



DESIGN PATTERNS & PHP

Gérald Croës - gerald@copix.org

<http://gcroes.com/forumphp2006/dp.odt>

Introduction

- ✿ Pourquoi ?

- ✿ Objectifs de la présentation.

Définition

- ✿ En français - Modèles de conception et aussi « Motifs de conception », « Patron de conception ».
- ✿ Solutions à des problèmes classiques de conception.
- ✿ « L'algorithmie de la conception »

Définition (2)

Un design pattern nomme, motive et explique un design général qui répond à un problème fréquent dans un système orienté objet.

Il décrit le problème, la solution, quand appliquer la solution et ses conséquences.

Il donne aussi des conseils d'implémentation et des exemples.

[GoF]

Définition (3)

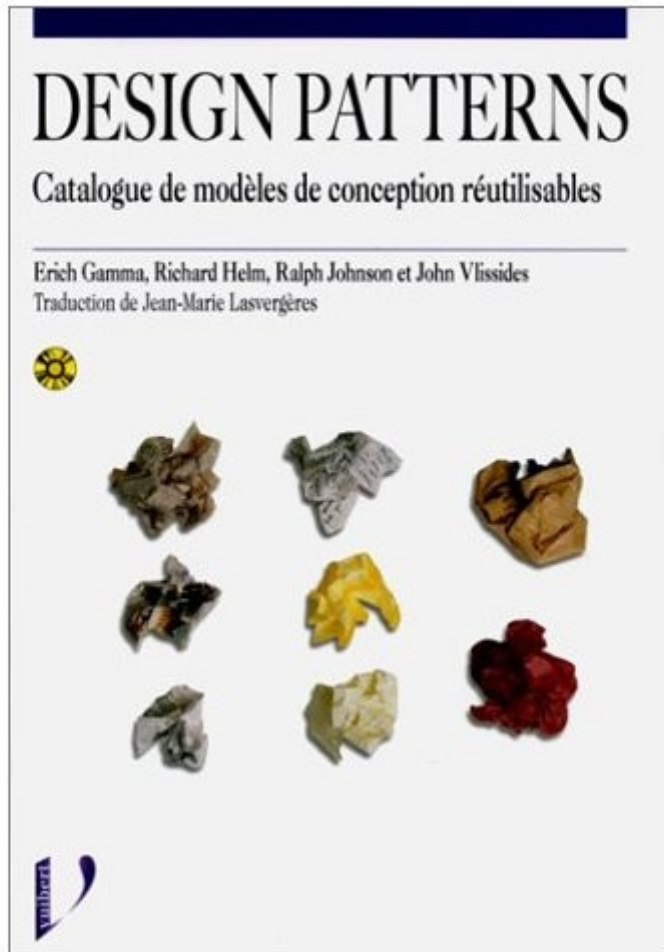
- ⚙ Solutions connues pour résoudre des problèmes connus
- ⚙ Indépendant de tout langage informatique
- ⚙ « Bibliothèque conceptuelle »

Un peu d'histoire (1)

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

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

Un peu d'histoire (2)



⚙️ OOPSLA 1987

Kent Beck Ward Cunningham

⚙️ GoF « Gang of Four »

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

Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides

Pourquoi les connaître ?

« Chaque patron décrit un problème qui se manifeste constamment dans notre environnement et décrit le cœur de la solution à ce problème, d'une façon telle que l'on puisse réutiliser cette solution des millions de fois, sans jamais le faire deux fois de la même manière »

Christopher Alexander.

Pourquoi les connaître ? (2)

- ✿ Facilite la communication entre développeurs.
- ✿ Décompose un problème complexe en plusieurs problèmes simples.
- ✿ Facilite la conception. (Améliore sa compréhension, auto-documentation des concepts)

Pourquoi les utiliser ?

- ✿ Pour la même raison que l'on utilise une fonction, une classe, un objet, une bibliothèque ou un framework.
- ✿ Ne pas réinventer la roue.
- ✿ Bénéficier du savoir faire d'experts dans des contextes éprouvés. (fiables, robustes & connus)

Les familles

Gof

- ✚ Créateurs
- ✚ Structuraux
- ✚ Comportementaux

J2EE

- ✚ Présentation
- ✚ Business
- ✚ Intégration

Autres... chacun est libre de formaliser ses propres patterns

Fiche d'identité

- ⚙ Nom
- ⚙ Description du problème
- ⚙ Description de la solution
 - (a) **Quels sont les éléments qui participent à la solution ?**
 - (b) **Quels sont leur rôle**
 - (c) **Comment interagissent t ils entre eux ?**
- ⚙ Exemples
- ⚙ Critiques

« Par convention, 3 utilisations couvertes de succès »

Les 23 Patterns « GoF »

Créateurs

- (a) **Factory method (5)**
- (b) **Abstract factory (3)**
- (c) **Builder (3)**
- (d) **Prototype (3)**
- (e) **Singleton (5)**

Structuraux

- (a) **Adapter (4)**
- (b) **Bridge (2)**
- (c) **Composite (2)**
- (d) **Decorator (3)**
- (e) **Façade (5)**
- (f) **Flyweight (1)**
- (g) **Proxy (2)**

Comportementaux

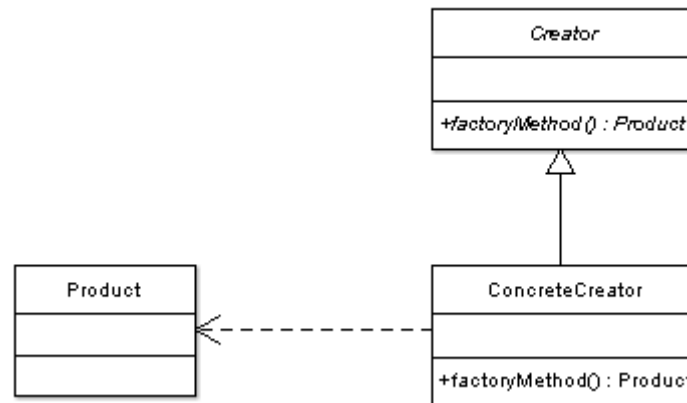
- (a) **Interpreter (3)**
- (b) **Template method (3)**
- (c) **Chain of responsibility (3)**
- (d) **Command (2)**
- (e) **Iterator (4)**
- (f) **Mediator (1)**
- (g) **Memento (2)**
- (h) **Observer (4)**
- (i) **State (4)**
- (j) **Strategy (2)**
- (k) **Visitor (1)**

[GoF] Créateurs

- ✿ Abstraction du processus de création.
- ✿ Encapsulation du choix de création.
- ✿ On ne sait pas à l'avance ce qui sera créée ou comment cela sera créé.

[GoF] Factory method / Factory / Fabrique

- On ne connaît pas encore l'objet qu'il nous faut à l'écriture du code.

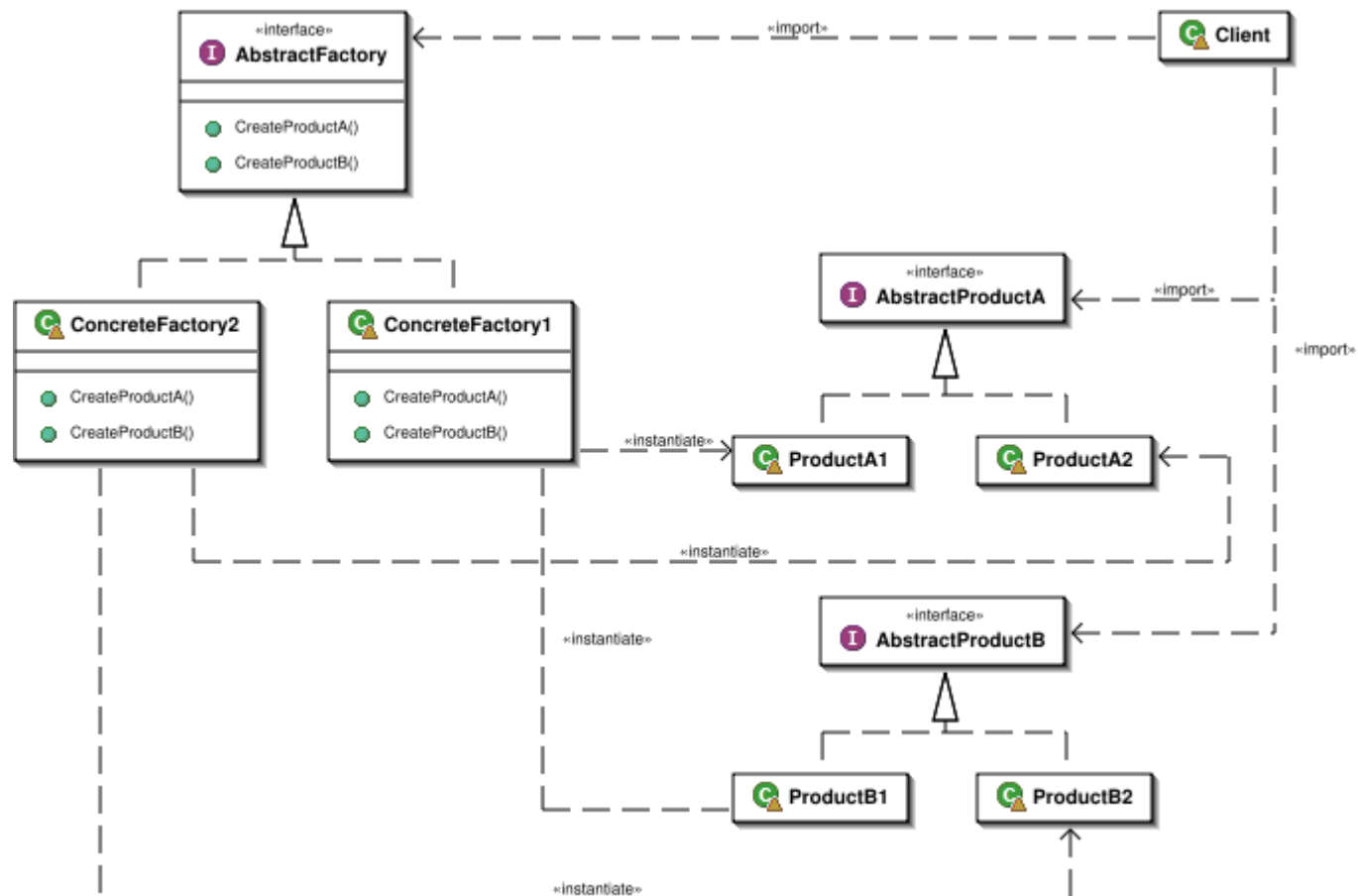


[GoF] Factory, exemple simple

```
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);  
        }  
    }  
}
```

[GoF] Abstract Factory / Fabrique abstraite

🧩 Création d'une famille d'objet



[GoF] Abstract Factory, exemple

```
abstract class AbstractDocumentFactory {  
    abstract public function getDevis ();  
    abstract public function getAdhesion ();  
}
```

```
class PdfDocumentFactory {}
```

```
class HTMLDocumentFactory {}
```

```
abstract class Devis ();
```

```
abstract class Adhesion ();
```

[GoF] Singleton

- ✿ Avoir une seule et unique instance d'un objet

Singleton
-instance : Singleton
-Singleton() +Instance() : Singleton

[GoF] Singleton, exemple

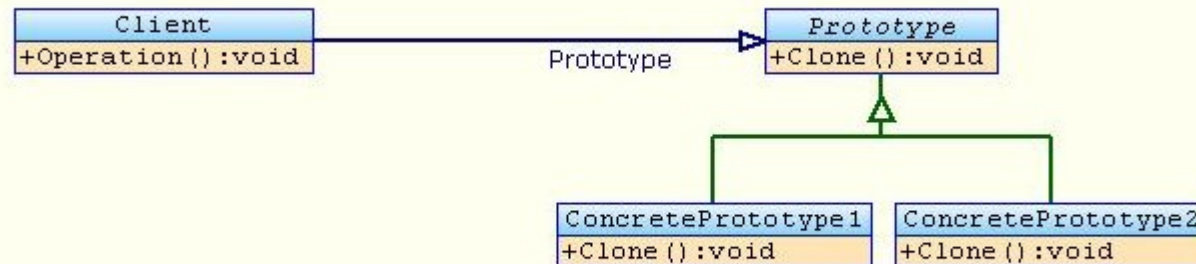
```
class ApplicationConfiguration {  
    private static $_instance = false;  
    public static function getInstance () {  
        if (self::$_instance === false) {  
            self::$_instance = new ApplicationConfiguration ();  
        }  
        return self::$_instance;  
    }  
    private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

[GoF] Singleton / Classe statique ?

- ✿ Une instance à manipuler.
- ✿ Conserve un contexte.
- ✿ Peut être passé en paramètre à n'importe quelle méthode.

[GoF] Prototype

- ⚙️ __clone () intégré à PHP
- ⚙️ A utiliser lorsque le processus de création ou d'initialisation est coûteux.

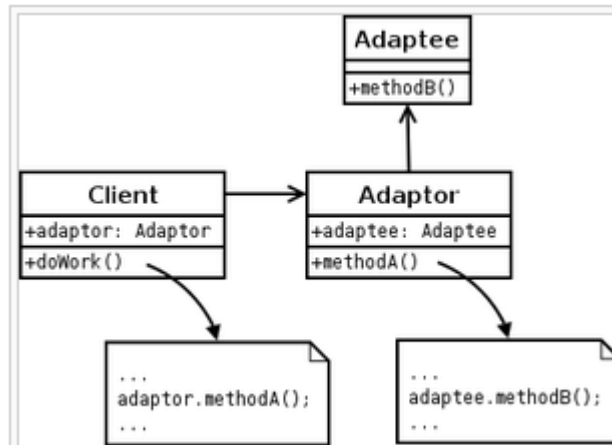


[GoF] Structuraux

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

[GoF] Adapter / Adaptater

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



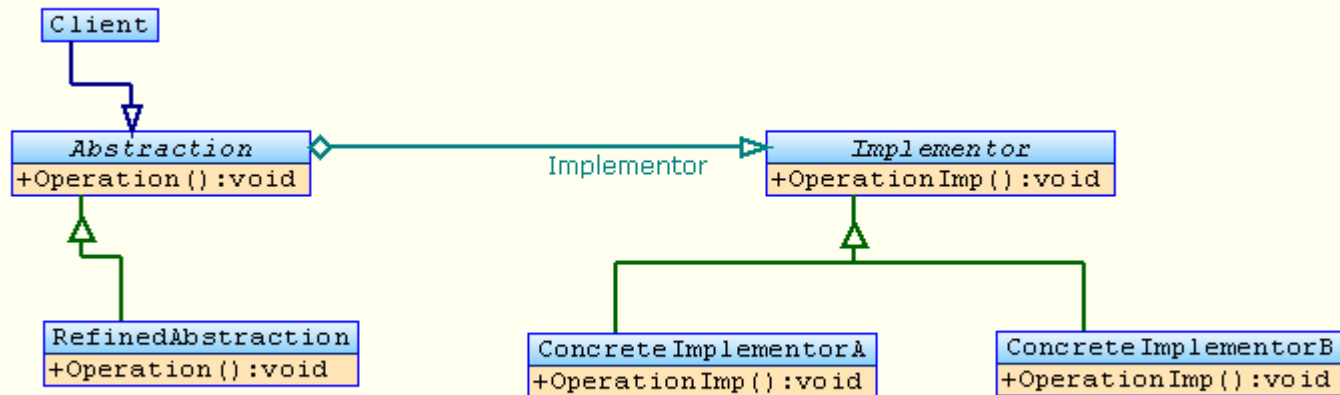
- ⚙️ Moins d'impacts en PHP que dans des langages à fort typage.

[GoF] Adaptateur, exemple

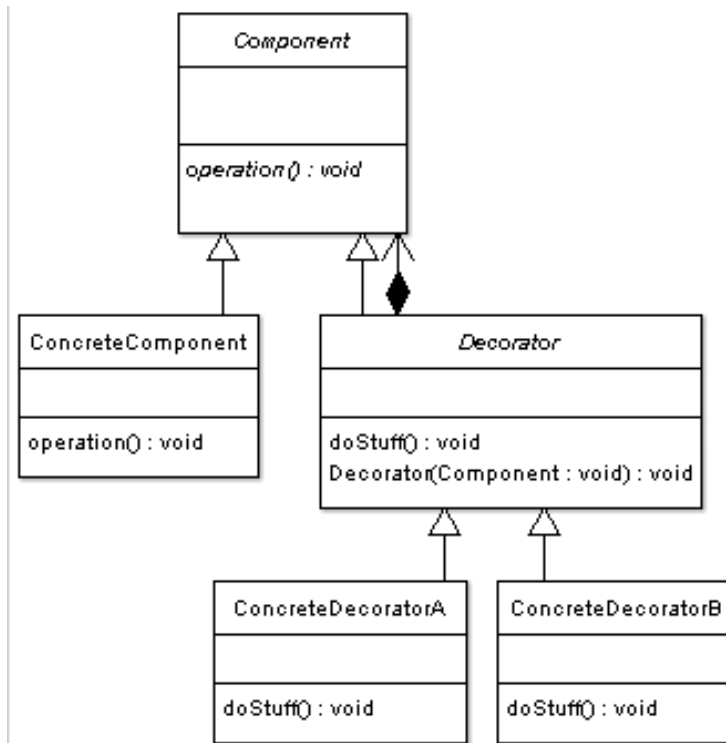
```
class AuthAutreBibliothèque {  
    public function getLevel (){//calcul le niveau de l'utilisateur}  
}  
  
class AuthAdapter implements Auth {  
    private $_adapted = null;  
    function __construct ($login){  
        $this->_adapted = $login;  
    }  
    function isConnected (){  
        return $this->_adapted->getLevel () >  
            AuthAutreApplication::ANONYMOUS;  
    }  
}
```

[GoF] Bridge / Pont

- ⚙️ Découpler l'abstraction de son implémentation.



[GoF] Decorator / Décorateur



- ✚ Rajouter des fonctionnalités à des composants existants
- ✚ Héritage impossible
- ✚ Une arborescence existante

[GoF] Decorator, exemple

class decorated {}

abstract class Decorator {

private \$_decorated;

function __construct (\$decorated){

 \$this->_decorated = \$decorated;

 }

function doStuff (){

return \$this->_decorated->doStuff ();

 }

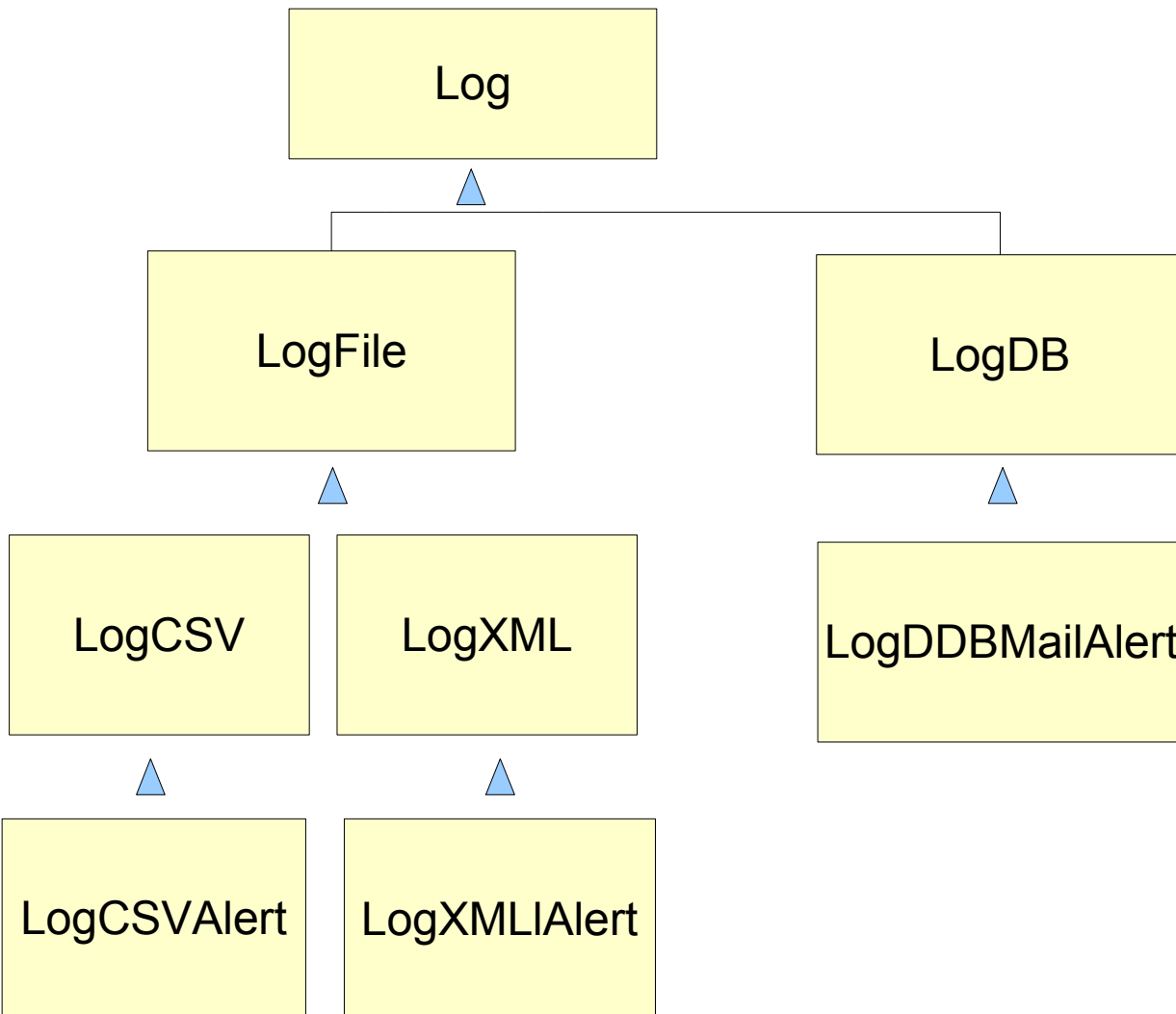
}

[GoF] Decorator exemple (2)

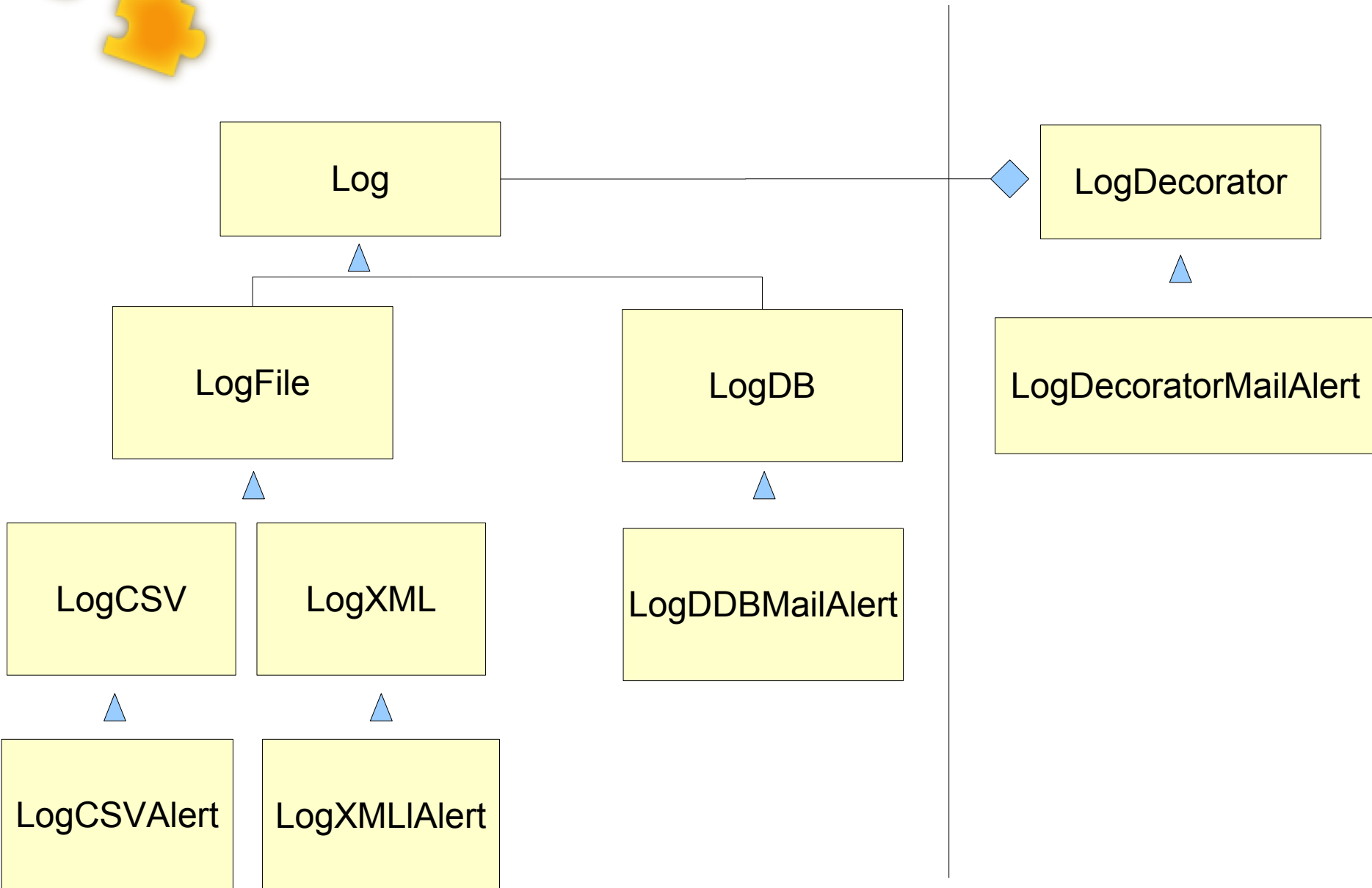
```
class Decorator1 extends Decorator {  
    function doOtherStuff () {  
        //autre traitement  
    }  
}  
  
class Decorator1 extends Decorator {  
    function yaS () {  
        //yet another stuff  
    }  
}
```

[GoF] Decorator / Héritage

Une arborescence
qui se complexifie

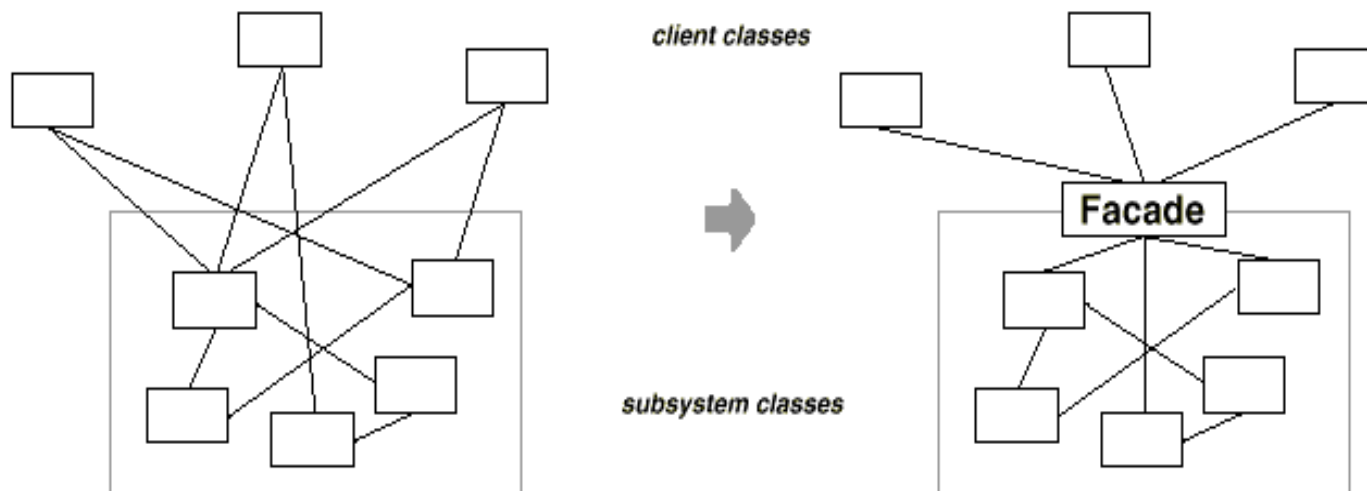


[GoF] Decorator / Héritage



[GoF] Façade

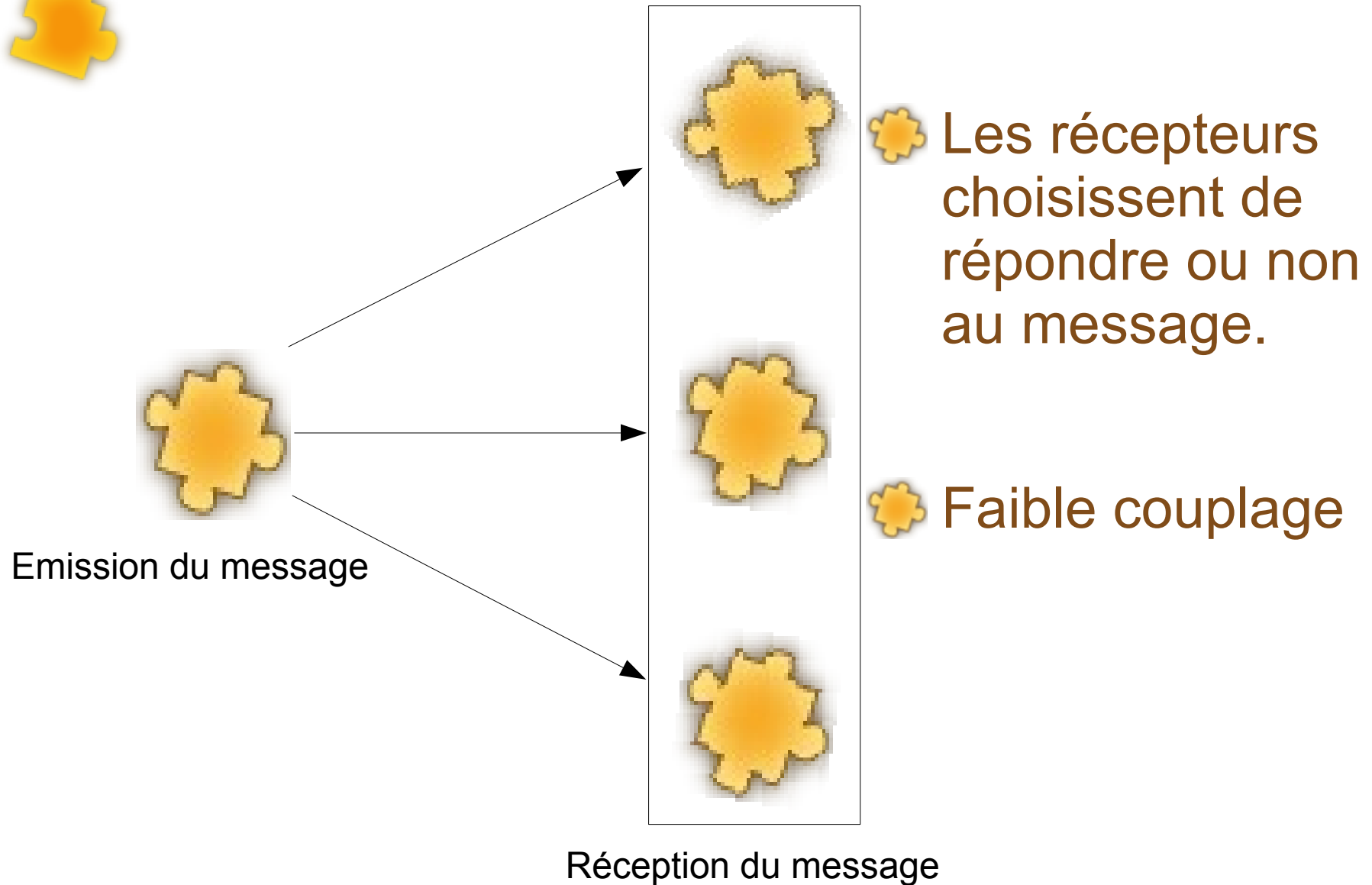
- ⚙️ Masquer la complexité d'une API.
- ⚙️ Créer une API simple pour un ensemble complexe.



[GoF] Comportementaux

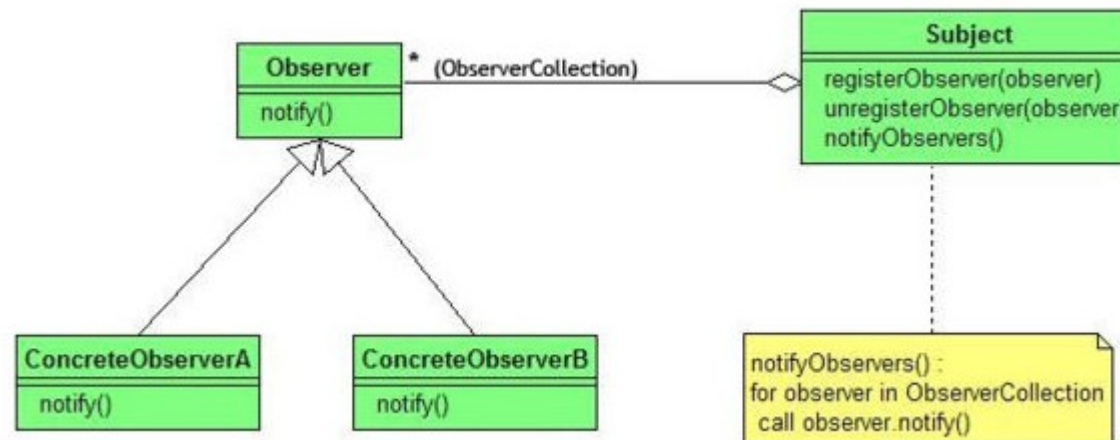
- Mode de communication entre les objets

[GoF] Chaine de responsabilité



[GoF] Observer / Listener

- ⚙ Réagir à des changements pouvant survenir.
- ⚙ « Poorman's events manager »



[GoF] Iterator / Itérateur

- ✚ Directement déclaré dans PHP
- ✚ Utilisable dans les foreach
- ✚ Idéal pour les sources de données

[GoF] Résumé

Factory

Interface de création d'objets

Abstract Factory

Fabrique de création pour des familles d'objets

Singleton

Instances uniques

Builder

Création d'objets complexes

Prototype

Création par copie

[GoF] Résumé (2)

Adapter

Adapte l'interface d'une classe à un autre objet.

Decorator

Ajout de fonctionnalités à un objet

Proxy

**Une même interface pour une invocation juste à temps,
invocation à distance.**

Flyweight

Objets réduits en grand nombre

[GoF] Résumé (3)

Bridge

Abstraction dé耦lée de l'implémentation

Composite

Arbres d'objets simples

Façade

API simplifiée pour un système complexe

[GoF] Résumé (4)

Chain of responsibility

Un message est passé au travers d'une chaine d'objet.

Command

Interpreter

Analyse syntaxique

Iterator

Parcours d'une liste d'objet

Mediator

Logique de communication d'objets tiers

[GoF] Résumé (5)

Memento

Stockage de l'état d'un objet.

Observer

Notification d'événements sur une classe

State

Etat d'un objet

Strategy

Implémentation d'algorithmes

Template method

Implémentation de comportements communs

[J2EE] DAO

- ⚙ Un composant pour lire les données.
- ⚙ Encapsulation de la logique de lecture / écriture.
- ⚙ Ne pas confondre avec le mapping objet / relationnel.

[J2EE] Front Controller

- ✿ Dans un site web, vous devez souvent procéder à de nombreuses tâches lors du traitement d'une requête. Ces tâches relèvent de la sécurité, de l'authentification, ou du rendu particulier à proposer à certains utilisateurs.
- ✿ Le front controller récupère l'ensemble des requêtes et implémente les comportements communs.

[Martin Fowler]

Critiques

- ⚙ Apporte des solutions peu efficaces
- ⚙ Standardisation des « best practices ».
- ⚙ En pratique, duplication de code inutile la 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) **Singletonitis**
- (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
- ⚙ <http://www.phppatterns.com/>
- ⚙ Wikipedia



Questions ?