Programmation avancée et répartie en Java, TP 1

Frédéric Gava
gava@u-pec.fr

Liens utiles

— Les outils de développement en ligne de commande JDK se trouvent sur
`https://www.oracle.com/java/technologies/downloads/`
— La documentation de référence de l'API se trouve sur le site
`https://docs.oracle.com/en/java/javase/17/docs/api/index.html`

Consignes générales pour les TPs

— Les TPs seront réalisés sous Linux.
— Pour écrire le code source, on utilisera un éditeur tel que gedit, kate, emacs, sublimText ... qui proposent tous la coloration syntaxique. Pour ce module, vous pouvez aussi utiliser un IDE tel que Eclipse ou Netbeans. Plutôt une compilation en ligne de commande !
— Il est recommandé de créer dans son répertoire personnel un dossier qui contiendra tous les TPs. A la racine de ce dossier, on créera un dossier pour les codes sources `src`, et un dossier pour les codes compilés, `build`. Dans le dossier `src`, on utilisera un dossier par TP (`tp1`, `tp2`, *etc.*) Ce dossier correspondra à un *package*, qui contiendra les différentes classes des exercices du TP.
— En suivant l'organisation décrite ci-dessus, la compilation des sources d'une séance de TP peut se faire avec la commande suivante, en se plaçant à la racine de l'arborescence de travail (qui contient les dossier `src` et `build`) :
`javac -d build src/tp1/*.java`
L'exécution d'une classe, par exemple `Exo1`, se fait alors avec la commande suivante :
`java -cp build tp1.Exo1`
— Il est demandé de respecter au maximum les conventions de style Java. En particulier,
  — Les noms de classe doivent commencer par une majuscule (ex : `class MaClasseTresImportante {`)
  — Les noms de variables et de méthodes doivent commencer par une minuscule (ex : variable `int unNombre`, méthode `void faireQuelqueChose()`)
  — Le code doit être *correctement indenté* !

Exercices

Exercice 1 (*Des listes génériques*)

Réécrire le code des listes afin qu'il soit générique.

Exercice 2 (*(facultatif) Des listes circulaires*)

Écrire une classe générique des listes circulaires (voir explication au tableau) qui contriendra les méthodes suivantes :
  — *Lire l'élément en cours*
  — *Se décaler sur l'élément suivant (sur sois-même si la liste ne contient qu'un seul*

élément)
- *Ajout/Suppression d'un élément*
- *Calcul de la longueur de la liste*
- *Recherche de la présence d'un élément dans la liste*

Rappel :

```
sealed interface Expr permits ConstantExpr, PlusExpr, TimesExpr {
    public int eval();
}

final class ConstantExpr implements Expr {
    int i;
    ConstantExpr(int i) { this.i = i; }
    public int eval() { return i; }
}

//Expr e = new PlusExpr(new ConstantExpr(1), new ConstantExpr(2));
//System.out.println(e.eval());
```

Exercice 3 (*Evaluateur*)

Complétez le code de cette mini-calculatrice. Testez. Pour les courageux (à la maison) rajoutez une petite interface graphique.

Exercice 4 (*Avec des bool*)

Rajoutez une conditionnelle (if-then-else) dans les expression de votre mini-calculatrice

Voici une autre manière d'avoir des arbres binaires contenant soit des feuilles avec un entier soit un String :

```
sealed interface Arbre permits BinNode, LeafInt, LeafString {
        public int size() ;
}
```

Chaque composante est doté de la méthode size et qui calcul le nombre de noeuds/feuilles de l'arbre (1 pour une feuille).

Exercice 5 (*Arbre*)

Codez ces composantes de 2 manières différentes. L'une avec des classes. L'autre avec des record. Comparez !

Exercice 6 (*Arbre statique*)

Rajoutez une méthode statique **static int** *size(Arbre) et qui calcul le nombre de noeuds/-feuilles de l'arbre (1 pour une feuille).*

Exercice 7 (*Arbre bis*)

Rajoutez des méthodes pour la hauteur de l'arbre et pour le plus grand entier.

Exercice 8 (*Arbre Ternaire*)

Rajoutez le cas où nous aurions aussi des noeuds ternaires !