**Université de BATNA**
**Faculté MI**
**Département d'Informatique**

**Filière : Master 1 IRC**
**Module : Technologie Objet**

# Travaux Dirigés N°2

**Exercice 1 :**

Le design pattern Iterator fourni un mécanisme d'accès à un objet agrégat (collection), indépendamment de son implémentation. L'API JAVA fournit un certains nombre de collections, contenues dans le paquetage java.util. On distingue deux types de collections : Simples et Associatives (maps, hashtables). Les collections simples sont définies à partir de la racine interface Collection <E> qui étend (extends) l'interface Iterable <E>, par contres les collections associatives à partir de la racine interface Map <K,V>.

1- Présenter une vue d'ensemble simplifiée de la hiérarchie des collections Java
2- Présenter l'interface Iterator et ListIterator
3- Faite correspondre l'interface Iterator avec celle du GOF
4- Considérons une collection d'entiers à parcourir en utilisant un Iterator :
   a- Présenter brièvement les quatre classes collection : ArrayList, LinkedList, HashSet, TreeSet
   b- Ecrire le programme JAVA permettant de créer et la collection.

**Exercice 2 :**

Un feu tricolore se compose de trois couleurs : Rouge, Orange, Vert. Itérer sur le feu tricolore consiste à parcourir sans fin les trois couleurs. Ecrire une classe JAVA pour représenter les feux tricolores, un itérateur de feu tricolore et un programme montrant une utilisation.

**Exercice 3 :**

Une Pizzeria offre à ses clients trois types de Pizza : Pizza au fromage, Pizza au fruit de Mer, Pizza végétarienne. La préparation du Pizza peut comprendre quatre opérations :

- Préparer
- Cuire
- Couper
- Emballer

Il est clair que le Factory Pattern peut être adopté pour modéliser la Pizzeria : Fabrique de Pizza. Par ailleurs, on veut écrire une implémentation en JAVA de la fabrique de Pizza en appliquant le Pattern Factory.

1- Conformément au modèle du Factory du GoF, illustrez schématiquement le Pattern de la fabrique de Pizza
2- Ecrire le code JAVA relatif à chaque élément du Pattern
3- En général, la manière de préparer le Pizza dépend des spécificités géographiques et régionales. Le même type de Pizza, dans deux pays différents, peut être préparé différemment. Ainsi, par exemple le Pizza de fruit de mer algérienne diffère de celle de l'Italie. Réécrire le code JAVA correspondant à cette différence.

```
Public interface Iterator <E> {
        T next()
        boolean hasNext ( ) ;
        v o i d remove ( ) ;
}
```

Methods of Iterator:

- hasNext()
- next()
- remove()

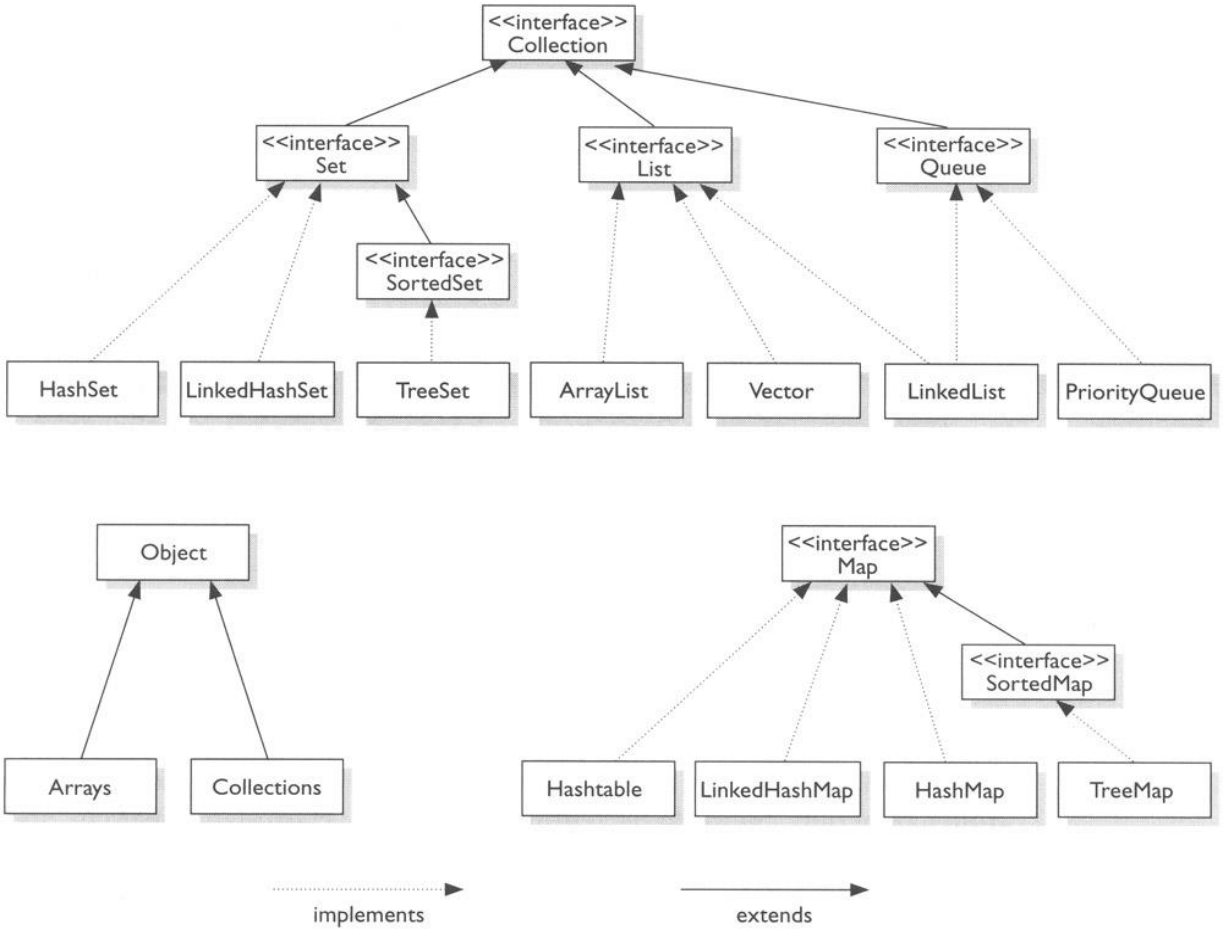
Methods of ListIterator:

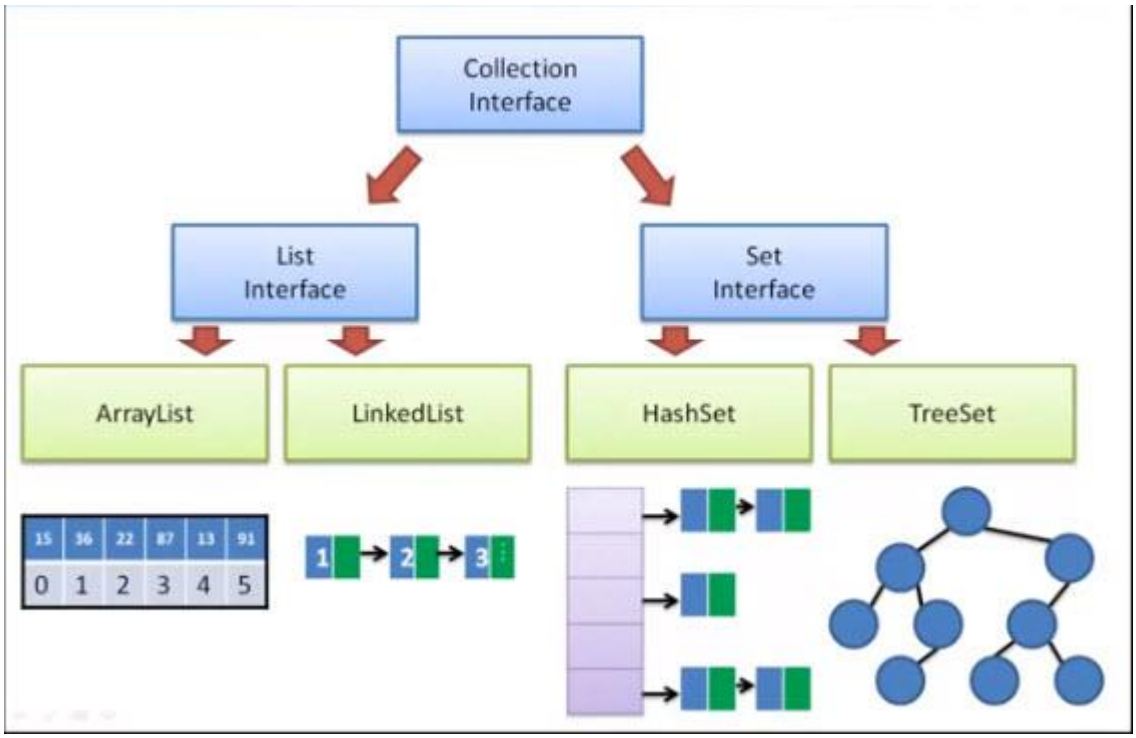
- add(E e)
- hasNext()
- hasPrevious()
- next()
- nextIndex()
- previous()
- previousIndex()
- remove()
- set(E e)

| *Java* | *Patron du GOF* |
|---|---|
| l'itérateur est positionné par défaut au début | First() |
| next() | CurrentItem() et Next() |
| hasNext() | isDone() |
| remove() | pas d'équivalent |

```
import java.util.Iterator;

import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Set;
import java.util.HashSet;
import java.util.TreeSet;

public class IterTest {
   public static void main(String[] args) {
      List <Integer> lis1 = new ArrayList<Integer>();
      List <Integer> lis2 = new LinkedList<Integer>();
      Set <Integer> set1 = new HashSet<Integer>();
      Set <Integer> set2 = new TreeSet<Integer>();

      for (int i=0; i<6; i++)
         lis1.add(i);
```

```
    for (int i=0; i<6; i++)
       lis2.add(i);

    for (int i=0; i<6; i++)
       set1.add(i);

    for (int i=0; i<6; i++)
       set2.add(i);

    Iterator <Integer> itl1 = lis1.iterator();
    Iterator <Integer> itl2 = lis2.iterator();
    Iterator <Integer> its1 = set1.iterator();
    Iterator <Integer> its2 = set2.iterator();

    System.out.println("ArrayList");
    while(itl1.hasNext()){
       System.out.print(itl1.next() + "  ");}

    System.out.println("\nLinkedList");
    while(itl2.hasNext()){
       System.out.print(itl2.next() + "  ");}

    System.out.println("\nHashSet");
    while(its1.hasNext()){
       System.out.print(its1.next() + "  ");}

    System.out.println("\nTreeSet");
    while(its2.hasNext()){
       System.out.print(its2.next() + "  ");}

    System.out.println("\n");
    }
}
```


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
<<interface>>
SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
Arrays
Collections
<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends


Collection
Interface
List
Interface
Set
Interface
ArrayList
LinkedList
HashSet
TreeSet
15
36
22
87
13
91
0
1
2
3
4
5
1
2
3