Design Pattern
MEDIATOR

CLASSE: L3GLSI B

Idrissa **SOW**

EMMANUEL **DIATTA**

Khady Mbacké **DIOP**

Cheikh Ibra **DIOP**

Ecole Supérieure Polytechnique

2020-2021

Introduction

Le développement orienté objet ordinaire distribue la responsabilité aussi loin que possible, avec chaque objet accomplissant sa tâche indépendamment des autres.

Par exemple, le pattern OBSERVER supporte cette distribution en limitant la responsabilité d'un objet que d'autres objets trouvent intéressant. Le pattern SINGLETON résiste à la distribution de la responsabilité et vous permet de la centraliser au niveau de certains objets que les clients localisent et réutilisent.

A l'instar de SINGLETON, le pattern MEDIATOR centralise la responsabilité mais pour un ensemble spécifique d'objets plutôt que pour tous les clients dans un système. Lorsque les interactions entre les objets reposent sur une condition complexe impliquant que chaque objet d'un groupe connaisse tous les autres, il est utile d'établir une autorité centrale.

La centralisation de la responsabilité est également utile lorsque la logique entourant les interactions des objets en relation est indépendante de l'autre comportement des objets.

Contexte

Différents objets ont des interactions : un événement sur l'un provoque une action ou des actions sur un autre ou d'autres objets.

Il se pose donc un besoin de centraliser le contrôle et les communications complexes entre objets apparentés.

Pour cela, il serait utile de construire un objet dont la vocation est la gestion et le contrôle des interactions complexes entre un ensemble d'objets sans que les éléments doivent se connaitre mutuellement.

Objectifs

L'objectif du pattern MEDIATOR est de définir un objet qui encapsule la façon dont un ensemble d'objets interagissent. Cela promeut un couplage lâche, évitant aux objets d'avoir à se référer explicitement les uns aux autres, et permet de varier leur interaction indépendamment.

Le pattern MEDIATOR:

- Gérer la transmission d'information entre des objets interagissant entre eux,
- Avoir un couplage faible entre les objets puisqu'ils n'ont pas de lien direct entre eux.
- Pouvoir varier leur interaction indépendamment.

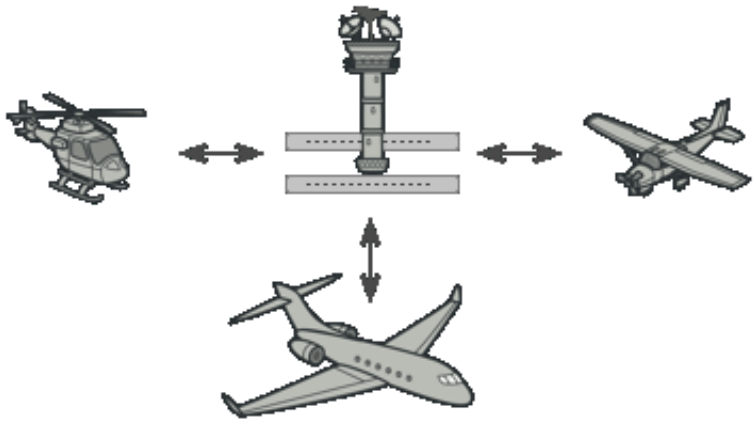
Problématique

Afin d'illustrer facilement le problème, prenons tout simplement comme illustration un aéroport et sa tour de contrôle.

Dans un aéroport, les différents avions doivent communiquer entre eux pour s'assurer des routes aériennes prises par les uns et les autres ainsi que s'assurer que la piste d'atterrissage est inoccupée.

Le problème est que réside dans le fait que lorsque chaque avion doit communiquer avec tous les autres avions d'un certain aéroport, la communication devient pénible voire impossible.

La solution qui a été proposée est celle de la tour de contrôle; ainsi la communication entre les deux se fera par le biais de ce dernier. Les objets avions qui étaient fortement couplés seront maintenant faiblement couplés.



Fonctionnement du pattern MEDIATOR (exemple de la tour de contrôle d'un aéroport)

Ce pattern nous permet de concevoir des composants réutilisables dont les dépendances entre eux démontre le phénomène du code spaghetti.

Définition du Design Pattern Médiateur

Le pattern MEDIATOR est un patron de conception comportemental qui diminue la complexité et les dépendances chaotiques entre les objets nécessairement couplés communiquant directement entre eux. Il restreint les communications directes entre les objets et les force à collaborer uniquement via un objet médiateur qui s'occupe de l'interaction entre les objets dépendants.

Voyons cela avec l'illustration ci-dessous d'un diagramme de classe représentant le fonctionnement d'un médiateur dans une entreprise :

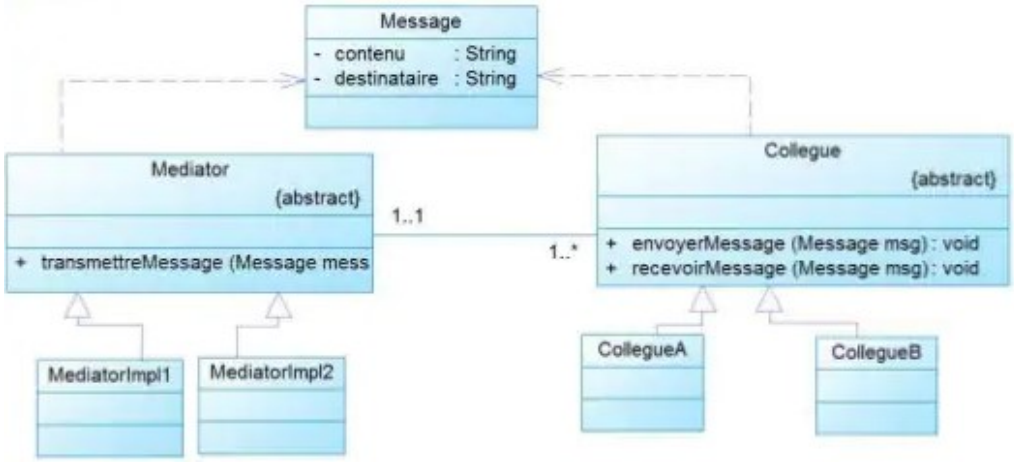


Diagramme UML (médiation entre collègues)

Avantages et inconvénients

Avantages

- Principe de responsabilité unique (*une classe ne doit s'occuper que d'une seule partie du code, ce pourquoi elle existe)* et Principe ouvert/fermé *(un objet doit être ouvert à l'extension et fermé à la modification)*.
- Simplifie la maintenance du système en centralisant la logique de contrôle.
- Augmente la réutilisabilité des objets pris en charge par le *médiateur* en les découplant du système.
- Réduit la variété des messages échangés par les différents objets du système.

Inconvénients

- Un médiateur peut évoluer en *God Object (objet, dans le domaine de POO, qui reconnaît trop de choses ou fait trop de choses)*
- En l'absence de conception soigneuse, l'objet Médiateur lui-même peut devenir exagérément complexe

Marche à suivre pour l'utilisation du pattern

1. Identifier une collection d'objets en interaction qui bénéficieraient d'un couplage faible
2. Encapsuler toutes les interactions d'objets dans une classe de *médiateur*
3. Restructurer toutes les classes de *« collègues »* pour n'interagir qu'avec l'objet *médiateur*
4. Reconfigurer dynamiquement les interactions d'objets du MEDIATOR si nécessaire.

Conclusion

Le Design Pattern MEDIATOR permet ici d'isoler la communication entre les objets.

Il promeut le couplage lâche, évitant à des objets en relation de devoir se référer explicitement les uns aux autres. Il intervient le plus souvent dans le développement d'applications GUI, lorsque vous voulez éviter d'avoir à gérer la complexité liée à l'actualisation mutuelle d'objets.

L'architecture de Java pousse dans cette direction, en encourageant à définir des objets qui enregistrent des listeners pour les événements GUI. Si l'on développe des interfaces utilisateur avec Java, on applique probablement ce pattern.

Ainsi, le médiateur peut se concentrer sur l'interaction entre les composants GUI, et la classe d'application peut se concentrer sur la construction des composants.

D'autres situations se prêtent à l'introduction d'un objet médiateur. Par exemple, on pourrait en avoir besoin pour centraliser la responsabilité de préserver l'intégrité relationnelle dans un modèle objet. Le pattern MEDIATOR pourra être utilisé chaque fois que l'on devra définir un objet qui encapsule la façon dont un ensemble d'objets interagissent.