Module: Object Oriented Programming with PHP

Module description:

Design a web application that can evolve over time and be reused by others. Learning the basics and technique of object-oriented programming. Study of development issues. Improving productivity, modularity and code clarification to save time .

- Object-Oriented Development Technique: Improving productivity, saving time through modularization, encapsulation, organization and code clarification.
- Object-oriented vs. process-oriented: Perception of development issues.
- Object conceptualization in a Web context: UML (Unified Modeling Language).
- Development environment - IDE : PhpStorm, Eclipse, etc.
- Classes & Objects.
- Class instantiation and references.
- Inferences, transformations and cloning.
- Visibility level.
- Handling and Belonging.
- Getter and Setter.
- Constructor and other predefined methods.
- Operators via keywords: This and Self.
- The implicit typing of arguments.
- Encapsulation and prioritization.
- Class inheritance.
- Principle of overload / abstraction / finalization / interface / line
- Design Pattern.
- Work with existing classes.
- Error management with exceptions.
- Namespace.
- PHP and MYSQL interfacing via Php Data Object (PDO).
- Use of Php libraries.
- Development of an object-oriented project: factorized, optimized, efficient and generic code.
- Commentary vs Code documentation.
- Object persistence: serialization, http transmission, database registration.
- Miscellaneous: Injection of addiction, polymorphism, collection, etc.

The benefits of OOP

Object Oriented Programming has many advantages, including but not limited to the following:

Reuse of code in different projects

Clearer and more organized design. Each part of the code becomes an object with a context, properties and specific actions ("Everything is object" and "1 class = 1 role")

The OOP is inspired by the real world for the description of objects (characteristics, functions, inheritance), which facilitates the learning of object logic (Example of animals, vehicles...etc)

Combined with exceptions, it improves and centralizes error management and debugging

It allows you to make modular code in order to work easily in a team

It allows to hide the complexity, the objects created are generally simple to use. On large and complex projects, some devs design the objects, others use them

Allows to go further by using design patterns to further improve the quality and structure of the code

**Instantiation

A class defines a structure (characteristics, behavior), which is then used to create objects.

We then speak of "class instance": each created object is an instance of the class that describes it.

Here `$dog` and `$cat` are instances of the Animal class, i. e. objects created from the model described in the Animal class.

```
class Animal {}

$dog = new Animal();

$cat = new Animal()
```

Each instance has its own property values, and a modification made on one instance does not affect the other instances.

Classes and properties

Classes

All the code describing an object is contained/encapsulated in a class.

The class is like a "mold" that describes the characteristics (class attributes/properties) and behavior (class methods) of an object.

The naming convention for declaring a class is the UpperCamelCase (1st letter of the words in upper case).

```
class MyObject {

// Code describing the object

}
```

Properties

The attributes/properties/characteristics of an object are described in a class as variables.

If no visibility is defined, we should at least use the keyword var to avoid error syntax. By default, attributes have public visibility (i.e. Visibility).

The naming convention for declaring a property is the underscore_case (each word separated by an underscore _).

```
class Animal {

  var $race;

  var $color;

  var $paws_count;

}


$animal = new Animal();

Animal->race ='dog'; // Access to the race attribute in writing, affects the value'dog'.

echo $animal->race; // Access to the race attribute in reading, displays "dog".
```

Static properties

Properties declared as static are accessible without instantiating the class, by using the keyword self::.

Example :

```php
class MyClass {

  public static $msg = "Hello world!";

  public static function sayHello() {

    echo self::$msg;

  }

}

MyClass::sayHello(); // Call sayHello and display Hello world !
```

$this

The methods of a class can access the properties and other methods of the class using the keyword $this.

It is followed by an arrow ->, and a property name without dollar $this->myPropertyorMethod$ this->myMethod()

Reminder: The keyword ***$this*** cannot be used in static methods (i.e. Static)

Example :

```php
class Product {

  private $name;

  public function getName() { {

    return $this->name;

  }


  public function setName($name) {

    $this->name = $name;

  }

}

$product = new Product();
```

```php
$product->setName('Product Name'); // Defines the value of the name property

echo $product->getName(); // Returns and displays the value of the name property
```

Constructor and destructor

The constructor and the destroyer are part of the magic methods of PHP.

These are methods declared and executed automatically by PHP under certain conditions.

The constructor

The constructor is called each time the class is instantiated.

In other words, each time a new object is declared $object = new MyClass(), the __construct() method is called in the MyClass class.

The __construct method receives all arguments passed to the instance.

Example :

```php
class Animal {

  public $race;

  public $color;

  public function __construct($race='', $color='') {

    $this->race = $race;

    $this->color = $color;

  }

}

$animal = new Animal(); // Implicit call to $animal->__construct()

dog = new Animal('dog','black'); // Implicit call to $dog->__construct('dog','black')
```

The destructor

The destroyer __destruct() is called at the end of the script execution or when there is no more reference to a given object.

Example :

```
class Animal {

  public function __destruct() {

    echo "Calling the destructor!

  }

}

$animal = new Animal();

// Implicit call $animal->__destruct() at the end of script loading

// Display the call of the destructor!
```

Methods

The actions/behaviors performed by an object are described in the class by functions called methods (or class methods).

By default, if no visibility is specified for a method, it is considered public (i.e. Visibility).

The methods of the class access the attributes of the class with the keyword $this(c.\ f.$this).

The naming convention for method declaration is the CamelCase (1st letter of the linked words in upper case).

Example :

```
class Animal {

  var $name;
```

```
  var $race;

  function getName() { {

    return $this->name;

  }

  showRace function() {

    echo $this->race;

  }

}

$animal = new Animal();

$animal->name ='doggy';

$animal->race = "dog";

echo $animal->getRace(); // Returns and displays doggy

$animal->showRace(); // Dog poster
```

Static methods

Methods declared as static are accessible without instantiating the class. They are called with the range resolution operator: ::.

They cannot access the other non-static properties and methods of the class and therefore cannot use the keyword $this.

They access the other static properties and methods of the class using the keyword self.

Example :

```
class MyClass {

  public static $msg = "Hello world!";


  public static function myStaticMethod() {
```

```
    echo self::$msg;
  }
  public static function anotherStaticMethod() {
    echo self::myStaticMethod();
  }
}
MyClass::anotherStaticMethod(); // Calls myStaticMethod() and displays Hello world !
```

Visibility

By default, all properties and methods of a class whose visibility is not declared are automatically considered public.

public (or var): the property or method is visible inside and outside the class

private: the property or method is only visible within the class

protected: the property or method is only visible inside the class and from its daughter classes (see Heritage)

By convention, the attributes and private/protected methods are prefixed with an underscore to better identify them.

Example :

```
class Animal {
  private $_race;
  public $color;
  private function _getRace() {
    return $this->_race;
  }
```

```
  public function showRace() {
    echo $this->_getRace();
  }

  public function setRace($race) {
    $this->_race = $race;
  }

  public function getColor() {
    echo $this->color;
  }
}

$animal = new Animal();
$animal->_race ='dog'; // Fatal error because the attribute _race is private so accessible only in the class
$animal->color ='black'; // No error because the attribute is public and therefore accessible from outside the class
$animal->setRace('dog'); // No error because the setRace method is public and therefore accessible from outside the class, and accesses the attribute _race inside the class, assigns the dog value to the attribute _race
$animal->_getRace(); // Fatal error because the _getRace method is private so accessible only in the class
$animal->showRace(); // No error because the showRace method is public and calls the _getRace method within the class, displays dog
$animal->getColor(); // No error because the getColor method is public
```

Inheritance and polymorphism

Inheritance

It is possible to extend classes: this mechanism is called inheritance.

For example, we can imagine a Car class that extends the Vehicle class.

```
class Vehicle {

/* Code */

}


class Car extends Vehicle {

/* Code */

}
```

The Car class will share the properties and methods of the Vehicle class.

It will add new features, to specialize it.

Example :

```
class Vehicle {

  private $color;

  public function drive() {

    /* ... */

  }

}


class Car extends Vehicle {

  private $electric windows;

  private $consumptionAt100;

  public function honk() {

    /* ... */

  }
```

```
}
```

A car can therefore move forward, but also honk its horn.

It has the possibility of having electric windows, it has a certain consumption, but it also has a colour (which is specific to all vehicles)

Child' classes can access the properties of the parent class, if these properties have been declared as "protected" (or "public", of course).

Method overload

It is possible to redefine a method created in the parent class.

Example :

```
class Vehicle {

  public function getMaxSpeed() {

    return 30;

  }

}


class Car extends Vehicle {

  public function getMaxSpeed() {

    return 180;

  }

}



$aCar = new Car();

echo $aCar->getMaxSpeed(); // Will display 180
```

It is possible, in a child class, to use a method from the parent class:

```
class Foo {

  public function printItem($string) {

    echo "Printing in Foo:". $string."<br />";

  }

}


class Bar extends Foo {

  public function printItem($string) {

    echo "Printing in Bar:". $string."<br />";

  }

}

$foo = new Foo();

$bar = new Bar();

$itemToPrint ='Test';

$foo->printItem($itemToPrint); // Will display Print in Foo ...

$bar->printItem($itemToPrint); // Will display Print in Bar ...
```

If we wanted, in addition, to use the Foo::printItem() method from the $bar object (Bar instance), we would have to declare the Bar class as follows:

```
class Bar extends Foo {

  public function printItem($string) {

    echo "Printing in Bar:". $string."<br />";

    parent::printItem($string);

  }

}
```

```php
$bar = new Bar();

$bar->printItem('Test');
```

Will display:

Printing in Bar: Test

Printing in Foo: Test