LE PATTERN COMPOSITE

Intention

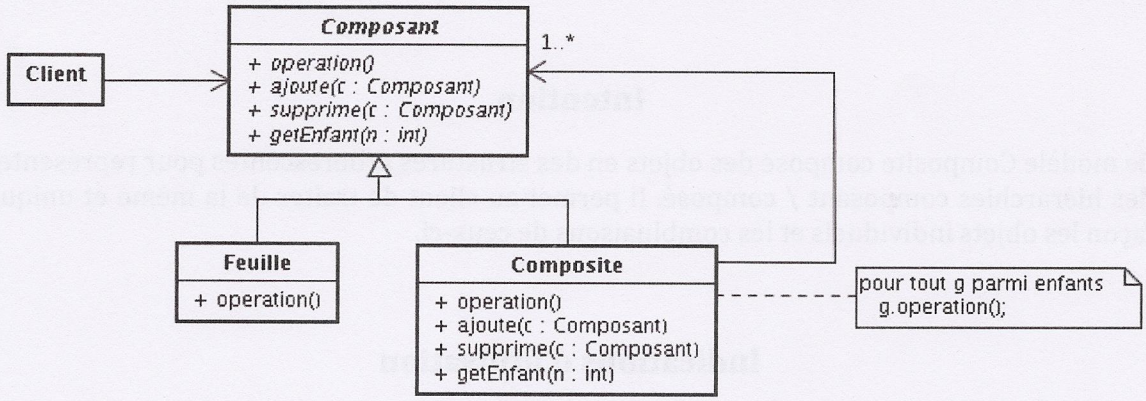
Le modèle Composite compose des objets en des strustures arborescentes pour représenter des hiérarchies composant / composé. Il permet au client de traiter de la même et unique façon les objets individuels et les combinaisons de ceux-ci.

Indications d'utilisation

On utilisera le modèle Composite lorsque :

- On souhaite représenter des hiérarchies de l'individu à l'ensemble.
- On souhaite que le client n'ait pas à se préoccuper de la différence entre combinaisons d'objets et objets individuels. Les clients pourront traiter de façon uniforme tous les objets de la structure composite.

Structure



Constituants

Composant

- Le Composant déclare l'interface des objets entrant dans la composition.
- Il implémente le comportement par défaut qui convient pour l'interface commune à toutes les classes.
- Il déclare une interface pour accéder à ses composants enfants et les gérer.
- Eventuellement, il définit une interface pour accéder à un parent du composant dans une structure récursive, et l'implémente si besoin est.

Feuille

- La feuille représente des objets feuille dans la composition. Une feuille n'a pas d'enfants.
- Elle définit le comportement d'objets primitives dans la composition.

Composite

- Le Composite définit le comportement des composants dotés d'enfants.
- Il stocke les composants enfants.
- Il implémente les opérations liées aux enfants dans l'interface Composant.

Client

- Le Client manipule les objets de la composition à l'aide de l'interface Composant.

Collaborations

Les clients utilisent l'interface de la classe Composant pour manipuler les objets de la structure composite. Si l'objet manipulé est une feuille, la requête est traitée directement. Si c'est un Composite, il transfère généralement cette requête à ses composants, en effectuant éventuellement des opérations supplémentaires avant et /ou après ce transfert.

Location de voitures

(Les classes/interfaces de cet exercice seront à ranger dans un paquetage agence.)

Une agence de location de voitures offre à ses clients la possibilité de choisir la voiture louée en fonction de différents critères.
Les voitures sont définies par une marque, un nom de modèle, une année de production et un prix de location à la journée. Pour simplifier les deux premiers paramètres seront des objets de la classe `String` et les deux derniers seront des `int`. Deux voitures sont considérées égales si tous leurs attributs sont égaux.

Q 1 . Donner le code de la classe `Voiture`

```
package agence;
public class Voiture {

      private final String saMarque; //sa marque
      private final String sonModele; // son modele
      private final int sonAnnee; // son année de production
      private final int sonPrix; // son prix de location / jour

   /** Constructeur initialisant comlètement cette voiture.
            * @param ma marque
            * @param mo modele
            * @param a annee de production
            * @param p prix de location / jour
            */
      public Voiture(String ma, String mo, int a, int p) {
            this.saMarque = ma;
            this.sonModele = mo;
            this.sonAnnee = a;
            this.sonPrix = p;
      }
      public String marque() {
            return this.saMarque;
      }
      public String modele() {
            return this.sonModele;
      }
      public int annee() {
            return this.sonAnnee;
      }
      public int prixLocation() {
            return this.sonPrix;
      }
      public boolean equals(Object o) {
          try {
            Voiture v = (Voiture) o;
            return (v.marque().equals(this.marque()) && v.modele().equals(this.modele())
                     && v.annee() == this.annee()
                     && v.prixLocation() == this.prixLocation());
            }
            catch (ClassCastException e) {
                  return false;
            }
      }
}
```

Il est possible de sélectionner dans la liste des voitures à louer toutes les voitures satisfaisant un critère donné.
On définit l'interface `Critere` ainsi :

```
public interface Critere {
   /** @return true si et seulement si l'objet o est conforme au
     * critère (on dit que o satisfait le critère)
   */
   public boolean correspond(Object o);
}
```

Q 2 . Donnez le code d'une classe `CritereMarque` qui est un critère satisfait par toutes les voitures d'une marque donnée, precisée à la construction du critère.

```
package agence;
public class CritereMarque implements Critere {
      // marque des voitures correspondant au critère
      private String marque;

      /** @param m marque */
      public CritereMarque(String m) {
            this.marque = m;
      }
      public boolean correspond(Object o) {
            if (!(o instanceof Voiture)) {
                  return false;
            }
            Voiture v = (Voiture) o;
            return v.marque().equals(this.marque);
      }
}
```

Q 3 . Donnez le code d'une classe `CriterePrix` qui est un critère satisfait par toutes les voitures dont le prix est inférieur à un prix fixé a la construction du critère.

```
package agence;

public class CriterePrix implements Critere {
      private int prix;

      /** @param p prix */
      public CriterePrix(int p) {
            this.prix = p;
      }
      public boolean correspond(Object o) {
            if (! (o instanceof Voiture)) {
                  return false;
            }
            return ((Voiture) o).prixLocation() <= this.prix;
      }
}
```

Q 4 . On suppose une classe `Agence` définie (au minimum) ainsi :

agence::Agence
- voitures : List ...
+ Agence(...) + selectionne(c : Critère) : List ..

Donnez le code de la méthode `selectionne` qui sélectionne parmi toutes les voitures de l'agence (contenues dans l'attribut `voitures`) celles qui satisfont le critère donné.

```
      public List selectionne(Critere c) {

            List result = new ArrayList();
            Iterator it = this.voitures.iterator();
            while (it.hasNext()) {
```

```
                Voiture v = (Voiture) it.next();
                if (c.correspond(v))
                    result.add(v);
            }
            return result;
        }
```

Q 5 . En supposant que la réference agence est de type `Agence` et a été initialisée, donnez la ou les lignes de code permettant de sélectionner toutes les voitures de cette agence dont le prix est inférieur à 100.

```
    public static void main(String[] args) {
        Agence agence = new Agence();
        agence.selectionne(new CriterePrix(100));
.....
```

Q 6 . On peut naturellement souhaiter faire des intersections c'est-a-dire appliquer le « et » logique, de critères (on pourrait de même souhaiter des unions de critères). On obtient alors un nouveau critère satisfait si tous les critères qui le compose sont satisfaits.

Définissez une classe `InterCritere` qui permet de définir des critères par intersection d'un nombre quelconque de critères. On demande à pouvoir ajouter un nouveau critère à l'intersection mais pas de pouvoir en retrancher.

Q 7 . Donnez la ou les lignes de code permettant de créer un critère intersection d'un critère pour la marque "Timoléon" et d'un critère pour un prix inferieur a 100.