Feuille de TP 5 : Ecriture d'un arbre dans un fichier

Le projet *graphviz* (`graphviz.org`) fournit des outils pour visualiser facilement des graphes et des arbres, décrits au moyen d'un langage simple.

L'exemple ci-dessous montre un arbre décrit dans ce langage (à gauche) et l'image produite (à droite).

```
graph untitled
{
        n [label="."];
        ng [label="a"];
        nd [label="b"];

        n -- ng;
        n -- nd;
}
```

Si l'arbre est décrit dans un fichier `arbre.dot`, l'image est produite avec la commande `dot` ci-dessous :

```
dot -T ps -o arbre.ps arbre.dot
```

La commande `dot` est disponible sur les machines des salles de TP (il se peut qu'elle affiche des erreurs pour des problèmes de fontes mais le fichier est tout de même correctement produit). Le format de sortie choisi ici (ps) est un format vectoriel (donc qui permet de zoomer sans altération de l'image). Le format pdf est lui aussi vectoriel et pris en charge par les versions plus récentes mais pas par cette que vous avez en TP (mais vous pouvez l'installer sur votre propre machine). D'autre formats non vectoriels sont aussi disponibles, comme jpg.

Vous trouverez sur Didel, dans *Documents et liens* > *TP* > *Divers* les fichiers dot et pdf de l'arbre pris comme exemple dans l'énoncé du projet.

`Remarque importante` pour la description des arbres, l'ordre des lignes à l'intérieur des accolades n'a pas d'importance, en particulier une arête peut être décrite avant les nœuds qui la composent (ce qui vous simplifiera beaucoup la tâche).

Exercice 1 :

Dans la classe `Arbre`, déclarer une méthode

```
abstract void toDotStream(PrintStream out, String nodeName);
```

et écrire une méthode

```
public void toDotFile(String filename)
```

qui prend en paramètre le nom du fichier à créer et effectue les opérations suivantes :

- crée un `PrintStream` à partir du fichier (voir les constructeurs de cette classe dans la doc Java 6) ;
- écrit le début du fichier (les deux premières lignes de l'exemple) ;
- appelle la méthode `toDotStream( )` en donnant comme deuxième paramètre la chaîne vide `""` ;
- écrit la fin du fichier (l'accolade fermante) ;
- ferme le `PrintStream` ;

– traite l'exception `FileNotFoundException` qui est susceptible d'être levée par la création du `PrintStream` :

```
try {
  /*
    code susceptible de lever une exception
  */
}
catch (FileNotFoundException e)
{
  /*
    code traitant l'exception, par exemple :
   */
  System.err.println("Erreur : " + e.getMessage());
}
```

Exercice 2 :

Créer une méthode `toDotStream` vide dans chacune des trois sous-classes d'Arbre et tester la méthode `toDotFile` dans le `main()` de `Arbre`.

La méthode `toDotStream` devra parcourir l'arbre récursivement, en écrivant les nœuds et les arêtes rencontrées au fur et à mesure. Le deuxième paramètre (`nodeName`) servira à nommer les nœuds dans le fichier dot. Le nom de la racine sera `n`, les fils d'un nœud unaire prendront le nom de leur père suivi de la lettre `f`, ceux d'un nœud binaire prendront le nom de leur père suivi de la lettre `g` ou `d` selon leur position.

Exercice 3 :

Remplir la méthode `toDotStream` de `Feuille`, puis de `Unaire`, puis de `Binaire`, en testant à chaque fois dans le main correspondant avec `System.out` comme premier paramètre.

Exercice 4 :

Tester le tout avec un arbre simple, puis avec une grosse expression (en utilisant la méthode du TP précédent pour construire l'arbre à partir d'une expression postfixe).