Listes, Pointeurs et Listes génériques

Exercices en Java

Auteur : E.Thirion – 24/02/2019

Document extrait du site cours.thirion.free.fr

Les exercices suivants sont en majorité des projets à complèter. L'interface graphique de ces projets est déjà réalisée ce qui vous permettra un gain de temps important. Ces projets sont disponibles par téléchargement. Pour savoir comment y accéder cliquez ici.

D'autre part, ce document fait partie d'un ensemble de cours du même auteur (programmation procédurale, programmation objet, programmation web, bases de données) auxquels vous pouvez accéder en cliquant ici.

Exercice 1 : Editeur de liste de morceaux

Parties du cours utilisées: pointeurs et listes.

Projet à ouvrir : Exo_Java-ProgObjet2/ELM0

Objectif du projet: il s'agit de réaliser un éditeur de liste de morceaux de musique permettant d'ajouter un nouveau morceau dans la liste, d'en supprimer, etc ... D'un point de vue pédagogique, cet exercice vous permettra de mieux assimiler la représentation de liste d'objets à l'aide de pointeurs (un attribut de l'objet sert de pointeur sur l'élément suivant).

Représentation des données

Un morceau est représenté par la classe **Morceau**, dont voici la déclaration (fichier **Morceau.java) :**

```
public class Morceau {
   String     Titre;
   Morceau Suivant;
   Morceau (String t) {
      Titre = t;
   }
}
```

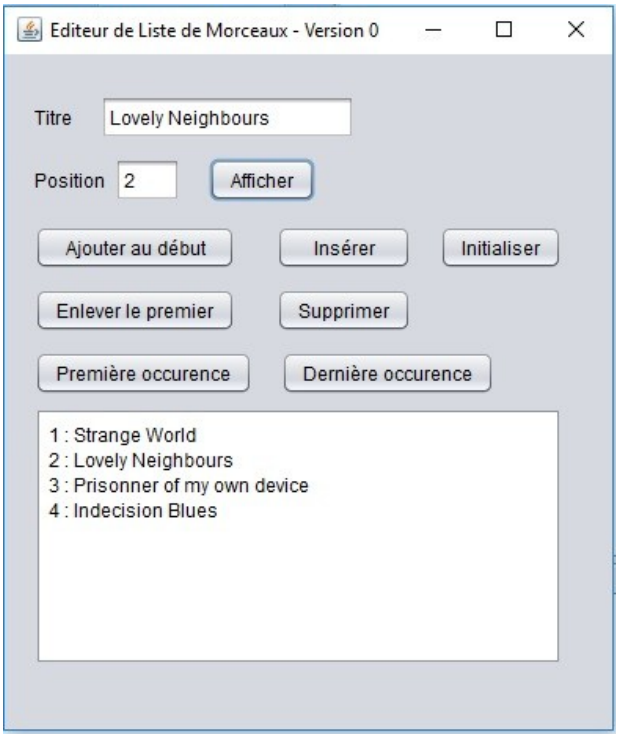
Chaque morceau de la liste est donc caractérisé par son titre et un pointeur vers le morceau suivant (attribut **Suivant**). Le constructeur de la classe est donné. Il prend en paramètre le titre d'un nouveau morceau.

La liste des morceaux est représentée la variable **LM** déclarée comme suit:

```
Morceau LM = null ;
```

Elle est donc initialement vide.

Fenêtre du projet



Composants de la fenêtre:

- Champs de texte: **CT_Titre** :titre du morceau, **CT_Pos** :position d'un morceau dans la liste.
- Zone de texte: **ZT_Liste** : liste des morceaux.

Question 1 : Initialisation de la liste

Le constructeur de la classe fenêtre appelle la méthode suivante:

```
void Initialiser () {
  LM = null;
  Ajouter_Au_Debut("Indecision Blues");
  Ajouter_Au_Debut("Prisonner of my own device");
  Ajouter_Au_Debut("Lovely Neighbours");
  Ajouter_Au_Debut("Strange World");
  AfficherListe();
}
```

Elle ajoute quatre morceaux dans la liste en appelant la procédure **Ajouter_Au_Debut**, puis affiche la liste des morceaux dans la zone de texte **ZT_Liste** en appelant la procédure **AfficherListe**.

Dans cette première question, il s'agit de complèter ces deux procédures afin de faire fonctionner la méthode **Initialiser**. L'affichage d'un titre de morceau sera précédé de sa position dans la liste (en démarrant à 1 pour le premier).

Si vos procédures sont bien écrites, les quatre morceaux devrait s'afficher dans la zone de texte dès le démarrage du programme. D'autre part, les boutons **Initialiser**, **Ajouter au début** et **Enlever le premier** devraient fonctionner. **Initialiser** vous permettra par la suite de tester plus facilement votre programme en réinitialisant la liste à n'importe quel moment.

Question 2 : Affichage du titre d'un élément de position donnée

Le bouton **Afficher** permet à l'utilisateur d'afficher le titre d'un morceau de position donnée.

Pour écrire la procédure évènementielle associée, vous utiliserez les deux fonctions suivantes, qu'il vous faudra tout d'abord complèter :

- Morceau ElementDePosition (**int** pos) : retourne le morceau de position donnée dans la liste (position 1 pour le premier).
- **int** TailleListe ( ) : retourne la taille de la liste **LM**.

On affichera un message d'erreur si la position n'est pas valide.

Question 3 : Insertion d'un élément

Le bouton **Insérer** permet à l'utilisateur d'insérer un nouveau morceau à une position donnée de la liste. Complètez la procédure évènementielle associée en utilisant encore une fois les fonctions **ElementDePosition** et **TailleListe**.

Notez que l'insertion n'est possible que si la position est comprise entre 1 et la taille de la liste + 1. Si ce n'est pas le cas, on affichera un message d'erreur.

Question 4 : Suppression d'un élément

Le bouton **Supprimer** permet à l'utilisateur de supprimer un morceau de position donnée. Complètez la procédure évènementielle associée en utilisant encore une fois les fonctions **ElementDePosition** et **TailleListe**. Vérifiez que la position donnée par l'utilisateur est valide et affichez un message d'erreur dans le cas contraire.

Question 5 : Première occurrence d'un morceau

Un morceau peut se retrouver plusieurs fois dans la liste. Le bouton **Première occurrence** affiche la position du premier morceau de titre donné. Si le titre n'est pas présent dans la liste, on l'indiquera par un message.

Pour effectuer vos tests, vous pouvez utiliser les boutons **Afficher** et **Insérer** pour introduire des titres en double dans la liste.

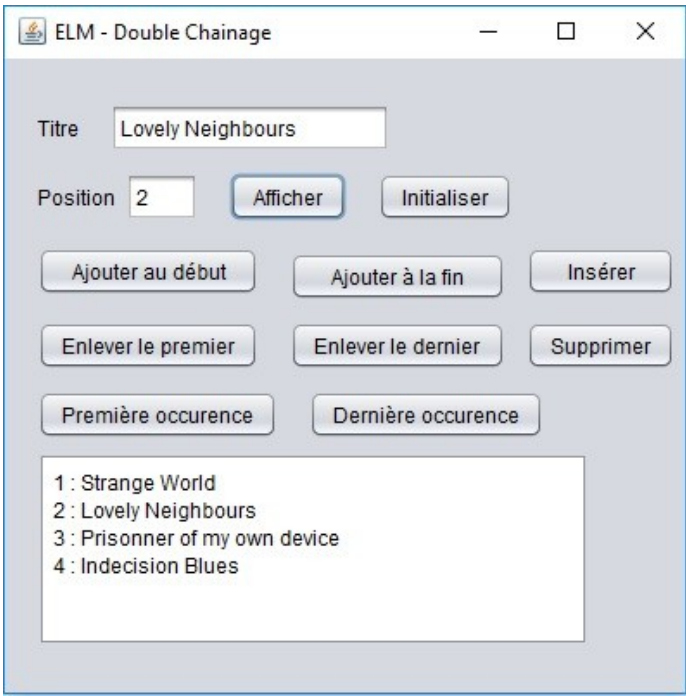
Question 6 : Dernière occurrence d'un morceau

Idem avec la position du dernier morceau de la liste possédant le titre donné.

Exercice 2 : Chainage double

Projet à ouvrir : **Exo_Java-ProgObjet2/ELM0_DC**

En gros, il s'agit de faire la même chose que dans l'exercice précédent, mais avec un double chainage. L'intergraphe graphique est quasiment la même, à part deux boutons supplémentaires permettant d'ajouter un morceau en fin de liste ou d'enlever le dernier morceau:



Représentation des données

La classe **Morceau:**

```
public class Morceau {
   String    Titre;
   Morceau Suivant;
   Morceau Precedent;
   Morceau (String t) {
      Titre = t;
   }
}
```

On a simplement ajouté un pointeur vers le morceau précédent.

La liste des morceaux est représentée par deux pointeurs déclarés dans la classe fenêtre:

```
Morceau LM_Debut = null;
Morceau LM_Fin = null;
```

LM_Debut pointe sur le début de la liste et **LM_Fin**, sur le dernier élément de la liste.

Question 1 : Initialisation de la liste

Comme dans l'exercice 1, sauf que le bouton **Enlever le premier** ne fonctionnera pas tout de suite.

Question 2 : Affichage du titre d'un élément de position donnée

Même objectif. A vous de voir s'il faut changer quelque chose au code.

Question 3 : Adjonction en fin de liste

Il s'agit ici de faire fonctionner le bouton **Ajouter à la fin** qui n'existait pas dans la version précédente. Le code de la procédure évènementielle associée est donné. Il utilise la fonction **Ajouter_A_La_Fin** que l'on vous demande de complèter.

Question 4 : Insertion d'un élément

Même objectif. Vous pourrez éventuellement utiliser la fonction **Ajouter_A_La_Fin** dans le cas où l'insertion se fait en fin de liste.

Question 5 : Suppression d'un élément

Il s'agit cette fois-ci de faire fonctionner trois boutons: **Enlever le premier, Enlever le dernier** et **Supprimer**.

Question 5-1 : Suppression du premier élément

Complètez la fonction **Enlever_Premier**, puis vérifiez que le bouton correspondant fonctionne.

Question 5-2 : Suppression du dernier élément

Complètez la fonction **Enlever_Le_Dernier**, puis vérifiez que le bouton correspondant fonctionne.

Question 5-3 : Suppression d'un élément de position quelconque.

Complètez la procédure évènementielle du bouton **Supprimer**.

Question 6 : Première occurrence d'un morceau

Même objectif que dans l'exercice précédent.

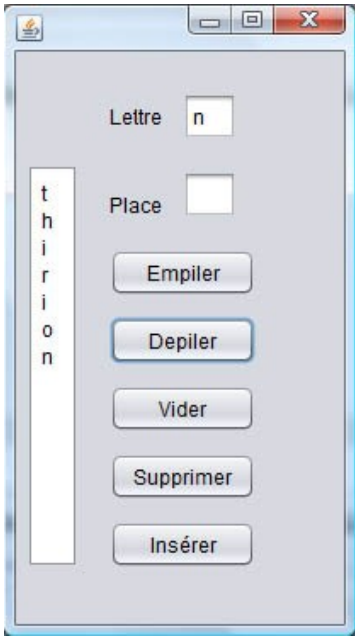
Question 7 : Dernière occurrence d'un morceau

Idem avec la position du dernier morceau de la liste possédant le titre donné. Je vous suggère de changer la méthode: commencez la recherche par la fin de la liste.

Exercice 3: Gestion d'un mot

Objectif: il s'agit re réaliser une nouvelle version du projet **Mot** (cf exercices du cours sur les tableaux). Dans la version précédente le mot était représenter par un tableau de **String**. Ici, nous allons le représenter par une liste générique de **String**. Cela vous permettra de comparer la représentation d'un ensemble de données avec un tableau avec la représentation sous-forme de liste.

Projet à Ouvrir: Exo_Java-ProgObjet2 /Mot2

Fenêtre de l'application	Nom des composants
	Zone de texte : ZT_Mot Champs de texte: CT_Lettre CT_Place

Représentation des données

Le mot est représenté par une liste générique **Mot** déclarée comme suit:

```
LinkedList <String> Mot = new LinkedList <String>();
```

Remarquez qu'il n'est plus nécessaire de définir une taille maximale pour un mot.

1) Affichage du mot

Complètez la procédure **AfficherMot ( )** afin qu'elle affiche toutes les lettres du mot dans la zone de texte. Cette procédure sera appelée après chaque modification du mot. Pour vous simplifier la vie, utilisez ici la boucle **pour chaque** adaptée aux listes génériques.

2) Adjonction d'une lettre à la fin du mot

Lorsque l'utilisateur clique sur le bouton **Empiler**, la lettre est ajoutée à la fin du mot.

3) Suppression de la dernière lettre

Lorsque l'utilisateur clique sur le bouton **Depiler**, la dernière lettre est supprimée. On pourra utiliser ici la méthode **removeLast**.

4) Effacement du mot

Lorsque l'utilisateur clique sur le bouton **Vider**, le mot devient vide.

5) Suppression d'une lettre (pas forcément la dernière)

Lorsque l'utilisateur clique sur le bouton **Supprimer**, la lettre à la place indiquée est supprimée. On supposera que l'utilisateur indique bien la position de la lettre. On aura donc en particulier **Place = 1** pour la première lettre et non pas 0.

5) Insertion d'une lettre

Lorsque l'utilisateur clique sur le bouton **Insérer**, la lettre donnée est insérée à la place indiquée. Même hypothèse que dans la question précédente sur la signification du champ de texte **Place**.

6) Protection du programme contre les erreurs

Pour terminer, protégez votre programme contre les erreurs de manipulation de l'utilisateur. Par exemple, interdire de dépiler lorsque le mot est vide .... Si cela se produit, on affichera un message d'erreur.

Protégez de cette manière chacun des boutons du formulaire contre les opérations qui pourrait provoquer une erreur d'exécution. Pour simplifier, on supposera:

- que le champ de texte **CT_Lettre** contient bien une chaine de caractères de longueur 1 contenant une lettre de l'alphabet.
- que le champ de texte **CT_Place** n'est pas vide et ne contient que des chiffres.