

# Cours d'Environnement de Développement

Arnaud Sangnier

## Partie 4

# La conception orientée-objet

- Un art difficile ...
  - Penser en terme d'objets plutôt qu'en fonction
- Une conception **réutilisable, adaptable, performante** est extrêmement difficile
- Quelle est la différence entre un novice et un concepteur expérimenté ?
  - Le novice hésité beaucoup entre différentes variantes
  - L'expert dispose d'outils pour trouver tout de suite la bonne solution
- Comme pour toute approche de programmation, le secret est **l'expérience**

# Expertise en conception orientée-objet

## Les enjeux

- Limiter l'impact d'une révision fonctionnelle ou technique
- Réutiliser efficacement certaines classes
- Identifier les éléments structurants :
  - Problème général  $\mapsto$  solution particulière
- Capitaliser les savoir-faire

## Principes appliqués

- Distribuer les responsabilités des objets
- Distinguer le **quoi** du **comment**

# Expertise en conception orientée-objet

## Comment profiter de l'expérience

- Ne pas réinventer la roue
- Réutiliser systématiquement des solutions qui ont fait leurs preuves
- Proposer une conception modulaire, élégante et adaptable

**Répétition de certains profils de classes ou collaboration d'objets**

⇒ Utilisation de **design patterns**  
(également appelés modèles de conception ou patrons de conception)

# Origines des design patterns

- Christopher Alexander, 1977 (architecture et urbanisme)  
*"Each pattern is three-part rule, which expresses a relation between a certain context, a problem and a solution"*
- Un exemple **historique**, 1988 :
  - Le modèle MVC de Smalltalk (Model-View-Controller)
- Le catalogue de la **bande des quatre** (*GoF design patterns*) :
  - Design patterns - Elements of reusable object oriented software  
Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides  
1995

# Bibliographie

- **DESIGN PATTERNS - Catalogues de modèles de conception réutilisables**  
Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides  
Éditions Vuibert
- **Les Design Patterns en Java - Les 23 modèles de conception fondamentaux**  
Steven John Metsker et William C. Wake  
Éditions Campus Press

# Définitions

# Patron de conception

## Définition

Un patron de conception décrit une structure commune et répétitive de composants en interaction qui résout un problème récurrent de conception dans un contexte particulier

Un patron de conception :

- Résout un problème
- Est un concept éprouvé
- Propose une solution pas forcément évidente
- Décrit une collaboration entre objets



# Formalisme

La description d'un patron de conception suit un formalisme fixe :

- ① Nom
  - Identification d'un concept
- ② Description du problème à résoudre
  - Situations dans lesquelles le patron s'implique
- ③ Description de la solution
  - Éléments du modèle de conception
- ④ Conséquences
  - Effets de l'application du modèle sur la conception

# Le catalogue de la bande des quatre des patrons de conception

	<b>Créateur</b>	<b>Structurel</b>	<b>Comportemental</b>
<b>Classe</b>	Fabrication	Adaptateur (classe)	Interprète Patron de méthode
<b>Objet</b>	Fabrique Absraite Monteur Prototype Singleton	Adaptateur (objet) Pont Composite Décorateur Façade Poids mouche Procuration	Interprète Commande Itérateur Médiateur Memento Observateur État Stratégie Visiteur

# Domaine des patrons

- **Classe**

- Modèles qui traitent des relations entre les classes et leurs sous-classes
- Ces relations sont statiques et établies préalablement à l'exécution

- **Objet**

- Modèles qui traitent des relations entre objets
- Ces relation peuvent évoluer au cours de l'exécution et sont donc plutôt dynamiques

# Rôle des patrons

- **Créateur**
  - Modèles concernant le processus de création d'objets
- **Structurel**
  - Modèles gérant la composition de classes ou d'objets
- **Comportemental**
  - Modèles spécifiant les façon d'interagir de classes et d'objets et de se répartir les responsabilités

## Boîte à outils

- Une application est fréquemment composée de classes issues d'une ou de plusieurs bibliothèques de classes prédéfinies appelées boîtes à outils
- Par exemple :
  - Les classes pour manipuler les listes, piles, tables associative, etc.
  - Les classes pour manipuler les entrées/sorties
- Les boîtes à outils fournissent des fonctionnalité qui facilitent la tâche des applications
- Les boîtes à outils sont les faire valoir de la réutilisation de code
- La conception de boîte à outils est difficile :
  - Elle doit pouvoir être utilisé dans de nombreuses applications
  - Le concepteur ne connaît pas ses applications

# Framework

## Définition

Un framework est un ensemble de classes qui coopèrent et permettent des conceptions réutilisables dans des catégories spécifiques de logiciels.

- Exemples :
  - Framework pour la fabrication d'éditeurs graphiques
  - Framework pour l'aide à la conception de compilateurs
  - Framework pour l'aide à la conception d'applications de modélisations financières
- Un framework impose une certaine architecture à une application
- Lorsqu'on utilise un framework, on réutilise le tronc principal et on réécrit le code que celui-ci appelle

# Patron de conception / Framework / Boite à outils

## Les patrons de conception ne sont ni des frameworks ni des boîtes à outils

- Les patrons de conception ne sont pas du code utilisables directement
- Les patrons de conception doivent être implémentés
- Une boîte à outils ou un framework contiennent souvent plusieurs patrons de conception
- Les modèles de conception sont moins spécialisé que les framework
- Comme les boîte à outils, les patrons de conception de la bande des quatre peuvent être utilisés dans à peu près n'importe quel type d'application

# Les patrons de création



# Les patrons de création

**Ils définissent un mécanisme de création de classes ou d'objets**

- Singleton
- Méthode de Fabrication
- Fabrication abstraite
- Monteur
- Prototype

# Le patron Singleton

L'objectif du patron Singleton est de garantir qu'une classe ne possède qu'une seule instance et de fournir un point d'accès global à celle-ci.

# Le patron Méthode de Fabrication

- En anglais : *Factory method*

L'objectif du patron Méthode de Fabrication est de laisser un autre développeur définir l'interface permettant de créer un objet, tout en gardant un contrôle sur le choix de la classe à instancier.

- Permettre à une classe de déléguer l'instanciation d'objets à des sous-classes
- Définir une interface pour la création d'un objet, mais en laissant à des sous-classes le choix des classes à instancier

# Le patron Fabrique Abstraite

- En anglais : *Abstract Factory*
- Parfois aussi appelé *Kit*

L'objectif du patron Fabrique Abstraite est de permettre la création de familles d'objets ayant un lien ou interdépendants.

- Un système doit être indépendant de la façon dont ses produits sont créés, combinés et représentés
- Un système est constitué d'une famille de produits, parmi plusieurs
- Ce patron sert à renforcer le caractère de communauté d'une famille de produits conçus pour être utilisés ensemble
- On isole l'endroit de création des objets, de leur utilisation

# Le patron Monteur

- En anglais : *Builder*

L'objectif du patron Monteur est de déplacer la logique de construction d'un objet en dehors de la classe à instancier

- L'algorithme de création d'un objet complexe doit être indépendant de la manière dont ses parties sont agencées
- Le processus de construction doit autoriser des représentations différentes
- Le patron Monteur est utilisé pour construire un objet pas à pas

# Le patron Prototype

L'objectif du patron Prototype est de fournir de nouveaux objets par la copie d'un exemple plutôt que de produire de nouvelles instances non initialisées d'une classe.

- Crée de nouveaux objets en copiant un exemple
- Par rapport, à l'appel d'un constructeur la copie inclut un certain état de l'objet original

# Récapitulatif sur les patrons de création

- **Classes créatrices** : La création des objets est partiellement externalisée, en distinguant :
  - ce qui est de la responsabilité des objets à créer
  - ce qui dépend du contexte
- Deux façons de paramétrer un système par les classes d'objet :
  - 1 **Dérivation** : Patron Methode de Fabrication
  - 2 **Composition** : Patron Fabrique Abstraite, Monteur et Prototype

# Les patrons structuraux



# Les patrons structuraux

**Ils définissent la façon de composer des classes et des objets pour réaliser des structures complexes**

- Adaptateur
- Pont
- Composite
- Décorateur
- Façade
- Poids-mouche
- Procuration

# Le patron Adaptateur

- En anglais : *Adapter*

L'objectif du patron Adaptateur est de fournir l'interface qu'un client attend en utilisant les services d'une classe dont l'interface est différente.

# Le patron Pont

- En anglais : *Bridge*
- Parfois aussi appelé *Driver*

L'objectif du patron Pont est de découpler une abstraction de l'implémentation de ses opérations abstraites, permettant ainsi à l'abstraction et son implémentation de varier indépendamment.

- Pas de lien définitif entre abstraction et implémentation
- Abstractions et implémentations peuvent être dérivées
- Une modification de l'implémentation ne doit pas avoir d'impact sur les classes clientes
- Plusieurs objets peuvent partager la même implé

# Le patron Composite

L'objectif du patron Composite est de permettre aux clients de traiter de façon uniforme des objets individuels et des compositions d'objets.

- Représentation de structures récursives
- Traitement uniforme de tous les objets du composite, qu'ils soient terminaux ou non

# Le patron Décorateur

- En anglais : *Decorator*
- Parfois aussi appelé *Wrapper*

L'objectif du patron Décorateur est de permettre de composer de nouvelles variations d'une opération lors de l'exécution.

- Ajouter dynamiquement des responsabilités à un objet sans modifier sa classe, et donc sans affecter les autres objets
- Permet d'éviter une explosion combinatoire du nombre de sous-classes

# Le patron Façade

- En anglais : *Facade*

L'objectif du patron Façade est de fournir une interface simplifiant l'emploi d'un sous-système.

- Disposer d'une interface simple avec un package
- Relations inter-packages trop nombreuses
- Structuration de packages en niveaux (une façade par niveau)
- Masque au client des composants du sous-système

# Le patron Poids mouche

- En anglais : *Flyweight*

L'objectif du patron Poids mouche est d'utiliser le partage pour supporter efficacement un grand nombre d'objets à forte granularité.

- Les poids mouches sont créés par un objet dédié (fabrique de poids mouche) qui les stocke et peut fournir une référence sur ces objets à la demande des objets client
- L'objet n'est créé que s'il n'existe pas déjà

# Le patron Proxy

- Aussi appelé : Subrogé (*Surrogate* en anglais)

L'objectif du patron Proxy est de contrôler l'accès à un objet en fournissant un intermédiaire pour cet objet.



# Les patrons comportementaux

# Les patrons structuraux

Ils définissent la façon de composer des classes et des objets pour partager les responsabilités entre objets

- Chaîne de responsabilité
- Commande
- Interpréteur
- **Itérateur**
- **Médiateur**
- Memento
- Observateur
- État
- Stratégie
- Patron de méthode
- Visiteur

# Le patron Itérateur

- En anglais : Iterator
- Aussi appelé : Curseur (*Cursor* en anglais)

L'objectif du patron Itérateur est de fournir un moyen d'accéder de façon séquentielle aux éléments d'une collection.

- Accéder aux éléments d'un conteneur sans dévoiler son implémentation
- Permettre plusieurs types de parcours
- Offrir une interface uniforme pour le parcours de différents types de conteneur

# Le patron Observateur

- En anglais : *Observer*
- Aussi appelé : diffusion-souscription

L'objectif du patron Observateur est définir une dépendance du type un-à-plusieurs entre des objets de manière que, lorsqu'un objet change d'état, tous les objets dépendants en soient notifiés afin de pouvoir réagir conformément.

- Une modification d'un objet nécessite la mise à jour d'autres objets
- Un objet doit faire une notification à d'autres sans faire d'hypothèse sur la nature de ceux-ci

# Le patron Visiteur

- En anglais : *Visitor*

L'objectif du patron Visiteur est de permettre de définir une nouvelle opération pour une hiérarchie sans changer ses classes.

- Effectuer plusieurs opérations sans relation entre elles sur les objets d'une structure en évitant de "polluer" les classes avec ces opérations
- En particulier adapté lorsque les classes de la structure changent peu et que l'on doit souvent définir de nouvelles opérations

# Le patron Stratégie

- En anglais : *Strategy*
- Aussi appelé : *policy*

L'objectif du patron Stratégie est d'encapsuler des approches, ou stratégies, alternatives dans des classes distinctes qui implémentent chacune une opération commune.