W1 - Piscine PHP

W-WEB-024

Jour 11

Programmation orientée objet

1.3

Jour 11

repository name: poolphpday11
language: php

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

Pour garder votre répertoire propre, regardez du côté de `gitignore`.

Informations

Avant de commencer

- Lisez attentivement toutes les consignes.
- Consultez vos mails plusieurs fois par jour, tous les jours.

Commencez par lire vos mails tout de suite à l'adresse : mail.office365.com.

- C'est une pangolinette (un programme) qui corrige vos exercices. Vérifiez le travail que vous allez rendre afin qu'il respecte scrupuleusement les consignes.
- Vous devez respecter les restrictions qui sont imposées dans chaque exercice. Dans le cas contraire, la pangolinette va considérer votre travail comme **triche** en attribuant la note de **-42**.
- Vous êtes susceptibles à tout moment de recevoir des corrections intermédiaires.

Pour bénéficier de corrections intermédiaires, vous devez chaque jour :

- Être inscrit au projet et aux activités dans l'intranet.
- Tenir à jour régulièrement le dépôt.

- Ne laissez jamais votre session ouverte sans surveillance.

JOUR 11

Afin de devenir de bons développeurs il vous faut être rigoureux et méthodique. Cela s'applique aussi à la qualité du code que vous allez produire.
Pour cela, vous devez respecter des normes de code ! Pour le PHP, par exemple, vous devez respecter la norme PSR-12.

Toute erreur de norme entraînera un malus conséquent sur votre note finale.

Votre répertoire ne doit pas contenir de fichiers inutiles (fichiers temporaires, …).
N'oubliez pas de push régulièrement vos fichiers, sans cela, pas de correction.

Pensez à créer votre répertoire en début de journée et à envoyer votre travail via **git**!
Le nom du répertoire est spécifié dans les instructions pour chaque étape/exercice.
Pour garder votre répertoire propre, regardez du côté de `gitignore`.

N'oubliez pas de vous inscrire à toutes les activités possibles de la semaine.

ÉTAPE 1 (4PTS)

Nom de rendu : 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.

error handling

ÉTAPE 2 (4PTS)

Nom de rendu : ex_02.php
Restrictions : Aucune.

Créez une interface **“iCars”** dans laquelle vous implémenterez les méthodes suivantes :
- `getPrice()` qui retournera la valeur de l’attribut privé `_price`
- `getWeight()` qui retournera 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 appelée sans instance d’objet.
La méthode `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 !”**.

static methodes

ÉTAPE 3 (4PTS)

Nom de rendu : 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**[**$namespace$**]**Spac**[hp] HP."** sans retour à la ligne.

Exemple :

Terminal

```
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";

// 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.
```

Étape 4 (4pts)

Nom de rendu : ex_04.php
Restrictions : Aucune.

En reprenant les classes de l'exercice précèdent, vous allez à présent créer une classe **Scanner** possédant une méth-ode 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.

A LIRE POUR LES EXERCICES 05 à 06 (OPTS)

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:

Terminal

```
class Soldat
{
    private $name;
    private $attack;
    private $hp;
    function __construct($_name, $_attack = 12, $_hp = 50)
    {
      list($this->name, $this->hp, $this->array) = array($_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 :

Terminal

```
$blemus_r = new Pangolin("Remi");
echo("$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 apparaître 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 vaudra 0.

ÉTAPE 5 (2PTS)

Nom de rendu : ex_05.php
Restrictions : N'utiliser aucune alternative aux `ReflectionClass`.

L'étudiant crée une classe `Arcaniste` qui implémente l'interface `iPerso`.
La classe `Arcaniste` étend la classe abstraite `aUnit`. `aUnit` ne doit pas pouvoir être instanciable.

Prototype: `void correct($arcanist)`

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

Barème :

- Test 0 : La classe Arcaniste existe et est instantiable.
- Test 1 : La classe Arcaniste implémente l'interface iPerso.
- Test 2 : La classe Arcaniste étends aUnit.
- Test 3 : La classe aUnit ne doit pas être instanciable.

ÉTAPE 6 (2PTS)

Nom de rendu : ex_06.php
Restrictions : N'utiliser aucune alternative aux `ReflectionClass`.

Vérifiez que toutes les classes créées par l'étudiant (se trouvant dans le tableau **$my_classes**) fassent partie d'au moins un namespace du 2ème tableau passé en paramètre.
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.
Vérifiez que chaque classe de l'étudiant ait les mêmes attributs et les mêmes méthodes que toutes les autres classes présentes 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: `void correct(array $my_classes, array $namespaces);`

Barème :

- Test 0 : Toutes les classes font partie d'au moins un namespace fourni.
- Test 1 : Toutes les classes doivent être finales, non clonables, ne doivent pas hériter ni implémenter d'autres classes / interfaces.
- Test 2 : Toutes les classes ont les mêmes attributs et méthodes, avec la même accessibilité.