R3.04

DESIGN PATTERN STRATEGY

VAN RIJSWIJK Adrian / TAERO REIA / RIGOT Edith / SANTOS João

Institut Universitaire de Technologie du Limousin

SOMMAIRE

1) Présentation des Patterns du GoF

2) Premier besoin concret
   - Contexte
   - Exemple à ne pas suivre
   - Réelle solution

3) Présentation du Pattern Stratégie
   - Nom / Type
   - Problématique / Intention
   - Diagramme de classe / séquence général

4) Liens avec SOLID

5) Lien avec le pattern Etat

6) Limites du Pattern

7) Live Coding

8) Second besoin concret (Canards)

9) QCM

10) Conclusion / Bibliographie

Les Patterns du GoF

Qu’est-ce que c’est ?

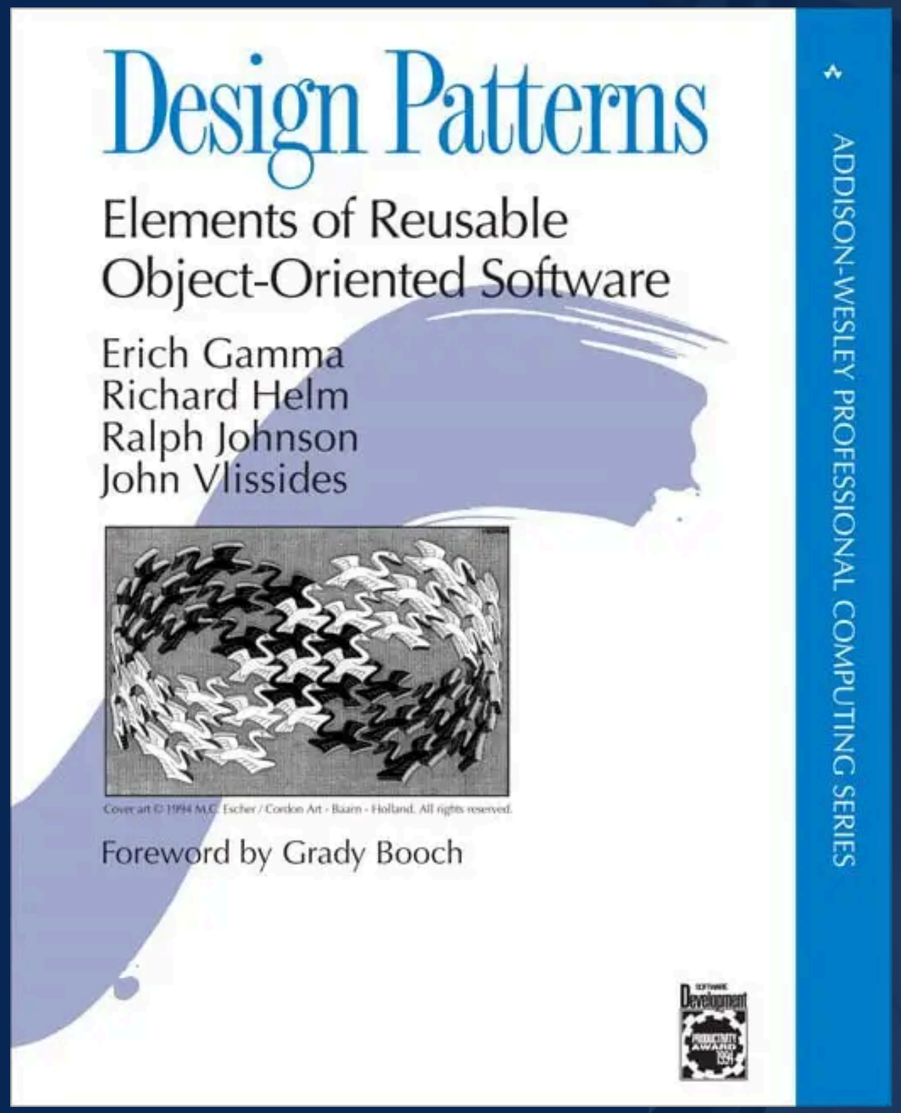
- Solutions génériques apportées à des problèmes récurrents du développement logiciel

Quelle utilité ?

- Améliorer la lisibilité / maintenabilité du code
- Faciliter la communication entre développeurs

Les différents Patterns

- 23 Pattern du GoF
- Créationnels / Structurels / Comportementaux



1ER BESOIN CLIENT

Contexte et exemple à ne pas suivre

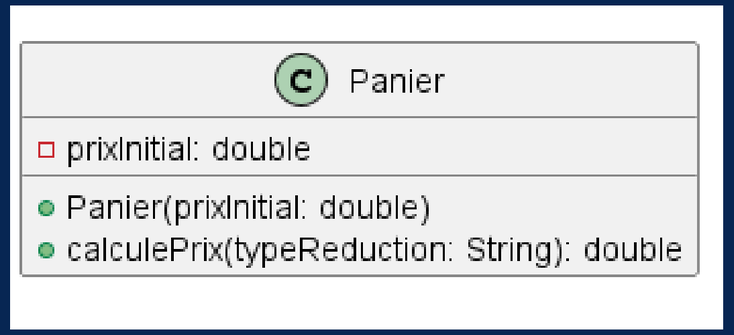
```java
public class Panier {
    private double prixInitial;

    public Panier(double prixInitial) {
        this.prixInitial = prixInitial;
    }

    // Méthode qui applique la réduction en fonction d'un type (ma
    public double calculerPrix(String typeReduction) {
        if (typeReduction.equals("Aucune")) {
            return prixInitial;
        } else if (typeReduction.equals("10%")) {
            return prixInitial * 0.9;
        } else if (typeReduction.equals("Fixe50")) {
            return prixInitial - 50;
        } else if (typeReduction.equals("20%")) {
            return prixInitial * 0.8;
        } else {
            System.out.println("Type de réduction inconnu !");
            return prixInitial;
        }
    }
}
```

Redondance

Non-respect des principes SOLID



RÉELLE SOLUTION AVEC LE PATTERN STRATEGY

```java
// --- Stratégie (interface) ---
interface StrategieReduction {
    double appliquerReduction(double prix);
}
```

```java
class PasDeReduction implements StrategieReduction {
    @Override
    public double appliquerReduction(double prix) {
        return prix; // pas de réduction
    }
}

class Reduction10Pourcent implements StrategieReduction {
    @Override
    public double appliquerReduction(double prix) {
        return prix * 0.9; // -10%
    }
}

class ReductionFixe50 implements StrategieReduction {
    @Override
    public double appliquerReduction(double prix) {
        return prix - 50; // -50 unités
    }
}
```

```java
class Panier {
    private StrategieReduction strategie;

    public Panier(StrategieReduction strategie) {
        this.strategie = strategie;
    }

    public void setStrategie(StrategieReduction strategie) {
        this.strategie = strategie;
    }

    public double calculerPrix(double prixInitial) {
        return strategie.appliquerReduction(prixInitial);
    }
}
```

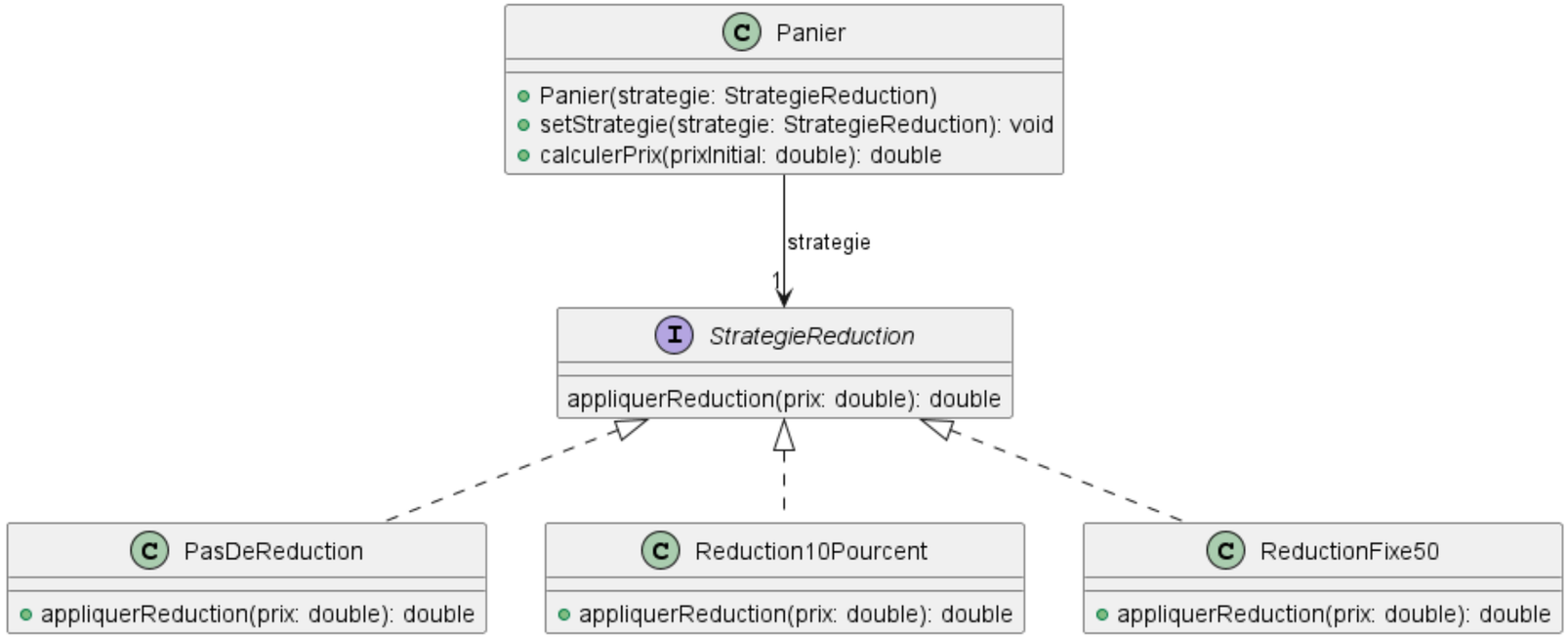
```java
public class Main {
    public static void main(String[] args) {
        double prixInitial = 500;

        Panier panier = new Panier(new PasDeReduction());
        System.out.println("Prix sans réduction : " + panier.calculerPrix(prixInit

        panier.setStrategie(new Reduction10Pourcent());
        System.out.println("Prix avec 10% de réduction : " + panier.calculerPrix(p

        panier.setStrategie(new ReductionFixe50());
        System.out.println("Prix avec réduction fixe de 50 : " + panier.calculerPr
    }
}
```

RÉELLE SOLUTION AVEC LE PATTERN STRATEGY

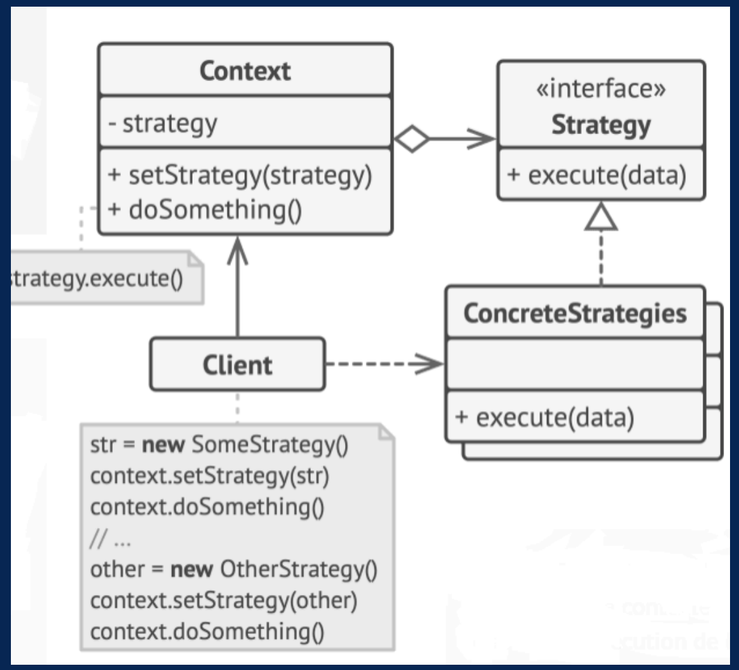


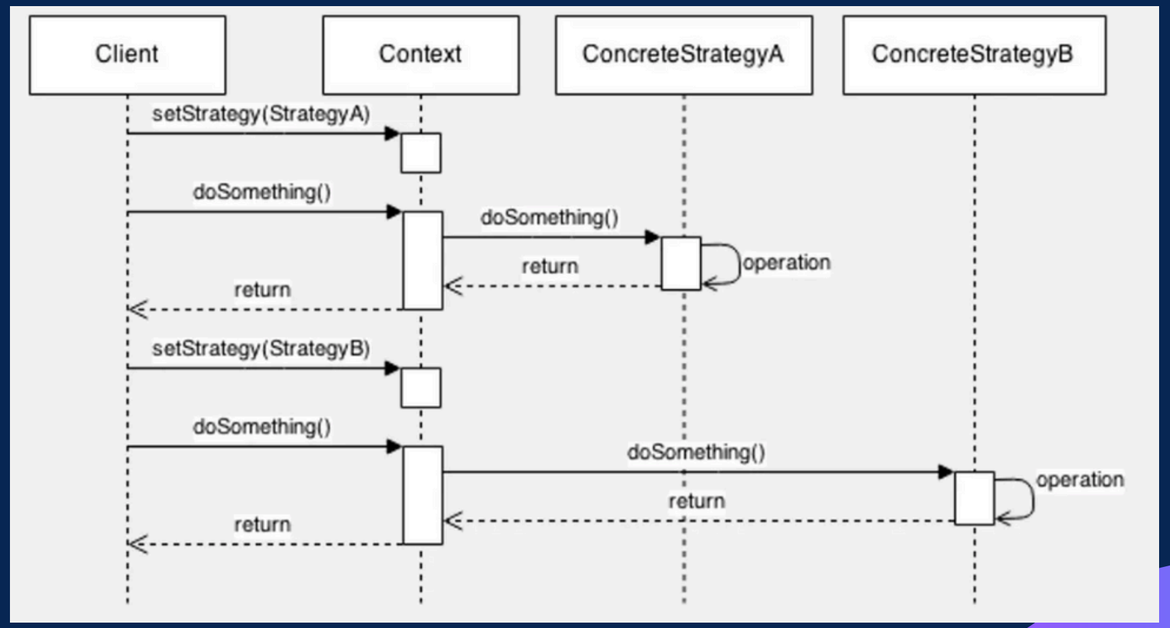
PRÉSENTATION DU PATTERN STRATEGY

1-Nom et type

2-Problématique du pattern

3- Intention/Définition





LIENS AVEC SOLID

SRP

Single Responsability Principle

Chaque Stratégie est unique

OCP

Open / Closed Principle

Programme exensible

DIP

Dependency Inversion Principle

Flexibilité du code grâce à linterface

LIEN AVEC LE PATTERN ÉTAT

STRATÉGIE

↓

ON CHOISIT LA STRATÉGIE

1. Composition,plutôt que l'heritage
2. Le contexte dépend de l'objet
3. Interface Commune
4. Réduction des "if...else if..."

ÉTAT

↓

STRATÉGIE DÉPEND DU CONTEXTE

Limites du Pattern

Multiplication des classes :

→ Chaque nouvelle stratégie = une **nouvelle classe**

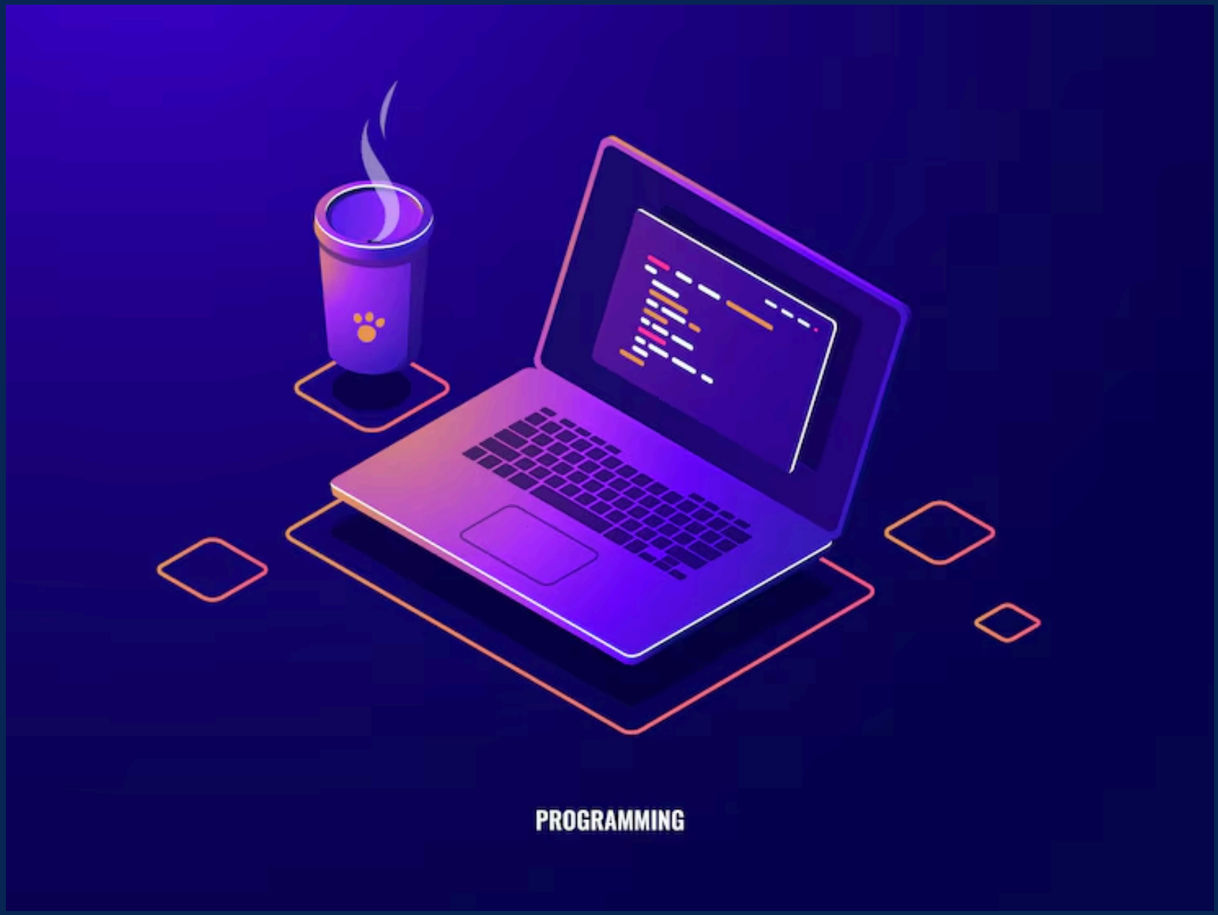
Complexité initiale accrue (pour un besoin simple) :

→ Parfois, un simple **if** ou un **switch** suffit si le nombre de variantes est faible et stable.

Compréhension pour les débutants :

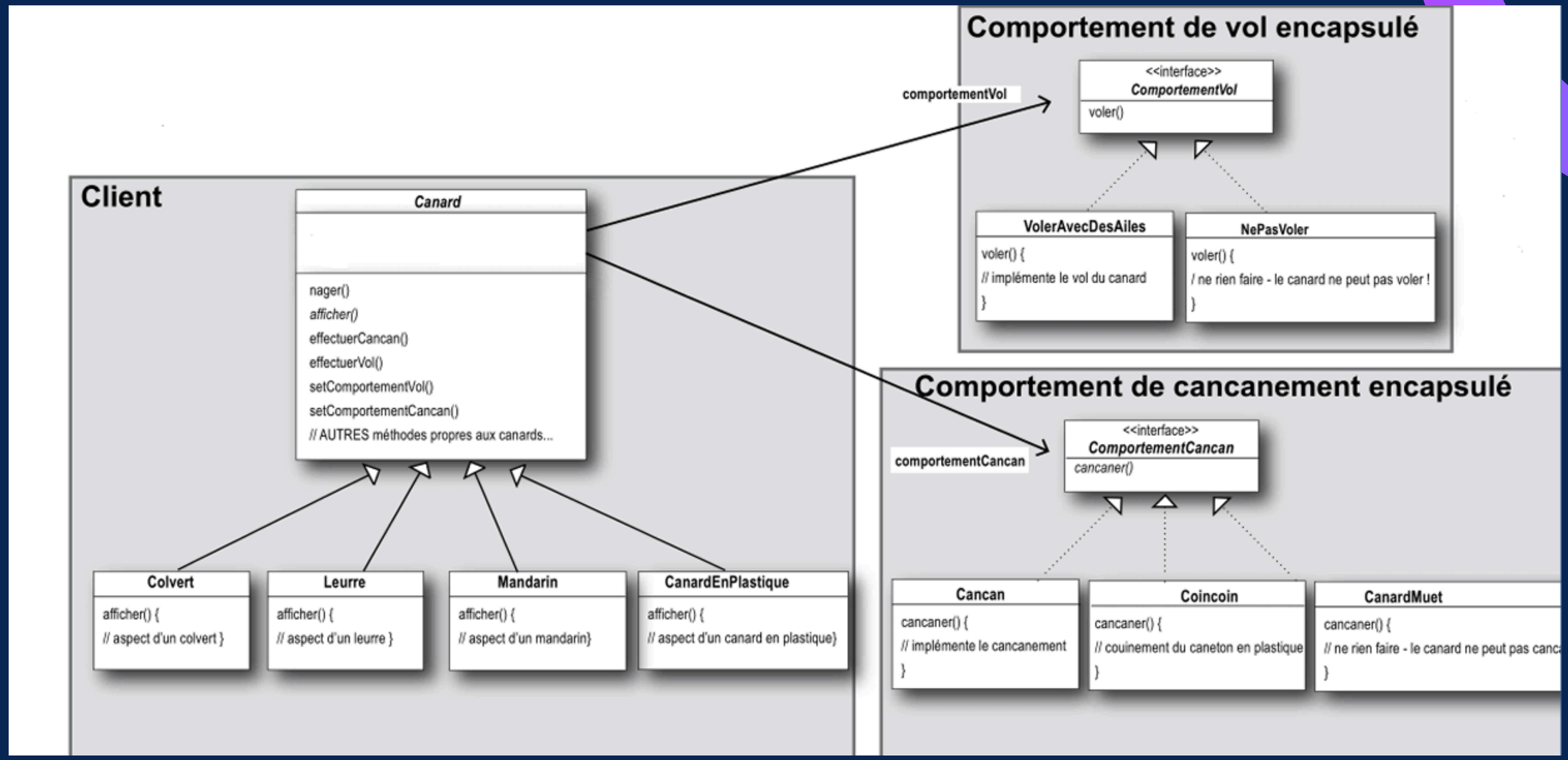
→ Structure plus **compliquée** pour un développeur non familier avec les patterns

LIVE CODING



https://youtu.be/cOK4RtvWftQ

DIAGRAMME DE CLASSES (CANARDS)



GUESS THE
CORRECT ANSWER
FORREST
GUMP
THE
TERMINATOR
JAWS
THE
WIZARD
OF OZ

QCM

CONCLUSION

BIBLIOGRAPHIE

github.com/iblasquez

Oracle Documentation — Design Patterns in Java

Refactoring Guru — Strategy Design Pattern

Baeldung — Strategy Pattern in Java