Design Pattern

Mediator Pattern

Elisa Gougerot & Victor Deyanovitch & Taj Singh

Mode opératoire

———

Pour la réalisation de ce second exposé, nous avons globalement suivi le même processus que lors de notre premier exposé.

Nous avons donc commencé la création de notre exposé le week-end précédant le rendu via un Google Slide afin d’y déposer en vrac nos idées.

Nous avons ensuite construit le plan en s’appuyant sur celui de notre premier exposé puis nous avons fait la synthèse de nos recherches.

Enfin nous avons finalisé notre présentation avec un petit entraînement oral afin de déterminer les temps de parole de chacun :)

Mediator Pattern

- Design pattern comportemental permettant de réduire les dépendances entre objets

- Restreint les communications directes entre les objets en les obligeants à collaborer par l’intermédiaire d’un objet médiateur

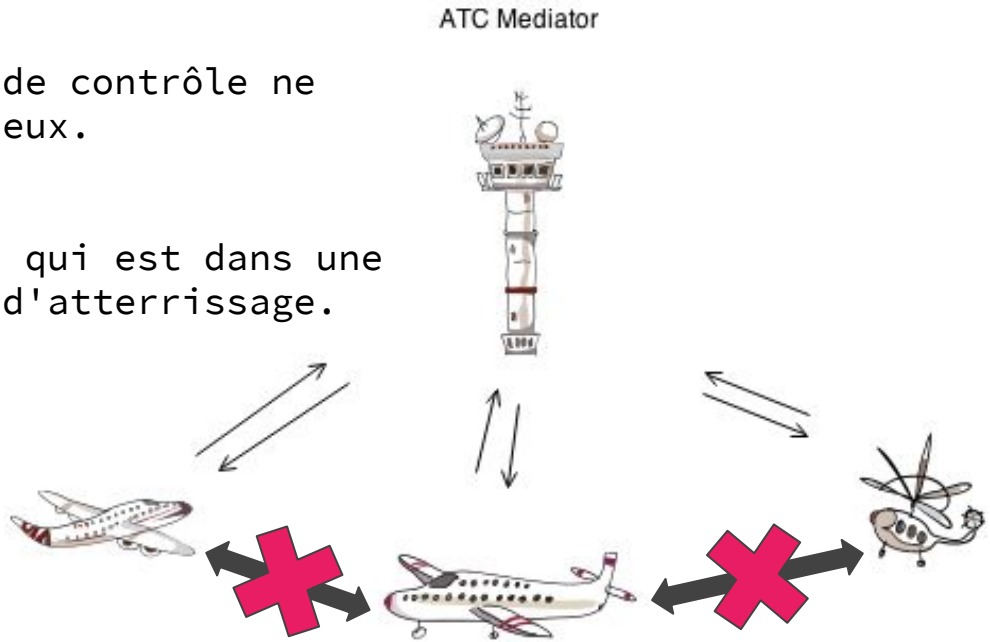
- Répondre aux problèmes courants de relations et comportements entre sous-programmes

- Permet la conception de composants réutilisables

La tour de contrôle

Les pilotes qui approchent la zone de contrôle ne communiquent pas directement entre eux.

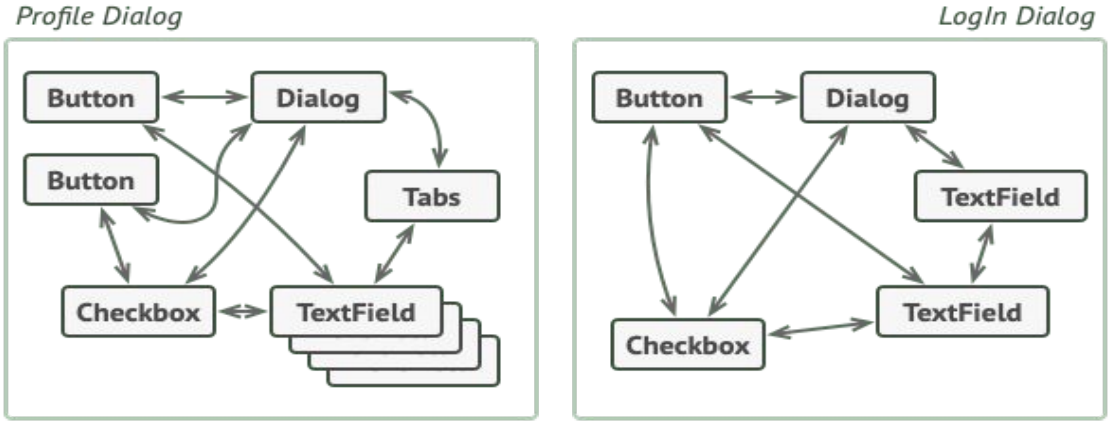
Ils parlent à un contrôleur aérien, qui est dans une tour quelque part près de la piste d'atterrissage.



Exemple d'usage : boîte de dialogue pour créer des profils

☹ Problem

Divers contrôles de formulaire : champs de texte, cases à cocher, boutons, etc

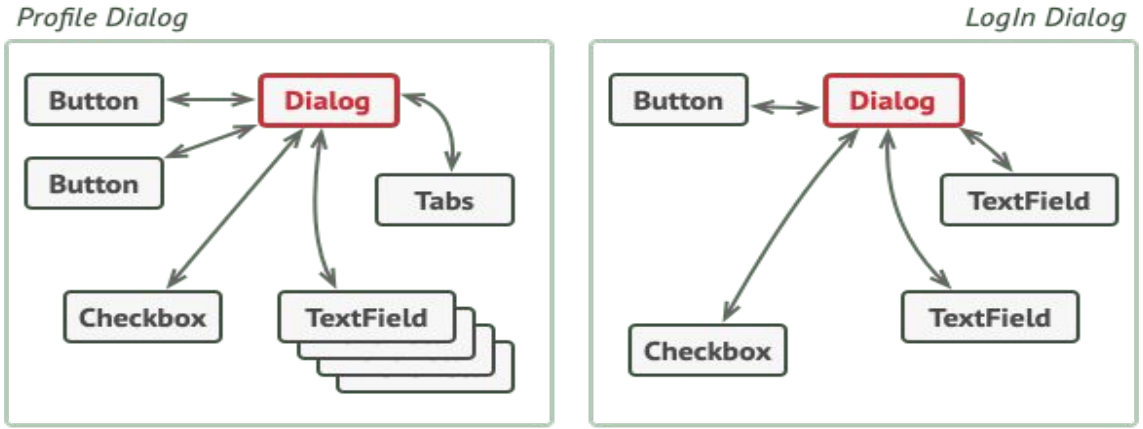


Les relations entre les éléments de l'interface utilisateur peuvent devenir chaotiques à mesure que l'application évolue

Exemple d'usage : boîte de dialogue pour créer des profils

😊 Solution

Le modèle Médiator permet d'encapsuler un réseau complexe de relations à l'intérieur d'un seul objet médiateur.



Moins une classe a de dépendances, plus il est facile de modifier, d'étendre ou de réutiliser cette classe.

Exemple de code

☹ Problem

```java
public class Button {
    private Fan fan;
    // constructor, getters and setters
    public void press(){
        if(fan.isOn()){
            fan.turnOff();
        } else {
            fan.turnOn();
        }
    }
}
```

```java
public class PowerSupplier {
    public void turnOn() {
        // implementation
    }

    public void turnOff() {
        // implementation
    }
}
```

```java
public class Fan {
    private Button button;
    private PowerSupplier powerSupplier;
    private boolean isOn = false;

    // constructor, getters and setters

    public void turnOn() {
        powerSupplier.turnOn();
        isOn = true;
    }

    public void turnOff() {
        isOn = false;
        powerSupplier.turnOff();
    }
}
```

Exemple de code

```
public class Mediator {
    private Button button;
    private Fan fan;
    private PowerSupplier powerSupplier;

    // constructor, getters and setters

    public void press() {
        if (fan.isOn()) {
            fan.turnOff();
        } else {
            fan.turnOn();
        }
    }

    public void start() {
        powerSupplier.turnOn();
    }

    public void stop() {
        powerSupplier.turnOff();
    }
}
```

😊 Solution

```
public class Fan {
    private Mediator mediator;
    private boolean isOn = false;

    // constructor, getters and setters

    public void turnOn() {
        mediator.start();
        isOn = true;
    }

    public void turnOff() {
        isOn = false;
        mediator.stop();
    }
}
```

Les conséquences de son utilisation

- Faciliter la maintenance et la compréhension du code
- Application d’un principe de la POO (Open/Closed), introduction possible de nouveau médiator sans avoir à modifier les anciens composants
- Réutilisation des composants de façon individuel

- Le médiator peut devenir un “objet divin”
- On cherche encore mais il n’y en a pas vraiment … :)

Mediator ≃ Observer

- La différence entre le Mediator et l’Observer est souvent insaisissable (dans la plupart des cas, vous pouvez appliquer l'un ou l'autre de ces modèles)

- L’objectif de Mediator est d'éliminer les dépendances mutuelles entre un ensemble de composantes du système, ces composants deviennent pour cela dépendants d'un seul objet : le médiator

- L'objectif d'Observer est d'établir des connexions dynamiques à sens unique entre les objets, où certains objets agissent comme des sous éléments d'autres

Sources

- https://refactoring.guru/design-patterns/mediator
- https://www.youtube.com/watch?time_continue=10&v=ZnbGf-8Q06E&feature=emb_logo
- https://sourcemaking.com/design_patterns/mediator
- https://www.baeldung.com/java-mediator-pattern