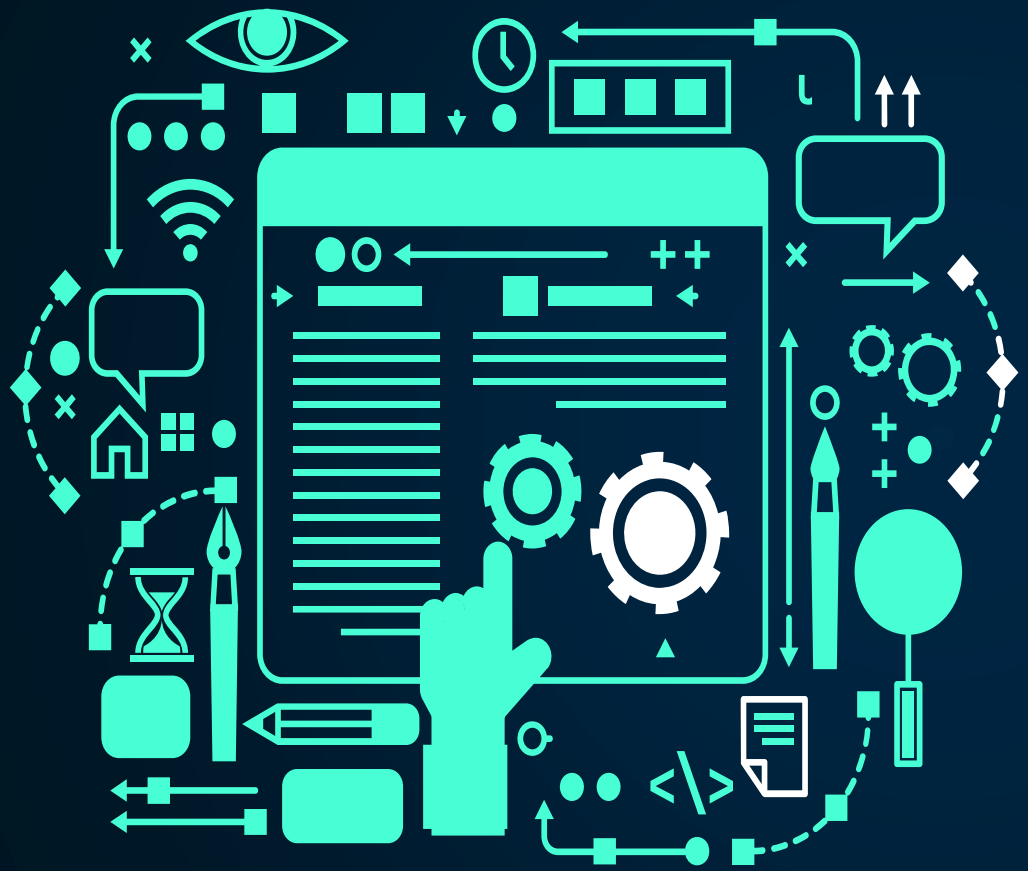


DESIGN Pattern MEDIATOR

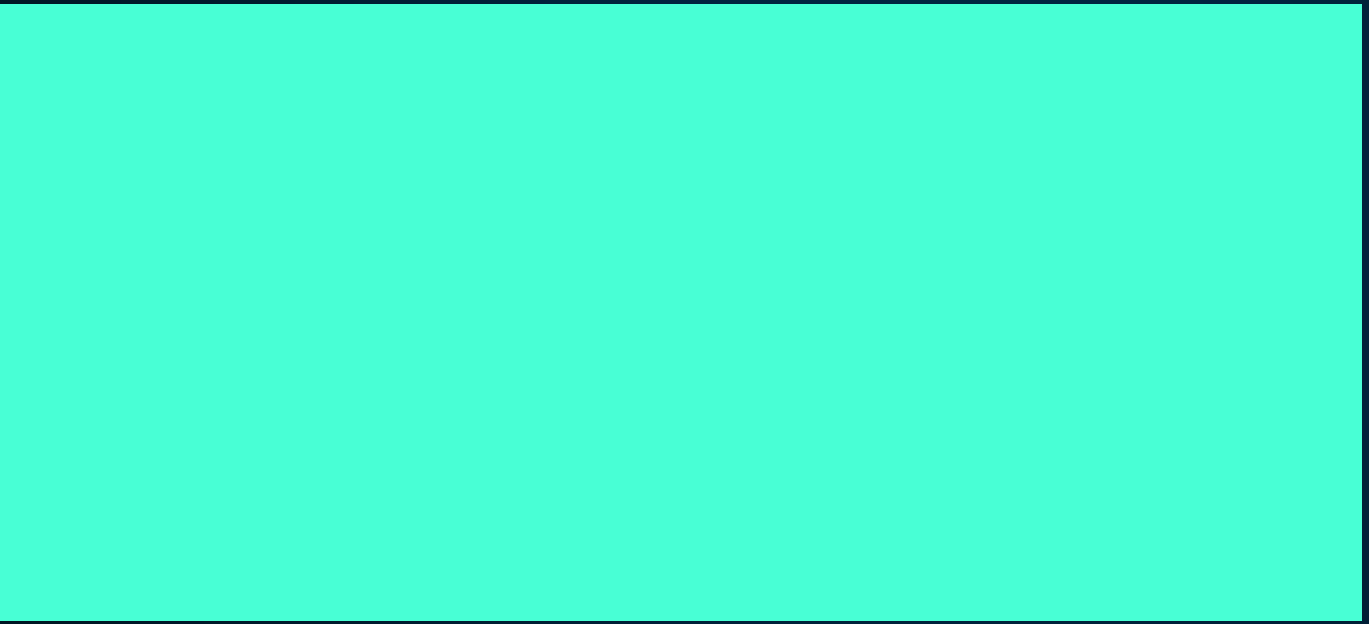
Bienvenue à notre présentation du design pattern MEDIATOR

Présenté par :

Idrissa SOW
Emmanuel DIATTA
Khady Mbacké Diop
Cheikh Ibra DIOP

Table des Matières

Définition du Design Pattern Mediator

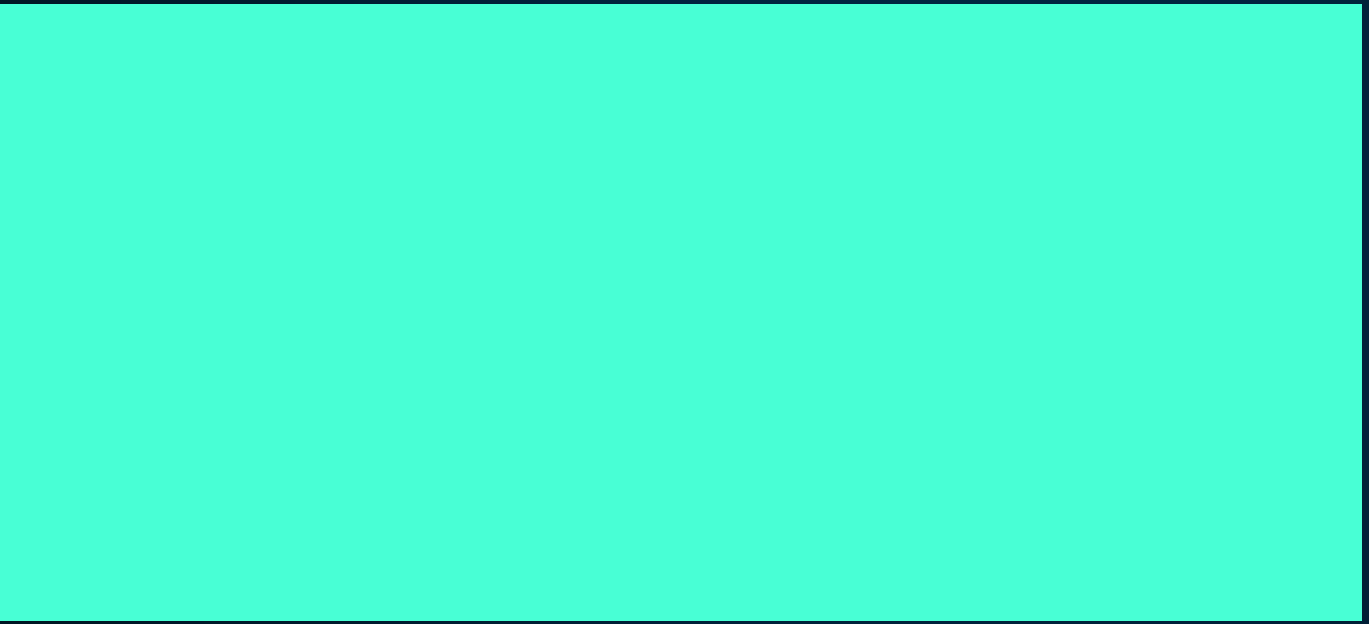


Définition du Design Pattern Mediator

- 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 dépendants.

Présentation du Design Pattern



Présentation du Pattern

- Mediator est un patron de conception comportemental.
- Il est utilisé pour réduire le couplage entre plusieurs classes et permettre ainsi une réutilisation directe de ces classes .
- Le patron de conception médiateur propose de mettre fin à toutes les communications directes entre les composants.

Présentation du Pattern

Contexte

Différents objets ont des interactions: un évènement sur l'un provoque une action sur d'autres objets. Pour cela, il pose donc:

- Un besoin de centraliser le contrôle et les communications complexes entre objets apparentés.
- Une utilité 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 connaître mutuellement.

Présentation du Pattern

Objectifs

L’objectif premier du pattern Mediator est de définir un 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.

Au-delà de cela, ce pattern permet de:

Présentation du Pattern

Objectifs

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

Présentation du Pattern

Problématique

Afin d'illustrer facilement le problème, prenons tout simplement l'exemple d'un feu tricolore.

De manière générale, d'un certain point de vue, les automobilistes ne communiquent pas directement entre eux. Ils regardent le feu tricolore et savent l'action à poser suivant la couleur active.

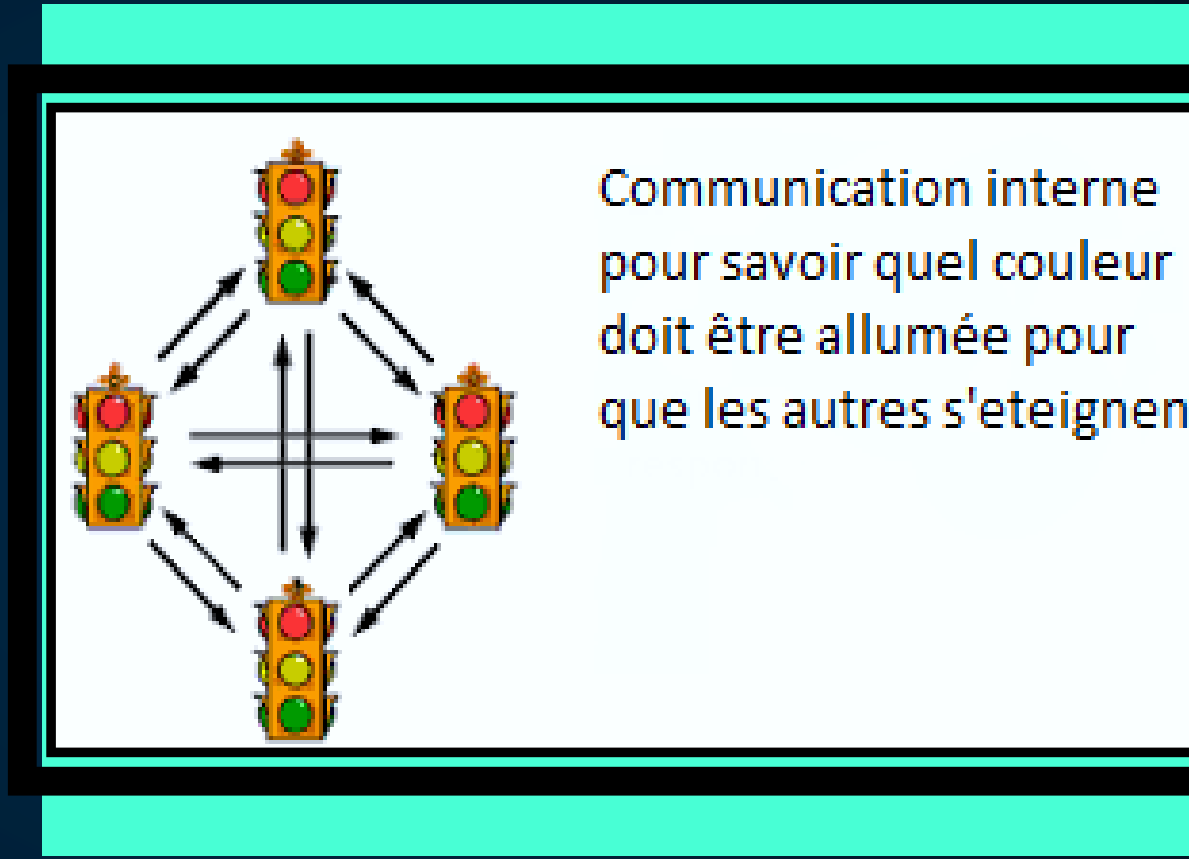
Présentation du Pattern

Problématique

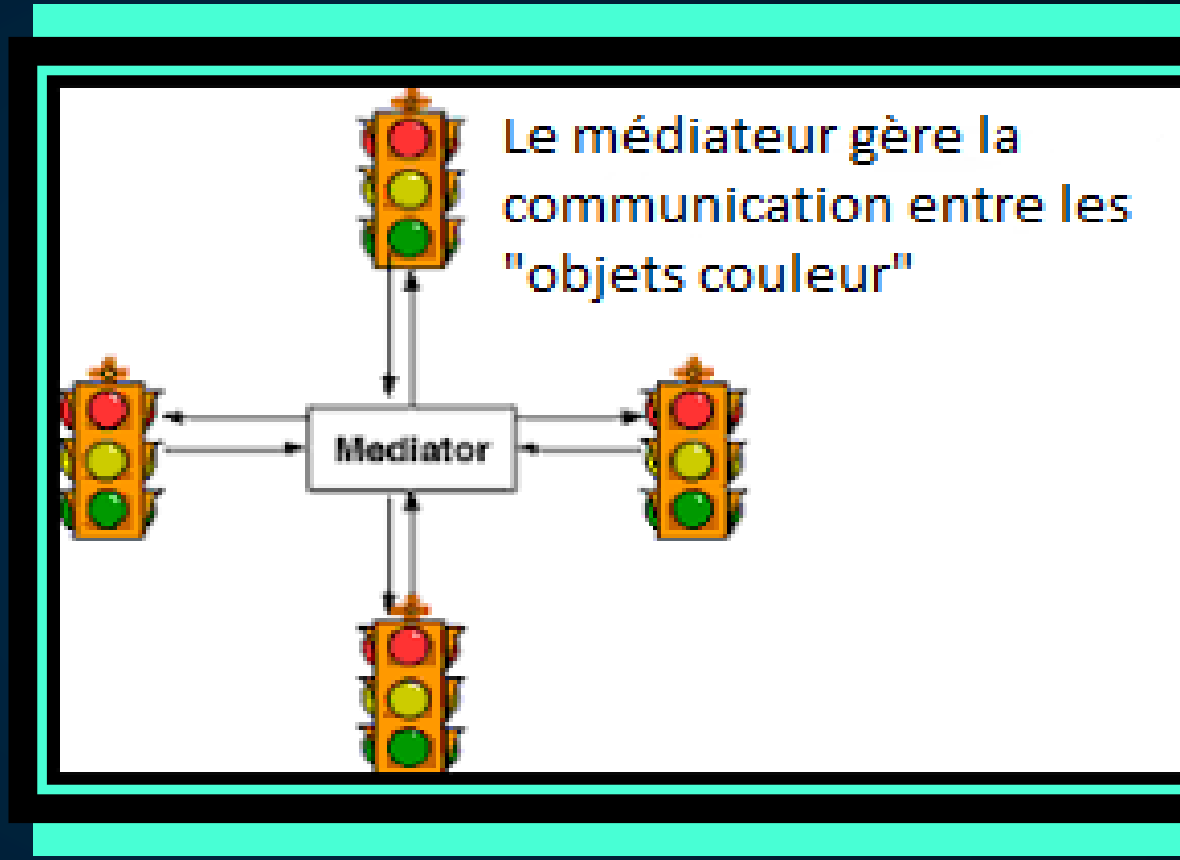
Cependant dans le fonctionnement interne du feu tricolore,

- Lorsqu'une couleur (Vert/Rouge/Jaune) est activée, les autres doivent s'éteindre et inversement.

- ✓ Les mécanismes de chaque couleur communiquent entre eux pour connaître l’état des autres et savoir l’état qu’elles doivent présenter (active/inactive).



- ✓ Le médiateur permet donc de réduire cette intercommunication en gérant les interactions.



CAS PRATIQUE

- *Solution avec l’utilisation du Pattern Mediator*

pattern_mediator - FeuMediator.java

```java
import java/util/hashset;

public class FeuMediator {
//Un feu est ON les autres sont OFF

//Hashset permet de garder une unicité pr les couleur.
    HashSet<Feu> feuSignal = new HashSet<Feu>();

//Enregistre l'objet feu dans le mediateur
    public void ajouteFeu(Feu feu) {
        feuSignal.add(feu);
    }

//Retire l'objet feu dans le mediateur
    public void retireFeu(Feu feu) {
        feuSignal.remove(Feu);
    }

    //Met toutes les couleurs en OFF sauf celle activée
    void offToutesLesCouleurs(Feu feu) {
        for (Feu l : feuSignal) {
            if (!(l.equals(feu))) {
                l.turnOFF();
            }
        }
        System.out.println("------------------------------");
    }

    /**
     * Quand le feu passe a 1 couleur le mediateur est notifié
     * Toutes les autres couleurs passent à OFF
     */
    public void notifMediator(Feu feu) {
        this.offToutesLesCouleurs(feu);
    }
}
```

- La classe FeuMediator gère l'interaction entre les couleurs du Feu tricolore afin de permettre à ces derniers de se charger de leur propre fonctionnement

pattern_mediator - Feu.java

```java
/**
 * Represente lle feu tricolore (Rouge, Vert, Jaune)
 */
class Feu {
//Les etats d'1 couleur du feu tricolore
    enum Etat {
        ON, OFF
    }

    enum Couleur {
        ON, OFF
    }


    private Couleur couleur;
    private Etat etatActuel;
    private FeuMediator mediator;

//Creer un objet Feu et l'enregistre au niveau du mediateur
    protected Feu(String couleur, FeuMediator mediator) {
        this.couleur = couleur;
        this.FeuMediator = mediator;
        mediator.ajouteFeu(this);
    }

//La couleur du feu passe à ON et notify le mediateur
    void turnON() {
        etatActuel = Etat.ON;
        System.out.printf("%s est en mode %s \n", this, etatActuel.ON);
        FeuMediator.notifMediator(this);
    }

//La couleur du feu passe à OFF
    void turnOFF() {
        etatActuel = Etat.OFF;
        System.out.printf("%s est en mode %s \n", this, etatActuel.OFF);
    }

//return couleur
    @Override
    public String toString() {
        return couleur;
    }
}
```

- Cette classe sert à créer les *«objets»* couleurs du feu tricolore (Vert/Rouge/Jaune)
- Elle présente aussi les méthodes qui définissent le comportement de ces objets

pattern_mediator - TestMediator.java

```java
import java.util.HashSet;

public class TestMediator {

    public static void main(String[] args) {

        FeuMediator mediator = new FeuMediator();
        Feu rouge = new Feu("Rouge", mediator);
        Feu vert = new Feu("Vert", mediator);
        Feu jaune = new Light("Jaune", mediator);

        rouge.turnON();
        vert.turnON();
        jaune.turnON();
    }
}
```

- La classe FeuMediator gère l'interaction entre les couleurs du Feu tricolore afin de permettre à ces derniers de se charger de leur propre fonctionnement

AVANTAGES & INCONVENIENTS

Avantages & Inconvénients

Avantages

Les avantages du patron de conception médiateur:

- Principe de responsabilité unique *(une classe ne doit s'occuper que d'une seule partie du code, ce pourquoi elle existe)*

- Principe ouvert/fermé *(un objet doit être ouvert à l'extension et fermé à la modification)*.

Avantages & Inconvénients

Avantages

- Simplification de la maintenance du système en centralisant la logique de contrôle.
- Augmentation de la réutilisabilité des objets pris en charge par le médiateur en les découplant du système.
- Réduction de la variété des messages échangés par les différents objets du système.

Avantages & Inconvénients

Inconvénients

Les inconvénients du patron de conception médiateur:

- Peut évoluer en God Object *(objet, dans le domaine de POO, qui reconnait trop de choses ou fait trop de choses)*.
- Peut lui-même peut devenir exagérément complexe en l'absence de conception soigneuse

DEMARCHE A SUIVRE

Démarche à suivre

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'objet du médiateur 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’application GUI. Si l’on développe des interfaces utilisateur avec java, on applique probablement ce pattern.

CONCLUSION

- Le médiateur se concentrer sur l’interaction entre les composants GUI, et la classe d’application se concentrer sur la construction des composants.

- 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.

“Merci de votre attention”