Design Patterns : Itérateur 

`https://tuleap-campus.org/plugins/git/m1miage2017/Iterator.git`

Master 1 MIAGE

Buts du TD :

- Présentation des itérateurs Java - comment les utiliser ?
- Rappels de notions UML (interfaces, classes, implémentation, généralisation, composition, visibilité) et liens avec langage Java
- Comment implémenter un itérateur spécifique ?

Partie 1 : interface `Iterator` et utilisation

En java, on dispose de l'interface `Iterator` : cette interface permet de parcourir les éléments stockés dans une collection générique (par exemple, List ou Set). Définition de l'interface Iterator<E> :

Boolean	hasNext()	Returns true if the iteration has more elements.
E	next()	Returns the next element in the iteration.
Void	remove()	Removes from the underlying collection the last element returned by the iterator (optional operation).

L'interface `Iterable` possède une méthode `iterator()` qui permet de renvoyer un objet qui implémente cette interface :

```
// Instanciates the list of Strings
List<String> myList = new List<String>(…);

// Gets the iterator
Iterator<String> myIterator = myList.iterator();
while(myIterator.hasNext()) {
   String s = myIterator.next();
}
```

Cette méthode `iterator()` est présente dans l'interface `Iterable` dont héritent généralement les collections standards du langage (interface `List`, classes `LinkedList`, etc.).

1- Représenter sur un diagramme de classe UML les interfaces `Iterator` et `Iterable` et les relations qui existent entre chacune de ces interfaces.
2- Positionner sur ce diagramme les classes concrètes : `List` et `ListIterator`. Bien définir les relations.

3- Ecrire un diagramme de séquence qui illustre le fonctionnement de notre itérateur en conformité avec le diagramme de classes obtenu précédemment.
4- Compléter l'implémentation des classes `List` et `ListIterator` en utilisant un tableau pour stocker les valeurs de la liste (//TODO). Soyez en conformité avec les tests unitaires.
5- Regarder la méthode statique `PrintElems` qui prend en paramètre un itérateur de String et qui permet d'écrire l'ensemble des éléments de la liste. Quel est l'intérêt de cette méthode ?

Partie 2 : implémentation de l'interface Iterator

On veut réaliser la classe MultiList qui permet le stockage d'objets dans une liste de tableaux de taille fixe, comme par exemple :

```
public class MultiList<T> implements Iterable<T>{

     List<List<T>>  multiList ;
     /* ???? */

}
```

1- Proposer une implémentation de l'interface `Iterator` qui permette d'itérer sur tous les éléments gérés par un objet de classe `MultiList` qui soit utilisable par la méthode `PrintElems` de la partie 1.

2- Quel est l'intérêt de la classe `Iterator` ?

3- Pour ceux qui ont le temps :

   a. Proposer une modélisation complète UML de la classe `MultiList` (attention aux relations avec les autres classes).
   b. Proposer une implémentation complète de la classe `MultiList`.

Rappel : Conception à adapter