Object georiënteerd programmeren: De basis uitgelegd

PHP en Classes

Na functies is het een kleine stap naar een class (klasse). Een class is namelijk een verzameling van functies. Dit klinkt misschien vrij abstract en vrijblijvend. En nee, een class hoort niet 30 willekeurige functies te bevatten. Om het minder abstract te maken: een class heeft samenhang.

Wat is een class?

Een class in PHP is een blauwdruk voor het creëren van objecten. Het definieert de eigenschappen (variabelen) en methoden (functies) die de objecten zullen hebben. Met classes kun je objectgeoriënteerd programmeren (OOP) toepassen, wat helpt bij het structureren en organiseren van je code. Een class kan ook de toegang tot zijn eigenschappen en methoden beperken met toegangsmodificatoren (public, protected, private). Daarnaast kunnen classes gebruikmaken van inheritance, abstracte klassen, interfaces, traits, en readonly eigenschappen voor extra functionaliteit en herbruikbaarheid.

Stel nu dat we een 'Medewerker' hebben. Dit is iets met bepaalde eigenschappen zoals: uurloon en aantal uren. Voor deze medewerker kunnen we bepalen hoeveel hij kost voor x aantal uren. Dit laatste is een 'functie'. In de context van een class noemen we dit een methode.

```php
<?php
class Medewerker {
    public $uurloon = 10;
    public $aantal_uren = 40;

    public function weekloon() {
       return $this->uurloon * $this->aantal_uren;
    }
}
```

Dit lijkt vrij veel op de functie die eerder beschreven is. We hebben een *class*, met daarachter de naam van de class: Medewerker. De inhoud van deze class staat tussen de `{` en `}`.

We zien vervolgens twee dingen: 'variabelen' en 'functies'. De variabelen zijn in dit geval attributen (properties) en de functies worden methodes (methods) genoemd.

Iets anders wat opvalt is de variabele `$this`. Deze variabele is een verwijzing naar de class zelf. Door middel van `$this` is het mogelijk om attributen en methodes binnen de class zelf aan te roepen.

Daarnaast hebben we '`public`'. Dit geeft de zichtbaarheid van een attribuut of methode aan. Andere opties zijn `protected` en `private`. Hierover verderop meer uitleg.

Class instantiëren

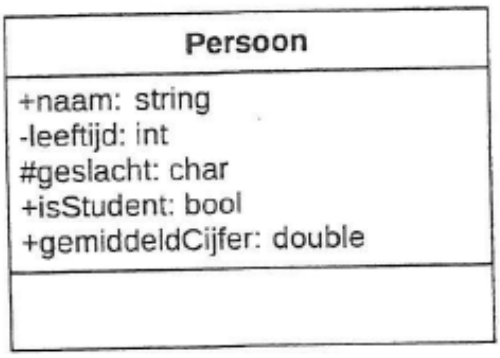
Het hebben van een class is leuk, maar we kunnen er niet nog iets mee. We moeten het eerst instantiëren en dan kunnen we de publieke functionaliteit ervan aanroepen. Als een class eenmaal is geïnstantieerd dan noemen we deze instantie ook wel het object.

```php
<?php
$medewerker = new Medewerker();
echo $medewerker->weekloon();
?>
```

Op het scherm wordt 400 afgedrukt!

Opgave 1

Maak een nieuwe map met de naam *OOPHP*. Open je editor en codeer de klasse uit de volgende figuur als **Persoon.php** in de map *OOPHP*.



Figuur 4.4 Klassendiagram

```
<?php
{
  class Persoon
  {
    public $naam;
    private $leeftijd;
    protected $geslacht;
    public $isStudent;
    public $gemiddeldCijfer;
  }
}
```

Constructors en Destructors

Constructors zijn speciale methoden die automatisch worden aangeroepen wanneer een object van een klasse wordt gemaakt. Ze worden gebruikt om het object te initialiseren.

```php
<?php
class MijnKlasse {
    public function __construct() {
        echo "Het object is aangemaakt.";
    }
}

$object = new MijnKlasse(); // Output: Het object is aangemaakt.
?>
```

Destructors worden aangeroepen wanneer een object wordt vernietigd of de scriptuitvoering eindigt. Ze worden gebruikt om op te ruimen.

```php
<?php
class MijnKlasse {
    public function __destruct() {
        echo "Het object is vernietigd.";
    }
}

$object = new MijnKlasse();
// Output: Het object is vernietigd wanneer het script eindigt.
?>
```

Vervang **'\n'** met **
**

Opgave 2

Ga naar je klasse Persoon en voeg de volgende constructor eraan toe. De code voor een enter is \n.

```
<?php
{
  class Persoon
  {
    public $naam;
    private $leeftijd;
    protected $geslacht;
    public $isStudent;
    public $gemiddeldCijfer;
    // constructor methode
    function __construct(string $naam, int $leeftijd, string $geslacht)
    {
      $this->naam = $naam;
      $this->leeftijd = $leeftijd;
      $this->geslacht = $geslacht;
      echo "\n Nieuw Persoon-object aangemaakt";
      echo "\n De property naam is " . $this->naam;
    }
  }
}
```

Opgave 3

Maak een nieuw bestand en sla het op met als naam **Program.php** in je map *OOPHP*. Voeg de volgende code eraan toe.

```
<?php
include 'Persoon.php';
$umut = new Persoon("Umut", 18, "M");
$demirel = new Persoon("Demirel",23,"M");
```

Output

```
Nieuw Persoon-object aangemaakt
De property naam is Umut
Nieuw Persoon-object aangemaakt
De property naam is Demirel
```

Opgave 4

Open de klasse `Persoon` en voeg de volgende code eraan toe.

```
function __destruct()
{
  // voer hier de benodigde acties uit;
  echo "\n Persoon object $this->naam wordt verwijderd";
}
```

Output

```
Nieuw Persoon-object aangemaakt
De property naam is Umut
Nieuw Persoon-object aangemaakt
De property naam is Demirel
Persoon object Demirel wordt verwijderd
Persoon object Umut wordt verwijderd
```

Methoden in Classes

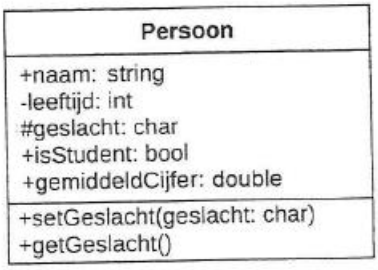
Methoden zijn functies die binnen een klasse worden gedefinieerd en gebruikt om het gedrag van objecten te definiëren. Methoden kunnen dezelfde acties uitvoeren als gewone functies, maar hebben toegang tot de eigenschappen van het object.

```php
<?php
class Auto {
    public $merk;
    public $model;

    public function __construct($merk, $model) {
        $this->merk = $merk;
        $this->model = $model;
    }

    public function beschrijf() {
        return "Deze auto is een $this->merk $this->model.";
    }
}

$auto = new Auto("Toyota", "Corolla");
echo $auto->beschrijf(); // Output: Deze auto is een Toyota Corolla.
?>
```

In dit voorbeeld definiëren we de methode `beschrijf` binnen de klasse `Auto`. Deze methode heeft toegang tot de eigenschappen `$merk` en `$model` van het object.

Opgave 6

Ga naar de klasse Persoon en codeer na destruct de get()- en set()-methodes uit het volgende klassendiagram:



Figuur 4.6 Klassendiagram met methodes

```
function setGeslacht(string $geslacht)
{
  $this->geslacht = $geslacht;
}
function getGeslacht()
{
  return $this->geslacht;
}
```

De $this-pointer

Omdat we de namen van de objecten niet van tevoren kunnen weten, gebruiken we `$this`-pointer. `$this` pointer is een verwijzing naar de naam van het aangemaakte object. Bijvoorbeeld, als we de volgende `getNaam()`-methode aanroepen binnen het `demirel`-object, verwijst `$this` naar de property `$naam` in het `demirel`-object.

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

Opgave 7

Ga naar de klasse `Persoon` en maak `get`- en `set`-methodes voor de volgende properties:

```
setLeeftijd();
getLeeftijd();
```

Opgave 8

Open **Program.php** en voeg de volgende code toe. Roep de `setLeeftijd()`-methode van het object `demirel` aan en verander de leeftijd in 24.

```
$demirel->setLeeftijd(24);
echo "\n De leeftijd van Demirel is: " . $demirel->getLeeftijd();
```

Output

```
De leeftijd van Demirel is: 24
```

Properties in Classes

Eigenschappen (properties) zijn variabelen die binnen een klasse worden gedefinieerd en die de staat van objecten beschrijven. Properties worden meestal geïnitialiseerd wanneer een object wordt gemaakt en kunnen publiek, beschermd of privé zijn.

```php
<?php
class Persoon {
    public $naam;
    protected $leeftijd;
    private $geslacht;

    public function __construct($naam, $leeftijd, $geslacht) {
        $this->naam = $naam;
        $this->leeftijd = $leeftijd;
        $this->geslacht = $geslacht;
    }

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

$persoon = new Persoon("John", 30, "Mannelijk");
echo $persoon->toonNaam(); // Output: John
?>
```

In dit voorbeeld definiëren we drie eigenschappen (`$naam` , `$leeftijd` , `$geslacht`) binnen de klasse `Persoon` . De eigenschappen kunnen verschillende toegangsmodificatoren hebben: public, protected en private. Public eigenschappen zijn toegankelijk buiten de klasse, protected eigenschappen zijn alleen toegankelijk binnen de klasse en afgeleide klassen, en private eigenschappen zijn alleen toegankelijk binnen de klasse zelf.

Access Modifiers (Toegangsmodificatoren)

Het is mogelijk om invloed uit te oefenen op de zichtbaarheid van attributen en methodes. Dit doen we met zogeheten toegangsmodificatoren. Over het algemeen zal er meestal gekozen worden voor `public` functionaliteit. Deze functionaliteit is benaderbaar binnen en buiten de class instantie.

Alternatieven zijn `protected` en `private` . Door een attribuut of methode `protected` te maken zorg je ervoor dat deze alleen bij de class zelf en bij de class die erft (inherit) beschikbaar is. Een `private` attribuut of methode is alleen toegankelijk voor de class zelf en dus niet daarbuiten.

Public

Een `public` eigenschap of methode is toegankelijk vanuit elke context.

```php
<?php
class MijnKlasse {
    public $publiekeEigenschap = "Ik ben publiek!";
}

$object = new MijnKlasse();
echo $object->publiekeEigenschap; // Output: Ik ben publiek!
?>
```

Private

Een `private` eigenschap of methode is alleen toegankelijk binnen de klasse zelf.

```php
<?php
class MijnKlasse {
    private $privéEigenschap = "Ik ben privé!";

    public function toonEigenschap() {
        return $this->privéEigenschap;
    }
}

$object = new MijnKlasse();
echo $object->toonEigenschap(); // Output: Ik ben privé!
?>
```

Datatypes in PHP: Een complete overzicht

In PHP zijn datatypes essentieel omdat ze bepalen welke soorten gegevens je kunt opslaan en welke bewerkingen je erop kunt uitvoeren. PHP is een dynamisch getypeerde taal, wat betekent dat variabelen geen vaste typen hebben en je geen datatype hoeft te declareren wanneer je een variabele aanmaakt.

Het datatype van een variabele wordt automatisch bepaald door de waarde die eraan wordt toegekend. Hieronder volgt een uitgebreide uitleg van de verschillende datatypes in PHP.

1. Strings

Een string is een reeks tekens, zoals tekst. Strings kunnen worden gedefinieerd met enkele of dubbele aanhalingstekens.

```php
<?php
$enkele_aanhalingstekens = 'Hallo wereld';
$dubbele_aanhalingstekens = "Hallo wereld";
?>
```

Strings kunnen ook variabelen bevatten als je dubbele aanhalingstekens gebruikt:

```php
<?php
$naam = "John";
$bericht = "Hallo, mijn naam is $naam";
echo $bericht; // Output: Hallo, mijn naam is John
?>
```

2. Integers

Een integer is een getal zonder decimalen. Het kan positief of negatief zijn.

```php
<?php
$positief = 42;
$negatief = -42;
?>
```

3. Floats

Een float (ook bekend als double) is een getal met decimalen.

```php
<?php
$float = 3.14;
?>
```

4. Booleans

Een boolean vertegenwoordigt een waar (true) of onwaar (false) waarde.

```php
<?php
$waar = true;
$onwaar = false;
?>
```

5. Arrays

Een array is een variabele die meerdere waarden kan opslaan. Arrays kunnen zowel numerieke indexen als associatieve sleutels gebruiken.

```php
<?php
$numerieke_array = [1, 2, 3];
$associatieve_array = ["a" => 1, "b" => 2, "c" => 3];
?>
```

6. Objecten

Een object is een instantie van een klasse. Klassen zijn blauwdrukken voor objecten die eigenschappen en methoden kunnen bevatten.

```php
<?php
class Persoon {
    public $naam;
    public function zetNaam($naam) {
        $this->naam = $naam;
    }
}

$persoon = new Persoon();
$persoon->zetNaam("John");
echo $persoon->naam; // Output: John
?>
```

7. Null

Null is een speciaal datatype dat een variabele zonder waarde vertegenwoordigt.

```php
<?php
$variabele = null;
?>
```

Type juggling en type casting

PHP doet automatisch type juggling, dit betekent dat het datatype van een variabele kan veranderen afhankelijk van de context.

```php
<?php
$waarde = "5" + 10; // $waarde is nu 15 (integer)
?>
```

Je kunt ook expliciet type casten om een variabele naar een specifiek datatype te converteren.

```php
<?php
$waarde = (int)"5"; // $waarde is nu 5 (integer)
?>
```

Datatypes wijzigen

We kunnen het ene datatype omzetten in een ander datatype.

Type	Omschrijving
(int)	Omzetten naar integer (zonder decimalen)
(bool)	Omzetten naar true of false
(float) of (double)	Omzetten naar getal met drijvende komma
(string)	Omzetten naar tekst
(array)	Omzetten naar array
(Object)	Omzetten naar object

Tabel 4.2 Omzetten

Hier volgt een voorbeeld:

```
$doubleCijfer = 7.6;
$intCijfer = (int)$doubleCijfer; // intCijfer is nu 7
```

Let op

Hier hebben we een grotere double omgezet in een kleinere int. De decimale informatie gaat verloren.

Het return type: wat geeft een methode terug

In de recentere PHP-versies is het mogelijk om aan te geven wat voor type een methode precies teruggeeft. Zo weet je als ontwikkelaar wat je kan verwachten bij de implementatie van classes en hoef je geen conversies meer te doen. Het return type kan een van de datatypes zijn uit PHP zelf of bijvoorbeeld een class die je zelf hebt geschreven.

```php
<?php
class SeniorMedewerker extends Medewerker {

    public function weekloon(): int {
       $weekloon = parent::weekloon();
       return $weekloon + 50;
    }
}
```

In het bovenstaande voorbeeld geven we met `:int` aan, dat er een integer wordt teruggeven. Geeft je methode niets terug, dan is er sprake van een `void` en ook dat kunnen we in de code aangeven.

```php
<?php
class SeniorMedewerker extends Medewerker {

    public function voorstellen(): void {
       echo 'Ik ben Andy';
    }
}
```

En je hebt ook de mogelijkheid om een combinatie zowel een `null` als een specifiek type terug te geven.

```php
<?php
class SeniorMedewerker extends Medewerker {
    private bool $isOntslagen = false;

    public function verantwoordelijkheden(): ?string {
       if ( $this->isOntslagen === true ) {
          return null;
       }

       return 'Geeft sturing';
    }
}
```

Opgave 10

Stap 1

Ga naar de klasse `Persoon` en voeg de volgende methode eraan toe.

```
function getGegevens(){
  $gegevens =
  "\nDe gegevens van " . $this->naam . " zijn: " .
  "\nLeeftijd: " . $this->leeftijd .
  "\nGeslacht: " . $this->geslacht;
  return $gegevens;
}
```

Stap 2

Open **Program.php** en toon het resultaat van de `getGegevens()`-methode van de `thamara`- en de `demirel`-objecten.

Het resultaat van de `getGegevens()`-methode zie je hieronder:

Output

```
De gegevens van Thamara zijn:
Leeftijd: 19
Geslacht: V

De gegevens van Demirel zijn:
Leeftijd: 24
Geslacht: M
```

Opgave 11

Ga naar de klasse `Persoon` en voeg de volgende methode eraan toe.

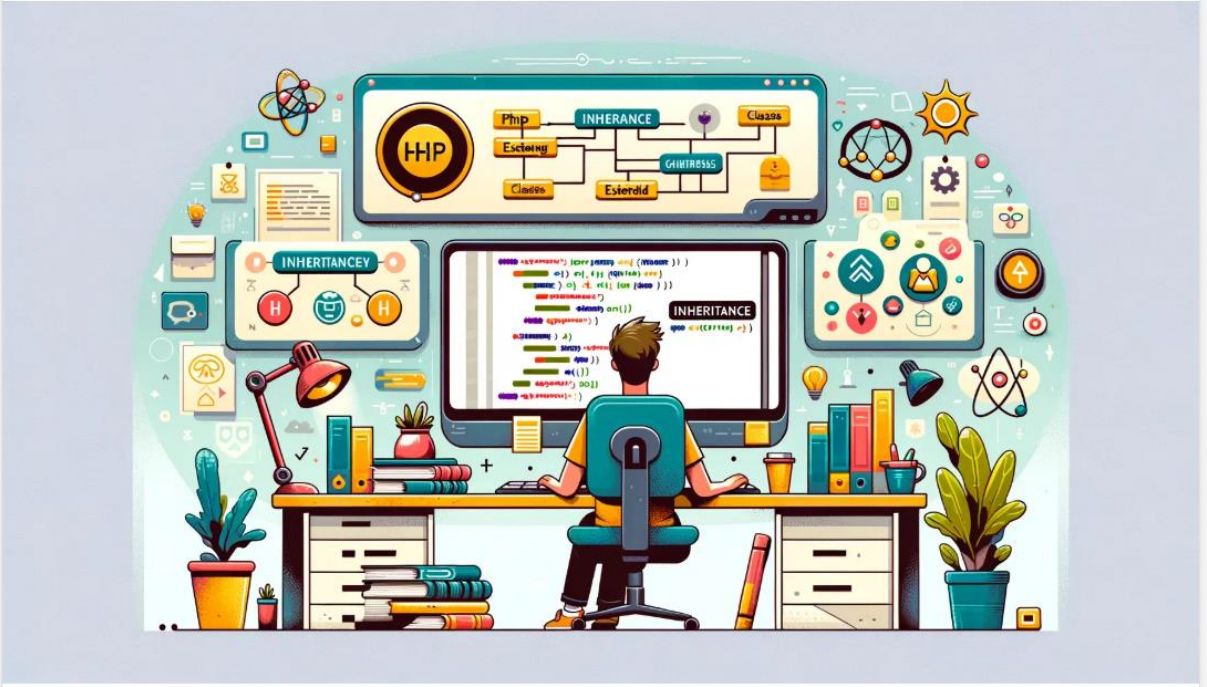
Een gebruikelijke manier om de properties van een klasse te tonen is door de `toString()`-methode als volgt te herschrijven:

```
// override toString() en print klasse gegevens
function toString()
{
  return("\nDe gegevens van " . $this->naam . " zijn:" .
  "\nLeeftijd: " . $this->leeftijd .
  "\nGeslacht: " . $this->geslacht);
}
```

Voeg aan **Program** het volgende toe:

```
echo $demirel->toString();
```

Inheritance in PHP: Hergebruik en uitbreiding van code



Inheritance is een van de fundamentele concepten in objectgeoriënteerd programmeren (OOP). Het stelt een klasse in staat om eigenschappen en methoden over te nemen van een andere klasse. Dit helpt bij het hergebruik van code, het uitbreiden van functionaliteit en het organiseren van je code op een logische manier.

Wat is inheritance in PHP?

Inheritance stelt je in staat om een nieuwe klasse te definiëren die de eigenschappen en methoden van een bestaande klasse overneemt. De bestaande klasse wordt de “superklasse” of “ouderklasse” genoemd, terwijl de nieuwe klasse de “subklasse” of “kindklasse” is. De subklasse kan nieuwe eigenschappen en methoden hebben of bestaande overschrijven.

Laten we een eenvoudig voorbeeld bekijken van hoe inheritance werkt in PHP.

Voorbeeld van inheritance

```php
<?php
class Dier {
    protected $naam;

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

    public function maakGeluid() {
        echo "Dit dier maakt een geluid.";
    }
}

class Hond extends Dier {
    public function maakGeluid() {
        echo "Woef!";
    }

    public function haalStok() {
        echo "$this->naam haalt de stok.";
    }
}

$hond = new Hond("Bobby");
$hond->maakGeluid();  // Output: Woef!
$hond->haalStok();    // Output: Bobby haalt de stok.
?>
```

In dit voorbeeld erft de `Hond` klasse van de `Dier` klasse. Dit betekent dat de `Hond` klasse toegang heeft tot de eigenschappen en methoden van de `Dier` klasse. De `Hond` klasse overschrijft de `maakGeluid` methode en voegt een nieuwe methode `haalStok` toe.

Opgave 12 Maak voorbeeld van inheritance, noem dit bestand `inheritance.php`

Plaats dit in de mak OOPHP

Let op: Dit is een oefening, daarom twee classes in één betand! Normaal nooit doen, anders werk autoloading niet in frameworks 😊

Voor het sparen en betalen van rekeningen kan je bij de bank twee soorten rekeningen openen:

1. Spaarrekening (saving account)
2. Betaalrekening (checking account)

Beide rekeningen hebben een saldo(balance), de hoeveelheid geld dat op de rekening staat. En je kan geld storten (deposit) of geld opnemen (withredraw)

We gaan eerst de algemene BankAccount class maken. (BankAccount.php)

```php
<?php

class BankAccount
{
    private $balance;

    public function getBalance()
    {
        return $this->balance;
    }

    public function deposit($amount)
    {
        if ($amount > 0) {
            $this->balance += $amount;
        }

        return $this;
    }
}
```

We gaan een spaarrekening (saving account) maken. (SavingAccount.php)

```php
<?php

class SavingAccount extends BankAccount
{
}
```

De klasse is nog leeg, maar we kunnen al wel objecten ervan maken. Je erft alles van BankAccount! (wel even BankAccount.php includen)

Maak een bestand `bankrekening.php` aan en kijk of onderstaande code werkt!

```php
<?php

require 'SavingAccount.php';

$account = new SavingAccount();
$account->deposit(100);
echo $account->getBalance();
```

Opgave 13 Maak BankAccount.php en SavingAccount.php en bankrekening.php

Als het goed staat er nu 100 euro op de spaarrekening.

Maak bij een spaarrekening hoort rente, vandaar wat nieuwe functies in SavingAccount

```php
<?php

class SavingAccount extends BankAccount
{
    private $interestRate;

    public function setInterestRate($interestRate)
    {
        $this->interestRate = $interestRate;
    }

    public function addInterest()
    {
        // calculate interest
        $interest = $this->interestRate * $this->getBalance();
        // deposit interest to the balance
        $this->deposit($interest);
    }
}
```

Nu kan er ook rente uitgekeerd worden. Even testen.

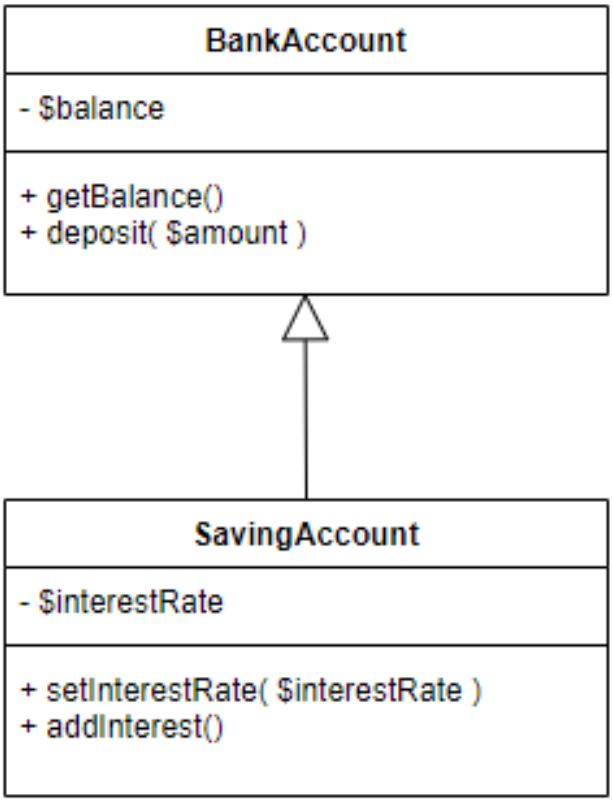
```php
$account = new SavingAccount();
$account->deposit(100);
// set interest rate
$account->setInterestRate(0.05);
$account->addInterest();
echo $account->getBalance();
```

Als het goed is staat je saldo (balance) nu op 105 euro. (`addInterest()` wel 1 keer per jaar aanroepen! 😊)

Opgave 14 Pas aan, BankAccount.php en SavingAccount.php en bankrekening.php

Inheritance en UML

We gebruiken een pijltje om de overerving (inheritance) aan te geven tussen twee classes



How to Call the Parent Constructor

We hebben nu een spaarrekening waar we geld op kunnen storten en rente op kunnen ontvangen. Alleen zou het wel fijn zijn als het begin saldo en het rentepercentage niet met methodes ingesteld moet worden (deposit en setInterest) maar middels een constructor.

```php
<?php

class BankAccount
{
        private $balance;

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

        public function getBalance()
        {
                return $this->balance;
        }

        public function deposit($amount)
        {
                if ($amount > 0) {
                        $this->balance += $amount;
                }

                return $this;
        }
}
```

```php
<?php


class SavingAccount extends BankAccount
{
        private $interestRate;

        public function __construct($balance, $interestRate)
        {
                parent::__construct($balance);

                $this->interestRate = $interestRate;
        }

        public function setInterestRate($interestRate)
        {
                $this->interestRate = $interestRate;
        }

        public function addInterest()
        {
                // calculate interest
                $interest = $this->interestRate * $this->getBalance();
                // deposit interest to the balance
                $this->deposit($interest);
        }
}
```

Nu hebben beide klassen een constructor. Wanneer je een SavingAccount object maakt wordt de constructor van SavingAccount aangeroepen. Daarin moet de constructor van de parent (BankAccount) aangeroepen worden!

En nu even testen.

```php
$account = new SavingAccount(100, 0.05);
$account->addInterest();
echo $account->getBalance();
```

Opgave 15 Pas aan, BankAccount.php en SavingAccount.php en bankrekening.php

Als het goed is heb je nu 105 euro op je bankrekening.

Opgave 16 Maak 5 spaarrekeningen aan voor de volgende personen

- Kwik
- Kwek
- Kwak
- Donald
- Katrien

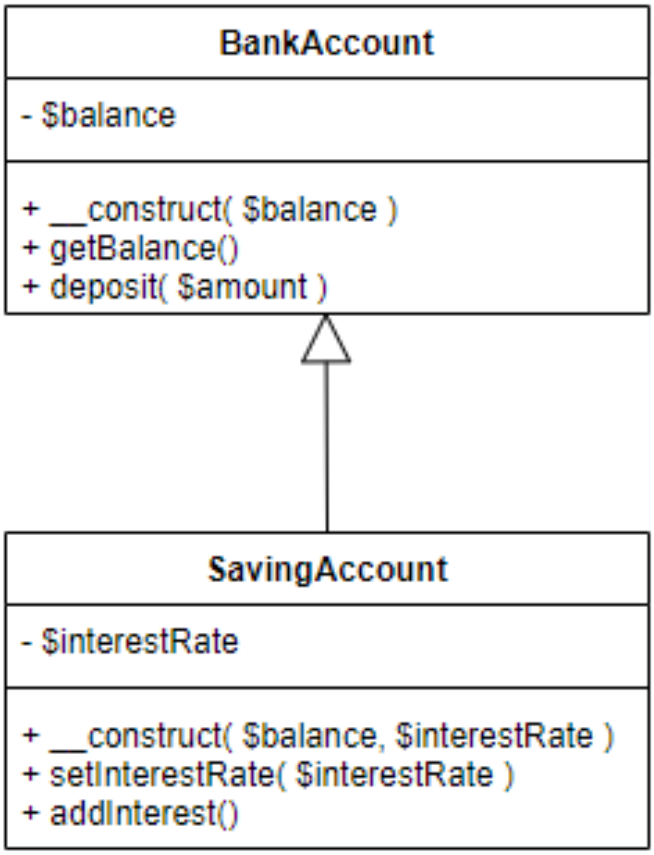
Om de naam te bewaren voeg je een nieuw attribuut(variabele) toe met te naam : name, maak hiervoor een getter aan. Zorg dat de naam geset wordt door de constructor. De constructor moet dus aangepast worden.

Bewaar de SavingAccount objecten in een array. Na een willekeurig aantal keer geld storten en het uitkeren van rente bij alle personen, druk je de naam en het saldo af in een tabel. Ze starten allemaal met 1000 euro saldo en de rente is 4%.

Gebruik hiervoor de foreach lus.

Let op deze code moet je laten controleren!

Class Diagram



Let op: klasse diagrammen verschillen soms een beetje van elkaar. De variabelen namen en de functie namen met de parameters moeten genoemd worden. Datatypes zijn optioneel.

PHP Override Method

Nu gaan we een betaalrekening maken, hiervoor moet de functie `withdraw($amount)` toegevoegd worden aan de klasse BankAccount.

```php
<?php

class BankAccount
{
        private $balance;

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

        public function getBalance()
        {
                return $this->balance;
        }

        public function deposit($amount)
        {
                if ($amount > 0) {
                        $this->balance += $amount;
                }
                return $this;
        }

        public function withdraw($amount)
        {
                if ($amount > 0 && $amount <= $this->balance) {
                        $this->balance -= $amount;
                        return true;
                }
                return false;
        }
}
```

Als het op te nemen geld lager is dan het saldo dan mag het. Saldo wordt verlaagd en er is een return waarde, `true` of `false`.

Nu de nieuwe Class CheckingAccount

```php
<?php
class CheckingAccount extends BankAccount
{
    private $minBalance;

    public function __construct($amount, $minBalance)
    {
        if ($amount > 0 && $amount >= $minBalance) {
            parent::__construct($amount);
            $this->minBalance = $minBalance;
        } else {
            throw new InvalidArgumentException( message: 'amount
            must be more than zero and higher than the minimum balance');
        }
    }

    public function withdraw($amount)
    {
        $canWithdraw = $amount > 0 &&
        $this->getBalance() - $amount > $this->minBalance;

        if ($canWithdraw) {
            parent::withdraw($amount);
            return true;
        }

        return false;
    }
}
```

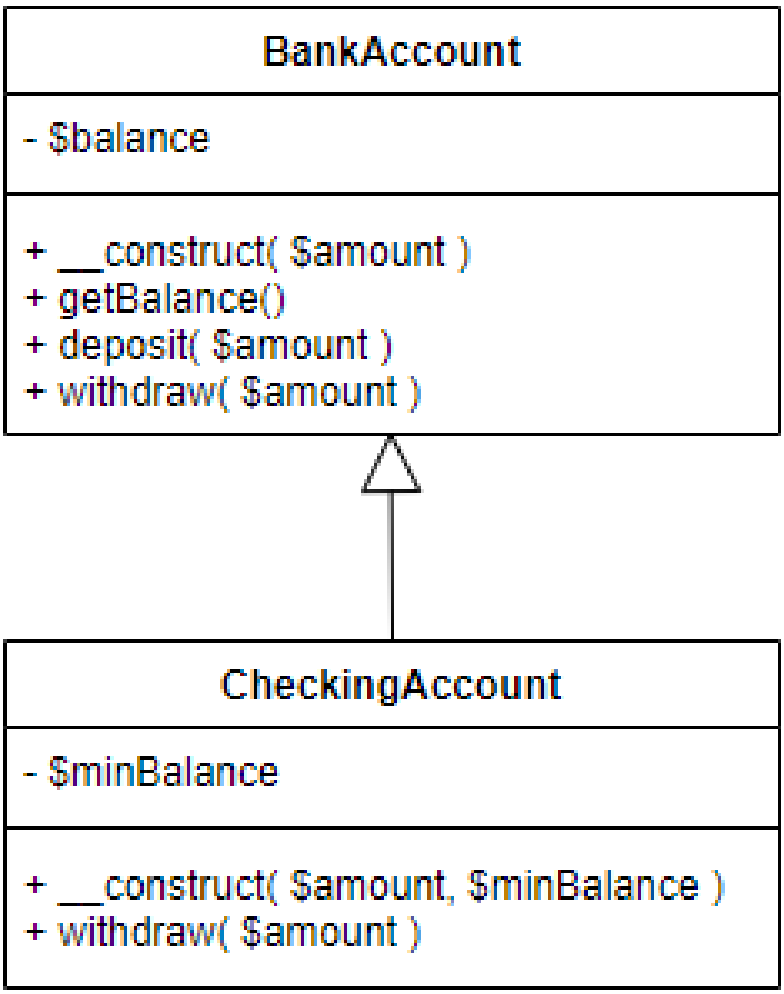
De functie `withdraw($amount)` is opnieuw gemaakt met een strengere controle. Je kan alleen het gewenste bedrag opnemen als er nog een minimum saldo overblijft.

Opgave 17 Maak CheckingAccount.php en pas bankrekening.php aan.

Maak een CheckingAccount object en probeer in geld op te nemen. Zorg dat het een keer lukt, balance =1000, min_balance=200, amount=500

En dat het een keer niet lukt.

Class diagram



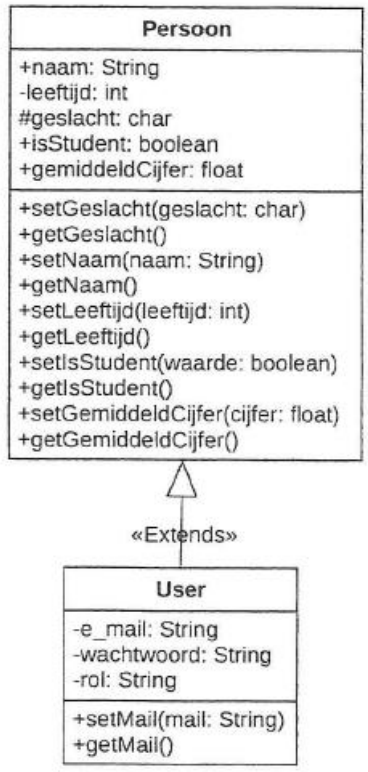
Opgave 18 Maak onderstaan de twee Classes aan en maak een user object.

Ga ervan uit dat Persoon voor de constructor de volgende variabelen nodig heeft

- Naam
- Leeftijd
- Geslacht

De User heeft voor zijn constructor de volgende variabelen nodig

- E_mail
- Wachtwoord
- Rol



Figuur 4.9 Klassendiagram met subklasse

Meer informatie?

Inheritace -> Inheritance in PHP: Hergebruik en uitbreiding van code - PHP Tutorial.nl

Interfaces -> Interfaces: ontdek de kracht van het publieke contract - PHP Tutorial.nl

Namespaces -> Namespaces in PHP: het hulpmiddel bij organiseren van code - PHP Tutorial.nl

Abstracte classes -> Abstracte classes: Een diepgaande uitleg over dit concept - PHP Tutorial.nl

Static in PHP Classes -> Static in PHP Classes: maak methodes en properties - PHP Tutorial.nl