Go OO!

Real-Life Design Patterns in PHP5

Stephan Schmidt, 1&1 Internet AG

international PHP2005 conference - spring edition -

Agenda

- What are Design Patterns?
- PHP5's new OO features
- Creational Patterns
- Structural Patterns
- Behavioral Patterns
- A Look at SPL
- Real-Life Appliances

The speaker

- Working for 1&1 Internet AG
- Founding member of PHP Application Tools (pat)
- Active developer of the PEAR community
- Writing for several PHP magazines

What are Design Patterns?

- standard solutions to common problems in software architecture
- independent of the programming language
- ease the communication between the developers
- typically describe the interaction of classes and objects

PHP5's new OO features

- OO-model has been completely revamped
- objects are not any longer just "improved arrays"
- new SPL extension

Pass-by-reference

```
class Foo {
   var $val = 'Foo';
}
$foo = new Foo();
$bar = $foo;
$bar->val = 'Bar';
echo $bar->val . "\n";
echo $foo->val . "\n";
```

PHP 4

```
Bar
Foo
```

PHP 5

```
Bar
Bar
```

Constructors/Destructors

- Constructor is named __construct()
- Destructor is named __destruct()

```
class Foo {
    public function __construct() {
        print "Foo created\n";
    }
    public function __destruct() {
        print "Foo destroyed\n";
    }
}
$bar = new Foo();
unset($bar);
```

Visibility

- Restrict access to properties and methods of a class.
- public := all
- protected := class and descendants
- private := only the class

Visibility (2): Example

```
class Foo {
   private   $foo = 'foo';
   public    $bar = 'bar';
   protected $tomato = 'tomato';
}
$bar = new Foo();
print "{$bar->bar}\n";
print "{$bar->foo}\n";
```

```
$ php visibility.php
bar

Fatal error: Cannot access private property Foo::$foo in
/home/schst/go-oo/visibility.php on line 9
```

Static methods/properties

- Available outside of the object context
- can be public/private/protected
- self::$foo instead of $this->foo

```
class Foo {
    public static $foo = 'bar';
    public static function getFoo() {
        return self::$foo;
    }
}
print Foo::$foo . "\n";
print Foo::getFoo() . "\n";
```

Object Abstraction

Abstract classes cannot be instantiated

```
abstract class AbstractClass {
    abstract public function doSomething();
}
class ConcreteClass extends AbstractClass {
    public function doSomething() {
        print "I've done something.\n";
    }
}
$foo = new ConcreteClass();
$bar = new AbstractClass();

Fatal error: Cannot instantiate abstract class
AbstractClass in ... on line 11
```

Interfaces

- Specify the methods a class has to implement without defining how they are implemented
- Classes may implement more than one interface
- If a class does not implement all methods of an interface => E_FATAL

Interfaces (2): Example

```
interface IRequest {
    public function getValue($name);
    public function getHeader($name);
}
class HttpRequest implements IRequest {
    public function getValue($name) {
        return $_REQUEST[$name];
    }
}


Fatal error: Class HttpRequest contains 1 abstract methods
and must therefore be declared abstract
(IRequest::getHeader) in /home/schst/go-oo/interfaces.php
on line 10
```

Property Overloading

Intercept access to properties, that do not exist

```
class Foo {
    private $props = array('foo' => 'bar');

    public function __get($prop) {
        if (!isset($this->props[$prop])) {
            return null;
        }
        return $this->props[$prop];
    }
}
$foo = new Foo();
print "{$foo->foo}\n";
```

Method Overloading

Intercept calls to methods, that do not exist

```
class Foo {
    public function __call($method, $args) {
        if (is_callable($method)) {
            return call_user_func_array($method, $args);
        }
        return null;
    }
}

$foo = new Foo();
print $foo->strrev('tomato') . "\n";
```

__toString()

Change string-cast behavior of an object

```
class User {
    private $id;
    private $name;
    public function __construct($id, $name) {
        $this->id = $id;
        $this->name = $name;
    }
    public function __toString() {
        return "{$this->name} ({$this->id})\n";
    }
}
$schst = new User('schst', 'Stephan Schmidt');
print $schst;
```

Object Iteration

Objects can be used in foreach-statements

```
class PearDevelopers {
    public $schst  = 'Stephan Schmidt';
    public $luckec = 'Carsten Lucke';
}
$users = new PearDevelopers();
foreach ($users as $id => $name) {
    print "$id is $name\n";
}

schst is Stephan Schmidt
luckec is Carsten Lucke
```

Misc additions

- Object cloning
  ```
  $bar = clone $foo;
  ```
- Type hints for objects
  ```
  public function foo (MyClass $obj) {}
  ```
- Exception Handling
- Reflection API
- __autoload()
- SPL

Creational Patterns

- Create objects
- Hide the creational process
- Your application does not need to know, how the object has to be created
- Allows you to hide the concrete implementation
- Dynamic object configuration

Factory Method

- Create objects based on some input
- Hides the object creation
- PEAR makes heavy use of this, like in PEAR::DB

```
$con = DB::connect('mysql://user:pass@host/db');
```

- The returned object is an instance of a subclass of DB_common based on the database type

Singleton

- Makes sure that only one instance of a class exists

```
class Registry {
    private static $instance = null;
    private function __construct() {}
    public $foo;

    public function singleton() {
        if (is_null(self::$instance)) {
            self::$instance = new Registry();
        }
        return self::$instance;
    }
}
```

Singleton (2): Usage

```
$reg1 = Registry::singleton();
$reg2 = Registry::singleton();
$reg1->foo = 'Bar';
print $reg2->foo . "\n";
```

- Commonly used for
  - Configurations / Registries
  - Sharing DB connections (combine it with factory method)
  - Request objects
  - ...

Structural Patterns

- Define the relationships between classes and/or objects
- Object-composition
- Often use inheritance and interfaces

Decorator

- Allows you to dynamically add functionality to an object
- Used for functionality that is used only in some cases
- Often used to avoid inheritance or when inheritance is not possible
- Decorator wraps the original objects

Decorator (2): Component

```
class String {
  private $string = null;
  public function __construct($string)  {
    $this->string = $string;
  }
  public function __toString()  {
    return $this->string;
  }
  public function getLength() {
    return strlen($this->string);
  }
  public function getString() {
    return $this->string;
  }
  public function setString($string) {
    $this->string = $string;
  }
}
```

Decorator (3): Abstract

An abstract decorator

```
abstract class String_Decorator {
  protected $obj;
  public function __construct($obj) {
    $this->obj = $obj;
  }
  public function __call($method, $args) {
    if (!method_exists($this->obj, $method)) {
      throw new Exception('Unknown method called.');
    }
    return call_user_func_array(
                array($this->obj, $method), $args);
  }
}
```

Decorator (4): Bold

Concrete Decorator

```
class String_Decorator_Bold extends String_Decorator {
  public function __toString() {
    return '<b>' . $this->obj->__toString() . '</b>';
  }
}
```

Usage

```
$str     = new String('Decorators are cool');
$strBold = new String_Decorator_Bold($str);
print $strBold;
```

Decorator (5): Reverse

Adding a method

```
class String_Decorator_Reverse extends String_Decorator {
  public function reverse(){
    $str = $this->obj->getString();
    $this->obj->setString(strrev($str));
  }
}
```

Usage

```
$str    = new String('Decorators are cool');
$strRev = new String_Decorator_Reverse($str);
$strRev->reverse();
print $strRev;
```

Decorator (6): Combination

Combining decorators

```
$str     = new String('Decorators are cool');
$strBold = new String_Decorator_Bold($str);
$strRev  = new String_Decorator_Reverse($strBold);
$strRev->reverse();
print $strRev;
```

Proxy

- Provides a placeholder for an objects to control or access this objects
- Very common with webservices or application servers

```
$client = new SoapClient(
                'http://api.google.com/GoogleSearch.wsdl');

$result = $client->doGoogleSearch(…);
```

Proxy (2): Implementation

Catch all method calls on the Proxy using __call()

```
class Proxy {
  public function __construct() {
  // establish connection to the original object
  }
  public function __call($method, $args) {
  // forward the call to the original object
  // using any protocol you need
  }
}
```

Delegator

- Similar to Proxy
  - Often referred to as "Chained Proxy"
- Acts as a proxy to several objects
- Intercept calls to all unknown methods and forward those to any of the delegates, that provides these methods
- Ready-to-use implementation for PHP: PEAR_Delegator

Delegator (2): Example

The Delegator

```
class Foo extends PEAR_Delegator {
  public function __construct() {
    parent::_construct();
  }
  public function __destruct() {
    parent::__destruct();
  }
  public function displayFoo() {
    print "foo\n";
  }
}
```

Delegator (3): Example

The Delegates

```
class Delegate1 {
  public function displayBar() {
    print "bar\n";
  }
}

class Delegate2 {
  public function displayTomato() {
    print "tomato\n";
  }
}
```

Delegator (4): Example

Usage

```
$delegator = new Foo();
$delegate1 = new Delegate1();
$delegate2 = new Delegate2();

$delegator->addDelegate($delegate1);
$delegator->addDelegate($delegate2);

$delegator->displayFoo();
$delegator->displayBar();
$delegator->displayTomato();
```

Behavioral patterns

- Provide solutions for communication between objects
- Increase the flexibility of the communication

Observer

- Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- First step to event-based development
- Often used for logging techniques

Observer (2): Subject

```
class Subject {
  private $observers = array();
  public $state = null;
  public function attach(Observer $observer) {
    $this->observers[] = $observer;
  }

  public function detach(Observer $observer) {
  // remove the observer
  }

  public function notify() {
    for ($i = 0; $i < count($this->observers); $i++) {
      $this->observers[$i]->update();
    }
  }
}
```

Observer (3): Observer

```
class Observer {
  private $subject;
  private $name

  public function __construct($subject, $name) {
    $this->subject = $subject;
    $this->name    = $name;
  }

  public function update() {
    $state = $this->subject->state;
    print $this->name.": State of subject is $state\n";
  }
}
```

Observer (4): Usage

```
$subj = new Subject();
$ob1  = new Observer($subj, 'Observer 1');
$ob2  = new Observer($subj, 'Observer 2');
$subj->attach($ob1);
$subj->attach($ob2);

$subj->state = "authenticated";
$subj->notify();
```

Standard PHP Library

- Bundled with PHP 5 and enabled by default
- Collection of interfaces
  - Iterators
  - ArrayAccess, Countable
  - Subject/Observer (PHP 5.1)
- Collection of Classes
  - Iterators
  - Exceptions, FileObject (PHP 5.1)

ArrayAccess

- Allows you to access any object as it were an array
- Interface provided by SPL
  - public function offsetExists($offset);
  - public function offsetGet($offset);
  - public function offsetSet($offset, $value);
  - public function offsetUnset($offset);
- Only works with [] array syntax

ArrayAccess (2): Example

```
class Foo implements ArrayAccess {
  private $props = array('foo' => 'Bar');
  public function offsetExists($offset) {
    return isset($this->props[$offset]);
  }
  public function offsetGet($offset) {
    return $this->props[$offset];
  }
  public function offsetSet($offset, $value) {
    $this->props[$offset] = $value;
  }
  public function offsetUnset($offset) {
    unset($this->props[$offset]);
  }
}
```

ArrayAccess (3): Example

```
$obj = new Foo();
print $obj['foo'] . "\n";
$obj['bar'] = 3452;
if (isset($obj['bar'])) {
  print $obj['bar'] . "\n";
}

$ php arrayAccess.php
Bar
3452
```

Abstracting HTTP Requests

- Create request object to access request properties
- Replaces $_GET, $_POST, $_SERVER
- High level of abstraction
- Provides flexibility
- Implements ArrayAccess interface to allow $request['property'] syntax for a shallow learning curve

Request (2): Example

```
abstract class Request implements ArrayAccess {
  protected $properties = array();
  public function offsetExists($offset) {
    return isset($this->properties[$offset]); }
  public function offsetGet($offset) {
    return $this->properties[$offset];
  }
  public function offsetSet($offset, $value) {
    $this->properties[$offset] = $value;
  }
  public function offsetUnset($offset) {
    unset($this->properties[$offset]);
  }
}
```

Request (3): HTTP

```
class Request_HTTP extends Request  {
  public function __construct() {
    $this->properties = $_REQUEST;
  }
}

$request = new Request_HTTP();
if (isset($request['foo'])) {
  echo $request['foo'];
} else {
  echo "property foo has not been set";
}
```

```
http://www.example.com/?foo=bar
```

Replacing the Request

- Request_HTTP can be replaced by any class with the same interface
  - Request_CLI
  - Request_SOAP
- Combine this with singleton and factory method:

```
$request = Request::get('HTTP');
```

Request (4): CLI

```
class Request_CLI extends Request  {
  public function __construct() {
    array_shift($_SERVER['argv']);
    foreach ($_SERVER['argv'] as $pair) {
      list($key, $value) = explode('=', $pair);
      $this->properties[$key] = $value;
    }
  }
}
$request = new Request_CLI();
if (isset($request['foo'])) {
  echo $request['foo'];
} else {
  echo "property foo has not been set";
}
```

```
$ ./script.php foo=bar
```

Intercepting filters

- Allows you to preprocess the request data
- Apply centralized authentication mechanism
- Validate/modify request properties
- Forward based on the URI
- Borrowed from J2EE Servlets

Intercepting filters (2)

Changes to Request

```
abstract class Request implements ArrayAccess {
  …
  protected $filters = array();
  public function addFilter(InterceptingFilter $filter) {
    $this->filters[] = $filter;
  }
  protected function applyFilters() {
    for ($i = 0; $i < $this->filters; $i++) {
      $this->filters[$i]->doFilter($this);
    }
  }
}
```

Intercepting filters (3)

Changes to Request_HTTP

```
class Request_HTTP extends Request {
  public function __construct() {
    $this->properties = $_REQUEST;
    $this->applyFilters();
  }
}
```

Simple filter interface

```
interface InterceptingFilter {
  public function doFilter(Request $request);
}
```

Iterators

- Interface that allows you to influence the way foreach-iterates over an object
  - mixed public function current();
  - mixed public function key();
  - void public function next();
  - boolean public function valid();
  - void public function rewind();
- Traverse any data using foreach (dir listing, array, text file, etc.)

Iterators (2): Example

```
class CSVFile implements Iterator {
  protected $file;
  protected $fp;
  protected $line;
  protected $key = -1;

  public function __construct($file) {
    $this->file  = $file;
    $this->fp    = @fopen($this->file, 'r');
    if (!$this->fp) {
      throw new Exception('Could not open file.');
    }
  }

  public function __desctruct() {
    fclose($this->fp);
  }
```

Iterators (3): Example cont.

```
 public function next(){
    if (!feof($this->fp)) {
      $this->key++;
      $this->line = fgetcsv($this->fp);
      $this->valid = true;
    } else {
      $this->valid = false;
    }
  }

  public function rewind() {
    $this->key = -1;
    fseek($this->fp, 0);
    $this->next();
  }
}
```

Iterators (4): Example cont.

```
  public function current() {
    return $this->line;
  }

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

  public function valid() {
    return $this->valid;
  }
}
```

Iterators (5): Example cont.

```
$csvFile = new CSVFile('users.csv');
foreach ($csvFile as $entry) {
  print_r($entry);
}
```

```
Array (
   [0] => Array (
               [0] => 'schst',
               [1] => 'Stephan Schmidt'
               ),
   [1] => Array (
               [0] => 'luckec',
               [1] => 'Carsten Lucke'
               ),
)
```

Recursive Iterators

- Extends the Iterator interface
  - boolean public function hasChildren();
  - Iterator public function getChildren();
- getChildren() returns an object that implements Iterator
- Traverse hierarchical data using the class RecursiveIteratorIterator

Abstracting data structures

Iterators allow you to abstract hierarchical structures in your application using foreach.

The user of the data does not know how it is computed, but only traverses it.

Simple example: Page definition files in a MVC-framework.

Example: Page defintions

One page per configuration file:

```
title = "Homepage"
desc  = "This is the homepage"
class = "Homepage"
```

Navigation structure in the filesystem:

```
index.ini
projects.ini
projects/
  pat.ini
  pear.ini
  pear/
    services_ebay.ini
    xml_serializer.ini
```

Example: Page Class

```
class Page {
  public $name;
  public $title;
  public $desc;

  public function __construct($basePath, $name) {
    $fname = $basePath . '/' . $name . '.ini';
    $tmp   = parse_ini_file($fname);
    $this->name  = $name;
    $this->title = $tmp['title'];
    $this->desc  = $tmp['desc'];
  }
}

$home = new Page('pages', 'index');
print $home->title;
```

Example: Sitemap Class

```
class Sitemap implements Iterator {
  protected $path;
  protected $pos = 0;
  protected $pages = array();

  public function __construct($path) {
    $this->path = $path;
    if (file_exists($this->path)) {
      $dir = dir($path);
      while ($entry = $dir->read()) {
        $this->pages[] = new Page($this->path, $entry);
      }
    }
  }
…
```

Example: Sitemap Class (2)

```
  public function current() {
    return $this->pages[$this->pos];
  }
  public function key() {
    return $this->pos;
  }
  public function next() {
    ++$this->pos;
  }
  public function rewind() {
    $this->pos = 0;
  }
  public function valid() {
    return isset($this->pages[$this->pos]);
  }
}
```

Example: Sitemap Usage

```
$sitemap = new Sitemap('pages');

foreach ($sitemap as $page) {
  echo $page->title . "<br />\n";
}
```

User does not need to know when and how the page objects are created and in which order they are traversed.

INI files can be substituted with XML, content from a database, etc.

Example: Going recursive

```
class Page extends Sitemap {
  …
  public function __construct($basePath, $name) {
    $fname = $basePath . '/' . $name . '.ini';
    $tmp   = parse_ini_file($fname);
    $this->name  = $name;
    $this->title = $tmp['title'];
    $this->desc  = $tmp['desc'];

    $subPath = $basePath . '/' . $this->name;
    parent::__construct($subPath);
  }
  public function hasPages() {
    return !empty($this->pages);
  }
}
```

Example: Going recursive

```
$sitemap = new Sitemap('pages');

foreach ($sitemap as $page) {
  echo $page->title . '<br />';
  foreach ($page as $subPage) {
    echo ' - ' . $subPage->title . '<br />';
  }
}
```

- Restricted to two levels :(
- Not intuitive

Example: Going recursive

```
class Sitemap implements RecursiveIterator {
  …
  public function hasChildren() {
    return $this->pages[$this->pos]->hasPages();
  }

  public function getChildren() {
    return $this->pages[$this->pos];
  }
}
```

Example: Done

```
$sitemap  = new Sitemap('pages');
$iterator = new RecursiveIteratorIterator($sitemap,
                                          RIT_SELF_FIRST);
foreach ($iterator as $page) {
  $depth = $iterator->getDepth();
  if ($depth > 0) {
    echo str_repeat(' ', $depth*2) . ' - ';
  }
  echo $page->title . '<br />';
}
```

```
Homepage
Projects
  - PAT-Projects
  - PEAR-Projects
    - Services_Ebay
    - XML_Serializer
```

Useful Resources

- Design Patterns
  http://www.dofactory.com/Patterns/Patterns.aspx
- phpPatterns()
  http://www.phppatterns.com
- J2EE Patterns
  http://java.sun.com/blueprints/corej2eepatterns/
- Microsoft patterns
  http://msdn.microsoft.com/architecture/patterns/

The end

Thank you for your attention!

ANY QUESTIONS ?

schst@php-tools.net
http://www.php-tools.net