Design Patterns
Part 4

Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

Supports de cours : http://fr.slideshare.net/mohamedyoussfi9

Chaîne vidéo : http://youtube.com/mohamedYoussfi

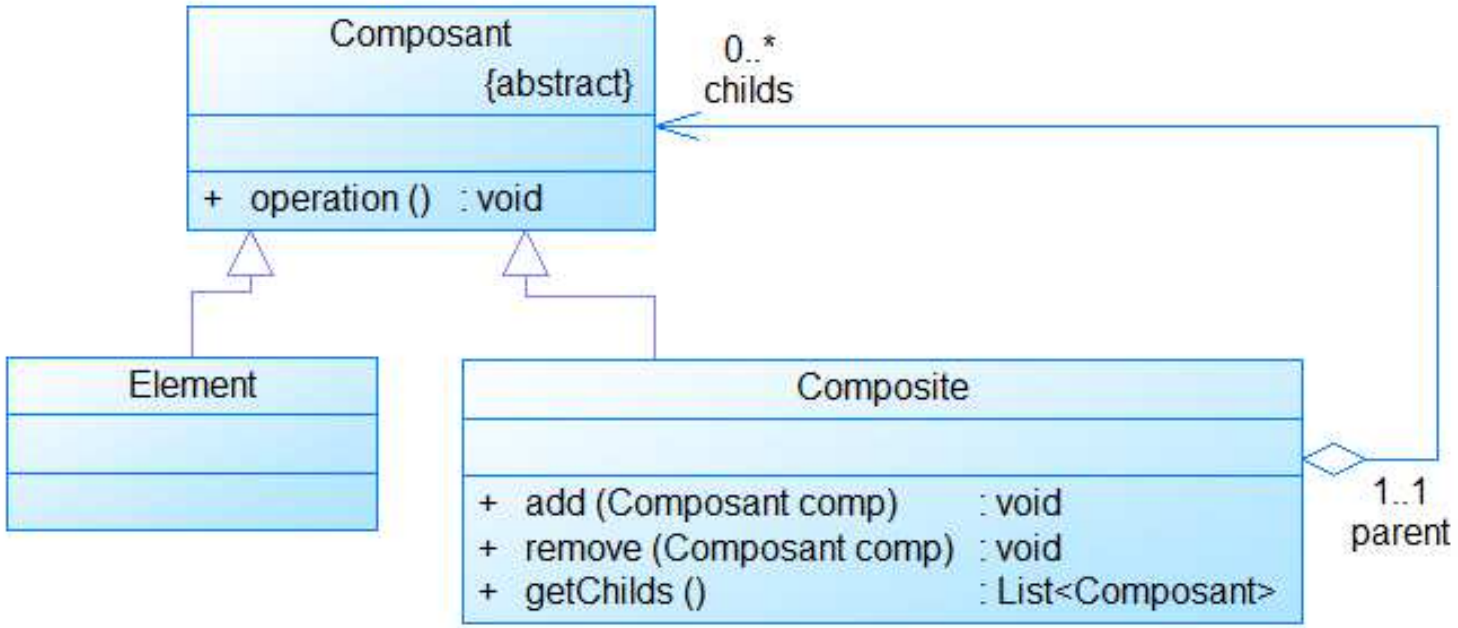
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Pattern Composite

Pattern Composite

- Catégorie :
  - Structure
- Objectif du pattern Décorateur
  - *Organiser les objets en structure arborescente afin de représenter une hiérarchie.*
  - *Permettre à la partie cliente de manipuler un objet unique et un objet composé de la même manière.*
- Résultat :
  - Le Design Pattern permet d'isoler l'appartenance à un agrégat.

Diagramme de classes



Raison d'utilisation

- Le système comporte une hiérarchie avec un nombre de niveaux non déterminé.
- Il est nécessaire de pouvoir considérer un groupe d'éléments comme un élément unique.
- Cela peut être le cas des éléments graphiques d'un logiciel de DAO. Plusieurs éléments graphiques peuvent être regroupés en un nouvel élément graphique.
- Chaque élément est un composant potentiel. En plus des éléments classiques, il y a un élément composite qui peut être composés de plusieurs composants.
- Comme l'élément composite est un composant potentiel, il peut être composé d'autres éléments composites.

Responsabilités

- **Composant :**
  - Définit l'interface d'un objet pouvant être un composant d'un autre objet de l'arborescence.
- **Element :**
  - implémente un objet de l'arborescence n'ayant pas d'objet le composant.
- **Composite :**
  - implémente un objet de l'arborescence ayant un ou des objets le composant.
- La partie client manipule les objets par l'interface **Composant.**

Implémentation

/* Composant.java */

```
public abstract class Composant {
   protected String nom;
   protected int level;
   public Composant(String nom) {
      this.nom = nom;
   }
   public abstract void operation();
}
```

/* Element.java */

```
public class Element extends Composant {
public Element(String nom) { super(nom); }
@Override
public void operation() {
   String tab=""; for(int i=0;i<level;i++) tab+="--";
   System.out.println(tab+"Opération sur l'élément ("+nom+")");
}
}
```

Implémentation

/* Composite.java */

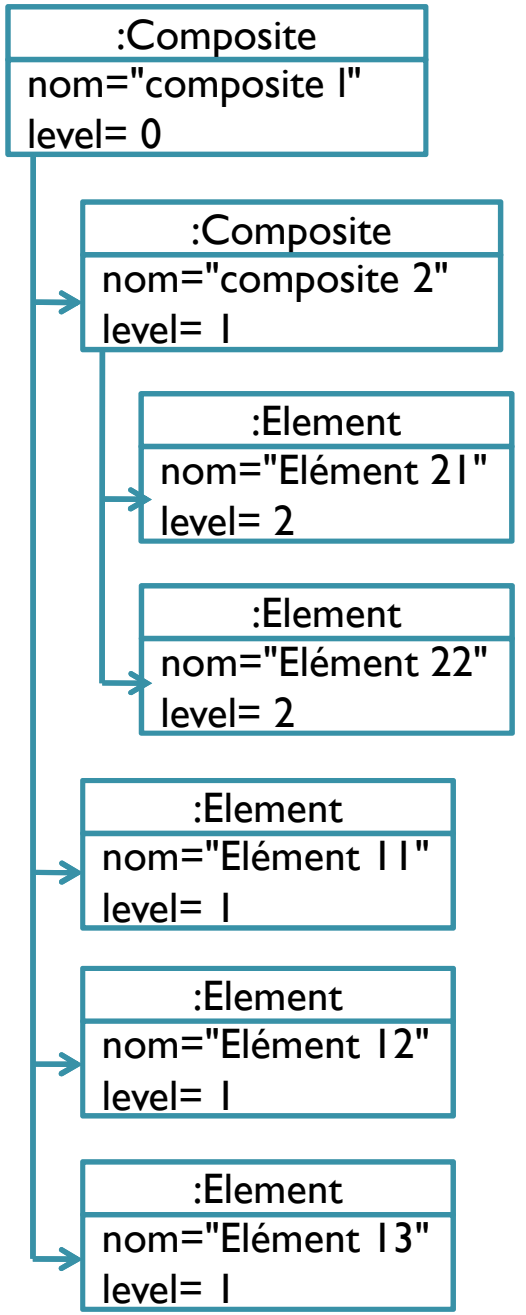
```
import java.util.ArrayList; import java.util.List;
public class Composite extends Composant {
private List<Composant> composants=new ArrayList<Composant>();
  public Composite(String nom) { super(nom); }
  @Override
  public void operation() {
     String tab=""; for(int i=0;i<level;i++) tab+="--";
     System.out.println(tab+"Opération sur un composite("+nom+")");
     for(Composant composant:composants)
        composant.operation();
}
public void add(Composant composant){
  composant.level=this.level+1; composants.add(composant);
}
public void remove(Composant composant){ composants.remove(composant); }
public List<Composant> getChilds(){ return composants; }
}
```

Implémentation

/* Application.java */

```
public class Application {
public static void main(String[] args) {
   Composite racine=new Composite("Composite 1");
   Composite composite2=new Composite("Composite 2");
   racine.add(composite2);
   racine.add(new Element("Elément 11"));
   racine.add(new Element("Elément 12"));
   racine.add(new Element("Elément 13"));
   composite2.add(new Element("Elément 21"));
   composite2.add(new Element("Elément 22"));
   racine.operation();
}}
```

Opération sur un composite(Composite 1)
-- Opération sur un composite(Composite 2)
---- Opération sur l'élément (Elément 21)
---- Opération sur l'élément (Elément 22)
-- Opération sur l'élément (Elément 11)
-- Opération sur l'élément (Elément 12)
-- Opération sur l'élément (Elément 13)



ssfi.net