DESIGN PATTERN

COMMAND PATTERN

LENOIR Quentin MOUSSET Nathan DECOMBE Sylvain

SOMMAIRE

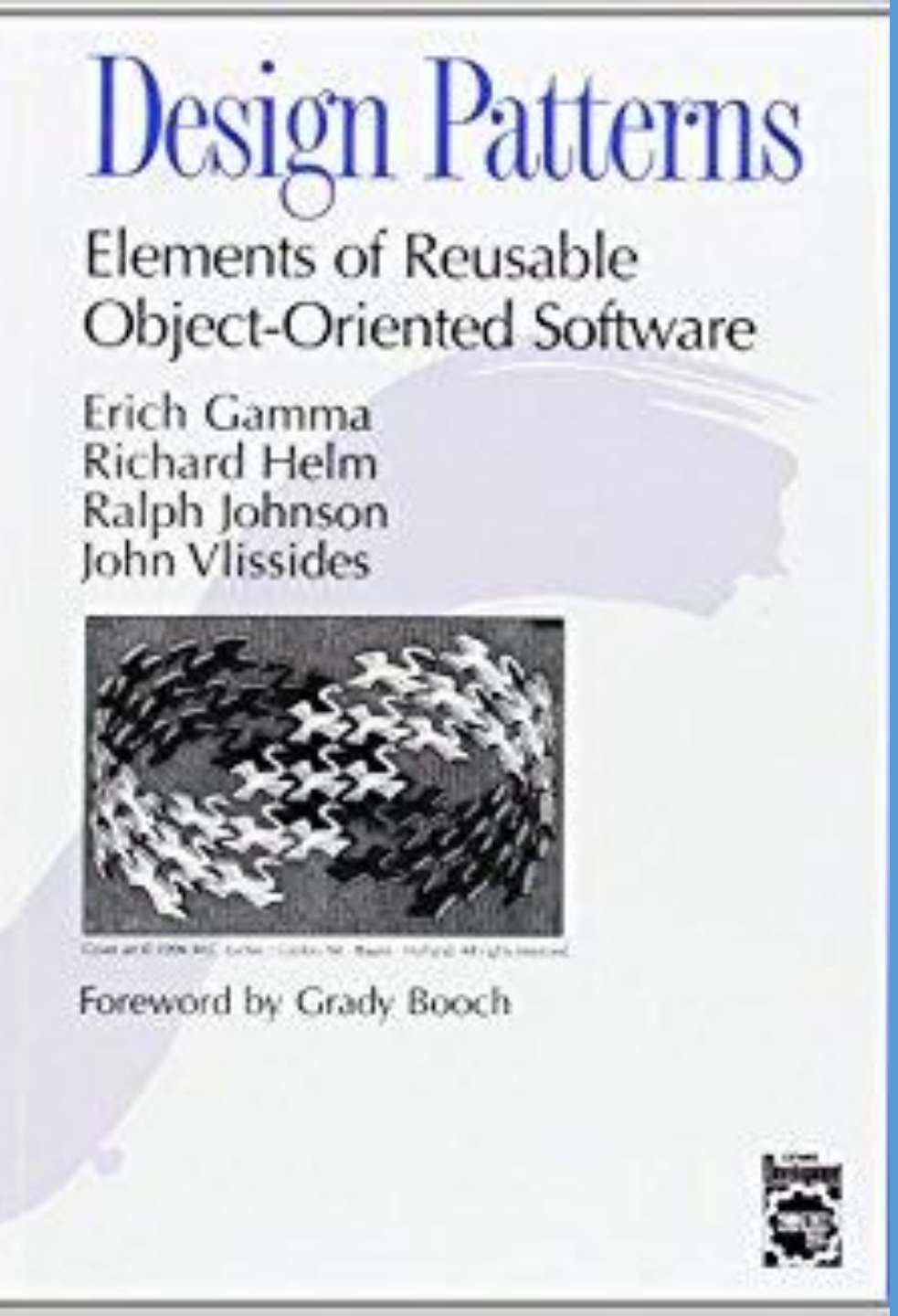
DESIGN PATTERN ?

COMMAND PATTERN ?

EXEMPLE

TUTORIEL

QCM



DESIGN PATTERN

- Solution fréquente pour un problème récurrent
- Regroupe un ensemble de pattern différents
- Améliorent la qualité de développement et diminuent la durée.

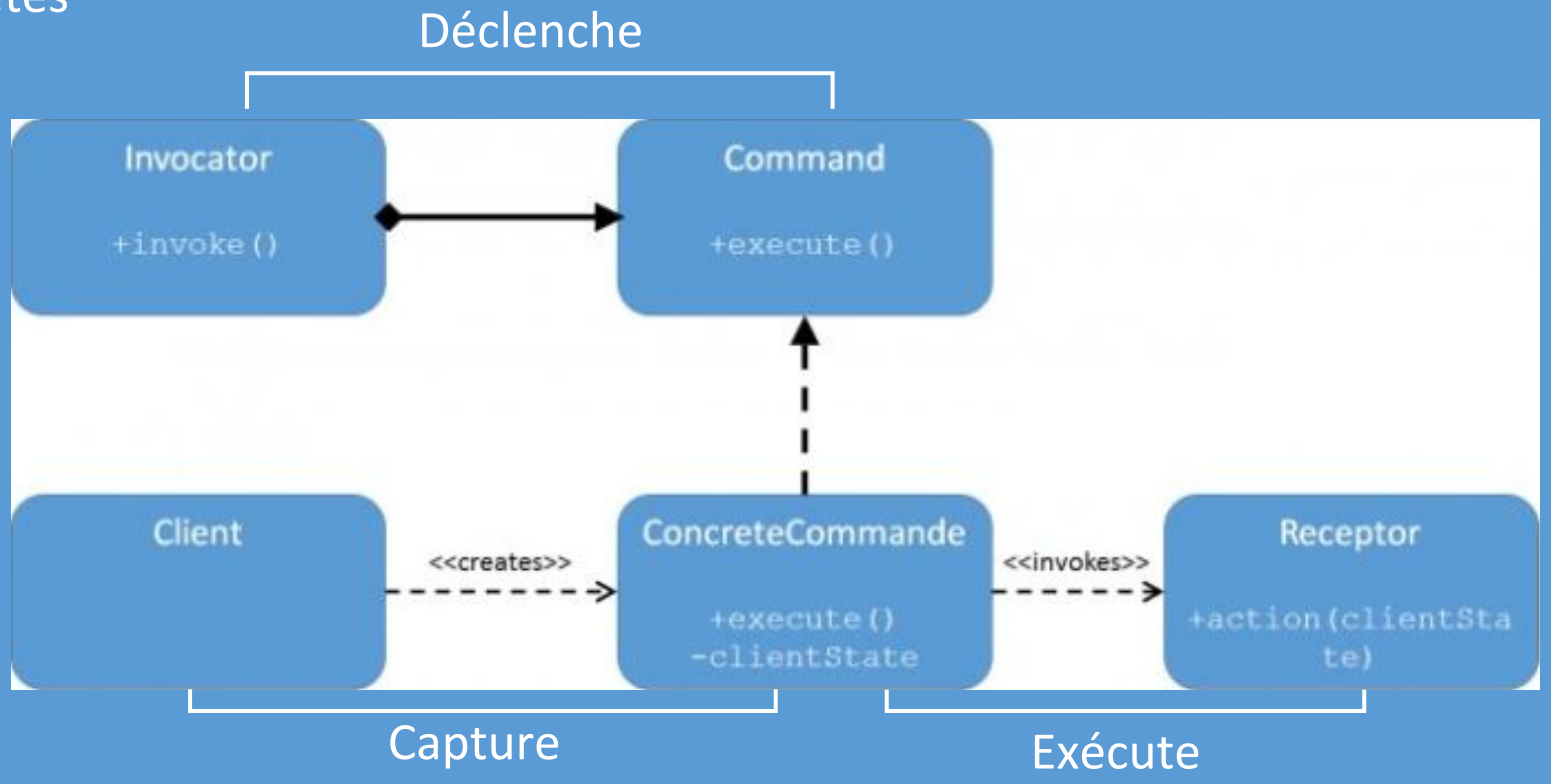
Patrons de création

Patrons de structure

Patrons comportementaux

COMMAND PATTERN

- Pattern comportemental
- Encapsuler une requête sous la forme d'un objet
- Paramétrer facilement des requêtes diverses
- Permettent des opérations réversibles



UTILITE ?

- Il y a prolifération de méthodes similaires, et que le code de l'interface devient difficile à maintenir.
  - Raccourcis clavier

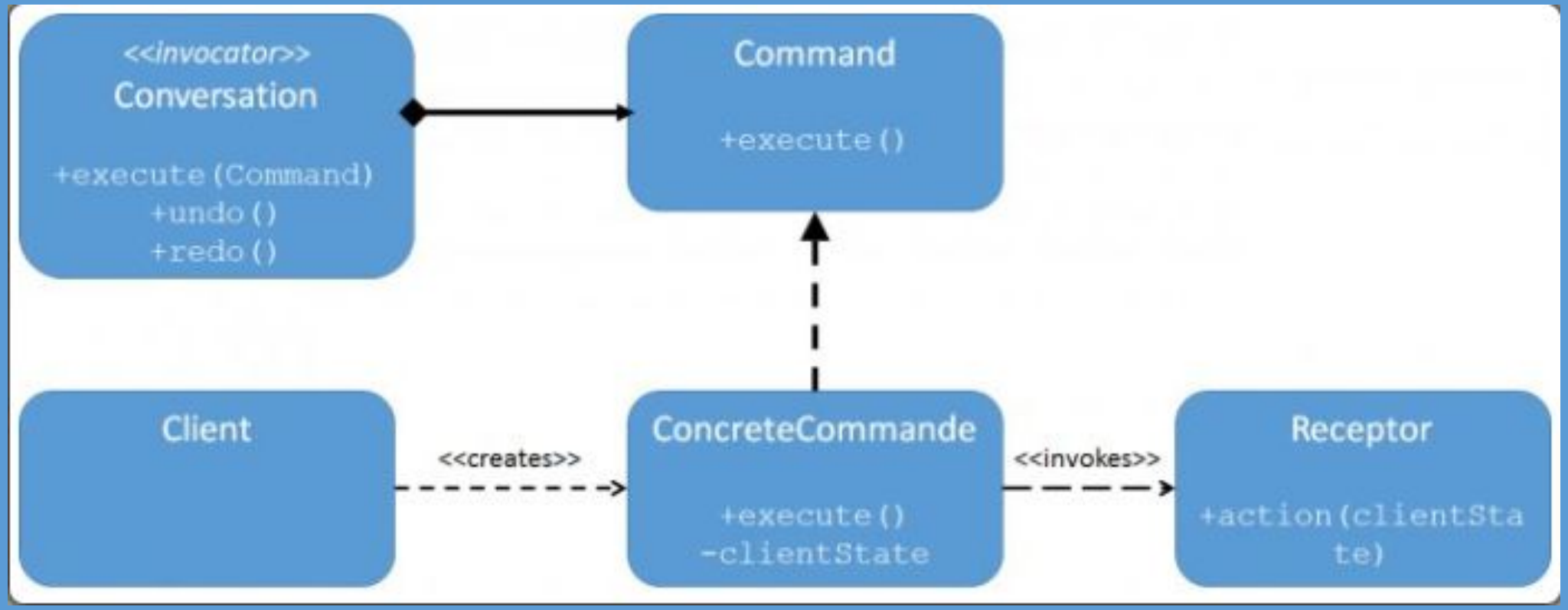
-> Les objets possèdent trop de méthodes publiques à l'usage d'autres objets.

-> L'interface est inexploitable et on la modifie tout le temps.

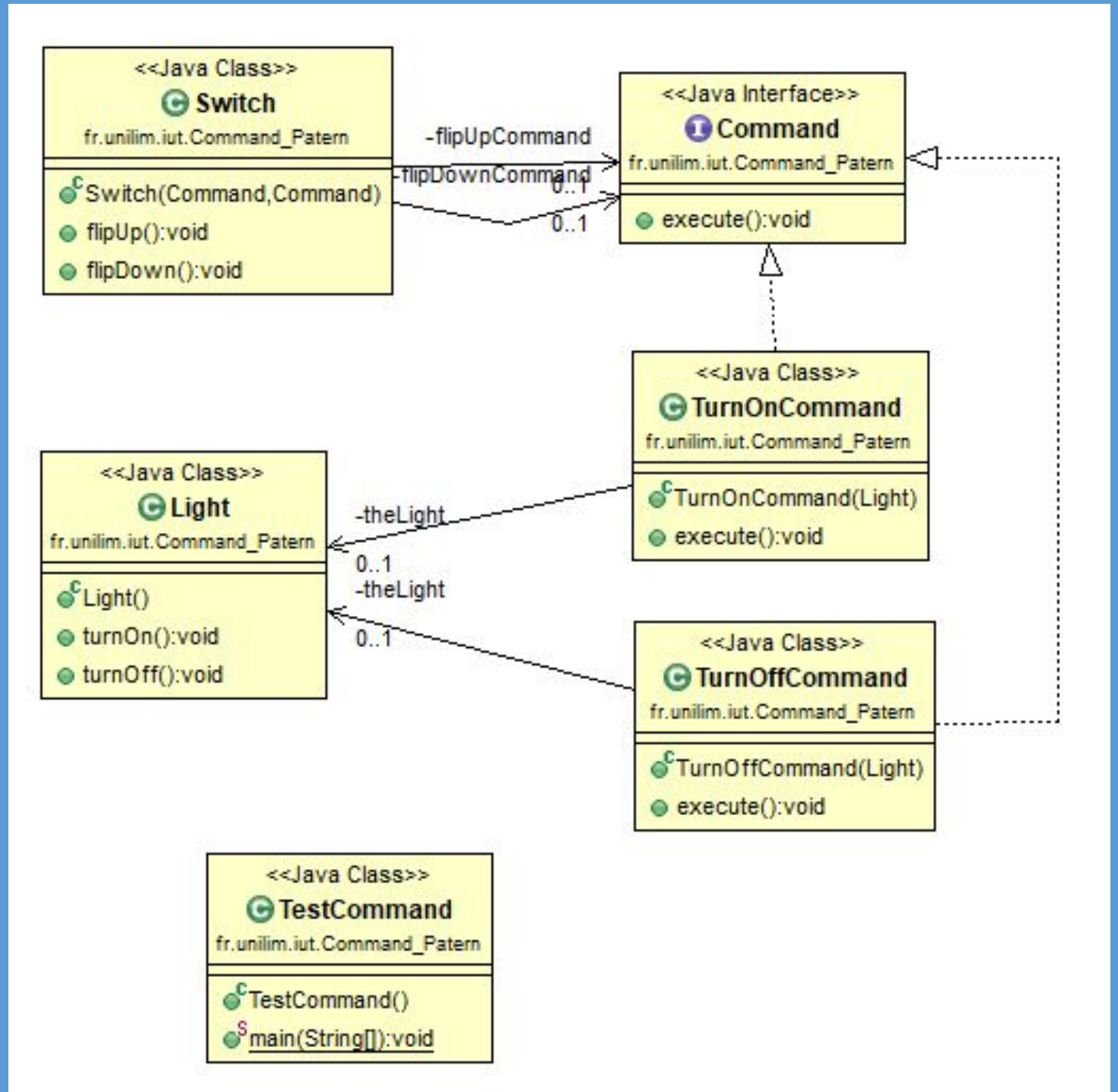
->Les noms des méthodes deviennent de longues périphrases.

LA COMMANDE

undo() & redo()



EXEMPLE


<<Java Class>>
Switch
fr.unilim.iut.Command_Patern
Switch(Command,Command)
flipUp():void
flipDown():void
-flipUpCommand
-flipDownCommand
0..1
0..1
<<Java Interface>>
Command
fr.unilim.iut.Command_Patern
execute():void
<<Java Class>>
TurnOnCommand
fr.unilim.iut.Command_Patern
TurnOnCommand(Light)
execute():void
<<Java Class>>
Light
fr.unilim.iut.Command_Patern
Light()
turnOn():void
turnOff():void
-theLight
0..1
-theLight
0..1
<<Java Class>>
TurnOffCommand
fr.unilim.iut.Command_Patern
TurnOffCommand(Light)
execute():void
<<Java Class>>
TestCommand
fr.unilim.iut.Command_Patern
TestCommand()
main(String[]):void

```java
/* Commande */
public interface Command
{
    void execute();
}
```

```java
/* Invocateur */
public class Switch
{
    private Command flipUpCommand;
    private Command flipDownCommand;

    public Switch(Command flipUpCmd,Command flipDownCmd)
    {
        this.flipUpCommand=flipUpCmd;
        this.flipDownCommand=flipDownCmd;
    }

    public void flipUp()
    {
        flipUpCommand.execute();
    }

    public void flipDown()
    {
        flipDownCommand.execute();
    }
}
```

```java
/* Commande concrète pour éteindre la lumière */
public class TurnOffCommand implements Command
{
    private Light theLight;

    public TurnOffCommand(Light light)
    {
        this.theLight=light;
    }

    public void execute()
    {
        theLight.turnOff();
    }
}
```

```java
/* Commande concrète pour allumer la lumière */
public class TurnOnCommand implements Command
{
    private Light theLight;

    public TurnOnCommand(Light light)
    {
        this.theLight=light;
    }

    public void execute()
    {
        theLight.turnOn();
    }
}
```

```java
/* Récepteur */
public class Light
{
    public Light() {  }

    public void turnOn()
    {
        System.out.println("The light is on");
    }

    public void turnOff()
    {
        System.out.println("The light is off");
    }
}
```

```java
/* Classe de test */
public class TestCommand
{
    public static void main(String[] args)
    {
        Light lamp = new Light();
        Command switchUp=new TurnOnCommand(lamp );
        Command switchDown=new TurnOffCommand(lamp );

        Switch s=new Switch(switchUp,switchDown);

        s.flipUp();
        s.flipDown();
    }
}
```

```
<terminated> TestCommand [Java Application]
The light is on
The light is off
```