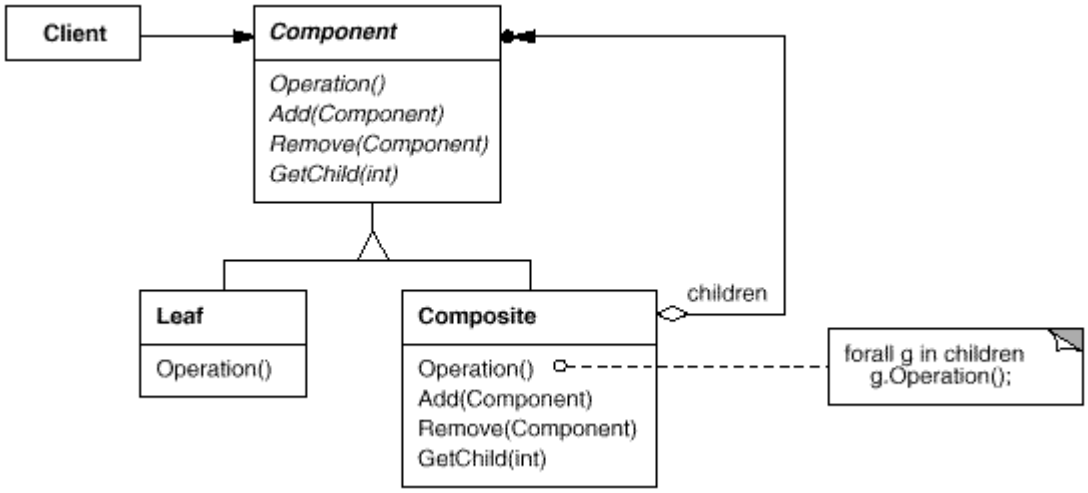
DESIGN PATTERN

TP n°1 – Instanciation du patron Composite

Nous rappelons la structure (statique) du pattern de conception Composite :



Et nous en donnons une implantation possible en Java :

```
import java.util.Collection;
import java.util.Iterator;

public abstract class Component {

	public void operation(){
		System.out.println("Opération par défaut");
	}

	// Ce sont les objets composites qui stockent les objets composants
	protected abstract Collection getChildren();

	public void add(Component c){
		Collection composite = this.getChildren();
		if (!composite.contains(c))
			composite.add(c);
	}

	public void remove(Component c){
		Collection composite = this.getChildren();
		if (composite.contains(c))
			composite.remove(c);
	}

	public Iterator iteratorOfChildren(){
		Collection composite = this.getChildren();
		return composite.iterator();
	}

}
```

```
import java.util.Collection;
import java.util.Iterator;

public class Leaf extends Component {

	// redéfinition de Opération spécifique à objet composant de base
```

```
        public void operation(){
                System.out.println("Opération sur objet composant de base");
        }

        // Un objet Feuille ne peut exécuter les opérations d'un objet Composite
        public void add(Component c){
                throw new UnsupportedOperationException("cet objet est une feuille !");
        }

        public void remove(Component c){
                throw new UnsupportedOperationException("cet objet est une feuille !");
        }

        public Iterator iteratorOfChildren(){
                throw new UnsupportedOperationException("cet objet est une feuille !");
        }

        // une feuille par définition ne contient pas d'autres objets composants
        protected Collection getChildren(){
                return null;
        }
}
```

```
import java.util.Collection;
import java.util.ArrayList;
import java.util.Iterator;

public class Composite extends Component {

        // La classe composite contient les références des objets composants.
        Collection children = new ArrayList();

        protected Collection getChildren(){
                return this.children;
        }

        //      redéfinition de Opération pour un objet Composite
         public void operation(){
                System.out.println("Action avant délégation");
                // for all g in children
                //            g.operation()
                Iterator composite = this.iteratorOfChildren();
                while(composite.hasNext())
                        ((Component)composite.next()).operation();
                System.out.println("Action après délégation");
         }
}
```

- En réutilisant par analogie le modèle de conception Composite, implanter en Java un évaluateur d'expression arithmétique.
    - Cet évaluateur ne traitera que des nombres entiers.
    - Les quatre opérations possibles sont :
        - addition : + (liste d'expressions arithmétiques) = $\sum$ expressions arithmétiques
        - multiplication : x (liste d'expressions arithmétiques) = $\prod$ expressions arithmétiques.
        - soustraction : - (liste d'expressions arithmétiques) = $expression_1 - expression_2 - \ldots expression_n$
        - division : / (liste d'expressions arithmétiques) = $expression_1 / expression_2 / \ldots expression_n$

- Ecrire un programme client permettant d'évaluer l'expression arithmétique suivante :

  $$2 + (3 * (5 - 2)) * 6 / 6 + 7$$

  En plaçant des traces (System.out.println(..)) dans les méthodes d'évaluation, le résultat à l'exécution devrait être :

```
valeur feuille : 2
valeur feuille : 3
valeur feuille : 5
valeur feuille : 2
valeur - : 3
valeur feuille : 6
valeur * : 54
valeur feuille : 6
valeur / : 9
valeur feuille : 7
valeur + : 18
2 + (3 * (5 -2)) * 6 /6 + 7 = 18
```