# Lab: OOP Encapsulation and Inheritance

This document defines the **exercise assignments** for the "[PHP Fundamentals" Course @ Software University](https://softuni.bg/courses/php-basics/) .

## Part I. Encapsulation

## Define a class Vehicle

Define a class **Vehicle** that has the following **private** properties: **numberDoors, color**. Define **a constructor** for the class that takes all properties as parameters and sets them. Create a new instance of Vehicle called **$myVehicle**.

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myVehicle with:  2 doors  orangle | Vehicle Object ( [color:Vehicle:private] => orange [numberDoors:Vehicle:private] => [numDooors] => orange ) |

## Vehicle Getters and Setters

1. Define two public setters called **setDoors** and **setColor** which set the properties **numberDoors** and **color**.
2. Define two public getters called **GetDoors** and **getColor** which will return the two properties of the object.
3. **Rewrite the constructor** to use **SetDoors** and **setColor**.

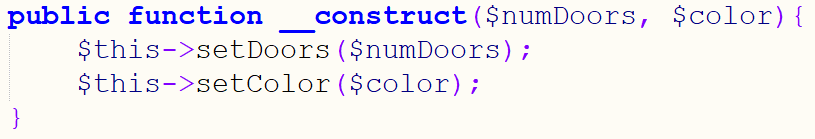
Write a magic getter that will **return the value** if it exists or return a string “property doesn’t exist”. Use setDoors to **set the number of doors to 4** and **get the number of doors** by the magic getter.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Set doors in $myVehicle to 4 | Get number of doors:  4 |

## Hint

For the magic getter use: **public function \_\_get($name)** in which you can put an if statement.

Your constructor should look like:



## Add a paint method to Vehicle

Imagine that somehow you go to the painter and you want to change the color of your vehicle. That would be equivalent to creating a method **paint** that will use the setter **setColor**. Write this method and call it on $myVehicle as you change the color to **blue**.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Paint $myVehicle in blue | Vehicle Object ( [color:Vehicle:private] => orange [numberDoors:Vehicle:private] => [numDooors] => blue ) |

## Part II. Inheritance

## Create Class Car that Extends Vehicle

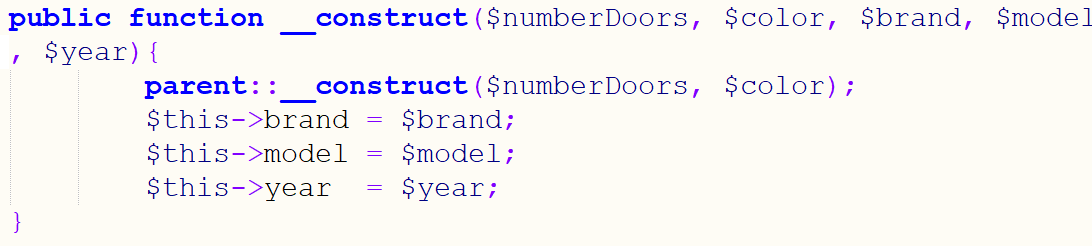
Use the Vehicle class by extending it to a new class Car. Add new properties to it: **brand**, **model**, **year**. Write a **constructor of Car** that will **call the constructor of Vehicle** and receive following parameters: **$numberDoors, $color, $brand, $model, $year**.

Seaparate the classes in two files: **Vehicle.php** and **Car.php**. Include Vehicle.php in Car.php. Create an instance of Car with the following parameters: Red Audi A4 2016 with 4 doors.

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCar  Red Audi A4 2016 with 4 doors | Car Object ( [brand:Car:private] => Audi [model:Car:private] => A4 [color:Car:private] => [numberDoors:Vehicle:private] => [color:Vehicle:private] => Red [numDoors] => 4 [year] => 2016 ) |

## Hint

Your constructor should look like:



### 

## Visibility of color

Imagine now that you want **to change the color of the Audi A4**. You can add a new method paint to Car:

|  |
| --- |
| Car.php  Class Car{  // …  public function **paint**($paint\_color){  $this->color = $paint\_color;  }  } |

Would that change the color of vehicle? The answer is **NO**. Lets dump $myCar:

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCar->paint('Green');  print\_r($myCar); | **Car** Object ( [brand:Car:private] => Audi [model:Car:private] => A4 [year:Car:private] => 2016 **[color:Vehicle:private] => Red** [numberDoors:Vehicle:private] => **4 [color] => Green** ) |

It has created a new property in Car called also **color** and assigned the new color to it. This is because color is defined as **private** in Vehicle and PHP will **overwrite** it in Car!

**How would you change the color that comes from Vehicle instead of creating a new property like:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCar->paint('Green');  print\_r($myCar); | **Car** Object ( [brand:Car:private] => Audi [model:Car:private] => A4 [year:Car:private] => 2016 **[color:Vehicle:private] => Green** [numberDoors:Vehicle:private] => **4** ~~[color] => Green~~ ) |

## Hint

We can use the parent setter to do the job: **parent::setColor($paint\_color);**

## Access to Color in Car but Not Outside

Since **color** is something significant to a car **why should it be a private member** of Vehicle? We want Car to be able to have **full access to color** but the color to be **NOT accessible outside car**. That is where protected comes in. **Rewrite Vehicle.php** so that **Car’s paint will use color** directly.

## Hint

The **visibility of color** in **Vehicle** must change. **Car’s** **paint** method should also change as it **doesn’t need the setter** in Vehicle anymore.

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCar->paint('Green');  print\_r($myCar); | Car Object ( [brand:Car:private] => Audi [model:Car:private] => A4 [year:Car:private] => 2016 [color:protected] => Green [numberDoors:Vehicle:private] => 4 ) |
| $myCar->color = 'Black';  print\_r($myCar); | **Setting color directly should give an error**  **Fatal error**: Uncaught Error: **Cannot access** protected property Car::$color |

## Protecting the Number of Doors Setter

Until now we had Vehicle **setDoors()** to be **public**. And we can access it in Car but also this way: **$myCar->setDoors();**

But somehow we **don’t want** to be able to **set the number of the doors outside of Car** because usually cars don’t change their number of doors. We want to **encapsulate** this functionality and have it in Vehicle, in Car but not outside of a parent or child class.

## Hint

The **visibility of setDoors** in **Vehicle** must change.

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCar->setDoors(15); | **Fatal error**: Uncaught Error: Call to protected method Vehicle::setDoors()) |

## Create Class Bicycle that Extends Vehicle

We can have more types of vehicles in the real world. The bicycle is another type of vehicle. An important fact for all bicycles is that **they do not have doors.** So the constructor of the Bicycle class should set the **numberDoors property** to **0**! Bicycles can have also a **brand**, **model** and **year** properties. Another property would be **forSkirt** = null |true | false, which shows whether it is suitable to be used when you wear a skirt (cars don’t have such a property).

**Write a method \_\_toString()** that would print all data about the bicycle in an HTML table.

**Add a property** that shows **whether someone rides the bike (true | false)**.

**Write a method** that sets the bicycle to ride mode. You may call the method **Ride**().

**Write a method** that stops the bike. You may call it **Stop()**.

**Write a new setDoors** which will **overwrite** the one in the parent class. It is without arguments because bikes don’t have doors. Change your constructor to use the new overwritten setDoors.

**Create two instances of the Bicycle class $b1 and $b2**, change the mode of b1 to ride and b2 to stop.

## Rethink Vehicle Again\*

The Car and Bicycle classes contain a **brand, model and year** properties. But this is something that all vehicles usually have. Now go back to your Vehicle class and **move the properties** there. What visibility should the properties have to be visible to Car and Bicycle, and all classes that would inherit them (children, grandchildren and etc.)?

How would **the Car and Bicycle** classes change? Change the setters, getters and constructors of Car and Bicycle.

**Important: A trap in OOP is to extend classes without the need to do so. So class hierarchies may become huge and horrific to maintain. A lazy approach is to extend classes without measure without optimization.**

## 