La première partie du TD concerne l'écriture des méthodes de bases d'insertion, de suppression et de parcours de listes chaînées. La deuxième partie est consacrée aux itérateurs. La troisième partie précise des points sur les méthodes récursives. Vous utiliserez la syntaxe Java.

1 Les listes chaînées : représentation sous forme de maillons

1.1 Qu'est-ce qu'une liste chaînée ?

Une liste chaînée correspond à un assemblage d'éléments reliés entre eux pour former une chaîne. Dans sa version simple, chacun des éléments contient une partie qui correspond aux données que l'on souhaite enregistrer dans la liste et une référence vers l'élément suivant de la liste. Ainsi, contrairement à un tableau qui correspond à des cases contiguës en mémoire, il n'y a pas de localité spatiale entre les éléments. On passe d'un élément à un autre grâce à des références. Ces éléments peuvent ainsi se trouver à des endroits très différents de la mémoire.

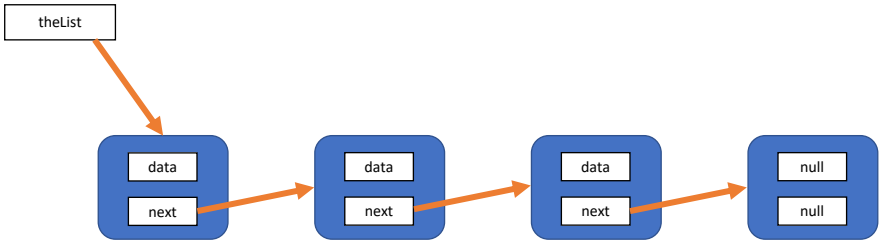


FIGURE 1

1.2 Conception orientée objet

Q 1. En vous inspirant des explications du cours, proposez une modélisation objet d'une liste chaînée. Nous considérerons qu'une liste peut-être représentée par un élément suivi d'une liste, éventuellement vide. Les éléments de la listes représentent chacun une chaîne de caractères. Écrivez les constructeurs associés.
Q 2. Modifiez vos classes de manière à obtenir une liste générique.
Q 3.Écrivez une méthode `public boolean estVide()` qui indique si la liste est vide ou non.
Q 4.Écrivez une méthode `public ListeChainee<E> ajoute(E valeur)` qui ajoute l'objet valeur en tête de liste et retourne le résultat. Comment faire pour faire cette opération sur la liste initiale ? Cela revient à implémenter la méthode `public void ajoute(E valeur)`
Q 5.Écrivez une méthode `public E supprimePremier()` qui supprime l'objet en tête de liste et le retourne.
Q 6.Écrivez une méthode `public String toString()` qui retourne la liste chaînée sous forme d'une chaine de caratères, en respectant la convention utilisée pour la classe `Collection` de Java, *i.e.* entre crochets et la virgule comme séparateur de valeurs.

2 Les itérateurs

2.1 Qu'est-ce qu'un itérateur ?

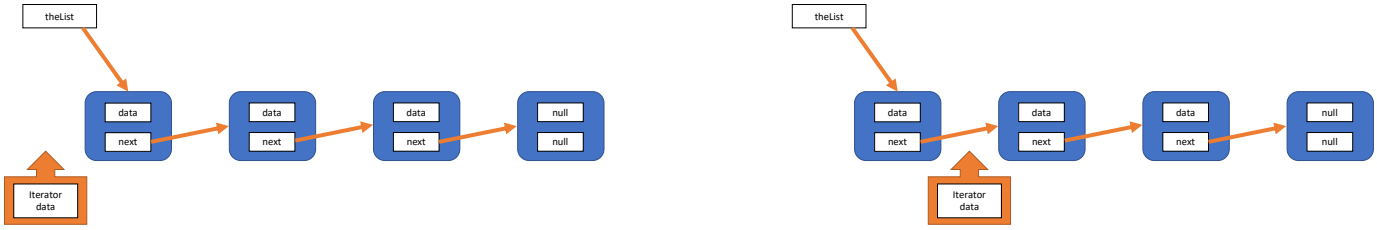
Un itérateur est un *curseur* que l'on peut déplacer sur les maillons d'une liste. L'itérateur peut réaliser les opérations suivantes :

- Se déplacer d'un élément vers la droite via la méthode `next()`
- Vérifier s'il est possible de se déplacer d'un élément vers la droite via la méthode `hasNext()`.

Remarque 1 : Comme indiqué en Figure 2a, à l'état initial l'itérateur est positionné avant le premier élément de liste. Un premier appel à `next()` est alors nécessaire pour que l'itérateur pointe sur le premier objet de la liste.

Remarque 2 : Il est tout à fait possible de déclarer plusieurs itérateurs sur une même liste.

Q 1.Écrivez une classe `ListeChaineeIterator<T>` qui implémente `Iterator<T>` et permettra d'itérer sur la liste déjà implémentée.
Q 2.Écrivez la méthode `boolean hasNext()` qui retourne vrai s'il est possible de déplacer le curseur à droite.



(a) Position initiale de l'itérateur

(b) Position de l'itérateur après un appel à `next()`

FIGURE 2 – Un itérateur

Q 3. Écrivez la méthode `E next()` qui déplace le curseur vers la la droite de la liste et retourne l'élément "pointé" par le curseur. La méthode soulève une exception de type `NoSuchElementException` s'il n'y a pas d"élément suivant.

Q 4. Rendez maintenant votre classe de liste chainée `Iterable<T>` et implémentez la méthode `Iterator<T> iterator()` qui retourne un itérateur associé à la liste.

3 Quelques algorithmes récursifs

Q 1. Écrivez une méthode récursive `boolean contains(Object o)` qui retourne vrai si la liste contient l'objet `o`, faux autrement.
Q 2. Écrivez une méthode récursive `int size()` qui retourne la taille de la liste.
Q 3. Votre méthode est-elle récursive terminale ? Sinon proposez-en une.
Q 4. Écrivez une méthode récursive `void afficheEnvers()` qui affiche la liste à l'envers, un élément par ligne.
Q 5. Le miroir d'une liste est la liste contenant les mêmes éléments mais dans l'ordre inverse. Écrivez une méthode `miroir`. Proposez une implémentation récursive et une implémentation itérative. Discutez les différences.