

Design Pattern Visitor

Plan

1) Les Design Pattern

- a) Introduction au design Pattern
- b) Définition

2) Le Pattern Visitor

- a) Principe/Fonctionnement du Pattern Visitor
- b) Schéma
- c) Avantages/Désavantages du Pattern Visitor

3) Exemple de Pattern Visitor

- a) Implémentation



Les Design Pattern

Introduction

- Pattern :
 - En général : événement observé de façon répété
- Design Pattern (patron de conception)
 - 1970 : Christopher Alexander
 - 1995 : Le “Gang of Four” (GoF)
 - *Design Patterns – Elements of Reusable Object-Oriented Software*
 - 23 patterns recensés



Les Design Pattern

Définition

- Solution à un problème récurrent dans la conception d'applications et logiciels.
- Intérêt : ajouter des fonctionnalités à une classe sans pour autant la modifier.
- 3 familles : Création, Structure, Comportements
- 4 caractéristiques pour chacun d'eux :
 - Un nom
 - Une problématique
 - Une solution
 - Des conséquences



Le Pattern Visitor

Principe

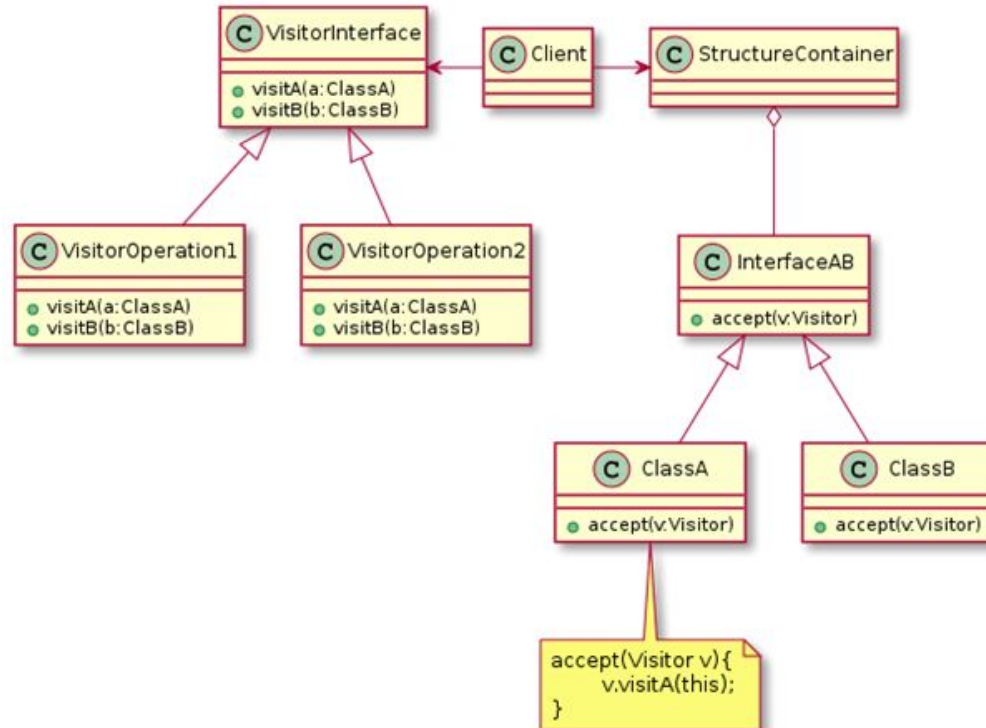
- Appartient à la famille des patterns comportementaux
- Séparer les données et leur traitements
- Attention : l'intention du visiteur n'est pas de visiter, mais de définir une nouvelle opération sans changer le code existant sur lequel il agit.

Fonctionnement

- Les classes objet ont une méthode publique nommé accept, cette méthode prend argument un objet de type visitor et appelle la méthode visiter. Les classes visitors connaissent grâce à cela la référence de l'objet. Cela permet de faire appelle aux méthodes publiques de cet objet.

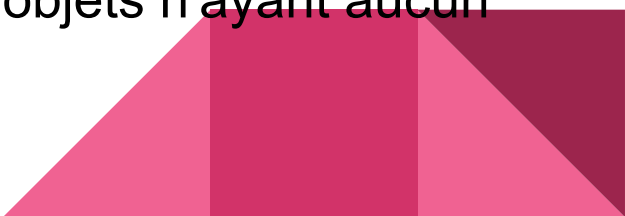
Le Pattern Visitor

Schéma



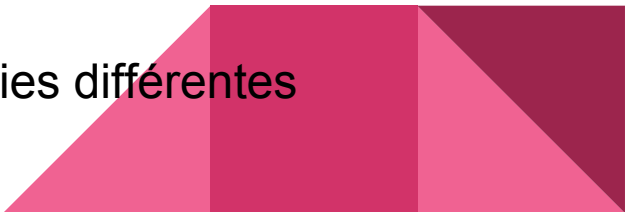
Le Pattern Visitor

Avantages

- Ajout de nouveau traitement sans toucher à la hiérarchie.
 - Code plus clair
 - Plusieurs personnes peuvent travailler sur des fonctionnalités différentes sans se gêner
 - On ne recompile que le code visiteur car le reste n'est pas modifié
 - Implémenter les mêmes méthodes sur des objets n'ayant aucun lien entre eux
- 

Le Pattern Visitor

Désavantages

- Le type de retour et les paramètres des méthodes doivent être connus à l'avance et il est déconseillé de les changer.
 - La hiérarchie doit être conçue avec l'utilisation du pattern Visitor. Sinon, il sera impossible de l'ajouter après.
 - Code difficilement réutilisable dans des hiérarchies différentes
- 

Exemple de Pattern Visitor

Implémentation

```
public interface Visitor {  
    public double visit(Presse presse);  
    public double visit(Necessite necessite);  
}
```



```
public interface Visitable {  
    public double accept(Visitor visitor);  
}
```

```
public class Presse implements Visitable {  
  
    private double prix;  
  
    public Presse(double prix) {  
  
        this.prix = prix;  
    }  
  
    public double getPrix() {  
        return prix;  
    }  
  
    public double accept(Visitor visitor) {  
  
        return visitor.visit(this);  
    }  
}
```

```
public class Necessite implements Visitable {  
  
    private double prix;  
  
    public Necessite(double prix) {  
  
        this.prix = prix;  
    }  
  
    public double getPrix() {  
        return prix;  
    }  
  
    public double accept(Visitor visitor) {  
  
        return visitor.visit(this);  
    }  
}
```

```
public class TaxVisitor implements Visitor {  
  
    public TaxVisitor() {  
  
    }  
  
    public double visit(Necessite necessite) {  
        System.out.println("Produit de premiere necessite: prix avec taxe");  
        return (necessite.getPrix() * .20) + necessite.getPrix();  
    }  
  
    public double visit(Presse presse) {  
        System.out.println("Presse: prix avec taxe");  
        return (presse.getPrix() * .05) + presse.getPrix();  
    }  
  
}
```



```
public class VisitorTest {  
    public static void main(String[] args) {  
        TaxVisitor taxeCalc =new TaxVisitor();  
  
        Necessite lait= new Necessite(2.10);  
        Presse lecho=new Presse(0.99);  
  
        System.out.println(lait.accept(taxeCalc)+"/n");  
        System.out.println(lecho.accept(taxeCalc)+"/n");  
    }  
}
```

prix du lait avec taxe: 2.52

prix du lait avec taxe: 1.0395

```
public class Presse {  
  
    private double prix;  
  
    public Presse(double prix) {  
        this.prix = prix;  
    }  
  
    public double prixAvecTaxe(double prix) {  
        return this.prix * .05 + this.prix;  
    }  
  
    public double getPrix() {  
        return prix;  
    }  
}
```

```
public class Necessite {  
  
    private double prix;  
  
    public Necessite(double prix) {  
        this.prix = prix;  
    }  
  
    public double prixAvecTaxe(double prix) {  
        return this.prix * .20 + this.prix;  
    }  
  
    public double getPrix() {  
        return prix;  
    }  
}
```

```
public class testNoDesign {  
    public static void main(String[] args) {  
        Necessite lait= new Necessite(2.10);  
        Presse lecho=new Presse(0.99);  
        System.out.println("prix du lait avec taxe: "+lait.prixAvecTaxe(lait.getPrix()));  
        System.out.println("prix du lait avec taxe: "+lecho.prixAvecTaxe(lecho.getPrix()));  
    }  
}
```

prix du lait avec taxe: 2.52

prix du lait avec taxe: 1.0395

