Outil de gestion d'images

Le projet devra être réalisé en Binôme (ou Trinôme après accord).
Les personnes seules qui n'arrivent pas à trouver de binôme s'en verront affecter un d'office par l'enseignant.
Les sources seront à remettre sous forme d'une archive zip à travers la plateforme communities. **Aucun rendu par mail ne sera accepté.**

▷ **Exercice 1** *Présentation générale du projet*

L'objectif du projet est de réaliser un outil de gestion de bibliothèque d'images simplifié avec des fonctions de sécurité. Pour cela, plusieurs fonctionnalités sont attendues dont des fonctions de bases :

- Mise en place d'opérations de traitement d'images.
- Gestion de tags pour identifier et chercher les images dans la bibliothèque.
- Mise en place d'une fonction de "chiffrement" d'image.

puis des fonctions un peu plus avancées pour les plus rapides d'entre vous :

- Mise en place d'un système de persistances des données (filtres + tags associés à une image).

Pensez à réaliser la conception générale de votre projet avant de vous lancer dans le développement. La qualité de la conception logicielle interviendra de manière importante dans la note.

A minima, vous devrez avoir une application respectant le `paradigme MVC` à la fin du projet.

▷ **Exercice 2** *Application réduite*

Dans cette première partie commencez par

- Essayer de lire une image et de l'afficher dans une fenêtre JavaFX. Pour cela, appuyez vous sur l'API de la classe `Image` et de la classe `ImageView`.
- Pour rendre votre projet plus lisible, créez dans le répertoire du projet, un nouveau répertoire nommé "resources" qui contiendra toutes les images à manipuler.
- ajoutez ensuite un `FileChooser` vous permettant de sélectionner l'image à afficher.

▷ **Exercice 3** *Les transformations d'image*

Dans cette partie, vous proposerez différentes transformations à appliquer aux images. Pour cela, on suppose que les images dont on dispose peuvent être representées via une matrice de pixels de dimension $m * n$. Chaque pixel a une couleur caractérisée

par trois composantes RGB (comprises entre 0 et 255) qui représentent chacune une couleur : rouge, vert et bleu. Appuyez vous une fois de plus sur la documentation du paquetage `javafx.scene.image` pour accèder aux pixels d'une image et comprendre comment manipuler les composantes RGB.

La première chose à mettre en place est un système de transformation permettant d'effectuer une rotation ou une symétrie d'image.

Dans un deuxième temps, vous proposerez un système de filtre pour traiter une image. Pour cela quelques contraintes de conception vous sont imposées. Vous devrez à minima :

- Utiliser la notion d'héritage et les interfaces pour éviter la redondance de code.
- Proposer des filtres permettant
  - d'échanger les composantes (R, G, B) pour obtenir (B, R, G)
  - d'afficher une image en noir et blanc (moyenne des composantes (R, G, B))
  - d'afficher une image en sepia (voir : `http://stackoverflow.com/questions/1061093/how-is-a-sepia-tone-created`)
  - de réaliser le filtre de Prewitt qui se rapproche du filtre de Sobel que vous avez vu dans un autre module pour effectuer de la détection de contours sur votre image.
- Intégrez ces filtres au sein de votre interface pour permettre à l'utilisateur de les appliquer sur une image de son choix

▷ **Exercice 4** *Support des tags et sauvegarde des transformations*

Cet exercice, bien que moins flashy que le précédent est très important dans la notation globale du projet.

Dans celui-ci, nous allons mettre au point un système permettant d'effectuer des recherches intuitives sur les images. En effet, même si les arborescences classiques restent le moyen privilégié pour stocker des informations sur un disque dur, elles ne sont souvent pas les plus adaptées pour retrouver rapidement de l'information. Pour cela nous allons proposer un système de Meta-données que nous stockerons dans des fichiers textes (version de base) ou des fichiers XML (version avancée).

Pour simplifier, toutes les métadonnées de la bibliothèque pourront être stockées dans le même fichier. La structuration du fichier est laissée à votre libre appréciation mais devra quand même s'appuyer sur les informations suivantes :

- Pour les tags, le fichier de métadonnée associé à une image contiendra une liste de tous les tags que l'utilisateur aura défini pour l'image.
- Pour les transformations, une liste des différentes transformations à appliquer (avec leurs paramètres) sera stockée. Cette liste permettra de réappliquer les transformations dans le même ordre au chargement de l'image.

La version texte est la plus rapide à mettre en oeuvre mais n'est pas la plus pratique à manipuler (en particulier pour le chargement et la création des objets). En général, on privilégie le format XML pour son expressivité. Son utilisation en Java est relativement simple et s'appuie sur JAXB (Java Architecture for XML Binding) qui est présente dans la JDK. Pour résumer de manière très rapide, cette bibliothèque permet d'annoter

une classe Java pour permettre de spécifier la structure du fichier XML assurant la persistance de cette classe. Un exemple très simple mais explicite peut être retrouvé à l'adresse suivante : `http://www.mkyong.com/java/jaxb-hello-world-example/`

Dans une version avancée, vous pouvez regarder le patron de conception DAO (Data Access Object) qui peut vous simplifier grandement la quantité de développement à réaliser.

▷ **Exercice 5** *Un peu de sécurité*

Pour que les étudiants intéressés par le master Cryptis ne se sentent pas floués par le début du projet, un fonction de sécurité supplémentaire sera ajoutée pour permettre de conserver des images de manière sécurisée.

Le concept est assez simple : l'idée est de mélanger les pixels d'une image dans un ordre prédicitif qui dépendra d'un mot de passe fourni par l'utilisateur.

Pour cela, nous nous appuierons sur l'objet `SecureRandom` fourni par la JDK. Comme dans de très nombreux langage, ce générateur de nombres aléatoires est controllé par une graine (seed) qui permet de générer des suites prédictibles de résultats. Ainsi, si vous utilisez deux fois de suite la même seed pour deux instances de générateurs de nombres aléatoires, ceux-ci vont effectuer les mêmes tirages. La graine est en général un nombre entier.

Pour passer d'un mot de passe fourni par l'utilisateur à un nombre entier servant à initialiser notre générateur de nombres aléatoires, une solution classique consiste à passer par une fonction de hachage. Ces fonctions standards en sécurité se retrouvent dans la JDK dans le paquetage `java.security`. Parmis toutes les fonctions proposées les plus classiques sont MD5 (considérée comme non sûre mais encore largement utilisée), SHA-1 (idem) et SHA-256. La classe `SecureRandom` permet de simplifier leur utilisation en spécifiant le type de fonction de hachage à utiliser pour générer les nombres aléatoires et en tolérant des seeds fournies sous forme de tableau d'octets.

D'un point de vue ergonomie ce filtre spécial modifiera directement l'image d'origine lorsqu'il sera appliqué et les méta données associées seront uniquement le nom du filtre. Bien entendu, le mot de passe pour déchiffrer l'image ne doit pas être conservé.

▷ **Exercice 6** *Les fonctions avancées*

Les exercices précédents vous donnent les fonctionnalités indispensables pour le projet mais vous pouvez aller plus loin (et donc avoir une meilleure note ;)).

Une première amélioration consiste à mettre en place un système de persistance des méta données dans une base de donnée plutôt que dans un fichier XML. Si vous avez bien conçu votre application cette modification ne devrait pas entrainer de changement important dans votre code. Utilisez une base H2 pour réaliser vos tests.

H2 est une base de données relationnelle de petite taille (1̃Mo) fournie avec la JDK qui utilise le langage SQL. Elle est très flexible pour du prototypage ou des tests au même

titre qu'Apache Derby ou HSQLDB par exemple.

La base de donnée H2 s'installe très simplement en dézippant une archive que vous pouvez retrouver sur le lien : `https://h2database.com/h2-2019-10-14.zip`. Sur le site principal vous pouvez retrouver les tutoriels nécessaires pour comprendre son utilisation. Un autre lien vous présentant un petit exemple est : `https://devstory.net/11895/installer-la-base-de-donnees-h2-et-utiliser-h2-console`.

Une fois la base de données installée, vous pouvez l'utiliser depuis votre application Java en utilisant la bibliothèque JDBC qui est fournie avec la JDK. Un petit cours présentant rapidement JDBC est disponible sur Communities et vous pouvez également retrouver sur le site de H2 des exemples de code permettant de l'utiliser avec la base de données : `https://www.h2database.com/html/tutorial.html#connecting_using_jdbc`

- Dans un terminal, rendez vous dans le répertoire : `/path/to/jdk/db/bin`
- Démarrez la base de données sur grâce au script `./startNetworkServer`
- Si tout se passe bien, la base de donnée est maintenant accessible sur l'adresse `http://localhost:1527/`
- Si cela ne fonctionne pas, vérifiez que vos variables `JAVA_HOME` et `JAVADB_HOME` sont bien positionnées.

Cette solution permet de démarrer une base de données externe, ce qui se rapproche de la réalité.