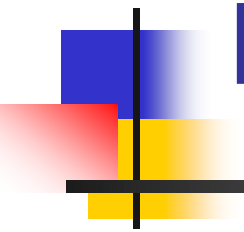
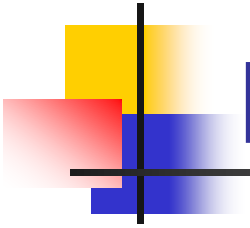


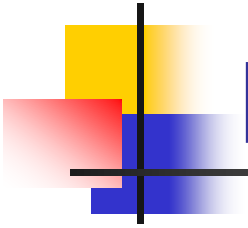
PHP5

Programmation orientée objet



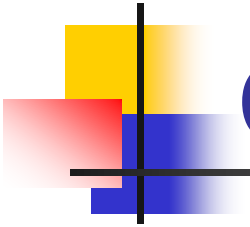
La POO en PHP

- Intérêt
  - Progrès par rapport à la version PHP4
  - Permet
    - de rassembler autour d'un même objet (concept), une définition, des attributs et des méthodes d'action
    - de réutiliser des objets existants en les complétant par ce qui est nouveau
  - D'où des avantages liés à :
    - La clarté du code
    - La modularité
    - La réutilisabilité
    - L'interopérabilité



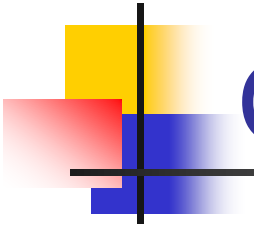
La POO en PHP

- Qu'est ce qu'un objet ?
  - Un objet est toute donnée manipulée
    - Exemples
      - la voiture de mon voisin
      - mon compte bancaire
- Qu'est ce que les attributs et les méthodes ?
  - Chaque objet a des attributs qui lui sont propres
    - Mon compte bancaire a 3 attributs
      - Le numéro du compte
      - Le solde actuel
      - Liste des différentes opérations
  - Les objets peuvent avoir des méthodes
    - Ce sont des actions que l'on peut appliquer à un objet
      - Solder
      - Créditer…



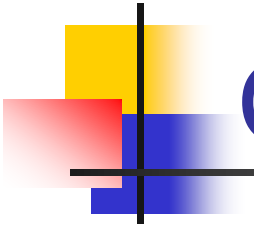
Classes et instances

- Qu'est ce qu'une classe ?
  - C'est un modèle de donnée
  - On peut la voir comme une famille d'objets
  - Tous les objets de la classe sont similaires
    - Partagent les mêmes attributs et les mêmes méthodes
  - Exemple
    - Classe rassemblant toutes les voitures
    - Attributs
      - Toutes les voitures (les objets) ont des plaques d'immatriculation, un moteur, un nombre de portières…
    - Méthodes
      - Toutes les voitures (les objets) ont des méthodes pour démarrer, freiner, accélérer…



Classes et instances

- Qu'est ce qu'une instance ?
  - Une instance est une représentation particulière d'une classe
  - Exemple
    - Mégane est une classe
    - La voiture que vous venez d'acheter est une instance
      - Elle est bleue, sans options
    - Une autre instance est
      - la voiture rouge garée en bas de chez vous



Classes et instances

- Utilisation simple des objets
  - Déclarer une classe

```
class voiture {
    //contenu de la classe
}
```

  - Attributs
    - Le contenu de la classe est structuré en deux parties
    - La première partie permet de déclarer les attributs
    - Les attributs sont déclarés en utilisant la syntaxe des variables et un des mots-clés suivant : public, private ou protected
  - Exemple

```
class voiture {
    public $marque;
}
```

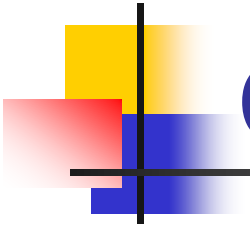
Classes et instances

- Il est possible d'utiliser une valeur par défaut

```
<?php
    class voiture {
            public $marque='Ferrari';
    }
?>
```

- Méthodes
  - La deuxième partie du contenu d'une classe permet la déclaration des méthodes
  - Ces méthodes se déclarent exactement comme des fonctions
  - Exemple

```
<?php
    class voiture {
        public $marque='Ferrari';
        function freiner($force_de_freinage){
                    //instructions pour faire freiner
        }
    }
?>
```



Classes et instances

- Créer un objet
  - On vient de voir que créer une classe n'est pas suffisant pour pouvoir l'utiliser
  - Pour l'utiliser proprement et avoir accès à ses paramètres, il faut en créer des instances : objets
    - $var_objet = new nom_classe()
  - Attention
    - le code de définition de la classe doit être dans le même script ou inclus au début du script à l'aide des fonctions require() ou include()
  - On peut créer plusieurs objets représentant des actions boursières conformes au modèle de la classe action :
    - $action1 = new action();
    - $action2 = new action();

Classes et instances

- Accès aux propriétés d'un objet
  - Pour accéder, aussi bien en lecture qu'en écriture, à la propriété prop d'un objet, on utilise : –>
    - $nom_objet–>prop;
  - ou encore :
    - $nom_objet–>prop[n];
  - si la propriété prop de l'objet est un tableau
  - Pour appeler une méthode de l'objet, appliquez la même notation :
    - $nom_objet–>nom_fonction();

- Exemple : objet4.php

```
<?php
    require("objet2.php");
    //Création d'une action
    $action1= new action();
    //Affectation de deux propriétés
    $action1->nom = "Mortendi";
    $action1->cours = 15.15;
    //utilisation des propriétés
    echo "<b>L'action $action1->nom cotée à la $action1->bourse vaut
    $action1->cours &euro;</b><hr>";
    //Appel d'une méthode
    $action1->info();
    echo "La structure de l'objet \$action1 est :  <br>";
    var_dump($action1);
    echo "<h4>Descriptif de l'action</h4>";
    foreach($action1 as $prop=>$valeur){
       echo "$prop = $valeur <br />";
    }
    if($action1 instanceof action) echo "<hr />L'objet \$action1 est du type
    action";
?>
```

- Exemple : objet4.php : résultat

L'action Mortendi cotée à la bourse de Paris vaut 15.15 €

Informations en date du 18/07/2011 14:40:52

Horaires des cotations

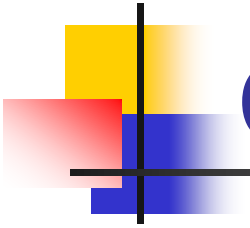
La Bourse de Paris est ouverte
La Bourse de New York est fermée

La structure de l'objet $action1 est :
object(action)#1 (3) { ["nom"]=> string(8) "Mortendi" ["cours"]=> float(15.15) ["bourse"]=> string(16) "bourse de Paris " }

Descriptif de l'action

nom = Mortendi
cours = 15.15
bourse = bourse de Paris

L'objet $action1 est du type action



Classes et instances

- Accès aux variables de la classe
  - Les variables propres de la classe ne sont pas accessibles directement à l'extérieur du script qui définit la classe
    - De même pour les méthodes
  - Cette particularité est nommée *encapsulation et permet en q*uelque sorte de protéger la « cuisine » interne, conçue pour créer une classe
  - De la même façon, si vous essayez d'utiliser dans une méthode une variable déclarée de la classe, vous n'obtenez aucun résultat

Classes et instances

- Accès aux variables
  - Pour qu'une méthode accède aux variables déclarées dans la classe, elle doit y faire appel à l'aide de la syntaxe suivante :
    - $this–>mavar
  - La pseudo-variable $this fait référence à l'objet en cours, ce qui permet d'utiliser la variable $mavar dans la méthode
  - La méthode info() de la classe action peut maintenant être enrichie et avoir comme fonctionnalité supplémentaire d'afficher toutes les propriétés d'un objet action
  - On peut, par exemple, remplacer la ligne de code d'objet4.php
    - echo "L'action $action1->nom cotée à la $action1->bourse vaut $action1->cours €<hr>";
  - par le code suivant, qui fera partie du corps de la fonction info() :

Classes et instances

```
if(isset($this–>nom) && isset($this–>cours))
{
   echo "<b>L'action $this–>nom cotée à la bourse de {$this–
   >bourse[0]}
   ➥vaut $this–>cours &euro;</b><br />";
}
```

- Cet accès aux variables de la classe est aussi valable si l'une de ces variables est un tableau
- Exemple, supposons que $montab a été déclaré comme suit :

  ```
  public $montab = array("valeur1","valeur2");
  ```

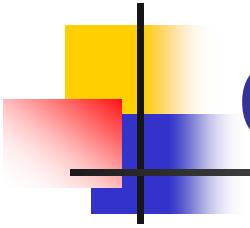
- Pour accéder à la valeur d'un de ses éléments, on peut écrire :

  ```
  $this–>montab[1]
  ```

Autre exemple : objet5.php montrant l'utilisation des variables et constantes de la classe. $_SERVER permet de lire le nom du serveur

```
<?php
class action
{
  //Définition d'une constante
  const PARIS="Palais Brognard";
  const NEWYORK="Wall Street";
  //Variables propres de la classe
  public $nom ;
  public $cours;
  public $bourse=array("Paris ","9h00","18h00");
  //fonctions propres de la classe
  function info(){
     global $client;
     //Utilisation de variables globales et d'un tableau superglobal
     echo "<h2> Bonjour $client, vous êtes sur le serveur:
     ",$_SERVER["HTTP_HOST"],"</h2>";
     echo "<h3>Informations en date du ",date("d/m/Y H:i:s"),"</h3>";
     echo "<h3>Bourse de {$this->bourse[0]} Cotations de {$this->bourse[1]} à
     {$this->bourse[2]} </h3>";
```

```
    //Informations sur les horaires d'ouverture
    $now=getdate();
    $heure= $now["hours"];
    $jour= $now["wday"];
    echo "<hr />";
    echo "<h3>Heures des cotations</h3>";
    if(($heure>=9 && $heure <=17)&& ($jour!=0 && $jour!=6))
        {echo "La Bourse de Paris ( ", self::PARIS ," ) est ouverte <br>"; }
    else {  echo "La Bourse de Paris ( ", self::PARIS ," ) est fermée <br>"; }
    if(($heure>=16 && $heure <=23)&& ($jour!=0 && $jour!=6) )
      {echo "La Bourse de New York ( ", self::NEWYORK ," ) est ouverte <hr>";}
    else
     {echo "La Bourse de New York ( ", self::NEWYORK ," ) est fermée <hr>";}
     //Affichage du cours
     if(isset($this->nom) && isset($this->cours))
      {echo "<b>L'action $this->nom cotée à la bourse de  {$this->bourse[0]} vaut
     $this->cours &euro;</b><br />";}
  }
}
?>
```



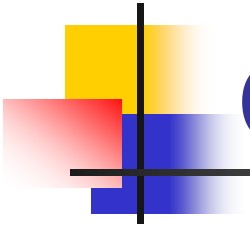
Classes et instances

- Accessibilité des propriétés : 3 options
  - Public
    - Permet l'accès universel à la propriété, aussi bien dans la classe que dans tout le script, y compris pour les classes dérivées, comme vous l'avez vu jusqu'à présent
  - Protected
    - La propriété n'est accessible que dans la classe qui l'a créée et dans ses classes dérivées
  - Private
    - C'est l'option la plus stricte : l'accès à la propriété n'est possible que dans la classe et nulle part ailleurs

Exemple : objet6-2.php

```php
<?php
class acces{
    //Variables propres de la classe
    public $varpub ="Propriété publique";
    protected $varpro="Propriété protégée";
    private $varpriv="Propriété privée";
    function lireprop() {
        echo "Lecture publique: $this–>varpub","<br />";
        echo "Lecture protégée: $this–>varpro","<br />";
        echo "Lecture privée: $this–>varpriv","<hr />";
    }
}
$objet=new acces();
$objet–>lireprop();
echo $objet–>varpub;
//echo $objet–>varpriv; Erreur fatale
//echo $objet–>varpro; Erreur fatale
echo "<hr />";
foreach(get_class_vars('acces') as $prop=>$val)
    { echo "Propriété ",$prop ," = ",$val,"<br />";}
?>
```

```
Lecture publique: Propriété publique
Lecture protégée: Propriété protégée
Lecture privée: Propriété privée

Propriété publique

Propriété varpub = Propriété publique
Propriété varpro = Propriété protégée
Propriété varpriv = Propriété privée
```



Classes et instances

- Accessibilité des méthodes
  - On retrouve les 3 options
  - Public
    - La méthode est utilisable par tous les objets et instances de la classe et de ses classes dérivées
  - Protected
    - La méthode est utilisable dans sa classe et dans ses classes dérivées, mais par aucun objet
  - Private
    - La méthode n'est utilisable que dans la classe qui la contient, donc ni dans les classes dérivées, ni par aucun objet
  - Tout appel d'une méthode en dehors de son champ de visibilité provoque une erreur fatale

Exemple : objet7.php

```
<?php
class accesmeth{
     //Variables propres de la classe
     private $code="Mon code privé";
    //protected $varpro="Mon code protégé";
    //****************************************
    //Méthodes
     //Méthode privée
    private function lirepriv(){
       echo "Lire privée ",$this->code,"<br />";
    }
    //Méthode protégée
    protected function lirepro(){
       echo "Lire protégée ",$this->code,"<br />";
    }
    //Méthode publique
    public function lirepub(){
       echo "Lire publique : ",$this->code,"<br />";
       $this->lirepro();
       $this->lirepriv();
 }
}
```

Exemple : objet7.php

```
//Appels des méthodes
$objet=new accesmeth();//7
$objet->lirepub();//8
$objet->lirepro();//Erreur//9
//$objet->lirepriv();//Erreur//10
?>
```

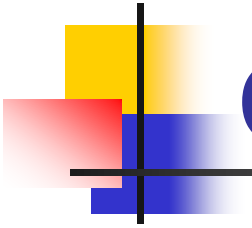
```
Lire publique : Mon code privé
Lire protégée Mon code privé
Lire privée Mon code privé

Fatal error: Call to protected method accesmeth::lirepro()
from context '' in C:\Program Files (x86)\EasyPHP-5.3.3.1
\www\LP-CISII-2011-2012\Exemples-POO\objet7.php on
line 32
```

➔ La création d'un objet et l'appel des différentes méthodes montrent que seule la méthode publique est utilisable par un objet

Constructeurs et destructeurs

- Constructeur
  - Manière de créer des objets et de définir leurs propriétés en une seule opération
    - void __construct(divers $argument1,…,argumentN)

  ➔ Méthode :
    - dite « méthode magique » comme toutes celles qui commencent par « __ »
    - porte le même nom, quelle que soit la classe, ce qui permet des mises à jour sans avoir à modifier le nom du constructeur
    - ne retourne aucune valeur
    - appelée automatiquement lors de la création de l'objet



Constructeurs et destructeurs

- Destructeur
  - void __destruct()

  ➔ Méthode :
    - s'utilise sans paramètre
    - ne retourne aucune valeur
    - appelée automatiquement soit après la destruction explicite de l'objet avec la fonction unset(), soit après la fin du script
    - contient typiquement des instructions qui permettent de gérer proprement la destruction d'un objet, comme la fermeture explicite d'un fichier ou d'une connexion à une base de données

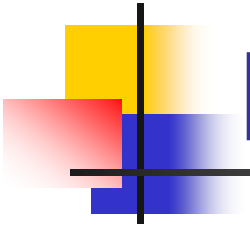
- Exemple : objet9.php

```
<?php
class action {
  private $propnom;
  private $propcours;
  protected $propbourse;
  function __construct($nom,$cours,$bourse="Paris"){
    $this->propnom=$nom;
    $this->propcours=$cours;
    $this->propbourse=$bourse;
  }
  function __destruct(){
    echo "L'action $this->propnom n'existe plus!<br />";
  }
}
```

Exemple : objet9.php (suite)

```
//Création d'objets
$alcotel = new action("Alcotel",10.21);
$bouch = new action("Bouch",9.11,"New York");
$bim = new action("BIM",34.50,"New York");
$ref=&$bim;
var_dump($alcotel);
echo "<hr />";
unset($alcotel);
unset($bim);
echo "<hr /><h4> FIN du script </h4><hr />";
?>
```

```
object(action)#1 (3) { ["propnom:private"]=> string(7) "Alcotel"
["propcours:private"]=> float(10.21) ["propbourse:protected"]=> string(5) "Paris" }

L'action Alcotel n'existe plus!

FIN du script

L'action Bouch n'existe plus!
L'action BIM n'existe plus!
```



Déférencement

- Principe
  - Il s’agit d’enchaîner des méthodes
  - Ceci est possible quand la première produit un objet
  - Syntaxe :
    - $varobj–>methode1()–>methode2();

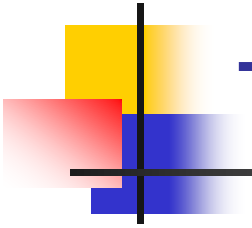
Exemple : objet10.php

```php
<?php
class varchar
{
  private $chaine;
  function __construct($a){
    $this->chaine= (string)$a;
  }
  function add($addch){
    $this->chaine.=$addch;
    return $this;
  }
  function getch(){
      return  $this->chaine;
  }
}
//Création d'objet
$texte=new varchar("Apache ");
echo $texte->getch(),"<hr />";
echo $texte->add( " PHP 5 ")->getch(),"<hr />";
echo $texte->add(" MySQL ")->add("SQLite ")->getch(),"<hr />";
?>
```

Apache

Apache PHP 5

Apache PHP 5 MySQL SQLite



Typage des paramètres

- Nouvelle possibilité
  - PHP 4 ne permettait pas de typer les paramètres d'une fonction personnelle
  - PHP 5 le permet mais uniquement pour les paramètres qui sont de type object ou array
  - Exemple

```
function (action $var)
{
   //Corps de la fonction
}
```

  - Dans ce cas, le paramètre $var doit être un objet instancié à partir de la classe action