

# PATTERN VISITOR

présenté par : DE POORTER Lucas, DETERNE  
Mateo, MARTY Benjamin, GORCE Max



BUT 2 - 2025



# PLAN

02/17

- 1 - Introduction générale
- 2 - Mise en situation
- 3 - Limites de la première solution
- 4 - Pattern Visitor
- 5 - Exemple détaillé
- 6 - Avantages & SOLID
- 7 - Limites du Visitor
- 8 - Comparaisons avec d'autres patterns
- 9 - Exemple métier (jeu vidéo)
- 10 - Live coding
- 11 - QCM interactif
- 12 - Conclusion & Bibliographie

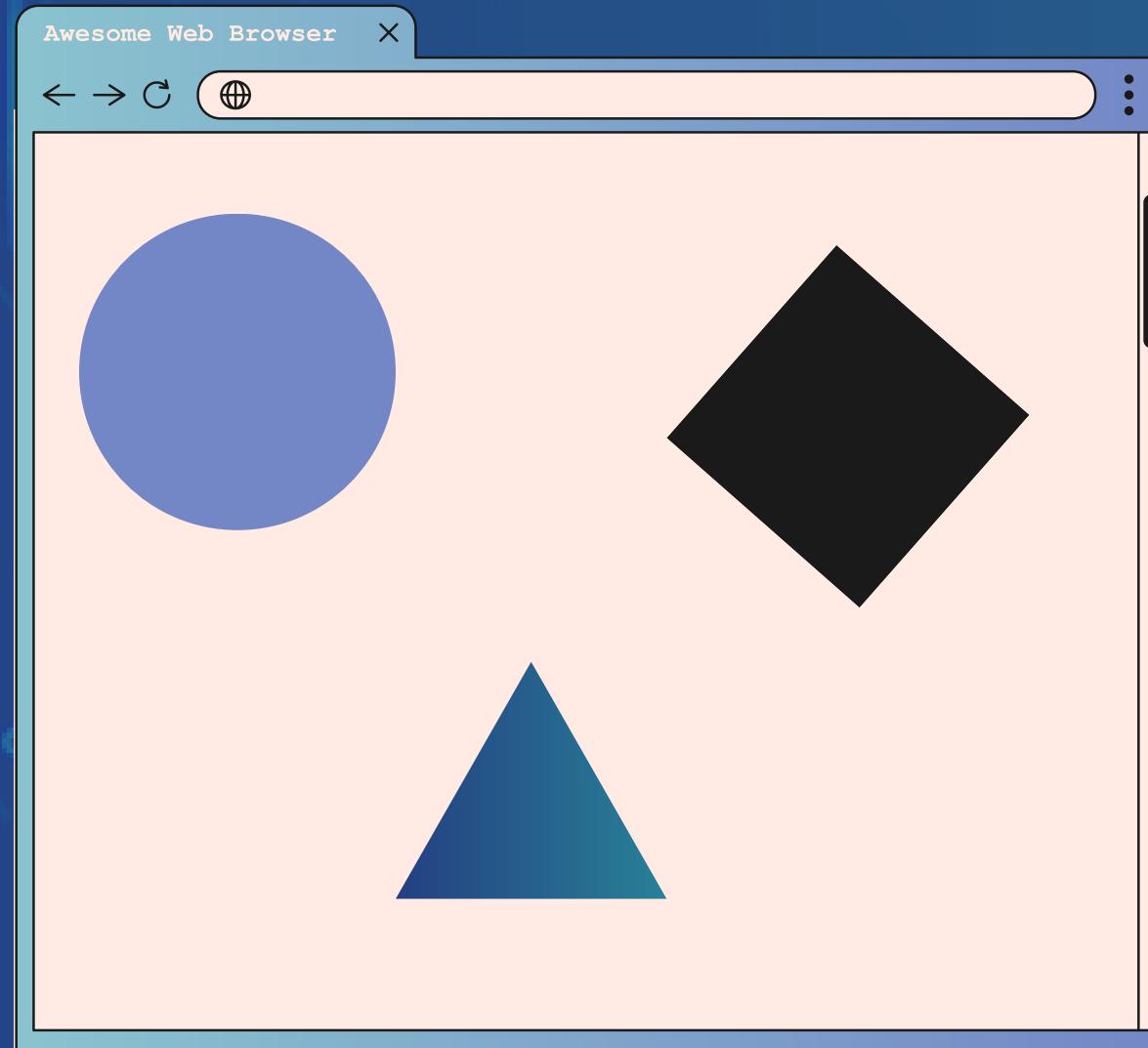
# INTRODUCTION GÉNÉRALE

**DESIGN PATTERNS** : solutions réutilisables à des problèmes courants.

Catalogue des Gang Of Four (1994)

- **Catégories :**
  - Créationnels
  - Structurels
  - Comportementaux

# MISE EN SITUATION



Besoin :

- calculer l'aire
- afficher
- exporter en JSON/XML
- générer du SVG

# PREMIÈRE MODÉLISATION

Cercle

- + afficher()
- +calculerAire()
- +exporterEnJSON()
- +genererSVG()

Triangle

- + afficher()
- +calculerAire()
- +exporterEnJSON()
- +genererSVG()

Carré

- + afficher()
- +calculerAire()
- +exporterEnJSON()
- +genererSVG()

# LIMITES DE LA MODÉLISATION

06/17

Cercle

+ afficher()  
+calculerAire()  
+exporterEnJSON()  
+genererSVG()

Triangle

+ afficher()  
+calculerAire()  
+exporterEnJSON()  
+genererSVG()

Carré

+ afficher()  
+calculerAire()  
+exporterEnJSON()  
+genererSVG()

Nouveau besoin:

- pouvoir les remplir d'une couleur
- changer la couleur des bords

+ changerCouleurBords() x3

+remplirDUneCouleur() x3

# INTRODUCTION AU VISITEUR

07/17

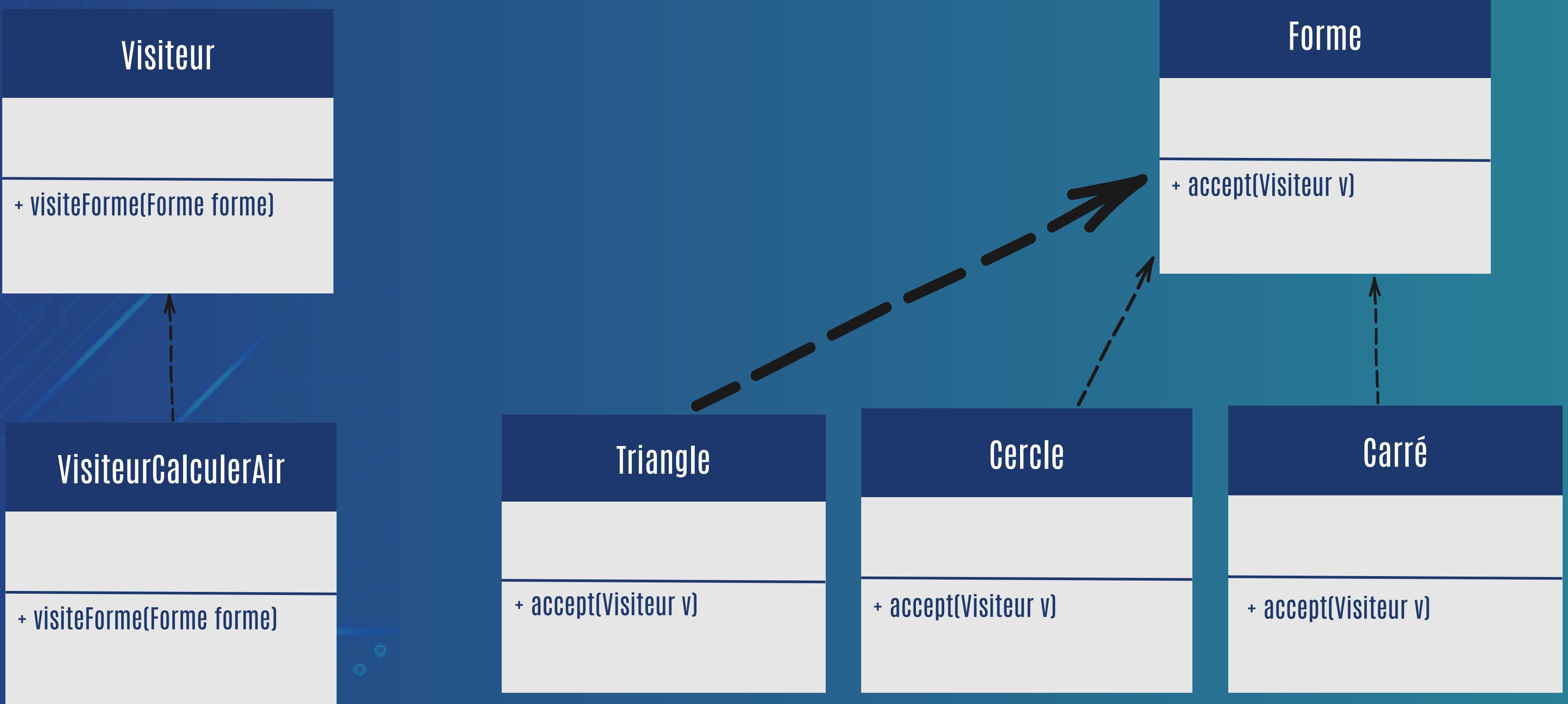
## Objectifs

### Sortir les Oppérations du modèle haut niveau

- Chaque forme devra accepter le visiteur
- Visiteur = Classe implémentant une Interface
- Le visiteur contiendra une Opérations

# UML DU MODEL AVEC VISITEUR

08/17



# EXEMPLE DE CODE

09/17

```
public interface Forme {  
    void accept(Visiteur v);  
}
```

```
public interface Visiteur {  
    void visitCercle(Cercle c);  
}
```

```
public class Cercle implements Forme {  
    double rayon;  
    Cercle(double r) { this.rayon = r; }  
    public void accept(Visiteur v) { v.visitCercle(this); }  
}
```

```
public class AireVisiteur implements Visiteur {  
    public void visitCercle(Cercle c) {  
        double aire = Math.PI * c.rayon * c.rayon;  
        System.out.println("Aire du cercle = " + aire);  
    }  
}
```

# AVANTAGES ET LIEN AVEC SOLIDE

Principe	avantage apporté par le visitor
S	Séparation données/logique métier
O	Nouveaux visiteurs sans modification
L	Visiteurs interchangeables
I	Interfaces claires et spécialisées
D	Dépendance vers l'abstraction

# LIMITES DU VISITOR

Visitor est utile quand les classes d'éléments sont stables

Mais quand on doit souvent en ajouter,  
il devient lourd

# COMPARAISON AVEC D'AUTRES PATTERNS

## Le Décorateur

ajoute des fonctionnalités à un objet particulier en l'enveloppant

## Le Visitor

ajoute de nouvelles opérations à une hiérarchie complète d'objets

## Le Composite

Le Visitor est souvent utilisé sur une structure Composite pour appliquer un traitement sur tous les éléments d'un arbre

## La Stratégie

La Stratégie encapsule un algorithme interchangeable

# EXEMPLE MÉTIER : UN JEU VIDÉO

**Classes de personnages :**

Guerrier  
Mage  
Archer

**Opérations nécessaires :**

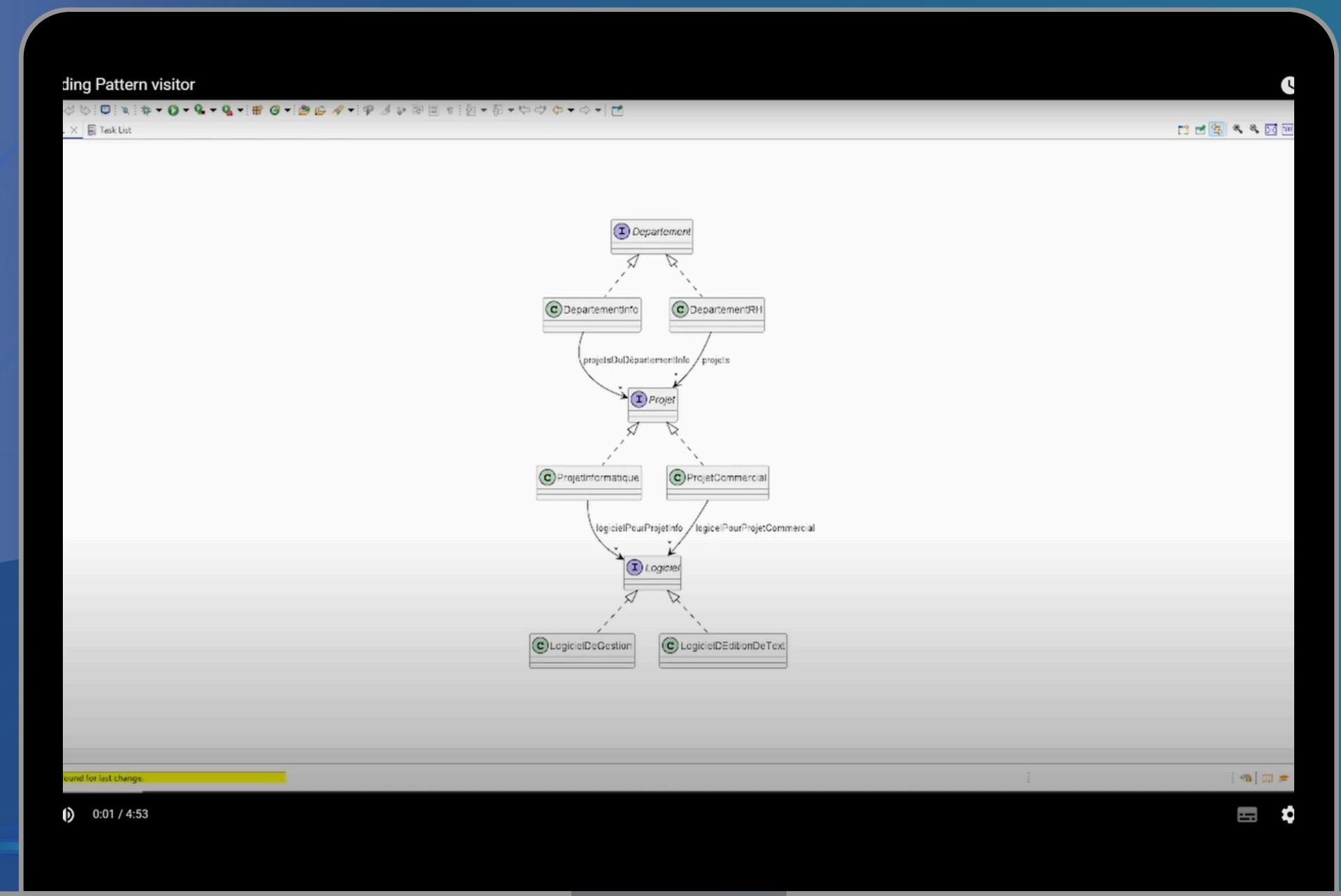
Calculer les dégâts  
Afficher les statistiques  
Appliquer un bonus magique

**Solution :**

<code>CombatVisitor</code>	→ <code>calcul des dégâts</code>
<code>AffichageVisitor</code>	→ <code>affichage statistiques</code>
<code>BuffVisitor</code>	→ <code>application de bonus</code>

# LIVE CODING

14/17



QCM!



# CONCLUSION



# BIBLIOGRAPHIE

Design Patterns: Elements of Reusable Object-Oriented Software (GoF).

UML – documentation officielle.

Chaîne YouTube Cours-en-ligne.