Algorithmique avancée
TD n° 8

Récursivité

Exercice 1.

On veut écrire une classe ListeRec utilisant des méthodes récursives. Pour cela les variables membres sont un objet (premier) et une liste de type ListeRec (reste). Par convention, si la liste est vide on met premier = reste = null. Grâce à cela, on peut demander le premier et le reste de la liste vide sans déclencher d'exception.

a. Compléter le code ci-dessous en utilisant des appels récursifs pour écrire les boucles et en commençant par les méthodes nbElem() et toString().

```
public class ListeRec {
    private Object premier ;
    private ListeRec reste ;
    // la liste vide est : premier == reste == null

    public boolean vide()      {
        return (premier==null && reste == null);      }

    public ListeRec() {
        premier = null;   reste = null ;
    }

    private ListeRec(Object prem, ListeRec res) {
         premier = prem ;
         reste = (prem != null && res == null) ? new ListeRec(): res;
    }

    public Object getPremier() {
        return(premier) ;
    }

    private ListeRec getPhysicalReste(){
         // Prive car peut renvoyer null
        return(reste) ;
    }

    public ListeRec getReste(){
    /* Quand la liste est vide, son reste est une liste vide ==> On peut encore appeler
getReste() sans provoquer une exception */
        return(reste == null ? new ListeRec() : reste) ;
    }

    public void rajouterEnTete(Object item) {

    }
```

```
    public void rajouterEnQueue(Object item) {

    }

    public Object retirerEnTete() throws ListeVideException      {

      }

    public Object retirerEnQueue() throws ListeVideException{

      }

    public int nbElem()      {

      }

    public String toString()    {

    }
// * ******************
    public class ListeVideException extends Exception {
        ListeVideException(String s) {
             super(s);
        }
    }
}
```

On utilisera la classe TestRec fournie pour tester le code.

b. Dans les deux fonctions testeAjout…() contenues dans TestRec, changez l'argument (taille de la liste) pour 10000. Expliquez les résultats.

c. ajoutez dans ListeRec un constructeur public qui crée une liste à partir d'un tableau.

Exercice 2.

On veut créer un vérificateur orthographique utilisant un dictionnaire implémenté dans un arbre binaire : chaque nœud contient un mot, tous les nœuds atteints par son fils gauche sont avant dans l'ordre alphabétique, tous les nœuds atteints par son fils droit sont après. La classe DicoNoeud contient un mot, un fils gauche et un fils droit ; la classe DicoArbre contient un DicoNoeud (la racine de l'arbre). Répondre aux questions ci-dessous en donnant les algorithmes récursifs puis en complétant le code fourni :

1. Ecrire une méthode **récursive** qui donne le premier mot du dictionnaire (dans l'ordre alphabétique)
2. Ecrire une méthode **récursive** de recherche dichotomique dans le dictionnaire : boolean contains(String mot) indique si le mot est dans le dictionnaire (son orthographe est correcte) ou pas.
3. Ecrire une méthode void add(String mot) d'insertion **récursive** pour ajouter des mots dans le dictionnaire
4. Quel est le problème de la suppression ? Proposez un algorithme qui le résolve, puis écrivez la méthode

Testez le code écrit à l'aide de la classe fournie TestDychoMap

```
import java.io.*;
public class DicoNoeud {
    String mot ;
    DicoNoeud filsg ;
    DicoNoeud filsd ;

    public DicoNoeud(String mot) {
        this.mot = mot ;
    }

    public String getMot() {
        return mot ;
    }
    public DicoNoeud getFilsg() {
        return filsg ;
    }
    public DicoNoeud getFilsd() {
        return filsd ;
    }
    public void setFilsg(DicoNoeud n) {
        filsg = n ;
    }
    public void setFilsd(DicoNoeud n) {
        filsd = n ;
    }

    public void sauver(ObjectOutputStream out) throws IOException {
        out.writeObject(mot);
        out.writeObject(filsg);
        out.writeObject(filsd);
    }


    public void charger(ObjectInputStream inp) throws IOException,
ClassNotFoundException {
        mot = (String)inp.readObject();
        filsg = (DicoNoeud)inp.readObject();
        filsd = (DicoNoeud)inp.readObject();
    }
}
```

```
import java.io.*;
public class DicoArbre {
    DicoNoeud racine ;

    public DicoArbre() {

    }
    public DicoArbre (String nomFich)
            throws FileNotFoundException, IOException, ClassNotFoundException {
        charger(nomFich) ;
```

```
    }

    public void sauver(String nomFich) throws FileNotFoundException, IOException {
        FileOutputStream fich = new FileOutputStream(nomFich);
        ObjectOutputStream out = new ObjectOutputStream(fich) ;
        racine.sauver(out) ;
        out.close();
        fich.close();
    }

    public void charger(String nomFich)
                throws FileNotFoundException, IOException,ClassNotFoundException {
        FileInputStream fich=new FileInputStream(nomFich);
        ObjectInputStream inp = new ObjectInputStream(fich) ;
        racine.charger(inp) ;
        inp.close();
        fich.close();
    }

    public String premier() throws ArbreVideException {
        if(racine == null) throw new ArbreVideException("Erreur : premier mot d'un
dictionnaire vide ") ;
        return premier(racine).getMot() ;
    }// fin de premier
    private DicoNoeud premier(DicoNoeud dn){
                                    /* A COMPLETER */
    }// fin de premier

    public boolean contains(String mot) {
                                    /* A COMPLETER */
    }// fin de contains

    private boolean contains(String mot, DicoNoeud dn) {
                                    /* A COMPLETER */
    }// fin de contains

    public void add(String mot) {
                                    /* A COMPLETER */
    }// fin de add
    public void add(String mot, DicoNoeud dn) {
                                    /* A COMPLETER */
    }// fin de add

    class ArbreVideException extends Exception {
        ArbreVideException(String s) {
            super(s);
        }
    }
}// fin de DicoArbre
```