C1- Code & Go

pool_php_d07

Day 07

RPG!!

CODING ACADEMY
BY EPITECH

Exercice 1

1 Pt(s)

Hand in: pool_php_d07/ex_01/Character.php

Create a class "Character" composed of the protected attributes "name", "life", "agility", "strength", "wit", and a constant attribute "CLASSE" and the corresponding getters.

These attributes will have the following values:

- name = argument passed to constructor
- life = 50
- agility = 2
- strength = 2
- wit = 2

Example:

```php
<?php
$perso = new Character("Jean-Luc");
echo $perso->getName()."\n";
echo $perso->getLife()."\n";
echo $perso->getAgility()."\n";
echo $perso->getStrength()."\n";
echo $perso->getWit()."\n";
echo $perso->getClasse()."\n";
// displays "Jean-Luc", "50", "2", "2", "2" and "Character"
?>
```

Exercice 2

1 Pt(s)

Hand in: pool_php_d07/ex_02/Character.php
pool_php_d07/ex_02/Warrior.php
pool_php_d07/ex_02/Mage.php

Copy the code of the previous exercise and create the class "Warrior" as well as a class "Mage" which inherits from "Character".

You will modify the attributes (that have been inherited from the class "Character") of each class in

the following manner:
Warrior:

- CLASSE = "Warrior"
- life = 100
- agility = 10
- strength = 8
- wit = 3

Mage:

- CLASSE = "Mage"
- life = 70
- agility = 10
- strength = 3
- wit = 10

These two classes will each implement a method "attack" that takes no parameter.

The method "attack" of class "Warrior" will display: "[NAME]: I'll crush you with my hammer!" followed by a new line.

The method "attack" of class "Mage" will display: "[NAME]: Feel the power of my magic!" followed by a new line.
[NAME] will of course be replaced by the name of your character.
Our characters are proud and they like to announce themselves on the battlefield. You will make sure that, when creating an object "Warrior" or "Mage", a message will be written in the following format:

- Warrior: "[NAME]: I'll engrave my name in history!" followed by a new line.
- Mage: "[NAME]: May the gods be with me." followed by a new line.

Even though they are powerful and proud, our characters can die. Upon destruction of one of the two objects, you will have to display the following message:

- Warrior: "[NAME]: Aarrg I can't believe I'm dead..." followed by a new line.
- Mage: "[NAME]: By the four gods, I passed away..." followed by a new line.

Example:

```php
<?php
$warrior = new Warrior("Jean-Luc");
$mage = new Mage("Robert");
$warrior->attack();
$mage->attack();

// displays
// "Jean-Luc: I'll engrave my name in history"
// "Robert: May the gods be with me."
// "Jean-Luc: I'll crush you with my hammer!"
// "Robert: Feel the power of my magic!"
// "Robert: By the four gods, I passed away..."
// "Jean-Luc: Aarrg I can't believe I'm dead..."
?>
```

Exercice 3

1 Pt(s)

Hand in: pool_php_d07/ex_03/Character.php
pool_php_d07/ex_03/Warrior.php
pool_php_d07/ex_03/Mage.php
pool_php_d07/ex_03/IMovable.php

In this exercise, you will reuse the classes of the previous exercises. Please copy them in your folder.
We now have characters which can be mages or warriors. They can attack, fair enough, but they still cannot move! This is bothersome. In order to add this behavior to our classes, we are going to create an interface "IMovable" that will contain the following methods: "moveRight", "moveLeft", "moveUp" and "moveDown".
You will then implement this interface to the class "Character".
Your methods will, respectively, display the following message:

- moveRight -> "[NAME]: moves right." followed by a new line.
- moveLeft -> "[NAME]: moves left." followed by a new line.
- moveUp -> "[NAME]: moves up." followed by a new line.
- moveDown -> "[NAME]: moves down." followed by a new line.

Example:

```
$warrior = new Warrior("Jean-Luc");
$warrior->moveRight();
$warrior->moveLeft();
$warrior->moveDown();
$warrior->moveUp();

// displays
// "Jean-Luc: I'll engrave my name in history!"
// "Jean-Luc: moves right."
// "Jean-Luc: moves left."
// "Jean-Luc: moves down."
// "Jean-Luc: moves up."
// "Jean-Luc: Aarrg I can't believe I'm dead..."
```

Exercice 4

1 Pt(s)

Hand in: pool_php_d07/ex_03/Character.php
pool_php_d07/ex_04/Warrior.php
pool_php_d07/ex_04/Mage.php
pool_php_d07/ex_04/IMovable.php

Paralysis is over! Our characters can now move but, being so proud, they want more! Our friend Warrior refuses to be compared to a small and skinny Mage. While the Warrior moves in a bold and virile manner, the Mage moves delicately! To satisfy our boorish Warrior, you will implement overrides for the methods "IMovable" inherited by "Character".
Thus, your methods will display the following messages corresponding to the class that overrides them: Warrior:

- moveRight -> "[NAME]: moves right like a bad boy." followed by a new line.
- moveLeft -> "[NAME]: moves left like a bad boy." followed by a new line.
- moveUp -> "[NAME]: moves up like a bad boy." followed by a new line.
- moveDown -> "[NAME]: moves down like a bad boy." followed by a new line.

Mage:

- moveRight -> "[NAME]: moves right furtively." followed by a new line.
- moveLeft -> "[NAME]: moves left furtively." followed by a new line.
- moveUp -> "[NAME]: moves up furtively." followed by a new line.
- moveDown -> "[NAME]: moves down furtively." followed by a new line.

Example:

```php
<?php
$warrior = new Warrior("Jean-Luc");
$warrior->moveRight();
$warrior->moveLeft();
$warrior->moveUp();
$warrior->moveDown();

$mage = new Mage("Robert");
$mage->moveRight();
$mage->moveLeft();
$mage->moveUp();
$mage->moveDown();

// displays

// "Jean-Luc: I'll engrave my name in history!"
// "Jean-Luc: moves right like a bad boy."
// "Jean-Luc: moves left like a bad boy."
// "Jean-Luc: moves up like a bad boy."
// "Jean-Luc: moves down like a bad boy."
// "Robert: May the gods be with me."
// "Robert: moves right furtively."
// "Robert: moves left furtively."
// "Robert: moves up furtively."
// "Robert: moves down furtively."
// "Robert: By the four gods, I passed away..."
// "Jean-Luc: Aarrg I can't believe I'm dead..."
```

Exercice 5

1 Pt(s)

Hand in: pool_php_d07/ex_05/Character.php
pool_php_d07/ex_05/Warrior.php
pool_php_d07/ex_05/Mage.php
pool_php_d07/ex_05/IMovable.php

Now our characters can talk, walk and attack in a customized way. Yet, they still cannot unsheathe their weapon! Being able to attack is nice, but attacking while the weapon is still in its sheath is going to be difficult…
You will agree that, whether Warrior or Mage, the character will draw his weapon the same way. This is why you will make sure that the class "Character" implements the method "unsheathe" so that
"Warrior" and "Mage" inherits from it. However, you will also make sure that the method "unsheathe" cannot be overrided by "Warrior" and "Mage".
This method will display the following text when it is called: "[NAME]: unsheathes his weapon."
Example:

```php
<?php

$perso = new Mage("Jean-Luc");

$perso->unsheathe();

// displays

// "Jean-Luc: May the gods be with me."

// "Jean-Luc: unsheathes his weapon."

// "Jean-Luc: By the four gods, I passed away..."
```

Exercice 6

1 Pt(s)

Hand in: pool_php_d07/ex_06/Pony.php

Create a class "Pony" with 3 public attributes: "gender", "name" and "color".

Make sure you can set these 3 attributes directly during construction (in the same order as mentioned before).

Make sure that, upon destruction of the pony the string "I'm a dead pony." followed by a new line is displayed.

Also make sure that, when you use the command "echo" on a "Pony", the message "Don't worry, I'm a pony!" followed by a new line is displayed.

Exercice 7

1 Pt(s)

Hand in: pool_php_d07/ex_07/Pony.php

Copy your previous exercise.

Add a public method called "speak" that displays "Hiii hiii hiii" followed by a new line.

Make sure that when you call a method that doesn't exist in the class "Pony", you display "I don't know what to do..." followed by a new line.

Exercice 8

1 Pt(s)

Hand in: pool_php_d07/ex_08/Pony.php
Mandatory: Use of : --get and --set

Copy the code from exercise 7.

Put all attributes in private, but make sure that you can still access them with get and set.

When you access with get, this must display "It's not right to get a private attribute!" followed by a new line, but still return the attribute.

When you access with set, this must display "It's not right to set a private attribute!" followed by a new line, then assign the attribute of the value.

If you set or get a non-existing attribute, it must display: "There is no attribute: " followed by the name of the attribute, a dot, and a new line.

Exercice 9

1 Pt(s)

Hand in: pool_php_d07/ex_09/Gecko.php

Create a class "Gecko" that will have 2 public attributes: "friends" of type array and "skills" of type Skill, a class we will define ourselves. You do not have to hand in this "Skill" class.

The constructor will take these 2 attributes as parameter (in the same order as mentioned above).

Make sure that it is not possible to pass a type which is different than the initial type of each attribute.

In addition, it will be possible to pass the type "null" to the argument corresponding to the attribute "friends".

Exercice 10

1 Pt(s)

Hand in: pool_php_d07/ex_10/character.class.php

Create a class named "Character" with private attributes "name", "strength, "magic", "intelligence" and "life". The value of the attribute “name” can be sent at the object creation. If it is not sent in parameters, it will take the value $i. The values by default during instantiation will be, respectively, $i, 0, 0, 0 and 100.

Make the protected methods "getName", "getStrength", "getMagic", "getIntelligence" and "getLife".

Make an implementation of the method "--toString" in order to match the displayed example.

Prototype: --construct([string $name]);

$i will have to be replaced by the occurrence of the number of characters created without any name. The first created "1", the second "2", the third "3", and so on and so forth...

```php
<?php
require_once("character.class.php");

foreach ([new Character, new Character("Julien"), new Character] as $character)
{
    echo $character;
}

?>

// displays:
// My name is Character 1.
// My name is Julien.
// My name is Character 2.
```

Exercice 11

1 Pt(s)

Hand in: pool_php_d07/ex_11/ex_11.php
Mandatory: Use of the function spl_autload_register

Create a file that, when included, will make it possible to use any class defined in a file of its own. These class definition files will be named like this: ''name_of_the_class.class.php''.

Example:

If the class ''Gecko'' exists, it will be defined in the file ''Gecko.class.php''. You should be able to instantiate this class just by including your file.

Exercice 12

1 Pt(s)

Hand in: pool_php_d07/ex_12/ex_12.php
Mandatory: Use of the keyword “clone”

Create a class "Dolly" with 3 public attributes: "age", "animal" and "doctor".
The constructor takes these 3 parameters as parameter (in the same order as mentioned above).

Make sure that when you clone an object of that class, "I will survive !" is displayed followed by a new line.

Outside of that class, create a function "clone_object" that takes as parameter an object named "object" and that returns a clone of the same object.

Exercice 13

1 Pt(s)

Hand in: pool_php_d07/ex_13/ex_13.php

Create a function "objects_comparison" which takes two objects as parameters named respectively "object1" and "object2".

It should display "Objects are equal." followed by a new line if both objects are equal.

And "Objects are the same." followed by a new line if both objects references have the same instance.

Otherwise, in all other cases, it should display a simple return to the line.

Exercice 14

1 Pt(s)

Hand in: pool_php_d07/ex_14/ex_14.php

In this exercise, you must call 5 times the function "call_gecko" (that doesn't take any parameter).

You must pay attention to catch errors if they are "thrown" by the function "call_gecko", and display them with the method of the Exception class "getMessage()" followed by a newline.

The call_gecko function will be created by the moulinette. Do not insert it in your file!

Exercice 15

2 Pt(s)

Hand in: pool_php_d07/ex_15/ex_15.php

Soldiers form the basis of an army, but the question is: to which army do they belong?

Your goal is to create 2 classes of Soldier. One will belong to the namespace Imperium and the other to Chaos. A soldier has 3 private attributes: hp, attack and name, as well as public getter/setter.

The constructor of Soldier takes as parameters name, hp and attack.

By default, the Imperium soldiers will have 50 hp and 12 attack. The Chaos soldiers will have 70 hp and 12 attack. The attribute "name" is necessary during object's creation.

A soldier also has a public method doDamage, taking as parameter a soldier object and reducing the hp of the latter by the attack quantity of the attacking soldier.

One last thing: when a soldier is called (by an echo for example), it will display "[$name] the [namespace] Space Marine : [$hp] HP." without a new line.
The following code must work:

```
<?php
$spaceMarine = new \Imperium\Soldier("Gessart");
$chaosSpaceMarine = new \Chaos\Soldier("Ruphen");

echo $spaceMarine,"\n";
echo $chaosSpaceMarine,"\n";

$spaceMarine->doDamage($chaosSpaceMarine);

echo $spaceMarine,"\n";
echo $chaosSpaceMarine,"\n";
?>
```

And display :

Gessart the Imperium Space Marine : 50 HP.
Ruphen the Chaos Space Marine : 70 HP.
Gessart the Imperium Space Marine : 50 HP.
Ruphen the Chaos Space Marine : 58 HP.

Exercice 16

2 Pt(s)

Hand in: pool_php_d07/ex_16/ex_16.php

Using the classes of the previous exercise, you will now create a class Scanner having a method "scan" taking a soldier as parameter. If the soldier belongs to the namespace Imperium, the function will display "Praise be, Emperor, Lord." followed by a new line. Otherwise, it will display "Xenos spotted." followed by a new line.

Exercice 17

2 Pt(s)

Hand in: pool_php_d07/ex_17/ex_17.php

Your army now needs a doctor called Apothecary. His mission is to heal soldiers of the class

Imperium and its children thanks to a method "heal" taking an object as parameter. When the unit is part of Imperium, your apothecary will shout "No servant of the Emperor shall fall if I can help it.",

followed by a new line. If the unit does not belong to Imperium, he will shout "The enemies of the Emperor shall be destroyed!" followed by a new line.

The unit will be called in the following way:

```
<?php
class Imperium { }
class SpaceMarine extends Imperium { }
class Heretic { }

Apothecary::heal(new Imperium());
Apothecary::heal(new SpaceMarine());
Apothecary::heal(new Heretic());
?>
```

And will display:

No servant of the Emperor shall fall if I can help it.
No servant of the Emperor shall fall if I can help it.
The enemies of the Emperor shall be destroyed!

It should be impossible to instantiate an apothecary!

Bonus

Informations

Your goal is to create a kind of “moulinette”.

For each exercise, you will create a class Gecko, with a private attribute _name that will be passed as parameter to its constructor and a public method correct($object).

For the method correct$\$object$:
The prototype of this method will be different for all exercises. Your method must make all necessary checks in this $object to verify that the student's work answers all points of the given scale.
For each successful check of an instruction, you must display:
"Test <number> : Good!" followed by a new line.
Otherwise, you must display:
"Test <number> : KO." followed by a new line.

Scale:

- The student creates a class Soldier with private attributes name, attack and hp.
- The constructor of a Soldier resets the private attributes with the parameters passed to it in the same order. By default, attack has for value: 50 and hp: 12.
- A Soldier has the public getters/setters of its 3 private attributes (get/setName/Attack/Hp).
- Soldier::gardeAVous() takes no parameters and displays: "Soldier <name> report on duty! I have <attack> in ATK and <hp> hitpoints!\n"

Student hand in:

```php
<?php
class Soldier
{
    private $name;
    private $attack;
    private $hp;

    function __construct($_name, $_attack = 12, $_hp = 50){
        list($this->name, $this->hp, $this->attack) = array($_name, $_hp, $_attack);
    }

    public function gardeAVous() {
        echo ("Soldier " . $this->name . " report on duty! I have " . $this->attack . " in ATK and
            " . $this->hp . " hit points!\n");
    }

    public function getName() { return ($this->name); }
    public function getAttack() { return ($this->attack); }
    public function getHP() { return ($this->hp); }
    public function setName($name) { $this->name = $name; }
    public function setAttack($attack) { $this->attack = $attack; }
    public function setHP($hp) { $this->hp = $hp; }

}
?>
```

Expected result on YOUR end:

```php
<?php
$gecko = new Gecko("Gecko");
echo ($gecko->getName() . " starts to check:\n");
$soldat = new Soldier("James Francis Ryan");
$gecko->correct($soldat);

/*
  Must display:
  Gecko starts to check:
  Test 0 : Good !
  Test 1 : KO.
  Test 2 : KO.
  Test 3 : Good !
*/

?>
```

Bonus 01

2 Pts

Hand in: pool_php_d07/ex_18/ex_18.php
Restriction: Use of ReflectionClasse alternatives

- The student creates a class Arcaniste that implements the IPerso interface.
- The class Arcaniste extends the abstract class AUnit. AUnit must not be instantiable.

You do not know what the classes AUnit and IPerso contain? It's on purpose! But you must still verify that the object Arcaniste inherits from them correctly!

Prototpe: bool correct($arcaniste);

Bonus 02

2 Pts

Hand in: pool_php_d07/ex_19/ex_19.php
Restriction: Use of ReflectionClasse alternatives

- Verify that all the classes created by the student (located in the array $my_classes) belong to at least one namespace of the second array passed as parameter.
- Verify that all the classes created by the student are not clonable, are final, implement no interface and do not inherit from another.
- Verify that each class of the student has the same attributes and methods as all other classes present in the array (with the same accessibility). You do not need to check if the functioning of these methods is identical from one class to another.

Prototpe: bool correct($my_classes, $my_workspaces);

Bonus 03

2 Pts

Hand in: pool_php_d07/ex_20/ex_20.php
Restriction: Use of ReflectionClasse alternatives

- The student creates a class Soldier with private attributes name, attack and hp.
- The constructor of a Soldier resets the private attributes with parameters passed in the same order. By default, attack has for value: 12 and hp: 50.
- A Soldier has the public getters/setters of its 3 private attributes (get/setName/Attack/Hp). They must run correctly.
- Soldier::doDamage($soldier) subtracts the attack value of the called Soldier from the HP of the Soldier passed as parameter.
- The public method doDamage($soldier) must display: "Haha! Take these <attack> points of damage, <name of victim>! \n"

Prototpe: bool correct(Soldier $soldier);