PHP - Programmation Orientée Objet (POO)

A - Modèle objet

A-1 – Introduction

Jusqu'à présent nous avons programmé en procédurale. C'est à dire que les « commandes » de votre programme se sont e*xécutées les unes à la suite des autres*.
Vos programmes contiennent des fonctions (méthodes) qui s'exécutent au fur et à mesure selon le plan de vos algorithmes.
Vous avez manipulés des variables et des méthodes indépendantes les unes des autres.

Ex une variable $prixHt
une méthode ReduireFacture($var1)
une variable $poids, ...
on peut très bien écrire ReduireFacture($prixHt) ou ReduireFacture($poids)

Cette dernière méthode bien que syntaxiquement correcte ne l'est pas au niveau logique. Votre IDE ou compilateur (en prog. Procédurale) ne vous indique que les erreurs syntaxiques, la programmation orientée objet vous permet d'avoir un niveau de sécurité supplémentaire et vous évite certaines erreurs logiques.

Maintenant vous allez manipuler des ensembles que l'on nomme des classes.

Une classe est tout simplement un moule pour faire des objets.

exemple les classes facture, voiture, bateau, commande, ...

Une classe est composé de membres ; parmi ces membres, on dispose de champs (les variables qui lui sont caractéristiques), de méthodes, de propriétés, ainsi que d'autres éléments que nous verrons plus tard.

A-2 – Terminologie

Une classe est une représentation abstraite d'un objet.
C'est la somme des propriétés de l'objet et des traitements dont il est capable.
Une Chaise, un Livre, un Humain, une Rivière sont autant d'exemples possibles de classes.
Une classe peut généralement être rendu concrète au moyen d'**une instance de classe**, que l'on appelle **objet** :
on parle d'**instancier la classe**. **Un objet** est donc un exemple concret de la définition abstraite qu'est la classe.

Exemple : la classe train, une instance de cette classe pourrait être l'objet train888

A-3 Syntaxe

C'est dans la classe que sont définis les membres (dont les champs et les méthodes).
Tout objet créé à partir d'une classe possède les membres que propose cette classe, vous comprenez donc pourquoi je parle de "moule à objet".

L'objet n'est pas seulement une somme d'informations, l'objet réagit. Cela n'est possible que s'il est muni d'opérations - méthodes- qui seront déclenchées en réponse à des sollicitations extérieures - d'un autre objet -. On traduit ce processus en introduisant la notion de message; la réception d'un message va engendrer la réaction de l'objet. Bien sûr il faut que l'objet possède dans son comportement une action capable de traiter le message.

exemple le cas du train
un train peut recevoir des messages stopper, ralentir, accélérer, des actions spécifiques seront déclenchées, par contre le message voler sera rejeté. Alors quand prog. procédurale rien n'empêche d'appeler une méthode voler alors que nous avons un train.

Une classe (en UML nous le verrons plus tard) est représentée par un rectangle, sa partie haute est réservée à son nom, la zone du milieu contient les attributs et la partie basse les opérations.

<Nom de la Classe>
Attributs
Méthodes

Exemples

Train
Vitesse EnMarche
Stopper() Ralentir() Accélerer()

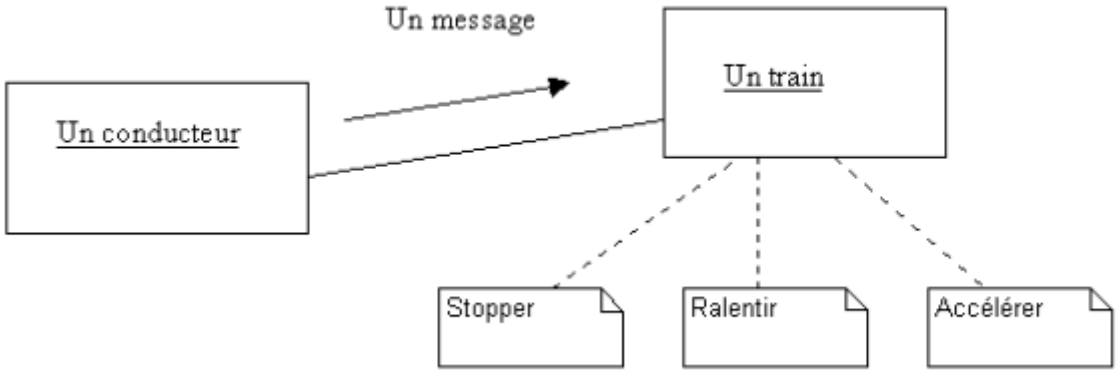
Classe train

En programmation objet, les opérations sont appelées méthodes.
En utilisant le langage algorithmique, nous pouvons écrire:

```
Classe Train
    Attributs:
      Vitesse : Entier
      EnMarche : Booleen
    Méthodes:
      Procédure Stopper()
      Procédure Ralentir( Donnée v : Entier )
      Procédure Accélerer( Donnée v : Entier)
FinClasse
```

```
public class Train ()          //ne vous souciez pas des mots private et public nous le
    {                           //verrons plus loin
    private $vitesse ;
    private $enMarche ;

    public function Stopper(){...}
    public function  Ralentir(v){...}
    public function  Accelerer(v){...}
    }
```



Ainsi un train peut recevoir des messages stopper, ralentir, accélérer, des actions spécifiques seront déclenchées, par contre le message voler sera rejeté

nous utilisons quatre éléments empruntés d'UML.
- **Un objet** est représenté par un rectangle, son nom est souligné:
- Le message est représenté par une flèche orienté.
- Le trait liant les deux objets symbolise une relation.
- La note représentée par un rectangle corné permet de préciser une notion.

Donc un objet est une instance d'une classe.
Après son **instanciation** l'objet bénéficie du comportement et des attributs de sa classe.

$train888 = **new** Train();

le mot **new** est un mot réservé , à n'utiliser que pour instancier une classe.

Après son **instanciation** l'objet bénéficie du comportement et des attributs de sa classe.

```
$train888->Ralentir( 50 ) ;
$train888->Stopper() ;
```

Notez la syntaxe: <Nom de l'objet> **->** Méthode

$this->vitesse ;

Notez la syntaxe: $this **->** variable sans le signe $

L'accès aux propriétés et méthodes de l'objet se fait par la flèche "->"

Exemple saisissez le code suivant :

```
<?php

class train{
	public $vitesse;

	public function Ralentir($vit){
	$this->vitesse-=$vit;		//équivalent à $this->vitesse=$this->vitesse-$vit
	return $this->vitesse;
	}

	public function Accelerer($vit){
	$this->vitesse+=$vit;		//équivalent à $this->vitesse=$this->vitesse-$vit
	return $this->vitesse;
	}

	public function Stopper(){
	$this->vitesse=0;
	}

}

$train888 = new train();		// on instancie un objet $train888

echo 'Bienvenue en gare des BTS, destination SIO, attention au départ<br/>';
```

```
echo $train888->Accelerer(10).'<br/>';
echo $train888->Accelerer(10).'<br/>';
echo $train888->Accelerer(10).'<br/>';

echo 'notre vitesse actuellement est de '.$train888->vitesse.' km/h';
echo '<br/>et le temps passe....';
echo '<br/>nous arrivons à destination<br/>';

echo $train888->Ralentir(10).'<br/>';
echo $train888->Ralentir(10).'<br/>';
echo $train888->Ralentir(10).'<br/>';
echo 'notre vitesse actuellement est de '.$train888->vitesse.' km/h<br/>';
echo 'nous sommes arrivé';

?>
```

TP

Améliorez le code ci-dessus :

-rajouter à la classe train la variable enMarche de type booléen.

Public $enMarche ; //état allumé ou éteint

-initialisez la vitesse à 0 dès sa déclaration.

-vérifiez que vous êtes à l'arrêt pour monter ou descendre.

-ne pas ralentir plus que possible sinon la vitesse sera négative.

-ne pas dépasser 100km/h.

-mettre un obstacle en milieu de route pour stopper net, puis repartir.

Correction

```
<?php


class train{
        public $vitesse=0;
        public $enMarche=False;

        public function Ralentir($vit){
        if((($this->vitesse-$vit)>=$vit) and $this->enMarche==True)
{               // pour ne pas avoir une vitesse négative
                 $this->vitesse=$this->vitesse-$vit;
        }else{$this->vitesse=0;
         $this->enMarche=False;}

        return $this->vitesse;
        }

        public function Accelerer($vit){
        if(($this->vitesse+$vit)<=100){       // on ne dépasse pas 100km/h
        if ($this->enMarche==False)$this->enMarche=True; }
                                // on confirme que la loco est allumée
        $this->vitesse=$this->vitesse+$vit;
        }else{$this->vitesse=100;}
        $this->enMarche=True;
        return $this->vitesse;
        }

        public function Stopper(){
        $this->enMarche=False;
        $this->vitesse=0;
        }

}


$train888 = new train();

if ($train888->enMarche==False) {         // il faut bien vérifier que le train est à
                                           //quai!
echo 'Bienvenue en gare des BTS destination SIO, attention au départ<br/>';
}
echo $train888->Accelerer(10).'<br/>';
echo $train888->Accelerer(10).'<br/>';
echo $train888->Accelerer(10).'<br/>';
echo $train888->Accelerer(60).'<br/>';     // ca y est le chauffeur s'excite
echo $train888->Accelerer(120).'<br/>';    // là il dort avec le pied sur
l'accélerateur
```

```
echo $train888->Accelerer(120).'<br/>';

echo 'notre vitesse actuellement est de '.$train888->vitesse.' km/h';
echo '<br/>et le temps passe....';
echo '<br/>une vache sur les rails!!!!!!!';
$train888->Stopper();                    //oufff
echo '<br/>elle est partie';
echo '<br/>on redémarre<br/>';
echo $train888->Accelerer(10).'<br/>';
echo $train888->Accelerer(20).'<br/>';
echo $train888->Accelerer(30).'<br/>';
echo $train888->Accelerer(60).'<br/>';

echo '<br/>nous arrivons à destination<br/>';

echo $train888->Ralentir(10).'<br/>';
echo $train888->Ralentir(20).'<br/>';
echo $train888->Ralentir(40).'<br/>';
echo $train888->Ralentir(60).'<br/>';  // ce conducteur n'est pas génial!!

echo 'notre vitesse actuellement est de '.$train888->vitesse.' km/h<br/>';
if ($train888->enMarche==False) {
echo 'nous sommes arrivé on peut descendre';}  // il faut bien vérifier que le
train est à quai!


?>
```

A-4 La portée

Comme vu plus haut en programmation procédurale on peut accéder de n'importe ou dans le code à une variable et la manipuler comme on le souhaite du moment que l'on respecte la portée de cette dernière.
La POO a rajoutée un niveau de plus de contrôle, l'**encapsulation.**
Au principe de portée, elle rajoute « des droits d'accès ou visibilité ».

Généralement on compte trois niveaux de visibilité - du plus restrictif au plus accessible :

Privé (private) : rend l'élément visible à la classe seule.

Protégé (protected) : rend l'élément visible aux sous-classes de la classe (la notion de sous-classe sera abordée plus loin avec l'héritage)

Public : rend l'élément visible à tout client de la classe.

Dans la classe train on remarque que la variable vitesse ne nécessite pas d'être

accessible de l'extérieur de la classe
donc la classe devient :

```
class train{
	private $vitesse;
	private $enMarche ;

	public function Ralentir($vit){
	$this->vitesse-=$vit;		//équivalent à $this->vitesse=$this->vitesse-$vit
	return $this->vitesse;
	}

	public function Accelerer($vit){
	$this->vitesse+=$vit;		//équivalent à $this->vitesse=$this->vitesse-$vit
	return $this->vitesse;
	}

	public function Stopper(){
	$this->vitesse=0;
	}

}
```

ainsi essayez ce code

```
$train888 = new train();
echo $train888->vitesse;
```

il vous renverra une erreur
Fatal error: Cannot access private property train::$vitesse...

Par convention et pour des mesures de sécurité il faut toujours mettre les variables d'une classe comme privée (private), on les encapsulent dans la classe.

L'**encapsulation** représente un des concept fondamentaux du monde objet. Par principe même, on ne peut accéder aux attributs ou opérations d'un objet en dehors de ceux explicitement déclarés public par la classe. Ainsi une classe n'est utilisable qu'au travers son **interface** c'est à dire l' ensemble des attributs et opérations publics.

En UML le niveau de visibilité est symbolisé par les caractères +, # et -, qui correspondent respectivement aux niveaux **public**, **protégé**, et **privé** (notation utilisée par la méthode UML).

Train
-vitesse -enMarche
+Stopper() +Ralentir() +Accélerer()

La donnée vitesse est encapsulée dans la classe Train, elle n'est pas accessible directement, de même pour l'état de marche.

Un conducteur envoie le message Stopper à son train, mais c'est de la responsabilité du train de s'arrêter, de plus le conducteur n'a pas besoin de savoir comment le train s'arrête (la même chose pour vous et votre véhicule !).

A-4-1 Les accesseurs et mutateurs

Non ceux ne sont pas des extra-terrestres mutants, mais des méthodes. Comme on le dit plus haut les variables doivent être déclarées « private » pour des raisons de sécurité et ainsi respecter le principe d'encapsulation.
Seule la classe peut les lire et les modifier.
La seule exception tolérée et selon la situation (selon les spécifications de votre programme).

A-4-1-1Les accesseurs (getters)

Ces méthodes ont un nom bien spécial : ce sont des **accesseurs (ou *getters*)**. Par convention, ces méthodes portent le même nom que l'attribut dont elles renvoient la valeur.
Exemple :

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

public  function EnMarche(){
        return $this->enMarche;
        }
```

A-4-1-2Les Mutateurs (setters)

On vous rappelle que le principe d'encapsulation est là pour vous

empêcher d'assigner un mauvais type de valeur à un attribut : si vous demandez à votre classe de le faire, ce risque est supprimé car la classe « contrôle » la valeur des attributs. La classe doit impérativement contrôler la valeur afin d'assurer son intégrité.
Et pour cela on a mis en place les mutateurs ou setters.
Ces méthodes sont de la forme *setNomDeLAttribut().*
Exemple pour le cas du train la vitesse doit être un entier positif

```
public function setVitesse($vit){
        $vit=(int)$vit ;                    //cast pour le type entier
        $vit=abs($vit) ;                    // empêche une vitesse négative
        $this->vitesse=$vit ;
        }
```

A-5 Le constructeur

L'approche objet propose des méthodes particulières qui ont comme rôle d'initialiser les attributs, ce sont les constructeurs.

Un constructeur est appelé automatiquement au moment de la création de l'objet à l'aide de l'opérateur new.

Les constructeurs ont une signature particulière, en php.

visibilité `function __construct()`. //On les reconnaît facilement car leur nom commence par deux underscores

Le constructeur peut ou ne pas prendre de paramètres et n'est pas obligatoire dans la déclaration d'une classe.

exemple :

```
class train{

private vitesse ;
private enMarche ;

 public function __construct($etat){
      $this->enMarche=$etat ;
      }
.....
}
$train888 = new train(True) ;
```

Si on ne déclare pas de constructeur php se charge automatiquement d'en crée un par défaut.

Ce constructeur par défaut va déclarer toutes les variables mais non les initialiser.
Exemple de constructeur par défaut crée par php.

```
public function __construct(){
      $this->vitesse='';
      $this->enMarche='';  // php ne connait pas le type des variables
     }
```

Il est fortement conseiller de toujours fournir au moins un constructeur à chaque classe.

A-6 Le destructeur

La destruction d'une instance provoque systématiquement l'appel du *destructeur* de la classe (la méthode d'instance `__destruct`). Le rôle avoué du destructeur est de pouvoir détruire proprement l'objet, par exemple en fermant les ressources utilisées par l'instance (comme un fichier ouvert avec fopen par exemple). Comme pour le constructeur, cette méthode n'est pas obligatoire. Dans la réalité, les usages du destructeur sont assez rares.

Exemple :

```
<?php

  class MaClasse {

      protected $ressource;

      function __construct ($fichier) {
          $this->ressource = fopen($fichier, 'r');
      }

     function lireUneLigne () {
         return fgets($this->ressource);
     }

     function __destruct () {
        // fermer proprement le fichier
        fclose($this->ressource);
     }
 }

 $mon_instance = new MaClasse('fichier.txt');
 echo $mon_instance->lireUneLigne(); // lit la première ligne
 echo $mon_instance->lireUneLigne(); // lit la seconde ligne
unset($mon_instance); // le destructeur est appellé, le fichier est fermé

 ?>
```

TP Compte bancaire

Soit la classe compteBancaire
Chaque compte bancaire possède :
-un numéro de compte (Trois premières lettres du nom puis heure et secondes en cours) ex : mat1545 issu de mathieu et 15h45
-un solde (positif ou négatif)
-un titulaire, son nom (maximum 25 caractères)
-decouvert, somme max en deça du solde 0 (par défaut -500)

Cette classe possède aussi des méthodes
-crediter pour ajouter une somme (positive sur le compte)
-debiter pour retirer de l'argent

Attention les opérations, de débit en particulier, ne doivent modifier le solde que si on le peut.
Ex : solde de 10€, découvert de 100, tentative de retrait de 200€, On a une opération impossible donc « niet » et message produit :
vous ne pouvez retirer que 110€ maximum.

L'affichage des messages doit être réalisé par une méthode particulière qui se chargera de l'affichage :

```
public function affichage($origin, $msg){
}
```

exemple pour Robert qui réalise un retrait de 500€

```
affichage ('debit', '500') ;
```

produira le message

Vous avez débitez votre compte de 500 €.
Vous êtes Robert votre compte est le rob1542, vous avez un solde après débit de 1420€ et votre découvert autorisé est de 200€

1) créez cette classe (constructeur prenant comme paramètre le nom)
2) Donnez un solde au compte en le créditant de 1500.
3) Débitez le de 450.
4) Créditez le de 25.
5) Débitez le de 2500.
6) Modifier le découvert à 2500.
7) Débitez le de 1500.

NB : pour obtenir l'heure et les secondes

```
$now=date('Hi');
```

Correction

```
<?php
class compteBancaire{



private $numCompte ;
private $solde ;
private $decouvert ;
private $titulaire ;

        public function __construct($nom){  //le constructeur

        $this->titulaire= substr($nom,0,25);
        $this->numCompte=substr($nom,0,3);
        $now=date('Hi');
        $this->numCompte.=$now;
        $this->solde=0;
        $this->decouvert=500;
        }

                // getters/accesseurs

        public function Titulaire(){ return $this->titulaire;}
        public function NumCompte() {return $this->numCompte;}
        public function Decouvert() {return $this->decouvert;}
        public function Solde() {return $this->solde;}

                // setters/mutateurs

        public function setDecouvert($val) {
        $this->decouvert=$val;}

        public function affichage($origin, $msg){
        $affiche='vous avez ';
        $finaffiche=' sur votre compte '.$msg.' &euro;<br/>';
        $finaffiche.='Vous &ecirc;tes '.$this->titulaire.' votre compte est le ';
        $finaffiche.=$this->numCompte;
        $finaffiche.=', vous avez un solde actuel de ';
        $finaffiche.=$this->solde;
        $finaffiche.=' &euro; et votre d&eacute;couvert autoris&eacute; est de '.
$this->decouvert.' &euro;';

        switch ($origin){
                case"debit":
                        $affiche.='d&eacute;bit&eacute;';
                        $affiche.=$finaffiche;
```

```
                        break;
                case"credit":
                        $affiche.='cr&eacute;dit&eacute;';
                        $affiche.=$finaffiche;
                        break;
                case"supdecouvert":
                        $affiche='Vous ne pouvez retirer que ';
                        $affiche.=$this->decouvert+$this->solde;
                        $affiche.='&euro; et non '.$msg.'&euro;';

                        break;
                default :
                        $affiche='Erreur interne m&ethode incorrecte: '.$origin;
                        $affiche.='Vous avez saisi'.$msg;
                        }

                echo $affiche.'<br/><br/>';
        }

        public function crediter($somme){

                if ($somme>0){
                $this->solde+=$somme;
                $this->affichage('credit',$somme);
                }else{ $this->affichage('cr&eacute;dit n&eacute;gatif',$somme);}
        }

        public function debiter($somme){

                if ($somme>0){
                $soldemini=($this->decouvert + $this->solde);
                $soldedemande=($this->solde+$somme);
                        if($soldemini>=$soldedemande){
                        $this->solde-=$somme;
                        $this->affichage('debit',$somme);
                        }else{$this->affichage('supdecouvert',$somme);}


                }else{ $this->affichage('d&eacute;bit n&eacute;gatif',$somme);}
        }

}

 $compteR = new compteBancaire('Robert');
$compteR->crediter(1500);
$compteR->debiter(450);
```

```
$compteR->crediter(25);
$compteR->debiter(2500);
$compteR->setDecouvert(2500);
$compteR->debiter(1500);


?>
```