TP noté
E.C. INFO-12401C
« Programmation objet »

F. BERTRAND

Année universitaire 2015–2016

Version avec éléments de correction

1 Un système de fichiers

Le but de ce TP est de mettre en œuvre la gestion simplifiée d'un système de fichiers comportant des répertoires et des fichiers.

Pour rappel, dans un système de fichiers :

— il existe un répertoire racine dans lequel se situent l'ensemble des sous-répertoires et des fichiers ;

— un répertoire peut contenir des fichiers mais pas l'inverse !...

Le diagramme de classes représentant les différentes relations entre celles-ci est présenté sur la Figure 1.

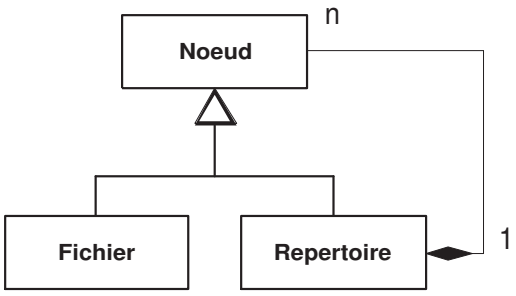

FIGURE 1 – Les différentes classes d'un système de fichiers

1.1 Mise en œuvre des classes `Fichier` et `Repertoire` à partir de la version initiale de la classe `Noeud`

Soit la classe abstraite `Noeud` regroupant les attributs et les méthodes communs aux deux-sous classes `Fichier` et `Repertoire` :

```
public abstract class Noeud {

    // à compléter...

    public abstract boolean ajouteElt(Noeud nouveau);
    public abstract boolean supprimeElt(Noeud existant);
    public abstract ArrayList<Noeud> donneElementsFils();
}
```

Dans un premier temps, le seul attribut commun aux classes `Fichier` et `Repertoire` sera un nom (un fichier et un répertoire sont identifiés par un nom).

Les méthodes `ajouteElt` et `supprimeElt` permettent de gérer le système de fichiers. Concernant la méthode `donneElementsFils`, elle retournera tous les éléments fils directs du répertoire sur lequel la méthode est appelée.

Écrire les classes `Fichier` et `Repertoire` sachant que :

— dans la classe `Fichier`, les méthodes `ajouteElt` et `supprimeElt` ne feront que lancer une exception de type `UnsupportedOperationException` (exception d'exécution) pour indiquer qu'il n'est pas possible d'ajouter des éléments (fichiers ou répertoires) à un fichier. La méthode `donneElementsFils` retournera une liste vide.

— dans la classe `Repertoire`, la méthode `ajouteElt` devra s'assurer que dans le répertoire où doit s'effectuer l'ajout, il n'existe pas déjà un élément (fichier ou répertoire) possédant le même nom. Pour les méthodes `ajouteElt` et `supprimeElt`, le renvoi d'une valeur `true` indiquera que l'opération s'est correctement effectuée.

Les tests 1, 2, 3 et 4 de la classe de test concernent cette première partie.

1.2 Calcul de la taille d'un élément

On décide d'ajouter à la classe `Noeud`, la méthode suivante :

```
    public abstract int taille();
```

Modifier les classes `Fichier` et `Repertoire` en conséquence sachant que :

— pour la classe `Fichier`, la taille sera représentée par un attribut qui sera initialisé à la création de l'objet ;

— pour la classe `Repertoire`, il n'y aura pas d'attribut, la taille sera calculée lors de l'appel à la méthode `taille` sachant que la taille d'un répertoire sera égale à la somme des tailles de ses éléments et qu'un répertoire vide aura une taille égale à zéro.

Le test 5 de la classe de test concerne cette seconde partie.

1.3 Recherche d'un élément

On décide d'ajouter à la classe `Noeud`, la méthode suivante :

```
    public abstract ArrayList<Noeud> rechercheElt(String nom);
```

Modifier les classes `Fichier` et `Repertoire` en conséquence sachant que :

— pour la classe `Fichier`, si le nom du fichier correspond au nom recherché alors une liste contenant un seul objet, l'instance de `Fichier`, sera retournée. Dans le cas contraire, une liste vide sera retournée ;

— pour la classe `Repertoire`, on vérifiera si le nom recherché correspond au nom du répertoire puis on effectuera *récursivement* cette recherche pour chaque élément fils. À chaque fois qu'un élément de même nom sera trouvé, il sera ajouté à la liste résultat. À noter que lors de la recherche, il n'y a pas besoin de distinguer si le fils est un fichier ou un répertoire, il suffira de tester la taille de la liste retournée. Si celle-ci est vide,

cela signifiera que la recherche a échouée. Dans le cas contraire, les éléments de la liste retournée seront ajouté à la liste résultat.

Les tests 6 et 7 de la classe de test concernent cette troisième partie.

1.4 Ajout de la notion de parent

On souhaite que chaque élément (fichier ou répertoire) possède une référence à son parent. Il existe une exception cependant : le répertoire racine n'aura pas de parent.

Modifier votre code en conséquence pour gérer cette information.

Le test 8 de la classe de test concerne cette quatrième partie.

1.5 Chemins d'accès à un élément

On décide d'ajouter à la classe `Noeud`, la méthode suivante :

```
public abstract ArrayList<Noeud> donneChemin(int identifiant);
```

Cette méthode retourne une liste contenant les éléments traversés pour accéder à un élément (répertoire ou fichier) représenté ici par son identifiant (qui peut correspondre soit à un répertoire, soit à un fichier donc, pour cette raison, la valeur retournée est une liste de `Noeud` et non une liste de `Repertoire`). Cette liste inclut le nom de l'élément et le répertoire à partir duquel cette méthode est appelée. Si l'identifiant ne correspond à aucun élément alors une liste vide sera retournée.

Par exemple, considérons un fichier `A.java` situé dans un répertoire `P2` lui-même contenu dans un répertoire `P1` :

— Si la méthode `donneChemin` est appelée dans `P2`, la liste retournée contiendra le répertoire `P2` et le fichier `A.java`;

— Si la méthode `donneChemin` est appelée dans `P1`, la liste retournée contiendra le répertoire `P1`, répertoire `P2` et le fichier `A.java`;

La mise en œuvre de cette méthode `donneChemin` s'effectuera en deux étapes :

1. Pour que cette méthode fonctionne, il faut attribuer un identifiant unique à chaque élément lors de sa création. En modifiant la classe `Noeud`, mettez en œuvre une solution permettant d'attribuer un identifiant unique à chaque élément lors de sa création (ne pas chercher à rendre à nouveau disponible l'identifiant d'un élément supprimé).
2. implémenter la méthode `donneChemin` dans les classes `Fichier` et `Repertoire` en sachant que, pour la classe `Fichier`, si l'identifiant correspond à celui du fichier alors une liste contenant uniquement l'objet représentant le fichier sera retournée.

Les tests 8 à 13 de la classe de test concernent cette dernière partie.

Éléments de correction

```
1
2   import java.util.ArrayList;
3
4   /**
5    * Represente le concept d'element d'un systeme de fichier.
6    * Les methodes abstraites seront definies dans les sous-classes .
7    *
8    */
9   public abstract class Noeud {
10
11      // nom du noeud
12      protected String nom;
13      // repertoire contenant le noeud
14      protected Repertoire parent;
15      // identifiant unique du noeud
16      protected int identifiant;
17
18      // permet de numeroter les elements crees
19      protected static int compteur = 0;
20
21      public Noeud(String nom) {
22          this.nom = nom;
23          this.parent = null;
24          this.identifiant = compteur++;
25      }
26
27      /** Ajoute un element (fichier ou repertoire) au noeud courant
28       *
29       * @param nouveau nouveau l'element a ajouter
30       * @return true si un element de meme nom n'existe pas deja, false sinon
31       */
32      public abstract boolean ajouteElt(Noeud nouveau);
33
34      /**
35       * Supprime un element (fichier ou repertoire) au noeud courant
36       * @param existant element a supprimer
37       * @return true si l'elt existe, false sinon
38       */
39      public abstract boolean supprimeElt(Noeud existant);
40
41      /**
42       * Retourne la liste des elements presents dans le noeud courant
43       * @return liste des elements presents dans le noeud courant
44       */
45      public abstract ArrayList<Noeud> donneElementsFils();
46
47      /** Recherche recursive des elements possedant le nom fourni en parametre
48       *  presents dans le noeud courant ou ses fils.
49       *
50       * @param nom nom des elts recherches
51       * @return liste des elements possedant ce nom
52       */
53      public abstract ArrayList<Noeud> rechercheElt(String nom);
54
55      /** Retourne la taille en octets egale a la somme des tailles de tous
56       *  les elements contenus dans le repertoire
57       *
58       * @return taille du repertoire
59       */
60      public abstract int taille();
61
62      /** Retourne la liste des elements traverses pour acceder a l'elt
63       *  represente par son identifiant
64       *
65       * @param identifiant identifiant de l'element
66       * @return le chemin pour acceder a cet element
67       */
68      public abstract ArrayList<Noeud> donneChemin(int identifiant);
69
70      /** Retourne le nom du noeud
71       *
72       * @return le nom du noeud
73       */
74      public String donneNom() {
75          return this.nom;
76      }
77
78      /** Retourne le repertoire parent du noeud
```

```
     *
     * @return le repertoire parent du noeud
     */
    public Repertoire donneParent() {
        return this.parent;
    }

    /** Retourne l'identifiant (unique) du Noeud
     *
     * @return l'identifiant du noeud
     */
    public int donneId() {
        return this.identifiant;
    }

    /** Retourne une description textuelle d'un noeud
     *
     * @return une chaine de caracteres decrivant l'objet
     */
    public String toString() {
        return "nom=" + nom;
    }
}
```

```

import java.util.ArrayList;

/**
 * Represente le concept de fichier dans un systeme de fichiers.
 *
 */
public class Fichier extends Noeud {

    // taille du fichier en octets
    private int taille;

    /** Cree un fichier dans un systeme de fichiers
     *
     * @param nom nom du fichier
     * @param taille taille du fichier en octets
     */
    public Fichier(String nom, int taille) {
        super(nom);
        this.taille = taille;
    }

    /** L'ajout d'elements a un fichier est interdite car un fichier ne
     *  peut contenir aucun element.
     *
     * @param nouveau element a ajouter
     * @throws UnsupportedOperationException
     * @return rien
     */
    public boolean ajouteElt(Noeud nouveau) {
        throw new UnsupportedOperationException(this.nom + " n'est pas un repertoire");
    }

    /** La suppression d'elements est interdite car un fichier ne
     *  peut contenir aucun element.
     *
     * @param existant element a supprimer
     * @throws UnsupportedOperationException
     * @return rien
     */
    public boolean supprimeElt(Noeud existant) {
        throw new UnsupportedOperationException(this.nom + " n'est pas un repertoire");
    }

    /** Un fichier n'a aucun element fils...
     *
     * @return une liste vide
     */
    public ArrayList<Noeud> donneElementsFils() {
        return new ArrayList<Noeud>();
    }

    /** Recherche si le fichier possede le nom indique en parametre
     *
     * @param nom nom du fichier recherche
     * @return une liste contenant le fichier si son nom correspond, vide sinon
```

```
     */
    public ArrayList<Noeud> rechercheElt(String nom) {
        ArrayList<Noeud> resultat = new ArrayList<Noeud>();
        if (this.nom.equals(nom)) {
            resultat.add(this);
        }
        return resultat;
    }

    /** Retourne la taille du fichier
     *
     * @return taille en octets du fichier
     */
    public int taille() {
        return this.taille;
    }

    /** Retourne une liste d'elements correspondant a l'identifiant
     *
     * @param identifiant identifiant du fichier recherche
     * @return une liste contenant le fichier si son id correspond, vide sinon
     */
    public ArrayList<Noeud> donneChemin(int identifiant) {
        ArrayList<Noeud> chemin = new ArrayList<Noeud>();
        if (this.identifiant==identifiant) {
            chemin.add(this);
        }
        return chemin;
    }

    /** Retourne une description textuelle d'un fichier
     *
     * @return une chaine de caracteres decrivant l'objet
     */
    public String toString() {
        return "Fichier{" + super.toString() + " ,taille=" + taille + '}';
    }

}
```

```

import java.util.ArrayList;

/**
 * Represente le concept de repertoire dans un systeme de fichiers.
 *
 */
public class Repertoire extends Noeud {

    private ArrayList<Noeud> elements;

    /**
     * Cree un repertoire dans un systeme de fichiers
     *
     * @param nom nom du repertoire a creer
     */
    public Repertoire(String nom) {
        super(nom);
        this.elements = new ArrayList<Noeud>();
    }

    /**
     * Ajoute un element (fichier ou repertoire) au repertoire
     *
     * @param nouveau l'element a ajouter
     * @return true si un element de meme nom n'existe pas deja, false sinon
     */
    public boolean ajouteElt(Noeud nouveau) {
        // recherche s'il n'existe pas deja un noeud avec le meme nom
        for (Noeud n : this.elements) {
            if (n.nom.equals(nouveau.nom)) {
                return false;
            }
        }
        // ajout de l'element
        this.elements.add(nouveau);
        // maj du parent
        nouveau.parent = this;
        return true;

```

```
41      }
42
43      /**
44       * Supprime un element (fichier ou repertoire) du repertoire
45       *
46       * @param existant element a supprimer
47       * @return true si l'elt existe, false sinon
48       */
49      public boolean supprimeElt(Noeud existant) {
50          // suppression de l'element
51          boolean sup = this.elements.remove(existant);
52          // maj du parent
53          existant.parent = null;
54          return sup;
55      }
56
57      /**
58       * Retourne la liste des elements presents dans le repertoire
59       *
60       * @return liste des elements du repertoire
61       */
62      public ArrayList<Noeud> donneElementsFils() {
63          return new ArrayList<Noeud>(this.elements);
64      }
65
66      /**
67       * Recherche recursive des elements possedant le nom fourni en parametre
68       * presents dans le repertoire ou les sous-repertoires
69       *
70       * @param nom nom des elts recherches
71       * @return liste des elements possedant ce nom
72       */
73      public ArrayList<Noeud> rechercheElt(String nom) {
74          ArrayList<Noeud> resultat = new ArrayList<Noeud>();
75          // on test d'abord le nom du repertoire
76          if (this.nom.equals(nom)) {
77              resultat.add(this);
78          }
79          // debut de la recherche recursive
80          for (Noeud n : this.elements) {
81              ArrayList<Noeud> recherche = n.rechercheElt(nom);
82              if (!recherche.isEmpty()) {
83                  resultat.addAll(recherche);
84              }
85          }
86          return resultat;
87      }
88
89      /**
90       * Retourne la taille en octets egale a la somme des tailles de tous les
91       * elements contenus dans le repertoire
92       *
93       * @return taille du repertoire
94       */
95      public int taille() {
96          int taille = 0;
97          for (Noeud n : this.elements) {
98              taille += n.taille();
99          }
100         return taille;
101     }
102
103     /**
104      * Retourne la liste des elements travers'es pour acceder a l'element
105      * correspondant a l'identifiant
106      *
107      * @param identifiant identifiant de l'element
108      * @return le chemin pour acceder a cet element
109      */
110     public ArrayList<Noeud> donneChemin(int identifiant) {
111         ArrayList<Noeud> chemin = new ArrayList<Noeud>();
112         // on verifie si l'elt recherche est un repertoire
113         if (this.identifiant == identifiant) {
114             chemin.add(this);
115         } else {
116             boolean trouve = false;
117             int indexMax = this.elements.size();
118             int i = 0;
119             ArrayList<Noeud> cheminFils;
120             while (!trouve && i < indexMax) {
```

```
                cheminFils = this.elements.get(i).donneChemin(identifiant);
                // si l'elt a ete trouve (liste non vide)
                if (!cheminFils.isEmpty()) {
                    // on ajoute le repertoire courant
                    chemin.add(this);
                    // on ajoute la liste correspondant au chemin trouve
                    chemin.addAll(cheminFils);
                    // la recherche est terminee...
                    trouve = true;
                } else {
                    // sinon on regarde le noeud suivant...
                    i++;
                }
            }
        }
        return chemin;
    }

    /**
     * Retourne une description textuelle d'un repertoire
     *
     * @return une chaine de caracteres decrivant l'objet
     */
    public String toString() {
        return "Repertoire{" + super.toString() + /* " ,elements=" + elements +*/ '}';
    }

}
```

```

import java.util.ArrayList;
import java.util.Collections;


public class TestSystemeFichiers {

    public static void main(String[] args) {
        Repertoire racine = new Repertoire("racine");
        Repertoire rep1 = new Repertoire("rep1");
        Fichier f1 = new Fichier("f1", 100);
        Fichier f2 = new Fichier("f2", 150);

        racine.ajouteElt(rep1);
        rep1.ajouteElt(f2);
        racine.ajouteElt(f1);

        System.out.println("Debut test...");
        // test 1 : ajout d'un repertoire de meme nom
        Repertoire rep2 = new Repertoire("rep1");
        if (racine.ajouteElt(rep2))
            throw new Error("ajout d'un element de meme nom");

        // test 2 : ajout d'un fichier de meme nom
        Fichier f2bis = new Fichier("f2", 150);
        if (rep1.ajouteElt(f2bis))
            throw new Error("ajout d'un element de meme nom");

        // test 3 : suppression d'un element inexistant
        Repertoire rep3 = new Repertoire("rep3");
        if (racine.supprimeElt(rep3))
            throw new Error("suppression d'un element inexistant");

        // test 4 : liste des elements fils
        if (!rep1.ajouteElt(rep3))
           throw new Error("ajout element " + rep3 + " impossible");
        ArrayList<Noeud> fils = rep1.donneElementsFils();
        ArrayList<Noeud> resultatFils = new ArrayList<Noeud>();
        resultatFils.add(f2);
        resultatFils.add(rep3);
        if (!fils.equals(resultatFils))
           throw new Error("elements fils differents");

/* A DECOMMENTER AU FUR ET A MESURE DE VOTRE PROGRESSION....   */
        // test 5 : calcul de la taille d'un repertoire
        if (racine.taille() != 250)
            throw new Error("calcul taille incorrect");

        // test 6 : recherche d'un element ayant le nom 'rep3'
        // et se situant a un niveau quelconque du systeme de fichiers
        ArrayList<Noeud> recherche = racine.rechercheElt("rep3");
```

```
52          ArrayList<Noeud> resultatRecherche = new ArrayList<Noeud>();
53          resultatRecherche.add(rep3);
54          if (!recherche.equals(resultatRecherche))
55              throw new Error("probleme sur le recherche d'un element");
56
57          // creation d'un repertoire et d'un ficher de meme nom
58          Repertoire testRep = new Repertoire("test");
59          Fichier testFic = new Fichier("test", 75);
60          if (!rep3.ajouteElt(testRep))
61             throw new Error("ajout element " + testRep + " impossible");
62          if (!rep1.ajouteElt(testFic))
63             throw new Error("ajout element " + testFic + " impossible");
64
65          // test 7 : recherche d'elements ayant le nom 'test'
66          // et se situant a un niveau quelconque du systeme de fichiers
67          recherche = racine.rechercheElt("test");
68          resultatRecherche.clear();
69          resultatRecherche.add(testFic);
70          resultatRecherche.add(testRep);
71          // pour eviter qu'il y ait des problemes d'ordre dans le resultat
72          if (!recherche.containsAll(resultatRecherche) ||
73                  !resultatRecherche.containsAll(recherche))
74              throw new Error("probleme sur le recherche d'un element");
75
76          // test 8 : verification du parent
77          if (racine.donneParent() != null)
78             throw new Error("probleme sur la gestion du parent");
79          if (rep3.donneParent() != rep1 || rep1.donneParent() != racine)
80             throw new Error("probleme sur la gestion du parent");
81          if (!racine.supprimeElt(f1))
82              throw new Error("suppression impossible de " + f1);
83          if (f1.donneParent() == racine)
84             throw new Error("probleme sur la gestion du parent");
85
86          // test 9 : verification du chemin absolu pour un fichier existant
87          Fichier f3 = new Fichier("f3", 35);
88          if (!rep3.ajouteElt(f3))
89             throw new Error("ajout element " + f3 + " impossible");
90          // maintenant le chemin absolu de f3 est : racine/rep1/rep3/f3
91          ArrayList<Noeud> cheminF3absolu = racine.donneChemin(f3.donneId());
92          ArrayList<Noeud> resultatCheminf3 = new ArrayList<Noeud>();
93          // ajout des elements a la liste representant le 'bon' resultat
94          Collections.addAll(resultatCheminf3,racine,rep1,rep3,f3);
95          if (!cheminF3absolu.equals(resultatCheminf3))
96             throw new Error("chemin absolu de " + f3 + " incorrect");
97
98          // test 10 : verification du chemin relatif pour un repertoire existant
99          ArrayList<Noeud> cheminRep3absolu = rep1.donneChemin(rep3.donneId());
100         ArrayList<Noeud> resultatCheminRep3 = new ArrayList<Noeud>();
101         Collections.addAll(resultatCheminRep3,rep1,rep3);
102         if (!cheminRep3absolu.equals(resultatCheminRep3))
103            throw new Error("chemin relatif de " + rep3 + " incorrect");
104
105         // test 11 : verification du chemin relatif pour un fichier existant
106         ArrayList<Noeud> cheminF3relatif = rep3.donneChemin(f3.donneId());
107         ArrayList<Noeud> resultatCheminF3relatif = new ArrayList<Noeud>();
108         Collections.addAll(resultatCheminF3relatif,rep3,f3);
109         if (!cheminF3relatif.equals(resultatCheminF3relatif))
110            throw new Error("chemin relatif de " + f3 + " incorrect");
111
112         // test 12 : verification du chemin relatif pour un fichier existant
113         cheminF3relatif = f3.donneChemin(f3.donneId());
114         resultatCheminF3relatif.remove(rep3);
115         if (!cheminF3relatif.equals(resultatCheminF3relatif))
116            throw new Error("chemin relatif de " + f3 + " incorrect");
117
118         // test 13 : verification du chemin pour un fichier inaccessible
119         ArrayList<Noeud> cheminVide = rep3.donneChemin(f2.donneId());
120         if (!cheminVide.isEmpty())
121            throw new Error("chemin de trouv'e pour un elt inaccessible");
122 //*/
123
124         System.out.println("Fin test.");
125     }
126
127 }
```