Object Oriented Programming

—

Fundamentals

Constructors

PHP allows you to declare a constructor method for a class with the name **__construct()** as follows:

```php
<?php
class Vehicle {
  // Properties
  public $speed = 0;
  public $wheel;

  // Methods
  function __construct($wheel) {
    $this->wheel += $wheel;
  }
  function speedInc($val) {
    $this->speed += $val;
  }
  function speedDec($val) {
    return $this->val -= speed;
  }
}
?>
```

```php
<?php
  $car = new Vehicle(4);
  $bus = new Vehicle(4);
  $bike = new Vehicle(2);
?>
```

When you create an instance of the class, PHP automatically calls the constructor method.

Typically, you use the constructor to initialize the properties of the object.

Ref: PHP constructor
Ref: PHP destructor

Access modifier

PHP has three access modifiers:

- Public
- Private and
- Protected.

We'll focus on the public and private access modifiers.

Public : The **public** access modifier allows you to access properties and methods from both inside and outside of the class.

Private : The **private** access modifier prevents you from accessing properties and methods from the outside of the class.

Access Modifier (continue)

```php
<?php
class Vehicle {
  // Properties
  public $speed = 0;
  private $key = "***";
}

$car = new Vehicle()
echo $car->speed      // 0
Echo $car->key        // Fatal Error
?>
```

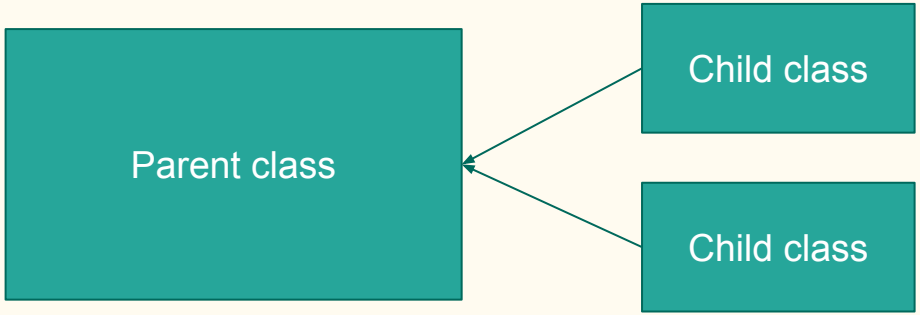
When you place the **public** keyword in front of a property or a method, the property or method becomes public. It means that you can access the property and method from both inside and outside of the class.

To prevent access to properties and methods from outside of the class, you use the **private** access modifier. The following example **$key** property of the Vehicle class is private

If you attempt to access the **$key** property from the outside of the class, you'll get an error.

Inheritances

- Inheritance allows a class to reuse the code from another class without duplicating it.
- In inheritance, you have a **parent class** with properties and methods, and a **child class** can use the code from the parent class.
- Inheritance allows you to write the code in the parent class and use it in both parent and child classes.
- The parent class is also called a **base class** or **super class**. And the child class is also known as a **derived class** or a **subclass**.
- To define a class inherits from another class, you use the **extends** keyword.



Inheritance (continued)

Suppose that you have a class called BankAccount as follows: The **BankAccount** class has the private property **$balance** and the public methods **getBalance()** and **deposit()**. To declare that the **SavingAccount** inherits from the **BankAccount**, you use the **extends** keyword as follows:

```php
<?php
class BankAccount {
    private $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 $interestRate;
    public function setInterestRate($interestRate) {
        $this->interestRate = $interestRate;
    }
}
?>
```

Inheritance (continued)

```php
require 'SavingAccount.php';

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

In this example, the **SavingAccount** can reuse all the non-private properties and methods from the **BankAccount** class. The following creates a new instance of the **SavingAccount** class, calls the **deposit()** method and shows the balance:

In this example, the **SavingAccount** class reuses the properties and methods from the **BankAccount** class.

Static methods

Static methods can be called directly - without creating an instance of the class first.

Static methods are declared with the static keyword:

```php
<?php
	class ClassName {
	 public static function staticMethod() {
	   echo "Hello World!";
	 }
	}
?>
```

```php
ClassName::staticMethod();
```

Static methods (example)

```php
<?php
class greeting {
 public static function welcome() {
   echo "Hello World!";
 }
}

// Call static method
greeting::welcome();
?>
```

Here, we declare a static method: **welcome()**. Then, we call the static method by using the class name, **double colon (::)**, and the method name (without creating an instance of the class first).

Static methods (more)

A class can have both static and non-static methods. A static method can be accessed from a method in the same class using the self keyword and double colon (::):

```php
<?php
class greeting {
 public static function welcome() {
   echo "Hello World!";
 }

 public function __construct() {
   self::welcome();
 }
}

new greeting();
?>
```

Static methods (more)

To call a static method from a child class, use the **parent** keyword inside the child class. Here, the static method can be **public** or **protected**.

```php
<?php
class domain {
 protected static function getWebsiteName() {
   return "W3Schools.com";
 }
}

class domainW3 extends domain {
 public $websiteName;
 public function __construct() {
   $this->websiteName = parent::getWebsiteName();
 }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

Static properties

Property and use case are similar to static methods.

Ref : Static Properties