DECORATEUR

DESIGN PATTERNS

DESIGN PATTERNS : DEFINITION

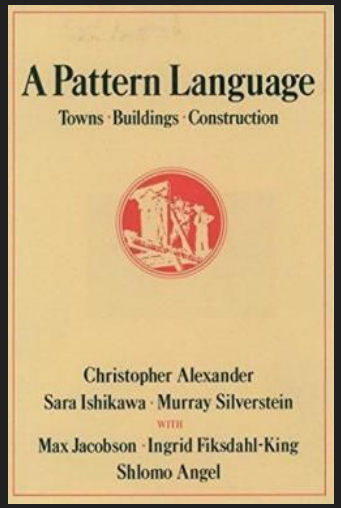
Problèmes de développement récurrents --> Invention Design Pattern

Donne une idée abstraite des solutions

Utilisable dans tous les langages

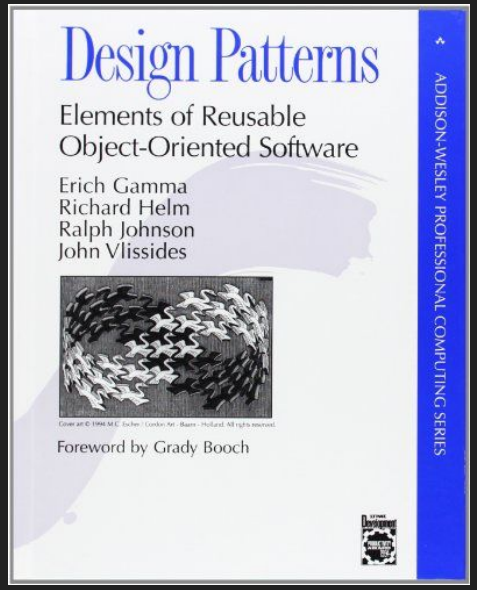
Facile à implementer

DESIGN PATTERNS : HISTOIRE



Dans les 1970

Paru en 1995



Patterns de création

- Abstract Factory
- Factory
- Builder
- Prototype
- Singleton

Patterns de structuration

- Facade
- Decorator
- Composite
- Flyweight
- Proxy
- Adaptor
- Bridge

Patterns de comportement

- Command
- Visiteur
- Observer
- State
- Strategy

MISTER O'TACOS

LES TACOS

GOUTEZ LA DIFFERENCE

(Boissons comprises)

SIMPLE (1 viande) 5,50

DOUBLE (2 viandes) 7,50

MAXI (3 viandes) 8,50

Viande

- Escalope poulet
- Viande hachée
- Cordon bleu
- Tenders
- Chicken Tandoori
- Chicken Curry
- Grec
- Poulet pané

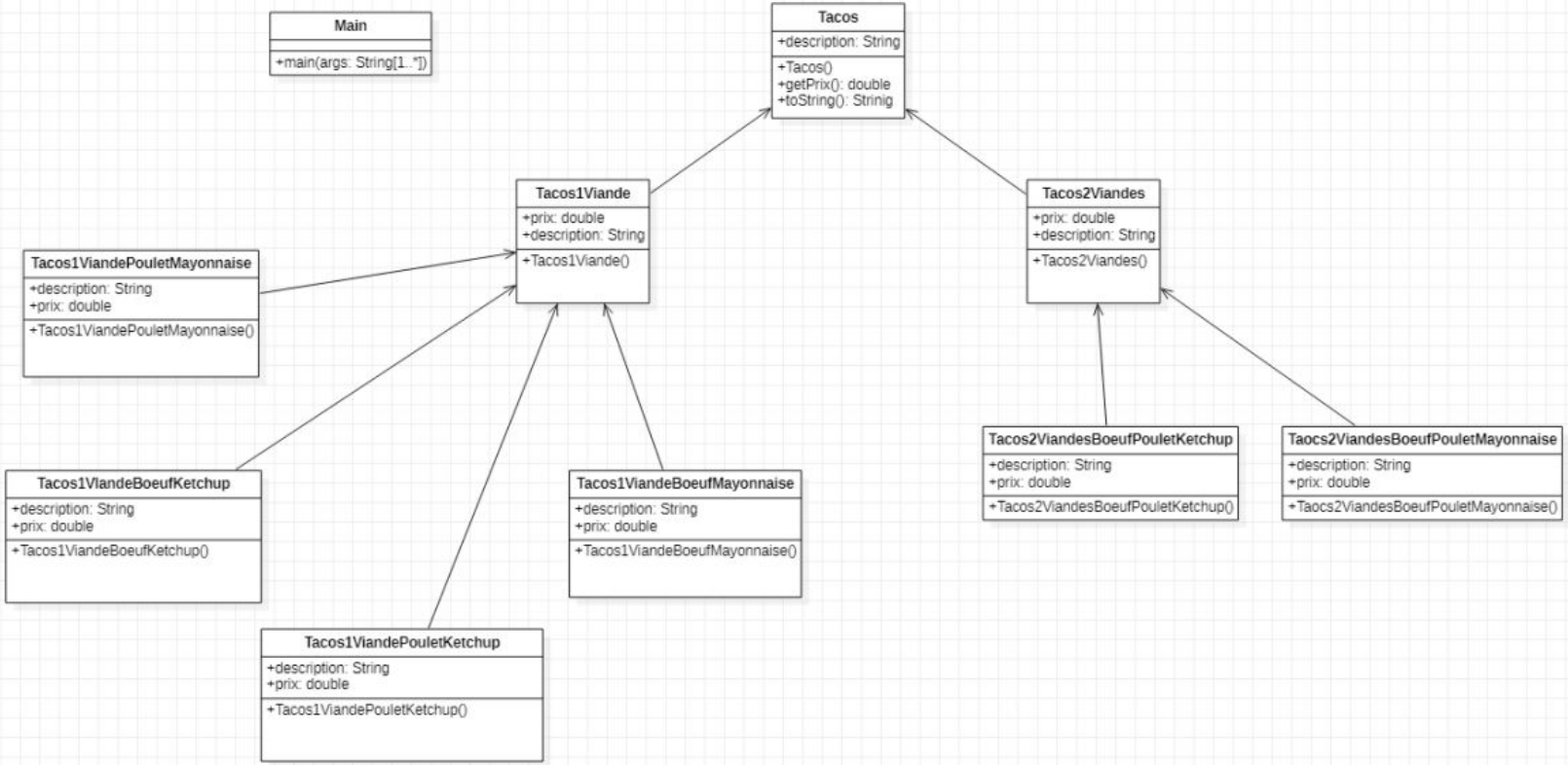
Sauce

- Algérienne
- Mayonnaise
- Curry
- Cheezy Easy
- Barbecue
- Blanche
- Fish to fish
- Biggy Burger
- Poivre
- Chili thaï
- Samuraï
- Harissa

Suppléments au choix

- FROMAGE
- BOURSIN 0,50
- CHEVRE
- FROMAGE RAPE
- OEUF

- GALETTE de POMME de Terre
- LARDON
- BACON 0,70
- POULET


Main
+main(args: String[1..*])
Tacos
+description: String
+Tacos()
+getPrix(): double
+toString(): Stringig
Tacos1Viande
+prix: double
+description: String
+Tacos1Viande()
Tacos2Viandes
+prix: double
+description: String
+Tacos2Viandes()
Tacos1ViandePouletMayonnaise
+description: String
+prix: double
+Tacos1ViandePouletMayonnaise()
Tacos1ViandeBoeufKetchup
+description: String
+prix: double
+Tacos1ViandeBoeufKetchup()
Tacos1ViandeBoeufMayonnaise
+description: String
+prix: double
+Tacos1ViandeBoeufMayonnaise()
Tacos1ViandePouletKetchup
+description: String
+prix: double
+Tacos1ViandePouletKetchup()
Tacos2ViandesBoeufPouletKetchup
+description: String
+prix: double
+Tacos2ViandesBoeufPouletKetchup()
Taocs2ViandesBoeufPouletMayonnaise
+description: String
+prix: double
+Taocs2ViandesBoeufPouletMayonnaise()

```java
public abstract class Tacos {
        protected String description= null;
        protected double prix;

        public String getDescription() {
            return this.description;
        }

        public Double getPrix() {
            return this.prix;
        }

        public String toString() {
            return "Le tacos "+this.description+" coûte "+this.prix+" euros.";
        }
}
```

```java
public class Tacos1ViandeBoeufKetchup extends Tacos1Viande {


    public Tacos1ViandeBoeufKetchup(String description, double prix) {
        super(description, prix);
        this.description=description+" 1 viande : boeuf+ketchup";

    }
}
```

```java
public class Tacos1ViandeBoeufMayonnaise extends Tacos1Viande {

    public Tacos1ViandeBoeufMayonnaise(String description, double prix) {
        super(description, prix);
        this.description=description+"1 viande : boeuff+ketchup";
    }
}
```

```java
public class Tacos1ViandePouletKetchup extends Tacos1Viande {

    public Tacos1ViandePouletKetchup(String description, double prix) {
        super(description, prix);
        this.description=description+" 1 viande : poulet+ketchup";
    }
}
```

```java
public class Tacos1Viande extends Tacos {

    public Tacos1Viande(String description, double prix) {
        this.description=null;
        this.prix=prix+5.0;
    }
}
```

```java
public class Tacos2Viandes extends Tacos {

    public Tacos2Viandes(String description, double prix) {
        this.description=null;
        this.prix=prix+7.0;
    }
}
```

```java
public class Tacos1ViandePouletMayonnaise extends Tacos1Viande {

    public Tacos1ViandePouletMayonnaise(String description, double prix) {
        super(description, prix);
        this.description=description+" 1 viande : poulet+mayonnaise";
    }
}
```

```java
public class Tacos2ViandesBoeufEtPouletKetchup extends Tacos2Viandes {

    public Tacos2ViandesBoeufEtPouletKetchup(String description, double prix) {
        super(description, prix);
        this.description=description+" 2 viandes : boeuf+poulet+ketchup";
    }
}
```

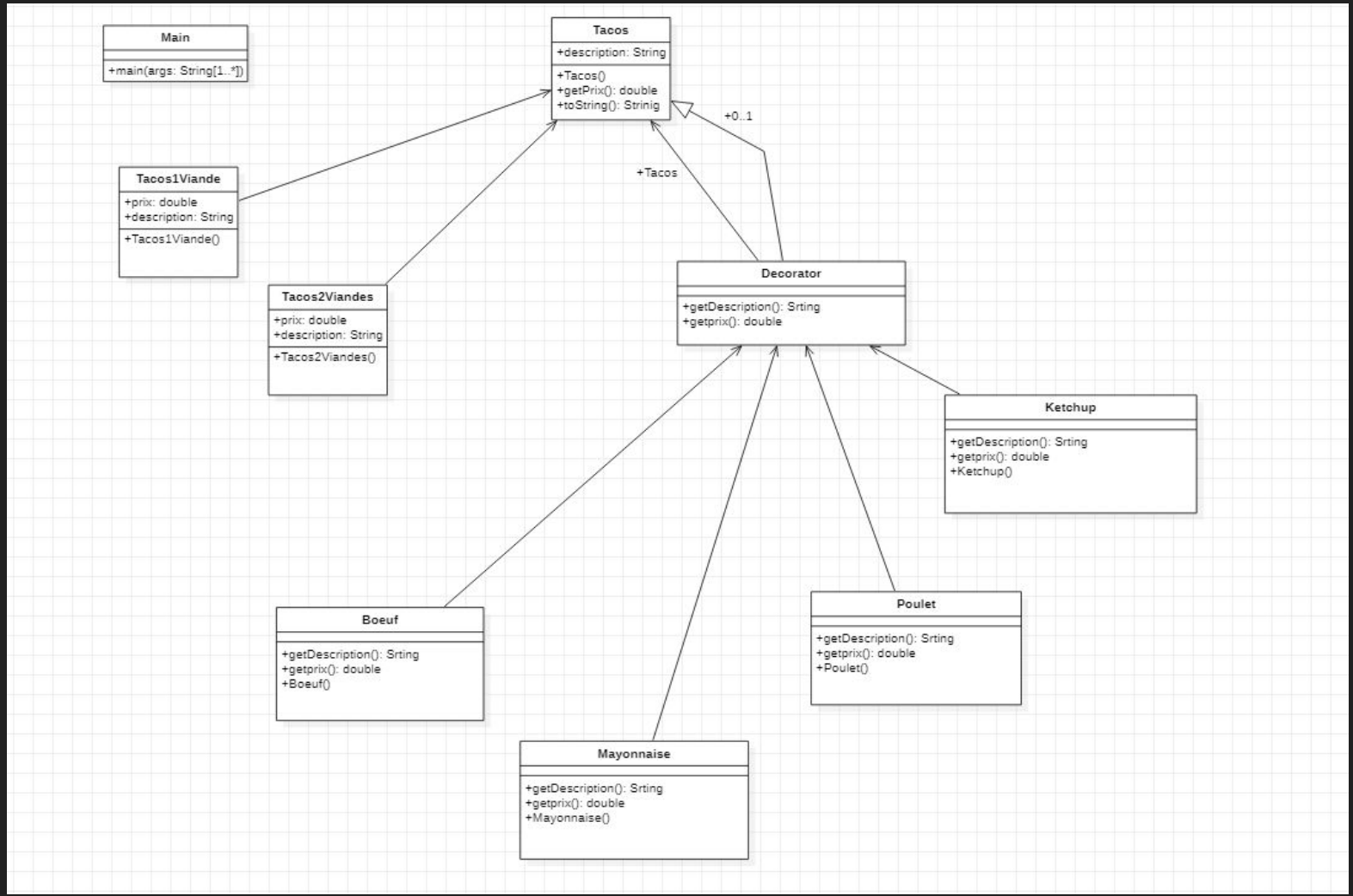
```java
public class Tacos2ViandesBoeufEtPouletKetchup extends Tacos2Viandes {

    public Tacos2ViandesBoeufEtPouletKetchup(String description, double prix) {
        super(description, prix);
        this.description=description+" 2 viandes : boeuf+poulet+ketchup";
    }

}
```

Les problèmes de l’héritage :

Duplication de code

Modifications de code compliquées et fastidieuses

Modèles très complexes


Main
+main(args: String[1..*])
Tacos
+description: String
+Tacos()
+getPrix(): double
+toString(): Strinig
+0..1
+Tacos
Tacos1Viande
+prix: double
+description: String
+Tacos1Viande()
Tacos2Viandes
+prix: double
+description: String
+Tacos2Viandes()
Decorator
+getDescription(): Srting
+getprix(): double
Ketchup
+getDescription(): Srting
+getprix(): double
+Ketchup()
Boeuf
+getDescription(): Srting
+getprix(): double
+Boeuf()
Poulet
+getDescription(): Srting
+getprix(): double
+Poulet()
Mayonnaise
+getDescription(): Srting
+getprix(): double
+Mayonnaise()

```java
public class Main {

    public static void main(String[] args) {

        Tacos tacos =new Tacos1Viande("",0.0);
        tacos=new Boeuf(tacos);
        tacos=new Ketchup(tacos);

        Tacos tacos2=new Tacos2Viandes("",0.0);
        tacos2= new Boeuf(tacos2);
        tacos2=new Poulet(tacos2);
        tacos2=new Mayonnaise(tacos2);

        System.out.println("Le "+tacos.getDescription()+ " coûte "+tacos.getPrix());
        System.out.println("Le "+tacos2.getDescription()+ " coûte "+tacos2.getPrix());
    }
}
```

```java
public abstract class Tacos {
        protected String description;
        protected double prix;

        public String getDescription() {
            return this.description;
        }

        public Double getPrix() {
            return this.prix;
        }
}
```

```java
public class Tacos2Viandes extends Tacos {

    public Tacos2Viandes(String description, double prix) {
        this.description=description+"Tacos 2 viandes";
        this.prix=prix+7.0;
    }
}
```

```java
public class Tacos1Viande extends Tacos {

    public Tacos1Viande(String description, double prix) {
        this.description=description+"Tacos 1 viande";
        this.prix=prix+5.0;
    }
}
```

```java
public abstract class Decorateur extends Tacos {
    @Override
    public abstract String getDescription();
    @Override
    public abstract Double getPrix();
}
```

```java
public class Boeuf extends Decorateur {

    Tacos tacos;
    public Boeuf(Tacos tacos) {
        this.tacos=tacos;
    }

    @Override
    public String getDescription() {
        return tacos.getDescription()+" boeuf";
    }

    @Override
    public Double getPrix() {
        return this.prix;
    }
}
```

```java
public class Poulet extends Decorateur {

    Tacos tacos;
    public Poulet(Tacos tacos) {
        this.tacos=tacos;
    }

    @Override
    public String getDescription() {
        return tacos.getDescription()+" poulet";
    }

    @Override
    public Double getPrix() {
        return this.prix;
    }
}
```

```java
public class Ketchup extends Decorateur {

    Tacos tacos;
    public Ketchup(Tacos tacos) {
        this.tacos=tacos;
    }

    @Override
    public String getDescription() {
        return tacos.getDescription()+" ketchup";
    }

    @Override
    public Double getPrix() {
        return this.prix;
    }
}
```

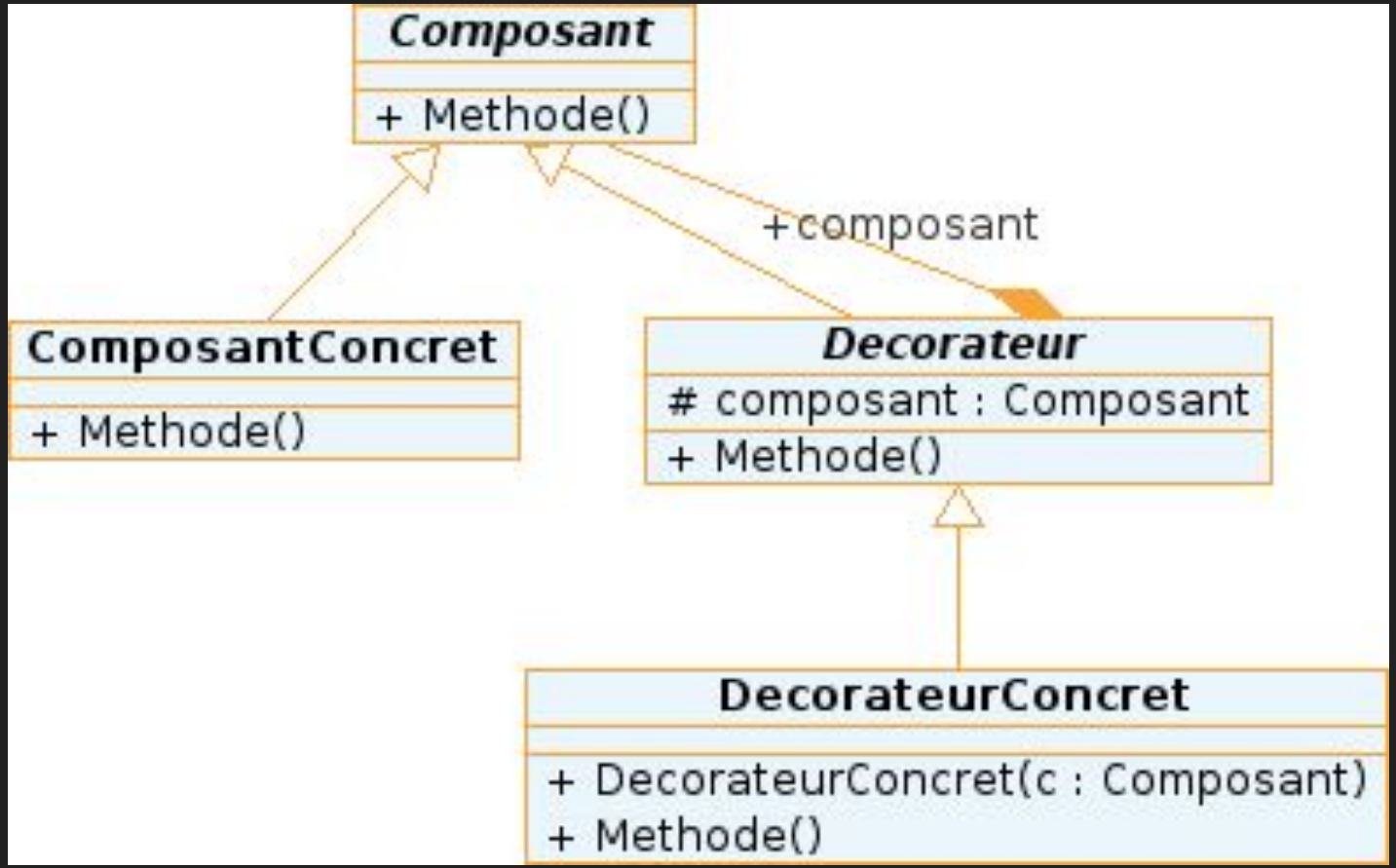
```java
public class Mayonnaise extends Decorateur {

    Tacos tacos;
    public Mayonnaise(Tacos tacos) {
        this.tacos=tacos;
    }

    @Override
    public String getDescription() {
        return tacos.getDescription()+" mayonnaise";
    }

    @Override
    public Double getPrix() {
        return this.prix;
    }
}
```

Avantages :

Pas de duplication de code

Redefinition simple

Modèle simple et compact


Composant
+ Methode()
+composant
ComposantConcret
+ Methode()
Decorateur
composant : Composant
+ Methode()
DecorateurConcret
+ DecorateurConcret(c : Composant)
+ Methode()

Decorateur :

Patterns de Structuration

Implémentation flexible

Réduction des dépendances

S.O.L.I.D

Srp (Responsibilité unique)

Ocp (Ouvert/fermé)

Lsp (Substitution de Liskov)

Isp (Ségrégation des interfaces)

Dip (Inversion des dependances)

Inconvénients

De très nombreux objets à instancier

Difficulté pour un supprimer un décorateur en particulier

Autres Patterns

Adapter

Composite

Strategy

Sitographie

Site :

- Design Patterns : Elements of Reusable Object-Oriented Software
- Decorator
- https://design-patterns.fr/introduction-aux-design-patterns#:~:text=L'origine%20des%20Design%20Patterns,de%20l'architecte%20Christopher%20Alexander.&text=Pour%20cela%20Alexander%20%C3%A9tabli%20un,fa%C3%A7on%20de%20concevoir%20une%20charpente).
- Christopher Alexander — Wikipédia
- enseignement-iut-m3105-conception-avancee/SOLID.pdf at master · iblasquez/enseignement-iut-m3105-conception-avancee

QCM

De quelle famille de Pattern fait partie le Décorateur ?

A. Création

B. Comportement

C. Structuration

Combien de fois peut-on “décorer” un objet avec les sous classes du décorateur ?

A. Une seule fois

B. Autant de fois qu’on veut

C. On ne peut pas “décorer” un objet avec les sous classes du décorateur

Pourquoi utilise-t-on un pattern décorateur ?

A. Pour rendre son code plus joli

B. Pour ajouter des fonctions dynamiquement

C. Pour faire de l’héritage multiple

Donner un inconvénient du Décorateur.

A. Il est difficile de supprimer un décorateur en particulier

B. Il est difficile de coder un décorateur

C. Il rend la lecture du code plus complexe

Quel est le titre du livre qui a popularisé les Design Patterns ?

A. Design Patterns : Reusable Solution for Software Design

B. Design Patterns : How to Build a Software

C. Design Patterns : Elements of Reusable Object-Oriented Software