In a room full of top software designers, if two agree on the same thing, that's a majority.

Bill Curtis

8 Object Oriented PHP

8.1 Introduction

PHP version 5 has a full object model, which means you can use all the principles and design patterns from object-oriented programming.

This chapter does not focus on design patterns, but gives a summary of the syntaxis for the most used elements form object-oriented php.

A complete overview of all object-oriented implementations in PHP can be found at `http://www.php.net/manual/en/language.oop5.php`

8.2 classes, properties and methods

A class in PHP is created with the `class` command, followed by the name of the class and curly brackets.

A property (or also called an instance variable) is defined at the beginning of the class, it can be public, private or protected. Public means that you can access the property within the class and also in other php files. Private means that you can only access the property within the class itself. And finally, protected means that you can only use the property within the class itself and in any of its subclasses.

A method is defined as a function within this class, it can also be public, private or protected. Examples of some methods can be found in the class Vehicle. A get method for the brand property is implemented, a set method for the brand property is implemented, and also some examples of other methods are implemented.

From each class, you can instantiate one or more objects. On each object you can call the methods defined within this class.

Listing 8.1 class object

```
<?php
class Vehicle
{
    public $color;
    private $brand;
    protected $kilometers = 0; // with a default value

    public function getBrand()
    {
        return $this->brand;
    }

    public function setBrand($brand)
    {
        $this->brand = $brand;
    }

    public function addKilometers($kilometres)
    {
        $this->kilometers = $this->kilometers + $kilometers;
    }

    public function printKilometers()
    {
        echo 'The vehicle has moved ' . $this->kilometers . ' kilometers.';
    }
}

// instantiation with the default constructor
$myVehicle = new Vehicle();

$myVehicle->addKilometers(5);
$myVehicle->printKilometers();
$myVehicle->addKilometers(75);
$myVehicle->printKilometers();

?>
```

8.3 Inheritance

Classes can inherit other classes with the keyword `extends`. This means that all properties and methods of the superclass are inherited by the subclass. In the subclass additional properties and methods can be defined.

Listing 8.2 inheritance of the vehicle

```
<?php

// ... the class Vehicle is defined as above

class Car extends Vehicle
{

    private $licenceplate = "1-ABC-123";

    public function printLicenceplate()
    {
        echo 'The car\'s licence plate is ' . $this->licenceplate;
    }

}

$myVehicle = new Car();

$myVehicle->addKilometers(15);
$myVehicle->printKilometers();
$myVehicle->printLicenceplate();
?>
```

8.4 Constructors, magic setters and getters and other magic methods

Magic methods are prefixed with a double underscore. There are a lot of these, but `__construct` and `__toString` are most commonly used.

8.4.1 constructors

A constructor is the method which is automatically executed whenever an instance of the class object is made.

Listing 8.3 constructor in superclass

```
<?php
class Vehicle
{
    // ... the rest is defined as above in the class Vehicle

    // constructor
    public function __construct($brand)
    {
        $this->brand = $brand;
    }
}

$vehicle = new Vehicle('XXX');

echo "The brand of the vehicle is: " . $vehicle->getBrand();

$vehicle->color = "black";
echo "The color of the vehicle is: " . $vehicle->color;

?>
```

In a subclass you can call the constructor of the superclass as given in the example below. In this constructor, the constructor of the super class is called and thus the brand is set, additionally the kilometers and the license plate is set in this constructor of the sub class.

Listing 8.4 constructor in subclass

```
<?php

// ... the class Vehicle is defined as above

class Car extends Vehicle
{
    // ... the rest is defined as above in the class Car

    function __construct($brand, $kms, $plate)
    {
        parent::__construct($brand);
        $this->kilometers = $kms;
        $this->licenceplate = $plate;
    }
}

$saab = new Car("Saab", 10000, "1-DMQ-901");

echo "The brand of the vehicle is: " . $saab->getBrand();
$saab->color = "black";
echo "The color of the vehicle is: " . $saab->color;

?>
```

8.4.2 method overriding

A method defined and implemented in a super class, can be overridden in a sub class. This means that a new implementation is given for this method on the level of the sub class. An example of such a method is given in the listings below (the method getNumberOfTies()).

Listing 8.5 method getNumberOfTies() in superclass

```
<?php
class Vehicle
{
    // ... the rest is defined as above in the class Vehicle

    public function getNumberOfTies()
    {
        return 0;
    }
}

$vehicle = new Vehicle("XXX");
echo "The number of ties of the vehicle are: " . $vehicle->getNumberOfTies();

?>
```

Listing 8.6 method getNumberOfTies() in subclass

```
<?php

// ... the class Vehicle is defined as above

class Car extends Vehicle
{
    // ... the rest is defined as above in the class Car

    public function getNumberOfTies()
    {
        return 4;
    }
}

$saab = new Car("Saab", 10000, "1-DMQ-901");

echo "The brand of the vehicle is: " . $saab->getBrand();
$saab->color = "black";
echo "The color of the vehicle is: " . $saab->color;
echo "The number of ties of the vehicle are: " . $saab->getNumberOfTies();

?>
```

It would even be better that the class Vehicle is an abstract class and that the method `getNumberOfTies()` is an abstract method on the level of the super class. This is possible in PHP , see `http://www.php.net/manual/en/language.oop5.abstract.php`.

8.4.3 toString

In order to print the properties of an object, you can define a magic method called the `__toString` method.

Listing 8.7 toString method

```
<?php

// ... the class Vehicle is defined as above

class Car extends Vehicle
{
    // ... the rest is defined as above in the class Car

    public function __toString()
    {
        return "The brand is: " . $this->brand . " - The kilometers of this car are:
            " . $this->kilometers . " - The color of this car is: " . $this->color .
            " - The licence plate of this car is: " . $this->licenceplate;
    }
}

$saab = new Car("Saab", 10000, "DMQ-901");

echo $saab;
?>
```

8.4.4 magic setters and magic getters

When you want to use properties (or instance variables) in a class, you have to define these at the beginning of the class. But if you want your class object to be more flexible, you can 'pretend' to have them by using the magic setter `__set` and magic getter `__get`. These two magic methods will provide a fallback mechanism whenever a variable is called which doesnt exists in the class model.

These can be helpful for logging and debugging as in the example below:

Listing 8.8 magic getters and setters for logging and debugging

```
<?php

class Person
{
    public function __set($name, $value)
    {
        echo 'Could not assign ' . $value . ' to ' . $name . '.';
    }

    public function __get($name)
    {
        echo 'Variable ' . $name . ' is not declared';
    }
}

$myPerson = new Person();

// these actions will produce some echo-statements
$myPerson->name = 'jan';
echo $myPerson->name;

?>
```

But you can also simply implement dynamic creation of variables for your object by using the magic methods to assign them to a container variable in your class.

Listing 8.9 magic getters and setters to mimic the existance of variables

```php
<?php

class Building
{
    protected $container = array();

    public function __set($name, $value)
    {
        $this->container[$name] = $value;
    }

    public function __get($name)
    {
        $return = false;

        if (array_key_exists($name, $this->container)) {
            $return = $this->container[$name];
        }

        return $return;
    }
}

$myBuilding = new Building();

// these actions will be executed as if $name was a variable declared inside the class Building
$myBuilding->name = 'G&T';
echo $myBuilding->name;

?>
```

More info can be found at `http://www.php.net/manual/en/language.oop5.magic.php`

8.5 static

Declaring class properties or methods as static makes them accessible without needing an instantiation of the class. For more information see `http://www.php.net/manual/en/language.oop5.static.php`.

In the following subsections examples of a static method and a static property are given.

8.5.1 static methods

In the example, the method sum is static because it is not necessary to make each time an object of the class Calculator in order to calculate the sum of two given numbers.

Listing 8.10 static method

```php
<?php

class Calculator
{
    public static function sum($number1, $number2)
    {
        return $number1+$number2;
    }
}

echo Calculator::sum(2, 4);

$className = "Calculator";
echo $className::sum(6, 4);
echo $className::sum(6, 6);
echo $className::sum(6, 8);

?>
```

8.5.2 static properties

In the example a static property is defined. Each time an object of this class is made, the static property is added by 1. At each moment in time, you can print the number of persons made without needing an instance of this class. And each time you create a new instance, the number of persons is added by 1 in the constructor of this class.

Think of this as if the variable `$numberOfPersons` was a global variable.

Remark that you don't use `$this` to refer to the current class because you don't make an instance of it when you use it.

Listing 8.11 static property

```php
<?php

class Person
{
    private $name;
    public static $numberOfPersons;

    public function __construct($name)
    {
        $this->name = $name;
        Person::$numberOfPersons = Person::$numberOfPersons + 1;
    }
}

$elke = new Person("Elke");
echo Person::$numberOfPersons;

$kurt = new Person("Kurt");
echo Person::$numberOfPersons;

?>
```