

TP 7

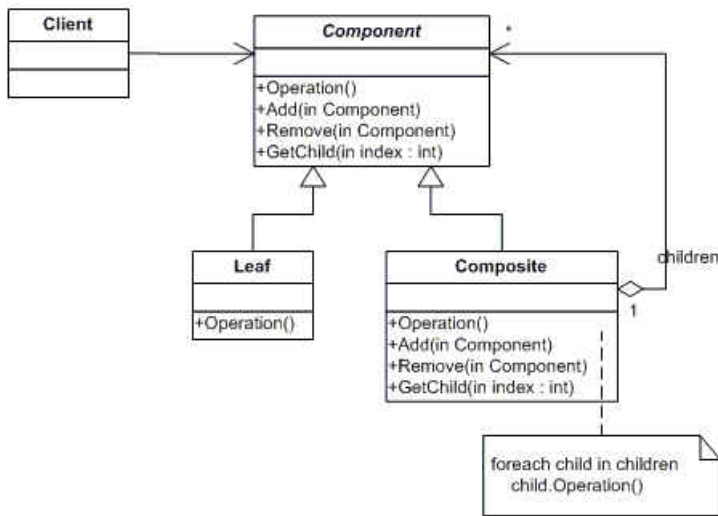
Thème :

- *Assemblage de patterns : les transactions*
- *Composite, Itérateur, Visiteur*
- *Commande, Template Method, Memento*

- Visualisez dans un navigateur le sujet en ouvrant `index.html` du répertoire qui a été créé à l'ouverture de `tp7.jar` par BlueJ; vous aurez ainsi accès aux différents liens qui sont proposés pour vous aider, et aux applettes.
- Soumettez chaque question à l'outil d'évaluation JNews.



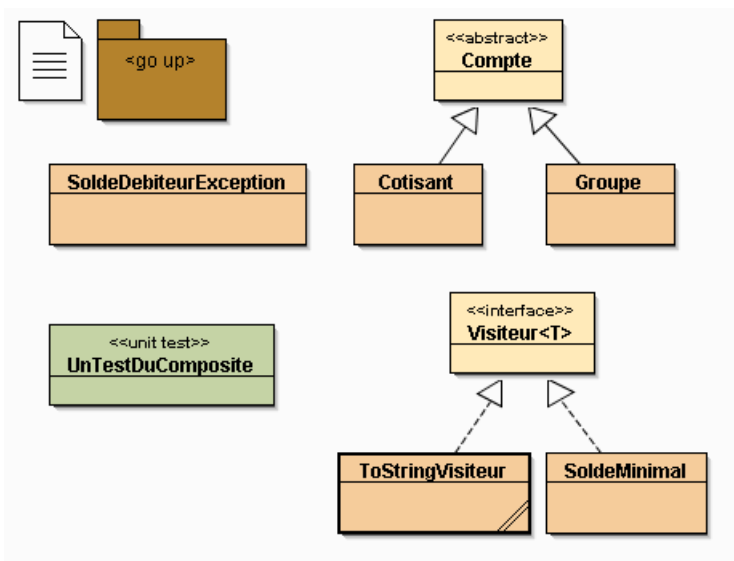
1.1) Complétez l'architecture du Composite `Compte`.



Le patron composite

Le Composite ci-dessous reflète, la "gestion" des comptes de cotisants solidaires.

- **Compte** est une classe abstraite, *soit le Component*
- **Cotisant** une classe concrète, feuille terminale dans ce Composite, *soit Leaf*.
- La classe **Groupe**, structure un ensemble hiérarchique de comptes, *Composite*.



```

public abstract class Compte{
    protected String nom;
    protected Compte parent;

    public Compte( String nom ) { this(nom,null); }
    public Compte( String nom, Compte parent ) { this.nom=nom; this.parent=parent; }

    public abstract void debit( int somme ) throws SoldeDebiteurException;
    public abstract void credit( int somme );
    public abstract int solde();
    public abstract int nombreDeCotisans();

    public String nom(){ return nom; }
    public boolean equals( Object o ) { return nom.equals( ((Compte)o).nom ); }
    public void setParent( Compte parent ) { this.parent=parent; }
    public Compte getParent() { return parent; }

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

```

à noter :

- La **somme** en paramètre du constructeur et des méthodes `credit` et `debit` ne peut-être que ≥ 0 .
 - Une exception de type `RuntimeException("nombre négatif !!!");` doit être levée dans le cas contraire
- Si un cotisant ou un groupe ne peut être débité de la somme souhaitée,
 - une exception `SoldeDebiteurException` est levée

```

public class Groupe extends Compte implements Iterable<Compte>{

```

- C'est la classe "*Composite*" de `Compte`
- La classe *Groupe* implémente les méthodes de la classe `Compte`, fournit le nombre de cotisans et également un itérateur (`Iterator<Compte> iterator()`)
- Utilisez la classe de tests fournie, afin de vérifier le comportement attendu

notamment les méthodes de test nommées

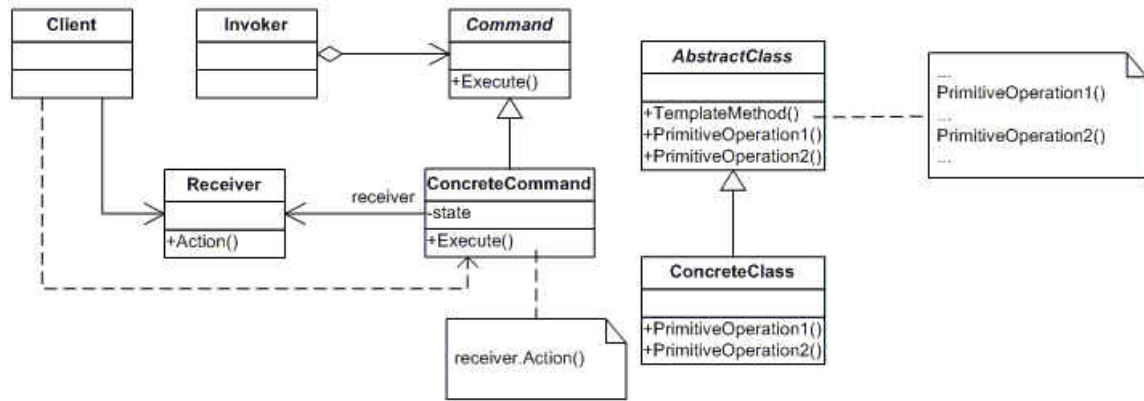
- `testUnGroupeAvecUnSoldeDebiteurDuPremierCotisant()`
- `testUnGroupeAvecUnSoldeDebiteurDuDernierCotisant()`
- `testXXXX`

Attention : A cette question, lors de la première occurrence d'un solde débiteur, la méthode *débit* d'un groupe s'interrompt (une exception est levée). **Certains cotisans ont été débités, d'autres non, !!!,** **C'est bien une erreur de conception,** ce débit doit être associé à une transaction et à cette question aucun "retour arrière" (rollback) n'est demandé.

1.2) Une proposition peu fiable : Complétez le visiteur `SoldeMinimal`, ce visiteur permet d'obtenir le solde minimal que l'on pourrait débiter d'un "Compte Composite" . C'est un test que l'utilisateur pourrait effectuer avant de débiter d'une somme précise. (*mais il pourrait aussi oublier de le faire...*).

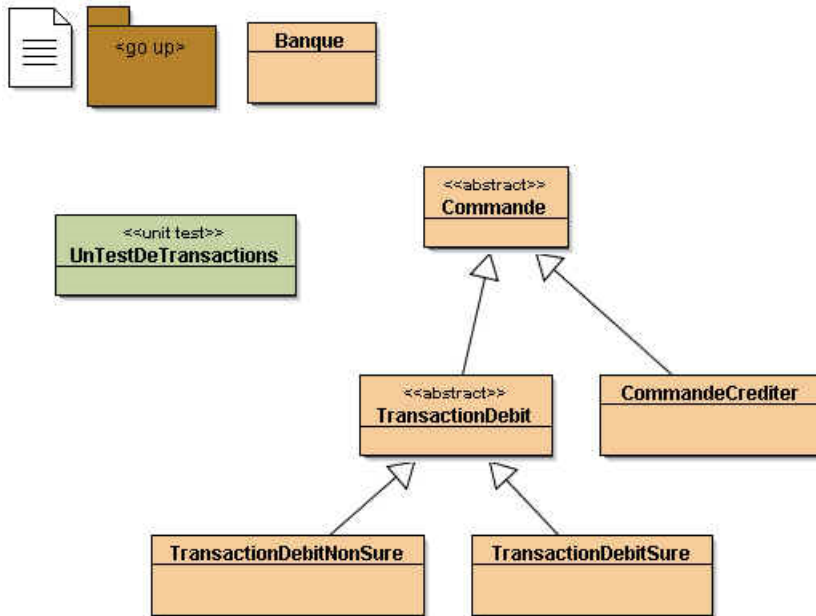
question2

Transaction et la combinaison de pattern Command/TemplateMethod



L'architecture ci-dessous reflète, l'usage combiné du pattern Command et du pattern Template Method.

- Complétez les classes Banque (*Invoker*), TransactionDebitSure (*ConcreteCommand*(du pattern Command)& *AbstractClass*(*Template Method*))



L'interface Commande

```

public abstract class Commande {
    public abstract void execute(int somme);
}

```

La commande concrète : **TransactionDebit**, utilise ici le pattern Template Method :

```

public abstract class TransactionDebit extends Commande{
    protected Compte compte;

    public abstract void beginTransaction(); // Template Method
    public abstract void endTransaction();
    public abstract void rollbackTransaction();

    public TransactionDebit(Compte compte){
        this.compte = compte;
    }
}

```

```
public void execute(int somme){ // commande atomique
    try{
        beginTransaction();
        compte.debit(somme);
        endTransaction();
    }catch(Exception e){
        rollbackTransaction();
    }
}
```

La méthode **rollbackTransaction** utilise le patron Memento; vous pouvez modifier en conséquence les classes de la question 1 et afin de rester compatible avec les tests unitaires. Les modifications doivent être des ajouts de champs, méthodes, classes internes ... (en compatibilité ascendante, voir si besoin le chapitre 13 de Java Language Specification http://java.sun.com/docs/books/jls/third_edition/html/binaryComp.html)

Aide :

- Dans Compte : interface Memento
- Dans Cotisant : classe MementoCotisant
- Dans Groupe : classe MementoGroupe

La classe TransactionDebit**NonSure** ci-dessous est complète

```
public class TransactionDebitNonSure extends TransactionDebit{

    public TransactionDebitNonSure(Compte compte){
        super(compte);
    }
    public void beginTransaction(){
    }
    public void endTransaction(){
    }
    public void rollbackTransaction(){
        throw new UnsupportedOperationException();
    }
}
```

Un test parmi d'autres :

```
public void testTransactionDebitSure1(){
    try{
        Groupe g = new Groupe("g");
        g.ajouter(new Cotisant("g_a",300));
        g.ajouter(new Cotisant("g_b",200));
        g.ajouter(new Cotisant("g_c",100));
        Banque b = new Banque();
        b.installerLesCommandes( new TransactionDebitSure(g), new CommandeCrediter(g));
        int ancienSolde = g.solde();
        b.debiter(120);
        assertTrue( " transaction erronée , l'ancien solde n'est pas restitué !!! ",
            g.solde()==ancienSolde );
    }catch(Exception e){
        fail("exception inattendue ??? "+ e.getClass().getName());
    }
}
```