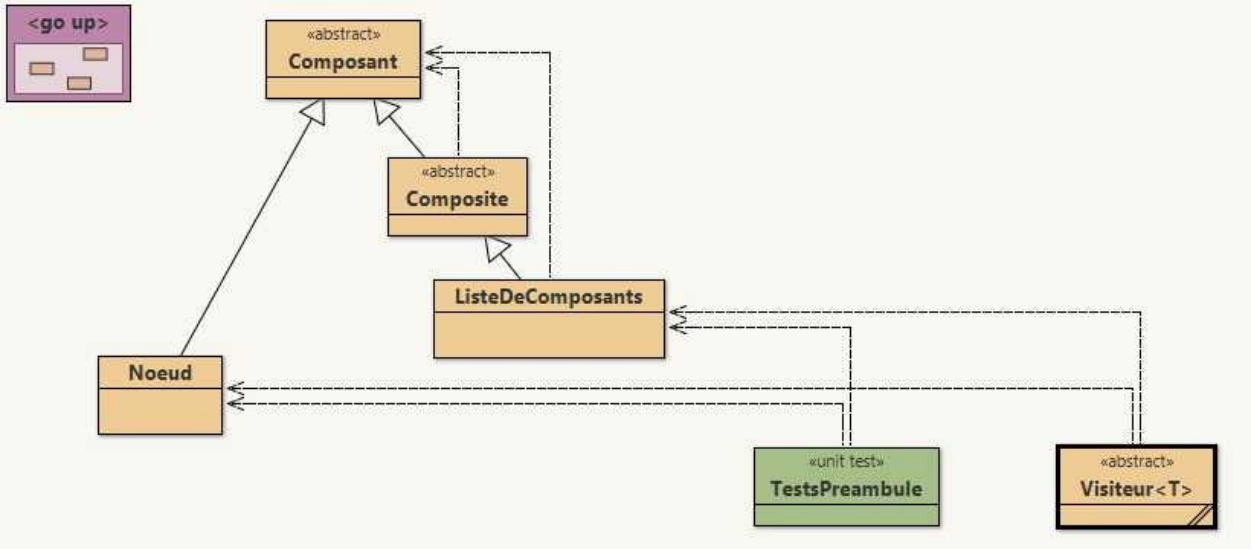
Exercices dirigés NFP121

Sommaire

Préambule : *Le patron composite*

Soit le patron Composite inspiré du cours (diapositives 16 à 21 du cours)



Préambule-1) Les patrons composite et interpréteur

Complétez toutes les classes afin que la classe de tests unitaires **TestsPreambule** ci-dessous puisse s'exécuter sans erreur (nb. pas de visiteurs demandés à ce niveau de préambule)

```
package preambule;

public class TestsPreambule extends junit.framework.TestCase{

   public void testComposite NombreDeNoeuds(){
      Noeud n1 = new Noeud("n1");
      Noeud n2 = new Noeud("n2");
      Noeud n3 = new Noeud("n3");
      assertEquals(1, n1.nombreDeNoeuds());
      ListeDeComposants l1 = new ListeDeComposants("l1");
      l1.ajouter(n1).ajouter(n2).ajouter(n3);
      assertEquals(3, l1.nombreDeNoeuds());

      ListeDeComposants l11 = new ListeDeComposants("l11");
      Noeud n11 = new Noeud("n11");
      l11.ajouter(n11);
      l1.ajouter(l11);
      assertEquals(4, l1.nombreDeNoeuds());
   }

   public void testComposite Interpreter(){
      Noeud n1 = new Noeud("n1");
      Noeud n2 = new Noeud("n2");
      Noeud n3 = new Noeud("n3");
      ListeDeComposants l1 = new ListeDeComposants("l1");
      l1.ajouter(n1).ajouter(n2).ajouter(n3);
      assertEquals("Liste:l1[Noeud:<n1>Noeud:<n2>Noeud:<n3>|",l1.interpreter());

      ListeDeComposants l11 = new ListeDeComposants("l11");
```

```
    Noeud n11 = new Noeud("n11");
    l11.ajouter(n11);
    l1.ajouter(l11);
    assertEquals("Liste:l1[Noeud:<n1>Noeud:<n2>Noeud:<n3>Liste:l11[Noeud:<n11>]]",l1.interpreter());
  }


  public void testComposite GetParent(){
    Noeud n1 = new Noeud("n1");
    Noeud n2 = new Noeud("n2");
    Noeud n3 = new Noeud("n3");
    assertEquals(null, n1.getParent());
    ListeDeComposants l1 = new ListeDeComposants("l1");
    l1.ajouter(n1).ajouter(n2).ajouter(n3);
    assertEquals(l1, n1.getParent());
    assertEquals(l1, n2.getParent());
    assertEquals(l1, n3.getParent());

    ListeDeComposants l11 = new ListeDeComposants("l11");
    Noeud n11 = new Noeud("n11");
    l11.ajouter(n11);
    l1.ajouter(l11);
    assertEquals(l1, l11.getParent());
    assertEquals(l11, n11.getParent());
    System.out.println(n11.getParent().getParent().interpreter());
    assertEquals(l1, n11.getParent().getParent());
  }
}
```

La classe abstraite Composante, un exemple possible et incomplet

```
public abstract class Composant{
    private String nom;
    public Composant(String nom){
        this.nom = nom;
    }
    public String getNom(){
        return this.nom;
    }
    public abstract int nombreDeNoeuds();

    public abstract String interpreter();

    public abstract <T> T accepter(Visiteur<T> visiteur);

    //...
```

La classe abstraite Composite, un exemple possible et incomplet

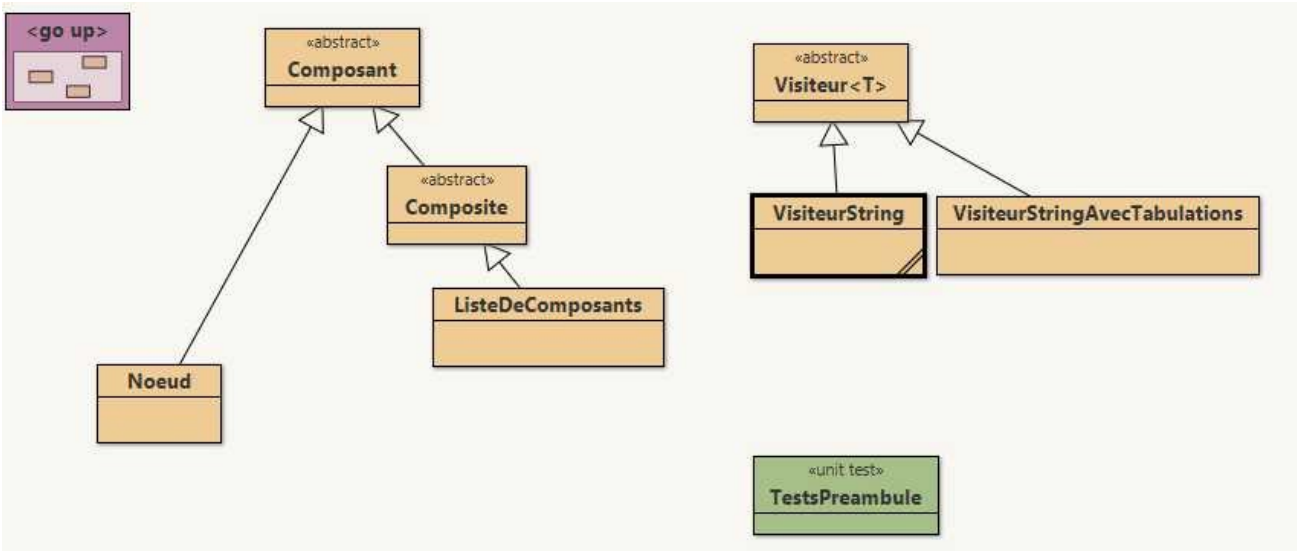
```
public abstract class Composite extends Composant implements Iterable<Composant>{
 // ...

    public Composite(String nom){
        super(nom);
    }

    public Composite ajouter(Composant composant){
        //...
        return this;
    }

    public int nombreDeNoeuds(){
        int nombre=0;
        //...
        return nombre;
    }

  //...

}
```

A ce stade du préambule, la classe visiteur est abstraite et complète

```
```

```
public abstract class Visiteur<T>{
  public T visite(Noeud n){ return null; }
  public T visite(ListeDeComposants n){ return null; }
}
```

Préambule-2)



Ajoutez à ce composite la prise en compte d'un visiteur dont son exécution engendre le même résultat que l'appel de la méthode interpreter, et vérifie le test ci-dessous

```
 public void testCompositeInterpreterEtVisiteur(){
       Noeud n1 = new Noeud("n1");
       Noeud n2 = new Noeud("n2");
       Noeud n3 = new Noeud("n3");
       ListeDeComposants l1 = new ListeDeComposants("l1");
       l1.ajouter(n1).ajouter(n2).ajouter(n3);
       assertEquals("Liste:l1[Noeud:<n1>Noeud:<n2>Noeud:<n3>]",l1.interpreter());
       Visiteur<String> visiteur = new VisiteurString();
       assertEquals(l1.accepter(visiteur), l1.interpreter());

       ListeDeComposants l11 = new ListeDeComposants("l11");
       Noeud n11 = new Noeud("n11");
       l11.ajouter(n11);
       l1.ajouter(l11);
       assertEquals("Liste:l1[Noeud:<n1>Noeud:<n2>Noeud:<n3>Liste:l11[Noeud:<n11>]]",l1.interpreter());
       assertEquals(l1.accepter(visiteur), l1.interpreter());
   }
```

Préambule-3)

Ajoutez un visiteur nommé **VisisteurStringAvecTabulations** dont son exécution vérifie le test ci-dessous

```
public void testCompositeVisiteurAvecTabulations(){
       Noeud n1 = new Noeud("n1");
       Noeud n2 = new Noeud("n2");
       Noeud n3 = new Noeud("n3");
       ListeDeComposants l1 = new ListeDeComposants("l1");
       l1.ajouter(n1).ajouter(n2).ajouter(n3);

       ListeDeComposants l11 = new ListeDeComposants("l11");
       Noeud n11 = new Noeud("n11");
       l11.ajouter(n11);
       l1.ajouter(l11);
       //assertEquals("Liste:l1[Noeud:<n1>Noeud:<n2>Noeud:<n3>Liste:l11[Noeud:<n11>]]",l1.interpreter());
       //System.out.println(l1.accepter(new VisiteurStringAvecTabulations()));
       String res = l1.accepter(new VisiteurStringAvecTabulations());
       assertTrue(res.startsWith("Liste:l1" + "\n" +
                         "\tNoeud:<n1>"+ "\n" +
                         "\tNoeud:<n2>"+ "\n" +
                         "\tNoeud:<n3>"+ "\n" +
                         "\tListe:l11" + "\n" +
                              "\t\tNoeud:<n11>" + "\n"));
```

Soit une sortie du même texte avec des tabulations, afin de représenter les différents stades de la structure de type composite

```
Liste:l1
   Noeud:<n1>
   Noeud:<n2>
   Noeud:<n3>
   Liste:l11
       Noeud:<n11>
```

Préambule-4)

Ajoutez un visiteur nommé **VisisteurCompositeValide** dont son exécution vérifie le test ci-dessous

```
public void testVisiteurCompositeValide(){
    Noeud n1 = new Noeud("n1");
    Noeud n2 = new Noeud("n2");
    Noeud n3 = new Noeud("n3");
    ListeDeComposants l1 = new ListeDeComposants("l1");
    l1.ajouter(n1).ajouter(n2).ajouter(n3);
    assertEquals("Liste:l1[Noeud:<n1>Noeud:<n2>Noeud:<n3>]",l1.interpreter());
    Visiteur<Boolean> visiteur = new VisiteurCompositeValide();
    assertTrue(l1.accepter(visiteur));

    ListeDeComposants l11 = new ListeDeComposants("l11");
    Noeud n11 = new Noeud("n11");
    l11.ajouter(n11);
    l1.ajouter(l11);
    visiteur = new VisiteurCompositeValide();
    assertTrue(l1.accepter(visiteur));

    ListeDeComposants l12 = new ListeDeComposants("n2");
    l12.ajouter(new Noeud("l1"));
    l1.ajouter(l12);
    visiteur = new VisiteurCompositeValide();
    assertTrue(l1.accepter(visiteur));

    l11.ajouter(new Noeud("n2"));
    assertEquals(6, l1.nombreDeNoeuds());
    visiteur = new VisiteurCompositeValide();
    assertFalse(l1.accepter(visiteur));

    ListeDeComposants l2 = new ListeDeComposants("l2");
    ListeDeComposants l22 = new ListeDeComposants("l22");
    visiteur = new VisiteurCompositeValide();
    assertFalse(l2.accepter(visiteur));
 }
```

- Un composite est valide si il n'existe pas de noeuds avec le même nom
- Un composite est valide si il n'existe pas de liste de composants avec le même nom
- Un composite est valide si il n'existe pas de liste de composants sans composant

/* Une idée...