XML et PHP

Technique 1: gestion manuelle (Bohf!)

- Idée =
  - XML en sortie : générer directement le fichier XML soit sur la sortie standard (via echo), soit dans un fichier,
  - XML en entrée: lire le fichier XML à partir des instructions de base de lecture d'un fichier.
- Inconvénients =
  - Pas malléable,
  - Très adHoc,
  - Obligation d'utiliser de la programmation d'assez-bas niveau.

Exemple 1:

```
<?php
header('Content-type: text/xml');
echo '<?xml version="1.0" encoding="ISO-8859-1"?>';
$mat_dist=array("paris" => array("nice" => 900, "lille" => 300),
                "nice" => array("paris" => 900, "tours" => 800, "lille" => 1200),
                 "tours" => array("nice" => 800, "lille" => 500),
                  "lille" => array("paris" => 300, "nice" => 1200, "tours" => 800));
echo "<distance_entre_ville>\n";
foreach ($mat_dist as $ville_origine => $destinations )
  foreach ($mat_dist[$ville_origine] as $ville_arrivee => $distance){
        $ville_origine = htmlspecialchars($ville_origine);
        $ville_arrivee = htmlspecialchars($ville_arrivee);
        $distance = htmlspecialchars($distance);
        echo " <liaison>\n";
        echo " <origine>$ville_origine</origine>\n";
        echo " <destination>$ville_arrivee</destination>\n";
        echo " <distance>$distance</distance>\n";
        echo " </liaison>\n";
}
echo "</distance_entre_ville>\n";
?>
```

htmlspecialchars(string)
encode en XML certains caractères qui sont des "méta" en HTML
par exemple remplace < par >

Fichier XML créé avec exemple 1

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<distance_entre_ville>
<liaison>
<origine>paris</origine>
<destination>nice</destination>
 <distance>900</distance>
</liaison>
<liaison>
<origine>paris</origine>
<destination>lille</destination>
 <distance>300</distance>
</liaison>
 <liaison>
<origine>nice</origine>
<destination>paris</destination>
<distance>900</distance> </liaison>
.....
 </distance_entre_ville>
```

Exemple 2 : BD vers XML manuellement

```php
<?php
$user = 'toto';  $password = 'secretoto';  $database = 'nosamisleschiens';  $table = 'chien';
$connect = mysql_connect('localhost',$user, $password)  or die ("erreur de connexion");
mysql_select_db($database, $connect)     or die ("erreur de connexion base");
$fic = fopen("rep/export.xml", "w");
fwrite($fic, '<?xml version="1.0"  encoding="ISO-8859-1"?>');
fwrite($fic, '<!-- DB to XML -->');
fwrite($fic, "<$table>");
$elements =array();
$result = mysql_query("describe $table"); // describe = donne nom de colonne
while ($row = mysql_fetch_array($result))
   $elements[] = $row[0];
$result = mysql_query("select * from $table");
$i = 0;
while ($row = mysql_fetch_array($result))
{ $i++;
  fwrite($fic, "<row num=\"$i\">");
  for ($j = 0; $j<count($elements) ; $j++)
     fwrite($fic, "<$elements[$j]>".htmlspecialchars($row[$j]) ."</$elements[$j]>") ;
  fwrite($fic, "</row>");
}
fwrite($fic, "</$table>");
mysql_close();
fclose($fic);
?>
```

Idée 2 = SimpleXML

- Disponible depuis PHP 5 :
  - présent par défaut ;
  - avant aucun outil de manipulation simple.
- SimpleXML = une interface PHP <-> XML
  - Adapté pour lecture et modification de fichiers simples.
- En background,
  - le DOM (Document Object Model) mais il n'est pas omniprésent,
  - la programmation objet.

simplexml_load_string

```
<?php
$string = "<?xml version='1.0'?>
<document>
 <title>hello</title>
 <from>Joe</from>
 <to>Jane</to>
 <body>
 Hello Jane !
 </body>
</document>" ; //fin de la chaîne XML
$xml = simplexml_load_string($string);

var_dump($xml); //affiche le contenu de la variable
?>
```

Exécution :

```
object(SimpleXMLElement)#1 (4) {
 ["title"]=>  string(5) "hello"
 ["from"]=>  string(3) "Joe"
 ["to"]=>  string(4) "Jane"
 ["body"]=>  string(16) " Hello Jane ! "
}
```

SimpleXML (PHP5)

```
<?xml version="1.0" encoding="utf-8"?>
<bibliotheque>
  <style id="roman">
    <livre>
      <titre>La fortune des Rougon</titre>
      <auteur>Emile Zola</auteur>
    </livre>
    <livre>
      <titre>Hernani</titre>
      <auteur>Victor Hugo</auteur>
    </livre>
  </style>
  <style id="fiction">
    <livre>
      <titre>Le seigneur des anneaux</titre>
      <auteur>J.R.R. Tolkien</auteur>
    </livre>
  </style>
</bibliotheque>
```

Import d'un fichier XML

simplexml_load_file(chaine décrivant le nom du fichier);

```
$bibliotheque = simplexml_load_file('livres.xml');
```

- Le fichier est lu en entier et
- Le contenu XML est validé par la bibliothèque libxml2.
- Si le document n'est pas compatible XML, alors
  - les erreurs d'analyse sont affichées dans la page,
  - et $bibliotheque reçoit le booléen false..

Exemple : simplexml_load_file

```php
<?php
$bibliotheque = simplexml_load_file('livres.xml');

foreach ($bibliotheque->style as $style) {
   echo "Type {$style['id']} <br/>";
   foreach ($style->livre as $livre) {
      echo "Titre : {$livre->titre} <br/>";
      echo "Auteur : {$livre->auteur} <br/>";
    }
 }
?>
```

Exécution :

Type roman
Titre : La fortune des Rougon
Auteur : Emile Zola
Titre : Hernani
Auteur : Victor Hugo
Type fiction
Titre : Le seigneur des anneaux
Auteur : J.R.R. Tolkien

Accès aux éléments XML

- La ressource retournée par simplexml_load_file est un objet qui représente l'élément racine du document xml.
- Accès :
  - Le contenu de chaque sous-élément est accessible à partir de son élément parent, comme attribut de cet élément.
  - Les sous-éléments de même nom sont regroupés dans un tableau (avec des indices numériques).
- Exemples
  - 1) On a deux balises style, et $bibliotheque contient un membre appelé style, qui est un tableau. On peut donc utiliser foreach() pour parcourir toutes les balises style.
  - 2) Chaque balise style contient deux balises distinctes : auteur et titre. Il suffit de les appeler par leur nom, les données contenues ont été enregistrées comme valeurs, sous forme de chaînes de caractère.

Accès aux attributs

- Les attributs sont aussi décodés et placés dans un tableau particulier. C'est en fait la même variable que l'objet que l'on utilise pour accéder aux différentes sous-balises.
  - *$style->livre* nous renvoie la sous-balise *livre* de balise *style* (ou le tableau s'il y a plusieurs livres).
  - *$style['livre']* nous renverrait la valeur de l'attribut *livre* de la balise *style*, s'il existait.
- SimpleXML utilise simultanément les deux notations : tableau et objet. Cela permet d'identifier rapidement les données manipulées.

attributes()

- Retourne la liste complète des attributs et les valeurs associées.

```
<?php
$string = "<a>
  <foo attribut1='un' attribut2='deux'>1</foo>
</a>";
$xml = simplexml_load_string($string);

foreach($xml->foo[0]->attributes() as $a => $b) {
  echo "$a = $b<br/>";
 }
?>
```

Exécution :
attribut1=un
attribut2=deux

children()

- Retourne la liste complète des éléments enfants et les valeurs associées.

```
<?php
$string = "
<a>
  <b>1</b>
  <b>2</b>
  <b>3</b>
  <b>4</b>
</a>";
$xml = simplexml_load_string($string);

foreach($xml->children() as $a => $b) {
  echo "$a = $b <br/>"; // $b : objet interprété par echo
  }
?>
```

Exécution :

b = 1
b = 2
b = 3
b = 44

SimpleXML et Xpath

- La méthode xpath() retourne un tableau avec le contenu de toutes les balises qui satisfont le chemin indiqué. Les objets sont retournés dans l'ordre où ils sont trouvés, quels que soient les éléments parents.

```
<?php
$string = <<<XML
<a>
  <b>
    <c>autre texte</c>
    <c>
      <d>texte</d>
      <d>texte2</d>
    </c>
  </b>
  <d>
     <c>contenu</c>
  </d>
</a>
XML;
$xml = simplexml_load_string($string);
$result = $xml->xpath('/a/b/c/d');
print_r($result);
?>
```

Exécution :

```
Array (
 [0] => SimpleXMLElement Object ( [0] => texte )
 [1] => SimpleXMLElement Object ( [0] => texte2 )
)
```

Exporter du XML

- Une fois qu'un fichier XML a été chargé avec simplexml_load_file ou simplexml_load_string, on peut l'exporter à nouveau en utilisant la méthode asXML().

```
<?php
$xml = <<<XML
<?xml version="1.0" encoding="iso-8859-1"?>
<a><b></b><c /></a>
XML;
$splxml = simplexml_load_string($xml);

file_put_contents('fichier.xml', $splxml ->asxml());
?>
```

Exécution :
Le fichier fichier.xml créé contient :
<?xml version="1.0" encoding="iso-8859-1"?>
<a><b/><c/> ← Observer formatage!

Modifier un fichier XML

```php
<?php
$xml = <<<XML
<?xml version="1.0" encoding="iso-8859-1"?>
<a><b version="1.0" type="classique"></b><c /></a>
XML;
$entries = simplexml_load_string($xml);

//Modification de la valeur d'une balise
$entries->b = 'contenu de b';
$entries->c = 'Contenu avec balises <> ';

//Ajout ou modification d'un attribut
$entries->b['langue'] = 'fr';
$entries->b['type'] = 'classique';

//Suppression d'un attribut
Unset($entries->b['version']);

echo $entries->asxml();
echo htmlspecialchars($entries->asxml());
?>
```

Résultat (reformaté)

```
contenu de bContenu avec balises <>
<?xml version="1.0" encoding="iso-8859-1"?>
<a>
<b type="classique" langue="fr">contenu de b</b>
<c>Contenu avec balises &lt;&gt; </c>
</a>
```

Plus de modifications

- ... via la programmation objet

```
<?php

$myxml = new simpleXMLElement("<a/>") ;
$myxml->addAttribute("val", "1") ;
$myxml->addChild("b","BBB") ;
$myxml->addChild("b","") ;
$myxml->addChild("c") ;
$c = $myxml->c ;
$c->addChild("d","DDD") ;
echo htmlspecialchars($myxml->asxml());
?>
```

Résultat (reformaté)

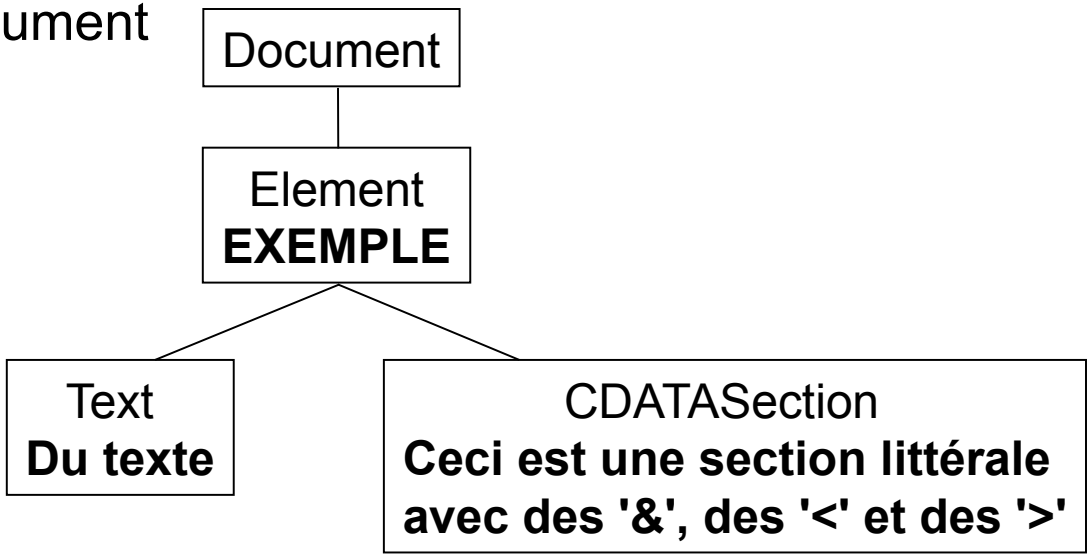
```
<?xml version="1.0"?>
<a val="1">
  <b>BBB</b>
  <b></b>
  <c>
    <d>DDD</d>
  </c>
</a>
```

Idée 3 = Manipuler le DOM

- Valable en PHP 4
- Plus riche que SimpleXML
- Avantage/inconvénient = verbosité
  - clair, facile à relire, mais long et détaillé
  - Permet d'être plus précis que SimpleXML.
- Avantage : Technique non liée à PHP
  - Le DOM est une API (Application Programming Interface) pour les documents HTML valide et les documents XML bien formés.
  - Norme spécifiée par le W3C.

Principe général de l'arbre DOM

- Chargement en mémoire complet du fichier XML sous forme d'un arbre.
- Chaque nœud comporte un des composants de la structure XML
- Types de nœuds courants (tous sous-types de Node) :
  - Element, Text, CDATASection, ProcessingInstruction, Comment, …
- La racine de l'arbre DOM est un nœud spécial, de type *Document*
- La racine DOM contient un seul objet fils de type Element qui est la racine du document



Extension Dom de PHP 5

- L'extension DOM de PHP5 est entièrement objet.
- Les principales classes sont les suivantes :
  - DomNode - objet nœud : documents, éléments, nœuds textuels...
  - DomDocument - objet document (hérite de DomNode)
  - DomElement - objet élément (hérite de DomNode)
  - DomAttr - objet attribut (hérite de DomNode)
  - DomNodeList - objet liste de DomNodes (ce n'est pas un tableau PHP !)
- Travailler avec spécifications pour attributs et méthodes (dans la suite que quelques exemples illustratifs)

Exemple

```
<?xml version="1.0"?>
<continents>
  <europe>
    <pays>France</pays>
    <pays>Belgique</pays>
    <pays>Espagne</pays>
  </europe>
  <asie>
    <pays>Japon</pays>
    <pays>Inde</pays>
  </asie>
</continents>
```

```
<?php
$dom = new DomDocument;
$dom->load("pays.xml");
$listePays = $dom->getElementsByTagName('pays');
foreach($listePays as $pays)
  echo $pays->firstChild->nodeValue . "<br />";
echo "---<br />";
$europe = $dom->getElementsByTagName('europe')->item(0);
$listePaysEurope = $europe->getElementsByTagName('pays');
 foreach($listePaysEurope as $pays)
   echo $pays->firstChild->nodeValue . "<br />";
?>
```

Lire les attributs

```
<?xml version="1.0" ?>
<!DOCTYPE continents SYSTEM "test.dtd">
<continents>
  <europe>
    <pays regime="republique">France</pays>
    <pays regime="monarchie">Belgique</pays>
    <pays regime="monarchie">Espagne</pays>
  </europe>
  <asie>
    <pays regime="empire">Japon</pays>
    <pays>Inde</pays>
  </asie>
</continents>
```

```
<?php
$dom = new DomDocument;
$dom->load("pays2.xml");
  $listePays = $dom->getElementsByTagName("pays");
  foreach($listePays as $pays)
  {
    echo $pays->nodeValue;
    if ($pays->hasAttribute("regime")) {
        echo  " - " . $pays->getAttribute("regime");
    }
    echo "<br />";
  }
?>
```

Chargement ou création d'un arbre Dom

- Chargement d'un fichier XML local
  - <?php $dom->load('fichier.xml'); ?>
- Chargement d'un document XML à partir d'une chaîne de caractères:
  - <?php $dom->loadXML($chaineXML); ?>
- Possibilité
  - d'ouvrir un document HTML grâce à la méthode DomDocument::loadHtmlFile,
  - d'importer un document depuis SimpleXML (grâce à la fonction dom_import_simplexml, cf. plus loin)

- Création d'un document XML vide à partir de rien :
  - <?php $dom->construct(); ?>

Enregistrement d'un fichier XML à partir de DOM

- Enregistrement d'un document XML
  - <?php $dom->save('nouveauFichier.xml'); ?>
- La méthode DomDocument::saveXML renvoie le document comme une chaîne de caractères, ce qui permet de récupérer le document XML dans une variable PHP.
  - <?php $chaineXML = $dom->saveXML(); ?>
  - On peut spécifier en paramètre une référence sur un objet DomNode, afin que seul le sous-arbre ayant cet objet pour racine soit transmis.

Passages Dom <---> SimpleXML

```php
<?php
$dom = new domDocument();
$dom->loadXML('<books><book><title>blah</title></book></books>');
if (!$dom) {
  echo 'Erreur de traitement XML';
  exit;
  }
$s = simplexml_import_dom($dom);
echo $s->book[0]->title;
?>
```

```php
<?php
$s = simplexml_load_string( '<livres><livre><titre>coucou</titre></livre></livres>');
if(!$s) {
  echo "Erreur de traitement du document XML \n";
  exit;
  }
$dom = dom_import_simplexml($s);
print $dom->ownerDocument->saveXML();
```

Ajouter un élément à un fichier XML

```
<?php
$dom = new domDocument();
$dom->loadXML('<livres><livre><titre>truc</titre></livre></livres>');
if (!$dom) {
  echo 'Erreur de traitement XML';
  exit;
  }

$liste = $dom->getElementsByTagName('livres');
$noeud = $liste->item(0);
$title = $dom->createElement("titre","coucou");
$node = $dom->createElement("livre");
$node->appendchild($title);

$noeud->appendChild($node);

$s = simplexml_import_dom($dom);

echo $s->asxml();
?>
```

Validation d'un document xml avec dom

- L'extension DOM autorise de manière très simple la validation d'un document relativement au document DTD spécifié dans le document XML :
  - <?php $dom->validate(); ?>
  - Renvoie *true* en cas de succès, *false* en cas d'échec de la validation.
  - En cas d'échec, des erreurs PHP de niveau *Warning* surviennent, décrivant les dérives par rapport au document de référence.
- Idée = SimpleXML --> DOM --> Validation

Lire un document

- Tous les résultats multiples (comprenant des nœuds) que retourne DOM sont sous la forme d'un objet DomNodeList.
- **Attention : un DomNodeList n'est pas un tableau : on n'accède pas à ses membres avec un index entre crochets.**
- La confusion peut venir du fait qu'un objet de type DomNodeListe peut être utilisé par foreach (car un itérateur a été défini).
- La méthode item() pour parcourir une DomNodeList, qui prend pour unique paramètre un index numérique.
  - <?php $element = $listeElements->item(0); ?>
  - récupère dans *$element* le premier objet pointé par le DomNodeList *$listeElements*. Si on fournit un mauvais index, la méthode ne renvoie rien. Si on exploite le résultat sans prendre de précautions, on récupère une erreur du style :

Notice: Trying to get property of non-object

Trouver des éléments

- On peut récupérer l'élément racine du document (dans ce cas-là, on récupère un objet DomElement, et pas un DomNodeList, puisqu'il n'y a qu'un seul élément racine) :
  - <?php $racine = $dom->documentElement;
    echo $racine->nodeName; ?>
  - les objets DomNode (et par conséquent les objets DomElement) ont une propriété *nodeName* qui renvoie le nom du nœud. Dans le cas d'un élément, c'est le nom de la balise. Dans le sens inverse de la propriété *documentElement* des objets DomDocument, les éléments ont une propriété *ownerDocument* qui est une référence sur le document.

- Recherche par nom de la balise
  - DomDocument::getElementsByTagName() ou DomElement::getElementsByTagName(). La première version fait une recherche dans tout le document, la deuxième dans les descendants de l'élément considéré. Ces fonctions retournent un objet DomNodeList.

Idée 4 = gérer un flux XML via SAX

- SAX est une API permet d'interpréter le XML au fur et à mesure de sa lecture (flux) sans chargement en mémoire complet préablable.
  - On parle de « **moteur événementiel** ».
  - Cet API se rencontre dans des contextes hors PHP.
- Plus précisément, au fur et à mesure de la lecture, SAX peut nous avertir dès qu'il rencontre:
  - une donnée textuelle,
  - une balise ouvrante ou fermante,
  - une entité,
  - ...

Principe général de mise en oeuvre

- 1) Définir le comportement de l'analyseur (le « parser »/parseur) pour chaque événement :
  - via des fonctions
- 2) Définir l'analyseur lui-même
- 3) Lancement de l'analyseur

Un exemple (les données)

- On reprend l'exemple du traitement d'un fichier contenant les ouvrages d'une bibliothèque comme celui de l'exemple du transparent 8 dont la dtd est:

```
<!ELEMENT bibliotheque (style+)>
<!ELEMENT style (livre*)>
<!ATTLIST style id CDATA #IMPLIED>
<!ELEMENT livre (titre, auteur)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
```

Un exemple (la sortie)

- On désire que la sortie ressemble à :

Quelques ouvrages de style roman:
- *La fortune des Rougon* de **Emile Zola**
- *Hernani* de **Victor Hugo**

Quelques ouvrages de style fiction:
- *Le seigneur des anneaux* de **J.R.R. Tolkien**

Traitement des événements textuels

- Remarques
  - trim : élimine les espaces inutiles.
  - utf8_decode :
    - transforme de l'ISO-8859-1 en UTF-8
    - inverse = utf8_encode
    - SAX en PHP, de même que PHP, reconnaît de base l'encodage ISO-8859-1. Ici le fichier XML est supposé être en UTF-8.

```
function traitement_texte( $sax, $texte) {
      $texte = trim($texte) ;
      if(!empty($texte)){echo utf8_decode($texte);}
 }
```

Traitements des balises

```
function ouvre( $sax, $nom, $attributs){
  if($nom == 'style'){
      echo "<b>Quelques ouvrages de style {$attributs['id']}:</b><br/>" ;
  }
  if($nom == 'livre'){
      echo '- ' ;
  }
  if($nom == 'auteur'){
      echo "<b>" ;
  }
  if($nom == "titre"){
      echo "<i>" ;
  }
}
```

```
function ferme( $sax, $nom){
    if($nom == 'style'){
        echo "<br>----------</br>" ;
    }
    if($nom == 'livre'){
        echo '</br>' ;
    }
    if($nom == 'auteur'){
        echo "</b> " ;
    }
    if($nom == "titre"){
        echo "</i> de " ;
    }
}
```

Définition analyseur et analyse

- $sax = xml_parser_create('UTF-8') ;
  - Définition de l'analyseur
- xml_parser_set_option($sax, XML_OPTION_CASE_FOLDING, 0);
  - Par défaut, l'analyseur sort les noms de balise en majuscule;
  - Ce comportement est ici redéfini (0 peut être remplacé par FALSE).
- xml_set_character_data_handler($sax, 'traitement_texte') ;
  - Définition du comportement de traitement de texte
- xml_set_element_handler($sax, 'ouvre', 'ferme') ;
  - Définition du comportement de traitement des balises
- $fic = file_get_contents('bibliotheque.xml') ;
  - Récupération du contenu du fichier dans une chaîne.
- xml_parse($sax, $fic) ;
  - lancement de l'analyse

Exemple complet

```
<?
 ffunction traitement_texte( $sax, $texte) { ...
 }

 function ouvre( $sax, $nom, $attributs) {...
 }

 function ferme( $sax, $nom){
 }

 $sax = xml_parser_create('UTF-8') ;
  xml_parser_set_option( $sax, XML_OPTION_CASE_FOLDING, 0 );
  xml_set_character_data_handler($sax, 'traitement_texte') ;
  xml_set_element_handler($sax, 'ouvre', 'ferme') ;

  $fic = file_get_contents('bibliotheque.xml') ;
  xml_parse($sax, $fic) ;
?>
```

Objets PHP 4

```
// Fichier rectangle.php
<?php
  class Rectangle {
    var $largeur;
    var $longueur;
    function Rectangle($la = 0, $lo = 0) {
      $this->largeur = $la;
      $this->longueur = $lo;
    }
    function surface() {
      return  $this->largeur * $this->longueur;
    }
    function perimetre() {
      return  2*($this->largeur + $this->longueur);
    }
  }
?>
```

Objets PHP 4

```
<?php
  require 'Rectangle.php';
  $rect = new Rectangle(2, 7.5);
  print('largeur du rectangle = '.$rect->largeur.'<br>');
  print('longueur du rectangle = '.$rect->longueur.'<br>');
  print('perimetre du rectangle = '.$rect->perimetre().'<br>');
  print('surface du rectangle = '.$rect->surface().'<br>');
  $rect->largeur++;
  print('nouvelle largeur du rectangle = '.$rect->largeur.'<br>');
  $rect0 = new rectangle();
  print('surface du rectangle rect0 = '.$rect0->surface().'<br>');
?>
```

Exécution :

largeur du rectangle = 2
longueur du rectangle = 7.5
perimetre du rectangle = 19
surface du rectangle = 15
nouvelle largeur du rectangle = 3
surface du rectangle rect0 = 0

Explications

- la classe Rectangle comporte :
  - 2 attributs (±variables) : largeur et longueur
  - 1 constructeur Rectangle
  - 2 méthodes (±fonctions) : surface et perimetre
- pour créer un objet de type Rectangle, on utilise l'opérateur new (qui fait appel au constructeur)
- l'acces aux attributs de la classe se fait :
  - pour un objet : $rect->largeur
  - dans la définition de classe : $this->largeur

Egalité d'objets

```
<?php
  require 'Rectangle.php';
  $rect = new Rectangle(2, 7.5);
  print('largeur de rect = '.$rect->largeur.'<br>');
  print('longueur de rect = '.$rect->longueur.'<br>');
  $copyrect = $rect;
  if ($rect == $copyrect)
        print('$rect == $copyrect<br>');
  else
        print('$rect != $copyrect<br>');
  $copyrect->largeur *= 2;
  print('doublement de la largeur de $copyrect<br>');
  print('surface de $copyrect = '.$copyrect->surface().'<br>');
  print('surface de $rect = '.$rect->surface().'<br>');
  $rect2 = new Rectangle(2, 7.5);
  print('largeur de $rect2 = '.$rect2->largeur.'<br>');
  print('longueur de $rect2 = '.$rect2->longueur.'<br>');
  if ($rect == $rect2)
        print('$rect == $rect2<br>');
  else
        print('$rect != $rect2<br>');
?>
```

Exécution :

largeur de rect1 = 2
longueur de rect1 = 7.5
$rect == $copyrect
doublement de la largeur de $copyrect
surface de $copyrect = 30
surface de $rect = 15
largeur de rect2 = 2
longueur de rect2 = 7.5
$rect == $rect2

- L'affectation de variables objets copie leur contenu.
- Pour comparer deux objets, on compare leurs attributs

Passage par référence

```
class Personne {
  var $nom;
  function Personne($nom) {
    $this-> = $nom;
    }
  function afficheNom() {
    echo "Nom=$this->nom";
    }
  }

$cedric = new Personne("Cédric");
$cedric2 = $cedric;
$cedric2->nom = "Sylvain";
$cedric->afficheNom();
$cedric2->afficheNom();
```

Nom=Cedric
Nom=Sylvain

```
$cedric = new Personne("Cédric");
$cedric2 = &$cedric;
$cedric2->nom = "Sylvain";
$cedric->afficheNom();
$cedric2->afficheNom();
```

Nom=Sylvain
Nom=Sylvain

Evolutions objets en php5

- Visibilité
  - PHP4 laissait tous les attributs et méthodes de classe publiques
  - avec PHP5 on peut limiter leur visibilité de ceux-ci, à l'aide des mots-clés : public, protected et private.
    - public : équivalent du mot-clé var de PHP4, il indique que l'attribut peut être appelé et modifié par l'ensemble du programme, même depuis l'extérieur de la class. C'est le type par défaut.
    - protected : l'attribut ne peut être utilisé que depuis sa propre classe ou ses classes héritées.
    - private : l'attribut ne peut être utilisé que depuis sa propre classe.

Evolutions objets en php5

- Méthodes __tostring(), __construct() et son pendant __destruct().
  - __tostring() : permet de définir un texte à afficher lorsque le programme essaye d'afficher directement l'objet (echo $xavier, par exemple), ce qui est plus explicite que le Object renvoyé par défaut.
  - __construct() : avec PHP4, pour initialiser des attributs ou lancer des méthodes dès la création de l'objet, il fallait que la classe dispose d'un méthode éponyme. Avec PHP5, l'équivalent est de passer par ce constructeur. Cette méthode prend en argument les attributs définis lors de l'instanciation de la classe.
  - __destruct() : inversement, le destructeur permet de terminer toute activité initiée par l'objet, quand celui-ci est détruit (par exemple avec unset()).

Evolutions objets en php5

- PHP4 passe les objets par copie par défaut, et par référence avec l'opérateur &, tandis que PHP5 les passe par référence implicitement.

- Ajout de la gestion des exceptions, try{} catch