40
Hoofdstuk 4 Arrays	42
Situatieschets en kennismaking.....	42
Arrays invullen	45
Arrays doorlopen.....	47
Hoofdstuk 5 Informatie onthouden.....	50
Noodzaak?	50
Een afgewerkt voorbeeld.....	50
Hoe sessions werken.....	54

Iets over cookies	55
Hoofdstuk 6 Programma's ontwikkelen	58
Splitting van presentatie en logica.....	58
In de praktijk.....	60
Hoofdstuk 7 Herhalingsoefeningen.....	65
Hoofdstuk 8 Object-oriëntatie	72
Basisprincipes.....	72
Overerving.....	76
Statische methodes en attributen.....	80
Abstracte klassen en functies.....	83
Interfaces	87
Hoofdstuk 9 Databanktoegang	91
Een korte inleiding.....	91
Verbinding maken.....	92
Gegevens ophalen	93
Gegevens toevoegen	95
Gegevens verwijderen	97
Gegevens bijwerken	102
Hoofdstuk 10 Meerlagenarchitectuur.....	111
Een sprong in het diepe.....	111
Het Model-View-Controller design pattern	115
Uitbreiding met een gegevensbank.....	117
Van nul... ..	119
Gegevens bundelen.....	137
Foutafhandeling.....	140
Namespaces	149
Autoloading.....	152
Hoofdstuk 11 Templating.....	158
De tegemoetkoming.....	158
Hoe werkt het?	160
Meer mogelijkheden.....	162

Appendix A Installatie.....	164
-----------------------------	-----

Een woordje vooraf...

Over deze cursus

Voor deze handleiding ter ondersteuning van de module PHP in het traject PHP ontwikkelaar is gekozen voor een inductieve methodiek. Dat betekent dat er reeds van bij het begin begonnen wordt met praktijkvoorbeelden, die de leidraad vormen tot het begrijpen van de theorie. Deze cursus werd volledig geschreven met het oog op zelfstudie.

De cursus bestaat uit drie delen.

Het eerste deel handelt over algemene aspecten van PHP. Je leert onmiddellijk hoe je op een relatief eenvoudige manier PHP-programma's in elkaar knutselt. Wanneer je deel 1 doorgenomen hebt, ben je in staat om de meest gangbare problemen op te lossen in PHP, zij het weliswaar zonder databanktoegang of geavanceerdere objectgeoriënteerde technieken.

Het tweede deel behandelt op zijn beurt deze objectgeoriënteerde begrippen. Hoewel van bij het begin reeds zoveel mogelijk wordt aangemoedigd om objectgeoriënteerd te werken, komt dit punt pas echt geaccentueerd in het tweede deel.

In het derde deel wordt onder meer ingegaan op het aanspreken van relationele gegevensbanken m.b.v. PHP. Op deze manier kan je gegevens uit databanktabellen gebruiken in je applicatie. Verder wordt uitgelegd hoe je een webapplicatie volgens een meerlagenarchitectuur ontwikkelt, en hoe je deze robuust en foutbestendig maakt.

Een aantal oefeningen breiden oplossingen van vorige oefeningen uit. Het zal dus soms noodzakelijk zijn dat je een bepaalde oefening oplost vooraleer je een andere kunt maken. Dit wordt steeds uitdrukkelijk vermeld.

Elke oefening is gemarkeerd met een aantal sterren die de moeilijkheidsgraad van de oefening aanduiden.



Gemakkelijk




Normaal



Moeilijk

Conventies

Doorheen de cursus gelden volgende afspraken:

- Klassenamen, namen van variabelen, objecten, en andere korte stukken code worden in een monotype font geschreven, bvb `$reken` is een object van de klasse `Rekenmachine`.
- Grotere stukken code worden omkaderd met een grijze achtergrond.
- Bestandsnamen en namen van mappen worden in *schuinschrift* gezet.
- Het teken  wordt gebruikt om aan te duiden dat de volgende regel aan de huidige geplakt dient te worden. Het is soms nodig lange statements in een aantal lijnen te splitsen.
- Soms wordt verwezen naar Internetadressen. URL's worden schuin gedrukt en met stippellijn onderlijnd, zoals *<http://www.php.net>*.
- Bij elke oefening hoort een voorbeeldoplossing. Deze zijn niet opgenomen in de cursus, maar zijn online opvraagbaar. Om een voorbeeldoplossing te zien te krijgen surf je naar *<http://www.vdabantwerpen.be/php>* en vul je de zescijferige code in die je bij de oefening in kwestie terugvindt. Soms zijn ook extra tips terug te vinden; deze werken op dezelfde manier. Slaag je er niet in een oefening op te lossen, bekijk dan eerst deze tips, en probeer het nogmaals.

Voorkennis

Voor het succesvol doornemen van deze cursus dien je over een basiskennis HTML te beschikken. Tevens moet je de module Programmatie logica afgelegd hebben. We komen immers niet terug op algemene basisprincipes van programmeren of hoe programma's werken. We behandelen uiteraard wel de typische syntaxregels van PHP (hoe maak ik een lus, hoe maak ik een voorwaardelijke uitdrukking,...).

Ontwikkelomgeving

Vanzelfsprekend heb je nood aan een ontwikkelomgeving, de nodige serversoftware, een browser en een editor. Voor het klaarstomen van een geschikte omgeving verwijzen we naar Appendix A achteraan deze handleiding.

Vermits PHP-code gewoon als tekstbestand wordt geschreven kan je voor het coderen van je applicaties beroep doen op een editor naar eigen keuze. Een kleine, zeer lightweight en snelle editor zonder al teveel "toeters en bellen" is de Crimson Editor/Emerald Editor. Je kan deze gratis downloaden op <http://sourceforge.net/projects/emeraldeditor/files/>.

Andere voorbeelden van editors zijn Notepad++ (<http://notepad-plus.sourceforge.net/nl/site.htm>) en PHPEdit (<http://www.phpedit.com/en>).

Voor projecten van grotere omvang (vooral wanneer we vanaf hoofdstuk 10 MVC-toepassingen begint uit te werken) kan je Netbeans (<http://netbeans.org/downloads/>) gebruiken. Deze gratis editor kent enkele zeer nuttige snelkoppelingen en plugins.

Vermits PHP een serverside programmeertaal is maakt het niet uit welke browser je gebruikt.

Extra hulp?

De cursus behandelt de belangrijkste, maar lang niet alle aspecten die de programmeertaal PHP rijk is. Je belangrijkste externe EHBO-instrument (Eerste Hulp Bij Ontwikkeling) wordt de officiële website van PHP zelf:

<http://www.php.net>

Op

<http://be.php.net/manual/en/funcref.php>

vind je de referentie van alle in PHP beschikbare functies.

DEEL

1

Hoofdstuk 1

Inleiding

In dit hoofdstuk:

- ✓ Een eerste kennismaking met PHP-code
- ✓ Enkele algemene principes van programmeren in PHP leren kennen

Een eerste blik

PHP is een server-side scripttaal waarmee snel en efficiënt dynamische webapplicaties kunnen gebouwd worden. Zowel syntactisch als semantisch is het gebaseerd op de programmeertaal C, hoewel er tevens elementen van Perl, Java, VB,... in terug te vinden zijn. We steken van wal met een eenvoudig voorbeeld. Bestudeer onderstaande code aandachtig.

```
<?php
class GreetingGenerator {
    public function getGreeting() {
        return "Hello world!";
    }
}
?>
<!DOCTYPE HTML>
<html>
<head>
    <meta charset=utf-8>
    <title>Hello world</title>
</head>
<body>
    <h1>
        <?php
        $gg = new GreetingGenerator();
        print($gg->getGreeting());
        ?>
    </h1>
</body>
</html>
```

Tik deze code in, sla het bestand op als *helloworld.php* en bekijk de uitvoer, alsook de broncode van de pagina die getoond wordt (in de browser – rechtermuisknopklik op de pagina, kies "Paginabron bekijken" uit het snelmenu).

We gaan op dit ogenblik nog niet al te diep in op de exacte werking van dit voorbeeld, maar willen wel een paar typische kenmerken overlopen.

Merk op dat je zowel PHP-code als HTML-code terugvindt in één pagina. Wanneer je alles wat HTML-code is aanduidt met een markeerstift valt duidelijk op dat PHP-code wordt begrensd door een speciale begin- en eindtag: respectievelijk `<?php` en `?>`.

In één PHP-bestand kunnen zich meerdere stukken PHP-code bevinden, zolang ze allemaal netjes afgesloten worden met deze speciale tags.

Verder valt op dat PHP-instructies worden afgesloten met een punt-komma. Probeer uit het voorbeeld één van de puntkomma's weg te laten en kijk wat er gebeurt als je dit probeert uit te voeren.

Duidelijk is ook dat het woord `class` gebruikt wordt om een klasse aan te duiden. Zoals je je misschien nog kan herinneren uit de cursus Objectgeoriënteerde Principes is een klasse een soort blauwdruk/sjabloon die de eigenschappen en het gedrag van een op zich staande entiteit beschrijft. De entiteit in dit voorbeeld is een `GreetingGenerator` (mocht je dit letterlijk willen vertalen: iets dat begroetingen "produceert"). Wat verderop in de PHP-code wordt een boodschap gestuurd naar deze entiteit:

```
$gg->getGreeting()
```

Dit resulteert in een begroeting die we vervolgens op het scherm afdrukken: "Hello World".

Als je de broncode van de uitgevoerde pagina in je browser bekijkt, is er geen spoor van PHP terug te vinden:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello world!</h1>
  </body>
</html>
```

Om te onthouden:

- ✓ PHP-code kan gewoon tussen HTML-code geschreven worden.
- ✓ PHP-code wordt afgesloten met de speciale `<?php` en `?>` tags, en op die manier gescheiden van wat HTML-code is.
- ✓ PHP ondersteunt het schrijven van programma's op een object-georiënteerde manier.

- ✓ PHP gedraagt zich transparant naar de eindgebruiker (= de bezoeker van de pagina) toe. Deze ziet enkel het resultaat van de door PHP gegenereerde uitvoer (in ons voorbeeld de begroeting), alsook de "gewone" HTML die in de pagina werd neergeschreven.

In sommige voorbeelden op het Internet wordt de PHP-code begrensd met `<? en ?>` i.p.v. `<?php en ?>`. Soms worden zelfs de oude ASP-tags `<% en %>` gebruikt. Welke moet ik gebruiken?

De tags die afwijken van de standaard `<?php en ?>` zijn slechts bruikbaar wanneer je bepaalde instellingen in het configuratiebestand `php.ini` wijzigt. Je gebruikt het best de tags `<?php en ?>`. Op die manier voorkom je dat je applicatie dienst weigert wanneer je ze plaatst op een server die toevallig deze tags niet ondersteunt, met alle gevolgen vandien...

Dynamische inhoud toevoegen

Bovenstaand voorbeeld toont bij elke uitvoer exact dezelfde inhoud: een pagina met daarop de begroeting "Hello world!". Dit hadden we evengoed kunnen bereiken zonder PHP, door enkel gebruik te maken van HTML-tags, en wel op de volgende manier:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Hello world!</h1>
  </body>
</html>
```

Het nut van PHP ligt in het feit dat we het gedrag van een webpagina (hier is het gedrag: het afdrukken van een boodschap) kunnen laten afhangen van programmacode.

We illustreren dit a.d.h.v. een tweede voorbeeld. We vormen de GreetingGenerator om zodat deze een datum en tijd toont.

```
...
class GreetingGenerator {
  public function getGreeting() {
    return "10/04/2012 - 16:19:48";
  }
}
```

...

Voer het programma uit. Uitvoer op het scherm:

10/04/2012 - 16:19:48

Merk op dat we als datum elke gewenste tekenreeks konden invullen. Vervang achtereenvolgens door andere data/tijden en voer telkens uit. Je ziet dat de datum/tijd die wordt afgedrukt totaal geen verband houdt met de werkelijkheid.

Wijzig opnieuw je code in onderstaand fragment:

```
...
class GreetingGenerator {
    public function getGreeting() {
        return date("d/m/Y - H:i:s");
    }
}
...
```

Wanneer je de nieuwe code uitvoert merk je dat de datum/tijd die verschijnt automatisch aangepast wordt aan de systeemtijd.

Noot: Krijg je een waarschuwing te zien wanneer je de pagina uitvoert, dan moet je wellicht je tijdzone in het PHP configuratiebestand instellen. Open daartoe het bestand `php.ini` dat zich in de submap `php` van je XAMPP-installatiemap bevindt met een editor naar keuze, en zoek naar `date.timezone`. Wijzig deze regel in `date.timezone = Europe/Brussels` (zorg ervoor dat de puntkomma aan het begin verdwijnt!).

De exacte werking van deze code is in dit stadium absoluut nog niet van belang.

We ronden af met een klein experimentje: Pas de systeemtijd van je toestel aan en voer het programma opnieuw uit. De inhoud van de pagina verandert telkens, zonder dat je één regel code aan je programma wijzigt. Vergeet niet om na je experiment de systeemtijd opnieuw te corrigeren.

Je merkt dat de actuele tijd op de pagina niet wijzigt, zolang je de pagina niet vernieuwt. De programmacode wordt pas uitgevoerd op het moment dat je de pagina opnieuw bezoekt.

Om te onthouden:

- ✓ PHP kan pagina's met dynamische inhoud genereren. Hoewel de code van je programma niet wijzigt, is het toch mogelijk dat je bij elk bezoek van de pagina iets anders te zien krijgt.
- ✓ Dynamische inhoud kan enkel wijzigen wanneer je de pagina opnieuw laadt (d.i. het opnieuw "bezoeken" van de pagina m.b.v. de knop "Vernieuwen" in je browser).

Presentatie versus logica

Bestudeer onderstaand voorbeeld aandachtig en bekijk de uitvoer.

```
<?php
class Rekenmachine {
    public function getKwadraat($getal) {
        $kwad = $getal * $getal;
        return $kwad;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Mijn rekenmachine</title>
    </head>
    <body>
        <h1>
            <?php
                $reken = new Rekenmachine();
                print($reken->getKwadraat(5));
            ?>
        </h1>
    </body>
</html>
```

Leg dit voorbeeld naast de voorgaande voorbeelden en zoek de overeenkomsten.

Telkens wordt gebruik gemaakt van een klasse (hier **Rekenmachine**, voordien **GreetingGenerator**) om een tekst/getal/... op te halen, en dit vervolgens tussen te voegen in HTML-code.

De regel

```
print($reken->getKwadraat(5));
```

betekent zoveel als: stuur de boodschap **getKwadraat**, met als parameter 5 naar het object met de naam **\$reken**, en druk (**print**) het resultaat af. Merk dus op dat we het berekenen van het kwadraat van 5 niet overlaten aan de

code die zich tussen de HTML-code bevindt, maar deze taak overlaten aan een speciaal daarvoor ontworpen klasse: **Rekenmachine**.

We maken dus een onderscheid tussen wat programmatielogica is enerzijds (hier: het berekenen van een kwadraat) en wat de presentatie aangaat anderzijds (hier: het afdrukken van het berekende kwadraat in een groot lettertype, d.m.v. de **<h1>** tags).

We breiden onze **Rekenmachine** uit met een somberekening.

Tekst
voorafgegaan door
twee schuine
streepjes // stelt
commentaar voor,
en wordt niet
uitgevoerd

```
<?php
class Rekenmachine {
    // Berekent het kwadraat van een meegegeven getal
    public function getKwadraat($getal) {
        $kwad = $getal * $getal;
        return $kwad;
    }

    /*
    Berekent de som van twee meegegeven getallen
    Dit is een tweede zelfgeschreven functie
    */
    public function getSom($getal1, $getal2) {
        $resultaat = $getal1 + $getal2;
        return $resultaat;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Mijn rekenmachine</title>
    </head>
    <body>
        <h1>
            <?php
            $reken = new Rekenmachine();
            print($reken->getKwadraat(5));
            ?>
        </h1>
        <h1>
            <?php
            print($reken->getSom(65, 8));
            ?>
        </h1>
    </body>
</html>
```

Commentaar kan je ook op
over meerdere lijnen spreiden.
Het commentaar begint met
/* en eindigt met */

Voer deze code uit en bekijk het resultaat. In een eerste kop wordt het kwadraat van 5, en in een volgende kop de som van 65 en 8 getoond.

Wat is er veranderd in de programmatielogica? We hebben een extra functie toegevoegd aan de klasse, waardoor deze laatste nu ook in staat is een som te berekenen.

Wat is er veranderd in de presentatie? We hebben de extra functie-aanroep toegevoegd, met als parameters de getallen 65 en 8.

Trek nu met een lat een horizontale lijn tussen deze twee regels:



```
?>  
<!DOCTYPE HTML>
```

Deze lijn is de scheiding tussen de programmatielogica en de presentatie.

De programmatielogica weet hoe iets berekend of bepaald moet worden, maar weet niet hoe het resultaat getoond zal worden aan de bezoeker. Deze "laag" maakt zich enkel zorgen over de correcte werking van de berekening van de invoerparameters en wat als resultaat gegeven moet worden.

De presentatie weet wat er wordt getoond en hoe iets wordt getoond aan de bezoeker (in een kop d.m.v. de `<h1>` tags in dit voorbeeld), maar niet hoe de berekening gebeurt.

Presentatie en logica hebben dus weliswaar een onderlinge afspraak met elkaar, maar weten van elkaar niet exact hoe ze werken. En dit is ook helemaal niet nodig, noch gewenst!

Niet gewenst? Welk nadeel is er verbonden aan het mengen van presentatie en logica?



De persoon die zich bezig houdt met de presentatie is dikwijls een designer, en geen PHP programmeur. Daarom is het van belang zo weinig mogelijk PHP-code te mengen in de HTML.



Tijd voor enkele kleine oefeningen.

Oefening 1.1: ★

Pas het voorbeeld aan zodat ook de som van de getallen 34 en 55 getoond wordt. Vraag jezelf af welke laag/lagen (presentatie of logica of beiden) je hebt aangepast.

→ Oplossing: 544125

Oefening 1.2: ★★

Zorg ervoor dat onze klasse ook in staat is om een vermenigvuldiging van twee getallen uit te voeren (de operator voor een optelling is een plusteken (+))

de operator voor een vermenigvuldiging is een sterretje (*)). Vraag jezelf af welke laag/lagen je hebt aangepast.

→ *Oplossing: 441544*

Oefening 1.3: ★★

Wijzig je vorige oplossing zodat de resultaten van je berekeningen in het rood worden getoond i.p.v. in het zwart. Maak hiervoor van de volgende CSS-code gebruik:

```
<style type="text/css">
  h1 { color: red; }
</style>
```

Vraag jezelf af welke laag/lagen je hebt aangepast.

→ *Oplossing: 447122*

Om te onthouden:

- ✓ We proberen de verantwoordelijkheden van een applicatie te scheiden in een programmatielogicadeel en een presentatiedeel. Elk deeltje heeft zijn eigen verantwoordelijkheid.
- ✓ Programmatielogica maakt zich zorgen over de eigenlijke werking van de applicatie. Dit kan berekeningen maken zijn, maar ook gegevens uit een databank ophalen, de inhoud van tekstbestanden wissen, wachtwoorden controleren, enz...
- ✓ De presentatie bepaalt wat er wordt getoond en op welke manier (in tabelvorm, in het rood, onderlijnd, enz...)

Hoofdstuk 2

De code uitgespit

In dit hoofdstuk:

- ✓ Wat zijn variabelen en hoe werken ze?
- ✓ Wat zijn de eigenschappen van expressies.
- ✓ Hoe werken controlestructuren?
- ✓ Hoe werken lusconstructies?

Variabelen

In de cursus Programmatie logica werd het concept van variabelen reeds uitvoerig uit de doeken gedaan. In een notepad zijn variabelen verwijzingen naar geheugenplaatsen waar men "iets" kan in opslaan. Later kan men dit "iets" terug opvragen, door gebruik te maken van de variabelenaam.

Een voorbeeld ter illustratie:

```
<?php
class Book {
    public function getTitle() {
        $title = "Handleiding HTML";
        return $title;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Boeken</title>
    </head>
    <body>
        <h1>
            <?php
            $boek = new Book();
            print($boek->getTitle());
            ?>
        </h1>
    </body>
</html>
```

Bestudeer en redeneer. Hoeveel variabelen werden in dit voorbeeld gebruikt? Waaraan kan je namen van variabelen herkennen?

Er wordt gebruik gemaakt van twee variabelen. In de klasse bevindt zich een variabele **\$title**, waar een tekst wordt in opgeslagen. In de presentatie is een variabele **\$boek** terug te vinden, dat een object bevat van de klasse **Book**.

Het mag duidelijk zijn dat variabelenamen te herkennen zijn aan het dollarteken dat de naam vooraf gaat.

What's in a name?

De naam van een variabele is vrij te kiezen, maar moet beginnen met een letter of underscore (_), gevolgd door een zelf te bepalen aantal letters, cijfers of underscores.

Kies bij voorkeur een naam die iets zegt over wat je er in gaat opslaan. Dit komt de leesbaarheid en onderhoudbaarheid enorm ten goede. Een naam als **\$inhoudKast** vertelt je exact wat je wil weten. Een programma met variabelenamen als **\$d76xKM** is - hoewel perfect technisch mogelijk - qua leesbaarheid ten dode opgeschreven.

En tot slot, niet te vergeten: variabelenamen zijn hoofdlettergevoelig!

Datatypes

In tegenstelling tot strikt getypeerde programmeertalen zoals Java en C++, wordt in PHP bij het eerste gebruik van variabelen niet vermeld wat voor soort data er in zal komen te staan. Dat betekent niet dat PHP geen datatypes kent. Het betekent enkel dat PHP een "weakly typed language" is. Dit is een dure benaming voor: "niet alleen de inhoud van een variabele kan wijzigen tijdens de uitvoer van het programma, maar ook het type".

Enkele voorbeelden van datatypes zijn gehele getallen (integer), kommagetallen (floating point), tekst (string), objecten, waar of onwaar (boolean), etc... Het lijstje gaat verder, maar is op dit moment niet van belang.

Enkele voorbeelden:

```
$budget = 18000;
```

slaat het gehele getal 18000 op in de variabele **\$budget**.

```
$resultaat = 22.5;
```

slaat het kommagetal 22.5 op in de variabele **\$resultaat**. Let op het gebruik van de punt om de decimalen aan te duiden!

```
$boek = new Boek();
```

slaat een verwijzing naar een boek-object op in de variabele **\$boek**.

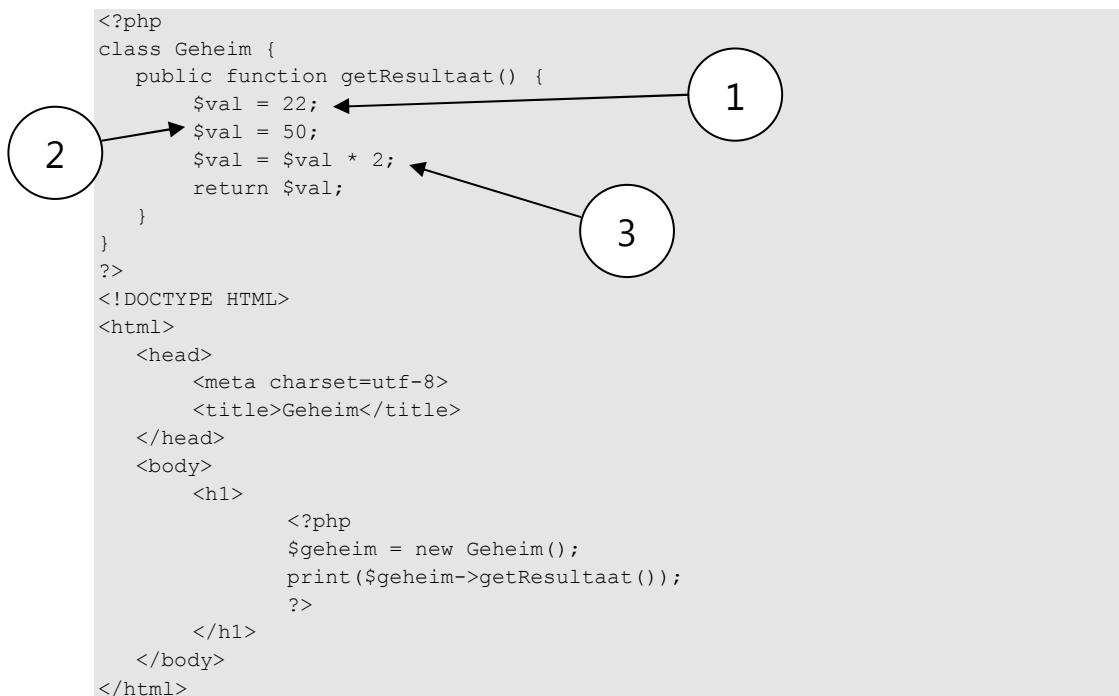
```
$geslaagd = true;
```

duidt aan dat de variabele **\$geslaagd** WAAR is. Voor ONWAAR schrijf je "false". Uiteraard zijn slechts twee waarden mogelijk.

Voor het toekennen van een waarde aan een variabele wordt een enkel is-gelijk-aan-teken gebruikt.

Aan het werk

Bekijk onderstaande code. Probeer vòòr het uitvoeren te voorspellen wat op het scherm tevoorschijn zal komen.



```
<?php
class Geheim {
    public function getResultaat() {
        $val = 22;
        $val = 50;
        $val = $val * 2;
        return $val;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Geheim</title>
    </head>
    <body>
        <h1>
            <?php
            $geheim = new Geheim();
            print($geheim->getResultaat());
            ?>
        </h1>
    </body>
</html>
```

Voer uit, en controleer je oplossing.

In essentie:

1. Het getal 22 wordt opgeslagen in de variabele **\$val**.
2. Het getal 50 wordt opgeslagen in de variabele **\$val**. De voorgaande waarde (22) wordt overschreven en is verdwenen.
3. Het resultaat van de berekening rechts van het is-gelijk-aan-teken (de huidige waarde van **\$val** vermenigvuldigd met 2, dus 100) wordt opgeslagen in de variabele **\$val**. De oude waarde van **\$val** (50) wordt overschreven.

Zodoende wordt het getal 100 afgedrukt.

kan ik in de print-opdracht niet gewoon de inhoud van \$val afdrukken? Deze variabele bevat toch het resultaat van de berekening?

De variabele \$val bestaat niet meer op het ogenblik van de print-opdracht. Ze werd gedefinieerd binnen de functie getResultaat() en is daarbuiten niet zichtbaar.

Is dat wat men omschrijft als de "scope" van variabelen?

Inderdaad. In dit geval spreekt men van een variabele met functiescope.

Oefening 2.1: ☆☆

Bekijk onderstaande code, maar **voer ze niet uit**:

```
<?php
$getal1 = 20;
$getal2 = 30;
...
print("Getal 1: " . $getal1 . "<br>");
print("Getal 2: " . $getal2);
?>
```

Schrijf op papier neer welke regel (of regels) code je toevoegt op de plaats van de drie puntjes om de waarden van de variabelen `$getal1` en `$getal2` om te wisselen. De bedoeling is dat men uiteindelijk te zien krijgt:

```
Getal 1: 30
Getal 2: 20
```

Controleer a.d.h.v. de voorbeeldoplossing.

→ *Oplossing: 172699*

Oefening 2.2: ☆☆

Bekijk onderstaande code, maar **voer ze niet uit**:

```
<?php
$_waarde = 10;
$waarde = $_waarde - 50;
$2eWaarde = 30;
print($waarde + 1);
?>
```

Deze code zou niet uitgevoerd kunnen worden. Waarom niet? Controleer je antwoord a.d.h.v. de voorbeeldoplossing.

→ *Oplossing: 556123*

Oefening 2.3: ★

Bestudeer onderstaande code aandachtig, en voer ze uit.

```
<?php
class Geheim {
    public function getResultaat() {
        $mijnGetal = 10;
        $mijnGetal = $mijnGetal * $mijnGetal;
        return $mijnGetal;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Oefening op variabelen</title>
    </head>
    <body>
        <h1>
            <?php
            $geheim = new Geheim();
            print($geheim->getResultaat());
            ?>
        </h1>
    </body>
</html>
```

Hoe verklaar je het resultaat?

→ *Oplossing: 742698*

Om te onthouden:

- ✓ Een variabele is een plaats in het geheugen waar een waarde in opgeslagen kan worden.
- ✓ Namen van variabelen zijn gebonden aan regels en hoofdlettergevoelig
- ✓ Verschillende datatypes (getallen, tekst,...) kunnen opgeslagen worden, maar het type wordt pas bepaald bij de toekenning van een waarde aan de variabele.
- ✓ De toekenning van een waarde aan een variabele gebeurt met een enkel is-gelijk-aan-teken.

Is het mogelijk om handmatig een variabele van het ene type naar het andere te converteren?

Ja. Dit kan door voor de naam van de variabele of expressie het type van het gewenste resultaat te vermelden tussen haken, bvb `$var = (integer)3.567`

Expressies

We hebben reeds meerdere keren gewerkt met berekeningen. Berekeningen zijn een deelverzameling van een grotere verzameling "expressies". In dit onderdeel willen we expressies eens nader bekijken, en een aantal voorbeelden aanhalen. Het is zeker niet de bedoeling een volledige opsomming te maken van alle soorten expressies. We willen je op dit ogenblik enkel laten kennismaken met deze constructies.

Een eenvoudig voorbeeld van een expressie:

```
5 + 2
```

Op papier is een expressie een bewerking met ten minste één operand en ten minste één operator. In bovenstaand voorbeeld zijn 5 en 2 de operanden, en is "+" de operator. In dit geval is de expressie een **berekening**.

Een andere expressie:

```
$resultaat = 7
```

Hierbij zijn **\$resultaat** en **7** de operanden, en "=" de operator. Het getal 7 wordt ondergebracht in de variabele **\$resultaat**. Deze expressie is een **toekenning**.

Expressies kunnen perfect andere expressies bevatten. Bijvoorbeeld:

```
$resultaat = 5 + 2
```

We hebben hier te maken met twee expressies in één regel:

- Een berekeningsexpressie waarin 5 en 2 de operanden zijn, en + de operator.
- De toekenningsexpressie waarin \$resultaat de eerste operand en de berekeningsexpressie de tweede operand zijn, en = de operator is

Een ander voorbeeld van een expressie:

```
$aantalPersonen == 20
```

We hebben hier te maken met een **vergelijking**. De operanden zijn `$aantalPersonen` en het getal 20. De operator is `==` (twee is-gelijk-aan-tekens. Denk aan het verschil met een toekenning!).

De waarde van een vergelijkingexpressie is steeds van het type boolean (TRUE of FALSE). In bovenstaand voorbeeld is het resultaat TRUE als de variabele `$aantalPersonen` inderdaad het getal 20 bevat.

Soms gebruikt men bij een rechtstreekse vergelijking DRIE is-gelijk-aan-tekens (`===`). Wat is het verschil met twee tekens?

Met twee is-gelijk-aan-tekens vergelijk je enkel de waarden van de operanden, bvb (`0 == 0.0`), terwijl je met drie tekens ook het type vergelijkt!

Met booleaanse waarden kunnen tevens bewerkingen uitgevoerd worden. We spreken dan over **logische** bewerkingen.

Een voorbeeld ter illustratie:

```
$geslaagd && $afgestudeerd
```

De AND-operator (`&&`) is een voorbeeld van een logische operator. De operanden zijn steeds booleaanse waarden.

Zoals reeds vermeld is het perfect mogelijk expressies te gaan nesten door gebruik te maken van haakjes.

```
$geslaagd || ($aantalOnvoldoendes < 2)
```

Hierbij is (`$aantalOnvoldoendes < 2`) een expressie die resulteert in een boolean: "bevat de variabele `$aantalOnvoldoendes` een waarde kleiner dan 2?". Een tweede expressie is een logische bewerking met `$geslaagd` als eerste en het resultaat van de eerste expressie als tweede operand. De operator is hier een logische OR (twee verticale streepjes `||`)

kan ik haakjes binnen andere haakjes gebruiken?

Zeker. De volgorde van bewerkingen werkt dan van binnen naar buiten. Eerst worden de bewerkingen tussen de binnenste haakjes uitgevoerd, daarna deze daarbuiten.

Er bestaan nog meer soorten expressies, waaronder het zgn. **pre- en postincrement en decrement**.

Bijvoorbeeld, het postincrement:

```
$getal++
```

betekent zoveel als:

```
$getal = $getal + 1
```

Postdecrement wordt dan weer gevormd met het minteken.

Verder kent PHP ook de volgende constructie:

```
$getal += 8;
```

wat gelijk staat aan:

```
$getal = $getal + 8;
```

Deze constructie werkt overigens ook met de operatoren -, *, / en %.

Oefening 2.4: ★★

Zoek op de officiële PHP website de overzichtjes op van alle door PHP gekende operatoren. Bestudeer de verschillende mogelijkheden aandachtig.

→ *Oplossing: 196334*

Om te besluiten volgen nog enkele oefeningen. Probeer je bij het oplossen hierbij te concentreren op de expressies. Andere (eventueel onbekende) constructies zijn nog niet aan de orde.

Oefening 2.5: ☆☆

Bestudeer en voer uit:

```
<?php
class Vergelijking {
    public function getSomIsStriktPositief($getal1, $getal2) {
        $antw = (($getal1 + $getal2) > 0);
        if ($antw == true) return "JA";
        else return "NEEN";
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Vergelijking</title>
    </head>
    <body>

        <h1>
            <?php
            $vgl = new Vergelijking();
            print($vgl->getSomIsStriktPositief(10, -9));
            ?>
        </h1>
    </body>
</html>
```

Voeg nu een functie **getSomIsStriktNegatief** toe, met drie parameters: **\$getal1**, **\$getal2** en **\$getal3**, en test deze uit. Bedoeling is dat bij het uitvoeren "JA" verschijnt op het scherm wanneer de som van de drie getallen strikt negatief is (kleiner dan nul), en "NEEN" in het andere geval.

→ *Oplissing: 842662*

Oefening 2.6: ☆

Bestudeer en beredeneer volgende code, maar voer ze niet uit.

```
<?php
$aantal = 2;
$gratisThuisbezorging = false;
$prijs = 6.5;

if ($aantal * $prijs > 10.0) $gratisThuisbezorging = true;
if ($gratisThuisbezorging) print("Uw pizza wordt gratis thuisbezorgd.");
else print("Thuislevering kost 1 euro.");
?>
```

Duidt de expressies aan. Schrijf neer wat je denkt dat de bedoeling is van deze code.

→ *Oplissing: 717225*

Om te onthouden:

- ✓ Een expressie is een bewerking waarbij minstens één operand en één operator gebruikt wordt.
- ✓ Expressies zijn er in verschillende soorten. Zo bestaan er expressies met rekenkundige operatoren, toekenningsoperatoren, vergelijkinsoperatoren, logische operatoren, enz...

Controlestructuren

Een controlestructuur is een constructie die in staat is de normale doorloop van een programma (instructie na instructie na instructie) te wijzigen in functie van een voorwaarde die, na evaluatie, leidt tot TRUE of FALSE.

De if-constructie

We nemen er opnieuw de code van voorgaande oefening bij (in lichtjes gewijzigde vorm):

```
<?php
$aantal = 2;
$gratisThuisbezorging = false;
$prijs = 6.5;

if ($aantal * $prijs > 10.0) {
    $gratisThuisbezorging = true;
}
if ($gratisThuisbezorging) {
    print("Uw pizza wordt gratis thuisbezorgd.");
} else {
    print("Thuislevering kost 1 euro.");
}
?>
```

en bestuderen eerst in het bijzonder deze regel:

```
if ($aantal * $prijs > 10.0) { $gratisThuisbezorging = true; }
```

De instructie **\$gratisThuisbezorging = true;** wordt enkel uitgevoerd als de expressie tussen de haakjes (**\$aantal * \$prijs > 10.0**) resulteert in een booleaanse waarde TRUE. Of anders gezegd: als het aantal vermenigvuldigd met de prijs een product geeft dat groter is dan 10, dan krijgt de klant een gratis thuislevering. In het andere geval wordt de toekenning niet uitgevoerd, en wordt onmiddellijk verder gegaan met:

```
if ($gratisThuisbezorging) print...
```

De else-constructie

Een if-constructie kan optioneel uitgebreid worden met een else-constructie, zoals in het voorbeeld:

```
if ($gratisThuisbezorging) {  
    print("Uw pizza wordt gratis thuisbezorgd.");  
} else {  
    print("Thuislevering kost 1 euro.");  
}
```

Indien de expressie tussen de haakjes TRUE is wordt de eerste printopdracht uitgevoerd. In het andere geval wordt de tweede uitgevoerd.

Oefening 2.7: ☆☆☆

Gegeven volgende code:

```
<?php  
class Oefening {  
    public function getAnalyse($getal1, $getal2) {  
        ...  
    }  
}  
?>  
<!DOCTYPE HTML>  
<html>  
    <head>  
        <meta charset=utf-8>  
        <title>Analyse van getallen</title>  
    </head>  
    <body>  
        <h1>  
            <?php  
                $oef = new Oefening();  
                print($oef->getAnalyse(7, 2));  
            ?>  
        </h1>  
    </body>  
</html>
```

Vul op de plaats van de drie puntjes code aan waardoor bij uitvoer van het programma:

- De tekst "Het eerste getal is groter dan het tweede" verschijnt als bijvoorbeeld 7 en 2 als getallen gebruikt worden.
- De tekst "Het eerste getal is niet groter dan het tweede" verschijnt als bijvoorbeeld 2 en 7 als getallen gebruikt worden.
- De tekst "Het eerste getal is niet groter dan het tweede" verschijnt als bijvoorbeeld 7 en 7 als getallen gebruikt worden.

Vergeet je oplossing ook niet uit te testen met enkele andere willekeurige getallen.

De elseif-constructie

De if-constructie en else-constructie kunnen indien gewenst samen met een elseif-gebruikt worden.

Bekijk volgend voorbeeld:

```
$seizoenNr = 2;  
if ($seizoenNr == 1) {  
    $seizoenNaam = "Lente";  
} elseif ($seizoenNr == 2) {  
    $seizoenNaam = "Zomer";  
} elseif ($seizoenNr == 3) {  
    $seizoenNaam = "Herfst";  
} else {  
    $seizoenNaam = "Winter";  
}
```

Hoe zit deze constructie in elkaar? Indien het **\$seizoenNr** de waarde 1 (getal) bevat, dan is de initiële expressie tussen haakjes TRUE, en dan wordt **\$seizoenNaam** ingesteld op "Lente" (tekst). Als dat niet het geval is, wordt een nieuwe expressie geanalyseerd, nl (**\$seizoenNr == 2**). Levert deze expressie TRUE op, dan wordt **\$seizoenNaam** ingesteld op "Zomer". Als dat ook niet het geval is, dan wordt nogmaals een nieuwe expressie geanalyseerd, nl (**\$seizoenNr == 3**). Is het resultaat van deze expressie TRUE, dan krijgt **\$seizoenNaam** de waarde "Herfst". Is tenslotte ook die expressie FALSE, dan wordt de tekst "Winter" in **\$seizoenNaam** gestoken.

Oefening 2.8: ★★

Pas je oplossing uit oefening 2.7 aan zodat:

- De tekst "Het eerste getal is groter dan het tweede" verschijnt als bijvoorbeeld 7 en 2 als getallen gebruikt worden.
- De tekst "Het eerste getal is kleiner dan het tweede" verschijnt als bijvoorbeeld 2 en 7 als getallen gebruikt worden.
- De tekst "Het eerste getal is gelijk aan het tweede" verschijnt als bijvoorbeeld 7 en 7 als getallen gebruikt worden.

Vergeet je oplossing ook niet uit te testen met enkele andere willekeurige getallen.

De switch-constructie

Wanneer je te maken hebt met een **if-elseif-else**-constructie die telkens de inhoud van eenzelfde variabele (of het resultaat van een expressie) controleert op gelijkheid met een vastgelegde waarde, dan wordt je code netter en aangenamer leesbaar door over te schakelen op een **switch**-constructie.

In zo'n geval schrijf je liever niet:

```
$browser = "Firefox";
if ($browser == "Internet Explorer") {
    print("Microsoft");
} elseif ($browser == "Firefox") {
    print("Mozilla");
} elseif ($browser == "Chrome") {
    print("Google");
} elseif ($browser == "Opera") {
    print("Opera Software");
} elseif ($browser == "Safari") {
    print("Apple");
} else {
    print("Een andere browser");
}
```

maar eerder:

```
$browser = "Firefox";
switch ($browser) {

    case "Internet Explorer":
        print("Microsoft");
        break;

    case "Firefox":
        print("Mozilla");
        break;

    case "Chrome":
        print("Google");
        break;

    case "Opera":
        print("Opera Software");
        break;

    case "Safari":
        print("Apple");
        break;

    default:
        print("Een andere browser");
        break;

}
```

De inhoud van **\$browser** wordt achtereenvolgens met de verschillende waarden vergeleken. Wanneer er een overeenkomst wordt gevonden, dan wordt het bijbehorende **case**-blok uitgevoerd. De **break**-instructie aan het

einde van elk blok zorgt ervoor dat na deze uitvoer de rest van de **switch**-constructie niet meer geanalyseerd wordt. Zorg er dus steeds voor dat dit **break**-sleutelwoord consequent aanwezig is in elk **case**-blok. Het **default**-blok werkt zoals de **else**-constructie, en zal dus enkel worden uitgevoerd als geen enkele "match" gevonden werd.

Oefening 2.9: ☆☆

Schrijf een programma waarin je een variabele `$getal` definieert met een numerieke waarde. Ligt het getal tussen 1 en 5 (grenzen inbegrepen), dan wordt het vol uitgeschreven ("Een", "Twee", enz...). Ligt het buiten deze grenzen dan wordt gewoon de getalwaarde getoond.

→ *Oplossing: 665899*

Om te onthouden:

- ✓ Om het gedrag van een programma te laten afhangen van een voorwaarde kan je een controlestructuur zoals de if-constructie gebruiken.
- ✓ De if-constructie kan uitgebreid worden met een else-constructie. Deze wordt uitgevoerd indien het resultaat van de expressie tussen haakjes FALSE is.
- ✓ Wil je de waarde van een variabele vergelijken met een lijst van mogelijkheden, stap dan over op een switch-constructie.

Lussen

Waar programmacode een bepaald of onbepaald aantal keren herhaald moet worden bieden lussen een uitkomst. Je hebt in de cursus Programmatie logica kennis gemaakt met lussen. Er bestaan verschillende varianten, en we overlopen even de belangrijkste.

De while-lus

```
$som = 0;
$basisgetal = 10;
while ($som < 100) {
    $som = $som + $basisgetal;
}
```

Bovenstaande code blijft een bepaald basisgetal bij zichzelf optellen tot de som 100 of groter wordt. De waarde van `$basisgetal` is variabel en vrij te

kiezen. We weten vooraf niet hoeveel keer de lus uitgevoerd zal worden. Tussen de haakjes staat de expressie waarvan het resultaat TRUE moet zijn opdat het blok tussen de accolades { en } uitgevoerd zou worden.

Belangrijk: de variabele die zich in de lusvoorwaarde bevindt (hier: `$som`) wordt beïnvloed tijdens de lus zelf. Dit is noodzakelijk om ervoor te zorgen dat de lus op zeker ogenblik zal stoppen en dus eindig is!

De for-lus

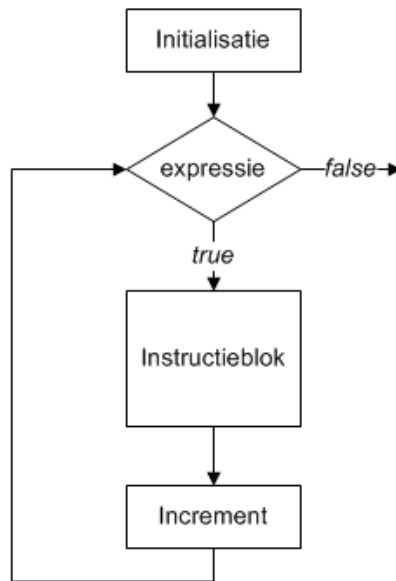
```
for ($teller = 1; $teller <= 100; $teller = $teller + 1) {  
    print($teller . "<br>");  
}
```

Dit stuk code drukt een oplopende teller af, startend bij 1 en eindigend bij 100 (het getal 100 wordt ook nog afgedrukt).

We weten exact hoeveel keer de lus zal worden uitgevoerd. De typische constructie van een for-lus bevat volgende onderdelen (van elkaar gescheiden door een puntkomma):

- Initialisatie-expressie: in ons voorbeeld is dit `$teller = 1`. Dit is de expressie die wordt uitgevoerd net vòòr de start van de lus (dus ook vòòr de voorwaarde-expressie geanalyseerd wordt)
- Voorwaarde-expressie: in ons voorbeeld `$teller <= 100`. De voorwaarde om de lus uit te voeren is dat de waarde in `$teller` kleiner dan of gelijk aan 100 is.
- Increment-expressie: hier `$teller = $teller + 1`. Deze expressie wordt uitgevoerd net vòòr er teruggekeerd wordt naar het begin van de for-lus, en dus vòòr de voorwaarde-expressie nogmaals geanalyseerd wordt.

Schematisch:



Merk op dat net zoals bij de while-lus het blok code dat herhaald moet worden omsloten wordt door accolades { en }.

- Belangrijk: de variabele die in de lushoofding gebruikt wordt en het aantal keren bepaalt dat de lus overlopen zal worden (hier **\$teller**), wordt niet beïnvloed in de lus zelf. De manier waarop deze variabele zal variëren tijdens de uitvoer wordt altijd bepaald in de lushoofding!

Oefening 2.10: ★

Schrijf een programma dat de getallen van 20 t.e.m. 50 op het scherm toont. Pas je oplossing daarna aan zodat er met stappen van 2 gewerkt wordt i.p.v. 1 (op het scherm verschijnt dus: 20 22 24 26 enz...

→ *Oplossing: 515663*

Oefening 2.11: ★★

Schrijf een programma dat de getallen van 1 t.e.m. een willekeurig gegenereerd getal tussen 100 en 200 uitschrijft.

→ *Tip: 651233*
→ *Oplossing: 326114*

Oefening 2.12: ★★

Schrijf een programma dat herhaaldelijke willekeurige getallen tussen 1 en 1000 op het scherm toont. Wanneer er een getal gegenereerd wordt dat groter is dan 600, stopt de uitvoer.

→ *Oplossing:* 547221

Oefening 2.13: ★★★

De Fibonaccireeks is een reeks getallen die begint met 0 en 1, en waarbij elk daaropvolgend getal de som is van de twee voorgaande. Dit resulteert in de volgende reeks:

```
0 1 1 2 3 5 8 13 21 34 ...
```

Schrijf een programma dat de opeenvolgende getallen uit de Fibonaccireeks afbeeldt, op zo'n manier dat het laatst uitgeschreven getal nog kleiner is dan 5000.

→ *Tip:* 793846

→ *Oplossing:* 344772

Om te onthouden:

- ✓ Om een bepaald blok code meerdere keren te laten uitvoeren kan gebruik gemaakt worden van een lusstructuur.
- ✓ Is het aantal keren dat een lus zal worden overlopen op voorhand niet gekend, gebruik dan een while-lus.
- ✓ Dient een lus daarentegen een exact aantal keren overlopen te worden, gebruik dan een for-lus.

Hoofdstuk 3

Gebruikersinvoer

In dit hoofdstuk:

- ✓ Gebruikersinvoer via de URL
- ✓ Gebruikersinvoer via een formulier
- ✓ Verschil tussen GET en POST en wanneer gebruik je welke methode

Via de URL met GET

Bestudeer en bekijk de uitvoer van onderstaand voorbeeld:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Gebruikersinvoer</title>
  </head>
  <body>
    <h1>
      Goeiemorgen,
      <?php
        print($_GET["naam"]);
      ?>
    </h1>
  </body>
</html>
```

Op het scherm verschijnt:

```
Goeiemorgen,
```

Nieuwe speler in het spel is hier de variabele `$_GET["naam"]`. De variabele `$_GET` (het is een variabele; merk het `$`-teken op) is een voorbeeld van een array. Wat arrays zijn en hoe ze werken is nog van geen belang. In het volgende hoofdstuk wordt dit item verder behandeld.

Voer nu voorgaande code opnieuw uit, maar voeg dit keer het volgende toe aan de URL in de adresbalk:

```
?naam=Paul
```

De volledige URL wordt dan iets in de aard van:

```
http://localhost/.../voorbeeldinvoer.php?naam=Paul
```

Bekijk de uitvoer:

```
Goeiemorgen, Paul
```

Vervang de naam Paul in de URL en bekijk het resultaat. Je merkt dat de inhoud van de variabele `$_GET["naam"]` overeenkomt met wat in de URL als waarde voor de parameter **naam** wordt ingegeven.

We breiden ons voorbeeld uit naar twee parameters:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Gebruikersinvoer</title>
  </head>
  <body>
    <h1>
      Goeiemorgen,
      <?php
      print($_GET["naam"]);
      ?>
      . Het is vandaag
      <?php
      print($_GET["dag"]);
      ?>
    </h1>
  </body>
</html>
```

Vul de URL verder aan met:

```
&dag=vrijdag
```

De volledige URL wordt dan iets in de aard van:

```
http://localhost/.../voorbeeldinvoer.php?naam=Paul&dag=vrijdag
```

Bekijk de uitvoer. De structuur van je URL is dus algemeen:

```
http://localhost/.../naamvanscript.php?varnaam1=varwaarde1&
varnaam2=varwaarde2&varnaam3=varwaarde3&...
```

Het onderdeel **varnaam1** is hierbij hetgeen overeenkomt met wat in de `$_GET`-variabele tussen aanhalingstekens staat, hier: `$_GET["varnaam1"]`. Het onderdeel **varwaarde1** is de inhoud van deze variabele.

Je bent nu in staat om gebruikersinvoer te vragen via de URL, waardoor de uitvoer en werking van het programma gewijzigd kunnen worden, zonder de code zelf te veranderen.

Oefening 3.1: ★★

Schrijf een programma dat een bezoeker in staat stelt zelf twee getallen te kiezen. Het programma schrijft de som uit van deze getallen.

Voorbeeld van de URL:

```
http://localhost/.../oefening31.php?getal1=25&getal2=72
```

→ Oplossing: 775954

Oefening 3.2: ★★

Werk voorgaande oefening verder uit zodat de bezoeker m.b.v. een derde parameter ook de bewerking kan kiezen, bijvoorbeeld:

```
http://localhost/.../oefening32.php?getal1=40&getal2=4&bewerking=4
```

Mogelijke waarden van de parameter zijn:

- 1 -> Optellen
- 2 -> Aftrekken
- 3 -> Vermenigvuldigen
- 4 -> Delen

Ter illustratie: de uitvoer van bovenstaande URL is

```
Resultaat: 10
```

→ Oplossing: 217449

Oefening 3.3: ★★★

Een programma voor een gokautomaat bedenkt een willekeurig getal tussen 1 en 10, waarbij de bezoeker op de URL een getal kan kiezen, bvb:

```
http://localhost/.../oefening33.php?mijngok=7
```

Het programma schrijft uit wat de gok van de bezoeker was, en welk getal de computer gekozen had. Bij elke uitvoer wordt dus een nieuw getal willekeurig gekozen.

Werk daarna je oplossing verder uit zodat ook wordt gezegd of er correct of foutief gegokt was.

→ Oplossing: 418974

Via een formulier met GET

We halen het eerste voorbeeld (*voorbeeldinvoer.php*) nog eens boven:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Gebruikersinvoer</title>
  </head>
  <body>
    <h1>
      Goeiemorgen,
      <?php
        print($_GET["naam"]);
      ?>
    </h1>
  </body>
</html>
```

Maak een tweede PHP-bestand aan en voorziet het van de nodige HTML-code, waarbij je tevens een formulier maakt:

```
<!DOCTYPE HTML>
<html>
  <head><title>Gebruikersinvoer</title></head>
  <body>
    <form action="voorbeeldinvoer.php" method="get">
      Vul je naam in:
      <input type="text" name="naam">
      <input type="submit" value="OK">
    </form>
  </body>
</html>
```

Sla dit bestand op als *voorbeeldinvoerform.php* en voer het uit.

Bestudeer het formulier aandachtig. Wat valt op:

- Het **action**-attribuut van de **form**-tag staat ingesteld op de bestandsnaam van ons voorbeeld (*voorbeeldinvoer.php*). Dit zal het bestand zijn dat ons formulier zal "verwerken".
- Het **method**-attribuut van de **form**-tag heeft de waarde "**get**". Door dit te doen worden de naam-waarde-paren in het formulier achteraan op de URL toegevoegd wanneer we het formulier versturen.
- Het **name**-attribuut van de eerste input-tag heeft de waarde "**naam**". Dit is de variabele die we in het bestand *voorbeeldinvoer.php* verwachten, nl: `$_GET["naam"]`.

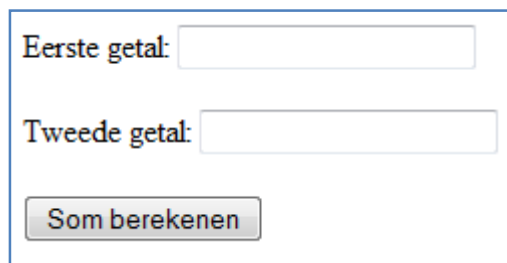
Bij uitvoer van het bestand *voorbeeldinvoerform.php* merk je dat de inhoud van het tekstveld wordt gebruikt om het programma in *voorbeeldinvoer.php* te laten werken.

Algemeen genomen kan je stellen dat wanneer je een formulierveld maakt met als naam **hiereenwillekeurigenaam** (als waarde van het **name**-attribuut van de overeenkomstige **input**-tag), de inhoud doorgegeven wordt aan de verwerkingsscript onder de vorm `$_GET["hiereenwillekeurigenaam"]`.

Oefening 3.4: ★★

Herneem oefening 3.1. Voeg een HTML-formulier toe, zodat de twee getallen die gesommeerd moeten worden, niet meer door de gebruiker op de URL moeten ingevuld worden, maar via een formulier kunnen doorgegeven worden.

Ter illustratie:



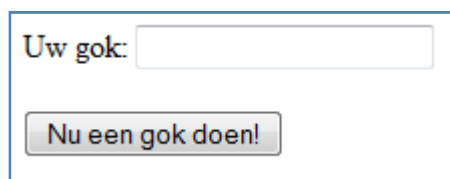
A screenshot of a web form with a blue border. It contains two text input fields. The first is labeled 'Eerste getal:' and the second is labeled 'Tweede getal:'. Below these fields is a button labeled 'Som berekenen'.

→ Oplossing: 715336

Oefening 3.5: ★★

Herneem oefening 3.3. Voeg een HTML-formulier toe, zodat de gok van de bezoeker ingegeven kan worden via het formulier, i.p.v. op de URL.

Ter illustratie:



A screenshot of a web form with a blue border. It contains a single text input field labeled 'Uw gok:'. Below the field is a button labeled 'Nu een gok doen!'.

→ Oplossing: 145875

Via een formulier met POST

We hernemen opnieuw het voorbeeld uit het voorgaande deel.

Een bestand *voorbeeldinvoerform.php*:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Gebruikersinvoer</title>
  </head>
  <body>
    <form action="voorbeeldinvoer.php" method="get">
      Vul je naam in:
      <input type="text" name="naam">
      <input type="submit" value="OK">
    </form>
  </body>
</html>
```

en een bestand *voorbeeldinvoer.php*:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Gebruikersinvoer</title>
  </head>
  <body>
    <h1>
      Goeiemorgen,
      <?php
      print($_GET["naam"]);
      ?>
    </h1>
  </body>
</html>
```

Wijzig het eerste bestand, zodat het formulier via de POST-methode wordt doorgegeven:

```
<form action="voorbeeldinvoer.php" method="post">
```

Wijzig tevens de code in het tweede bestand, zodat de `$_POST`-variabele wordt aangesproken:

```
print($_POST["naam"]);
```

Voer opnieuw uit en zoek het verschil tussen de versie met POST en de versie met GET.

Zonder op al teveel technische details in te gaan kunnen we nu reeds stellen dat de werking van het programma ogenschijnlijk niet verschilt. Het verschil echter, is dat gebruik makende van de GET-methode de naam-waarde-paren uit het formulier worden toegevoegd achteraan de URL. Bij gebruik van de POST-methode echter worden deze naam-waarde-paren ingekapseld/verborgen in het bericht dat naar de webserver verstuurd wordt.

Om te onthouden:

- ✓ Gebruikersinvoer kan gebeuren op verschillende manieren
- ✓ Bij invoer via de URL worden gegevens achteraan op de URL ingegeven d.m.v. parameters.
- ✓ Bij invoer via een formulier worden de gegevens doorgegeven naar de verwerkingsscript, hetzij via de URL (bij een GET), hetzij ingekapseld in het aanvraagbericht (bij een POST).

GET of POST?

Uit wat vooraf ging volgt dat de werking van een formulier met GET min of meer dezelfde is als die van een formulier met POST. De vraag die dan rijst is: "wanneer gebruik ik welke methode?"

Als het versturen van het formulier een handeling teweeg brengt die iets in de applicatie wijzigt (bvb. Een formulier met klantgegevens, voor het aanmaken van een nieuwe klant, het aanbrengen van wijzigingen in bestaande gegevens, enz...), dan gebruik je de POST-methode.

Bij handelingen die slechts gegevens opvragen, en niets wijzigen (bvb een zoekformulier) gebruik je de GET-methode. Een uitzondering op deze regel kan wanneer je expliciet wilt voorkomen dat de gegevens die je invult in de velden in de URL getoond worden (zoals formulier met gebruikersnaam en wachtwoord). In zo'n geval is gebruik van de POST-methode gewenst en nodig.

Hoofdstuk 4

Arrays

In dit hoofdstuk:

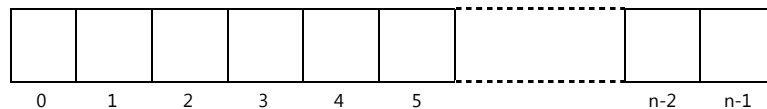
- ✓ Een eerste kennismaking met arrays.
- ✓ Hoe een array invullen?
- ✓ Hoe arrays doorlopen?

Situatieschets en kennismaking

Probleemstelling: we willen een programma maken dat 100 willekeurige getallen tussen 1 en 1000 genereert en bewaart. Het spreekt vanzelf dat we geen 100 verschillende variabelen kunnen gebruiken...

```
$getal1 = ...  
$getal2 = ...  
$getal3 = ...  
...  
$getal99 = ...  
$getal100 = ...
```

Neen, daar denken we beter niet aan. Een oplossing voor dit probleem wordt geboden door arrays (vrij vertaald: "tabellen"). Bij arrays maken we gebruik van één variabelenaam die fungeert als naam van de array, terwijl we een specifieke positie in de tabel aanduiden m.b.v. een "index" (ook wel "sleutel" genoemd). Grafisch kan je een array met n elementen dus voorstellen als:



We hebben reeds kennis gemaakt met een array. We grijpen terug naar het voorbeeld uit het vorige hoofdstuk:

```

<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Gebruikersinvoer</title>
  </head>
  <body>
    <h1>
      Goeiemorgen,
      <?php
        print($_GET["naam"]);
      ?>
    </h1>
  </body>
</html>

```

De **\$_GET** variabele is een array. Kenmerkend is de structuur:

dollarteken + variabelenaam + open vierkant haakje + sleutel + sluit vierkant haakje

De sleutel van een array kan zowel een string (tussen aanhalingstekens) als een getal zijn. PHP is uiterst flexibel wat dit betreft.

Terug naar onze probleemstelling. We schrijven een programma dat 100 willekeurige getallen tussen 1 en 1000 genereert en bewaart.

Bekijk en bestudeer aandachtig de oplossing van dit probleem, in het bijzonder de methode **getArray()**. De rest van het programma is op dit ogenblik minder van belang.

```

<?php
class GetalArrayGenerator {
  public function getArray() {
    $tab = array();
    for ($i=0; $i<100; $i++) {
      $willekeurigGetal = rand(1, 1000);
      $tab[$i] = $willekeurigGetal;
    }
    return $tab;
  }
}
?>
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Willekeurige getallen</title>
  </head>
  <body>
    <pre>
      <?php
        $arrGen= new GetalArrayGenerator();
        print_r($arrGen->getArray());
      ?>
    </pre>
  </body>
</html>

```

De functie `print_r` drukt de inhoud van een complexere variabele (zoals een array) in een handig leesbare vorm af.

De array die hier wordt gebruikt is `$tab`. De index wordt bepaald door een geheel getal (0 t.e.m. 99). Het eerste element van deze array is dus `$tab[0]`, het tweede `$tab[1]` en het laatste `$tab[99]`.

Controleer de uitvoer van het programma. Je merkt dat de array wordt opgevuld met 100 willekeurige getallen tussen 1 en 1000.

Oefening 4.1: ★

Wis de volledige inhoud van de methode `getArray()` en probeer ze zonder hulp opnieuw te construeren, zodat je programma 20 willekeurige getallen tussen -50 en 50 genereert en opslaat

→ *Oplossing: 759332*

Oefening 4.2: ★★

Wis de volledige inhoud van de methode `getArray()` en vul ze opnieuw in zodat je programma een array invult van 50 getallen, te beginnen met 0, zodat elk getal in de reeks telkens 1 méér verschilt van het vorige getal dan de voorgaande twee.

De array bevat dus volgende getallen:

0	1	3	6	10	15		1176	1225
0	1	2	3	4	5		48	49

→ *Oplossing: 474569*

Oefening 4.3: ★★★

Wis de volledige inhoud van de methode `getArray()`. Herneem oefening 2.13 (Fibonaccireeks). Sla dit keer de eerste 30 fibonacci-getallen op in een array.

→ *Oplossing: 254793*

Om te onthouden:

- ✓ Arrays zijn structuren die meerdere waarden kunnen opslaan onder de vorm van één enkele variabele.
- ✓ De plaats/positie van een bepaald element in een array wordt een index of sleutel genoemd.

- ✓ De sleutel van een array kan zowel een getalwaarde als een stringwaarde zijn.

Arrays invullen

Je kan arrays op verschillende manieren invullen. In wat voorafging hebben we reeds één manier gezien. Met een toekenning-expressie berg je een waarde op in de array op een bepaalde positie. Bijvoorbeeld:

```
$mijntabel[6] = 52;  
$kleuren["herfst"] = "Rood en geel";
```

Soms wil je gewoon een verzameling van gegevens bijhouden in een array, en speelt de volgorde van deze gegevens minder een rol. De exacte sleutel of positie van de elementen in de array is daarbij niet zo van belang. Bekijk onderstaand voorbeeld:

```
<?php  
class IngredientenArrayGenerator {  
    public function getIngredienten() {  
        $ing = array();  
        array_push($ing, "bloem");  
        array_push($ing, "zout");  
        array_push($ing, "suiker");  
        array_push($ing, "gist");  
        array_push($ing, "water");  
        return $ing;  
    }  
}  
?  
<!DOCTYPE HTML>  
<html>  
    <head>  
        <meta charset=utf-8>  
        <title>Ingredienten</title>  
    </head>  
    <body>  
        <pre>  
            <?php  
            $ingredienten= new IngredientenArrayGenerator();  
            print_r($ingredienten->getIngredienten());  
            ?>  
        </pre>  
    </body>  
</html>
```

Bekijk de uitvoer.

De volgorde waarin de elementen zich in de array bevinden is dezelfde als deze waarin ze om beurten zijn toegevoegd. Het toevoegen van een element aan een array kan gebeuren met de functie

```
array_push(naamarray, element)
```

Bij elke aanroep wordt het element achteraan de array toegevoegd.

Een alternatieve manier is een gewone toekenning waarbij je geen index tussen de vierkante haakjes vermeldt. Zo voegt

```
$sing[] = "plantaardig vet";
```

de string "plantaardig vet" achteraan toe aan de array `$sing`.

Oefening 4.4: ★

Schrijf een programma dat de namen van de vier seizoenen opslaat in een array. Gebruik hiervoor de `array_push()` functie. Controleer de uitvoer.

→ *Oplossing: 158446*

Oefening 4.5: ★★

Schrijf een programma dat willekeurige getallen tussen 1 en 100 genereert en opslaat in een array, en stopt wanneer een getal gegenereerd wordt dat groter is dan 80. Het laatste getal (het getal dat groter was dan 80) wordt ook nog opgeslagen in de array. Voer je programma verschillende keren uit en controleer de correcte werking.

→ *Oplossing: 828142*

Oefening 4.6: ★★

Maak een programma dat alle oneven getallen van 1 t.e.m. 49 in een array steekt, gevolgd door de reeks van even getallen van 2 t.e.m. 50.

De array bevat dus volgende getallen:

1	3	5		49	2	4		48	50
0	1	2		24	25	26		48	49

→ *Oplossing: 712612*

Om te onthouden:

- ✓ Arrays invullen kan rechtstreeks gebeuren door waarden in te vullen m.b.v. een toekenningsexpressie
- ✓ Arrays invullen kan ook gebeuren door nieuwe elementen toe te voegen met de `array_push()` instructie. Hierdoor wordt telkens een nieuw element achteraan de array toegevoegd.

Arrays doorlopen


Bestudeer onderstaande code. Raad eerst wat de uitvoer zal voorstellen, en controleer pas daarna.

```
<?php
class GetallenGenerator {
    public function getGetallen() {
        $getallen = array();
        for ($i=0; $i<20; $i++) {
            $getallen[$i] = rand(1, 100);
        }
        return $getallen;
    }
}
?>

<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Hello world</title>
    </head>
    <body>
        <ul>

            <?php
            $gg= new GetallenGenerator();
            $tabel = $gg->getGetallen();
            $tabelgrootte = count($tabel);
            for ($k=0; $k < $tabelgrootte; $k++) {
                print("<li>" . $tabel[$k] . "</li>");
            }
            ?>

        </ul>
    </body>
</html>
```



Deze index verhoogt in elke iteratie van de lus

De klassieke manier om een array te doorlopen m.b.v. een for-lus werkt prima wanneer de array geïndexeerd is met getallen:

In bovenstaande code wordt een array opgevuld met 20 willekeurige getallen tussen 1 en 100. In het hoofdprogramma wordt de grootte van de tabel opgevraagd met de functie **count(tabelnaam)**. Daarna wordt een for-lus gebruikt om beurtelings elk element van de array op te vragen (d.m.v. **\$tabel[\$k]**, waarbij **\$k** een teller is).

Je kan evenwel geen beroep doen op een for-lus om een array te overlopen die geïndexeerd werd met stringwaarden.

Bekijk even onderstaande code ter illustratie:

```
<?php
class Kalender {
    public function getAantalDagenInMaand() {
        $dagen = array();
        $dagen["januari"] = 31;
        $dagen["februari"] = 28;
        $dagen["maart"] = 31;
        $dagen["april"] = 30;
        $dagen["mei"] = 31;
        $dagen["juni"] = 30;
        $dagen["juli"] = 31;
        $dagen["augustus"] = 31;
        $dagen["september"] = 30;
        $dagen["oktober"] = 31;
        $dagen["november"] = 30;
        $dagen["december"] = 31;
        return $dagen;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Hello world</title>
    </head>
    <body>
        <ul>
            <?php
            $kal = new Kalender();
            $tabel = $kal->getAantalDagenInMaand();
            foreach ($tabel as $sleutel=>$waarde) {
                print("<li>");
                print($sleutel);
                print(" bevat ");
                print($waarde);
                print(" dagen");
                print("</li>");
            }
            ?>
        </ul>
    </body>
</html>
```

In zulke gevallen is een foreach-constructie adequaat. De algemene vorm is:

```
foreach (naamtabel as naamsleutel=>naamwaarde)
```

In elke iteratie van de lus heb je nu beschikking over zowel de sleutelwaarde als de waarde van het element dat op die positie in de array opgeslagen zit.

Ben je enkel geïnteresseerd in de waarde van die elementen, en niet in de sleutel, dan kan je de foreach constructie korter schrijven als:

```
foreach (naamtabel as naamwaarde)
```

Tip: Je mag de foreach-constructie altijd gebruiken i.p.v. een for-lus om een array te overlopen. Een foreach werkt altijd waar een klassieke for-lus ook zou werken, maar niet andersom.

Oefening 4.7: ☆☆

Schrijf een programma dat 100 willekeurige getallen tussen 1 en 40 genereert. Het programma houdt in een array bij hoeveel keer elk getal gegenereerd werd.

Druk de resultaten af in volgende vorm:

```
...
Getal 24 werd 3 keer gegenereerd.
Getal 25 werd 0 keer gegenereerd.
Getal 26 werd 4 keer gegenereerd.
...
```

De getallen staan op volgorde en worden slechts één keer getoond.

→ *Oplossing: 234912*

Om te onthouden:

- ✓ Het overlopen van arrays gebeurt doorgaans met de foreach instructie.
- ✓ Het overlopen van een array met een eenvoudige for-lus is slechts mogelijk als de sleutels van de array een aaneengesloten reeks getallen vormen, die m.b.v. een for-lus overlopen kunnen worden.

Hoofdstuk 5

Informatie onthouden

In dit hoofdstuk:

- ✓ Wat zijn sessies en hoe gebruik je ze?

Noodzaak?

Door de aard van het http-protocol, waar we hier niet verder op ingaan, gaat na het uitvoeren van een script alle informatie (waarden van variabelen) verloren. Telkens wanneer de pagina herladen wordt, wordt de code opnieuw uitgevoerd.

We grijpen even terug naar een oefening uit één van de voorgaande hoofdstukken. De bedoeling was om een getal te raden, dat door de computer willekeurig gekozen werd (binnen bepaalde grenzen). Het “probleem” dat hierbij optrad is dat de bezoeker slechts één kans krijgt om het getal te raden. Dit is logisch, vermits we tot nu toe niet in staat waren om het getal dat de computer willekeurig koos te onthouden. Elke keer wanneer het formulier met onze gok verzonden werd, werd een nieuw getal gekozen.

In dit hoofdstuk introduceren we het gebruik van sessies en sessievariabelen.

Een afgewerkt voorbeeld

Wat volgt is de inhoud van een bestand *getalradenverwerk.php*. Tik de code niet in, maar overloop ze en markeer de elementen waarvan je denkt dat ze bijdragen tot het onthouden van informatie.



```
<?php
session_start();
class Casino {
    public function getGokResultaat($mijnGok) {
        // De inhoud van gokresultaat is 0 als het getal
        // gevonden is, 1 als de gok te groot was, en -1 als de
        // gok te klein was.
        if (!isset($_SESSION["teRadenGetal"])) {
            $_SESSION["teRadenGetal"] = rand(1, 100);
        }

        if ($mijnGok == $_SESSION["teRadenGetal"]) {
            $gokResultaat = 0;
            unset($_SESSION["teRadenGetal"]);
        } elseif ($mijnGok < $_SESSION["teRadenGetal"]) {
            $gokResultaat = -1;
        } else {
            $gokResultaat = 1;
        }
        return $gokResultaat;
    }
}
?>
<!DOCTYPE HTML>
<html>
<head>
    <meta charset=utf-8>
    <title>Juist gegokt?</title>
</head>
<body>
    <?php
    $gok = $_GET["gok"];
    $casino = new Casino();
    $resultaat = $casino->getGokResultaat($gok);
    if ($resultaat == 0) {
        print("Getal is geraden!");
    } elseif ($resultaat == -1) {
        print("Uw gok is te klein");
    } else {
        print("Uw gok is te groot");
    }
    ?>
    <br><br>
    <a href="getalradenform.php">Nog eens proberen</a>
</body>
</html>
```

Ga er even van uit dat de variabele `$_GET["gok"]` een waarde heeft gekregen door een formulier dat hier niet getoond werd, maar vooraf ging aan deze code. Voer nu het programma "in je hoofd" uit en tracht te achterhalen wat het doet en hoe het werkt.

Wat voor type variabele is `$_SESSION["teRadenGetal"]` ?

Wat kunnen we besluiten uit dit voorbeeld?

Informatie die je wenst te onthouden over verschillende aanvragen heen, moet opgeslagen worden in een sessievariabele. In bovenstaande code moeten we slechts één sessievariabele gebruiken. Deze bevat het getal dat gegenereerd werd door de computer met de `rand()` functie. De toewijzing van deze

waarde aan de variabele gebeurt net zoals je met elke andere variabele zou doen: met de toekenningoperator "=".

Concreet:

```
$_SESSION["teRadenGetal"] = rand(1, 100);
```

Deze expressie genereert een willekeurig getal tussen 1 en 100 en slaat dit op als een sessievariabele `$_SESSION["teRadenGetal"]`. Sessievariabelen maken dus onderdeel uit van de speciale array `$_SESSION[]`.

Met de code

```
if (!isset($_SESSION["teRadenGetal"])) { ... }
```

ga je na of de sessievariabele `$_SESSION["teRadenGetal"]` reeds een waarde heeft gekregen. Het genereren van een willekeurig getal moet immers enkel gebeuren als dat nog niet eerder gedaan werd.

Met de code

```
unset($_SESSION["teRadenGetal"]);
```

verwijder je de inhoud van de sessievariabele in kwestie. Dit mag gebeuren wanneer het getal geraden werd. Op die manier kan de volgende keer een nieuw getal gegenereerd worden.

- Belangrijk: om toegang te krijgen tot het gebruik van sessievariabelen moet je programmacode beginnen met de instructie `session_start()`; (zie voorbeeld).

In sessievariabelen kan je alles opslaan wat je in elke andere variabele zou kunnen opslaan: getallen, stringwaarden, booleaanse waarden, objecten, andere arrays enz...

- Belangrijk: Vooraleer je een object van een bepaalde klasse in een sessie opslaat moet je dit object **serialiseren**! Door een object te serialiseren maak je er een stringwaarde van, die op zijn beurt klaar is voor opslag in een sessie (maar ook bvb in een bestand, een databank, ...). Om een object te serialiseren gebruik je de functie `serialize(object)`. De functie geeft een stringwaarde terug die je zonder meer in de `$_SESSION[]` array kunt opslaan. Wanneer je het object daarna van de sessie wilt halen dien je eerst de functie `unserialize(opgeslagen object)` te gebruiken. Daarmee herstel je het object opnieuw in zijn originele staat.

Voorbeeld:

```
// Op de sessie zetten
$boek = new Boek("De Kreutzersonate", "Leo Tolstoj");
$strBoek = serialize($boek);
$_SESSION["boek"] = $strBoek;

// Van de sessie halen
$sessieBoek = $_SESSION["boek"];
$origineelBoek = unserialize($sessieBoek);
```

We gaan het gebruik van sessies even inoefenen.

Oefening 5.1: ★★

Schrijf een programma dat een willekeurig getal tussen 1 en 100 genereert en afdrukt op het scherm. Zorg ervoor dat het genereren slechts één keer plaatsvindt.

Tip: Verwijder de cookies in je browser om je programma herhaaldelijk te testen met verschillende getallen.

→ *Oplossing:* 716985

Oefening 5.2: ★★

Pas je oplossing van oefening 5.1 aan, zodat het automatisch een nieuw getal genereert, telkens wanneer je de pagina tien keer vernieuwd hebt.

→ *Oplossing:* 191546

Oefening 5.3: ★★

Schrijf een programma dat een teller bijhoudt die elke keer wanneer je de pagina uitvoert verhoogt.

→ *Oplossing:* 465266

Oefening 5.4: ★★

Pas je oplossing van oefening 5.3 aan, zodat de teller terug op 0 gezet wordt wanneer 20 bereikt wordt.

→ *Oplossing:* 463982

Hoe sessions werken

Je kunt je afvragen hoe sessions nu eigenlijk werken, welke kunstgreep er wordt uitgehaald om de variabelen die je in `$_SESSION[]` opslaat toch te kunnen onthouden, niettegenstaande HTTP een stateless protocol is (elke aanvraag-antwoord-communicatie staat volledig los van eventuele vorige of volgende communicaties).

Het verhaal begint eigenlijk bij het aanroepen van de instructie `session_start()`. De PHP scripting engine controleert de aanwezigheid van een specifieke cookie in de HTTP-aanvraag. De cookie bevat een unieke identificatiecode, de session ID. Is deze cookie niet aanwezig (zoals het geval zal zijn bij de allereerste `session_start()` aanroep), dan genereert de scripting engine een session ID en maakt in een bepaalde directory in het bestandssysteem een nieuw bestand aan dat de session ID als naam heeft. De session ID wordt in een cookie opgeslagen, en deze cookie wordt in het HTTP-antwoordbericht teruggezonden naar de browser. De browser slaat de cookie op in het eigen, lokale bestandssysteem.

Wanneer je een waarde in de array `$_SESSION[]` opslaat, dan wordt deze waarde tevens "fysiek" opgeslagen in het bestand op de webserver. Telkens wanneer de scripting engine de instructie `session_start()` interpreteert, wordt gecontroleerd of de gezochte cookie aanwezig is. Is dat het geval, dan wordt de session ID uit de cookie gelezen en krijgt de bezoeker toegang tot zijn "persoonlijke" bestand, en dus al de variabelen die in `$_SESSION[]` werden opgeslagen.

Waarom wordt er in de URL's van sommige PHP applicaties een parameter `PHPSESSID` meegegeven?



Het kan gebeuren dat de bezoeker het gebruik van cookies heeft uitgeschakeld. Om de session ID toch door te kunnen geven wordt deze achteraan de URL toegevoegd. Men noemt dit "URL rewriting"



Om te onthouden:

- ✓ Om informatie over verschillende aanvragen te kunnen onthouden maak je gebruik van sessievariabelen.
- ✓ Een sessievariabele aanmaken kan gebeuren met een eenvoudige toewijzing: `$_SESSION["naamvariabele"] = waarde;`
- ✓ Een variabele kan "leeggemaakt" worden met de `unset()` functie. Dit werkt ook voor sessievariabelen, bvb
`unset($_SESSION["naamvariabele"]);`
- ✓ Nagaan of een variabele een waarde heeft gekregen doe je met de functie `isset`. Dit werkt ook voor sessievariabelen, bvb
`if (isset($_SESSION["naamvariabele"])) ...`

Iets over cookies

Het feit dat sessions waar mogelijk met cookies proberen te werken impliceert dat PHP het gebruik van cookies ondersteunt. Zelf cookies maken is dan ook geen probleem.

Bestudeer onderstaande regel code en tracht te voorspellen wat haar functie is:

```
setcookie("taal", "nl", time() + 3600);
```

Het antwoord volgt.

Er wordt een cookie ingesteld met de naam "taal" en waarde "nl". De cookie is geldig gedurende één uur. De functie `time()` geeft het huidige aantal seconden weer sinds de Unix Epoch. Hierbij tellen we 3600 seconden op, en we krijgen het gewenste tijdstip waarop de cookie dient te vervallen.

De Unix Epoch?



De Unix Epoch is het tijdstip 1 januari 1970, om stipt middernacht. Voer een eenvoudige opdracht `print(time());` uit om te zien hoeveel seconden er sinds dat tijdstip verlopen zijn.



Bestudeer en bekijk de uitvoer van onderstaand voorbeeld.

```
<?php
    if (!empty($_POST["txtNaam"])) {
        setcookie("ingevuldeNaam", $_POST["txtNaam"], time() + 120);
        $naam = $_POST["txtNaam"];
    } else {
        $naam = $_COOKIE["ingevuldeNaam"];
    }
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Cookies</title>
    </head>
    <body>
        <?php if (isset($naam)) { print("Welkom, ") . $naam; } ?>
        <form action="cookies.php" method="post">
            Uw naam: <input type="text" name="txtNaam"
                        value="<?php print($naam);?>"
            <input type="submit" value="Versturen">
        </form>
        <br>
        <a href="cookies.php">Vernieuw de pagina</a>
    </body>
</html>
```

Deze code toont een formulier waar je je naam kunt ingeven. Deze naam wordt in een cookie ondergebracht en is twee minuten geldig vanaf het tijdstip dat deze aangemaakt werd.

De functie **setcookie()** heeft meerdere optionele parameters. Voor een volledig overzicht verwijzen we naar

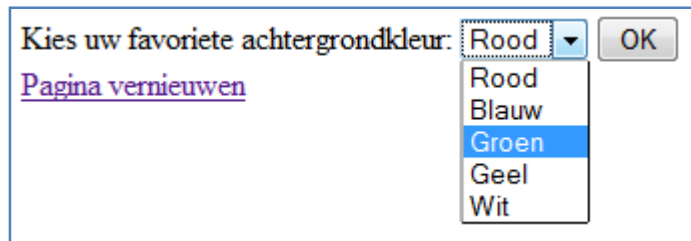
<http://be2.php.net/manual/en/function.setcookie.php>.

Om te onthouden:

- ✓ Een nieuwe cookie maken doe je met de **setcookie()** functie. Deze functie heeft een aantal verplichte en een aantal optionele parameters.
- ✓ De naam-waarde-paren van actieve cookies worden door PHP automatisch in de array **\$_COOKIE** opgeslagen vòòr de uitvoer van het script. Eén bepaalde cookiewaarde bevindt zich dus simpelweg in de variabele **\$_COOKIE["naamcookie"]**.
- ✓ Een bezoeker kan via de browserinstellingen het gebruik van cookie weigeren.
- ✓ Een cookie wordt weggeschreven in een leesbaar tekstbestand en mag om die reden geen vertrouwelijke informatie bevatten.

Oefening 5.5: ☆☆

Ontwikkel een pagina die er ongeveer als volgt uitziet:



The screenshot shows a web form with a label "Kies uw favoriete achtergrondkleur:" followed by a dropdown menu. The dropdown menu is open, showing a list of colors: Rood, Blauw, Groen, Geel, and Wit. The "Groen" option is currently selected and highlighted in blue. To the right of the dropdown menu is an "OK" button. Below the label and dropdown menu is a link that says "Pagina vernieuwen".

Wanneer je een kleur uit de keuzelijst kiest en op de knop OK klikt wordt de achtergrondkleur van de gehele pagina daarmee ingekleurd. Bovendien wordt deze kleur gedurende één dag bewaard.

→ *Oplossing:* 755632

Hoofdstuk 6

Programma's ontwikkelen

In dit hoofdstuk:

- ✓ Hoe is de opbouw van PHP programma's?
- ✓ Hoe splits je een probleem op in presentatie en logica?

Tot nu toe zijn we telkens uitgegaan van een bestaand programma dat slechts werd aangepast. De bedoeling van dit hoofdstuk is na te gaan hoe de structuur van een goed opgebouwd PHP programma er uit kan zien.

Splitsing van presentatie en logica

In het allereerste hoofdstuk hebben we reeds kort beschreven dat dynamische websites best worden opgesplitst in ten minste twee delen: de presentatielaag en de logicalaag.

Dit opsplitsen gebeurt m.b.v. klassen. We hernemen eens het allereerste voorbeeld:

```
<?php
class GreetingGenerator {
    public function getGreeting() {
        return "Hello world!";
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Hello world</title>
    </head>
    <body>
        <h1>
            <?php
            $gg = new GreetingGenerator();
            print($gg->getGreeting());
            ?>
        </h1>
    </body>
</html>
```

Het gedeelte boven de horizontale lijn is de logicalaag, het gedeelte eronder de presentatielaag.

In de praktijk worden de klassen met de logica ondergebracht in een apart bestand. Voordeel hiervan is dat ze makkelijk kunnen worden hergebruikt in meer dan één presentatiebestand.

- Belangrijk: Vanaf nu zullen we in bestanden die enkel PHP-code bevatten steeds de sluit-tag `?>` aan het einde weglaten. Als na deze sluit-tag geen HTML-code meer te vinden is, zal de PHP interpreter hem tijdens de programma-uitvoer zelf toevoegen. Dit is een zgn "best practice" die ook in vele frameworks en CMS-toepassingen zoals bvb Drupal toegepast wordt.

Concreet:

We maken een bestand *greetinggenerator.php* met als inhoud:

```
<?php
class GreetingGenerator {
    public function getGreeting() {
        return "Hello world!";
    }
}
```

Dit is de logicalaag. De presentatielaag is dan een bestand met de naam *greeting.php*:

```
<?php
require_once("greetinggenerator.php");
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Hello world</title>
    </head>
    <body>
        <h1>
            <?php
            $gg = new GreetingGenerator();
            print($gg->getGreeting());
            ?>
        </h1>
    </body>
</html>
```

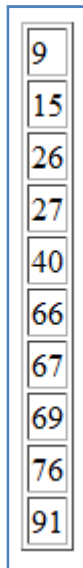
We gebruiken de functie **`require_once("locatieVanBestand")`** om de nodige klasse te importeren voor gebruik in deze pagina. De werking van ons programma blijft uiteraard ongewijzigd.

Noot: Er bestaan nog enkele andere functies waarmee je de inhoud van bestanden kunt importeren in andere bestanden, maar deze zijn nog niet van belang. We komen hier later in de cursus nog op terug.

In de praktijk

Om een goede werkwijze uit te leggen introduceren we een eenvoudige opgave. Gevraagd wordt een pagina te schrijven die een tabel toont van 10 willekeurige getallen, gesorteerd van klein naar groot.

Bijvoorbeeld:



9
15
26
27
40
66
67
69
76
91

Eerste vraag die we ons moeten stellen is: wát moeten we tonen op de pagina (dus niet hoe)? Of anders gezegd: Hoe ziet de logica er uit?

De logica...

We hebben "iets" nodig dat ons een gesorteerde array van 10 willekeurige getallen kan geven. Hoe die getallen gegenereerd worden, of hoe ze gesorteerd worden is nog niet aan de orde. Het "iets" dat ons deze gemaakte array zal aanreiken is een klasse, die we de toepasselijke naam **GetallenReeksMaker** gaan geven. We maken een nieuw bestand *getallenreeksmaker.php* en voorzien deze van de standaardcode die nodig is om een gewone klasse te maken:

```
<?php
class GetallenReeksMaker {
}
```

We hebben nu een lege klasse. We moeten deze voorzien van een functie, die zal instaan voor het aanreiken van de getallenarray aan iedereen die er om vraagt. We noemen deze functie **getReeks()**. De functie in kwestie heeft geen parameters, daar we zullen veronderstellen dat ze totaal autonoom een reeks getallen kan genereren en sorteren.

We krijgen dus:

```
<?php
class GetallenReeksMaker {
    public function getReeks() {
    }
}
```

Hoewel deze functie ver van af is, gaan we ons aandachtspunt verleggen naar de presentatielaag. Echter, om één en ander in de uitvoer reeds te kunnen controleren voorzien we de functie van zgn. "dummy data". We laten de functie een array teruggeven met getallen die hardgecodeerd zijn.

```
<?php
class GetallenReeksMaker {
    public function getReeks() {
        return array(3, 8, 60, 90);
    }
}
```

We doen dit omdat we op die manier straks ons programma kunnen uitvoeren, en kunnen controleren of er inderdaad "een" array in de vereiste tabelvorm wordt afgedrukt. Later zullen we dan de "dummy data" vervangen door de echte programmacode, die de getallen in de array willekeurig genereert en sorteert.

...de presentatie...

De presentatielaag is een bestand dat we de naam *toongetallen.php* geven. Bij het schrijven van dit bestand is het van belang:

- Waar in het document wordt de data getoond?
- Hoe wordt de data getoond?

Wat moet achtereenvolgens gebeuren?

1. De logicalaag importeren, zodat we er gebruik van kunnen maken.
2. De HTML-code (**DOCTYPE**, **<html>**, **<head>**, **<body>** enz...) beschrijven.
3. Een nieuw object van de klasse **GetallenReeksMaker** aanmaken.
4. De functie **getReeks()** uit de klasse **GetallenReeksMaker** aanroepen en het resultaat opslaan in een lokale variabele.
5. Deze variabele als een array overlopen in een lus, en in elke iteratie van de lus de nodige HTML-code schrijven teneinde een echte tabel te zien te krijgen.

Wat volgt is de volledige uitwerking hiervan.

```
<?php
require_once("getallenreeksmaker.php");
?>
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Getallenreeks</title>
  </head>
  <body>
    <?php
    $getReeks = new GetallenReeksMaker();
    $tabel = $getReeks->getReeks();
    ?>
    <table border="1">
      <tbody>
        <?php
        foreach ($tabel as $getal) {
          ?>
          <tr>
            <td>
              <?php print($getal);?>
            </td>
          </tr>
        <?php
        }
        ?>
      </tbody>
    </table>
  </body>
</html>
```

The diagram illustrates the structure of a PHP script with five numbered callouts:

- 1: Points to the first PHP opening tag `<?php` at the top of the script.
- 2: Points to the `<html>` tag, indicating the start of the HTML document.
- 3: Points to the opening PHP tag `<?php` inside the `<body>` tag, where the `GetallenReeksMaker` object is instantiated.
- 4: Points to the `$tabel = $getReeks->getReeks();` line, which retrieves the data for the table.
- 5: Points to the opening PHP tag `<?php` inside the `<tbody>` tag, marking the start of the loop that iterates over the array.

Opdracht: duid met een markeerstift in bovenstaande code alles aan dat PHP-code is (vanaf **<?php** t.e.m. de eerstvolgende `?>`). Op die manier krijg je een beter overzicht en zal je de structuur sneller onder de knie hebben.

Test het programma uit. Op het scherm wordt een tabel van getallen getoond; bij elke uitvoer zijn dit dezelfde getallen 3, 8, 60, 90.

Dit is logisch, vermits dit de array is die we van onze `GetallenReeksMaker` aangereikt kregen. We moeten inderdaad nog de "dummy data" vervangen door de echte programmacode.

Na de vorige hoofdstukken zou je zelf nu reeds in staat moeten zijn om dit te doen. Probeer dit bij wijze van oefening, eventueel gebruik makend van volgende tips.

Tip 1: Probeer eerst een versie die de getallen wel genereert, maar niet sorteert.

Tip 2: Zoek op het Internet op hoe je een array van getallen in PHP kunt sorteren.

...en de afwerking van de logica.

Hieronder vind je de volledig uitgewerkte code van de klasse **GetallenReeksMaker**. De functie **getReeks()** maakt eerst een lege array aan. Daarna wordt met een for-lus (we kennen immers op voorhand het aantal iteraties exact, nl. 10) telkens een willekeurig getal tussen 1 en 100 gegenereerd, waarna dit wordt toegevoegd aan de array.

```
<?php
class GetallenReeksMaker {
    public function getReeks() {
        $stab = array();
        for ($i=0; $i<10; $i++) {
            array_push($stab, rand(1, 100));
        }
        sort($stab);
        return $stab;
    }
}
```

Conclusie

We splitsen programma's steeds op in een logicagedeelte (welke data moet aangeleverd worden en hoe bekomen we deze data?) en een presentatiegedeelte (waar in ons HTML-document moet deze data ingevoegd worden, en op welke manier wordt deze getoond?).

Door deze werkwijze aan te houden ontwerp je een overzichtelijke programma. De extra tijd die je spendeert aan de analyse van de opsplitsing betaalt zich automatisch terug wanneer je dergelijke programma's moet onderhouden of wijzigen. Vergeet ook de herbruikbaarheid van zelfgeschreven klassen niet. De klasse **GetallenReeksMaker** is bruikbaar in elk presentatiebestand dat ze nodig kan hebben.

Oefening 6.1: ★

Wijzig de code uit het voorbeeld met de **GetallenReeksMaker** zo, dat de getallen worden gesorteerd van groot naar klein.

Welke laag (presentatie of logica) moet je aanpassen?

→ Tip: 765931
→ Oplossing: 754669

Oefening 6.2: ☆☆

Wijzig de code uit het voorbeeld met de **GetallenReeksMaker** zo, dat de getallen van klein naar groot in horizontale richting getoond worden i.p.v. verticale richting.

Voorbeeld van een mogelijke uitvoer:

17	24	50	57	68	70	83	87	91	91
----	----	----	----	----	----	----	----	----	----

Welke laag (presentatie of logica) moet je aanpassen?

→ *Oplossing: 231444*

Om te onthouden:

- ✓ Werk bij het ontwikkelen van PHP-programma's steeds aan een zo zuiver mogelijke opsplitsing tussen wat de gebruiker ziet (de presentatielaag) en wat er op de achtergrond gebeurt (de logicalaag).
- ✓ De logicalaag regelt de herkomst van de data en levert ze aan, maar vraagt zich niet af hoe de data getoond zal worden, noch waarvoor ze gebruikt zal worden.
- ✓ De presentatielaag mag rekenen op de logicalaag om de data te ontvangen en bepaalt tevens wat er met deze data moet gebeuren, hoe ze getoond zal worden en op welke manier.

Hoofdstuk 7

Herhalingsoefeningen

In dit hoofdstuk:

- ✓ Herhalingsoefeningen over de voorgaande hoofdstukken

Oefening 7.1: ★★

Ontwerp een programma dat volgende uitvoer genereert:



PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH
PHP is FANTASTISCH

De eerste regel heeft een lettergrootte van 10 pixels. De tweede van 20 pixels enz...

→ Tip 1: 186449

→ Tip 2: 822643

→ Oplossing: 827556

Oefening 7.2: ☆☆

Ontwerp een programma, bestaande uit twee pagina's. De eerste pagina bevat een invulformulier waar men een grondtal kan invullen.

Geef een grondtal:

Na verwerking van het formulier wordt een nieuwe pagina getoond, met daarop de tafel van het grondtal dat werd ingevuld, in tabelvorm.

De tafel van 13	
1 maal 13	13
2 maal 13	26
3 maal 13	39
4 maal 13	52
5 maal 13	65
6 maal 13	78
7 maal 13	91
8 maal 13	104
9 maal 13	117
10 maal 13	130

→ Oplossing: 195642

Oefening 7.3: ☆☆

Schrijf een programma dat de tafels van vermenigvuldiging genereert (1 t.e.m. 10) en in tabelvorm uitschrijft. Op de volgende bladzijde vind je een voorbeeld van de uitvoer.

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

→ Oplossing: 266482

Oefening 7.4: ★★☆☆

Schrijf een programma dat een tabel met de getallen 1 t.e.m. 42 op het scherm toont, waarbij 6 lottonummers in een verschillende achtergrondkleur aangeduid worden.

Denk er aan dat het hier steeds om 6 verschillende getallen gaat. Telkens de pagina vernieuwd wordt, worden de getallen opnieuw willekeurig gekozen.

De winnende lotto-getallen

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42

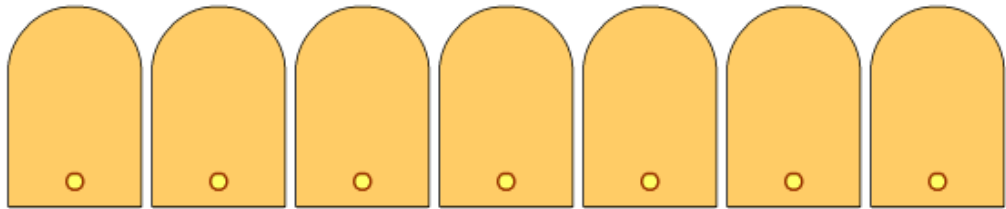
→ Tip 1: 664963

→ Tip 2: 133544

→ Oplossing: 448826

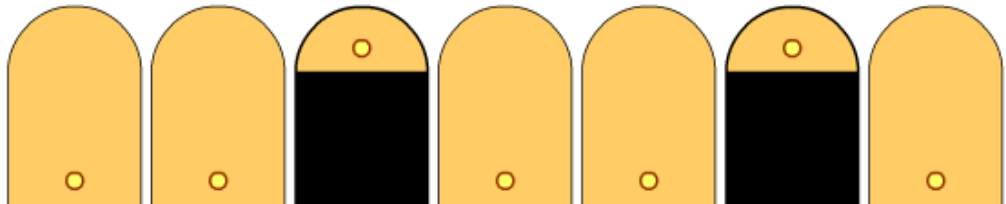
Oefening 7.5: ★★★

Schrijf een spelletje waarbij de speler 7 gesloten deuren te zien krijgt. De figuren *doorclosed.png*, *dooropen.png* en *doortreasure.png* vind je in de oefenmap terug.

Kies een deur

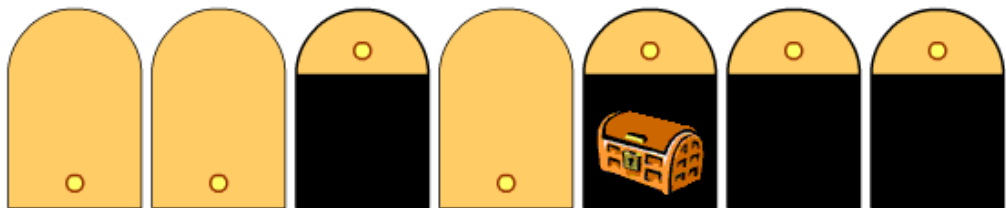
Klik [hier](#) om een nieuw spel te beginnen.

Achter één van deze deuren bevindt zich de schat. De speler kan deuren één voor één openen door op een deur naar keuze te klikken. Eenmaal geopend blijft een deur open.

Kies een deur

Klik [hier](#) om een nieuw spel te beginnen.

De speler kan te allen tijde het spel herbeginnen door op de link onderaan te klikken.

Kies een deur

Klik [hier](#) om een nieuw spel te beginnen.

Eventuele uitbreiding: hou het aantal pogingen bij die de speler moest ondernemen om de schat te vinden. Zorg er ook voor dat wanneer de schat gevonden is, er geen deuren meer kunnen worden geopend.

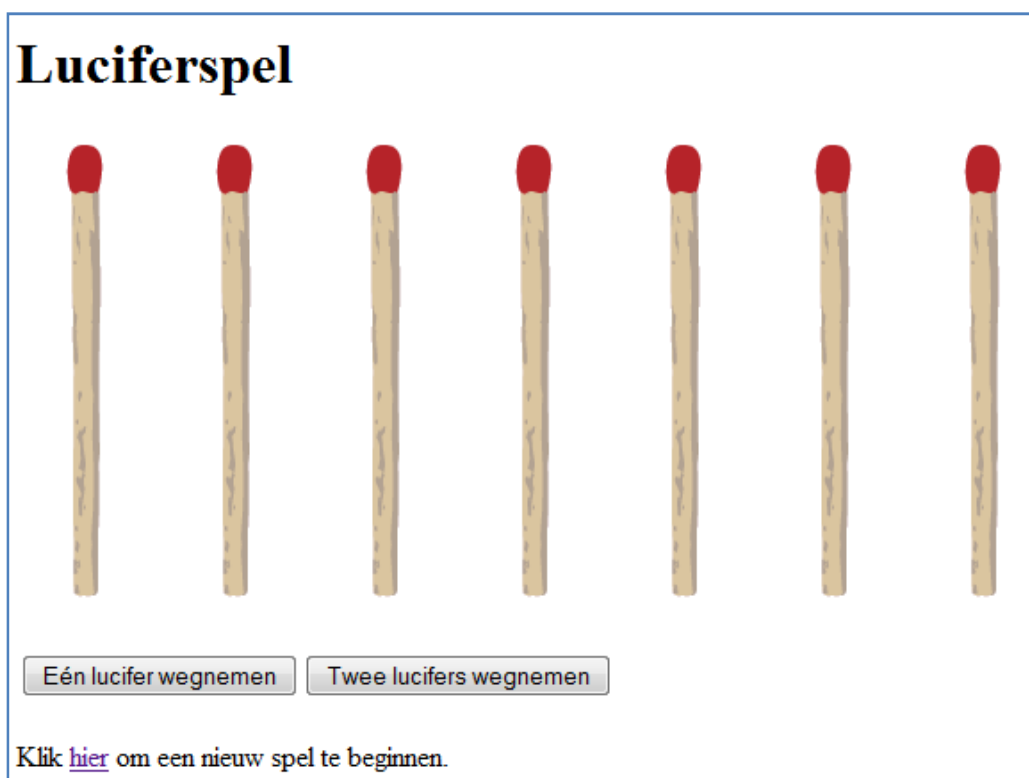
→ Tip 1: 736133

→ Tip 2: 127843

→ Oplossing: 735124

Oefening 7.6: ★★☆☆

Ontwerp een spelprogramma waarbij initieel 7 lucifers worden getoond. In de oefenmap vind je het bestand *lucifer.png* terug.



Eronder bevinden zich twee knoppen waardoor twee spelers beurtelings één of twee lucifers wegnemen. Wie de laatste lucifer weg moet nemen is verloren. Wanneer de laatste lucifer weggenomen wordt, wordt een melding gemaakt dat het spel afgelopen is.

Luciferspel

Het spel is afgelopen.

Klik [hier](#) om een nieuw spel te beginnen.

Er is te allen tijde een link beschikbaar waarmee je het spel kan hervatten, en er opnieuw 7 lucifers getoond worden.

→ *Oplossing: 828472*

DEEL

2

Hoofdstuk 8

Object-oriëntatie

In dit hoofdstuk:

- ✓ Klassen, objecten, attributen en functies nader bekeken
- ✓ Gevorderde technieken: overerving, abstracte klasse en interfaces

Basisprincipes

Klassen maken en gebruiken

We hebben ondertussen reeds veelvuldig gebruik gemaakt van klassen. Het maken van een klasse kan heel eenvoudig met volgende structuur.

```
<?php
class Klant {
}
?>
```

Eens je klasse gedefiniëerd is kan je er objecten van gaan aanmaken, elders in je PHP pagina:

```
<?php
    $mijnKlant = new Klant();
?>
```

Een klasse heeft:

- een **toestand**: de toestand wordt bepaald door de waarde van zgn. attributen. Bijvoorbeeld: een auto kan een attribuut **kleur**, een attribuut **aantalPersonen**, een attribuut **verbruik**, enz... hebben.
- een **gedrag**: het gedrag wordt bepaald door de functies van de klasse. Bijvoorbeeld: een auto kan een functie **rijden()**, een functie **tanken(aantalLiter)**, een functie **getAantalDeuren()** enz... hebben.

Functies die in de klasse gedefiniëerd worden hebben toegang tot de attributen van deze klasse. Op die manier kan je bijvoorbeeld ook functies hebben die er enkel toe dienen de waarden van de attributen terug te geven.

Bijvoorbeeld:

```
<?php
class Auto {
    private $kleur;
    private $aantalDeuren;
    private $verbruik;

    public function getKleur() {
        return $this->kleur;
    }

    public function getAantalDeuren() {
        return $this->aantalDeuren;
    }

    public function getVerbruik() {
        return $this->verbruik;
    }
}
?>

<?php
    $auto = new Auto();
?>
```

Dit soort functies worden dikwijls “getters” genoemd. Merk op dat de speciale impliciete variabele **\$this** wordt gebruikt als referentie naar “de eigen instantie” van een object. Elk object van een klasse is immers een instantie van die klasse. De waarden van gewone attributen kunnen dan ook verschillen per object van de klasse.

Bestudeer aandachtig onderstaand voorbeeld en tracht de uitvoer te voorspellen.

```
<?php
class Auto {
    private $aantalDeuren;

    public function setAantalDeuren($aantal) {
        $this->aantalDeuren = $aantal;
    }

    public function getAantalDeuren() {
        return $this->aantalDeuren;
    }
}
```

```
$kleineAuto = new Auto();
$groteAuto = new Auto();
$kleineAuto->setAantalDeuren(3);
$groteAuto->setAantalDeuren(5);
$groteAuto->setAantalDeuren($kleineAuto->getAantalDeuren()+2);

?>
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Klassen in de praktijk</title>
  </head>
  <body>
    Aantal deuren in een kleine auto:
    <?php print($kleineAuto->getAantalDeuren());?>
    <br>
    Aantal deuren in een grote auto:
    <?php print($groteAuto->getAantalDeuren());?>
  </body>
</html>
```

Test het voorbeeld uit en controleer je antwoord.

Oefening 8.1: ★★

Maak een klasse **Thermometer**. Een thermometer houdt steeds zijn eigen temperatuur bij. Voorzie de mogelijkheid om de temperatuur te verhogen met een functie **verhoog(aantalGraden)** en te verlagen met een functie **verlaag(aantalGraden)**. Test je oplossing uit.

→ *Oplossing: 655712*

Om te onthouden:

- ✓ Je maakt een object van een bepaalde klasse aan met het sleutelwoord **new**.
- ✓ De attributen van een klasse vormen de toestand van die klasse
- ✓ De functies van een klasse vormen het gedrag van die klasse

Constructors

De constructor van een klasse is de functie die wordt uitgevoerd op het ogenblik dat een nieuw object van deze klasse aangemaakt wordt. De constructor wordt gedefiniëerd door `__construct` als naam van de functie te gebruiken. Een constructor kan net zoals gewone functies nul of meerdere parameters bezitten.

Een voorbeeld ter illustratie:

Deze functie wordt uitgevoerd zodra er een nieuw object wordt aangemaakt van deze klasse.

```
<?php
class Auto {
    private $aantalDeuren;

    public function __construct() {
        $this->aantalDeuren = 2;
    }

    public function setAantalDeuren($aantal) {
        $this->aantalDeuren = $aantal;
    }

    public function getAantalDeuren() {
        return $this->aantalDeuren;
    }
}
?>
```

Op het moment dat er een nieuw object van de klasse **Auto** gemaakt wordt, worden alle instructies in de constructor onmiddellijk uitgevoerd. In dit geval wordt het aantal deuren ingesteld op 2.

Oefening 8.2: ★

Breid je oplossing van oefening 8.1 uit zodat bij het aanmaken van een nieuwe thermometer de temperatuur automatisch ingesteld wordt op 25.

→ Oplossing: 644288

Oefening 8.3: ★

Vervang de constructor van oefening 8.2 door een andere die toelaat dat je zelf de begintemperatuur kan instellen m.b.v. een parameter. Zo moet het bijvoorbeeld mogelijk zijn om te schrijven:

```
$therm = new Thermometer(25);
```

Zorg er nadien voor dat de klasse **Thermometer** geen temperatuurwaarden kan bevatten die lager zijn dan -50, en geen temperaturen hoger dan 100.

→ Oplossing: 165944

Om te onthouden:

- ✓ De constructor van een klasse is een functie die wordt uitgevoerd wanneer een nieuw object wordt aangemaakt van die klasse.
- ✓ De constructorfunctie heeft altijd de naam **__construct**.
- ✓ Een constructor kan nul of meerdere parameters bezitten.

Overerving

Het principe van overerving ("inheritance") werd reeds besproken in de cursus Objectgeoriënteerde Principes. PHP biedt ondersteuning voor enkelvoudige inheritance (= je kan een klasse van maximaal één andere klasse afleiden).

Bestudeer volgend voorbeeld, maar voer het niet uit. Ga na welk onderdeel van de code zorgt voor de overerving.

```
<?php
class Product {
    private $prijs;

    public function getPrijs() {
        return $this->prijs;
    }

    public function setPrijs($prijs) {
        $this->prijs = $prijs;
    }
}

class Dvd extends Product {
    private $speelduur;

    public function getSpeelduur() {
        return $this->speelduur;
    }

    public function setSpeelduur($speelduur) {
        $this->speelduur = $speelduur;
    }
}

class Boek extends Product {
    private $auteur;

    public function getAuteur() {
        return $this->auteur;
    }

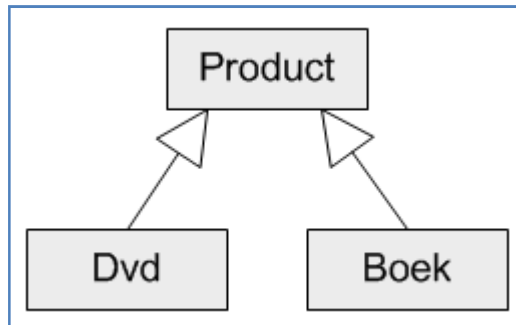
    public function setAuteur($auteur) {
        $this->auteur = $auteur;
    }
}
?>
```

Er worden drie klassen gedefiniëerd: **Product**, **Dvd** en **Boek**. Enkel de klasse **Dvd** beschikt over een speelduur. Enkel de klasse **Boek** beschikt over een auteur. Vermits zowel **Dvd** als **Boek** afgeleid zijn van de klasse **Product** beschikken beide over een prijs. De aanduiding **extends** **Naamklasse** zorgt voor de overerving.

Dat betekent dat de volgende code

```
$boek = new Boek();
print($boek->getPrijs());
```

probleemloos werkt. De klasse **Boek** heeft zelf geen functie **getPrijs()**, maar vermits deze van de klasse **Product** is afgeleid, erft deze alle functies en wordt het opvragen van de prijs toch mogelijk.



Oefening 8.4: ★★

Van een cursist moet worden bijgehouden hoeveel cursussen hij/zij volgt, van een medewerker wordt bijgehouden hoeveel cursisten hij/zij begeleidt. Van zowel cursisten als medewerkers wordt een familienaam en voornaam bijgehouden. Schrijf een programma waardoor volgende code kan worden uitgevoerd:

```
<?php
$cursist = new Cursist();
$medewerker = new Medewerker();
$cursist->setFamilienaam("Peeters");
$cursist->setVoornaam("Jan");
$medewerker->setFamilienaam("Janssens");
$medewerker->setVoornaam("Tom");
?>
<!DOCTYPE HTML>
<html>
  <head>
    <title>Cursisten en medewerkers</title>
  </head>
  <body>
    <h1>Namen</h1>
    <ul>
      <li><?php print($cursist->getVollNaam());?></li>
      <li><?php print($medewerker->getVollNaam());?></li>
    </ul>
  </body>
</html>
```

De uitvoer van dit programma is:

Namen

- Peeters, Jan
- Janssens, Tom

→ Oplossing: 746992

Om te onthouden:

- ✓ Een afgeleide klasse is een klasse die overerft van een basisklasse. De afgeleide klasse erft daarbij alle eigenschappen van de basisklasse.
- ✓ Je gebruikt het sleutelwoord **extends** om afleiding te realiseren.

Constructors en overerving

Zoals reeds eerder beschreven is de constructor van een klasse een speciale functie die wordt uitgevoerd wanneer we een nieuw object aanmaken van die klasse. Het komt dikwijls voor dat je extra parameters wilt meegeven met deze constructor, die bvb. zorgen voor het invullen van attributen van de klasse. Wanneer je een klasse B afgeleid hebt van een klasse A, zal je bij het aanmaken van een object van de klasse B parameters willen meegeven die worden verwerkt door de constructor van klasse A.

Bekijk onderstaande code:

```
<?php
class Product {
    private $prijs;

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

    public function getPrijs() {
        return $this->prijs;
    }

    public function setPrijs($prijs) {
        $this->prijs = $prijs;
    }
}

class Dvd extends Product {
    private $speelduur;

    public function __construct($speelduur, $prijs) {
        parent::__construct($prijs);
        $this->setSpeelduur($speelduur);
    }

    public function getSpeelduur() {
        return $this->speelduur;
    }

    public function setSpeelduur($speelduur) {
        $this->speelduur = $speelduur;
    }
}
```

```

class Boek extends Product {
    private $auteur;

    public function getAuteur() {
        return $this->auteur;
    }

    public function setAuteur($auteur) {
        $this->auteur = $auteur;
    }
}

```

Duid met een markeerstift de regels code aan die nieuw zijn tegenover die in de vorige sectie. Welke nieuwe constructie bemerk je?

Met de code **parent::__construct(...)**; voer je de constructor uit van de bovenliggende klasse (d.i. de klasse waarvan je hebt afgeleid).

Oefening 8.5: ★★

Breid oefening 8.4 uit zodat onderstaande code werkt:

```

<?php
    $cursist = new Cursist("Peeters", "Jan", 3);
    $medewerker = new Medewerker("Janssens", "Tom", 8);
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Cursisten en medewerkers</title>
    </head>
    <body>
        <h1>Namen</h1>
        <ul>
            <li><?php print($cursist->getVollNaam() . " volgt " .
                $cursist->getAantalCursussen() . " cursus(sen)");?></li>
            <li><?php print($medewerker->getVollNaam() . " begeleidt " .
                $medewerker->getAantalCursisten() .
                " cursist(en)");?></li>
        </ul>
    </body>
</html>

```

→ Oplossing: 436917

Om te onthouden:

- ✓ De constructor van de basisklasse waarvan een andere klasse afgeleid is, is toegankelijk via **parent::__construct(parameters)**.
- ✓ Algemeen kan je elke functie die in de basisklasse gedefiniëerd werd aanroepen via **parent::naamvanfunctie(parameters)**.

Toegankelijkheid tot klassen

In alle voorgaande voorbeelden waar objectoriëntatie bij kwam kijken werden alle attributen van een klasse steeds gedefiniëerd met het sleutelwoord `private`, terwijl alle functies voorafgegaan werden door het sleutelwoord `public`.

Klassen op deze manier opmaken zorgt ervoor dat alle code die functionaliteit van jouw klasse gebruikt:

- probleemloos toegang heeft tot alle functies van de klasse
- geen rechtstreekse toegang heeft tot de attributen van de klasse

Dat betekent dat, wanneer je waarden wil toekennen aan attributen van een klasse, dit via een “setter” functie moet gebeuren.

Probeer zelf te beredeneren waarom deze werkwijze gehandhaafd wordt.

Om te onthouden:

- ✓ Tijdens de opbouw van klassen hanteer je zoveel mogelijk het “black-box” principe: zorg ervoor dat attributen enkel bereikbaar, opvraagbaar en instelbaar zijn via functies.
- ✓ Je maakt attributen van een klasse `private`, om te voorkomen dat ze rechtstreeks (zonder controle) ingesteld kunnen worden van buitenaf (bvb bij een Rekening om te voorkomen dat het rekeningnummer compleet willekeurig gekozen en ingesteld kan worden). Deze controle kan je wel doen via een functie.

Statische methodes en attributen

Telkens wanneer een object aangemaakt wordt, wordt er geheugenruimte voorzien voor elk van zijn eigen attributen, en kan je a.d.h.v. methodes het gedrag van het object beïnvloeden.

Tot nu toe hebben we enkel gebruik gemaakt van dit soort attributen en methodes. Twee objecten `$k1` en `$k2` van een klasse `MijnKlasse` werkten onafhankelijk van elkaar.

Dit in het achterhoofd houdend, bestudeer onderstaande code:

```
<?php
class MeetkundeKlasse {

    private static $pi = 3.14;
    private $aantal = 2;

    public static function getOppervlakteCirkel($straal) {
        return ($straal * $straal * self::$pi);
    }
}
?>

<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Oppervlakte cirkel</title>
    </head>
    <body>
        <h1>
            <?php
                $meet1 = new MeetkundeKlasse();
                $meet2 = new MeetkundeKlasse();
                print($meet1->getOppervlakteCirkel(4.5));
            ?>
        </h1>
    </body>
</html>
```

Opgave: Markeer de nieuwe elementen in dit voorbeeld.

Het sleutelwoord **static** voor het attribuut **\$pi** duidt aan dat dit een klasseattribuut is. De waarde van deze variabele is dezelfde voor *alle* objecten van deze klasse, hoeveel er ook gemaakt worden.

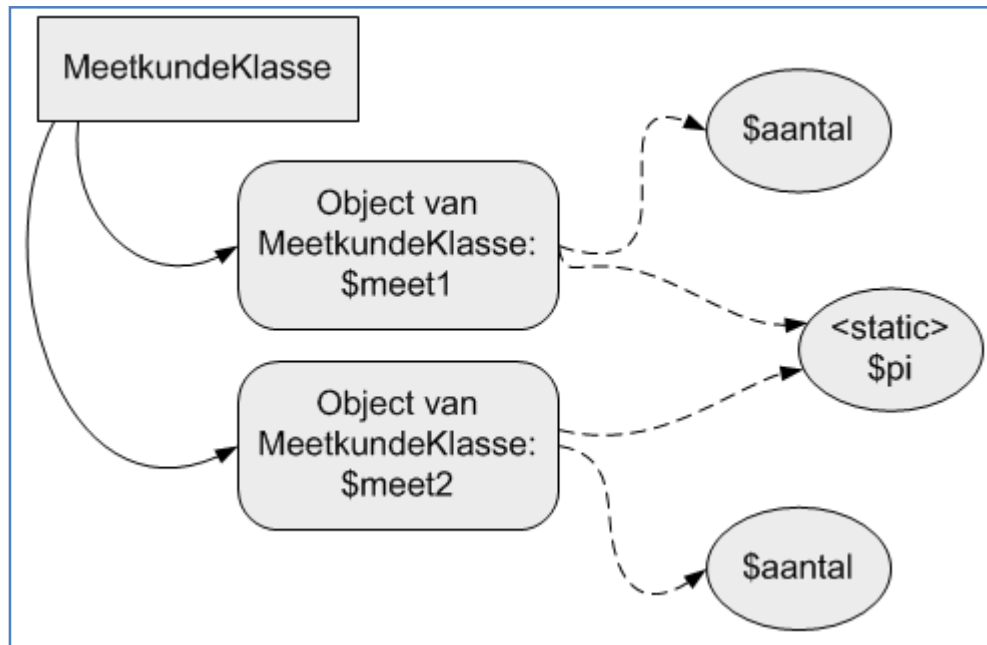
Hetzelfde sleutelwoord **static** in de header van de functie **getOppervlakteCirkel** duidt aan dat het gedrag van deze functie niet afhangt van attributen die eigen zijn aan een specifiek object van deze klasse. Dat betekent in ons voorbeeld dat je binnen de functie **getOppervlakteCirkel** geen gebruik kan/mag maken van de variabele **\$this->aantal** (vermits dit geen klassevariabele is). Probeer je dit wel, dan wordt je een foutmelding voorgeschoteld:

Fatal error: Using \$this when not in object context in C:\xampp\htdocs\cursusphp\vb1\vb31.php on line 8

Er worden twee objecten aangemaakt van de klasse **MeetkundeKlasse**: **\$meet1** en **\$meet2**. Elk van deze objecten heeft toegang tot twee variabelen: **\$pi** en **\$aantal**. De waarde van **\$aantal** van **\$meet1** kan verschillen van die van **\$meet2**. Beide objecten maken gebruik van eenzelfde statische variabele **\$pi**.

M.b.v. het sleutelwoord **self::** (voorafgaand aan de variabelenaam) krijg je toegang tot de statische variabele in de klasse, in dit geval **self::\$pi**.

Op de volgende pagina vind je hiervan een schematisch overzicht.



Oefening 8.6: ★★

Bouw een klasse **Rekening** die een bankrekening voorstelt. Bij het aanmaken van een nieuwe rekening moet het rekeningnummer worden meegegeven. Verder wordt in de klasse de intrest opgeslagen. Deze intrest bedraagt steeds 3%, voor alle rekeningen.

Werk de rekening verder uit zodat er een bedrag kan gestort worden met een functie **stort(\$bedrag)**. Voorzie tevens een functie **getSaldo()** die het huidige saldo opvraagt, alsook een functie **voerIntrestDoor()**. Deze laatste functie verhoogt het saldo één keer met de intrestvoet.

Test je klasse uit met dit programma:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Rekeningnummers</title>
  </head>
  <body>
    <h1>
      <?php
        $rek = new Rekening("091-0122401-16");
        print("Het saldo is: " . $rek->getSaldo() . "<br>");
        $rek->stort(200);
        print("Het saldo is: " . $rek->getSaldo() . "<br>");
        $rek->voerIntrestDoor();
```

```
print("Het saldo is: " . $rek->getSaldo() . "<br>");  
?>  
  
</h1>  
</body>  
</html>
```

Het resultaat op het scherm is dan:

Het saldo is: 0
Het saldo is: 200
Het saldo is: 206

→ *Oplossing:* 822469

Om te onthouden:


- ✓ Je gebruikt statische attributen en functies wanneer hun functionaliteit niet afhangt van de rest van de klasse.
- ✓ De inhoud van een statische variabele is altijd dezelfde, voor alle objecten die van een bepaalde klasse gemaakt worden.
- ✓ Een "gewone" functie kan gebruik maken van statische attributen en "gewone" attributen.
- ✓ Een statische functie kan gebruik maken van statische attributen, maar niet van "gewone" attributen.

Abstracte klassen en functies

In een online winkel verschillen de verhandelbare items van elkaar. Dvd's hebben andere attributen (speelduur, genre,...) dan boeken (uitgever, auteur,...). Ze hebben evenwel ook een aantal gemeenschappelijke attributen. Zowel dvd's als boeken hebben een omschrijving, prijs,... Het zijn allemaal producten. Het ontwerpen van een klasse **Product** is een logisch gevolg, net als de klasse **Dvd** en **Boek** die overerven van **Product**.

Echter, we willen voorkomen dat er objecten van de klasse **Product** kunnen aangemaakt worden, vermits deze zonder verdere specificatie geen bestaansrecht hebben. We maken **Product** daarom abstract.

Hieronder vind je een illustratief voorbeeld van hoe dit er zou kunnen uitzien.



```
<?php
abstract class Product {
    private $omschrijving;
    private $prijs;
    public function setOmschrijving($omschrijving) {
        $this->omschrijving = $omschrijving;
    }
    public function setPrijs($prijs) {
        $this->prijs = $prijs;
    }
    public function getOmschrijving() {
        return $this->omschrijving;
    }
    public function getPrijs() {
        return $this->prijs;
    }
}

class Dvd extends Product {
    private $speelduur;
    private $genre;

    public function setSpeelduur($speelduur) {
        $this->speelduur = $speelduur;
    }
    public function setGenre($genre) {
        $this->genre = $genre;
    }
    public function getSpeelduur() {
        return $this->speelduur;
    }
    public function getGenre() {
        return $this->genre;
    }
}

class Boek extends Product {
    private $uitgever;
    private $auteur;

    public function setUitgever($uitgever) {
        $this->uitgever = $uitgever;
    }
    public function setAuteur($auteur) {
        $this->auteur = $auteur;
    }
    public function getUitgever() {
        return $this->uitgever;
    }
    public function getAuteur() {
        return $this->auteur;
    }
}

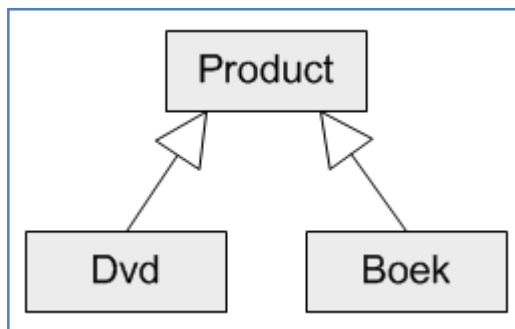
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Boeken</title>
    </head>
    <body>
```

```
<h1>

    <?php
        $b = new Boek();
        $b->setPrijs(25.0);
        print($b->getPrijs());
    ?>

</h1>
</body>
</html>
```

Let op het sleutelwoord "abstract" bij de definitie van de klasse **Product**. De rest van de code is een geval van "gewone" overerving:



In het voorgaande voorbeeld is er enkel sprake van abstract klassen. Echter, ook een functie kan abstract gedefiniëerd worden.

```
<?php
abstract class Product {

    ...
    abstract public function drukAf();
    ...
}
```

Door de functie **drukAf()** abstract te maken verplicht je alle klassen die erven van de klasse **Product** (in bovenstaand voorbeeld zijn dat **Dvd** en **Boek**) deze functie te definiëren en inhoud te geven. Merk immers op dat de functie **drukAf()** van de klasse **Product** alleen maar gedefiniëerd is, en geen inhoud heeft (er staat enkel een puntkomma op het einde van de functiedefinitie, en geen accolades { }).

Abstracte functies worden meestal gebruikt voor die functies waarvan het gedrag heel afhankelijk is van het type klasse waarin ze gebruikt wordt. De programmacode voor het "afdrukken" van een dvd (bvb titel, omschrijving, genre, speelduur, lijst van acteurs,...) ziet er heel anders uit dan voor het afdrukken van een boek (bvb de inhoud van de bladzijden). Daarom wordt er geen code geschreven in de functie **drukAf()** van de klasse **Product**, maar wordt dat overgelaten aan de klassen die van **Product** erven: **Dvd** en **Boek**.

Uiteraard wordt het sleutelwoord "abstract" weggelaten wanneer je de functie inhoud geeft.

```
class Dvd extends Product {  
    ...  
    public function drukAf() {  
        // Hier komt de code voor het afdrukken van een Dvd  
    }  
    ...  
}  
  
class Boek extends Product {  
    ...  
    public function drukAf() {  
        // Hier komt de code voor het afdrukken van een Boek  
    }  
    ...  
}
```

Enkele spelregels die je moet volgen wanneer je abstracte functies gebruikt:

- Een abstracte functie mag zelf geen inhoud hebben wanneer ze gedefiniëerd wordt in de superklasse.
- Elke klasse B die afgeleid is van een klasse A moet alle abstracte functies van A implementeren (inhoud geven), tenzij de klasse B zelf ook abstract is. In dat geval wordt de verantwoordelijkheid verder doorgegeven naar de klasse C, afgeleid van B.
- Vanaf het ogenblik dat een klasse A ten minste één abstracte functie bevat, MOET de klasse A zelf ook als abstract gedefiniëerd worden.

Oefening 8.7: ☆☆☆

Pas de klasse **Rekening** uit oefening 8.6 aan zodat je geen objecten meer kunt aanmaken van deze klasse. In plaats daarvan maak je twee klassen **Zichtrekening** en **Spaarrekening** die hiervan overerven. Verplaats tevens de statische variabele **\$intrest** naar deze subklassen. De intrest op een spaarrekening is steeds 3%, deze op een zichtrekening 2.5%.

Wijzig ook het hoofdprogramma, zodat je achtereenvolgens de correcte werking van de spaarrekening en de zichtrekening kunt testen.

→ *Oplossing: 191734*

Om te onthouden:

- ✓ Abstracte klassen worden gebruikt wanneer je een deel van een klasse wenste te implementeren, maar waarbij die klasse nog niet volledig is, en waarbij je dus wilt voorkomen dat er objecten van aangemaakt worden.
- ✓ Er kunnen geen objecten aangemaakt worden van een abstracte klasse.

Interfaces

Een interface is een constructie die toelaat een aantal functies te definiëren die van inhoud moeten voorzien worden in de klasse die deze interface implementeert. Je legt a.h.w. een aantal afspraken vast die elke klasse moet volgen als ze de interface implementeert.

Een **Persoon** en een **Oppervlakte** beschikken beiden over een functie **getGrootte()**. De grootte van een persoon wordt weliswaar op een totaal andere manier berekend dan de grootte van de oppervlakte, maar deze berekening wordt niet vastgelegd in een interface. Wat wel vastgelegd wordt in de interface is het "feit" dat er een functie **getGrootte()** bestaat. Elke klasse die de interface wil gebruiken moet een functie **getGrootte()** bevatten. Vanzelfsprekend kan elke klasse dan zelf kiezen hoe deze grootte berekend wordt.

In code:

```
<?php
interface Omvang {

    public function getGrootte();

}

class Persoon implements Omvang {

    private $lengte;

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

    public function getGrootte() {
        return $this->lengte;
    }

}

class Oppervlakte implements Omvang {

    private $breedte;
    private $lengte;

    public function __construct($breedte, $lengte) {
        $this->breedte = $breedte;
        $this->lengte = $lengte;
    }

    public function getGrootte() {
        return $this->lengte * $this->breedte;
    }

}
?>
```

Let op! De functie heeft geen inhoud, dus wordt ook NIET begrensd door accolades { en }, maar eindigt op een puntkomma !

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Interfaces voorbeeld</title>
  </head>
  <body>
    <h1>
      <?php
        $p = new Persoon(190);
        print($p->getGrootte() . "<br>");
        $o = new Oppervlakte(20, 30);
        print($o->getGrootte());
      ?>
    </h1>
  </body>
</html>
```

Verwijder de functie **getGrootte()** uit de klasse **Persoon**, voer uit en kijk wat er gebeurt.

Let op: hoewel je een klasse van slechts één andere klasse (of abstracte klasse) kunt laten afleiden, kan een klasse wel meerdere interfaces implementeren.

Oefening 8.8: ☆☆

Breid oefening 8.7 verder uit zodat een **Spaarrekening** en een **Zichtrekening** een interface **Omschrijving** implementeren. De functie **getOmschrijving()** geeft bij een **Zichtrekening** de tekst "Kortetermijnrekening" en bij een **Spaarrekening** de tekst "Langetermijnrekening" terug.

Test hun functionaliteit uit in je hoofdprogramma.

→ *Oplossing: 448629*

Om te onthouden:

- ✓ Interfaces zijn structuren waarin enkel wordt gedefiniëerd welke functies aanwezig zijn en welke parameters worden meegegeven.
- ✓ Interfaces bepalen niet hoe een functie wordt ingevuld.
- ✓ Een klasse die een interface implementeert moet ook alle functies die de interface voorschrijft implementeren (tenzij het natuurlijk zelf een abstracte klasse is).

DEEL

3

Hoofdstuk 9

Databanktoegang

In dit hoofdstuk:

- ✓ Nut van databanken
- ✓ Verbinding maken met een databank
- ✓ SQL-queries uitvoeren

Een korte inleiding...

Nut van databanken

Tot hiertoe hebben we nog geen gebruik kunnen maken van het opslaan van gegevens. We hebben weliswaar met sessievariabelen gewerkt, die het mogelijk maken om data te bewaren over verschillende HTTP-aanvragen, maar deze strategie is slechts beperkt bruikbaar:

- De inhoud van sessievariabelen gaat verloren wanneer men de cookies wist of alle instanties van de browser sluit.
- De inhoud van sessievariabelen is slechts opvraagbaar door de persoon waarvoor deze aangemaakt werd.

Je kan databanken gebruiken om gegevens beschikbaar te stellen voor meerdere bezoekers, terwijl deze gegevens niet verdwijnen wanneer men de sessie verlaat (browser sluiten, cookies wissen,...).

PDO

Hoewel we in deze cursus werken met MySQL als databanksysteem en het in PHP perfect mogelijk is MySQL-gerichte opdrachten uit te voeren, kiezen we voor het abstract behandelen van databanktoegang. Om dit te realiseren maken we gebruik van PDO (PHP Data Objects). Dit zorgt ervoor dat we functies kunnen uitvoeren op de databank, zonder ons zorgen te hoeven maken over het gebruikte databanksysteem (MySQL, MS SQL, Oracle, PostgreSQL, SQLite,...).

Verbinding maken

Het maken van een verbinding met een databankserver en het selecteren van een databank gaan vooraf aan het ophalen van gegevens uit of wegschrijven van gegevens naar tabellen uit de databank.

De **PDO** klasse (standaard geactiveerd vanaf PHP versie 5.1) heeft een constructor waarmee je de verbinding kunt realiseren. Gebruik makend van het verkregen object voer je via functies de nodige SQL-statements uit.

We geven een voorbeeld ter illustratie. De database "cursusphp" bevat een tabel "personen". Bekijk de manier waarop de verbinding werd gemaakt.

```
<?php
class PersonenLijst {

    public function getLijst() {
        $lijst = array();

        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
                      "cursusgebruiker", "cursuspwd");
        $dbh = null;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Databanken introductie</title>
    </head>
    <body>
        <?php
        $pl = new PersonenLijst();
        $pl->getLijst();
        ?>
    </body>
</html>
```



Merk op dat de verbinding onmiddellijk daarna opnieuw wordt vernietigd om ongebruikte systeembronnen vrij te geven, via

```
$dbh = null;
```

In de praktijk wil je voor dit gebeurt uiteraard eerst nog gegevens ophalen, maar voorlopig is het enkel van belang dat je weet hoe je een verbinding opzet.

De constructor van de klasse **PDO** bevat drie parameters:

- De connection string: hierin wordt aangeduid welke SQL-variant zal worden gebruikt (in dit geval MySQL), alsook het IP-adres/hostname van de databankserver en de naam van de databank die zal worden aangesproken.
- De gebruikersnaam waarmee PHP een verbinding zal trachten te maken
- Het wachtwoord dat bij de gebruikersnaam hoort, en nodig is voor het aanmelden bij de databankserver.

Krijg je bij uitvoer van dit voorbeeld een foutmelding, dan is er wellicht iets misgegaan bij de aanmelding. Controleer de connection string, je gebruikersnaam en wachtwoord op eventuele (tik)fouten.

Krijg je geen foutmelding, dan is de verbinding met succes tot stand gebracht. Tijd om gegevens op te halen!

Gegevens ophalen

We maken een array aan die wordt opgevuld met stringwaarden als elementen. Eén element bevat telkens:

familienaam + komma + spatie + voornaam

Bedoeling is dat we alle records uit de tabel personen ophalen, van elk record de velden familienaam en voornaam extraheren en deze gebruiken om de elementen van de array te vormen.

Daarna wordt de array in de presentatielaag overlopen, en worden de namen netjes in een lijstvorm getoond.

Hieronder vind je de volledige code hiervan terug. Het gedeelte dat zorgt voor het leggen van de verbinding is uiteraard ongewijzigd gebleven.

```
<?php
class PersonenLijst {

    public function getLijst() {
        $lijst = array();

        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
                        "cursusgebruiker", "cursuspwd");

        $resultSet = $dbh->query("select familienaam, voornaam
                                from personen");

        foreach ($resultSet as $rij) {
            $naam = $rij["familienaam"] . ", " . $rij["voornaam"];
            array_push($lijst, $naam);
        }
        $dbh = null;
        return $lijst;
    }
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Databanken introductie</title>
    </head>
    <body>
        <?php
        $pl = new PersonenLijst();
        $stab = $pl->getLijst();
        ?>
        <ul>

            <?php
            foreach ($stab as $naam) {
                print("<li>" . $naam . "</li>");
            }
            ?>

        </ul>

    </body>
</html>
```

Opgave: Pas deze code minimaal aan zodat de personen gerangschikt op familienaam verschijnen.

Merk op dat logica en presentatie zoals gewoonlijk gescheiden worden. Voer deze code uit. Een voorbeeld van de uitvoer:

- Peeters, Bram
- Van Dessel, Rudy
- Vereecken, Marie
- Maes, Eveline
- Vangeel, Joke
- Van Heule, Pieter
- Naessens, Katleen
- Sleenwaert, Koen

Oefening 9.1: ☆☆

De databank "cursusphp" bevat een tabel modules. Per module wordt een uniek id, de naam en de prijs bijgehouden.

Ontwerp een pagina die de bezoeker toelaat een minimale en maximale prijs in te geven.

Modules

Geef een minimumprijs: euro

Geef een maximumprijs: euro

Het formulier wordt verwerkt door een tweede pagina. Deze pagina toont enkel de modules waarvan de prijs ligt tussen de twee opgegeven waarden. Geef je bijvoorbeeld minimumwaarde 90 en maximumwaarde 100 in, dan is het resultaat:

Zoekresultaat

- UML (90 euro)
- SQL (99.9 euro)

Controleer je oplossing met verschillende waarden. Ter vereenvoudiging mag je van de veronderstelling uitgaan dat er twee correcte getalwaarden ingevuld worden in de formulervelden.

→ Oplossing: 479336

Gegevens toevoegen

Zoals je wellicht reeds weet gebeurt het toevoegen van nieuwe gegevens en het bijwerken van bestaande gegevens in een tabel m.b.v. respectievelijk de SQL-instructies INSERT en UPDATE. In dit onderdeel behandelen we het toevoegen van gegevens.

De volgende code voegt een extra module Access toe in de tabel "modules", met een prijs van 85.0 euro.

```
<?php
class ModuleLijst {

    public function createModule($naam, $prijs) {

        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
                        "cursusgebruiker", "cursuspwd");

        $sql = "insert into modules (naam, prijs) values ('" .
                $naam . "', " . $prijs . ")";

        $dbh->exec($sql);
        $dbh = null;
    }

}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Modules</title>
    </head>
    <body>
        <h1>Module toevoegen</h1>
        <?php
            $mlijst = new ModuleLijst();
            $mlijst->createModule("Access", 85.0);
        ?>
    </body>
</html>
```

Zowel INSERT- als UPDATE-instructies worden uitgevoerd met de functie **`exec(sqlstatement)`**.

De functie **`lastInsertId()`** kan je direct na de **`exec()`** functie gebruiken om het ID van het laatste record te weten te komen, wanneer er (zoals in ons geval) autonummering gebruikt wordt in de tabel.

Bijvoorbeeld:

```
$dbh->exec($sql);
$laatsteId = $dbh->lastInsertId();
print($laatsteId);
```

Oefening 9.2: ☆☆☆

Ontwerp één pagina die alle films uit de gelijknamige tabel in lijstvorm toont, en die het tevens mogelijk maakt een nieuwe film toe te voegen.

Alle films

- Forrest Gump (142 min)
- Godfather, The (168 min)
- Green Mile, The (188 min)
- Lord of the Rings: The Return of the King, The (201 min)
- Once Upon a Time in the West (165 min)
- Pulp Fiction (154 min)
- Schindler's List (195 min)
- Se7en (127 min)
- Shawshank Redemption, The (142 min)

Film toevoegen

Titel:

Duurtijd: minuten

Extra uitbreidingen:

- Om een film toe te kunnen voegen moet de titel ingevuld zijn.
- De duurtijd moet numeriek en groter zijn dan 0.
- Zorg voor een gepaste foutmelding als het toevoegen mislukt is.

→ Oplossing: 913364

Gegevens verwijderen

Het verwijderen van bestaande gegevens uit een tabel gaat even eenvoudig als het toevoegen van nieuwe gegevens. Het enige bijkomende element is de noodzaak aan een parameter die ons toelaat het te verwijderen record uniek te identificeren.

Bestudeer onderstaande code:

```
<?php
class ModuleLijst {

    public function deleteModule($id) {

        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
                        "cursusgebruiker", "cursuspwd");
        $sql = "delete from modules where id = " . $id;
        $dbh->exec($sql);
        $dbh = null;
    }
}
```

```

}
?>
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Modules</title>
  </head>
  <body>
    <h1>Module verwijderen</h1>
    <?php
      $mlijst = new ModuleLijst();
      $mlijst->deleteModule(12);
    ?>
  </body>
</html>

```

Hierbij wordt de module met ID 12 gewist uit de tabel. Merk op dat er net zoals bij het INSERT-statement gebruik gemaakt wordt van de **exec()** functie.

Controleer na het uitvoeren van dit programma of de module met ID 12 inderdaad gewist is.

In de praktijk gebeurt het verwijderen van een element uit een zichtbare lijst via een hyperlink die zich bij elk element bevindt. De hyperlink bevat een extra parameter die het ID van het element voorstelt. Het verwerkingsscript kan dan op die manier het gewenste element (hier module) verwijderen.

Volgende code zouden we in de presentatielaag kunnen gebruiken bij het uitlijsten van de modules. Bestudeer aandachtig de structuur van de hyperlink.

We laten deze aanhalingsstekens voorafgaan door een backslash om te voorkomen dat de stringwaarde wordt afgesloten. Het gaat hier immers om aanhalingsstekens die we letterlijk willen opnemen als onderdeel van de string.

```

<ul>
  <?php
    foreach ($stab as $module) {
      print("<li>" . $module . " (<a href=\"vb40.php?action=verwijder&
        id=\" . $moduleId . "\">Verwijderen</a>) </li>");
    }
  ?>
</ul>

```

Het enige probleem dat hier opduikt is dat **\$moduleId**, dat het ID van de module zou moeten bevatten, helemaal geen waarde heeft. Het probleem is dat de lijst die wordt afgedrukt een lijst van stringwaarden is (modulenames), terwijl er van ID's geen sprake is.

Om dit probleem op te kunnen lossen introduceren we het gebruik van zgn. *entities*. Een entity is een klasse die functioneert als element waarin gegevens kunnen worden opgeslagen, terwijl haar functies zich grotendeels beperken tot getters en setters (om haar attributen te kunnen ophalen en wijzigen).

Het plan is tijdens het overlopen van de recordset (het resultaat van de SQL-SELECT-instructie) geen namen toe te voegen aan de array, maar objecten van

een zelf ontworpen klasse **Module**. Dit is een voorbeeld van een entity, en kan er bijvoorbeeld als volgt uitzien:

```
class Module {  
  
    private $id;  
    private $naam;  
    private $prijs;  
  
    public function __construct($id, $naam, $prijs) {  
        $this->id = $id;  
        $this->naam = $naam;  
        $this->prijs = $prijs;  
    }  
  
    public function getId() {  
        return $this->id;  
    }  
  
    public function getNaam() {  
        return $this->naam;  
    }  
  
    public function getPrijs() {  
        return $this->prijs;  
    }  
}
```

Merk op dat we minstens een getter voorzien voor elk veld uit de overeenkomstige tabel "modules".

We passen nu de klasse **ModuleLijst** aan, zodat er bij elke iteratie van de lus die de recordset overloopt een nieuw object aangemaakt wordt van de klasse **Module**, en toegevoegd wordt aan de array.

```
class ModuleLijst {  
  
    public function getLijst() {  
        $lijst = array();  
  
        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",  
                       "cursusgebruiker", "cursuspwd");  
        $sql = "select id, naam, prijs from modules order by naam";  
        $resultSet = $dbh->query($sql);  
        foreach ($resultSet as $rij) {  
            $module = new Module($rij["id"], $rij["naam"], $rij["prijs"]);  
            array_push($lijst, $module);  
        }  
        $dbh = null;  
        return $lijst;  
    }  
}
```

We gebruiken daarbij de waarden die we uit het record **\$rij** ophalen als parameters voor de constructor van **Module**, zodat deze attributen onmiddellijk ingevuld worden.

De regel **return \$lijst** geeft nu geen array van stringwaarden meer terug zoals voorheen het geval was, maar een array van objecten van de klasse **Module**.

We kunnen nu in de presentatielaag van dit voordeel gebruik maken, door via de getterfuncties van **Module** de attributen id, naam, enz... op te vragen.

```
<ul>
  <?php
  foreach ($stab as $module) {
    $moduleNaam = $module->getNaam();
    $moduleId = $module->getId();

    print("<li>" . $moduleNaam . " (<a href=\"vb40.php?action=verwijder&
      id=" . $moduleId . "\">Verwijderen</a>) </li>");
  }
  ?>
</ul>
```

Het enige dat ons nog te doen staat is ModuleLijst voorzien van een functie **verwijderModule** die een module kan verwijderen op basis van een meegegeven ID,...

```
public function deleteModule($id) {
    $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
        "cursusgebruiker", "cursuspwd");
    $sql = "delete from modules where id = " . $id;
    $dbh->exec($sql);
    $dbh = null;
}
```

...en ervoor zorgen dat het script deze functie aanroept op het ogenblik dat de GET-parameter action in de URL de waarde "verwijder" heeft:

```
if (isset($_GET["action"]) && $_GET["action"] == "verwijder") {
    $modLijst->deleteModule($_GET["id"]);
}
```

Op de volgende bladzijden vind je de volledige code van dit voorbeeld.

```
<?php
class Module {

    private $id;
    private $naam;
    private $prijs;

    public function __construct($id, $naam, $prijs) {
        $this->id = $id;
        $this->naam = $naam;
        $this->prijs = $prijs;
    }

    public function getId() {
        return $this->id;
    }

    public function getNaam() {
        return $this->naam;
    }
}
```

```

    }

    public function getPrijs() {
        return $this->prijs;
    }
}

class ModuleLijst {

    public function getLijst() {
        $lijst = array();

        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
                        "cursusgebruiker", "cursuspwd");
        $sql = "select id, naam, prijs from modules order by naam";
        $resultSet = $dbh->query($sql);
        foreach ($resultSet as $rij) {
            $module = new Module($rij["id"], $rij["naam"], $rij["prijs"]);
            array_push($lijst, $module);
        }
        $dbh = null;
        return $lijst;
    }

    public function deleteModule($id) {
        $dbh = new PDO("mysql:host=localhost;dbname=cursusphp",
                        "cursusgebruiker", "cursuspwd");
        $sql = "delete from modules where id = " . $id;
        $dbh->exec($sql);
        $dbh = null;
    }
}

$modLijst = new ModuleLijst();

if (isset($_GET["action"]) && $_GET["action"] == "verwijder") {
    $modLijst->deleteModule($_GET["id"]);
}
?>
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Modules</title>
    </head>
    <body>
        <h1>Modules</h1>
        <?php
            $tab = $modLijst->getLijst();
            ?>
            <ul>
                <?php
                    foreach ($tab as $module) {
                        $moduleNaam = $module->getNaam();
                        $moduleId = $module->getId();
                        print("<li>" . $moduleNaam . " (<a
                                                                    href=\"vb41.php?action=verwijder&id=" .
                                                                    $moduleId . "\">Verwijderen</a>) </li>");
                    }
                ?>
            </ul>

        </body>
    </html>

```

Oefening 9.3: ☆☆

Breid oefening 9.2 uit zodat je ook films kan verwijderen. Het verwijderen gebeurt door op een icoontje te klikken naast elke film. Gebruik hiervoor het bestand *delete.png* uit de oefenmap.

→ *Oplossing: 449631*

Gegevens bijwerken

Naast het toevoegen en verwijderen van gegevens is het dikwijls noodzakelijk dat je bestaande records in tabellen kan aanpassen. Zoals je reeds weet gebeurt dit op het laagste niveau met de SQL-instructie UPDATE. Deze instructie vereist dat we een sleutel meegeven a.d.h.v. dewelke het record dat gewijzigd dient te worden kan worden geïsoleerd van de andere records.

De manier waarop we te werk gaan lijkt goed op deze die gebruikt wordt bij het verwijderen van gegevens:

1. Toon een lijst van "alle" gegevens waaruit we één element kiezen. Zorg ervoor dat het ID van het element meegegeven wordt als parameter op de URL.
2. Toon de detailgegevens van dat ene specifieke element. Voorzie formulervelden voor elke eigenschap die moet kunnen worden aangepast. Voorzie tevens een knop "Opslaan".
3. Klikken op de knop verstuurt het formulier naar een verwerkingsscript. Samen met het formulier wordt ook het ID van het gewijzigde element meegegeven (meestal op de URL, als GET-parameter) .
4. Je voert een SQL UPDATE-instructie uit. Pas alle benodigde velden aan (via de ingevulde formulervelden) voor het record dat overeenkomt met het meegegeven ID.

We passen dit toe op ons voorbeeld van de modulelijst.

We beginnen met het afzonderen van onze entities (in dit geval enkel de entity Module). We maken een klasse **Module** en slaan deze op in een bestand *module.class.php* (de aanduiding "class" hoeft niet, maar het blijft op deze manier wel te allen tijde duidelijk dat dit geen gewone PHP-pagina, maar een klassedefinitie betreft).

```
<?php
class Module {

    private $id;
    private $naam;
    private $prijs;

    public function __construct($id, $naam, $prijs) {
        $this->id = $id;
        $this->naam = $naam;
        $this->prijs = $prijs;
    }

    public function getId() {
        return $this->id;
    }

    public function getNaam() {
        return $this->naam;
    }

    public function getPrijs() {
        return $this->prijs;
    }

    public function setNaam($naam) {
        $this->naam = $naam;
    }

    public function setPrijs($prijs) {
        $this->prijs = $prijs;
    }
}
```

Merk op dat we dit keer onze klasse naast getters ook voorzien van setters.

Vervolgens ontwerpen we in een bestand *moduleLijst.class.php* de klasse **ModuleLijst**, die de functies bevat die zullen worden uitgevoerd op de databank.

```
<?php
require_once("module.class.php");
class ModuleLijst {

    private $dbConn;
    private $dbUsername;
    private $dbPassword;

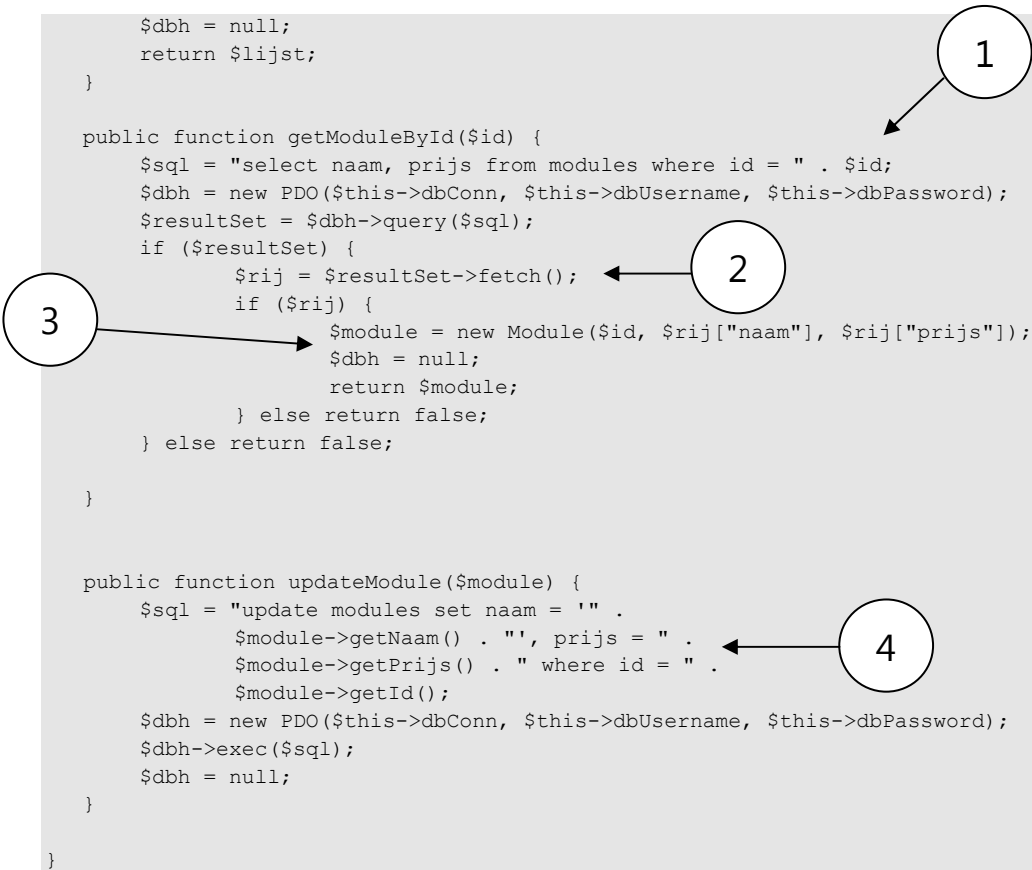
    public function __construct() {
        $this->dbConn = "mysql:host=localhost;dbname=cursusphp";
        $this->dbUsername = "cursusgebruiker";
        $this->dbPassword = "cursuspwd";
    }

    public function getLijst() {
        $lijst = array();
        $sql = "select id, naam, prijs from modules order by naam";
        $dbh = new PDO($this->dbConn, $this->dbUsername, $this->dbPassword);
        $resultSet = $dbh->query($sql);
        foreach ($resultSet as $rij) {
            $module = new Module($rij["id"], $rij["naam"], $rij["prijs"]);
            array_push($lijst, $module);
        }
    }
}
```

```
$dbh = null;
return $lijst;
}

public function getModuleById($id) {
    $sql = "select naam, prijs from modules where id = " . $id;
    $dbh = new PDO($this->dbConn, $this->dbUsername, $this->dbPassword);
    $resultSet = $dbh->query($sql);
    if ($resultSet) {
        $rij = $resultSet->fetch();
        if ($rij) {
            $module = new Module($id, $rij["naam"], $rij["prijs"]);
            $dbh = null;
            return $module;
        } else return false;
    } else return false;
}

public function updateModule($module) {
    $sql = "update modules set naam = '" .
        $module->getNaam() . "', prijs = '" .
        $module->getPrijs() . "' where id = " .
        $module->getId();
    $dbh = new PDO($this->dbConn, $this->dbUsername, $this->dbPassword);
    $dbh->exec($sql);
    $dbh = null;
}
}
```



Zoals je kan zien hebben we dit keer geopteerd om de gegevens die nodig zijn om verbinding te maken met de databank onder te brengen als attributen van de klasse (**\$dbConn**, **\$dbUsername** en **\$dbPassword**). In de constructor krijgen deze variabelen hun waarde toegekend. Op die manier hoeven we deze gegevens niet meer te herhalen in elke functie die verbinding moet maken met een databank.

De functie **getLijst()** kenden we al, de functie **getModuleById(\$id)** is nieuw. De functie voert een SELECT-instructie uit op basis van het meegegeven ID ❶. Van dit resultaat wordt het eerste record opgehaald (met de functie **fetch()**) ❷, en omgevormd tot object van de klasse **Module** ❸. Dit object wordt teruggegeven.

De functie **updateModule(\$module)** heeft één parameter: een object van de klasse **Module**. De getters van dit object laten ons toe de SQL-opdrachtstring te vervolledigen ❹. Merk op dat we een update uitvoeren op alle velden van het record, behalve het ID. Indien deze waarden niet gewijzigd werden worden ze gewoon met dezelfde waarde overschreven.

De benodigde klassen zijn nu gemaakt. We leggen ons nu toe op de zichtbare PHP-pagina's. We ontwerpen eerst de pagina *overzicht.php*:


```

<?php
require_once("module.class.php");
require_once("modulelijst.class.php");
?>
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Modules</title>
  </head>
  <body>
    <h1>Modules</h1>
    <?php
    $modLijst = new ModuleLijst();
    $tab = $modLijst->getLijst();
    ?>
    <ul>

        <?php
        foreach ($tab as $module) {
            $moduleNaam = $module->getNaam();
            $moduleId = $module->getId();

            print("<li>" . $moduleNaam . "
                (<a href=\"modulebewerken.php?id=
                . $moduleId . \">Bewerken</a>) </li>");

        }
        ?>

    </ul>

  </body>
</html>

```

Let op de twee **require_once** functie-aanroepen. Ze zorgen ervoor dat we in onze pagina zowel van de klasse **Module** als van de klasse **ModuleLijst** gebruik kunnen maken. Van elke module wordt een hyperlink gemaakt. Deze hyperlink bevat een vermelding van het ID van de module (de pagina die de details van module zal tonen en klaar zal zetten voor wijziging zal immers moeten weten welke module we willen aanpassen).

Het verwerkingsscript zelf wordt ondergebracht in een ander bestand *modulebewerken.php*, en kan er bijvoorbeeld als volgt uitzien:

```

<?php
require_once("module.class.php");
require_once("modulelijst.class.php");

if ($_GET["action"] == "verwerk") {
    $module = new Module($_GET["id"], $_POST["naam"], $_POST["prijs"]);
    $modLijst = new ModuleLijst();
    $modLijst->updateModule($module);
    $updated = true;
}
?>

```

```

<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Modules</title>
  </head>
  <body>
    <h1>Module bewerken</h1>
    <?php
      if ($updated) print("Record bijgewerkt!");
      $modLijst = new ModuleLijst();
      $module = $modLijst->getModuleById($_GET["id"]); ← 3
    ?>

    <form action="modulebewerken.php?action=verwerk&id=
      <?php print($_GET["id"]); ?>" method="post">

      Naam:
      <input type="text" name="naam" value="
        <?php print($module->getNaam()); ?>"><br><br>
      Prijs:
      <input type="text" name="prijs" value="
        <?php print($module->getPrijs()); ?>"> euro<br>
      <input type="submit" value="Opslaan">

    </form>
    <br>
    <a href="overzicht.php">Terug naar overzicht</a>

  </body>
</html>

```

Merk opnieuw de twee **require_once** functieaanroepen op ❶. We gebruiken in dit voorbeeld dezelfde pagina om enerzijds het formulier te tonen dat ons toelaat de gegevens van de module aan te passen, en anderzijds de eigenlijke update (de verwerking) te doen. We gaan daarom eerst na ❷ in welk van de twee scenario's we ons bevinden. We willen vervolgens de huidige waarden van de eigenschappen (naam en prijs) van de module kunnen tonen in formulierform. Vermits we het ID van de aan te passen module kennen (die hebben we meegekregen via de URL (`$_GET["id"]`), en we via de klasse **ModuleLijst** toegang hebben tot een functie **getModuleById(*id*)** wordt dit een fluitje van een cent ❸.

Tijd voor het formulier nu. Het formulier bevat invulvelden die reeds voorzien zijn van een standaardwaarde: de waarde van de eigenschappen van de module zoals ze op dit ogenblik zijn. We maken hiervoor gebruik van het attribuut **value** van de INPUT tag ❹.

Tip: Het helpt ook hier weer om de PHP-code te markeren, om ze makkelijk te onderscheiden van HTML-code.

NOOT: Wanneer een formulier onvolledig of oncorrect ingevuld werd, word je na verzenden meestal teruggestuurd naar de formulierpagina, waarbij de waarden die je ingevuld had reeds opnieuw ingevuld staan, en wordt netjes

aangeduid welke velden fouten bevatten. We gaan verder in op deze strategie in het hoofdstuk over MVC (Model-View-Controller)

Oefening 9.4: ★★

Breid oefening 9.3 uit zodat je de eigenschappen van films ook kunt aanpassen. Het klikken op een hyperlink brengt je naar een invulformulier dat je toelaat de titel en duurtijd van de film aan te passen.

→ Oplossing: 964873

Oefening 9.5: ★★

Ontwerp een rudimentair gastenboek. Het moet mogelijk zijn om de berichten te lezen en een nieuw bericht toe te voegen. Voor elk bericht moet verplicht een auteur en boodschap ingevuld worden.

Berichten

Auteur: Admin
Het werkt inderdaad.

Auteur: Bezoeker
Even testen of het gastenboek werkt...

Bericht toevoegen

Auteur:

Boodschap:

Extra's:

- Zorg ervoor dat de boodschap maximaal 200 karakters lang kan zijn
- Zorg ervoor dat de berichten aflopend volgens datum gesorteerd getoond worden.

→ Oplossing: 745996

Oefening 9.6: ★★★

Ontwerp een eenvoudig chatprogramma. Wanneer een deelnemer voor het eerst de pagina opvraagt krijgt hij/zij een zgn. nickname toegewezen onder de vorm `p<willekeurig nummer tussen 111 en 999>`, bvb `p437`, `p755` enz... Deze naam blijft behouden zolang de sessie loopt. Wanneer de browser wordt afgesloten of de cookies worden verwijderd, wordt bij het eerstvolgende bezoek een nieuwe nickname gegenereerd.

Alle berichten die reeds toegevoegd werden worden getoond, met het meest recente bericht bovenaan.

Eenzelfde nickname kan per toeval tweemaal gegenereerd worden. Je hoeft hier echter geen rekening mee te houden.

Extra's:

- Zorg ervoor dat de nickname niet gegenereerd wordt, maar dat de bezoeker bij zijn eerste bezoek een eigen nickname kan ingeven. Vanaf dan wordt deze nickname gebruikt, zolang de sessie loopt.
- Zorg ervoor dat enkel de 20 meest recente berichten worden getoond.

→ *Oplissing: 944268*

Oefening 9.7: ★★★

Ontwerp een versie van het spel Vier-Op-Een-Rij. In de oefenmap vind je de bestanden *emptyslot.png*, *yellowslot.png* en *redslot.png* terug. Het speelbord bevat 6 rijen en 7 kolommen. Initieel zijn alle vakjes leeg (emptyslot). Twee spelers proberen beurtelings vier van hun munten op één rij te plaatsen. Het spel dient gespeeld te worden over het netwerk.

Initieel surfen beide spelers naar de pagina *kleurkeuze.php*. Hier wordt hen gevraagd een kleur te kiezen. Beide spelers dienen uiteraard de tegenovergestelde kleur te kiezen, maar ter vereenvoudiging hoeft dit niet gecontroleerd te worden.

Vier op een Rij

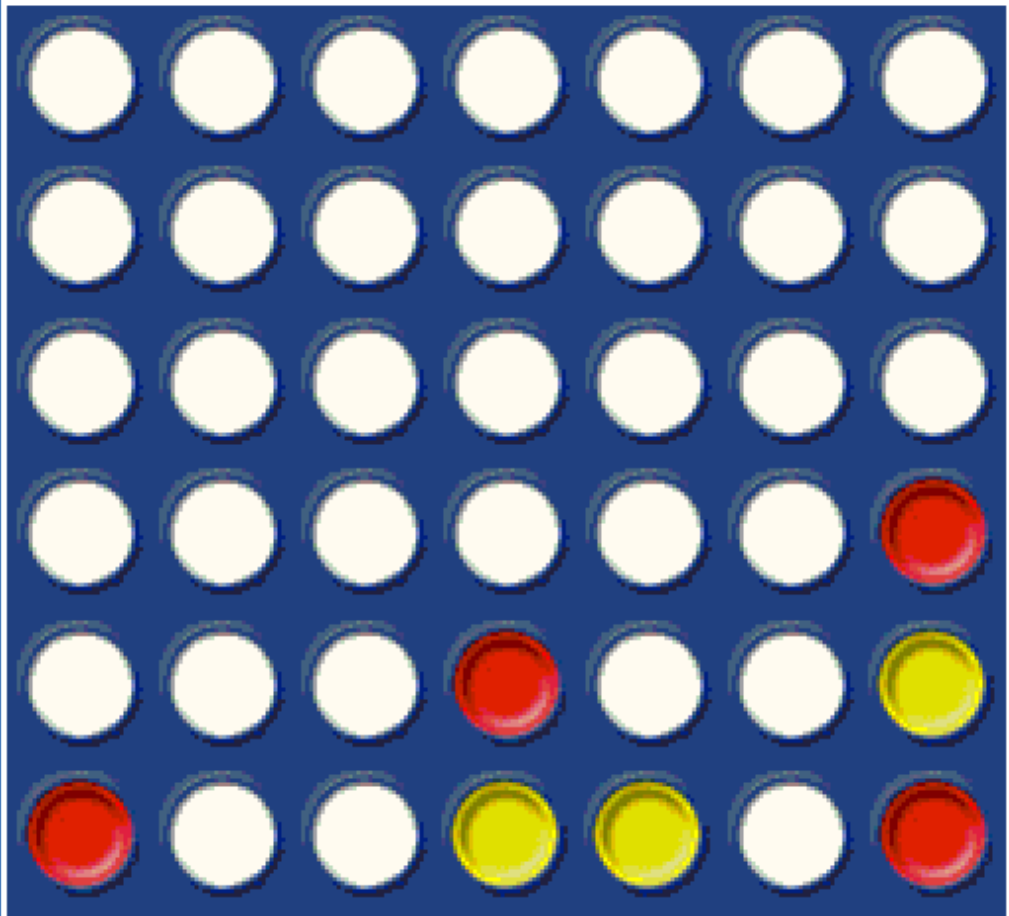
Kies de kleur waar je mee wilt spelen

- [Klik hier om met geel te spelen](#)
- [Klik hier om met rood te spelen](#)

Een klik op één van de links brengt je naar het bestand *spelen.php*, waarop het eigenlijke speelbord te zien is.

Maak gebruik van de tabel "vieropeenrij_spelbord". Deze tabel bevat 3 velden: een rijnummer, kolomnummer en een statusnummer. De positie linksbovenaan het bord heeft rijnummer 1 en kolomnummer 1. De positie rechtsonderaan heeft rijnummer 6 en kolomnummer 7. Status 0 betekent dat er zich geen munt op die plaats bevindt. Status 1 staat voor een gele munt en status 2 voor een rode.

Vier op een Rij



[Vernieuw bord](#)

[Spel herstarten](#)

Het plaatsen van een munt gebeurt door het klikken op een willekeurig vakje. Of dit vakje reeds een munt bevat of niet speelt geen rol, enkel de kolom waarin het aangeklikte vakje zich bevindt is van belang. In deze kolom wordt een munt gedeponeerd, die zo ver mogelijk naar beneden "valt".

Het spel moet enkel munten kunnen plaatsen in vakjes. Het controleren of er vier-op-een-rij is gevormd zal door de spelers zelf gebeuren, niet door het programma. Er hoeft dus geen melding gemaakt te worden wanneer een speler gewonnen is. Het is tevens onbelangrijk om na te gaan wiens beurt het is. Beide spelers werpen om beurten een munt in, en wachten daarna zelfstandig op een zet van de tegenstander.

Een klik op de link "Spel herstarten" maakt het bord leeg.

→ Oplossing: 464982

Hoofdstuk 10

Meerlagenarchitectuur

In dit hoofdstuk:

- ✓ Het uitwerken van een oefening met MVC (Model-View-Controller)

Een sprong in het diepe

We hebben reeds uitgebreid kennisgemaakt met het splitsen van applicaties in twee lagen: een presentatielaag en een logica laag. In dit hoofdstuk werken we deze splitsing nog verder uit.

We steken van wal met een voorbeeld. We maken een applicatie die eenvoudigweg de namen van een aantal personen op het scherm zet, in lijstvorm. De opsplitsing die we maken zal op het eerste zicht misschien overdreven lijken, maar toch zal deze manier van werken zichzelf erg snel “terugbetalen” naar onderhoudbaarheid en overzichtelijkheid toe.

Bestudeer elk stuk code aandachtig en neem het over in de genoemde bestanden, zodat je op het einde de goede werking kan uittesten.

Stap 1: De entities

Maak een subfolder aan met de naam */entities*.

In ons voorbeeld hebben we enkel een entity **Persoon**. Deze zal per persoon alle benodigde informatie bevatten, toegankelijk via getters en setters. Dit is een erg eenvoudige klasse, die we opslaan in een bestand *persoon.class.php* in de folder */entities*.

```
<?php
class Persoon {

    private $familienaam;
    private $voornaam;

    public function __construct($familienaam, $voornaam) {
        $this->setFamilienaam($familienaam);
        $this->setVoornaam($voornaam);
    }

    public function getFamilienaam() {
        return $this->familienaam;
    }

    public function getVoornaam() {
        return $this->voornaam;
    }

    public function setFamilienaam($familienaam) {
        $this->familienaam = $familienaam;
    }

    public function setVoornaam($voornaam) {
        $this->voornaam = $voornaam;
    }

}
```

Stap 2: De data laag

De data laag bevat alle klassen die verantwoordelijk zijn voor het ophalen van gegevens uit externe bronnen (gegevensbanken, bestanden, etc...) en het omvormen van deze gegevens tot entity-objecten.

In de praktijk komen deze gegevens vanzelfsprekend uit een databank. Maar voorlopig stellen we ons tevreden met het hardcoderen van een aantal Persoon-objecten in een array, dit teneinde ons voorbeeld niet te complex te maken.

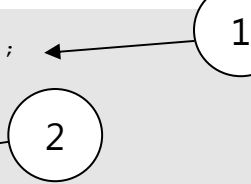
We maken een subfolder met de naam `/data`, waarin we klasse **PersoonDAO** onderbrengen. DAO staat voor **Data Access Object**.

```
<?php
require_once("entities/persoon.class.php");

class PersoonDAO {

    public static function getAll() {
        $lijst = array();
        array_push($lijst, new Persoon("Peeters", "Bram"));
        array_push($lijst, new Persoon("Van Dessel", "Rudy"));
        array_push($lijst, new Persoon("Vereecken", "Marie"));
        array_push($lijst, new Persoon("Maes", "Eveline"));
        return $lijst;
    }

}
```



Merk op dat deze klasse uiteraard toegang moet hebben tot de klasse **Persoon** ❶ (om er entity objecten van te kunnen maken).

De functie **getAll()** ❷ is verantwoordelijk – zoals de naam zelf impliceert – voor het ophalen van alle personen.

De aandachtige lezer zal gezien hebben dat we de functie **getAll()** statisch gemaakt hebben. Vanaf nu zullen we dit toepassen op alle functies van zowel de DAO-klassen als de service-klassen. Het is namelijk niet de bedoeling dat we telkens objecten aanmaken van van dit soort klassen. Ze hebben immers geen eigenschappen (attributen). Enkel hun gedrag (de functies) interesseert ons. We maken ze daarom **static**, zodat slechts één keer geheugenruimte wordt vrijgemaakt.

Stap 3: De businesslaag

Nog een niveau hoger bevindt zich de businesslaag of servicelaag. Regel is: één functie in de businesslaag komt overeen met één "taak". De taak in kwestie is hier het ophalen van alle personen, om ze later te kunnen tonen op het scherm. Let wel dat het tonen zelf NIET de verantwoordelijkheid is van de deze laag, die eer komt de presentatielaag toe (zie verder).

Maak een klasse **PersoonService** in een bestand *persoonservice.class.php*, en plaats deze in een submap met de naam */business*.

Let op! Ook in de serviceklassen maken we de functies statisch.

```
<?php
require_once("data/persoondao.class.php");

class PersoonService {

    public static function getAllPersonen() {
        $personen = PersoonDAO::getAll();
        return $personen;
    }

}
```

3

Een klasse in de servicelaag, zoals deze, spreekt elke DAO-klasse aan die hij nodig heeft om zijn taak te kunnen afwerken. In dit geval is enkel de **PersoonDAO** klasse noodzakelijk. Het antwoord (de return-waarde) van de DAO is in dit voorbeeld gelijk aan het antwoord van de serviceklasse zelf ❸.

Belangrijke opmerking: de serviceklasse weet dus niet waar de gegevens waar hij om vraagt vandaan komen (een databank, een bestand, Active Directory,...). Ze hoeft dit echter ook niet te weten om haar opdracht te kunnen invullen: alle personen ophalen.

Stap 4: De presentatielaag

De presentatielaag bevat alle bestanden die verantwoordelijk zijn voor hetgeen de gebruiker uiteindelijk op het scherm te zien krijgt. Deze bestanden bevatten zo weinig mogelijk PHP code. Er worden dus ook geen objecten meer gemaakt – in tegenstelling tot wat we tot nu toe over het algemeen wel deden. We gaan er simpelweg van uit dat alle objecten waarvan we de inhoud willen tonen reeds bestaan (ze zullen worden “klaargezet” door de controller, zie verder).

We maken in een submap */presentation* een bestand *personenlijst.php* aan.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Hello world</title>
  </head>
  <body>
    <h1>Personenlijst</h1>
    <ul>
      <?php
        foreach ($personen as $persoon) {
          ?>
          <li>
            <?php print($persoon->getFamilienaam() .
              ", " . $persoon->getVoornaam());?>
          </li>
        <?php
        }
      ?>
    </ul>
  </body>
</html>
```

Markeer de PHP-code in dit bestand. Er is enkel PHP-code voorzien voor het overlopen van de lijst met een foreach-lus, en om de inhoud van de attributen (familienaam en voornaam) af te drukken. De rest is HTML.

Stap 5: De controller

De logicalaag bestaat, net als de presentatielaag. Nu is er enkel nog een element nodig dat deze twee aan elkaar koppelt. Deze taak is weggelegd voor de controller.

Bedoeling is dat de bezoeker enkel nog surft naar controllers, vermits zij het ingangspunt van de applicatie zijn. We maken (niet in een subfolder, maar in de root van de applicatie) een bestand *toonallepersonen.php* aan.

Nota m.b.t. de
functie include()
op de volgende
pagina !

```
<?php
require_once("business/persoonservice.class.php");
$personen = PersoonService::getAllePersonen();
include("presentation/personenlijst.php");
```

Dit is exact het bestand waar de bezoeker naartoe surft, en is dus het ingangspunt van de applicatie. De controller spreekt de nodige serviceklassen aan, in dit geval `PersoonService`, en steekt het resultaat van de functie `getAllePersonen` in een variabele `$personen`. Deze variabele staat nu "klaar" om ingelezen te worden door de presentatielaag. De presentatielaag wordt m.b.v. de functie **`include(bestand)`** ingevoegd, en de bezoeker krijgt alle personen te zien.

Test deze applicatie nu uit.

Wellicht heb je gemerkt dat we voor het importeren van de presentatiepagina voor het eerst de functie **`include()`** gebruikt hebben, i.p.v. de gebruikelijke **`require_once()`**. Daarnaast bestaan er ook de functies **`include_once()`** en **`require()`**.

Het verschil tussen **`include()`** en **`require()`** ontstaat wanneer het gerefereerde bestand niet gevonden wordt. In zo'n geval genereert de functie **`include()`** een waarschuwing en gaat vervolgens gewoon door met de uitvoer van het script. De functie **`require()`** genereert een "fatal error" en stopt het script onmiddellijk, waarbij alle daaropvolgende regels genegeerd worden.

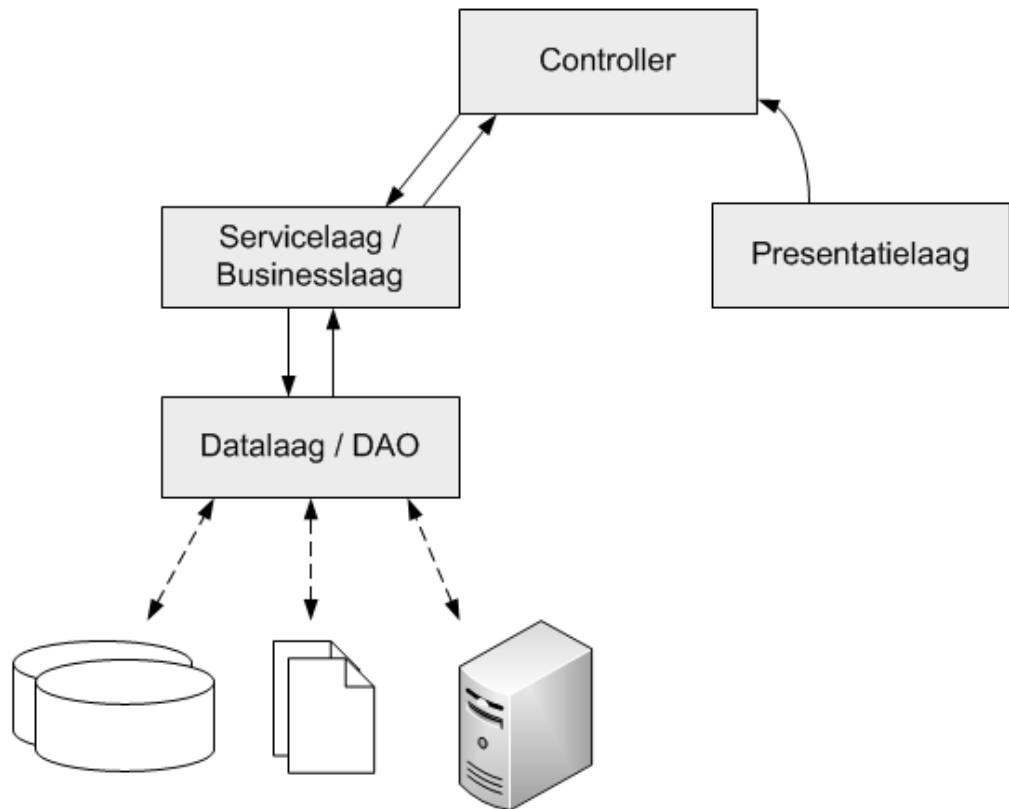
Beide functies hebben ook een **`_once`**-versie. Bij gebruik hiervan wordt er vooraf een controle uitgevoerd of het gerefereerde bestand al niet een keer was geïmporteerd op de pagina. Zo ja, dan wordt de functie-aanroep genegeerd.

Het Model-View-Controller design pattern

Wat we tot hertoe beschreven hebben is eigenlijk niets meer dan een implementatie van het befaamde MVC (Model-View-Controller) design pattern.

Vrijwel elke webapplicatie is relatief eenvoudig uit te breiden en te onderhouden wanneer ze op dit design pattern geënt is.

De data laag haalt gegevens op, schrijft ze weg of verwijdert ze. Deze laag bevat meestal één DAO-klasse per entity. Deze klasse bevat dan een functie per CRUD-operatie op de gegevensbron (Create Read Update Delete).



De servicelaag of businesslaag bevat doorgaans een serviceklasse per entity. Deze klasse bevat dan alle taken die uitgevoerd moeten worden en betrekking hebben op die specifieke entity. Een serviceklasse bepaalt per taak welke DAO's er aangesproken dienen te worden om de taak uit te kunnen voeren.

De controller is het ingangspunt van de applicatie en er zal enkel gesurft worden naar controller-pagina's. Ook formulieren worden enkel verstuurd naar een controller. De controller bekijkt de aanvraag en beslist welke serviceklasse(n) moet(en) worden aangesproken. De controller bepaalt daarna tevens welke presentatiepagina('s) moet(en) worden geïmporteerd.

Merk op dat dit hele proces volledig transparant blijft naar de bezoeker.

Wanneer je een webapplicatie volgens het MVC-model wilt schrijven is de volgende volgorde over het algemeen de meest eenvoudige:

1. Maak een directorystructuur aan. De root van de applicatie bevat minstens de volgende mappen:
 - */business*
 - */data*
 - */entities*
 - */presentation*

2. Maak in */entities* de nodige entity-klassen aan (meestal één per tabel in de databank), bvb **Boek**.
3. Maak in */data* de nodige DAO-klassen aan, minstens één per entity, bvb **BoekDAO**.
4. Maak in */business* de nodige serviceklassen, minstens één per entity, bvb **BoekService**.
5. Ontwerp in */presentation* de pagina's die betrekking hebben op wat de gebruiker uiteindelijk te zien krijgt. Dit zijn pagina's die zeer weinig PHP-code bevatten, maar daarentegen wel HTML, CSS, Javascript,...
6. Schrijf per applicatietaak een controllerpagina, bvb *toonalleboeken.php*. Plaats de controllers in de root van de applicatie.

Uitbreiding met een gegevensbank

In het voorbeeld dat we vooraf geïntroduceerd hadden werd de lijst van personen hardgecodeerd in de DAO-klasse. In de praktijk zal je uiteraard een gegevensbank willen gebruiken.

Hier steekt onmiddellijk één van de grote voordelen van MVC de kop op. De enige laag die zaken heeft met de manier waarop gegevens worden opgeslagen is de data laag. Dat betekent dat wanneer we onze applicatie gebruik willen laten maken van een databank, we enkel de data laag (m.a.w. de DAO-klassen) moeten aanpassen.

We werpen eerst nog eens een blik op hoe onze **PersoonDAO** klasse er momenteel uitziet.

```
<?php
require_once("entities/persoon.class.php");

class PersoonDAO {

    public static function getAll() {
        $lijst = array();
        array_push($lijst, new Persoon("Peeters", "Bram"));
        array_push($lijst, new Persoon("Van Dessel", "Rudy"));
        array_push($lijst, new Persoon("Vereecken", "Marie"));
        array_push($lijst, new Persoon("Maes", "Eveline"));
        return $lijst;
    }
}
```

We moeten enkel de inhoud van de functie **getAll()** wijzigen, zodat een databank wordt aangesproken. Aan de header van de functie wijzigt niets.

Voor de serviceklasse **PersoonService** wijzigt er niets, deze spreekt nog steeds de functie **getAll()** van **PersoonDAO** aan.

Vermits een applicatie meestal uit meer dan één DAO-klasse bestaat is het verstandig om de gegevens die betrekking hebben op de databankverbinding (locatie, gebruikersnaam, wachtwoord,...) onder te brengen in een aparte klasse **DBConfig**. We gebruiken publieke statische variabelen. We hoeven immers niet telkens een object te maken van **DBConfig** om de waarde van deze variabelen te kunnen lezen.

```
<?php
class DBConfig {

    public static $DB_CONNSTRING = "mysql:host=localhost;dbname=cursusphp";
    public static $DB_USERNAME = "cursusgebruiker";
    public static $DB_PASSWORD = "cursuspwd";

}
```

Je kan nu overal naar deze variabelen verwijzen door gebruik te maken van **DBConfig::\$DB_CONNSTRING**, **DBConfig::\$DB_USERNAME** en **DBConfig::\$DB_PASSWORD**.

Rest ons nu alleen nog de DAO-klasse aan te passen, zodat die verbinding maakt met een gegevensbank, gebruik makend van bovengenoemde variabelen en de records omvormt tot objecten van de klasse **Persoon**.

Op de volgende bladzijde vind je de nieuwe code voor onze **PersoonDAO** klasse.

```
<?php
require_once("entities/persoon.class.php");
require_once("dbconfig.class.php");

class PersoonDAO {

    public static function getAll() {
        $lijst = array();
        $sql = "select familienaam, voornaam from personen";

        $dbh = new PDO(DBConfig::$DB_CONNSTRING,
            DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);
        $resultSet = $dbh->query($sql);
        foreach ($resultSet as $rij) {
            $persoon = new Persoon($rij["familienaam"], $rij["voornaam"]);
            array_push($lijst, $persoon);
        }
        $dbh = null;
        return $lijst;
    }

}
```

Test de applicatie opnieuw uit.

Van nul...

Bij wijze van oefening maken we eens een applicatie die twee tabellen gebruikt. Bedoeling is deze volgens het MVC design pattern te schrijven, waarbij het mogelijk is om boeken op te halen, weg te schrijven, te verwijderen en bij te werken.

We starten met het ontwerp van de tabellen. Maak deze ineens aan (geef de tabellen de namen *mvc_boeken* en *mvc_genres*). We houden per tabel niet te veel attributen bij, om onze applicatie niet te complex te maken.

mvc_boeken
<u>id</u> (INTEGER)
titel (VARCHAR 100)
genreid (INTEGER)

mvc_genres
<u>id</u> (INTEGER)
omschrijving (VARCHAR 40)

Voeg zelf handmatig enkele genres en boeken toe. Bedoeling zal zijn dat we via de applicatie geen extra genres kunnen toevoegen, maar wel boeken. Voorzie daarom zelf een voldoende aantal genres. Denk eraan dat de velden id van beide tabellen een autonummeringveld (en primary key) zijn.

We maken alvast een mappenstructuur op, die zich bevindt in de root van de applicatie:

- */entities*
- */data*
- */business*
- */presentation*

en we gaan van start.

Stap 1: De entities

De entities zijn op zich het eenvoudigst: maak één getter/setter per veld in de tabel. We zullen echter onze entity klassen iets anders moeten schrijven dan we tot nu toe gedaan hebben. Er is immers sprake van een JOIN tussen twee tabellen (**genreid** in *mvc_boeken* is een verwijzing naar **id** in *mvc_genres*). Dat betekent dat we voor elk **Boek**-object tevens een object van een klasse **Genre** zullen moeten bijhouden.

Het probleem is echter dat we niet zomaar per boek een nieuw **Genre**-object mogen aanmaken. Stel bijvoorbeeld dat we een boek ophalen uit de tabel *mvc_boeken* dat een genre heeft waar we in een vorige iteratie al eens een

object van aangemaakt hebben. Dan is het belangrijk dat er geen nieuw **Genre**-object wordt aangemaakt, maar dat het object gebruikt wordt dat voordien reeds aangemaakt werd.

Dat betekent dat we ergens per klasse lijsten zullen moeten bijhouden van objecten die reeds van de klasse in kwestie aangemaakt werden, zodat we er kunnen naar verwijzen wanneer nodig.

Bestudeer onderstaande code voor de klasse **Genre** aandachtig.

```
<?php
class Genre {

    private static $idMap = array();

    private $id;
    private $omschrijving;

    private function __construct($id, $omschrijving) {
        $this->id = $id;
        $this->omschrijving = $omschrijving;
    }

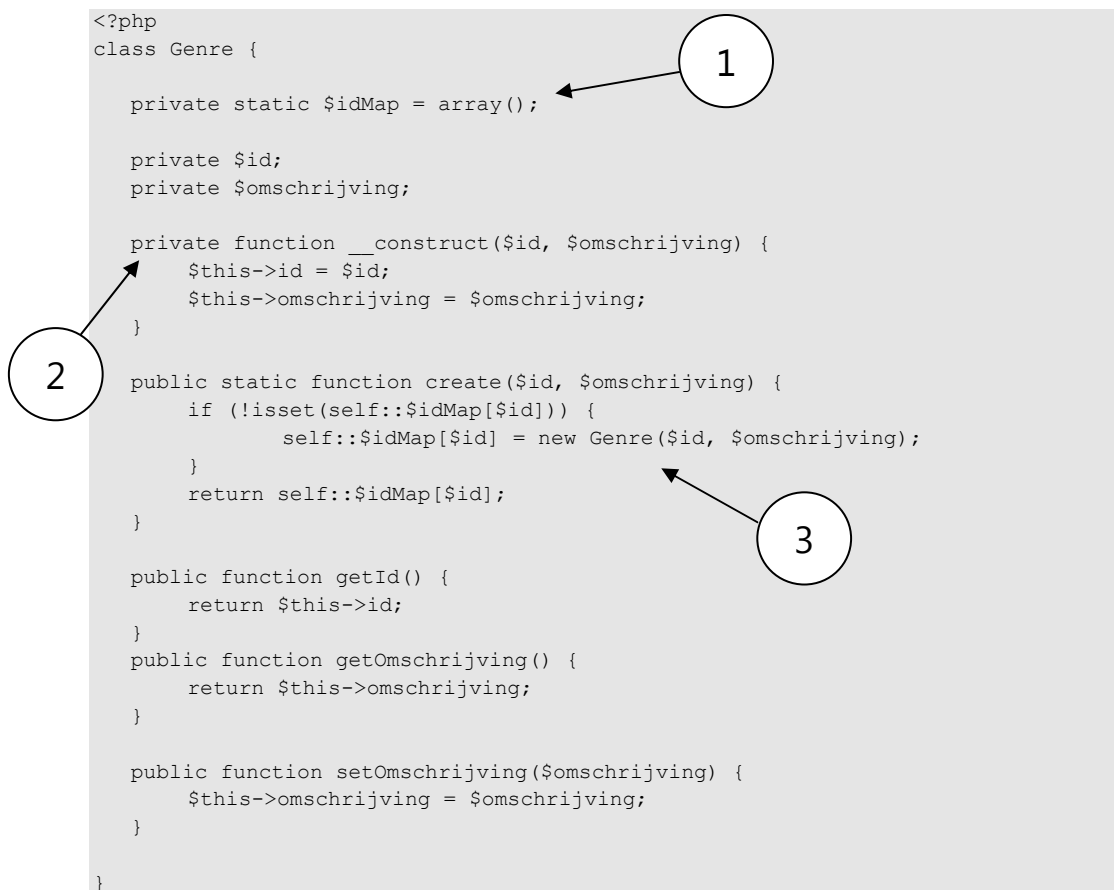
    public static function create($id, $omschrijving) {
        if (!isset(self::$idMap[$id])) {
            self::$idMap[$id] = new Genre($id, $omschrijving);
        }
        return self::$idMap[$id];
    }

    public function getId() {
        return $this->id;
    }

    public function getOmschrijving() {
        return $this->omschrijving;
    }

    public function setOmschrijving($omschrijving) {
        $this->omschrijving = $omschrijving;
    }

}
```



We maken een privaats attribuut **\$idMap** ❶. Dit wordt de array die alle reeds aangemaakte objecten van de klasse **Genre** bevat. De array wordt geïndexeerd met het id van het **Genre**-object. Op die manier kunnen we in één stap controleren of er een **Genre**-object met een bepaald id werd aangemaakt, zonder de hele array te moeten overlopen. Merk op dat **\$idMap** een static attribuut is. Er wordt dus (zoals uiteraard de bedoeling is) slechts één lijst aangemaakt, voor alle **Genre**-objecten.

We maken de constructor private ❷! Vanaf nu kunnen we dus van buitenaf geen objecten meer aanmaken van de klasse **Genre**. Het zal enkel nog mogelijk zijn om dit te doen vanuit de klasse **Genre** zelf.

Een nieuw object aanmaken doen we i.p.v. via de constructor, via de functie **create(parameters)** ③. We geven dezelfde parameters mee aan deze functie als we aan de constructor zouden meegeven. De functie controleert simpelweg of onze objectenlijst **\$idMap** reeds een **Genre**-object bevat die hetzelfde id heeft. Zo neen, dan wordt een nieuw **Genre**-object aangemaakt (m.b.v. de constructor) en aan de lijst toegevoegd. Zo ja, dan wordt het reeds bestaande object teruggegeven.

Doorheen onze applicatie zullen we dus nooit meer

```
$obj = new Genre(1, "Avontuur");
```

schrijven, maar wel

```
$obj = Genre::create(1, "Avontuur");
```

Een kleine aanpassing dus, maar wel noodzakelijk, vermits we van buitenaf de constructor van een entity niet meer zullen kunnen aanspreken.

Tijd voor de implementatie van de klasse **Boek**, die volledig analoog is aan die van de klasse **Genre**.

```
<?php
class Boek {

    private static $idMap = array();

    private $id;
    private $titel;
    private $genre;

    private function __construct($id, $titel, $genre) {
        $this->id = $id;
        $this->titel = $titel;
        $this->genre = $genre;
    }

    public static function create($id, $titel, $genre) {
        if (!isset(self::$idMap[$id])) {
            self::$idMap[$id] = new Boek($id, $titel, $genre);
        }
        return self::$idMap[$id];
    }

    public function getId() { return $this->id; }
    public function getTitel() { return $this->titel; }
    public function getGenre() { return $this->genre; }

    public function setTitel($titel) { $this->titel = $titel; }
    public function setGenre($genre) { $this->genre = $genre; }
}
```

Merk op dat we een attribuut **\$genre** bijhouden, en geen **\$genreid**. **\$genre** zal nl. het echte **Genre**-object bevatten i.p.v. zomaar een nummer.

Onze entities zijn gemaakt, tijd voor de volgende stap.

Stap 2: De datalaag

De datalaag bevat de code die voor de verbinding zorgt met de databank, en de records in de tabellen omvormt naar **Boek**- en **Genre**-objecten.

Bestudeer de uitgewerkte code voor de klasse **BoekDAO**.

```
<?php
require_once("data/dbconfig.class.php");
require_once("entities/genre.class.php");
require_once("entities/boek.class.php");

class BoekDAO {

    public static function getAll() {
        $lijst = array();

        $dbh = new PDO(DBConfig::$DB_CONNSTRING,
                        DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);

        $sql = "select mvc_boeken.id as boekid, titel,
                    genreid, omschrijving from mvc_boeken,
                    mvc_genres where genreid =
                    mvc_genres.id";

        $resultSet = $dbh->query($sql);
        foreach ($resultSet as $rij) {
            $genre = Genre::create($rij["genreid"],
                                   $rij["omschrijving"]);

            $boek = Boek::create($rij["boekid"],
                                 $rij["titel"], $genre);

            array_push($lijst, $boek);
        }
        $dbh = null;
        return $lijst;
    }
}
```



De functie **getAll()** haalt alle boeken op uit de tabel. Zoals je ziet worden naast de gegevens specifiek voor boeken ook per boek de gegevens specifiek voor genres opgehaald. Dit wordt bekomen met een JOIN instructie.

We gebruiken de functie **create(parameters)** van **Genre** om een nieuw genre te maken ❶. Bedoeling is natuurlijk dat, als dit genre reeds werd aangemaakt in een vorige iteratie, dat object gebruikt wordt i.p.v. een nieuw object. Dat gedrag hebben we echter reeds geïmplementeerd in onze entities. Bijgevolg hoeven we er hier niets bijzonders voor te doen.

Nadat het **Genre**-object gemaakt is, gebruiken we dit als parameter in de functie **create(parameters)** van **Boek**, om het **Boek**-object te maken ❷. Ook hier geldt weer hetzelfde: het gebruik van een "oud" **Boek**-object dan wel het aanmaken van een nieuw gebeurt hier volautomatisch.

We controleren nu alvast eens of onze BoekDAO goed werkt. Maak een bestandje *test.php* aan in de root van je applicatie. Dit bestand zal geen deel uitmaken van onze uiteindelijke applicatie, maar zullen we gebruiken om snel bepaalde delen van het programma uit te testen.

De inhoud van *test.php*:

```
<?php
require_once("data/boekdao.class.php");

$lijst = BoekDAO::getAll();
print("<pre>");
print_r($lijst);
print("</pre>");
?>
```

Voer het bestand uit. Je zou een array te zien moeten krijgen, met daarin alle boek-objecten, met hun eigen gekoppelde genre.

Als tweede en laatste DAO ontwerpen we de GenreDAO klasse, die volledig analoog opgemaakt is:

```
<?php
require_once("data/dbconfig.class.php");
require_once("entities/genre.class.php");

class GenreDAO {

    public static function getAll() {
        $lijst = array();

        $dbh = new PDO(DBConfig::$DB_CONNSTRING,
            DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);
        $sql = "select id, omschrijving from mvc_genres";
        $resultSet = $dbh->query($sql);
        foreach ($resultSet as $rij) {
            $genre = Genre::create($rij["id"],
                $rij["omschrijving"]);
            array_push($lijst, $genre);
        }
        $dbh = null;
        return $lijst;
    }

}
```

Opgelet: ons SQL-statement bevat dit keer geen JOIN, vermits we simpelweg een lijst opvragen van alle genres, en er van hieruit geen verwijzing is naar de tabel *mvc_boeken*. Logischerwijs zijn de enige objecten die aangemaakt worden dan ook **Genre**-objecten.

We testen uiteraard ook deze DAO klasse, met volgende code in het bestand *test.php*:

```
<?php
require_once("data/genredao.class.php");
```

```
$lijst = GenreDAO::getAll();  
print("<pre>");  
print_r($lijst);  
print("</pre>");
```

De uitvoer toont een array van alle genres met hun id en omschrijving.

Tijd voor de businesslaag!

Stap 3: De businesslaag

Even een opfrissing: de businesslaag bevat per entity een serviceklasse die het aanspreekpunt zullen zijn voor de controller om een specifieke taak uit te voeren die betrekking heeft op die entity. Een serviceklasse bevat één functie per taak.

In ons geval is dit eerder beperkt. We hebben voorlopig slechts één taak te implementeren: alle boeken ophalen. We zullen dus een functie **toonAlleBoeken()** inbouwen in de klasse **BoekService**. De functie heeft geen parameters, en geeft een lijst van boeken terug.

Zoals blijkt uit de code van *boekservice.class.php* wordt dit een uiterst eenvoudige functie:

```
<?php  
require_once("data/boekdao.class.php");  
class BoekService {  
  
    public static function toonAlleBoeken() {  
        $lijst = BoekDAO::getAll();  
        return $lijst;  
    }  
  
}
```

Naar goede gewoonte testen we de goede werking van de serviceklasse, met ons bestand *test.php*:

```
<?php  
require_once("business/boekservice.class.php");  
  
$lijst = BoekService::toonAlleBoeken();  
print("<pre>");  
print_r($lijst);  
print("</pre>");
```

Krijg je netjes een array van alle boeken in de tabel *mvc_boeken* te zien, dan werkt de serviceklasse naar behoren.

Ook hier herhalen we even dat de servicelaag niet weet op welke manier de data die ze aanlevert getoond wordt. De klasse **BoekService** zorgt er enkel voor dat alle data die zullen moeten getoond worden teruggegeven worden.

In dat verband zijn we klaar om de eerste steen van onze presentatielaag te leggen.

Stap 4: De presentatielaag

We dienen een pagina te schrijven met een minimum aan PHP-code, die een lijst toont van boeken. Merk op dat we zeggen: "een lijst van boeken", en niet "een lijst van alle boeken".

Ons opzet is immers een pagina te schrijven die niet alleen gebruikt kan worden om alle boeken te tonen, maar ook boeken die aan een bepaald criterium voldoen. Op die manier kan eenzelfde pagina voor meerdere doeleinden gebruikt worden. Let er evenwel op dat het blijft gaan om een lijst van boeken. Mochten we op termijn een lijst van bijvoorbeeld genres willen tonen, dan zullen we dit laten afhandelen door een andere pagina.

Hierna vind je een voorbeeld van het bestand *boekenlijst.php* in de folder */presentation*. Uiteraard is de opmaak van de tabel slechts illustratief en mag je zelf de opmaak verder verfijnen.

Merk op dat de hoeveelheid PHP-code op deze pagina uiterst beperkt is.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Boeken</title>
    <style>
      table { border-collapse: collapse; }
      td, th { border: 1px solid black; padding: 3px; }
      th { background-color: #ddd }
    </style>
  </head>
  <body>
    <h1>Boekenlijst</h1>
    <table>
      <tr>
        <th>Titel</th>
        <th>Genre</th>
      </tr>
      <?php
        foreach ($boekenLijst as $boek) {
          ?>
          <tr>
            <td>
              <?php print($boek->getTitel());?>
            </td>
            <td>
              <?php
                print($boek->getGenre()
                  ->getOmschrijving());?>
            </td>
          </tr>
        <?php
          }
        }
      </table>
    </body>
  </html>
```

```
        ?>
    </table>
</body>
</html>
```

Door het feit dat bij het ophalen van de boeken ineens de data m.b.t. de genres (nl. de omschrijving) die erbij horen mee opgehaald werden, kunnen we in één moeite door deze omschrijving ophalen met:

```
$boek->getGenre()->getOmschrijving();
```

Enkel het gouden schaalpje ontbreekt nog...

Als laatste stap ontwerpen we de controller.

Stap 5: De controller

De controller is zo mogelijk nog korter dan de serviceklasse. Als ingangspunt van de applicatie dient deze louter de juiste servicelaag aan te spreken (**BoekService**) en de variabelen die in de presentatielaag gebruikt worden "klaar te zetten". Tenslotte wordt met een eenvoudige **include()**-functie de presentatiepagina geïmporteerd.

De controller noemen we hier *toonalleboeken.php*. Vergeet niet dat deze niet in een subfolder wordt geplaatst, maar gewoon rechtstreeks in de root van de applicatie.

```
<?php
require_once("business/boekservice.class.php");
$boekenLijst = BoekService::toonAlleBoeken();
include("presentation/boekenlijst.php");
?>
```

Test je applicatie uit door te surfen naar de controller *toonalleboeken.php*.

De meeste controllers zullen de omvang hebben van bovenstaand bestand. Er moet hier immers niet veel beslist worden:

1. Welke serviceklasse levert de data aan?
2. Welke presentatiepagina moet geïmporteerd worden?

Het eerste deel van de applicatie is hiermee af.

Objecten per ID ophalen

We breiden onze applicatie uit zodat we naast alle boeken en genres ook één enkel boek of één enkel genre kunnen ophalen op basis van een ID.

We voorzien altijd een dergelijke functie per entiteit. Naarmate je applicatie groeit zal je merken dat ze op dit soort functies dikwijls een beroep doet. Het kan dus geen kwaad (lees: is aan te raden) ze onmiddellijk te implementeren, zelfs al heb je ze op het eerste zicht niet direct nodig.

Het ophalen van slechts één boek of één genre op basis van een ID lijkt goed op het ophalen van alle boeken/genres. Alleen gebruik je nu eenvoudigweg een **fetch()** opdracht.

Merk op dat we er van uit gaan dat er wel degelijk een boek/genre bestaat in de databank met het meegegeven ID. Later bouwen we hier de nodige controles voor in, teneinde onze applicatie robuuster en ongevoeliger voor fouten te maken.

Voeg volgende functie toe aan de klasse **BoekDAO** en bestudeer ze.

```
public static function getById($id) {
    $dbh = new PDO(DBConfig::$DB_CONNSTRING,
        DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);
    $sql = "select mvc_boeken.id as boekid, titel, genreid,
        omschrijving from mvc_boeken, mvc_genres where
        genreid = mvc_genres.id and
        mvc_boeken.id = " . $id;
    $resultSet = $dbh->query($sql);
    $rij = $resultSet->fetch();
    $genre = Genre::create($rij["genreid"],
        $rij["omschrijving"]);
    $boek = Boek::create($rij["boekid"], $rij["titel"], $genre);
    $dbh = null;
    return $boek;
}
```

Een gelijkaardige functie voeg je toe aan de klasse **GenreDAO**.

```
public static function getById($id) {
    $dbh = new PDO(DBConfig::$DB_CONNSTRING,
        DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);
    $sql = "select omschrijving from mvc_genres
        where id = " . $id;
    $resultSet = $dbh->query($sql);
    $rij = $resultSet->fetch();
    $genre = Genre::create($id, $rij["omschrijving"]);
    $dbh = null;
    return $genre;
}
```

Vanzelfsprekend testen we onze functie onmiddellijk uit via *test.php*:

```
<?php
require_once("data/boekdao.class.php");

$boek = BoekDAO::getById(1);
print("<pre>");
```

```
print_r($boek);  
print("</pre>");  
?>
```

Test zelf ook de nieuwe functie van **GenreDAO** uit.

Een boek toevoegen

We implementeren nu de mogelijkheid om een nieuw boek toe te voegen. We zullen een formulier nodig hebben met een tekstveld om een titel in te vullen, en liefst een drop-down keuzelijst met alle genres om een genre uit te kiezen. Wanneer het formulier verstuurd wordt zien we het nieuwe boek onmiddellijk verschijnen in de lijst.

Maar first things first...

We beginnen met het uitbreiden van de data laag. Het toevoegen van een nieuw record gebeurt uiteraard in **BoekDAO**.

```
public static function create($titel, $genreId) {  
    $sql = "insert into mvc_boeken (titel, genreid)  
        values ('" . $titel . "', " . $genreId . ")";  
    $dbh = new PDO(DBConfig::$DB_CONNSTRING,  
        DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);  
    $dbh->exec($sql);  
    $boekId = $dbh->lastInsertId();  
    $dbh = null;  
    $genre = GenreDAO::getById($genreId);  
    $boek = Boek::create($boekId, $titel, $genre);  
    return $boek;  
}
```

Merk opnieuw op dat we geen controle uitvoeren op correcte waarden van **\$titel** of **\$genreId**. We zullen later een gepaste controle uitvoeren.

Ook de klasse **BoekService** voorzien we van een extra functie **voegNieuwBoekToe**. Dit zal de functie zijn die door onze controller opgeroepen wordt:

```
public static function voegNieuwBoekToe($titel, $genreId) {  
    BoekDAO::create($titel, $genreId);  
}
```

Vermits we in het formulier een drop-down keuzelijst met alle genres willen plaatsen, zal de controller in staat moeten zijn alle genres op te halen. Genres ophalen gebeurt door de klasse **GenreService**. We hebben deze nog niet gemaakt, dus we doen dit nu. Breng volgende code onder in het bestand *genreservice.class.php*.


```
<?php
require_once("data/genredao.class.php");
class GenreService {

    public static function toonAlleGenres() {
        $lijst = GenreDAO::getAll();
        return $lijst;
    }

}
```

We breiden de presentatielaag uit met één extra bestand: *nieuwboekform.php*. Deze presentatiepagina bevat het invulformulier met een tekstveld en een drop-down keuzelijst met daarin alle genres.

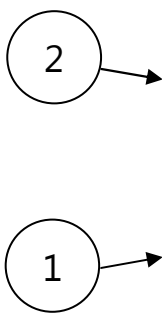
```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Boeken</title>
  </head>
  <body>
    <h1>Nieuw boek toevoegen</h1>
    <form method="post" action="voegboektoe.php?action=process">
      <table>
        <tr>
          <td>Titel:</td>
          <td>
            <input type="text" name="txtTitel">
          </td>
        </tr>
        <tr>
          <td>Genre:</td>
          <td>
            <select name="selGenre">
              <?php
                foreach ($genreLijst as $genre) {
                  ?>
                  <option value="<?php print($genre->getId());?>">
                    <?php print($genre->getOmschrijving());?></option>
                  <?php
                    }
                  ?>
                </select>
              </td>
            </tr>
            <tr>
              <td></td>
              <td>
                <input type="submit" value="Toevoegen">
              </td>
            </tr>
          </table>
        </form>
      </body>
    </html>
```

Markeer hierin de PHP-code. Je merkt dat deze eerder beperkt blijft. Er is enkel code voorzien om een foreach-lus te maken die doorheen alle genres in een genrelijst loopt, en van elk genre het ID en de omschrijving opvraagt.

Blijft nog één ding over: de controller. Noem deze *voegboektoe.php*.

```
<?php
require_once("business/genreservice.class.php");
require_once("business/boekservice.class.php");

if ($_GET["action"] == "process") {
    BoekService::voegNieuwBoekToe($_POST["txtTitel"], $_POST["selGenre"]);
    header("location: toonalleboeken.php");
    exit(0);
} else {
    $genreLijst = GenreService::toonAlleGenres();
    include("presentation/nieuwboekform.php");
}
```



Zoals je ziet zijn er twee manieren om in deze controller te geraken: met of zonder een action-attribuut "process".

Zonder dit attribuut (dus als men gewoon surft naar *voegboektoe.php*, ❶) worden alle genres opgehaald, en wordt de nieuwboekform-presentatiepagina geïmporteerd.

Als het attribuut bestaat en de waarde "process" heeft ❷, werd het invulformulier met de gegevens van het nieuwe boek verstuurd naar deze controller. In dat geval geven we de klasse **BoekService** de opdracht om een nieuw boek aan te maken. Als parameters geven we de inhoud van de formulervelden mee (het formulier wordt met de POST-methode verzonden).

Opgelet! Nadat dit gebeurd is importeren we géén presentatiepagina, maar sturen we de bezoeker onmiddellijk verder door naar de controller *toonalleboeken.php*.

Het doorsturen naar een andere locatie gebeurt met

```
header("location: <nieuwe_locatie_hier>");
exit(0);
```

De controller *toonalleboeken.php* zal dan op zijn beurt, zoals steeds, alle boeken opvragen en netjes in een lijst tonen.

Test de applicatie uit door te surfen naar *voegboektoe.php*.

Waar toe dient de instructie
`exit(0)`?



Wanneer de PHP interpreter deze instructie uitvoert wordt het script op die pagina onmiddellijk afgesloten. Alle instructies daarna worden genegeerd.



Is dat van belang? Die regel was
op zich al de laatste van de
pagina.



Denk eraan dat de meeste PHP scripts geleidelijk aan groeien en soms aangevuld worden. Het is de bedoeling dat de bezoeker bij het uitvoeren van de header instructie onmiddellijk doorgestuurd wordt. Zonder de regel `exit(0)` gebeurt dit pas aan het einde van het script.



Vraag jezelf af waarom na het toevoegen van een nieuw boek, de controller de bezoeker doorstuurt naar een andere controller, en niet gewoon de presentatiepagina *boekenlijst.php* importeert.

Vind je het antwoord niet, neem dan de proef op de som en vervang in *voegboektoe.php* de lijn met de **header**-functie door een **include**-opdracht:

```
include("presentation/nieuwboekform.php");
```

Voer de applicatie opnieuw uit, voeg een boek toe bekijk de URL in de adresbalk van je browser onmiddellijk na het toevoegen. Vernieuw nu de pagina (meestal de knop F5) en kijk wat er gebeurt.

Zorg er dus voor dat een controller enkel een presentatiepagina importeert wanneer hij slechts een lees-operatie (bvb records ophalen uit een databank) heeft uitgevoerd, en nooit na een schrijf-operatie (bvb records bijwerken, toevoegen of verwijderen). Betreft het wel een schrijf-operatie, voeg dan een **header("location: ...")** opdracht uit, zodat er automatisch doorgestuurd wordt naar een andere controller.

Een boek verwijderen

Om een boek uit de lijst te verwijderen moeten we een ankerpunt inbouwen, bijvoorbeeld een hyperlink naast elk boek waarop we kunnen klikken om dat specifieke boek te verwijderen. We roepen een nieuwe controller aan en geven het boek ID mee op de URL (als GET-parameter). De controller roept op zijn beurt de **BoekService** klasse op, die dan weer op zijn beurt de **BoekDAO** zal aanroepen om het gewenste record uit de databank te verwijderen.

We beginnen op de diepste laag, zoals gewoonlijk, bij **BoekDAO**. Volgende code zal op basis van een ID een DELETE-operatie uitvoeren.

```
public static function delete($id) {
    $sql = "delete from mvc_boeken where id = " . $id;
    $dbh = new PDO(DBConfig::$DB_CONNSTRING,
        DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);
    $dbh->exec($sql);
    $dbh = null;
}
```

De hoeveelheid code die op de businesslaag (**BoekService**) toegevoegd moet worden is minimaal:

```
public static function verwijderBoek($id) {
    BoekDAO::delete($id);
}
```

Voor het verwijderen van het boek is geen zichtbare pagina nodig, maar we maken in deze stap wel een aanpassing aan de reeds bestaande presentatiepagina *boekenlijst.php*. We voegen op elke rij een hyperlink toe die ons naar de nieuwe controller (die we zo dadelijk zullen maken) stuurt, met als extra parameter het ID van het boek.

Het bestand *boekenlijst.php* wordt dan:

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Boeken</title>
        <style>
            table { border-collapse: collapse; }
            td, th { border: 1px solid black; padding: 3px; }
            th { background-color: #ddd}
        </style>
    </head>
    <body>
        <h1>Boekenlijst</h1>

        <table>
            <tr>
                <th>Titel</th>
                <th>Genre</th>
                <th></th>
            </tr>
            <?php
            foreach ($boekenLijst as $boek) {
                <tr>
                    <td>
                        <?php print($boek->getTitel());?>
                    </td>
                    <td>
                        <?php print(
                            $boek->getGenre()->getOmschrijving());?>
                    </td>
                    <td>
```

```

        <a href="verwijderboek.php?id=
        <?php print($boek->getId());?>">
        Verwijder</a>
    </td>
</tr>
<?php
}
?>
</table>
</body>
</html>

```

Tot slot maken we de controller *verwijderboek.php*:

```

<?php
require_once("business/boekservice.class.php");
BoekService::verwijderBoek($_GET["id"]);
header("location: toonalleboeken.php");
exit(0);

```

Merk op dat we na het verwijderen van een boek géén presentatiepagina importeren, maar de bezoeker automatisch verder doorsturen naar een andere controller (in dit geval *toonalleboeken.php*).

Het mag onderhand duidelijk zijn dat we in een URL nooit rechtstreeks een presentatiepagina aanspreken. In de applicatie zal elke hyperlink naar een controller verwijzen, zal elke automatische doorsturing naar een controller gebeuren, en zal elk formulier doorgestuurd worden naar een controller.

De controllers zijn en blijven te allen tijde het aanspreekpunt, doorheen de hele applicatie.

Een boek bijwerken

In tegenstelling tot het verwijderen van een boek zal het bijwerken van de eigenschappen van een boek wel een eigen presentatiepagina introduceren. Deze pagina zal een formulier moeten bevatten met een tekstveld voor de titel en een drop-down keuzelijst voor een genre. Uiteraard zijn de huidige waarden van de boekeigenschappen ingevuld.

We beginnen opnieuw onderaan, in **BoekDAO**. We voegen een functie toe die slechts één parameter heeft: een object van de klasse **Boek**. Van dit object werken we met een UPDATE elke eigenschap bij, met uitzondering van het ID. Het ID gebruiken we om te bepalen welk record in de tabel bijgewerkt moet worden, en dit ID halen we eveneens uit het **Boek**-object.

We breiden **BoekDAO** uit:

```

public static function update($boek) {
    $sql = "update mvc_boeken set titel='" . $boek->getTitel() .

```

```

        ", genreid=" . $boek->getGenre()->getId() .
        " where id = " . $boek->getId();

$dbh = new PDO(DBConfig::$DB_CONNSTRING,
    DBConfig::$DB_USERNAME, DBConfig::$DB_PASSWORD);
$dbh->exec($sql);
$dbh = null;
}

```

We voegen twee functies toe aan **BoekService**.

```

public static function haalBoekOp($id) {
    $boek = BoekDAO::getById($id);
    return $boek;
}

public static function updateBoek($id, $titel, $genreId) {
    $genre = GenreDAO::getById($genreId);
    $boek = BoekDAO::getById($id);
    $boek->setTitel($titel);
    $boek->setGenre($genre);
    BoekDAO::update($boek);
}

```

De bestaansreden van **updateBoek (eigenschappen)** is duidelijk. De functie **haalBoekOp (id)** zullen we in de nieuwe controller nodig hebben. Om het formulier te kunnen invullen met de huidige waarden van de boekeigenschappen moet de controller in staat zijn om de servicelaag opdracht te geven één enkel boek op te halen. De functie **haalBoekOp (id)** zal dit mogelijk maken.

Er komt een extra presentatiepagina bij: *updateboekform.php*:

```

<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Boeken</title>
    </head>
    <body>
        <h1>Boek bijwerken</h1>
        <form method="post" action="updateboek.php?action=process&id=
            <?php print($boek->getId());?>">

            <table>
                <tr>
                    <td>Titel:</td>
                    <td>
                        <input type="text" name="txtTitel"
                            value="<?php print($boek->getTitel());?>"
                        </td>
                </tr>
                <tr>
                    <td>Genre:</td>
                    <td>
                        <select name="selGenre">
                            <?php
                                foreach ($genreLijst as $genre) {
                                    if ($genre->getId() == $boek->getGenre()->getId()) {
                                        $selWaarde = " selected";
                                    } else {

```

```

        $selWaarde = "";
    }
    ?>
    <option value="<?php print($genre->getId());?>"<?php
        print($selWaarde);?>>
    <?php print($genre->getOmschrijving());?>
    </option>
    <?php
        }
    ?>
    </select>
    </td>
</tr>
<tr>
    <td></td>
    <td>
        <input type="submit" value="Bijwerken">
    </td>
</tr>
</table>
</form>
</body>
</html>

```

De presentatiepagina zal er dus van uit gaan dat de controller (die we zo dadelijk maken) twee variabelen "klaar" zet: een variabele **\$boek** die alle eigenschappen van het te bewerken boek bevat, en een variabele **\$genreLijst**, die een array bevat van alle genres.

We ontwerpen een nieuwe controller *updateboek.php*:

```

<?php
require_once("business/genreservice.class.php");
require_once("business/boekservice.class.php");

if ($_GET["action"] == "process") {
    BoekService::updateBoek($_GET["id"], $_POST["txtTitel"],
        $_POST["selGenre"]);
    header("location: toonalleboeken.php");
    exit(0);
} else {
    $genreLijst = GenreService::toonAlleGenres();
    $boek = BoekService::haalBoekOp($_GET["id"]);
    include("presentation/updateboekform.php");
}

```

Analoog aan de controller voor het toevoegen van een nieuw boek zijn er ook hier twee mogelijke ingangspunten.

Ofwel wordt de controller opgeroepen waarbij een parameter "action" de waarde "process" heeft. Dat gebeurt wanneer we het aanpassingsformulier versturen naar deze controller. In dat geval roepen we **BoekService** aan en sturen we alle eigenschappen van het boek mee. De ID halen we uit de URL (via **\$_GET**); de andere eigenschappen uit het formulier.

In alle andere gevallen is het de bedoeling dat een presentatiepagina wordt geïmporteerd met daarop een formulier met de huidige eigenschappen van het boek reeds ingevuld (dit is de reden waarom **BoekService** een functie **haalBoekOp()** nodig heeft).

Denk eraan: na het bijwerken van de gegevens wordt géén presentatiepagina geïmporteerd, maar wel verder doorgestuurd naar de controller *toonalleboeken.php*.

Er staat ons nog één ding te doen. We moeten de presentatiepagina *boekenlijst.php* aanpassen, zodat de titels hyperlinks worden die ons bij een klik erop doorsturen naar de controller *updateboek.php*.

Vervang in *boekenlijst.php* de lijn

```
<?php print($boek->getTitel());?>
```

door

```
<a href="updateboek.php?id=<?php print($boek->getId());?>">
<?php print($boek->getTitel());?>
</a>
```

en test de applicatie uit door naar *toonalleboeken.php* te surfen. Een eerste basisversie van onze toepassing is hiermee afgewerkt.

Samengevat:

- Het ingangspunt van de applicatie is altijd een controller
- De controller vangt eventuele formulierwaarden of parameters van de URL op en beslist welke serviceklasse/serviceklassen moeten worden aangesproken.
- De serviceklasse spreekt de nodige DAO-klassen aan en geeft de gewenste data terug aan de controller.
- Bij een leesoperatie zet de controller deze data "klaar" in variabelen en wordt een presentatiepagina geïmporteerd.
- Bij een schrijfoperatie stuurt de controller de bezoeker verder door naar een andere controller (via de functie **header()**).

Oefening 10.1 ★★

Ontwerp een pagina met "geheime informatie". Bedoeling is dat je ook een formulier ontwerpt waar de bezoeker als gebruikersnaam "admin" en wachtwoord "geheim" moet invullen. Wanneer getracht wordt de geheime pagina te lezen zonder het formulier correct ingevuld te hebben, wordt de bezoeker doorgestuurd naar het formulier.

→ *Oplossing:* 849638

Oefening 10.2 ★★

Maak in de databank een tabel "gebruikers" aan, met drie velden: een autonummeringsveld id, een gebruikersnaam en een wachtwoord.

Breid oefening 10.1 uit zodat de persoon kan inloggen wanneer hij een correcte combinatie gebruikersnaam-wachtwoord heeft ingevuld.

→ *Oplossing:* 496398

Gegevens bundelen

Je weet reeds dat de controller de servicelaag kan aanspreken om gegevens op te halen die nodig zijn bij het tonen van een presentatiepagina. Bekijk nog even het bestand *voegboektoe.php* uit het voorgaande stuk.

```
<?php
require_once("business/genreservice.class.php");
require_once("business/boekservice.class.php");

if ($_GET["action"] == "process") {

    ...

} else {

    $genreLijst = GenreService::toonAlleGenres();
    include("presentation/nieuwboekform.php");

}
```

Om een nieuw boek toe te voegen hebben we een lijst nodig van alle beschikbare genres. We willen immers een keuzelijst genereren op de presentatiepagina *nieuwboekform.php*. Deze lijst verkregen we van **GenreService**. Veronderstel eens dat we ook alle soorten boekkaften nodig hadden. Dan hadden we moeten schrijven:

```
$genreLijst = GenreService::toonAlleGenres();
$kaftLijst = KaftService::toonAlleKaften();
```

Het is evenwel af te raden om per taak de servicelaag meer dan één keer aan te spreken. De taak kan namelijk alleen goed uitgevoerd worden als én alle genres, én alle kaften worden opgehaald. Het is geen goed idee om dan te “vertrouwen” op het feit dat de controller beide functies zal aanroepen, en niet slechts één van de twee (hou er rekening mee dat de controllers door andere programmeurs geschreven kunnen worden dan zij die de servicelaag onderhouden).

Vergelijk even met een overschrijving in het bankwezen. De serviceklasse wordt dan door de controller niet aangeroepen met

```
BankService:haalAf($van, $hoeveel);  
BankService:stort($naar, $hoeveel);
```

maar wel met

```
BankService::schrijfOver($van, $naar, $hoeveel);
```

Als programmeur van **BankService** kan je de ontwikkelaar van de controller immers niet verplichten om na het uitvoeren van **haalAf** ook **stort** uit te voeren. Echter, met slechts één functie **schrijfOver**, waarin de beide uitgevoerd worden, heeft hij geen keuze.

Hetzelfde geldt voor het ophalen van gegevens. In plaats van

```
$genreLijst = GenreService::toonAlleGenres();  
$kaftLijst = KaftService::toonAlleKaften();
```

schrijf je dus beter

```
$gegevensLijst = BoekService::haalGegevensOpVoorNieuwBoek();
```

We introduceren hier evenwel een probleempje... Wat geeft de functie **haalGegevensOpVoorNieuwBoek()** terug aan de controller? Het zou een soort van object moeten zijn waarin zowel een array van genres zit, als een array van kaften. Natuurlijk kun je dit oplossen door de functie als volgt te definiëren:

```
public static function haalGegevensOpVoorNieuwBoek() {  
    $genreLijst = GenreService::toonAlleGenres();  
    $kaftLijst = KaftService::toonAlleKaften();  
    $gegevensLijst = array();  
    $gegevensLijst["genres"] = $genreLijst;  
    $gegevensLijst["kaften"] = $kaftLijst;  
    return $gegevensLijst;  
}
```

en dus een extra associatieve array **\$gegevensLijst** te maken waarin je beide arrays (genres en kaften) op een verschillende index stockeert.

In de presentatiepagina kan je dan schrijven:

```
<select name="selGenre">
    <?php
        foreach ($gegevensLijst["genres"] as $genre) {
            ...
        }
    ?>
</select>
```

Op zich is dit niet fout, maar misschien kan het netter.

Je zou ook een extra klasse kunnen schrijven met twee properties, die je kan invullen met de beide lijsten, een zgn DTO-klasse (**Data Transfer Object**):

```
class GenresEnKaftenDTO {
    public $genres;
    public $kaften;
}
```

en in de servicelaag:

```
public static function haalGegevensOpVoorNieuwBoek() {
    $genreLijst = GenreService::toonAlleGenres();
    $kaftLijst = KaftService::toonAlleKaften();
    $gegevensDTO = new GenresEnKaftenDTO();
    $gegevensDTO->genres = $genreLijst;
    $gegevensDTO->kaften = $kaftenLijst;
    return $gegevensDTO;
}
```

De presentatiepagina wordt dan:

```
<select name="selGenre">
    <?php
        foreach ($gegevensLijst->genres as $genre) {
            ...
        }
    ?>
</select>
```

Dat lijkt er al beter op, maar als je voor elke vorm van gegevensoverdracht een aparte klasse moet gaan schrijven, alleen maar omdat een functie slechts één waarde terug kan geven, dan raak je ver van huis.

In PHP bestaat er een speciale constructie die de naam **stdClass** (let op de eerste **kleine** letter 's') draagt, en exact de oplossing voor dit probleem biedt. Een **stdClass**-object kan je vergelijken met een klasse met publieke properties, die je last-minute kunt toekennen, zonder dat je eerst de klasse schrijft en de properties expliciet definieert, zoals je met "echte" PHP-klassen zou doen. Je kan dus schrijven:

```
$obj = new stdClass();
$obj->titel = "...";
$obj->jaarVanUitgave = 2013;
```

Specifiek voor onze situatie zou je de servicelaag als volgt kunnen inkleden:

```
public static function haalGegevensOpVoorNieuwBoek() {  
    $gegevensDTO = new stdClass();  
    $gegevensDTO->genres = GenreService::toonAlleGenres();  
    $gegevensDTO->kaften = KaftService::toonAlleKaften();  
    return $gegevensDTO;  
}
```

We hoeven geen extra klasse te schrijven, en toch zijn onze beide lijsten netjes gebundeld. Onze presentatielaag ziet er net zo uit alsof we een eigen klasse geschreven zouden hebben:

```
<select name="selGenre">  
    <?php  
        foreach ($gegevensLijst->genres as $genre) {  
            ...  
        }  
    ?>  
</select>
```

Een **stdClass**-object kan elk soort property bevatten die een "echt" object ook zou kunnen bevatten, dus volgende code is geen probleem:

```
$obj = new stdClass();  
$obj->jaar = 2013;  
$obj->coach = new Coach();  
$obj->deelnemers = array();  
$obj->deelnemers[0] = new Deelnemer();  
$obj->deelnemers[0]->setNaam("Paul");  
$obj->reglement = new stdClass();  
$obj->reglement->wedstrijdreglement = new Reglement();
```

Vooraleer we dit stuk afsluiten, toch nog een waarschuwing. Een object van **stdClass** is **geen vervanger voor een volwaardige klasse**. Gebruik ze liefst

- enkel voor het bundelen van meerdere stukken informatie die de servicelaag als één geheel moet doorgeven aan de controller. De reden daarvoor laat zich raden: objecten van **stdClass** zijn immers objecten waarvan alle properties public zijn. Je geeft daarbij alle controle over de inhoud ervan uit handen, en dit zondigt tegen de regels van het blackbox-principe. Gebruik deze constructie verstandig!

Foutafhandeling

Tot nu toe zijn we er bij zowat alle opdrachten en oefeningen van uit gegaan dat de bezoeker geldige, correcte waarden invult waar iets ingevuld moet worden. In de praktijk echter zijn mensen die het minste vakkennis hebben de beste testers van een applicatie. Foutieve input is steeds mogelijk. Een correcte foutafhandeling is noodzakelijk.

Bestudeer volgend voorbeeld. Beredeneer en markeer de elementen die zorgen voor foutafhandeling. Voer daarna uit.

```
<?php
class Rekening {
    private $saldo;

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

    public function storten($bedrag) {
        if ($bedrag < 0) throw new Exception();
        $this->saldo += $bedrag;
    }

    public function getSaldo() {
        return $this->saldo;
    }
}
?>

<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Test exception</title>
    </head>
    <body>
        <?php
        $rek = new Rekening();
        try {
            print("<p>Saldo: " . $rek->getSaldo() . " &euro;</p>");
            $rek->storten(200);
            print("<p>Saldo: " . $rek->getSaldo() . " &euro;</p>");
        } catch (Exception $ex) {
            print("<p>Een negatief bedrag storten is niet mogelijk!</p>");
        }
        ?>
    </body>
</html>
```

We komen zo dadelijk nog op de structuur terug. Wijzig eerst de lijn

```
$rek->storten(200);
```

In

```
$rek->storten(-200);
```

Tracht het resultaat zelf eerst te verklaren.

We breiden dit voorbeeld verder uit naar twee zelfgemaakte exceptions. Laat ons er eens van uit gaan dat de saldolimiet van onze rekening 1000 € is.

```
<?php
class NegatieveStortingException extends Exception {
}
class RekeningVolException extends Exception {
}
```

```

class Rekening {
    private $saldo;

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

    public function storten($bedrag) {
        if ($bedrag < 0) throw new NegatieveStortingException();
        if ($this->saldo + $bedrag > 1000) throw new RekeningVolException();
        $this->saldo += $bedrag;
    }

    public function getSaldo() {
        return $this->saldo;
    }
}
?>

<!DOCTYPE HTML>
<html>
    <head>
        <meta charset=utf-8>
        <title>Test exception</title>
    </head>
    <body>
        <?php
            $rek = new Rekening();
            try {
                print("<p>Saldo: " . $rek->getSaldo() . " &euro;</p>");
                $rek->storten(200);
                $rek->storten(600);
                $rek->storten(300);
                print("<p>Saldo: " . $rek->getSaldo() . " &euro;</p>");
            } catch (NegatieveStortingException $ex) {
                print("<p>Een negatief bedrag storten is niet mogelijk!</p>");
            } catch (RekeningVolException $ex) {
                print("<p>Dit bedrag kan niet gestort worden, de limiet
                    van de rekening is 1000 &euro;!</p>");
            }
        ?>
    </body>
</html>

```

Bestudeer, voer uit en verklaar het resultaat.

Oefening 10.3 ★

Breid het voorbeeld uit. Maak een derde Exceptionklasse die opgeworpen wordt wanneer men tracht in één keer een bedrag te storten dat groter is dan 500 €. Zorg voor een gepaste foutmelding. Test je oplossing uit met verschillende bedragen.

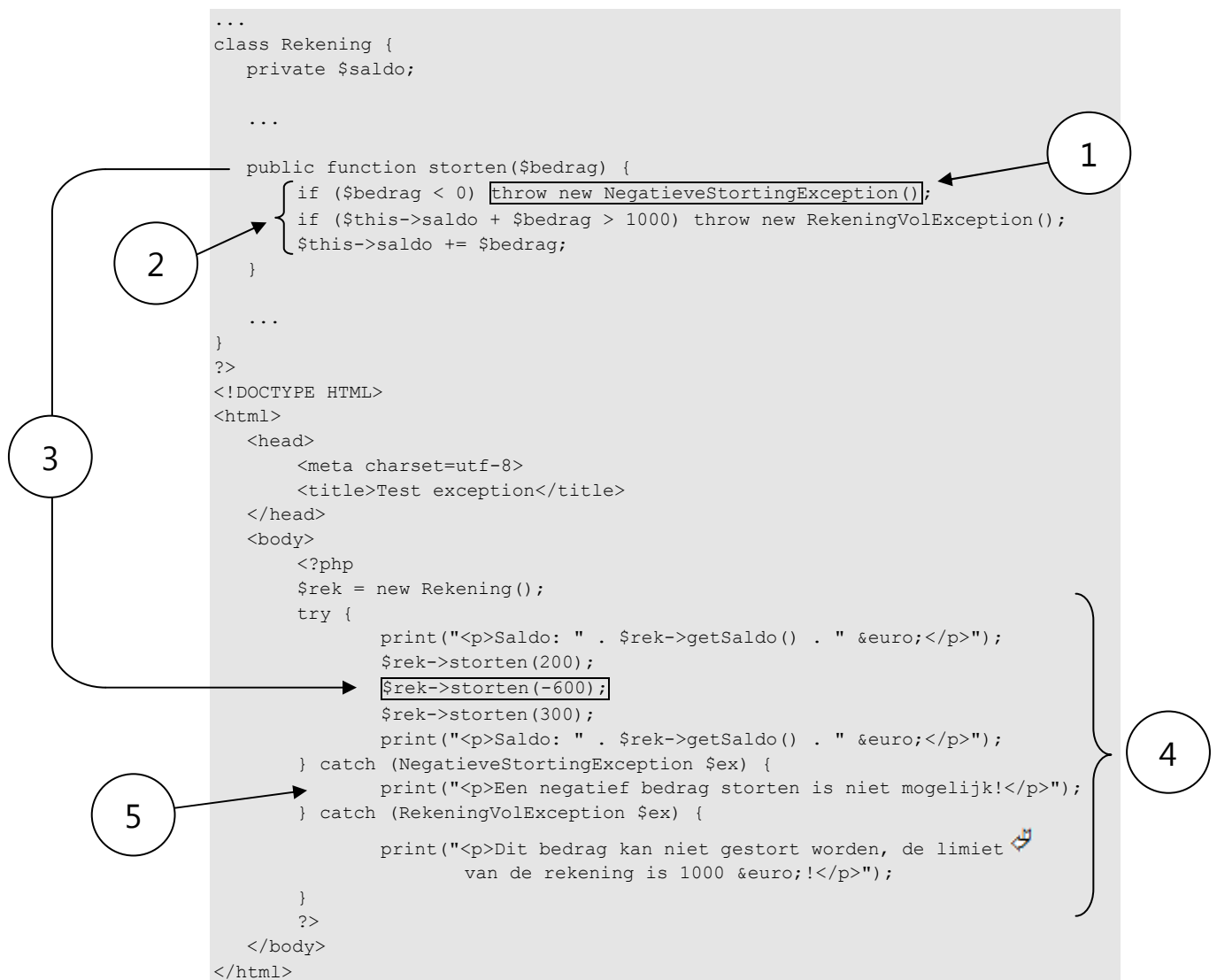
→ *Oplossing: 193655*

De theorie

Wanneer zich een fout voordoet in het programma kan men een exception "opwerpen". Dit opwerpen gebeurt simpelweg met

```
throw new NaamExceptionKlasse();
```

De exception zoekt dan naar de met die klasse overeenkomende **catch**-structuur die een onderdeel is van het **try-catch**-blok waarin de foutmelding werd opgeworpen. Wordt er geen geldig catch-blok gevonden, dan wordt de exception verder naar een "hoger niveau" opgeworpen, waar er opnieuw gezocht wordt naar een overeenstemmend catch-blok.



Concreet voor het geval van een negatieve storting wordt de fout eerst opgeworpen ❶. Daarna wordt er gezocht naar het eerste buitenliggende try-catch-blok waarbinnen de fout werd opgeworpen. In dit geval is er in het hele blok ❷ geen enkele try-catch te vinden. Dus wordt de fout verder opgeworpen naar boven, naar de regel waarop de functieaanroep gebeurde ❸. Opnieuw wordt gekeken in de onmiddellijke "omgeving" naar een try-catch-blok ❹. Dit keer wordt er wel één gevonden, en dus wordt er nagekeken of er een catch-blok is die dit specifieke type fout (**NegatieveStortingException**) opvangt en afhandelt. Dit blok is

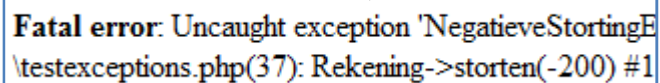
inderdaad aanwezig **5** en de instructies erin worden één voor één uitgevoerd.

Mocht dit blok niet aanwezig geweest zijn, dan zou er getracht geweest zijn de fout nog verder naar boven op te werpen. Echter, er is geen sprake meer van een functieaanroep – we zitten in ons "hoofdprogramma". PHP zal in dit geval een (niet zo mooie) foutmelding op het scherm tonen.

Test dit uit door de volgende twee regels te wissen:

```
} catch (NegatieveStortingException $ex) {  
    print("<p>Een negatief bedrag storten is niet mogelijk!</p>");
```

Voer de applicatie opnieuw uit.



Fatal error: Uncaught exception 'NegatieveStortingException' with message 'testexceptions.php(37): Rekening->storten(-200) #1'

De term "Uncaught exception" spreekt voor zich...

Van nu af aan zullen we spreken van "exceptions" i.p.v. uitzonderingen.

Foutafhandeling in MVC

Hoewel foutafhandeling bij het optreden van uitzonderingen op zich niets te maken heeft met Model-View-Controller willen we toch even uitleggen hoe je een correcte foutafhandeling maakt in applicaties die volgens dat design pattern opgebouwd zijn.

De vraag die we ons voornamelijk moeten stellen is: welke laag is verantwoordelijk voor welke controles? Veronderstel – teruggrijpend naar onze boekenapplicatie uit de vorige sectie – dat we er ons van willen verzekeren dat er geen twee boeken met dezelfde titel in de databank terecht komen. Het idee is vrij eenvoudig: controleer bij het toevoegen van een nieuw boek en bij het bewerken van de titel van een bestaand boek of er al geen boek met dezelfde titel in de databank aanwezig is.

Willen we dat laatste kunnen nagaan, dan hebben we een functie nodig die ons een boek-object geeft a.d.h.v. een titel i.p.v. een ID. Bestaat dit boek-object, dan is de titel reeds aanwezig. Deze functie, die we **getByTitel(\$titel)** zullen noemen ontbreekt op dit ogenblik. Het is een goede oefening om deze nu zelf eerst te proberen implementeren, vooraleer verder te lezen.

Hieronder volgt de code voor deze functie, die we toevoegen aan **BoekDAO**.

```
public static function getByTitel($titel) {
    $dbh = new PDO(DBConfig::$DB_CONNSTRING, DBConfig::$DB_USERNAME,
        DBConfig::$DB_PASSWORD);
    $sql = "select mvc_boeken.id as boekid, titel, genreid, omschrijving
            from mvc_boeken, mvc_genres where genreid = mvc_genres.id
            and titel = '" . $titel . "'";
    $resultSet = $dbh->query($sql);
    $rij = $resultSet->fetch();
    if (!$rij) {
        return null;
    } else {
        $genre = Genre::create($rij["genreid"], $rij["omschrijving"]);
        $boek = Boek::create($rij["boekid"], $rij["titel"], $genre);
        $dbh = null;
        return $boek;
    }
}
```

Merk op dat we het NULL-object teruggeven als de rij niet gevonden was. In het andere geval wordt het Boek-object aangemaakt en teruggegeven. Hier komen voorlopig nog geen exceptions bij kijken. Het is namelijk perfect "legaal" om een boek te proberen ophalen dat niet bestaat in de databank. In dat geval krijgen we gewoon NULL terug.

We hebben nu de functie, vraag is enkel nog wanneer we moeten controleren of bij het toevoegen van een boek er reeds een gelijkgetiteld boek bestaat: in de controller, in de servicelaag of in de data laag?

Steken we deze controle in de controller, dan moeten we dezelfde controle herhalen voor alle controllers die op de één of andere manier een boek proberen toevoegen. Steken we deze in de servicelaag, dan moeten we dezelfde controle herhalen voor elke servicelaag die een boek probeert toe te voegen. Akkoord, op dit ogenblik is er slechts één controller en één servicelaag die een boek kan toevoegen. Maar we mogen niet vergeten dat onze DAO-klassen herbruikbaar moeten zijn. Dat betekent dat het kan voorkomen dat er op een andere plaats in de applicatie de mogelijkheid geboden kan worden om een boek toe te voegen. We hebben er dus alle belang bij de controle zo "diep" mogelijk te steken, m.a.w. op de data laag.

Volgende twee regels worden de allereerste van de functie **create(\$titel, \$genreId)** van **BoekDAO**:

```
$bestaandBoek = self::getByTitel($titel);
if (isset($bestaandBoek)) throw new TitelBestaatException();
```

Dit impliceert dat we een klasse **TitelBestaatException** moeten ontwerpen. Maak in de root van de applicatie een nieuwe map */exceptions*. Hierin brengen we alle zelfgemaakte exception-klassen onder.

De opbouw van **TitelBestaatException** is zeer eenvoudig:

```
<?php
class TitelBestaatException extends Exception {
}
```

We doen in feite niets méér dan deze klasse slechts definiëren. Merk op dat het hier wel degelijk om een echte klasse gaat. Extra attributen, methodes en/of een constructor toevoegen is dus perfect mogelijk (en soms nodig)!

We hebben nu de exception én we hebben de plaats waar we ze opwerpen. We moeten ze uiteraard nog ergens opvangen en afhandelen ook.

Een goede plaats om dit te doen is de controller, vermits we graag een gepaste foutmelding op het scherm afdrukken, en dit zal gebeuren in de presentatielaag (en de controller importeert de presentatielaag).

We passen *voegboektoe.php* aan:

```
<?php
require_once("business/genreservice.class.php");
require_once("business/boekservice.class.php");
require_once("exceptions/titelbestaatexception.class.php");

if ($_GET["action"] == "process") {
    try {
        BoekService::voegNieuwBoekToe($_POST["txtTitel"], $_POST["selGenre"]);
        header("location: toonalleboeken.php");
        exit(0);
    } catch (TitelBestaatException $tbe) {
        header("location: voegboektoe.php?error=titleexists");
        exit(0);
    }
} else {
    $genreLijst = GenreService::toonAlleGenres();
    $error = $_GET["error"];
    include("presentation/nieuwboekform.php");
}
```

Op het ogenblik dat er bij het toevoegen van een nieuw boek een **TitelBestaatException** optreedt, dan wordt de bezoeker verder doorgestuurd naar dezelfde controller, maar deze keer met een GET-parameter "error", met als waarde "titleexists". Uiteraard mag je deze benamingen zelf kiezen, zolang je hierin consequent te werk gaat. Bij het aanroepen van deze controller wordt de inhoud van deze GET-parameter "klaargezet" in een variabele **\$error**.

We werken nu onze presentatiepagina nieuwboekform.php lichtjes bij, zodat er een gepaste foutmelding getoond wordt, wanneer **\$error** de vooropgestelde waarde bevat:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Boeken</title>
  </head>
  <body>
    <h1>Nieuw boek toevoegen</h1>
    <?php
      if ($error == "titleexists") {
        ?>
        <p style="color: red">Titel bestaat al</p>
        <?php
      }
      ?>
      <form method="post" action="voegboektoe.php?action=process">
        <table>
          <tr>
            <td>Titel:</td>
            <td>
              <input type="text" name="txtTitel">
            </td>
          </tr>
          <tr>
            <td>Genre:</td>
            <td>
              <select name="selGenre">
                <?php
                  foreach ($genreLijst as $genre) {
                    ?>
                    <option value="<?php print($genre->getId());?>">
                      <?php print($genre->getOmschrijving());?></option>
                    <?php
                  }
                ?>
              </select>
            </td>
          </tr>
          <tr>
            <td></td>
            <td>
              <input type="submit" value="Toevoegen">
            </td>
          </tr>
        </table>
      </form>
    </body>
  </html>
```

Test de applicatie opnieuw uit door te surfen naar *voegboektoe.php* en een titel in te geven die reeds bestaat in de tabel *mvc_boeken*.

Ook bij het bijwerken van de eigenschappen van een bestaand boek zullen we een dergelijke controle moeten inbouwen. Echter, het is dit keer onvoldoende om te controleren of er al een boek met de opgegeven titel bestaat. De

tweede voorwaarde is uiteraard dat het ID van het gevonden boek verschilt van het ID van het boek dat we willen bijwerken.

We passen de functie **update (\$boek)** in **BoekDAO** aan en voegen de volgende controleregels toe, aan het begin van de functie:

```
$bestaandBoek = self::getByTitel($boek->getTitel());
if (isset($bestaandBoek) && $bestaandBoek->getId() != $boek->getId()) throw
    new TitelBestaatException();
```

En ook de controller voor het updaten van een boek wordt bijgewerkt:

```
<?php
require_once("business/genreservice.class.php");
require_once("business/boekservice.class.php");
require_once("exceptions/titelbestaatexception.class.php");

if ($_GET["action"] == "process") {
    try {
        BoekService::updateBoek($_GET["id"], $_POST["txtTitel"],
                                $_POST["selGenre"]);
        header("location: toonalleboeken.php");
        exit(0);
    } catch (TitelBestaatException $tbe) {
        header("location: updateboek.php?id=".$_GET["id"]."&error=titleexists");
        exit(0);
    }
} else {
    $genreLijst = GenreService::toonAlleGenres();
    $boek = BoekService::haalBoekOp($_GET["id"]);
    $error = $_GET["error"];
    include("presentation/updateboekform.php");
}
```

Als de TitelBestaatException wordt opgevangen wordt de bezoeker verder doorgestuurd naar dezelfde controller, maar dit keer met de GET-parameter "error" toegevoegd. Vergeet ook niet om het ID mee door te sturen, want zoals je in het bovenste stuk van de code ziet, rekent de controller daarop ❶.

De GET-parameter "error" wordt overgezet naar **\$error**, en staat klaar voor de presentatielaag. Deze laatste – updateboekform.php – wordt eveneens bijgewerkt, zodat er een foutmelding getoond kan worden.

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Boeken</title>
  </head>
  <body>
    <h1>Boek bijwerken</h1>
    <?php
    if ($error == "titleexists") {
        ?>
        <p style="color: red">Titel bestaat al</p>
        <?php
    }
    ?>
```

```

<form method="post" action="updateboek.php?action=process&id=
    <?php print($boek->getId());?>">

    <table>
        <tr>
            <td>Titel:</td>
            <td>
                <input type="text" name="txtTitel"
                value="<?php print($boek->getTitel());?>"
                </td>
        </tr>
        <tr>
            <td>Genre:</td>
            <td>
                <select name="selGenre">
                    <?php
                        foreach ($genreLijst as $genre) {
                            if ($genre->getId() == $boek->getGenre()->getId()) {
                                $selWaarde = " selected";
                            } else {
                                $selWaarde = "";
                            }
                        }
                    <?>
                    <option value="<?php print($genre->getId());?>"
                    <?php print($selWaarde);?>"
                    <?php print($genre->getOmschrijving());?></option>
                    <?php
                        }
                    <?>
                </select>
            </td>
        </tr>
        <tr>
            <td></td>
            <td>
                <input type="submit" value="Bijwerken">
            </td>
        </tr>
    </table>
</form>
</body>
</html>

```

Test de applicatie opnieuw uit. Surf naar **toonalleboeken.php**, klik op een boek naar keuze en wijzig de titel naar een reeds bestaande titel.

Namespaces

De wereld van PHP omvat een brede waaier aan klassenlibraries, die allerlei problemen aanpakken, of frequent voorkomende taken eenvoudig en efficiënt uitvoeren. Je wil deze libraries kunnen gebruiken in je eigen projecten, of misschien wil je zelf wel een library schrijven en verspreiden. Vroeg of laat stuit je op problemen met naamgevingen. Je zal niet de eerste programmeur zijn die een **UtilManager**- of **ConnectionHandler**-klasse

schrijft. Wanneer je in eenzelfde script gebruik wilt maken van zowel je eigen **UtilManager** als die van een project dat je geïmporteerd hebt, weet PHP niet welke klasse je bedoelt wanneer je

```
$manager = new UtilManager();
```

schrijft. Namespaces vormen hiervoor een oplossing. Concreet gebruik je namespaces in hun eenvoudigste vorm door simpelweg bovenaan elk bestand dat je maakt de namespace waarin het zich bevindt te definiëren:

```
<?php
namespace MijnProject;
class UtilManager {
...
}
```

Wanneer je nu een nieuw object wilt aanmaken van de klasse **UtilManager**, kan dat d.m.v.:

```
$manager = new \MijnProject\UtilManager();
```

Wil je een object aanmaken van de **UtilManager** uit de geïmporteerde library, dan gebruik je:

```
$manager = new \NamespaceVanAndereLibrary\UtilManager();
```

De geïmporteerde library moet dan uiteraard wel een namespace hebben, wat doorgaans wel het geval is.

Wat structuur aanbrengen...

Je mag je namespaces zelf verder onderverdelen in verschillende subelementen en subsubelementen. Dit is ook aan te raden, omdat het wat meer structuur in je project brengt. Je mag dus schrijven:

```
<?php
namespace BookApplication\Service;
class BookService {
    public static function getAllBooks() {
        ...
    }
}
```

Let op: BACKSLASH !

en in je controller:

```
<?php
require_once("business/bookservice.php");
$lijst = \BookApplication\Service\BookService::getAllBooks();
...
```

Namespaces of niet, merk op dat het importeren van het bestand nog altijd nodig is !

Opgelet: de structuur van de namespaces in PHP heeft op zich niets met de onderliggende directorystructuur te maken, die heel verschillend kan zijn. Voor sommige functionaliteiten binnen PHP, zoals autoloading (waar we later op terugkomen), kan het evenwel handig zijn om de structuur van de namespaces dezelfde te kiezen als die van de directories. Meer daarover later.

... en korter schrijven

Bekijk nog even dit stuk code:

```
<?php
require_once("business/bookservice.php");
$lijst = \BookApplication\Service\BookService::getAllBooks();
...
```

Omdat **BookService** zich in de namespace **\BookApplications\Service** bevindt, moeten we deze context meegeven wanneer we van **BookService** gebruik willen maken. In bovenstaand voorbeeld gebeurt dat één keer. Als we de klasse 10 keer binnen een script willen kunnen gebruiken, zouden we de namespace 10 keer moeten opnieuw schrijven. Het kan ook korter, m.b.v. van het sleutelwoord **use**:

Duid aan in welke namespace BookService gevonden kan worden ...

... en gebruik dan BookService zonder vermelding van namespace.

```
<?php
require_once("business/bookservice.php");
use BookApplication\Service\BookService;
$lijst = BookService::getAllBooks();
...
```

Je hoeft dan slechts één keer per scriptbestand het **use**-sleutelwoord te gebruiken, waarna je de namespace niet meer hoeft te herhalen.

Het gebruik van een alias is toegelaten en kan soms nuttig zijn:

```
<?php
require_once("business/bookservice.php");
use BookApplication\Service\BookService as BServ;
$lijst = BServ::getAllBooks();
...
```

Zo'n alias is vooral handig als je in een situatie terechtkomt waarbij je binnen hetzelfde script zowel de je eigen **BookService** als die van een 3rd-party library wil aanspreken:

```
<?php
require_once("business/bookservice.php");
use Some3rdPartyLibrary\BookService as OtherBServ;
use BookApplication\Service\BookService as MyBServ;
$lijst1 = OtherBServ::doSomething();
$lijst2 = MyBServ::getAllBooks();
...
```

We willen er nogmaals de nadruk op leggen dat het gebruik van namespaces niets te maken heeft met de onderliggende directory-structuur (al **màg** die wel dezelfde zijn), en niets te maken heeft met de "bereikbaarheid" van een klasse. Als je geen **require_once("pad_naar_bestand_bookservice")** inbouwt, zal PHP een foutmelding geven wanneer je de **BookService** probeert aan te spreken, ook al gebruik je namespaces!

Autoloading

Naarmate uitgewerkte projecten toenemen in omvang, krijg je te maken met twee extra problemen.

- 1 Om te beginnen neemt het aantal klassen en dus ook het aantal scriptbestanden toe. Uiteindelijk eindig je met een hele rits aan **require_once**-opdrachten in je controller, want alle klassebestanden moeten gevonden kunnen worden.
- 2 Daarnaast zit je nog met het verschijnsel dat, wanneer je een **require_once**-opdracht gebruikt, dit geïmporteerde bestand sowieso wordt uitgevoerd, los van het feit of de zich daarin bevindende klasse wordt aangeroepen of niet. Kijk even naar onderstaande code, in een bestand *eenklasse.php*:

```
<?php
print("eenklasse.php wordt aangeroepen!");

class EenKlasse {

    public function doeIets() {
    }

}
```

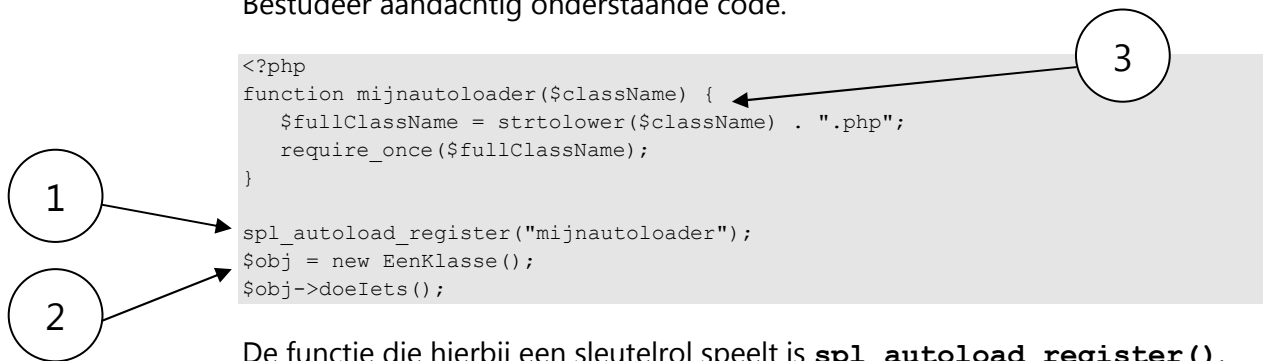
Merk op dat het bestand een **print**-instructie bevat. Maken we nu een ander bestand *test.php* met deze inhoud:

```
<?php
require_once("eenklasse.php");
```

en voeren we dit uit, dan wordt "*eenklasse.php wordt aangeroepen!*" op het scherm afgebeeld. Uiteraard zullen we in de praktijk geen code buiten de klasse schrijven, maar je kan hieruit wel opmaken dat de inhoud van het bestand **eenklasse.php** wordt uitgevoerd en geanalyseerd, zonder dat we de klasse **EenKlasse** effectief ergens gebruikt hebben. Het ware beter dat dit bestand pas zou uitgevoerd worden, op het ogenblik dat we **EenKlasse** voor het eerst aanspreken.

De oplossing van beide problemen ligt in een techniek die men **autoloading** noemt. Bij autoloading laten we alle **require_once**-opdrachten achterwege, en laten we ingrijpen in op het moment dat de PHP engine op het punt staat een fout op te werpen omdat een klasse niet gevonden kan worden.

Bestudeer aandachtig onderstaande code.



De functie die hierbij een sleutelrol speelt is `spl_autoload_register()`.

Deze heeft een parameter, die in stringvorm de naam bevat van de functie die zal worden uitgevoerd alvorens een Class Not Found-fout op te werpen.

Concreet gebeurt het volgende:

In ❶ registreert de engine de functie `mijnautoloader` als een autoloading-functie. In ❷ roepen we `EenKlasse` aan, die onbekend is bij de engine. Deze werpt evenwel nog geen exception op, maar voert eerst de functie `mijnautoloader` uit (❸). De parameter `$className` bevat op dat ogenblik de stringwaarde "`EenKlasse`". We zetten deze stringwaarde om naar kleine letters, en voegen er de extensie `.php` aan toe, en wat we bekomen is de bestandsnaam die we in een `require_once` opnemen.

Echter, deze eenvoudige vorm is weinig flexibel. Je kan bijvoorbeeld `EenKlasse` niet opnemen in een namespace `\MijnProject`. Als we dit zouden doen, en vervolgens

```
<?php
use MijnProject\EenKlasse;

function mijnautoloader($className) {
    $fullClassName = strtolower($className) . ".php";
    require_once($fullClassName);
}

spl_autoload_register("mijnautoloader");
$obj = new EenKlasse();
$obj->doeIets();
```

zouden uitvoeren, dan zouden we een foutmelding krijgen. `$className` bevat dan immers niet meer "`EenKlasse`", maar "`MijnProject\EenKlasse`", zijnde de volledige namespace + de klassenaam.

We zouden dit kunnen oplossen door de waarde van de parameter `$className` te gaan analyseren, uit elkaar te halen en enkel het laatste gedeelte te gebruiken.

Dat hebben we in onderstaande code dan ook gedaan.

```
<?php
use MijnProject\EenKlasse;

function mijnautoloader($className) {
    $classNameParts = explode('\\', $className);
    $lastPart = end($classNameParts);
    $fullClassName = strtolower($lastPart) . ".php";
    require_once($fullClassName);
}

spl_autoload_register("mijnautoloader");
$obj = new EenKlasse();
$obj->doeIets();
```

Dit werkt wel, en is al iets flexibeler, maar hiermee zijn we er toch nog niet helemaal. Bovenstaande code verplicht je immers om alle klassebestanden in dezelfde directory als waar het controllerbestand staat onder te brengen, vermits de **require_once**-opdracht enkel daarin zal zoeken naar het gevraagde bestand.

Een betere versie van onze autoloader zou bijvoorbeeld de namespace zelf kunnen bekijken en op basis daarvan beslissen in welke directory de bestanden gezocht moeten worden.

```
if ($namespace == "MijnProject\service") include_once("business/" . $filename);
else if ($namespace == "MijnProject\data") include_once("data/" . $filename);
else if ($namespace == "MijnProject/entities") include_once("entities/" . $filename);
```

Op die manier zullen alle klassen gevonden worden, op voorwaarde dat je ze onderbrengt in de juiste namespace en in de juiste directory.

Je merkt dat er op deze manier een verband gelegd is tussen namespaces enerzijds, en de directorystructuur anderzijds. We zouden de lijn kunnen doortrekken en de namespaces dezelfde structuur kunnen geven als die van de directories waarin de bestanden staan. Uit het voorgaande stuk weet je reeds dat dit niet verplicht is, maar wel mag. Op die manier bekomen we een flexibelere autoloader.

Autoloaders allerhande

Vanzelfsprekend bestaan er reeds heel wat kant-en-klare autoloaders op de markt. Allen doen ze uiteindelijk hetzelfde, alleen de afspraken en conventies verschillen.

Het is belangrijk dat je weet wat autoloading is en hoe het werkt. In praktische projecten zal je wellicht gebruik maken van 3rd-party autoloaders. Het warme

water steeds opnieuw uitvinden heeft weinig zin, en deze autoloaders zijn al zeer efficiënt opgebouwd, en hebben hun snelheid bewezen.

Bij wijze van voorbeeld zullen we de autoloader van Doctrine even onder de loep nemen. Deze ClassLoader (zoals hij noemt) rekent er op dat je namespace-structuur een weerspiegeling is van de directory-structuur van je project. Daarom gaan we onze directories die we in de voorgaande hoofdstukken gehanteerd hebben een ietsje wijzigen, en wel als volgt:

```
xampp/  
 htdocs/  
    mijnproject/  
      src/  
        VDAB/  
          MijnProject/  
            Business/  
              BookService.php  
            Data/  
              BookDAO.php  
            Entities/  
              Book.php  
            Exceptions/  
              BookNotFoundException.php  
  
        toonalleboeken.php  
        voegboektoe.php  
        ...
```

Een aantal zaken vallen op. Omdat we onze namespaces het liefst laten beginnen met een hoofdletter (conventioneel in de PHP-wereld), start ook elke directorynaam met een hoofdletter. De ClassLoader zal ook de exacte naam van de klasse gebruiken om het overeenkomstig bestand te vinden, dus ook de bestandsnamen volgen deze conventie (bvb de klasse **BoekService** staat in een bestand **BoekService.php**). Voor de controllers kunnen we blijven gebruik maken van kleine letters; het zijn immers geen klassen.

Wat ook nog opvalt is dat we onze mappenstructuur naar een dieper gelegen niveau verplaatst hebben. In plaats van rechtstreeks onder de rootmap staan deze nu onder VDAB/MijnProject. Dit zal dan ook de hoofdnamespace zijn die we gebruiken in ons project. Al onze zelfgemaakte klassen worden verzameld onder de *src/* directory.

Tot slot moeten we er voor zorgen dat de namespaces in alle klassen netjes ingevuld zijn. Dat betekent:

- Voor alle klassen van de servicelaag:
namespace VDAB\MijnProject\Business;
- Voor alle klassen van de data laag:
namespace VDAB\MijnProject\Data;
- Voor alle entity-klassen:
namespace VDAB\MijnProject\Entities;

Enzovoort.

Denk eraan dat geen enkele klasse nog de functie **require_once** aanroept (we gaan immers autoloading gebruiken). Uitzondering op de regel zijn natuurlijk de **require**- of **include**-instructies die geen klasse importeren, maar een los stuk code, zoals bvb de **include** van een presentatiepagina binnen de controller.

Nu de projectstructuur geprepareerd is, is het tijd om er de Doctrine ClassLoader op los te laten. Je kan deze downloaden op

<https://github.com/doctrine/common/archive/master.zip>

Pak dit bestand uit, en kopiëer de directory

common-master\lib\Doctrine

integraal onder de root-map van je eigen project. Je krijgt dus:

```
xampp/  
 htdocs/  
    mijnproject/  
      src/  
        VDAB/  
          MijnProject/  
            Business/  
              BookService.php  
            Data/  
              BookDAO.php  
            Entities/  
              Book.php  
            Exceptions/  
              BookNotFoundException.php  
  
          toonalleboeken.php  
          voegboektoe.php  
          ...  
          Doctrine/  
            Common/  
              ...
```

Ook de Doctrine library zelf maakt gebruik van namespaces...

De enige klasse die we nog expliciet vanuit de controller met een **require_once** zullen importeren is uiteraard de ClassLoader zelf. Onderstaande controller maakt gebruik van de Doctrine ClassLoader om de benodigde klassen te autoloaden. Bestudeer deze aandachtig.

```
<?php  
use VDAB\MijnProject\Business\BookService;  
use Doctrine\Common\ClassLoader;  
require_once("Doctrine/Common/ClassLoader.php");  
$classLoader = new ClassLoader("VDAB", "src");  
$classLoader->register();  
$alleBoeken = BookService::getAllBooks();
```

Zoals je ziet werkt het gebruik van de ClassLoader heel eenvoudig. Je maakt er een nieuw object van, en geeft aan de constructors twee parameters mee. De eerste parameter is de top-namespace waar deze classloader voor verantwoordelijk is (je mag er immers meer dan één aanmaken). De tweede parameter is de top-directory waaronder de classloader moet afdalen om de gezochte klassen te kunnen vinden (in ons geval is dat de *src/* map).

Tot slot registreer je de classloader met de functie **register**.

Vooraleer de PHP engine een fout opwerpt omdat de klasse **\VDAB\MijnProject\Business\BookService** niet gevonden kan worden, treedt de autoloading van de Doctrine ClassLoader in werking en vindt het benodigde bestand *VDAB/MijnProject/Business/BookService.php*, hoewel we nooit zelf een **include** of **require** hiervan gedaan hebben.

Als we de **BookService** niet gebruiken, zal het bestand ook nooit geïmporteerd worden, wat performant en efficiënt werkt.

Hoofdstuk 11

Templating

In dit hoofdstuk:

- ✓ Het toepassen van templating op presentatiepagina's

De tegemoetkoming

Je hebt ondertussen onder de knie hoe je toepassingen ontwikkelt volgens een MVC-patroon. Waar we nog geen extra aandacht aan besteed hebben is de manier waarop gegevens worden getoond. Je weet reeds dat de gegevens die in een presentatiepagina worden getoond opgehaald en "klaar" werden gezet door de controller die deze pagina geïmporteerd heeft.

Werp een blik op deze presentatiepagina:

```
<!DOCTYPE HTML>
<html>
  <head>
    ...
  </head>
  <body>
    <h1>Boekenlijst</h1>
    <table>
      <tr>
        <th>Titel</th>
        <th>Genre</th>
      </tr>
      <?php
        foreach ($boekenLijst as $boek) {
          ?>
          <tr>
            <td>
              <?php print($boek->getTitel()); ?>
            </td>
            <td>
              <?php
                print($boek->getGenre()->getOmschrijving());
              ?>
            </td>
          </tr>
        }
      <?php
    </table>
  </body>
</html>
```

Het is mogelijk dat de persoon die dit soort presentatiepagina's moet ontwikkelen weinig kaas gegeten heeft van PHP (bvb omdat het in de eerste plaats een designer is). We hebben al ons best gedaan om de variabelen die de te tonen data bevatten netjes klaar te zetten, zodat ze met relatief eenvoudige print-instructies kunnen afgedrukt worden. Maar een nog betere tegemoetkoming naar deze persoon toe zou zijn om alle PHP-tags achterwege te kunnen laten.

Dit wordt het uiteindelijke doel:

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=utf-8>
    <title>Boeken</title>
  </head>
  <body>
    <h1>Boekenlijst</h1>
    <table>
      <tr>
        <th>Titel</th>
        <th>Genre</th>
      </tr>
      {% for boek in boekenLijst %}
        <tr>
          <td>
            {{ boek.titel }}
          </td>
          <td>
            {{ boek.genre.omschrijving }}
          </td>
        </tr>
      {% endfor %}
    </table>
  </body>
</html>
```

De dollar-teken-aanduiding voor variabelen is verdwenen, net als alle PHP-tags en PHP-functies. De logicablokken zijn vervangen door zgn. template-tags, en de properties van variabelen zijn (schijnbaar) toegankelijk zonder getter-functies (hier komen we zo dadelijk nog op terug).

Om dit te kunnen bereiken maken we gebruik van een **templating engine**. Een templating engine heeft als doel een presentatiepagina in te lezen, te evalueren, te interpreteren en opnieuw uit te voeren. De tags die zorgen voor extra mogelijkheden (zoals een for-constructie, een if-constructie, een schrijf-opdracht) binnen een template (vanaf nu zal onze presentatiepagina door het leven gaan als "template") zijn dan ook specifiek voor een bepaalde templating engine.

Templating engines zijn er in verschillende smaken. Voor deze cursus is gekozen voor Twig, een snelle en lightweight bibliotheek die de klus perfect klaart.

Zoals je kan zien in het vorige template worden de tags {% %} en {{ }} door Twig herkend en geïnterpreteerd. We geven enkele voorbeelden

Zo maak je een foreach lus m.b.v.

```
{% for boek in boekenLijst %}  
...  
{% endfor %}
```

Een if-constructie bereik je met:

```
{% if boek.titel == 'De Naam van de Roos' %}  
...  
{% endif %}
```

Wil je een de inhoud van een variabele uitschrijven, dan gebruik je

```
<p>Welkom, {{ gebruikersnaam }}</p>
```

! Let op de dubbele accolades {{ en }} bij het uitschrijven, t.o.v. de Twig-operaties, die worden omgeven door {% en %}.

Gebruik de punt-operator om een property van een variabele aan te spreken:

```
<p>Welkom, {{ gebruiker.naam }}</p>
```

Twig controleert eerst of **naam** een geldige, toegankelijke property is van het object **gebruiker**. Zoniet controleert het of **naam** een geldige functie is van het object **gebruiker**. Zoniet controleert het of **getNaam()** een geldige functie is. Zoniet controleert het of **isNaam()** een geldige functie is.

Je kan dus zonder expliciete vermelding van de prefix "get" een property aanspreken.

Hoe werkt het?

Zonder de Twig-bibliotheek te importeren in ons project zullen we niet ver springen. Ook de variabelen die beschikbaar zijn in de controller zullen niet meer zonder meer gebruikt kunnen worden in het template, zoals je vroeger wel kon doen in "gewone" PHP-views. Je moet uitdrukkelijk aangeven welke variabelen je onder welke naam ter beschikking wilt hebben.

Om te beginnen halen we de bibliotheek van Twig in huis. Dit kan op verschillende manieren (deze vind je op de website van Twig:

<http://twig.sensiolabs.org/>). Wij zullen hier gebruik maken van een gewone download:

<https://github.com/fabpot/Twig/archive/master.zip>.

In de map *Twig-master/lib* vind je een map *Twig* terug. Deze mag je uitpakken naar de map van je project. Het liefst maak je een subdirectory *libraries* onder de root van je project, met daarin alle externe bibliotheken die door je toepassing gebruikt worden. Je krijgt dan:

```
xampp/  
  htdocs/  
    mijnproject/  
      libraries/  
        Twig/  
          Autoloader.php  
          Compiler.php  
          ...
```

De controller die de boekenlijst ophaalt kan er bvb als volgt uitzien:

```
<?php  
require_once("libraries/Twig/Autoloader.php");  
require_once("service/BoekenService.php");  
  
Twig_Autoloader::register();  
  
$boeken = BoekenService::getAlleBoeken();  
  
$loader = new Twig_Loader_Filesystem("presentation");  
$twig = new Twig_Environment($loader);  
$view = $twig->render("boeken.twig", array( "boekenLijst" => $boeken ));  
print($view);
```

Twig heeft een eigen autoloader voor zijn klassen. Je moet enkel een **require_once** doen van de autoloader zelf (*Autoloader.php*), en deze activeren met

```
Twig_Autoloader::register();
```

Om de templates van je webapplicatie terug te kunnen vinden maakt Twig gebruik van een loader. Er bestaan verschillende loaders, maar hier maken we een loader die templates op het bestandssysteem kan ophalen:

```
$loader = new Twig_Loader_Filesystem("presentation");
```

We duiden ineens de directory aan (vanaf de root van de applicatie, want daar bevindt zich de controller waar we in werken) waar Twig de templates terug kan vinden.

Vervolgens maken we met

```
$twig = new Twig_Environment($loader);
```

een Twig-omgeving met een bepaalde configuratie aan, op basis van de loader die we net gedefiniëerd hebben. Deze configuratie bevat een aantal parameters die je kan wijzigen, maar voorlopig volstaan voor ons de standaardwaarden. Raadpleeg de Twig-documentatie als je hier meer wilt over weten (<http://twig.sensiolabs.org/doc/api.html#environment-options>).

De regel

```
$view = $twig->render("boeken.twig", array( "boekenLijst" => $boeken ));
```

zet Twig effectief aan het werk. Het templatebestand *boeken.twig* (let op de extensie *.twig*) wordt ingelezen en geïnterpreteerd. Merk op dat, hoewel dit bestand zich in de map *presentation* bevindt, we dit niet meer expliciet vermelden. We hebben deze informatie immers reeds meegegeven tijdens het maken van de Twig-loader.

De functie **render** heeft nog een tweede parameter: een associatieve array die een mapping bevat van alle variabelen die je in het template wilt kunnen gebruiken. Zoals we reeds eerder aangehaald hebben kan je de variabele **\$boeken** niet zomaar uitlezen in het template: je moet deze expliciet meegeven. In een notedop zorgt bovenstaande regel ervoor dat in het template *boeken.twig* een variabele **boekenLijst** ter beschikking is, die dezelfde inhoud heeft als de variabele **\$boeken** in onze controller.

Het resultaat van de functie **render** is de inhoud van de hele presentatiepagina, nadat deze geïnterpreteerd is door Twig. Het enige wat ons nu nog te doen staat is deze inhoud uit te schrijven in het HTTP-antwoord:

```
print($view);
```

Meer mogelijkheden

Het zou weinig zinvol zijn alle mogelijkheden van Twig uitgebreid uit de doeken te doen in deze cursus. De officiële documentatie van Twig is uitstekend opgebouwd en volstaat ruimschoots om complexere templates te kunnen schrijven.

Je vindt deze documentatie op <http://twig.sensiolabs.org/documentation>. Je kan ze eveneens als PDF-document downloaden.

Lees minstens het hoofdstuk "Twig for Template Designers" zeer goed door. Voor zij die steviger de tanden in Twig willen zetten kunnen we het hoofdstuk "Twig for Developers" ten eerste aanbevelen.

Naast Twig bestaan er nog meer templating engines: Smarty, Dwoo, Savant3, Rain TPL, enz... Elk hebben ze hun eigen manier van werken. Twig is voornamelijk interessant omdat het hand in hand gaat met een van de meest populaire MVC-frameworks voor PHP: Symfony2, maar dat verhaal is voor een andere keer...

Appendix A

Installatie

Om met PHP te kunnen werken en experimenteren zijn volgende elementen nodig:

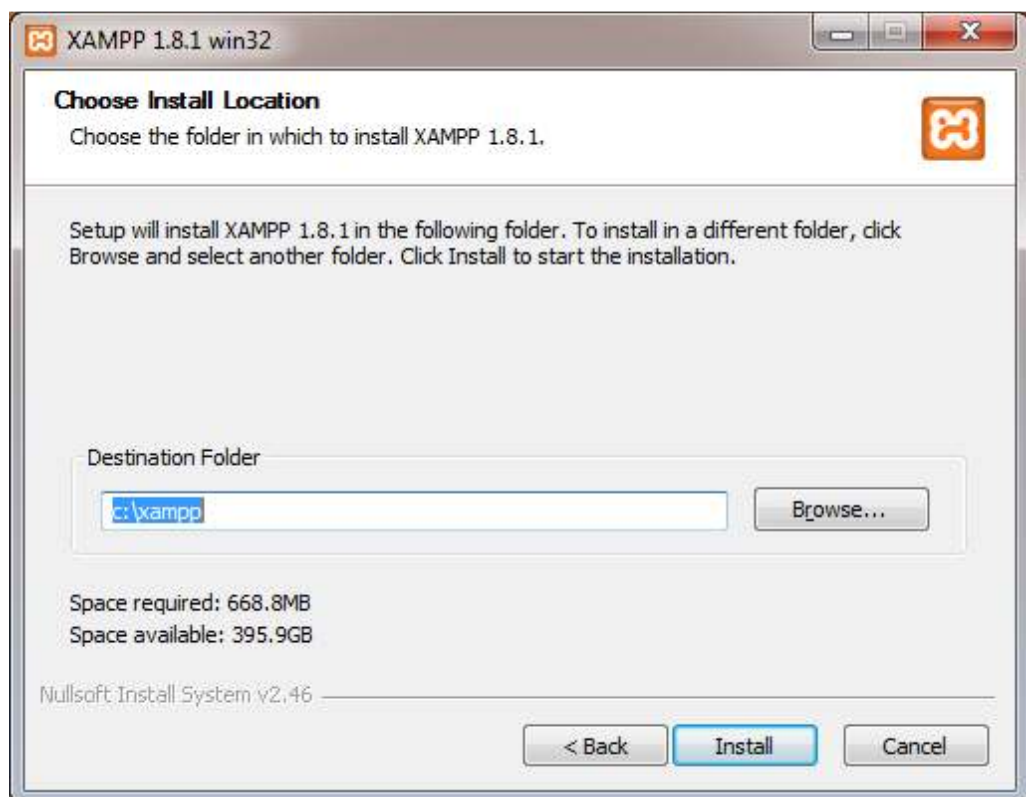
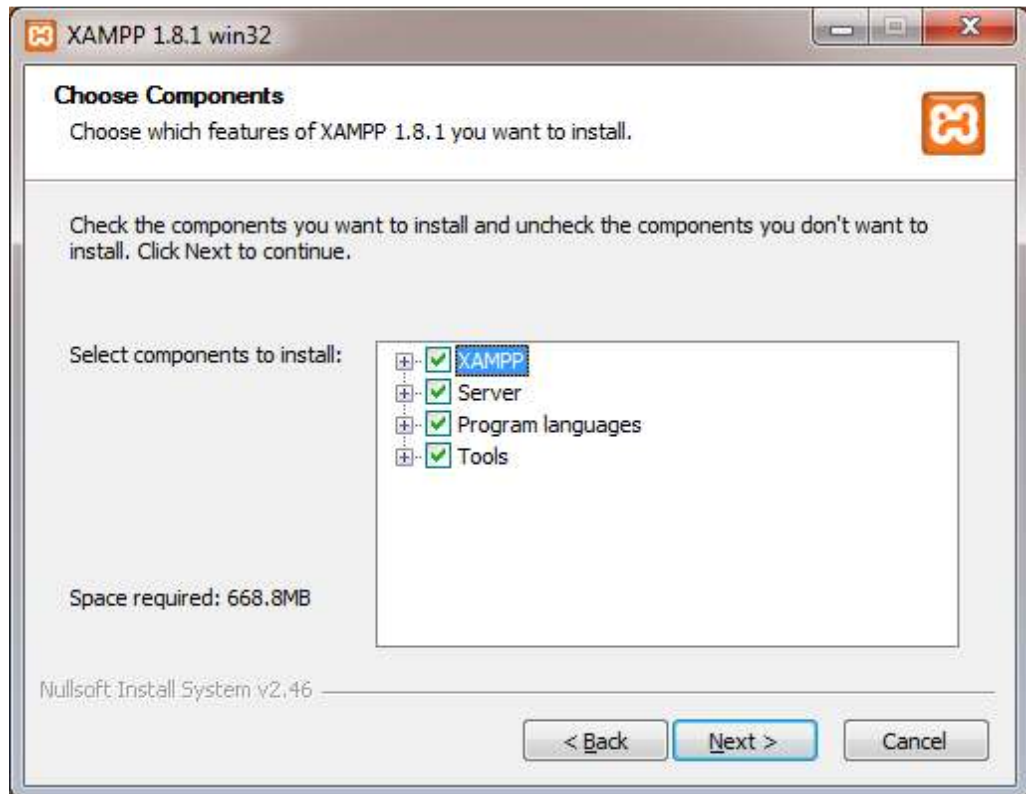
- Een webserver (Apache, Internet Information Server, ...)
- De PHP scripting engine software, die ondersteuning voor PHP biedt
- Een browser om je applicaties uit te testen
- Eventueel extra bronnen (databanken, bestanden, ...)

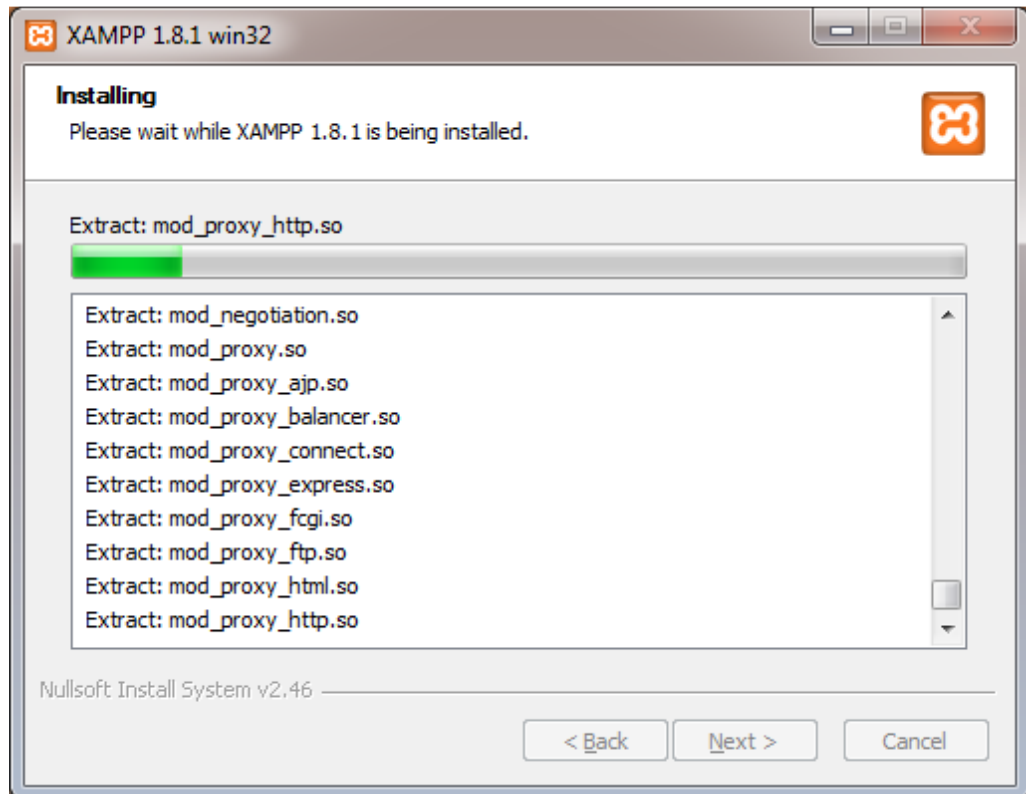
Voor deze cursus opteren we voor een all-in-one oplossing. XAMPP is een gratis totaalpakket dat in één installatieprocedure de Apache webserver, PHP, MySQL en PhpMyAdmin (een applicatie om MySQL databanken via een webinterface te beheren) installeert.

De laatste versie van XAMPP is steeds te vinden op

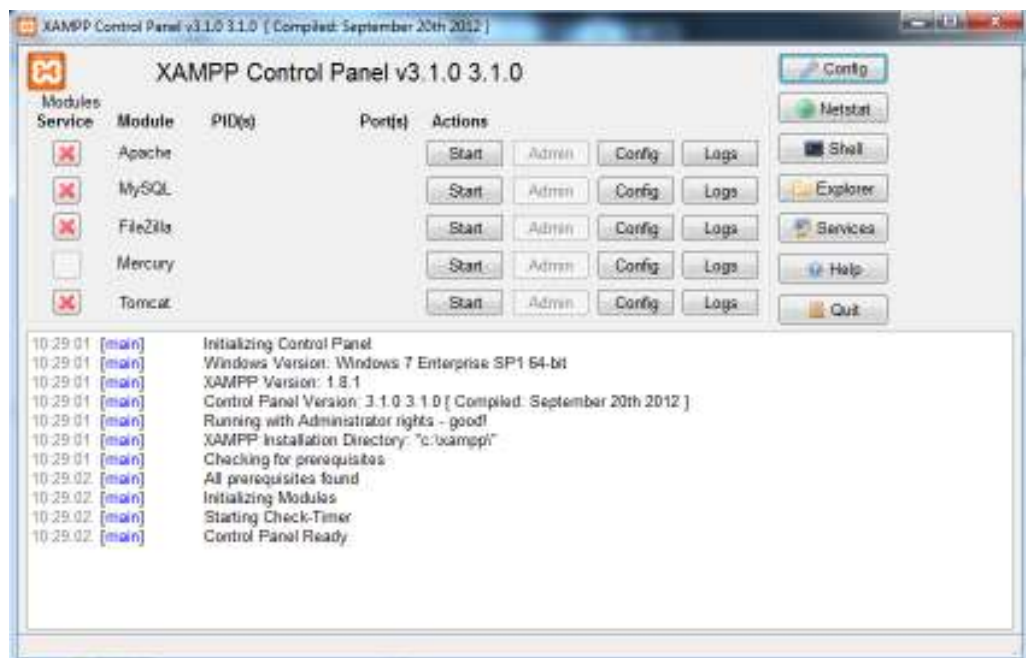
<http://www.apachefriends.org/en/xampp.html>

Download en installeer het "Basic Package" in zelfuitpakkend EXE-formaat.





Na de installatie kan je het controlepaneel van XAMPP rechtstreeks openen door bevestigend te antwoorden op de laatste vraag die je wordt gesteld.



Klik éénmaal op de knop Start rechts van het woord "Apache" (dit start de webserver), en éénmaal op de knop Start van het woord MySQL (dit start de MySQL DBMS). Indien Windows je in een waarschuwing komt vragen of deze programma's een poort mogen openzetten, laat je dit toe.

Noot: Bij een volgende heropstart van je computer dien je dit controlepaneel wederom te openen (er is een snelkoppeling beschikbaar) en de gewenste servers opnieuw te activeren.

We testen nu de omgeving. Open een browser naar keuze en surf naar

`http://localhost/`



Krijg je bovenstaand scherm, dan is de installatie succesvol verlopen. Klik op "Nederlands".

Als volgende stap maken we de databank, nodig voor deze cursus, klaar voor gebruik. Klik op in het navigatiemenu links op "phpMyAdmin". Kies uit het menu bovenaan "Importeer".



In de sectie "Te importeren bestand", kies je het bestand **cursusphp.sql** uit de oefenmap en klik op de knop Start links onderaan.

Controleer of de databank en de nodige tabellen aangemaakt zijn. Links in het scherm is een nieuwe databank te zien met de naam "cursusphp". Klik hierop om in het rechtergedeelte de verschillende tabellen te zien verschijnen.

Maak in de map `C:\xampp\htdocs` een submap aan met je eigen naam. Bij het maken van de oefeningen plaats je je code in je eigen map. Op die manier voorkom je dat je eigen bestanden gaat plaatsen tussen bestanden die XAMPP zelf nodig heeft om correct te werken.

Om je eigen pagina's en scripts in deze submap te kunnen testen, open je een browser en surf je naar `http://localhost/naamvandesubmap`. Je krijgt dan een bestandslijst te zien die de inhoud van deze map voorstelt. Klik op het gewenste bestand om het te testen.

COLOFON

Sectorverantwoordelijke:	
Cursusverantwoordelijke:	Paul Kiekens
Didactiek:	Paul Kiekens
Lay-out:	Paul Kiekens
Medewerkers:	
Versie:	27/05/13