

Interface, classe abstraite, table de hachage, generics

Exercice 1 - Arbre d'héritage et expression

Le but de cet exercice est de construire un évaluateur d'expressions arithmétiques simples. Ces expressions sont représentées sous forme d'arbre.

On veut un type commun `fr.uml.v.calc.Expr` représentant des expressions arithmétiques qui peuvent être soit une valeur réelle (de type `fr.uml.v.calc.Value`) soit une opération d'addition (de type `fr.uml.v.calc.Add`) qui permet d'effectuer l'addition de deux expressions.

On veut de plus être capable d'évaluer de ces expressions en utilisant la méthode `eval()`.

```
Expr value=new Value(7.0);
System.out.println(value.eval()); // affiche 7.0
Expr add=new Add(new Value(3.0),value);
System.out.println(add.eval());   // affiche 10.0
Expr expr=new Add(new Value(2.0),add);
System.out.println(expr.eval());  // affiche 12.0
```

- 1 Écrire les trois classes `Expr`, `Value` et `Add` et leurs méthodes `double eval()` ainsi qu'une classe `Main` de test.
- 2 Implanter la méthode `toString()` sur les différentes expressions.
- 3 Discuter sur le fait de transformer la classe `Expr` en classe abstraite ou en interface. Faites les changements qui s'imposent.
- 4 Ajouter les classes `Mult` et `Divide` permettant d'effectuer respectivement la multiplication et la division. Refactoriser le code pour mettre à un seul endroit les codes communs.
- 5 Écrire une classe `Main` et faites des tests.

Exercice 2 - List, LinkedList et ArrayList

Expliquer ce que fait le code suivant :

```
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;

...public static void main(String[] args) {
    List list;
    if (args.length==0)
        list=new ArrayList();
    else
        list=new LinkedList();

    for(String s:args)
        list.add(s);
}
...
```

- 1 Quel différence y a-t'il entre `LinkedList` et `ArrayList` ?
- 2 À quoi sert l'interface `List` ?

Le code précédent compile avec un warning, que doit-on faire pour supprimer celui-ci ?
Ré-écrire le code du `main` en conséquence

De la même façon, transformer le code suivant :

```
public static void main(String[] args) {  
    List list=new ArrayList();  
    Collections.addAll(list,args);  
  
    for(int i=0;i<list.size();i++) {  
        String s=(String)list.get(i);  
        System.out.printf("%s:%d\n",s,s.length());  
    }  
}
```

Quel est l'intérêt des generics en Java ?

Exercice 3 - Table de hachage

On souhaite afficher l'animal préférée de bob, alice ou june.

Alice a pour animal préféré edith le singe.

Bob a pour animal préféré izard le chamoix.

June a pour animal préféré gold le poisson rouge.

Ecrire un programme qui prend en paramètre le nom d'un enfant (parmi bob, alice et june) et affiche son animal préféré.

```
java Animal bob  
L'animal préféré de bob est izard le chamoix
```

Ecrire un programme qui prend en paramètre le nom d'un enfant (parmi bob, alice et june) et affiche son animal préféré.

Ré-écrire le code utilisant une table de hachage `java.util.HashMap` pour éviter les `if ... else`.