Cours PHP

PHP Avancé : PHP5 un langage orienté objet

Par Gauthier GARNIER

Contact mail et jabber : garnier.gauthier@gmail.com

PHP avancé – un langage orienté objet

Sommaire

- Utilisation simple des objets
  - Déclaration d'une classe
  - Instanciation
  - Vérification du type
- Copie et référence
  - Le comportement PHP4
  - PHP5, le passage par référence
  - Garder la compatibilité avec PHP4
  - La copie explicite ou clonage
  - Egalité et identité
- Constructeurs et destructeurs
- La notion d'héritage
  - Définition
  - Surcharge d'attributs ou de méthodes
  - Classes et méthodes finales
  - Héritage multiple

PHP avancé – un langage orienté objet

Sommaire (suite)

- Sûreté de programmation
  - Typage
  - Classes abstraites
  - Interfaces
- Accès statiques
- Les constantes
- Chargement automatique
- Utilisation via les sessions
- Affichage d'un objet
- Itérateurs et la SPL
- Rétro-conception avec Réfection
- Les motifs de conception (Design Patterns)

PHP avancé – un langage orienté objet

- Utilisation simple des objets / Déclaration d'une classe

  En PHP4, tous les attributs et méthodes sont visibles à l'extérieur de la classe. PHP5 introduit les modificateurs d'accès qui fournissent le mode d'accessibilité (ou de visibilité) des membres d'une classe.

  - Il existe trois types d'accessibilité en PHP :
    - public : les membres publiques sont visibles sans restriction dans tout le programme,
    - protected : les membres protégés ne peuvent être accédés que par la classe qui les a déclarés et par les classes qui héritent de cette dernière,
    - private : les membres privés ne sont visibles que dans la classe qui les a déclarés.
  - Les attributs d'une classe ne peuvent être accédés dans la hiérarchie de classes et en dehors de celles-ci, qu'en utilisant respectivement la référence à l'objet en cours $this et qu'à partir d'une instance de classe.
  - Par souci de compatibilité avec les versions antérieures, la déclaration des modificateurs d'accès n'est pas obligatoire.
  - Dans la mesure du possible, tous les attributs d'une classe doivent être privés afin de respecter les principes de l'encapsulation.

PHP avancé – un langage orienté objet

- Utilisation simple des objets / Déclaration d'une classe
  - Des méthodes dites accesseurs et mutateurs permettent de respectivement accéder et modifier les attributs privés, créant ainsi une interface publique pour leur accès et leur modification. Leur rôle est de contrôler l'intégrité des données.
  - PHP défini ainsi deux méthodes __get et __set. Ces méthodes sont appelées automatiquement si l'on tente d' utiliser une variable membre inexistante.

Classe PHP4

```
class example
{
  var $variable ;

  ...
}
```

Classe PHP5

```
class example
{
  private $variable ;

  public function __get($name)
  {
    return $this->$name ;
  }

  public function __set($name, $value)
  {
    $this->$name = $value ;
  }
}
```

PHP avancé – un langage orienté objet

- Utilisation simple des objets / Vérification du type
  - PHP5 introduit le mot clé instanceof qui permet de vérifier le type d'un objet. Ce mot clé est en réalité un opérateur conditionnel qui vérifie si un objet est une instance d'une classe donnée.

Vérification du type

```
class example
{
  var $variable ;

  public function test1($param)
  {
    if ($param instanceof example)
      ...
  }
}
```

Héritage et vérification de type

```
class children extends example
{
  ...
}

$var1 = new example() ;
$var2 = new children() ;

$var1 instanceof example ➔ true

$var1 instanceof children ➔ false

$var2 instanceof example ➔ true
```

PHP avancé – un langage orienté objet

- Copie et référence / Comportement PHP4
  - Le langage PHP4 manipule uniquement les valeurs des variables. On travaille donc toujours sur une copie des objets.
  - L'esperluette (signe &) permet d'indiquer à PHP que l'on travaille par référence avec les objets.
  - En PHP4, il n'est pas possible d'enchaîner les appels de fonctions à partir d'une même variable.

PHP4, affectation par copie

```
class example
{
  var $variable ;
}

$object1 = new example() ;

$object1->variable = 2 ;

$object2 = $object1 ;

$object2->variable = 3 ;

echo $object1->variable ; // 2
```

PHP4, affectation par référence

```
class example
{
  var $variable ;
}

$object1 = & new example() ;

$object1->variable = 2 ;

$object2 = & $object1 ;

$object2->variable = 3 ;

echo $object1->variable ; // 3
```

PHP avancé – un langage orienté objet

- Copie et référence / PHP5, le passage par référence
  - En PHP5 tous les objets sont gérés par référence.
  - Le passage par référence consomme moins de mémoire et est ainsi plus performant.

PHP5 affectation

```
class example
{
  public $variable ;
}

$object1 = new example() ;

$object1->variable = 2 ;

$object2 = $object1 ;

$object2->variable = 3 ;

echo $object1->variable ; // 3
```

PHP5 appel de fonctions

```
class example1
{
  private $variable ;

  function example1()
  {
    $this->variable = new example2() ;
  }

  function getVariable()
  {
    return $this->variable ;
  }
}

class example2
{
  function test() { ... }
}

$object = new example() ;

$object->getVariable()->test() ;
```

PHP avancé – un langage orienté objet

- Copie et référence / Egalité et identité
  - La comparaison d'objets permet de vérifier si un objet est égal à un autre.
  - Il peut y avoir deux types d'égalité d'objets :
    - soit les objets possèdent une même référence,
    - soit les objets sont des instances d'une même classe et que leurs états sont identiques.
  - L'opérateur === indique si deux objets ont des références égales.
  - L'opérateur == indique si les deux objets comparés proviennent d'une instanciation de la même classe et si les attributs et leur valeur sont identiques.
  - Un objet ayant la même référence qu'un autre et ayant le même état, alors ces objets seront identiques dans les deux type de comparaisons.
  - Les opérateurs !== et != effectuent l'opération de comparaison inverse à ceux précités, en l'occurrence === et ==.

PHP5, égalité et identité

```
class example
{
  public $variable ;
}
$obj1 = new example() ;
$obj2 = $obj1 ;
$obj3 = new example() ;
```

PHP5, égalité et identité

```
$obj1->variable = 2 ;
$obj2->variable = 3 ;
$obj3->variable = 2 ;

$obj1 == $obj3 ➔ true
$obj1 == $obj2 ➔ false
$obj1 === $obj2 ➔ true
$obj1 === $obj3 ➔ false
```

PHP avancé – un langage orienté objet

- Copie et référence / La copie explicite ou clonage
  - L'instruction clone permet de copier un objet, soit entièrement, soit en personnalisant la copie.
  - Par défaut, PHP5 fait une copie systématique des propriétés de l'objet source, vers l'objet cible. Ceci a pour conséquence un clonage profond des objets, soit une copie des valeurs des propriétés de l'objet source vers l'objet cible, tant que les propriétés de l'objet source ne sont pas des objets. En effet, dans le cas des propriétés instances d'une classe quelconque, la copie est alors superficielle, c'est à dire une transmission des références d'objet des propriétés vers l'objet cible.
    - Si les propriétés sont des objets, une copie de référence est exécutée.
    - Si les propriétés sont des chaînes de caractères ou des valeurs alphanumériques, une copie de valeur est accomplie.
  - L'appel de la fonction clone produit le clonage d'un objet source vers un objet cible dans les conditions précitées.

PHP avancé – un langage orienté objet

- Copie et référence / La copie explicite ou clonage
  - La redéfinition de la méthode __clone permet d'obtenir un comportement plus approprié pour le clonage d'un objet source. Il devient possible d'obtenir un objet cible indépendant de l'objet source, mais en conservant l'état et le comportement de ce dernier. Avec ce type de clonage, la modification de l'état de l'un des objets, n'entraîne pas de changement dans l'autre.
  - Chaque objet possède une méthode __clone par défaut, qui effectue une copie superficielle de toutes les propriétés de l'objet source vers l'objet cible.

PHP5 copie d'un objet

```
class example
{
  public $variable ;

  function __clone() { ... }
}

$object1 = new example() ;

$object1->variable = 2 ;

$object2 = clone $object1 ;

$object2->variable = 3 ;

echo $object1->variable ; // 2
echo $object2->variable ; // 3
```

PHP avancé – un langage orienté objet

- Constructeurs et destructeurs
  - Le constructeur est invoqué à la création d'un objet et effectue des tâches d'initialisation.
  - En PHP4, le constructeur porte le même nom que la classe. En PHP5, le constructeur porte un nom spécial __construct. Si aucune méthode portant le nom __construct n'est trouvée alors PHP recherche une fonction de même nom que la classe.
  - Le destructeur a pour vocation de libérer des ressources allouées. Il porte le nom spécial __destruct. Le destructeur est automatiquement appelé lorsque l'objet est détruit implicitement (par le ramasse-miettes PHP) ou bien explicitement avec l'utilisation de la fonction unset.

Constructeurs PHP4

```
class example
{
  var $variable ;

  function example($param)
  {
    $var->variable = $param ;
  }
}
```

Constructeurs et destructeurs PHP5

```
class example
{
  private $variable ;

  public function __construct($param)
  {
    $this->variable = $param ;
  }

  public function __destruct() { ... }
}
```

PHP avancé – un langage orienté objet

- La notion d'héritage / Définition
  - L'héritage est un principe fondamental de la programmation orientée objet, elle permet à une nouvelle classe de posséder les caractéristiques d'une autre appelée super-classe ou classe générique. La classe ayant héritée d'une autre, est quant à elle dénommée sous-classe, classe étendue ou classe dérivée.
  - Une classe peut être déclarée comme sous-classe d'une autre classe en spécifiant le mot clé extends.
  - Les modificateurs d'accès private et protected peuvent être utilisés pour contrôler l'héritage. Un attribut ou une méthode déclaré en private ne peut pas être hérité. Un attribut ou une méthode déclaré en protected n'est pas visible en dehors de la classe mais peut être hérité.

Héritage PHP5, classe mère

```
class mother
{
  protected $member ;

  public function __construct() { ... }
}
```

Héritage PHP5, classe fille

```
class children extends mother
{
  ...
}
```

PHP avancé – un langage orienté objet

- La notion d'héritage / Surcharge d'attributs ou de méthodes
  - Une classe peut redéclarer les mêmes attributs et méthodes. La surcharge est utile pour donner à un attribut une valeur par défaut différente de la classe mère ou bien pour donner à une méthode de la classe fille un comportement différent.
  - Une classe fille hérite de tous les attributs et méthodes de la classe mère. Une méthode surchargée dans une classe fille devient prépondérante et remplace la méthode d'origine (polymorphisme).
  - Le mot clé parent permet d'accéder à la méthode parente portant la même définition. PHP utilise les valeurs d'attributs de la classe courante.

Classes parentes

```
class grand_mother
{
  public function test($param)
  {
    // do something
  }
}
class mother
{
  public function test($param)
  {
    // do something
  }
}
```

Accès aux méthodes des classes parentes

```
class children extends mother
{
  public function test($param)
  {
    parent::test($param) ;

    grand_mother::test($param) ;

    // do something
  }
}
```

PHP avancé – un langage orienté objet

- La notion d'héritage / Surcharge d'attributs ou de méthodes
  - La surcharge des méthodes peut s'opérer par l'intermédiaire de la méthode __call(). Cette surcharge consiste à utiliser un nom de méthode identique pour des appels de méthodes avec un nombre variable d'arguments. La méthode __call prend un premier paramètre qui est le nom de la méthode et un second qui est la liste des paramètres.
  - La surcharge des méthodes reflète un concept de la programmation objet. Il s'agît du polymorphisme, soit la capacité d'un objet à s'adapter aux différentes invocations possibles de méthodes avec un nom commun mais une liste d'arguments différents (signature).
  - Si les méthodes existent au sein de la classe, alors la fonction __call n'aura aucun effet.

Surcharge de méthode avec __call

```
class example
{
  public function test()
  {
    // do something
  }

  public function __call($method, $params)
  {
    $this->$method
  }
}
```

PHP avancé – un langage orienté objet

- La notion d'héritage / Classes et méthodes finales
  - Lorsque le modificateur final apparaît dans une déclaration de classe, cela signifie que la classe ne peut être héritée.
  - Une méthode portant le modificateur final ne peut pas être remplacée dans une sous-classe.
  - En outre, une méthode finale pourra être optimisée par le compilateur, puisqu'elle ne peut être sous-classée.
  - Par ailleurs, les méthodes déclarées avec le modificateur static ou private sont implicitement finales.

Classe et méthode finales

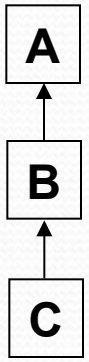
```
final class example
{
   ...
}

class example
{
   final public function doSomething()
   {
      ...
   }
}
```

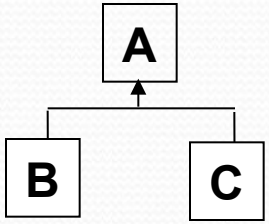
PHP avancé – un langage orienté objet

- La notion d'héritage / Héritage multiple
  - PHP ne prend pas en charge l'héritage multiple. Chaque classe ne peut hériter que d'une seule classe parente. En revanche aucune restriction n'impose une limite d'enfants engendrée par une même classe parente.

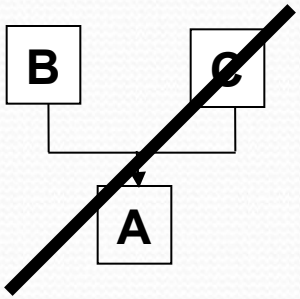
Les types d'héritage



Héritage unique



Héritage unique



Héritage multiple

PHP avancé – un langage orienté objet

- Sûreté de programmation / Typage
  - PHP5 autorise le typage des paramètres d'une méthode et fournit donc la possibilité d'indiquer le type d'objet permis pour le passage d'arguments. L'indicateur de classe est spécifié avant le paramètre. Ainsi si le type de la valeur du paramètre ne correspond pas une erreur est déclenchée. La vérification de type est l'équivalent de instanceof.
  - En fait, cette écriture ne correspond pas à un typage réel, à l'instar des langages tels que Java, mais plutôt un moyen de vérifier l'instance de l'objet passé en argument au moment de l'exécution.
  - La vérification du type du paramètre est valable uniquement pour les objets et les tableaux.

Vérification du type

```
class example
{
  var $variable ;

  public function test1(example $param)
  {
    ...
  }
}

$object = new example();

$object->test(3) ; // error
```

PHP avancé – un langage orienté objet

- Sûreté de programmation / Classes abstraites
  - L'utilisation du modificateur abstract dans une déclaration de classe signifie qu'une ou plusieurs des méthodes à l'intérieur du corps de la classe sont déclarées comme abstraites.
  - Si une méthode à l'intérieur de la classe est déclarée avec le modificateur abstract, alors cette classe doit être définie explicitement comme abstraite.
  - Les classes déclarées avec le modificateur abstract ne peuvent être instanciées. Les classes abstraites contraignent les classes qui les étendent, à un comportement imposé. Les classes dérivées doivent fournir un bloc d'instructions pour chacune des méthodes abstraites héritées de la classe portant le modificateur abstract. La nouvelle classe dérivée pourra alors être instanciée.

Classe abstraite

```
abstract class mother
{
  private $variable ;

  abstract public function test() ;
}
```

Classe fille

```
class children extends mother
{
  public function test()
  {
    // do something here
  }
}
```

PHP avancé – un langage orienté objet

- Sûreté de programmation / Classes abstraites
  - Une classe dérivée d'une classe abstraite doit impérativement implémenter toutes les méthodes abstraites de cette dernière, sinon elle doit porter le modificateur abstract dans sa déclaration.
  - Une interface pouvant être considérée comme étant une classe abstraite, une classe qui en implémente une doit être déclarée abstraite si toutes les méthodes de l'interface n'y sont pas implémentées.
  - Les classes abstraites sont utilisées pour fournir des méthodes et des variables communes à l'ensemble de leurs sous-classes.
  - Les classes abstraites, au contraire des interfaces, peuvent contenir des méthodes qui possèdent une implémentation.

PHP avancé – un langage orienté objet

- Sûreté de programmation / Interfaces
  - Une interface est un modèle d'implémentation, permettant de mettre en relation des entités sans rapports.
  - Une interface, définissant un protocole de comportement, est un élément que des objets sans liens utilisent pour interagir entre eux.
  - La déclaration d'une interface s'effectue par l'intermédiaire de l'instruction interface.
  - Les interfaces possèdent en général dans leur bloc d'instructions, des déclarations de méthodes sans corps ainsi que des déclarations de variables.
  - Toutes les méthodes contenues dans une interface sont implicitement déclarées publiques et abstraites.
  - Les méthodes ne peuvent être déclarées avec le modificateur static.
  - Toutes les variables, en fait des constantes, contenues dans une interface doivent être déclarées avec le modificateur const. Elles doivent avoir une valeur constante d'affectation.
  - Les interfaces ne peuvent implémenter d'autres interfaces. Par contre, elles peuvent hériter de plusieurs autres interfaces.
  - Le bloc d'instructions des interfaces contient essentiellement des déclarations de constantes et de méthodes abstraites.
  - Les interfaces constituent une catégorie particulière des classes abstraites.

PHP avancé – un langage orienté objet

- Sûreté de programmation / Interfaces
  - L'instruction implements permet d'indiquer que la classe fournit une implémentation pour une ou plusieurs interfaces. Ce mot clé doit toujours être placé après la clause d'extension introduite par extends.
  - Toutes les classes implémentant une interface doivent possèder les méthodes déclarées dans cette dernière en y ajoutant un bloc d'instructions.
  - De plus, PHP ne permettant pas l'héritage multiple, l'utilisation des interfaces permet de palier à cette carence dans la mesure où une classe peut désormais implémenter plusieurs interfaces.

Déclaration d'interfaces

```
interface example1
{
  abstract public function test1() ;
  abstract public function test2() ;
}

interface example2
{
  abstract public function test3() ;
  abstract public function test4() ;
}
```

Implémentation d'une ou plusieurs interfaces

```
class example3
implements example1, example2
{
  public function test1() { ... }
  public function test2() { ... }
  public function test3() { ... }
  public function test4() { ... }
}
```

PHP avancé – un langage orienté objet

- Accès statique
  - Une méthode dite statique est invoquée sans instancier la classe auquelle elle appartient.
  - Si une méthode est invoquée de manière statique et qu'elle n'est pas déclarée comme telle, PHP produit un avertissement.
  - Un attribut statique a une valeur commune à l'ensemble des objets appartenant à la même classe.
  - On utilise l'opérateur de portée :: pour accéder à une méthode ou un attribut statique.

Méthode statique

```
class example
{
  static public function test($param)
  {
    // do something
  }
}

example::test('hello') ;

echo example::$variable ;
```

Attribut statique

```
class example
{
  static public $variable ;
}

// call a static method

echo example::$variable ;
```

PHP avancé – un langage orienté objet

- Les constantes
  - Les constantes contiennent une valeur fixe et sont caractérisées par un identificateur et un type de données à l'instar des variables.
  - Les constantes ne peuvent être déclarées que dans les classes, en utilisant le mot-clé const. Les méthodes ne peuvent contenir des déclarations de constantes.
  - Les constantes ne sont pas précédées par le signe $ spécifique aux variables.
  - Par convention, le nom des constantes est toujours en majuscules afin de les distinguer sans équivoques des variables.
  - L'accès à une constante depuis une méthode de la classe est réalisé via le mot-clé self.

Déclaration et utilisation d'une constante

```
class example
{
  const MY_CONST = 'hello' ;

  function test()
  {
    $var = self::MY_CONST ;
  }
}

echo example::MY_CONST ;
```

PHP avancé – un langage orienté objet

- Chargement automatique
  - La fonction __autoload est un mécanisme de chargement automatique des classes, selon leur nom , lorsque elles ne sont pas préalablement déclarées.
  - Lorsqu'une fonction utilisée n'est pas encore définie, PHP recherche si la fonction __autoload() est implémentée. Si tel est le cas elle est exécutée.
  - La fonction __autoload est particulièrement utile dans les frameworks PHP
  - La SPL offre une alternative avec la fonction spl_autoload_register qui permet d'enregistrer sa propre fonction de chargement personnalisée.

Autochargement de classes

```
function __autoload($class)
{
  require $class . '.php' ;
}
```

Autochargement de classes

```
function my_autoload($class)
{
  ...
}

spl_autoload_register('my_autoload') ;
```

PHP avancé – un langage orienté objet

- Utilisation avec les sessions
  - Les fonctions spéciales __sleep et __wakeup sont appelées respectivement par les commandes serialize et unserialize afin de traiter l'objet ou la chaîne de caractères représentant un objet avant la linéarisation ou délinéarisation.
  - La fonction serialize recherche la méthode __sleep dans une classe afin de la lancer avant le processus de linéarisation. La fonction spéciale __sleep peut effectuer un traitement préliminaire de l'objet dans le but de terminer proprement toutes les opérations relatives à cet objet, comme la fermeture des connexions sur les bases de données...
  - De même, la fonction unserialize recherche la méthode __wakeup dans une classe afin de la lancer avant le processus de délinéarisation. La fonction __wakeup peut accomplir des opérations de reconstruction de l'objet en ajoutant des informations, en réouvrant des connexions vers des bases de données...

Sérialisation avec la méthode __sleep

```
class example
{
  public function __sleep { ... }
}

$obj = new example() ;

serialize($obj) ;
```

Désérialisation avec la méthode __wakeup

```
class example
{
  public function __wakeup { ... }
}

unserialize($obj) ;
```

PHP avancé – un langage orienté objet

- Affichage d'un objet
  - La fonction __toString est chargée de retourner une représentation textuelle d'un objet.
  - De cette manière un objet peut fournir un aperçu de son état, sous la forme d'une chaîne de caractères.
  - La fonction __toString est appelée automatiquement lorsque l'objet est appelé automatiquement à partir d'une commande echo ou print.
  - Néanmoins dans certaines conditions, la fonction toString ne pourra être invoquée implicitement à l'exécution du programme :
    - Lorsque l'objet est associé à un opérateur de concaténation.
    - Lorsque l'objet se trouve au sein d'une chaîne de caractères.
    - Lorsque l'objet subi une opération de conversion.

Implémentation de la méthode __toString

```
class example
{
  private $variable ;

  public function __toString()
  {
    return '[' . $this->variable . ']' ;
  }
}
```

Appel implicite de la méthode __toString

```
$object = new example() ;

echo $object ;
```

PHP avancé – un langage orienté objet

- Itérateurs et la SPL
  - La SPL (Standard PHP Library) est une extension intégrée par défaut à PHP5. Elle apporte une collection de classes abstraites et d'interfaces qui permettent de solutionner élégamment des problèmes récurrents concernant le parcours de données en boucle.
  - Un **itérateur** est une interface qui définit des méthodes de parcours.
  - Principaux itérateurs :
    - Iterator
    - ArrayIterator
    - CachingIterator
    - DirectoryIterator
    - FilterIterator
    - LimitIterator
    - RecursiveDirectoryIterator

PHP avancé – un langage orienté objet

- Itérateurs et la SPL

Les méthodes introduites par Iterator sont :

- current : retourne l'élément courant
- key : retourne la valeur de la clef courante
- next : positionne sur l'élément suivant (ne retourne rien)
- rewind : positionne sur le premier élément (ne retourne rien)
- valid : indique si on a atteint la fin de la liste d'éléments.

Utilisation de l'interface Iterator

```
class objectIterator implements Iterator
{
  private $objectList ;
  private $count ;
  private $index ;

  public function __construct(array $valueList)
  {
    $this->objectList = $valueList ;
    $this->count = count($valueList) ;
    $this->index = 0 ;
  }

  public function rewind() { $this->index = 0 ; }
  public function key() { return $this->index ; }
  public function next() { $this->index++ ; }

  public function current()
  {
    return $this->objectList[$this->index] ;
  }

  public function valid()
  {
    return $this->index < $this->count ;
  }
}

$object = objectIterator(array(2,4,7,5)) ;
echo $object[1] ;
```

PHP avancé – un langage orienté objet

- Itérateurs et la SPL

La classe RecursiveDirectoryIterator propose des méthodes pour parcourir un répertoire et récupérer des informations sur les différentes entrées rencontrées. Elle implémente l'interface Iterator.

Quelques méthodes : getFilename, getPathname, getSize, getType, …

Implémentation de la classe RecursiveDirectoryIterator

```
$directory = new RecursiveIteratorIterator(
  RecursiveDirectoryIterator('.'),
  true
) ;

foreach ($directory as $file)
{
  echo str_repeat(' ', $dir->getDepth()) . ' '.$file.'<br/>';
}
```

PHP avancé – un langage orienté objet

- Rétro-conception avec Reflection

Réfection est une API orientée objet constituée d'une interface et d'un ensemble de classes disponibles par défaut. On désigne par réflexion la possibilité d'interroger des classes et des objets existants afin de se renseigner sur leur structure et leur contenu.

- Reflector : Interface. Elle définit deux méthodes abstraites : export et __toString.
- Reflection : Classe de plus haut niveau
- ReflectionFunction : Récupère des informations sur les fonctions. Elle implémente l'interface Reflector
- ReflectionObject : Récupère des informations sur les Objets. Cette classe étend les fonctionnalités de la classe ReflectionClass.
- ReflectionExtension : Récupère des informations sur les extensions. Elle implémente l'interface Reflector.
- ReflectionClass : Récupère des informations sur les classes. Cette classe implémente l'interface Reflector.
- ReflectionMethod : Récupère des informations sur les méthodes
- ReflectionParameter : Récupère des informations sur les paramètres (des méthodes et des fonctions). Elle implémente l'interface Reflector
- ReflectionProperty : … sur les propriétés d'un objet
- ReflectionException : Classe de gestion des exceptions, elle hérite de la classe Exception

PHP avancé – un langage orienté objet

- Rétro-conception avec Reflection

  L'API de réflexion peut être utile pour effectuer un interfaçage avec des classes inconnues ou non documentées.

Rétro-conception avec Reflection

```
$class = 'Exception' ;

try
{
  $reflection = new ReflectionClass($class) ;

  $methodList = $reflection->getMethods() ;

  foreach($methodList as $method)
  {
    echo $method . '<br/>';
  }
}
catch (ReflectionException $e)
{
  ...
}
```

PHP avancé – un langage orienté objet

- Les motifs de conception (Design Patterns)
  - En génie logiciel, un motif de conception ("design pattern" en anglais) est un concept destiné à résoudre les problèmes récurrents suivant le paradigme objet. En français on utilise aussi les termes "modèle de conception" ou "patron de conception".
  - Les patrons de conception décrivent des solutions standards pour répondre à des problèmes d'architecture et de conception des logiciels. À la différence d'un algorithme qui s'attache à décrire d'une manière formelle comment résoudre un problème particulier, les patrons de conception décrivent des procédés généraux de conception.
  - L'objectif général des patrons de conception est de minimiser les interactions qu'il peut y avoir entre les différentes classes (ou modules, plus généralement) d'un même programme. L'avantage de ces patrons est de diminuer le temps nécessaire au développement d'un logiciel et d'augmenter la qualité du résultat.

PHP avancé – un langage orienté objet

- Les motifs de conception (Design Patterns)
  - Les patrons de conception les plus répandus sont au nombre de 23. On distingue 3 familles de patron de conception selon leur utilisation : créationnistes, structuraux et comportementaux. Le patron Modèle-Vue-Contrôleur (MVC) est une combinaison des motifs Observateur, Stratégie et Composite.
  - **Création** : Ils définissent comment faire l'instanciation et la configuration des classes et des objets : Fabrique, Prototype, Singleton…
  - **Structure** : Ils définissent comment organiser les classes d'un programme dans une structure plus large (séparant l'interface de l'implémentation) : Adaptateur, Objet composite, Décorateur.
  - **Comportement** : Ils définissent comment organiser les objets pour que ceux-ci collaborent (distribution des responsabilités) et expliquent le fonctionnements des algorithmes impliqués : Commande, Itérateur, Observateur, Etat, …

PHP avancé – un langage orienté objet

- Les motifs de conception (Design Patterns) / Le singleton
  - Le Singleton est l'un des modèles les plus simples à implémenter. Son objectif est de restreindre les capacités d'instanciation d'une classe, le plus souvent à un seul objet.
  - Sa création répond au besoin de ne pas dupliquer certaines instances de classe (accès à la base de données, gestion des threads, gestion du cache) et permet ainsi de limiter l'usage des ressources. Plusieurs instances de différentes classes peuvent se reposer sur une seule instance de la classe appliquant ce modèle.

Motif de conception, le singleton

```
class singletonPattern
{
  private static $instance ;

  private function __construct() { ... }

  public static function instance()
  {
    if (isset(self::$instance))
    {
      return self::$instance ;
    }

    return new singletonPattern() ;
  }
}

$object = singletonPattern::instance() ;
```

PHP avancé – un langage orienté objet

- Les motifs de conception (Design Patterns) / La fabrique
  - La fabrique, ou usine, est responsable de la "fabrication" d'objets. Ce modèle est très pratique dans le cadre d'un ensemble de classes abstraites et concrètes, souvent semblables, mais au fonctionnement légèrement différent et qui s'applique dans des cas précis. Par exemple, une classe d'abstraction de base de données pourrait implémenter une usine : l'usine instanciera la classe dédiée à MySQL, à PostgreSQL ou Oracle selon les cas.

Motif de conception, la fabrique

```
class driverFactory
{
  public static function factory($type)
  {
    $driver = 'drivers/'
            . $type
            . '.php' ;

    if (include_once $driver)
    {
      $class = 'driver_' . $type;

      return new $class ;
    }
    else
    {
      throw new Exception('Failed to
                  load driver ' . $type);
    }
  }

  $database = driverFactory::factory(
    'mysqli'
  ) ;
```

PHP avancé – un langage orienté objet

- Les motifs de conception (Design Patterns) / Le modèle MVC

Le but de modèle MVC est de séparer les couches d'une application (en 3 couches distinctes, au minimum, et le plus souvent). On va donc distinguer :

- **Le modèle :** il encapsule la logique métier et la manipulation des sources de données.
- **La vue :** elle présente les données à l'utilisateur.
- **Le contrôleur :** c'est tout le reste, c'est lui qui analyse la requête client, accède aux données, formate le tout, et balance le tout à la partie vue, qui va afficher ça.