Projet Zoo

Etape 1: Installation

Architecture

Créez un nouveau dossier et dedans, créer un fichier composer.json avec le contenu suivant.

```
{
    "name": "votre-nom/zoo",
    "autoload": {
        "psr-4": {
            "App\\":"src"
        }
    }
}
```

Pour la suite du projet, le dossier aura cette structure:

```
.
├── app.php
├── composer.json
└── src
```

Dans le fichier `app.php` il faut ajouter le contenu suivant:

```
<?php
require __DIR__ . '/vendor/autoload.php';


// Here comes your code.
```

Pour tester son travail, il faut faire

```
php app.php
```

à cette étape du projet, ce fichier lance une erreur. C'est normal.

Téléchargement et installation de Composer

Pour commencer, il faut installer composer, un gestionnaire de dépendances en PHP. Toutes les informations d'installation se trouvent sur getcomposer.org/doc/00-intro.md

Initialisation du GIT

On va définir le dossier comme étant un projet GIT.

Pour cela, on va effectuer un :

```
git init
```

la totalité du projet doit être commité avec la commande `git commit -m "mon message de commit"` .

A chaque fin d'étapes, penser à commiter avec un message **explicite**.

Mise en place de composer

On va initialiser le projet en faisant un `composer install` dans le répertoire du projet.

Cela aura pour conséquence de créer un dossier `vendor` qui **ne doit pas être commit**.

A partir de maintenant, tester ses instances se fera dans le fichier app.php, et les classes devront être déclarées dans le namespace 'App' dans le dossier src.

> Tip: on n'oublie pas de commit

Etape 2: Des animaux en pagaille

On va commencer par créer une classe abstraite `App\Animal` .

Cette classe aura: - un attribut name, privé, qui sera définie au constructeur, et un accesseur en lecture seulement. - méthode protégée abstraite appelé `getNoise` qui ne prendra pas de paramètres, et qui renverra une chaine de caractères. :warning: il faut typer ses valeurs de retour. - Une méthode publique appelée `noise` qui fera un appel à la méthode précédement créée et renverra son résultat.

On va également créer des classes qui vont hériter de la classe `App\Animal` :

```
App\Animals\Fish
App\Animals\BubbleFish
App\Animals\CatFish
App\Animals\ClownFish
App\Animals\Whale
```

```
App\Animals\Zebra
App\Animals\Elephant
App\Animals\Parrot
App\Animals\Dove
```

et implémenter les différentes méthodes de la classe qu'ils héritent. :warning: on veut pousser l'héritage plus loin avec certains animaux en particulier, sans faire de classes abstraites.

Lexique des cris d'animaux :

- L' éléphant : `tooooooout`
- Le zèbre : `hiiiii`
- Le poisson : `bloubloublou`
- La balaine : `splash`
- Le perroquet : `coco`
- La colombe : `Rou Rouuu`

Test du code

Pour tester le code effectué, on va modifier le fichier `app.php` :

- on va instancier un tableau `$animals` qui contiendra 5 poissons, 3 boisson bulle, 2 poissons chat, 1 poisson clown, 2 êlephants, 1 Zèbre, 10 perroquets et 2 colombes.
- Une fois cela fait, il faudra, avec une simple boucle sur le tableau `$animals` afficher le nom de l'animal et son cri.

> Tip: on n'oublie pas de commit

Etape 3: Des Enclos adaptés

On veut maintenant savoir si les enclos sont adaptés aux besoin. Notre Zoo aura 3 enclos : - Aquarium - Aviary - Fence

La règle est simple : Si un animal peut voler, il faut le mettre dans la volière, s'il ne sait que marcher il faut le mettre dans un enclos, et s'il ne peut que nager, il faut le mettre dans l' aquarium.

:warning: ne mettez pas les Zèbres dans l'aquarium.

On va donc créer des interfaces: `App\Interfaces\CanSwim` , `App\Interfaces\CanFly` , `App\Interfaces\CanWalk` .

Ces interfaces ne déclareront pas de méthodes supplémentaires. Ajouter une ou plusieurs de ces interfaces aux différentes classes.

Il faut créer une classe `Enclosure` qui va contenir un tableau `animals` avec une méthode `addAnimal` qui prendra un Animal en paramètre. et une méthode **to String** qui nous permettra de voir (et d'entendre crier) les animaux !

On va enfin créer une classe `App\Zoo` qui aura 3 attributs privés statiques de type Enclosure : aquarium, aviary et fence. Par défaut, ces propriétés sont à `null` .

Cette classe aura 3 méthodes statiques pour récupérer l'aquarium, la volière et la cage.

Créer la méthode statique publique qui s'appellera également `addAnimal` et qui va nous permettre de mettre un Animal dans l'enclos (qui devra être instancié pour l'occasion et seulement à ce moment) qui lui correspond en fonction de l'interface qu'ils instancient.

Pour terminer sur cette partie, on va créer une méthode static `visitTheZoo` , qui va afficher le contenu des différentes cages.

Test du code

On va modifier l'affichage précédent du app.php pour que cette fois, chaque animaux soit mis dans la classe Zoo.

> on n'oublie pas de commit

Enfin

envoyez votre code sur GitHub dans un repository privé.