Développement d'applications Web avec le langage de programmation PHP

Nicolas Le Sommer
Nicolas.Le-Sommer@univ-ubs.fr
Université Bretagne Sud, IUT de Vannes, Département Informatique

Plan du cours

1. Présentation du langage PHP
2. Les bases du langage PHP
3. Programmation orientée objet en PHP
4. PHP et les bases de données avec PDO
5. PHP et le protocole HTTP
6. Gestion des sessions

Présentation du langage PHP

Un peu d'histoire...

- Le langage PHP est un langage de scripts dédié à la génération dynamique de pages Web.
- 1994 : proposition du langage Rasmus Lerdorf
- Oct. 2000 : dernière version de PHP 3
- Mai 2000 : première version de PHP 4
- Juil. 2004 : première version de PHP 5
  - Introduction d'un modèle de programmation orienté objet
- Déc. 2015 : première version de PHP 7
  - La version 7.4 est la plus couramment utilisée en production (dernière version 7.4.29 d'avril 2022)
- Nov. 2020 : première versions de PHP 8
- La dernière version : 8.1.8 (juillet 2022)
- La version 6 n'a jamais existé

Un langage interprété, orienté objet et à typage dynamique

- Langage interprété, orienté objet et à typage dynamique
  - Programmation orientée objet comparable à celle de Java à partir de PHP5
  - Typage dynamique plus contraint avec PHP7
- Le code peut être interprété et exécuté
  - en ligne de commande
  - par un serveur Web « étendu »
    - Exemple : apache avec module PHP
- PHP dispose de nombreuses APIs pour manipuler des bases de données, des annuaires, des fichiers, des images, ….
- De nombreux cadres de conception existent pour développer des applications Web
  - Laravel, Symphony, CodeIgniter, Zend, FuelPHP, CakePHP, Slim, …

Principe de fonctionnement

- Les fichiers PHP sont usuellement suffixés par « .php ».
  - Le traitement des fichiers .php sont délégués à l'interpréteur PHP.
- Le code PHP peut être mélangé au code HTML dans une page Web.
- Des balises « PHP » de début et de fin permettent de séparer le code HTML (non interprété) du code PHP à interpréter.
- 3 façons de définir un bloc d'instructions PHP dans une page HTML:
  - Définition d'une balise HTML de type script précisant que le type de langage utilisé est PHP.
  - Utilisation d'une balise spécifique.

```
<html>
  <body>
    <?php echo "hello !"; ?>
    <?= "hello !" ?> // équivalent à la ligne précédente
    <script language="php">echo "hello !";</script>
  </body>
</html>
```

Les bases du langage

(version de référence : PHP 7.4)

Les commentaires

- Les commentaires peuvent s'écrire de 3 manières:
  - Sur une seule ligne avec
    - « // », comme en C++
    - « # », comme dans les langages shell
  - Sur plusieurs lignes avec « /* » et « */ »

```
<?php
    # Un premier commentaire
    // Un second commentaire
    /* Encore un
       commentaire */
?>
```

Les variables

- En PHP, les variables sont typées dynamiquement
  - Il n'est pas nécessaire de définir le type qui leur est associé
  - Leur type peut changer en cours d'exécution
- Les variables sont toujours préfixées par le symbole « $ » et sont sensibles à la casse
- Il n'y a pas de déclaration explicite de variables en PHP
  - PHP crée automatiquement une nouvelle variable dès qu'un nouveau symbole préfixé par « $ » apparaît dans le script
- PHP définit 3 types de variables : « automatique », statique et globale
- Une référence à une variable peut être obtenue en préfixant celle-ci par le symbole « & »

```
$a = 3;
$b = &$a; // $b est une référence à $a

print "$a\n"; // affiche 3
print "$b\n"; // affiche 3
```

Les variables « automatiques »

- Les variables automatiques sont celles qui sont le plus couramment utilisées.
- Elles sont définies automatiquement dès qu'un nouveau symbole $ apparaît dans le code du programme.
- La portée d'une variable dynamique est limitée au bloc d'instructions dans lequel elle est définie.
  - Si une telle variable est définie au sein d'une fonction, alors sa portée se limite au corps de la fonction (i.e. elle n'est pas accessible depuis l'extérieur de la fonction).

```php
<?php
    $myVar = "0";  // $myVar est une chaîne de caractères (ASCII 48)
    $myVar += 2;   // $myVar est maintenant du type entier (2)
    $myVar = $myVar + 1.3;  // $myVar est maintenant du type double (3.3)
?>
```

Les variables statiques

- Les variables statiques sont des variables qui ne sont pas visibles à l'extérieur des fonctions au sein desquelles elles sont définies.
- Elles conservent leur valeur entre deux appels à la fonction.
- Pour définir une variable statique, il suffit de faire précéder le nom de la variable du mot-clé `static` lors de la définition de celle-ci.

```php
<?php
function maFonction () {
    static $cpt = 0;        // Ceci est une variable statique
    ...
    $cpt++;                 // on incrémente le compteur
    echo "compteur = $cpt";
}
?>
```

Les variables globales

- Les variables globales:
  - Les variables globales sont des variables définies en dehors de toute fonction, et rendues accessibles à l'intérieur des fonctions grâce au mot-clé global.

```php
$a = 1;
$b = 2;
function add(){
 global $a, $b;
 $b = $a+$b;
}
add();
echo "somme = $b"; // affiche somme = 3
```

- À partir de PHP 7.3, une valeur par défaut peut être affectée aux variables automatiques, statiques et globales si celles-ci ne sont pas définies.

```php
// PHP 7.3
$a = $a ?? 'valeur par défaut';

// PHP 7.4
$a ??= 'valeur par défaut';
```

Les constantes

- Les constantes sont définies à l'aide de la fonction `define()` ou du mot-clé `const`.
- Contrairement au nom d'une variable, le nom d'une constante ne commence jamais par le symbole « $ ».

```php
define("CONSTANTE", "Bonjour le monde.");
echo CONSTANTE; // affiche "Bonjour le monde."

const CONSTANT = 'Bonjour le monde !';

const ANOTHER_CONST = CONSTANT.'; Au revoir le monde !';
echo ANOTHER_CONST;
```

- La fonction `defined()` permet de vérifier si une constante existe ou non. Elle renvoie 1 si la constante existe, et 0 dans le cas contraire.

Les types

- PHP distingue
  - 4 types scalaires :booléen, entiers, flottants, chaînes de caractères
  - 4 types composés : tableaux, classes, iterable, callable
  - 2 types spéciaux : NULL, resource

- Les types numériques et booléens:

```
 $myVar = 1;             // Entier en notation décimale
 $myVar = 011;           // Notation octale < PHP 8.1 (9 en décimal)
 $myVar = 0o11;          // Notation octale PHP 8.1 (9 en décimal)
 $myVar = 0x11;          // Notation hexadécimale (17 en décimal)
 $myVar = 3.14116;       // flottant
 $myVar = 0.3e-3;        // Notation exponentielle
 $myVar = 7E-10;
 $myVar = TRUE;          // Valeur booléenne (TRUE et FALSE
                         // sont des constantes prédéfinies)
$myVar = 1_234.567;
```

Les types

- Les types chaînes de caractères:

```
 $myVar = 'C\'est une chaîne de caractères';
 $myVar = "Toto";

 echo "C'est un texte écrit par $myVar";

// Syntaxe heredoc
echo <<<END

    $myVar \n
    sdfsf
END;
```

- Remarque:
  - Les chaînes entre guillemets simples ne peuvent pas contenir de variables ou de caractères d'échappement (comme '\n').

Déclaration de type

- Les déclarations de types peuvent être ajoutées
  - aux arguments des fonctions,
  - valeurs de retour des fonctions,
  - aux propriétés des classes.
- La déclaration de type permet de s'assurer que le type est correct au moment de l'appel, sinon une TypeError est levée

```php
<?php
class User {
    private int $id;
    private string $name;

    public function get_id() : int{
     return $this->id;
    }

    public function set_id(int $i) : void {
     $this->id = $i;
    }
}
?>
```

Transtypage

- PHP permet de modifier explicitement le type d'une variable.
  - Les transtypages (array) et (object) sont autorisés.
  - (integer) est synonyme de (int).
  - (float) et (real) sont synonymes de (double).

```
$myVar = 11.2;              // $myVar est un double
$myVar = (int) $myVar;      // $myVar est maintenant un entier (valeur 11)
$myVar = (double) $myVar;   // $myVar est désormais un double (valeur 11.0)
$myvar = (string) $myVar;   // $myVar est maintenant une chaîne.
```

Fonctions associées aux variables

fonction	description
`gettype()`	Retourne "unknown type" si le type de la variable ne peut être déterminé, ou une chaîne de caractères définissant le type de la variable.
`settype()`	Permet de définir le type d'une variable de manière explicite. Le type est spécifié par une chaîne de caractères
`unset()`	Permet de détruire une variable, et ainsi de libérer la mémoire utilisée par cette variable.
`isset()`	Retourne vrai si la variable possède une valeur, et faux dans le cas contraire.

Les opérateurs associés aux types scalaires

- Opérateurs arithmétiques
  - – +, -, *, /, %
- Opérateurs logiques
  - && (ou and), || (ou or), xor, !
- Opérateurs sur les bits
  - &, |, ^, <<, >>
- Opérateurs de comparaison
  - ==, !=, >, <, <=, >=, <=>,!==, ===
- Concaténation des chaînes de caractères
  - En PHP, l'opérateur de concaténation de chaînes de caractères est le symbole « . ».
- Operateur « Nullsafe » (PHP 8)

Opérateur de comparaison ternaire introduit en PHP7

```
// Integers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
```

```
// Strings
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
```

```
// PHP7
$country =  null;
if ($user !== null) {
    $address = $user->getAddress();

    if ($address !== null) {
      $country = $address->country;
    }
}
```

```
// PHP 8
$country = user?->getAddress()?->country;
```

Les tableaux

- PHP gère dynamiquement la taille des tableaux, ce qui permet d'ajouter ou de supprimer des valeurs sans se soucier de l'espace mémoire.
- Il existe deux catégories de tableaux en PHP :
  - Les tableaux indicés:
    - Les données sont référencées par leur position en débutant à 0
  - Les tableaux associatifs:
    - Les données sont référencées par des « clés » définies explicitement par le programmeur.

Les tableaux indicés

- Un même tableau peut contenir des données de types différents.
- Il faut éviter de définir des tableaux pouvant contenir des données de types différents car
  - le code devient difficile à comprendre et à maintenir,
  - Il y a un risque d'erreur lors des itérations

```php
$myArray [0]= "ma chaîne";
$myArray [1]= 12;
$myArray [2]= 3.14116;

// L'instruction array de PHP offre le moyen
// d'initialiser facilement un tableau.
$myArray2 = array ("ma chaîne", 12, 3.14116);
```

Les tableaux indicés

- Un indice peut être affecté automatiquement à un nouvel élément du tableau
  - l'indice est le numéro de la première cellule vide
- Déconseillé car le code est peu lisible
- En utilisant cette façon de programmer, le code précédent pourrait s'écrire de la manière suivante:

```php
$myArray []= "ma chaîne";   // $myArray [0]
$myArray []= 12;            // $myArray [1]
$myArray []= 3.14116;       // $myArray [2]
```

Les tableaux associatifs

- Les tableaux associatifs permettent d'identifier une valeur à partir d'une clé.
- La clé peut être un entier (ce qui équivaut alors aux tableaux avec accès par indice) ou une chaîne de caractères.

```php
// le tableau étudiant peut être défini et initialiser de cette manière
$etutiant["numEtudiant"] = 124;
$etudiant["nomEtudiant"] = "Torvald";
$etudiant["prenomEtudiant"]= "Linus";

// le tableau étudiant aurait pu être également défini de la manière suivante :
$etutiant2 = array ("numEtudiant" => 124,"nomEtudiant" => "Torvald",
                    "prenomEtudiant" => "Linus");

// On accéder aux éléments du tableau de la manière suivante :
$myVar = $etudiant["numEtudiant"];
```

Constantes et tableaux

- Les tableaux peuvent être des constantes

```
const ANIMALS = array('chien', 'chat', 'oiseaux');
echo ANIMALS[1]; // affiche "chat"

define('ANIMALS', array(
    'chien',
    'chat',
    'oiseaux'
));
echo ANIMALS[1]; // affiche "chat
```

Les tableaux à plusieurs dimensions

- Les tableaux à plusieurs dimensions sont définis de la même manière que les tableaux à une dimension.

```php
$myArray = array ("v1" => array ("a1", "b1"),
                  "v2" => array ("a2", "b2"));

$myVar = $myArray["v2"][1]; //$myVar prend la valeur "b2"
```

Fonctions associées aux tableaux

fonction	description
array_keys()	Retourne les clés du tableau spécifié en paramètre
array_values()	Retourne les valeurs du tableau passé en paramètre
sort()	Trie le tableau par ordre croissant
rsort()	Trie le tableau par ordre décroissant
reset()	Permet de se positionner au début du tableau
end()	Permet de se positionner à la fin du tableau
prev()	Passe à l'élément précédent
next()	Passe à l'élément suivant

Opérateurs sur les tableaux

opérateur	description
$a + $b	Union de $a et $b
$a == $b	TRUE si $a et $b contiennent les mêmes paires clés/valeurs.
$a === $b	TRUE si $a et $b contiennent les mêmes paires clés/valeurs dans le même ordre et du même type.
$a != $b	TRUE si $a n'est pas égal à $b.
$a !== $b	TRUE si $a n'est pas identique à $b.
$a <> $b	TRUE si $a n'est pas égal à $b.

Opérateurs sur les tableaux

Concaténation de tableaux avec la fonction « `array_merge()` » ou l'opérateur « `...` » (à partir de PHP 7.4)

```
$array1 = array ("v1","v2") ;
$array2 = array ("a1", "b1") ;
$array3 = array.merge($array1,$array2) ;
$array3 = [...$array1, ...$array2] ;
$array4 = [...$array1, ...$array2, 1, 2,3] ;
$array5 = array("a" => "a1", "b" => "b1");
$array6 = array("a" =>"a2", "b" => "b2", "c" => "c2");
$array7 = $array5 + $array6 // contient ("a" => "a1", "b" => "b1", "c" => "c2")
$array8 = $array6 + $array5 // contient ("a" => "a2", "b" => "b2", "c" => "c2")
```

Enumerations

- À partir de PHP 8.1

```php
enum Status
{
    case Draft;
    case Published;
    case Archived;
}
function acceptStatus(Status $status) {…}

acceptStatus(Status::Published) ;
```

Les structures de contrôle

- Conditionnelles
  - `if, if/else, if/elseif/else`
  - `switch/case`
  - `match (PHP 8)`
- Boucles
  - `for, foreach,`
  - `while, do/while`
- Directives
  - Exécution de blocs : `declare()`
  - Inclusion :
    - `require, include, require_once, include_once`

```php
foreach ($arr as $value) {
    echo "Valeur : $value<br />\n";
}

foreach ($arr as $key => $value) {
    echo "Clé : $key; Valeur : $value<br />\n";
}

// inclusion du fichier a.php
include_once "a.php";

define('__ROOT__', dirname(dirname(__FILE__)));
require_once(__ROOT__.'/config.php');
```

Structures de contrôle

- match (PHP 8)

```
// Avec PHP 7

switch (8.0) {
  case '8.0':
    $result = "Oh no!";
    break;
  case 8.0:
    $result = "This is what I expected";
    break;
}
echo $result;
//> Oh no!
```

```
// Avec PHP 8

echo match (8.0) {
  '8.0' => "Oh no!",
   8.0 => "This is what I expected",
};
//> This is what I expected
```

Les fonctions

- Une fonction peut posséder un nombre arbitraire, mais fixe d'arguments, et retourner une valeur en résultat via l'instruction `return`.
- À partir de PHP 5, les arguments peuvent être typés, mais uniquement s'ils sont de type `Array`, `Callable`, classe ou interface
- À partir de PHP 7
  - les arguments peuvent être typés, y compris pour les types scalaires
  - le type de retour des fonctions peut être spécifié
  - Vérification « stricte » ou « coercitive » des types des fonctions
    - `declare(strict_types=1);`
- Les fonctions doivent être définies avant leur appel.
- Une union de type peut être spécifié comme type d'argument et type de retour (PHP 8.1)
- Le type `never` peut être spécifié pour indiquer qu'une fonction ne retourne rien (PHP 8.1)
- Les arguments peuvent être nommés

Les fonctions: exemples

- Avec PHP5

```
function addition ($val1, $val2) {
    $somme = $val1 + $val2;
    return $somme;
}

echo addition(2.5,3) ; // affiche 5.5
echo addition(2.5,'3') ; // affiche 5.5
```

```
function display (array $t{
    for($i=0 ; $i < count($t) ; $i++){
       echo $t[$i] ;
     }
}
```

- Avec PHP 7

```
function addition (int $val1, int $val2):int {
    $somme = $val1 + $val2;
    return $somme;
}
echo addition(2,3) ; // affiche 5

echo addition('2',3) ;
// en mode coercitif affiche 5
// en mode strict (declare(strict_types=1);) affiche EXCEPTION (Uncaught TypeError)
```

```
// PHP 8
function display (int|float $arg) : never
     echo $arg ;
}

function process($idx, $value){…}

process(value:12,id:1) ; // paramètres nommés
```

Les fonctions fléchées

- Les fonctions fléchées sont introduites à partir de la version 7.4 de PHP
- Elles offrent une notation plus simple pour définir des fonctions anonymes
- Les variables définies dans la portée parent sont implicitement capturées par valeur dans les fonctions fléchées
- Elles peuvent être passées en paramètres d'autres fonctions pour réaliser des *callbacks*
  - *Les callbacks* sont des fonctions qui seront appelées pour transmettre les résultats. Elles sont utilisées essentiellement dans les traitements/exécutions asynchrones.

```
function cube($n){
     return ($n * $n * $n);
}
$a = [1, 2, 3, 4, 5];
$b = array_map('cube', $a);
print_r($b);
```

```
// Avec PHP 7.4
$a = [1, 2, 3, 4, 5];
$b = array_map(fn($n) => $n * $n * $n, $a);
print_r($b);
```

```
$factor = 10;
$calc = fn($num) => $num * $factor;
```

Passage des arguments par valeur

- Les arguments sont passés par valeur
  - une copie des variables est faite au moment de l'appel de la fonction
  - les éventuelles modifications de ces arguments au sein du corps de la fonction n'ont qu'un effet local.
- Exemple:

```
function addition ($val1, $val2) {
   $somme = $val1 + $val2;
   $val1 = 2;
   $val2 = 3;
   return $somme;
}

// Appel de la fonction
$val1 = 1;
$val2=1;
$result = addition($val1,$val2);
//$val1 et $val2 valent toujours 1 ici !
```

Passage des arguments par adresse

- Il est également possible de passer les arguments par adresse en préfixant l'argument par &.
- C'est alors une référence à la variable qui est passée à la fonction et pas une copie de sa valeur.

```
function addition2 ($val1, $val2, &$somme){
  $somme=$val1+$val2;
}

// Appel de la fonction addition2
$val1=1;
$val2=1;
addition2($val1, $val2, $somme);

// $val1 et $val2 valent toujours 1 ici, mais $somme vaut 2 !
```

Valeur par défaut des paramètres de fonctions

- Le langage PHP offre le moyen de définir des valeurs par défaut pour les arguments d'une fonction.
- À éviter

  - Code difficile à lire et à maintenir

```php
function connexion ($login, $password, $host="ulysse", $database="base") {
  // Ici le corps de la fonction
}

// Connexion à la base par défaut
$connexion1 = connexion("guest", "passwdGuest");

// Connexion à la base Film sur le serveur escobez
$connexion2 = connexion("tournesol", "passwdTournesol", "escobez", "Film");
```

Programmation orientée objet

Programmation orientée objet en PHP

- Depuis PHP 5, le modèle de programmation orientée objet repose sur des
  - interfaces
  - classes abstraites
  - classes
- À partir de PHP 7.4, il est possible de définir le type des attributs d'une classe.
- En plus de l'invariance, PHP 7.4 introduit la covariance et la contravariance sur les types de paramètres et les types de retour.

Constructeurs et destructeurs d'objets

- À partir de PHP 5, le constructeur est défini par la méthode `__construct()`
- La méthode `__destruct()` permet d'effectuer certains traitements avant la destruction des objets
- Les objets sont créés à l'aide du mot-clé `new`

Portée et visibilité des attributs et méthodes

- Le modèle de programmation objet de PHP 5 permet de définir la portée et la visibilité des méthodes et des attributs d'une classe à l'aide des mots-clés suivants:

Mot-clé	Propriétés
private	Les attributs ou les méthodes précédés de ce mot-clé sont accessibles uniquement au sein de la classe dans laquelle ils sont définis.
protected	Les attributs ou les méthodes possédant cette caractéristique sont accessibles au sein de la classe dans laquelle ils sont définis et au sein des classes héritières uniquement.
public	Les attributs ou les méthodes ayant cette propriété sont visibles depuis l'extérieur des classes, et peuvent de fait être accédés par n'importe quel autre élément d'un programme.
final	Le mot-clé final permet de préciser que la méthode sur laquelle il porte ne peut être réimplantée au sein d'une classe héritière.
static	Les attributs ou les méthodes précédés du mot-clé static sont des attributs ou des méthodes de niveau classe. Ils doivent donc être invoqués sur la classe.

Interfaces et classes abstraites

- Les interfaces sont définies à l'aide du mot-clé `interface`.
- Le mot-clé `implements` permet de préciser qu'une classe implante une interface.
- Le mot-clé `abstract` est utilisé pour définir des classes abstraites et préciser qu'une méthode est abstraite
  - toute classe contenant une méthode de type `abstract` doit elle-même être de type `abstract`
- `extends` permet d'indiquer qu'une classe (ou interface) étend une autre classe (ou une autre interface).

Exemple : définition d'une interface et d'une classe abstraite

```php
interface Connection {
   //Définition du prototype des méthodes

   public function connect();
   public function disconnect();
   public function isConnected();
}
```

```php
abstract class DataBaseConnection implements Connection {
   protected $login ;    // login de l'utilisateur
   protected $password ; // mot de passe de l'utilisateur
   protected $host ;     // nom du serveur
   protected $database ; // nom de la base de données

   function __construct($userLogin, $userPassword, $hostDB, $db) {
      $this->login = $userLogin ;   $this->password = $userPassword ;
      $this->host = $hostDB ;       $this->database = $db ;
   }

   public abstract function getLogTime() ;

   public function get_login() { return $this->login ; }
   public function get_password() { return $this->password ; }
   public function get_host() { return $this->host ; }
   public function get_dataBase() { return $this->database ; }

   function __destruct() { echo 'Destructeur déclenché' ; }
}
```

Exemple: définition d'une classe

```
class MySQLConnection extends DataBaseConnection {
   private $isConnected = false ;          private $hostConnection ;
   private $dbConnection ;                 public static $numberOfConnection = 0 ;

   function __construct($userLogin, $userPassword, $hostDB, $db) {
      parent::__construct($userLogin, $userPassword, $hostDB, $db) ;
   }

   public final function connect() {
      …
     MySQLConnection::$numberOfConnection ++ ;
   }
   public final function disconnect() {
      ...
      MySQLConnection::$numberOfConnection -- ;
   }
   public final function isConnected() { return $this->isConnected ; }

   function __destruct() { parent::__destruct() ; }
}
$myConnection = new MySQLConnection("nico", "pass", "localhost", "db") ;
```

Type des attributs des classes et objets

- À partir de PHP 7.4, il est possible de définir un type pour les attributs de niveau classe et de niveau objet
  - Tous les types sont supportés, à l'exception de `void` et `callable`
  - `bool`, `int`, `float`, `string`, `array`, `object`, `iterable`, `self`, `parent`, tout nom de classe ou d'interface, et les types nullables types (?type).

```php
<?php
class User {
     private int $id;
     private string $name;
     public static int $numberOfUsers;
     private ?string $phoneNumber = null;
}
?>
```

Héritage et constructeur

- Lors de l'héritage de classe, le constructeur et le destructeur de la classe mère ne sont pas automatiquement appelés.
  - Il est possible de les invoquer explicitement.
    - `parent::__construct()`

Méthodes et variables de classes

- Les méthodes et les variables statiques de la classe courante peuvent être utilisées via la mot clé `self`
- Les méthodes statiques d'une classe peuvent être utilisées par `nomDeLaClasse::NomDeLaMéthode()`
- Exemple:

```
<?php
class test {
     public static $variable="Variable";

     public static function afficherVariable()
     {
           print(self::$variable);
     }
}
test::afficherVariable();
?>
```

Invariance, covariance et contravariance

- La variance est une propriété des hiérarchies de classes qui décrit comment les types d'un constructeur de type affectent les sous-types. En général, un constructeur de type peut être :
  - Invariant : si le type du super-type limite le type du sous-type.
  - Covariant : si l'ordre des types est préservé (les types sont classés de plus spécifique à plus générique).
  - Contravariant : s'il inverse l'ordre (les types sont classés de plus générique à plus spécifique).
- En plus de l'invariance, PHP 7.4 introduit la covariance et la contravariance sur les types de paramètres et les types de retour.

```
// covariance
interface Factory {
     function make(): object;
}

class UserFactory implements Factory {
     function make(): User;
}
```

```
// contravariance
interface Concatable {
     function concat(Iterator $input);
}

class Collection implements Concatable {
     // accepte tous les iterables,
    // et pas seulement Iterator
     function concat(iterable $input) {...}
}
```

Copie d'objets

- Des copies conformes d'objets peuvent être effectuées en invoquant l'instruction `clone` sur les objets devant être copiés
- Il est possible de définir des copies personnalisées des objets en redéfinissant les modalités de recopie au sein d'une méthode `__clone()` de la classe considérée.
- Exemple

```
class MaClasse{
     private $attr1, $attr2;

     function __construct(){ $this->attr1 = 0; $this->attr2 = false;}

     function __clone () {$this->attr2 = true;}
}
$monObj1 = new MaClasse();    $monObj2 = clone $monObj1;
```

Classes anonymes

- Les classes anonymes ont été introduites dans PHP 7

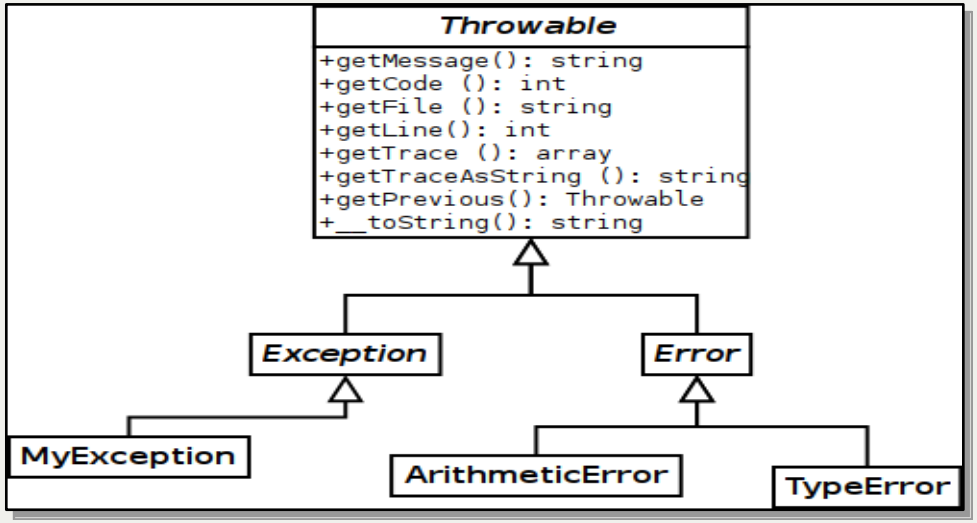
```
function createObj() {
    return new class {
        public function toString() {
            echo "Hello world !";
        }
    };
}

$obj = createObj();
$obj->toString(); // "Hello world !"
```

Gestion des exceptions

- La gestion des exceptions repose sur `try, catch, throw` et sur la classe `Exception`
  - La gestion des exceptions est comparable à celle du langage Java.
- La classe `Exception` offre les méthodes
  - `getMessage()` : retourne le message d'erreur
  - `getLine()` : retourne le numéro de ligne où l'exception est générée
  - `getFile()` : retourne le nom du fichier où l'exception s'est produite.
- La classe `Exception` peut être étendue afin de définir de nouvelles exceptions

Gestion des exceptions et erreurs

- Amélioration du traitement des erreurs dans PHP 7
  - Les erreurs peuvent être capturées à l'aide d'un bloc `try/catch`
  - Les erreurs ne donnent plus lieu à un arrêt brutal du programme
- Introduction d'une interface *`Throwable`* pour hiérarchiser les exceptions et les erreurs




```
try {
    (new MaClasse())->run();
}catch (Error $e) {echo $e;}
```

Gestion des exceptions: exemple

```php
<?php
class MyException extends Exception {

public function __toString() {
    return "exception '".__CLASS__ ."' with message '"
           .$this->getMessage()."' in ".$this->getFile().
           ":".$this->getLine()."\nStack trace:\n".
           $this->getTraceAsString();
  }
}
?>
```

```php
<?php
 class MaClasse {
 public function run() {
   ....
   if ($error){
     throw new MyException('Runtime error');
   }else{...}
}
?>
```

```php
<?php
 try {
     (new MaClasse())->run();
 }catch (MyException $e) {echo $e;}
?>
```

Espaces de nommage

- Souvent, le développeur est confronté à des problèmes de recouvrement : deux variables, deux fonctions ou deux classes ayant le même nom, qui entrent en conflit.
- Les espaces de noms permettent d'éviter ce problème en proposant une séparation de chaque brique en paquetage dont l'environnement sera protégé. Ainsi, la sécurité et l'intégrité des applications seront améliorées.

Espaces de nommage

```php
<?php
namespace MyProject/DB;

const CONNECT_OK = 1;
class Connection {
     public function connect(){...}
}
?>
```

```php
<?php
require 'MyProject/Db/Connection.php';
$x = new MyProject::DB::Connection();
$x->connect();
?>
```

- Les espaces de noms peuvent être importés dans le contexte global courant (pas classe ou fonction) en utilisant l'opérateur `use`

```php
<?php
require 'MyProject/Db/Connection.php';
use MyProject/DB/Connection as DbConnection;
$x = new  DbConnection();
$x->connect();
?>
```

Accès aux bases de données via PDO

PDO (PHP *Data Object*)

- PDO a été introduit dans la version 5.1 de PHP
- PDO facilite l'utilisation et offre une abstraction plus importants que les anciennes fonctions natives propres à chaque SGBD (`mysql_..., sqlite_...`)
- Offre un accès aux bases de données comparable à celui de JDBC
  - Support de différents types de drivers (mysql, sqlite, oracle, ...)
  - Utilisation de méthodes spécifiques
    - `exec(), query(), …`
- Avant PDO, des méthodes natives propres à chaque SGBD devaient être utilisées
  - Elles sont dépréciées depuis PHP 7

L'API PDO

PDO

```
+beginTransaction()
+commit()
+__construct()
+errorCode()
+errorInfo()
+exec()
+getAttribute()
+getAvailableDrivers()
+lastInsertId()
+prepare()
+query()
+quote()
+rollBack()
+setAttribute()
```

PDOException

```
+getMessage()
+getPrevious()
+getCode()
+getFile()
+getLine()
+getTrace()
+getTraceAsString()
+__toString()
+__clone()
```

PDOStatement

```
+bindColumn()
+bindParam()
+bindValue()
+closeCursor()
+columnCount()
+execute()
+fetch()
+fetchAll()
+fetchColumn()
+fetchObject()
+getAttribute()
+nextRowset()
+rowCount()
+setAttribute()
+setFetchMode()
...
```

Connexion à une base de données avec PDO

- Connexion à un serveur MySQL

```php
<?php
$user='lesommer'; $pass='test';
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
     ....
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
}
?>
```

- Connexion à une base de données SQLite

```php
<?php
  try {
    $db = new PDO('sqlite:/home/lesommer/music.db', array(PDO::ATTR_PERSISTENT => true));
     $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
     ....
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
}
?>
```

Requête SELECT simple avec PDO

```php
<?php
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);

     $sqlQuery = 'SELECT firstname, lastname FROM Employee ORDER BY lastname';
     $result = $connection->query($sqlQuery)->fetchAll();

     foreach ($result as $row) {
        print $row['firstname'] . "\t";
        print $row['lastname'] . "\n";
    }
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Requête SELECT avec *prepare*

```php
<?php
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);

     $sqlQuery = 'SELECT firstname, lastname FROM Employee WHERE age >:age
                  AND salaire < :salaire';
     $statement = $connection->prepare($sqlQuery);
     $statement->execute(array(':age' => 30, ':salaire' => 1000));
     $result = $statement->fetchAll();

     $sqlQuery2 = 'SELECT firstname, lastname FROM Employee WHERE age > ?
                  AND salaire < ?';
     $statement2->execute(array(30,1000));
     $statement2 = $connection->prepare($sqlQuery2);
     $result2 = $statement2->fetchAll();

} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Les méthodes `fetch()` et `fetchAll()` de `PDOStatement`

- Les méthodes `fetch()` et `fetchAll()` permettent d'obtenir la ligne suivante d'un jeu de résultats ou tous les résultats
- Les résultats peuvent être stockés dans des structures de données différentes selon les «modes `FETCH`» choisis
  - `FETCH_ASSOC`: les tuples sont stockés dans des tableaux associatifs
  - `FETCH_BOTH`: les tuples sont stockés dans des tableaux indicés et associatifs
  - `FETCH_CLASS`: les tuples sont des instances d'une classe prédéfinie
  - `FETCH_OBJ`: les tuples sont des objets anonymes dont les attributs sont les champs du jeu de résultats
  - `FETCH_INTO`: stocke le résultat dans un objet existant
  - ...

Traitement des données avec FETCH

```php
<?php
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);

     $sqlQuery = 'SELECT firstname, lastname FROM Employee ORDER BY lastname';
     $st = $connection->prepare($sqlQuery);
     $st->execute();
     $result=$st->fetch(PDO::FETCH_ASSOC);
     print_r($result);

     $result = $st->fetch(PDO::FETCH_BOTH);
     print_r($result);


} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

```
Array(
[firstname] => titi
[lastname] => toto
)
```

```
Array(
[firstname] => titi
[0]=>titi
[lastname] => toto
[1]=>toto
)
```

Traitement des données avec FETCH_OBJ

```
<?php
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);

     $sqlQuery = 'SELECT firstname, lastname FROM Employee ORDER BY lastname';
     $st = $connection->prepare($sqlQuery);
     $st->execute();
     $obj=$st->fetch(PDO::FETCH_OBJ);

     echo $obj->firstname;
     echo $obj->lastname;


} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Traitement des données avec FETCH_CLASS

```php
<?php
public class Employee{
     public $firstname;
    public $lastname;
     Public function __toString(){ return $this->lastname.' '.$this->firstname;}
}
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);

     $sqlQuery = 'SELECT firstname, lastname FROM Employee ORDER BY lastname';
     $st = $connection->prepare($sqlQuery);
     $st->execute();
     $obj=$st->fetch(PDO::FETCH_CLASS,'Employee');
     foreach($obj as $emp){
            echo $emp->__toString().'<br />';
        }
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Traitement des données avec FETCHALL

- `FetchAll` retourne un tableau contenant tous les jeux de résultats
- `FetchAll` peut être paramétré avec `FETCH_ASSOC, FETCH_BOTH, ...`
- Exemple:

```
$connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
$sqlQuery = 'SELECT firstname, lastname FROM Employee ORDER BY lastname';
$stmt = $connection->prepare($sqlQuery);
$stmt->execute();
$arrValues = $stmt->fetchAll(PDO::FETCH_ASSOC);

print "<table wdith=\"100%\">\n";
print "<tr>\n";
foreach ($arrValues[0] as $key => $useless){  print "<th>$key</th>"; }
print "</tr>";

foreach ($arrValues as $row){
     print "<tr>";
     foreach ($row as $key => $val){ print "<td>$val</td>"; }
     print "</tr>\n";
}

print "</table>\n";
```

Requêtes INSERT, UPDATE, DELETE avec PDO

- L'insertion, la mise-à-jour et la suppression de données se fait avec l'instruction `exec()` de la classe PDO ou avec la méthode `execute()` de la classe `PDOStatement`
- Exemple:

```
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
     $sqlQuery = 'DELETE FROM Employee WHERE numEmp = 12';
     $count = $connection->exec($sqlQuery);
     ....
} catch (PDOException $e) {...}
```

```
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
     $query = "insert into Employee(lastName, firstName) values (:ln,:fn)";
          $stmt = $dbc->prepare($query);

          $stmt->bindValue(':ln',$user->getLastname(),PDO::PARAM_STR);
          $stmt->bindValue(':fn',$user->getFirstName(),PDO::PARAM_STR);
     ....
} catch (PDOException $e) {...}
```

Transactions

- Les transactions reposent sur 3 méthodes: `beginTransaction()`, `commit()`, `rollBack()`
- Exemple:

```
<?php
$user='lesommer'; $pass='test';
try {
    $connection = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
     $connection->beginTransaction();

     $connection->exec("insert into ....");
     $connection->exec("insert into ...");

     $connection->commit();
} catch (PDOException $e) {
     $connection->rollBack();
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

PHP et le protocole HTTP

Protocole HTTP

- Il n'existe pas en PHP d'API spécifique pour manipuler les requêtes et réponses HTTP
- Les données transmises par les requêtes peuvent être obtenues via des variables globales
  - `$_SERVER, $_REQUEST, $_POST, $_GET, …`
- Les données transmises dans les réponses HTTP sont les données écrites sur le flux de sortie de la sortie standard
  - Dans le cas d'une connexion TCP, il s'agit du flux de sortie associé à cette connexion
- Les entêtes des réponses HTTP peuvent être modifiées via l'instruction `header`

La variable globale $_SERVER

- La variable $_SERVER permet d'accéder à des informations telles que le type de requête HTTP, l'adresse de la machine distante, ...

```php
<?php

 if($_SERVER['REQUEST_METHOD']=='POST'){
   echo 'la méthode HTTP est POST';
 }
 // le type de client utilisé
 echo "le client utilisé est $_SERVER["HTTP_USER_AGENT"]";

 // l'adresse de la machine distante
 echo "l'adresse de la machine distante est $_SERVER["REMOTE_ADDR"]";

 // le nom de la machine distante
 echo "l'adresse de la machine distante est $_SERVER["REMOTE_HOST"]";

 ?>
```

$_POST et $_GET

- Les requêtes HTTP POST et GET sont modélisées par `$_GET` et `$_POST`.
- Les paramètres CGI d'une requête GET et le contenu d'une requête POST peuvent être obtenus de la manière suivante:

```
<html>...
<form action="valider.php" method="POST">
<select name="objet">
...
<input name="nom" type="text" />
...
</html>
```

```
<?php

 $objet = $_POST['objet '];
 $genre = $_POST['nom'];

 ?>
```

```
<html>
...
<a href="valid.php?id=123">Valider</a>
...
</html>
```

```
<?php

 $user_id = $_GET['user_id'];

 ?>
```

$_REQUEST

- La variable globale `$_REQUEST` permet d'accéder aux données fournies par les méthodes HTTP POST et GET
  - Un tableau associatif qui contient par défaut le contenu des variables `$_GET`, `$_POST` et `$_COOKIE`.
- `$_REQUEST` est à privilégier

Gestion des entêtes des réponses

```php
<?php
/* Redirige le client vers le site PHP */
header("Location: http://www.php.net/");

/* Garantie que le code ci-dessous n'est jamais exécuté. */
exit();
?>
```

```php
<?php
header("HTTP/1.0 404 Not Found");
?>
```

```php
<?php
// Vous voulez afficher un pdf
header('Content-type: application/pdf');

// Il sera nommé downloaded.pdf
header('Content-Disposition: attachment; filename="downloaded.pdf"');

// Le source du PDF original.pdf
readfile('original.pdf');
?>
```

Utilisation des cookies en PHP

- Pour définir un cookie, il faut utiliser la méthode `setcookie()`
- Exemple:

```
<?php

$compteur++;

//expire le 05/12/2020 à 18h30
setcookie (”compteur”, $compteur, mktime (18,30,0,5,12, 2020));

?>
```

Utilisation des cookies en PHP

- Les cookies déposés préalablement sur la machine du client peuvent être obtenus via `$_COOKIE`
- `$_COOKIE` retourne par défaut un tableau associatif contenant l'ensemble des cookies pouvant être consultés.
- `$_COOKIE["key1"]` permet d'obtenir la valeur du cookie `key1`.
- Exemple:

```
<?php

 print_r($_COOKIE["$compteur"]);

?>
```

Gestion des sessions

Session PHP

- Une session PHP offre le moyen de partager des informations entre des plusieurs pages.
  - Une session PHP permet donc de stocker des informations de l'utilisateur sur le serveur (son panier, ses identifiants de connexion...)
- Principe de fonctionnement
  - Les informations sont stockées dans des fichiers, sur le serveur, à chaque session correspond un fichier.
  - Chaque session possède un identifiant.
  - Lorsque le visiteur accepte les cookies, l'identifiant de la session est stocké dans un cookie, dans le cas contraire, il existe un autre moyen de stocker l'identifiant.

Création d'une session

- Si un fichier existe sur le serveur pour cette session, les variables de session seront récupérées, si ce n'est pas le cas, un nouveau fichier sera créé.
- Les variables de session sont accessibles, une fois que la session est démarrée, via un tableau super global : `$_SESSION`
- Le code est à placer tout au début de votre page, avant tout code HTML.

```
<?php

session_start(); // création ou restauration d'une session
session_destroy() ; // destruction d'une session

 ?>
```

Bibliographie

- https://www.php.net/manual/fr/