formation.tech

PHP Objet

PHP Objet - Introduction

- Programmation orientée objet apparue dans les années 70 avec le langage Smalltalk
- Application de principe de tous les jours à la programmation
- Paradigme de programmation (style de programmation)
- Architecture l'application autour des données plutôt que des traitement (programmation procédurale)
- Facilite les tests, la réutilisation de code, le travail à plusieurs

PHP Objet - Classe vs Objet

- Classe
  - Un concept (Ordinateur)
  - Un type
  - Permet la création d’objets (un moule)
- Objet
  - Représentation en mémoire d’une classe
  - Plusieurs objets possibles pour une classe

PHP Objet - Classe

- Une classe : un concept
- Un objet : une représentation en mémoire de ce concept

```php
<?php

class Computer
{
    public $brand;
    public $model;
}

$macbook = new Computer;
$macbook->brand = 'Apple';

echo "This MacBook was built by $macbook->brand";
```

PHP Objet - this

- Le mot clé this est une référence interne à l'objet

```php
<?php

class Computer
{
    public $brand;
    public $model;

    public function showInfos() {
        echo "This $this->model was built by $this->brand";
    }
}

$macbook = new Computer;
$macbook->brand = 'Apple';
$macbook->model = 'MacBook';
$macbook->showInfos();
```

PHP Objet - Encapsulation

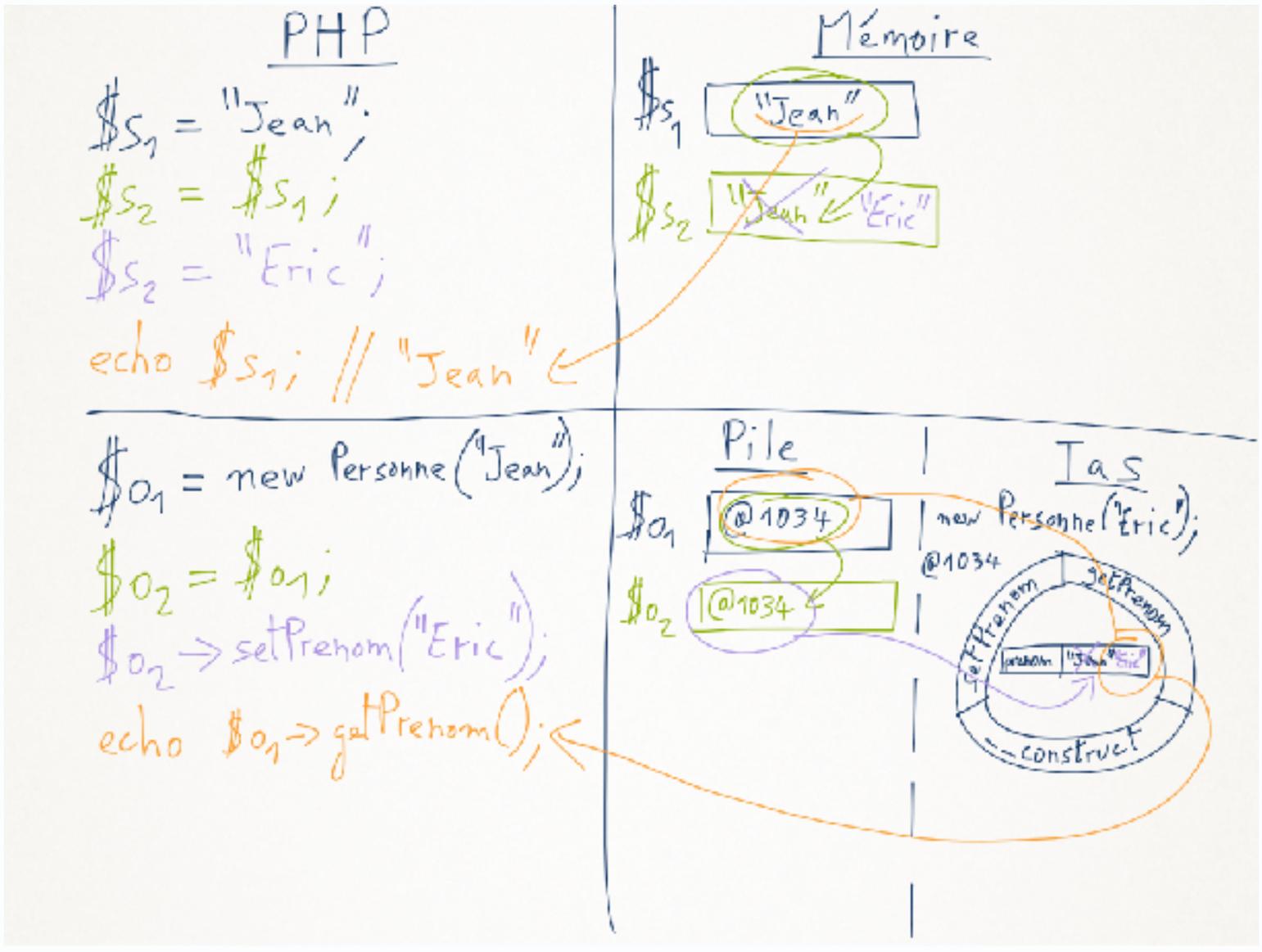
- Principe d'encapsulation
  Un objet doit être vu comme une boite noire, son fonctionnement interne peut-être complexe, son utilisation doit être simple. Les propriétés ne sont jamais publiques. Utilisation de getters/setters, permet également de limiter à la lecture seule et d'encapsuler les règles de validation.

```php
<?php

namespace Application\Entity;

class Contact
{
    protected $prenom;
    protected $nom;

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

    public function setNom($nom)
    {
        $this->nom = $nom;
    }

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

    public function setPrenom($prenom)
    {
        $this->prenom = $prenom;
    }
}
```

PHP Objet - Encapsulation

Référence

A la différence des autres types (int, boolean, float, string, array…) les objets se manipulent au travers de références.



PHP Objet - Association

Association

Un des propriété d'un objet contient une référence vers un autre objet.

```php
<?php

namespace Application\Entity;


class Contact
{
    protected $id;
    protected $prenom;
    protected $nom;
    protected $email;
    protected $telephone;

    /**
     * @var Societe
     */
    protected $societe;
}
```

```php
<?php

namespace Application\Entity;


class Societe
{
    protected $nom;
    protected $siteweb;
}
```

PHP Objet - Héritage

Héritage

Une classe réutilise les membres d'une autre classe.

```php
<?php

namespace Application\Entity;


class Contact
{
    protected $id;
    protected $prenom;
    protected $nom;
    protected $email;
    protected $telephone;

    /**
     * @var Societe
     */
    protected $societe;
}
```

```php
<?php

namespace Application\Entity;


class Salarie extends Contact
{
    protected $salaire;
}
```

PHP Objet - Interface

- ## Interface

  La définition d'une liste de méthodes, implémenter une interface oblige à implémenter ses méthodes.

  Dans ZF2 : leur nom se termine toujours par Interface

```php
<?php

namespace Application\Entity;

use Zend\Stdlib\ArraySerializableInterface;

class Contact implements ArraySerializableInterface
{
    // ...

    public function exchangeArray(array $array)
    {
        foreach ($array as $key => $value) {
            if (property_exists($this, $key)) {
                $this->$key = $value;
            }
        }
    }

    public function getArrayCopy()
    {
        return get_object_vars($this);
    }
}
```

```php
<?php

namespace Zend\Stdlib;

interface ArraySerializableInterface
{
    public function exchangeArray(array $array);

    public function getArrayCopy();
}
```

PHP Objet - Classe abstraite

- ## Classe abstraite

  Une classe qui n'a pour vocation à être utilisé qu'au travers d'un héritage. Peut également imposer l'implémentation de certaines méthodes comme une interface.
  Comme pour une interface, c'est la garantie que certaines méthodes seront bien présentes (programmation par contrat).
  Dans ZF2 leur nom commence toujours par Abstract.

```php
<?php

namespace Zend\Mvc\Controller;

abstract class AbstractActionController extends AbstractController
{
    // ...
}
```

```php
<?php

namespace AddressBook\Controller;

use Zend\Mvc\Controller\AbstractActionController;

class ContactController extends AbstractActionController
{
// ...
}
```

PHP Objet - Classe abstraite

- PHP FIG
  PHP Framework Interop Group, groupe de travail regroupant les principaux créateurs de frameworks/bibliothèques créant des normes PHP.
  http://www.php-fig.org/
- PSR-1
  PSR-0 + Basic Coding Standard
- PSR-2
  PSR-1 + Coding Style Guide
- PSR-3
  Logger Interface
- PSR-4
  Improved Autoloading
- PSR-7
  HTTP Message Interface

PHP Objet - Composition

- ## Composition

  Une composition est un type d'association forte entre 2 objet. La destruction d'un objet entrainerait la destruction de l'objet associé.
  Exemple : Un objet Tasse est composée de Café

```php
<?php

namespace ExpressoComposition;

class Cafe
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```php
<?php

namespace ExpressoComposition;

class Tasse
{
    protected $contenu;

    public function __construct() {
        $this->contenu = new Cafe("Arabica", "Mexique");
    }
}
```

```php
<?php

require_once 'autoload.php';

$tasseDeCafe = new \ExpressoComposition\Tasse();
```

- ## Mauvaise Pratique

  La composition est désormais considéré comme une mauvaise pratique. Premièrement la classe Tasse n'est très réutilisable, elle ne peut contenir que du café. De plus il n'est pas possible d'écrire d'écrire un test unitaire de Tasse, puisqu'il faudrait en même temps tester Café.
  Globalement il faut essayer de proscrire l'utilisation de new il l'intérieur d'une classe (à l'exception des Values Objects (DateTime, ArrayObject, etc…)

PHP Objet - Injection de Dépendance

- ## Solution

  La solution est simple, pour éviter le new dans cette classe nous allons injecter la dépendance.

```php
<?php

namespace ExpressoInjection;

interface Liquide {

}
```

```php
<?php

namespace ExpressoInjection;

class Cafe implements Liquide
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```php
<?php

namespace ExpressoInjection;

class Tasse
{
    protected $contenu;

    public function __construct(Liquide $contenu) {
        $this->contenu = $contenu;
    }
}
```

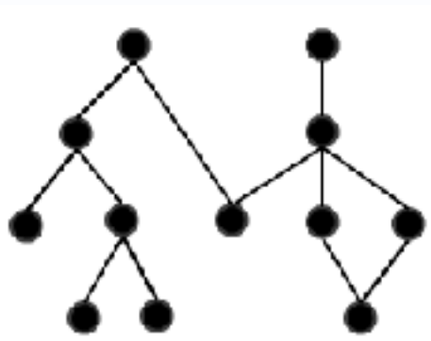
```php
<?php

require_once 'autoload.php';

$cafe = new \ExpressoInjection\Cafe();
$tasseDeCafe = new \ExpressoInjection\Tasse($cafe);
```

- La classe Tasse peut désormais recevoir n’importe quel contenu qui implémente l’interface Liquide.

PHP Objet - Injection de Dépendance

- ## Conteneur d'injection de dépendance (DIC)

  Le problème lorsqu'on injecte les dépendances est qu'on peut parfois se retrouver avec des dépendances complexes :



- Dans ce cas il devient utile d'utiliser un conteneur d'injection de dépendance qu'on aura configuré au préalable. En PHP il existe quelques DIC connus :
  - Pimple par Fabien Potencier
  - Dice par Tom Butler
  - PHP-DI par Matthieu Napoli
  - Symfony\Container intégré Symfony2
  - Zend\Di & Zend\ServiceManager intégrés ZF 2

- Zend\Di est très simple à configurer mais peu performant (utilise la Reflection), Zend\ServiceManager est le plus utilisé.
  Documentation : http://framework.zend.com/manual/current/en/modules/zend.service-manager.quick-start.html