W1 – Piscine PHP

Jour 11

Responsables Pédagogiques

pedagowac@epitech.eu

Bienvenue dans cette onzième journée de piscine Web@cadémie ! ☺

Dans cette journée nous allons apprendre de nouvelles notions telles que les exceptions, les namespaces ainsi que la réflexivité (ou comment créer une pangolinette).

Sommaire

Exercice 01 (3 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_01.php

Restrictions : Aucune

Dans cet exercice vous devrez appeler 5 fois la fonction “call_pangolin” (qui ne prend aucun paramètre). Vous devrez faire attention à catch les erreurs si jamais elles sont “throw” par la fonction “call_pangolin” et à les afficher avec la méthode de la classe Exception “getMessage()”.

La fonction call_pangolin sera créée par la pangolinette, ne l’insérez pas dans votre fichier.

Exercice 02 (3 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_02.php

Restrictions : Aucune

Créez une interface “iCars” dans laquelle vous implémenterez les méthodes suivantes :

- getPrice() qui retourna la valeur de l’attribut privé “_price”
- getWeight() qui retourna la valeur de l’attribut privé “_weight”
- mineIsBigger($obj) sur laquelle nous reviendrons plus tard dans le sujet.

Créez ensuite les classes “BMW” et “Suzuki” qui implémenteront l’interface “iCars”. Vous ajouterez les attributs privés “$_price” et “$_weight”. Il doit être possible d’instancier des objets issus de ces 2 classes en indiquant leur prix ET leur poids ou en indiquant seulement leur prix. Si aucun poids n’est passé en paramètre vous assignerez “4242” à l’attribut “$_weight”.

Implémentez la méthode statique “lessExpensive” dans ces deux classes. Cette méthode devra retourner 15000 pour la classe “BMW” et 5000 pour la classe “Suzuki” et pourra être appelé sans instance d’objet.

La méhode mineIsBigger devra prendre un objet en paramètre. S’il s’agit d’une “Toyota” vous devrez afficher “Mine is bigger”, s’il s’agit d’une “Smart” vous devrez afficher “Mine is way bigger !” et enfin, s’il s’agit d’un “Velib” vous devrez afficher “LOL”. Dans tous les autres cas vous afficherez “Show me !”.

Exercice 03 (3 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_03.php

Restrictions : Aucune

Les soldats sont la base d'une armée, mais à quelle armée appartiennent-ils, là est la question. Votre objectif sera de créer 2 classes Soldier. L'une fera partie du namespace Imperium et l'autre du Chaos. Un soldier possède 3 attributs privés : hp, attack et name, ainsi que leurs getter/setter publiques (get/setHp, get/setAttack)

Le constructeur du Soldier prendra en paramètre un name, un hp et un attack. Par défaut, les soldats de l'Imperium auront 50 hp et 12 attack, les soldats du Chaos auront, quant à eux, 70 hp et 12 attack. Un soldier possède aussi une méthode publique doDamage, prenant en paramètre un object soldier et réduisant les hp de ce dernier par la quantité d'attack du soldat attaquant.

Une dernière chose : Lorsqu'un soldat est appelé (via un echo par exemple), il devra afficher "[$name] the [namespace] Space Marine : [$hp] HP." sans retour à la ligne

Le code suivant devra fonctionner :

```
$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";
```

Et devra afficher :

```
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 04 (3 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_04.php

Restrictions : Aucune

En reprenant les classes de l'exercice précèdent, vous allez à présent créer une classe Scanner possédant une méthode statique "scan" prenant en paramètre un soldat. Si le soldat fait partie du namespace Imperium, la fonction affichera "Praise be, Emperor, Lord." suivi d'un retour à la ligne. Sinon, elle affichera "Xenos spotted." suivi d'un retour à la ligne.

Exercice 05 (3 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_05.php

Restrictions : Aucune

Votre armée a maintenant besoin d'un médecin appelé Apothecary. Sa mission est de soigner les soldats de la class Imperium et de ses enfants grâce à sa méthode heal prenant en paramètre un objet. Lorsque l'unité fait bien partie de l'imperium, votre apothicaire criera "No servant of the Emperor shall fall if I can help it.", suivi d'un retour à la ligne. Si l'unité ne fait pas partie de l'Imperium, il criera alors "The enemies of the Emperor shall be destroyed!" suivi d'un retour à la ligne.

L'unité sera appelée de cette manière :

```
class Imperium { }
class SpaceMarine extends Imperium { }
class Heretic { }

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

Et devra afficher :

```
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!
```

Il ne doit pas être possible d'instancier un Apothecary !

A lire pour les exercices 06 à 08

Pour chaque exercice, vous devez créer une classe **Pangolin**, ayant un attribut privé **_name** qui sera passé en paramètre à son constructeur et une méthode publique **correct(**$object**)**.

Méthode correct($object) :

$object est une instance d'une classe écrite par l'étudiant que vous corrigez. Le prototype de cette méthode ne sera pas le même pour tous les exercices, il vous sera donné à chaque fois. Votre méthode doit faire toutes les vérifications nécessaires sur cet objet pour vérifier que le rendu de l'étudiant réponde bien à chaque point du barème donné. Pour chaque consigne respectée il faudra afficher :

« Test <numero> : Good !\n »

Sinon, vous devez afficher : « Test <numero> : KO.\n »

Barème :

1. L'étudiant crée une classe Soldat avec pour attributs privés name, attack et hp.

2. Le constructeur d'un Soldat initialise ses attributs privés avec les paramètres qui lui sont passés dans le même ordre. Par défaut, attack aura pour valeur : 50 et hp : 12.

3. Un Soldat a les getters/setters publiques de ses 3 attributs privés (get/setName/Attack/Hp).

4. Soldat::gardeAVous() ne prend pas de paramètre et affiche :
« Soldat <name> au rapport ! J'ai <attack> en ATK et <hp> points de vie !\n »

Rendu de l'étudiant :

```
class   Soldat
{
        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 ('Soldat ' . $this->name . ' au rapport ! J\'ai ' . $this->attack . ' en ATK et '
. $this->hp . ' points de vie !\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; }
}
```

Résultat attendu de VOTRE rendu :

```
$blemus_r = new Pangolin(“Remi”);
echo ($blemus_r->getName() . ‘ commence a corriger :\n’) ;
$soldat = new Soldat(“James Francis Ryan”);
$blemus_r->correct($soldat);
```

Doit afficher :

```
Remi commence a corriger :
Test 0 : Good !
Test 1 : KO.
Test 2 : KO.
Test 3 : Good !
```

Ne rendez jamais les classes « de votre étudiant ». Vous devez uniquement rendre une classe Pangolin dont seule la méthode correct() changera d’un exercice à un autre.

Nous ne devons rien lire hormis les résultats de votre correction. Même si vous devez vérifier ce qu’affiche une méthode de l’étudiant, elle ne doit pas apparaitre à l’écran.

Tous vos exercices doivent fonctionner en utilisant des ReflectionClass pour analyser celles de vos « étudiants ». Un appel direct aux méthodes ou propriétés d’une instance immédiate de leur classe vous rapportera un 0.

Exercice 06 (2 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_06.php

Restrictions :

- N'utiliser aucune alternative aux ReflectionClass.

1. L'étudiant crée une classe Arcaniste qui implémente l'interface iPerso.

2. La classe Arcaniste étends la classe abstraite aUnit. aUnit ne doit pas pouvoir être instanciable.

Vous ne savez pas ce que contiennent la classe Unit et iPerso ? C'est voulu ! Mais vous devez tout de même vérifier que l'objet Arcaniste en hérite bien comme il faut !

Prototype : bool correct($arcanist);

Exercice 07 (2 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_07.php

Restrictions :

- N'utiliser aucune alternative aux ReflectionClass.

1. Vérifiez que toutes les classes créées par l'étudiant (se trouvant dans le tableau $my_classes) fassent partie d'au moins un namespace du 2è tableau passé en paramètre.

2. Vérifiez que toutes les classes créées par l'étudiant ne soient pas clonables, soient finales, n'implémentent aucune interface et n'héritent d'aucune autre.

3. Vérifiez que chaque classe de l'étudiant ait les mêmes attributs et les mêmes méthodes que toutes les autres classes présentent dans le tableau (avec la même accessibilité). Vous ne devrez toutefois pas vérifier que le fonctionnement de ces méthodes est identique d'une classe à l'autre.

Prototype : bool correct(array $my_classes, array $namespaces);

Exercice 08 (1 pts)

Nom du rendu : Piscine_PHP_Jour_11/ex_08.php

Restrictions :

- N'utiliser aucune alternative aux ReflectionClass.

1. L'étudiant crée une classe Soldier avec pour attributs privés name, attack et hp.

2. Le constructeur d'un Soldier initialise ses attributs privés avec les paramètres qui lui sont passés dans le même ordre. Par défaut, attack aura pour valeur : 12 et hp : 50.

3. Un Soldier a les getters/setters publiques de ses 3 attributs privés (get/setName/Attack/Hp). Ils doivent être fonctionnels

4. Soldier::doDamage($soldier) retire la valeur d'attaque de Soldier appelé aux HP du Soldier passé en paramètre.

5. La méthode publique doDamage($soldier) doit écrire :
« Haha ! Encaisse ces <attack> points de degats, <nom de la victime> !\n »

Prototype : bool correct(Soldier $soldier);