Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

ATL – Ateliers Logiciels

Design pattern Commande

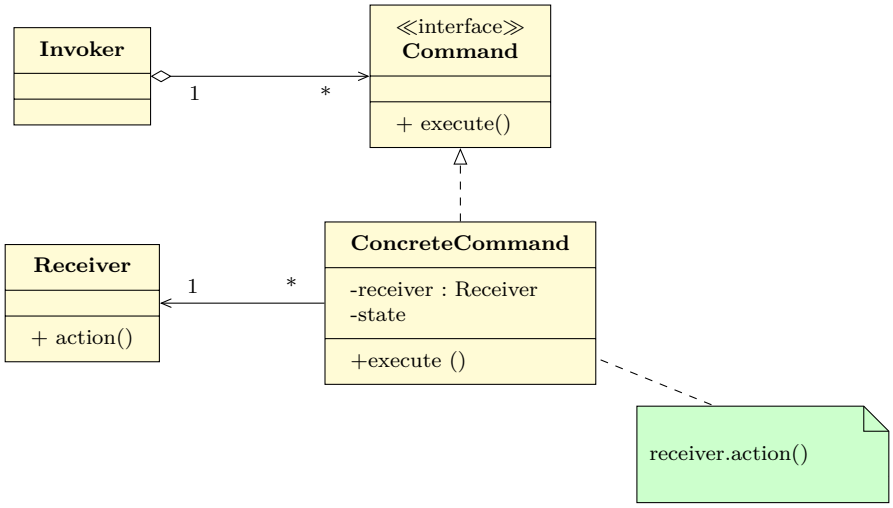
Introduction

Consignes

Ce document introduit le *design pattern* (patron de conception) *Commande*.
Vous pouvez obtenir les ressources de cette fiche via le lien `https://git.esi-bru.be/ATL/designpatterncards`.
N'oubliez pas de vous **connecter** au serveur `git.esi-bru.be` pour avoir accès au téléchargement.

1 Commande

Dans le *Design Pattern Commande*, une *action* représentant une fonctionnalité d'une application est encapsulée par un objet, qui est la seule entité capable d'effectuer cette action et possède toutes les informations nécessaire pour déclencher celle-ci. Il permet de refactoriser le code en s'assurant qu'une même fonctionnalité, appelable depuis de nombreux composants d'une application, se déroulera toujours de façon unique.



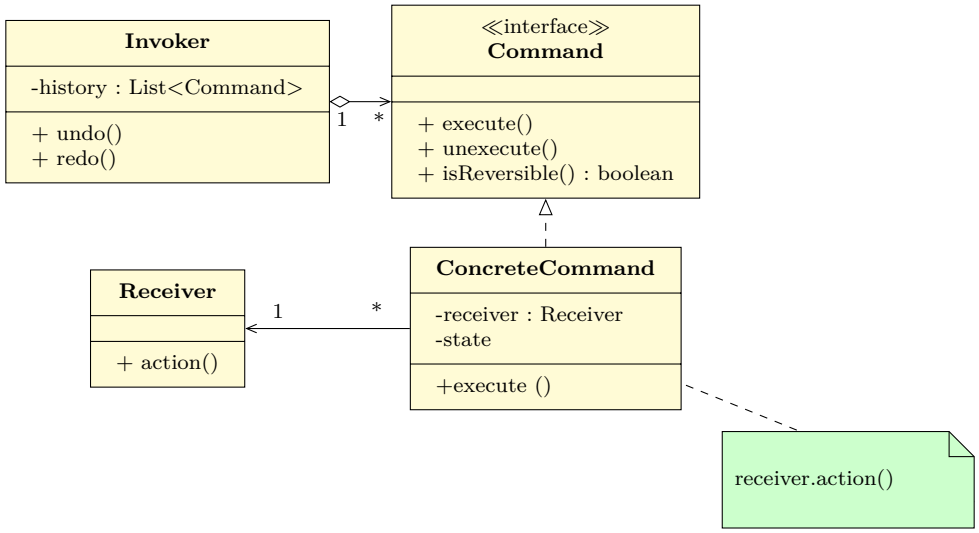
L'invocateur (*invoker*) est l'objet en charge de déclencher une commande. Tout autre élément de l'application (on parlera parfois de client) va donc demander à l'invocateur d'exécuter une commande via une requête pour déclencher celle-ci. Cette séparation entre l'événement qui déclenche l'appel d'une commande avec l'implémentation de celle-ci permet de réduire le couplage entre les classes, puisqu'on peut dès lors déterminer quelle commande employer durant l'exécution plutôt qu'à la compilation.

L'objet commande est en charge d'implémenter l'exécution concrète de la commande. Ces commandes vont typiquement affecter d'autres composants de l'application, nommés ici récepteurs *Receiver*.

Version du document
17 octobre 2022

Ce *design pattern* permet en outre d'implémenter des fonctionnalités d'annulation : il suffit pour cela que l'invocateur garde une liste des commandes effectuées, et que les commandes reçoivent une méthode supplémentaire permettant l'annulation de la commande.

Annuler (ou refaire) des actions revient alors à naviguer dans cette liste.



2 Exemple

2.1 Cas simple : allumer la lumière

À titre d'illustration, deux exemples sont disponibles sur le serveur *gitlab* de l'école : `https://git.esi-bru.be/ATL/designpatterncards`.

L'application `PressSwitch` présente un exemple simple de la séparation des responsabilités quand le *design pattern Commande* est employé. Cet exemple représente une lampe qui peut être allumée ou éteinte.

- La classe `SwitchManager` est un invocateur : il possède une liste de commandes pour lesquelles il peut appeler la méthode `execute()`. Le choix de la commande est ici effectué via un `switch` sur une chaîne de caractère afin de pouvoir ajouter ou supprimer des commandes de façon transparente du point de vue de l'invocateur.
- La classe `Light` est un récepteur pour les commandes implémentées ci-dessous : il s'agit d'un objet de l'application avec ses méthodes propres permettant d'allumer ou éteindre la lampe.
- L'interface `Command` est implémentée par deux commandes : `FlipUpCommand` et `FlipDownCommand`. Toutes deux ont besoin d'un récepteur `Light` lors de leur création pour savoir quelle lampe elles contrôlent.

Notez qu'il est tout à fait possible que différentes commandes disposent de récepteurs différents ; l'interface `Command` garantit que le fonctionnement restera transparent.

2.2 Cas plus complexe : éditeur de texte

L'application `TextEditor` présente un exemple plus complexe : un éditeur de texte implémenté en JavaFX. Cet éditeur est doté de commandes permettant de copier, couper ou coller du texte, ainsi que d'une commande d'annulation.

Dans cette application :

- L'invocateur est l'application `TextEditorApp` (qui hérite de l'application JavaFX). Différents boutons placés dans des menus appellent les différentes commandes.
- Le récepteur est la classe `Editor`, un composant de l'interface graphique sur lequel on effectue des modification (copier, couper, coller).
- Chacune de ces modifications prend la forme d'une commande. Pour pouvoir être annulée, chaque commande stocke des informations sur l'état du récepteur avant d'exécuter la commande, afin de pouvoir rétablir cet état.
- L'annulation est implémentée via un historique des commandes géré par l'invocateur.