Objets, patterns et génie logiciel en PHP

Qui suis-je ?

- Julien PAULI ; jpauli@php.net
  - Architecte système et logiciel
  - Expert spécialiste plateforme LAMP (PHP)
  - Contributeur open source   
  - Consultant
  - Co-Auteur (Eyrolles)

https://github.com/jpauli


@julienpauli

Voyons voir ...

- Développement objet PHP
  - Ou en est-on ?
  - Historique PHP
  - Différence avec d'autres langages
- Patterns et génie logiciel
  - Rappels sur les principes SOLID
  - Rappels sur les patterns
- Objets et patterns en PHP
  - Modèle objet de PHP détaillé
  - Quelques patterns en PHP

PHP, rappelez moi ce que c'est ?

- Langage web interprété, crée en ~1998
  - Langage procédural
  - Syntaxe Inspirée de C
  - Possède un modèle objet, inspiré de Java
  - Ecrit en C (~800.000 lignes), Machine virtuelle
  - Extensible
  - Performant
  - Simple et efficace
  - Massivement déployé
    - Y compris chez les "gros"
      - Facebook
      - Yahoo!

PHP où en est-on ?

- 2009 : PHP 5.3
  - Enrichissement du modèle objet (namespaces)
  - Performances du moteur accrues (~+15%)
  - Meilleure gestion de la mémoire
  - Fonctions anonymes
  - Prévention de l'utilisation de fonctions dépréciées
  - Pilote MySQL natif (mysqlnd : licence PHP)
  - FastCGI refondu (PHP FPM)

PHP où va-t-on ?

- ~11/2010 : PHP 5.4
  - Traits (pseudo héritage horizontal)
  - Nettoyage d'anciennes fonctionnalités dépréciées
  - Serveur web intégré
  - Support dtrace
  - Syntaxe améliorée

PHP et OOP

- Pas un langage orienté objet
  - Mais modèle objet présent
  - Proche de celui de Java (pas d'héritage multiple)
- Quelques objets/interfaces natif
  - Iterator / ArrayAccess / Traversable / JsonSerializable
  - DateTime / DateTimeZone/ PDO / SimpleXml / Dom
- Des frameworks full OO
  - ZendFramework
  - Symfony
  - CodeIgniter
  - ...

OOP PHP face à OOP Java

- Pas de typage des retours de méthodes
- Pas de déclaration des exceptions lancées
- Pas de typage fort des attributs
- Pas de finally (exceptions)
- Pas de typage dynamique (duck typing)
- Pas de visibilité "package"
- ...

- PHP n'est pas Java
- Il s'inspire globalement de son modèle objet

Modèle objet de PHP

- namespaces
- classes abstraites
- interfaces
- Héritage simple (non multiple)
- Pas de persistence
  - Sérialization

```
namespace JPDO\Result;

abstract class SavableObjects extends Objects implements SomeIface
{
    public function save() { }
    abstract protected function load() { }
}
```

Principes SOLID

- Règles régissant la conception orientée Objet
- **S**ingle Responsability
- **O**pen/Close Principle
- **L**iskov substitution Principle
- **I**nterface Seggregation
- **D**ependency Injection

PHP et SOLID

Liskov Subsitution

- Un objet utilisant A doit pouvoir utiliser un fils de A sans s'en rendre compte
- L'héritage doit être strict, le type conservé

```
class A
{
    public function foo($b, $c) { }
}
```

```
class B extends A
{
    public function foo($b, $c, $d) { }
}
```

Strict standards: Declaration of B::foo() should be compatible with that of A::foo()

PHP et SOLID

Dependency Injection

- Un objet A ayant besoin de B ne doit pas chercher B lui-même, il doit le lui être injecté (agrégation)

```
class A
{
    protected $b;

    public function setB(B $b)
    {
        $this->b = $b
    }

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

PHP et SOLID

Interface Seggregation

- Un objet A ne doit pas utiliser un objet B directement, mais une interface Ib de B

```
class A
{
    public function setB(Ib $b) { }
}
```

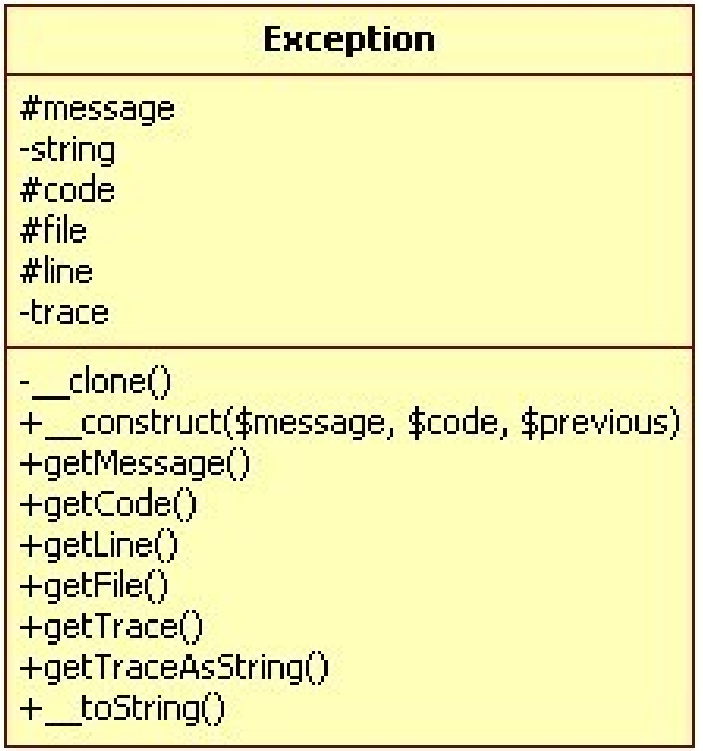
```
interface Ib
{
    function someMethod() { }
}
```

```
class B implements Ib
{
    public function someMethod() { }
}
```

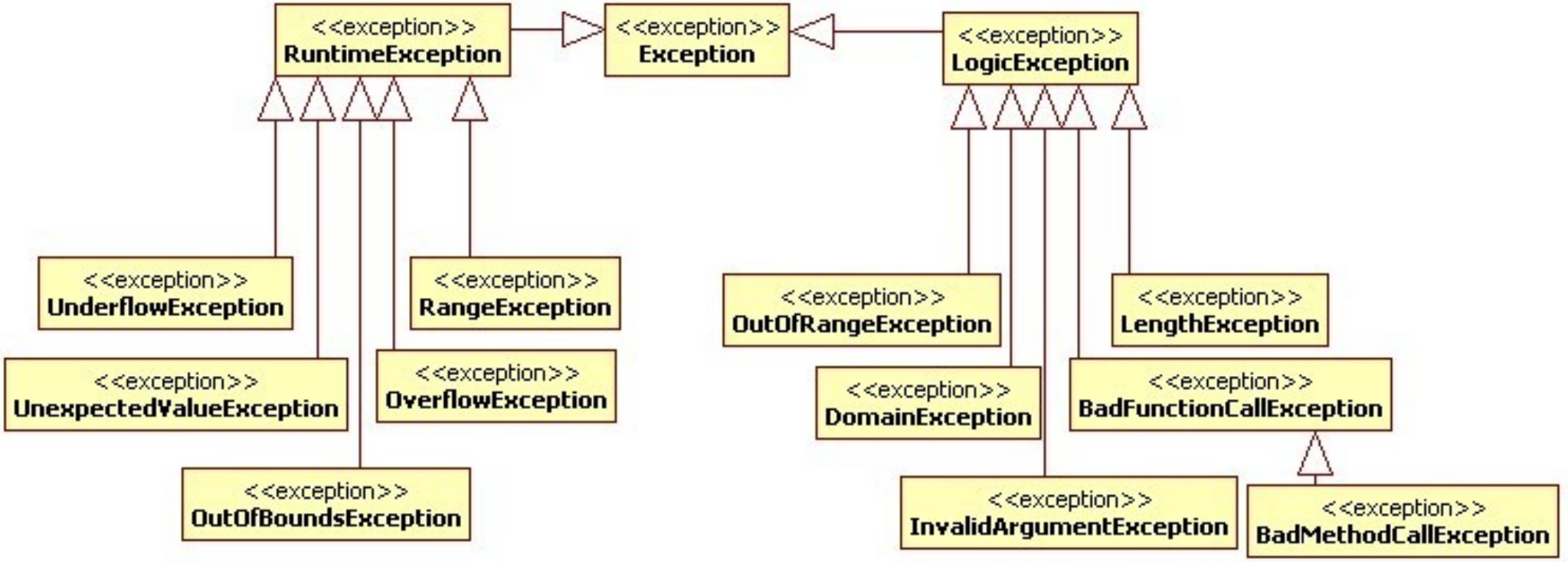
PHP Exceptions

- try { throw } catch () { }
- Pas de finally
- Pas d'interface Throwable

```
try{
      $result = $db->getResult();
}catch (BadArgumentException $e){
// une erreur dans les arguments d'une fonction
}catch (DatabaseException $e){
// une erreur de base de données
}catch (Exception $e){
echo 'erreur !' . $e->getMessage();
}
```



PHP plein d'exceptions



Finalement, on a de quoi utiliser des patterns !

- Interfaces et classes abstraites
- Héritage
- Namespaces
- Visibilité, encapsulation

- PHP est tout à fait capable de supporter la plupart des patterns existants grâce à la fléxibilité de son modèle objet.

- PHP reste un langage pour le web
  - >> Inutile de le "polluer" de fonctionnalités non adaptées à cet environnement

Design Patterns ?

- Solutions OO
  - Remarquables
  - Implémentant SOLID
  - Répondent à des problèmes récurrents
  - Modélisables avec UML
  - Très connues des archi. logiciels
  - Organisées par familles
    - Créateurs, structuraux, comportementaux
  - Souvent passer de l'un à l'autre c'est changer quelques lignes de code
  - **Permettent d'assurer la pérénité dans le code dans le temps**

Singleton

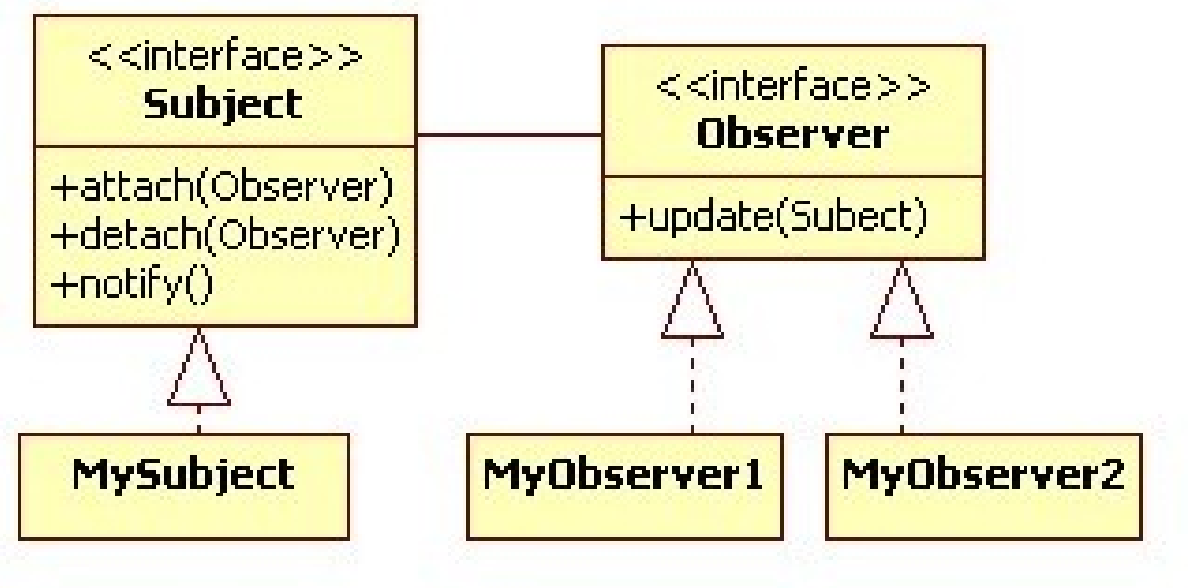
- Attention à son utilisation anti-pattern...

```
class Singleton
{
    protected static $_instance;

    protected function __construct() { }
    protected function __clone() { }

    public static function getInstance() {
        if (!self::$_instance) {
            self::$_instance = new self;
        }
        return self::$_instance;
}
```

```
assert(Singleton::getInstance() ===
        Singleton::getInstance());
```

Observateur / Sujet

- Un pattern très utile, pouvant être implémenté de plusieurs manières différentes

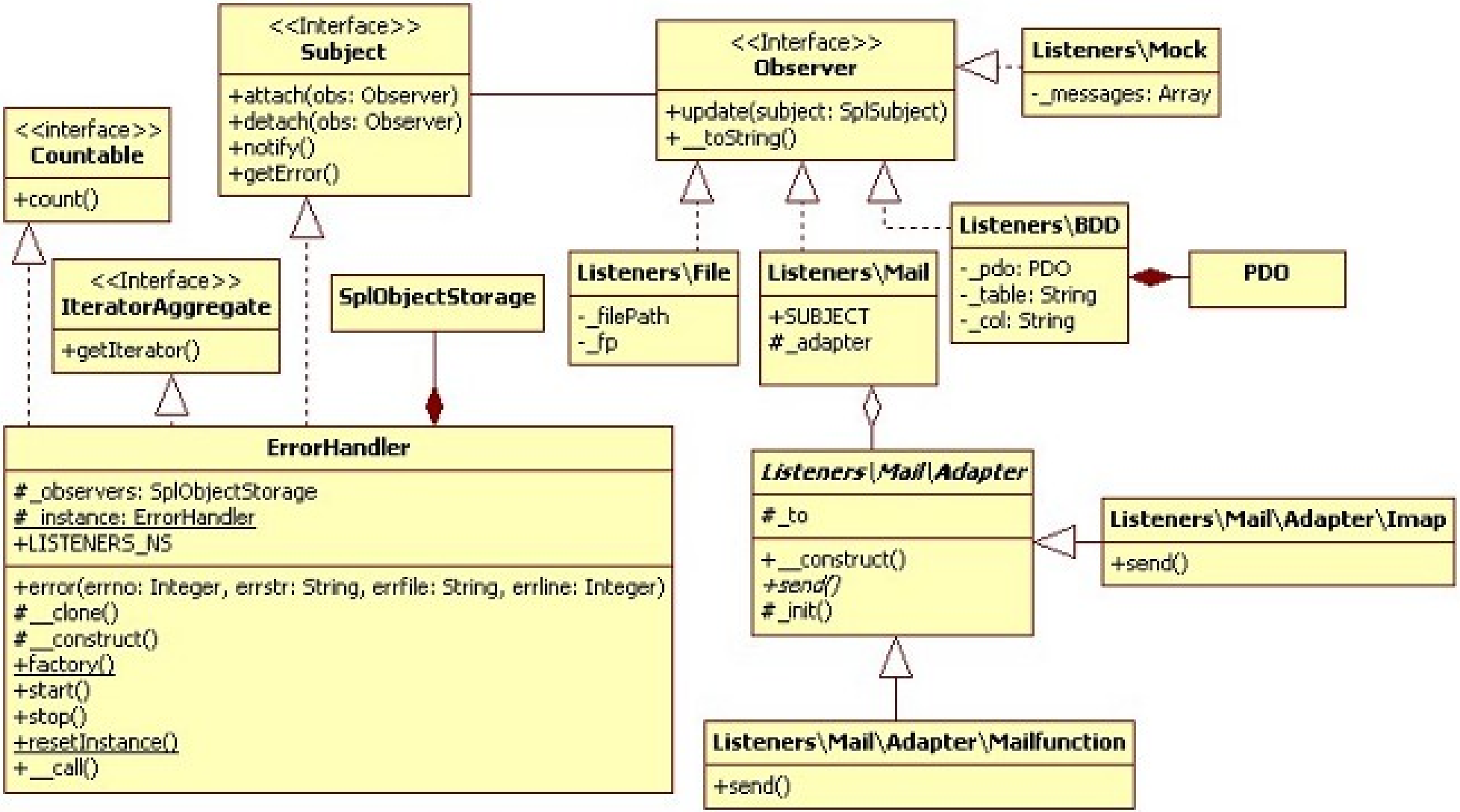


Observateur / Sujet codé

```
class SomeSubject implements Subject
{
    protected $observers;
    protected function __construct() {
        $this->observers = new SplObjectStorage();
    }
    protected function attach(Observer $o) {
        $this->observers->attach($o);
    }
    public function detach(Observer $o) {
        $this->observers->detach($o);
    }
    public function doSomething() {
        foreach ($this->observers as $o) {
            $o->update($this);
        }
    }
}
```

```
class SomeObserver
implements Observer
{
    public function
update(Subject $s) {
        // let's do something here
    }
}
```

Observateur / Sujet concrêt



Décorateur

- Permet de résoudre la problématique de l'héritage multiple

```
class Stylo {
    public function ecrire($what) {
        return "Le stylo écrit $what";
    }
}
```

```
class StyloGras extends Stylo {
    public function ecrire($what) {
        return "<b>" . parent::ecrire($what) . "</b>";
    }
}
```

```
class StyloItalic extends Stylo {
    public function ecrire($what) {
        return "<i>" . parent::ecrire($what) ."</i>";
    }
}
```

Décorateur

Implémentation PHP

```
interface Ecrivant {
    public function ecrire($what);
}
```

```
abstract class Decorator implements Ecrivant {
    protected $_decorator;
    public function __construct(Ecrivant $s) {
        $this->_decorator = $s;
    }
}
```

```
class DecoratorItalic extends Decorator {
    public function ecrire($what) {
        return "<i>" . $this->_decorator->ecrire($what) . "</i>";
    }
}
```

Décorateur

- Passage d'un modèle vertical (héritage) à un modèle horizontal (agrégation)

```
$obj = new DecoratorItalic(new DecoratorBold(new Stylo));
echo $obj->ecrire("Voyez vous cela ?");
```

Itérateur

- Déjà présent dans PHP par défaut

- Redéfinit l'opérateur **foreach**
- Définit la manière dont PHP parcourt un objet

```
$obj->rewind();
while ($obj->valid()) {
     printf("%s : %s \n", $obj->key(), $obj->current());
    $obj->next();
}
```



```
foreach ($obj as $k => $v) {
     printf("%s : %s \n", $k, $v);
}
```

Encore des patterns !

- Proxy
- Poids plume
- Pont
- Adaptateur
- MVC
- Visiteur
- ...

Où trouver des patterns ?

- Sur mon GitHub :)
  - https://github.com/jpauli
  - Ils sont là à titre éducatif
- Concrètement :
  - Dans les frameworks hautement orientés objets
    - ZendFramework
    - Symfony
- On les trouve aussi dans beaucoup d'autres projets PHP
  - Pour peu qu'ils soient un minimum réfléchis et pros

Se documenter sur les patterns

- Les langages OO sont mis en avant
  - >> C++ , Java
- Les exemples en PHP sont donc rares
- Il faut comprendre les langages OO pour faire du bon design applicatif en PHP

Se documenter sur les patterns en PHP

- Lecture et analyse de codes existants
  - https://github.com/symfony/symfony
  - https://github.com/zendframework/zf2
  - https://github.com/manuelpichler/phpmd
- Utilisation d'autres langages OO
  - Java
  - C++
  - Ruby&Rails

Merci !

- Questions ?