DEFINING A CLASS

A CLASS IS DEFINED BY USING THE KEYWORD `class` FOLLOWED BY ITS NAME

```
class className [extends parentClass] [implements interface]
{
...properties  // private, public or protected

...methods     // private, public or protected

}
```

BY USING THE OPTIONAL KEYWORD `extends` WE DECLARE THE CLASS `className` AS A CHILD CLASS OF `parentClass`

BY USING THE OPTIONAL KEYWORD `implements` WE DECLARE THAT THE CLASS `className` MUST IMPLEMENT THE SPECIFIED INTERFACE

PROPERTIES

PROPERTIES ARE DECLARED INSIDE THE CURLY BRACKETS

```
class Person
{
  private $name;
  private $address;
  ...
}
```

METHODS

METHODS ARE DECLARED INSIDE THE BRACKETS BY USING THE KEYWORD function

PROPERTIES ARE REFERENCED INSIDE A METHOD BY USING $this->property

```
class Person
{
  private $name;
  private $address;

  function setName($name) {
   $this->name=$name;
  }
  function getName() {
   return $this->name;
  }
  ...
}
```

CREATING OBJECTS

AN OBJECT IS CREATED USING THE KEYWORD `NEW` FOLLOWED BY THE NAME OF THE CLASS AND A PAIR OF PARENTHESIS (WITH OR WITHOUT PARAMETERS)

```
$objectName=new className();
```

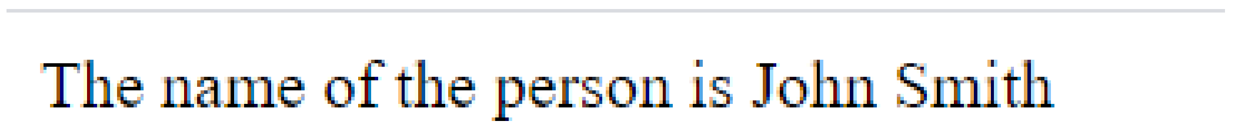
WE HAVE INSTANTIATED THE CLASS IN ORDER TO CREATE THE OBJECT. THEREFORE, `objectName` IS AN INSTANCE OF THE CLASS `className`

PROPERTIES AND METHODS ARE REFERENCED WITH `$objectName->`

```
$name=$objectName->name;
$objectName->methodName();
```

WORKING WITH OBJECTS PROPERTIES AND METHODS

```php
$onePerson=new Person();
$onePerson->setName("John Smith");
echo "The name of the person is ".$onePerson->getName();
```



```
The name of the person is John Smith
```

ACCESS MODIFIERS

ACCESS MODIFIERS CONTROL WHERE PROPERTIES AND METHODS CAN BE ACCESSED

THE ACCESS MODIFIERS ARE:

• `public`	universal access (both from inside and outside the class)
• `private`	only can be accessed within the class
• `protected`	can be accessed within the class and derived classes

BY DEFAULT, EVERYTHING IS `public`

ACCESS MODIFIERS

```
class Person
{
  private $name;
  public $address;
  protected $birthDate;

   ...
}

$onePerson=new Person();
$onePerson->name="John Smith";   // ERROR
$onePerson->address="Trafalgar Square"; // IT WORKS
```

INSTANCEOF KEYWORD

WE CAN CHECK IF AN OBJECT BELONGS TO A SPECIFIC CLASS BY USING THE KEYWORD `INSTANCEOF`

```
$onePerson=new Person();
$onePerson->setName("John Smith");
echo "The name ".$onePerson->getName();
if($onePerson instanceof Person) {
   echo " belongs to a person";
}
else {
   echo " doesn't belong to a person";
}
```

CONSTRUCTOR METHOD

IN PHP WE DECLARE CONSTRUCTOR METHODS BY USING THE FUNCTION NAME `__construct` WITH OR WITHOUT PARAMETERS

```
class Person
{
 private $name;
 public function __construct($name) {
     $this->name=$name;
 }
 …
}

$onePerson=new Person("John");
// the constructor is automatically invoked when the object
// is created. It's useful to initialize properties
```

CONSTRUCTOR METHOD

UNLIKE OTHER LANGUAGES LIKE JAVA, PHP DOES NOT SUPPORT MULTIPLE CONSTRUCTORS HAVING DIFFERENT NUMBERS OF PARAMETERS. TRY TO RUN THIS AND SEE WHAT HAPPENS...

```
class Person
{
 private $name;
 private $surname;
 public function __construct($name) {
    $this->name=$name;
    $this->surname="Not defined yet";
 }
 public function __construct($name, $surname) {
    $this->name=$name;
    $this->surname=$surname;
 }
}
```

```
Fatal error: Cannot redeclare Person::__construct() in C:\xampp\htdocs\a.php on line 10
```

INHERITANCE

INHERITANCE ALLOWS A CLASS TO ACQUIRE THE MEMBERS OF ANOTHER CLASS BY USING THE `extends` KEYWORD. IN THE EXAMPLE, THE `Square` CLASS (CHILD) INHERITS FROM `Rectangle` (PARENT). IN ADDITION TO ITS OWN MEMBERS, `Square` GAINS ALL ACCESSIBLE (NON-PRIVATE) MEMBERS IN `Rectangle`, INCLUDING ANY CONSTRUCTOR.

```
// Parent class (base class)
class Rectangle {
    public $x, $y;
    function __construct($a, $b) {
        $this->x = $a;
        $this->y = $b;
    }
}
// Child class (derived class)
class Square extends Rectangle {}
```

INHERITANCE

When creating an instance of `Square`, two arguments must now be specified because `Square` has inherited `Rectangle`'s constructor.

```
$s = new Square(5,10);
```

The properties inherited from `Rectangle` can also be accessed from the `Square` object.

```
$s->x = 5; $s->y = 10;
```

A class in PHP may only inherit from one parent class and the parent must be defined before the child class in the script file.

CONSTRUCTOR METHOD AND INHERITANCE

A CHILD CLASS INHERITES THE CONSTRUCTOR METHOD FROM ITS PARENT CLASS. THE CONSTRUCTOR IS AUTOMATICALLY INVOKED WHEN AN OBJECT (BOTH OF PARENT OR CHILD CLASSES) IS CREATED.

THEREFORE, WHEN THE CHILD CLASS HAS ITS OWN CONSTRUCTOR THE PARENT CONSTRUCTOR IS NOT AUTOMATICALLY INVOKED. SO, IF WE NEED TO DO THIS, WE MUST DO IT EXPLICITLY WITH `parent::__construct`.

```
class Teacher extends Person
{
  private $speciality;
  function __construct($name,$surname,$speciality) {
    parent::__construct($name,$surname);
    $this->speciality=$speciality;
  }
  …
}
```

DESTRUCTOR METHOD

A DESTRUCTOR IS AUTOMATICALLY CALLED WHEN THE LAST INSTANCE OF AN OBJECT IS DESTRUCTED OR WHEN THE SCRIPT IS STOPPED OR EXITED

WE CAN CREATE A METHOD `__destruct()` IN ORDER TO FORCE THE DESTRUCTOR TO DO SOMETHING ELSE BESIDES JUST "GARBAGE COLLECTING"

```
class Person
{
  function __destruct()
  {
    …// the code goes here
  }
}
```

WE CAN MANUALLY REMOVE ALL THE REFERENCES TO AN OBJECT BY USING THE `unset` FUNCTION

```
unset($object)
```

FINAL KEYWORD

BY USING THE KEYWORD `final` WE PREVENT A CLASS FROM BEING EXTENDED, OR A METHOD FROM BEING OVERRIDEN.

```
final class MyFinalClass {
   ...
}

class MyExtendedClass extends MyFinalClass{};  // ERROR

 class MyNormalClass {
    final function dontOverrideMe() {
       … // the class can be extended, but this method
         // can't be overriden
    }
 }
```

CONST KEYWORD

BY USING THE KEYWORD `const` WE CAN DECLARE A CONSTANT WITHIN A CLASS (WITHOUT THE $).

```
class Product {
   const IVA = 21;
}
```

CLASS CONSTANTS ARE CASE-SENSITIVE. HOWEVER, IS HIGHLY RECOMMENDED TO NAME THE CONSTANTS USING UPPERCASE LETTERS.

CLASS CONSTANTS ARE ACCESSED FROM OUTSIDE THE CLASS BY USING THE CLASS NAME FOLLOWED BY THE OPERATOR :: AND THE CONSTANT NAME

```
echo Product::IVA;
```

```
And from a method of this class use: self::IVA
```

OBJECTS COMPARISON

THE EQUAL TO OPERATOR (==) RETURNS TRUE IF THE TWO COMPARED OBJECTS ARE INSTANCES OF THE SAME CLASS AND THEIR PROPERTIES HAVE THE SAME VALUES AND TYPES.

THE IDENTITY OPERATOR (===) RETURNS TRUE IF THE TWO COMPARED OBJECTS REFERENCE THE SAME INSTANCE OF THE CLASS

STATIC PROPERTIES AND METHODS

STATIC PROPERTIES AND METHODS CAN BE CALLED WITHOUT CREATING AN OBJECT. THEY ARE DEFINED BY USING THE KEYWORD `static`

STATIC PROPERTIES ARE USEFUL WHEN EVERY INSTANCE OF THE CLASS SHARES THE SAME VALUE FOR THE PROPERTY. UNLIKE CONSTANTS, THE VALUES OF STATIC PROPERTIES CAN BE MODIFIED: THE NEW VALUE WILL SPREAD TO ALL INSTANCES.

OUTSIDE THE CLASS THEY ARE ACCESSED BY USING THE NAME OF THE CLASS FOLLOWED BY THE OPERATOR `::` AND THE NAME OF THE PROPERTY / METHOD

```
class WorkTable {
// public by default
   static function jumpALine() {
       echo "<br/>";
    }
}
WorkTable::jumpALine();
```

STATIC PROPERTIES AND METHODS

STATIC PROPERTIES CAN BE ACCESSED WITHIN THE CLASS BY USING THE `self` KEYWORD FOLLOWED BY THE OPERATOR `::` AND THE NAME OF THE PROPERTY

```
class MathThings {
   public static $valueOfPi = 3.14159;
   function getPI() {
      return self::$valueOfPi;
   }
}
echo "<br/>The value of PI is ".MathThings::$valueOfPi;
$OneMathThing = new MathThings();
echo "<br/>The value of PI is ".$OneMathThing->getPI();
```

```
You can also modify the static property:
MathThings::$valueOfPi=3.14;
```

STATIC PROPERTIES AND METHODS

IF YOU NEED TO ACCESS A STATIC PROPERTY FROM A CHILD CLASS, USE THE KEYWORD `parent` INSTEAD OF `self`

```
class MathThings {
   public static $valueOfPi = 3.14159;
   function getPI() {
      return self::$valueOfPi;
   }
}

class ChildMathThings extends MathThings {
   function getPI() {
      return parent::$valueOfPi;
   }
}
```

TRAITS

LIKE JAVA, PHP ONLY SUPPORTS SINGLE INHERITANCE

BY USING TRAITS, WE CAN IMPLEMENT “MULTIPLE” INHERITANCE: MULTIPLE CLASSES CAN USE METHODS FROM MULTIPLE TRAITS

A TRAIT

```
trait EnglishMessages
{
    function WelcomeEnglish()
    {
        echo "Welcome!";
    }
  …
}
```

USING A TRAIT IN A CLASS

```
class Messages
{
    use EnglishMessages;
    …
}

$message=new Messages();
echo $message->WelcomeEnglish();
```

TRAITS

SINCE WE CAN USE VARIOUS TRAITS IN A CLASS, SOMEHOW WE ARE IMPLEMENTING “MULTIPLE” INHERITANCE

```
trait EnglishMessages {
   function WelcomeEnglish() {
      echo "<br/>Welcome!";
   }
}
trait SpanishMessages {
   function WelcomeSpanish() {
      echo "<br/>Bienvenida!";
   }
}
```

```
class Messages {
   use EnglishMessages, SpanishMessages;
}
$message = new Messages();
echo $message->WelcomeEnglish();
echo $message->WelcomeSpanish();
```

INTERFACES

WE CAN DEFINE INTERFACES BY USING THE KEYWORD `interface`

A CLASS CAN IMPLEMENT ONE OR MORE INTERFACES BY USING THE KEYWORD `implements` AND THEN THE NAME OF THE INTERFACE

WHEN A CLASS IMPLEMENTS AN INTERFACE, DEVELOPING ALL ITS METHODS BECOMES MANDATORY

```
interface WorkTable
{
   function jumpALine();

}
```

```
class MyTable implements WorkTable
{
   function jumpALine() {
       // function code goes here
    }
}
```

ABSTRACT CLASSES

WE CAN DEFINE ABSTRACT CLASSES WITH THE KEYWORD `abstract`

AN ABSTRACT CLASS MUST INCLUDE AT LEAST ONE ABSTRACT METHOD. WHEN AN ABSTRACT CLASS IS INHERITED, THE PARENT ABSTRACT METHODS MUST BE IMPLEMENTED IN THE CHILD CLASS.

AN ABSTRACT CLASS CAN ALSO INCLUDE NON ABSTRACT METHODS. THESE METHODS ARE INHERITED BY THE CHILD CLASS AS USUAL.

```
abstract class WorkTable {
   abstract function jumpALine();
}
```

```
class MyTable extends WorkTable
{
   function jumpALine(){
       // the code goes here
   }
   …
}
```

TRAITS VS INTERFACES VS ABSTRACT CLASSES

TRAIT	INTERFACE	ABSTRACT CLASS
CONTAINS ONLY DEVELOPED METHODS THAT CAN BE CALLED IN THE CLASS THAT USES THE TRAIT	CONTAINS ONLY EMPTY METHODS THAT MUST BE DEVELOPED IN THE CLASS THAT IMPLEMENTS THE INTERFACE	CAN CONTAIN BOTH EMPTY (ABSTRACT) AND DEVELOPED (NON ABSTRACT) METHODS

ANONYMOUS CLASSES

ANONYMOUS CLASSES WERE INTRODUCED IN PHP 7. WE CAN USE THEM, INSTEAD OF A NAMED CLASS, WHEN ONLY A SINGLE AND THROWABLE OBJECT IS NEEDED

```
$obj = new class('Hi') {
    public $x;
    public function __construct($a) {
        $this->x = $a;
    }
};

echo $obj->x; // "Hi";
```

NAMESPACES

NAMESPACES ARE USEFUL TO ORGANIZE AND GROUP CLASSES. NAMESPACES ALSO ALLOW DEFINING MORE THAN A CLASS WITH THE SAME NAME (BUT IN DIFFERENT NAMESPACES).

WE DEFINE A NAMESPACE BY USING THE KEYWORD `namespace`. IT MUST BE THE FIRST SENTENCE IN THE PHP FILE.

```
<?php
namespace group01;
class SameName {
   …
}
?>
```

```
<?php
namespace group02;
class SameName {
   …
}
?>
```

NAMESPACES

FILE namespace01.php

```php
<?php
namespace group01;
class SameName {
   …
}
?>
```

FILE namespace02.php

```php
<?php
namespace group02;
class SameName {
   …
}
?>
```

FILE checkingNameSpaces.php

```php
<?php
include 'namespace01.php';  // including the external file
include 'namespace02.php';  // including the external file
use group01 as g1;  // an alias for the namespace
use group02 as g2;  // an alias for the namespace
$object01 = new g1\SameName(); // using external namespace
$object02 = new g2\SameName(); // using external namespace
?>
```

SOME CLASS USEFUL FUNCTIONS

Function	Description
`class_exists($className)`	Returns TRUE if the class exists, FALSE otherwise
`get_class_methods($className)`	Returns an array with the class method names
`get_class_vars($className)`	Returns an array with the class property names and their default values (only if they are visible from the current scope)
`get_declared_classes()`	Returns an array with the declared class names
`get_declared_interfaces()`	Returns an array with the declared interface names

SOME OBJECT USEFUL FUNCTIONS

Function	Description
`is_object($var)`	Returns TRUE if `$var` is an object
`get_class($obj)`	Returns the name of the class to which the object belongs to
`method_exists($obj,$meth)`	Returns TRUE if the obj has the specified method
`get_object_vars($obj)`	Returns an array with the object properties and their values (only if they are visible from the current scope)
`get_parent_class($obj)`	Returns the name of the parent class (FALSE if there is none)