W1 – Piscine PHP

Jour 09

Responsables Pédagogiques

pedagowac@epitech.eu

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

Au cours de cette journée nous allons aborder la programmation orientée-objet, notion très importante dans le domaine du développement informatique.

Prévoyez quelques courses pour le « Jour 11 », à savoir, le nécessaire pour bien faire les choses. Nous vous communiquerons une liste d'individus à attraper dans le sujet de mercredi. Chaque personne qui sera correctement attachée sous cellophane rapportera des avantages non négligeables.

Sommaire

Exercice 01 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_01.php

Restrictions : Aucune

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

Exemple :

```
$foo = new MyLittleDisplay();

$foo->display();

// affiche "Hello"
```

Exercice 02 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_02.php

Restrictions : Aucune

Créez une classe "MyLittleAttribute" 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 MyLittleAttribute("Jean-Luc");

$foo->display();

// affiche "Jean-Luc"
```

Exercice 03 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_03.php

Restrictions : Aucune

Créez une classe "MyLittleAttributes" 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 MyLittleAttributes("Hello", "World");

$foo->display();

// affiche "Hello World"
```

Exercice 04 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_04.php

Restrictions : Aucune

Créez une classe "MyCalculator", composée de 3 attributs privés "a", "b" et "result", ainsi que de leurs getters et setters.

"result" contiendra le résultat de la dernière opération effectuée.

Ce résultat sera accessible grâce à la méthode "getLastResult".

La classe contiendra 4 méthodes publiques ne prenant aucun paramètre : "addition", "subtraction", "multiplication" et "divide". Elles renverront le résultat de leur opération respective.

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

Exemple :

```
$foo = new MyCalculator(30, 12);

echo $foo->addition()."\n";

echo $foo->subtraction()."\n";

echo $foo->multiplication()."\n";

echo $foo->divide()."\n";

echo $foo->getLastResult()."\n";

// affiche "42", "18", "360", "2.5" et "2.5"
```

Exercice 05 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_05.php

Restrictions : Aucune

Créez une classe "Personnage" composée des attributs protégés "name", "life", "agility", "strength", "wit", de la constante "CLASSE" et des getters qui y correspondent.

Ces attributs auront les valeurs suivantes:

- name = argument passé au constructeur
- life = 50
- agility = 2
- strength = 2
- wit = 2

Exemple :

```
$perso = new Personnage("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";

// affiche "Jean-Luc", "50", "2", "2", "2" et "Personnage"
```

Exercice 06 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_06.php

Restrictions : Aucune

Copiez le code de l'exercice précédent et créez une classe "Warrior" ainsi qu'une classe "Mage" héritant de "Personnage".

Vous modifierez les attributs de chacune de ces classes de la manière suivante :

- Warrior :
  - CLASSE = "Warrior"
  - life = 100
  - strength = 10
  - agility = 8
  - wit = 3
- Mage :
  - CLASSE = "Mage"
  - life = 70
  - strength = 3
  - agility = 10
  - wit = 10

Ces deux classes devront implémenter chacune une méthode "attack" qui ne prend aucun paramètre.

La méthode "attack" de la classe "Warrior" 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]: Fell 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 "Warrior" ou "Mage", un message soit écrit, ce dernier devra être de la forme suivante :

- Warrior : "[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 :

- Warrior : "[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 :

```
$warrior = new Warrior("Jean-Luc");

$mage = new Mage("Robert");

$warrior->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: Fell the power of my magic !", "Robert: By the four gods, I passed away..." et "Jean-Luc: Aarrg I can't believe I'm dead..."
```

Exercice 07 (2 pts)

Nom du 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 "iMovable" qui contiendra les méthodes suivantes: "moveRight", "moveLeft", "moveUp" et "moveDown".

Vous implémenterez ensuite cette interface à la classe "Personnage".

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 :

```
$warrior = new Warrior("Jean-Luc");

$warrior->moveRight();

$warrior->moveLeft();

$warrior->moveDown();

$warrior->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." et "Jean-Luc: Aarrg I can't believe I'm dead... "
```

Exercice 08 (2 pts)

Nom du rendu : Piscine_PHP_Jour_09/ex_08.php

Restrictions : Aucune

Finit la paraplégie ! Nos personnages peuvent maintenant bouger, cependant, non content de pouvoir se mouvoir, nos personnages toujours dans leur élan d'égo en veulent toujours plus ! Notre ami Warrior 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 Warrior, vous implémenterez des surcharges pour les méthodes de "iMovable" dont hérite "Personnage".

Ainsi, vos méthodes move afficheront les messages suivants en accord avec la classe qui les surcharge:

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

Exemple :

```
$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();
```

// 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 furtively.", "Robert: moves left furtively.", "Robert: moves up furtively.", "Robert: moves down furtively. ", "Robert: By the four gods, I passed away..." et "Jean-Luc: Aarrg I can't believe I'm dead... "

Exercice 09 (2 pts)

Nom du 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 Warrior ou un Mage, ce dernier dégainera son arme de la même façon. C'est pourquoi vous ferez en sorte que la classe "Personnage" implémente la méthode "unsheathe" afin que "Warrior" et "Mage" en hérite, cependant vous devrez également faire en sorte que la méthode "unsheathe" ne puisse pas être surchargée par "Warrior" et "Mage".

Cette méthode devra afficher le texte suivant lorsqu'elle sera appelée: "[NAME]: unsheathe his weapon." suivi d'un retour à la ligne.

Exemple :

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

$perso->unsheathe();

// affiche "Jean-Luc: May the gods be with me.", "Jean-Luc: unsheathe his weapon." et "Jean-Luc: By the four gods, I passed away..."
```

Exercice 10 (2 pts)

Nom du 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 définie dans le fichier "Pangolin.class.php". On devra pouvoir instancier cette classe rien qu'en incluant votre fichier.