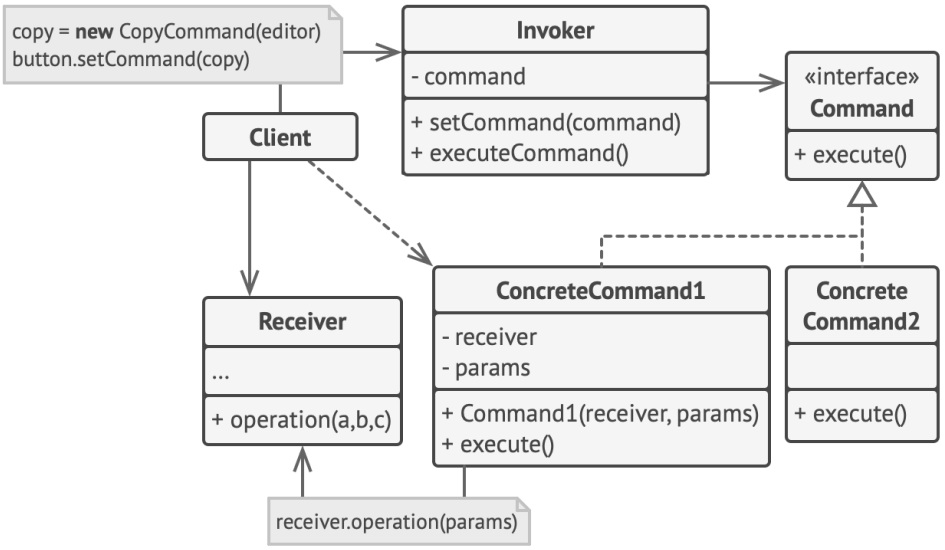
Pattern Commande : fiche résumée

Problématique et intentions du pattern :

Le design pattern Commande est une approche qui prend une action à réaliser et la convertit en un objet autonome qui encapsule tous les détails de cette action. Cette conversion permet de paramétrer des méthodes avec différentes actions, de planifier leur exécution, de les mettre en file d'attente ou d'annuler des opérations déjà effectuées. Il fait partie de la famille des patterns comportementaux.

Diagramme de classe :



Classes participantes :

Client : Le client crée et configure les objets des commandes concrètes. Il les configure en y mettant les paramètres nécessaires. C'est lui qui effectue les commandes avec les objets créés.

Invoker : L'invoker est responsable de l'envoi des commandes, càd qu'il va déclencher la commande que le client lui a mis en paramètre indépendamment de ce même paramètre. Il transmet juste le fait d'exécuter une commande.

Command : l'interface command sert juste à déclarer la méthode execut.

Concrete command : elle permet d'exécuter la commande d'un objet métier. Elle prend en paramètre un reciver qui va faire son opération.

Receiver : c’est lui qui exécute la commande avec les paramètres que la commande concrète lui a transmis.

Exemple de contexte métier :

On peut prendre l’exemple utilisé dans notre exposé.
Dans cet exemple de système de domotique, le design pattern Commande est utilisé pour structurer la logique d’une télécommande intelligente capable de contrôler plusieurs appareils tels qu’une lumière, un volet roulant ou une alarme. L’objectif principal est de découpler les commandes envoyées par la télécommande des actions réelles effectuées par les appareils, afin de rendre le système extensible et facile à maintenir.

On peut aussi prendre l’exemple d’un logiciel de dessin où l’on peut annuler ou rétablir les actions faites par l’utilisateur.

SOLIDité du pattern :

SRP : une classe à une responsabilité. Chaque classe a sa propre responsabilité.

OCP : ouvert aux extensions, fermées aux modifications. On peut ajouter une commande sans endommager le code existant.

LSP : Les sous-types doivent pouvoir être substitués à leur type de base. Respecté, car il n’y a aucun héritage

ISP : Un client ne doit pas être forcé d’utiliser un service dont il n’a pas besoin. Respecté, car il n’y a qu'une seule interface avec une méthode qui est implémentée par toutes les classes.

DIP : Respecté, car il n’y a pas de module de bas ou de haut niveau.

Limites pattern :

Le code peut être compliqué à lire, car il y a une multiplication des classes. Grande complexité càd pour utiliser une commande, il faut créer un invoker, une concrete command, un receiver et que chaque classe effectue leur opération. Il est aussi difficile de savoir quelle classe a effectué quelle méthode, il est donc plus difficile de debugger.