Bases de la POO / Java

Les listes chaînées

LINKEDLIST
ARRAYLIST

Présentation de la classe

- **Agrégat ordonné d'objets quelconques**
- **Chaque élément est obligatoirement un objet**
- **Type unique recommandé**
- **Chaînage avant et arrière**
- **Accès direct à la tête et à la queue de la liste**
- **Itérateurs bidirectionnels**
- **Constructions récurrentes valides**

La hiérarchie d'héritage en Java

java.util
Class LinkedList

java.lang.Object
  └ java.util.AbstractCollection
    └ java.util.AbstractList
      └ java.util.AbstractSequentialList
        └ java.util.LinkedList

All Implemented Interfaces:
Cloneable, Collection, List, Serializable

1.4.2

Version générique (Cf. cours suivant)

java.lang.Object
  └ java.util.AbstractCollection<E>
    └ java.util.AbstractList<E>
      └ java.util.AbstractSequentialList<E>
        └ java.util.LinkedList<E>

5.0

Principaux services

Taille courante d'une liste (**size**)
Accesseurs de consultation (**get**, **getFirst**, ...)
Accesseurs de modification (**set**)
Accesseurs de position (**indexOf**)
Ajouter un nouvel élément (**add**)
Retirer un élément (**remove**)
Vider une liste (**clear**)
Contrôle d'appartenance (**contains**)
Transférer dans un tableau (**toArray**)

Créer et initialiser une liste


```
import java.util.*;
public class Exemple_1 {
    public void main (String[] args) {
    LinkedList notes= new LinkedList ();

        notes.add(new Integer(17)) ;
        notes.add(new Integer(11)) ;
        notes.add(new Integer(6)) ;
        ---
```

Visualiser une liste


```
import java.util.*;
public class Exemple_1 {
   public void main (String[] args) {
   LinkedList notes= new LinkedList ();

      notes.add(new Integer(17)) ;
      notes.add(new Integer(14)) ;
      notes.add(new Integer(6)) ;

      System.out.println ("Notes= " + notes);
   }
}
```

➔ **Exécution**

```
Notes= [17, 14, 6]
```

Parcourir une liste de réels

```
public static float moyenne (final LinkedList notes) {
float somme=0.0f;

    Iterator i= notes.iterator();
    while (i.hasNext())
        somme += ( (float) i.next() );
     return somme/notes.size();
}
```

Dupliquer une liste (1)


```
import java.util.*;
public class Exemple_4_1 {
   public void main (String[] args) {
   LinkedList source= new LinkedList ();
   LinkedList copie;

      source.add("jaune") ;
      source.add("bleu") ;
      source.add("vert") ;

      copie= (LinkedList)source.clone();
      ---
```

➔ **Exécution**

```
Source = [bleu, jaune, vert]
Copie = [bleu, jaune, vert]
```

Dupliquer une liste (2-1)


```
import java.util.*;
public class Exemple_4_2 {
   public void main (String[] args) {
   LinkedList source= new LinkedList ();
   LinkedList copie;

   Ville w1= new Ville("C",  500000);

      source.add(new Ville("A", 200000) ;
      source.add(new Ville("B", 350000) ;
      source.add(w1) ;
      source.add(new Ville("D", 40000)) ;
```

Dupliquer une liste (2-2)


```
        ---

        copie= (LinkedList )source.clone();

        w1.setPopulation (600000);

        System.out.println ("Source = " + source);

        System.out.println ("Copie =  " + copie);
    }
}
```

➔ **Exécution**

```
Source= [A/200000, B/350000, C/600000, D/40000]
Copie=  [A/200000, B/350000, C/600000, D/40000]
```