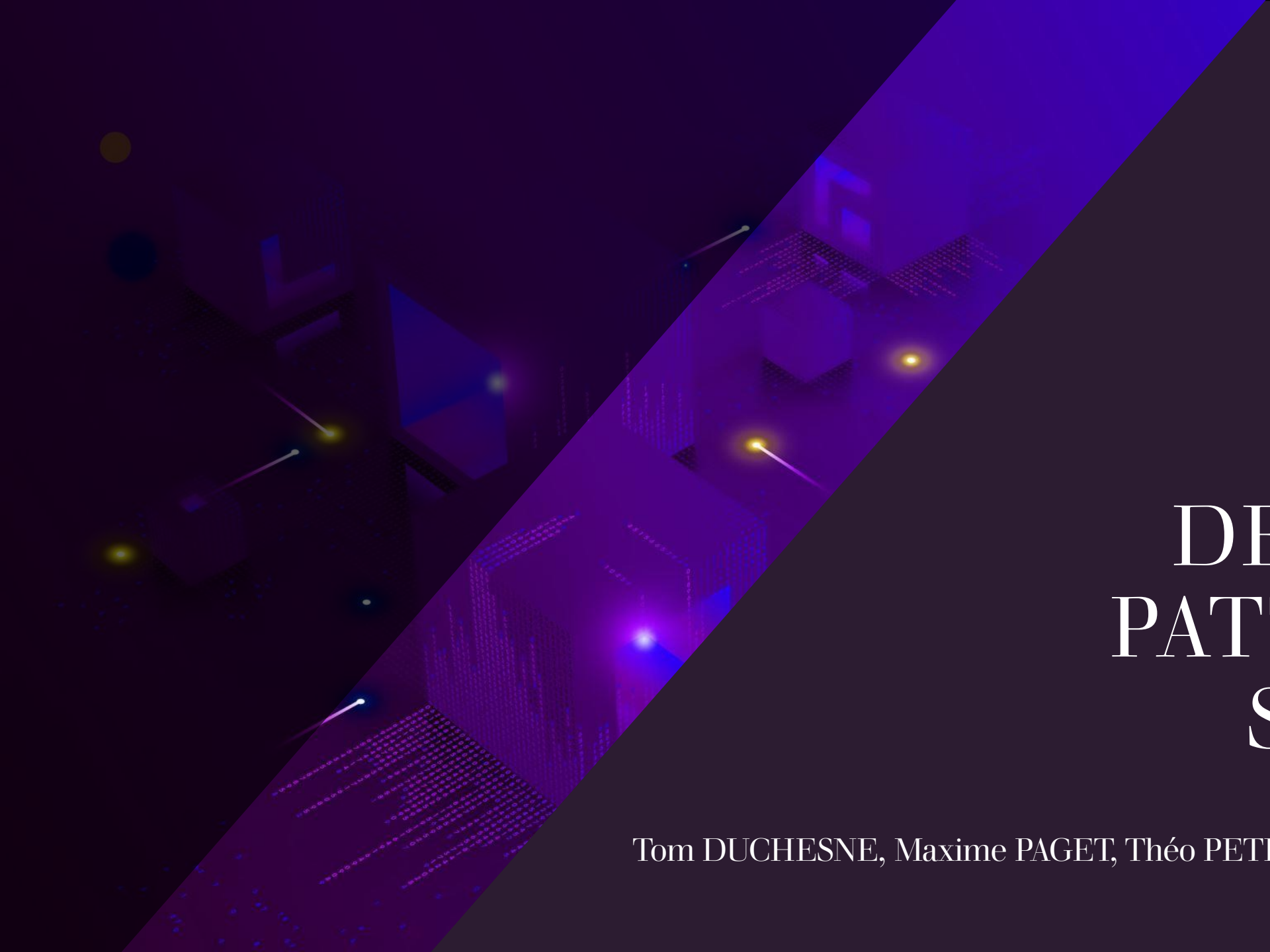
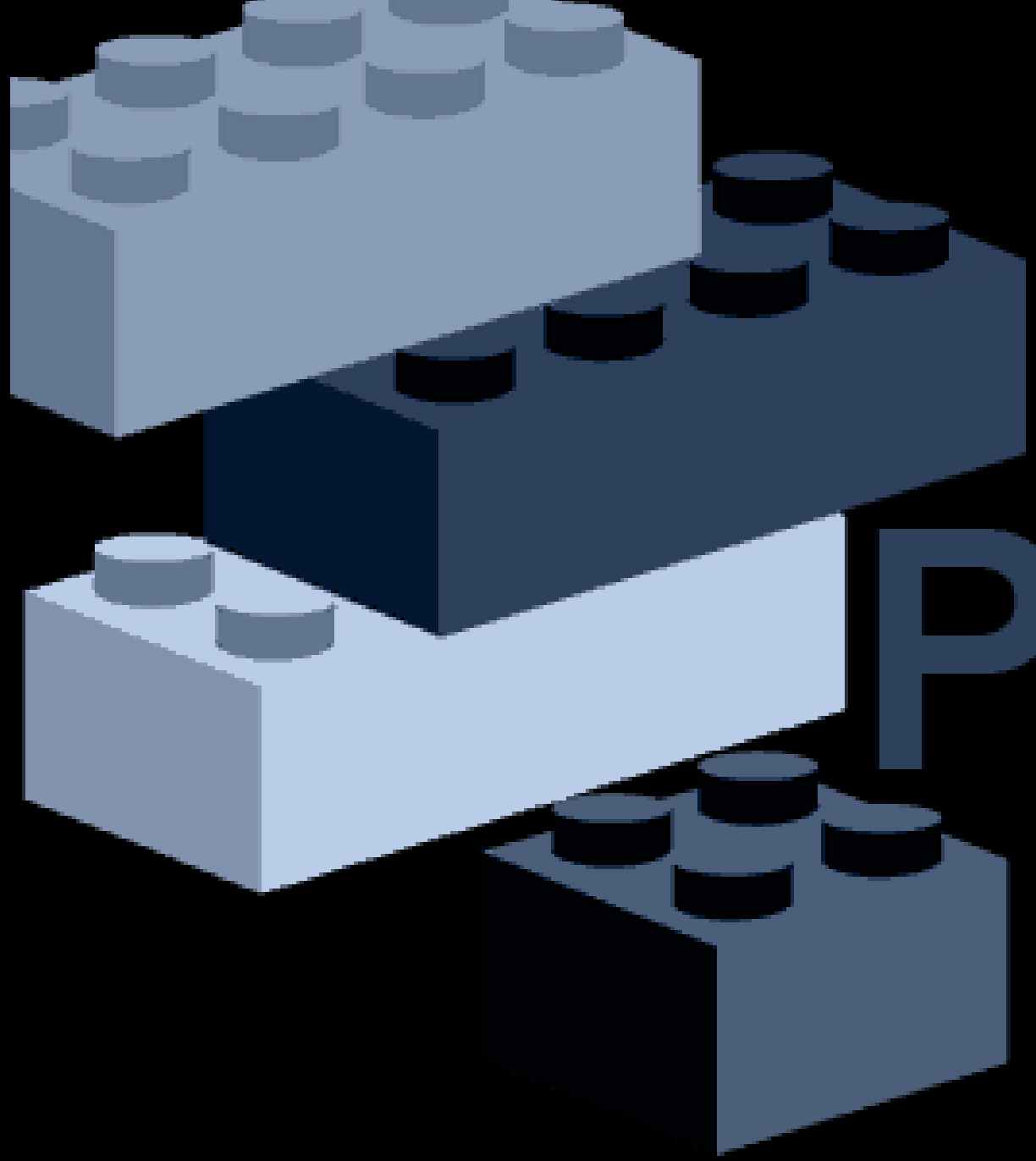


DESIGN PATTERN STATE

Tom DUCHESNE, Maxime PAGET, Théo PETIT, Maxime NGUYEN

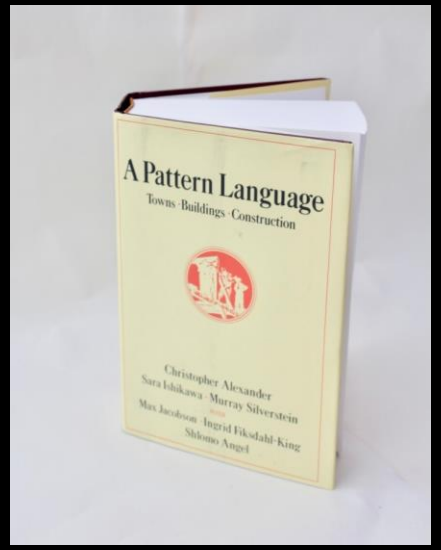
Sommaire



Design Patterns

Design Patterns

L'origine



1977

CHRISTOPHER ALEXANDER

A PATTERN LANGUAGE, TOWNS, BUILDINGS, CONSTRUCTION

Design Patterns

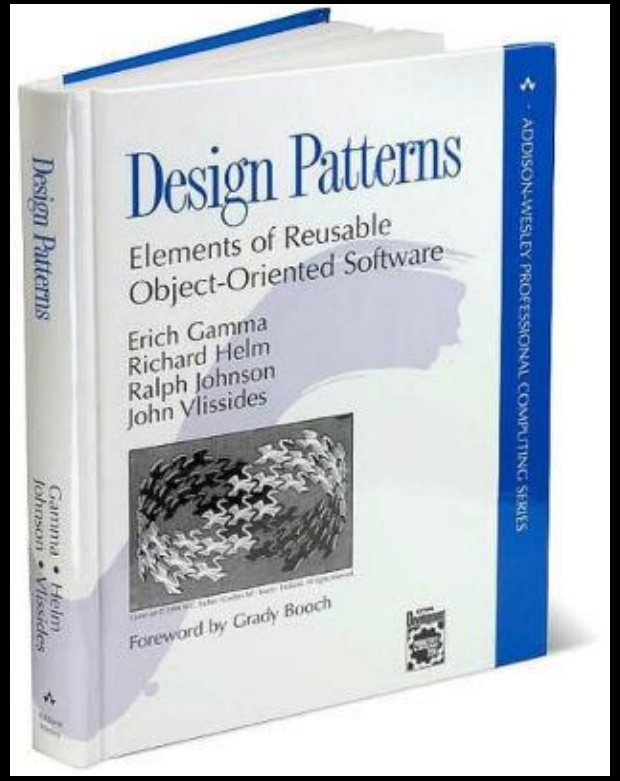
Gang of Four (GoF)

Erich Gamma

Richard Helm

Ralph Johnson

John Vlissides



Il permet d’optimiser le code informatique afin de le rendre plus lisible et de meilleure qualité.

Creation design patterns

Structural design patterns

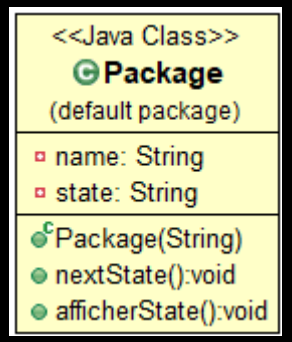
Behavioral design patterns

MISE EN SITUATION

Suivi d’un colis

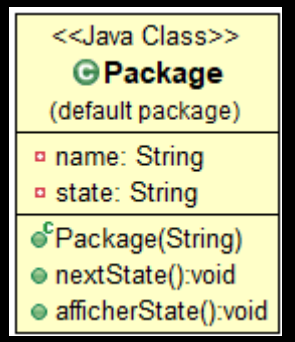
Suivi d’un colis

```java
public class Package {

    private String name;
    private String state;

    public Package(String name) {
        this.name = name;
        this.state = "Ordered";
    }

    public void nextState() {
        switch(state) {
        case "Ordered" :
            state = "Prepared";
            break;
        case "Prepared" :
            state = "Delivery";
            break;
        case "Delivery" :
            state = "Delivered";
            break;
        case "Delivered":
            System.out.println("Le colis a été livré, nous ne pouvons plus interagir avec");
            break;
        default :
            System.out.println("Une erreur est survenue");
            break;
        }
    }

    public void afficherState() {
        System.out.println(this.state);
    }
}
```



Suivi d’un colis

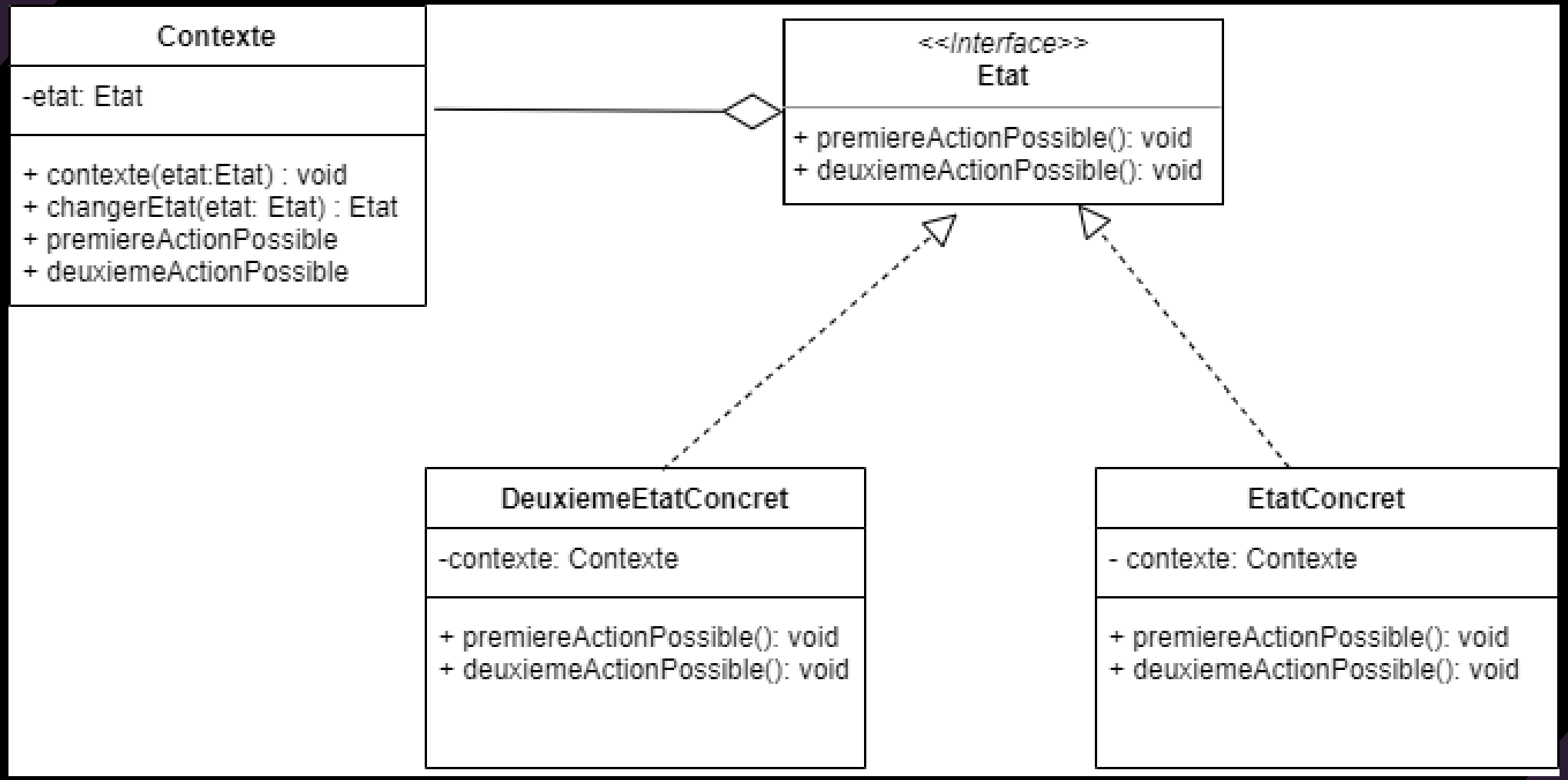
- Trop simple ?
- Difficile à améliorer ?
- Non respect des principes SOLID ?



LE STATE

LE STATE

Problème ? Solution !


Contexte
-etat: Etat
+ contexte(etat:Etat) : void
+ changerEtat(etat: Etat) : Etat
+ premiereActionPossible
+ deuxiemeActionPossible
<<Interface>>
Etat
+ premiereActionPossible(): void
+ deuxiemeActionPossible(): void
DeuxiemeEtatConcret
-contexte: Contexte
+ premiereActionPossible(): void
+ deuxiemeActionPossible(): void
EtatConcret
- contexte: Contexte
+ premiereActionPossible(): void
+ deuxiemeActionPossible(): void

Quand est-il utilisable ?

- Le code contient plusieurs états voués à changer
- Le comportement d'un objet varie en fonction de l'état
- Réduire la maintenance
- Classe polluée par du code changeant le comportement de la classe en fonction de l'attribut
- Duplication de code dans des états similaires
- Quand on veut instaurer une hiérarchie entre nos classes

- Organise le code lié aux différents états dans des classes séparées.
- Ajoutez de nouveaux états sans modifier les classes état ou le contexte existants.
- Élimine les grandes parties conditionnelles de l'automate
- Efficace avec plusieurs états divers

- Exagérer lorsqu'il comporte peu d'états

LA SITUATION AVEC STATE

Suivi d’un colis

Suivi d'un colis – Avec State

```java
public class Package {
    private String name;
    private State state;

    public Package(String name) {
        this.name = name;
        this.state = new Ordered(this);
    }

    public void setState(State state) {
        this.state = state;
    }

    public State state() {
        return this.state;
    }
}
```

- String state devient un objet State
- Class Package plus simple

Suivi d’un colis – Avec State

```
public interface State {

    abstract void action();
    abstract void nextState();
}
```

- Une interface simple
- Méthodes abstraites

Suivi d'un colis – Avec State

```java
public class Ordered implements State {

    Package purchase;

    public Ordered(Package purchase) {
        this.purchase = purchase;
    }

    @Override
    public void nextState() {
        purchase.setState(new Prepared(this.purchase));
    }



    @Override
    public void action() {
        System.out.println("Ordered");

    }

}
```

- Un état pour un package spécifique
- Un comportement spécifique à chaque état

Suivi d’un colis – Avec State

```java
public class Delivered implements State {

    Package purchase;
    public Delivered(Package purchase) {
        this.purchase =purchase;
    }
    @Override
    public void action() {
        System.out.println("Delivered");

    }
    @Override
    public void nextState() {
        // TODO Auto-generated method stub
    }


}
```

- Cas particulier : Dernier état
- Exemple de comportement spécifique

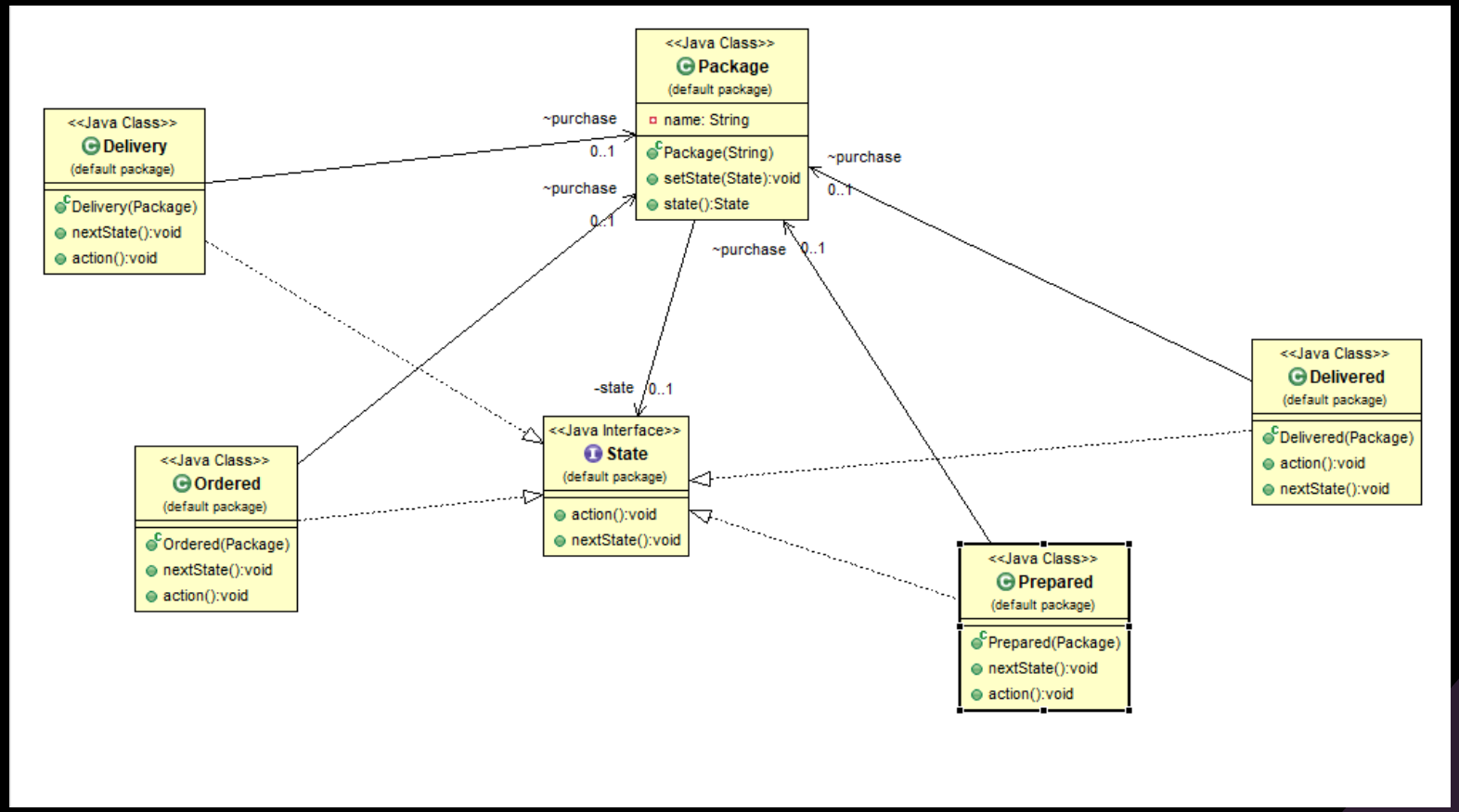
Suivi d'un colis – Avec State

```java
public class Main {

    private final static Package purchase = new Package("Teddy bear");

    public static void main(String[] args){

        purchase.state().action();

        purchase.state().nextState();
        purchase.state().action();

        purchase.state().nextState();
        purchase.state().action();

        purchase.state().nextState();
        purchase.state().action();
    }
}
```

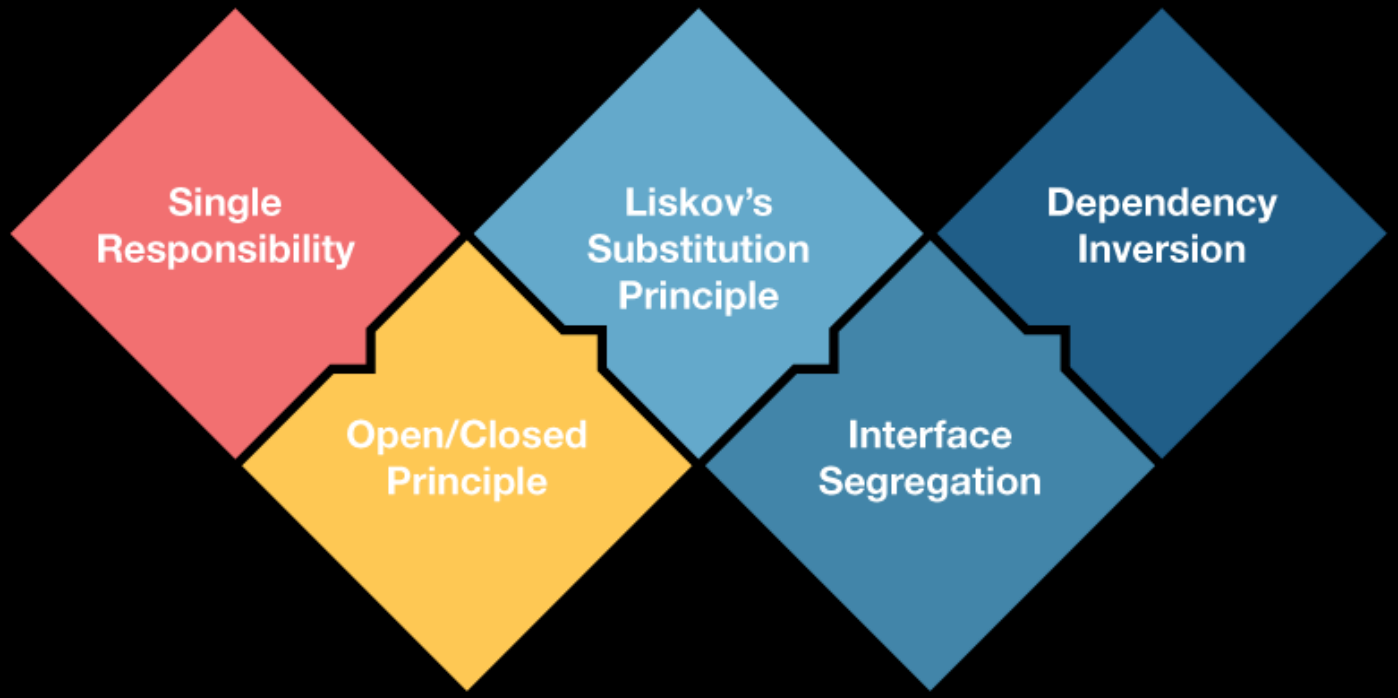
```
Ordered
Prepared
Delivery
Delivered
```

Suivi d'un colis – Avec State

- Mieux ordonné
- Simple à améliorer
- Respecte les principes SOLID

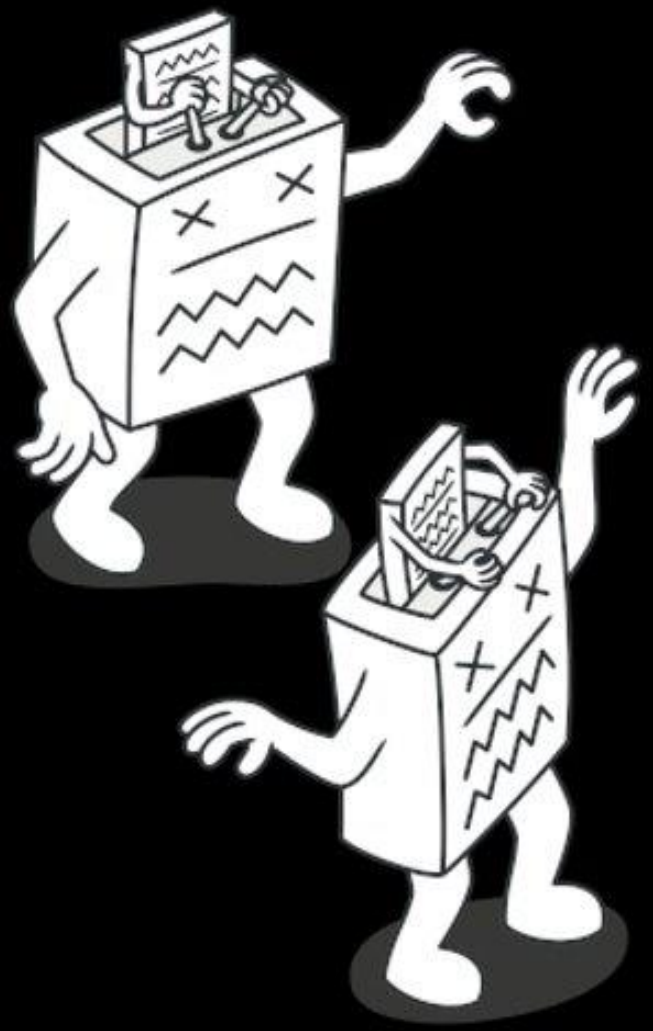


S.O.L.I.D.

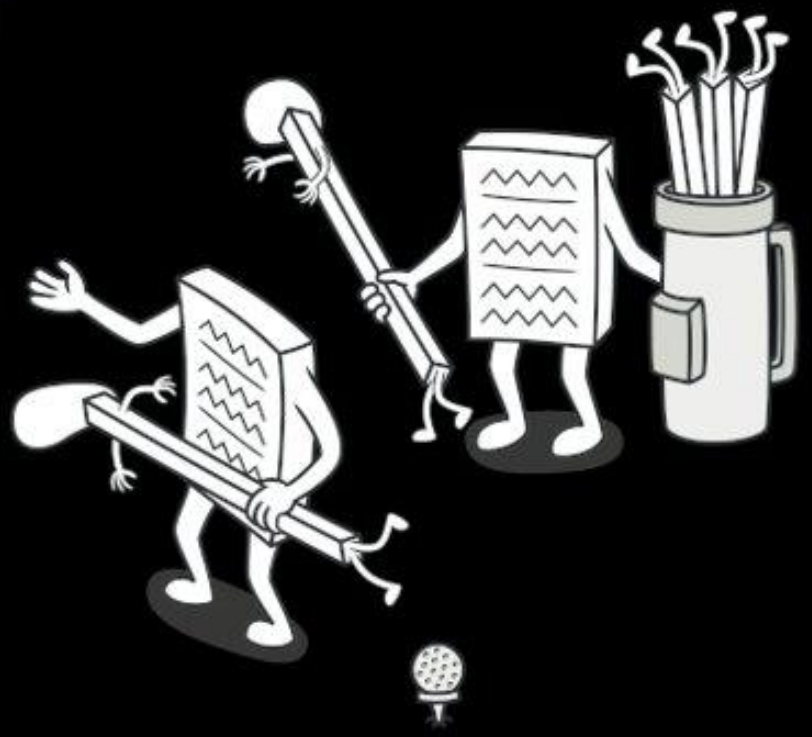

Single Responsibility
Open/Closed Principle
Liskov's Substitution Principle
Interface Segregation
Dependency Inversion

Single Responsibility Principle	
Open/Closed Principle	
Liskov’s Substitution Principle	
Interface Segregation Principle	
Dependancy Inversion Principle	

Où sont les limites ?



VS



STATE

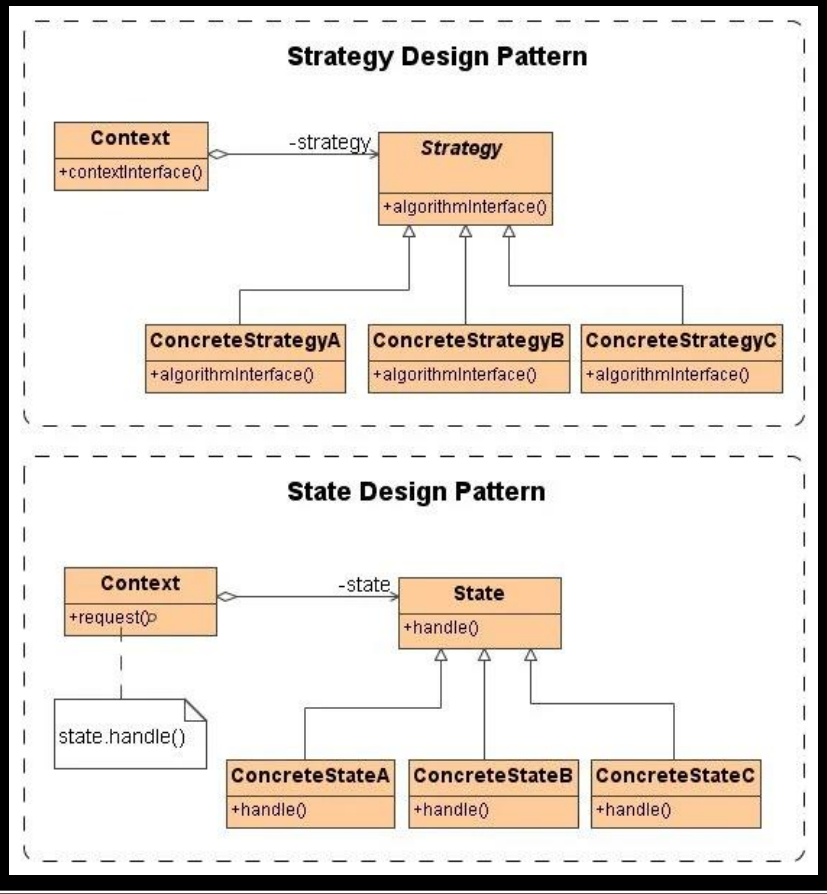
STRATEGY

STRATEGY C'EST QUOI ?

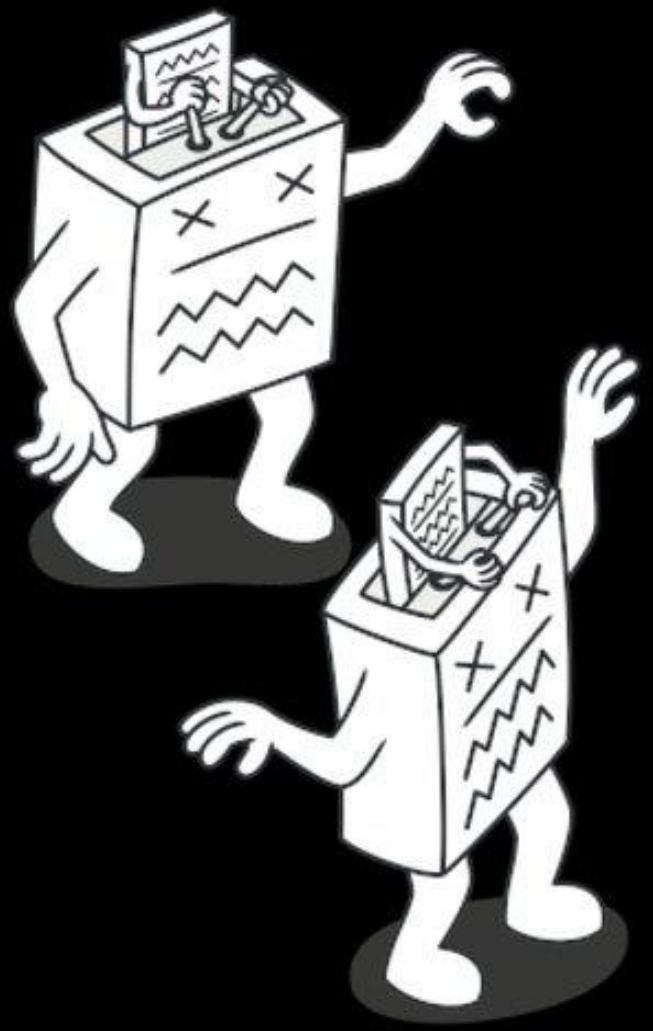
State VS Strategy – Pourquoi les comparer ?

- Structure similaire :

- Basés sur OCP



State	VS	Strategy
Contient référence		Ignore objet
Fait partie de l'objet		Est un paramètre
Définit « Quoi » et « Quand »		Définit « Comment»
Changement par lui-même		Changement par client

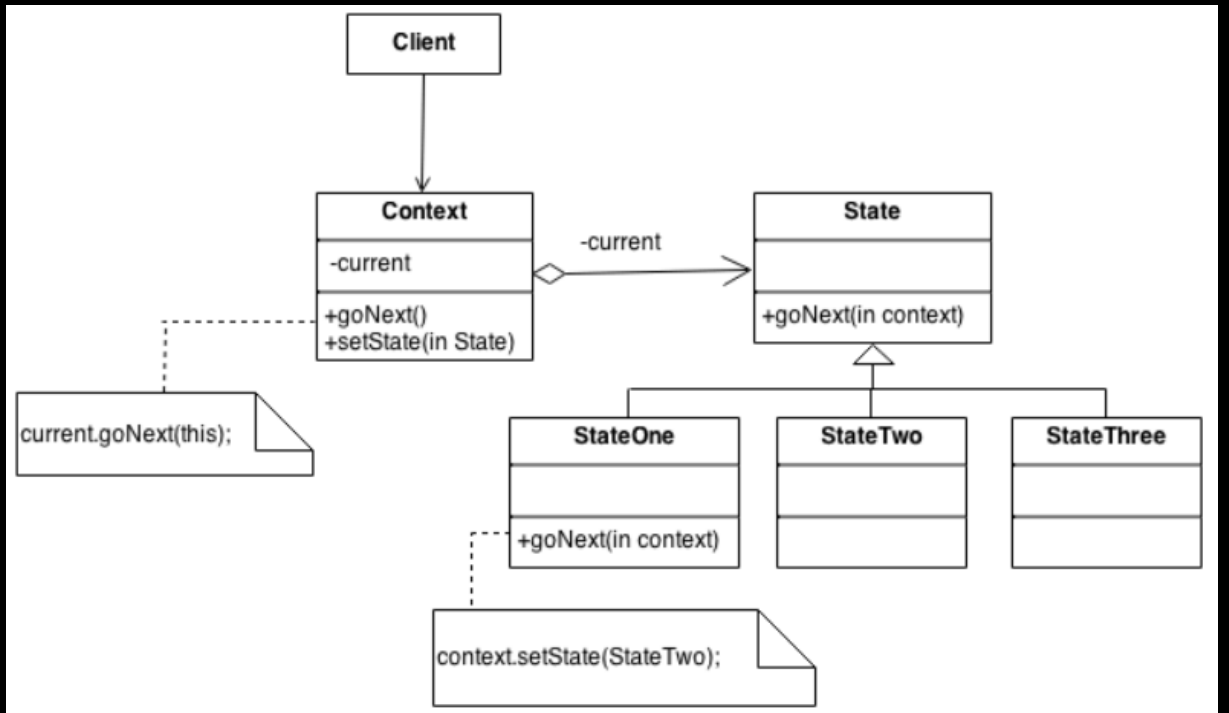


STATE

VS

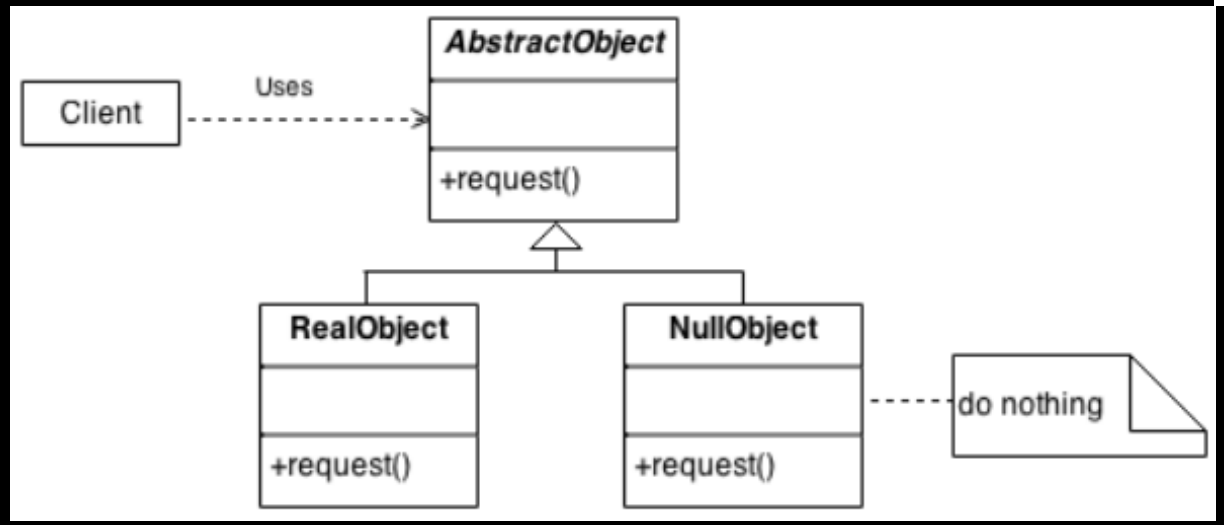
NULL OBJECT

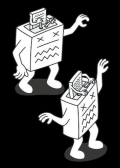
NULL OBJECT



STATE

NULL OBJECT





STATE

Utilise plusieurs instances

Vérifie si l'objet est NULL

VS

NULL OBJECT

Utilise plusieurs fois une instance

Outrepasse les conditions NULLS

Ne peut être un objet concret

Le pattern State appartient au

A) Gang of Four

B) Patron de création

C) Patron de comportement

D) Patron de structure

Le pattern State appartient au

A) Gang of Four

B) Patron de création

C) Patron de comportement

D) Patron de structure

Le pattern State appartient au

A) Gang of Four

B) Patron de création

C) Patron de comportement

D) Patron de structure

Ce pattern State respecte

A) Le SCP

B) Le ISP

C) La famille

D) Le OCP

Ce pattern State respecte

A) Le SCP

B) Le ISP

C) La famille

D) Le OCP

Ce pattern State respecte

A) Le SCP

B) Le ISP

C) La famille

D) Le OCP

Le pattern State n'est pas recommandé quand

A) L'objet a peu d'états et beaucoup de changement

B) L'objet a beaucoup d'états et beaucoup de changement

C) L'objet a peu d'états et peu de changement

D) L'objet a beaucoup d'états et peu de changement

Le pattern State n'est pas recommandé quand

A) L'objet a peu d'états et beaucoup de changement

B) L'objet a beaucoup d'états et beaucoup de changement

C) L'objet a peu d'états et peu de changement

D) L'objet a beaucoup d'états et peu de changement

Le pattern State n'est pas recommandé quand

A) L'objet a peu d'états et beaucoup de changement

B) L'objet a beaucoup d'états et beaucoup de changement

C) L'objet a peu d'états et peu de changement

D) L'objet a beaucoup d'états et peu de changement

Les patterns présentés sont

A) State

B) Architecture

C) Null Object

D) Strategy

Les patterns présentés sont

A) State

B) Architecture

C) Null Object

D) Strategy

Les patterns présentés sont

A) State

B) Architecture

C) Null Object

D) Strategy

Quel exemple(s) avons-nous utilisé dans cette présentation ?

A) Une pizza avec plusieurs états

B) Un ordinateur avec plusieurs états

C) Une commande avec plusieurs états

D) Un lecteur vidéo avec plusieurs états

Quel exemple(s) avons-nous utilisé dans cette présentation ?

A) Une pizza avec plusieurs états

B) Un ordinateur avec plusieurs états

C) Une commande avec plusieurs états

D) Un lecteur vidéo avec plusieurs états

Quel exemple(s) avons-nous utilisé dans cette présentation ?

A) Une pizza avec plusieurs états

B) Un ordinateur avec plusieurs états

C) Une commande avec plusieurs états

D) Un lecteur vidéo avec plusieurs états