Retrouver des patterns ? :o

Notes Pattern.

Bisous sur vos culs <3

Bruno Loverius

Decorator :

On recherche une classe qui implémente une interface et qui possède un attribut du même type que cette interface. Dans le main, une chaine de news doit être créée pour utiliser le Decorator.
new Captain(new BattleFishingBoat(new CrewMate(…))) ;

Coposite :

On recherche une classe qui implémente une interface et qui retient en attribut une collection d'objets du même type que cette interface. Dans le main, une chaine de news doit être créée aussi.

Facade

Dans le cas d'un projet contenant plein de classes, si on ne se sert que d'une seule d'entre elles, on peut considérer que c'est notre clase facade. C'est l'équivalent des UCC dans le projet AE.

Template Method

Si on à une classe abstraite qui implémente une méthode concrète ET une méthode abstraite et que la méthode concrète appelle la méthode abstraite, on est sur une Template method.

Factory Method

C'est un cas d'exception de la Template Méthod. Si l'implémentation de la méthode abstraite renvoie un new, on est sur une Factory Method.

Strategy

Quand on à une interface qui donne la signature d'UNE méthode et que cette méthode est implémentée de différentes façons dans d'autres classes implémentant cette interface, on retrouve une strategy. La strategy est passée en paramètre d'une méthode dont le résultat varie en fonction de ladite strategy.

Visitor

Ressemble assez bien à la strategy. Il ne faut pas confondre. (Souvent en accord avec un composite). Le pattern Visitor doit contenir deux interfaces. Une interface d'action (le visiteur) avec plusieurs méthodes. Il y aura autant de méthodes que d'objets implémentant l'autre interface (interface élément).
L'interface élément implémente d'office une méthode (la méthode accept le nom peut changer). Cette méthode doit prendre un Visitor en paramètres. Donc, un élément de l'autre interface celle contenant autant de méthodes que de classes implémentant l'interface Element.
Vu que « Accept » prend un visitor en paramètre, on peut appeler la méthode du visitor correspondant à la classe dans laquelle on se trouve en passant « this » en paramètre de la méthode correspondante.

Adaptor

Le But derrière le pattern Adaptor est de faire un passer un objet d'une certaine classe pour ce qu'il n'est dans l'optique de permettre à plusieurs interfaces de travailler entre elles. On le reconnait avec une classe qui implémente une interface. Cette classe contient un objet d'une autre classe du projet et toutes les méthodes implémentées de l'interface sont « Renvoyées » à cet objet. Elles sont généralement gérées sur une seule ligne. En gros toutes les méthodes de la classe implémentant les méthodes de l'interface vont déléguer le travail à l'objet en attribut qui agira comme bon semble à ce dernier.

Singleton

On ne le présente plus. Tu cherches une instance, un constructeur private et une méthode qui call le constructeur.

Observer

Pour trouver un Observer, il faut qu'une classe possède une liste d'objets X (X étant l'observer). Il faut que chacun de ces Observers aient une méthode d'action. Tous les observeurs effectueront cette méthode à la demande de la classe subject. La classe sujette donc possède une méthode dont le but est de parcourir la collection d'observers et de leur faire effectuer la méthode.

Iterator

On ne le présente plus. On cherche des méthodes next et has next et toute méthode qui se rapproche de près ou de loin au parcours d'une liste particulère.

Prototype

Si tu vois une méthode de clonage, il est très possible que tu aies un prototype. Sans clône, pas de prototype. Donc, ça doit implémenter l'interface cloneable. Dans le main (ou ailleurs), il faut checker si les objets sont réellement créés ou si ils sont clonés.

Flyweigth

Pense à celui-là s'il faut créer un grand nombre d'objets de la même classe. On doit considérer des propriétés qui seront dites intrinsèques et extrinsèques. Prenons l'exemple de quelqu'un qui veut dessiner 100 000 rectangles dans une fenêtre. Ça prendrait quelques secondes de systématiquement devoir créer tous les carrés et ça prendrait masse mémoire. On peut donc considérer des propriétés intrinsèques (disons la couleur) et extrinsèques (la position, la taille…). On va donc mettre en place une liste de couleurs et instancier un seul carré par couleurs dans un hashmap. Ensuite, on réutilisera le même carré qu'on posera à des positions et tailles différentes.

Quand on utilise beaucoup d'objets, on se servira de Flyweight. On aura une map qui à pour but de garder en mémoire une petite quantité de la masse d'objets qu'on crée pour les réutiliser.

Command Pattern

https://www.youtube.com/watch?v=7Pj5kAhVBlg

De rien.

Builder

Super simple, On crée un objet avec une chaine de méthodes qui portent chacune le nom de ses attributs. On utilise ça quand on doit créer des objets qui peuvent avoir toute une série de champs à null. On doit avoir un objet et un builder de cet objet. L'objet doit avoir un constructeur private et tous ses champs en final quand le constructeur de l'objet n'a que les champs requis en final et tous les autres ne le sont pas. Le constructeur de l'objet est appelé pour créer l'objet suivi des méthodes dont le but est de remplir les attributs. On termine par une méthode build qui renverra l'objet et non pas le constructeur de l'objet.

NB : quand je parle de constructeur ici, je parle d'un objet dont le but est de construire un autre. Absolument rien à vois avec le constructeur de la classe. Ne pas confondre les deux.

Abstract Factory

La vidéo est sur moodle. Très simple.

Chain of responsability

On le retrouve easy dans son instansiation. C'est comme un décorator sauf que ça se termine par null.

State

La vidéo est sur moodle, très simple.