

Année: 2020-2021



GLSI-B

Thème:

Patron de conception Mediateur

Présenté par: Ndoumbe TOURE



**PLAN**

# INTRODUCTION

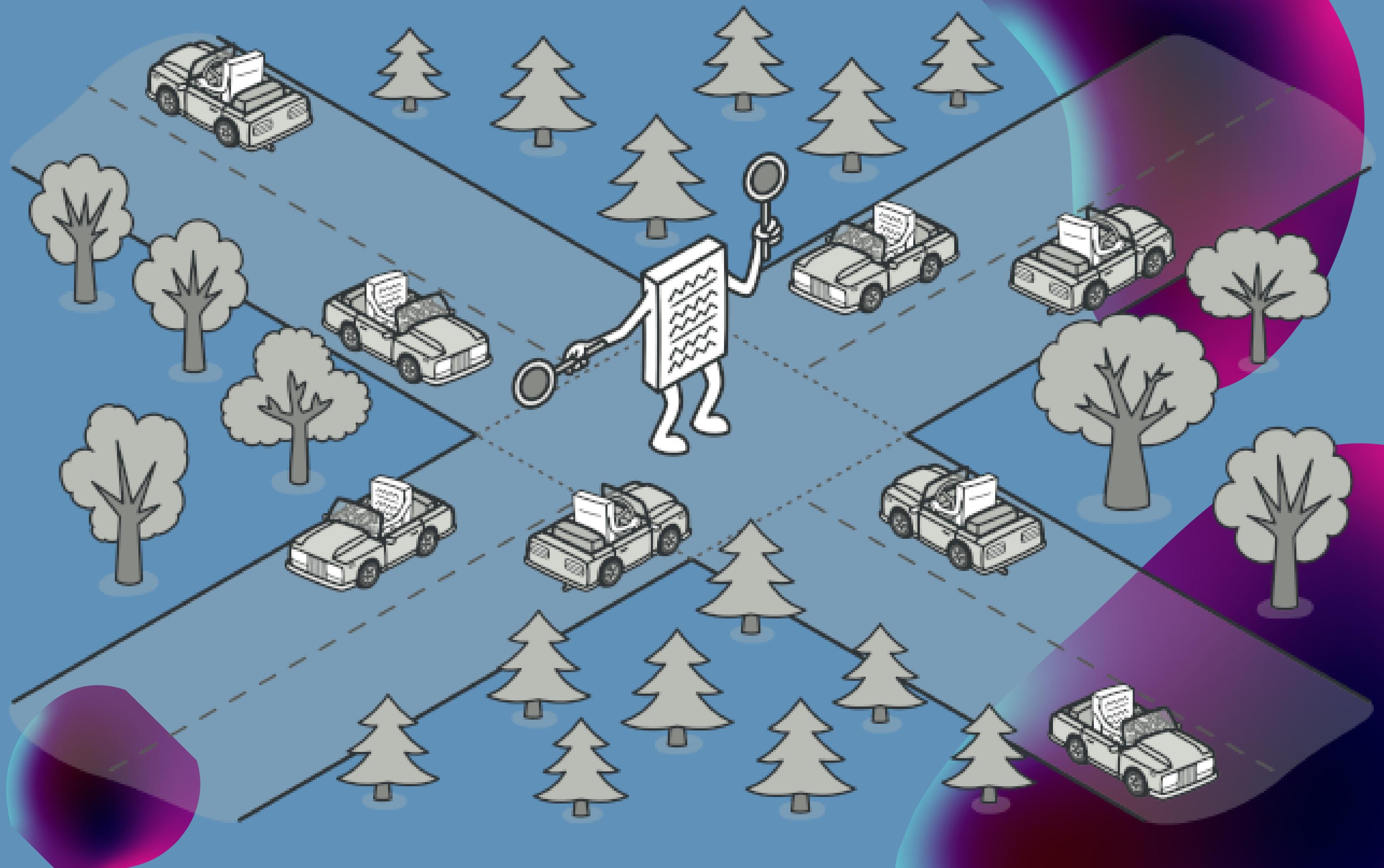
# CAS ILLUSTRATIF

# IMPLEMENTATION

# CONCLUSION

# INTRODUCTION

Médiateur est un patron de conception comportemental qui diminue les dépendances entre les objets en fournissant une interface unifiée pour un ensemble d'interfaces d'un sous-système . Il restreint les communications directes entre les objets et les force à collaborer uniquement via un objet médiateur.



# CAS ILLUSTRATIF

# Probleme

Supposons que dans une boîte de dialogue qui crée et modifie des profils client. Elle comporte plusieurs contrôles de formulaires comme des champs de texte, des cases à cocher, des boutons, etc.

Certains éléments du formulaire peuvent interagir.

En écrivant directement la logique dans le code des éléments du formulaire, vous rendez les classes de ces éléments bien plus difficiles à réutiliser dans l'application. Par exemple, vous ne pourrez pas utiliser la classe de la case à cocher dans un autre formulaire, car elle est couplée avec un champ de texte . Vous êtes par conséquent obligé d'utiliser toutes les classes du formulaire du profil si vous voulez vous servir de l'une d'entre elles.

## Solution

Le patron de conception médiateur vous propose de mettre fin à toutes les communications directes entre les composants et de rendre ces derniers indépendants les uns des autres. À la place, ces composants collaborent indirectement en utilisant un objet spécial médiateur qui redirige les appels vers les composants appropriés.

Ainsi, les composants ne reposent que sur une seule classe médiateur plutôt que d'être couplés à de nombreux collègues.

Dans notre exemple qui porte sur l'édition du formulaire d'un profil, la classe dialogue peut prendre le rôle du médiateur. Elle connaît déjà probablement tous ses sous-éléments, donc pas besoin d'y ajouter des dépendances.

# EXEMPLE D'IMPLEMENTATION

```
public class Discussion {  
    public static void voirMessage (User user, String message){  
        System.out.println( " [" + user.getName() + "] : " + message);  
    }  
}
```

La class Discussion va ici être notre médiateur. Elle comprend une méthode permettant de mettre en forme le message d'un utilisateur et de le faire apparaître .

```
public class User {  
    private String name;  
    public String getName() {  
        return name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    public User(String name){  
        this.name = name;  
    }  
    public void sendMessage(String message){  
        Discussion.voirMessage(this,message);  
    }  
}
```

Ici, on implémente le constructeur de User, et on lui attribue une fonction sendMessage qui la relie à notre médiateur,

```
public class Mediateur {  
    public static void main(String[] args) {  
        User mandy = new User("Mandy");  
        User raymond = new User("Raymond");  
        mandy.sendMessage("Salut! Raymond! pas de cours demain");  
        raymond.sendMessage("Salut Mr le RP! Message bien recu");  
    }  
}
```

Pour terminer on implémente la classe principale main. A l'intérieur, on définit deux utilisateurs User, Mandy et Raymond. Chacun d'eux envoient un message avec la commande sendMessage, définie dans le constructeur User. Mais cette commande passe par la Discussion, et c'est là que se crée le médiateur. Discussion est la seule entité à avoir accès aux messages de chacun. Elle les met en forme, et les affiche .

# CONCLUSION

Le pattern MEDIATOR 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. Vous pouvez appliquer MEDIATOR chaque fois que vous devez définir un objet qui encapsule la façon dont un ensemble d'objets interagissent.