Les modèles structuraux

- Ces modèles englobent tous les patterns dont le rôle est de s'occuper de la composition de classes et d'objets. Nous étudierons les patterns :

- **Adaptateur**
- **Composite**
- **Facade**
- **Decorateur**
- **Procuration**

Le pattern structurel Adaptateur

- **La motivation**

Soit un éditeur de dessin qui gère des textes , des lignes et des polygones. Chaque objet pouvoir s'afficher, s'éditer et se déplacer. L'éditeur voit ces objets à travers l'interface Forme

```
Interface Forme
    afficher() // pour afficher
    editer() // pour éditer
    // pour récupérer un objet qui gère les déplacements
    creerManipulateur()
```

Le pattern structurel Adaptateur

- On dispose d'une classe VueTexte en open source mais dont l'interface d'utilisation est **editerTexte** pour l'édition et **afficherTexte** pour l'affichage.

- Il faut adapter cette interface pour qu'elle soit conforme à celle utilisée par l'éditeur de dessin.

- On doit donc créer une classe d'adaptation **FormeTexte** qui implémente **Forme** et utilise la classe **VueTexte**.

Le pattern structurel Adaptateur

- **L'intention**
  - Ce pattern doit convertir l'interface d'une classe en une interface conforme à l'attente de l'utilisateur.
  - L'adaptateur permet donc à des classes n'ayant pas des interfaces compatibles de travailler ensemble.

Le pattern structurel Adaptateur

- **Les constituants sont** :
  - Une interface **But** qui définit l'interface qu'utilise le client.
  - Une classe **Client** qui collabore avec les objets en se conformant à l'interface But.
  - Une classe **Adapte** qui réalise ce que doit utiliser le client à condition d'adapter son interface.
  - Une classe **Adaptateur** adapte l'interface de l'adapté pour se conformer à l'interface But.

Le pattern structurel Adaptateur

- Les **correspondances** avec l'exemple sont :
  - L'interface **But** est l'interface Forme
  - La classe **Client** est l'éditeur de dessin
  - La classe **Adapte** est la classe VueTexte
  - La classe **Adapateur** est la classe FormeTexte

Le pattern structurel Composite

- **La motivation**

  Soit un éditeur de dessin pour créer des objets graphiques.

  Ces objets graphiques sont :

  - soit des objets primitifs tels que lignes, Polygones et textes
  - soit des objets définissant un agrégat d'objets graphiques.

Le pattern structurel Composite

La motivation

On désire traiter de la même façon les objets primitifs et les objets agrégats d'objets graphiques en constatant qu'ils ont en commun d'être des graphiques

Le pattern structurel Composite

- **La motivation**
  - Image, la classe définissant un agrégat d'objets graphiques.
  - Graphique, une classe abstraite pour manipuler des objets graphiques qui peuvent être :
    - des objets de la classe Image
    - des objets primitifs dont les classes sont Ligne, Polygone et Texte

Le pattern structurel Composite

- On en déduit qu'un objet Image est une arborescence dont :
  - les noeuds internes sont des agrégats d'objets graphiques
  - les noeuds externes (ou feuilles) sont des formes primitives (lignes, polygones, texte)

Le pattern structurel Composite

- D'un point de vue général, avec un objet nœud, on doit pouvoir :
  - Le dessiner
  - Ajouter un noeud enfant (un objet graphique)
  - Supprimer un noeud enfant (un objet graphique)
  - Récupérer l'un de ses nœuds enfants.

Le pattern structurel Composite

- Tous les nœuds héritent d'une classe abstraite Graphique qui :
  - déclare l'opération **dessine()**
  - implémente les trois autres opérations avec le code quasiment vide pour les objets primitifs

- Chaque objet primitif implémente son opération dessine().

- Chaque objet de la classe Image implémente l'opération dessine() avec le code suivant :

Pour tout objet graphique g de l'image

f.dessine()

Le pattern structurel Composite

- **L'intention de ce pattern** est d'organiser des objets en structure arborescente.

- Un objet Composite permet de traiter les objets individuels (feuilles de l'arbre) de la même façon que les ensembles organisés (noeuds de l'arbre).

Le pattern structurel Composite

- **Les constituants** sont :
  - Une classe abstraite Composant (Graphique) qui définit l'interface qu'utilise le client.
  - Des classes Feuille (texte, ligne, polygone) qui dérive de la classe Composant et définit le comportement des objets primitifs.

Le pattern structurel Composite

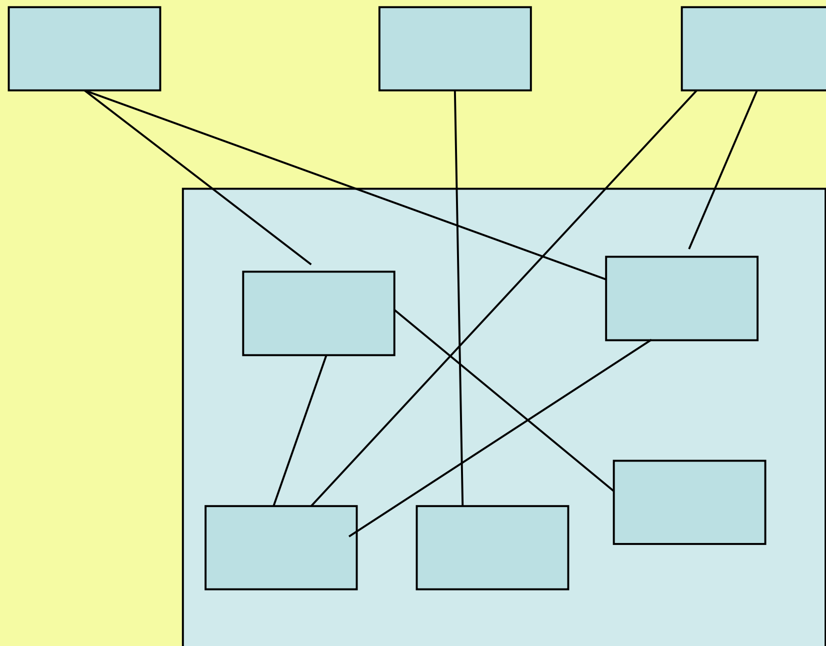
- **Les constituants**
  - Une classe Composite (Image) qui dérive de la classe Composant
  - Elle définit les opérations liées aux enfants (supprimer,ajouter, récupérer)
  - stocke des composants enfants
  - définit les opérations liées aux composants d'enfants (dessine)

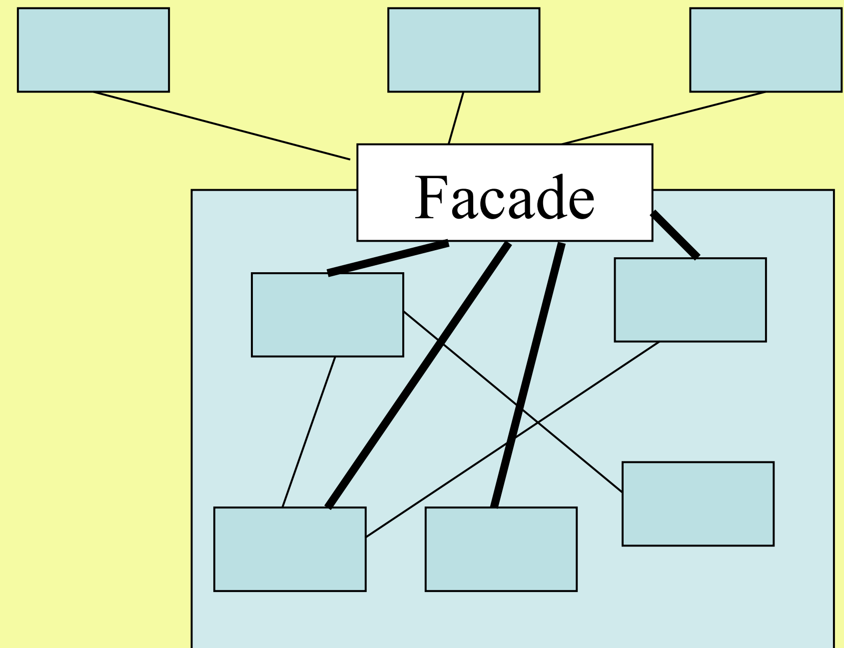
Le pattern structurel Composite

- Les constituants
  - Une classe **Client** (éditeur de dessin) dont les objets manipulent des objets de composition en utilisant des références de la classe abstraite Composant.

Le pattern structurel Façade

- **La motivation**
  - Structurer un système en sous systèmes pour réduire la complexité de l'ensemble.





Le pattern structurel Façade

- **Intention**
  - Ce pattern fournit une interface unifiée à l’ensemble des interfaces d’un sous système
  - La façade fournit une interface de plus haut niveau.

Le pattern structurel Façade

- Conséquences de l'Intention
  - Le sous système est plus facile à utiliser.
  - L'intérieur du sous système peut être mis à jour sans impacter le ou les clients du sous système

Le pattern structurel Façade

- **Les constituants**
  - La façade qui
    - connaît les classes du sous système qui savent traiter les requêtes des clients
    - délègue le traitement des requêtes des clients à ces classes

Le pattern structurel Façade

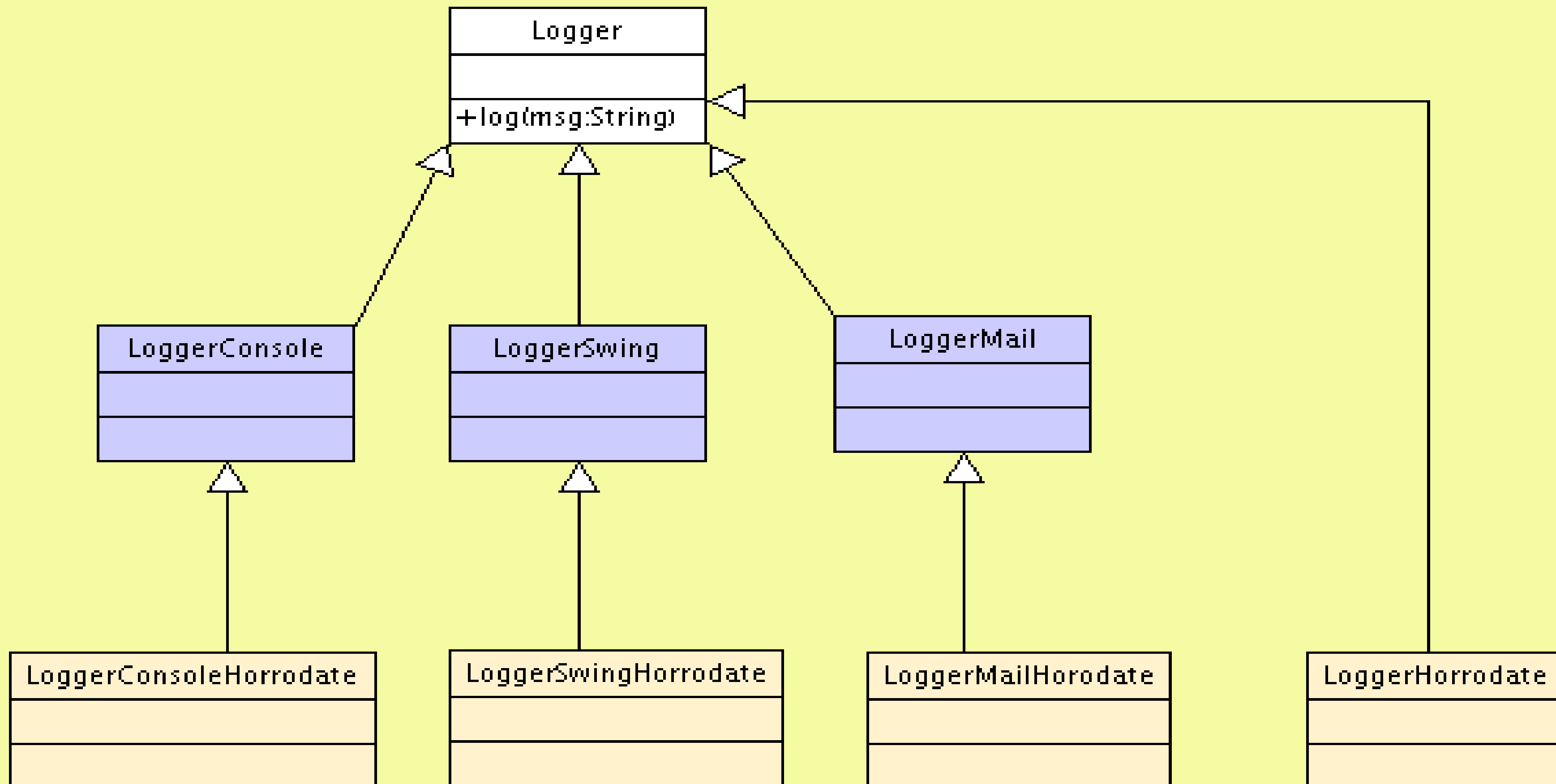
- **Les constituants**
  - Les classes du sous système
    - qui implémentent les fonctionnalités du sous système
    - qui gèrent les travaux assignés par l'objet Facade
    - Elles ne connaissent pas la Facade

Le pattern structurel Decorateur

- **La motivation**
  - On dispose d'une classe Logger qui permet de journaliser des messages.
  - On veut ajouter à chaque message, l'heure et la date d'émission
  - On veut aussi ajouter l'affichage des messages
    - dans une console texte
    - dans une fenêtre graphique Swing
    - dans un fichier pour envoyer un mail

Le pattern structurel Decorateur

- Une solution par héritage qui manque de souplesse
- 7 nouvelles classes sont nécessaires !!!!

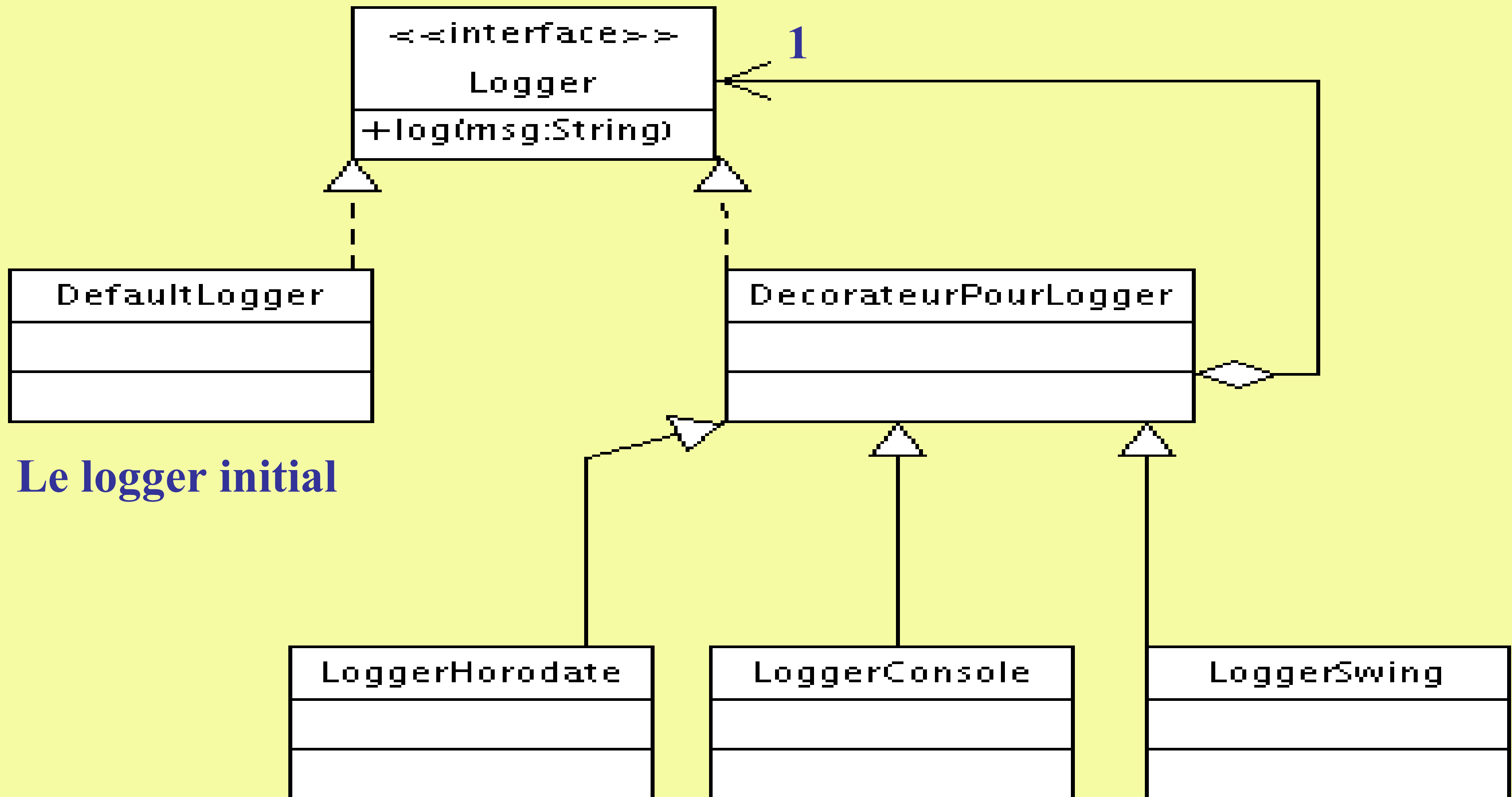


Le pattern structurel Decorateur

- **L'intention**
  - Ce pattern doit permettre d'attacher dynamiquement des responsabilités supplémentaires à un objet
  - Il permet une alternative plus souple que la dérivation pour étendre les fonctionnalités

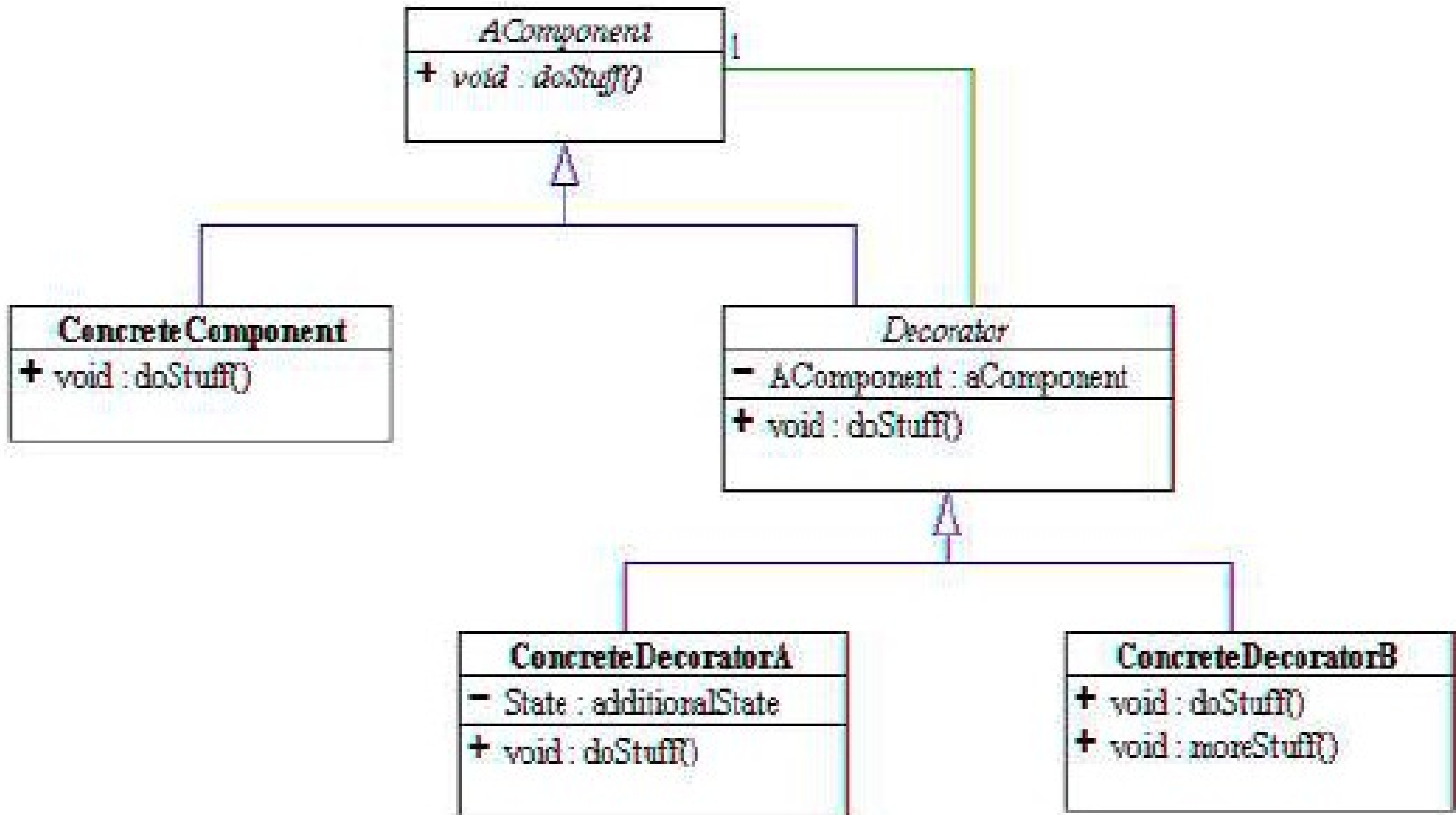
Le pattern structurel Decorateur

- Une solution par composition



Le pattern structurel Decorateur

- Les constituants et leur structure



Le pattern Decorateur en Java

```
interface AComponent {
    public void operation();
}

class ConcreteComponent implements AComponent {
    public void operation() {
        System.out.println("racine");
    }
}
```

Le pattern Decorateur en Java

```
abstract class Decorateur implements AComponent {
    private AComponent component;

    public Decorateur(AComponent component) {
        this.component = component;
    }

    public AComponent getComponent() {
        return component;
    }
}
```

Le pattern Decorateur en Java

```
class ConcreteDecorateurA extends Decorateur {
    public ConcreteDecorateurA(AComponent component) {
        super(component);
    }
    public void operation() {
        try {
            System.out.print("Decorateur A ");
            this.getComponent().operation();
        }
        catch(Exception e) {
            System.out.println(" pas de composant");
        }
    }
}
```

Le pattern Decorateur en Java

```
class TestDecorateur {
    public static void main(String [] args) {
        Decorateur dec =
        new ConcreteDecorateurB(
                (AComponent)new ConcreteDecorateurA((
                                AComponent)new
                        ConcreteComponent()));

        dec.operation();
    }
}
```

Le pattern structurel Procurateur

- **La motivation**
  - On désire surveiller l'utilisation d'un objet X d'une façon transparente pour l'objet Client qui s'en sert
  - On désire faire des statistiques d'utilisation d'un objet X d'une façon transparente pour l'objet Client qui s'en sert

Le pattern structurel Procurateur

- **La motivation**
  - On désire minimiser l’espace mémoire utilisé par un objet X d’une façon transparente pour l’objet Client qui se sert de X
  - On désire rendre transparent à un objet Client l’utilisation d’objets distants

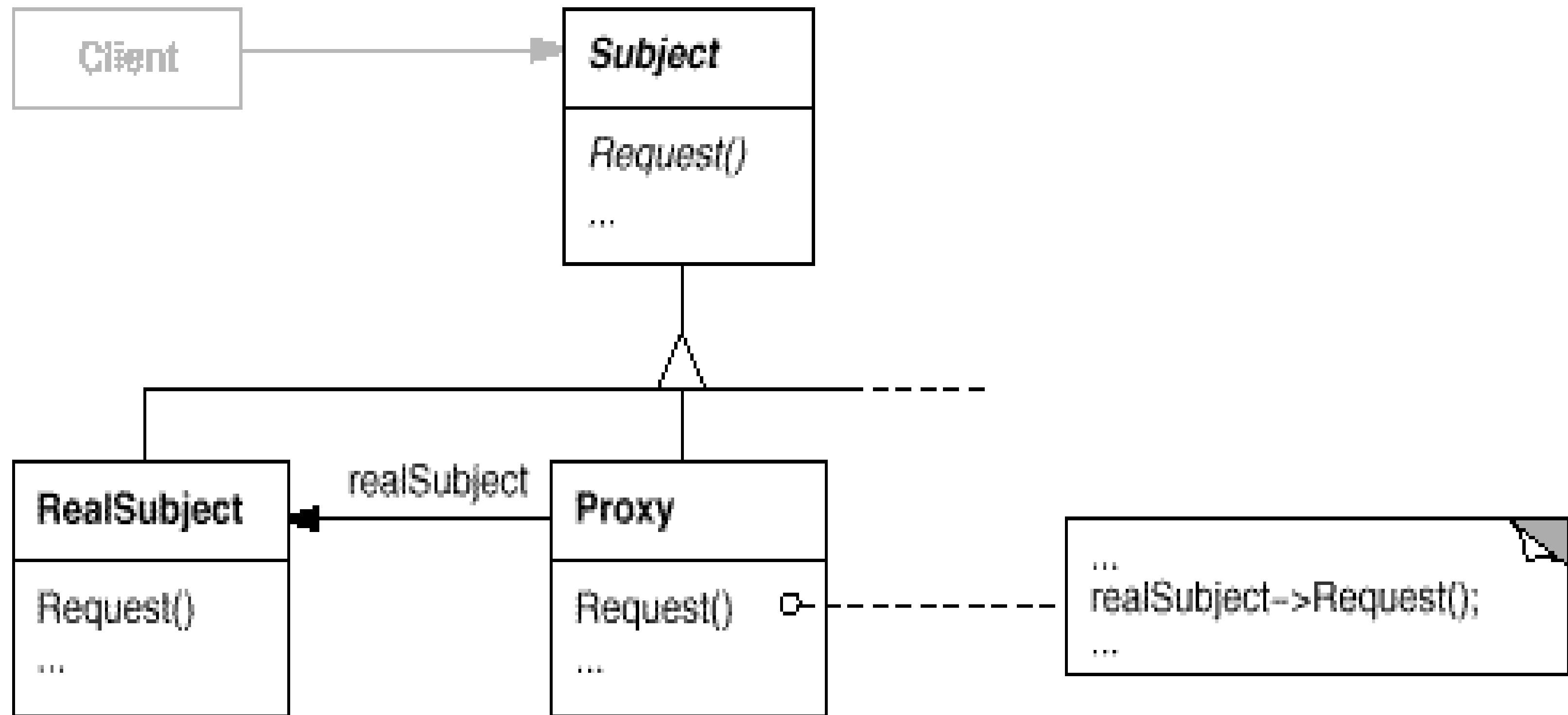
Le pattern structurel Procurateur

- **L'intention**

  Ce pattern fournit un objet intermédiaire entre un objet Client et un objet X pour contrôler l'accès à l'objet X.

Le pattern structurel Proxy

- Les constituants et leur structure



Le pattern structurel Proxy

- La communication entre les objets