TP PHP 2 - Objets, réflexion, Espace de noms, Tests unitaires

Téléchargez les fichiers annexes de l'espace Moodle et suivre les instructions du README :

- `IEmployee.php` : à placer dans `src`.
- `IManager.php` : à placer dans `src`.
- `ManagerTest.php` : à placer dans `tests`.

Exercice 1. Développez un script `Employee.php` qui définit la classe `Employee` comportant quatre propriétés :

- `id` : entier, accès privé.
- `name` : chaîne de caractères, accès public.
- `salary` : réel, accès protégé.
- `age` : entier, accès privé.

`Employee` doit implémenter l'interface `IEmployee` qui déclare les méthodes suivantes :

- Constructeur avec `id`, `name`, `salary` et `age` en paramètres.
- Accesseurs (*getters*) et mutateurs (*setters*).
- Méthode magique d'affichage des propriétés de l'objet.

Exercice 2. Développez un script `employee_display.php` qui crée un tableau de trois employés et les affiche à l'écran. On affichera également le salaire moyen de ces trois employés. Pensez à utiliser les fonctions `array_*`.
Exemple d'affichage :

```
employee: id=1 name=superman salary=1.27 age=80
employee: id=2 name=batman salary=1 age=73
employee: id=3 name=spiderman salary=0.82 age=50
mean salary = 1.03
```

Exercice 3. Développez un script `employee_raise.php` qui utilise le programme précédent et augmente la salaire de chacun des employés de 5%. Ce calcul s'effectuera avec une fonction `employee_raise` qui attend en paramètre un employé, vérifie que le paramètre est bien un objet et est de classe `Employee`, ou sinon lève une exception.
Exemple d'appel correct et incorrect :

```
Avant augmentation :
employee: id=1 name=superman salary=1.27 age=80
employee: id=2 name=batman salary=1 age=73
employee: id=3 name=spiderman salary=0.82 age=50
Après augmentation :
employee: id=1 name=superman salary=1.3335 age=80
employee: id=2 name=batman salary=1.05 age=73
employee: id=3 name=spiderman salary=0.861 age=50

Notice: Array to string conversion in .../employee_raise.php on line 13
Le paramètre n'est pas une instance de Employee
```

Exercice 4. Développez un script `employee_sort.php` qui trie un tableau d'employés en ordre de salaire croissant.
Exemple :

```
Key-preserving salary-increasing sorting
Array
(    [spider] => Employee Object
        (
            [id:Employee:private] => 3
            [name] => spiderman
            [salary:protected] => 0.82
            [age:Employee:private] => 50
        )
    [bat] => Employee Object
        (
            [id:Employee:private] => 2
            [name] => batman
            [salary:protected] => 1
            [age:Employee:private] => 73
        )
    [super] => Employee Object
        (
            [id:Employee:private] => 1
            [name] => superman
            [salary:protected] => 1.27
            [age:Employee:private] => 80
        )
)
```

Exercice 5. Le développement piloté par les tests (test-driven development) consiste à prédéfinir des tests unitaires sur le code source à développer. Ces tests sont ensuite exécutés de manière automatique sur chaque version produite à l'aide d'outils d'intégration continue (eg. `jenkins`). PHPUnit est le framework de référence en PHP pour développer des tests unitaires.
Lisez les instructions du **README** concernant l'organisation de votre répertoire, la mise en place des espaces de noms et des fonctionnalités d'autochargement avec l'outil `composer` (archive PHP fournie) et la mise en place et l'exécution de tests avec `PHPUnit` (fourni).
Le script `ManagerTest.php` contient un ensemble de tests unitaires destinés à être exécuter avec `PHPUnit` sur une classe `Manager` à implanter. Un manager est un employé qui a sous ses ordres un ensemble d'employés (ses subordonnés). Implantez d'abord la classe `Manager` (fichier `Manager.php`) qui hérite de `Employee` et implémente l'interface `IManager.php`. Les subordonnés d'un manager pourront être stockés sous forme d'un tableau contenant leurs identifiants.
Afin de tester votre classe :

(1) Exécutez les tests définis dans `ManagerTest.php` avec la commande :

```
./phpunit tests/ManagerTest
```

(2) Examinez le rapport d'exécution produit par `PHPUnit`.

(3) Si tout s'est bien passé, provoquez volontairement un échec du test en introduisant une erreur dans votre classe ou dans l'une des assertions de votre classe test.

Complétez ensuite la classe `ManagerTest.php` (marqueurs `TODO`) :

- En ajoutant un test pour la méthode `setAge` de `Manager`.
- En complétant la méthode de test `testAddEmployee`.

Re-testez votre classe avec `PHPUnit`.

Exercice 6. Développez une classe `Team` (fichier `Team.php`) qui permet de stocker des employés et des managers. Implantez la méthode magique d'affichage d'une équipe. Pour une équipe comportant un manager, on affichera le nom des employés qu'il a sous ses ordres. Exemple d'affichage d'équipe :

```
employee: id=1 name=superman salary=1.27 age=80

employee: id=2 name=batman salary=1 age=73

employee: id=3 name=spiderman salary=0.82 age=50

employee: id=4 name=wonder woman salary=3.14 age=71
subordinates=[superman batman spiderman ]
```

Exercice 7. Développez le fichier `employee_reflex1.php` qui affiche les informations suivantes concernant un employé en utilisant les fonctions de réflexion (`get_object_vars`, ...) :

- nom de la classe.
- nom de la classe parente.
- nom des champs et valeurs.

```
**Classe :
Employee
**Classe parente :
Pas de classe parente
** Propriétés visibles ayant une valeur par défaut :
Array
(
    [name] => anonymous
)
** Propriétés publiques :
Array
(
    [name] => euler
)
** Toutes les propriétés :
Array
(
    [id] => 0
    [name] => euler
    [salary] => 2.718
    [age] => 305
)
```

Exercice 8. Développez le fichier `employee_reflex2.php` qui affiche les mêmes informations que l'exercice précédent mais en utilisant l'API Reflexion.

```
Employee

Array
(
    [0] => ReflectionProperty Object
        (
            [name] => id
            [class] => Employee
        )

    [1] => ReflectionProperty Object
        (
            [name] => name
            [class] => Employee
        )
...
        )
)
Array
(
    [0] => ReflectionMethod Object
        (
            [name] => __construct
            [class] => Employee
        )

    [1] => ReflectionMethod Object
        (
            [name] => getId
            [class] => Employee
        )
        ...
)
```