

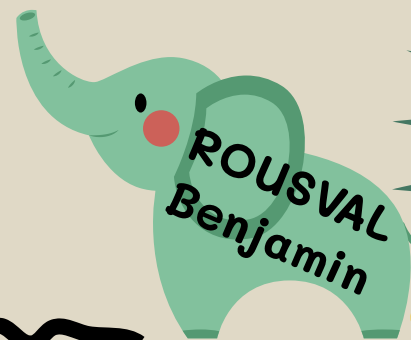
# United STATES Of Pattern

Oui, c'est du comic sans MS

Here is where the design begin

Pas celui là

Ici aussi



# Sommaire

**01**

**Problèmes**

**02**

**Solutions**

**03**

**Implémentation**

**04**

**Pour et contre**





**C'est parti !**



# Qu'est ce qu'un state ?

Le **State** est un design pattern comportemental permettant à un objet de modifier son comportement lorsque son état interne change. Il apparaît comme si l'objet changeait de fonctionnement.



# 01 Problèmes

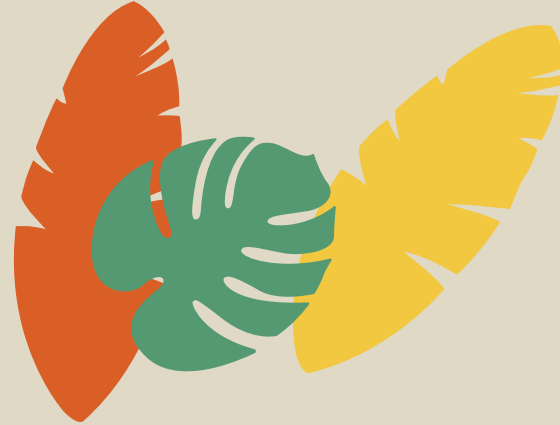


# Mise en situation



1

Dans chaque état unique, le programme se comporte différemment et peut passer d'un état à l'autre instantanément.




2

Selon l'état actuel, le programme peut ou non passer à certains autres états.

3

Ces règles de commutation, appelées transitions, sont également finies et prédéterminées.





**Ca me paraît très  
compliqué tout ça..!**



A stylized tropical illustration. On the left, a tall palm tree with green fronds and a small cluster of coconuts stands on a green mound. The foreground is filled with various tropical plants: a red fan palm on the left, a dark green monstera leaf, a yellow monstera leaf, and a large green fern on the right. The background is a solid light beige color.

**En vrai ?  
Non.**





Oké

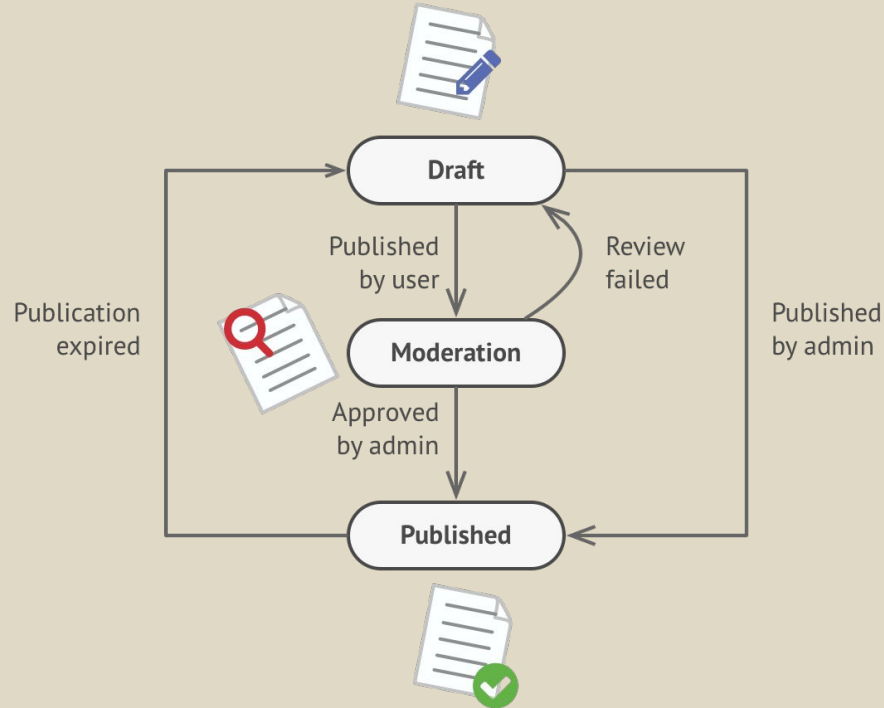
# Exemple : la classe Document

Un document peut se trouver dans l'un des trois états suivants :

- **Rédaction**
- **Modération**
- **Publication**



# Exemple : la classe Document



# Exemple : la classe Document

```
class Document is
  field state: string
  // ...
  method publish() is
    switch (state)
      "draft":
        state = "moderation"
        break
      "moderation":
        if (currentUser.role == 'admin')
          state = "published"
          break
      "published":
        // Do nothing.
        break
  // ...
```



# Problèmes



Oblige un code avec beaucoup de conditions pour vérifier l'état du state



Rend le code très difficilement maintenable et brouillon



Plus le projet évolue plus les problèmes grandissent. Il est impossible de prévoir tous les states au design de l'application



## 02 Solutions





# Aux problèmes, des solutions

**STEP 1**



Une classe par état  
d'un objet

**STEP 2**



Faire appel à un  
contexte qui délègue  
les traitements aux  
classes States


**STEP 3**



Pour changer d'état,  
appelez une nouvelle  
classe state (elles  
doivent avoir la  
même interface)





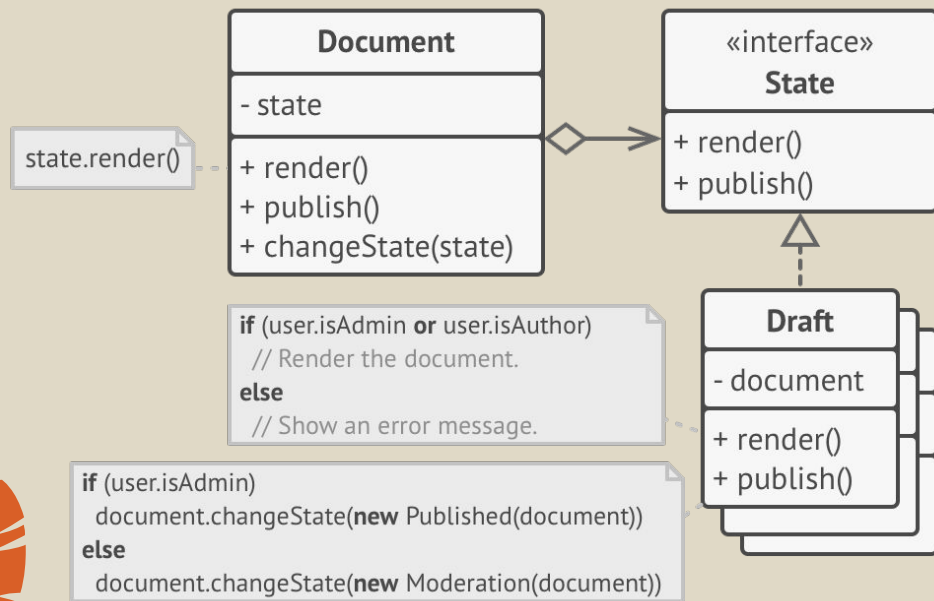


**03**

# **Implementation**



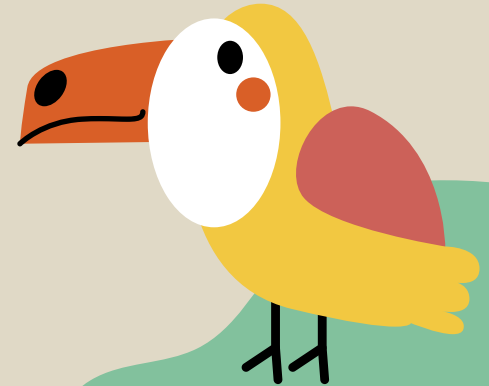
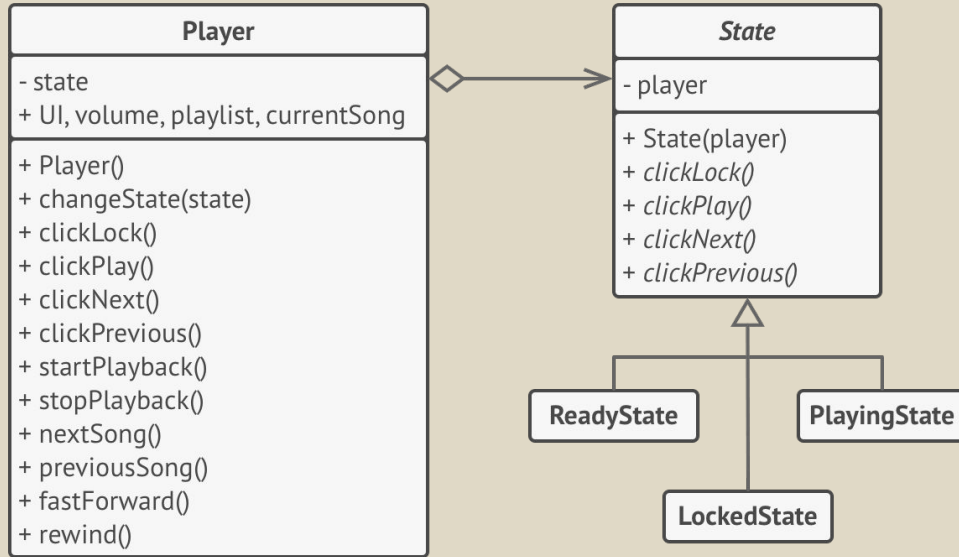
# Implémentation



A stylized illustration of a tropical jungle scene. In the background, a tall palm tree with a cluster of coconuts stands on the left. The foreground is filled with various tropical plants, including large green and yellow leaves, and smaller green bushes. The overall color palette is warm, featuring shades of green, yellow, and orange against a light beige background.

**Complicquons  
les choses**

# la classe Player






```
class AudioPlayer is
  field state: State
  field UI, volume, playlist, currentSong

  constructor AudioPlayer() is
    this.state = new ReadyState(this)

  UI = new UserInterface()
  UI.lockButton.onClick(this.clickLock)
  UI.playButton.onClick(this.clickPlay)


  method changeState(state: State) is
    this.state = state
  method clickLock() is
    state.clickLock()
  method clickPlay() is
    state.clickPlay()
```





```
abstract class State is  
    protected field player: AudioPlayer  
    constructor State(player) is  
        this.player = player  
abstract method clickLock()  
abstract method clickPlay()
```

```
state  
method click  
state.clickPlay()
```



```
class LockedState extends State is
    method clickLock() is
        if (player.playing)
            player.changeState(new PlayingState(player))
        else
            player.changeState(new ReadyState(player))

    method clickPlay() is
```


```
class ReadyState extends State is
    method clickLock() is
        player.changeState(new LockedState(player))

    method clickPlay() is
        player.startPlayback()
        player.changeState(new PlayingState(player))
```

```
class PlayingState extends State is
    method clickLock() is
        player.changeState(new LockedState(player))

    method clickPlay() is
        player.stopPlayback()
        player.changeState(new ReadyState(player))
```





04

# Pours et contres





## Pour

- Single Responsibility Principle.
- Open/Closed Principle.

## Contre

- Le modèle de l'application peut être exagérée si une classe n'a que quelques états ou change rarement.



# Merci!

Avez-vous des questions ?

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.

