NOVEI FORMATION

LA MANU

L'ÉCOLE DES MÉTIERS DU NUMÉRIQUE
Manufacture de compétences

SUPPORT APPRENANT

Programmation Orientée Objet

CONFIDENTIEL

Programmation Orientée Objet

Ce que l'on a vu…

La **P**rogrammation **O**rientée **O**bjet, souvent appelée par son acronyme POO, est une façon de coder plus modulaire en utilisant des objets, ce qui permet notamment d'avoir un code plus facilement réutilisable.
Les concepts de la POO voient le jour dans les années 1970 dans des laboratoires de recherche en informatique, et sont appliqués dans des langages comme le C++ à partir des années 1980. Ce n'est que dans la décennie suivante que l'avènement de la POO aura lieu dans de nombreux secteurs du développement logiciel. Avant d'utiliser la POO, on utilisait principalement la programmation procédurale qui consiste à séparer le traitement des données, des données elles-mêmes.
L'architecture des applications utilisant la POO est différente de celle des applications utilisant la programmation procédurale. Avant de commencer à coder son application, on peut passer par une phase de modélisation orientée objet. Pour cela on utilise des diagrammes réalisés grâce au langage de modélisation UML.

POO

En POO, il est important de bien comprendre la notion d'objet qui est le cœur de ce type de programmation.
Un objet en programmation, comme dans la vie, correspond à une entité qui représente un élément. Un **objet** est défini par des caractéristiques et des actions, en d'autres termes un objet possède des propriétés et peut faire ou subir des actions. Avant de pouvoir utiliser un objet, il faut réaliser une instanciation. L'**instanciation** permet de créer un objet à partir « d'un moule » appelé **classe**. Elle se fait à l'aide de l'opérateur **new** de la manière suivante :

```
$nomObjet = new NomClasse() ;
```

C'est dans la **classe** que l'on définit tout ce qui caractérise les objets (propriétés et actions). Par convention le nom des classes commence par une majuscule. La déclaration d'une classe se fait ainsi :

```
class NomClasse
{
    //déclaration des attributs
    //déclaration des méthodes
}
```

Les propriétés d'un objet s'appellent des **attributs**, et les actions que cet objet peut réaliser sont définies par des fonctions qui s'appellent des **méthodes**. L'appel d'une méthode, déclarée dans une classe, par un objet se fait grâce à l'opérateur « **->** » :

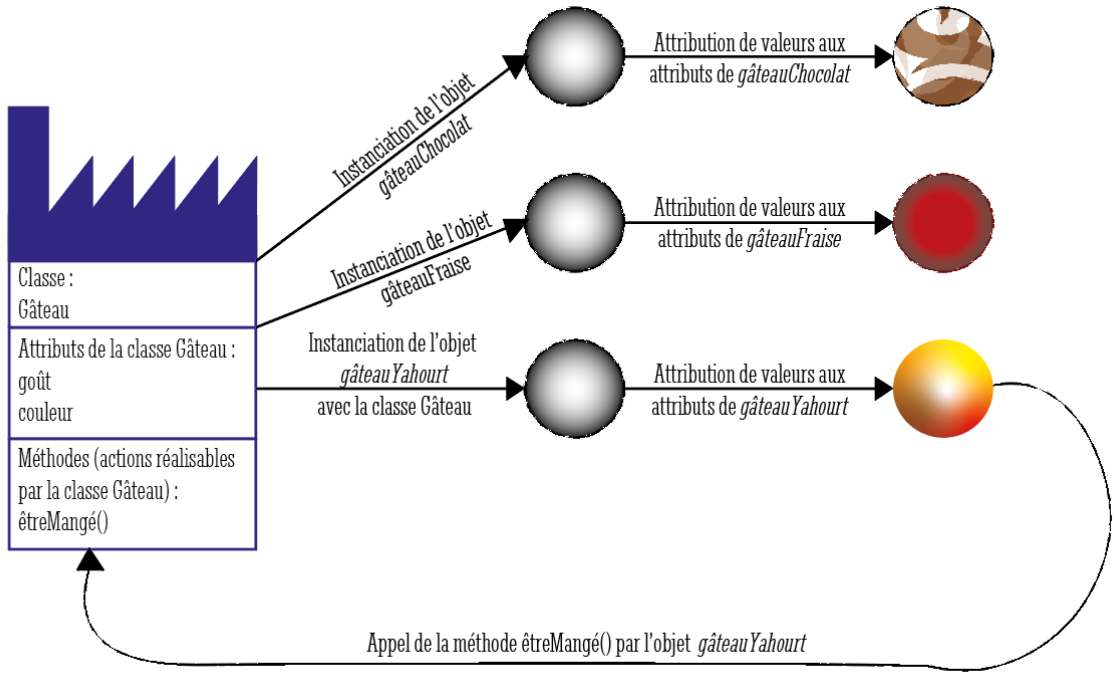
```
$nomObjet->nomMéthode() ;
```

On utilise également cet opérateur pour accéder à la valeur d'un attribut de l'objet ou pour attribuer une valeur à un attribut de l'objet :

```
//attribution de la valeur de l'attribut de l'objet à la variable
$nomVariable = $nomObjet->nomAttribut ;
//attribution de la valeur à l'attribut de l'objet
$nomObjet->nomAttribut = valeur ;
```

Si on utilise une analogie pour mieux comprendre, la classe représente un moule et avec ce moule on peut faire des objets. Le moule « Gâteau » permet de faire des objets qui sont des gâteaux. Un gâteau sera défini par des propriétés telles qu'une couleur et un goût, et par des actions comme « être mangé » par exemple.
Avec une même classe il est possible de faire de nombreux objets qui ont les mêmes propriétés et qui peuvent faire les mêmes actions mais la valeur de leurs propriétés est propre à chaque objet, ainsi

chaque objet est différent. De la même façon avec un moule à gâteaux, il possible de faire de nombreux gâteaux qui ont tous une couleur et un goût et qui peuvent tous « être mangé », mais la couleur et le goût de ces gâteaux peuvent être différentes, ces valeurs sont propres à chaque gâteau. Ainsi, on peut avoir un gâteau au chocolat marron, un gâteau au yahourt jaune, un gâteau à la fraise rouge… Chaque gâteau est un objet différent, mais ils sont tous issus du même moule à gâteaux et possèdent tous les mêmes propriétés. De même qu'ils peuvent tous « être mangé » en faisant appel à cette méthode.



La classe donne en quelques sortes la définition de l'objet. Il s'agit d'un concept abstrait alors que l'objet est la représentation concrète de ce concept abstrait.
Un objet étant le résultat de l'instanciation d'une classe on peut parfois parler d'instance d'objet ou d'instance de classe.

Lorsqu'on instancie un objet on fait automatiquement appel à une méthode particulière que l'on nomme un constructeur. Les constructeurs font parties de ce que l'on appelle des méthodes magiques. Les méthodes magiques sont des fonctions PHP spéciales. Elles se reconnaissent à leur syntaxe qui commence avec 2 underscores suivis du nom de la méthode. Il existe plusieurs méthodes magiques mais les plus utilisées sont les méthodes **__construct()** et **__destruct()**. L'utilisation de ces méthodes n'est pas obligatoire mais elle est très fortement recommandée.
La méthode __construct() d'une classe est appelée à chaque instanciation d'un nouvel objet avec cette classe. Elle permet notamment d'initialiser la valeur des attributs d'un objet à sa création ou de faire la connexion à une base de données par exemple. En revanche, la méthode __destruct() permet de détruire un objet. Elle est appelée dès qu'il n'y a plus de référence à un objet ou s'il n'existe plus dans le contexte de l'application.
Les méthodes magiques doivent toujours être déclarées en public.
En revanche, les autres méthodes et les attributs peuvent être aussi bien déclarés en public, private ou protected selon les besoins. Les mots-clés public, private et protected correspondent à la visibilité

des éléments. Ils définissent la portée des éléments, celle-ci est différente selon le niveau de visibilité employé. On parle aussi d'encapsulation ou d'accessibilité.

Visibilité	
public	Attributs et/ou méthodes accessibles dans la classe où ils ont été déclarés et à l'extérieur.
private	Attributs et/ou méthodes accessibles uniquement dans la classe où ils ont été déclarés.
protected	Attributs et/ou méthodes accessibles uniquement dans la classe où ils ont été déclarés et dans ses héritiers.

Lorsque le niveau de visibilité des attributs est en private, il n'est pas possible d'y accéder à l'extérieur de la classe dans laquelle ils ont été déclarés. Cela permet de protéger le code de mauvaises manipulations, notamment lorsqu'il va être mis à disposition d'autres développeurs, en ne laissant accessible que le strict nécessaire pour l'utilisation de la classe. La convention de nommage d'un attribut ou d'une méthode en private veut qu'ils soient déclarés en précédant le nom de l'élément d'un underscore.
Par exemple :

```
private $_nomAttribut ;
```

Lorsqu'on utilise ce niveau d'encapsulation il faut prévoir des accesseurs pour permettre d'accéder aux attributs et de les modifier en dehors de leur classe.
Les accesseurs sont des méthodes publiques qui permettent d'accéder à des attributs privés. Il y a des accesseurs qui accordent la lecture des attributs et d'autres qui autorisent la modification de ces attributs. Un accesseur qui permet de lire la valeur d'un attribut privé est un getter. Alors qu'un accesseur qui permet de modifier la valeur d'un attribut privé est un setter.
Pour accéder à l'attribut d'un objet dans une méthode on utilise la pseudo-variable **$this** suivi de l'opérateur « **->** » et du nom de l'attribut auquel on souhaite accéder.
Exemple :

```
public function getName()
{
   return $this->name ;
}
```

La pseudo-variable $this se réfère à l'objet qui appelle la méthode. Dans l'exemple ci-dessus, la méthode *getName()* retourne la valeur de l'attribut *name* de l'objet qui appelle cette méthode. Ainsi les valeurs retournées pourront être différentes si la méthode est appelée par différents objets.
Le nom d'un getter ou d'un setter contient généralement le mot get ou set suivi du nom de l'attribut auquel il permet d'accéder :

```
//getter
public function getNomAttribut()
{
   return $this->nomAttribut ;
}

//setter
public function setNomAttribut(paramètre)
{
   $this->nomAttribut = paramètre ;
}
```

En plus d'autoriser l'accès aux attributs d'un objet les accesseurs rendent également possible d'effectuer des contrôles supplémentaires. Ils permettent par exemple de vérifier que les données transmises sont bien celles attendues, limitant ainsi le risque d'attribuer des valeurs au mauvais format à un attribut.

La notion d'héritage fait partie des concepts principaux de la Programmation Orientée Objet. L'héritage permet de créer des classes à partir d'autres classes déjà existantes.
La classe existante est alors considérée comme une classe mère (ou parent) à partir de laquelle il est possible de créer une classe fille (ou enfant). La classe fille hérite des attributs et des méthodes de la classe mère. Pour cela les attributs et les méthodes du parent doivent être déclarés en public ou en protected.

La classe mère a des caractéristiques plus générales que celles d'une classe fille, et une classe fille a des caractéristiques plus spécialisées que la classe mère.
Par exemple, un chien ou un poisson sont tous deux des animaux. On peut dire qu'ils héritent tous deux des caractéristiques générales d'un animal, mais chacun a des caractéristiques supplémentaires plus spécifiques à leur espèce.
Avec l'héritage simple, un parent peut avoir plusieurs enfants, mais un enfant ne peut avoir qu'un seul parent. Contrairement à l'héritage multiple qui autorise qu'une classe fille hérite de plusieurs classes mères.

En PHP, pour spécifier qu'une classe hérite d'une autre il faut la déclarer grâce au mot-clé **extends** et en précisant la classe parente.
Pour que la classe fille hérite des méthodes de la classe mère il faut d'abord déclarer une méthode du même nom et comportant les mêmes paramètres que la classe mère. Ensuite, à l'intérieur de cette méthode il faut écrire le mot-clé **parent** suivi de 2 symboles « **:** » puis du nom de la méthode dont on souhaite hériter.
Exemple :

```
class NomClasseFille extends NomClasseMère
{
   //déclaration d'attributs spécifiques à la classe fille (facultatif)
   public function __construct()
   {
      parent::__construct() ;
   }
   //déclaration de méthodes spécifiques à la classe fille
   public function __destruct()
   {
      parent::__destruct() ;
   }
}
```

Lorsqu'une classe hérite d'une autre classe, il est possible de surcharger les méthodes de la classe parente qu'elle utilise. C'est ce qu'on appelle du polymorphisme. Le polymorphisme est une notion liée à celle d'héritage.
Du grec « plusieurs formes », le polymorphisme permet à une méthode d'avoir différentes formes. Lorsqu'une méthode est déclarée dans une classe fille, elle a la même forme que la méthode déclarée dans la classe mère. Cependant en ajoutant une surcharge à la méthode dans la classe fille elle prendra une forme différente. Le polymorphisme permet donc à une classe enfant de surcharger une méthode de la classe parente pour en modifier le fonctionnement.

Cela peut être utile si une classe fille emploie une méthode de la classe mère mais avec des valeurs d'attributs différentes par exemple.

```
class NomClasseFille extends NomClasseMère
{
   //déclaration d'attributs spécifiques à la classe fille (facultatif)
   public function __construct()
   {
      //surcharge de la méthode
      $this->nomAttribut = valeurDifférenteDuParent ;
      parent::__construct() ;
   }
   …
}
```

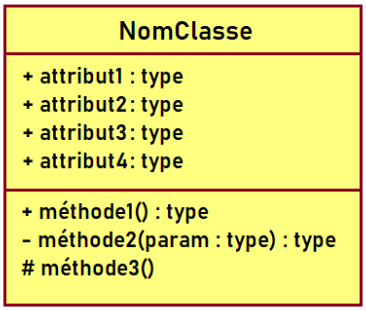
UML

L'acronyme UML signifie **U**nified **M**odeling **L**anguage, soit langage de modélisation unifié en français. Ce langage est constitué de diagrammes permettant de représenter de manière visuelle les besoins d'une application à développer. Il existe 13 types de diagrammes (diagramme de classe, diagramme de package, diagramme de cas d'utilisation…) qui donnent chacun une vision du projet, cependant il n'y a pas de méthode particulière à suivre, que ce soit pour les diagrammes à utiliser ou l'ordre dans lequel les réaliser. Il existe tout de même certaines normes à respecter pour qu'ils puissent être compris de la même manière par tous.
Les diagrammes UML peuvent être réalisés à la main ou avec des logiciels tels que Umbrello, StarUml ou PowerDesign.

Diagramme de classe

Le diagramme de classe est un des types de diagrammes faisant partie de la norme UML. C'est un diagramme qui permet d'avoir une bonne représentation graphique de toutes les classes d'une application à développer en programmation orientée objet. Il donne ainsi une vision des fonctions et des informations qui seront utilisées par l'application.
Il s'agit d'une représentation universelle qui peut être comprise par tous les développeurs, quelques soit le langage de programmation orientée objet qu'ils utilisent. Pour que le diagramme de classe soit universel il doit répondre à certaines normes.
Les classes sont représentées par un rectangle divisé en 3 parties : le nom de la classe, ses attributs et ses méthodes. Comme dans le code le nom de la classe commence par une majuscule. Les attributs et les méthodes de la classe sont précédés d'un symbole représentant la visibilité de l'élément :

- **+** : public
- **-** : private
- **#** : protected



Le type retourné par les méthodes et celui des attributs est également présent. Si des méthodes prennent un paramètre, celui-ci figure entre les parenthèses de la méthode suivi de son type.
Sur un diagramme de classes, on retrouve également les interactions entre les différentes classes de l'application. Celles-ci répondent elles aussi à des normes. Les interactions peuvent être de différents types, telles que l'héritage, l'association, l'agrégation ou la composition.

Les relations d'héritage sont symbolisées par une flèche allant de la classe fille à la classe mère, soit la pointe de la flèche sur la classe parent. Dans le logiciel Umbrello, les relations d'héritage peuvent se faire à l'aide du bouton « Generalization ».

Liens utiles

https://www.php.net/
https://openclassrooms.com/fr/courses/1665806-programmez-en-oriente-objet-en-php/1665911-introduction-a-la-poo
https://umbrello.kde.org/
https://docs.kde.org/trunk5/en/kdesdk/umbrello/uml-elements.html#class-diagram
https://bpesquet.gitbooks.io/programmation-orientee-objet-csharp/content/chapters/01-initiation-poo.html
https://www.grafikart.fr/formations/programmation-objet-php
https://www.pierre-giraud.com/php-mysql/cours-complet/php-poo-classes-objets.php
https://pear.php.net/manual/en/standards.php
https://www.commentcamarche.net/contents/811-poo-le-polymorphisme#le-polymorphisme-d-heritage
https://www.mediaforma.com/php-mysql-polymorphisme/
https://tutowebdesign.com/poo-php.php
http://igm.univ-mlv.fr/~cherrier/download/imac/P004.pdf

Bonnes pratiques

Rappel des principales bonnes pratiques	
☐	1 fichier par classe.
☐	Nom des classes doit être explicite et commencer par une majuscule.
☐	Mettre un _ devant les noms d'éléments privés. Ex : $*_nomAttribut* ; *_nomMéthode*()
☐	Les méthodes magiques doivent être déclarées en public.
☐	L'utilisation de l'anglais est obligatoire pour les noms de variables, des classes, des id…
☐	Pour une meilleure compréhension du code, il faut indenter son code.
☐	Utiliser les commentaires.
☐	Utilisation des simples quotes.
☐	Vocabulaire spécifique : • Objet • Classe • Instance • Héritage • Méthodes • Attributs • Diagramme de classe