Les vingt-trois patrons GoF :

Fabrique et Fabrique abstraite

Ce patron fournit une interface pour créer des familles d'objets sans spécifier la classe concrète11. Le patron fabrique, ou méthode fabrique (en anglais factory ou factory method) est un patron récurrent. Une fabrique simple retourne une instance d'une classe parmi plusieurs possibles, en fonction des paramètres qui ont été fournis. Toutes les classes ont un lien de parenté, et des méthodes communes, et chacune est optimisée en fonction d'une certaine donnée10. Le patron fabrique abstraite (en anglais abstract factory) va un peu plus loin que la fabrique simple. Une fabrique abstraite est utilisée pour obtenir un jeu d'objets connexes. Par exemple pour implémenter une charte graphique : il existe une fabrique qui retourne des objets (boutons, menus) dans le style de Windows, une qui retourne des objets dans le style de Motif, et une dans le style de Macintosh. Une fabrique abstraite est obtenue en utilisant une fabrique simple.

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 incompatibles11. Il est utilisé dans le cas où un programme se sert d'une bibliothèque de classe 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 (en anglais adapter) expose alors l'ancienne interface en utilisant les fonctionnalités de la nouvelle.

Pont

Ce patron permet de découpler une abstraction de son implémentation, de telle manière qu'ils peuvent évoluer indépendamment11. Il consiste à diviser une implémentation en deux parties : une classe d'abstraction qui définit le problème à résoudre, et une seconde classe qui fournit une implémentation. Il peut exister plusieurs implémentations pour le même problème et la classe d'abstraction comporte une référence à l'implémentation choisie, qui peut être changée selon les besoins13[source insuffisante]. Le patron pont (en anglais bridge) est fréquemment utilisé pour réaliser des récepteurs d'événements.

Monteur

Ce patron sépare le processus de construction d'un objet du résultat obtenu. Permet d'utiliser le même processus pour obtenir différents résultats11. C'est une alternative au pattern 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 (en anglais builder). 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 pattern est particulièrement utile quand il y a de nombreux paramètres de création, presque tous optionnels.

Chaîne de responsabilité

Le patron chaîne de responsabilité (en anglais chain of responsibility) vise à découpler l'émission d'une requête de la réception et le traitement de cette dernière en permettant à plusieurs objets de la traiter successivement11. 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 emboîte une demande dans un objet, permettant de paramétrer, mettre en file d'attente, journaliser et annuler des demandes11. Dans ce patron un objet commande (en anglais command) 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 implémente cette interface — qui comporte une méthode execute. L'opération est lancée lorsque la méthodeexecute est utilisée. Ce patron est notamment utilisé pour les barres d'outils.

Composite

Le patron composite (même nom en anglais) permet de composer une hiérarchie d'objets, et de manipuler de la même manière un élément unique, une branche, ou l'ensemble de l'arbre11. Il permet en particulier de créer des objets complexes en reliant différents objets selon une structure en arbre. Ce patron impose que les différents objets aient une même interface, ce qui rend uniformes les manipulations de la structure. Par exemple dans un traitement de texte, les mots sont placés dans des paragraphes disposés dans des colonnes dans des pages ; pour manipuler l'ensemble, une classe composite implémente une interface. Cette interface est héritée par les objets qui représentent les textes, les paragraphes, les colonnes et les pages.

Décorateur

Le patron décorateur (en anglais decorator) permet d'attacher dynamiquement des responsabilités à un objet. Une alternative à l'héritage11. Ce patron est inspiré des poupées russes. Un objet peut être caché à l'intérieur d'un autre objet décorateur qui lui rajoutera 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 implémentent la même interface, qui est typiquement définie par une classe abstraite.

Façade

Le patron façade (en anglais facade) fournit une interface unifiée sur un ensemble d'interfaces d'un système11. 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.

Flyweight

Dans le patron flyweight (en français poids-mouche), un type d'objet est utilisé pour représenter une gamme de petits objets tous différents11. 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 factory 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.

Interpreter

Le patron comporte deux composants centraux : le contexte et l'expression20 ainsi que des objets qui sont des représentations d'éléments de grammaire d'un langage de programmation11. Le patron est utilisé pour transformer une expression écrite dans un certain langage de programmation — un texte source — en quelque chose de manipulable par programmation20 : 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 interpreter est responsable de transformer le texte source en objets.

Iterator

Ce patron permet d'accéder séquentiellement aux éléments d'un ensemble sans connaitre les détails techniques du fonctionnement de l'ensemble11. C'est un des patrons les plus simples et les plus fréquents. Selon la spécification originale, il consiste en une interface qui fournit les méthodes Next et Current. L'interface en Java comporte généralement une méthode nextElement et une méthode hasMoreElements.

Mediator

Dans ce patron il y a un objet qui définit comment plusieurs objets communiquent entre eux en évitant à chacun de faire référence à ses interlocuteurs11. Ce patron est utilisé quand il y a un nombre non négligeable de composants et de relations entre les composants. Par exemple dans un réseau de 5 composants il peut y avoir jusqu'à vingt relations (chaque composant vers quatre autres). Un composant médiateur est placé au milieu du réseau et le nombre de relations est diminué : chaque composant est relié uniquement au médiateur22[source insuffisante]. Le mediator joue un rôle similaire à un sujet dans le patron observer et sert d'intermédiaire pour assurer les communications entre les objets.

Memento

Ce patron vise à externaliser l'état interne d'un objet sans perte d'encapsulation. Permet de remettre l'objet dans l'état où il était auparavant11. Ce patron permet de stocker l'état interne d'un objet sans que cet état ne soit rendu public 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.

Observer

Ce patron établit une relation un à plusieurs entre des objets, où lorsqu'un objet change, plusieurs autres objets sont avisés du changement11. Dans ce patron, un objet le sujet tient une liste des objets dépendants des observateurs qui seront avertis des modifications apportées au sujet. Quand une modification est apportée, le sujet émet un message aux différents observateurs. Le message peut contenir une description détaillée du changement23[source insuffisante]. Dans ce patron, un objet observer comporte une méthode pour inscrire des observateurs. Chaque observateur comporte une méthode Notify. Lorsqu'un message est émis, l'objet appelle la méthode Notify de chaque observateur inscrit.

Prototype

Ce patron permet de définir le genre d'objet à créer en dupliquant une instance qui sert d'exemple — le prototype11. 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 duplicatas. Des langages de programmation comme PHP ont une méthode clone incorporée dans tous les objets26[source insuffisante].

Proxy

Ce patron est un substitut d'un objet, qui permet de contrôler l'utilisation de ce dernier11. Un proxy est un objet destiné à protéger un autre objet. Le proxy a la même interface que l'objet à protéger. Un proxy peut être créé par exemple pour permettre d'accéder à distance à un objet (via un middleware). Le proxy peut également être créé 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 proxy 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 manipuler11. C'est un des patrons les plus simples. L'objet qui ne doit exister qu'en une seule instance comporte une méthode pour obtenir cette unique instance et un mécanisme pour empêcher la création d'autres instances.

State

Ce patron permet à un objet de modifier son comportement lorsque son état interne change11. Ce patron est souvent utilisé pour implémenter une machine à états. Un exemple d'appareil à états est le lecteur audio — dont les états sont lecture, enregistrement, pause et arrêt. Selon ce patron il existe 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.

Strategy

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 sert11. Il comporte trois rôles : le contexte, la stratégie et les implémentations. La stratégie est l'interface commune aux différentes implémentations — typiquement une classe abstraite. Le contexte est l'objet qui va associer un algorithme avec un processus.

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 générale11. C'est un des patrons les plus simples et les plus couramment utilisés en programmation orientée objet. Il est utilisé lorsqu'il y a plusieurs implémentations possibles d'un calcul. Une classe d'exemple (anglais template) comporte des méthodes d'exemple, qui, utilisées ensemble, implémentent 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.

Visitor

Ce patron représente une opération à effectuer sur un ensemble d'objets. Permet de modifier l'opération sans changer l'objet concerné ni la structure11. 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.

Source WIKIPEDIA : https://fr.wikipedia.org/wiki/Patron_de_conception