Atelier

DESIGN PATTERNS

Gérald Croës - gerald@copix.org

http://gcroes.com/conf/quebec/2009/atelier_design_pa

Objectifs de la présentation

- Comprendre ce qu'est un pattern
- En connaître les principaux représentants
- Exemples d'utilisation
- Des ressources pour aller plus loin

- Apporter des idées conceptuelles

Définition

- Modèles de conception

  parfois aussi « Motifs de conception » ou « Patrons de conception ».

- Solutions à des problèmes classiques.

  (Règle de 3 utilisations)

- Indépendants du langage.

Ils sont partout...

- Composition musicale (cadence, apogiatures, tonalité, rubato, ...)
- Communication (intonations dans le discours, métaphores et exemples, ...)
- Graphisme, ergonomie (positionnements, codes couleurs, ...)
- ...

Tout ce qui représente une façon de procéder pour arriver à un résultat

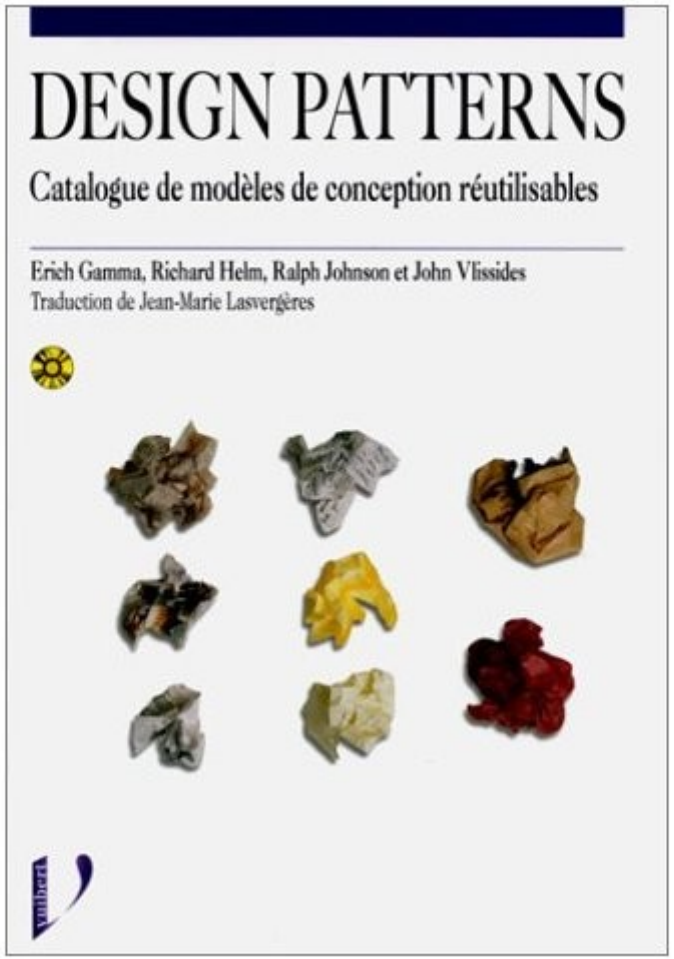
Apparition des patterns

- C. Alexander « A Pattern Language: Towns, Buildings, Construction » [1977]

« Les utilisateurs connaissent mieux le bâtiment dont ils ont besoin que les architectes »

« Chaque modèle décrit un problème qui se manifeste constamment dans notre environnement, et donc décrit le coeur de la solution à ce problème, de telle façon que l'on puisse la réutiliser des millions de fois et ce jamais de la même manière » [AIS+ 77]

L'ouvrage de référence



- GoF « Gang of Four »

Design patterns. Elements of reusable Object-Oriented Software [1994]

Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides

Pourquoi les étudier ?

- Catalogue de solutions.
- Bénéficier du savoir faire d'experts dans des contextes éprouvés. (fiables, robustes & connus)
- Facilite la conception.

Pourquoi les utiliser ?

- Ne pas réinventer la roue.
- Facilite la communication entre développeurs.
- Pour résoudre un problème

Fiche d'identité

- Nom
- Description du problème
- Description de la solution
  - exemples, modèles, liste des éléments et des relations
- Conséquences
  - critiques, impacts sur l'application

« Par convention, 3 utilisations couvertes de succès »

Les 5 patterns de création

- Création
  - Factory, AbstractFactory, Builder, Prototype, Singleton
  -
- Abstraction du processus de création.
- Encapsulation de la logique de création.
- On ne sait pas à l'avance ce qui sera créée ou comment cela sera créé.

Les 7 patterns de structure

- Structure

  Adaptator, Bridge, Composite, Decorator, Interface, Flyweight, Proxy

  d

- Comment sont assemblés les objets.
- Découpler l'interface de l'implémentation.

Les 11 patterns comportementaux

- Comportement
  - Interpretor, Template method, Chain of responsability, Command, Iterator, Mediator, Memento, Observator, State, Strategy, Visitor
- Mode de communication entre les objets

Quelques exemples pour commencer

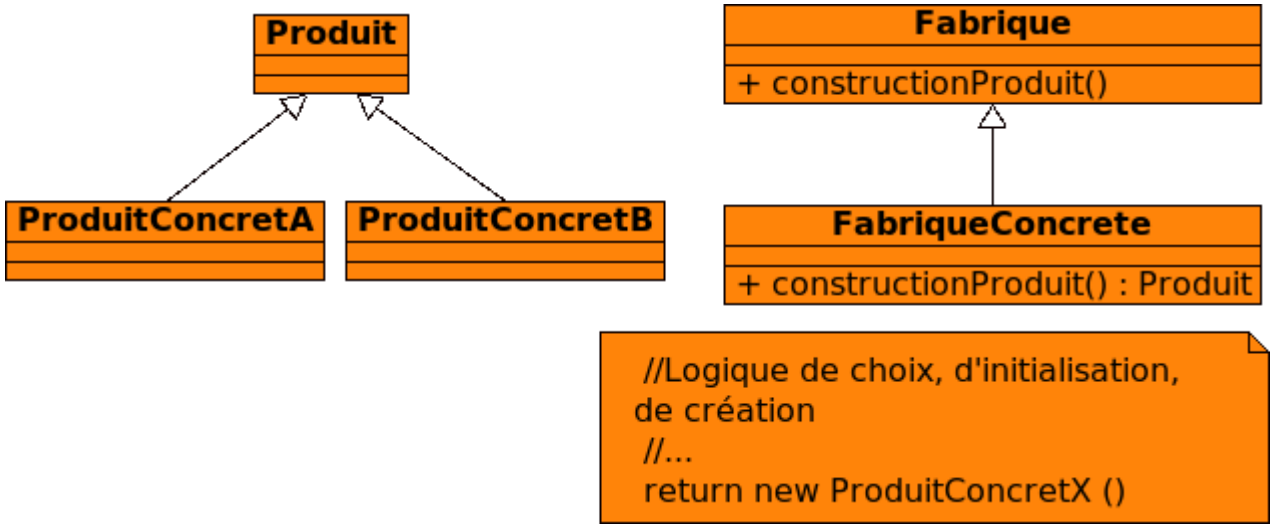
- Factory
- Singleton
- Adaptator
- Decorator
- Observer
- Iterator

[Création] Factory

- Problématique : Obtenir facilement un objet prêt à l'emploi et qui correspond à nos besoins.

- Solution : Une classe / Une méthode qui encapsule la logique de création des objets en question.

[Création] Factory, diagramme



[Création] Factory, exemple

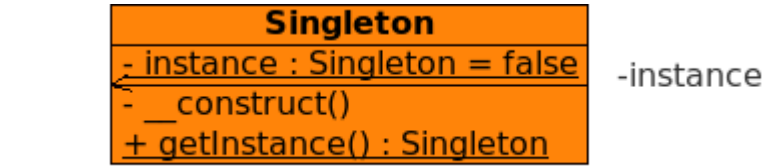
```
require_once ('DBAbstract.class.php');
class DBFactory {
   static function create ($dataSourceId){
      switch (self::_getDriver ($dataSourceId)){
         case self::MySQL :
            require_once ('DBMySQL.class.php');
            return new DBMySQL ($dataSourceId);
         case self::MySQLI :
            require_once ('DBMySQLI.class.php');
            return new DBMySQLI ($dataSourceId);
      }
   }
}
```

[Création] Singleton

- Problématique : S'assurer qu'il existe une seule instance d'un objet donné pour toute l'application.

- Solution : Une méthode statique pour contrôler l'instanciation. Rendre ce processus d'instanciation l'unique solution possible pour la classe en question.

[Création] Singleton, diagramme




```
if (self::$instance === false){
  self::$instance = new Singleton ();
}
return self::$instance;
```

[Création] Singleton, exemple

```
class ApplicationConfiguration {
   private static $_instance = false;
   public static function getInstance (){
      if (self::$_instance === false){
         self::$_instance = new ApplicationConfiguration ();
      }
      return self::$_instance;
   }
   private function __construct (){
      //chargement du fichier de configuration
   }
}
```

[Création] Singleton, pièges & différences

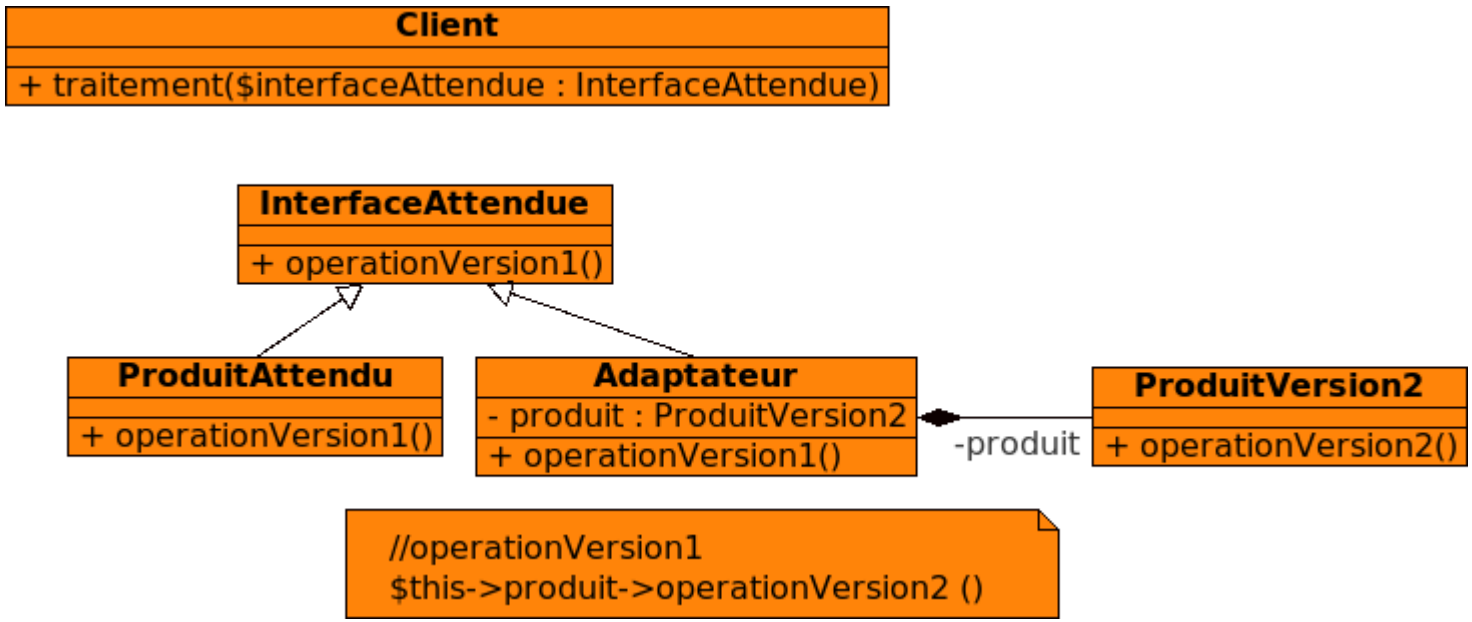
- N'est pas une classe statique
  - Une instance à manipuler.
  - Conserve un contexte
  - Peut être passé en paramètre à une méthode
  -
- N'est pas une variable globale
  - Eviter la singletonite

[Structure] Adaptator

- Problématique : Ajuster l'interface d'un objet à celle attendue par le code client.

- Solution : L'adaptateur conserve une instance de la classe adaptée et convertit les appels d'une interface existante vers l'interface implémentée.

[Structure] Adaptator, diagramme



[Structure] Adaptator, exemple

```
class ForumUser implements IForumUser {
   public function isModerator (){/** Implementation */}
}
class CompanyUser implements ICompanyUser {
   public function isTheBoss (){/** Implementation */}
}
class Forum  {
   public function getAstuceDuJour (IForumUser $user){
      If ($user->isModerator ()){
         /** Actions spéciales **/
      }
   }
}
```

[Structure] Adaptator, exemple (2)

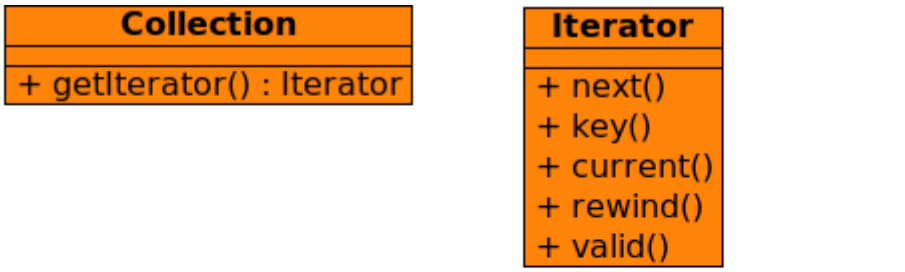
```
class CompanyUserForumAdaptator implements IforumUser {
   private $_companyUser;
   public function __construct (ICompanyUser $user){
      $this->_companyUser = $user;
   }
   public function isModerator (){
      return $this->_companyUser->isTheBoss ();
   }
}
//utilisation
$forum->astuceDuJour (new CompanyUserForumAdaptator ($user));
```

[Comportement] Iterator

- Problématique : Parcourir des collections d'objets diverses, éventuellement de différentes façons, sans risque pour le contenu.

- Solution : Utiliser un objet qui dispose de la connaissance nécessaire à la navigation dans la collection avec une interface unique.

[Comportement] Iterator, diagramme




```
foreach ($collection->getIterator () as $element){
  //...
}
```

[Comportement] Iterator & PHP

- SPL à la rescousse
  - « Iterator » Utilisable dans les foreach
  -
  - ArrayIterator, RecursiveArrayIterator, DirectoryIterator, RecursiveDirectoryIterator, RegexIterator, SimpleXMLIterator, ...
  -

[Comportement] SPL (Interfaces)

- Iterator
  - [current, key, next, valid, rewind]
- SeekableIterator
  - [(iterator), seek]
- RecursiveIterator
  - [(iterator), getChildren, hasChildren]
- FilterIterator
  - [accept]

[Comportement] SPL (plus d'Interfaces)

- ArrayAccess
  - [offsetGet, offsetSet, offsetExists, offsetUnset]
  - $monObjet['index'] = « valeur »;
- Countable
  - [count]
  - count ($monObjet);

[Comportement] SPL (Classes)

- DirectoryIterator, RecursiveDirectoryIterator
  - foreach ($directories = new DirectoryIterator ('./') as $file){ //...
- RecursiveIteratorIterator
  - Parcours un iterateur récursif comme un iterateur
  - foreach (new RecursiveIteratorIterator (new RecursiveDirectoryIterator ('./'))){
  - //...

[Comportement] Exemple sans Iterator

```
$hdl = opendir('./');
  while ($dirEntry = readdir($hdl))  {
    if (substr($dirEntry, 0, 1) != '.')  {
        if(!is_file($dirEntry)) {
            continue;
            }
            echo $dirEntry, '<br />;
        }
    }
  closedir($hdl);
```

[Comportement] Exemple avec Iterator

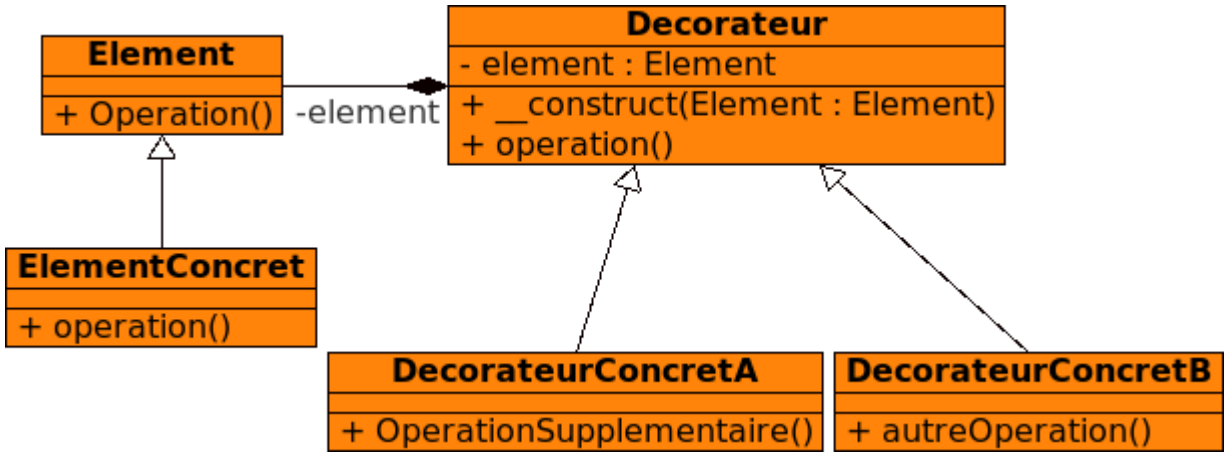
```
foreach ( new DirectoryIterator('./') as $Item )  {
   echo $Item.'<br />';
}
```

[Structure] Decorator

- Problématique : Rajouter des fonctionnalités à des composants existants sans utiliser l'héritage.

- Solution : Encapsuler l'objet existant et y ajouter des comportements nouveaux.

[Structure] Decorator, diagramme



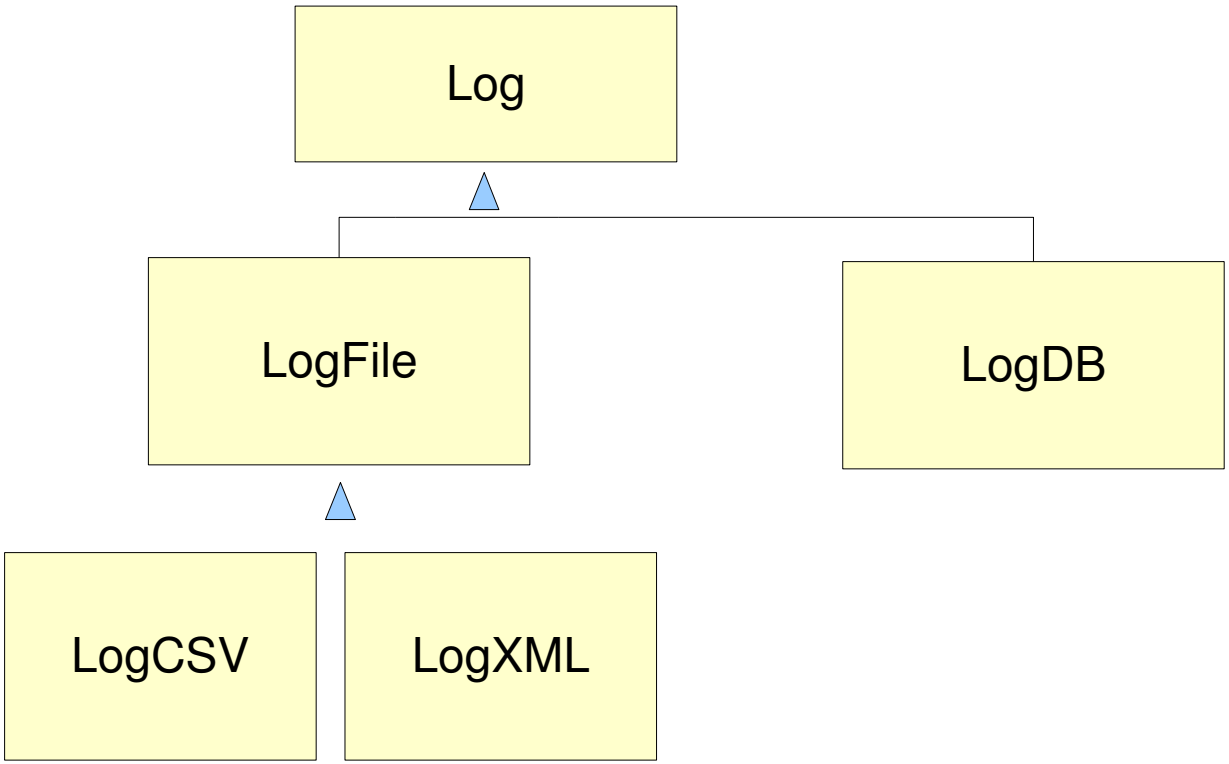
[Structure] Decorator, exemple

```
class decorated {}
abstract class Decorator {
    private $_decorated;
    function __construct ($decorated){
        $this->_decorated = $decorated;
    }
    function operaton (){
        return $this->_decorated->operation ();
    }
}
```

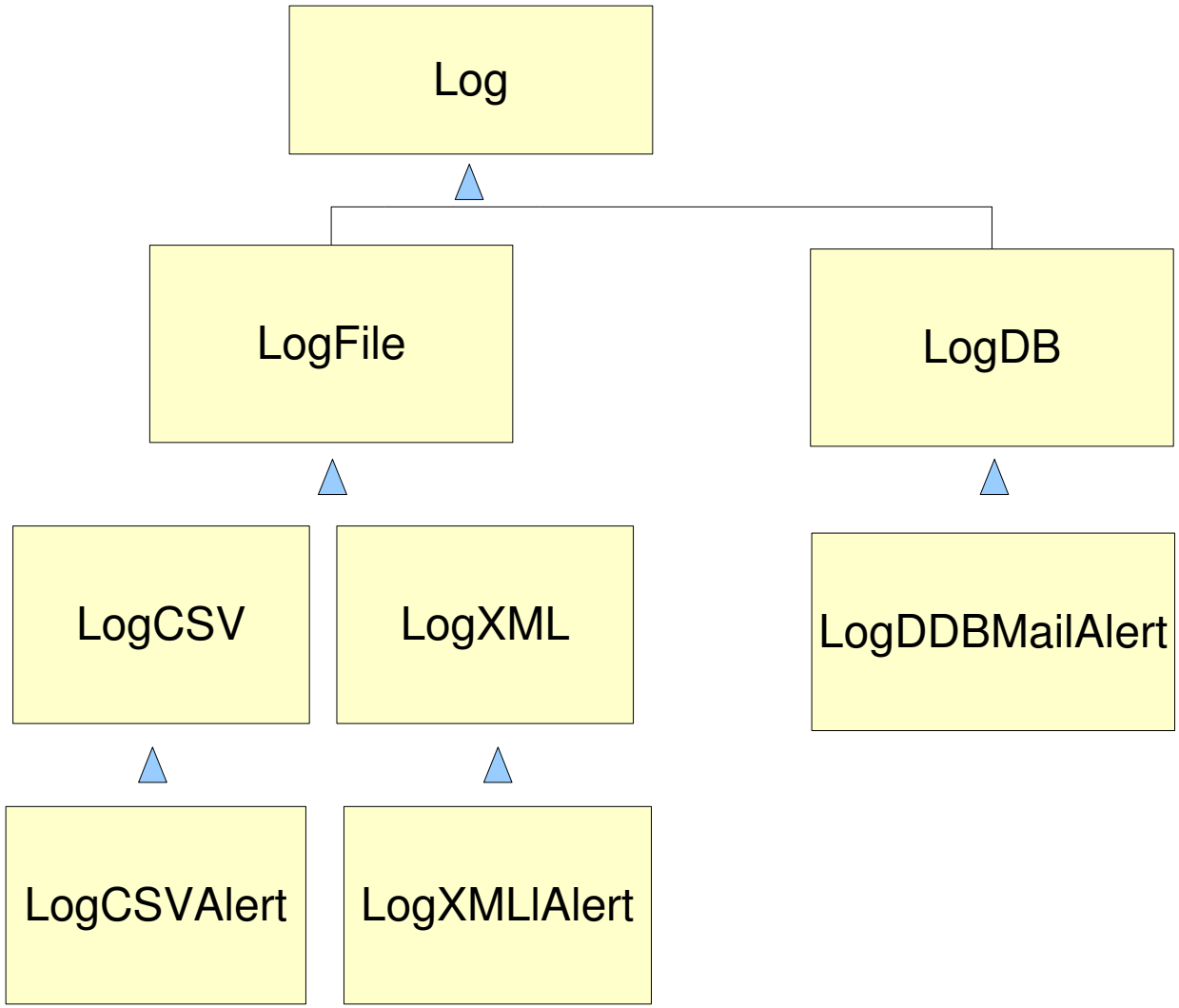
[Structure] Decorator, exemple (2)

```
class Decorator1 extends Decorator {
    function operation2 (){
        //autre traitement
    }
}
class Decorator2 extends Decorator {
    function operation3 (){
        //...
    }
}
```

[Structure] Decorator VS Héritage

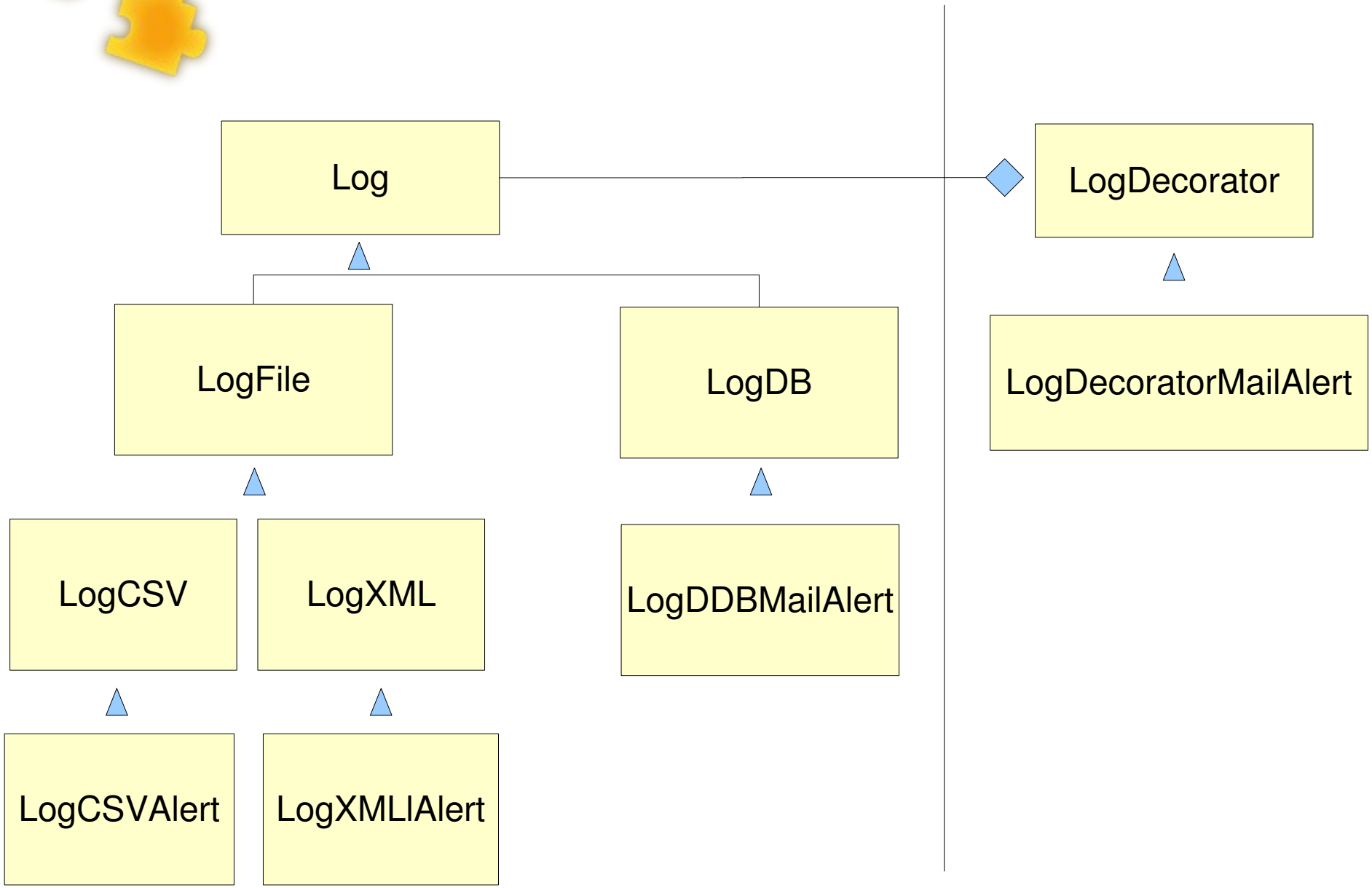


[Structure] Decorator VS Héritage



- Une arborescence qui se complexifie

[Structure] Decorator VS Héritage



[Comportement] Exercice

En utilisant un pattern que nous avons vu précédemment et une interface de la SPL, comment faire un itérateur capable de parcourir un ensemble de fichiers, que l'on pourra filtrer selon l'extension, respectant l'interface ci dessous ?

```
interface IExtensionFilterIterator implements Iterator {
   public function setExtension ($pExt);
}
```

[Comportement] Exercice

```
class ExtensionFilterIteratorFileDecorator extends FilterIterator {
  private $_ext;
  public function accept (){
    if (file_exists ($this->current ())){
      return substr ($this->current (), -1 * strlen ($this->_ext)) === $this->_ext;
    }
    return false;
  }
  public function setExtension ($pExt){
    $this->_ext = $pExt;
  }
}
```

[Comportement] Exercice (2)

```
//Utilisation dans un seul répertoire

$directories = new ExtensionFilterIteratorFileDecorator (new DirectoryIterator
('./framework/'));

$directories->setExtension ('.class.php');



//Utilisation avec une arborescence

$directories = new ExtensionFilterIteratorFileDecorator (new RecursiveIteratorIterator
(new RecursiveDirectoryIterator ('./framework/')));

$directories->setExtension ('.class.php');



foreach ($directories as $directory){

   echo $directory->getFileName (), '<br />';

}
```

Quizz

Quels patterns sont mis a contribution dans

RecursiveIteratorIterator ?

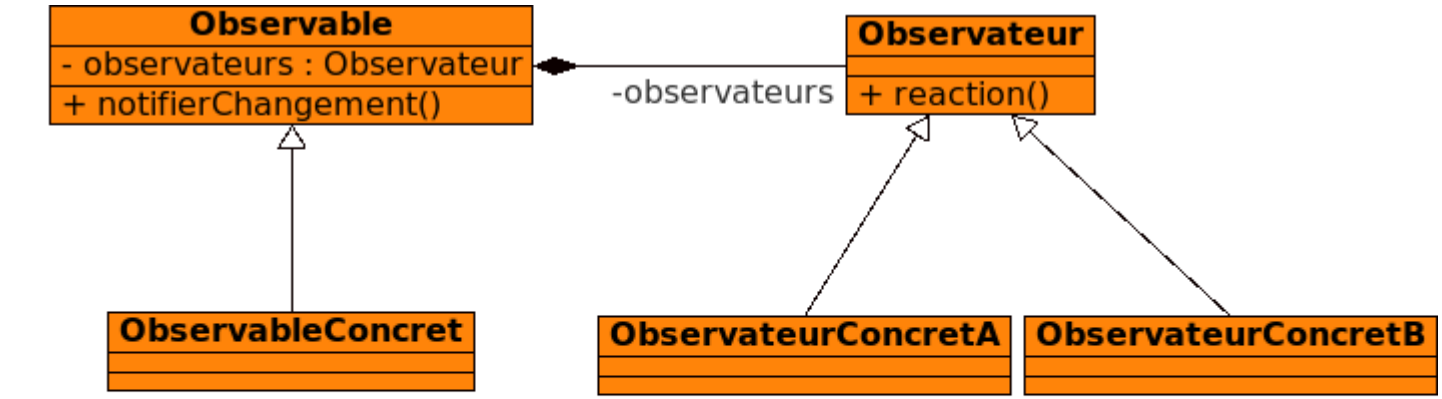
FilterIterator ?

[Comportement] Observer

- Problématique : Permettre à un objet de réagir aux comportement d'un autre sans pour autant les lier « en dur ».

- Solution : Définir un objet comme « observable » et donc capable de notifier des changements à des observateurs, quels qu'ils soient.

[Comportement] Observer




```
//notifierChangement
foreach ($this->obervateurs as $o){
   $o->reaction ();
}
```

[Comportement] Observer & Spl

- SplSubject
  - attach (SplObserver $observer)
  - dettach (SplObserver $observer)
  - notify ()
  -
- SplObserver
  - update (SplSubject $subject)

[Comportement] Exercice

- En utilisant SplSubject et SplObserver, comment faire un système de log capable d'envoyer des mails / écrire dans un fichier / afficher à l'écran les informations envoyées ?

[Comportement] Exercice (2)

```
class LogBase implements SplSubject {
  private $_lastMessage;
  private $_observers = array ();
  public function add ($pMessage) {
    $this->_lastMessage     = $message;
    $this->notify ();
  }
  public function getLastMessage (){
     return $this->_lastMessage;
  }
```

[Comportement] Exercice (3)

```
public function attach(SPLObserver $pObserver) {
  $this->_observers[] = $pObserver;
  return $this;
}
public function detach (SPLObserver $pObserver)  {
  if (is_int ($key = array_search ($pObserver, $this->_observers, true))) {
    unset ($this->_observers[$key]);
  }
  return $this;
}
public function notify () {
  foreach ($this->_observers as $observer) {
      $observer->update ($this);
}
}
```

[Comportement] Exercice (4)

```
class FileSaver implements SplObserver {
   public function update (SplSubject $log){
      //écriture dans un fichier du log
   }
}
class MailSender implements SplObserver {
   public function update (SplSubject $log){
      //Envois du log par mail
   }
}
/Utilisation
$logBase = new LogBase ();
$logBase->attach (new FileSaver ('/tmp/application/log.txt'))
            ->attach (new MailSender ('someadress'));
$logBase->add ('Mon message');
```

Quizz

Quel pattern pourrait-on mettre a contribution pour avoir une seule instance de logBase accessible dans notre application ?

Quel pattern pourrait-on mettre a contribution pour obtenir rapidement un « LogBase » correctement configuré avec ses observers ?

Et plus encore....

- Abstract Factory
- Builder
- Proxy
- Interface
- Visitor
- Chain of Responsability

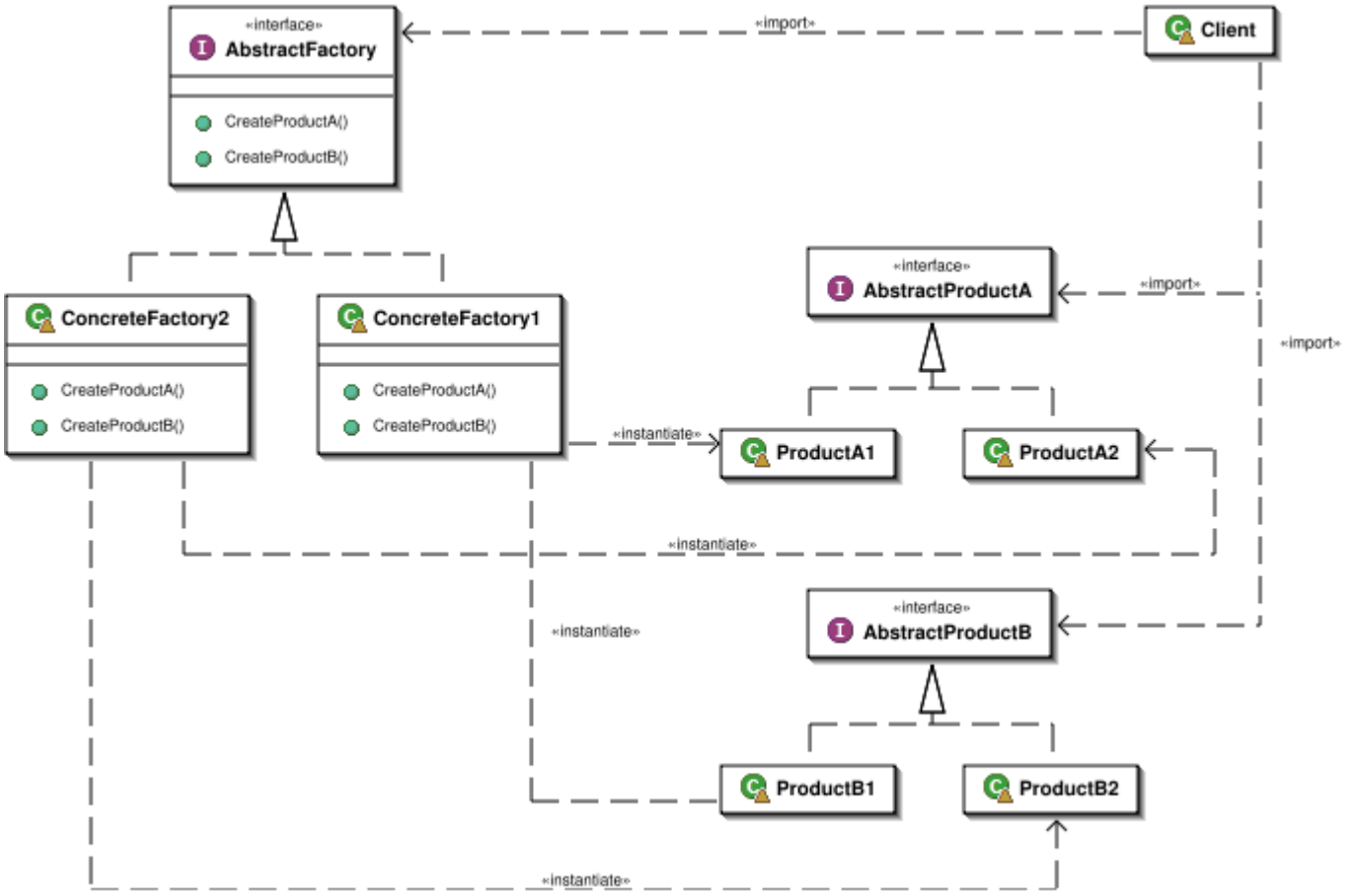
d

[Créateur] Abstract Factory

- Problématique : Disposer d'une interface pour créer des familles d'objets.
- Solution : Une fabrique de fabrique.

[Créateur] Abstract Factory, diagramme

Création d'une famille d'objet



[Créateur] Abstract Factory, exemple

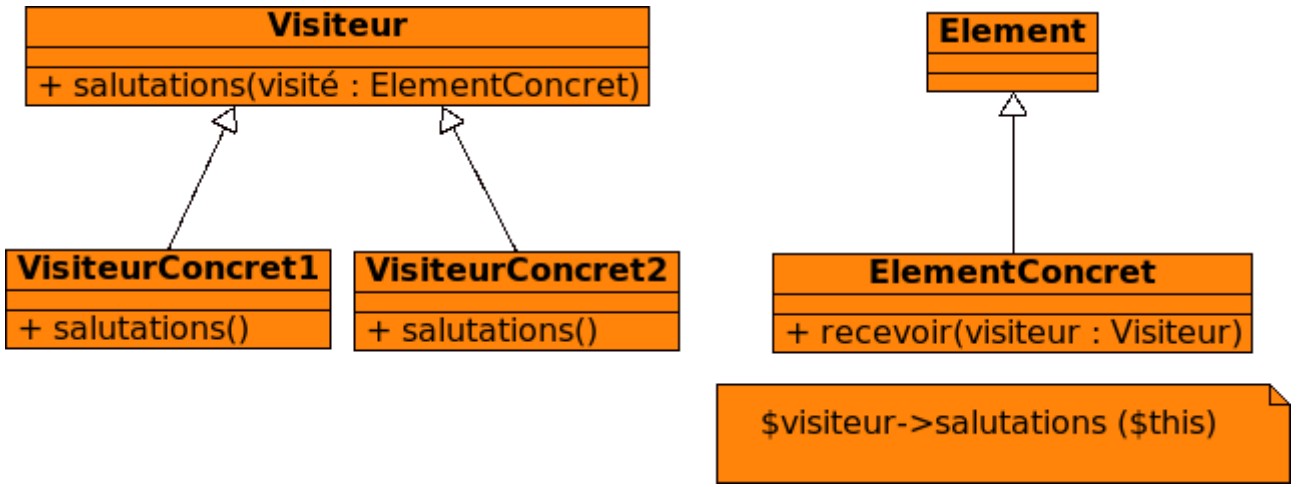
```
abstract class AbstractDocumentFactory {

   abstract public function getDevis ($dataDevis);

   abstract public function getAdhesion ($dataAdhesion);

}

class PdfDocumentFactory extends
      AbstractDocumentFactory {}

class HTMLDocumentFactory extends
      AbstractDocumentFactory {}


abstract class Devis ();

abstract class Adhesion ();
```

[Structuraux] Visitor

- Problématique : On souhaite réaliser des opérations sur les éléments d'un objet sans pour autant connaître à l'avance le résultat à obtenir.

- Solution : On utilise un objet tiers (visiteur) qui sera capable d'obtenir le résultat souhaité à partir des données.

[Structuraux] Visitor, diagramme



[Structuraux] Visitor, Exercice

- Nous avons une classe DocumentHTML succeptible de contenir des définitions, fautes d'orthographes, liens obsolètes, … qu'il est nécessaire de traiter avant affichage.
- Comment peut-on profiter du pattern visiteur afin de prendre en charge ces problèmes ?

[Structuraux] Visitor, Exercice (2)

```
interface IDocument {
   public function getContent ();
   public function setContent ($pContent);
   public function filtre (IDocumentFilter $visitor);
}

interface IDocumentFilter {
   public function apply (IDocument $document);
}
```

[Structuraux] Visitor, Exercice (3)

```
class Document implements IDocument {
   private $_content;
   public function getContent (){
      return $this->_content;
   }
   public function setContent ($pContent){
      $this->_content = $pContent;
   }
   public function filtre (IDocumentFilter $visitor){
      $visitor->apply ($this);
   }
}
```

[Structuraux] Visitor, Exercice (4)

```
class DocumentAccentFilter implements
IDocumentFilter {

   public function apply (IDocument $document){

      $document->setContent (str_replace ('é', '&eacute;',
$document->getContent ()));

   }

}

class DocumentLinkFilter //...

class DocumentMetaFilter //..

class DocumentStatsFilter //..
```

[Structuraux] Visitor, Exercice (5)

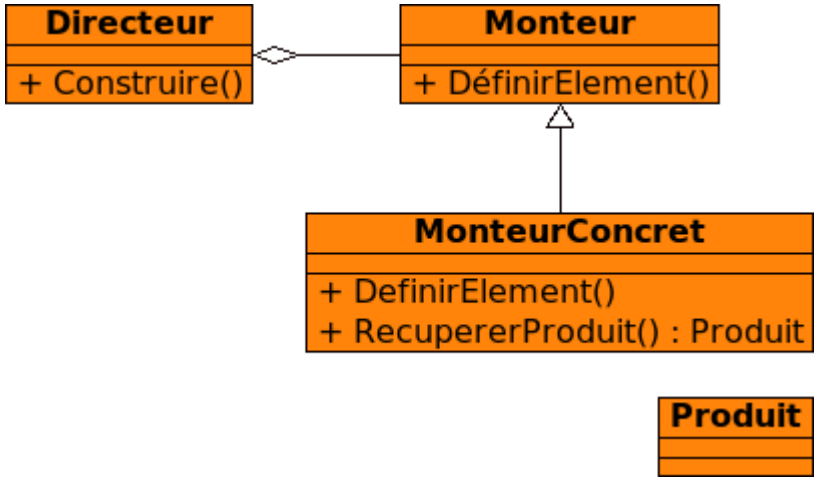
```
public function showDocument (IDocument $document){
   foreach (ApplicationConfiguration::get ()->filters as $filterName){
      $document->filter (FilterFactory::create ($filterName));
   }
}
```

[Créateur] Builder

- Problématique : Bien que la façon d'initialiser l'objet soit la même, il doit être possible d'obtenir différents résultats.

- Solution : Utiliser un Monteur qui implémente les étapes de créations.

[Créateur] Builder, diagramme



[Créateur] Builder, Exercice

- Notre application construit des documents en assemblant des composants classiques (paragraphes, titres, ...).
- Il doit être possible, a partir de recettes connues, de générer ces documents dans divers formats.

[Créateur] Builder, exemple

```
abstract class DocumentBuilder {
   private $_document = false;
   function getDocument (){
      if ($this->_document === false){
         $this->_document = new TextDocument ();
      }
      return $this->_document;
   }
   abstract public function doTitre ();
   abstract public function doParagraphe ();
}

class DocumentBuilderFactory {
   function getDocumentBuilder (){
      switch (Config::getInstance ()->documentType){
         case 'HTML' : return new HTMLBuilder ();
         case 'Wiki' : return new WikiBuilder ();
      }
   }
}
```

[Créateur] Builder, exemple 2

```
class Document {
   private $_content = '';
   function __construct ($baseContent = ''){
      $this->_content = $baseContent;
   }
   function getContent (){
      return $this->_content;
   }
   function add ($text){
      $this->_content .= $text;
   }
}
```

[Créateur] Builder, exemple 3

```
class HTMLBuilder extends DocumentBuilder {
   function doTitre ($texte, $niveau){
      $this->getDocument ()->add ("<h$niveau> $texte </h$niveau>");
   }
   function doParagraphe ($texte){
      $this->getDocument ()->add ("<p>$texte</p>");
   }
}

class WikiBuilder extends DocumentBuilder {
   functon doTitre ($texte, $niveau){
      $this->getDocument ()->add (str_repeat ('!', $niveau).$texte."\n\r");
   }
   function doParagraphe ($texte){
      $this->getDocument ()->add ("\n\r".$texte."\n\r");
   }
}
```

[Créateur] Builder, exemple 4

```
class Devis {
   private $_client = false;
   private $_produits = array ();

   //...

   function getDocument ($documentBuilder){
      $documentBuilder->doTitre ("Client");
      $documentBuilder->doParagraphe ($this->_client->toString ());
      foreach ($this->_produits as $produit){
         documentBuilder->doTitre ($produit->getNom ());
         documentBuilder->doParagraphe ($produit->getDescription ());
      }
   }
}
```

[Créateur] Builder, exemple code client

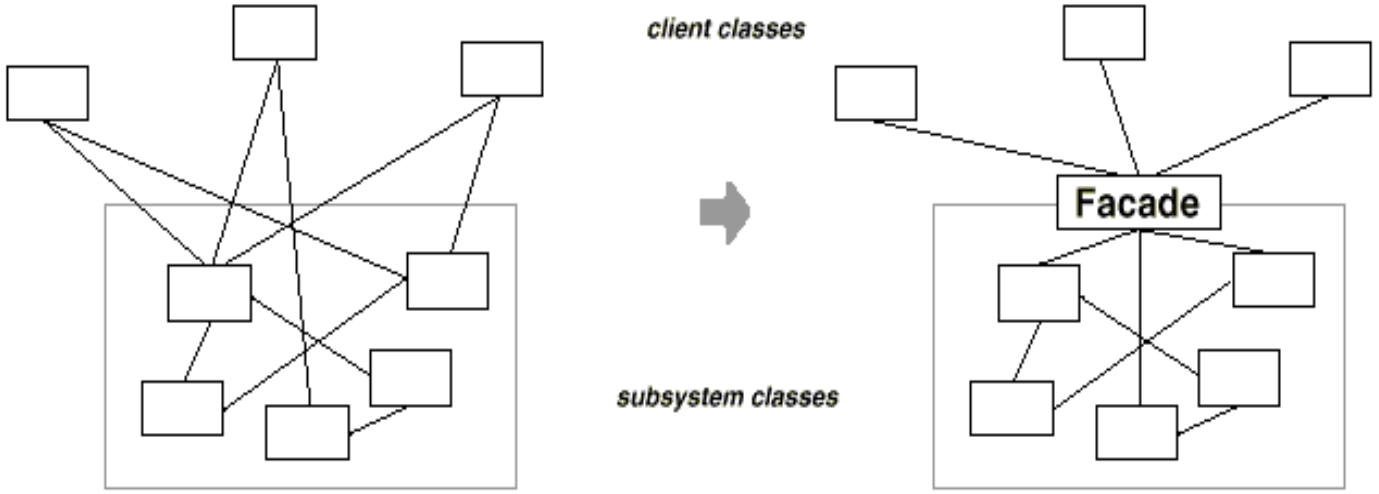
```
//...
function doDevis (){
   $devis = new Devis ();
   $devis->setClient (ClientSession::get ());
   foreach (HttpRequest::getProduit () as $id){
      $devis->addProduit ($id);
   }

   return new DownloadResponse ($devis->getDocument
        (DocumentBuilderFactory::getDocumentBuilder ()));
}
```

[Structuraux] Façade

- Problématique : Comment masquer la complexité d'une API qui réponds à un besoin simple.

- Solution : Un objet qui s'occupe de masquer la complexité des objets sous jacents.

[Structuraux] Façade, Schéma

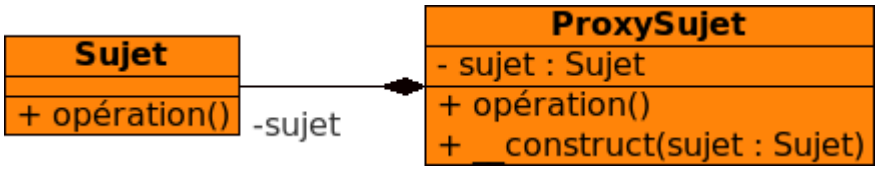


[Structuraux] Proxy

- Problématique : Donner l'accès à un objet sans avoir à le transmettre.

- Solution : Utiliser un Proxy qui va agréger l'objet et utiliser l'objet source pour réaliser les opérations.

[Structuraux] Proxy, diagramme



[Structuraux] Exercice

Comment profiter de certaines fonctions magiques de PHP pour créer facilement un proxy ?

[PHP] Fonctions magiques

- __get
- __set
- __isset
- __unset
- __call
- __callStatic (5.3+)
- __sleep
- __wakeup
- __clone

[Structuraux] Proxy, exemple

```
abstract class ClassProxy {
  protected $_object = false;
  public function __construct ($pObject){
    $this->_object = $pObject;
  }
  public function __call ($pName, $pArgs){
    $this->_beforeRemoteAction ();
    try {
      $toReturn = call_user_func_array (array ($this->getRemoteObject (),
      $pName), $pArgs);
    }catch (Exception $e){
      $this->_afterRemoteAction ();
      throw $e;
   }
   $this->_afterRemoteAction ();
   return $toReturn;
  }
```

[Structuraux] Proxy, exemple 2

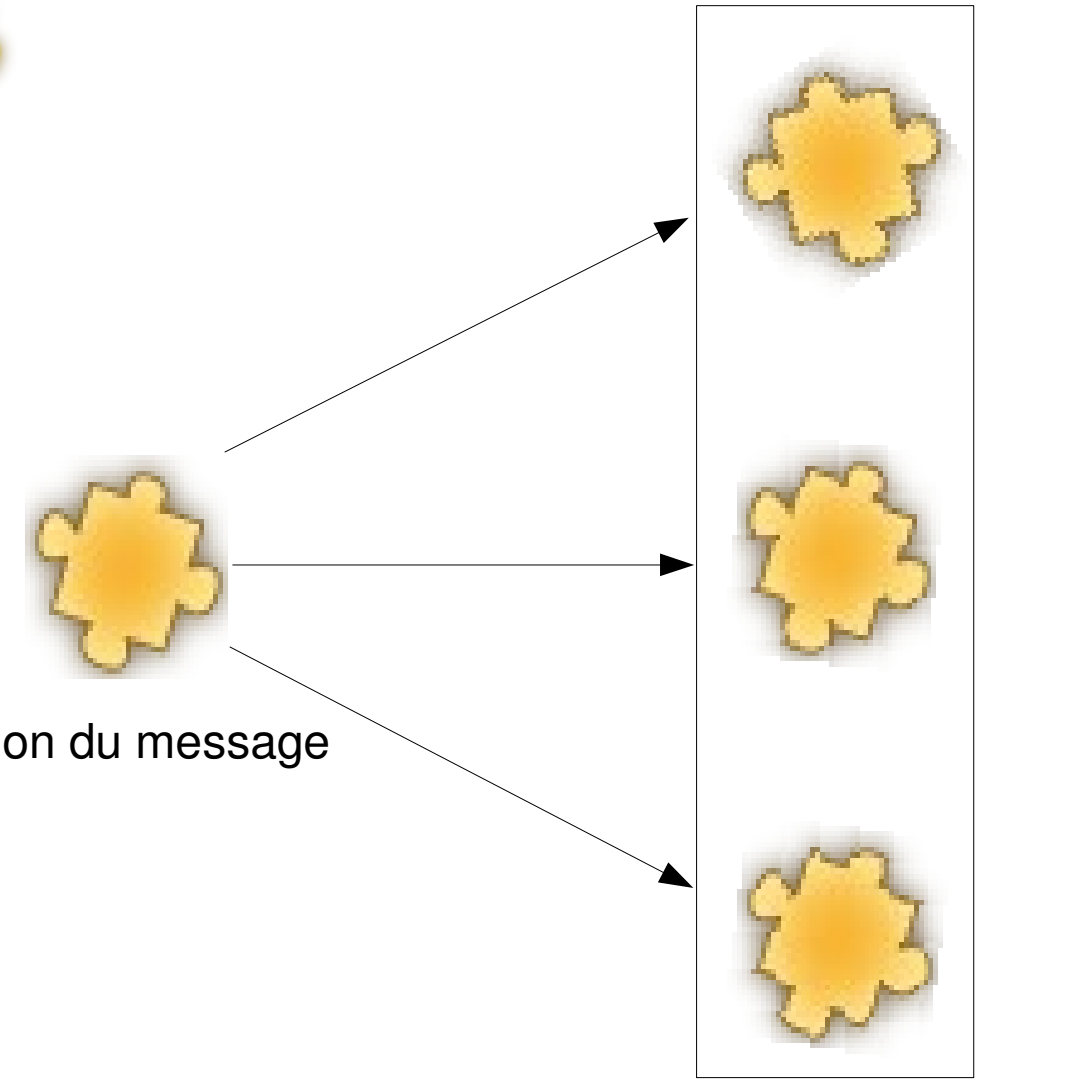
```
public function __get ($pName){
        $this->_beforeRemoteAction ();
        try {
                $toReturn = $this->getRemoteObject ()->$pName;
        }catch (Exception $e){
                $this->_afterRemoteAction ();
                throw $e;
        }
        $this->_afterRemoteAction ();
        return $toReturn;
}
public function __set ($pName, $pValue){
        $this->_beforeRemoteAction ();
        try {
                $this->getRemoteObject ()->$pName = $pValue;
        }catch (Exception $e){
                $this->_afterRemoteAction ();
                throw $e;
        }
        $this->_afterRemoteAction ();
```

[Comportement] Chain of Responsability

- Problématique : Plusieurs objets sont capables de traiter la demande, mais on ne sait pas lequel.

- Solution : On va passer successivement l'information à la collection d'objets.

[Comportement] Chain of Responsability



[Comportement] Exercice

- Un code client cherche une personne. Ce code client est en relation avec plusieurs moyens de recherche afin de la trouver. Comment profiter de ces moyens de recherches afin de trouver la personne en question ?

[Comportement] Exercice (2)

```
Interface IMoyenDeRecherche {
   public function trouve ($personne);
}

class PagesJaunes implements IMoyenDeRecherche {/**/}
class ChienDeBerger implements IMoyenDeRecherche {/**/}
class Internet implements IMoyenDeRecherche {/**/}
class BigBorther implements IMoyenDeRecherche {/**/}
```

[Comportement] Exercice (3)

```
class Enquete {
   /* ... */
   public function recherche ($nom){
      foreach ($this->_moyens as $moyen){
         If (($personne = $moyen->trouve ($nom)) !== false){
            return $personne;
         }
      }
      return null;
   }
}
```

Exercice (encore !)

A partir de tout ce que nous avons vu, créer un autoloader capable de charger automatiquement les classes situées dans une arborescence donnée.

Exercice (2)

```
class Autoloader {
  private $_preloadFiles = false;
  private function _preloadFiles (){
    if ($this->_preloadFiles === false){
      $files = new ExtensionFilterIteratorFileDecorator (new
     RecursiveIteratorIterator (new RecursiveDirectoryIterator ('.')));
      $files->setExtension ('.class.php');
      foreach ($files as $file){
        $this->_preloadFiles[strtolower ($file->getFileName ())] = $file-
>getRealPath ();
      }
    }
  }
  public function load ($pClassName) {
    $this->_preloadFiles ();
    if (isset ($this->_preloadFiles[strtolower ($pClassName)])){
      require ($this->_preloadFiles[strtolower ($pClassName)]);
    }
  }
}
//Enregistrement de notre autoloader dans la liste.
spl_autoload_register (array (new Autoloader (), 'load'));
```

Quizz

Quel pattern peut être identifié avec spl_autoload_register ?

Au delà du GoF

- J2EE
- Patterns of Enterprise Application Architecture (Martin Fowler)

d

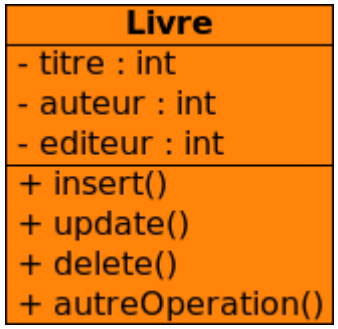
Au delà du GoF

- Active Record
- DAO / Data Mapper
- Lazy Load
- Front Controller
- MVC
- Query Object

Active record

- Problématique : Le modèle de données s'approche du modèle objet.
- Solution : Un objet qui contient l'ensemble

Active record, diagramme



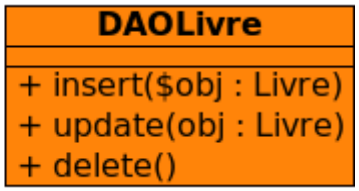

```php
<?php

//Exemple basique en utilisant le ZendFramework

$GoF = new Livre();

$GoF->setTitre('GoF');

$GoF->setAuteur('GoF');

$GoF->setEditeur ('...');

$GoF->save();

?>
```

DAO

- Problématique : Accéder aux données sans connaître la logique de stockage ou de représentation.

- Solution : Un objet qui encapsule la logique de récupération / mise à jour de données.

DAO, diagramme




```
//Exemple basique en utilisant le framework Copix

foreach (CopixDaoFactory::create ('Livre')-
    >findAll () as $record){

   echo $record->titre;

}
```

Lazy Load

- Problématique : Ne pas surcharger l'application de traitements pour la récupération de données qui ne sont pas toujours nécessaires

- Solution : Un objet qui ne contient pas l'ensemble des données mais qui sait comment les récupérer.

Quizz

- Dans quel exemple avons nous vu du lazy loading ?
- Quelles sont les méthodes / interfaces PHP permettant de mettre en oeuvre du LazyLoading de façon transparente pour le code client ?

Front controller

- Problématique : Pouvoir réaliser des opérations diverses relatives à l'ensemble des pages d'un site avant / après leur traitement.

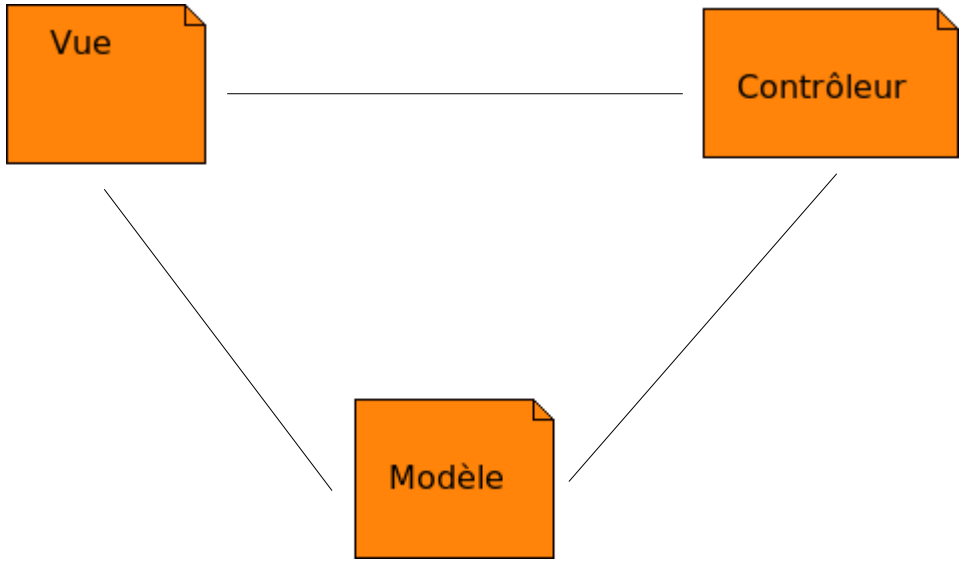
- Solution : Un objet qui intercepte l'ensemble des requêtes de l'application, objet pouvant être décoré.

MVC

- Problématique : Comment organiser les interactions utilisateurs dans l'application ?

- Solution : Définition de 3 rôles : Modèle, Vue, Contrôleur

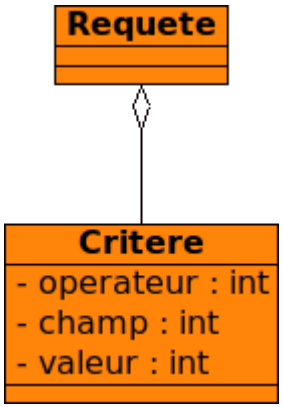
MVC, Schéma



Query Object

- Problématique : Pouvoir manipuler les données de l'application depuis sa représentation objet.
- Solution : Un objet qui représente une requête (modèle interprète)

Query Object




```php
<?php
//Exemple avec ezComponents
require 'ezc-setup.php';
$db = ezcDbFactory::create( 'mysql://root@localhost/geolocation' );
$sq = $db->createSelectQuery();
$stmt = $sq->select( 'name', 'country', 'lat', 'lon' )
    ->from( 'city' )
    ->where(
        $sq->expr->like(
            'normalized_name', $sq->bindValue( 'sofia%' )
        )
    )
    ->orderBy( 'country', 'name' )
    ->prepare();
$stmt->execute();
foreach ( $stmt as $entry )
{
    list( $name, $country, $lat, $lon ) = $entry;
    printf( '%s, %s is @ %.2f%s %.2f%s<br/>',
        $name, $country,
        abs( $lat ), ($lat > 0 ? "N" :"S" ),
        abs( $lon ), ( $lon > 0 ? "E" : "W" ) );
}
?>
```

Au dela du GoF 2

- Plugin
- Registry
- Strategy (GoF)
- Template View
- 2 step View

Plugin

- Problématique : Comment embarquer des fonctionnalités optionnelles sans alourdir l'application générale ?

- Solution : Lier les objets au moment de la configuration et non au moment de l'écriture du code.

Registre

- Problématique : Comment mettre des données à disposition sans devoir transmettre une multitude de paramètres ou sans passer par des variables globales ?

- Solution : Utiliser un objet connu de tous capable de conserver l'état d'autres objets

Registre

Quizz

- Quelles solutions sont à notre disposition pour que le registre soit accessible de partout dans l'application ?

Registre, Exercice

- Créer un registre qui, une fois configuré, sera capable de vérifier la validité des données que l'on lui assigne.
- Ce registre devra être manipulé comme un tableau de données.
- Ce registre devra être parcourable dans un foreach
- Il doit être possible de connaître, grâce a la fonction php count, le nombre d'éléments contenus dans le registre

Registre, Exercice, Validateur (2)

```
interface IValidator {
   public function __construct ($pOptions = array ());
   public function validate ($pValue);
   public function assertValidate ($pValue);
}
abstract class AbstractValidator implements IValidator {
   protected $_options;
   public function __construct ($pOptions = array ()) {
      $this->_options = $pOptions;
   }
   public function assertValidate ($pValue){
      If (($result = $this->validate ($pValue)) !== true){
         throw new Exception ($result);
      }
   }
}
```

Registre, Exercice, Interface, Restrictions (3)

```
interface IRegistry extends ArrayAccess, Countable, Iterator {
   public function set ($pName, $pValue);
   publc function get ($pName);
   public function setConstraints ();
}

class Registry implements IRegistry {
   protected $_data = array ();
   private $_restrictions = array ();
   public function setConstraints (Iterator $restrictions) {
      foreach ($restrictions as $property=>$validator){
         if ($validator instanceof IValidator){
            $this->_restrictions[$property] = $validator;
         }else{
            throw new Exception ('Les contraintes doivent être exprimées avec des
Validateurs');
         }
      }
   }
```

Registre, Exercice, get / set (4)

```
public function get ($pName){
   If ($this->exists ($pName)){
      return $this->_data[$pName];
   }
   return null;
}

public function set ($pName, $pValue){
   if (array_key_exists ($pName, $this->_restrictions)){
      if (($valid = $this->_restrictions[$pName]->validate ($pValue)) !== true){
         throw new Exception ('La donnée '.$pValue.' est invalide : '.$valid);
      }
   }
   $this->_data[$pName] = $pValue;
}
```

Registre, Exercice, ArrayAccess (5)

```
public function offsetGet ($pOffset){
   return $this->get ($pOffset);
}

public function offsetSet ($pName, $pValue){
   return $this->set ($pName, $pValue);
}

public function offsetExists ($pOffset){
   return array_key_exists ($pOffset, $this->_data);
}

public function offsetUnset ($pOffset){
   if ($this->offsetExists ($pOffset)){
      unset ($this->_data[$pOffset]);
   }
}
```

Registre, Exercice, Iterator (6)

```
private $_iteratorPosition = 0;

public function current (){
   return $this->get ($this->key ());
}
public function key (){
   $keys = $this->index ();
   return isset ($keys[$this->_iteratorPosition]) ? $keys[$this->_iteratorPosition] : false;
}
public function valid (){
   return (($key = $this->key ()) !== false) && $this->offsetExists ($key);
}
public function next (){
   $this->_iteratorPosition++;
}
public function rewind (){
   $this->_iteratorPosition = 0;
}
```

Registre, Exercice, Countable (7)

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

Registre, Exercice, Factory (8)

```
class RegistryFactory {
   static $_configuration = false;
   public static function createConfiguration (){
      If (self::$_configuration === false){
         self::$_configuration = new Registry ();
         self::$_configuration->setConstraints (new Registry (array ( 'session_name'=>new
StringValidator (),
    'layout'=>new FileExistsValidator ())));
      }
      return self::$_configuration;
   }
}

//Utilisation
$configuration = RegistryFactory::createConfiguration ();
$configuration['session_name'] = array ();//lance une exception !
```

Template View

- Problématique : Comment obtenir une vue HTML avec les données de l'application ?

- Solution : Utiliser des fichiers intermédiaires qui contiennent la structure de la page ainsi que la logique de présentation.

Template View, Exemple

```
interface IView {
   public function __construct ($pFileId);
   public function display (IData $pData);
   public function render (IData $pData);
}
```

Template View, Exemple (2)

```
class View implements IView {
  private $_file;

  public function __construct ($pFileId){
    $fileValidator = new FileExistsValidator ();
    $fileValidator->assertValidate ($pFileId);
    $this->_file = $pFileId;
  }
  public function display (IData $data){
    echo $this->render ($data);
  }
  public function render (IData $data){
    ob_start ();
    include ($this->_file);
    $renderedView = ob_get_clean ();
    return $renderedView;
  }
}
```

Template View, Exemple (3)

```
if ($lastOnly){
   $listNouvelles = _dao ('nouvelles')->findLast ();
}else{
   $listeNouvelles = _dao ('nouvelles')->findAll ();
}
$view = new View ($mode === 'XML' ?
'nouvelles.rss' : 'nouvelles.html');
$view->display ($listeNouvelles);
```

Retour au GoF : Strategy (comportement)

- Problématique : La façon de traiter une donnée dépend du contexte, et il existe plusieurs algorithmes pour arriver au résultat. Ces algorithme ne sont connus qu'au dernier moment.
- Solution : Déléguer le traitement à un objet tiers qui implémente l'algorithme à utiliser.

Strategy, Exercice

- Les contrôlleurs de notre application sont succeptibles de retourner des réponses de type différent. Chaque type de réponse doit être interprété différemment. Comment s'y prendre ?

Strategy, Exercice (2)

```
interface IResponse {}
abstract class Response implements IResponse {}

class ResponseRedirect extends Response {
   private $_redirect = null;
   public function setRedirect ($pUrl){$this->_redirect = $pUrl;}
   public function get (){return $this->_redirect;}
}

class ResponseDisplay extends Response {
   private $_view;
   private $_data;
   public function setData ($pData){$this->_data = $pData;}
   public function getData (){return $this->_data;}
   public function setView ($pView){$this->_view = $pView;}
   public function getView (){return $this->_view; }
}
```

Strategy, Exercice (3)

```
interface IResponseStrategy {
   public function handle (IResponse $response);
}

class ResponseDisplayStrategy implements IResponseStrategy {
   public function handle (ResponseDisplay $response){
      $response->getView ()->display ($response->getData ());
   }
}

class ResponseRedirectStrategy implements IResponseStrategy {
   public function handle (IResponse $response){
      header ('location: '.$response->get ());
   }
}
```

Strategy, Exercice (4)

```
protected function getResponseStrategy (IResponse $response){
   if ($response instanceof ResponseDisplay){
      return new ResponseDisplayStrategy ();
   }elseif ($response instanceof ResponseRedirect){
      return new ResponseRedirectStrategy ();
   }else{
      return new ResponseNotFoundStrategy ();
   }
}

protected function sendUser ($strategy, $response){
   /* */
   $strategy->handle ($response);
   /* */
}

//code client
$response = $controller->action ();
$strategy = $this->getResponseStrategy ($response);
$this->sendUser ($strategy, $response);
```

Strategy, Quizz

Quel pattern aurait put être mis a contribution pour trouver la stratégie capable de répondre a notre réponse ?

Quel pattern pourait être mis a contribution si nous avions des modes de fonctionnement de l'application différent, rendant le traitement des réponses différents ? (par exemple un navigateur qui ne supporterait pas les messages en en tête et dans lequel nous devrions adresser le message de redirection avec un meta redirect)

2 Step View

- Problématique : Lorsque toute les pages d'un site disposent de la même présentation, faire en sorte qu'une modification d'apparence générale soit localisée en un endroit.
- Solution : Effectuer le calcul de la sortie finale en 2 passes.

2 Step View, Exemple

```
class Response2StepViewStrategy {
   public function handle (ResponseDisplay
$response){
      $view1Result = $response->getView ()-
>render ($response->getData ());
      $mainView = new View
(FrontController::getInstance ()-
>getConfiguration ()->get ('layout'));
      $mainView->display (new Data (array
('content'=>$view1Result)));
   }
}
```

Critiques

- Apporte des solutions peu efficientes
- Standardisation des « best practices ».
- En pratique, duplication de code inutile là ou une solution spécifique bien pensée aurait put être meilleure qu'un pattern « faisant l'affaire ».

Anti Patterns

- Catalogue des choses à ne pas faire

(a) **Copier/Coller**

(b) **Magic numbers**

(c) **Singletonite**

(d) **Code en dur**

(e) **Masquage d'erreur**

(f) **Coulée de lave (maintenance d'un mauvais code dans le temps)**

(g) **Utilisation des types au lieu d'interfaces**

(h) **Confiance aveugle**

(i) **Sur-Abstraction**

(j) **« Super objet » (trop de responsabilités)**

(k) **...**

Références

- Design Patterns. Elements of Reusable Object-Oriented Software.
- Design patterns Tête la première
- Pattern of Enterprise Application Architecture
- php|architect's Guide to PHP Design Pattern
- http://www.phppatterns.com/
- Wikipedia

Un design pattern pour résoudre un problème et non pas l'inverse.

Un design pattern n'est pas une solution figée

Questions ?