

Algorithmique avancée TD n° 5

Écritures infixée, préfixée, postfixée.

Introduction.

La classe Arbre comporte une variable membre de type Nœud (la racine) et une méthode qui renvoie la racine. La classe **Noeud** est détaillée ci-dessous (vous remarquerez qu'elle a deux variables membre de type **Noeud**). La classe **DefaultMutableTreeNode** est une classe modèle pour l'affichage graphique d'arbres dans le schéma MVC. Elle sert à la visualisation de l'arbre.

- a. En laissant de côté les lignes marquées //v qui servent à la visualisation, cette classe est-elle conforme à l'implémentation des arbres donnée en cours ?
- b. Quel est le principe de l'appel aux fonctions de visualisation ?

```
import java.io.*;
import javax.swing.tree.* ;
import javax.swing.* ;

// Classe Nœud pour la construction d'arbres, avec affichage graphique incorporé.

public class Noeud implements Serializable    {

    private DefaultMutableTreeNode noeud; //v
    private Noeud filsGauche;
    private Noeud frere;

    // ----- CONSTRUCTEURS

    public Noeud()
    {   noeud = new DefaultMutableTreeNode();
        filsGauche = null;
        frere = null;
    }

    public Noeud(String nomFich)throws FileNotFoundException, IOException,
                                   ClassNotFoundException
    {charger(nomFich);}

    // ----- CREATIONS

    public void creerFilsGauche() // sans y mettre de valeur
    {   filsGauche= new Noeud();
        noeud.insert(filsGauche.noeud,0); //v
    }

    public void creerFilsGauche(Noeud n)
    {   filsGauche= n;
        noeud.insert(filsGauche.noeud,0); //v
    }

    public void creerFrere() // sans y mettre de valeur
    {   frere= new Noeud();
        DefaultMutableTreeNode pere = (DefaultMutableTreeNode) noeud.getParent() ; //v
        pere.add(frere.getNoeud()); //v
    }
}
```

```

public void creerFrere(Noeud n)
{
    frere= n ;
    DefaultMutableTreeNode pere = (DefaultMutableTreeNode) noeud.getParent() ; //v
    pere.add(frere.getNoeud());//v
}

// ----- METHODES DE SERVICE

public Object getObjet() { return (noeud.getUserObject()); }

public Noeud getFilsGauche() { return (filsGauche) ; }

public Noeud getFrere() { return (frere) ; }

private DefaultMutableTreeNode getNoeud() { return(noeud); }

public void setObjet(Object o) { noeud.setUserObject(o); }

public String toString() {return getObjet().toString() ; }

// ----- IDENTITE

public boolean estRacine(){return(noeud.isRoot()); } //v

public boolean estFeuille(){ return(noeud.isLeaf()); } // peut être fait directement

// ----- SAUVEGARDE

public void sauver(String nomFich) throws FileNotFoundException, IOException
{
    FileOutputStream fich=new FileOutputStream(nomFich);
    ObjectOutputStream out = new ObjectOutputStream(fich) ;
    out.writeObject(noeud); //v
    out.writeObject(filsGauche);
    out.writeObject(frere);
    out.close();
    fich.close();
}

public void charger(String nomFich) throws FileNotFoundException, IOException,
                    ClassNotFoundException
{
    FileInputStream fich=new FileInputStream(nomFich);
    ObjectInputStream inp = new ObjectInputStream(fich) ;
    noeud=(DefaultMutableTreeNode)inp.readObject(); //v
    filsGauche=(Noeud)inp.readObject();
    frere=(Noeud)inp.readObject();
    inp.close();
    fich.close();
}

// INTERFACAGE GRAPHIQUE

public JTree getJTree() //v
{
    return new JTree(new DefaultTreeModel(noeud));
}

} // end class Noeud

```

Expressions arithmétiques :

Une expression arithmétique peut prendre trois formes : **préfixée, postfixée et infixée.**

Cette expression peut s'insérer dans un arbre binaire

A une expression arithmétique donnée correspond donc trois écritures et un arbre binaire.

L'objectif du td est de rédiger trois classes **Infixee** , **Prefixee** , et **Postfixee** permettant de passer d'une forme d'écriture à l'arbre binaire et réciproquement.

Ces trois classes ayant la même structure, elles dériveront d'une même classe abstraite **Expression** qui stocke à la fois l'écriture et l'arbre (il suffit de pointer sur sa racine), et dont voici le code (disponible dans tpinfo) :

```
import java.io.*;

public abstract class Expression    {
    protected String expression;
    protected Noeud arbre;

    public Expression(String s)
    { expression = new String(s);
      fabriquerArbre();
    }

    public Expression(Noeud a)
    { arbre = a;
      reconstituerExpression();
    }

    // constructeur à partir d'un arbre enregistré
    public Expression(String chemin,String nomFich)
        throws FileNotFoundException, IOException,ClassNotFoundException
    { nomFich = chemin + nomFich;
      arbre = new Noeud(nomFich);
      reconstituerExpression();
    }

    protected abstract void reconstituerExpression();
    protected abstract void fabriquerArbre();

    public String getExpression() {return(expression);}
    public Noeud getArbre() { return(arbre);}

    public void sauver(String nomFich)    throws FileNotFoundException, IOException
        {arbre.sauver(nomFich);}

    public int valeur()
        { // à compléter
          // end valeur
        }
    } // end class Expression
```

Question 1 :

Pour chacune des trois classes **Prefixee** , **Postfixee** et **Infixee** quel algorithme permet de reconstituer une expression à partir de l'arbre ? Implémenter cet algorithme dans la méthode :

protected void reconstituerExpression();

qui est appelée quand on crée une expression en lui passant un arbre, et qui calcule l'écriture (préfixée, postfixée, infixée) de l'arbre (un algorithme et une méthode par classe !). Pour les tests, on pourra utiliser la classe `TreeMaker` fournie : la méthode `newtree(String s)` de cette classe reçoit une expression infixée totalement parenthésée, crée un arbre et renvoie un `Nœud` qui est la racine de cet arbre. A l'aide de cet arbre, on peut tester que la reconstitution de l'expression est correcte. Dans le cas d'une expression infixée, la méthode **reconstituerExpression()** fournira une chaîne entièrement parenthésée, identique à celle passée en entrée, aux espaces près.

La classe `Test.java` fournie contient un exemple de test.

Question 2

Pour chacune des trois classes **Prefixee**, **Postfixee** et **Infixee**, quels sont les algorithmes qui permettent de reconstituer l'arbre à partir de l'expression ? Implémenter cet algorithme dans la méthode :

```
protected abstract void fabriquerArbre();
```

Pour la mise au point, on utilisera la classe `FenEssai.class` dans laquelle on peut saisir une expression et afficher l'arbre obtenu, et les programmes `TestPostfixe.java`, `TestPrefixe.java` et `TestInfixe.java`.

Dans `TPINFO/AlgoA/td5` :

`libexpression.jar` contient les classes `Nœud`, `Expression`, `FenEssai`

`pile.jar` contient `Pileint` et une classe `PileObject` qui empile les types non-primitifs

`TreeMaker.class` est fourni en version compilée

les fichiers de test sont en code source.