Exercice 14

ListeSort : une deuxième implémentation plus aboutie

Objectif : Implémentation d'une classe ListeSort qui maintient une liste ordonnée croissante des éléments grâce à un arbre binaire infixé et dont l'itérateur des éléments utilise un double chainage. De plus on veut être générique sur le type d'élément

1. Cahier des charges

Nous voulons aller plus loin dans l'implémentation de la liste ordonnée de l'exercice 13 précédent :

- être générique sur le type d'élément de la liste.
- créer un iterator plus optimisé que dans l'exercice 13 : l'iterateur utilise un chainage des éléments au lieu de transformer l'arbre en un ArrayList afin de retourner son iterator.

Rappel de l'exercice 13 :

Nous voulons implémenter une liste de type <T> dont les éléments sont toujours ordonnés croissants et dont le moyen de stockage est un arbre binaire afin d'optimiser la recherche d'un élément.

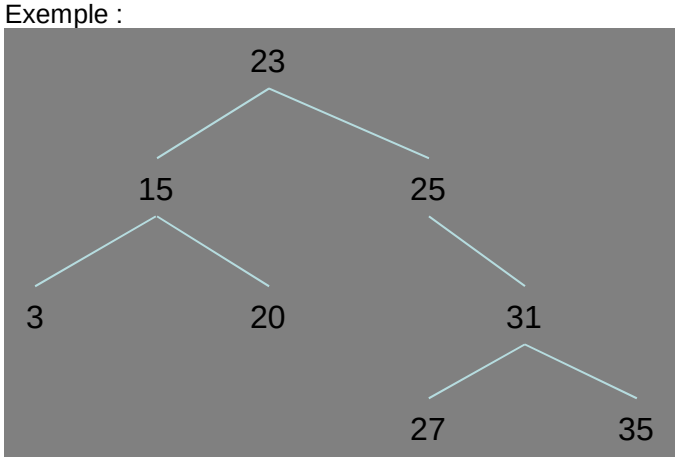
Le principe est le suivant les éléments sont stockés dans un arbre binaire dont la racine est le premier élément ajouté dans la liste.

Ensuite les éléments sont ajoutés à gauche d'un nœud si l'élément ajouté est inférieur à la valeur du nœud sinon à droite.

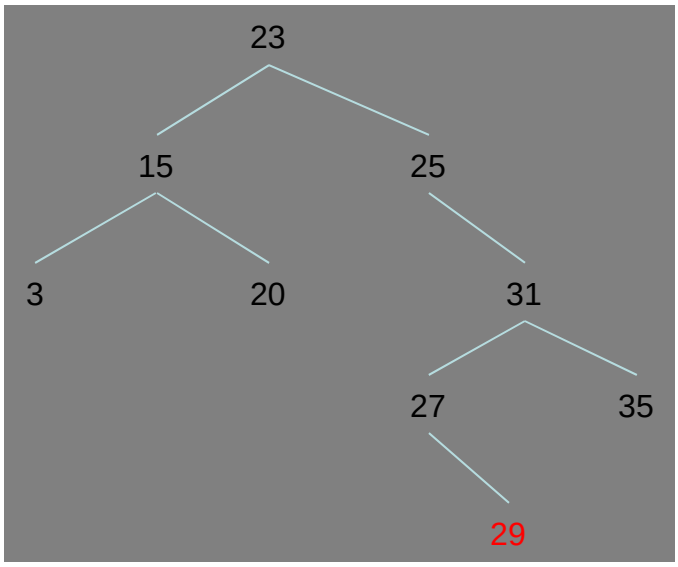
On s'aperçoit ainsi que si on parcourt l'arbre suivant un parcours infixé les éléments sont ordonnés croissants.

De plus cela permet de rechercher un élément très rapidement. Sur un arbre équilibré de 1024 feuilles, on retrouve un élément en au plus 10 tests.

Exemple :

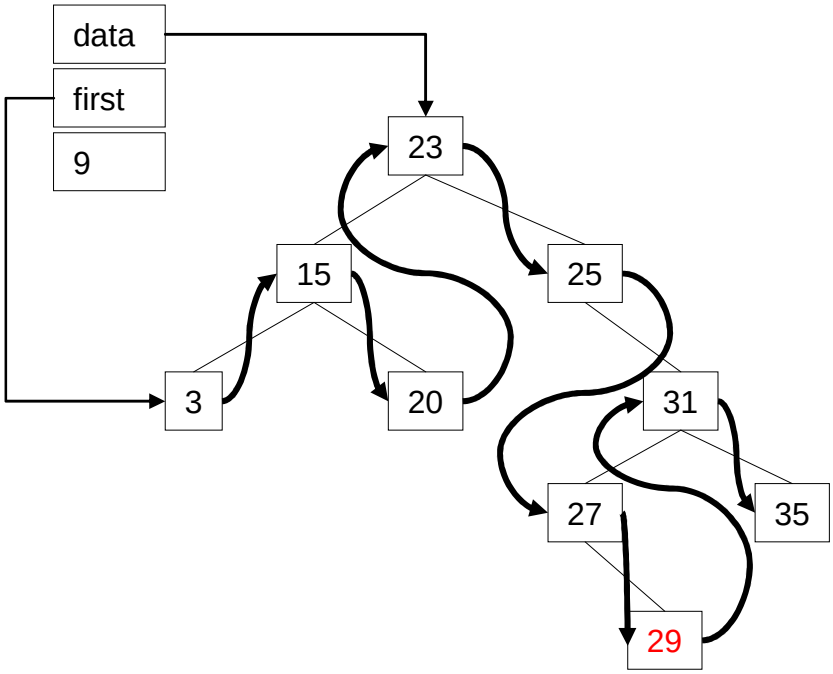


Ajouter l'élément 29 :



Nouveauté :

Les éléments sont chainés entre eux dans l'ordre croissant (afin de ne pas surcharge le schéma, n'est pas représenté le double-chainage).



La classe ListeSort devient :

```
  public class ListeSort<T extends Comparable<T>> implements Iterable<T>,
Iterator<T>
```

Le programme principal permet de tester la classe.