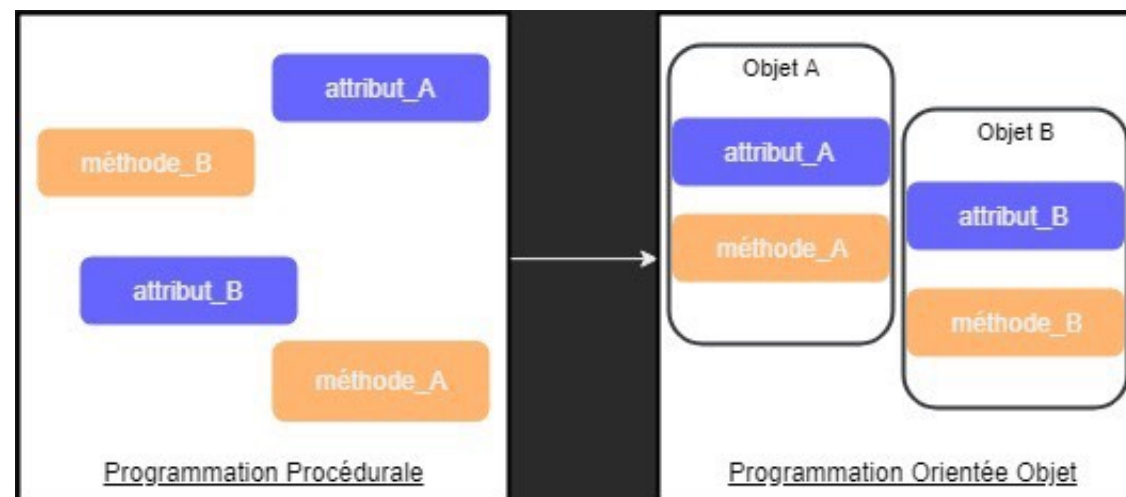


POO

BTS SIO :: SLAM

POO Introduction

- La prog procédurale -> séparation données et traitements
- POO : données & traitements rassemblés dans un même concept (objet)
- L'objet est l'élément où l'on retrouve données et traitements



Un objet

- Attention à ne pas confondre avec l'objet de la vie courante. Ici un séjour peut être un objet. Un objet n'est pas toujours tangible (comme les entités)
- Un objet est un concept à qui on attribue des valeurs grâce aux variables (on parle d'attributs dans le cas d'un objet) mais aussi des traitements ou actions grâce à des fonctions (on parle de méthodes pour les objets)

Une classe

- La fabrication de l'objet est confiée à un moule qu'on appelle **une classe** en POO.
- C'est la classe qui contient les éléments (attributs et méthodes) qu'il faudra incorporer pour construire l'objet.
- Pour créer un objet on utilise l'opérateur **new()**
- Le fait de créer un objet à partir d'une classe s'appelle : **instancier**

Classe **Personnage**

\$force
\$localisation
\$experience
\$degats

fonction frapper()
fonction gagnerExperience()
fonction deplacer()

Une classe

- Comme c'est un objet que l'on va créer on ne doit pouvoir interagir qu'à travers son interface (ses méthodes).
- On va donc rendre inaccessible ses propriétés dans la grande majorité des cas. Pour cela on mettra le mot clef : `private`.
- On reviendra sur ce principe qu'on appelle l'**encapsulation**.
- Pour accéder à un attribut de notre objet on utilise `$this` suivi du nom de l'attribut. ex : `$this->_nom`. Cela évite aussi de confondre avec une variable dans le programme.

Exemple de classe

```
<?php
class Personnel {
    // -- Propriétés --
    private $_nom = "Dupont";
    protected $prenom = "Pierre";
    public $age = 55;

    // Méthodes
    public function AffichePersonne() {
        echo "Personne : ";
        echo $this->_nom."\t".$this->prenom."\t".$this->age;
        echo PHP_EOL;
    }
}

// Programme
// Instanciation
$salarie = new Personnel();
// Utilisation de la méthode de l'objet
$salarie->AffichePersonne();
?>
```

Norme : notation PEAR -> \$_nom pour les données private

L'objet salarie est créé avec le new()

Pour appeler une méthode on utilise le signe ->

On peut donner des valeurs par défaut \$_nom = « Dupont »

PHP_EOL renvoie une chaîne correspondant au saut de ligne sur la plateforme

Utilisation d'une classe

- Si on utilise notre classe comme telle on ferait que des Pierre Dupont de 55 ans.
- De plus l'attribut **\$age** est accessibles à l'extérieur
- On va donc rendre les attributs **private**
- Découvrir l'intérêt d'un **constructeur**

```
<?php
class Personnel {
    // --- Propriétés ---
    private $_nom = "Dupont";
    protected $prenom = "Pierre";
    public $age = 55;

    // Méthodes
    public function AffichePersonne() {
        echo "Personne : ";
        echo $this->_nom."\t".$this->prenom."\t".$this->age;
        echo PHP_EOL;
    }
}

// Programme
// Instanciation
$salarie = new Personnel();
// Utilisation de la méthode de l'objet
$salarie->AffichePersonne();
?>
```

Constructeur de classe

```
1  <?php
2  class Personnel {
3      // -- Propriétés --
4      private $_nom ;
5      private $_prenom ;
6      private $_age ;
7
8      // Constructeur & destructeur
9      function __construct($n, $p, $a) {
10         $this->_nom = $n;
11         $this->_prenom = $p;
12         $this->_age = $a;
13     }
14
15     function __destruct() {
16         unset ($this->_nom);
17         unset ($this->_prenom);
18         unset ($this->_age);
19         echo "Destruction de l'objet";
20     }
21
22     // Methodes
23
24     public function AffichePersonne() {
25         echo "Personne : ";
26         echo $this->_nom."\t".$this->_prenom."\t".$this->_age;
27         echo PHP_EOL;
28     }
29 }
30
31
32 // Programme
33 $salarieB = new Personnel("dupond", "jean", 45);
34 $salarieB->AffichePersonne();
35 unset($salarieB);
36 ?>
```

Exercices

Exercice 1.1 : Ecrire une classe **Point** qui permet de représenter un point dans un espace en 2 dimensions grâce à ses coordonnées x et y. Ajouter à la classe **Point** un constructeur prenant en paramètre 2 nombres réels afin d'initialiser ses coordonnées.
Ecrire la méthode **affichePoint()** qui permet d'afficher ses coordonnées.

Exercice 1.2 : Pour afficher le point utiliser maintenant la méthode magique : `__toString()`. Regarder ce qu'est une méthode magique en PHP et étudier en particulier `__toString()` afin de l'utiliser dans vos exercices.

Utilisation d'une classe

- Les méthodes qui permettent d'accéder à la valeur d'un attribut depuis l'extérieur de l'objet sont appelées **accesseurs** ou **getters**.
- Les méthodes qui permettent de modifier les valeurs sont appelées **mutateurs** ou **setters**.
- En effet on ne peut directement modifier un attribut (normalement)

Accès aux propriétés

```
1 <?php
2 class Personnel {
3     const AGE_MIN = 14;
4     const AGE_MAX = 65;
5
6     // -- Propriétés --
7     private $_nom="";
8     private $_prenom="";
9     private $_age=0 ;
10
11     // Getter
12     public function getNom() {
13         return $this->_nom;
14     }
15     public function getPrenom() {
16         return $this->_prenom;
17     }
18     public function getAge() {
19         return $this->_age;
20     }
21
22     // Setter
23     public function setNom($n) {
24         $this->_nom = $n;
25     }
26     public function setPrenom($p) {
27         $this->_prenom = $p;
28     }
29     public function setAge($a) {
30         if (($a>=self::AGE_MIN)&&($a<self::AGE_MAX)) {
31             $this->_age=intval($a);
32         }
33     }
34
35     // Methodes
36
37     public function AffichePersonne() {
38         echo "Personne : ";
39         echo $this->_nom."\t".$this->_prenom."\t".$this->_age;
40         echo PHP_EOL;
41     }
42 }
43
44 // Programme
45 $salarie = new Personnel();
46 $salarie->setNom("dupont");
47 $salarie->setPrenom("jean");
48 $salarie->setAge("25");
49 echo $salarie->getNom()." ";
50 echo $salarie->getPrenom()." ";
51 echo $salarie->getAge().PHP_EOL;
52 ?>
```

Héritage

Lorsqu'on dit que la classe Commercial **hérite** de la classe Personnel, c'est que la classe Commercial **hérite** de tous les attributs et méthodes de la classe Personnel.

Si l'on déclare des méthodes dans la classe Personnel, et qu'on crée une instance de la classe Commercial, alors on pourra appeler n'importe quelle méthode déclarée dans la classe Personnel **du moment qu'elle est publique**.

```
1  <?php
2  class Personnel {
3      protected static $NbPersonne ;
4
5      // -- Propriétés --
6      private $_nom;
7      private $_prenom;
8      private $_age;
9
10     public function __construct($n, $p, $a) {
11         $this->_nom = $n;
12         $this->_prenom = $p;
13         $this->_age = $a;
14         self::$NbPersonne++;
15     }
16
17     // Methode statique
18     static function NbrPersonne() {
19         return self::$NbPersonne;
20     }
21     // Methodes
22
23     public function AffichePersonne() {
24         echo self::$NbPersonne." Personne : ";
25         echo $this->_nom."\t".$this->_prenom."\t".$this->_age;
26         echo PHP_EOL;
27         echo "<br/>";
28     }
29 }
30
31 class Commercial extends Personnel {
32     private $_CArealiser;
33
34     public function __construct($n, $p, $a, $ca) {
35         parent::__construct($n, $p, $a);
36         $this->_CArealiser = $ca;
37     }
38
39     public function AffichePersonne() {
40         parent::AffichePersonne();
41         echo $this->_CArealiser;
42     }
43 }
44 // Programme
45 $salarie = new Personnel("Combette","Roland",50);
46 $salarie1 = new Commercial("Ferreira","Antoine",32,5000);
47 echo $salarie1->AffichePersonne();
48 ?>
```

Exercices

Exercice 1.4 : Ecrire une classe **Rectangle** comme héritant de la classe **Geometrie** (**voir ci dessous**). Un Rectangle est caractérisé par une longueur et une largeur (tous deux des nombres réels). Faire en sorte que la classe redéfinisse correctement la méthode aire et possède un constructeur approprié.

Ajouter à la classe **Rectangle** les méthodes *getLongueur* et *getLargeur* qui retournent respectivement la longueur et la largeur du Rectangle.

```
class Geometrie {  
    private $_aire=0;  
  
    public function aire() {  
        return $this->_aire;  
    }  
}
```

Abstraction

- On peut avoir des classes abstraites. Cela veut dire qu'elle ne peuvent servir à instancier un objet. Elles ont un rôle de modèle lors de l'héritage.

abstract class Personnage

- On peut avoir des méthodes abstraites. Dans la même idée une méthode abstraite à son corps vide. Elle oblige le développeur à écrire cette méthode lors de l'héritage.

abstract public function()

- **Dès qu'une classe à au moins une méthode abstraite elle devient abstraite.**

Exercices

Exercice 1.5 : Ecrire une classe **Cercle** comme héritant de la classe **Géométrie**.

La classe géométrie devient *abstraite*.

Un **Cercle** est caractérisé par un *Point central* et un *rayon* (nombre réel).

Faire en sorte que la classe Cercle redéfinisse correctement la méthode aire et possède un constructeur approprié.

Modifier votre classe Point en la mettant abstraite. Que se passe-t-il ?

Exercices

Exercice 1.5bis : Ne rien changer aux classes mais l'on veut que les classes soient dans des fichiers indépendant donc l'extension est `.class.php`.
On devra alors utiliser **require** dans le programme test qui peut s'appeler `POOFigures15bis.php`

Exercices

Exercice 1.5ter: Si on a 30 ou 40 classes différentes ce qui peut être assez courant cela devient fastidieux de faire pour chaque classe un *require*. Il existe une manière de les charger automatiquement lors qu'on **instancie** une classe. En cherchant sur le web essayer d'implémenter cette solution.

Exercices

Exercice 1.6 :

Rendre la méthode aire de la classe Geometrie abstraite.

Les constantes de classe

- Comme le nom l'indique ces constantes appartiennent à la classe et non aux objets.
- On utilise l'opérateur de portée `::` pour y accéder
- Une constante de classe se déclare avec l'opérateur **const**
- Pour y accéder depuis l'intérieur de la classe on utilise **self::**
- Pour y accéder depuis l'extérieur de la classe on utilise **Classe::CONST**

Les constantes de classe

`$perso = new Personnage(Personnage::FORCE_MOYENNE);`

```
<?php
class Personnage
{
    private $_force;
    private $_localisation;
    private $_experience;
    private $_degats;

    // Déclarations des constantes en rapport avec la force.

    const FORCE_PETITE = 20;
    const FORCE_MOYENNE = 50;
    const FORCE_GRADE = 80;

    public function __construct($forceInitiale)
    {
        // N'oubliez pas qu'il faut assigner la valeur d'un attribut uniquement depuis son
        // setter !
        $this->setForce($forceInitiale);
    }

    public function deplacer()
    {
    }

    public function frapper()
    {
    }

    public function gagnerExperience()
    {
    }

    public function setForce($force)
    {
        // On vérifie qu'on nous donne bien soit une « FORCE_PETITE », soit une « FORCE_MOYENNE
        // », soit une « FORCE_GRADE ».
        if (in_array($force, [self::FORCE_PETITE, self::FORCE_MOYENNE, self::FORCE_GRADE]))
        {
            $this->_force = $force;
        }
    }
}
```

Méthodes statiques

- Une méthode statique dépend de la classe et non du ou des objets.
- Pour l'appeler de l'extérieur de l'objet :
Classe::methode()
Exemple : **Personnage::parler()**
- Pour l'appeler de l'intérieur de l'objet :
self::parler()

Méthode static

```
1 <?php
2 class Personnel {
3     protected static $NbPersonne ;
4
5     // -- Propriétés --
6     private $_nom;
7     private $_prenom;
8     private $_age;
9
10    public function __construct($n, $p, $a) {
11        $this->_nom = $n;
12        $this->_prenom = $p;
13        $this->_age = $a;
14        self::$NbPersonne++;
15    }
16
17    // Methode statique
18    static function NbrPersonne() {
19        return self::$NbPersonne;
20    }
21    // Methodes
22
23    public function AffichePersonne() {
24        echo self::$NbPersonne." Personne : ";
25        echo $this->_nom."\t".$this->_prenom."\t".$this->_age;
26        echo PHP_EOL;
27        echo "<br/>";
28    }
29 }
30
31 // Programme
32 $salarie = new Personnel("Combette","Roland",50);
33 $salarie1 = new Personnel("Fereira","Antoine",32);
34 echo Personnel::NbrPersonne();
35 ?>
```

Méthode statique

```
<?php
class Personnage
{
    private $_force;
    private $_localisation;
    private $_experience;
    private $_degats;

    const FORCE_PETITE = 20;
    const FORCE_MOYENNE = 50;
    const FORCE_GRADE = 80;

    public function __construct($forceInitiale)
    {
        $this->setForce($forceInitiale);
    }

    public function deplacer(){}

    public function frapper(){}

    public function gagnerExperience(){}

    public function setForce($force)
    {
        // On vérifie qu'on nous donne bien soit une « FORCE_PETITE », soit une « FORCE_MOYENNE », soit une « FORCE_GRADE ».
        if (in_array($force, [self::FORCE_PETITE, self::FORCE_MOYENNE, self::FORCE_GRADE]))
        {
            $this->_force = $force;
        }
    }

    // Notez que le mot-clé static peut être placé avant la visibilité de la méthode (ici c'est public).
    public static function parler()
    {
        echo 'Je vais tous vous tuer !';
    }
}
```

```
$perso = new Personnage(Personnage::FORCE_GRADE);
echo Personnage::parler();
```

Attributs statiques

- Un attribut statique dépend de la classe et non du ou des objets.
- Pour l'appeler de l'extérieur de l'objet :
Exemple : **Personnage::\$monAttribut**
- Pour l'appeler de l'intérieur de l'objet :
self::\$monAttribut

Attribut static

```
1 <?php
2 class Personnel {
3     protected static $NbPersonne ;
4
5     // -- Propriétés --
6     private $_nom;
7     private $_prenom;
8     private $_age;
9
10    public function __construct($n, $p, $a) {
11        $this->_nom = $n;
12        $this->_prenom = $p;
13        $this->_age = $a;
14        self::$NbPersonne++;
15    }
16
17    // Methodes
18
19    public function AffichePersonne() {
20        echo self::$NbPersonne." Personne : ";
21        echo $this->_nom."\t".$this->_prenom."\t".$this->_age;
22        echo PHP_EOL;
23        echo "<br/>";
24    }
25 }
26
27 // Programme
28 $salarie = new Personnel("Combette","Roland",50);
29 $salarie->AffichePersonne();
30 $salarie1 = new Personnel("Fereira","Antoine",32);
31 $salarie1->AffichePersonne();
32 ?>
```

Exercices

Exercice 1.7 : Ecrire une classe Vehicule qui possède comme propriétés démarré (boolean), une vitesse actuelle et une vitesse maximum pour le véhicule. Cette classe à également 4 méthodes abstraites :

- **demarrer()**
- **eteindre()**
- **ralentir(valeur)**
- **accelerer(valeur)**

Construire une classe Voiture qui hérite de Véhicule. Implémenter les méthodes en essayant d'être le plus près de la réalité. Par exemple :

- **on ne peut accélérer si le moteur n'est pas allumé.**
- **on ne peut ralentir de 30 km/h si on est à 20 km/h.**
- **lorsqu'on accélère on ne peut dépasser la vitesse maximum.**
- **implémenter la méthode de classe (méthode statique) qui donne le nombre de voiture instanciés. On suppose qu'on ne détruit pas de véhicule.**
- **implémenter une constante de classe VITESSE_MAX = 260 qui interdit d'instancier un véhicule avec une vitesse supérieur.**

Pour ceux qui veulent pousser plus loin : on peut interdire ou limiter l'accélération à 20 km/h par appel de fonction ou à 20 % de la vitesse courante.