|  |  |
| --- | --- |
| 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:

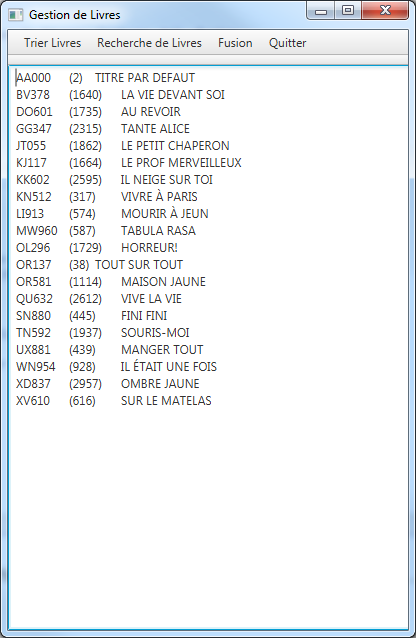
* **Utiliser les différents algorithmes de tri, recherche et fusion de vecteurs d’objets.**
* Utiliser les composants de base dans un environnement fenêtré.
* Gérer une interface graphique plus évoluée.
* Accéder à des données à partir d’un fichier.

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.

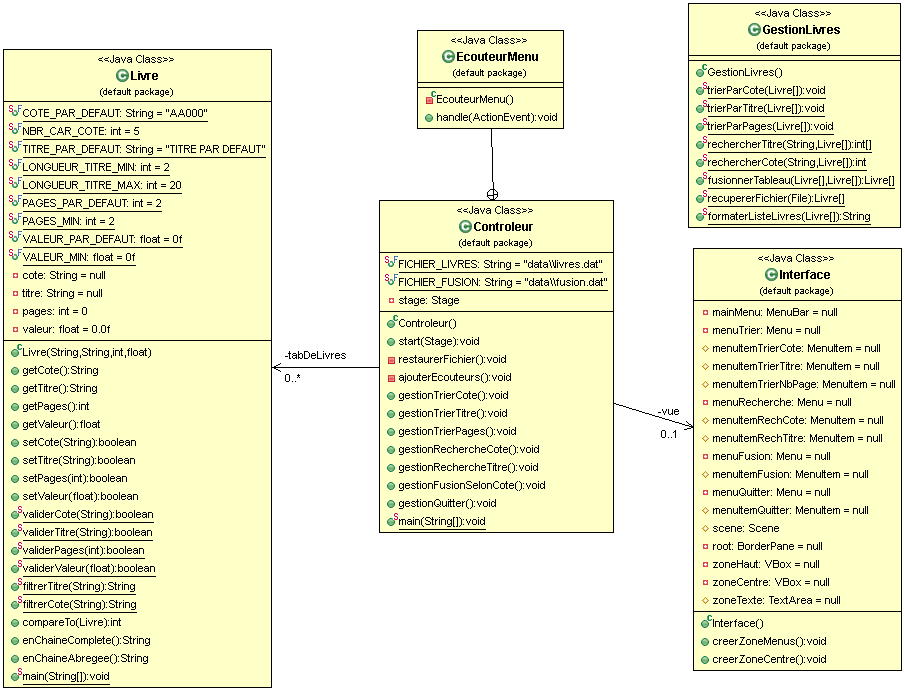
Les opérations que vous aurez à effectuer sur ces données sont les suivantes:

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

Échéancier pour les remises

Dépôt 1 : durée 2 semaines : La classe Livre, la classe GestionLivres et les classes JUnit

(60 points)

1. Coder toutes les méthodes de la classe **Livre**.
2. Faire les tests unitaires JUnit de la classe Livre dans la classe **LivreTest** (à créer). Attention aux tests réalisés! Toutes les méthodes de la classe qui ont été codées doivent être testées ainsi que tous les cas possibles de validation. Pour qu'il fonctionne bien, vous devrez copier le dossier « data » (il contient les fichiers livres.dat et fusion.dat).

**NOTE** : Essayez de coder vos tests avant de coder vos méthodes (Test Driven Development (TDD) -- <http://fr.wikipedia.org/wiki/Test_Driven_Development>)

* À partir de la classe **GestionLivres**, coder toutes les méthodes de cette classe (sauf recupererFichier() qui est offert.
* Faire les tests unitaires JUnit de la classe **GestionLivres** dans la classe **GestionLivresTest** (à créer). Attention aux tests réalisés! Toutes les méthodes de la classe qui ont été codées doivent être testées ainsi que tous les cas possibles de validation.
* Aucune JavaDoc n’est demandée mais vous devez commenter l’intérieur des méthodes lorsque ceci est pertinent (complexité plus grande).
* Voir **Lab2.exe**

1. **À remettre au début de la période de laboratoire** :

* Un **fichier** « .zip », du nom de « **Votre nom‑Lab2 sem1-2.zip** », du code source complet (.java) bien formaté et bien commenté des classes **Livre, LivreTest, GestionLivres et GestionLivresTest** déposé sur LEA.

Dépôt 2 : durée 1 semaine: La classe Interface et la classe Controleur (voir Lab2.exe)

(40 points)

1. Coder toutes les méthodes de la classe **Interface**.
2. À partir de la classe **Controleur**, coder toutes les méthodes de cette classe (sauf restaurerFichier() qui est offerte).
3. Aucune JavaDoc n’est demandée mais vous devez commenter l’intérieur des méthodes lorsque ceci est pertinent (complexité plus grande).
4. À remettre :

* Un **fichier** « .zip », du nom de « **Votre nom‑Lab2 sem3.zip** », du code source complet (.java) bien formaté et bien commenté de toutes les classes, déposé sur LEA.