ÉTS
Le génie pour l'industrie

LOG121

Conception orientée objet

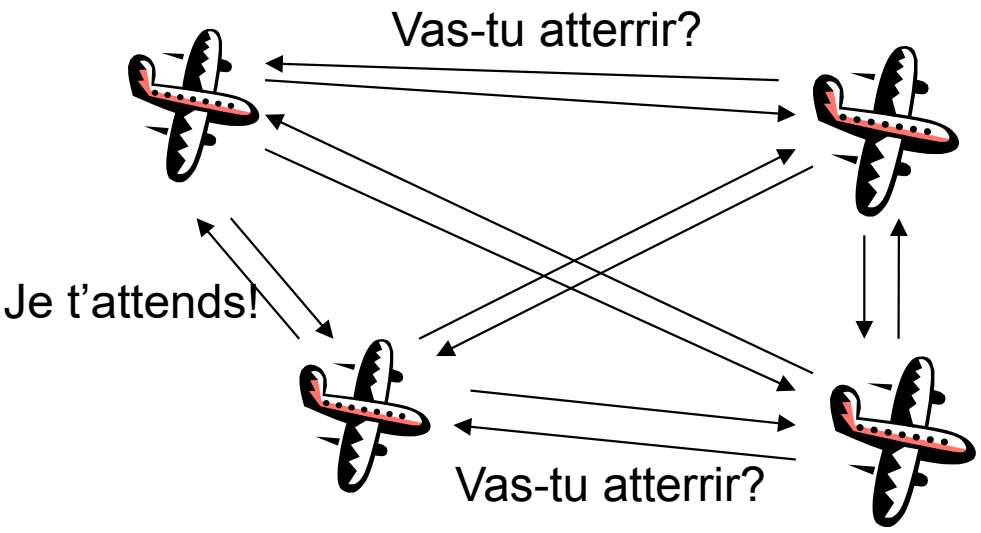
Patron Médiateur

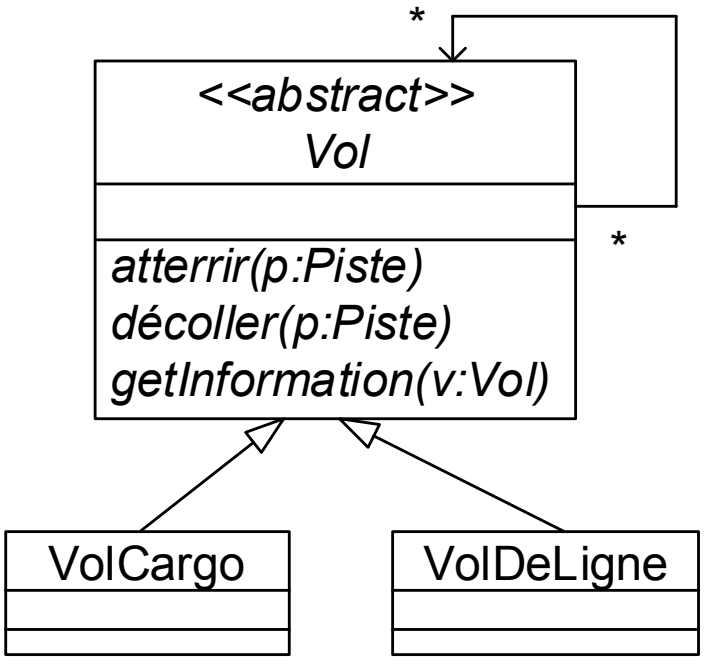
Enseignante: Souad Hadjres

Exemple d'interactions complexes

- Exemple du trafic aérien
  - Il y a de nombreux vols qui transitent par un aéroport
  - Les vols peuvent être des vols de ligne, des vols cargo, etc.
  - Les vols ne doivent pas entrer en collision
    - L'atterrissage et le décollage des avions doivent être synchronisés

Exemple d'interactions complexes



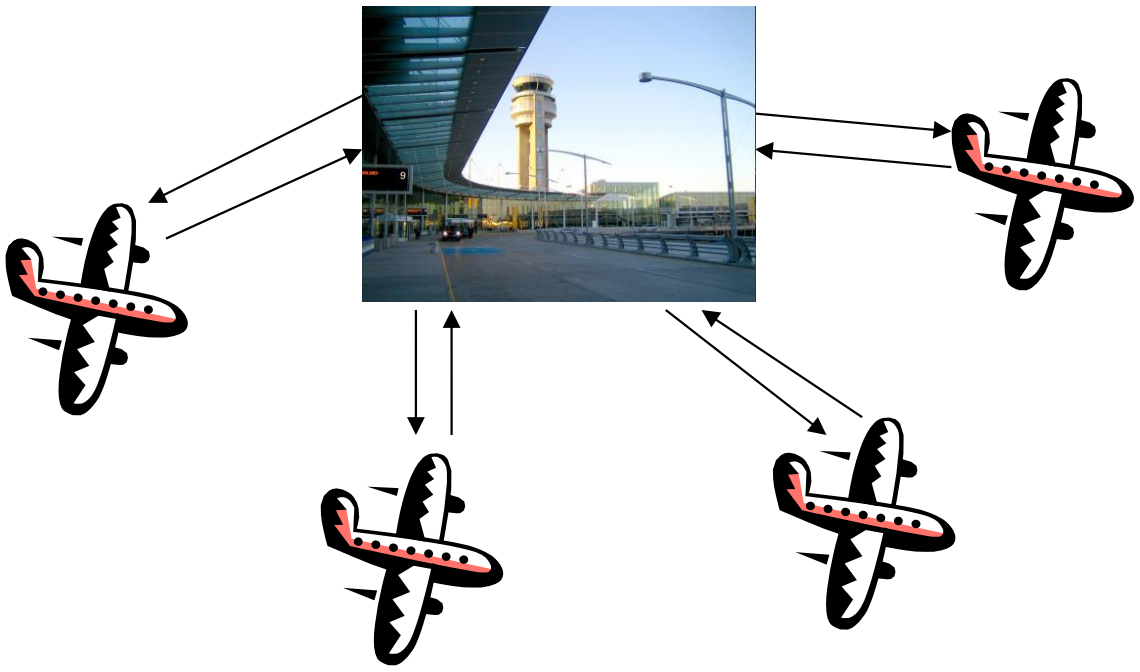


- Quel est le problème avec cette conception?
- Comment régler le problème?

Exemple d'interactions complexes

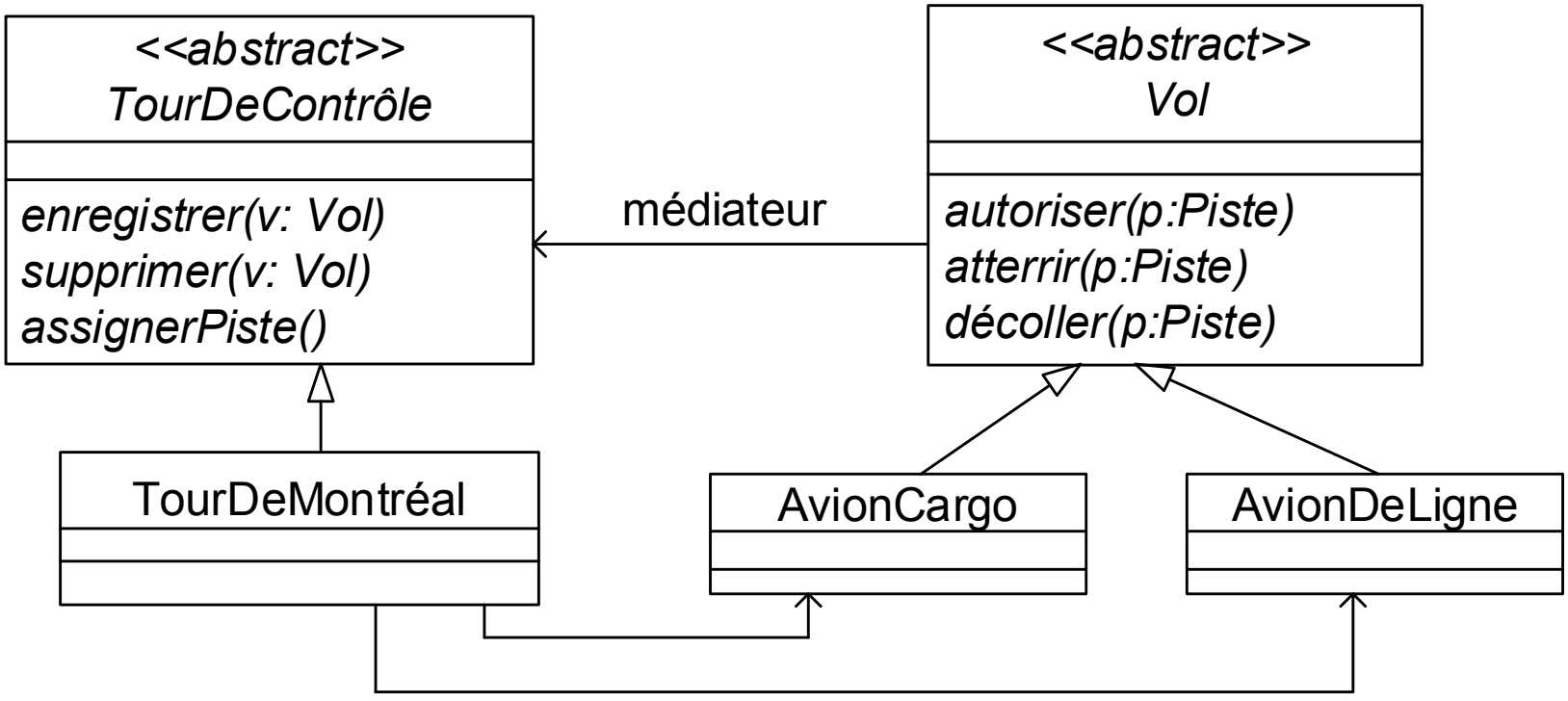
- Problèmes avec ma conception précédente
  - Trop de couplage
  - Selon la configuration de l'aéroport, les interactions entre les vols peuvent changer
    - Un aéroport peut avoir une ou plusieurs pistes
    - On doit changer l'implémentation des vols à chaque fois qu'on change d'aéroport!

Solution au problème



- Chaque vol communique avec la tour de contrôle.
- La tour de contrôle gère l'interaction entre les vols.

Solution au problème



- Un vol communique son information à la tour de contrôle.
- La tour de contrôle utilise cette information pour gérer les atterrissages et les décollages.
- La tour de contrôle envoie des autorisations aux vols.

```java
public abstract class TourDeControle {

    public abstract void enregistrer(Vol v);
    public abstract void supprimer(Vol v);
    public abstract void assignerPiste();

}
```

```java
public abstract class Vol {

    public abstract void autoriser(Piste p);
    public abstract void atterir(Piste p);
    public abstract void decoler(Piste p);

}
```

```java
public class VolDeLigne extends Vol {
    private TourDeControle aeroportActuel;
    boolean PretPourAtterissage;
    boolean PretPourDecolage;
    public VolDeLigne(TourDeControle _a, boolean _att, boolean _decol){
        aeroportActuel = _a;
        PretPourAtterissage = _att;
        PretPourDecolage = _decol;
    }
    public void atterir(Piste p) {
        System.out.println("Atterissage en cours sur la piste assignée...");
        try {
            TimeUnit.MINUTES.sleep(1);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
      }
    public void decoler(Piste p) {
        System.out.println("Decolage en cours sur la piste assignée...");
        try {
            TimeUnit.MINUTES.sleep(1);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public void autoriser(Piste p) {
        if (PretPourAtterissage) {
            atterir(p);
        } else if(PretPourDecolage) {
            decoler(p);
        }
    }
}
```

```java
import java.util.*;

public class TourDeMontreal extends TourDeControle {

    private Piste piste = new Piste();
    private Queue<Vol> fileAttente = new LinkedList<Vol>();

    public void enregistrer(Vol v) {
        fileAttente.add(v);
    }

    public void supprimer(Vol v) {
        fileAttente.remove(v);
    }

    public void assignerPiste() {
        Vol v = fileAttente.peek();
        v.autoriser(piste);
    }
```

```java
public class Gestionnaire {
    public static void main(String args[]) {
        //On crée le médiateur
        TourDeControle tour1 = new TourDeMontreal();

        //on crée les vols en leur passant le médiateur en paramètre
        //création d'un vol qui veut atterir
        Vol vol1 = new VolDeLigne(tour1, true, false);

        //le vol s'enregistre dans la file d'attente
        tour1.enregistrer(vol1);

        //création d'un vol qui veut decoler
        Vol vol2 = new VolDeLigne(tour1, false, true);
        tour1.enregistrer(vol2);

        tour1.assignerPiste();
        tour1.supprimer(vol1);

        tour1.assignerPiste();
        tour1.supprimer(vol2);
    }
}
```

une classe qui utilise le tout

Solution au problème


```java
public abstract class TourDeControle {

    public abstract void enregistrer(Vol v);
    public abstract void supprimer(Vol v);
    public abstract void assignerPiste();

}
```

```java
public abstract class Vol {

    public abstract void autoriser(Piste p);
    public abstract void atterir(Piste p);
    public abstract void decoler(Piste p);

}
```

```java
import java.util.*;

public class TourDeMontreal extends TourDeControle {

    private Piste piste = new Piste();
    private Queue<Vol> fileAttente = new LinkedList<Vol>();

    public void enregistrer(Vol v) {
        fileAttente.add(v);
    }

    public void supprimer(Vol v) {
        fileAttente.remove(v);
    }

    public void assignerPiste() {
        Vol v = fileAttente.peek();
        v.autoriser(piste);
    }
```

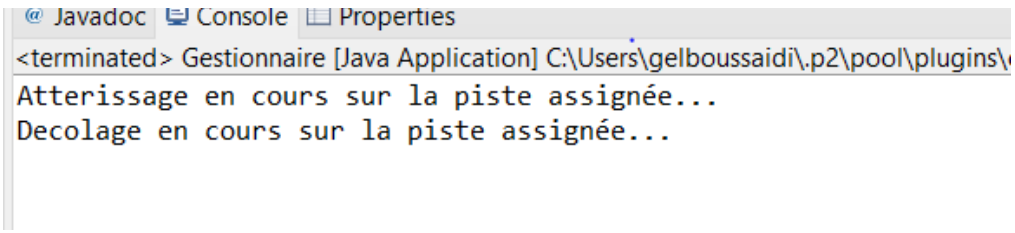
Solution au problème


```java
public class VolDeLigne extends Vol {
    private TourDeControle aeroportActuel;
    boolean PretPourAtterissage;
    boolean PretPourDecolage;
    public VolDeLigne(TourDeControle _a, boolean _att, boolean _decol){
        aeroportActuel = _a;
        PretPourAtterissage = _att;
        PretPourDecolage = _decol;
    }
    public void atterir(Piste p) {
        System.out.println("Atterissage en cours sur la piste assignée...");
        try {
            TimeUnit.MINUTES.sleep(1);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
      }
    public void decoler(Piste p) {
        System.out.println("Decolage en cours sur la piste assignée...");
        try {
            TimeUnit.MINUTES.sleep(1);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    public void autoriser(Piste p) {
        if (PretPourAtterissage) {
            atterir(p);
        } else if(PretPourDecolage) {
            decoler(p);
        }
    }
}
```

Solution au problème

Une classe qui utilise le tout.

```
public class Gestionnaire {
    public static void main(String args[]) {
        //On crée le médiateur
        TourDeControle tour1 = new TourDeMontreal();

        //on crée les vols en leur passant le médiateur en paramètre
        //création d'un vol qui veut atterir
        Vol vol1 = new VolDeLigne(tour1, true, false);

        //le vol s'enregistre dans la file d'attente
        tour1.enregistrer(vol1);

        //création d'un vol qui veut decoler
        Vol vol2 = new VolDeLigne(tour1, false, true);
        tour1.enregistrer(vol2);

        tour1.assignerPiste();
        tour1.supprimer(vol1);

        tour1.assignerPiste();
        tour1.supprimer(vol2);
    }
}
```



Le patron Médiateur

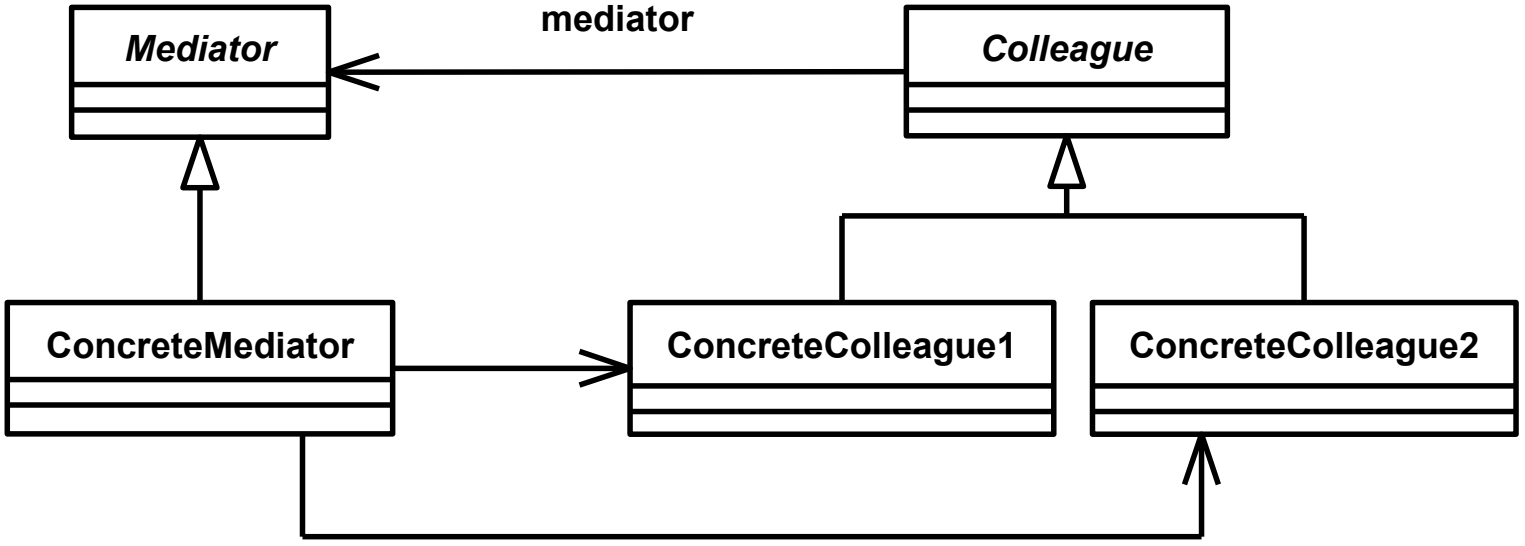
- Contexte
  - On a un ensemble d'objets qui interagissent d'une manière complexe mais bien définie. Cela peut aboutir à des dépendances non structurées et difficiles à comprendre.
  - La réutilisation d'un objet est difficile car il est couplé à beaucoup d'autres objets.
  - On a un comportement qui est distribué entre plusieurs objets et qu'on aimerait personnaliser.

Le patron Médiateur

- Solution
  - Définir une interface (Mediator) de communication entre les objets (Colleague) interagissant.
  - Définir un objet (ConcreteMediator) qui implémente concrètement cette interaction en coordonnant les objets Colleague. ConcreteMediator connaît ses objets Colleague.
  - Chaque objet Colleague connaît son médiateur et il communique avec lui.

Le patron Médiateur

- La structure du patron dans GoF



Nom dans le patron	Nom dans l'exemple du trafic aérien
Mediator	Tour_De_Contrôle
ConcreteMediator	Tour_De_Montréal
Colleague	vol
ConcreteColleague	VolDeLigne ou VolCargo

Le patron Médiateur

- Pouvez-vous citer des exemples de situations où le patron médiateur peut s'appliquer?
  - Le patron médiateur est souvent utilisé pour coordonner l'interaction entre les composants d'une interface (GUI).
  - On peut utiliser le médiateur pour implémenter un forum de discussion.

Le patron Médiateur

- Quelles sont les conséquences du patron médiateur?
  - Limiter le besoin d'étendre plusieurs classes: pour changer l'interaction entre les objets de type Colleague, on a besoin d'étendre le médiateur.
  - Le médiateur permet de découpler les objets de type Colleague. On peut varier et réutiliser les classes Colleague et Mediator de façon indépendante.
  - Les interactions sont plus faciles à comprendre, à maintenir et à étendre: le médiateur remplace les interactions *plusieurs-à plusieurs* par des interactions *un-à-plusieurs*.
  - Le médiateur centralise le contrôle: le médiateur peut devenir trop complexe et donc difficile à maintenir.

Question

- Est-ce qu'on aurait pu régler le problème de gestion de trafic aérien autour de l'aéroport en utilisant le patron Observateur?
  - Non, il y aurait un nombre trop élevé d' objets observables et d'objets observateurs. De plus les objets observateurs seront aussi des objets observables.