

# PATTERN VISITEUR

Justin PINHEIRO

Kévin WANG

Thibault DUFOUR

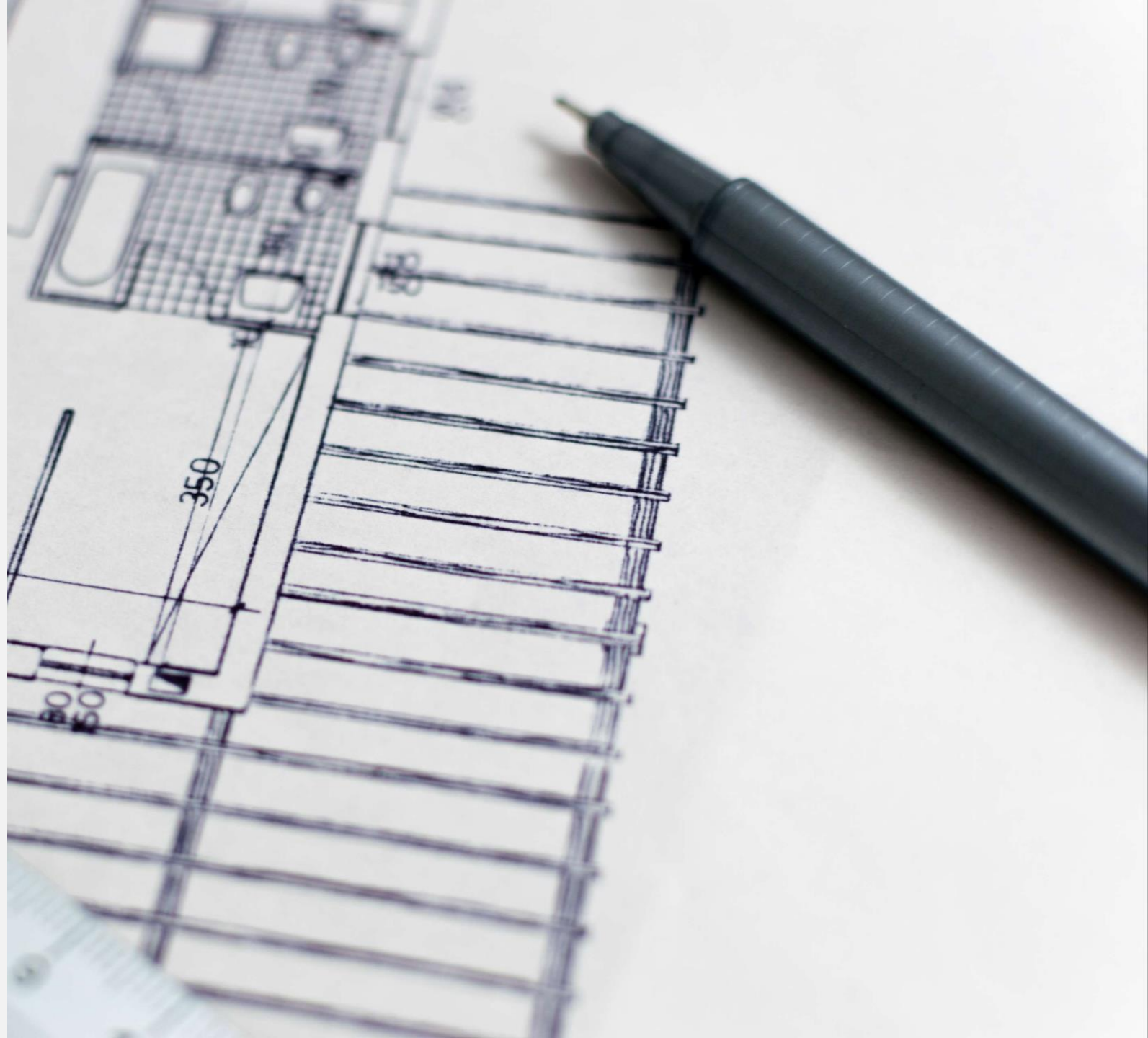
Romain ALLEMAND

# SOMMAIRE

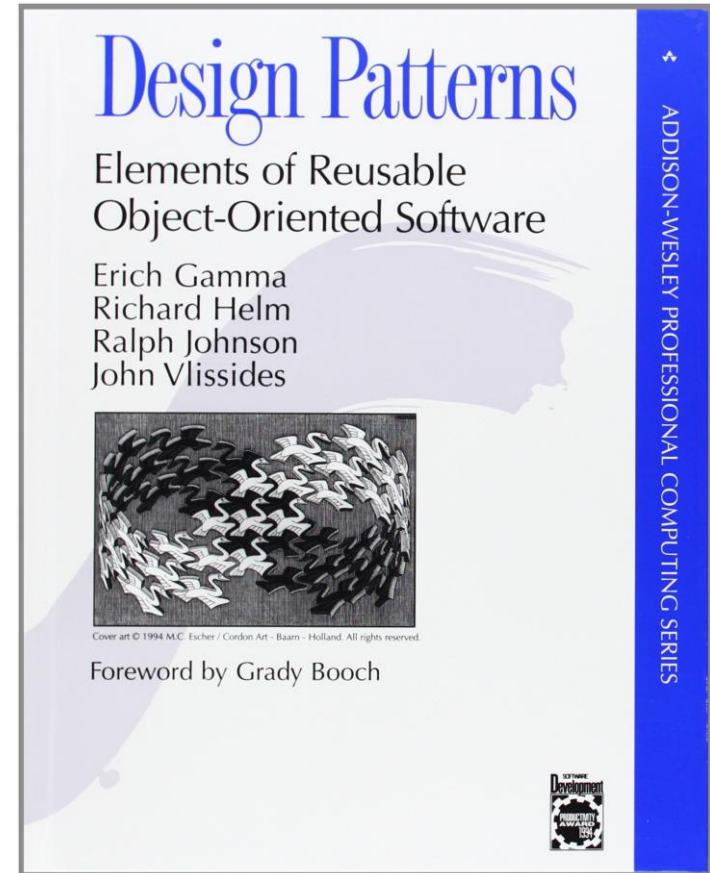
- I. Design patterns
- II. Pattern Visiteur
  - 1) Introduction par l'exemple
  - 2) Présentation du pattern
  - 3) Principes SOLID
- III. Limites et rapprochements

# I. DESIGN PATTERNS

- Solution classique à un problème récurrent de conception logiciel
- Nom
- Description du problème à résoudre
- Description de la solution
- Résultats issus de la solution



- *Design Patterns : Elements of Reusable Software*
- GoF (Gang of Four)



# PATTERN VISITEUR – INTRODUCTION PAR L'EXEMPLE

- Exemple : création d'un supermarché virtuel
- Implémentation d'une TVA différente pour chacun de nos produits

📦 Livre		
f 🔒	nom	String
f 🔒	auteur	String
f 🔒	prix_ht	float
m 📦	Livre(String, String, float)	

📦 Medicament		
f 🔒	nom	String
f 🔒	prix_ht	float
m 📦	Medicament(String, float)	

📦 Carburant		
f 🔒	nom	String
f 🔒	prix_ht	float
m 📦	Carburant(String, float)	

📦 Supermarche	
m 📦	Supermarche()



## PATTERN VISITEUR – INTRODUCTION PAR L'EXEMPLE

- Taux normal (20 %) : majorité des biens
- Taux réduit (5,5 %) : livres par exemple
- Taux particulier (2,1 %) : médicaments remboursables par la sécurité sociale)

- Première implémentation

```
public float prix_ttc()  
{  
    return prix_ht * 1.055f;  
}
```

```
public float prix_ttc()  
{  
    return prix_ht * 1.2f;  
}
```

Livre		
f	nom	String
f	auteur	String
f	prix_ht	float
m	Livre(String, String, float)	
m	prix_ttc()	

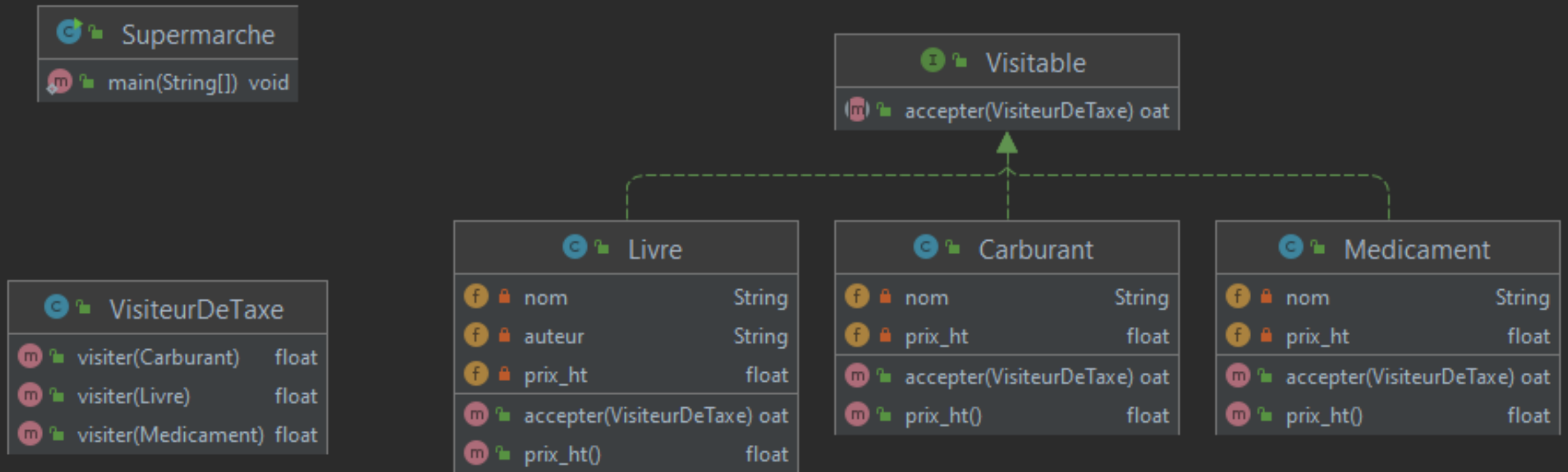
Medicament		
f	nom	String
f	prix_ht	float
m	Medicament(String, float)	
m	prix_ttc()	

Carburant		
f	nom	String
f	prix_ht	float
m	Carburant(String, float)	
m	prix_ttc()	

Supermarche		
m	Supermarche()	
m	main(String[]) void	



- Implémentation du pattern Visiteur





- Implémentation du pattern Visiteur

Supermarche  
main(String[]) void

Visitable  
accepter(VisiteurDeTaxe) oat

Livre

nom	String
auteur	String
prix_ht	float
accepter(VisiteurDeTaxe) oat	
prix_ht()	float

Carburant

nom	
prix_ht	
accepter(VisiteurDeTaxe)	
prix_ht()	

```
public class Medicament implements Visitable {
    private String nom;
    private float prix_ht;

    public Medicament(String nom, float prix_ht)
    {
        this.nom = nom;
        this.prix_ht = prix_ht;
    }

    public float prix_ht() { return prix_ht; }

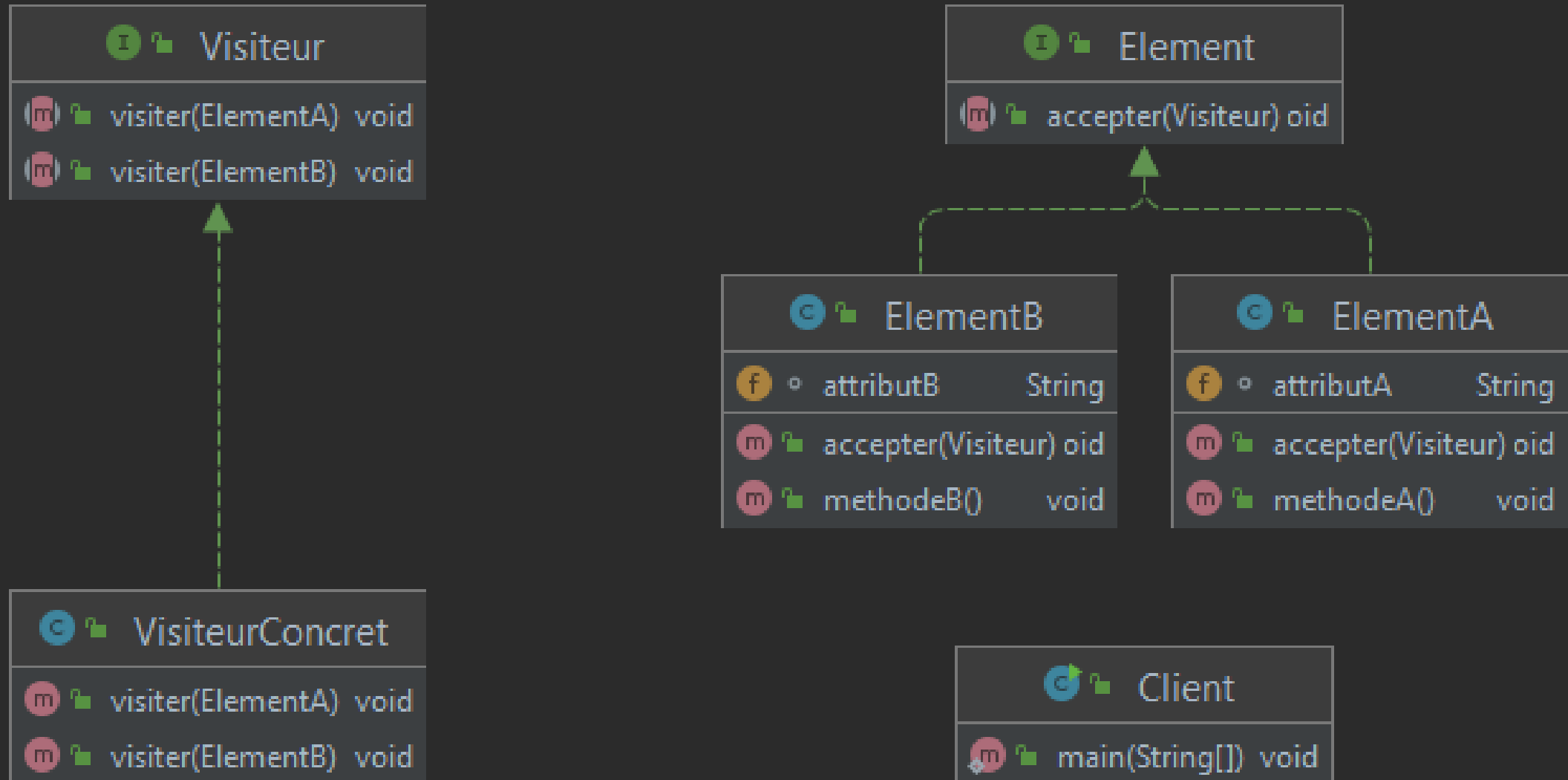
    @Override
    public float accepter(VisiteurDeTaxe visiteur)
    {
        return visiteur.visiter( medicament: this);
    }
}
```

```
public class VisiteurDeTaxe
{
    public float visiter(Carburant carburant)
    {
        return carburant.prix_ht() * 1.2f;
    }

    public float visiter(Livre livre)
    {
        return livre.prix_ht() * 1.055f;
    }

    public float visiter(Medicament medicament)
    {
        return medicament.prix_ht() * 1.021f;
    }
}
```

- Généralisation du pattern Visiteur



# PATTERN VISITEUR – PRÉSENTATION

## Principe

- Appartient à la catégorie des patterns comportementaux
- Sépare l'algorithme de la structure de l'objet sur lequel il travaille
  - Informe du type d'instances d'un ensemble de classes
  - Nouvelle classe dérivée pour de nouveau traitement

# PATTERN VISITEUR – PRÉSENTATION

## Intérêts

- Un code plus clair
- Des équipes différentes peuvent travailler dessus
- Pas obligé de tout recompiler à chaque fois

# PROBLEMATIQUE

Livre		
nom	String	
auteur	String	
prix_ht	float	
Livre(String, String, float)		
prix_ttc()	float	

Medicament		
nom	String	
prix_ht	float	
Medicament(String, float)		
prix_ttc()	float	

Carburant		
nom	String	
prix_ht	float	
Carburant(String, float)		
prix_ttc()	float	

Supermarche		
Supermarche()		
main(String[]) void		

Supermarche		
main(String[]) void		

VisiteurDeTaxe		
visiter(Carburant)	float	
visiter(Livre)	float	
visiter(Medicament)	float	

Livre		
nom	String	
auteur	String	
prix_ht	float	
accepter(VisiteurDeTaxe) oat		
prix_ht()	float	

Visitable		
accepter(VisiteurDeTaxe) oat		

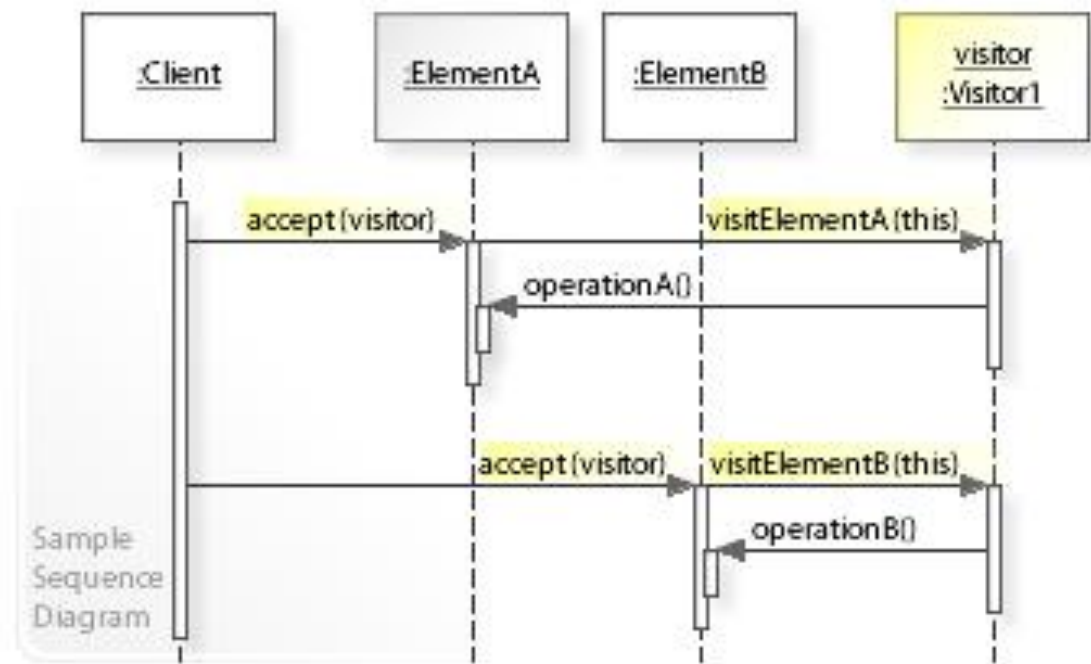
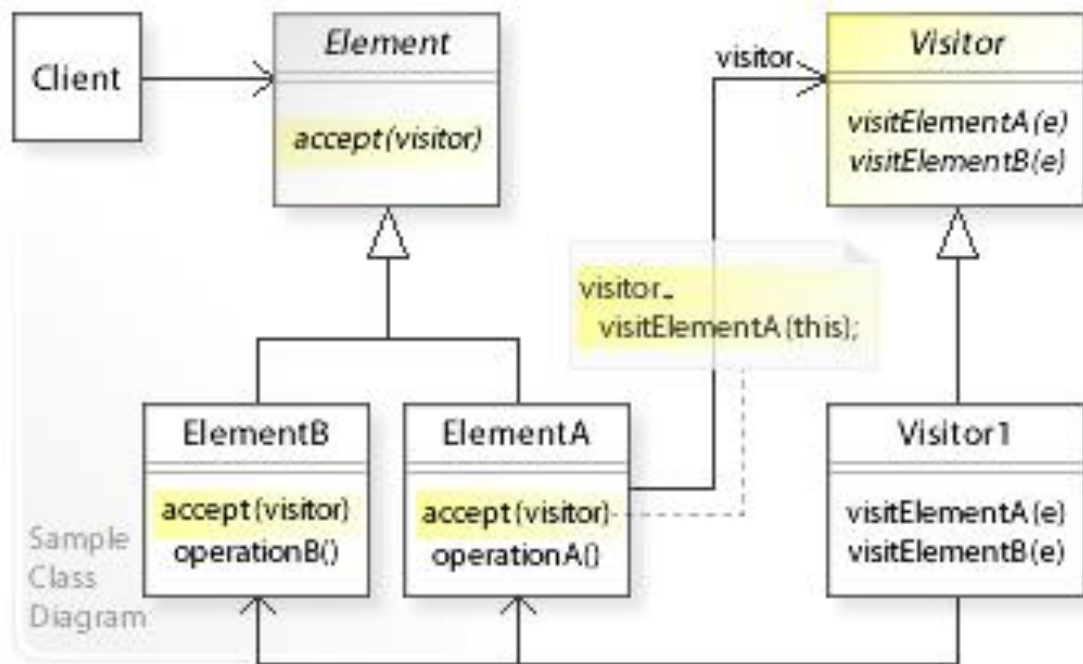
Carburant		
nom	String	
prix_ht	float	
accepter(VisiteurDeTaxe) oat		
prix_ht()	float	

Medicament		
nom	String	
prix_ht	float	
accepter(VisiteurDeTaxe) oat		
prix_ht()	float	



# LA SOLUTION

- ✓ • Ajouter un nouveau comportement ou une fonctionnalité dans une classe séparée appelée visiteur.
- ✗ • Inclure directement dans les classes existantes.



### Modèle de conception type

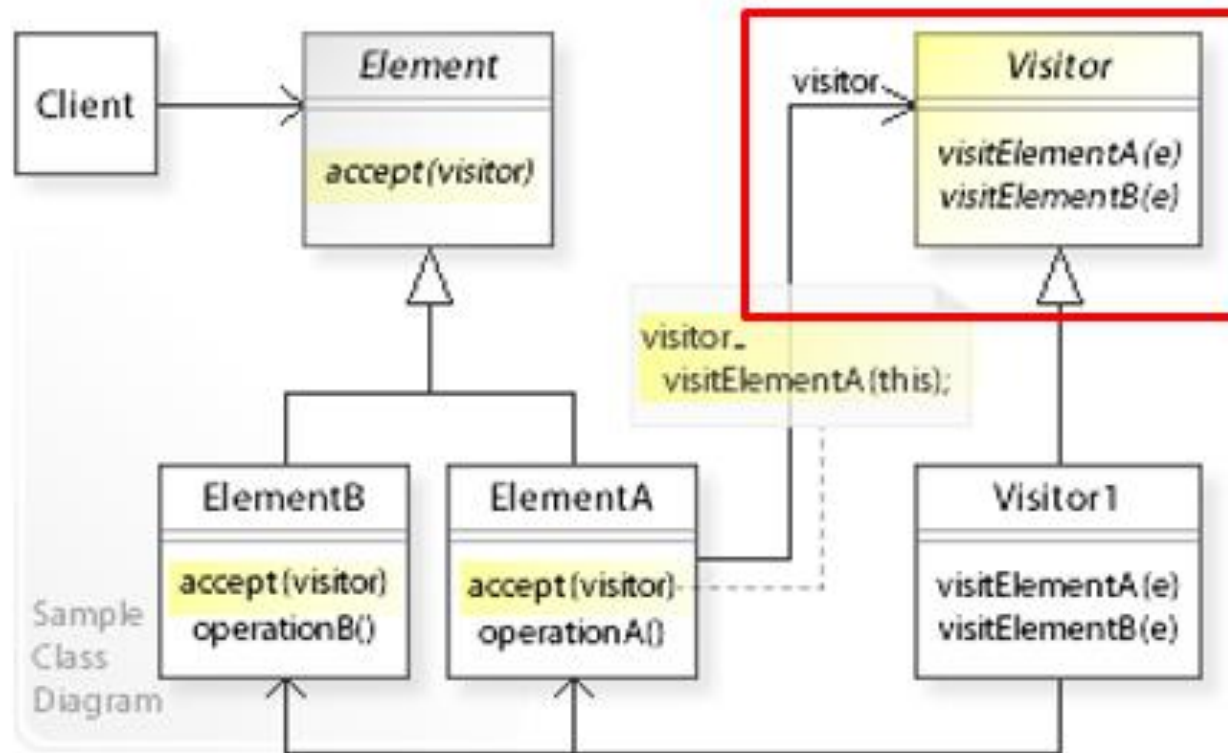
Le diagramme de classe	Le diagramme de séquences
Compilation Les classes et les relations	Exécution Les objets et les interactions entre eux.



# VISITOR

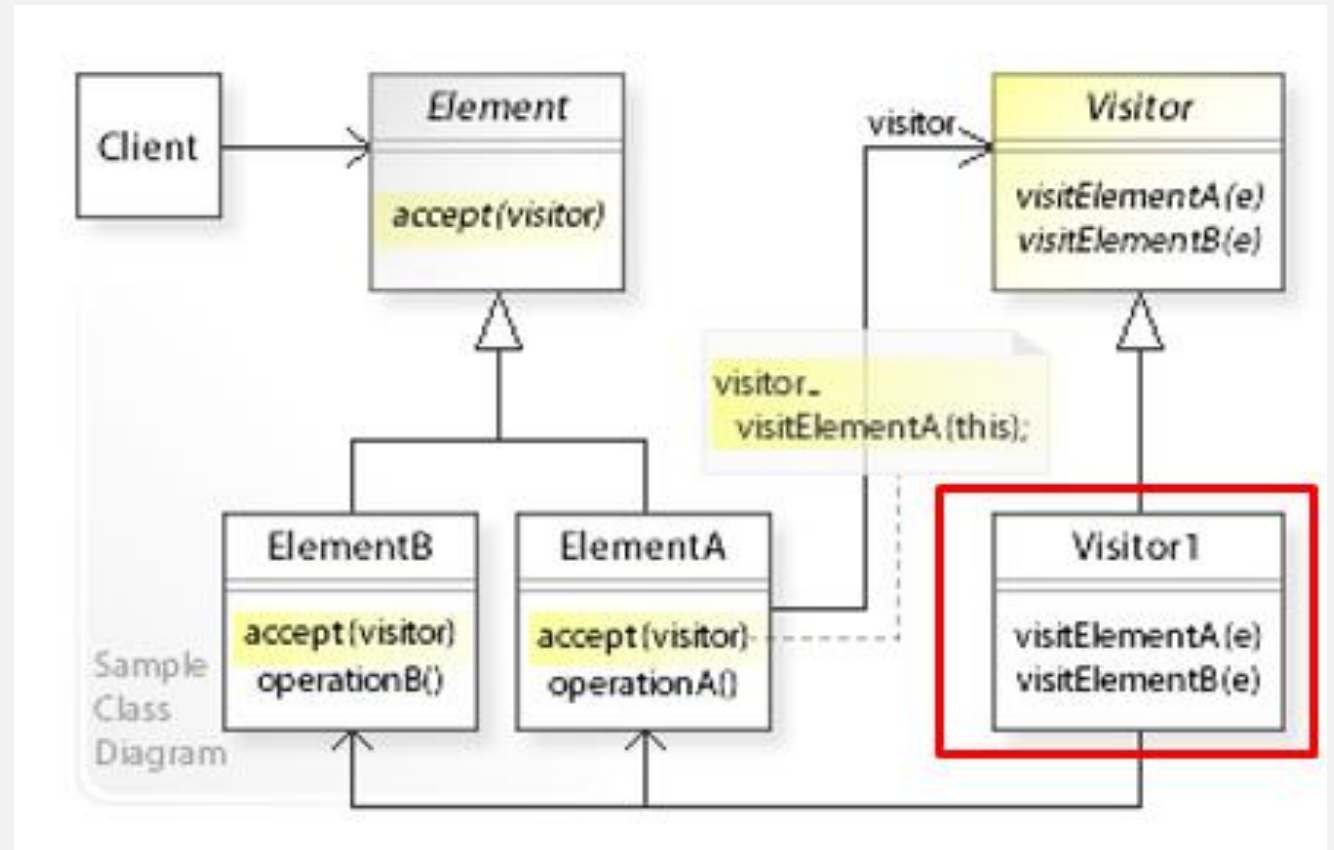
Visitor = interface

- méthodes de parcours
- méthode <- éléments concrets



# VISITOR I

- Visitor I = classe concrète  
=> Visiteur concret.
- Il implémente plusieurs versions des mêmes comportements en fonction des classes et des éléments concrets,
- Ex :Visiteur de taxe

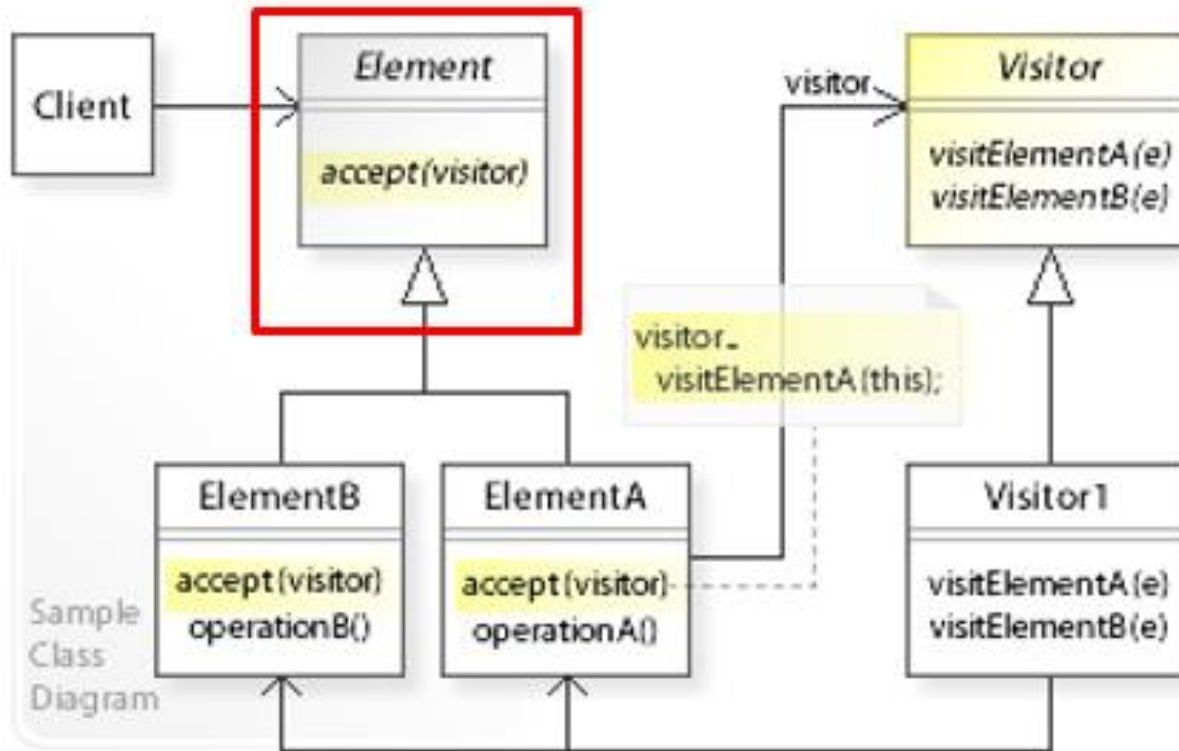


# ELEMENT

Élément = interface

- déclare une méthode qui « accepte » les visiteurs.
  - Prend en paramètre un visiteur

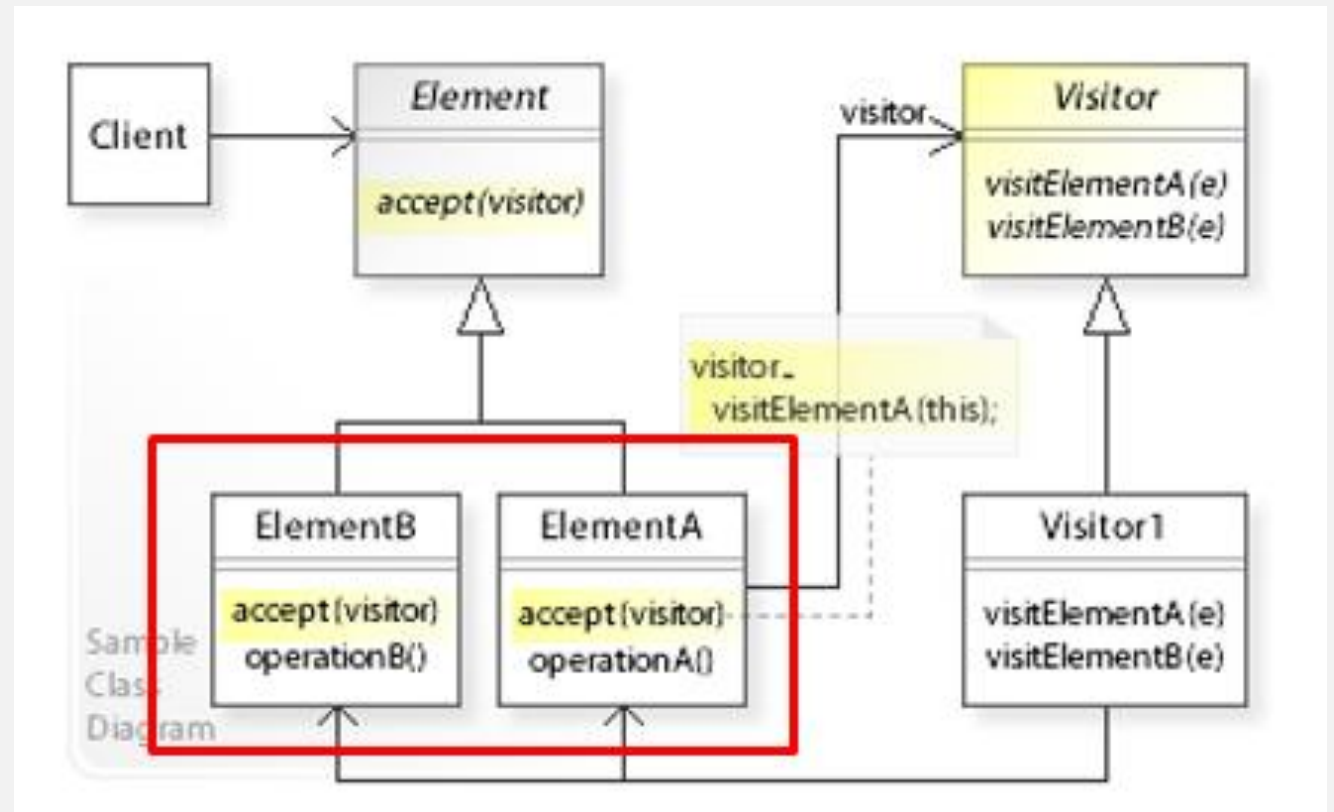
Ex : un produit



# ELEMENT A ELEMENT B

## Éléments Concrets

- Implémentent une méthode d'acceptation

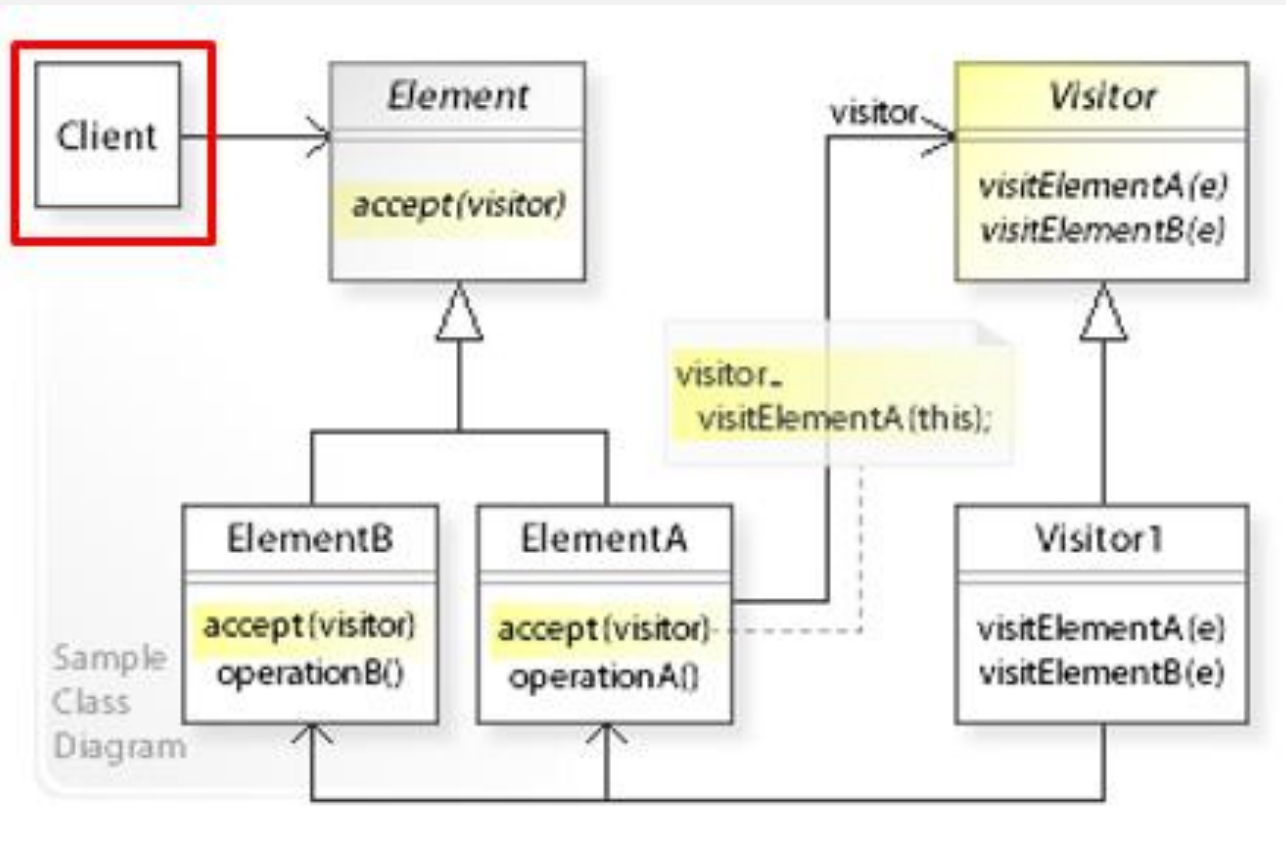


Ex: Les différents produits

# CLIENT

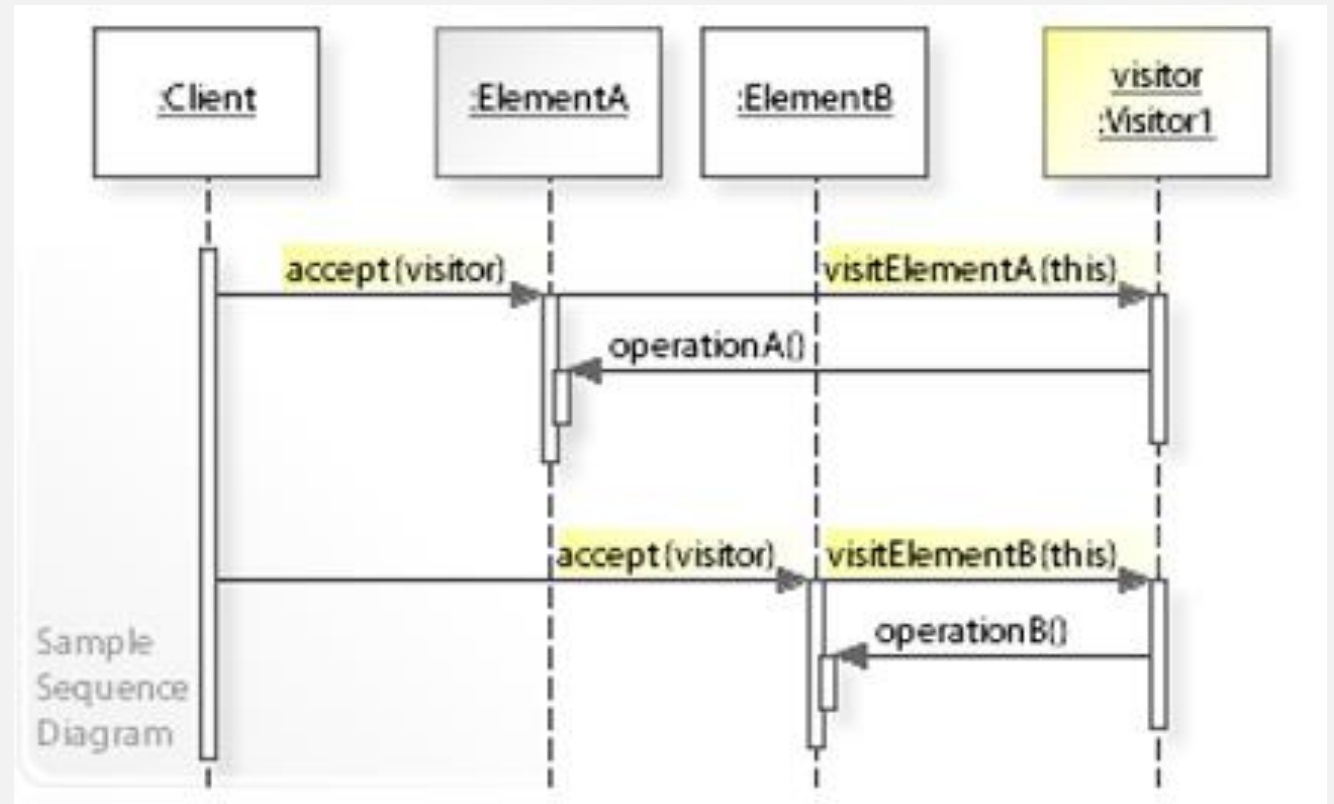
- souvent une collection ou autre objet complexe.
- Les clients n'ont pas de visibilité sur les classes des éléments concrets,
- ils manipulent via une interface abstraite.

Ex : le supermarché

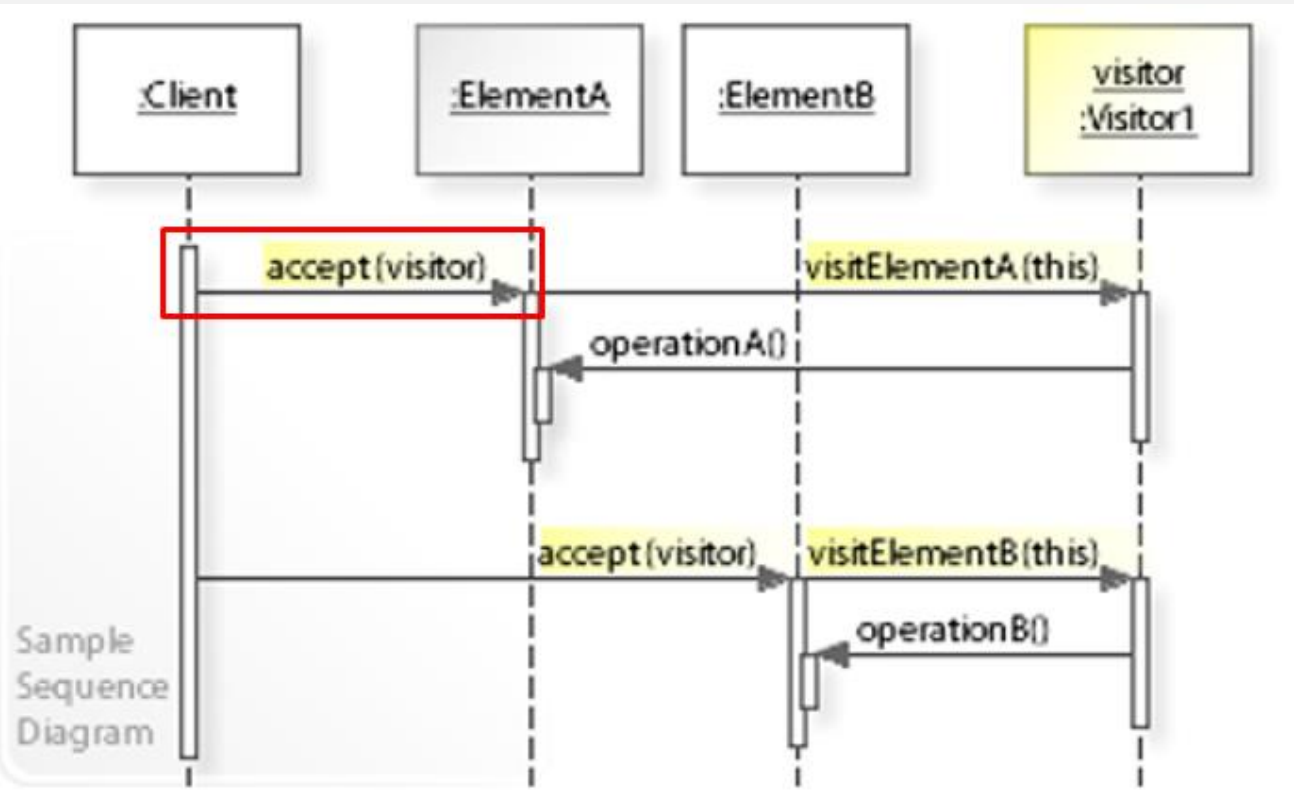


# EXEMPLE DE SCÉNARIO

- Un objet Client traverse les éléments A et B et appelle `accept(visitor)` sur chaque élément.



## EXEMPLE DE SCÉNARIO

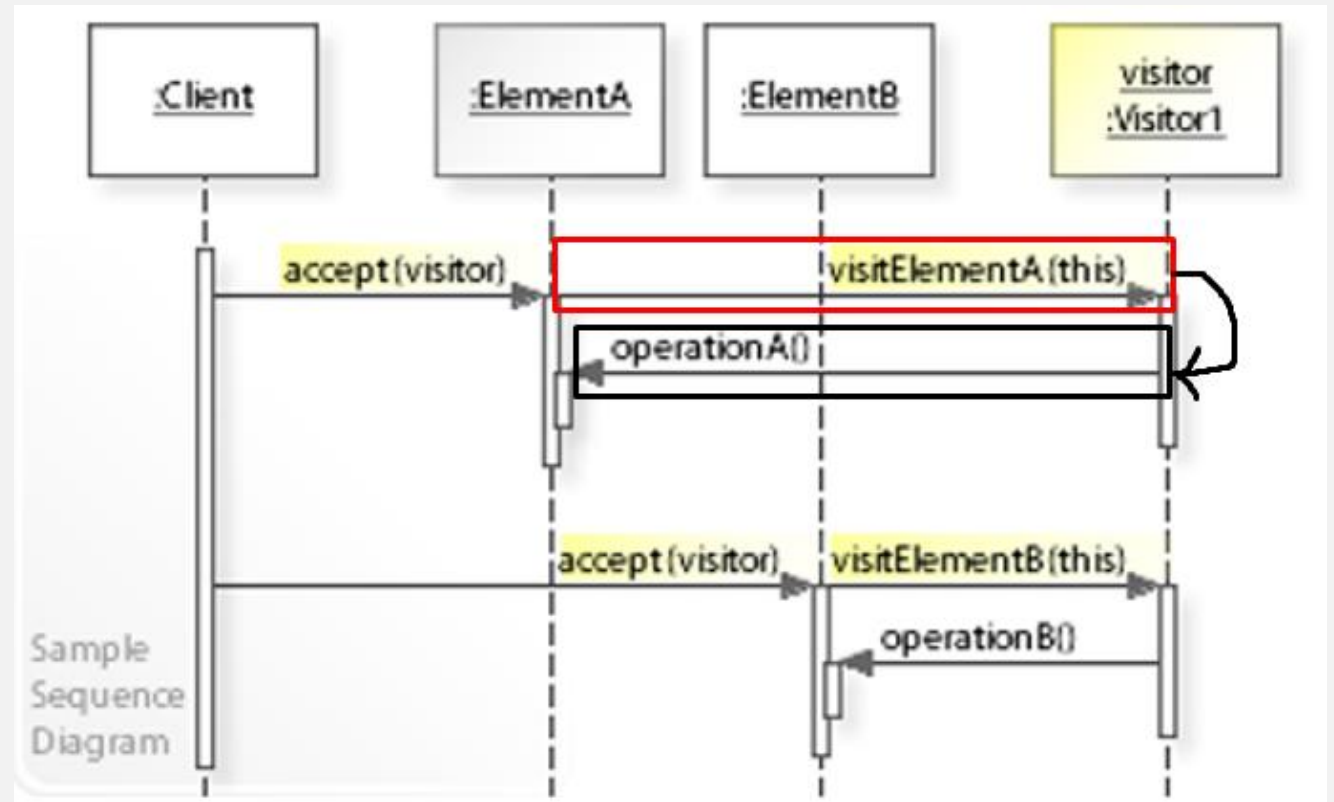


- Supposons que le Client fournisse un objet Visitor I.
- Alors il appelle `accept(visitor)` sur l'objet ElementA.

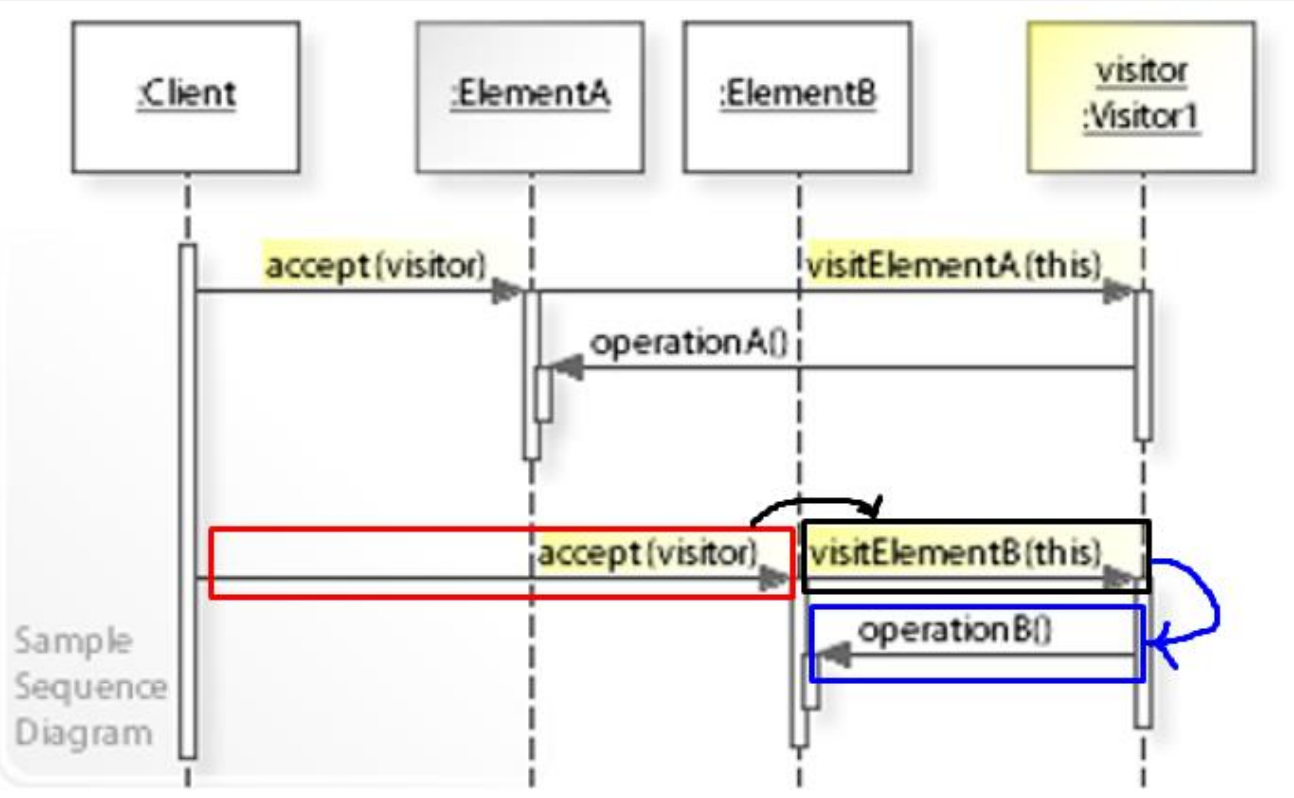


## EXEMPLE DE SCÉNARIO

- L'opération de délégation `accept(visitor)` de `ElementA` appelle `visitElementA(this)` sur l'objet `Visitor I`.
- Puis `Visitor I` visite `ElementA` et renvoie `operationA()`.



## EXEMPLE DE SCÉNARIO



- le Client appelle `accept(visitor)` sur `ElementB`, qui appelle `visitElementB(this)` sur l'objet `Visitor I`.
- Puis `Visitor I` revoie alors `operationB()`.

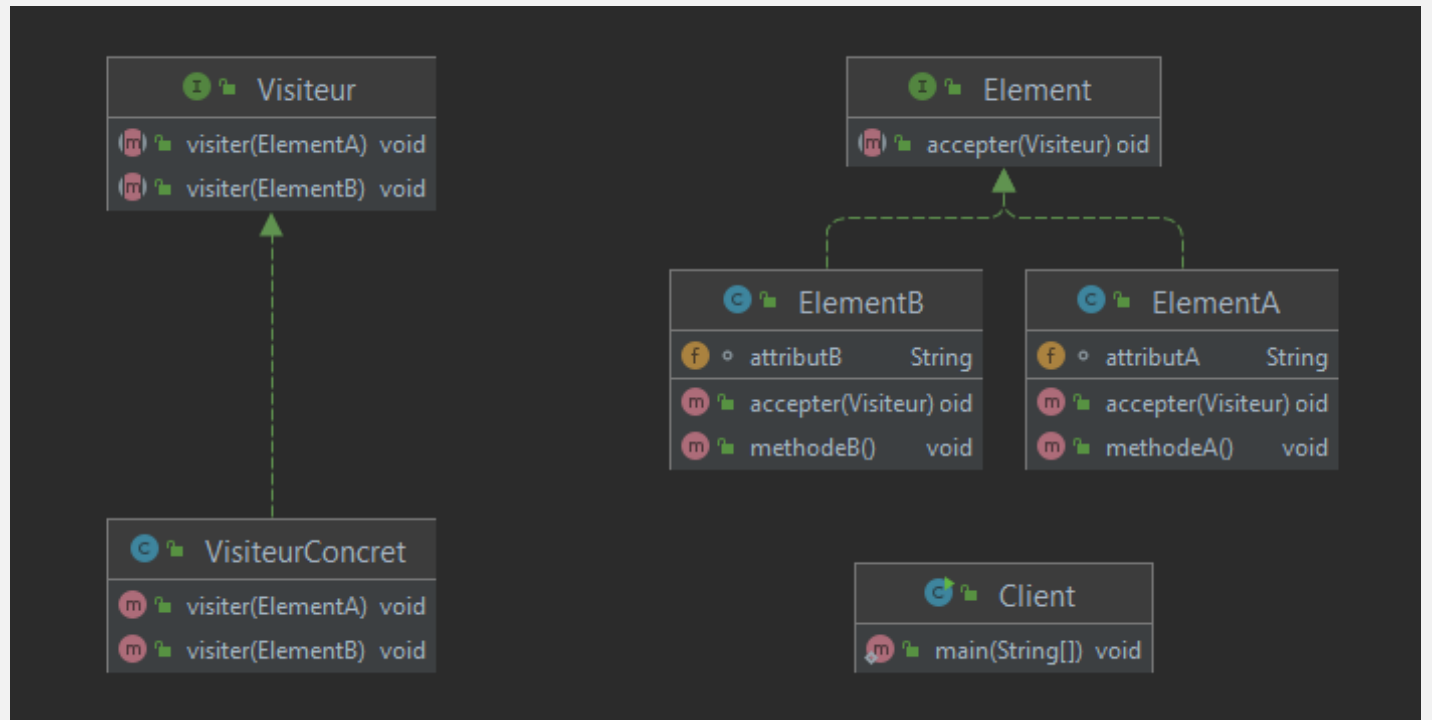


## SOLID

- ✓ • SRP (Single responsibility principle)
- ✓ • OCP (Open/closed principle)
- ✓ • LSP (Liskov substitution principle)
- ✓ • ISP (Interface segregation principle)
- ✓ • DIP (Dependency inversion principle)

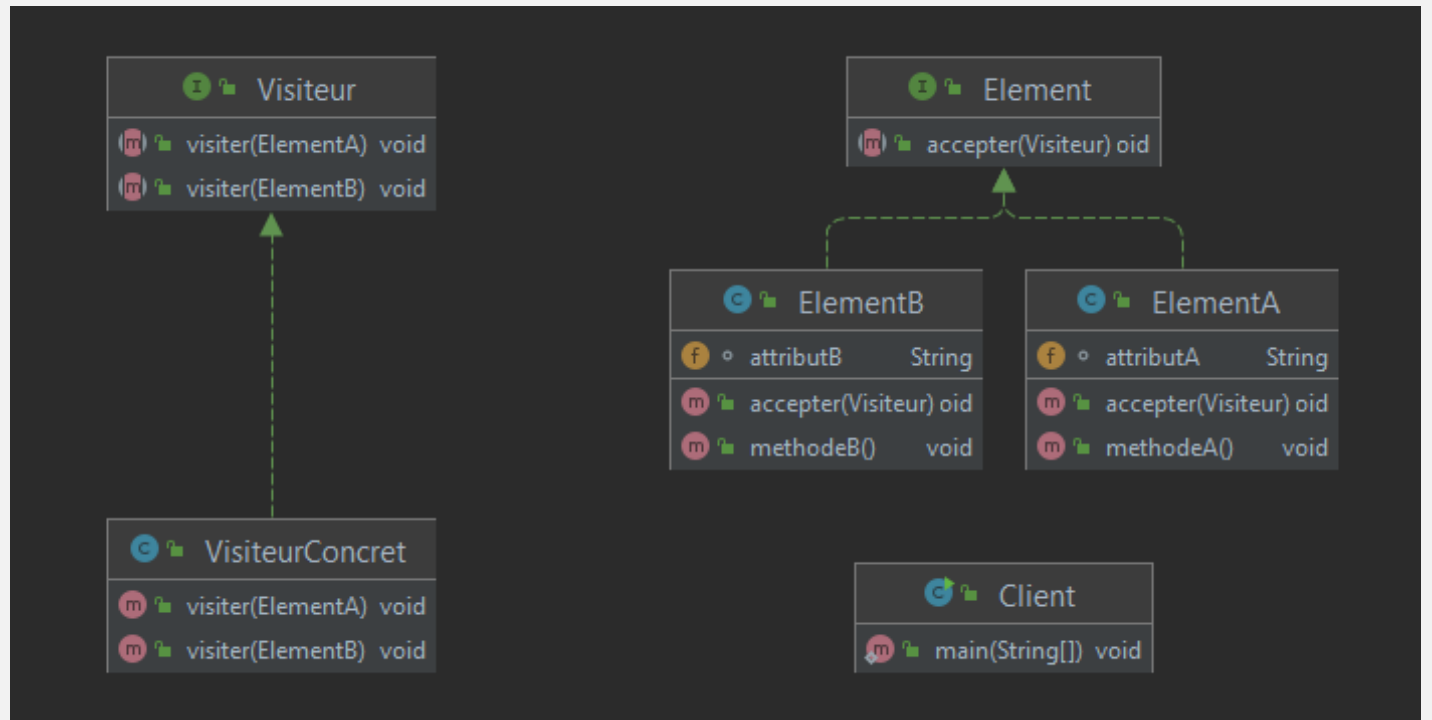
# SRP - SINGLE RESPONSIBILITY PRINCIPLE

- Chaque classe doit avoir une et une seule responsabilité



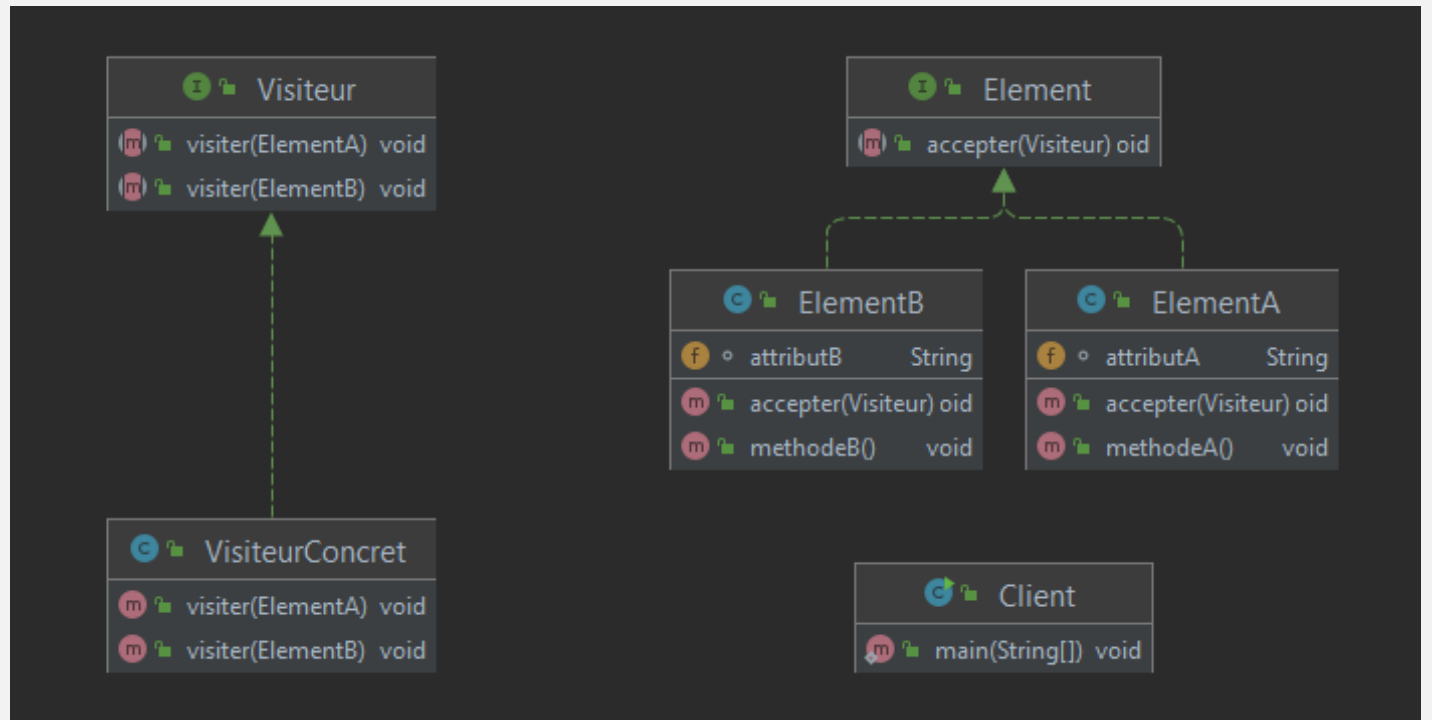
# OCP - OPEN/CLOSED PRINCIPLE

- Les entités sont ouvertes aux extensions mais fermées aux modifications



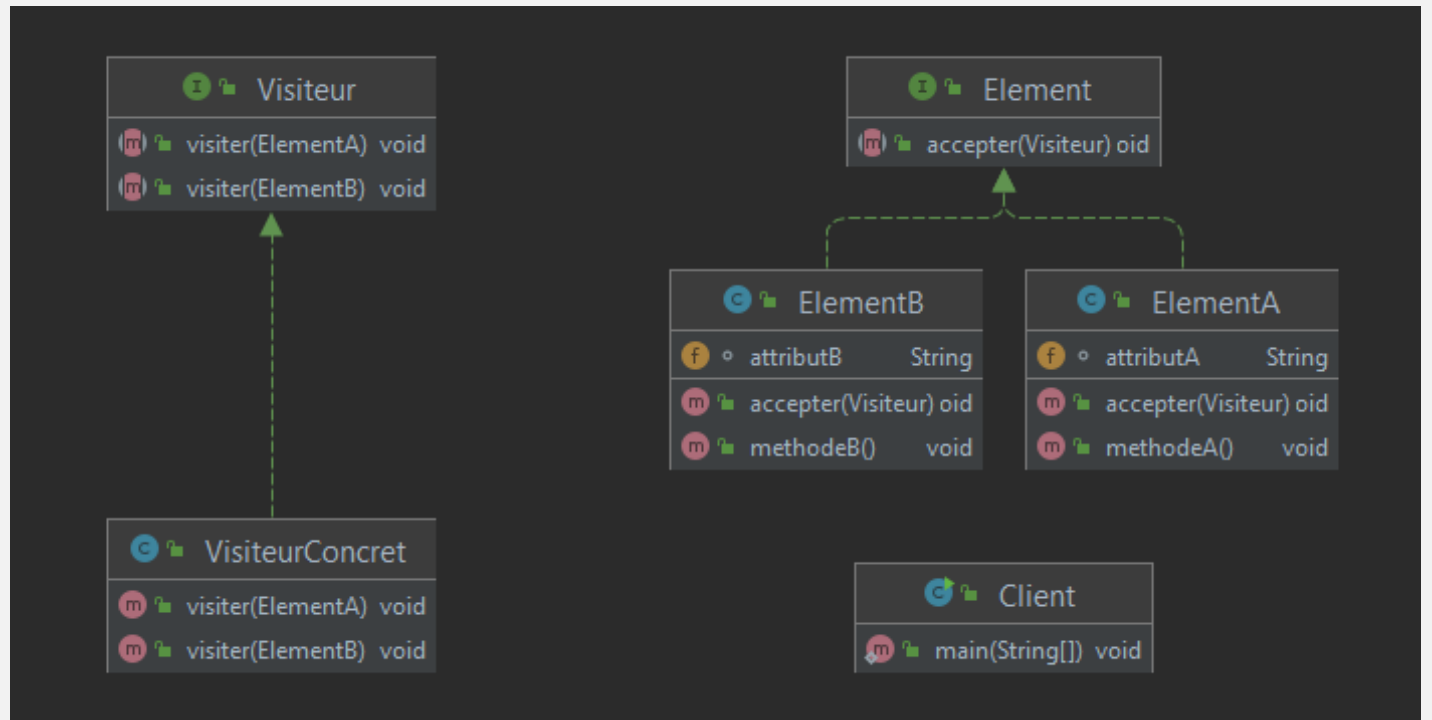
# LSP - LISKOV SUBSTITUTION PRINCIPLE

- Les sous-types doivent pouvoir être substitués à leurs types de base



# ISP - INTERFACE SEGREGATION PRINCIPLE

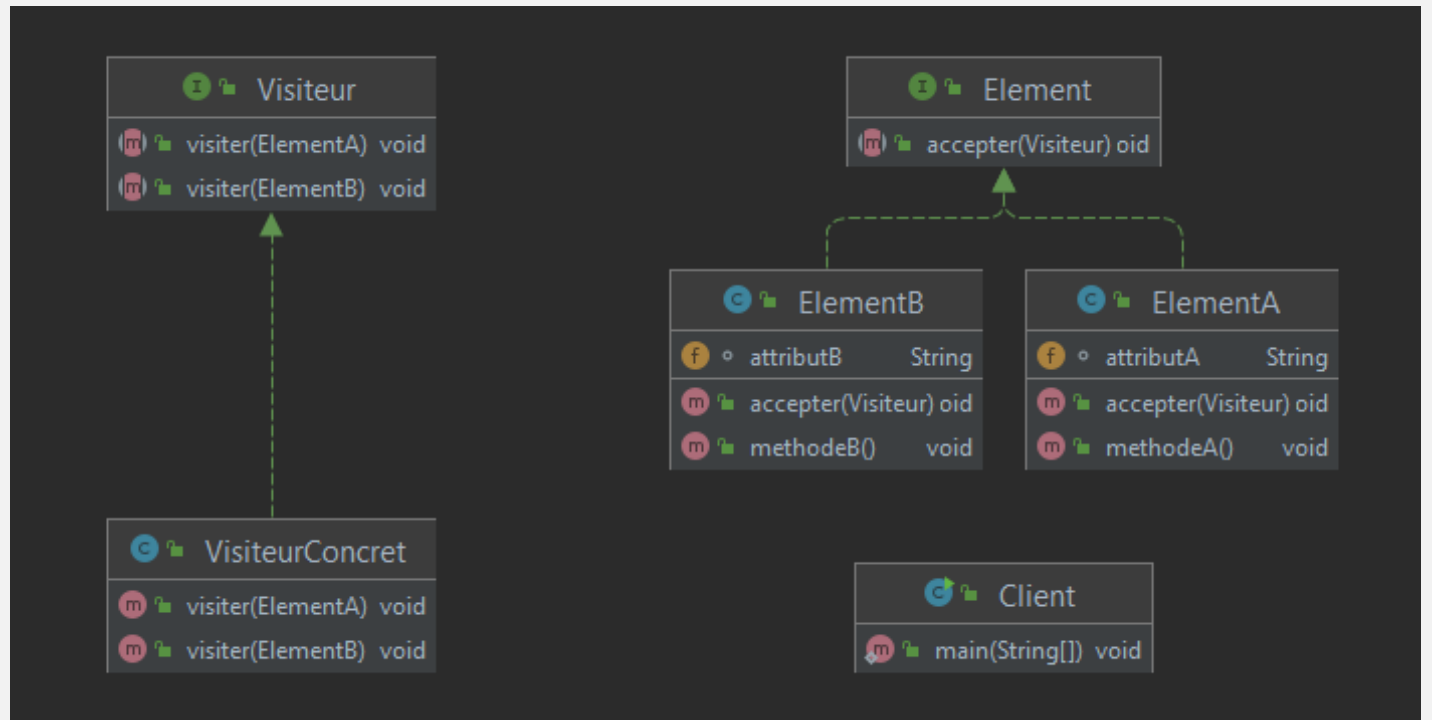
- Il vaut mieux avoir plusieurs interfaces spécifiques qu'une générale



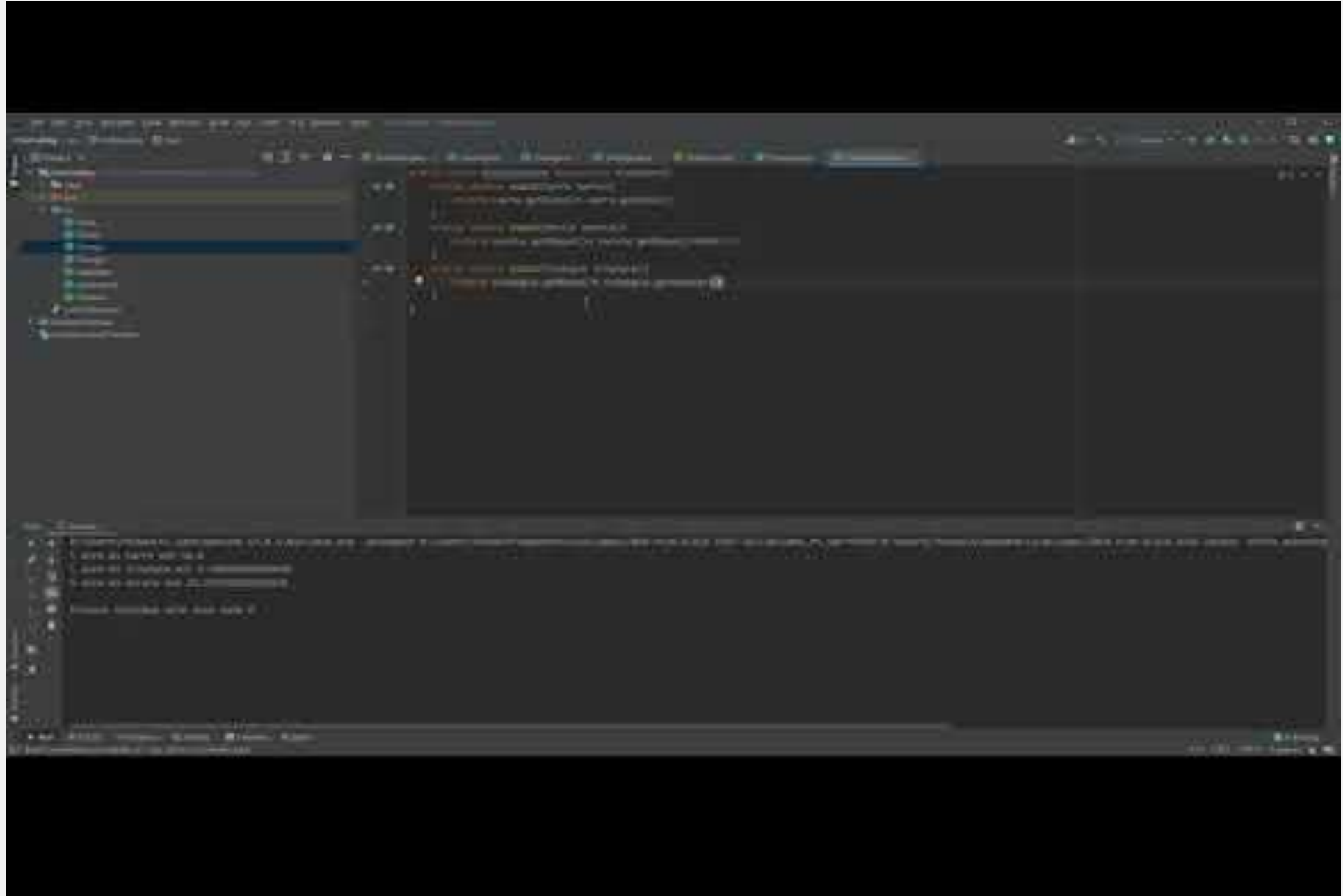


# DIP - DEPENDENCY INVERSION PRINCIPLE

- Les modules doivent dépendre d'abstractions, pas d'implémentations



LIVE CODING



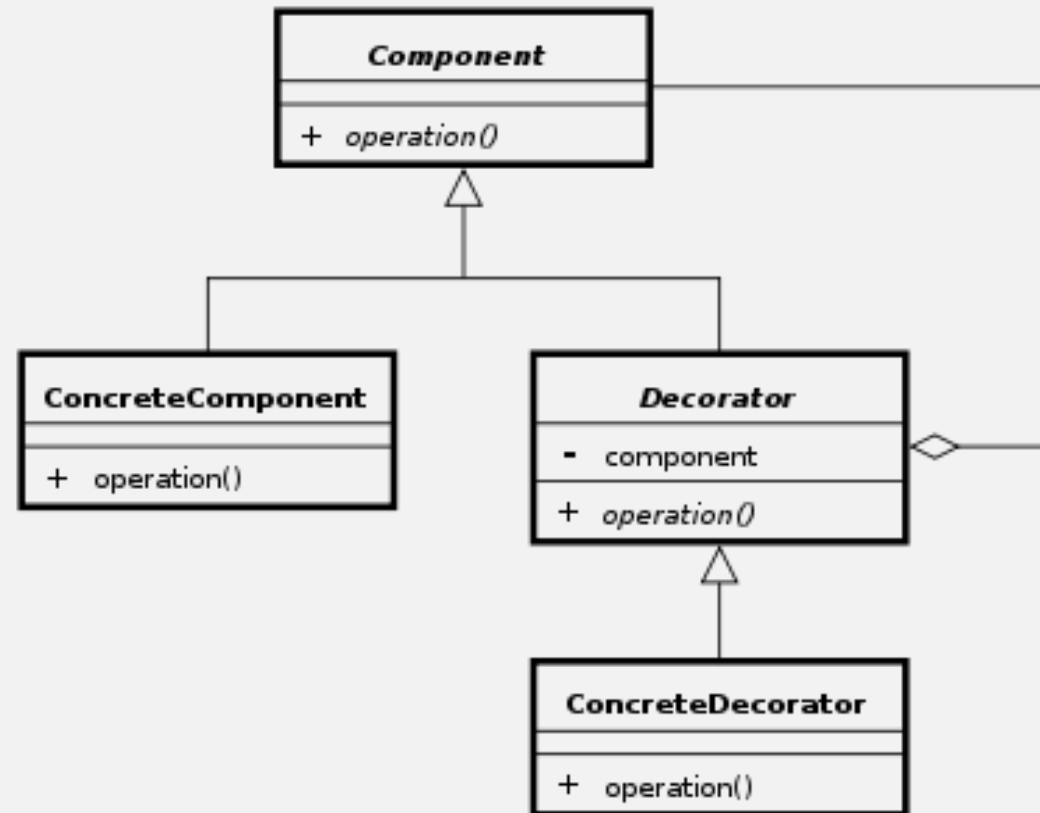
[https://www.youtube.com/watch?v=JT\\_W44foLsE&ab\\_channel=thibaultdufour](https://www.youtube.com/watch?v=JT_W44foLsE&ab_channel=thibaultdufour)

## PATTERN VISITEUR – LIMITES

- Rend la structure de la classe figée
- Problèmes d'accès aux attributs d'une classe

# RAPPROCHEMENT AVEC LE PATTERN DECORATEUR

- Création de classes externes pour ajouter des fonctionnalités



QCM

<https://tech.io/playgrounds/59260/quiz-design-pattern-visiteur>

# REFERENCES

- <https://www.ionos.fr/digitalguide/sites-internet/developpement-web/visitor-pattern/>
- <https://stackoverflow.com/questions/1673841/examples-of-gof-design-patterns-in-javas-core-libraries>
- [https://en.wikipedia.org/wiki/Visitor\\_pattern](https://en.wikipedia.org/wiki/Visitor_pattern)
- [https://fr.wikibooks.org/wiki/Patrons\\_de\\_conception/Visiteur](https://fr.wikibooks.org/wiki/Patrons_de_conception/Visiteur)
- <https://deptmedia.cnam.fr/new/spip.php?pdoc11796>
- <https://www.codingame.com/playgrounds/8339/design-pattern-visitor/exemple-dutilisation>
- <http://w3sdesign.com/?gr=b11&ugr=struct#gf>
- <https://www.youtube.com/watch?v=VI3TJQqCUvM>
- <https://www.youtube.com/watch?v=vG9gEI1dr1o>
- <https://alexsoyes.com/solid/>
- <https://refactoring.guru/fr/design-patterns/visitor/java/example>
- [https://www.youtube.com/watch?v=JT\\_W44foLsE](https://www.youtube.com/watch?v=JT_W44foLsE)