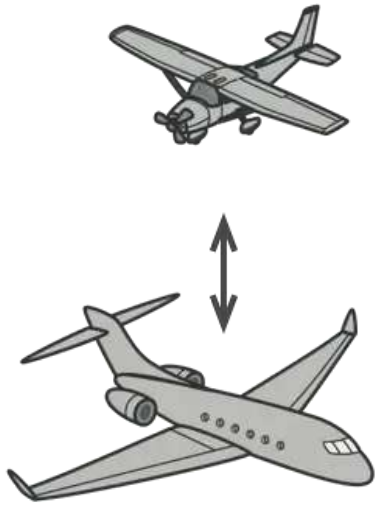
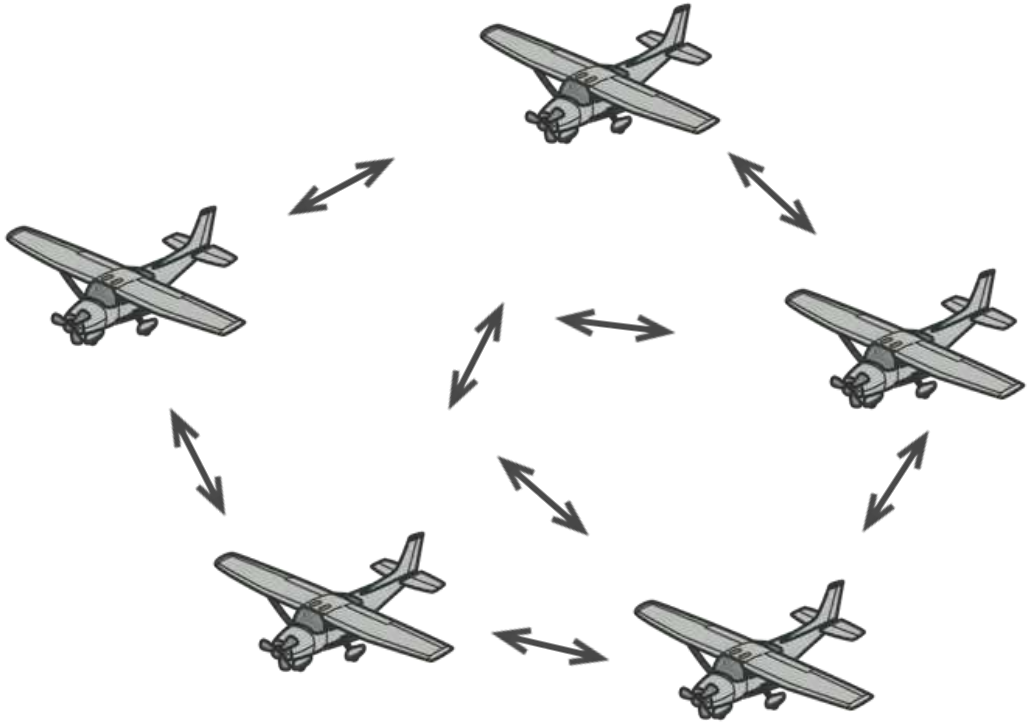
Mediator

Ondřej Tichavský

Motivace

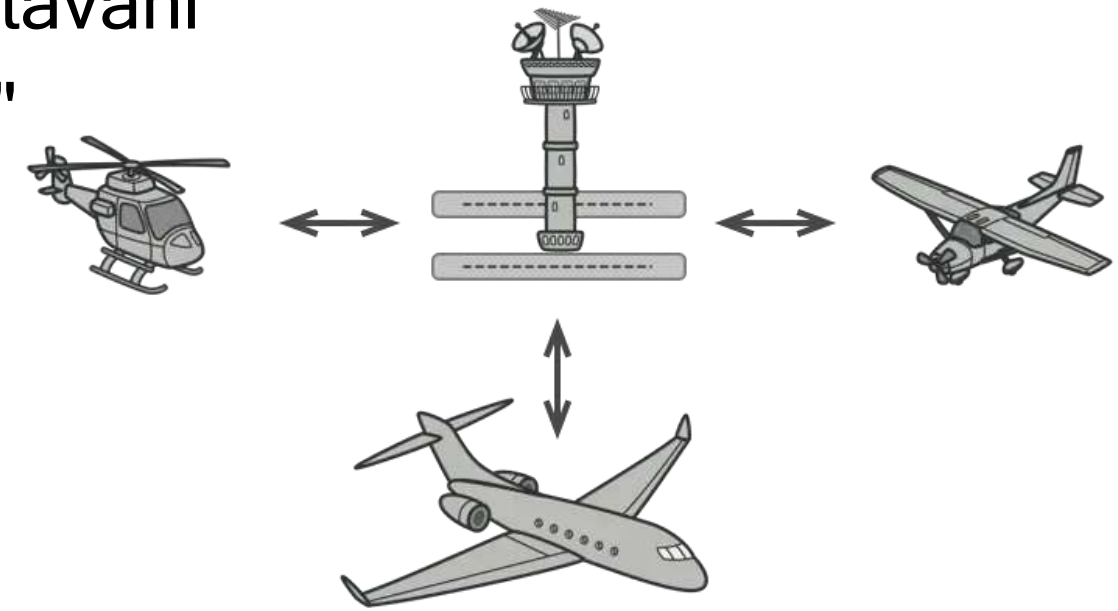
- Komunikace mezi latadly při snaze přistát na letišti
- Domluva, kdy, kde, kdo přistane





Řešní s mediatorem

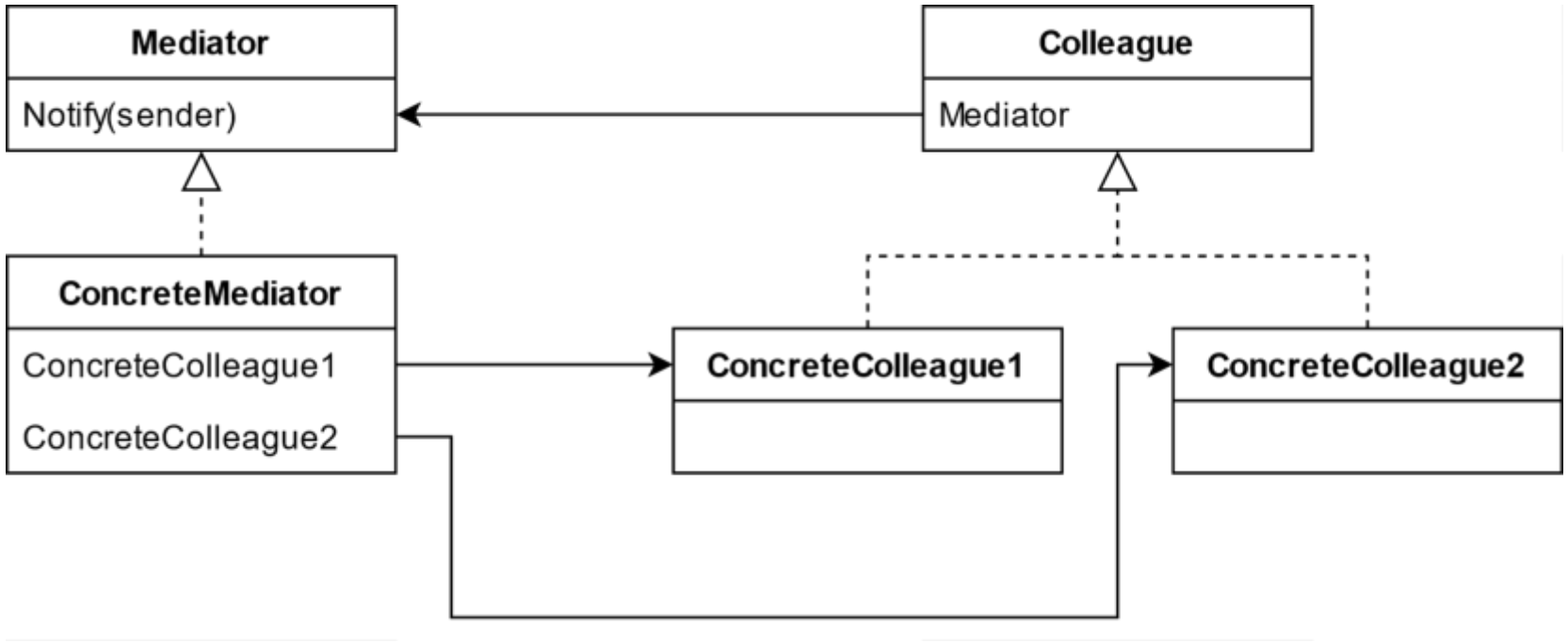
- Komnikace přes řídící věž
- Letadla neřeší scheduling přistávání
- Letadla o sobě "nemusí vědět"



Mediator - Definice

- Behaviorální návrhový vzor
- Interakce mezi objekty je zapouzdřena pomocí objektu Mediator
- Objekty nekomnikují přímo, ale skrze Mediator
- Objekty o sobě navzájem neví

Struktura



- Colleague
  - komunikující entity, neví o sobě navzájem, pouze o Mediatoru
- Abstract Mediator
  - Umožňuje změnu chování náhradou konkrétního
- Concrete Mediator
  - Implementuje chování

Příklad implementace - Chatroom

```
public abstract class Colleague {
    private Mediator chatRoom;
    public String name;
    public abstract void receiveMessage(String message);
}

public interface class Mediator {
    void sendMessage(String message,
         Colleague colleague, String messageType);
}
```

Concrete collegue

```
public class ConcreteColleague1 extends Colleague {
    public ConcreteColleague1(Mediator mediator,
            String name) {
        this.chatRoom = mediator;
        this.name = name;
    }

    public void sendMessage(String message) {
        mediator.sendMessage(message, this, "toAll");
    }

    @Override
    public void receiveMessage(String message) {
        println("Colleague1 received message: " + message);
    }
}
```

```
public abstract class Colleague {
    private Mediator chatRoom;
    public String name;
    public abstract void receiveMessage(String message);
}

public interface class Mediator {
    void sendMessage(String message,
        Colleague colleague, String messageType);
}
```

Concrete Mediator - Chatroom

```
public class ConcreteMediator implements Mediator {
    public Colleague alena { get; set; }
    public Colleague jan { get; set; }
    public Colleague tom { get; set; }

    @Override
    public void sendMessage(String message,
              Colleague colleague) {
        alena.receiveMessage(message);
        jan.receiveMessage(message);
        tom.receiveMessage(message);
    }
}
```

```
public abstract class Colleague {
    private Mediator chatRoom;
    public String name;
    public abstract void receiveMessage(String message);
}

public interface class Mediator {
    void sendMessage(String message,
         Colleague colleague, String messageType);
}
```

```
public static void main(String[] args) {
    var mediator = new ConcreteMediator();
    var alena = new ConcreteColleague1(mediator, "Alena");
    var jan = new ConcreteColleague2(mediator, "Jan");
    var tom = new ConcreteColleague3(mediator, "Tom");

    mediator.alena = alena;
    mediator.jan = jan;
    mediator.tom = tom;

    alena.sendMessage("Hello, how are you?");
    jan.sendMessage("I am fine, thank you!");
}
```

Změna v mediatoru – pomocí nové třídy

```
public class ConcreteMediator2 implements Mediator {
    public Colleague alena { get; set; }
    public Colleague jan { get; set; }
    public Colleague tom { get; set; }

    @Override
    public void sendMessage(String message,
             Colleague colleague, String messageType) {
        if (!messageType.equals("toAll")
              return;

        if (colleague == alena) {
            jan.receiveMessage(message);
            tom.receiveMessage(message);
        }
    }
}
```

Mediator as publisher

```
public class ConcreteMediator implements Mediator {
    private Subscribers subscribers = new Subscribers();

    @Override
    public void sendMessage(String message,
            Colleague colleague, String messageType) {
        subscribers.forEach(subscriber -> {
            if (subscriber != colleague) {
                subscriber.receiveMessage(message);
            }
        });
    }

    public void addSubscriber(Colleague colleague) {
        subscribers.add(colleague);
    }
}
```

Vhodné použít... když je

- dobře definovaný, ale složitý způsob komunikace.
  - S vysokou provázaností
- složité znovu použít objekt kvůli vysoké provázanosti.
- pro příjemce důležitá bezpečnost při komunikaci.

- chtěné customizovat chování, které je rozprostřeno mezi několik tříd.

Klady

- Změna chování vyžaduje změnu pouze v Mediatoru
- Zvýšení koheze
  - Oddělení interakce mezi objekty
- Volnější coupling
  - Snižuje provázanost tříd
- Jednodušší znovupoužitelnost tříd
- Zjednodušení komunikace
  - m:n -> 1:n

Zápory

- Negativní vliv na performance
- Nebezpečí vzniku příliš komplexního Mediatoru

Kde se lze setkat

- GUI interakce
- chatrooms
- Send/receive protokoly, JMS Java message service

Souvislosti s dalšími navrhovými vzory

- Façade
  - Definuje zjednodušený interface pro komunikaci
  - Upravuje vztahy mezi provázanými objekty
- Observer
  - Běžně používán pro komunikaci mezi mediátorem a kolegy

Zdroje

- https://refactoring.guru/design-patterns/mediator
- GoF Design Patterns, Benneth Christiansson (Ed.) Mattias Forss, 2008
- https://www.digitalocean.com/community/tutorials/mediator-design-pattern-java
- https://stackoverflow.com/questions/41266495/what-is-an-example-in-the-real-world-that-uses-the-mediator-pattern