TP n° 3
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

1 Système de fichiers

Dans ce TP nous allons reprendre l'exemple vu en cours concernant la représentation d'un système de fichiers. Soit une représentation très simplifiée d'un système de fichiers dont le diagramme de classes est donné ci-dessous sur la Figure 1.

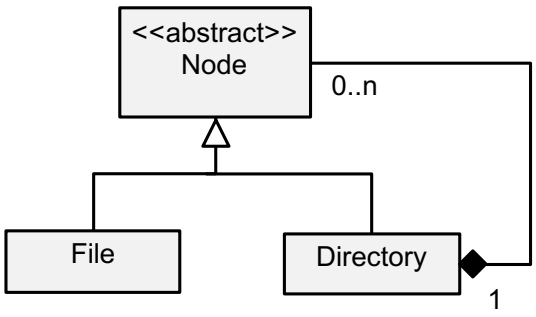

FIGURE 1 – Représentation (simplifiée) d'un système de fichiers

Éléments à noter :

— La classe `Node` factorise les éléments communs aux fichiers (`File`) et aux répertoires (`Directory`) comme la notion de nom mais elle pourrait contenir d'autres éléments comme les droits en écriture/lecture.
— Un répertoire peut contenir des fichiers et d'autres (sous-)répertoires alors qu'un fichier est un élément ne pouvant contenir aucun autre élément.

Le code Java donné ci-dessous décrit l'implémentation de ces trois classes :

```
1  import java.util.ArrayList;
2
3  abstract class Node
4  {
5
6      private String name;
7
8      public Node(String nom)
9      {
10         this.name = nom;
11     }
12
13     public String getName()
14     {
15         return this.name;
16     }
17
```

```
18      public void explore()
19      {
20          System.out.println(this.getName());
21          if (this.getClass() == Directory.class)
22          {
23              Directory rep = (Directory) this;
24              ArrayList<Node> content = rep.getChildren();
25              for (Node n : content)
26              {
27                  n.explore();
28              }
29          }
30      }
31  }
32
33  class File extends Node
34  {
35
36      public File(String name)
37      {
38          super(name);
39      }
40  }
41
42  class Directory extends Node
43  {
44
45      private ArrayList<Node> children;
46
47      public Directory(String name)
48      {
49          super(name);
50          this.children = new ArrayList<Node>();
51      }
52
53      public void addNode(Node n)
54      {
55          this.children.add(n);
56      }
57
58      public ArrayList<Node> getChildren()
59      {
60          return this.children;
61      }
62  }
63
64  public class TestFileSystem
65  {
66      public static void main(String[] args)
67      {
68          Directory r1 = new Directory("A");
69          Directory r2 = new Directory("B");
70          File f1 = new File("f1");
71          File f2 = new File("f2");
72          r1.addNode(r2);
73          r2.addNode(f2);
74          r1.addNode(f1);
75          r1.explore();
76      }
77  }
```

Travail à réaliser :

— Tout d'abord créer une classe par fichier (bonne habitude à prendre...);

— Créer une classe `FileSystem` représentant un système de fichiers ayant une racine unique de type `Directory` qui sera créée automatique lors de l'instanciation de la classe `FileSystem`;

— Modifier ce code pour retirer le test de la ligne 24 en utilisant la notion de polymorphisme. L'idée générale (expliquée dans le cours) est de permettre à chaque entité (fichier ou répertoire) de posséder des méthodes `getChildren` et `addNode` mais de se comporter de manière différente selon la classe où cette méthode sera implémentée.

— Sur le même modèle (déclaration comme méthode abstraite dans `Node` et implémentation dans les sous-classes) implémenter une méthode `getSize` qui :

  — pour un fichier retournera sa taille (attribut à ajouter à la classe `File`);

— pour un répertoire retournera la taille des éléments qu'il contient (ici on simplifie car dans la réalité un répertoire possède une taille même si celui-ci ne contient aucun élément).

— Mettre en œuvre la notion de parent (qui ne pourra être que de type `Directory`) de telle manière que toute entité, fichier ou répertoire, ait un parent. Le test présent dans la classe de test présentée ci-dessous montre un exemple de ce qu'on souhaite implémenter.

— Pour terminer, question optionnelle, créer une classe `SymbolicLink` représentant un lien symbolique.

La nouvelle version de la classe de test :

```
public class TestFileSystem {
    public static void main(String[] args) {
        Directory dirA = new Directory("A");
        Directory dirB = new Directory("B");
        FileSystem fs = new FileSystem();
        fs.getRoot().addNode(dirA);
        File f1 = new File("f1",120);
        File f2 = new File("f2", 340);
        dirA.addNode(dirB);
        dirB.addNode(f2);
        dirA.addNode(f1);
        if (fs.getSize() != 460) {
            throw new Error("Calcul taille repertoire incorrect");
        }
        if (f1.getParent() != dirA || f2.getParent() != dirB || dirB.getParent() != dirA) {
            throw new Error("Mauvaise gestion du lien parent");
        }
        dirA.explore();
    }
}
```

2 Composants informatiques

On souhaite représenter, sous forme de classes, des composants intervenant comme éléments d'un PC. Nous avons les trois composants suivants :

— le disque dur possédant les informations suivantes : identifiant, prix, consommation, capacité ;

— l'alimentation possédant les informations suivantes : identifiant, prix, puissance ;

— le processeur possédant les informations suivantes : identifiant, prix, consommation, fréquence, type de connecteur (*socket*) ;

— une barrette mémoire possédant les informations suivantes : identifiant, prix, fréquence, consommation.

Travail à réaliser :

1. Mettre en œuvre une hiérarchie de classes ayant comme racine la classe `Composant` regroupant les attributs communs à l'ensemble des classes.
2. Puis créer une classe `PC` qui possédera une liste de composants.
3. En considérant que chaque composant à une méthode `donneEnergie` qui retourne soit une valeur négative si le composant consomme de l'énergie, soit une valeur positive si le composant en fourni (cas de l'alimentation), vérifier lors de la constitution du PC (de préférence avant l'appel du constructeur) que le bilan énergétique des composants est positif (ou nul à la rigueur).
4. Créer une classe `TestAssemblagePC` vérifiant l'assemblage correct et incorrect d'un PC.