PROJET NFA032

FOD – Année 2018/2019 – Semestre 1

1. Enoncé

Le but de ce projet est d'écrire en Java, en utilisant les possibilités offertes par la programmation orientée objets, un programme permettant la manipulation d'images. Plus précisément, le programme devra permettre, étant donnée une image quelconque au format JPG ou PNG, d'effectuer différentes opérations sur celle-ci et de visualiser le résultat à l'écran. Les opérations que l'on effectuera consisteront à dessiner sur l'image des formes géométriques simples (trait, rectangle horizontal, cercle, triangle), d'épaisseur paramétrable, ainsi que différentes opérations de manipulation de l'image (rotation à 90°, réduction par 2 de la taille de l'image, extraction d'une partie de l'image.

Le projet comporte une version de base et des extensions possibles. Il sera impératif d'utiliser l'héritage. Une vidéo de présentation du projet est disponible sur votre espace moodle (lecnam.net).

2. Utilitaires et représentation des images en java

Deux classes sont mises à votre disposition afin d'afficher les images : la classe **`ImageUtil`** et la classe **`Afficheur`**. Il n'est pas nécessaire de comprendre leur fonctionnement pour réaliser le projet.

Une image est un tableau de points appelés pixels. Ce tableau comporte un certain nombre de lignes et de colonnes. Le nombre de lignes est appelé hauteur de l'image et le nombre de colonnes est appelé la largeur de l'image. Chaque pixel est un point coloré dont la couleur est définie par un mélange de rouge, de vert et de bleu. En plus de la couleur, chaque point d'une image a un niveau d'opacité : le paramètre alpha[1], qui sera dans ce projet toujours égal à 255.

En java, le paramètre alpha et chacune des composantes de couleurs rouge, verte et bleue prend une valeur comprise entre 0 et 255. Un pixel est ainsi caractérisé par 4 nombres tenant chacun sur un octet. L'ensemble des 4 nombres peut être stocké dans un entier qui comporte 4 octets.

Prenons un exemple : un point opaque noir. L'alpha a la valeur maximale 255. Ensuite, pour faire du noir, il ne faut mettre aucune couleur donc 0 de rouge, 0 de vert et 0 de bleu. Ces quatre nombre 255, 0, 0 et 0 peuvent être agglomérés dans un entier. Cet entier est : -16777216. Pour passer de l'entier aux 4 valeurs ou des 4 valeurs à l'entier, il y a un certain calcul à faire.

Une image entière est représentée par un tableau d'entiers. Mais alors qu'une image a deux dimensions (lignes et colonnes), le tableau d'`int` utilisé pour la coder n'a qu'une dimension (une rangée de cases). Dans ce tableau sont placées bout à bout d'abord la première ligne, puis la deuxième, puis la troisième etc.

Prenons l'exemple d'une image qui a une largeur 4 et un hauteur 3 et donnons des noms aux différents pixels.

[1] si le niveau d'opacité est élevé, le point est opaque, on ne voit pas ce qui est au-dessous. Si ce niveau est nul, le point est totalement transparent et si cette image est affichée sur un fond, un voit le fond et pas du tout la couleur du point. Pour les niveaux intermédiaires, on voit la couleur du point et un peu ce qu'il y a dessous par un phénomène de transparence. Le niveau d'opacité s'appelle le paramètre alpha. Dans ce projet, nous ne gérerons pas le paramètres alpha : tous les pixels seront complètement opaques (visibles), le paramètre alpha vaudra donc toujours sa valeur maximale : 255.

i = 0	p00	p10	p20	p30
i = 1	p01	p11	p21	p31
i = 2	p02	p12	p22	p32
	j = 0	j = 1	j = 2	j = 3

Cette image sera représentée par un tableau d'`int` :

p00	p10	p20	p30	p01	p11	p21	p31	p02	p12	p22	p32
0	1	2	3	4	5	6	7	8	9	10	11

Une classe appelée **`ImageUtil`** vous fournit un certain nombre de méthodes statiques qui vous permettent de faire certaines opérations[2] :

- **`ImageUtil.readImage(String fnam)`** renvoie le tableau représentant une image donnée par un fichier png ou jpg. Le paramètre `fnam` est le nom du fichier qui contient cette image.
- **`ImageUtil.getWidth(String fnam)`** renvoie la largeur de l'image contenue dans un fichier dont `fnam` est le nom.
- **`ImageUtil.getHeight(String fnam)`** renvoie la hauteur de l'image contenue dans un fichier dont `fnam` est le nom.
- **`ImageUtil.explodePixel(int pix)`** permet de décomposer un entier en 4 éléments : alpha, rouge, vert et bleu. Cette méthode renvoie un tableau de 4 `int`.
- **`ImageUtil.computePixel(int a, int r, int g, int b)`** fait l'opération inverse : elle calcule l'entier à partir de 4 `int` passés en paramètres.
- **`ImageUtil.computePixel(int[] pt)`** fait la même chose, mais les 4 composantes sont dans un tableau de 4 `int` passé en paramètre.
- **`ImageUtil.alphaFromPix(int pix)`** renvoie la composante alpha stockée dans un entier.
- **`ImageUtil.redFromPix(int pix)`** renvoie la composante rouge stockée dans un entier.
- **`ImageUtil.greenFromPix(int pix)`** renvoie la composante verte stockée dans un entier.
- **`ImageUtil.blueFromPix(int pix)`** renvoie la composante bleue stockée dans un entier.

Les méthodes de la classe **`ImageUtil`** font les calculs nécessaires pour coder et décoder les pixels. Il n'est pas nécessaire de connaître et comprendre ces calculs pour les utiliser.

La classe **`Afficheur`** quant à elle permet d'afficher à l'écran une image. Elle dispose de deux constructeurs, qui tous les deux prennent en paramètres un tableau d'int représentant une image (lue par exemple à l'aide des outils de la classe ImageUtil), la hauteur et la largeur de l'image, et, pour le deuxième constructeur, un nom pour la fenêtre d'affichage ainsi que les positions de la fenêtre d'affichage sur l'écran, dans un repère commençant au coin supérieur gauche de l'écran, l'axe des ordonnées étant vers le bas).

La classe **`Afficheur`** dispose également de la méthode **`update()`**, qui est à appeler après toute modification apportée sur une image pour que les modifications soient visibles à l'écran, et de la méthode **`masquer()`** qui permet de masquer l'affichage d'une image à l'écran. Par ailleurs la classe **`Afficheur`** est dotée de ce que l'on

[2] Vous n'aurez sûrement l'utilité que d'un certain nombre de ces méthodes

nomme un « écouteur » d'évènements (non étudié en cours) : lorsqu'une image est affichée, si vous cliquez dans l'image, les coordonnées du pixel cliqué apparaîtront dans la console.

3. Les fonctionnalités à mettre en œuvre

Partant d'une image initiale au format jpg, comme l'image **`parthenon.jpg`** fournie représentant un temple grec, votre programme devra proposer à l'utilisateur, dans la fenêtre de console, différentes fonctionnalités (sous forme d'un menu par exemple) :

Image initiale

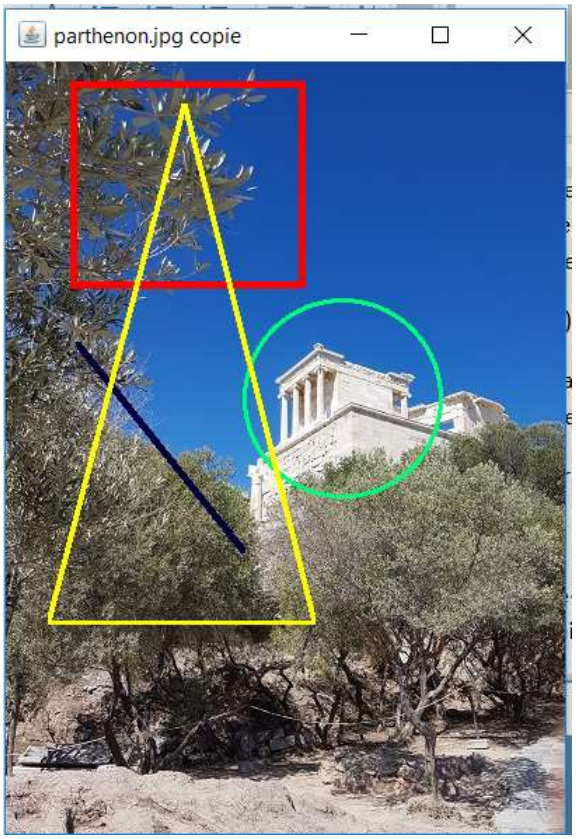
- Ajout sur l'image de formes géométriques pour mettre en évidence certains éléments de l'image :
  - **Ajout d'un rectangle horizontal creux** : vous devrez demander à l'utilisateur les caractéristiques d'un rectangle à ajouter dans l'image, à savoir :
    - Les coordonnées (numéros de lignes et de colonnes) du pixel correspondant au coin supérieur gauche du rectangle, sa hauteur (en nombre de pixels), sa largeur (en nombres de pixels
    - La couleur du rectangle
    - l'épaisseur (en nombre de pixels) du contour
  - **Ajout d'un trait,** compris entre deux pixels de l'écran dont on aura spécifié les coordonnées, d'une épaisseur donnée. Ce trait ne sera pas forcément horizontal ni vertical. Pour cela, vous pourrez par exemple déterminer l'équation de la droite passant par les deux pixels (dans un repère dont l'origine est le coin supérieur gauche de l'image, l'axe des x étant cette fois vertical vers le bas et l'axe des y horizontal vers la droite), puis déterminer pour chaque pixel de l'image si ses coordonnées (i,j) vérifient l'équation de la droite (à l'épaisseur du trait près).
  - **Ajout d'un triangle,** dont on aura fourni les coordonnées des trois points (pixels) et l'épaisseur.

- **Ajout d'un cercle,** dont on aura fourni les coordonnées du centre, le rayon et l'épaisseur. On pourra procéder de la même façon que pour le trait oblique : si les coordonnées du centre sont iC et jC, le rayon r, alors un pixel (i,j) est sur le cercle si ses coordonnées vérifient :

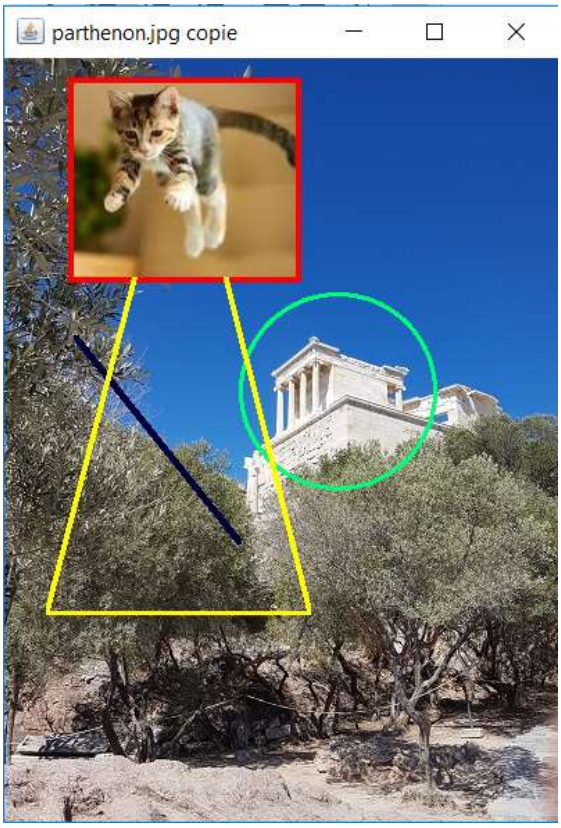
$$(iC-i)^2 + (jC-j)^2 = r^2$$

Là encore il faudra tenir compte de l'épaisseur du trait et ne pas vérifier l'égalité stricte par rapport à r^2, mais plutôt que le membre de gauche est égal à r^2 à l'épaisseur près.

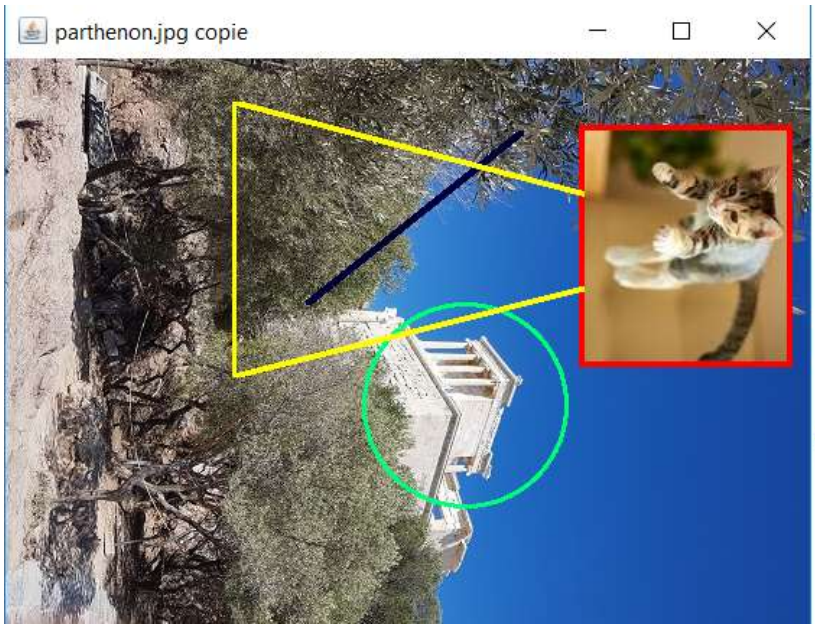
Exemple : image initiale à laquelle on a rajouté différentes formes de différentes épaisseurs :



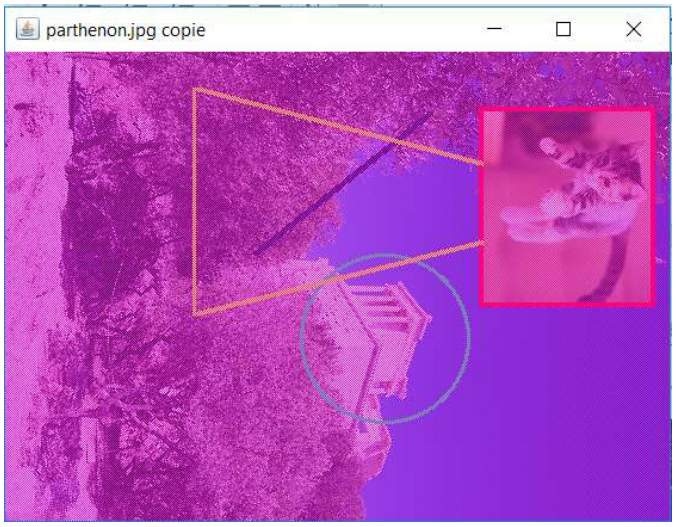
- **Incrustation d'une image dans une autre** : votre programme devra permettre d'incruster une partie d'autre image (par exemple l'image chaton.jpg fournie) dans l'image de départ, après avoir indiqué les caractéristiques d'un rectangle dans lequel l'image cible (le chaton) doit s'incruster :



- **Manipulation de l'image** : Votre programme devra proposer les opérations suivantes sur l'image :
  - **Rotation à 90°,** dans le sens des aiguilles d'une montre :



  - **Ajout d'un filtre de couleur à l'image,** en coloriant un pixel sur deux d'une certaine couleur, par exemple en violet :



- **Réduction par 2 de la taille de l'image** (en supprimant des pixels)
- **Extraction d'une partie de l'image (crop ou rogner),** en spécifiant un rectangle autour duquel on va découper l'image. Exemple, en repartant de l'image initiale dans laquelle on a incrusté l'image du chaton :

4. Variantes possibles

Vous pouvez faire évoluer librement ce projet au-delà de la version de base demandée, par exemple :

- en incrustant une autre image non plus à l'intérieur d'un rectangle, mais d'un cercle (ou d'un triangle)
- en transformant l'image en noir et blanc (en transformant pour chaque pixel son niveau de rouge, de vert et de bleu par la moyenne des trois niveaux).
- en proposant des formes pleines (avec contour et remplissage de différentes couleurs),
- en gérant l'annulation des différentes commandes (pour revenir en arrière),
- ou bien en essayant de mieux gérer l'interface graphique, par exemple en ajoutant à l'interface des boutons correspondant aux commandes. Ce dernier point n'a pas du tout été abordé en cours, il serait naturellement uniquement en bonus.

5. Qualités attendues pour votre programme

Il doit être écrit dans le style orienté objet, avec un découpage en classes rationnel. Les méthodes doivent être incluses dans les bonnes classes. Les méthodes ne doivent pas être trop longues. Les erreurs doivent être gérées avec des exceptions. Vous représenterez les différentes données au moyen de classes différentes **en utilisant l'héritage** ou des interfaces (éventuellement les deux).

Le programme doit être bien présenté, avec une indentation correcte.

Le programme doit montrer vos capacités à programmer. Pour cela, il faut privilégier les classes et méthodes que vous écrivez vous-même par rapport aux classes et méthodes de la librairie. Vous pouvez utiliser ArrayList si vous jugez cela utile.

Le programme devra être divisé en méthodes de taille raisonnable (maximum une page d'écran). Cela vaut aussi pour la méthode main qui ne doit pas être trop longue.

6. Consignes

Le projet est individuel : chaque auditeur doit le réaliser et doit en écrire seul chacune des lignes de code. Le projet donne lieu à une soutenance au cours de laquelle l'auditeur doit justifier les choix qu'il a fait.

Pendant la période de préparation du projet, vous pouvez demander l'aide de votre tuteur. Vous pouvez lui poser toutes les questions et lui soumettre tous vos problèmes.

7. Remise et soutenance du projet

Il y aura trois étapes dans l'évaluation du projet.

- ***Première étape*** : l'analyse. Vous devrez fournir la structure de votre programmes (classes et variables d'instance, entête des méthodes à écrire). Date limite : semaine du 7 janvier 2019.
- ***Deuxième étape*** : la remise du programme complet. Vous remettrez le code source d'un programme compilé et testé. Date limite : semaine du 11 février 2019.
- ***Troisième étape*** : la soutenance, en face-à-face avec votre tuteur. Il sera nécessaire de prendre un rendez-vous par mail, pour une soutenance également la semaine du 11 février 2019 ou la suivante. Pour les auditeurs inscrits dans un centre CNAM de province, la soutenance peut se faire en visio-conférence.