|  |  |
| --- | --- |
| Cours 420-EAB-LI **Développement orienté objets** Hiver 2015 | Laboratoire 2 (8%)  "Gestion de livres" |

**Objectifs**: Après avoir complété ce travail, vous devriez être capable de:

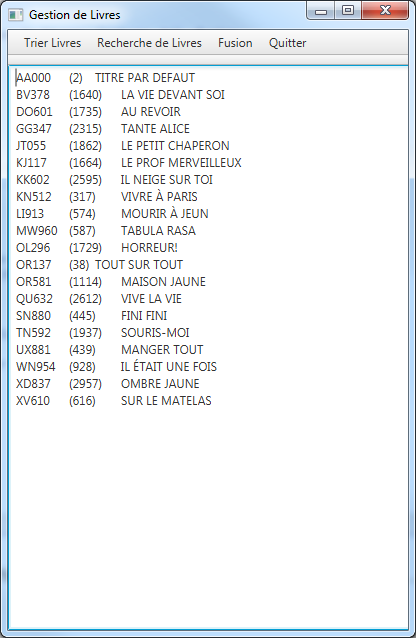
* **Utiliser les différents algorithmes de tri, recherche et fusion de vecteurs d’objets.**
* Utiliser les composants de base dans un environnement fenêtré.
* Gérer une interface graphique plus évoluée.
* Accéder à des données à partir d’un fichier.

Mise en contexte

Ce laboratoire permet de créer un système très simplifié de gestion de livres. Cette application doit être conviviale pour les utilisateurs, c'est pourquoi vous devrez la développer avec une interface graphique.

Un **livre** est caractérisé par une cote, un titre, un nombre de pages et une valeur (prix). Ces informations seront regroupées sous une même entité objet, c'est-à-dire un « **Livre** ». Il s'agira donc des données que votre système aura à manipuler.

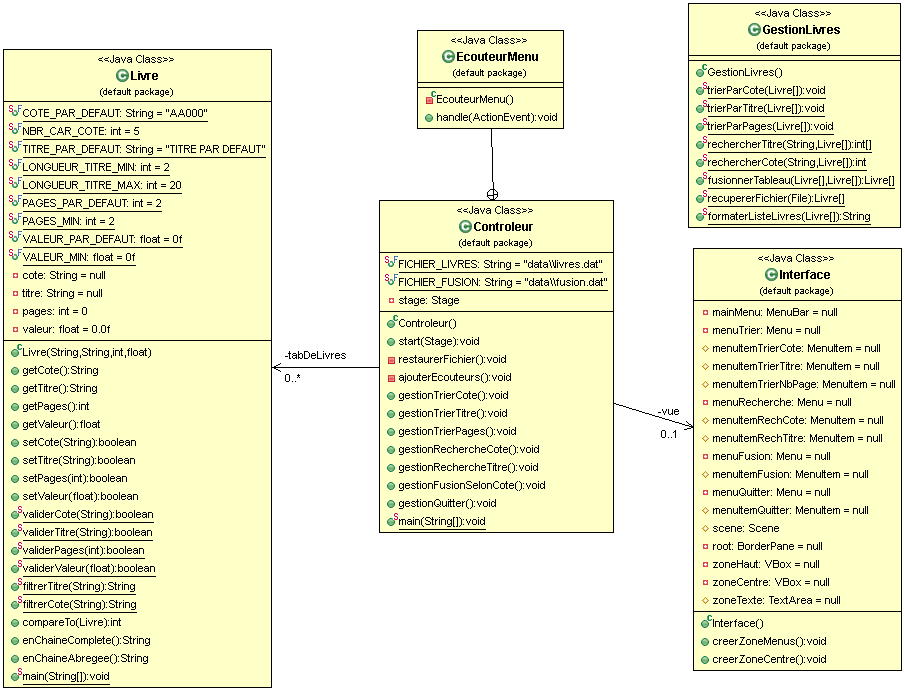
Les opérations que vous aurez à effectuer sur ces données sont les suivantes:

1. Récupérer un fichier de livres.
2. Trier les données provenant de ce fichier.
3. Faire la recherche dichotomique d'un livre à partir de sa cote à l’identique.
4. Faire la recherche séquentielle d'un livre avec son titre commençant par une chaîne.
5. Fusionner des tableaux de données provenant de fichiers différents.
6. Afficher les messages d'informations et d'erreurs nécessaires.
7. Voir à ce que l'application se ferme correctement.

Voici l’interface graphique à réaliser.

Pour réaliser ce travail, vous devez créer 4 classes. Une classe **Livre**, une classe **GestionLivres,** une classe **Interface** et une classe **Controleur**. On vous fournit aussi un modèle de classes et les fichiers HTML de la « Javadoc » de cette application.

Description des classes



1. Aucune JavaDoc n’est demandée mais vous devez commenter l’intérieur des méthodes lorsque ceci est pertinent.
2. À remettre :

* Un **fichier** « .zip », du nom de « **Votre nom‑TP2.zip** », du code source complet (.java) bien formaté et bien commenté de toutes les classes, déposé sur LEA.

**public** **class** Livre **implements** Comparable<Livre>

{

/\*\*

\* Valider la cote du livre. La cote doit être obligatoirement du format

\* suivant soit 2 lettres majuscules suivies de 3 chiffres.

\*

\* **@param** pCote la cote à valider

\*

\* **@return** boolean, vrai si la valeur est valide

\*/

**public** **static** **boolean** validerCote(String pCote)

{

/\*

\* return ((pCote != null) && (pCote.length() == Livre.NBR\_CAR\_COTE) &&

\* (Character.isUpperCase(pCote.charAt(0))) &&

\* (Character.isUpperCase(pCote.charAt(1))) &&

\* (Character.isDigit(pCote.charAt(2))) &&

\* (Character.isDigit(pCote.charAt(3))) &&

\* (Character.isDigit(pCote.charAt(4))));

\*/

// OU

// Expressions régulières

**return** ((pCote != **null**) && (pCote.matches("[A-Z][A-Z][0-9][0-9][0-9]")));

}

/\*\*

\* Permet de mettre en forme, sans validation, une chaîne de caractères qui

\* représente le titre d'un livre. Enlève les espaces inutiles de début et

\* de fin et met la chaîne en majuscules.

\*

\* **@param** pTitre la chaîne à filtrer.

\*

\* **@return** String, la chaîne filtrée

\*/

**public** **static** String filtrerTitre(String pTitre)

{

**if** (pTitre != **null**)

{

pTitre = pTitre.trim().toUpperCase();

}

**return** pTitre;

}

/\*\*

\* Permet de mettre en forme, sans validation, une chaîne de caractères qui

\* représente une cote. Enlève les espaces inutiles de début et de fin, met

\* la chaîne en majuscules et ajuste sa longueur.

\*

\* **@param** pCote la chaîne à filtrer.

\*

\* **@return** String, la chaîne filtrée

\*/

**public** **static** String filtrerCote(String pCote)

{

**if** (pCote != **null**)

{

pCote = pCote.trim().toUpperCase();

// pCote = (pCote.length() > Livre.NBR\_CAR\_COTE) ?

// pCote.substring(0, Livre.NBR\_CAR\_COTE) : pCote);

// OU

**if** (pCote.length() > Livre.***NBR\_CAR\_COTE***)

{

pCote = pCote.substring(0, Livre.***NBR\_CAR\_COTE***);

}

}

**return** pCote;

}

/\*\*

\* Permet de comparer deux livres en considérant leur nombre de pages.

\*

\* <br>

\* <br>

\* - Si le livre courant a un nombre de pages plus petit que le livre reçu

\* en entrée, une valeur négative sera retournée.

\*

\* <br>

\* - Si le livre courant a un nombre de pages plus grand que le livre reçu

\* en entrée, une valeur positive sera retournée.

\*

\* <br>

\* - Si le livre courant a un nombre de pages égale au livre reçu en entrée,

\* la valeur 0 sera retournée.

\*

\* <br>

\* <br>

\* <b>Attention:</b> L'adresse de l'objet livre reçu en entrée peut être

\* nulle, dans ce cas, le livre courant est toujours plus grand.

\*

\* **@param** pLivre le livre avec lequel il faut comparer le livre courant.

\*

\* **@return** int, une valeur négative si inférieur, positive si supérieur et 0

\* si égale.

\*/

**public** **int** compareTo(Livre pLivre)

{

**int** valRetour = 1;

**if** (pLivre != **null**)

{

valRetour = **this**.getPages() - pLivre.getPages();

}

**return** valRetour;

}

**public** **static** **void** main(String[] args)

{

Livre l1 = **new** Livre("AA111", "Titre", 12, 30);

System.***out***.println(l1.getCote()+"\n"+

l1.getTitre()+"\n"+l1.getValeur()+"\n"+l1.getPages());

Livre l2 = **new** Livre("BB111", "Titre", 14, 35);

System.***out***.println(l1.compareTo(l2));

}

AA111

Titre

30.0

12

-2

**package** TP2;

**import** java.io.\*;

**import** javafx.scene.control.Alert;

**import** javafx.scene.control.Alert.AlertType;

**public** **class** GestionLivres

{

/\*\*

\* Effectue le tri en ordre croissant d'un tableau, reçu en entrée, en fonction de la cote des livres. Utilise l'algorithme de tri à bulle

\* amélioré.

### Tri à bulle amélioré:

courant = (n-2)

//Il faut le faire au moins une fois

Faire

flag = 0

Pour position variant de 0 à courant

Si a[position] > a[position+1]

valTemp = a[position+1]

a[position+1] = a[position]

a[position] = valTemp

flag = 1

Fin si

Fin pour

courant = courant - 1

Tant que (flag est différent de 0)

\* **@param** pTab le tableau de livres à trier

\*/

**public** **static** **void** trierParCote(Livre[] pTab)

{

// Déclaration des variables

Livre livreTemp = **null**;

**boolean** flag = **false**;

**int** courant = (pTab.length - 2);

// Pour chaque passe

**do**

{

// Pour chaque livre dans la passe

**.....**

// Si la cote du livre (pTab[position].getCote()) est plus grande que la cote du suivant

// les permuter

**....**

// permutation des livres à position et position + 1

}

}

}

**while** (flag);

}

/\*\*

\* Effectue le tri en ordre croissant d'un tableau, reçu en entrée, en fonction du titre des livres. Utilise l'algorithme de tri par insertion.

### \*

### Tri par insertion :

Pour courant variant de 1 à (n-1)

valTemp = a[courant]

position = courant

Tant que (( position > 0 ) et ( a[position - 1] > valTemp ))

a[position] = a[position - 1]

position = position - 1

Fin tant que

a[position] = valTemp

Fin pour

\*

\* **@param** pTab le tableau à trier

\*/

**public** **static** **void** trierParTitre(Livre[] pTab)

{

// Déclaration des variables

// Pour chaque livre du tableau

// Tant que le titre du précédent est plus grand

// Le décaler vers la droite

// insérer le courant

}

/\*\*

\* Effectue le tri en ordre croissant d'un tableau, reçu en entrée, en

\* fonction du nombre de pages des livres. Utilise l'algorithme de tri par

\* sélection.

\*

### \*

### Tri par sélection :

Pour courant variant de 0 à (n-2)

indiceTemp = courant

Pour position variant de courant+1 à (n-1)

Si (a[position] < a[indiceTemp])

indiceTemp = position //valeur de l’indice du vecteur A

Fin si

Fin pour

Si indiceTemp <> courant

valTemp = a[indiceTemp]

a[indiceTemp] = a[courant]

a[courant] = valTemp

Fin si

Fin pour

\*

\* **@param** pTab le tableau à trier

\*/

**public** **static** **void** trierParPages(Livre[] pTab)

{

// Déclaration des variables

// Pour chaque livre du tableau

}

/\*\*

\* Recherche séquentielle, dans le tableau reçu en entrée (pTab), d'un titre

\* qui commence par la chaîne reçue en entrée (pTitre). Les livres trouvés

\* seront tous ceux pour lesquels le titre commence par la chaîne pTitre. Il

\* peut y avoir plusieurs titres pour lesquels le titre (ou le début du

\* titre) correspond à la chaîne.

\*

\* <br>

\* <br>

\* Cette méthode retourne un tableau des indices des livres trouvés ou

\* <b>null</b> s’il y en a aucun. (Voir algorithme de recherche

\* séquentielle). Attention aux minuscules et majuscules.

\*

\* <br>

\* <br>

\* <b>Attention:</b> Vous n'avez pas besoin de vérifier si "pTab" est égale

\* à null, mais le tableau peut être vide (0 objet).

\*

\* <br>

\* <br>

\* <b>IMPORTANT : la méthode "startsWith()" de la classe "String" pourrait

\* peut-être vous aider.</b>

\*

\* **@param** pTitre le début du titre à rechercher

\* **@param** pTab le tableau de livres

\*

\* **@return** int[], un tableau des indices des livres correspondant ou "null"

\*/

**public** **static** **int**[] rechercherTitre(String pTitre, Livre[] pTab)

{

// Déclaration des variables

**int** indice = 0, nombreTrouve = 0;

**int**[] vect = **null**;

// Compter le nombre de livres dans pTab dont le titre commence par

// pTitre

// si le titre commence par pTitre, le compter

**if** (pTab[indice].getTitre().toLowerCase()

.startsWith(pTitre.toLowerCase()))

{

nombreTrouve++;

}

// si des livres ont été trouvés

**if** (nombreTrouve > 0)

{

vect = **new** **int**[nombreTrouve];

**int** j = 0;

// Pour chaque livre du tableau

// si le titre commence par pTitre, ajouter l'indice au

// tableau

}

}

// retourner le tableau des indices

**return** vect;

}

/\*\*

\* Recherche dichotomique, dans un tableau de livres trié par cote, reçu en

\* entrée (pTab), d'un livre ayant une cote correspondant de façon identique

\* à celle reçue en entrée (pCote) (Voir algorithme de recherche

\* dichotomique).

\*

\* <br>

\* <br>

\* La cote est unique et un seul livre pourra être trouvé. La méthode

\* retourne l'indice du livre trouvé ou –1 si la cote n'existe pas.

\* Attention aux minuscules et majuscules.

\*

\* <br>

\* <br>

\* <b>Attention:</b> Vous n'avez pas besoin de vérifier si "pTab" est égale

\* à null, mais le tableau peut être vide (0 objet).

\*

\* <br>

\* <br>

\* <b>IMPORTANT : la classe "String" est "Comparable".</b>

\*

\* **@param** pCote la cote du livre à rechercher

\* **@param** pTab un tableau de livres

\*

\* **@return** int, l'indice du livre trouvé ou -1 si pas trouvé

\*/

L’algorithme suivant fait la recherche d’un élément dans un tableau **OBLIGATOIREMENT** trié.

### Recherche dichotomique :

inf = 0 //Indice inférieur de la recherche

sup = n - 1 //Indice supérieur de la recherche

indice = -1 //Valeur retournée si pas trouvé

Tant que (( inf <= sup ) et ( indice = -1 ))

milieu = (inf + sup) / 2

Si (a[milieu] > elementRechercher )

sup = milieu - 1

Sinon Si (a[milieu] < elementRechercher )

inf = milieu + 1

Sinon

indice = milieu

Fin si

Fin tant que

Retourner indice

**public** **static** **int** rechercherCote(String pCote, Livre[] pTab)

{

// Déclaration des variables

// tant que la borne inférieure < borne supérieure

// et qu'on n'a pas trouvé

// si la cote du milieu > cote cherchée, chercher à gauche

// sinon si la cote du milieu < cote cherchée, chercher à droite

// sinon, on l'a trouvé

**return** indice;

}

/\*\*

\* Effectue la fusion de 2 tableaux de livres <b>triés par cote</b>, reçus

\* en entrée. Elle retourne le nouveau tableau fusionné. (Voir algorithme de

\* fusion).

\*

\* <br>

\* <br>

\* <b>Attention:</b> Vous n'avez pas besoin de vérifier si "pTab1" ou

\* "pTab2" sont égale à null, mais les tableaux peuvent être vides (0

\* objet).

\*

\* <br>

\* <br>

\* <b>IMPORTANT : la classe "String" est "Comparable".</b>

\*

\* **@param** pTab1 un premier tableau de livres

\* **@param** pTab2 un deuxième tableau de livres

\*

\* **@return** Livre[], un tableau de livres, le résultat de la fusion

\*/

**Identificateurs :**

v1: Un tableau (vecteur) trié de nv1 éléments

v2: Un tableau (vecteur) trié de nv2 éléments

v3: Un tableau (vecteur) trié résultant de la fusion (nv1 + nv2 éléments)

iv1: Indice où on est rendu dans le tableau v1

iv2: Indice où on est rendu dans le tableau v2

iv3: Indice où on est rendu dans le tableau v3

nv1: Nombre d'éléments du tableau v1

nv2: Nombre d'éléments du tableau v2

**Algorithme :**

iv1 = 0

iv2 = 0

iv3 = 0

Faire

Si (v1[iv1] < v2[iv2])

v3[iv3] = v1[iv1]

iv1 = iv1 + 1

Sinon

v3[iv3] = v2[iv2]

iv2 = iv2 + 1

Fin si

iv3 = iv3 + 1

Tant que (( iv1 < nv1 ) et ( iv2 < nv2 ))

//Placer le reste des éléments dans vecteur v3

Si (iv1 = nv1)

Pour iv2 variant de iv2 à nv2-1

v3[iv3] = v2[iv2]

iv3 = iv3 + 1

Fin pour

Sinon

Pour iv1 variant de iv1 à nv1-1

v3[iv3] = v1[iv1]

iv3 = iv3 + 1

Fin pour

Fin si

**public** **static** Livre[] fusionnerTableau(Livre[] pTab1, Livre[] pTab2)

{

// Déclaration des variables

// Faire tant que on n'est pas rendu à la fin d'un des tableaux

// si la cote du livre dans tableau1 est plus petite que celle dans

// tableau2

// placer le livre dans le tableauFusion

// sinon placer le livre du tableau2 dans le tableauFusion

}

// si on est à la fin du tableau1, placer les livres restant

// dans le tableau2 dans le tableauFusion

// si on est à la fin du tableau2, placer les livres restant

// dans le tableau1 dans le tableauFusion

**return** tabFusion;

}

/\*\*

\* Créer un tableau de livres à partir d'un fichier de données lues.

\*

\* **@param** pFichier le chemin et le nom (path) du fichier à récupérer.

\*

\* **@return** Livre[], un tableau d'objets "Livres".

\*/

**public** **static** Livre[] recupererFichier(File pFichier)

{

DataInputStream dataInpStr = **null**;

**int** compteur = 0;

**int** i = 0;

Livre[] tabRetour = **null**;

// Lire le fichier une fois pour compter combien il y a de livres.

**try**

{

dataInpStr = **new** DataInputStream(**new** BufferedInputStream(

**new** FileInputStream(pFichier)));

**try**

{

**do**

{

dataInpStr.readUTF();

.....

}

**while** (**true**);

}

// Fermer le fichier lorsque fin de fichier détecté

**catch** (EOFException except)

{

dataInpStr.close();

}

}

// Afficher message d'erreur lorsque le fichier n'est pas ouvert

**catch** (java.io.IOException excep)

{

// **TODO** Faire attention à l'affichage de message direct!!!

Alert alert = **new** Alert(AlertType.***ERROR***);

alert.setTitle("Erreur IO:");

alert.setHeaderText("");

alert.setContentText("Erreur" + excep.getMessage());

alert.showAndWait();

}

// Ne rien faire si on n'a pas de livre

................

// Instancier le tableau selon le nombre de livres comptés à la

// lecture précédente.

tabRetour = **new** Livre[compteur];

// Lire le fichier une seconde fois pour remplir le tableau

**try**

{

dataInpStr = **new** DataInputStream(**new** BufferedInputStream(

**new** FileInputStream(pFichier)));

**try**

{

**do**

{

// Ajouter le livre lu dans le tableau

................

}

**while** (**true**);

}

// Fermer le fichier lorsque fin de fichier détecté

**catch** (EOFException except)

{

dataInpStr.close();

}

}

// Afficher message d'erreur lorsque fichier pas ouvert

**catch** (java.io.IOException excep)

{

// **TODO** Faire attention à l'affichage de message direct!!!

Alert alert = **new** Alert(AlertType.***ERROR***);

alert.setTitle("Erreur IO:");

alert.setHeaderText("");

alert.setContentText("Erreur" + excep.getMessage());

alert.showAndWait();

}

}

**return** tabRetour;

}

/\*\*

\* Construire une String contenant l'information (cote et titre, i.e. sous

\* forme de chaîne abrégée) sur tous les livres du tableau, un livre par

\* ligne. Elle servira pour l’affichage dans la liste de la fenêtre

\* graphique.

\*

\* **@param** pTab le tableau de livres

\*

\* **@return** String, la chaîne construite

\*/

**public** **static** String formaterListeLivres(Livre[] pTab)

{

String texte = "";

..............

**return** texte;

} }

//Test

**public** **static** **void** main(String[] args)

{

Livre[] TLivre;

Livre l1 = **new** Livre("AA111", "Titre1", 12, 40);

Livre l2 = **new** Livre("BB111", "Titre4", 24, 38);

Livre l3 = **new** Livre("CC444", "Titre2", 18, 30);

Livre l4 = **new** Livre("DD555", "Titre3", 32, 37);

TLivre=**new** Livre[4];

TLivre[0]=l1;

TLivre[1]=l2;

TLivre[2]=l3;

TLivre[3]=l4;

//trierParCote(TLivre);

//System.out.println(formaterListeLivres(TLivre));

//trierParTitre(TLivre);;

//System.out.println(formaterListeLivres(TLivre));

//trierParPages(TLivre);;

//System.out.println(formaterListeLivres(TLivre));

//String ct=JOptionPane.showInputDialog(null, "entrer une cote:");

// System.out.println(TLivre[rechercherCote(ct, TLivre)].getTitre()+"\n"+TLivre[rechercherCote(ct, TLivre)].getValeur()+"\n"+TLivre[rechercherCote(ct, TLivre)].getPages());

String pt=JOptionPane.*showInputDialog*(**null**, "entrer un titre:");

**int** taille=*rechercherTitre*(pt,TLivre).length;

**for**(**int** i=0;i<taille;i++)

{

System.***out***.println(TLivre[*rechercherTitre*(pt, TLivre)[i]].getCote()+ " "+ TLivre[*rechercherTitre*(pt, TLivre)[i]].getTitre()+" "+TLivre[*rechercherTitre*(pt, TLivre)[i]].getPages()+"\n");

}

}

**package** TP2;

**import** java.io.File;

**import** java.util.Optional;

**import** javafx.application.Application;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.scene.control.Alert;

**import** javafx.scene.control.Alert.AlertType;

**import** javafx.scene.control.ButtonType;

**import** javafx.scene.control.TextInputDialog;

**import** javafx.stage.Stage;

/\*\*

\* Classe permettant la manipulation des écouteurs et les actions associées.

\*

\* **@version** 2.0 (Hiver 2015)

\* **@author**

\*/

**public** **class** Controleur **extends** Application

{

/\*\*

\* Chemin et nom du fichier de livres lu à l'ouverture de l'application.

\*/

**public** **final** **static** String ***FICHIER\_LIVRES*** = "livres.dat";

/\*\*

\* Chemin et nom du fichier de livres lu lors de la demande de fusion.

\*/

**public** **final** **static** String ***FICHIER\_FUSION*** = "fusion.dat";

/\*\*

\* Un pointeur pour l'interface

\*/

/\*\*

\* Un pointeur Stage pour le stage

\*/

**....**

/\*\*

\* Un tableau (vecteur) de livres. Il sera rempli au début à partir du

\* contenu du fichier livres.dat depuis data

\*/

**......**

/\*\*

\* Méthode qui instancie l'interface, ajoute les écouteurs sur les sous

\* menu, récupère le fichier de livres et affiche l'interface de départ

\*/

**public** **void** start(Stage pStage)

{

..................

}

/\*\*

\* Méthode qui permet de récupérer le fichier, le trier par cote et le

\* placer dans la zone de texte

\*/

**private** **void** restaurerFichier()

{

....

}

/\*\*

\* Méthode qui permet d'ajouter un écouteur d'évenements à chacun des sous

\* menu de la zone de menus

\*/

**private** **void** ajouterEcouteurs()

{

.................

}

/\*\*

\* Classe permettant d'implémenter la méthode pour l'écoute des clics sur

\* les sous menu de la zone de menu

\*/

**private** **class** EcouteurMenu **implements** EventHandler<ActionEvent>

{

@Override

**public** **void** handle(ActionEvent e)

{

..........

}

}

/\*\*

\* Effectue les opérations nécessaires pour que le tableau de livres soit

\* trié par cote (trierParCote) et réaffiche correctement la liste de livres

\* dans le TextArea.

\*/

**public** **void** gestionTrierCote()

{

....................

}

/\*\*

\* Effectue les opérations nécessaires pour que le tableau de livres soit

\* trié par titre (trierParTitre) et réaffiche correctement la liste de

\* livres dans le TextArea.

\*/

**public** **void** gestionTrierTitre()

{

....

}

/\*\*

\* Effectue les opérations nécessaires pour que le tableau de livres soit

\* trié par nombre de pages (trierParPages) et réaffiche correctement la

\* liste de livres dans le TextArea.

\*/

**public** **void** gestionTrierPages()

{

...............

}

/\*\*

\* Effectue la gestion de la recherche d'un livre selon une cote.<br>

\* <br>

\*

\* Elle demande la cote. Assurez-vous que la saisie de la cote n’a pas été

\* annulée et que la cote filtrée soit valide (Saisie avec feedback).

\*

\* Effectue ensuite la recherche et affiche le livre trouvé ou un message

\* approprié si cote inexistante.

\*/

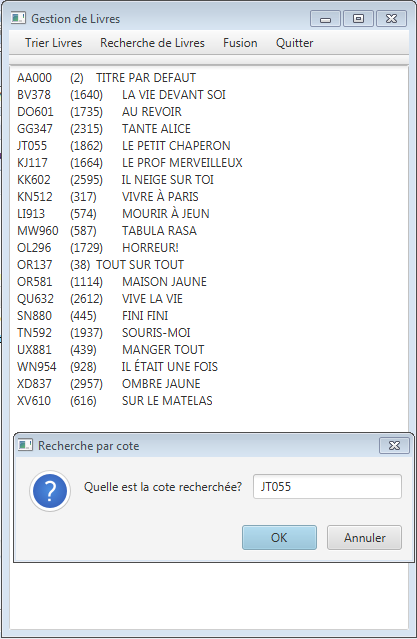
**public** **void** gestionRechercheCote()

{

String coteCherchee = **null**;

**int** coteTrouvee = 0;

// demander la cote à chercher



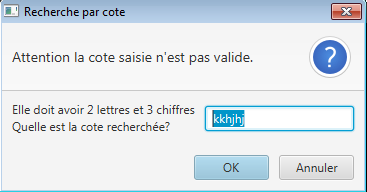
// boucle de saisie, si présence de contenu et que la chaine n'est pas

// valide après filtration

**while** ((result.isPresent())

&& (!Livre.*validerCote*(Livre.*filtrerCote*(result.get()))))

{



}

**if** (result.isPresent())

{

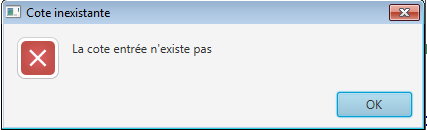
coteCherchee = Livre.*filtrerCote*(result.get());

// rechercher la cote, avant de chercher, il faut trier le tableau.

// si pas trouvé afficher un message

**if** (coteTrouvee == -1)

{



}

// si la cote a été trouvée, les renseignements du livre sont

// affichés

**else**

{



}

}

}

/\*\*

\* Effectue la gestion de la recherche de titres de livres. <br>

\* <br>

\*

\* Elle demande un titre. Assurez-vous que la saisie du titre n’a pas été

\* annulée et que le titre filtré ait au moins un caractère. (Saisie avec

\* feedback).<br>

\* <br>

\*

\* Effectue ensuite la recherche et affiche dans des boîtes de message le ou les

\* livres dont le titre <b>commence</b> par la chaîne saisie ou un message

\* approprié si aucun livre n'existe avec ce titre.

\*/

**public** **void** gestionRechercheTitre()

{

// Déclaration des variables

// Demander le titre recherché

// boucle de saisie

// rechercher les indices des livres trouvés sinon tableau null

// si aucun livre trouvé, afficher un message

// sinon, afficher les livres trouvés

}

/\*\*

\* Effectue la gestion de la fusion. <br>

\* <br>

\*

\* Elle récupère le contenu du fichier FICHIER\_FUSION (Appel de la méthode

\* recupererFichier) et trie par cote le tableau résultant de cette

\* récupération de fichier s'il n'est pas à "null". <br>

\* <br>

\*

\* Elle trie aussi par cote le tableau courant "tabDeLivres" s'il n'est pas

\* à "null" et fusionne ces 2 tableaux triés.<br>

\* <br>

\*

\* Le résultat de la fusion est remis dans le tableau courant "tabDeLivres".<br>

\* <br>

\*

\* Attention, il faut prévoir les cas où un des 2 tableaux (ou les 2) soit

\* null.

\*

\* Après la fusion, elle réaffiche correctement les informations du tableau

\* fusionné dans le TextArea et désactive le sous menu "Fusion".

\*/

**public** **void** gestionFusionSelonCote()

{

Livre[] tabTemp = GestionLivres.*recupererFichier*(**new** File(

***FICHIER\_FUSION***));

// Le nouveau tableau est-il null

**if** (tabTemp != **null**)

{

// Le trier selon la cote

GestionLivres.*trierParCote*(tabTemp);

// Le tableau interne est-il null

**if** (tabDeLivres != **null**)

{

// Le trier selon la cote

// Fusionner les deux tableaux

tabDeLivres = GestionLivres.*fusionnerTableau*(tabDeLivres,

tabTemp);

}

**else**

{

tabDeLivres = tabTemp;

}

}

// Le deuxième tableau est-t-il null?

**else** **if** (tabDeLivres != **null**)

{

GestionLivres.*trierParCote*(tabDeLivres);

}

vue.zoneTexte.setText(GestionLivres.*formaterListeLivres*(tabDeLivres));

// Désactive l'option

vue.menuItemFusion.setDisable(**true**);

}

/\*\*

\* Effectue les opérations nécessaires pour permettre une sortie conviviale

\* du programme.

\*/

**public** **void** gestionQuitter()

{

}

**public** **static** **void** main(String[] args)

{

// **TODO** Auto-generated method stub

Application.*launch*(args);

}

}