Cours Programmation ObjetS en PHP

Ph. Declerck

1.Généralités

a. Motivation

Terminologie

Classe : modèle d'objets composé de données (attributs, propriétés) et de fonctions (méthodes)
Objet : une instance de la classe (création d'objets à partir du moule qui peut donc en avoir plusieurs)
Chaque objet transporte avec lui ses données qui pourront être traitées par les fonctions (méthodes) de la classe de l'objet.

L'intérêt de l PO est présent quand :
- à une classe correspond plusieurs objets.
- Une "classification" d'objets existe.
- Un objet transporte un ensemble de données personnelles
- Un objet a un ensemble d'actions réservées(fonctions, méthodes)

Avant de faire de la PO, vérifiez que cela présente un intérêt pour le problème considéré, c'est à dire la présence naturelle de plusieurs objets pour chaque classe et/ou l'existence d'un héritage de classes ! Si le programme se résume à un seul objet déduit d'une classe unique (généré per l'instruction NEW) qui contient plusieurs méthodes qui semblent complexes, on peut s'interroger sur la pertinence du choix du langage Objet : est-ce de la PO ou du langage Objet qui est programmé de manière procédurale ?...

Domaines potentiels : comptes bancaires, abonnés d'une bibliothèque, classification d'espèces animales en zoologie, gestion d'images, ...

Par rapport à l'AF, chaque classe correspond à une boîte. Chaque boîte contient des données (appelées ici propriétés) et des activités (appelées ici méthodes). Un objet peut être vu comme un exemplaire d'un mécanisme des actigrammes. L'arbre hiérarchique est remplacé par la notion d'héritage où on "hérite" de la classe de niveau supérieur : le parent de deux enfants représente la partie commune et pas une fonction plus générale. Le modèle entités/relations (renommé diagramme de classe en UML) représente les échanges entre les objets de classe différente.

b. Langages JavaScript et PHP

Le langage JavaScript utilise des objets dans un arbre complexe mais n'a pas la notion de classe (jusqu'à version 1.5 de Netscape). L'instruction NEW n'existe que pour les tableaux. Ce

n'est donc pas réellement un langage Objet mais plutôt événementiel.

Nouveau en PHP4, la programmation orientée Objets (plutôt Classes/Objets) s'améliore grossièrement en PHP5. Il faudra cependant s'attendre à des changements des instructions dans les nouvelles versions.

Limites
- L'héritage est unique et pas multiple en PHP4 et 5.
- Constructeurs d'objets : nouveau en PHP5
- Destructeurs d'objets : à voir
- Classes abstraites et interfaces : nouveau en PHP5.
- variables PUBLIC, PRIVATE,... : tout est public en PHP4 ; nouveau en PHP5.

AVERTISSEMENT : les informations disponibles dans les livres sur PHP5 ne sont pas cohérent à 100 % avec les tests avec EasyPhp (Ex : livre PHP5 de Eric Daspet et Cyril Pierre de Geyer) et il vaut mieux tester les instructions avant exploitation.

2. Les classes

```
<?PHP
//============================== début de la class
CLASS voiture
{ //----attributs----------
PUBLIC $marque=99 ;
PUBLIC $v ;
//PUBLIC $t ;  // pas indispensable

//----Méthodes--------------
FUNCTION klaxonner ()
{ echo "Elle klaxonne !"."<BR>" ;
}

FUNCTION avancer ($t)
{ $d=$this->v * $t ;  // marche pas avec voiture
echo "Elle avance de ".$d." km <BR>" ;
}

} //============================== fin de la class

$mavoit= NEW voiture() ;  // création de l'objet
echo "marque=".$mavoit->marque."<BR>" ;  // $ // valeur de
l'initialisation
$mavoit->klaxonner () ;

$mavoit->v=100 ; //
echo "vitesse=".$mavoit->v."<BR>" ;

$mavoit->avancer (3) ;

$mavoit->t=5 ; //
echo "temps=".$mavoit->t."<BR>" ;
$mavoit->avancer ($mavoit->t) ;

//echo "distance=".$mavoit->d."<BR>" ; //inaccessible

?>
```

3. Héritage

```
///============================== début de la class GRAND-PARENT
CLASS transport
{ //----attributs----------
PUBLIC $v ;
FUNCTION bouger ($s)
{ echo "Le GRAND-PARENT bouge de ".$s." !<BR>" ;}
} //============================== fin de la class  GRAND-PARENT

//============================== début de la class PARENT
CLASS vehicule EXTENDS transport
{ //----attributs----------
PUBLIC $marque=99 ;
PUBLIC $v ;
//PUBLIC $t ;  // pas indispensable
FUNCTION avancer ($t)
{ $d=$this->v * $t ;  // marche pas avec voiture
echo "Le PARENT avance de ".$d." km <BR>" ;
}
} //============================== fin de la class  PARENT

//============================== début de la class ENFANT
CLASS voiture EXTENDS vehicule
{ //----attributs----------
//----Méthodes-------------
FUNCTION klaxonner ()
{ echo "Elle klaxonne !"."<BR>" ;
echo "-------Dans enfant<BR>" ;
parent::avancer (4) ;
vehicule::avancer (4) ;
transport::bouger (18) ;
echo "-------<BR>" ;
}
} //============================== fin de la class ENFANT
//============principal==============================
$mavoit= NEW voiture() ;  // création de l'objet
echo "marque=".$mavoit->marque."<BR>" ;  // $

$mavoit->v=33 ; //
$mavoit->klaxonner () ;

$mavoit->v=100 ; //
echo "vitesse=".$mavoit->v."<BR>" ;

$mavoit->avancer (3) ;

$mavoit->t=5 ; //
echo "temps=".$mavoit->t."<BR>" ;
$mavoit->avancer ($mavoit->t) ;
//echo "distance=".$mavoit->d."<BR>" ; //inaccessible
```

4.Accès

a) Accès aux méthodes et attributs

Accès public : PUBLIC
La méthode ou l'attribut est accessible depuis toute l'application. C'est le cas par défaut. On peut le préciser explicitement avec le mot-clé PUBLIC.

Accès privé : PRIVATE
La méthode ou l'attribut n'est accessible qu'à l'intérieur de la classe. On le précise avec le mot-clé PRIVATE.

Accès protégé : PROTECTED
La méthode ou l'attribut n'est accessible qu'à l'intérieur de la classe et de ses classes dérivées. On le précise avec le mot-clé PROTECTED.

```
//============================== début de la class PARENT
CLASS vehicule
{ //----attributs----------
PUBLIC $marque=99 ;
PROTECTED $v=100 ;
//----Méthodes-------------
} //============================== fin de la class  PARENT

//============================== début de la class ENFANT
CLASS voiture EXTENDS vehicule
{ //----attributs----------
PRIVATE $vv=30 ;
PUBLIC FUNCTION avancer ()
{ echo "avance à la vitesse de ENFANT de".$this->vv ."km/h"."<BR>"
;  // OK car PRIVATE
echo "avance à la vitesse du PARENT de".$this->v ."km/h"."<BR>" ;
//OK car PROTECTED
}
} //============================== fin de la class  ENFANT

$mavoit= NEW voiture() ;  // création de l'objet
echo "marque=".$mavoit->marque."<BR>" ;  // $

$mavoit->avancer () ;  // OK
//echo "avance à la vitesse de".$mavoit->vv ."km/h"."<BR>" ;
//NON
//Fatal error: Cannot access private property voiture::$vv

//echo "avance à la vitesse du PARENT de".$mavoit->v
."km/h"."<BR>" ;  //NON
//Fatal error: Cannot access protected property voiture::$v
```

b.Accès statique

Accès direct aux données de la classe (statique) sans création d'objet.

```
//============================== début de la class
CLASS voiture
{ //----attributs----------
CONST pi=3.14 ;
STATIC $roues=4 ;
//----Méthodes-------------
FUNCTION compter()
{echo "nb roues interne à la classe ".self::$roues."<BR>" ;
RETURN self::$roues ;
}
} //============================== fin de la class

echo "La constante pi =".voiture::pi."<BR>" ;
echo "La nb de roues ext avec variable =".voiture::$roues."<BR>" ;

echo "nombre de roues ext  avec
fonction=".voiture::compter()."<BR>" ;
```

5. Signe égal

Il faut faire la différence entre 2 cas :
1) 2 objets qui ont les mêmes attributs. Ex. 2 chapeaux achetés ayant la même marque, couleur, forme,...
Le clonage crée un deuxième objet différent mais identique pour les attributs. En PHP4, c'est un simple signe "=". En PHP5, c'est "CLONE".
Le test d'égalité : "==" (2 signes "=")

2) 2 objets qui sont en réalité un seul. Ex. Mon compte en banque que j'alimente sur le web et le compte en banque à mon nom que le banquier gère. Une modification (retrait, frais de gestion,..) va apparaître sur les deux.
Un simple signe "=" fait un autre nom pour l'objet (on parle aussi d'aléas). Cela permet d'écrire de manière plus courte le nom d'un objet.
Le test d'égalité : "===" (3 signes "=")
Voir aussi le passage par référence.

Clonage

```
CLASS voiture
{ //----attributs----------
PUBLIC $vv ;

PUBLIC FUNCTION affiche ()
{ echo "affiche la vitesse de".$this->vv ."km/h"."<BR>" ;
}

} //============================== fin de la class

$mavoit1= NEW voiture() ;  // création de l'objet
$mavoit1->vv=45 ;
//_____________________________Cas
1__________CLONE_____________________________
$savoit2 = CLONE $mavoit1 ;
if( $savoit2===$mavoit1) {echo "savoit2 et mavoit1 même
objet"."<BR>" ;}
else {echo "savoit2 et mavoit1 pas même objet"."<BR>" ;}
if( $savoit2==$mavoit1) {echo "mêmes attributs"."<BR>" ;}
else {echo "pas mêmes attributs"."<BR>" ;}
$savoit2->vv=100 ;
echo "vv pour mavoit1 =".$mavoit1 ->vv."<BR>" ;
echo "vv pour savoit2 =".$savoit2->vv."<BR>" ;
if( $savoit2==$mavoit1) {echo "mêmes attributs"."<BR>" ;}
else {echo "pas mêmes attributs"."<BR>" ;}
//_____________________________Cas
2________("=")_____________________________
$c=$mavoit1 ;
if( $c===$mavoit1) {echo "c et mavoit1 même objet"."<BR>" ;}
else {echo "c et mavoit1 pas même objet"."<BR>" ;}

if( $c==$mavoit1) {echo "c et mavoit1 mêmes attributs"."<BR>" ;}
else {echo "c et mavoit1 pas mêmes attributs"."<BR>" ;}
```

6. FONCTION

a) Passage par référence

FUNCTION avancer ($t,&$d)

$t est l'entrée et $d la sortie. Si on ne met pas le "&", la valeur calculée dans la fonction n'est répercutée dans le principal.

Exemple

```
//============================== début de la class
CLASS voiture
{ //----attributs----------
PUBLIC $marque=99 ;
PUBLIC $v=0 ;
PUBLIC $d=0 ; // pas indispensable
PUBLIC $t ;  // pas indispensable
CONST pi=3.14 ;
//----Méthodes--------------
FUNCTION klaxonner ()
{ echo "Elle klaxonne !"."<BR>" ;
$this->marque=5 ;   // influence externe
$mavoit->marque=2 ;  // pas d'effet ext.
echo "Dans fonction :marque=".$mavoit->marque."<BR>" ;  //
}

FUNCTION avancer ($t,&$d)
{ $d=$this->v * $t ;  // marche pas avec voiture
echo "Dans la fonction : Elle avance de ".$d." km <BR>" ;

$this->t=$this->t+5 ; // influence externe
$t=$t+3 ; // pas d'influence externe

}

} //============================== fin de la class

echo "Dans principal : La constante =".voiture::pi."<BR>" ;

$mavoit= NEW voiture() ;  // création de l'objet

$mavoit->v=100 ; //
echo "Dans principal : vitesse=".$mavoit->v."<BR>" ;

echo "Dans principal :marque=".$mavoit->marque."<BR>" ;  // $  //
valeur de l'initialisation
$mavoit->klaxonner () ;
echo "Dans principal :marque=".$mavoit->marque."<BR><BR>" ; // pas
influencé par l'appel de la fonction : valeur de l'initialisation
```

```
//===============TEST FONCTION======================
$mavoit->avancer (10,$mavoit->d) ;
echo "Dans principal après fonction :distance=".$mavoit->d."<BR>"
;

$mavoit->t=5 ; //
echo "Dans principal avant fonction :temps=".$mavoit->t."<BR>" ;
$mavoit->avancer ($mavoit->t,$mavoit->d) ;

echo "Dans principal après fonction : mavoit->temps=".$mavoit-
>t."<BR>" ; // influencé par l'appel de la fonction avec this
echo "Dans principal après fonction : distance=".$mavoit->d."<BR>"
; // accessible avec &
```

Résultat à l'écran

```
Dans principal : La constante =3.14
Dans principal : vitesse=100
Dans principal :marque=99
Elle klaxonne !
Dans fonction :marque=2
Dans principal :marque=5

Dans la fonction : Elle avance de 1000 km
Dans principal après fonction :distance=1000
Dans principal avant fonction :temps=5
Dans la fonction : Elle avance de 500 km
Dans principal après fonction : mavoit->temps=10
Dans principal après fonction : distance=500
```

Appel à un niveau supérieur

Exemple

```
//============================== début de la class PARENT
CLASS vehicule
{ //----attributs----------
PUBLIC $marque=99 ;
PUBLIC $v=0 ;
PUBLIC $d=0 ; // pas indispensable

FUNCTION avancer ($t,&$d)  // & indispensable
{ echo "Dans fonction avancer: t=".$t."<BR>" ; //
$d=$this->v * $t ;
echo "Dans la fonction avancer: Elle avance de ".$d." km <BR>" ;

}

} //============================== fin de la class  PARENT

//============================== début de la class ENFANT
CLASS voiture EXTENDS vehicule
{ //----attributs----------

//----Méthodes--------------
FUNCTION klaxonner($t,&$d)  // & indispensable
{ echo "Elle klaxonne !"."<BR>" ;
echo "Dans fonction klaxonner: t=".$t."<BR>" ; //
$this->t=$this->t+1 ;  // $t différent de $this->t
vehicule::avancer ($t,$d) ;
//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<APPEL
echo "Dans fonction klaxonner: d=".$d."<BR>" ; //
}

} //============================== fin de la class


$mavoit= NEW voiture() ;  // création de l'objet

//==============TEST FONCTION======================

$mavoit->v=100 ; //
$mavoit->t=5 ; //
echo "Dans principal avant fonction :mavoit->t=".$mavoit->t."<BR>"
;

$mavoit->klaxonner($mavoit->t,$mavoit->d) ;
//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<APPEL

echo "Dans principal après fonction : mavoit->t=".$mavoit-
>t."<BR>" ; // influencé par l'appel de la fonction avec this
echo "Dans principal après fonction : mavoit->d=".$mavoit-
>d."<BR>" ; // accessible avec &
```

```
echo "Dans principal après fonction : t=".$t."<BR>" ; // inconnu
echo "Dans principal après fonction : d=".$d."<BR>" ; // inconnu
```

Résultat à l'écran

```
Dans principal avant fonction :mavoit->t=5
Elle klaxonne !
Dans fonction klaxonner: t=5
Dans fonction avancer: t=5
Dans la fonction avancer: Elle avance de 500 km
Dans fonction klaxonner: d=500
Dans principal après fonction : mavoit->t=6
Dans principal après fonction : mavoit->d=500

Notice: Undefined variable: t in C:\Program Files\EasyPHP
2.0b1\www\objets\function-parent.php on line 55
Dans principal après fonction : t=

Notice: Undefined variable: d in C:\Program Files\EasyPHP
2.0b1\www\objets\function-parent.php on line 56
Dans principal après fonction : d=
```

Interaction entre objets

Contrairement à PHP4, PHP5 peut faire une affectation par référence de manière implicite au moins pour l'exemple suivant ce qui peut être source d'erreurs.

Exemple

```
CLASS compte
{ //----attributs----------
PUBLIC $montant ;

PUBLIC FUNCTION virer($valeur, $destination)
{ $this->montant=$this->montant-$valeur ;
$destination->montant=$destination->montant+$valeur ;
}

} //============================== fin de la class

$lui= NEW compte() ;  // création d'un compte
$lui->montant =100 ;
$elle= NEW compte() ; // création d'un autre compte
$elle->montant =200 ;

$lui->virer(50,$elle) ;
echo "Lui : montant de son compte =".$lui->montant."<BR>" ;
echo "Elle : montant de son compte =".$elle->montant."<BR>" ;
```

En entrée, $valeur=50 et en sortie $destination=$elle qui sera modifiée. En PHP4, il aurait fallu mettre un "&" afin qu'il y ait une modification : FUNCTION virer($valeur, &$destination)

b) FONCTION : filtrer les entrées

Une fonction peut accepter qu'un type d'objet en entrée. Il faut préciser la classe dans les paramètres de la fonction.

Dans le script précédent, on rajoute la classe

```
CLASS essai
{ //----attributs----------
PUBLIC $montant ;
} //============================== fin de la class
```

et on modifie

```
PUBLIC FUNCTION virer($valeur, compte $destination)
```

On a alors un message d'erreur avec l'appel suivant.

```
$lobjet= NEW essai() ;  // création d'un objet
$lui->virer(50,$lobjet) ;
```

Message d'erreur.

Catchable fatal error: Argument 2 passed to compte::virer() must be an instance of compte, instance of essai given

Gestion des images avec la bibliothèque GD

Permet de créer et de modifier des images du type JPEG, PNG...

- Faire des formes primaires comme des ellipses, des arcs, des lignes,...
- redimensionner,tourner,.. des images
- Superposition d'images
- Diagrammes à barre, camemberts,...
- Faire des graphiques,...

Il faut que PHP soit installé avec l'extension GD:
- verifier les Extensions dans EasyPhp, Administration
- Instruction get_loaded_extensions()

```
echo "=========get_loaded_extensions()==========<BR>" ;
$A=get_loaded_extensions() ;
reset($A) ;
while (list($indice, $valeur) = each($A))
          echo "\$A[$indice] : $valeur<BR>" ;
```

Supprimer ou neutraliser le HTML pour la suite sous peine d'erreur.

Démarche

1. Création du modèle de l'image (allocation des ressources mémoire). on crée une feuille blanche ou on charge une image existante
2. Travail sur le modèle (ajouts, modification de formes, couleurs,...)
3. Fabrication de l'image finale (envoi au navigateur ou sauvegarde comme fichier sur le disque)
4. Effacement des données de la mémoire

Exemple

```
$largeur= 400 ;
$hauteur= 400 ;
$image= ImageCreate($largeur,$hauteur) ;  // reserve mémoire

$jaune=ImageColorAllocate($image, 255, 255, 0) ;
$noir=ImageColorAllocate($image, 0, 0, 0) ;

$couleur=$jaune ;
ImageFilledRectangle($image, 0 , 0, $largeur,$hauteur, $couleur) ;

Header("Content-type: image/jpeg") ;
imagejpeg($image) ;

imagedestroy() ;
```

Pour créer un fichier et le sauvegarder :

```
imagepng($image,'carre.png')
```

Pour écrire : imagestring (image, police, x, y, texte, couleur)
police entre 1 et 5
Ex : imagestring ($image, 3, 10, 90, 'Beau !', $noir)