Sujet 2 : Comprendre un objet simple	
Buts :	découvrir et différencier les notions de classe et d'instance, utiliser une documentation, découvrir l'implémentation d'une classe, compléter l'implémentation d'une classe

Avant de commencer

Créez un répertoire `POO`, dans lequel vous créerez un sous-répertoire `TP2`. Vous y stockerez l'ensemble des développements produits tout au long de ce TP. Votre enseignant devra y retrouver la trace de tout le travail effectué, ne remplacez donc pas vos solutions par d'autres, privilégiez, en cas de nécessité, une mise en commentaire. Pour une question de lisibilité, veillez à reporter les numéros des questions dans votre code. De la même façon les noms des auteurs devront y apparaître en haut de chaque script. Les réponses aux questions *d'ordre théorique* (qui ne sont pas du code) seront enregistrées dans un fichier TP2.txt.

Création du dépôt Git

Votre travail sera impérativement déposé sur un dépôt Git. Si vous avez opté pour un dépôt commun à tous les TPs, n'oubliez pas de le compléter systématiquement en créant un répertoire par sujet de TP. Dans le cas contraire initialisez un nouveau dépôt en suivant les directives d'initialisation de dépôt décrites ci-dessous.

Consignes

Vous veillerez à respecter scrupuleusement les consignes suivantes :

- Vous effectuerez un commit après chaque exercice.
- A la fin de chaque séance, vous effectuerez un commit. Si ce commit contient du code incomplet ou ne fonctionnant pas, mentionnez-le dans le message de commit. Vous pousserez ensuite votre travail vers le dépôt distant.
- Ce dépôt sera utilisé par votre enseignant(e) de TP pour évaluer votre travail. Assurez-vous donc régulièrement que tous les fichiers que vous souhaitez lui rendre sont bien présents dans le dépôt.
- Le dépôt lui-même sera évalué : soignez l'écriture de vos messages.

Initialisation du dépôt

- Ouvrez un terminal et utilisez la commande *cd* pour vous placer dans le répertoire $TP2$
- Initialisez le dépôt *Git* du projet.
- Si nécessaire, éditez le fichier *.gitignore*
- Effectuez un premier commit
- Connectez-vous à l'application *Gitlab* et créez un dépôt nommé $POO - TP2$

- Assignez le rôle *Reporter* à votre enseignant(e) de TP.
- Poussez le dépôt local vers le dépôt distant.

Liens utiles

- Aide-mémoire Git : *https://iut-info.univ-reims.fr/users/nourrit/git_aide_memoire.pdf*
- Gitlab : *https://iut-info.univ-reims.fr/gitlab*

Les bons usages

D'après les recommandations PSR-2, un script PHP débute par la balise <?*php* mais ne se termine par aucune balise. Vous éviterez de fermer vos scripts avec la balise ? >.

Afin que le typage de vos fonctions et de vos méthodes soit réellement pris en compte nous procéderons dorénavant à une typage strict. Vous rajouterez pour ceci la directive suivante en haut de vos scripts :

```
declare(strict_types = 1);
```

Exercice 1. La classe Country : manipulation d'un objet existant

Dans cet exercice nous nous intéresserons à la manipulation d'un objet simple. Pour ceci nous manipulerons la classe `Country` dont la documentation vous est fournie. Vous êtes invités à la consulter tout au long de cet exercice. Le fichier contenant l'implémentation (code) de la classe est disponible en ligne à l'adresse :
https://iut-info.univ-reims.fr/users/romaniuk/restricted/S2BasesPOO/ressources/Country.txt
Récupérez le ficher `Country.txt` et enregistrez-le dans le répertoire TP2 sous le nom `Country.php`. Vous n'ouvrirez pas ce fichier. L'objectif ici est d'utiliser une classe et non d'étudier et de comprendre son implémentation.

Documentation de la classe *Country*

Un `Country` est caractérisé ici par son nom *name*, sa *surface* et sa *population*. Ces caractéristiques sont propres à chaque pays.

Constructeur(s)

`__construct ([string $name="Unknown" [,float $surface=0 [,int $population=0)]]])`
Méthode d'instance qui permet de créer une instance (un objet) de la classe `Country`, c'est-à-dire de créer une variable dont le type est `Country`. Ce constructeur est appelé de manière implicite avec 0, 1, 2 ou 3 paramètres.

Destructeur

`__destruct()`
Méthode d'instance qui permet d'effectuer les tâches nécessaires juste avant la destruction des attributs dans une instance de la classe `Country`. Cette méthode est appelée de manière implicite (automatique) lors de la destruction d'un objet.

Accesseurs et modificateurs

`getPopulation () : int`
Méthode d'instance qui permet de récupérer le nombre d'habitants d'un pays particulier.

`getSurface () : float`
Méthode d'instance qui permet de récupérer la superficie d'un pays particulier.

`getName () : string`
Méthode d'instance qui permet de récupérer le nom d'un pays particulier.

`setPopulation ( int $population ) : void`
Méthode d'instance qui permet de remplacer le nombre d'habitants d'un pays par la valeur $population.

`setSurface ( float $surface ) : void`
Méthode d'instance qui permet de remplacer la superficie d'un pays par la valeur $surface.

`setName ( string $name ) : void`
Méthode d'instance qui permet de remplacer le nom d'un pays par la valeur $nom.

Divers

`print ( ) : void`
Méthode d'instance qui permet d'afficher les caractéristiques d'un pays particulier.

`getDensity() : float`
Méthode d'instance qui permet de calculer puis de retourner la densité d'un pays particulier. Si les valeurs de la surface ou de la densité ne sont pas strictement positives cette méthode retourne 0.

`isEqual (Country $autrePays) : bool`
Méthode d'instance qui permet de vérifier si les caractéristiques de l'objet courant (le pays par rapport auquel est appelée la méthode) sont le même que celles de $autrePays. Cette méthode retourne *true* si c'est le cas, $false$ sinon.

Avant de commencer à manipuler des objets de type `Country` voici quelques règles de base :

Construction :
Pour faire appel à un constructeur la syntaxe à adopter est la suivante :

```
$inconnu = new Country ; #appel au constructeur sans paramètre
$chine = new Country ("Chine" ) ; #appel au constructeur avec un seul paramètre
```

Destruction :
On ne fait pas appel au destructeur, il est appelé automatiquement.

Appel d'une méthode :
Pour faire appel à une méthode quelconque la syntaxe générique à adopter est la suivante :

```
[... = ] $objet -> nomMethode( [ valeurs des paramètres ] ) ;
```

Question 1.
Créez un script `testCountry.php` dans le répertoire TP2. Afin de pouvoir tester les différentes fonctionnalités de la classe `Country`, à l'intérieur de votre script vous inclurez l'instruction suivante :

```
require_once "Country.php";
```

Instanciez l'objet *$france*, dont les caractéristiques sont {France - 641185.0 km^2 - 66600000 habitants}. Vous utiliserez pour ceci le constructeur avec 3 paramètres.

En utilisant la méthode `print` affichez l'objet *$france*. Exécutez votre script. Que constatez-vous ?

Les caractéristiques de l'objet *$france* sont : le nom `name`, la `surface` et la `population`. Ces caractéristiques sont les **attributs d'instance** de la classe `Country`. Quels sont, selon vous, leurs types respectifs ? Quelles sont leurs valeurs. En quoi a consisté le travail du constructeur préalablement appelé ?

Question 2.
Utilisez le constructeur en lui passant en paramètre uniquement une chaîne de caractères pour instancier l'objet *$italie* dont les caractéristiques sont : {Italie - 301336.0 km^2 - 60626442 habitants}. Affichez l'objet *$italie*. Quelles sont les valeurs des attributs d'instance de l'objet *$italie* ? En quoi a consisté le travail du constructeur appelé ? L'ensemble des attributs d'instance de cet objet ont-ils les valeurs initialement désirées ? Mettez à jour votre objet, si ceci est nécessaire, pour qu'il contienne les caractéristiques initialement désirées. On pourra pour ceci faire appel aux modificateurs. Aurait-il été plus judicieux d'appeler le constructeur avec un nombre de paramètres différent pour créer l'instance *$italie* ? Si oui, lequel ?

Question 3.
Créez deux nouvelles instances de la classe `Country`, une construite à partir d'aucun paramètre et une autre construite à partir de deux paramètres. Comment sont initialisés les attributs d'instance de ces deux objets ?

Question 4.
On souhaite dupliquer l'instance *$france* dans l'objet *$copieFrance*. Afin d'éviter d'attribuer un second nom de variable à un même emplacement mémoire en utilisant un simple =, la création de $copieFrance nécessitera l'appel du constructeur. Proposez une solution à ce problème. Affichez le contenu des deux instances. Comparez *informatiquement* le contenu des objets *$france* et *$copieFrance*.

Question 5.
Testez l'ensemble des méthodes d'instance de la classe `Country` commençant par les mots `get` et `set` en les appelant de manière judicieuse sur l'ensemble des instances de la classe `Country` créées auparavant. Vous pourrez par exemple démontrer grâce à ces appels que les objets *france* et *copieFrance* sont deux objets biens distincts.

Question 6.
Dans le script `testCountry.php`, définissez la fonction `copie` qui prend en paramètre un `Country` et qui retourne sa copie. Testez votre fonction.

Question 7.
Dans le script `testCountry.php`, définissez la fonction `printCountry` qui prend en paramètre un `Country` et qui affiche ses caractéristiques. Cette fonction devra produire le même rendu que la méthode `print` mais sans y faire appel. Testez votre fonction.

Question 8.
Quelle est la différence entre une fonction et une méthode d'instance selon vous ?

Exercice 2. Bilan sur la manipulation d'un objet (répondre dans un fichier texte)

Question 9.

En vous plaçant du côté d'un utilisateur, définissez avec des mots simples les notions suivantes :

- classe,
- instance,
- objet,
- méthode d'instance,
- constructeur.

Si ceci est possible, vous veillerez à donner des définitions qui permettent de différencier toutes ces notions. Chacune de ces définitions ne nécessite pas plus de 20 mots.

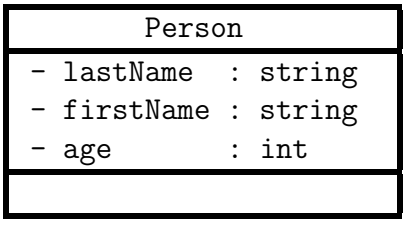
Exercice 3 : la classe Person

L'objectif de cet exercice est d'implémenter (définir) des classes simples tout en respectant le principe d'encapsulation des données, c'est à dire en cachant à l'utilisateur le contenu de la classe (partie interne) et en lui proposant des spécifications (partie externe, présentée aux utilisateurs de la classe) intuitive permettant de facilement manipuler les objets de cette classe.

Dans le projet *TP2*, créez un script nommé `Person.php`. Il contiendra la définition de la classe `Person`. Éditez ce fichier et complétez la au fur et à mesure.

En parallèle créez un script `TestPerson.php`, n'oubliez pas de commencer votre script par l'instruction : `require_once "Person.php";` . Vous compléterez cette classe et l'exécuterez à chaque ajout d'une nouvelle méthode d'instance dans la classe `Person`.

L'objectif de cet exercice est de proposer une implémentation simple de la classe `Person`. Une personne sera caractérisée par son nom `lastName`, prénom `firstName` et son `age`. Ci-dessous vous trouverez le diagramme de classe correspondant :



Les attributs d'instance et constructeurs

On considère la définition incomplète de la classe `Person` suivante :

```
class Person
{
    private string $lastName;
    private string $firstName;
    private int $age;
}
```

Question 10.
Recopiez cette définition dans la classe `Person`. Quels sont les attributs d'instance de la classe `Person` ? Quels sont leurs noms et quels sont leurs types ?

A quoi correspond le mot `private` ? Quelle est la différence entre `private` et `public` ? Peut-on accéder aux attributs d'instance de la classe `Person` à partir du script `TestPerson.php` ?

Définissez simplement la notion d'encapsulation des données.

Question 11.
On considère la méthode d'instance de la classe `Person` suivante que vous allez ajouter dans la classe sans pour autant vous attarder sur sa définition :

```
public function print() : void
{
    echo "Nom : {$this->lastName}\n";
    echo "Prenom : {$this->firstName}\n";
    echo "Age : {$this->age}\n" ;
}
```

Essayez d'instancier un objet de type `Person` par le biais de l'instruction :

```
$inconnu = new Person ;
```

Si ceci fonctionne, affichez l'instance par le biais de la méthode `print`. Que constatez-vous ?

Utilisez maintenant l'instruction `var_dump ( $inconnu ) ;` pour inspecter le contenu de cette instance. Quelles sont les valeurs des ses attributs d'instance et pourquoi ?

Question 12.
Le constructeur est une méthode d'instance très particulière dans une classe. Dans quel contexte fait-on appel à un constructeur ? Quel opérateur associe-t-on à l'appel d'un constructeur ? Quel est l'objectif que doit remplir un constructeur ? Combien d'objets peut-on instancier grâce à un même constructeur ?

Un constructeur a-t-il un type de retour ? Quel est le nom que porte obligatoirement un constructeur ?

Question 13.
Dans la classe `Person`, ajoutez la définition du constructeur avec arguments suivante :

```
public function __construct (string $lastName, string $firstName, int $age)
{
    $this->lastName = $lastName;
    $this->firstName = $firstName;
    $this->age = $age;
}
```

Quels sont les paramètres de ce constructeur ? Que réalise ce constructeur ? A quoi correspond le mot clé `$this` ? Expliquez la signification de l'instruction `$this->lastName = $lastName;`.

L'instruction `$inconnu = new Person ;` est-elle toujours valide ? De combien de façons peut-on actuellement instancier un objet ? Expliquez pourquoi ? Instanciez ensuite un objet de type `Person` en faisant appel au constructeur avec arguments. Quels sont les valeurs des ses attributs d'instance ?

Dans le script `TestPerson.php` faites appel à ce constructeur.

Question 14.
On souhaiterait pouvoir instancier un objet de type `Person` de la manière suivante :

```
$jacques = new Person("Durand" , "Jacques" );
```

où "Durand" est la valeur que prendra l'attribut d'instance `$lastName` de l'instance `jacques`, "Jacques" est la valeur que prendra son attribut d'instance `$firstName` et 0 est la valeur que prendra par défaut son attribut d'instance `$age`.

En PHP il n'est pas possible de surcharger des méthodes, c'est-à-dire de définir dans une même classe ou un même script plusieurs fonctions qui ont le même nom. On se propose pour ceci de modifier le constructeur précédent comme suit :

```
public function __construct (string $lastName, string $firstName, int $age = 0)
{
    $this->lastName = $lastName;
    $this->firstName = $firstName;
    $this->age = $age;
}
```

Il s'agit ici d'une initialisation par défaut du paramètre $*age*. Lorsque à l'appel du constructeur ce dernier n'est pas renseigné il prendra pour valeur 0, la valeur passée en paramètre sinon. Vous remarquerez ici que l'initialisation par défaut se trouve en fin de liste de paramètres.

Vérifiez que vous pouvez actuellement instancier des objets de type `Person` de deux façons différentes en créant une instance de `Person` faisant appel à un constructeur avec deux paramètres.

Question 15.
On souhaiterait pouvoir instancier des objets de type `Person` de la manière suivante :

```
$inconnu = new Person;
     #constructeur qui initialise les attributs à "Doe", "John" et 0
$jules = new Person("Durand");
     #constructeur qui initialise les attributs à "Durand", "John" et 0
```

Modifiez le constructeur initial en conséquence et testez sa validité dans le script `TestPerson.php`.

Les accesseurs et modificateurs

Question 16.
On considère la méthode d'instance de la classe `Person` suivante :

```
public function getLastName() : string
{
    return $this->lastName;
}
```

Quel est le type de retour de la méthode d'instance `getLastName` ? A-t-elle des paramètres ?

Cette méthode s'appelle un accesseur sur l'attribut d'instance `$lastName`. Après avoir ajouté cette méthode d'instance dans votre classe, vérifiez de manière pertinente le bon fonctionnement de votre accesseur, ajoutez les lignes suivantes à la fin du script `TestPerson.php` :

```
$nomInconnu = $inconnu->getLastName();
echo "$nomInconnu\n";
```

L'accesseur `getLastName()` retourne la valeur de l'attribut d'instance `$lastName` de l'objet `Person`. Autorise-t-il sa modification ? Justifiez votre réponse.

Question 17.
Ajoutez les accesseurs aux deux autres attributs d'instance de la classe `Person`. Complétez le script `TestPerson.php` et testez l'ensemble.

Question 18.
Complétez le script `TestPerson.php` avec l'instanciation d'un objet `copie` correspondant à la copie de l'objet `jacques`. On veillera à éviter tout partage de la mémoire.

Question 19.
On considère la méthode d'instance de la classe `Person` suivante :

```
public function setAge(int $age) : void
{
    $this->age = $age;
}
```

Afin de vérifier de manière judicieuse le bon fonctionnement de la méthode d'instance `setAge` ajoutez les lignes suivantes à la fin du script `TestPerson.php` puis testez l'ensemble :

```
echo "{$jacques->getAge()}\n";
$jacques->setAge(26);
echo "{$jacques->getAge()}\n";
```

Question 20.
Ajoutez les modificateurs des deux autres attributs d'instance de la classe `Person`. Complétez le script `TestPerson.php` et testez l'ensemble.

Les méthodes (autres)

Question 21.

On souhaite pouvoir afficher les attributs d'instance d'une `Person` de la manière suivante :

```
echo "$jacques\n";
```

Ajoutez cette instruction à la fin du script `TestPerson.php`. Exécutez le et commentez le résultat obtenu.

On peut obtenir un affichage *correct* (celui de la méthode d'instance `print` par exemple) en définissant la méthode d'instance `__toString` dans la classe `Person`. Cette méthode retourne une chaîne de caractères contenant les informations de l'instance de la classe `Person`. Elle détermine en effet comment l'objet doit réagir lorsqu'il est traité comme une chaîne de caractères, erreur que vous avez constatée préalablement en faisant appel à `echo` sur votre instance. Cette méthode d'instance pourrait s'écrire ici comme suit :

```
public function __toString() : string
{
    $res = "Nom : {$this->lastName}\n";
    $res = $res."Prenom : {$this->firstName}\n";
    $res = $res."Age : {$this->age}" ;
    return $res;
}
```

Complétez votre classe avec la méthode d'instance `__toString`. Exécutez à nouveau votre script `TestPerson.php`.