Le pattern Command

Jules DUFOSSÉ
Mathis BREMONT
Lucas MAURI
Gautier MECHAUSSIE

iut du limousin
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
Université de Limoges

Introduction : qu'est ce qu'un Design Pattern ?

- Solution utilisée avec un langage orienté objet (ex : c++)
- Bonne pratique et modèle à suivre
- Éviter les erreurs de programmation
- Il existe 3 types de Design Pattern :
    - Création : organise les classes, exemple pattern Composite
    - Structure : crée et configure des objets, exemple pattern Builder
    - Comportemental : bonne communication entre les objets, exemple pattern Strategy

Plan

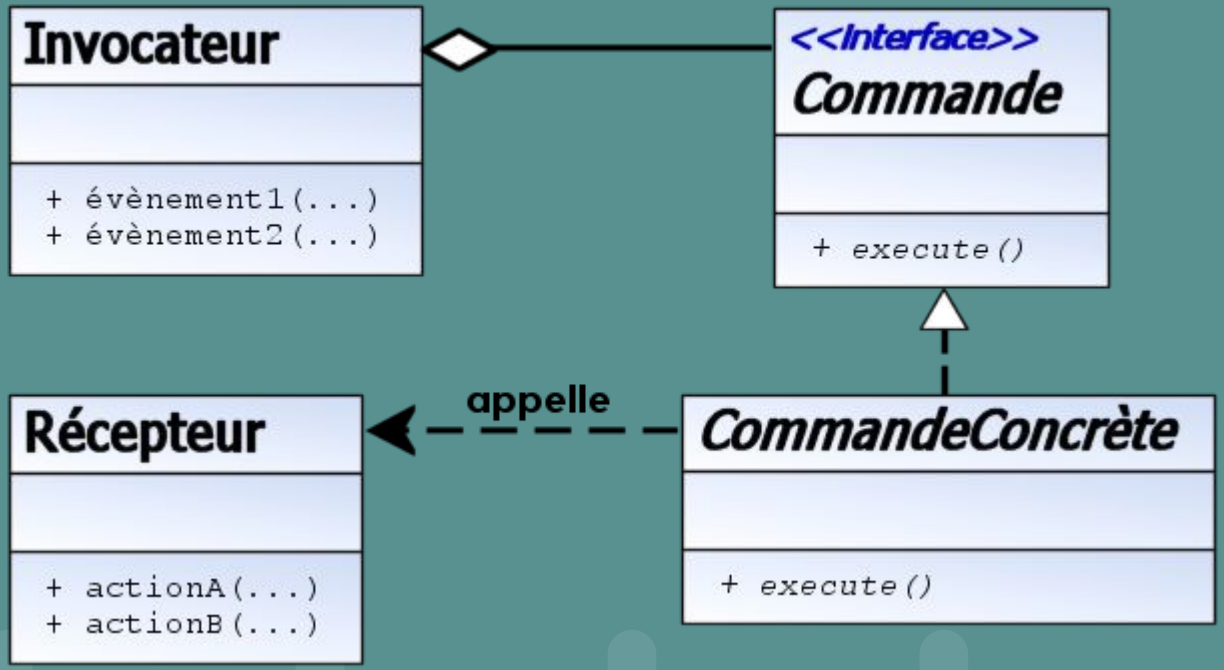
Problématique : Quels sont les raisons qui poussent à utiliser le pattern Command ?

1. L’objectif du pattern Command
2. Le fonctionnement général du pattern
3. Rappel : SOLID
4. Exemple d’utilisation du pattern Command

Utilité et objectifs du pattern

- Encapsuler la logique métier d’un objet derrière une interface standardisée
- Apporter des modifications aux commandes sans avoir à tout changer
- Tracer une requête
- Enregistrer des séquences de commandes
- Méthodes “Undo” et “Rollback”
- Établir des barres de progression
- Les interfaces graphiques (bouton et menu)

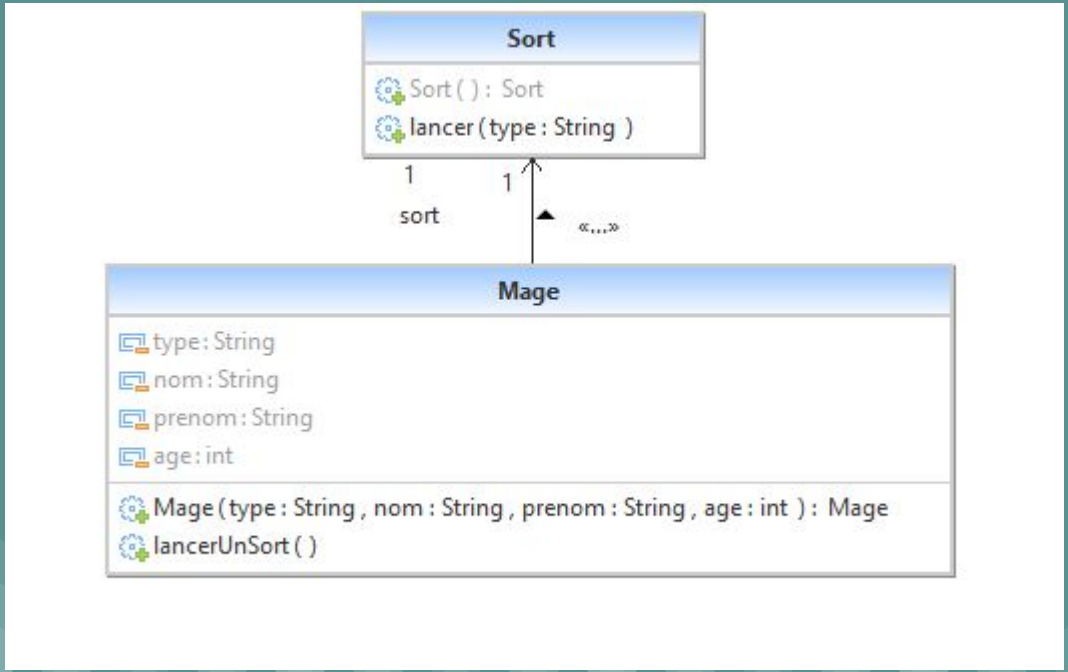
Fonctionnement du pattern



Rappel du principe SOLID :

- **S**ingle responsibility
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface segregation Principle
- **D**ependency Inversion Principle

Diagramme de classe :



La classe Mage :

```java
public class Mage {
    private String type;
    private String nom;
    private String prenom;
    private int age;
    private Sort sort;

    public Mage(String type, String nom, String prenom, int age) {
        this.type = type;
        this.nom = nom;
        this.prenom = prenom;
        this.age = age;
        this.sort = new Sort();
    }

    public void lancerUnSort() {
        if (this.type == "Glace") {
            this.sort.lancer("Glace");
        }
        if (this.type == "Feu") {
            this.sort.lancer("Feu");
        }
        if (this.type == "Terre") {
            this.sort.lancer("Terre");
        }
        if (this.type == "Air") {
            this.sort.lancer("Air");
        }

    }
}
```

La classe sort :

```
public class Sort {
    public Sort() {
    }

    public void lancer(String type) {
        System.out.println("Sortilège de type : \""+type+"\" a été lancé ");
    }

}
```

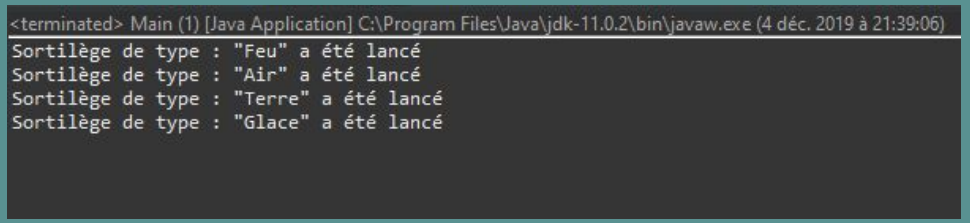
Un petit scénario :

```
public class Main {

    public static void main(String[] args) {
        Mage gandalf = new Mage("Feu","Gandalf","Le Gris",24000);
        gandalf.lancerUnSort();
        gandalf.setType("Air");
        gandalf.lancerUnSort();
        gandalf.setType("Terre");
        gandalf.lancerUnSort();
        gandalf.setType("Glace");
        gandalf.lancerUnSort();
    }

}
```

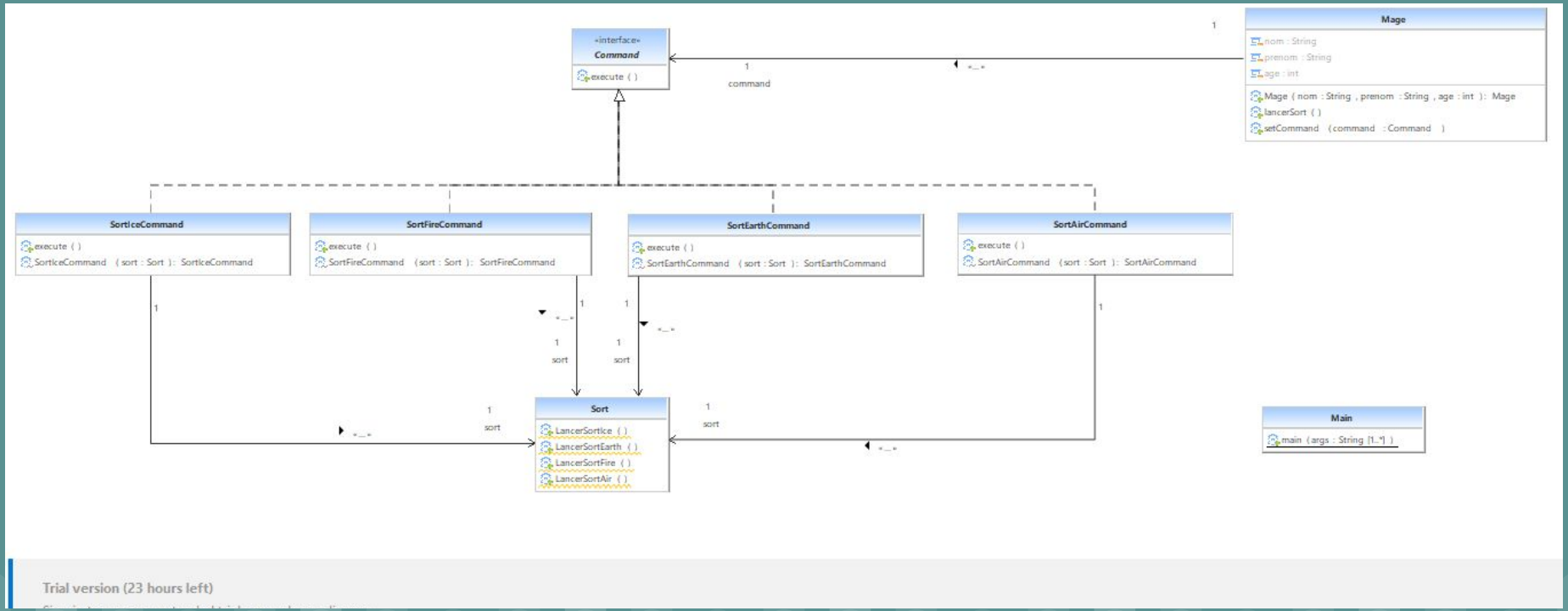
Le résultat :


<terminated> Main (1) [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (4 déc. 2019 à 21:39:06)
Sortilège de type : "Feu" a été lancé
Sortilège de type : "Air" a été lancé
Sortilège de type : "Terre" a été lancé
Sortilège de type : "Glace" a été lancé

Le changement :

```
public void lancerUnSort() {
    if (this.getType() == "Glace") {
        this.sort.lancer("Glace");
    }
    if (this.getType() == "Feu") {
        this.sort.lancer("Feu");
    }
    if (this.getType() == "Terre") {
        this.sort.lancer("Terre");
    }
    if (this.getType() == "Air") {
        this.sort.lancer("Air");
    }
    if (this.getType() == "Metal") {
        this.sort.lancer("Metal");
    }
}
```

Diagramme de classe:



Interface Command

```
public interface Command {
    public void execute();
}
```

Classe SortAirCommand

```
public class SortAirCommand implements Command{

    private Sort sort;

    SortAirCommand(Sort sort){
        this.sort= sort;
    }

    public void execute() { this.sort.LancerSortAir(); }
}
```

Classe SortFireCommand

```
public class SortFireCommand implements Command{

    private Sort sort;

    SortFireCommand(Sort sort){
        this.sort= sort;
    }

    public void execute() {
        this.sort.LancerSortFire();
    }
}
```

Classe SortEarthCommand

```
public class SortEarthCommand implements Command{

    private Sort sort;

    SortEarthCommand(Sort sort){
        this.sort= sort;
    }

    public void execute() {
        this.sort.LancerSortEarth();
    }
}
```

Classe SortIceCommand

```
public class SortIceCommand implements Command{

    private Sort sort;

    SortIceCommand(Sort sort){
        this.sort= sort;
    }

    public void execute() {
        this.sort.LancerSortIce();
    }
}
```

Classe Sort

```java
public class Sort {



    public void LancerSortIce(){
        System.out.println("Sortilege de type : \"glace\" a été lancé");
    };
    public void LancerSortEarth(){
        System.out.println("Sortilege de type : \"terre\" a été lancé");
    };

    public void LancerSortFire(){
        System.out.println("Sortilege de type : \"feu\" a été lancé");
    };

    public void LancerSortAir(){
        System.out.println("Sortilege de type : \"air\" a été lancé");
    };
}
```

Classe Mage

```java
public class Mage{

    private String nom;
    private String prenom;
    private int age;

    public Mage(String nom, String prenom, int age) {
        this.nom=nom;
        this.prenom=prenom;
        this.age=age;
    }

    private Command command;

    public void setCommand(Command command){
        this.command = command;
    }

    public void lancerSort(){
        command.execute();
    }
}
```

Classe Main

```java
public class Main {
    public static void main(String[] args){
        Mage gandalf = new Mage( nom: "Leblanc", prenom: "Gandalf", age: 24000);
        Sort sortilege = new Sort();

        Command sortFeu = new SortFireCommand(sortilege);
        gandalf.setCommand(sortFeu);
        gandalf.lancerSort();

        Command sortAir = new SortAirCommand(sortilege);
        gandalf.setCommand(sortAir);
        gandalf.lancerSort();

        Command sortEarth = new SortEarthCommand(sortilege);
        gandalf.setCommand(sortEarth);
        gandalf.lancerSort();

        Command sortIce = new SortIceCommand(sortilege);
        gandalf.setCommand(sortIce);
        gandalf.lancerSort();

    }
}
```

Résultats

```
Sortilege de type : "feu" a été lancé
Sortilege de type : "air" a été lancé
Sortilege de type : "terre" a été lancé
Sortilege de type : "glace" a été lancé
```

Conclusion et Sources

https://tech.io/playgrounds/36502/design-pattern-command/le-pattern-command

https://github.com/leger50/DesignPattern_Command/blob/master/presentation_pattern_command_m3105.pdf

https://fr.wikibooks.org/wiki/Patrons_de_conception/Commande

https://fr.wikipedia.org/wiki/Commande_(patron_de_conception)

https://www.codingame.com/playgrounds/36502/design-pattern-command/le-pattern-comma