## Patrons fondateurs

Erich GAMMA, Richard HELM, Ralph JOHNSON et John VLISSIDES (dits le *Gang of Four*) publient en 1994 chez Addison-Wesley "*Design Patterns: Elements of Reusable Object-Oriented Software*" qui présente 23 patrons répartis en 3 familles.

## 3 familles de patrons

- de création
  *(instancier des classes et configurer des objets)*
  Fabrique (abstraite et concrète), Singleton, Monteur, Prototype
- de structure
  *(organiser des classes dans une structure plus large)*
  Adaptateur, Décorateur, Façade, Proxy, Composite…
- de comportement
  *(organiser des objets pour qu'ils collaborent)*
  Itérateur, Stratégie, Visiteur, Observateur …

## Fabrique simple (*factory*)

- Ce patron fournit une interface pour créer des objets sans donner leur classe concrète.
- Une fabrique simple retourne une instance d'une classe parmi plusieurs possibles, en fonction des paramètres fournis. Toutes les classes ont un lien de parenté, et des méthodes communes.
- L'objet retourné est un produit dont l'utilisateur a fait appel à la Fabrique Simple (l'usine) pour l'obtenir. C'est la Fabrique Simple qui est chargée d'instancier et de retourner le Produit Concret attendu. L'utilisateur du produit est donc fortement couplé uniquement à la Fabrique Simple et non à tous les produits qu'il utilise.

## Fabrique (*abstract factory*)

- Ce patron fournit une interface pour créer des objets sans donner leur classe concrète.
- L'utilisateur d'un produit le demande à la fabrique qui le fait instancier par la fabrique simple (l'usine) qui fabrique ce produit.
- La fabrique (l'entreprise) contient toutes les méthodes permettant de manipuler les produits exceptée la méthode creerProduit qui est abstraite. Les fabriques simples (usines) implantent la méthode creerProduit qui instancie et retourne les produits. Chaque fabrique simple peut donc créer des produits dont elle a la responsabilité.
- Tous les produits implantent la même interface afin que les classes utilisant les produits puissent s'y référer sans connaître les types concrets.

## Adaptateur

- Ce patron convertit l'interface d'une classe en une autre interface exploitée par une application.
- Permet d'interconnecter des classes qui sans cela seraient incompatibles. Il est utilisé dans le cas où un programme se sert d'une bibliothèque de classes qui ne correspond plus à l'utilisation qui en est faite, à la suite d'une mise à jour de la bibliothèque dont l'interface a changé. Un objet adaptateur expose alors l'ancienne interface en utilisant les fonctionnalités de la nouvelle.

## Pont

- Ce patron permet de découpler une abstraction de son implantation, de telle manière qu'elles peuvent évoluer indépendamment. Il consiste à diviser une implantation en deux parties : une classe d'abstraction qui définit le problème à résoudre, et une seconde classe qui fournit une implantation.
- Il peut exister plusieurs implantations pour le même problème et la classe d'abstraction comporte une référence à l'implantation choisie, qui peut être changée selon les besoins.
- Le patron pont est fréquemment utilisé pour réaliser des récepteurs d'événements.

## Monteur (*builder*)

- Ce patron sépare le processus de construction d'un objet du résultat obtenu, ce qui permet d'utiliser le même processus pour obtenir différents résultats.
- C'est une alternative au patron fabrique.
- Au lieu d'une méthode pour créer un objet, à laquelle est passée un ensemble de paramètres, la classe fabrique comporte une méthode pour créer un objet - le monteur. Cet objet comporte des propriétés qui peuvent être modifiées et une méthode pour créer l'objet final en tenant compte de toutes les propriétés.
- Ce patron est particulièrement utile quand il y a de nombreux paramètres de création, presque tous optionnels.

## Chaîne de responsabilité

- Le patron Chaîne de responsabilité vise à découpler l'émission d'une requête de la réception et du traitement de cette dernière en permettant à plusieurs objets de la traiter successivement.
- Dans ce patron chaque objet comporte un lien vers l'objet suivant, qui est du même type. Plusieurs objets sont ainsi attachés et forment une chaîne. Lorsqu'une demande est faite au premier objet de la chaîne, celui-ci tente de la traiter, et s'il ne peut pas, il fait appel à l'objet suivant, et ainsi de suite.

## Commande

- Ce patron encapsule une demande dans un objet, permettant de paramétrer, mettre en file d'attente, journaliser et annuler des demandes.
- Dans ce patron, un objet commande correspond à une opération à effectuer. L'interface de cet objet comporte une méthode execute. Pour chaque opération, l'application va créer un objet différent qui implante cette interface - qui spécifie une méthode execute.
- L'opération est lancée lorsque la méthode execute est utilisée.
- Ce patron est notamment utilisé pour les barres d'outils.

## Composite

- Le patron composite permet de créer des objets complexes en reliant différents objets selon une structure en arbre. Il permet de composer une hiérarchie d'objets, et de manipuler de la même manière un élément unique, une branche, ou l'ensemble de l'arbre.
- Ce patron impose que les différents objets aient une même interface, ce qui rend uniformes les manipulations de la structure.

## Décorateur

- Le patron décorateur permet d'attacher dynamiquement des responsabilités à un objet.
- Alternative à l'héritage, ce patron est inspiré des poupées russes. Un objet peut être caché à l'intérieur d'un autre objet décorateur qui lui ajoute des fonctionnalités. L'ensemble peut être décoré avec un autre objet qui lui ajoute des fonctionnalités et ainsi de suite.
- Cette technique nécessite que l'objet décoré et ses décorateurs implantent la même interface, qui est typiquement définie par une classe abstraite.

## Façade

- Le patron façade fournit une interface unifiée sur un ensemble d'interfaces d'un système.
- Il est utilisé pour réaliser des interfaces de programmation.
- Si un sous-système comporte plusieurs composants qui doivent être utilisés dans un ordre précis, une classe façade sera mise à disposition, et permettra de contrôler l'ordre des opérations et de cacher les détails techniques des sous-systèmes.

## Poids plume

- Dans le patron poids-plume, un type d'objet est utilisé pour représenter une gamme de petits objets tous différents.
- Ce patron permet de créer un ensemble d'objets et de les réutiliser. Il peut être utilisé par exemple pour représenter un jeu de caractères : un objet fabrique va retourner un objet correspondant au caractère recherché. La même instance peut être retournée à chaque fois que le caractère est utilisé dans un texte.

## Interpréteur

- Le patron Interpreteur comporte deux composants centraux : le contexte et l'expression ainsi que des objets qui sont des représentations d'éléments de grammaire d'un langage de programmation.
- Ce patron est utilisé pour transformer une expression écrite dans un certain langage de programmation - un texte source - en quelque chose de manipulable par programme : le code source est écrit conformément à une ou plusieurs règles de grammaire, et un objet est créé pour chaque utilisation d'une règle de grammaire. L'objet interpreteur est responsable de transformer le texte source en objets.

## Itérateur

- Ce patron permet d'accéder séquentiellement aux éléments d'un ensemble sans connaitre les détails techniques du fonctionnement de l'ensemble.
- Selon la spécification originale, il consiste en une interface qui fournit les méthodes Next et Current. L'interface en Java comporte une méthode next, une méthode hasNext et une méthode optionnelle remove.

## Médiateur

- Ce patron permet à plusieurs objets de communiquer entre eux en évitant à chacun de faire référence aux autres.
- Ce patron est utilisé quand il y a un nombre non négligeable de composants et de relations entre les composants.
- Un médiateur est placé au milieu des objets communiquant et le nombre de relations est diminué puisque chaque composant est relié uniquement au médiateur.
- Le mediateur sert d'intermédiaire pour assurer les communications entre les objets.

## Mémento

- Ce patron vise à externaliser l'état interne d'un objet sans perte d'encapsulation et permet de remettre l'objet dans l'état où il était auparavant.
- Ce patron permet de stocker l'état interne d'un objet sans que cet état ne soit rendu publique par une interface. Il est composé de trois classes : l´origine - d'où l'état provient, le memento - l'état de l'objet d'origine, et le gardien qui est l'objet qui manipulera le memento. L´origine comporte une méthode pour manipuler les memento. Le gardien est responsable de stocker les memento et de les renvoyer à leur origine.
- Ce patron ne définit pas d'interface précise pour les différents objets, qui sont cependant toujours au nombre de trois.

## Observateur

- Ce patron établit une relation un à plusieurs entre des objets, où lorsqu'un objet change, plusieurs autres objets sont avisés du changement.
- Dans ce patron, un objet sujet tient une liste des objets dépendants (des observateurs) qui seront avertis des modifications apportées au sujet. Quand une modification est apportée, le sujet demande aux observateurs de s'actualiser.
- La demande d'actualisation peut contenir une description détaillée du changement. Certains changements étant sans effet sur une partie des observateurs, la description du changement permet aux observateurs de déterminer s'ils doivent s'actualiser ou s'ils ignorent la demande.

## Prototype

- Ce patron permet de définir le genre d'objet à créer en dupliquant une instance qui sert d'exemple - le prototype.
- L'objectif de ce patron est d'économiser le temps nécessaire pour instancier des objets.
- Selon ce patron, une application comporte une instance d'un objet, qui sert de prototype. Cet objet comporte une méthode clone pour créer des duplicata.

## Fondé de pouvoir (*proxy*)

- Le fondé de pouvoir  se substitut à un objet, ce qui permet de contrôler l'utilisation de ce dernier.
- Un fondé de pouvoir est un objet destiné à protéger un autre objet. Il a la même interface que l'objet à protéger. Un fondé de pouvoir peut être créé par exemple pour permettre d'accéder à distance à un objet ou dans le but de retarder la création de l'objet protégé - qui sera créé immédiatement avant d'être utilisé.
- Dans sa forme la plus simple, un fondé de pouvoir ne protège rien du tout et transmet tous les appels de méthode à l'objet cible.

## Singleton

- Ce patron vise à assurer qu'il n'y a toujours qu'une seule instance d'une classe en fournissant une interface pour la manipuler.
- La classe instanciable une seule fois comporte une méthode pour obtenir cette unique instance et un mécanisme pour empêcher la création d'autres instances.

## Etat

- Ce patron permet à un objet de modifier son comportement lorsque son état interne change
- Ce patron est souvent utilisé pour implanter une machine à états.
- Selon ce patron il y a une classe machine à états, et une classe pour chaque état. Lorsqu'un événement provoque le changement d'état, la classe machine à états se relie à un autre état et modifie ainsi son comportement.

## Stratégie

- Dans ce patron, une famille d'algorithmes est encapsulée de manière qu'ils soient interchangeables. Les algorithmes peuvent changer indépendamment de l'application qui s'en sert.
- Ce patron comporte trois rôles : le contexte, la stratégie et les implantations. La stratégie est l'interface commune aux différentes implantations - typiquement une classe abstraite. Le contexte est l'objet qui va associer un algorithme avec un processus.

## Modèle (*Template method*)

- Ce patron définit la structure générale d'un algorithme en déléguant certains passages. Permettant à des sous-classes de modifier l'algorithme en conservant sa structure générale.
- Il est utilisé lorsqu'il y a plusieurs implantations possibles d'un calcul. Une classe d'exemple (anglais template) comporte des méthodes d'exemple, qui, utilisées ensemble, implantent un algorithme par défaut. Certaines méthodes peuvent être vides ou abstraites. Les sous-classes de la classe template peuvent remplacer certaines méthodes et ainsi créer un algorithme dérivé.

# Visiteur

- Ce patron représente une opération à effectuer sur un objet ou un ensemble d'objets.
- Il permet de modifier l'opération sans changer l'objet concerné ni la structure.
- Selon ce patron, les objets à modifier sont passés en paramètre à une classe tierce qui effectuera des modifications.
- Une classe abstraite Visitor définit l'interface de la classe tierce. Ce patron est utilisé notamment pour manipuler un jeu d'objets, où les objets peuvent avoir différentes interfaces, qui ne peuvent pas être modifiés.