Module 6 – PHP orienté objet

Pour tester rapidement du code PHP : *http://sandbox.onlinephpfunctions.com/*

Exercice 1 : pour s'échauffer…

Le but de ce premier exercice est de vous familiariser avec la syntaxe de l'orienté objet en PHP. Vous pouvez le réaliser sur le site Sandbox plutôt que d'utiliser un fichier situé sur le serveur.

Étape 1

Définissez une classe `Personnage` dont chacun des objets sera décrit par un nom et un nombre de points de vie (il s'agira de deux attributs privés). Ajoutez également une méthode `subitBlessure($n)` simulant une blessure : le nombre de points de vie sera diminué de $n (mais sans jamais passer en-dessous de 0).

Ajoutez un constructeur à la classe et créez un personnage appelé Homer et possédant 20 points de vie dans la variable `$homer`.

Ajoutez une méthode « magique » pour afficher un personnage sous le format suivant : nom (pv). Testez la en exécutant l'instruction

```
echo $homer;
```

qui devrait afficher Homer (20).

Infligez 5 points de dégâts à Homer (en utilisant la méthode subitBlessure) puis affichez à nouveau le contenu de `$homer`.

Étape 2

Définissez deux classes héritant de `Personnage` : `Guerrier` et `Magicien`. Lors de la construction d'un guerrier, on précisera, en plus de son nom et de ses points de vie, l'arme qu'il utilise (par exemple « une hache »).

Définissez une méthode magique pour l'affichage faisant appel à la méthode d'affichage de la classe `Personnage` et complétant la chaîne avec [G] pour un guerrier et [M] pour un magicien.

Ainsi, si Homer est un guerrier armé de « une hache » et possédant 20 points de vie et si Harry est un magicien avec 10 points de vie, les instructions

```
echo $homer;
echo $harry;
```

devront afficher respectivement Homer (20) [G] et Harry (10) [M].

Étape 3

Transformez la classe `Personnage` en une classe abstraite et ajoutez-lui la méthode abstraite `blesse($adversaire)` qui représentera le fait qu'un personnage en blesse un autre.

Implémentez la méthode `blesse` pour les guerriers et pour les magiciens, sachant que les premiers infligent 5 points de dégâts à leurs adversaires et que les seconds n'infligent que 3 points de dégâts. La méthode `blesse` devra tout d'abord infliger des dégâts à l'adversaire (en utilisant la méthode `subitBlessure`) puis afficher un message du style « Homer (20) [G] a blessé Harry (5) [M] ».

Pour tester le tout, exécutez les instructions suivantes,

```
$homer = new Guerrier('Homer', 20, 'une hache');
$harry = new Magicien('Harry', 10);
$homer->blesse($harry);
$harry->blesse($homer);
```

qui devraient afficher

```
Homer (20) [G] a blessé Harry (5) [M]
Harry (5) [M] a blessé Homer (17) [G]
```

Étape 4

Les guerriers possèdent un nombre maximal de points de vie égal à 25, alors que les magiciens ont au plus 20 points de vie. Définissez ces deux valeurs sous la forme de constantes de classe appelées `MAXPV`.

Ajoutez dans la classe Guerrier une méthode `afficheEtat()` qui produit l'affichage d'une ligne similaire à

```
Homer : 20/25 pv
```

en faisant référence à la constante de classe définie plus haut.

Pour rendre le nom et le nombre de points de vie accessibles dans la classe `Guerrier`, vous pouvez soit définir un « getteur » normal dans la classe `Personnage` soit définir une fonction magique `__get` dans `Guerrier`. Dans les deux cas, on pourra accéder à ces attributs uniquement en lecture.

Étape 5

On pourrait définir une fonction `afficheEtat()` similaire dans la classe `Magicien`, mais son code serait quasiment identique à celle de la classe `Guerrier` vu qu'on fait appel à une constante de classe.

Pour faire d'une pierre deux coups, déplacez plutôt la définition de la méthode `afficheEtat()` au niveau de la classe `Personnage`.

Cela devrait générer une erreur car aucune constante `MAXPV` n'est définie dans la classe `Personnage`. Comment résoudre ce problème ? Suffit-il d'ajouter une constante `MAXPV = 50` dans la classe `Personnage` ?

Testez votre code grâce aux lignes suivantes,

```
$homer = new Guerrier("Homer", 20, "une hache");
$harry = new Magicien("Harry", 10);
$homer->afficheEtat();
$harry->afficheEtat();
$homer->blesse($harry);
$harry->afficheEtat();
$homer->blesse($homer);
$homer->afficheEtat();
```

qui devraient produire ces sorties.

```
Homer : 20/25 pv
Harry : 10/20 pv
Homer (20) [G] a blessé Harry (5) [M].
Harry : 5/20 pv
Homer (15) [G] a blessé Homer (15) [G].
Homer : 15/25 pv
```

Étape 6

Les méthodes `blesse($adversaire)` des classes `Guerrier` et `Magicien` sont plutôt similaires : la seule différence concerne le nombre de dégâts infligés.

Placez ce nombre dans des constantes de classe et remplacez les deux méthodes `blesse` par une définition située dans la classe `Personnage` (qui peut donc redevenir une classe concrète désormais).

Exercice 2 : le programme des cours

Le but de cet exercice est de générer différents affichages à partir d'informations modélisées en utilisant une classe, de tirer avantage de la fonction `__autoload` et de protéger son code PHP.

Étape 1

Dans un premier temps, utilisez la sandbox pour créer une classe `Cours` en suivant les instructions suivantes.

Pour chaque cours, on retiendra un nom (de cours), le nombre d'heures et un booléen indiquant s'il s'agit d'un cours de spécialité (informatique) ou pas. Ces trois attributs seront privés.

Prévoyez un constructeur compatible avec le code suivant, qui est censé remplir un tableau $programme avec divers cours. Pour rappel, la syntaxe `$programme[]` permet d'ajouter un nouvel élément « à la fin » du tableau.

```
$programme[] = new Cours ("Business Intelligence", 50, false);
$programme[] = new Cours ("Environnements de développement de
logiciel", 65, true);
$programme[] = new Cours ("Programmation mobile", 50, true);
```

```
$programme[] = new Cours ("Programmation web orientée objet", 45,
true);
$programme[] = new Cours ("Développement d'applications web", 25,
true);
$programme[] = new Cours ("BDD avancées", 55, true);
$programme[] = new Cours ("Langues", 50, false);
$programme[] = new Cours ("Stage", 320, true);
$programme[] = new Cours ("TFE", 25, true);
```

Étape 2

Ajoutez à la classe `Cours` une méthode `sortieSimple()` renvoyant une chaîne de caractères correspondant aux exemples suivants (le premier concerne un cours de spécialité et le second, un cours hors spécialité).

```
BDD avancées pendant 55 heures (info)
Langues pendant 50 heures
```

Ajoutez également une méthode `ratio()` renvoyant le pourcentage horaire correspondant à un cours. Ainsi, si les cours couvrent un total de 500 heures, cette méthode devrait renvoyer 10 (pour 10%) dans le cas d'un cours de 50 heures.

Pour pouvoir calculer ce pourcentage, vous devrez mémoriser le nombre d'heures total. Pour ce faire, utilisez une variable de classe qui sera mise à jour chaque fois qu'un nouveau cours sera construit.

Testez toutes ces méthodes sur la Sandbox.

Étape 3 : premières sorties

Créez maintenant un fichier `cours.php` destiné à produire une page HTML. Celle-ci devra afficher un titre « Vos cours » en balise h1 suivi d'une liste (balises ul et li) présentant les cours un par un (la sortie pour chacun des cours correspond à la phrase produite par la méthode `sortieSimple`).

Quelques consignes :

- La définition de la classe `Cours` se trouvera dans un fichier séparé appelé `Cours.class.php`.
- Dans `cours.php`, vous demanderez l'incorporation de ce fichier via `require_once`. Juste après, vous construirez le tableau `$programme` en reprenant le code de l'étape 1.
- Utilisez un « `foreach` » pour parcourir tout le tableau et afficher la liste.

Étape 4 : protéger son code php

Le fichier « `Cours.class.php` » se trouve dans le répertoire `public`, qui est accessible par n'importe qui. Cela veut dire que le code qu'il contient pourrait être lu par tout le monde.

Pour plus de sécurité, on place généralement ce genre de fichiers (contenant uniquement du code) sur une partie privée du serveur, dans un endroit auquel le processeur PHP peut accéder mais qui est interdit au reste du monde.

Sur le serveur, créez un répertoire « php » au même niveau que le répertoire « public ». Le contenu de ce répertoire « php » sera donc accessible aux programmes tournant sur le serveur (comme le processeur PHP) mais pas au monde extérieur.

Modifiez la ligne « require_once » pour qu'elle dirige vers le nouvel emplacement de `Cours.class.php` (utilisez une adresse relative !).

Étape 5

Supprimez la ligne require_once !

Plutôt que de demander au préprocesseur d'aller chercher la définition de la classe Cours, vous allez lui indiquer où il pourra aller trouver (automatiquement) toutes les définitions de classe nécessaire.

Pour ce faire, définissez une fonction `__autoload` qui lui indique d'aller récupérer le fichier de définition de classe dans le répertoire php que vous venez de créer. On supposera que toute classe portant le nom XXX sera définie dans un fichier appelé `XXX.class.php` placé dans le répertoire php.

Étape 6 : des sorties améliorées

Dans la définition de la classe Cours, ajoutez deux constantes appelées CSS_CLASSINFO et CSS_CLASSNOM et possédant pour valeur (respectivement) « coursSpec » et « nomCours ».

Il s'agira du nom des classes CSS utilisées pour écrire les cours de spécialité et les noms des cours. Partout ailleurs, que ce soit dans la définition de ces classes ou dans leur utilisation, vous devrez faire référence à ces constantes plutôt que d'utiliser les mots « coursSpec » et « nomCours » directement. Votre code doit donc rester correct même si on change les valeurs de ces constantes (pour éviter un conflit futur avec d'autres classes par exemple).

Dans le fichier `cours.php`, ajoutez un script PHP pour produire les définitions CSS de ces styles. N'oubliez pas de faire appel aux constantes de classes ! Le style associé aux cours de spécialité utilisera une couleur de texte « blue ». Le style associé aux noms des cours utilisera des caractères gras.

Dans la définition de la classe `Cours`, ajoutez une méthode `sortieHTML` qui produira le texte correspondant à un cours. Le contenu textuel sera le même que celui produit par `sortieSimple`, mais il sera suivi du pourcentage horaire représenté par le cours et comportera des annotations HTML portant sur les styles :

- les noms des cours seront écrits dans le style correspondant (utilisez la balise HTML span pour cibler les noms de cours et leur attribuer un style) ;
- dans le cas des cours de spécialité, l'ensemble du texte sera présenté avec le style correspondant (là aussi, utilisez la balise).

Finalement, modifiez la boucle d'affichage pour qu'elle utilise la nouvelle méthode.

Voici le résultat que vous devriez obtenir.

Vos cours

- **Business Intelligence** pendant 50 heures pour 7.3%
- **Environnements de développement de logiciel** pendant 65 heures (info) pour 9.49%
- **Programmation mobile** pendant 50 heures (info) pour 7.3%
- **Programmation web orientée objet** pendant 45 heures (info) pour 6.57%
- **Développement d'applications web** pendant 25 heures (info) pour 3.65%
- **BDD avancées** pendant 55 heures (info) pour 8.03%
- **Langues** pendant 50 heures pour 7.3%
- **Stage** pendant 320 heures (info) pour 46.72%
- **TFE** pendant 25 heures (info) pour 3.65%

Étape 7

Ajoutez une nouvelle méthode à la classe `Cours`. Cette méthode, appelée `divRatio`, produira le code HTML nécessaire à l'affichage d'une barre grisée horizontale dont une partie (correspondant au pourcentage horaire du cours en question) sera noircie.

Voici par exemple l'affichage correspondant au code HTML attendu dans le cas de Stage.

La méthode `divRatio` prendra un argument optionnel indiquant la longueur en pixels du cadre grisé. Par défaut, celui-ci vaudra 80.

Le code HTML à produire sera composé de deux imbriqués.

- Le extérieur correspondant au cadre grisé aura une hauteur fixée à 12 pixels et une longueur dépendant de l'argument donnée. Il sera affiché en position « relative » et avec un fond de couleur lightgrey.
- Le intérieur, lui, aura une hauteur de 12 pixels et une longueur qui dépend du pourcentage horaire correspondant au cours (utilisez la méthode demandée plus haut). Il sera affiché en position « absolute » avec un déplacement indiqué par « left : 0 » et « top : 0 ».

Votre but final est d'obtenir l'affichage suivant.

Nom	Nb heures	Proportion
Business Intelligence	50	
Environnements de développement de logiciel	65	
Programmation mobile	50	
Programmation web orientée objet	45	
Développement d'applications web	25	
BDD avancées	55	
Langues	50	
Stage	320	
TFE	25	

Vous devrez donc utiliser un tableau HTML et sans doute écrire une nouvelle méthode qui produira le code HTML correspondant à chacun des cours (sous la forme d'une ligne du tableau). Utilisez à nouveau le style correspondant aux cours de spécialité.

Le contenu des dernières cases de chaque ligne sera produit grâce à la méthode divRatio construite plus haut.

Pour plus de clarté, vous ajouterez également un tooltip sur chacun des graphiques (chacune des barres grisées). Le tooltip devra afficher le pourcentage correspondant au cours au format numérique (par exemple 3,65% pour Développement web).