Liste SD

☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺

Un grand pas vers une implémentation d'une liste très peu « coûteuse » !

☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺☺

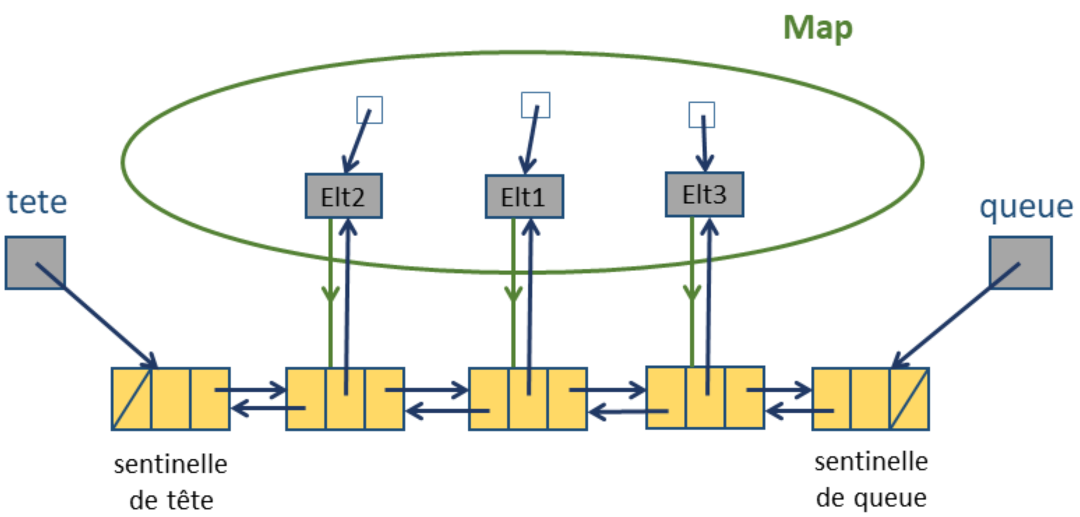
Exercices obligatoires

A Implémentation de l'interface *ListeSD*

A1 Sur moodle, faites le codeRunner *ListeSD*.

Celui-ci va vous proposer d'écrire quelques méthodes de l'interface.

Voici l'implémentation choisie.

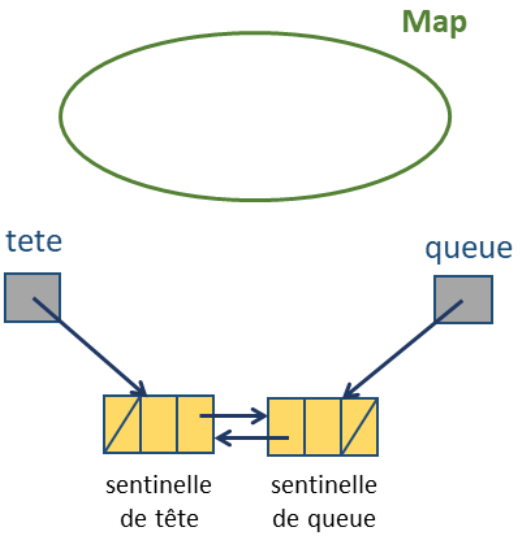


Cette implémentation utilise une liste doublement chaînée avec sentinelles et un map

Une amélioration bien connue de l'implémentation de liste est l'ajout de faux éléments.
L'ajout de ces faux éléments évite de devoir tester de nombreux cas particuliers.
Dans le cas d'une liste doublement chaînée, on ajoute 2 faux éléments. Ces faux éléments sont placés en tête et en queue de liste dans des nœuds appelés « **sentinelle** » ou encore « bidon ».
Les éléments contenus dans ces nœuds n'appartiennent pas à la liste.

De nombreuses méthodes doivent accéder au nœud contenant l'élément passé en paramètre.
Pour récupérer ce nœud, on pourrait parcourir la liste. Mais un parcours de liste se fait en O(N) !
L'utilisation d'un *map* permet la recherche d'un nœud en O(1).
La clé est l'élément recherché, la valeur associée est le nœud contenant cet élément.

Au départ, la liste est vide, mais contient 2 nœuds sentinelles :



Remarque sur le codeRunner :
Si la méthode modifie la liste, les tests du codeRunner vérifient le nombre d'éléments de la liste, le contenu de la liste en suivant le chaînage direct (tête → queue) et celui obtenu en suivant le chaînage inverse (queue → tête).
Aucune des méthodes demandées ne parcourent la liste, mais les tests le font.
Si vous avez mal prévu le chaînage, il peut y avoir une boucle infinie lors du test.
Le but de parcourir la liste dans les 2 sens est de vérifier si tous les raccords ont été bien faits.
Exemple :

	Test	Expected	Got
✖	String[] tableAcopier = {"pierre","paul","jean"}; ListeSDImpl<String> l1 = new ListeSDImpl<String>(tableAcopier); l1.insererEnTete("marie"); System.out.print(l1.taille());\t System.out.print(l1.teteQueue()); System.out.print(l1.queueTete());	4(marie,pierre,paul,jean)(jean,paul,pierre,marie)	3boucle infinie(jean,paul,pierre)

La liste testée est : (pierre,paul,jean)
Après avoir inséré en tête marie :
La liste devrait contenir 4 noms : (marie,pierre,paul,jean)
Cette même liste en sens inverse est : (jean,paul,pierre, marie).
La liste obtenue ne contient que 3 noms. Marie n'a pas été ajoutée. Lors du parcours tête → queue, il y a eu une boucle infinie !

A2 Complétez la classe *ListeSDImpl*.

A3 Sur moodle, répondez au questionnaire à choix multiples *ListeSDImpl*.

B Rallye Automobile

Vous allez développer une application pour la gestion des positions des pilotes lors d'un rallye automobile.

Voici le menu attendu :

```
1 -> Afficher toute la course
2 -> Afficher le pilote en tête
3 -> Enregistrer un dépassement
4 -> Retirer un pilote de la course
5 -> Donner la position d'un pilote (encore dans la course)
6 -> Faire franchir la ligne d'arrivée au pilote de tête
```

Pour cela, vous avez besoin d'une liste ordonnée de tous les pilotes. Le pilote en tête est en tête de liste, et le dernier en queue.

B1 Ecrivez une classe *RallyeAutomobile*.
Cette classe aura comme attribut un objet de la classe *ListeSDImpl*
Vous pouvez ajouter d'autres attributs.
Pour simplifier l'exercice (et les tests !), n'introduisez pas de classe *Pilote*. Le pilote sera représenté par un String.
C'est cette classe qui va présenter les méthodes :

```
String donnerPiloteEnTete()
boolean supprimer(String pilote)
…
```

Elle possédera le constructeur :
`RallyeAutomobile(String[] lesPilotes)`
La table passée en paramètre contient les pilotes enregistrés selon l'ordre de départ.

B2 Ecrivez une classe *GestionRallyeAutomobile*.
Cette classe va permettre dans un premier temps d'enregistrer les pilotes selon l'ordre de départ.
Ensuite cette classe va présenter le menu ci-dessus.
Chaque fois qu'un pilote franchit la ligne d'arrivée, il est retiré.
Le programme s'arrête lorsqu'il n'y a plus de pilotes sur le parcours.
C'est dans cette classe (et uniquement dans celle-ci) que se font les interactions avec un utilisateur.
Dans un premier temps les affichages peuvent être très simples.

Voici un exemple de ce que pourrait être l'affichage à la suite d'une exécution de votre programme :

```
*********************************************
Programme de gestion d'un Rallye Automobile
*********************************************

Entrez le nombre de pilotes : 3
Entrez le nom du pilote 1 : pilote1
Entrez le nom du pilote 2 : pilote2
Entrez le nom du pilote 3 : pilote3


1 -> Afficher toute la course
2 -> Afficher le pilote en tête
3 -> Enregistrer un dépassement
4 -> Retirer un pilote de la course
5 -> Donner la position d'un pilote
6 -> Faire franchir la ligne d'arrivée au pilote de tête


Votre choix : 1
 pilote1 pilote2 pilote3

1 -> Afficher toute la course
2 -> Afficher le pilote en tête
3 -> Enregistrer un dépassement
4 -> Retirer un pilote de la course
5 -> Donner la position d'un pilote
6 -> Faire franchir la ligne d'arrivée au pilote de tête


Votre choix : 3
Entrez le pilote qui dépasse : pilote2

1 -> Afficher toute la course
2 -> Afficher le pilote en tête
3 -> Enregistrer un dépassement
4 -> Retirer un pilote de la course
5 -> Donner la position d'un pilote
6 -> Faire franchir la ligne d'arrivée au pilote de tête


Votre choix : 1
 pilote2 pilote1 pilote3

1 -> Afficher toute la course
2 -> Afficher le pilote en tête
3 -> Enregistrer un dépassement
4 -> Retirer un pilote de la course
5 -> Donner la position d'un pilote
6 -> Faire franchir la ligne d'arrivée au pilote de tête


Votre choix : 4
 Entrez le pilote à supprimer : pilote1

1 -> Afficher toute la course
2 -> Afficher le pilote en tête
3 -> Enregistrer un dépassement
4 -> Retirer un pilote de la course
5 -> Donner la position d'un pilote
6 -> Faire franchir la ligne d'arrivée au pilote de tête


Votre choix : 1
 pilote2 pilote3

Etc…
```

B3 Dans la classe *GestionRallyeAutomobile*, complétez le menu :

```
7 -> Remettre un pilote dans la course (après un autre pilote)
8 -> Afficher les pilotes hors course
9 -> Afficher le classement
```

Attention, de ne pas remettre un pilote en course qui est déjà dans la course ou qui a déjà franchi la ligne d'arrivée. Il faut aussi que le pilote soit bien un pilote qui a pris part à la course.
Les pilotes hors course sont les pilotes qui ont été supprimés.
Dans le classement, on retrouve les pilotes qui ont franchi la ligne d'arrivée. L'ordre des pilotes a de l'importance. Le premier qui a franchi la ligne d'arrivée est le premier et ainsi de suite.
Commencez par compléter la classe *RallyeAutomobile*.
Discutez du choix des structures de données à ajouter avec un professeur.
Il est nécessaire de revoir certaines méthodes et d'en ajouter d'autres.

Exercices supplémentaires

C Mécanisme de LRU

Version semaine 4 : la structure de données utilisée est une liste de documents (String).

METHODE	**COUT**
`ouvrirDocument()`	**O(N)** ☹
`toString()`	**O(N)**

Version semaine 8 (juin 2022) : la structure de données utilisée est une « *listeSD* » de documents (String).

METHODE	**COUT**
`ouvrirDocument()`	**O(1)** ☺
`toString()`	**O(N)**

Mécanisme LRU :
cfr *ExListe* semaine 4

ListeLRU

Les documents vont être placés dans une liste sans doublon (liste doublement chaînée avec sentinelles et un *map*)

Nous vous demandons de compléter la classe *ListeLRU.*
Elle contient les méthodes utiles pour implémenter le mécanisme de LRU.
Testez votre classe à l'aide de la classe *TestListeLRU.*

DocumentsLRU

Nous vous demandons de compléter la classe *DocumentsLRU.*

Cette classe possède une liste de documents (String).
Pour la liste, vous utiliserez un objet de la classe *ListeLRU*.

Le jeu de tests de la classe *TestDocumentsLRU* suit le scénario suivant :

```
Au depart :
 doc1 doc2 doc3 doc4 doc5

ouvrir doc3
 doc3 doc1 doc2 doc4 doc5

ouvrir doc4
 doc4 doc3 doc1 doc2 doc5

ouvrir doc4
 doc4 doc3 doc1 doc2 doc5

ouvrir doc5
 doc5 doc4 doc3 doc1 doc2

ouvrir doc6
 doc6 doc5 doc4 doc3 doc1

ouvrir doc3
 doc3 doc6 doc5 doc4 doc1

ouvrir doc6
 doc6 doc3 doc5 doc4 doc1

ouvrir doc7
 doc7 doc6 doc3 doc5 doc4
```