TP 5

Développement d'une plateforme de travail collaboratif (en PHP OO)

I. POO : Rappel - Introduction

1. Définition d'un objet

L'exemple le plus pertinent quand on fait un cours sur la POO est d'utiliser l'exemple du personnage dans un jeu de combat. Ainsi, imaginons que nous ayons un objet **Personnage** dans notre application. Un personnage a des **caractéristiques** :

- Une force ;
- Une localisation ;
- Une certaine expérience ;
- Et enfin des dégâts.

Toutes ses caractéristiques correspondent à des **valeurs**. Comme vous le savez sûrement, les valeurs sont stockées dans des **variables**.

Mis à part ces caractéristiques, un personnage a aussi des **capacités**. Il peut :

- Frapper un autre personnage ;
- Gagner de l'expérience ;
- Se déplacer.

Ces capacités correspondent à des **fonctions**. Comme pour les variables, ce sont des fonctions un peu spéciales.
Chaque objet est défini selon des caractéristiques et un plan bien précis. En POO, ces informations sont contenues dans ce qu'on appelle des **classes**.

2. Définition d'une classe

Prenons l'exemple le plus simple du monde : les gâteaux et leur moule. Le moule, il est unique. Il peut produire une quantité infinie de gâteaux. Dans ces cas-là, les gâteaux sont les **objets** et le moule est la **classe** : le moule va définir la forme du gâteau. La classe contient donc le plan de fabrication d'un objet et on peut s'en servir autant qu'on veut afin d'obtenir une infinité d'objets.
Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité.

3. Définition d'une instance

Une instance, c'est tout simplement le résultat d'une instanciation. Une instanciation, c'est le fait d'instancier une classe. Instancier une classe, c'est se servir d'une classe afin qu'elle nous crée un objet. En gros, une instance c'est un objet.

Classe Personnage

```
$force
$localisation
$experience
$degats

Fonction frapper()
Fonction gangerExperience()
Fonction deplacer()
```

Exemple : création d'une classe Personnage
Prenons l'exemple du personnage dans un jeu de combat. Nous avons donc un objet **Personnage** dans notre application. Un personnage a des **caractéristiques** :

- Une force ;
- Une localisation ;
- Une certaine expérience ;
- Et enfin des dégâts.

Toutes ses caractéristiques correspondent à des **valeurs**.

Mis à part ces caractéristiques, un personnage a aussi des **capacités**. Il peut :

- Frapper un autre personnage ;
- Gagner de l'expérience ;
- Se déplacer.

On va créer une classe **Personnage**. Celle-ci doit contenir la liste des variables et des fonctions citées plus haut : c'est la base de tout objet **Personnage**. Chaque instance de cette classe possèdera ainsi toutes ces variables et fonctions. Voici donc cette classe :
Vous voyez donc les variables et fonctions stockées dans la classe **Personnage**. Sachez qu'en réalité, on ne les appelle pas comme ça : il s'agit d'**attributs** (ou **propriétés**) et de **méthodes**. Un attribut désigne une variable de cette classe et une méthode désigne une fonction de celle-ci.

Ainsi, tout objet **Personnage** aura ces attributs et méthodes. On pourra modifier la valeur de ces attributs et appeler (invoquer) ces méthodes sur l'objet afin de modifier ses caractéristiques ou son comportement.

4. Le principe d'encapsulation

L'un des gros avantages de la POO est que l'on peut **masquer** le code à l'utilisateur (l'utilisateur est ici le développeur qui se servira de la classe, pas celui qui exploite le programme fini). Le concepteur de la classe a englobé dans celle-ci un code qui peut être assez complexe et il est donc inutile voire **dangereux** de laisser l'utilisateur manipuler ces objets sans aucune restriction. Ainsi, il est important d'interdire à l'utilisateur de modifier directement les attributs d'un objet.

Prenons l'exemple d'un avion où sont disponibles des centaines de boutons. Chacun de ces boutons constituent des actions que l'on peut effectuer sur l'avion. C'est l'**interface** de l'avion. Le pilote se moque de quoi est composé l'avion : son rôle est de le piloter. Pour cela, il va se servir des boutons afin de manipuler les composants de l'avion. Le pilote ne doit pas se charger de modifier manuellement ces composants : il pourrait faire de grosses bêtises.

Le principe est exactement le même pour la POO : l'utilisateur de la classe doit se contenter d'invoquer les méthodes en ignorant les attributs. Pour instaurer une telle contrainte, on dit que les attributs sont **privés**.

II. Créer une classe

1. Syntaxe de base

Voici la syntaxe de base de toute classe en PHP:

```
1 <?php
2     class Personnage // Présence du mot-clé class suivi du nom de la classe.
3     {
4         // Déclaration des attributs et méthodes ici.
5     }
6 ?>
```

On vient de créer le **moule**
La déclaration d'attributs dans une classe se fait en écrivant le nom de l'attribut à créer, précédé de sa **visibilité**.

2. La visibilité d'un attribut ou d'une méthode

La visibilité d'un attribut ou d'une méthode indique à partir d'où on peut avoir accès à telle méthode ou tel attribut. Nous allons voir ici 3 types de visibilité : public, private (privé) et protected (protégé)

- Accès public : c'est le plus simple. Si un attribut ou une méthode est public, alors on pourra avoir accès à cet attribut ou cette méthode depuis n'importe où, autant depuis l'intérieur de l'objet (dans les méthodes qu'on a créées, on aura accès aux éléments public), mais aussi depuis l'extérieur.
- Accès privé : il impose quelques restrictions. On aura accès aux attributs et méthodes **que** depuis **l'intérieur de la classe**, c'est-à-dire que seul le code voulant accéder à un attribut privé ou une méthode privée écrit(e) **à l'intérieur** de la classe fonctionnera. Les héritier n'ont pas accès à un élément privé.
- Accès protégé : C'est comme l'accès privé sauf qu'un tel attribut (ou méthode) est accessible aussi par les classes qui héritent de la classe mère.

Ceci est l'un des **principes de l'encapsulation**.

3. Création d'attributs

Pour déclarer des attributs, on va donc les écrire entre les accolades, les uns à la suite des autres, en faisant précéder leurs noms du mot-clé private :

```
1 <?php
2     class Personnage
3     {
4         private $_force; // La force du personnage.
5         private $_localisation; // Sa localisation.
6         private $_experience; // Son expérience.
7         private $_degats; // Ses dégâts.
8     }
9 ?>
```

Chaque attribut est précédé d'un **underscore**. Ceci est une notation qu'il est préférable de respecter (il s'agit de la notation PEAR). Cette notation dit que chaque nom d'élément privé (attributs ou méthodes) doit être précédé d'un **underscore**.

On peut initialiser les attributs lors de la. Exemple :

```
1 <?php
2     class Personnage
```

```
3     {
4         private $_force = 50; // La force du personnage, par défaut à 50.
5         private $_localisation = 'Sousse'; // Sa localisation, par défaut à Sousse.
6         private $_experience = 1; // Son expérience, par défaut à 1.
7         private $_degats = 0; // Ses dégâts, par défaut à 0.
8     }
9 ?>
```

La valeur par défaut doit être une expression. Par conséquent, leur valeur ne peut pas être issue d'un appel à une fonction , d'une opération ou d'une concaténation .
Exemples : Expliquer pourquoi les initialisations suivantes sont fausses :

```
private $_attribut = 1 + 1 ;
private $_attribut = 'Mon ' . 'super ' . 'attribut' ;
private $_attribut = intval ('007') ;
```

4. Création de méthodes

Il suffit de faire précéder le mot-clé function de la visibilité de la méthode. Les types de visibilité des méthodes sont les mêmes que les attributs.

```
1 <?php
2     class Personnage
3     {
4         private $_force; // La force du personnage.
5         private $_localisation; // Sa localisation.
6         private $_experience; // Son expérience.
7         private $_degats; // Ses dégâts.
8
9         public function deplacer()
10        {
11
12        }
13
14        public function frapper()
15        {
16
17        }
18
19        public function gagnerExperience()
20        {
21
22        }
23    }
24 ?>
```

Dans notre cas toutes les méthodes sont de type public.

III. QCM :

Comment appelle-t-on une variable dans une classe ? Et une fonction ?
- Une variable est une méthode; une fonction est un attribut
- Une variable est un attribut ; une fonction est une méthode

Quelle est la différence entre classe et objet ?
- Une classe est un ensemble de fonctions et variables ; un objet est une instance de cette classe
- Un objet est un ensemble de fonctions et variables ; une classe est une instance de cet objet

Par quel mot-clé déclare-t-on une classe ?
- classe
- class
- object
- objet

IV. Utiliser la classe

Après avoir défini la classe on peut l'utiliser pour créer plusieurs objets afin de s'en servir.

1. Créer un objet

Pour créer un nouvel objet, il faut faire précéder le nom de la classe à instancier du mot-clé new, comme ceci :

```
1 <?php
2     $perso = new Personnage();
3 ?>
```

Ainsi, $perso sera un objet de type Personnage. On dit qu'on a instancié la classe Personnage ou qu'on a créé une instance de la classe Personnage.

2. Appeler les méthodes de l'objet

Pour appeler une méthode d'un objet, il va falloir utiliser l'opérateur ->.Il s'utilise de la manière suivante : À gauche de cet opérateur, on place **l'objet** que l'on veut utiliser, à droite de l'opérateur on spécifie le nom de la méthode qu'on veut invoquer (appeler)

```
1  <?php
 2 // Nous créons une classe "Personnage"
 3 class Personnage
 4 {
 5   private $_force;
 6   private $_localisation;
 7   private $_experience;
 8   private $_degats;
 9
10   // Nous déclarons une méthode dont le seul but est d'afficher un texte
11   public function parler()
12   {
13     echo 'Je suis un personnage !';
14   }
15 }
16
17 $perso = new Personnage();
18 $perso->parler();
```

La ligne 18 signifie « va chercher l'objet $perso, et invoque (appelle) la méthode parler() sur cet objet ».

3. Accéder à un attribut de la classe

L'opérateur -> permet aussi d'atteindre un attribut. On va essayer de créer un personnage et d'augmenter son expérience :

```
 1  <?php
 2  class Personnage
 3  {
 4    private $_force;
 5    private $_experience;
 6    private $_degats;
 7  }
 8
 9  $perso = new Personnage();
10  $perso->_experience = $perso->_experience + 1; // erreur fatale ! Pourquoi ??
```

On est hors de la classe et on a essayé d'accéder à un attribut privé. Ceci est interdit, donc PHP lève une erreur. Dans notre exemple (qui essaye d'augmenter de 1 l'expérience du personnage), il faudra demander à la classe d'augmenter l'expérience. Pour cela, nous allons écrire une méthode experience() :

```
1 <?php
2 class Personnage
3 {
4   private $_experience;
5
6   public function gagnerExperience()
7   {
8     // Cette méthode doit ajouter 1 à l'expérience du personnage
9     $this->_experience ++;
10  }
11 }
12
13 $perso = new Personnage();
14 $perso->gagnerExperience();
```

$this est appelé 'pseudo variable' et permet d'accéder aux propriété d'une classe alors qu'on n'a même pas d'objet instancié !

V. Développement de la plateforme de travail collaboratif

Lancer MovAmp, ouvrir phpMyAdmin et créer une nouvelle bdd appelée « travail_collaboratif ». Créer ensuite une nouvelle table « utilisateur » ayant les attribut du point « V.2 » vu en cours.
Dans le dossier « mnt/var/www » de MovAmp, créer un nouveau dossier appelé « projet » et y créer l'arborescence suivante :

- /projet/classes/

- /projet/js/
- /projet/inc/
- /projet/images/
- /projet/admin/

ATTENTION :
Toutes les classes seront créée dans le dossier « /classes/ ».
Tous les fichiers de traitement relatifs à l'admin seront dans le dossier « /admin/ »

Pour chaque partie on donne le squelette qu'il faudra compléter.

1. La base de données

On demande de développer la classe « Mysql » qui permet de se connecter à la bdd. Cette classe contient :

- des attributs privés :
  - serveur (host) : définir « locahost » comme valeur par défaut
  - login
  - mot de passe
  - nom de la bdd
  - identifiant de connexion : la valeur retourné par mysql_connect()
- Les méthodes publiques suivantes :
  - set_serveur($s), set_login($s), set_mdp($s), set_bdd($s) : initialise la variable respective.
  - connexion() : permet de se connecter à la bdd et d'affecter l'identifiant de la connexion à l'attribut privé.
  - get_cnx() : renvoi la valeur de l'identifiant de connexion.
  - deconnexion() : permet de fermer la connexion à la bdd
  - requete($q) : permet d'exécuter une requête $q et de renvoyer son résultat.

Dans chaque méthode, en cas d'erreur afficher un message et interrompre le traitement en utilisant la fonction exit()
Ci-dessous le squelette du fichier qu'il faut compléter :

```
<?php
class Mysql
{
	private $_serveur = "……";
	private $_login;
	private $_mdp;
	private $_bdd;
	private $_cnx;

	public function set_serveur($s)
	{
		$this->_serveur = $s;
	}

	public function set_login($s)
	{
		$this->…… = $s;
	}

	public function …… ($s)
	{
		$this->_mdp = $s;
	}

	public function …… ($s)
	{
		$this->……= $s;
	}

	public function ……_cnx()
	{
		return $this->_cnx;
	}

	public function connexion()
	{
		$this->_cnx = mysql_connect($this->_serveur, $this->……, ……);
		if (……)
			exit("Erreur de connexion bdd : " . mysql_error());
		if (!mysql_select_db(……))
			exit("Erreur : bdd inexistante : " . ……());

	}

	public function requete($q)
	{
		$res = ……($q);
		if (!$res)
```

```
                exit("<pre>Erreur dans la requete [$q] : " . ……..() . "</pre>");
        return ……..;
    }
}
?>
```

Enregistrer le fichier (dans « /classes/Mysql.php » Créer et compléter le fichier « /inc/connexion.php » dont le code est le suivant :

```
<?php
    include(".../Mysql.php");

    $bdd = new ...;
    ...->set_serveur("localhost");
    ...->set_login("...");
    ...-> ...(" ...");
    ...->set_bdd("...");
    ...->connexion();
?>
```

2. Gestion des utilisateurs

a. La classe

Créer la classe « Utilisateur » dont le contenu est :

- Les attributs privés :
  - id : identifiant unique, auto-increment
  - nom : obligatoire, vc(50), not null
  - prenom : obligatoire, vc(50), not null
  - d_naissance : non obligatoire, date
  - mail : taille entre 5 et 50 caractères, unique, obligatoire, vc(50), not null
  - mdp : taille entre 4 et 15 caractères, la valeur par défaut est ‘1234’
- Les méthodes publiques suivantes :
  - Les différent setters (set_nom($s) …) en prenant en compte les remarques relatives à chaque attribut
  - enregistrer($bdd) : insérer un nouvel enregistrement
  - supprimer($bdd) : supprimer un enregistrement dont l’identifiant est défini par set_id()

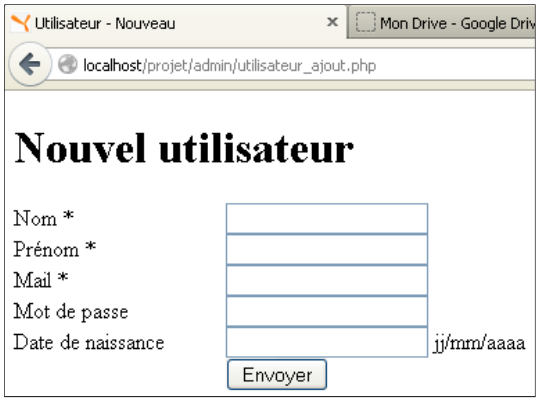
On commence par développer l’interface d’ajout, donc pour le moment ces méthodes sont suffisantes. Ci-dessous le code du fichier /classes/Utilisateur.php qu’il faut compléter :

```
<?php
class ...
{
    private $_id;
    private $_nom;
    private ...
    private ...
    private ...
    private ...

    public function set_nom($s)
    {
        if (strlen($s) == 0)
            exit("Utilisateur : le nom est obligatoire");
        $this->... = $s;
    }

    public function set_prenom($s)
    {
        ...
        ...
    }

    public function set_mail($s)
    {
        if (...)
            exit("Utilisateur : le ... est obligatoire");
        ...
    }

    public function set_mdp($s)
    {
        if (...)
            $s = "1234";

        if (...)
            exit("Utilisateur : le mdp doit être compris entre 4 et 15 caractères");
        ...
    }
```

```
        public function set_d_naissance($s) // format d'entrés : jj/mm/aaaa
        {
                $this->_d_naissance = $s;
        }

        public function set_id($x)
        {
                $this->_id = $x;
        }

        public function enregistrer(Mysql $bdd)
        {
                $q = "INSERT INTO utilisateur (id, nom, ...)
                          VALUES (null, '$this->_nom', ... )";
                return $bdd->...;
        }


        function supprimer()
        {
                ...
                ...
        }
}
?>
```

b. Ajout d'un nouvel utilisateur

Créer le formulaire d'ajout (ci-contre) qui sera appelé « /admin/utilisateur_ajout.php ». Les noms des champs (attributs 'id' et 'name' des input) seront dans l'ordre : nom, prenom, mail, mdp, d_naissance, b_ajouter



Pour la balise FORM, définir la valeur de l'attribut ACTION : « utilisateur_ajout_action.php »

```
<form id="form1" name="form1" method="post"
action="utilisateur_ajout_action.php">
```

Créer et compléter le fichier « /admin/utilisateur_ajout_action.php » dont le code est le suivant :

```
<?php
        include(".../connexion.php");
        include("...");
        $u = new Utilisateur();
        $u->set_nom($_REQUEST['nom']);
        $u->...
        $u->...
        $u->...
        $u->set_mdp($_REQUEST['mdp']);

        if (...->...($bdd))
                print "Ajout utilisateur ok.";
?>
```

Tester le formulaire de saisie en ouvrant l'url « http://localhost/projet/admin/utilisateur_ajout.php »
Vérifier depuis phpmyadmin que les données sont bien enregistrées. En cas d'erreurs PHP **corriger** le code source et tester de nouveau.

Si l'ajout fonctionne bien, on va à l'étape suivante qui consiste à développer la page qui affiche la liste des utilisateurs.

c. Liste des utilisateurs existants

Modifier le fichier « /classes/Utilisateur » en **ajoutant** les méthodes suivantes. ATTENTION : effectuer une copie de sauvegarde de votre fichier !

- Les différent getters (get_nom() …)
- modifier($bdd) : mettre à jour un enregistrement existant
- get_un($bdd, $id) : envoi un objet rempli avec les données de l'utilisateur dont l'id est passé en paramètre.
- get_liste($bdd, $order_by='id', $order_type='ASC') : renvoi le contenu de la table sous forme de tableau d'objets. Si le 2ème paramètre est spécifié, le tri se fait sur le nom de la colonne défini, sinon le tri se fait sur l'id. Le tri se fait selon le 3ème paramètre : ASCendant (par défaut) ou DESCendant.

…

```
public function get_id() { return $this->_id; }
```

```
public function get_nom() { return $this->... }
public function get_...() { return ... }
public function ... { ... }
public function ...
public function ...

public function get_un(..., ...)
{
	$q = "SELECT ... WHERE ...";
	$res = ...->requete(...);
	$row = mysql_fetch_array($res);

	$u = new ...;
	$u->set_d_naissance($row['d_naissance']);
	$u->set_id($row['...']);
	$u->set_mail(...);
	$u->set_... (...);
	$u->...
	$u->...
	... $u;
}

public function get_liste(..., ..., ...)
{
	$q = "SELECT * FROM utilisateur ORDER BY $order_by $order_type";
	$res = $bdd->requete($q);
	while($row = mysql_fetch_array($res))
	{
		$u = new Utilisateur();
		$u->set_d_naissance($row['d_naissance']);
		$u->set_id($row['id']);
		$u->set_mail($row['mail']);
		$u->set_mdp($row['mdp']);
		$u->set_nom($row['nom']);
		$u->set_prenom($row['prenom']);

		$a_user[] = $u;
	}
	return $a_user;
}
```

3. Gestion des utilisateurs – Les écrans.

Pour la création des utilisateurs, créer et développer les pages php suivantes :

- « utilisateur_ajout.php » : il s'agit d'un simple formulaire pour saisir les données d'un nouvel utilisateur. Attention, on n'aura pas de champ « id » dans cet écran.
- « utilisateur_ajout_action.php » : cette page sera appelée quand on valide la page précédente. Elle récupère les données depuis le formulaire et utilise les classes développés ci-dessus afin d'insérer un nouvel enregistrement dans la bdd.

Utiliser le même principe pour développer les 2 pages nécessaires à la modification d'un utilisateur dont l'identifiant est passé en paramètre (par la méthode get())

Développer une page « utilisateur_supp_action.php » qui permet de supprimer un utilisateur dont l'identifiant est passé en paramètre (par la méthode get())

Développer une page « utilisateur_liste.php » qui affiche la liste des utilisateurs comme vu en cours, avec des liens nécessaires vers les pages créés ci-dessus.

4. Perfectionnement

On remarque des problèmes d'affichage et de stockage avec les dates.

Référence :
http://fr.scribd.com/doc/3774696/PHP-POO-fr-avec-exemples
http://www.siteduzero.com/tutoriel-3-147180-programmez-en-oriente-objet-en-php.html