Functional Programming with SPL Iterators

Marcus Börger

international **PHP**2004 conference

Functional Programming with SPL Iterators

- ☑ Functional programming?
- ☑ Iterators
- ☑ Recursion using Iterators
- ☑ Filtering using Iterators

Functional programming?

- ☑ Abstract from the actual data (types)
- ☑ Implement algorithms without knowing the data

Example: Sorting

- ☞ Sorting requires a container for elements
- ☞ Sorting requires element comparison
- ☞ Containers provide access to elements

- ☞ Sorting and Containers must not know data

What are Iterators

- ☑ Iterators are a concept to iterate anything that contains other things. Examples:
  - ☑ Values and Keys in an array
  - ☑ Text lines in a file
  - ☑ Database query results
  - ☑ Files in a directory
  - ☑ Elements or Attributes in XML
  - ☑ Bits in an image
  - ☑ Dates in a calendar range
- ☑ Iterators allow to encapsulate algorithms

The basic concepts

- ☑ Iterators can be internal or external also referred to as active or passive
- ☑ An internal iterator modifies the object itself
- ☑ An external iterator points to another object without modifying it
- ☑ PHP always uses external iterators at engine-level

The big difference

Arrays

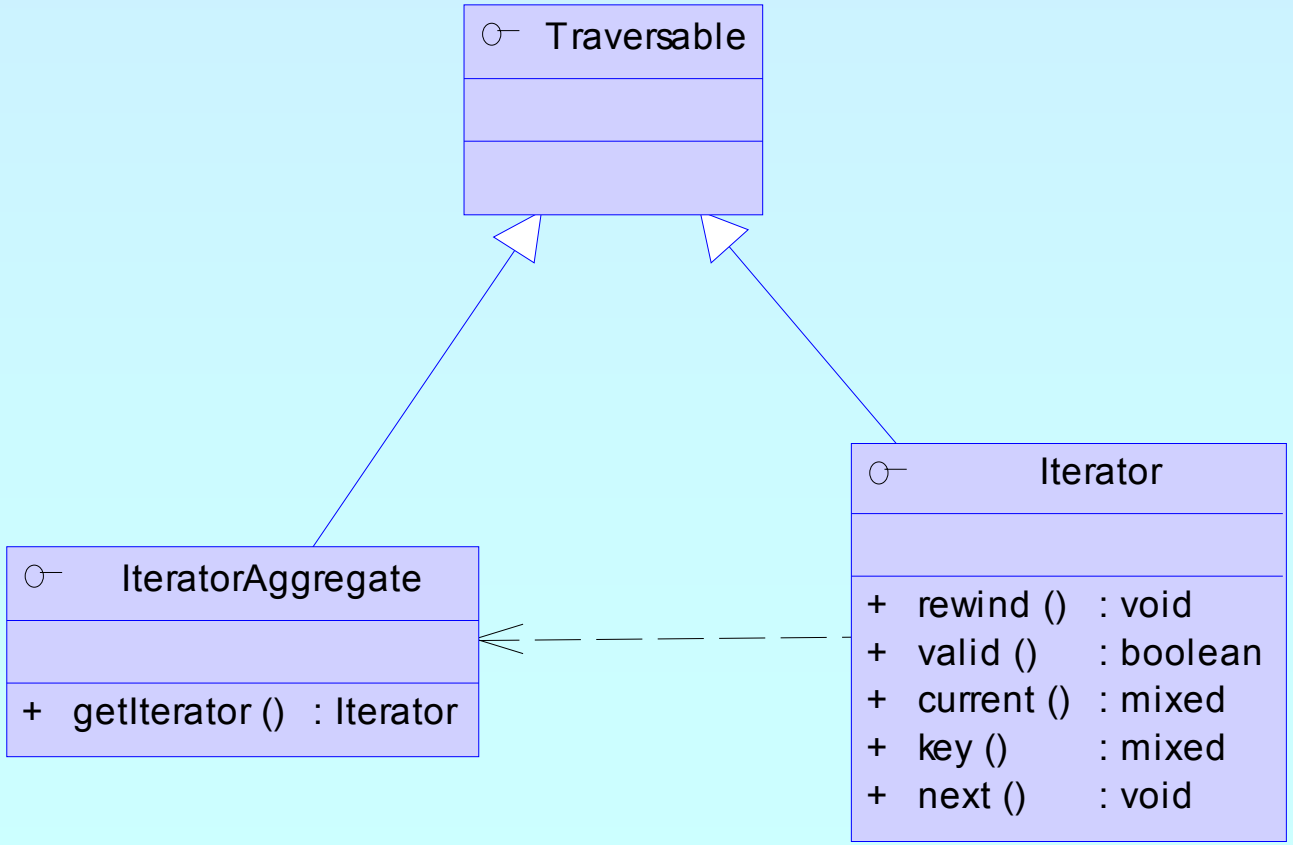
- ☑ require memory for all elements
- ☑ allow to access any element directly

Iterators

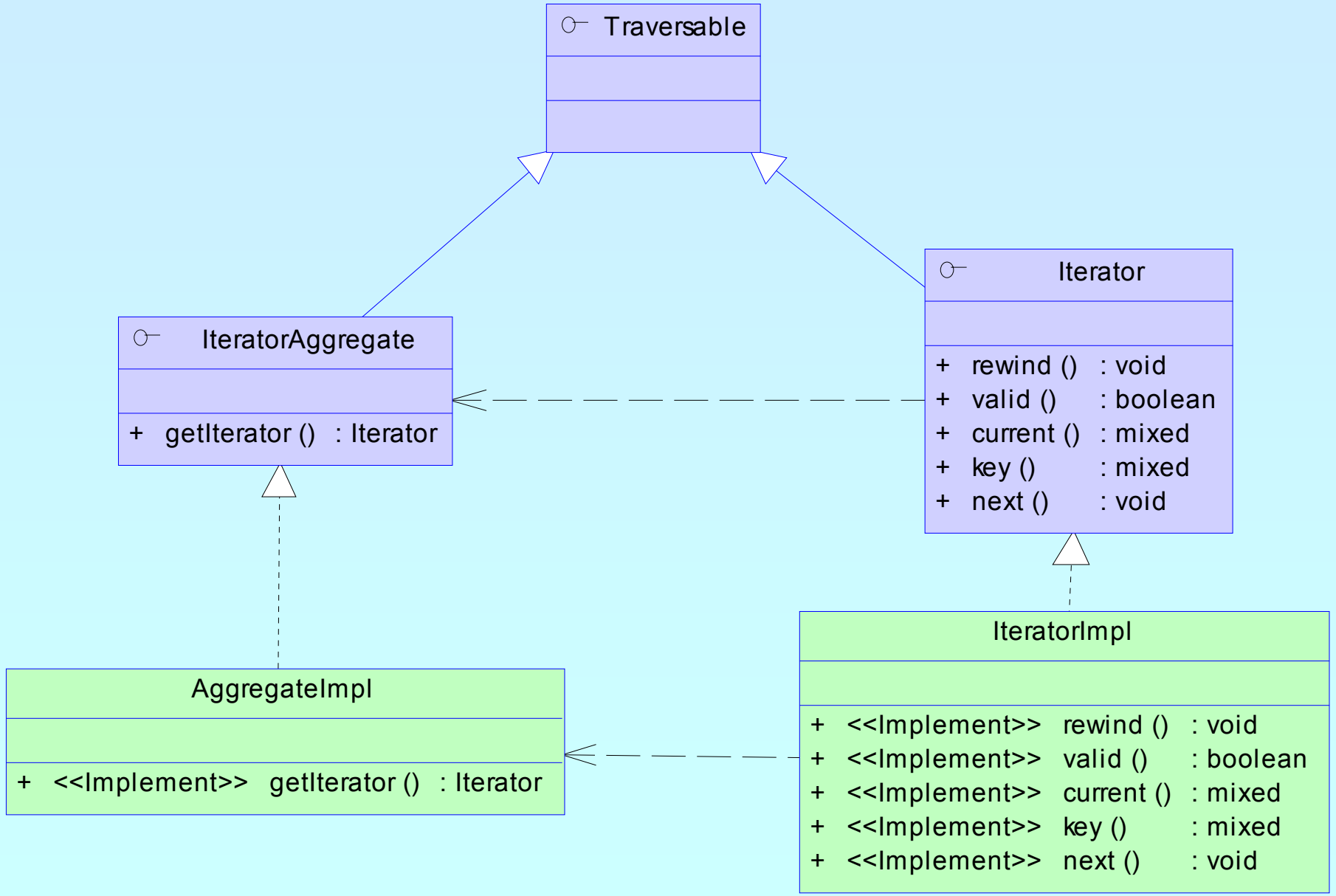
- ☑ only know one element at a time
- ☑ only require memory for the current element
- ☑ forward access only
- ☑ Access done by method calls

PHP Iterators

- ☑ Anything that can be iterated implements **Traversable**
- ☑ User classes cannot implement **Traversable**
- ☑ **Aggregate** is used for objects that use external iterators
- ☑ **Iterator** is used for internal traversal or external iterators



Implementing Iterators



How Iterators work

- ☑ Iterators can be used manually

```php
<?php
$o = new ArrayIterator(array(1, 2, 3));
$o->rewind();
while ($o->valid()) {
    $key = $o->key();
    $val = $o->current();
    // some code
    $o->next();
}
?>
```

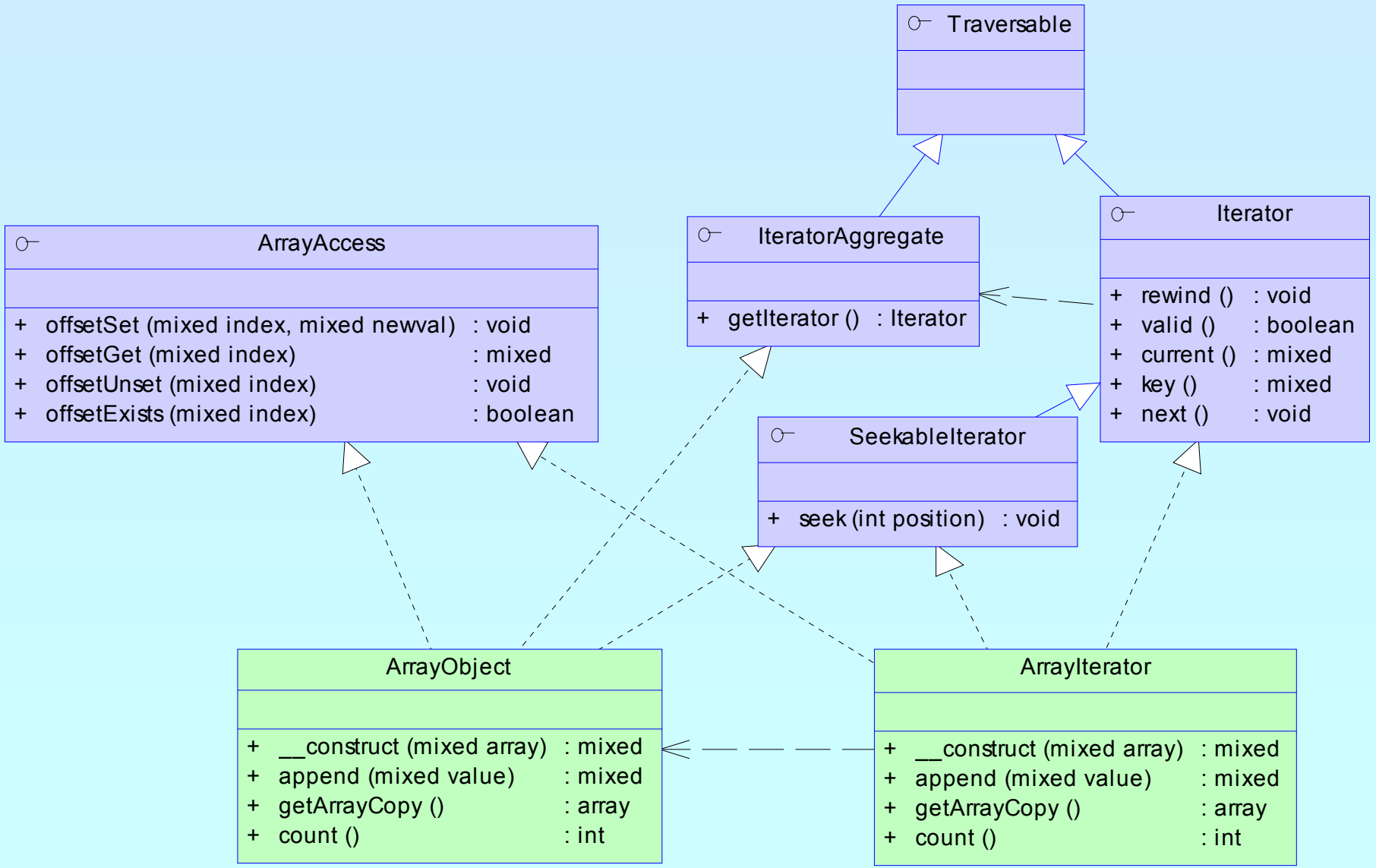
- ☑ Iterators can be used implicitly with **foreach**

```php
<?php
$o = new ArrayIterator(array(1, 2, 3));
foreach($o as $key => $val) {
    // some code
}
?>
```

Array and property traversal

- ☑ **ArrayObject** allows external traversal of arrays
- ☑ **ArrayObject** creates **ArrayIterator** instances
- ☑ Multiple **ArrayIterator** instances can reference the same target with different states
- ☑ Both implement **SeekableIterator** which allows to 'jump' to any position in the Array directly.

Array and property traversal



An example

- ☑ Reading a menu definition from an array
- ☑ Writing it to the output

Problem

- ☞ Handling of hierarchy
- ☞ Detecting recursion
- ☞ Formatting the output

Recursion with arrays

☑ A typical solution is to directly call array functions

☠ No code reuse possible

```php
<?php
function recurse_array($ar)
{
    // do something before recursion
    while (!is_null(key($ar))) {
        if (is_array(current($ar))) {
            recurse_array(current($ar));
        }
        // do something
        next($ar);
    }
    // do something after recursion
}
?>
```

Detecting Recursion

- An array is recursive
  - If the current element itself is an Array
  - In other words **current**() has children
  - This is detectable by **is_array**()
  - Recursing requires creating a new wrapper instance for the child array
  - **RecursiveIterator** is the interface to unify Recursion
  - **RecursiveIteratorIterator** handles the recursion

```
class RecursiveArrayIterator
    extends ArrayIterator implements RecursiveIterator
{
    function hasChildren() {
        return is_array($this->current());
    }
    function getChildren() {
        return new RecursiveArrayIterator($this->current());
    }
}
```

Debug Session

```php
<?php
$a = array('1','2',array('31','32'),'4');
$o = new RecursiveArrayIterator($a);
$i = new RecursiveIteratorIterator($o);
foreach($i as $key => $val) {
    echo "$key => $val\n";
}
?>
```

```
0 => 1
1 => 2
0 => 31
1 => 32
3 => 3
```

```php
<?php
class RecursiveArrayIterator implements RecursiveIterator {
   protected $ar;
   function __construct(Array $ar) {
      $this->ar = $ar; }
   function rewind() {
      reset($this->ar); }
   function valid() {
      return !is_null(key($this->ar)); }
   function key() {
      return key($this->ar); }
   function current() {
      return current($this->ar); }
   function next() {
      next($this->ar); }
   function hasChildren() {
      return is_array(current($this->ar)); }
   function getChildren() {
      return new RecursiveArrayIterator($this->current()); }
}
?>
```

Making ArrayObject recursive

- ☑ Change the class type of `ArrayObjects` Iterator
  - ☞ We simply need to change `getIterator()`

```
<?php
class RecursiveArrayObject
    extends ArrayObject
{
    function getIterator() {
       return new RecursiveArrayIterator($this);
    }
}
?>
```

Output HTML

- Problem how to format the output using
  - Detecting recursion begin/end

```
<?php
class MenuOutput
   extends RecursiveIteratorIterator
{
   function __construct(Menu $m) {
      parent::__construct($m);
   }
   function beginChildren() {
      echo str_repeat('  ',$this->getDepth())."<ul>\n";
   }
   function endChildren() {
      echo str_repeat('  ',$this->getDepth())."</ul>\n";
   }
}
?>
```

Output HTML

☑ Problem how to write the output

☞ Echo the output within foreach

```
<ul>
<li>1</li>
<li>2</li>
  <ul>
  <li>31</li>
  <li>32</li>
  </ul>
<li>4</li>
</ul>
```

```php
<?php
class MenuOutput
   extends RecursiveIteratorIterator
{
   function __construct(Menu $m) {
      parent::__construct($m);
   }
   function beginChildren() {
      echo str_repeat('  ',$this->getDepth())."<ul>\n";
   }
   function endChildren() {
      echo str_repeat('  ',$this->getDepth()+1)."</ul>\n";
   }
}
$ar = array('1','2',array('31','32'),'4');
$it = new MenuOutput(new RecursiveArrayIterator($ar));
echo "<ul>\n"; // for the intro
foreach($it as $m) {
   echo str_repeat('  ',$it->getDepth()+1)."<li>$m</li>\n";
}
echo "</ul>\n"; // for the outro
?>
```

Filtering

Problem

- ☞ Only recurse into active `Menu` elements
- ☞ Only show visible `Menu` elements
- ☠ Changes prevent `recurse_array` from reuse

```php
<?php
class Menu
{
    function isActive()  // return true if active
    function isVisible() // return true if visible
}
function recurse_array($ar)
{
    // do something before recursion
    while (!is_null(key($ar))) {
        if (is_array(current($ar)) && current($ar)->isActive()) {
            recurse_array(current($ar));
        }
        if (current($ar)->current()->isActive()) {
            // do something
        }
        next($ar);
    }
    // do something after recursion
}
?>
```

Filtering

Solution Filter the incoming data

- Unaccepted data simply needs to be skipped
- Do not accept inactive menu elements
- Using a `FilterIterator`

```php
<?php
class Menu extends RecursiveArrayIterator
{
    function isActive()  // return true if active
    function isVisible() // return true if visible
}
?>
```

Filtering

☑ Using a FilterIterator

```
<?php
class MenuFilter extends FilterIterator
    implements RecursiveIterator
{
    function __construct(Menu $m) {
        parent::__construct($m);
    }
    function accept() {
        return $this->current()->isVisible();
    }
    function hasChildren() {
        return $this->current()->hasChildren()
            && $this->current()->isActive();
    }
    function getChildren() {
        return new MenuFilter($this->current());
    }
}
?>
```

Putting it together

- ☑ Make `MenuOutput` operate on `MenuFilter`
  - Pass a `Menu` to the constructor (guarded by type hint)
  - Create a `MenuFilter` from the `Menu`
  - `MenuFilter` implements `RecursiveIterator`

```
<?php
class MenuOutput extends RecursiveIteratorIterator
{
   function __construct(Menu $m) {
      parent::__construct(new MenuFilter($m));
   }
   function beginChildren() {
      echo "<ul>\n";
   }
   function endChildren() {
      echo "</ul>\n";
   }
}
?>
```

What now

- ☑ If your menu structure comes from a database
- ☑ If your menu structure comes from XML

  - ☞ You have to change `Menu`
  - ☞ Detection of recursion works differently

  - ☞ No single change in `MenuOutput` needed
  - ☞ No single change in `MenuFilter` needed

Using XML

☑ Change `Menu` to inherit from `SimpleXMLIterator`

```php
<?php
class Menu extends SimpleXMLIterator
{
    static function factory($xml)
    {
        return simplexml_load_string($xml, 'Menu');
    }
    function isActive() {
        return $this['active']; // access attribute
    }
    function isVisible() {
        return $this['visible']; // access attribute
    }
    // getChildren already returns Menu instances
}
?>
```

Using PDO

☑ Change `Menu` to read from database

- ☞ PDO supports Iterator based access
- ☞ PDO can create and read into objects
- ☞ PDO will be available with PHP 5.1
- ✋ PDO is under heavy development

```
<?php
$db = new PDO("mysql://...");
$menu = $db->query("SELECT ... FROM Menu ...", "Menu");
foreach ($menu as $m) {
      // fetch now returns Menu instances
      echo $m->fetch()->__toString();
}
?>
```

Conclusion

- ☑ Iterators require a new way of programming
- ☑ Iterators allow to implement algorithms abstracted from data
- ☑ Iterators promote code reuse
- ☑ Some things are already in SPL
  - ☑ Filtering
  - ☑ Handling recursion

THANK YOU

http://somabo.de/talks/

http://php.net/~helly/php/ext/spl