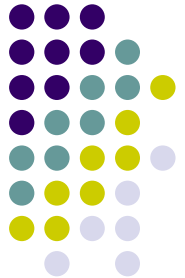
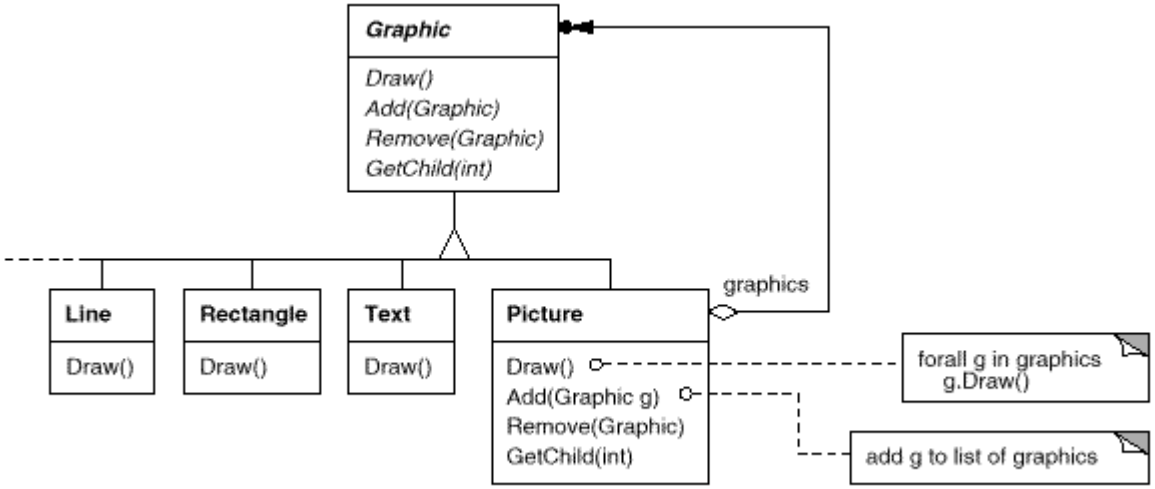
Un exemple complet : le patron Composite



- **Modèle Composite / Objet / Structurel**
- **Intention**
  - **Compose des objets en des structures arborescentes pour représenter des hiérarchies composant/composé. Le client traitera de manière uniforme les objets individuels (feuilles) et composites (nœuds internes)**

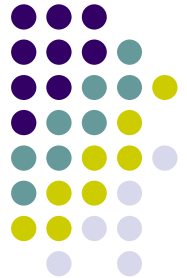
Un exemple complet : le patron Composite

- **Motivation**



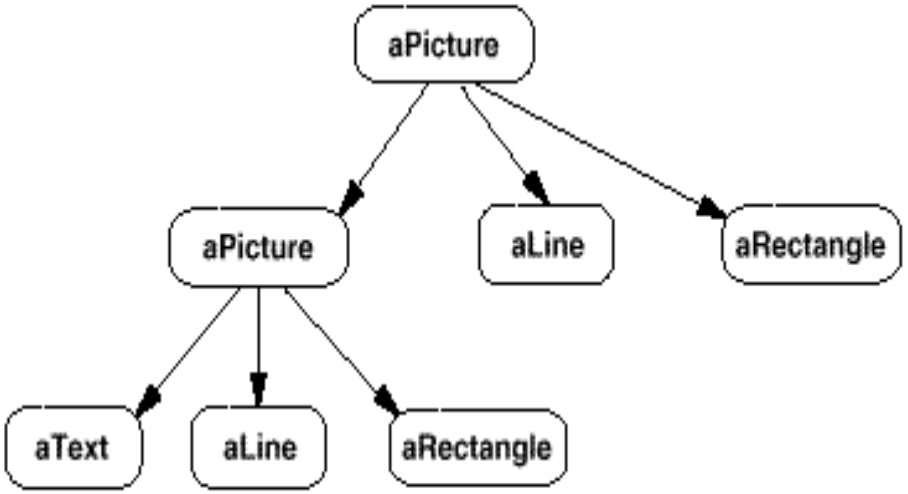
• Construction de diagrammes complexes (Picture) à partir d'objets graphiques simples.
• Le client gère uniquement des Objets graphiques pouvant se composer et se dessiner.

Un exemple complet : le patron Composite

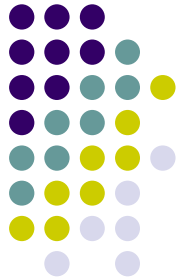


- **Motivation**

• Structure typique d'objets graphiques composés récursivement.



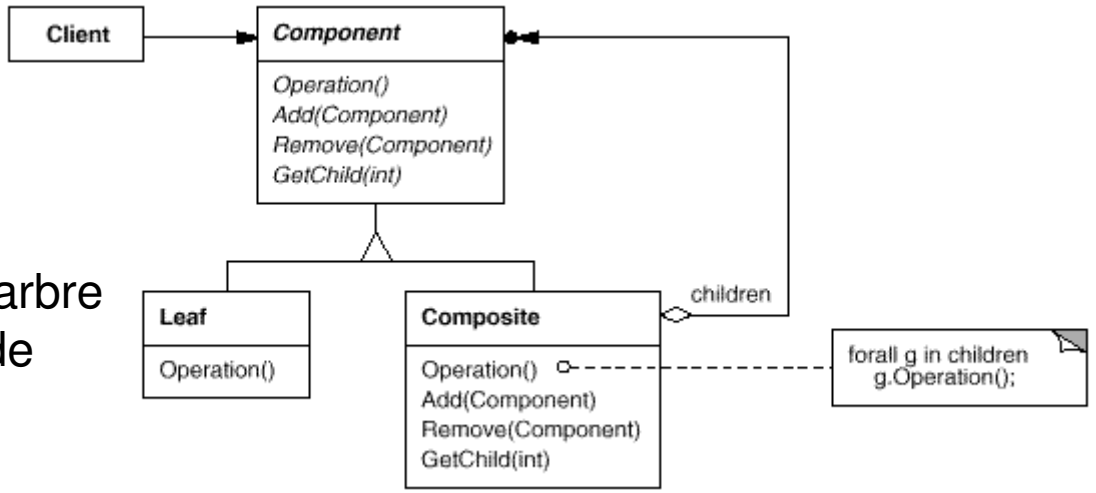
Un exemple complet : le patron Composite



- **Indication d'utilisation**
  - On veut manipuler des objets composés ou emboîtés récursivement le plus simplement possible.
  - Les programmes utilisateurs utilisent un protocole commun pour traiter ces objets.

Un exemple complet : le patron Composite

- Structure



Client (application) :
manipule les objets de l'arbre de composition à l'aide de l'interface Composant.

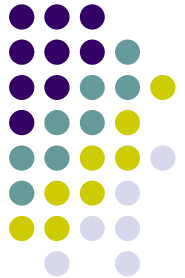
- Constituants

Composant (classe abstraite) :
Interface commune à tous les objets
Gestion de la relation de composition

Feuille (classe concrète) :
Implante l'interface commune pour des objets « simples ».

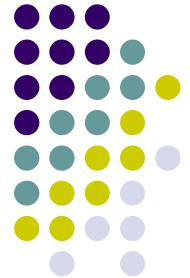
Composite (classe concrète) :
Stocke les composants enfants
Interface commune à tous les objets

Un exemple complet : le patron Composite



- Collaborations
  - Les clients utilisent l'interface Composant pour
    - Créer et assembler des arbres d'objets composites.
    - Gérer des traitements sur les objets de la structure.
  - Les Feuilles traitent les messages directement.
  - Les Composites délègue les messages à leurs enfants en ajoutant éventuellement un comportement.
- Conséquences
  - Définit des hiérarchies d'objets primitifs et composites.
  - Simplifie les programmes clients.
  - Permet l'ajout de nouveaux types de composants sans changer les programmes clents.
  - Bémol : excessivement général, rajouter à la main des contrôles sur le type des composants permises.

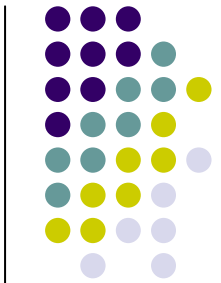
Un exemple complet : le patron Composite



- **Implémentation** (classe Component)

```
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();}
}
```

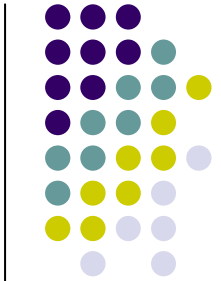
Un exemple complet : le patron Composite



- **Implémentation** (classe Feuille)

```
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;}
}
```

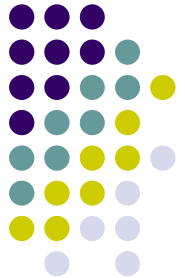
Un exemple complet : le patron Composite



- **Implémentation** (classe Composite)

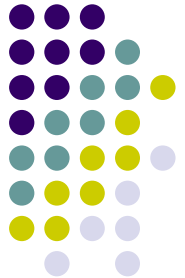
```
import java.util.Collection;
import java.util.ArrayList;
import java.util.Iterator;
public class Composite extends Component {

        // 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");
         }
}
```

Un exemple complet : le patron Composite



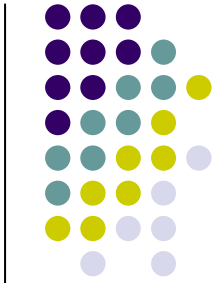
- **Utilisations remarquables**
  - **Le package java.awt,… A voir maintenant !**

Un exemple complet : le patron Composite



- **Modèles apparentés**
  - Décorateur
  - Poids mouche
  - Itérateur
  - Interpréteur
  - Visiteur

Un exemple complet : le patron Composite



- **Exemples de code**
  - 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) = ∑ expressions arithmétiques
      - multiplication : x (liste d'expressions arithmétiques) = ∏ expressions arithmétiques.
      - soustraction : - (liste d'expressions arithmétiques) = expression1 – expression2 - … expressionn
      - division : / (liste d'expressions arithmétiques) = expression1 / expression2 / … expressionn
  - Ecrire un programme client permettant d'évaluer l'expression arithmétique suivante : 2 + (3 * (5 - 2)) * 6 / 6 + 7