Programmation Internet
Cours : programmation PHP objet

Jean-Michel Ilié

IUT Paris 5 - département informatique

Approche orientée-objet en PHP

Motivation de l'approche objet

- pages = structuration à partir des interactions IHM
- objet : encapsulation des données et services
  - données : gestion bds, fichiers ...
  - gestion des objets de l'application
  - approches métiers (développement séparé MVC)

Définition et instanciation des classes en PHP5

- class
- new
- $this
- public
- protected
- private

```
<? php
class Caddie
{   //Préfixés les propriétés et méthodes par
    //un attribut d'accessibilité :
                        //public (optionel, défaut),
                        //protected (visible pour  la classe et ses dérivées)
                        //private (visible pour la classe seule).
public $items;
function ajout_item ($Article,$nb)  {$this->items[$Article] += $nb;}
}
```

```
$cart = new Caddie();   //référence sur un nouvel objet Caddie
$cart -> items = array("ski" => 1);    //ne pas écrire $cart->$items
$cart -> ajout_item("casque", 1);
                                    ?>
```

Initialisation et terminaison automatique (PHP5)

- void __construct()

  void __destruct()

- unset ($obj)

  // destruction

```
<?php
class Caddie
{$item;
 function ajout_item ($Article,$nb) {...}
 function __construct()
    {$this->ajout_item ("cadeau", 1);
     }
}
```

```
$cad = new Caddie();
?>
```

Réutilisation par héritage de classe

☞ **extends**

```
<?php
class Caddie_perso
      extends Caddie
{   private $proprietaire;
    function proprio ($nom)
       { $this->proprietaire = $nom;
        }
}
```

```
$cad = new Caddie_perso();
$cad->proprio ("jmi");
echo $cad->proprietaire;
$cad->ajout_item ("batons", 2);
                ?>
```

Référence à une classe non instanciée

- :: *// référence à une **classe***
- static *//déclarations*

  //propriété ou méthode
- const *//déclaration de*

  //constante
- self *//utile pour référencer*

  //la classe courante.

```
<?php
class  Caddie_forcé  extends Caddie
{ const NbMIN=1;
  protected static function annonce ()
     {echo 'pour vous les prix bas<br>'
      echo 'à partir de'. self::NbMIN ' . articles<br>'}
}
```

```
echo "vendeur attention : <br>";
echo "ne pas oublier l'annonce : ". Caddie_forcé::annonce() ;
                                                    ?>
```

Référence à une méthode surchargée

```
<?php
class Caddie_forcé {
  protected static function annonce() {echo "pour vous les prix bas
<br>\n";}
}
class Caddie_soldes extends Caddie_forcé {
  function annonce() {
    Caddie_forcé :: annonce();        //ou ici parent::annonce();
    echo "et plus bas encore. <br>\n";
      }
}
Caddie_forcé::annonce();   //affiche pour vous les prix bas
$b = new Caddie_soldes() ;
$b->annonce();      //affiche pour vous les prix bas
                    //      et plus bas encore
?>
```

l'invocation de méthode surchargée est explicite

Comparaison d'objets

☞ **==** *// 2 objets d'une même classe*

//sont égaux s'ils possèdent

//les mêmes valeurs de propriétés

☞ **===** // tester que 2 références

//correspondent au même objet

(référence = alias)

```
<php
$c = new Caddie();
$o = new Caddie();
$p &= $o;
$r = new Caddie_forcé();   ... ?>
```

```
$c == $o   //TRUE
$c === $o  //FALSE
$c == $o        //TRUE
$p === $o       //TRUE
$c ==$r              //FALSE
$c === $r            //FALSE
```

Itérations sur les propriétés d'un objet

☞ **foreach (as =>)**

similaire au parcours des collections et tableaux

```
$a = new Caddie_perso();
foreach ($a as $index => $valeur)
{ echo 'article ['. $index .' ]: ' . $valeur .'<br>';
}
```

//voir aussi les itérateurs

Duplication d'un objet

- ☞ void __clone()

  //spécifie la façon de cloner

- ☞ clone //crée une copie clonée

  //copie les valeurs propriétés

  //y compris les propriétés référence

  // et __clone() est invoquée.

- = //alternative : copie exacte des

  //valeurs des propriétés.

```
<?php
class Caddie_indexé
      extends Caddie
{   private $num=0;
    function __clone ()
       { $this->$num +=1 ;
        }
}
```

```
$a = new Caddie_indexé();
$b = clone $a;
          ?>
```

// Les propriétés non specifiées par le clonage sont recopiées exactement.

Sauvegarde d'objets

- $ch=serialize($objet);
- $obj=unserialize($ch);

Sérialisation = sauvegarde sous forme d'une chaîne de caractères de l'ensemble des variables d'un objet.

- la sérialisation et la désérialisation nécessite la définition de la classe
  --- *connaissance des propriétés de la classe*

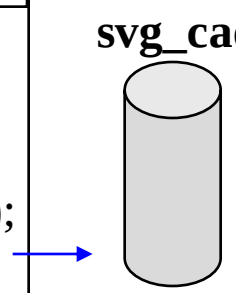
Exemple de sérialisation

classDef.inc

```
<?php    //fichier commun de définition
  class Caddie {
private items;
function ajout_item ($Article,$nb)  {$this>items[$Article] += $nb;} }
?>
```

page1.php

```
<?php
 include("classDef.inc");
 $a = new Caddie;
 $ch = serialize($a);
 $fp = fopen("svg_cad", "w");
 fputs($fp, $ch);
 fclose($fp);          ?>
```



page2.php

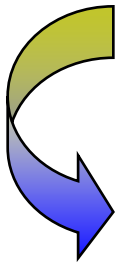
```
<?php
 include("classDef.inc");
 $ch = implode("", @file("svg_cad"));
 $a=unserialize($ch);   lecture
 $a->ajout_item(10,1);
?>
```

$ch=implode ($separ, array pieces): regroupe des chaînes (idem join, inverse : explode ou split)
$tab=file (nomFic [, int includePath]) // idem readfile mais retourne un tableau

Exploitation des variables sessions

script 1

```
<php start_session();
include("classDef.inc");
$_SESSION['obj'] = new Caddie();
?>
```



script 2

```
<php start_session();
include("classDef.inc");
echo $_SESSION['obj'] -> Items;
?>
```