ÉNONCE – CLASSES ET NAMESPACE EN PHP

Objectif :

L'objectif est de s'initier à la programmation orientée-objet de **PHP**. Les différents exercices mettront en évidence les syntaxes particulières du langage, notamment sur l'écriture du constructeur. Une fois les premiers mécanismes assimilés, nous pourrons mettre en évidence la problématique des noms de classes et sa résolution par les espaces de nom. La dernière partie concernera l'utilisation de **composer**, et son utilisation de l'**autoloader**.

Ressources indispensables :

- *https://www.php.net/docs.php*
- *https://getcomposer.org/download/*

Nous allons implémenter une classe Personnage. Le but étant d'obtenir un module logiciel, qui comme dans un jeu de rôle, permettra de faire évoluer des « personnages ». Voici le Diagramme de Classe UML qui représente notre base de départ.

Etape 1 : La classe

Un dépôt du projet est à récupérer sur Github. Vous pouvez récupérer le squelette du code avec la commande **git clone** https://github.com/laurentgiustignano/R3.01-ClassesEtNamespace-Enonce dans votre répertoire de travail. Ou bien depuis **PhpStorm**, en choisissant : **Get from Version Control** avec la même URL.

Le projet comporte 4 éléments :

- **index.php** qui correspond à la page d'accueil de notre projet.
- **css** et **js** qui sont des dossiers contenant Bootstrap.
- **header.php** qui contient l'entête pour l'affichage de notre page et la fonction **debug()** permettant d'afficher distinctement des portions à débugger

Avec un clic-droit sur votre dossier de projet, choisissez **new-> Php Class**. Dans la fenêtre qui apparaît, saisir le nom de la classe « **Personnage** » et laissez les autres champs par défaut. **PhpStorm** va créer un nouveau fichier dans votre projet, portant le nom de votre classe avec l'extension **.php**.
Remarque : En PHP, le nom du fichier doit correspondre au nom de la classe.

- Dans le fichier **Personnage.php**, ajouter l'attribut **$pointDeVie** dans la classe. Pour l'instant nous garderons la visibilité « **public** », spécifierons le type de l'attribut à **int** :

```php
public int $pointDeVie;
```

- Dans le fichier index.php, nous allons créer un objet **$mario** de type **Personnage**. La création d'une instance d'un objet a de multiple syntaxe, nous utiliserons celle qui appelle le constructeur de la classe (même s'il n'existe pas encore dans notre projet) : **$nomDeLObjet = new NomDeLaClasse() ;**
  Nous remarquons qu'une erreur s'affiche sur notre page. PHP n'arrive pas à localiser la classe

Personnage. Pour cela, nous allons utiliser au début du script l'instruction **require_once** avec le nom du fichier entre guillemet.

- Toujours dans le fichier **index.php**, remplacer le texte de l'appel de **debug()** par l'objet **$mario**. Celui-ci apparaitra à l'écran mais vide. En effet, pour l'instant son seul attribut n'a pas de valeur, donc il n'est pas présent. Si dans le programme, on modifie la valeur de **$pointDeVie** pour l'objet **$mario**, l'affichage devra confirmer l'affectation. Mais nous observons le danger d'avoir cet attribut public dans la classe **Personnage**.
- Dans la classe **Personnage**, changez la visibilité de l'attribut **$pointDeVie** en la passant à **private**. Notez l'erreur qu'il en retourne. Nous n'avons plus le droit de changer de valeur cet attribut à l'extérieur de la classe (et c'est bien ainsi). Il faut commenter/supprimer la ligne provoquant l'erreur qui est spécifié dans le message d'erreur.
- Pour spécifier la valeur commune à tous les objets de type **Personnage**, nous avons besoin d'un constructeur. En PHP, nous ne pouvons utiliser qu'un seul constructeur, et il s'appelle **__construct()**. Pour l'ajouter facilement, nous faisons un clic-droit dans **class Personnage**, et on choisit **Generate…**, puis **Constructor…** On clique sur la classe pour désélectionner le nom de la propriété existante pour avoir un constructeur par défaut, puis **Ok**. Dedans, nous allons donner **40** comme valeur par défaut à **$pointDeVie**. Pour désigner l'objet pour lequel s'exécute le constructeur, il faut utiliser **$this** et la flèche (**->**) pour désigner l'attribut ou la méthode.

Etape 2 –

Pour compléter la classe, nous allons ajouter un nouvel attribut **$maxDeVie** et une nouvelle fonctionnalité **jeSuis()** qui affichera dans un premier temps le texte « Je suis un Personnage ». L'attribut **$maxDeVie** permet de connaître le nombre de **$pointDeVie** maximum qu'un Personnage peut avoir quand il est en parfaire santé.

- Faire les ajouts nécessaires dans **Personnage.php**
- Vérifier le comportement pour l'objet **$mario** en utilisant sa méthode **jeSuis()**.

Pour respecter le principe d'encapsulation, les données membres ont été déclaré privés. Pour continuer le TP, nous allons avoir besoin d'implémenter des getters et setters, afin de faciliter la compréhension. Encore une fois, **PhpStorm** va nous aider.

- Faire un clic-droit dans la classe Personnage, Choisissez **Generate…**, puis **getters and setters…**, sélectionner les deux attributs **$pointDeVie** et **$maxDeVie**, puis **Ok**.

*Remarque : Vous noterez qu'à la fin de la définition des fonctions, PhpStorm rajoute une syntaxe « **: type**» qui correspond au type de retour de la méthode. Même si le typage est intuitif dans PHP, il reste une bonne pratique de typer les éléments pour confirmer des valeurs de retour et aider les relecteurs de code !*

Étape 3 –

Pour limiter les problématiques des noms de classes similaires, il faut s'habituer à travailler dans des espaces de noms. Nous allons devoir placer la classe Personnage dans ce nouvel espace, puis ensuite spécifier dans le fichier index.php le nom complet de la classe avec son namespace devant. Néanmoins, il est possible dans un script PHP, d'avertir l'interpréteur qu'on utilise une classe d'un espace de nom, cela permet de simplifier l'écriture.

- Faire un clic-droit sur la classe **Personnage** et choisir **Refactor**, puis **Move Class…** . Dans le menu, spécifier votre **namespace**, par exemple **App**, puis **Ok**. **PhpStorm** va placer ajouter la syntaxe **namespace App ;** au début du fichier **Personnage.php** et placer ce fichier dans un dossier **App**. De plus, au début du fichier **index.php** qui utilisait la classe **Personnage**, la syntaxe **App/Personnage ;** informe que l'usage du terme **Personnage** fait référence à la classe

Personnage du namespace **App**. Donc, tout va (presque) bien. Comme un **Personnage.php** est contenu dans le répertoire **App**, il faut mettre à jour le **require_once**.

Maintenant le projet est mieux organisé, mais il est tout de même embêtant d'avoir à répéter l'utilisation d'une classe entre **use** et **require_once**. Le langage PHP permet de charger les classes utiles pour un script par le biais d'un **autoloader**. Cela signifie que si l'organisation des fichiers dans des dossiers et namespace est suffisamment claire, on peut charger les fichiers automatiquement. Il faut respecter le **PSR-4** (PHP Standard Recommendation) qui indique que l'espace de nom doit être calqué sur les chemins d'accès du système de fichier. Ainsi, on peut remplacer les **require** de toutes les classes utilisées par le **require** d'un seul fichier **autoload**.
Il est possible de créer un autoloader « maison » mais il est habituel d'utiliser **Composer** dans les projets PHP, qui contient son autoloader. **Composer** est un gestionnaire de librairies pour PHP. Nous allons voir comment utiliser **Composer** et son autoloader.
Remarque : Si Composer a été installé en global sur votre machine, vous pouvez sauter cette première phase d'installation. Pour vérifier, tapez la commande composer dans un terminal.

- Pour initialiser **Composer** pour votre projet, il faut commencer par l'installer. Vous trouverez les commandes PHP à saisir. Il faut ouvrir un terminal au niveau de votre projet. La première commande effectue le téléchargement. Si une erreur apparaît à ce niveau, vérifier dans le fichier de configuration php.ini que l'extension **openssl** est bien décommanté. La suivante effectue une vérification de la valeur de hachage, la troisième installe **Composer** et crée le fichier **composer.phar**. Et la dernière supprime le fichier d'installation **composer-setup.php**.
- Nous pouvons initialiser **Composer** pour le projet en tapant la commande **php composer.phar init**. Les premières questions concerneront votre nom et celui de votre projet, vous pouvez laisser vides, les questions sur la Description, la Stabilité, la License. Puis pour les Dépendencies, répondez **n** pour non. Un descriptif Json va apparaître et vous pouvez accepter la génération du fichier **composer.json**.
- Adaptez le fichier **composer.json** avec vos informations avec l'image ci-dessous comme référence.

```json
{
    "name": "laurent/init",
    "authors": [
        {
            "name": "Laurent Giustignano",
            "email": "laurent.giustignano@u-paris.fr"
        }
    ],
    "autoload": {
        "psr-4": {
            "App\\": "App"
        }
    },
    "require": {}
}
```

Remarque : Le terme init dans la clé « name » correspond au nom de mon projet. Le vôtre peut ne pas porter le même nom. Les informations de la clé psr-4 correspondent, en premier à votre namespace, et en second au nom du répertoire de votre projet qui représentera la racine de votre namespace. Notez

enfin que les doubles anti-slashs « \\ » sont obligatoire pour ne pas échapper le guillemet de la chaine de caractères.

- Enfin, pour générer l'autoloader de composer, il faut lancer la commande **php composer.phar dump-autoload**. À présent, on peut utiliser l'**autoload** de composer en ajoutant au début du fichier **index.php** l'instruction suivante :

Bon courage…