TD/TP

ITERATOR, IMPLEMENTATION D'INTERFACES

UNIVERSITÉ CAEN NORMANDIE

Un Iterator est un objet que peut délivrer une structure de données (ArrayList, Set, Vector, etc.) afin de proposer un parcours de tous les éléments qu'elle contient.

L'idée est de proposer une façon générique de parcourir l'ensemble de n'importe quelle structure. Par exemple, le code ci-dessous est spécifique aux ArrayList :

```
public void afficheElements(ArrayList<?> liste)
    {
    for (int i=0; i<liste.size();i++)
                    System.out.println(liste.get(i));
    }
```

En revanche, le code suivant peut parcourir n'importe quelle collection, pourvu que cette dernière soit capable de fournir un Iterator sur ses données :

```
public static void afficheElements(Iterator<?> iterateur)
    {
    while (iterateur.hasNext())
            System.out.println(iterateur.next());
    }
```

Par exemple, si l'on dispose de ArrayList<String> al = new ArrayList<String>(), on peut lancer la méthode afficheElements(al.iterator()).

Pour ce faire, Iterator<E> déclare trois méthodes :

+ hasNext() : boolean // renvoie true tant qu'il y a des éléments à voir
+ next() : E // renvoie une référence vers un objet non encore vu
+ remove() : void // retire de la structure le denier élément renvoyé par next()

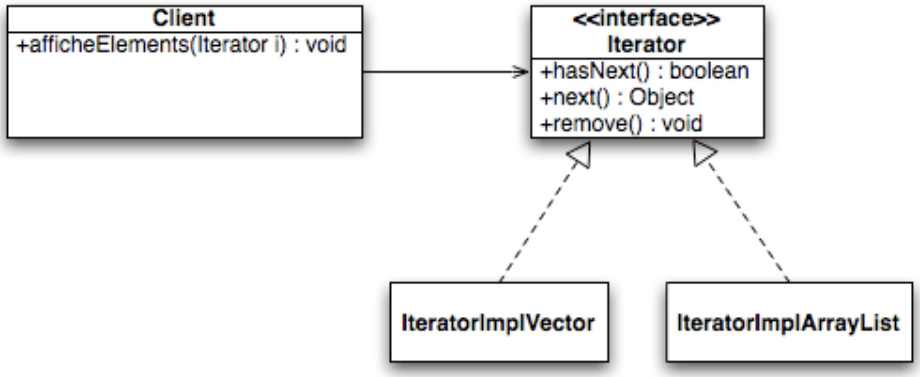
Remarque :

La méthode remove() permet de retirer de la collection observée le dernier élément visité par l'itération. Si une implémentation ne prévoit pas une telle implémentation, elle doit lever une UnsupportedOperationException par le morceau de code suivant :

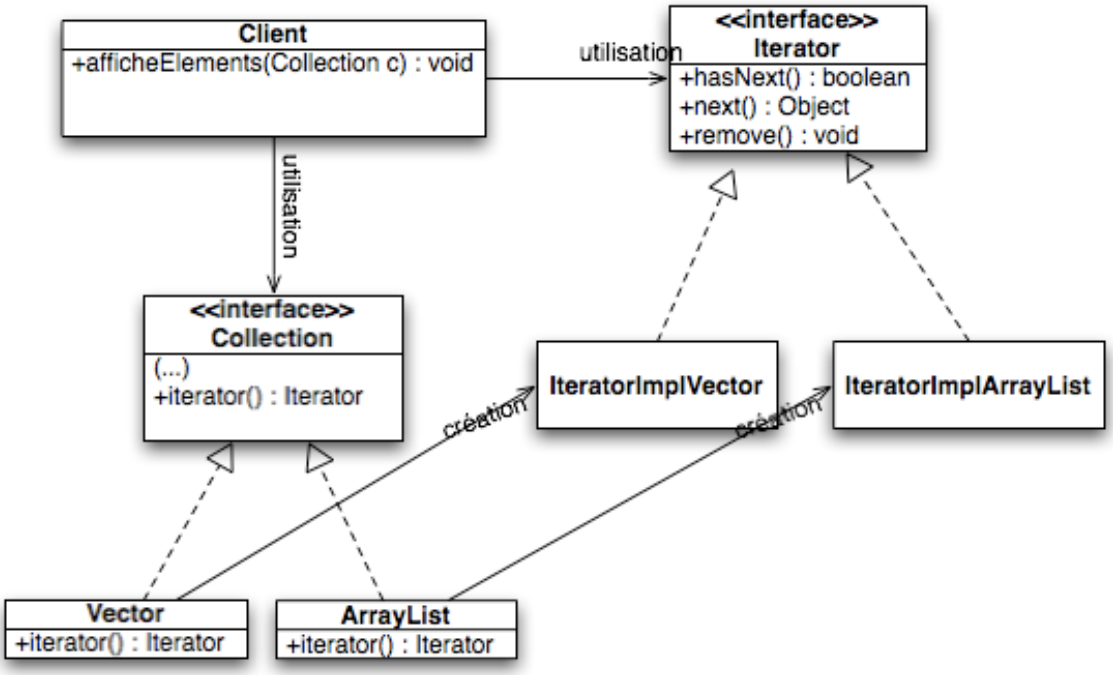
```
throw new UnsupportedOperationException("opération non implémentée");
```

Si elle est implémentée mais invoquée de façon non conforme (sans que next() n'ait jamais été invoquée ou si elle est invoquée deux fois de suite sans appel à next() entre-temps), on lèvera une IllegalStateException.

Le diagramme de classes simplifié est le suivant :



De façon plus pratique, l'interface Collection est le super-type d'un grand nombre de structures de données en Java, et prévoit que chacune de ces dernières fournisse son Iterator. Le diagramme complet est donc :



1. Classe listant un contenu via un itérateur

Créer la classe AfficheurIterator qui propose la méthode statique suivante :

public static void afficheElements(Iterator<?> i)

Voir le code ci-dessus.

Testez-la en lui passant un Iterator produit par un ArrayList que l'on aura rempli avec quelques objets (String par ex.) :

```
ArrayList<String> al=new ArrayList<>();
al.add("A");
al.add("B");
al.add("C");
AfficheurIterator.afficheElements(al.iterator());
```

2. Création d'Iterators "maison" de ArrayList

Le code sera testé au fur et à mesure grâce à la classe de l'exercice 1. L'idée est ici de réinventer soi même l'Iterator de la classe ArrayList. La première version (exercice 1) consiste à faire exactement ce que fait l'itérateur existant d'ArrayList. La seconde version (exercice 3) consiste à faire une version qui propose un parcours à l'envers.

1. MyArrayListIterator

Créer la classe MyArrayListIterator<E> qui implémente Iterator<E>, prend un ArrayList<E> en argument de constructeur, et se comporte, bien sûr, comme itérateur sur cet arrayList<E>.

2. MyArrayList

Créer la classe MyArrayList<E> qui hérite de ArrayList<E> et qui en redéfinit la méthode iterator() de sorte à fournir une instance de notre propre MyArrayListIterator<E>.

3. MyArrayListReverseIterator

Cet itérateur restitue les éléments dans l'ordre inverse. Créer conjointement MyArrayList2

3. Création d'un Iterator sur votre classe ListeChainee

Enrichissez votre classe ListeChainee ou ListeChainee<E> de sorte qu'elle renvoie un iterator permettant de parcourir son contenu.