



EXAMEN DE FIN D'ÉTUDES SECONDAIRES TECHNIQUES 2017

BRANCHE	SECTION(S)	ÉPREUVE ÉCRITE
Programmation	GI	Durée de l'épreuve 4 heures
		Date de l'épreuve 29/05/2017
		Numéro du candidat

Indication importante :

- ☞ Renommez d'abord le dossier EEEE_GI_x-xx en remplaçant EEEE par le code de votre établissement et x-xx par votre numéro d'examen !

1. BooksBst – Bibliothèque dans un ABR (19 p.)

- ☞ Vous allez compléter le projet **BooksBst** qui se trouve dans le dossier QUESTION1.

La vidéo **BooksBst.mov** qui se trouve dans votre dossier vous sert de référence.

Le projet **BooksBst** permet de lire les livres contenus dans le fichier **Biblio.txt** (qui se trouve dans le dossier du projet) et de les transférer dans un ABR (*arbre binaire de recherche* ; **anglais** : *BST* : *Binary Search Tree*). Dans l'ABR les données sont organisées (triées) selon le code de référence des livres. Ensuite, il est possible de rechercher un livre par son code de référence ou de rechercher tous les livres dont la description contient un mot clé donné (pour les détails voir plus bas).

La classe **Book** qui représente un livre est donnée et n'a pas besoin d'être changée. Le constructeur **Book** crée un nouveau livre à partir d'une ligne lue dans le fichier. La méthode **toString** retourne les informations les plus importantes du livre, mais pas toutes (voir code de la classe).

La classe **MainFrame** qui contient l'interface graphique est donnée, mais quelques détails doivent être complétés ou ajoutés.

Ouvrez le projet **BooksBst** et suivez les étapes de développement indiquées ci-dessous.

Ouvrez et complétez la classe **BooksBst** qui sert à gérer les livres dans un ABR. Ajoutez les méthodes publiques décrites ci-dessous. Il est nécessaire de définir des méthodes supplémentaires pour réaliser les parties récurrentes des méthodes **add**, **size**, **get** et **find**. Ces méthodes supplémentaires ne doivent pas être accessibles à l'extérieur de la classe.

add (Book book) : ajout d'un livre en organisant l'ABR selon **referenceCode**

int size() : retourne le nombre de livres actuellement contenus dans l'ABR

loadFromFile(String fileName) : lecture des livres contenus dans le fichier dont le nom est donné comme paramètre. Des exceptions éventuelles sont traitées dans la vue. Complétez la réaction du bouton **loadFileButton**. Le nombre de livres est affiché dans le libellé **sizeLabel**. Si une exception survient, le message de l'exception est affiché dans ce libellé.

Book get(String refCode) : retourne le livre dont le code de référence est fourni comme paramètre, ou **null** si un tel livre n'existe pas. La réaction du bouton **getButton** est déjà définie, il suffit de la décommenter.

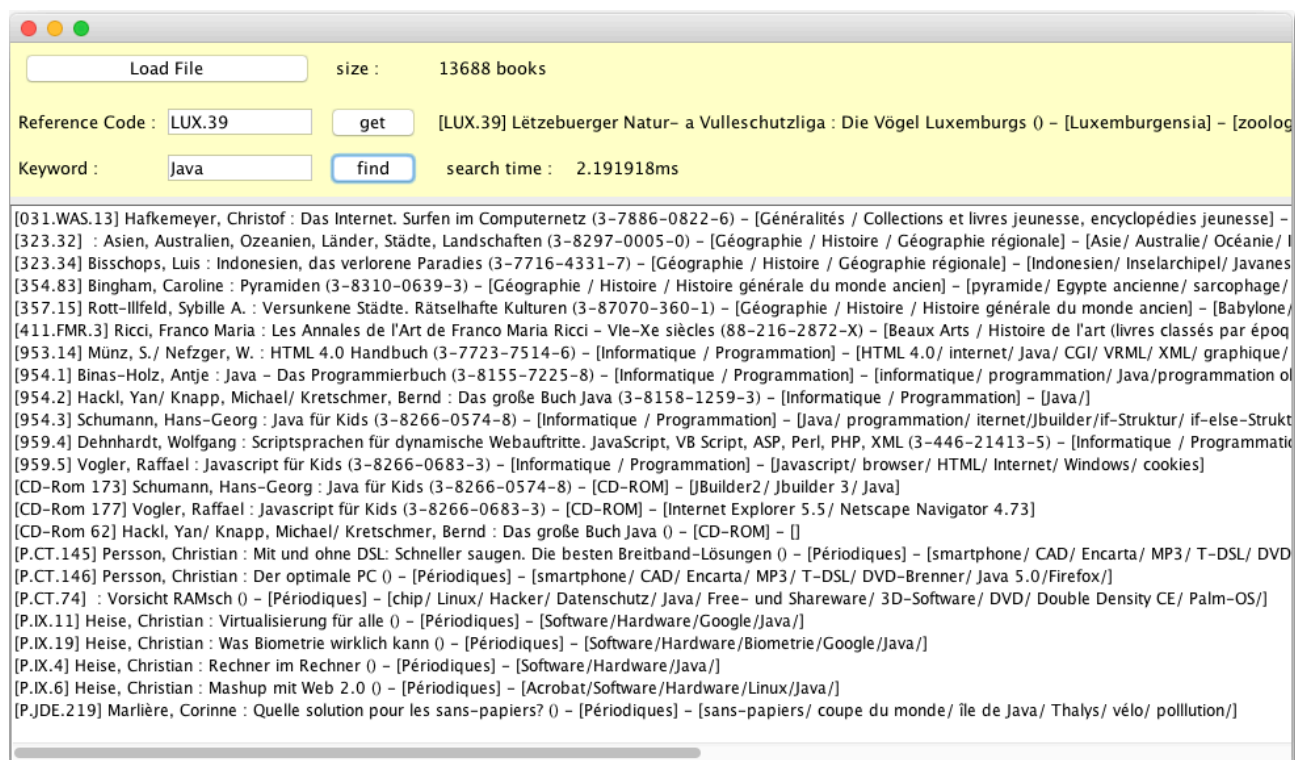
long find(String keyWord) : recherche tous les livres dont l'auteur, le titre, la catégorie ou les mots clés contiennent **keyWord** (en entier ou comme sous-chaîne). Les livres ainsi trouvés sont mémorisés dans la liste privée **alResult**. La méthode retourne la durée de la recherche en nanosecondes.

Object[] resultToArray() : retourne les livres contenus dans **alResult** de façon à ce qu'ils puissent facilement être affichés dans **resultList (JList)** de **MainFrame**.

IMPORTANT :

La méthode **find** doit être réalisée de façon à ce que tous les changements dans **alResult** soient immédiatement affichés dans **resultList** de **MainFrame**. Utilisez pour cela le schéma **Observer/Observable**. Modifiez **BooksBst** et **MainFrame** en conséquence.

Barème : add: 3p size: 1.5p loadFromFile: 2.5p get: 3p find: 9p



2. Flakes – fractales en mouvement (41 p.)

- ☞ Vous allez créer et réaliser le projet **Flakes** dans le dossier QUESTION2. Définissez d'abord les paquets nécessaires (voir UML). La vidéo **Flakes.mov** qui se trouve dans votre dossier vous sert de référence.

Il s'agit de développer une application qui dessine des fractales en forme de flocons de neige qui tombent et qui sont supprimées dès qu'elles sont sorties complètement en bas de la fenêtre. Les flocons de neige sont tous créés à l'aide de la même classe et de la même méthode. Les différentes apparences sont dues uniquement à des paramètres différents.

Chaque flocon (classe **Flake**) est composé d'un certain nombre de lignes (classe **Line**). L'ensemble des flocons est géré dans la classe **Flakes**. Un diagramme UML se trouve à la fin de l'énoncé.

Classe **Line** [6 points] :

- Les attributs **start** et **end** définissent les points de départ et d'arrivée de la ligne. L'attribut **length** détermine la longueur de la ligne. Ces points sont du type **Point2D** pour pouvoir mémoriser les coordonnées précises (type **Double**).
- Le constructeur calcule les coordonnées du point d'arrivée **end** à partir de coordonnées polaires données (point de départ **start**, angle de la ligne et longueur de la ligne).
- La méthode **move** déplace la ligne de **diffX** points en horizontale et **diffY** points en verticale.
- Veillez à ce que chaque ligne possède sa propre instance du point de départ, sinon la méthode **move** peut provoquer des effets indésirables.
- Lors du dessin, l'épaisseur de chaque ligne est d'un dixième de sa longueur. Utilisez la méthode **setStroke** de **Graphics2D** et la classe **BasicStroke**. (Consultez **JavaDoc** si nécessaire.)

Classe **Flake** [14 points] :

Les flocons ont un nombre défini de **branches** qui subdivisent le cercle en parties de même angle. P.ex. si **branches**=6, chaque subdivision a un angle de $360/6 = 60^\circ$ (donc $2 \cdot \pi / 6$ radians). Sur la pointe de chaque branche est placée récursivement un flocon avec la même couleur et le même nombre de branches, mais d'une taille réduite par le facteur **factor**. Il n'y a pas de flocons avec des branches d'une longueur inférieure ou égale à 5.

Chaque flocon (fractale) **Flake** est donc composé de lignes (classe **Line**) de même couleur **color**. Ces lignes sont mémorisées dans une liste **allLines** (**ArrayList**) et elles peuvent être redessinées au besoin. Les déplacements sont effectués en modifiant les coordonnées des lignes dans la liste. Ainsi les calculs récursifs sont effectués une seule fois pour chaque fractale : lors de leur construction.

- Le constructeur remplit **allLines** par toutes les lignes qui représentent le flocon. Pour cela, il utilise la méthode récursive **makeFlake**. Paramètres de **makeFlake** :
 - **p** : le centre du flocon
 - **branches** : le nombre de branches (subdivisions) du flocon
 - **angle** : angle de la première branche. A chaque niveau récursif, cet angle est modifié de la moitié de l'angle d'une subdivision (voir représentation et vidéo modèles).
 - **length** : la longueur des branches
 - **factor** : le facteur de réduction de la longueur des branches pour le prochain niveau récursif. Ce facteur a une valeur positive inférieure à 1.

- Chaque flocon possède un attribut **radius**, qui exprime la distance du centre jusqu'au point extérieur le plus éloigné du centre (voir aussi le schéma à la page suivante). La valeur de **radius** est calculée lors de la construction du flocon par la méthode **calculateRadius**. Le rayon reste inchangé pendant l'existence du flocon. (Idée : Profitez du fait que toutes les coordonnées se trouvent dans la liste).
- Le centre d'un flocon doit être le point de départ de la première ligne de la liste.
- La méthode **draw** dessine le flocon en utilisant sa couleur.
- La méthode **move** déplace le flocon de **diffX** points en horizontale et **diffY** points en verticale.

Classe **Flakes** [12 points]:

La classe **Flakes** organise les flocons dans une liste chaînée. Cette liste chaînée est à définir par vos soins, sans recourir aux structures prédéfinies.

- La méthode **add** ajoute un flocon à la fin de la liste.
- La méthode **draw** dessine tous les flocons.
- La méthode **move** déplace tous les flocons de **diffY** points vers le bas. Le déplacement horizontal est une valeur aléatoire entre **-diffY** et **+diffY**. Ce déplacement horizontal est différent pour chaque flocon.
- La méthode **cleanUp** supprime de la liste tous les flocons qui sont sortis entièrement en bas de la surface de dessin. Pour ce faire, elle a besoin de la hauteur de la surface de dessin, du centre et du rayon de chaque flocon.

ATTENTION : Cette opération n'est pas si triviale : dessinez un ou plusieurs schémas pour vous orienter. Il est recommandé d'effectuer la suppression des flocons en construisant une nouvelle liste contenant uniquement les flocons qui doivent rester dans la liste.

Classe **DrawPanel** [2 points] :

Cette classe est responsable pour le dessin de tous les flocons sur un fond noir. Pour améliorer le dessin, activez l'anti-alias par la commande suivante :

```
((Graphics2D)g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,  
                                RenderingHints.VALUE_ANTIALIAS_ON);
```

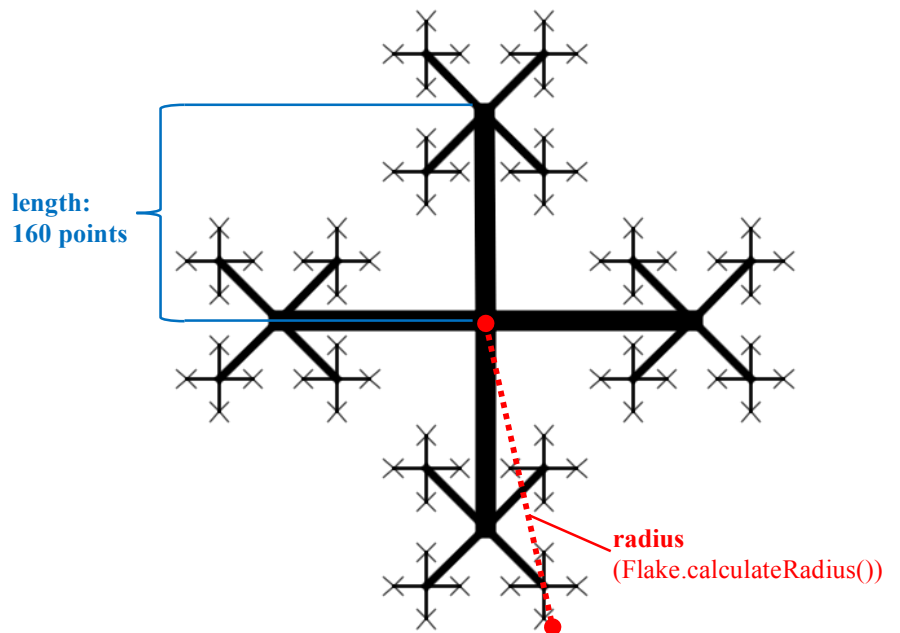
Classe **MainFrame** [7 points] :

- Les flocons sont déplacés et redessinés 10 fois par seconde. Le déplacement vertical est de 5 points vers le bas. Après chaque déplacement, les flocons qui sont sortis sont supprimés.
- Lors d'un clic de la souris avec le bouton droit, le flocon de démonstration est ajouté en blanc (voir représentation à la page suivante) en prenant comme centre la position de la souris.
- Lors d'un clic de la souris avec le bouton gauche, un flocon aléatoire est ajouté en prenant comme centre la position de la souris. Valeurs pour les paramètres aléatoires :
 - o la couleur est à moitié transparente et les valeurs aléatoires pour les composantes RGB varient individuellement entre 100 et 256
 - o le nombre de branches varie entre 3 et 11
 - o l'angle varie entre 0 et Pi
 - o la longueur des branches (1^{er} niveau) varie entre 20 et 60
 - o le facteur de réduction varie entre 0,3 et 0,55

Figure de démonstration et de test (clic droit), représentée ici en inversé (noir sur blanc).

Paramètres :

- branches : 4
- angle : 0
- longueur des branches (1^{er} niveau) : 160 points
- facteur : 0,4



UML :

