Université Constantine 2-Abdelhamid Mehri
Faculté des Nouvelles Technologies de l'Information et de la Communication
Département de Tronc Commun MI

ASD – Algorithmique et Structures de Données
Semestre I, Année Universitaire 2023-2024

TP N°5: Implémentation (en Java) du TAD LISTE et son Exploitation (Partie I)

L'implémentation contigüe, du TAD LISTE, respectivement en pseude-code (colonne gauche) et en langage Java (colonne droite) est présentée dans le tableau suivant :

Implémentation Contigüe, du TAD LISTE, en pseudocode	Implémentation Contigüe, du TAD LISTE, en langage Java

Type:
Liste=enregistrement
Tab: tableau [**Lmax**] d'élément
Longueur-liste: entier
fin
Tab est déclaré avec une taille maximale
	public class Liste{
	private static final int **LMAX=10000 ;**
	private Element tab[] ;
	private int longueur_liste ;
	}

Fonction liste-vide ( ): liste
Déclaration
L: liste
Début
L.longueur-liste = 0
Retourner L
fin
	public Liste listeVide(){
	Liste L = new Liste() ;
	L.tab = null ;
	L.longueur_list = 0 :
	return L ;
	}

Fonction insérer (L: liste, p: Entier, e:élement): liste
Déclaration
j: Entier
Début
Si L.longueur-liste < Lmax
Alors Si L.longueur-liste≠0 et p≠ L.longueur-liste+1
Alors
Pour j = L.longueur-liste à p pas = -1 faire
L.Tab[j+1] = L.Tab[j]
Finpour
Fsi
L.Tab[p] = e
L.longueur-liste = L.longueur-liste+1
Sinon écrire ("l'insertion est impossible, la liste est saturée")
Finsi
retourner(L)
Fin
	public Liste inserer(Liste L, int p, Element e){
	……
	
	else{
	System.out.println("l'insertion est impossible, la liste est saturée") ;
	}
	return L ;
	}

Fonction supprimer(L: liste, p: Entier): liste
Déclaration
	public Liste supprimer (Liste L, int p){
	……
	}

j: Entier Début Si L.longueur-liste ≠ 1 Alors Pour j = p à L.longueur-liste -1 faire L.Tab[j] = L.Tab[j+1] Finpur finsi L.longueur-liste = L.longueur-liste - 1 retourner(L) Fin suppresion avec un seul élément dans L	

Fonction acces (L: liste, p:entier): élément
Début
Retourner L.Tab[p]
fin
	public Element acces(Liste L, int p){
	……
	}

Fonction longueur (L: liste): entier
Début
Retourner (L.longueur-liste)
Fin
Fonction récursive
Fonction longueur (L: liste): entier
Début
Si L==liste-vide ( ) /*ou Est-vide (L) == vrai
alors retourner 0
Sinon retourner 1+ longueur (supprimer(L,1))
Fsi
fin
	public int longueur-iter(Liste L){
	
	}
	
	public int longueur-rec(Liste L){
	
	}

Fonction est-vide (L: liste): booleen
Début
retourner (L==liste-vide( ))
Fin
	public boolean estVide(Liste L){
	……
	}

Travail à faire :

<u>1ère Partie:</u>

1. Créer un projet Java contenant 3 classes : **Element, Liste et Test.**
2. Donner le programme Java de la classe **Element** qui contient :
   - L'attribut privé **id** de type entier.
   - Le constructeur **Element(int id)** qui permet d'initialiser l'attribut **id**.
   - La méthode **int getId()** qui permet de retourner la valeur de l'attribut **id**.
3. Compléter le code de la classe **liste** présenté dans la colonne droite du tableau.
4. Donner le programme Java de la classe **Test** qui contient
   - La méthode **public static Liste creationListe(Liste L)** qui permet de créer une liste contenant plusieurs éléments et la retourner. Le nombre des éléments n'est connu qu'au moment de l'exécution. La lecture des éléments à insérer dans la liste doit se faire à partir du clavier.

Remarque importante : **toutes les méthodes, tel que creationListe(Liste L), affichageListe(liste L) doivent réutiliser les fonctions de base du TAD LISTE (insérer (..), supprimer(..) ;……. Voir la** colonne droite)

- La méthode **public static void affichageListe(liste L)** qui permet d'afficher les éléments de la liste '**L'.**
- La méthode **public static void main(String[] args)** et dans laquelle vous allez :

5. Déclarer un objet **liste1** de type Liste.
6. Appeler la méthode **creationListe** afin d'insérer des éléments dans la liste **liste1.**
7. Tester si la liste **liste1** est vide.
    a. Si la liste **liste1** est vide alors afficher le message **« La liste est vide »**.
    b. Sinon afficher le message **« Les élément de la liste sont : »**, puis appeler la méthode **affichageListe** afin d'afficher les éléments de la liste **liste1.**

2ième Partie:

1. Rajouter la méthode **public Element queue (Liste L)** dans la classe liste. Cette méthode permet de retourner le dernier élément d'une liste.
2. Dans la méthode main tester:
    2.1. Si la liste **liste1** est vide alors afficher le message **« La liste est vide »**.
    2.2. Si la liste **liste1** contient un seul élément alors appeler la méthode **affichageListe** pour afficher **liste1.**
    2.3. Sinon appeler la méthode **queue** afin de récupérer le dernier élément de **liste1**.
        2.3.1. Afficher le résultat retourné.

NB : la méthode **queue (Liste L) à développer** doit être issue d'une opération que vous avez à définir (profile, préconditions et axiomes. Puis vous réutilisez la partie droite de votre axiome dans la méthode. Ce principe est à respecter avec toutes les méthodes suivantes **à développer**).

3ième Partie:

1. Rajouter la méthode **public Liste concatener (Liste L1, Liste L2)** dans la classe liste. Cette méthode permet de concaténer deux listes.
2. Dans la méthode main :

2.1. Déclarer un objet **liste2** de type liste.
2.2. Répéter les étapes 6 et 7 de la 1ère partie pour créer et afficher **liste2**.
2.3. Tester :

- Si les deux listes **liste1** et **liste2** sont vides alors afficher le message **« le résultat de la concaténation est une liste vide ».**

- Si la liste **liste1** est vide alors afficher le message **« le résultat de la concaténation est : »,** puis appeler la méthode **AffichageListe** afin d'afficher les éléments de la liste **liste2.**
- Si la liste **liste2** est vide alors afficher le message **« le résultat de la concaténation est : »,** puis appeler la méthode **AffichageListe** afin d'afficher les éléments de la liste **liste1.**
- Sinon si (la taille de **liste1** + la taille de **liste2>LMAX)** alors afficher le message **«la concaténation est impossible »**.
- Sinon appeler la méthode **concatener** afin de concaténer les éléments de la liste **liste1** et la liste **liste2.**

3. Appeler la méthode **affichageListe** pour afficher le résultat trouvé.

4ième Partie:

Dans le but d'enrichir le TAD LISTE implémenté de manière contigüe :

1. Rajouter la méthode **public Liste inverse(liste L)** dans la classe liste. Cette méthode permet d'inverser les éléments d'une liste.
2. Dans la méthode main tester:
   2.1. Si la liste **liste1** est vide alors afficher le message **« La liste est vide»**.
   2.2. Si la liste **liste1** contient un seul élément alors appeler la méthode **affichageListe** pour afficher **liste1.**
   2.3. Sinon appeler la méthode **inverse** afin d'inverser les éléments de **liste1**.
      2.3.1. Appeler la méthode **affichageListe** pour afficher la liste inversée.

Bon travail de conception abstraite, de réutilisation et de développement de boite à outils.