|  |  |
| --- | --- |
| Dans sa définition première, **un paradigme est une représentation du monde**, ou une manière de voir les choses. Le paradigme est un modèle, un exemple, ou encore une référence sur laquelle on se base pour illustrer une règle.  Le monde de la programmation dispose d'un bon nombre de paradigmes, dont le but est d'**aider le développeur à mieux penser son application** : Objet bien sûr, mais également MVC, programmation orientée Aspect, programmation déclarative...   |  | | --- | |  |   C'est ainsi que le développeur programmant à l'aide d'un langage Objet doit porter sa réflexion sur un **modèle centré sur l'objet** : les programmes qu'il construit sont composé d'objets (ou instances de classe) qui communiquent entre eux, et travaillent ensemble.  Plutôt que d'écrire un programme de manière séquentielle, le développeur devra réfléchir à la **pertinence de tel objet ou telle méthode dans le cadre du programme global**, chaque objet disposant d'une responsabilité précise.  Le paradigme Objet nécessite donc des objets qui interagissent, ce qui influe profondément l'approche du code. Pour un même résultat, un code Objet sera bien différent d'un code utilisant la programmation fonctionnelle, où le programme est une séquence d'évaluations de fonctions.  **Un paradigme n'est pas forcément lié à un langage** : certains peuvent aussi bien utiliser du procédural, de l'orienté objet, du fonctionnel et du générique, pour une même syntaxe. En ce sens, le paradigme objet n'est pas lié directement aux interfaces Java.  Cela étant, les interfaces de Java, par l'abstraction qu'elles offrent, permettent un héritage plus poussé, par encapsulation d'objets. En cela, les interfaces de Java renforcent le paradigme Objet, même si elles ne lui sont pas nécessaires. |
| |  | | --- | |  | |

Tandis que la **programmation procédurale** est un paradigme qui se fonde sur le concept d'appel procédural.

Une procédure, aussi appelée *routine*, *sous-routine* ou *fonction* (à ne pas confondre avec les fonctions de la programmation fonctionnelle reposant sur des fonctions mathématiques), contient simplement une série d'étapes à réaliser. N'importe quelle procédure peut être appelée à n'importe quelle étape de l'exécution du programme, y compris à l'intérieur d'autres procédures, voire dans la procédure elle-même (récursivité).

1) **namespace**

Un **espace de nom** ou **namespace** représente un moyen de séparer ses éléments au sein du code de telle sorte à éviter les conflits (ou collisions). Ces collisions sont dues à des duplications de noms (ou identifiants) d'éléments comme les fonctions, les constantes ou les classes.

On peut se service des espaces de noms quand on veut, mais les développeurs expérimentés les déconseillent sur des petits projets, car on risque de se retrouver avec de nombreux namespaces qui encapsulent chacun un ou deux éléments, ce qui peut donner lieu à du travail supplémentaire qui consiste à se déplacer entre ces espaces-là bien que le risque de collision n'était pas vraiment considérable. Donc, le jeu n'en vaut pas vraiment la chandelle. Cependant, on peut y recourir dès les cas suivants:

* **Dans un gros projet**: où le nombre de fonctions, de constantes et de classes mises en jeu est énorme, donc le risque de collision est élevé.
* **Dans des projets qui utilisent des librairies externes**: dans ce cas, il se peut que l'on utilise des éléments qui peuvent avoir le même nom que leur homologue qui sont embarquées dans la librairie.

Toutefois, il n'existe pas une règle fixe qui dicte quand est ce qu'on doit vraiment se servir des namespaces. Alors, ça revient à vous d'estimer quand est ce qu'ils pourront vous être utiles.

### **Que est-ce qu'un objet ?**

#### **Définition**

Un « objet » est une représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés et des actions.

Exemple : une voiture, une personne, un personnage, un animal, un nombre ou bien un compte bancaire peuvent être vus comme des objets.

#### **Attributs ou données membres**

Les « attributs » (aussi appelés « données membres ») sont les caractères propres à un objet.

Exemple : Une personne, par exemple, possèdent différents attributs qui lui sont propres comme le nom, le prénom, la couleur des yeux, le sexe, la couleur des cheveux, la taille...

#### **Méthodes**

Les « méthodes » sont les actions applicables à un objet.

Exemple : Un objet personne dispose des actions suivantes : manger, dormir, boire, marcher, courir...

### **Qu'est ce qu’ une classe ?**

Une classe est une entité regroupant des variables et des fonctions. Chacune de ces fonctions aura accès aux variables de cette entité. Dans le cas du personnage, nous aurons une fonction frapper (). Cette fonction devra simplement modifier la variable $degats du personnage en fonction de la variable $force. Une classe est donc un regroupement logique de variables et fonctions que tout objet issu de cette classe possédera.

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 et une méthode désigne une fonction.

Exemple

Les classes contiennent la définition des objets que l'on va créer par la suite. Prenons l'exemple le plus simple du monde : les gâteaux et leur moule. Le moule est unique. Il peut produire une quantité infinie de gâteaux. Dans ce 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 un modèle de données définissant la structure commune à tous les objets qui seront créés à partir d'elle. Plus concrètement, nous pouvons percevoir une classe comme un moule grâce auquel nous allons créer autant d'objets de même ***type*** et de même structure qu'on le désire.

Par exemple, pour modéliser n'importe quelle personne, nous pourrions écrire une classe Personne dans laquelle nous définissons les attributs (couleurs des yeux, couleurs des cheveux, taille, sexe...) et méthodes (marcher, courir, manger, boire...) communs à tout être humain.

**4) Classes abstraites**

De la même façon qu'il est possible d'empêcher quelqu'un d'étendre une classe, ou de surcharger une méthode, il est possible de forcer l'extension ou la surcharge, en utilisant le mot-clé abstract.

Formellement une classe abstraite n'est pas différente d'une classe normale. Simplement, elle est déclarée en ajoutant le mot-clé abstract, comme dans l'exemple suivant.

**. Classe abstraite**

On peut déclarer autant de méthodes abstraites que l'on veut dans une classe abstraite, y compris ne pas en déclarer du tout. Une méthode abstraite, comme dans notre exemple, ne comporte qu'une déclaration de signature ; elle n'a pas de corps. On peut alors se poser la question : comment fait-on pour exécuter une telle méthode ? La réponse est simple : on ne peut pas l'exécuter en l'état.

Une classe abstraite ne peut pas être instanciée. Il est nécessaire de créer une classe concrète (c'est-à-dire non abstraite !) qui l'étende, et d'instancier cette classe concrète. Toute classe concrète qui étend une classe abstraite doit fournir un corps de méthode (on parle alors d'implémentation, un beau mot de la langue française...) pour toutes les méthodes abstraites de cette classe abstraite.

5)METHODE STATIQUE

## Définition des propriétés et méthodes statiques

Une propriété ou une méthode statique est une propriété ou une méthode qui ne va pas appartenir à une instance de classe ou à un objet en particulier mais qui va plutôt appartenir à la classe dans laquelle elle a été définie.

Les méthodes et propriétés statiques vont donc avoir la même définition et la même valeur pour toutes les instances d’une classe et nous allons pouvoir accéder à ces éléments sans avoir besoin d’instancier la classe.

Pour être tout à fait précis, nous n’allons pas pouvoir accéder à une propriété statique depuis un objet. En revanche, cela va être possible dans le cas d’une méthode statique.

Attention à ne pas confondre propriétés statiques et constantes de classe : une propriété statique peut tout à fait changer de valeur au cours du temps à la différence d’une constante dont la valeur est fixée. Simplement, la valeur d’une propriété statique sera partagée par tous les objets issus de la classe dans laquelle elle a été définie.

De manière générale, nous n’utiliserons quasiment jamais de méthode statique car il n’y aura que très rarement d’intérêt à en utiliser. En revanche, les propriétés statiques vont s’avérer utiles dans de nombreux cas.

5)METHODE CONSTRUCTEUR

En **PHP**, le **constructeur** est une **méthode** qui doit s'appeler \_\_construct(). Le code suivant, instanciera un objet de la classe Ma Classe A Moi en appelant la **méthode** \_\_construct (qui selon l'exemple affichera au passage "Ca y est je suis prêt").

6) **encapsulation**

L'**encapsulation** est un pilier de la programmation orientée objet, où chaque **classe** définit des méthodes ou des propriétés pour interagir avec les données membres, mais ce principe peut se rencontrer dans d'autres styles de programmation (par exemple la programmation modulaire).

La visibilité des propriétés et des méthodes d'un objet constitue une des particularités élémentaires de la programmation orientée objet. Ce tutoriel a pour objectif de présenter les différents niveaux de visibilité que propose le modèle objet de PHP 5. Nous les passerons en revue un par un au travers d'exemples pratiques et nous apporterons quelques bonnes pratiques à adopter lorsqu'on les utilise.

### Qu'est-ce que la visibilité des propriétés et méthodes d'un objet ?

La visibilité permet de définir de quelle manière un attribut ou une méthode d'un objet sera accessible dans le programme. Comme Java, C++ ou bien ActionScript 3, PHP introduit 3 niveaux différents de visibilité applicables aux propriétés ou méthodes de l'objet.

Il s'agit des visibilités ***publiques, privées*** ou **protégées** qui sont respectivement définies dans la classe au moyen des mots-clés **public, private**et ***protected***.

L'exemple suivant illustre la syntaxe de déclaration de données membres (attributs) et de méthodes ayant des visibilités différentes. Nous expliquerons juste après chaque particularité des niveaux de visibilité.

## Get and Set

Vous avez appris dans le chapitre précédent que private variables private ne sont accessibles que dans la même classe (une classe externe n'y a pas accès). Cependant, il est possible d'y accéder si nous fournissons des méthodes publiques **get** et **set**.

La méthode get renvoie la valeur de la variable et la méthode set définit la valeur.

La syntaxe des deux est qu'ils commencent par get ou set, suivis du nom de la variable, avec la première lettre en majuscule

#### **Exemple expliqué**

La méthode get renvoie la valeur du name de la variable.

La méthode set prend un paramètre ( newName ) et l'affecte à la variable de name . Le mot-clé this est utilisé pour faire référence à l'objet actuel.

Cependant, comme la variable name est déclarée private , nous **ne pouvons pas y** accéder depuis l'extérieur de cette classe:

f) **surcharge**

La **surcharge** (également connue sous le nom de **surdéfinition**, [**polymorphisme**](https://fr.wikipedia.org/wiki/Polymorphisme_(informatique))**ad hoc** ou ***overloading*** en anglais) est une possibilité offerte par certains [langages de programmation](https://fr.wikipedia.org/wiki/Langage_de_programmation) qui permet de choisir entre différentes versions d'une même fonction ou méthode selon le nombre et le [type](https://fr.wikipedia.org/wiki/Type_(informatique)) des arguments fournis.

Le polymorphisme ad hoc ne doit pas être confondu avec le polymorphisme d'inclusion des [langages à objets](https://fr.wikipedia.org/wiki/Programmation_orient%C3%A9e_objet), permis par l'[héritage](https://fr.wikipedia.org/wiki/H%C3%A9ritage_(informatique)) de [classe](https://fr.wikipedia.org/wiki/Classe_(informatique)) et la redéfinition de [méthode](https://fr.wikipedia.org/wiki/M%C3%A9thode_(informatique)) (*overriding* en anglais).

La surcharge peut être [statique](https://fr.wikipedia.org/wiki/Typage_statique) (le choix de la version est alors fait en fonction du nombre d'arguments et de leur type statique [déclaré](https://fr.wikipedia.org/wiki/D%C3%A9claration_(informatique)) à la [compilation](https://fr.wikipedia.org/wiki/Compilateur)) ou [dynamique](https://fr.wikipedia.org/wiki/Typage_dynamique) (le choix de la version est alors fait en fonction du type dynamique des arguments constaté à l'exécution). La surcharge dynamique est également appelée « [dispatch multiple](https://fr.wikipedia.org/wiki/Dispatch_multiple) » et une méthode surchargée dynamiquement « multiméthode »

La surcharge des opérateurs tels que +, -, ou \* peut surprendre le lecteur d'un programme, en autorisant le programmeur à donner à une même fonction des fonctionnalités subtilement différentes en fonction des types de ses opérandes, pour gagner en lisibilité

Objet

Un « objet » est une représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés et des actions.

*Exemple : une voiture, une personne, un personnage, un animal, un nombre ou bien un compte bancaire peuvent être vus comme des objets.*

### **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 celui qui se servira de la classe, pas celui qui chargera la page depuis son navigateur). 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. Comme le pilote de l'avion, il n'a pas à les trifouiller. Pour instaurer une telle contrainte, on dit que les attributs sont **privés**.

### **Déclaration d'une classe**

Nous venons de définir le vocabulaire propre à la programmation orientée objet. Entrons à présent dans le vif du sujet, c'est à dire la déclaration et l'instanciation d'une classe. Nous allons déclarer une classe Personne qui nous permettra ensuite de créer autant d'instances (objets) de cette classe que nous le souhaitons.

Comme vous pouvez le constater, nous avons déclaré 5 attributs publics, 5 constantes, 1 méthode constructeur et 2 méthodes classiques. Détaillons chaque élément.

### **Les attributs**

Comme nous l'avons expliqué au début de ce cours, les attributs sont les caractéristiques propres d'un objet. Toute personne possède un nom, un prénom, une date de naissance, une taille, un sexe... Tous ces éléments caractérisent un être humain.

Par cet exemple, nous déclarons les attributs de notre classe public.

Le mot-clé public permet de rendre l'attribut accessible depuis l'extérieur de la classe. Ce n'est pas une bonne pratique à adopter mais pour ce premier exemple à gauche, nous l'utiliserons comme.

En programmation orientée objet, un attribut n'est ni plus ni moins qu'une variable.

**Notez également que nous avons juste déclaré les attributs. En revanche, aucun type ni aucune valeur ne leur ont été attribués. Ils sont donc par défaut initialisés à la valeur NULL.**

### **Les constantes**

Il est aussi possible de déclarer des constantes propres à la classe. Contrairement au mode procédural de programmation, une constante est déclarée avec le mot-clé ***const***.

### **Le constructeur**

Le constructeur est une méthode particulière. C'est elle qui est appelée implicitement à la création de l'objet (instanciation).

Dans notre exemple, le constructeur n'a ni ne paramètre ni instruction. Le programmeur est libre de définir des paramètres obligatoires à passer au constructeur ainsi qu'un groupe d'instructions à exécuter à l'instanciation de la classe. Nous nous en passerons pour simplifier notre exemple.

Nous le détaillerons dans l'article suivant.

### **Les méthodes**

Les méthodes sont les actions que l'on peut appliquer à un objet. Il s'agit en fait de fonctions qui peuvent prendre ou non des paramètres et retourner ou non des valeurs / objets. S'agissant d'actions, nous vous conseillons de les nommer avec un verbe à l'infinitif.

Elles se déclarent de la même manière que des fonctions traditionnelles

Au même titre que les attributs, on déclare une méthode avec un niveau de visibilité. Le mot-clé public  
indique que l'on pourra appliquer la méthode en dehors de la classe, c'est à dire sur l'objet lui-même.

#### **Exiger des objets en paramètre**

Reprenons l'exemple de la méthode frapper (). Celle-ci accepte un argument : un personnage à frapper. Cependant, qu'est-ce qui vous garantit qu'on passe effectivement un personnage à frapper ?

On pourrait très bien passer un argument complètement différent, comme un nombre par exemple.

Et là, qu'est-ce qui se passe ? Une erreur est générée car, à l'intérieur de la méthode frapper (), nous essayons d'appeler une méthode sur le paramètre qui n'est pas un objet.

### **Instanciation d'une classe**

L'instanciation d'une classe est la phase de création des objets issus de cette classe.

Lorsque l'on instancie une classe, on utilise le mot-clé ***new*** suivant du nom de la classe.

Cette instruction appelle la méthode constructeur (\_\_construct() ) qui construit l'objet et le place en mémoire.

Voici un exemple qui illustre 4 instances différentes de la classe Personne.

### **Accès aux attributs**

#### **Accès en écriture**

Nous venons de créer 4 objets de même type et de même structure. Dans l'état actuel, ce sont des clones car leurs attributs respectifs sont tous déclarés mais ne sont pas initialisés. Nous affectons à présent des valeurs à chacun des attributs de chaque objet.

Nous avons maintenant des objets ayant chacun des caractéristiques différentes.

Dans notre exemple, nous accédons directement à la valeur de l'attribut. Cela est possible car nous avons défini l'attribut comme étant public. Si nous avions déclaré l'attribut avec les mots-clés private ou protected, nous aurions dû utiliser un autre mécanisme pour accéder à sa valeur ou bien la mettre à jour. Nous verrons cela dans un prochain cours.

#### **Accès en lecture**

La lecture de la valeur d'un attribut d'un objet se fait exactement de la même manière que pour une variable traditionnelle.

Le code suivant présente comment afficher le nom et le prénom de la personne 1.

#### **Accès aux méthodes**

Leur utilisation est exactement la même que pour les attributs.

Rappelez-vous, nous avions défini deux méthodes pour notre classe Personne. Il s'agit des méthodes boire () et manger (). Toutes deux affichent une chaine de caractères sur la sortie standard lorsqu'elles sont exécutées.

Reprenons nos 4 objets précédents. Nous simulons que ces 4 personnes sont à table et qu'elles dînent. Simultanément les deux premières personnes boivent le contenu d'un verre pendant que les deux autres mangent. Cela se représente donc par l'appel de la méthode boire () sur les objets personne1 et personne2, puis par l'appel de la méthode manger() sur les objets personne3 et personne4.