W1- Piscine PHP

W-WEB-024

Jour 09

Programmation orientée objet

v1.0

EPITECH.
L'ECOLE DE L'INNOVATION ET DE L'EXPERTISE INFORMATIQUE

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. Le cas contraire, la pangolinette va considérer 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 :

- Etre inscrit au projet et aux activités dans l'intranet.
- Avoir créé le dépôt avec BLIH.
- Tenir à jour régulièrement le dépôt.

- Ne laissez jamais votre session ouverte sans surveillance.

Jour 09

Programmation orientée objet

Nom du répertoire:	Piscine_PHP_Jour_09
Droits de ramassage:	ramassage-tek
langage:	php
Taille du groupe:	1

- Votre repertoire ne doit pas contenir de fichiers inutiles (fichiers temporaires, ...)
- Vous ne devez pas oublier votre fichier *auteur*, si vous l'oubliez, la moulinette ne pourra pas vous corriger.
- 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 possible de la semaine.

Etape 1

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_01.php`
Restrictions: Aucune

Créez une classe **"MyDisplay"** composée d'une méthode publique **"display"** qui se contentera d'afficher **"Hello World !"** suivi d'un retour à la ligne à chaque appel.

Exemple:

```
$foo = new MyDisplay();
$foo->display();
/* affiche
Hello World !
*/
```

Etape 2

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_02.php`
Restrictions: Aucune

Créez une classe **"MyAttribute"** composée d'une méthode publique **"display"** qui se contentera d'afficher l'attribut passé en paramètre du constructeur, suivi d'un retour à la ligne.

Exemple:

```
$foo = new MyAttribute("Jean-Luc");
$foo->display();
/* affiche
Jean-Luc
*/
```

Etape 3

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_03.php`
Restrictions: Aucune

Créez une classe **"MyAttributes"** composée de deux attributs privés, de leurs getters et setters (**"getA"**, **"getB"**, **"setA"** et **"setB"**) ainsi que d'une méthode **"display"** qui se contentera d'afficher les 2 attributs séparés d'un espace et suivi d'un retour à la ligne.

Prototype du constructeur: public function __construct(string $a, string $b);
Exemple:

```
$foo = new MyAttributes("Hello", "World");
$foo->display();
/* affiche
Hello World
*/
```

Etape 4

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_04.php`
Restrictions: Aucune

Créez une classe **"MyTinyCalculator"**, composée de 3 attributs privés **"a"**, **"b"** et **"result"**, ainsi que de leurs getters et setters (getA, setA, getB, setB, setResult, getResult).
"result" contiendra le résultat de la dernière opération effectuée.
Ce résultat sera accessible grâce à la méthode public **"showResult"**.
La classe contiendra 4 méthodes publiques ne prenant aucun paramètre : **"add"**, **"subtract"**, **"multiply"** et **"divide"**.
Elles renverront le résultat de leur opération respective.

Prototype du constructeur: public function __construct(number $a, number $b);
Exemple:

```
$foo = new MyTinyCalculator(30, 12);
echo $foo->add()."\n";
echo $foo->subtract()."\n";
echo $foo->multiply()."\n";
echo $foo->divide()."\n";
echo $foo->showResult()."\n";
/* affiche
42
18
360
2.5
2.5
*/
```

Etape 5

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_05.php`
Restrictions: Aucune

Créez une classe **"Character"** composée des attributs protégés **"name"**, **"endurance"**, **"agility"**, **"strength"**, **"mana"**, de la constante **"CLASSE"** et des getters qui y correspondent. Ces attributs auront les valeurs suivantes:

- name = argument passé au constructeur
- endurance = 50
- agility = 2
- strength = 2
- mana = 2

Exemple:

```
$perso = new Character("Jean-Luc");
echo $perso->getName()."\n";
echo $perso->getEndurance()."\n";
echo $perso->getAgility()."\n";
echo $perso->getStrength()."\n";
echo $perso->getMana()."\n";
echo $perso->getClasse()."\n";
/* affiche
Jean-Luc
50
2
2
2
Character
*/
```

Etape 6

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_06.php`
Restrictions: Aucune

Copiez le code de l'exercice précédent et créez une classe **"Paladin"** ainsi qu'une classe **"Mage"** héritant de **"Character"**. Vous modifierez les attributs de chacune de ces classes de la manière suivante :

- Paladin :
  - CLASSE = "Paladin"
  - endurance = 100
  - strength = 10
  - agility = 8
  - mana = 3
- Mage :
  - CLASSE = "Mage"
  - endurance = 70
  - strength = 3
  - agility = 10
  - mana = 10

Ces deux classes devront implémenter chacune une méthode **"attack"** qui ne prend aucun paramètre. La méthode **"attack"** de la classe **"Paladin"** devra afficher: **"[NAME]: I'll crush you with my hammer !"** suivi d'un retour à la ligne. La méthode **"attack"** de la classe **"Mage"** devra afficher: **"[NAME]: Feel the power of my magic !"** suivi d'un retour à la ligne. [NAME] sera bien évidement remplacé par le nom de notre personnage.
Nos personnages sont assez orgueilleux et aiment bien s'annoncer sur le champ de bataille.

Vous ferez donc en sorte qu'à la création d'un objet **"Paladin"** ou **"Mage"**, un message soit écrit ce dernier devra être de la forme suivante :

- Paladin : **"[NAME]: I'll engrave my name in the history !"** suivi d'un retour à la ligne
- Mage : **"[NAME]: May the gods be with me."** suivi d'un retour à la ligne

Bien que puissants et orgueilleux, nos personnages peuvent mourir.
Vous ferez donc en sorte d'afficher un message à la destruction d'un de ces deux objets, ce message sera le suivant :

- Paladin : **"[NAME]: Aarrg I can't believe I'm dead..."** suivi d'un retour à la ligne
- Mage : **"[NAME]: By the four gods, I passed away..."** suivi d'un retour à la ligne

Exemple:

```
$paladin = new Paladin("Jean-Luc");
$mage = new Mage("Robert");
$paladin->attack();
$mage->attack();
/* affiche
Jean-Luc: I'll engrave my name in the 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...
*/
```

Etape 7

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_07.php`
Restrictions: Aucune

Dans cet exercice vous réutiliserez les classes de l'exercice précédent, copiez les donc dans votre répertoire de rendu.
Nous avons maintenant des personnages qui peuvent être magiciens ou guerrier et qui peuvent attaquer, c'est bien

mais ils ne peuvent toujours pas bouger !
C'est assez embêtant.
Afin d'ajouter ce comportement à nos classes, nous allons créer une interface **"iMove"** qui contiendra les méthodes

suivantes: **"moveRight"**, **"moveLeft"**, **"moveUp"** et **"moveDown"**.
Vous implémenterez ensuite cette interface à la classe **"Character"**.
Vos méthodes devront respectivement afficher le message suivant:

- moveRight -> **"[NAME]: moves right."** suivi d'un retour à la ligne
- moveLeft -> **"[NAME]: moves left."** suivi d'un retour à la ligne
- moveUp -> **"[NAME]: moves up."** suivi d'un retour à la ligne
- moveDown -> **"[NAME]: moves down."** suivi d'un retour à la ligne

Exemple:

```
$paladin = new Paladin("Jean-Luc");
$paladin->moveRight();
$paladin->moveLeft();
$paladin->moveDown();
$paladin->moveUp();
/* affiche
Jean-Luc: I'll engrave my name in the 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...
*/
```

Etape 8

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_08.php`
Restrictions: Aucune

Fini la paraplégie ! Nos personnages peuvent maintenant bouger, cependant, non-contents de pouvoir se mouvoir, nos personnages toujours dans leur élan d'égo en veulent toujours plus !
Notre ami Paladin refuse d'être comparé à un minable et frêle petit Mage, en effet, ce dernier ayant une démarche très prononcée et virile se démarque totalement du Mage qui lui se déplace en toute délicatesse !
Afin de satisfaire ce rustre de Paladin, vous implémenterez des surcharges pour les méthodes de **"iMove"** dont hérite **"Paladin"**.
Ainsi, vos méthodes move afficheront les messages suivants en accord avec la classe qui les surcharge:

- Paladin:
  - moveRight -> **"[NAME]: moves right like a bad boy."** suivi d'un retour à la ligne
  - moveLeft -> **"[NAME]: moves left like a bad boy."** suivi d'un retour à la ligne
  - moveUp -> **"[NAME]: moves up like a bad boy."** suivi d'un retour à la ligne
  - moveDown -> **"[NAME]: moves down like a bad boy."** suivi d'un retour à la ligne
- Mage:
  - moveRight -> **"[NAME]: moves right with grace."** suivi d'un retour à la ligne
  - moveLeft -> **"[NAME]: moves left with grace."** suivi d'un retour à la ligne
  - moveUp -> **"[NAME]: moves up with grace."** suivi d'un retour à la ligne
  - moveDown -> **"[NAME]: moves down with grace."** suivi d'un retour à la ligne

Exemple:

```
$paladin = new Paladin("Jean-Luc");
$paladin->moveRight();
$paladin->moveLeft();
$paladin->moveUp();
$paladin->moveDown();
$mage = new Mage("Robert");
$mage->moveRight();
$mage->moveLeft();
$mage->moveUp();
$mage->moveDown();
/* affiche
Jean-Luc: I'll engrave my name in the 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 with grace.
Robert: moves left with grace.
Robert: moves up with grace.
Robert: moves down with grace.
Robert: By the four gods, I passed away...
Jean-Luc: Aarrg I can't believe I'm dead...
*/
```

Etape 9

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_09.php`
Restrictions: Aucune

Maintenant nos personnages peuvent parler, marcher et attaquer de façon totalement personnalisée.
Cependant ils ne peuvent toujours pas dégainer leur arme ! C'est bien joli de pouvoir attaquer mais avec l'arme dans son fourreau cela risque d'être assez difficile...
Vous serez d'accord pour dire que peu importe que notre personnage soit un Paladin ou un Mage, ce dernier dégainera son arme de la même façon.
C'est pourquoi vous ferez en sorte que la classe **"Character"** implémente la méthode **"takeWeapon"** afin que **"Paladin"** et **"Mage"** en hérite, cependant vous devrez également faire en sorte que la méthode **"takeWeapon"** ne puisse pas être surchargée par **"Paladin"** et **"Mage"**.
Cette méthode devra afficher le texte suivant lorsqu'elle sera appelée: **"[NAME]: take out hisweapon."** suivi d'un retour à la ligne.

Exemple:

```
$perso = new Mage("Jean-Luc");
$perso->takeWeapon();
/* affiche
Jean-Luc: May the gods be with me.
Jean-Luc: take out his weapon.
Jean-Luc: By the four gods, I passed away...
*/
```

Etape 10

2 points

Nom de rendu: `Piscine_PHP_Jour_09/ex_10.php`
Restrictions: Vous devrez utiliser la fonction **"spl_autoload_register"**

Créez un fichier qui, lorsqu'il sera inclus, permettra d'utiliser n'importe quelle classe définie dans un fichier qui lui est propre. Ces fichiers de définitions de classe seront nommés comme tel :**"nom_de_la_classe.class.php"**.

Exemple:

```
/*
S'il existe une classe "Pangolin", elle sera definie dans le fichier "Pangolin.class.php". On
    devra pouvoir instancier cette classe rien qu'en incluant votre fichier.
*/
```