

LP PRISM DÉVELOPPEMENT WEB

Programmer en PHP côté serveur

PROGRAMMATION OBJET EN PHP

Bases du PHP objet, utilisation de la base de données pour créer des objets

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**

```
livre.php
1  <?php
2  class Livre {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38  }
39  ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38  }
39  ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38  }
39  ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38  }
39  ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)
- On ne peut déclarer qu'un seul constructeur : **__construct(...)**

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11     // constructeur d'un objet Livre
12     public function __construct($num, $titre, $auteur) {
13         $this->num = $num;
14         $this->titre = $titre;
15         $this->auteur = $auteur;
16     }
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38 }
39 ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)
- On ne peut déclarer qu'un seul constructeur : **__construct(...)**
- On peut créer des getter et setter classiques

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11     // constructeur d'un objet Livre
12     public function __construct($num, $titre, $auteur) {
13         $this->num = $num;
14         $this->titre = $titre;
15         $this->auteur = $auteur;
16     }
17
18     // un exemple de getter et de setter - méthodes d'instance
19     public function getTitre() {return $this->titre;}
20     public function setAuteur($a) {$this->auteur = $a;}
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38 }
39 ?>
```


Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)
- On ne peut déclarer qu'un seul constructeur : **__construct(...)**
- On peut créer des getter et setter classiques
- On peut ajouter des méthodes d'instances : ici, la méthode **afficher()**.

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11     // constructeur d'un objet Livre
12     public function __construct($num, $titre, $auteur) {
13         $this->num = $num;
14         $this->titre = $titre;
15         $this->auteur = $auteur;
16     }
17
18     // un exemple de getter et de setter - méthodes d'instance
19     public function getTitre() {return $this->titre;}
20     public function setAuteur($a) {$this->auteur = $a;}
21
22     // une méthode d'affichage - méthode d'instance
23     public function afficher() {
24         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
25     }
26
27
28
29
30
31
32
33
34
35
36
37
38 }
39 ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)
- On ne peut déclarer qu'un seul constructeur : **__construct(...)**
- On peut créer des getter et setter classiques
- On peut ajouter des méthodes d'instances : ici, la méthode **afficher()**.
- On peut ajouter des méthodes de classe (mot-clé **static**) : ici, **afficherLivres()** et **ajouterLivre(\$unLivre)**

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11     // constructeur d'un objet Livre
12     public function __construct($num, $titre, $auteur) {
13         $this->num = $num;
14         $this->titre = $titre;
15         $this->auteur = $auteur;
16     }
17
18     // un exemple de getter et de setter - méthodes d'instance
19     public function getTitre() {return $this->titre;}
20     public function setAuteur($a) {$this->auteur = $a;}
21
22     // une méthode d'affichage - méthode d'instance
23     public function afficher() {
24         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
25     }
26
27     // une méthode pour afficher tous les livres - méthode de classe
28     public static function afficherLivres() {
29         foreach (self::$lesLivres as $unLivre) {
30             $unLivre->afficher();
31         }
32     }
33
34     // une méthode pour ajouter un livre - méthode de classe
35     public static function ajouterLivre($unLivre) {
36         self::$lesLivres[] = $unLivre;
37     }
38 }
39 ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)
- On ne peut déclarer qu'un seul constructeur : **__construct(...)**
- On peut créer des getter et setter classiques
- On peut ajouter des méthodes d'instances : ici, la méthode **afficher()**.
- On peut ajouter des méthodes de classe (mot-clé **static**) : ici, **afficherLivres()** et **ajouterLivre(\$unLivre)**
- Le mot-clé **self** est, pour la classe, l'équivalent de **\$this** pour l'instance. **self::\$lesLivres** désigne l'attribut de la classe Livre. Notez la syntaxe avec les **::** pour tout ce qui est static

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11     // constructeur d'un objet Livre
12     public function __construct($num, $titre, $auteur) {
13         $this->num = $num;
14         $this->titre = $titre;
15         $this->auteur = $auteur;
16     }
17
18     // un exemple de getter et de setter - méthodes d'instance
19     public function getTitre() {return $this->titre;}
20     public function setAuteur($a) {$this->auteur = $a;}
21
22     // une méthode d'affichage - méthode d'instance
23     public function afficher() {
24         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
25     }
26
27     // une méthode pour afficher tous les livres - méthode de classe
28     public static function afficherLivres() {
29         foreach (self::$lesLivres as $unLivre) {
30             $unLivre->afficher();
31         }
32     }
33
34     // une méthode pour ajouter un livre - méthode de classe
35     public static function ajouterLivre($unLivre) {
36         self::$lesLivres[] = $unLivre;
37     }
38 }
39 ?>
```

Une première classe : Livre

- La classe est déclarée dans un fichier **livre.php**
- On déclare de façon classique les attributs d'instance. **\$this** désigne l'instance. Ce mot-clé est inutilisable pour nommer une variable.
- Les attributs d'instance sont accessibles par **\$this->...**
- On peut aussi déclarer des attributs de classe (mot-clé **static**)
- On ne peut déclarer qu'un seul constructeur : **__construct(...)**
- On peut créer des getter et setter classiques
- On peut ajouter des méthodes d'instances : ici, la méthode **afficher()**.
- On peut ajouter des méthodes de classe (mot-clé **static**) : ici, **afficherLivres()** et **ajouterLivre(\$unLivre)**
- Le mot-clé **self** est, pour la classe, l'équivalent de **\$this** pour l'instance. **self::\$lesLivres** désigne l'attribut de la classe Livre. Notez la syntaxe avec les **::** pour tout ce qui est static
- A l'extérieur de la classe, on utilisera **Livre::\$lesLivres**

```
livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // attribut de la classe Livre (static), ici un tableau
9      static private $lesLivres = array();
10
11     // constructeur d'un objet Livre
12     public function __construct($num, $titre, $auteur) {
13         $this->num = $num;
14         $this->titre = $titre;
15         $this->auteur = $auteur;
16     }
17
18     // un exemple de getter et de setter - méthodes d'instance
19     public function getTitre() {return $this->titre;}
20     public function setAuteur($a) {$this->auteur = $a;}
21
22     // une méthode d'affichage - méthode d'instance
23     public function afficher() {
24         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
25     }
26
27     // une méthode pour afficher tous les livres - méthode de classe
28     public static function afficherLivres() {
29         foreach (self::$lesLivres as $unLivre) {
30             $unLivre->afficher();
31         }
32     }
33
34     // une méthode pour ajouter un livre - méthode de classe
35     public static function ajouterLivre($unLivre) {
36         self::$lesLivres[] = $unLivre;
37     }
38 }
39 ?>
```

Utilisation de la classe Livre

testLivre.php

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28     ?>
29   </body>
30 </html>
```

Utilisation de la classe Livre

testLivre.php

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28   ?>
29 </body>
30 </html>
```

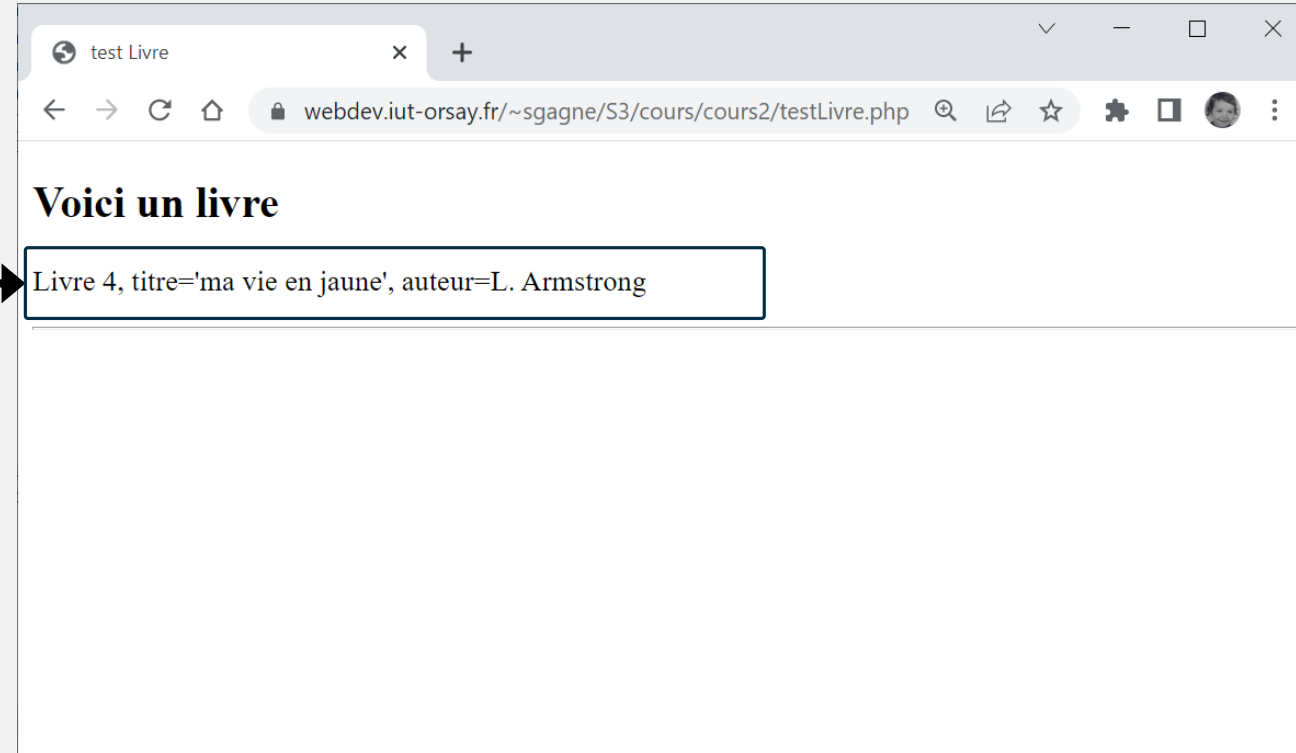
Utilisation de la classe Livre

testLivre.php

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16
17
18
19
20
21
22
23
24
25
26
27
28   ?>
29   </body>
30 </html>
```

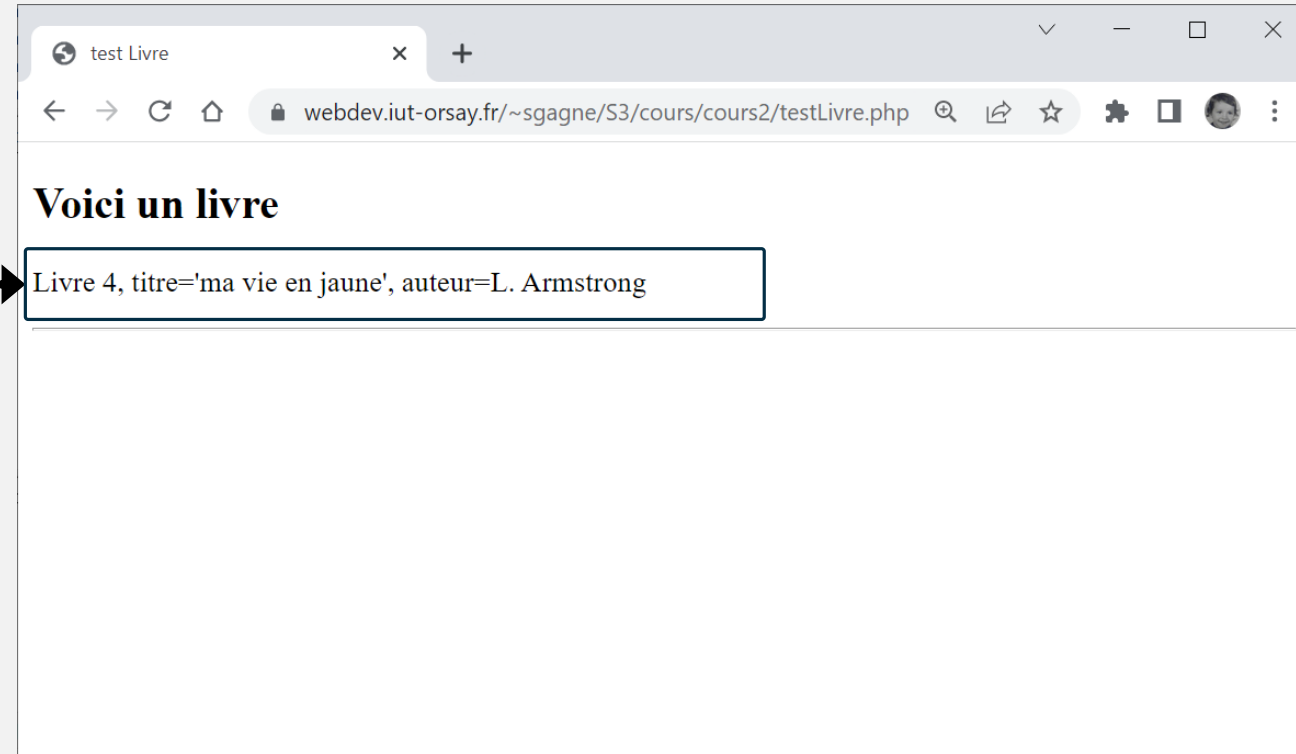
Utilisation de la classe Livre

```
testLivre.php
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16      // appel à une méthode d'instance
17      echo "<h2>Voici un livre</h2>";
18      $livre4->afficher();
19      echo "<hr>";
20
21
22
23
24
25
26
27
28   ?>
29 </body>
30 </html>
```



Utilisation de la classe Livre

```
testLivre.php
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16      // appel à une méthode d'instance
17      echo "<h2>Voici un livre</h2>";
18      $livre4->afficher();
19      echo "<hr>";
20      // appel à une méthode (static) de la classe Livre
21      Livre::ajouterLivre($livre1);
22      Livre::ajouterLivre($livre2);
23      Livre::ajouterLivre($livre3);
24      Livre::ajouterLivre($livre4);
25
26
27
28   ?>
29 </body>
30 </html>
```



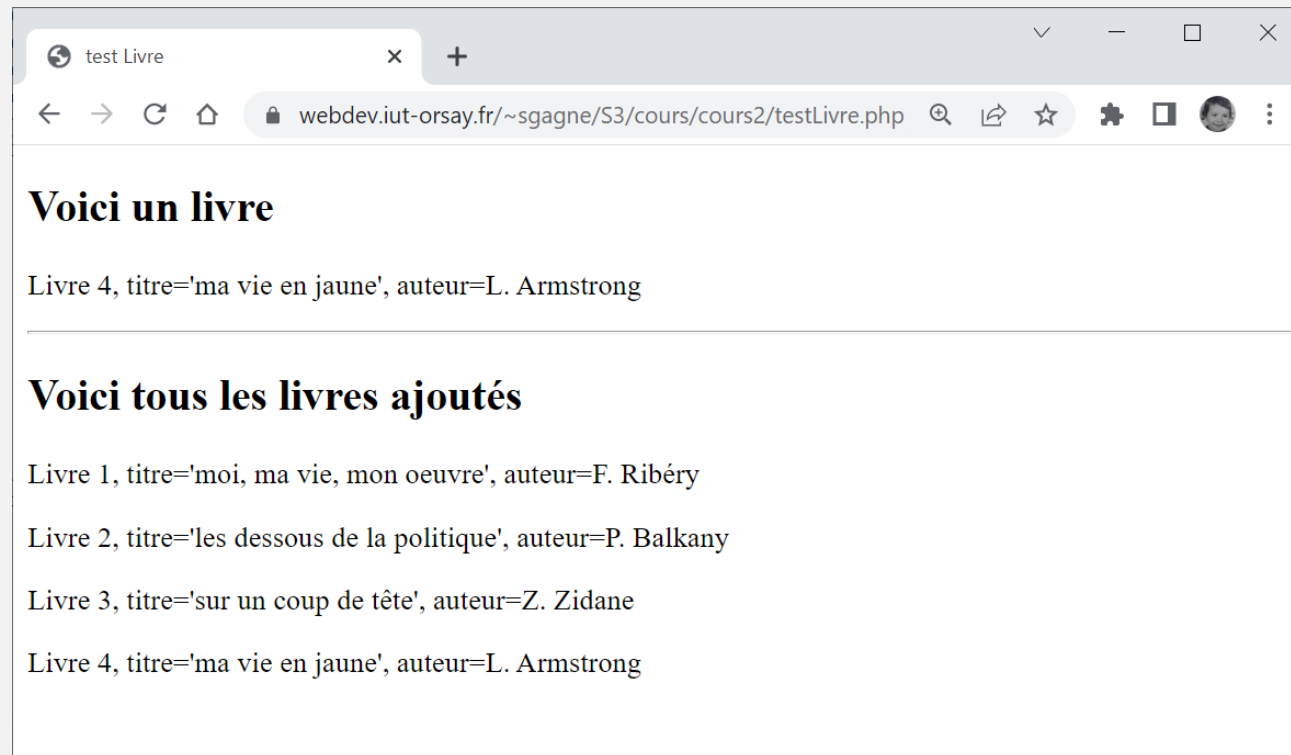
Utilisation de la classe Livre

```
testLivre.php
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16      // appel à une méthode d'instance
17      echo "<h2>Voici un livre</h2>";
18      $livre4->afficher();
19      echo "<hr>";
20      // appel à une méthode (static) de la classe Livre
21      Livre::ajouterLivre($livre1);
22      Livre::ajouterLivre($livre2);
23      Livre::ajouterLivre($livre3);
24      Livre::ajouterLivre($livre4);
25      // appel à une autre méthode de la classe Livre
26      echo "<h2>Voici tous les livres ajoutés</h2>";
27      Livre::afficherLivres();
28    ?>
29  </body>
30 </html>
```



Utilisation de la classe Livre

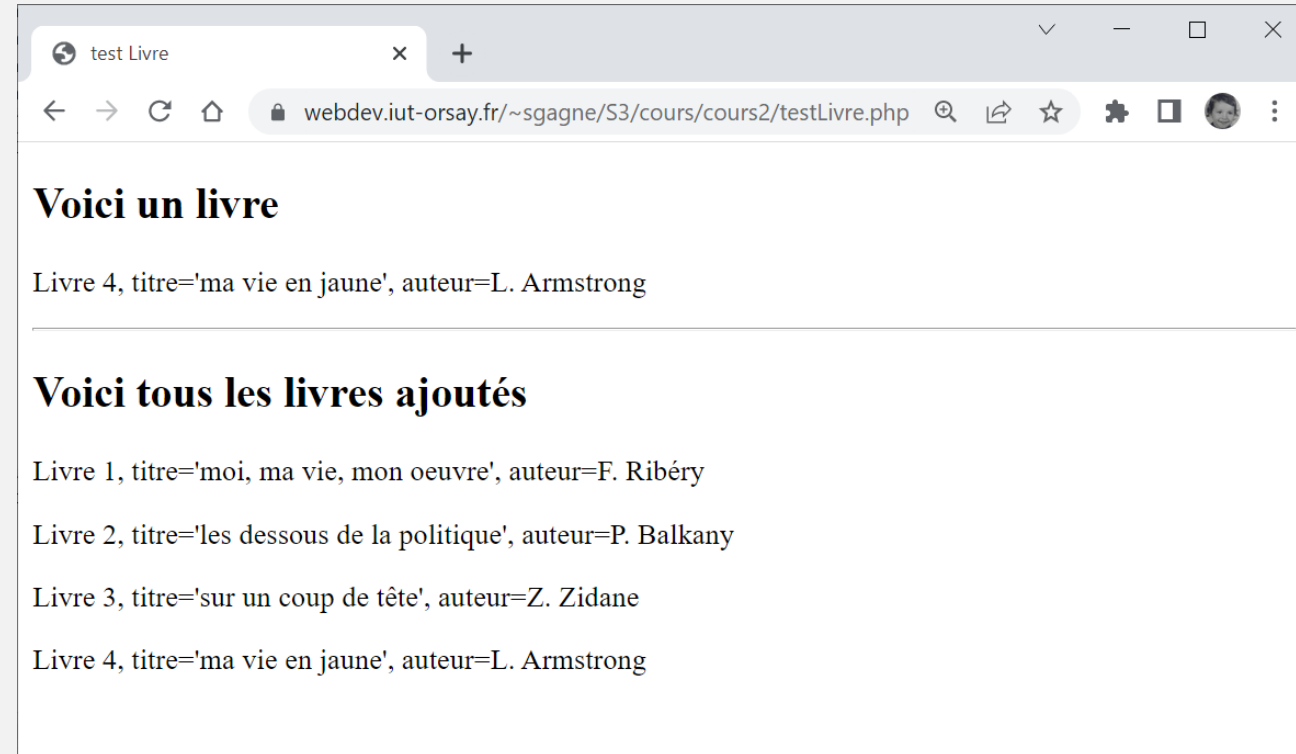
```
testLivre.php
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16      // appel à une méthode d'instance
17      echo "<h2>Voici un livre</h2>";
18      $livre4->afficher();
19      echo "<hr>";
20      // appel à une méthode (static) de la classe Livre
21      Livre::ajouterLivre($livre1);
22      Livre::ajouterLivre($livre2);
23      Livre::ajouterLivre($livre3);
24      Livre::ajouterLivre($livre4);
25      // appel à une autre méthode de la classe Livre
26      echo "<h2>Voici tous les livres ajoutés</h2>";
27      Livre::afficherLivres();
28    ?>
29  </body>
30 </html>
```



- Aucun lien avec une base de données !

Utilisation de la classe Livre

```
testLivre.php
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16      // appel à une méthode d'instance
17      echo "<h2>Voici un livre</h2>";
18      $livre4->afficher();
19      echo "<hr>";
20      // appel à une méthode (static) de la classe Livre
21      Livre::ajouterLivre($livre1);
22      Livre::ajouterLivre($livre2);
23      Livre::ajouterLivre($livre3);
24      Livre::ajouterLivre($livre4);
25      // appel à une autre méthode de la classe Livre
26      echo "<h2>Voici tous les livres ajoutés</h2>";
27      Livre::afficherLivres();
28    ?>
29  </body>
30 </html>
```

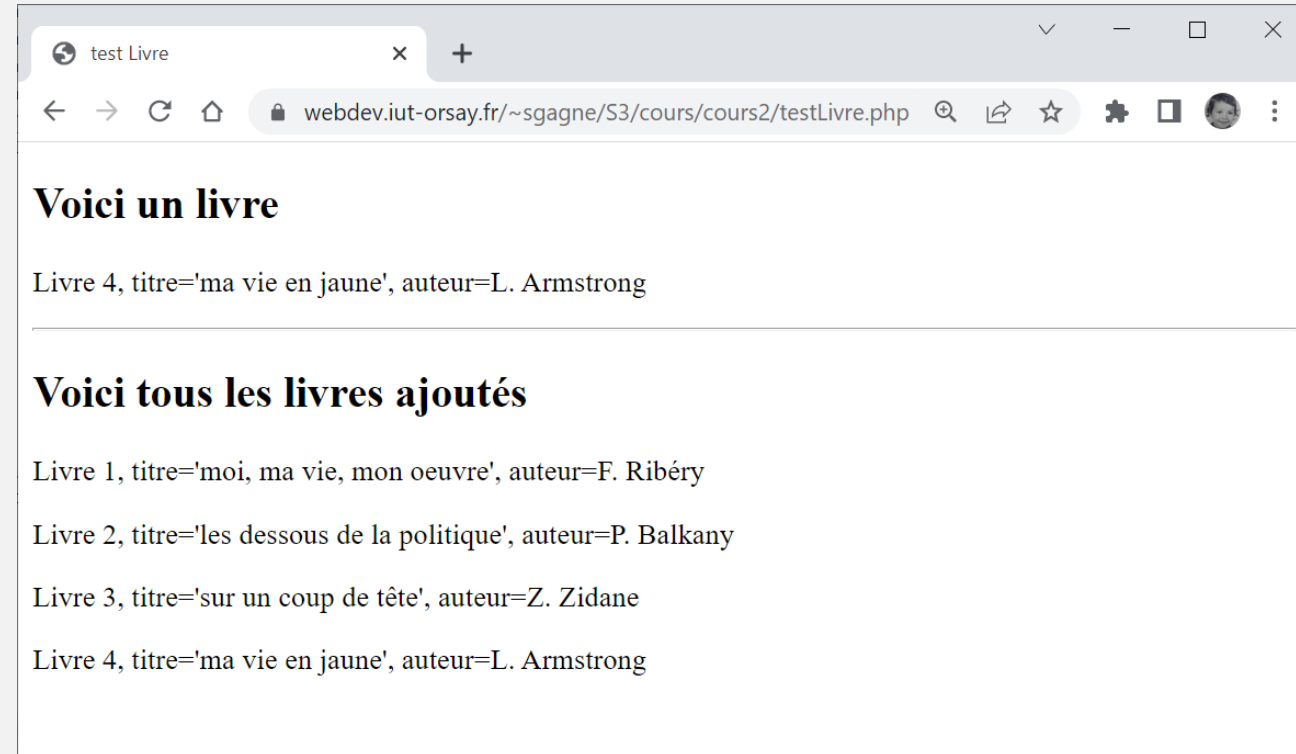


- Aucun lien avec une base de données !

➡ Intérêt très limité !

Utilisation de la classe Livre

```
testLivre.php
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>test Livre</title>
6   </head>
7   <body>
8     <?php
9       // insertion de la classe Livre
10      require_once("livre.php");
11      // construction d'instances de la classe Livre
12      $livre1 = new Livre(1,"moi, ma vie, mon oeuvre","F. Ribéry");
13      $livre2 = new Livre(2,"les dessous de la politique","P. Balkany");
14      $livre3 = new Livre(3,"sur un coup de tête","Z. Zidane");
15      $livre4 = new Livre(4,"ma vie en jaune","L. Armstrong");
16      // appel à une méthode d'instance
17      echo "<h2>Voici un livre</h2>";
18      $livre4->afficher();
19      echo "<hr>";
20      // appel à une méthode (static) de la classe Livre
21      Livre::ajouterLivre($livre1);
22      Livre::ajouterLivre($livre2);
23      Livre::ajouterLivre($livre3);
24      Livre::ajouterLivre($livre4);
25      // appel à une autre méthode de la classe Livre
26      echo "<h2>Voici tous les livres ajoutés</h2>";
27      Livre::afficherLivres();
28    ?>
29  </body>
30 </html>
```



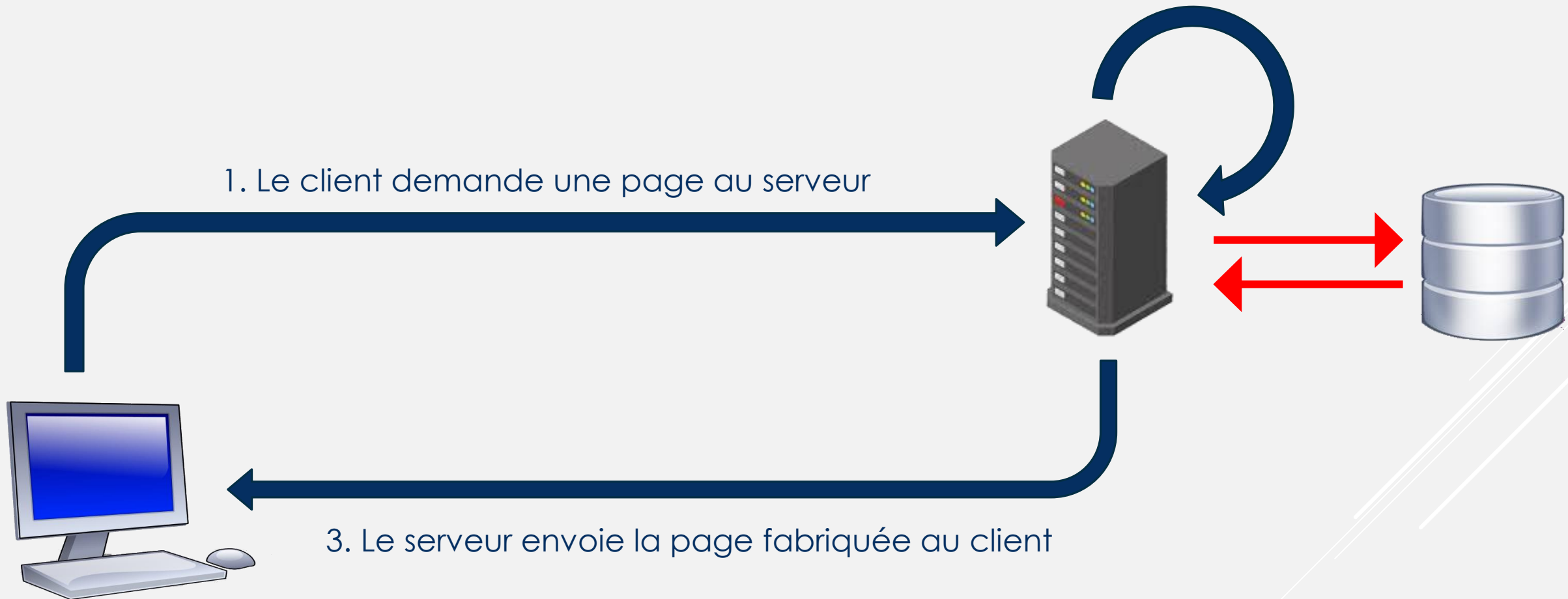
- Aucun lien avec une base de données !

➡ Intérêt très limité !

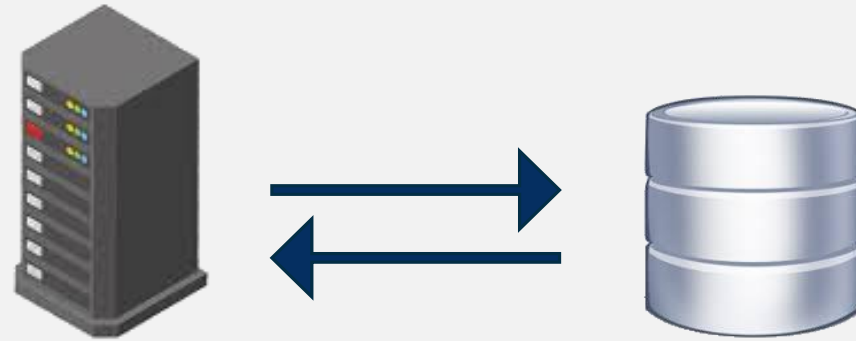
- L'idéal : récupérer les données d'une BD pour construire les objets...

Principe de l'appel serveur en lien avec une BD

2. Le serveur génère la page html / css, en utilisant des données **récupérées de la BD**



Principe de l'appel serveur en lien avec une BD



- Le script PHP doit permettre la **connexion** à la base de données (phase essentielle).
- Cette connexion permettra de **recupérer**, mais aussi d'**insérer**, **modifier** ou **supprimer** des données.
- Grâce aux données récupérées, on pourra **construire des objets PHP** par une utilisation adaptée du constructeur.
- Le pilier de la programmation web : le lien avec des **données persistantes**.

UTILISATION D'UNE BASE DE DONNÉES

Connexion à la base

Connexion à la base de données

- On a plusieurs outils pour se connecter à une base de données.
- On peut notamment utiliser la classe PHP **PDO** (**P**HP **D**ata **O**bjects).
- Une instance de la classe PDO permet la connexion avec une base de données de type MySQL ou **MariaDB**.
- La classe PDO fournit de nombreuses méthodes statiques qui permettent d'utiliser simplement des fonctionnalités essentielles à la gestion et à la persistance des données, au sein du code PHP.
- On peut utiliser l'interface **phpMyAdmin** qui est un outil pratique et simple pour structurer la base de données qui sera utilisée dans le code PHP.

Interface classique phpMyAdmin

The screenshot shows the phpMyAdmin interface in a web browser. The address bar displays the URL: `webdev.iut-orsay.fr/phpmyadmin/sql.php?server=1&db=sgagne&table=livre&pos=0`. The left sidebar shows the database structure with 'sgagne' selected and 'livre' highlighted. The main panel shows the 'Table: livre' view with a toolbar containing 'Parcourir', 'Structure', 'SQL', 'Rechercher', 'Insérer', 'Exporter', 'Importer', 'Privileges', and 'Plus'. A green status bar indicates 'Affichage des lignes 0 - 3 (total de 4, traitement en 0.0004 seconde(s).)'. The SQL query `SELECT * FROM `livre`` is entered in the query box. Below the query box, there are options for 'Profilage', 'Éditer en ligne', 'Éditer', 'Expliquer SQL', 'Créer le code source PHP', and 'Actualiser'. The 'Options' section shows 'Tout afficher', 'Nombre de lignes: 25', 'Filtrer les lignes: Chercher dans cette table', and 'Trier par clé: Aucun(e)'. The results table has columns 'num', 'titre', and 'auteur'. The data rows are:

num	titre	auteur
1	moi, ma vie, mon oeuvre	F. Ribéry
2	comment réussir sa vie	P. Balkany
3	ma vie en jaune	L. Armstrong
4	sur un coup de tête	Z. Zidane

At the bottom, there is a section for 'Opérations sur les résultats de la requête' with a 'Console de requêtes SQL' button.

Connexion à la base de données – un premier exemple de code

```
testConnexion.php
1  <?php
2      // création de 4 variables pour les éléments essentiels de connexion
3      $hostname = 'localhost';
4      $database = '*****';          // votre id court
5      $login = '*****';              // votre id court
6      $password = '*****';          // votre mdp changé selon la procédure db_informations.txt
7
8      // création d'un tableau associatif pour le paramétrage UTF-8.
9      // avec cet argument, toutes les chaînes de caractères
10     // en entrée et sortie de la base seront dans le codage UTF-8
11     $tabUTF8 = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
12
13     try {
14         // création d'un objet instance de la classe PDO qui matérialise la connexion.
15         $pdo = new PDO("mysql:host=$hostname;dbname=$database",$login,$password,$tabUTF8);
16
17         // On active le mode d'affichage des erreurs, et le lancement d'exception en cas d'erreur
18         $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
19
20     } catch(PDOException $e) {
21         echo "erreur de connexion : ".$e->getMessage()."<br>";
22     }
23     ?>
```

Connexion à la base de données – organisation du code – PHP objet

```
connexion.php
1  <?php
2  class Connexion {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27  }
28  ?>
```

Connexion à la base de données – organisation du code – PHP objet

```
connexion.php
1  <?php
2  class Connexion {
3      // les attributs static caractéristiques de la connexion
4      static private $hostname = 'localhost';
5      static private $database = '*****'; // votre id court
6      static private $login = '*****'; // votre id court
7      static private $password = '*****'; // votre mdp
8
9      static private $tabUTF8 = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27  }
28  ?>
```

Connexion à la base de données – organisation du code – PHP objet

```
connexion.php
1  <?php
2  class Connexion {
3      // les attributs static caractéristiques de la connexion
4      static private $hostname = 'localhost';
5      static private $database = '*****'; // votre id court
6      static private $login = '*****'; // votre id court
7      static private $password = '*****'; // votre mdp
8
9      static private $tabUTF8 = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
10
11     // l'attribut static qui matérialisera la connexion
12     static private $pdo;
13
14     // le getter public de cet attribut
15     static public function pdo() {return self::$pdo;}
16
17
18
19
20
21
22
23
24
25
26
27 }
28 ?>
```

Connexion à la base de données – organisation du code – PHP objet

```
connexion.php
1  <?php
2  class Connexion {
3      // les attributs static caractéristiques de la connexion
4      static private $hostname = 'localhost';
5      static private $database = '*****'; // votre id court
6      static private $login = '*****'; // votre id court
7      static private $password = '*****'; // votre mdp
8
9      static private $tabUTF8 = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
10
11     // l'attribut static qui matérialisera la connexion
12     static private $pdo;
13
14     // le getter public de cet attribut
15     static public function pdo() {return self::$pdo;}
16
17     // la fonction static de connexion qui initialise $pdo et lance la tentative de connexion
18     static public function connect() {
19         $h = self::$hostname; $d = self::$database; $l = self::$login; $p = self::$password; $t = self::$tabUTF8;
20         try {
21             self::$pdo = new PDO("mysql:host=$h;dbname=$d",$l,$p,$t);
22             self::$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
23         } catch(PDOException $e) {
24             echo "erreur de connexion : ".$e->getMessage()."<br>";
25         }
26     }
27 }
28 ?>
```

Connexion à la base de données – organisation du code – PHP objet

testConnexion.php

```
1  <?php
2      // insertion de la classe Connexion
3      require_once("connexion.php");
4
5      // appel de la fonction static de connexion à la base
6      Connexion::connect();
7  ?>
```


UTILISATION D'UNE BASE DE DONNÉES

Premières requêtes – récupérer des données et construire des objets PHP

Préliminaire important – modification du constructeur

Préliminaire important – modification du constructeur

- En PHP, on ne dispose que d'un seul constructeur, contrairement à beaucoup d'autres langages objet.

Préliminaire important – modification du constructeur

- En PHP, on ne dispose que d'un seul constructeur, contrairement à beaucoup d'autres langages objet.

```
// constructeur d'un objet Livre  
public function __construct($num, $titre, $auteur) {  
    $this->num = $num;  
    $this->titre = $titre;  
    $this->auteur = $auteur;  
}
```

Préliminaire important – modification du constructeur

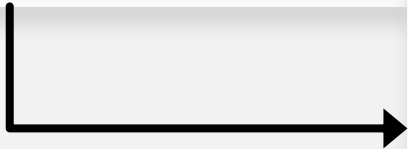
- En PHP, on ne dispose que d'un seul constructeur, contrairement à beaucoup d'autres langages objet.
- Mais il y a une technique simple pour différencier la façon d'appeler le constructeur. En gros, données en provenance de la BD ou non.

```
// constructeur d'un objet Livre  
public function __construct($num, $titre, $auteur) {  
    $this->num = $num;  
    $this->titre = $titre;  
    $this->auteur = $auteur;  
}
```

Préliminaire important – modification du constructeur

- En PHP, on ne dispose que d'un seul constructeur, contrairement à beaucoup d'autres langages objet.
- Mais il y a une technique simple pour différencier la façon d'appeler le constructeur. En gros, données en provenance de la BD ou non.
- Ainsi, le constructeur (unique) va avoir deux comportements différents en fonction de la provenance des données.

```
// constructeur d'un objet Livre
public function __construct($num, $titre, $auteur) {
    $this->num = $num;
    $this->titre = $titre;
    $this->auteur = $auteur;
}
```

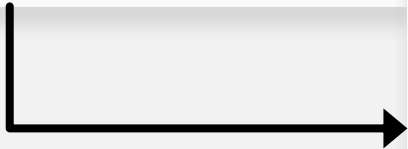


```
// constructeur polyvalent d'un objet Livre
public function __construct($num = NULL, $titre = NULL, $auteur = NULL) {
    if (!is_null($num)) {
        $this->num = $num;
        $this->titre = $titre;
        $this->auteur = $auteur;
    }
}
```

Préliminaire important – modification du constructeur

- En PHP, on ne dispose que d'un seul constructeur, contrairement à beaucoup d'autres langages objet.
- Mais il y a une technique simple pour différencier la façon d'appeler le constructeur. En gros, données en provenance de la BD ou non.
- Ainsi, le constructeur (unique) va avoir deux comportements différents en fonction de la provenance des données.

```
// constructeur d'un objet Livre
public function __construct($num, $titre, $auteur) {
    $this->num = $num;
    $this->titre = $titre;
    $this->auteur = $auteur;
}
```



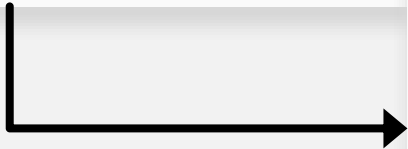
```
// constructeur polyvalent d'un objet Livre
public function __construct($num = NULL, $titre = NULL, $auteur = NULL) {
    if (!is_null($num)) {
        $this->num = $num;
        $this->titre = $titre;
        $this->auteur = $auteur;
    }
}
```

- Si on passe des paramètres au constructeur, alors il les utilise,

Préliminaire important – modification du constructeur

- En PHP, on ne dispose que d'un seul constructeur, contrairement à beaucoup d'autres langages objet.
- Mais il y a une technique simple pour différencier la façon d'appeler le constructeur. En gros, données en provenance de la BD ou non.
- Ainsi, le constructeur (unique) va avoir deux comportements différents en fonction de la provenance des données.

```
// constructeur d'un objet Livre
public function __construct($num, $titre, $auteur) {
    $this->num = $num;
    $this->titre = $titre;
    $this->auteur = $auteur;
}
```



```
// constructeur polyvalent d'un objet Livre
public function __construct($num = NULL, $titre = NULL, $auteur = NULL) {
    if (!is_null($num)) {
        $this->num = $num;
        $this->titre = $titre;
        $this->auteur = $auteur;
    }
}
```

- Si on passe des paramètres au constructeur, alors il les utilise,
- Sinon, les données de la BD seront utilisées.

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2  // insertion des classes Livre et Connexion
3  require_once("livre.php");
4  require_once("connexion.php");
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25  ?>
```

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2      // insertion des classes Livre et Connexion
3      require_once("livre.php");
4      require_once("connexion.php");
5
6      // lancement de la méthode statique Connexion::connect()
7      // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8      Connexion::connect();
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25  ?>
```

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2    // insertion des classes Livre et Connexion
3    require_once("livre.php");
4    require_once("connexion.php");
5
6    // lancement de la méthode statique Connexion::connect()
7    // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8    Connexion::connect();
9
10   // une première requête
11   $requete = "SELECT * FROM livre;";
12
13
14
15
16
17
18
19
20
21
22
23
24
25  ?>
```

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2      // insertion des classes Livre et Connexion
3      require_once("livre.php");
4      require_once("connexion.php");
5
6      // lancement de la méthode statique Connexion::connect()
7      // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8      Connexion::connect();
9
10     // une première requête
11     $requete = "SELECT * FROM livre;";
12
13     // on lance la requête par l'attribut statique $pdo de la classe Connexion
14     // et on récupère le résultat sous forme d'un objet PDOStatement
15     $resultat = Connexion::pdo()->query($requete);
16
17
18
19
20
21
22
23
24
25  ?>
```

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2      // insertion des classes Livre et Connexion
3      require_once("livre.php");
4      require_once("connexion.php");
5
6      // lancement de la méthode statique Connexion::connect()
7      // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8      Connexion::connect();
9
10     // une première requête
11     $requete = "SELECT * FROM livre;";
12
13     // on lance la requête par l'attribut statique $pdo de la classe Connexion
14     // et on récupère le résultat sous forme d'un objet PDOStatement
15     $resultat = Connexion::pdo()->query($requete);
16
17     // la variable $resultat a besoin d'un traitement avant d'être exploitée
18     // 1. On indique que chaque ligne de la table de données doit être vue comme une instance de "Livre"
19     $resultat->setFetchmode(PDO::FETCH_CLASS,"Livre");
20
21
22
23
24
25  ?>
```

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2      // insertion des classes Livre et Connexion
3      require_once("livre.php");
4      require_once("connexion.php");
5
6      // lancement de la méthode statique Connexion::connect()
7      // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8      Connexion::connect();
9
10     // une première requête
11     $requete = "SELECT * FROM livre;";
12
13     // on lance la requête par l'attribut statique $pdo de la classe Connexion
14     // et on récupère le résultat sous forme d'un objet PDOStatement
15     $resultat = Connexion::pdo()->query($requete);
16
17     // la variable $resultat a besoin d'un traitement avant d'être exploitée
18     // 1. On indique que chaque ligne de la table de données doit être vue comme une instance de "Livre"
19     $resultat->setFetchmode(PDO::FETCH_CLASS,"Livre");
20     // 2. on recueille les objets créés dans un tableau d'objets PHP
21     $tableau = $resultat->fetchAll();
22
23
24
25  ?>
```

Première requête – une requête SELECT

```
lireLivres.php
1  <?php
2      // insertion des classes Livre et Connexion
3      require_once("livre.php");
4      require_once("connexion.php");
5
6      // lancement de la méthode statique Connexion::connect()
7      // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8      Connexion::connect();
9
10     // une première requête
11     $requete = "SELECT * FROM livre;";
12
13     // on lance la requête par l'attribut statique $pdo de la classe Connexion
14     // et on récupère le résultat sous forme d'un objet PDOStatement
15     $resultat = Connexion::pdo()->query($requete);
16
17     // la variable $resultat a besoin d'un traitement avant d'être exploitée
18     // 1. On indique que chaque ligne de la table de données doit être vue comme une instance de "Livre"
19     $resultat->setFetchmode(PDO::FETCH_CLASS,"Livre");
20     // 2. on recueille les objets créés dans un tableau d'objets PHP
21     $tableau = $resultat->fetchAll();
22
23     // affichage du tableau
24     echo "<pre>"; print_r($tableau); echo "</pre>";
25     ?>
```

Première requête – une requête SELECT

lireLivres.php

```
1 <?php
2 // insertion des classes Livre et Connexion
3 require_once("livre.php");
4 require_once("connexion.php");
5
6 // lancement de la méthode statique Connexion::connect()
7 // qui crée une instance d'objet PDO stockée dans l'attribut statique $pdo
8 Connexion::connect();
9
10 // une première requête
11 $requete = "SELECT * FROM livre;";
12
13 // on lance la requête par l'attribut statique $pdo de la classe Connexion
14 // et on récupère le résultat sous forme d'un objet PDOStatement
15 $resultat = Connexion::pdo()->query($requete);
16
17 // la variable $resultat a besoin d'un traitement avant d'être exploitée
18 // 1. On indique que chaque ligne de la table de données doit être vue comme une
19 $resultat->setFetchmode(PDO::FETCH_CLASS,"Livre");
20 // 2. on recueille les objets créés dans un tableau d'objets PHP
21 $tableau = $resultat->fetchAll();
22
23 // affichage du tableau
24 echo "<pre>"; print_r($tableau); echo "</pre>";
25 ?>
```

Array
(
 [0] => Livre Object
 (
 [titre:Livre:private] => moi, ma vie, mon oeuvre
 [auteur:Livre:private] => F. Ribéry
 [num:Livre:private] => 1
)
 [1] => Livre Object
 (
 [titre:Livre:private] => comment réussir sa vie
 [auteur:Livre:private] => P. Balkany
 [num:Livre:private] => 2
)
 [2] => Livre Object
 (
 [titre:Livre:private] => ma vie en jaune
 [auteur:Livre:private] => L. Armstrong
 [num:Livre:private] => 3
)
 [3] => Livre Object
 (
 [titre:Livre:private] => sur un coup de tête
 [auteur:Livre:private] => Z. Zidane
 [num:Livre:private] => 4
)
 [4] => Livre Object
 (
 [titre:Livre:private] => la loi du préau
 [auteur:Livre:private] => ZEP
 [num:Livre:private] => 5
)
)

Amélioration du code de la classe Livre

- Les lignes de code qui permettent de récupérer les livres de la base sont encapsulées dans une méthode, nommée ici getAllLivres().

```
Livre.php
1  <?php
2  class Livre {
3      // attributs d'une instance de la classe Livre
4      private $titre;
5      private $auteur;
6      private $num;
7
8      // constructeur polyvalent d'un objet Livre
9      public function __construct($num = NULL, $titre = NULL, $auteur = NULL) {
10         if (!is_null($num)) {
11             $this->num = $num;
12             $this->titre = $titre;
13             $this->auteur = $auteur;
14         }
15     }
16
17     // un exemple de getter et de setter - méthodes d'instance
18     public function getTitre() {return $this->titre;}
19     public function setAuteur($a) {$this->auteur = $a;}
20
21     // une méthode d'affichage - méthode d'instance
22     public function afficher() {
23         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
24     }
25
26     // une méthode static qui retourne le tableau des livres de la base
27     public static function getAllLivres() {
28         $requete = "SELECT * FROM livre;";
29         $resultat = Connexion::pdo()->query($requete);
30         $resultat->setFetchmode(PDO::FETCH_CLASS, "Livre");
31         $tableau = $resultat->fetchAll();
32         return $tableau;
33     }
34 }
35 ?>
```

Amélioration du code de la classe Livre

- Les lignes de code qui permettent de récupérer les livres de la base sont encapsulées dans une méthode, nommée ici `getAllLivres()`.
- Cette méthode retourne le tableau des objets Livre représentés dans la base.

lireLivres2.php

```
1 <?php
2 // insertion des classes Livre et Connexion
3 require_once("livre.php");
4 require_once("connexion.php");
5
6 // connexion
7 Connexion::connect();
8
9 // méthode qui récupère les livres de la BD sous forme d'objets PHP
10 $tableau = Livre::getAllLivres();
11
12 // affichage du tableau retourné par la méthode getAllLivres()
13 echo "<pre>"; print_r($tableau); echo "</pre>";
14 ?>
```

Livre.php

```
1 <?php
2 class Livre {
3     // attributs d'une instance de la classe Livre
4     private $titre;
5     private $auteur;
6     private $num;
7
8     // constructeur polyvalent d'un objet Livre
9     public function __construct($num = NULL, $titre = NULL, $auteur = NULL) {
10         if (!is_null($num)) {
11             $this->num = $num;
12             $this->titre = $titre;
13             $this->auteur = $auteur;
14         }
15     }
16
17     // un exemple de getter et de setter - méthodes d'instance
18     public function getTitre() {return $this->titre;}
19     public function setAuteur($a) {$this->auteur = $a;}
20
21     // une méthode d'affichage - méthode d'instance
22     public function afficher() {
23         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
24     }
25
26     // une méthode static qui retourne le tableau des livres de la base
27     public static function getAllLivres() {
28         $requete = "SELECT * FROM livre;";
29         $resultat = Connexion::pdo()->query($requete);
30         $resultat->setFetchmode(PDO::FETCH_CLASS, "Livre");
31         $tableau = $resultat->fetchAll();
32         return $tableau;
33     }
34 }
35 ?>
```

Amélioration du code de la classe Livre

- Les lignes de code qui permettent de récupérer les livres de la base sont encapsulées dans une méthode, nommée ici `getAllLivres()`.
- Cette méthode retourne le tableau des objets Livre représentés dans la base.
- On peut alors évoquer cette méthode dans notre fichier `lireLivres.php`.

lireLivres2.php

```
1 <?php
2 // insertion des classes Livre et Connexion
3 require_once("livre.php");
4 require_once("connexion.php");
5
6 // connexion
7 Connexion::connect();
8
9 // méthode qui récupère les livres de la BD sous forme d'objets PHP
10 $tableau = Livre::getAllLivres();
11
12 // affichage du tableau retourné par la méthode getAllLivres()
13 echo "<pre>"; print_r($tableau); echo "</pre>";
14 ?>
```

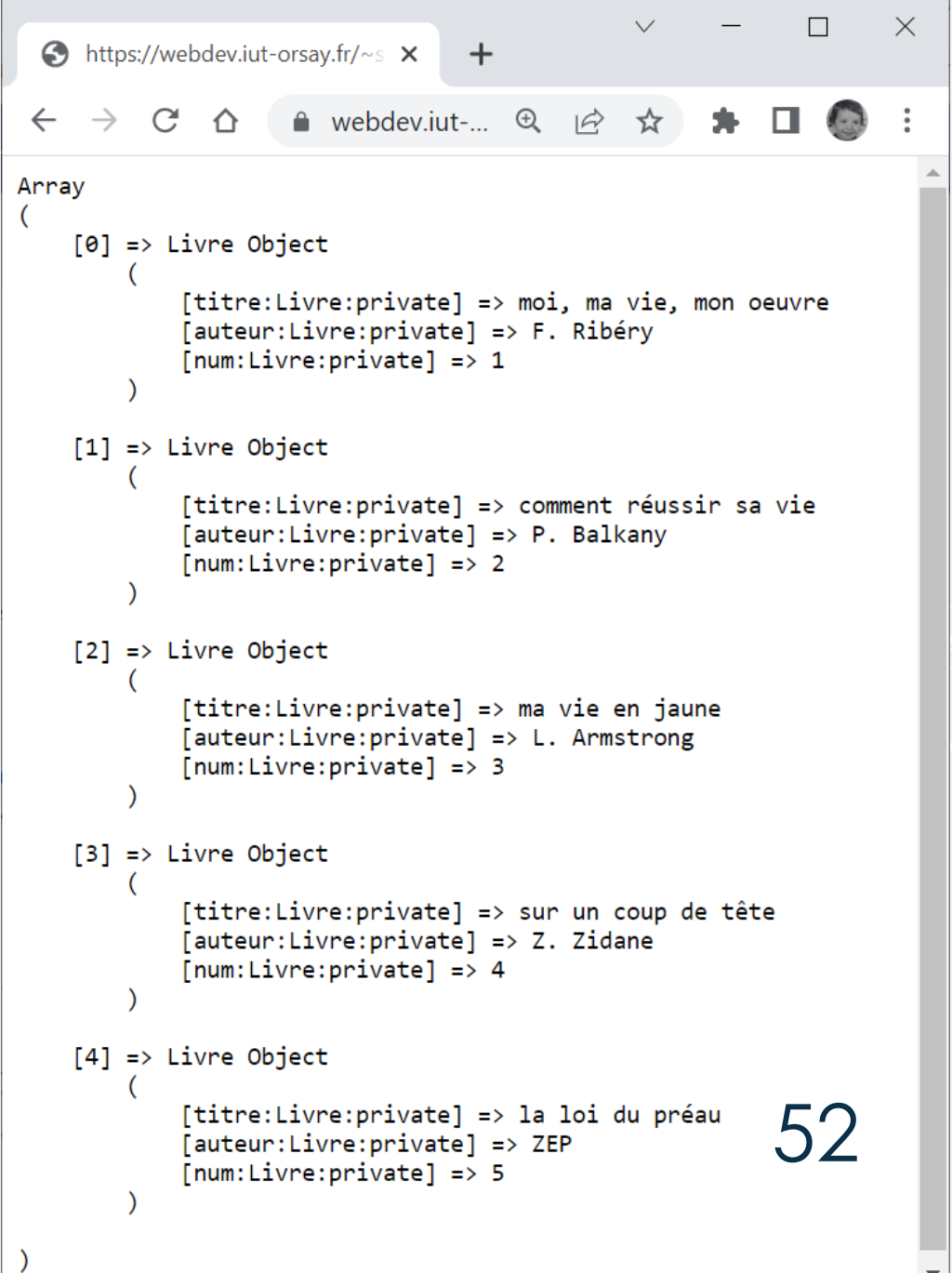
Livre.php

```
1 <?php
2 class Livre {
3     // attributs d'une instance de la classe Livre
4     private $titre;
5     private $auteur;
6     private $num;
7
8     // constructeur polyvalent d'un objet Livre
9     public function __construct($num = NULL, $titre = NULL, $auteur = NULL) {
10         if (!is_null($num)) {
11             $this->num = $num;
12             $this->titre = $titre;
13             $this->auteur = $auteur;
14         }
15     }
16
17     // un exemple de getter et de setter - méthodes d'instance
18     public function getTitre() {return $this->titre;}
19     public function setAuteur($a) {$this->auteur = $a;}
20
21     // une méthode d'affichage - méthode d'instance
22     public function afficher() {
23         echo "<p>Livre $this->num, titre='$this->titre', auteur=$this->auteur</p>";
24     }
25
26     // une méthode static qui retourne le tableau des livres de la base
27     public static function getAllLivres() {
28         $requete = "SELECT * FROM livre;";
29         $resultat = Connexion::pdo()->query($requete);
30         $resultat->setFetchmode(PDO::FETCH_CLASS, "Livre");
31         $tableau = $resultat->fetchAll();
32         return $tableau;
33     }
34 }
35 ?>
```

Amélioration du code de la classe Livre

- Les lignes de code qui permettent de récupérer les livres de la base sont encapsulées dans une méthode, nommée ici `getAllLivres()`.
- Cette méthode retourne le tableau des objets Livre représentés dans la base.
- On peut alors évoquer cette méthode dans notre fichier `lireLivres.php`.

```
lireLivres2.php
1  <?php
2  // insertion des classes Livre et Connexion
3  require_once("livre.php");
4  require_once("connexion.php");
5
6  // connexion
7  Connexion::connect();
8
9  // méthode qui récupère les livres de la BD sous forme d'objets PHP
10 $tableau = Livre::getAllLivres();
11
12 // affichage du tableau retourné par la méthode getAllLivres()
13 echo "<pre>"; print_r($tableau); echo "</pre>";
14 ?>
```



```
Array
(
    [0] => Livre Object
        (
            [titre:Livre:private] => moi, ma vie, mon oeuvre
            [auteur:Livre:private] => F. Ribéry
            [num:Livre:private] => 1
        )
    [1] => Livre Object
        (
            [titre:Livre:private] => comment réussir sa vie
            [auteur:Livre:private] => P. Balkany
            [num:Livre:private] => 2
        )
    [2] => Livre Object
        (
            [titre:Livre:private] => ma vie en jaune
            [auteur:Livre:private] => L. Armstrong
            [num:Livre:private] => 3
        )
    [3] => Livre Object
        (
            [titre:Livre:private] => sur un coup de tête
            [auteur:Livre:private] => Z. Zidane
            [num:Livre:private] => 4
        )
    [4] => Livre Object
        (
            [titre:Livre:private] => la loi du préau
            [auteur:Livre:private] => ZEP
            [num:Livre:private] => 5
        )
)
```