Sujet 5 : L'héritage simple	
Buts :	comprendre les mécanismes de l'héritage simple, produire une documentation

Avant de commencer

Dans le répertoire `POO` créez un sous-répertoire `TP5`. Vous y stockerez l'ensemble des solutions développées 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.

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 $TP5$
- 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 $POO - TP5$
- 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 vérifié de manière stricte nous procéderons dorénavant à une typage strict. Vous ajouterez pour cela la directive suivante en tant que première instruction de vos scripts :

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

Vos classes doivent être clairement documentées. Pour générer une documentation associée à votre classe vous le ferez à partir d'un terminal. Vous vous placerez pour ceci au niveau du répertoire contenant les fichiers à documenter. Voici le ligne de commande permettant d'obtenir la documentation de la classe `MaClasse` :

```
phpdoc -t Documentation --visibility public -f MaClasse.php
```

L'objectif de ce sujet est de comprendre les mécanismes de l'héritage simple.

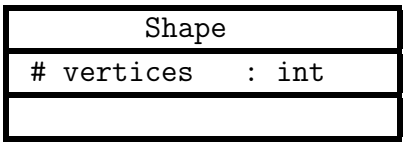
Avant de commencer

L'implémentation des classes `Shape`, `Rectangle` et `Square` est disponible en ligne à l'adresse : *https://iut-info.univ-reims.fr/users/romaniuk/restricted/S2BasesPOO/ressources/*
Télechargez les fichiers `Shape.txt`, `Rectangle.txt` et `Square.txt` et placez-les dans le répertoire `TP5`. Renommez respectivement les fichiers en `Shape.php`, `Rectangle.php` et `Square.php`. Créez un script `TestShapes.php` que vous compléterez progressivement pour tester les trois classes.

Partie 1 : Comprendre une implémentation dans le contexte de l'héritage

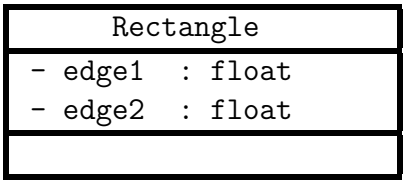
Dans cet exercice nous nous intéresserons à la manipulation d'objets simples dans le contexte de l'héritage. Afin de faciliter la compréhension des mécanismes de l'héritage nous étudierons les classes figure (`Shape`), `Rectangle` et carré (`Square`).

Pour rappel, `Shape` est une classe mère caractérisée par son nombre de sommets. Son diagramme de classe est le suivant :



Question 1. Instanciez l'objet `$figure1` de type `Shape` dont le nombre de sommets est 6. Affichez l'instance `$figure1`. Caractérisez ses attributs d'instance (nom, type et valeur). Quels sont leurs modes d'accès ? Quelle est la différence entre les modes `public`, `private` et `protected`.

La classe `Rectangle` hérite de la classe `Shape`. Elle est donc la classe fille (ou la classe dérivée) de la classe `Shape`. Cette classe est caractérisée, outre par le nombre de sommets, par la longueur de ses deux arrêtes. Son diagramme de classe est le suivant :



Question 2. En étudiant la déclaration de la classe `Rectangle`, identifiez la syntaxe permettant d'obtenir le lien d'héritage qui existe entre les classes `Shape` et `Rectangle` ?

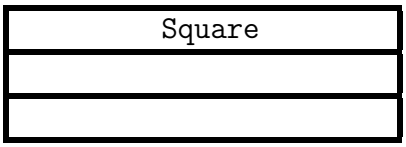
Question 3. Instanciez l'objet `$rectangle1` de type `Rectangle` dont les longueurs respectives des arrêtes sont 25 et 63.5. Affichez l'instance `$rectangle1`. Caractérisez ses attributs d'instance (nom, type et valeur). Que constatez vous ? L'instance `$rectangle1` est-elle une instance de la classe `Shape` ?

Étudiez l'implémentation du constructeur de la classe `Rectangle`. Expliquez à quoi correspond l'instruction : `parent::__construct(4);` . Que se passe-t-il lorsque cette instruction n'est pas présente dans le constructeur ? Expliquez pourquoi. L'ordre des instructions dans le constructeur a-t-il de l'importance ?

Question 4. Étudiez l'implémentation de la méthode `print` dans la classe `Rectangle`. A quoi correspond l'instruction : `parent::print();` . Que se passe-t-il lorsque cette instruction n'est pas présente dans cette méthode ? L'ordre des instructions dans cette méthode a-t-il de l'importance ?

Par quelle instruction pourrait-on remplacer l'instruction : `parent::print();` dans la méthode d'instance `print` de la classe `Rectangle` ? Expliquez pourquoi.

La classe `Square` hérite de la classe `Rectangle`. Cette classe n'a pas de caractéristiques supplémentaires par rapport à celles d'un `Rectangle`. Son diagramme de classe est le suivant :



Question 5. Instanciez l'objet `$carre1` de type `Square` dont la longueur des arrêtes est 14. Affichez l'instance `$carre1`. Caractérisez ses attributs d'instance (nom, type et valeur). Que constatez vous ? L'instance `$carre1` est-elle une instance de la classe `Shape` ? Est-elle un `Rectangle` ?

Étudiez l'implémentation du constructeur de la classe `Square`. A quoi correspond l'instruction `parent::__construct($a1,$a1);` . Est-il possible d'appeler le constructeur de la classe `Shape` dans ce constructeur pour construire l'objet `$this` ?

Question 6. Est-il possible de redéfinir la méthode `print` dans la classe `Square`, permettant ainsi de simplifier l'affichage proposé dans la classe `Rectangle` (affichage d'une unique longueur d'arête) sans intervenir sur l'implémentation des classes `Rectangle` et/ou `Shape` ? Justifiez votre réponse.

Question 7. Documentez les 3 classes.

Partie 2 : Autour des Logements

L'objectif de cet exercice est de modéliser le concept de `Lodgement` (`Apartment` ou `House`).
La classe `Lodgement` est une classe dont dérivent les classes `Apartment` et `House`.

La classe `Lodgement` est caractérisée par sa surface, son type (T2, Studio, F4, ...) ainsi que le prix du m^2. La classe `Apartment` contient en plus l'information sur l'étage auquel se trouve l'appartement et l'information sur la présence d'un ascenseur ou non. La classe `House` est caractérisée en plus par la surface du jardin.

La classe Lodgement

Un logement est caractérisé par sa `surface` en m^2, son `type`, ainsi que le prix du m^2 (`meterPrice`).
Voici le diagramme de classe correspondant :

Lodgement
- surface : float - type : string - meterPrice : float

Dans le répertoire `TP5`, créez une classe nommée `Lodgement`. Créez un script `TestLodgements.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Lodgement`.

Question 1. Définissez la classe `Lodgement` en y ajoutant la déclaration de ses trois attributs d'instance.

Question 2. On souhaite pouvoir créer un `Logement` de la manière suivante :

```
$logement1 = new Lodgement(46.7, "T2", 6805.);
```

Le premier paramètre de ce constructeur sert à initialiser la surface, le dernier le prix du m^2.
Définissez le constructeur correspondant.

Question 3. On souhaite pouvoir accéder et modifier le prix du m^2 du logement en dehors de la classe `Lodgement`. Définissez l'accesseur `getMeterPrice` et du modificateur `setMeterPrice` dans la classe `Lodgement`.

Question 4. On souhaite pouvoir afficher les caractéristiques d'un `Logement` en utilisant l'instruction suivante :

```
echo "$logement1\n";
```

L'affichage souhaité correspondant à l'instance `$logement1` est :

```
Surface : 46.7
Type : T2
Prix du m : 6805
```

Définissez les modalités d'affichage permettant d'obtenir le résultat souhaité.

Question 5. Dans la classe `Lodgement` est-il nécessaire de définir la méthode `__clone` afin de permettre un clonage propre de ses instances ? Justifiez votre réponse. Clonez une instance de la classe `Lodgement` et vérifiez votre réponse.

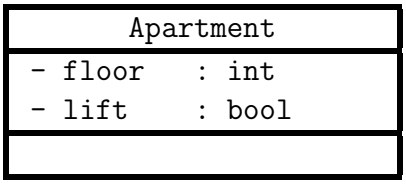
Question 6. On souhaite disposer du prix du `Logement` en utilisant la méthode `getPrice`. Cette méthode retourne le prix du logement en fonction de sa surface et de son prix au m^2. Définissez la méthode `getPrice`.

La classe Apartment

La classe `Apartment` hérite de la classe `Lodgement`. Cette classe a pour caractéristiques (outre celles de la classe `Lodgement`) :

- l'étage auquel se trouve l'appartement ;
- l'information sur la présence d'un ascenseur.

Voici le diagramme de classe correspondant :



Question 7. Dans le répertoire `TP5`, créez une classe nommée `Apartment`. Vous compléterez et exécuterez le script `TestLodgements.php` à chaque ajout de nouvelle méthode dans la classe `Apartment`.

Question 8. Complétez la définition de la classe `Apartment` en y ajoutant la déclaration de ses attributs d'instance.

Question 9. On souhaite disposer d'un constructeur avec arguments dans la classe `Apartment`. Ce constructeur prendra en paramètres toutes les informations nécessaires pour initialiser l'ensemble des caractéristiques de l'objet. Définissez ce constructeur de la classe `Apartment`.

Question 10. On souhaite afficher l'ensemble des caractéristiques d'un `Apartment` comme suit :

```
Surface : 82.5
Type : F4
Prix du m : 9805
Etage : 5
Ascenseur : oui
```

Définissez la méthode `__toString` dans la classe `Apartment`.

Question 11. Est-il nécessaire de définir la méthode `__clone` dans la classe `Apartment`. Justifiez votre réponse.

Question 12. Le prix d'un appartement peut varier en fonction des prestations offertes par l'immeuble. Ainsi, le prix de base d'un `Apartment` :

- se situant au rez-de-chaussée (étage égal à 0) verra son prix diminué de 15% (par rapport à son prix de base) ;

- se situant au delà du 2^e étage (celui-ci compris) verra son prix augmenté de 5% par étage par rapport au prix de l'étage précédent en cas de présence d'un ascenseur, diminué de 5% par étage par rapport au prix de l'étage précédent en cas d'absence d'ascenseur. Cette augmentation, ou diminution, se fera donc progressivement par rapport au prix de l'étage précédent et non par rapport à celui du prix de base. Par ailleurs, au delà du 4^e étage aucune augmentation ou diminution supplémentaire du prix ne sera appliquée.

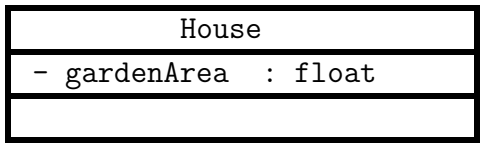
Redéfinissez la méthode `getPrice` dans la classe `Apartment`.

La classe House

La classe `House` hérite de la classe `Lodgement`. Cette classe a pour caractéristique (outre celles de la classe `Lodgement`) :

- la surface du jardin.

Voici le diagramme de classe correspondant :



Question 13. Dans le répertoire `TP5`, créez une classe nommée `House`. Vous compléterez et exécuterez le script `TestLodgements.php` à chaque ajout de nouvelle méthode dans la classe `House`. Complétez la définition de la classe `House` en y ajoutant la déclaration de ses attributs d'instance.

Question 14. On souhaite disposer d'un constructeur avec arguments dans la classe `House`. Ce constructeur prend en paramètres toutes les informations nécessaires pour initialiser l'ensemble des caractéristiques de l'objet. Définissez ce constructeur dans la classe `House`.

Question 15. Le prix d'une maison est majoré (augmenté) par le prix du jardin. Le prix d'un m^2 de jardin est égal à 10% du prix du m^2 du logement. Redéfinissez la méthode `getPrice` dans la classe `House`.

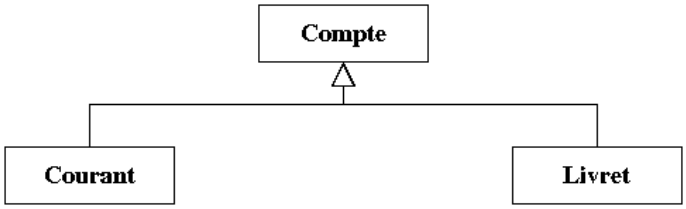
Le script TestLogements.php

Si ceci n'est pas encore fait, dans le script `TestLogements` testez l'ensemble des fonctionnalités développées dans les classes `Lodgement`, `Apartment` et `House`. Le jeu de tests proposé mettra en évidence les mécanismes d'héritage simple et de polymorphisme en créant par exemple une fonction `testLodgment` prenant en paramètre un `Lodgement` qui serait ensuite appellée avec au minimum un logement de chaque type.

Partie 3 : Les comptes bancaires

L'objet de cet exercice est de proposer une solution à la gestion des comptes bancaires. On définira pour ceci une classe `Compte` que l'on spécialisera ensuite en deux classes différentes :

`Courant` (compte chèque) et `Livret` (livret d'épargne), *c.f.* figure ci-dessous. Ces comptes sont régis par des règles différentes que l'on rappellera tout au long de ce sujet.



La classe Compte

On s'intéresse dans cette partie à la classe `Compte` qui regroupe les informations communes à tous les types de comptes. Elle représente le **concept général** d'un compte. Un `Compte` est caractérisé par : un `numero`, son `titulaire` (on se limitera ici à son nom), un `solde` exprimé en euros, positif ou négatif. Voici le diagramme de classe correspondant :

Compte
- numero : int - titulaire : string - solde : float

Question 1. Dans le répertoire `TP5`, créez une classe nommée `Compte`. Créez me script `TestComptes.php` que vous compléterez et exécuterez à chaque ajout de nouvelle méthode dans la classe `Compte`, si ceci est possible. Définissez la classe `Compte` en vous limitant dans la classe à la déclaration de ses attributs.

Question 2. On souhaiterait pouvoir instancier des `Compte` des deux manières suivantes :

```
$compte1 = new Compte ("Lagaffe" , 50408); //50408 est le numéro du compte
$compte2 = new Compte ("Dupont" , 50409 , 3000.);
                        //50409 et 3000 sont les numéro et solde du compte
```

Définissez le constructeur correspondant.

Question 3. Dans la classe `Compte`, on souhaite disposer de l'accesseur sur le solde d'un compte. Définissez cet accesseur.

Question 4. Dans la classe `Compte`, on souhaite disposer de la méthode magique `__toString`. La chaîne de caractères construite contiendra l'ensemble des valeurs des attributs d'instance. Redéfinissez la méthode `__toString`.

Question 5. Dans la classe `Compte`, on souhaite disposer des méthodes d'instance `effectuerRetrait` et `effectuerDepot`. Un compte pourra toujours être crédité ou débité quel que soit le montant et le solde du compte que concerne cette opération. Outre le fait d'effectuer l'opération, ces méthodes afficheront le numéro, le type de l'opération (dépôt ou retrait), le montant de l'opération ainsi que le nouveau solde du compte concerné par l'opération. Cet affichage pourra

être assimilé à l'édition d'un ticket de retrait ou de dépôt. La méthode `effectuerRetrait` retournera ici *true* puisque au niveau de la classe `Compte` un retrait est toujours possible.

La classe compte Courant

Un compte `Courant` est un `Compte` dont le montant maximal de `decouvert` (float) autorisé est fixé lors de l'ouverture.

Question 6. Dans le répertoire `TP5`, créez une classe nommée `Courant`. Vous compléterez et exécuterez le script `TestComptes.php` à chaque ajout de nouvelle méthode dans la classe `Courant`. Définissez la classe `Courant` en vous limitant dans la classe à la déclaration de ses attributs d'instance.

Question 7. Dans la classe `Courant`, définissez un constructeur qui pourra être appelé au minimum de deux manières différentes, l'ensemble des possibilités d'appel du constructeur de la classe `Compte` devant être sollicité.

Question 8. Dans la classe `Courant`, définissez un accesseur et un modificateur sur l'attribut `decouvert`.

Question 9. Dans la classe `Courant`, redéfinissez la méthode `effectuerRetrait`. Lorsqu'un retrait est impossible cette méthode retournera $false$, sinon elle retournera *true*. Outre le fait d'effectuer l'opération de retrait, cette méthode affichera les mêmes informations que celles qui ont été affichées par la méthode `effectuerRetrait` définie au niveau de la classe mère.

Est-il nécessaire de redéfinir la méthode `effectuerDepot` dans la classe `Courant` ? Justifiez votre réponse.

Question 10. Définissez la méthode magique `__toString` pour les objets de la classe `Courant`.

La classe Livret

Un `Livret` est un `Compte` dont le solde doit toujours être strictement positif. Il est rémunéré sur la base d'un `taux` d'intérêts (2.25% pour un livret jeune) et les intérêts ne pourront être calculés que sur une somme inférieure au `plafond` (1600 euros pour un livret jeune) même si ce compte contient plus que le `plafond` euros.

Un `Livret` étant sujet à des prises d'intérêts, une donnée supplémentaire s'impose dans chacune des instances de cette classe : le solde minimum `soldeMin` du livret au cours de la période écoulée depuis la dernière prise d'intérêts. Les intérêts d'un compte sont calculés sur ce `soldeMin` (ou le `plafond` si `soldeMin` l'excède) et non le `solde` du `Livret`. Le `soldeMin` sera mis à jour à chaque opération de retrait et de prise d'intérêts.

Question 11. Dans le répertoire `TP5`, créez une classe nommée `Livret`. Vous compléterez et exécuterez le script `TestComptes.php` à chaque ajout de nouvelle méthode dans la classe `Livret`. Définissez la classe `Livret` en vous limitant dans la classe à la déclaration de ses attributs.

Question 12. Dans la classe `Livret`, définissez un constructeur qui pourra être appelé au minimum de deux manières différentes, l'ensemble des possibilités d'appel du constructeur de la classe `Compte` devant être sollicité.

Question 13. Dans la classe `Livret`, redéfinissez la méthode `effectuerRetrait`. Lorsqu'un retrait est impossible cette méthode retournera $false$, sinon elle retournera $true$. Outre le fait d'effectuer l'opération de retrait, cette méthode affichera les mêmes informations que celles qui ont été affichées dans la méthode `effectuerRetrait` définie au niveau de la classe mère.

Est-il nécessaire de redéfinir la méthode `effectuerDepot` dans la classe `Livret` ? Justifiez votre réponse.

Question 14. Définissez la méthode `priseInterets` dans la classe `Livret`. Cette méthode calculera, puis créditera les intérêts sur un livret.

Question 15. Définissez la méthode magique `__toString` pour les objets de la classe `Livret`.

Polymorphisme

Dans le script `TestComptes.php` simulez la gestion des comptes bancaires en créant au minimum un compte de chaque type.

Question 16. Testez les fonctionnalités développées dans les différentes classes. Vous serez entre autres amenés à :
- simuler une prise d'intérêts avant retrait puis à la fin de l'ensemble des opérations ;
- effectuer un dépôt sur chacun des comptes ;
- effectuer un retrait sur chacun des comptes.

Question 17. On souhaite pouvoir effectuer un virement entre deux comptes. Dans la classe `Compte` définissez une méthode d'instance `effectuerVirement`. Cette méthode prendra en paramètre le compte destinataire du virement ainsi que la somme à transférer entre les deux comptes. Cette méthode sera, dans certains cas, amenée à ne pas effectuer de virement. Elle retournera $false$ dans ce cas là.

Est-il nécessaire de redéfinir la méthode d'instance `effectuerVirement` dans les classes `Courant` et `Livret` ? Justifiez votre réponse.

Effectuez des virements entre l'ensemble des comptes instanciés.

Question 18. Le polymorphisme est automatique en *PHP*. Définissez succinctement en quoi consiste le polymorphisme.

Question 19. On souhaite maintenant implémenter une version alternative de la méthode d'instance permettant d'effectuer un virement entre deux comptes. La méthode `effectuerVirementBis` sera déclarée abstraite au niveau de la classe `Compte`. Peut-on définir cette méthode au niveau de la classe `Compte` ? Quel impact sur la classe `Compte` a cette déclaration ? Quelle(s)

conséquence(s) concrète(s) a cet impact sur la classe `Compte`.

Est-il nécessaire de redéfinir la méthode `effectuerVirementBis` dans les classes `Courant` et `Livret` ? Justifiez votre réponse.