

# INF2010 – Structures de données et algorithmes

## Automne 2017

### Travail Pratique 1

### Héritage, polymorphisme et interfaces en Java

Objectifs :

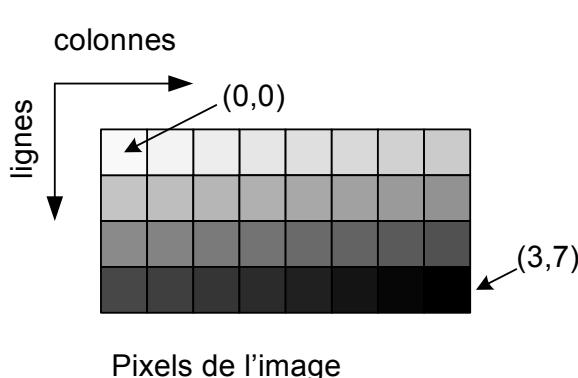
L'objectif visé par ce travail pratique est de vous familiariser avec les concepts orientés objets de la programmation en Java. Au terme de ce travail, vous devriez être en mesure de comprendre et de manipuler les éléments suivants :

- Classe
- Héritage
- Polymorphisme
- Classes abstraites
- Interfaces

### PROBLÈME :

Nous voulons disposer d'une librairie Java qui nous permette de manipuler des images. Ultimement, nous allons exploiter cette librairie pour sécuriser l'envoie de formulaire sur le web. Afin de simplifier notre tâche, nous utilisons le format de fichiers images dit PNM (portable anmap format). Il s'agit d'un format où les données de l'image sont transcris en ASCII dans un fichier texte. Le lecteur intéressé pourra se documenter sur le web sur le format PNM<sup>1</sup>. Il ne vous sera pas demandé d'écrire ou de lire ces fichiers, puisque le code nécessaire à cette tâche vous est fourni. Le code servant à afficher ces images vous est également fourni.

Une image peut être vue comme un tableau en deux dimensions. Chaque élément du tableau correspond à un pixel de l'image. Les coordonnées de l'image débutent en haut à gauche; le pixel à cette position est donné par les coordonnées (0,0).



entête {  
P2  
# Ceci est un commentaire  
8 4  
255  
0 8 16 24 32 40 48 56  
64 72 80 88 96 104 112 120  
128 136 144 152 160 168 176 184  
192 200 208 216 224 232 240 248

Fichier PNM (PGM en l'occurrence)

On considère quatre types d'images : noir et blanc (le pixel ne peut prendre que les valeurs 0/1 (nous utiliserons un booléen)), tons de gris (le pixel prend une valeur entre 0 et 255), couleurs (le pixel est représenté par un triplet d'entiers ( $r, g, b$ ) entre 0 et 255, le premier pour le rouge, le second pour le vert, le troisième pour le bleu) ou transparent (il définit une composante de plus que le pixel couleur pour représenter la transparence).

<sup>1</sup> Voir par exemple Wikipédia : [http://en.wikipedia.org/wiki/Netpbm\\_format](http://en.wikipedia.org/wiki/Netpbm_format).

## Exercice 1 : Classes abstraites, héritage et polymorphisme (1.5 pts)

Il vous est demandé de terminer l'implémentation des classes implémentant les quatre types de pixels considérés ici. Les classes associées aux pixels héritent de `AbstractPixel`, une classe abstraite qui vous est entièrement fournie. Les autres classes sont `BWPixel` (pour pixel noir et blanc), `GrayPixel` (pour pixel en tons de gris), `ColorPixel` (pour pixel en couleurs) et `TransparentPixel` (pour pixel transparent). La classe `BWPixel` est entièrement implémentée pour vous servir d'exemple. Il vous est demandé de compléter l'implémentation des trois autres classes. Pour ce travail, veillez à ce que vos classes respectent le protocole de la table suivante:

		Pixel de départ			
		BW	Gray	Color	Transparent
Converti en:	BW	-	$x \leq 127 \rightarrow 0$ $x > 127 \rightarrow 1$	moyenne( $r, g, b$ ) $\leq 127 \rightarrow 0$ moyenne( $r, g, b$ ) $> 127 \rightarrow 1$	moyenne( $r, g, b$ ) $\leq 127 \rightarrow 0$ moyenne( $r, g, b$ ) $> 127 \rightarrow 1$
	Gray	$0 \rightarrow 0;$ $1 \rightarrow 255$	-	moyenne( $r, g, b$ )	moyenne( $r, g, b$ )
	Color	$0 \rightarrow (0,0,0);$ $1 \rightarrow (255,255,255)$	$x \rightarrow (x, x, x)$	-	$(r,g,b,a) \rightarrow (r,g,b)$
	Transparent	$0 \rightarrow (0,0,0,255);$ $1 \rightarrow (255,255,255,255)$	$x \rightarrow (x, x, x, 255)$	$(r, g, b) \rightarrow (r, g, b, 255)$	-
Négatif		$0 \rightarrow 1;$ $1 \rightarrow 0$	$x \rightarrow 255 - x$	$(r, g, b) \rightarrow (255-r, 255-g, 255-b)$	$(r, g, b, a) \rightarrow (255-r, 255-g, 255-b, a)$

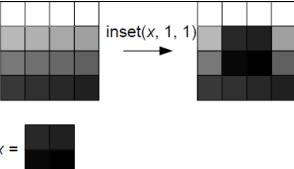
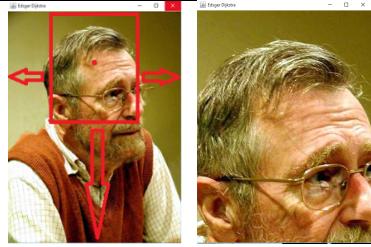
Il vous est également demandé de compléter l'implémentation de la classe image : `PixelMap`. Il s'agit principalement d'initialisation des données et de conversion de type. À la fin de ce travail, vous devriez être en mesure d'exécuter la partie `Exercice 1` du fichier principal `Main.java` (mettez la partie `Exercice 2` en commentaires) et obtenir les quatre figures suivantes :

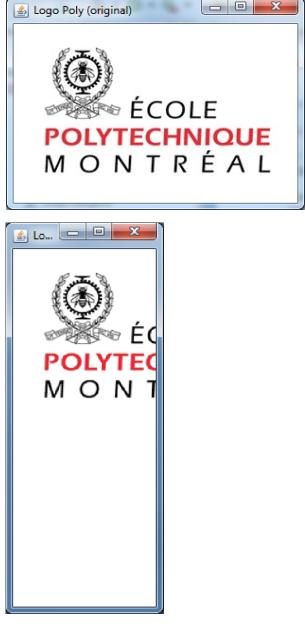


## Exercice 2 : Héritage et interfaces (2.5 pts)

Dans cette partie, il vous est demandé de compléter l'implémentation de la classe `PixelMapPlus` qui hérite de la classe `PixelMap` et qui implémente les méthodes de l'interface `ImageOperations`. Le tableau qui suit explique le fonctionnement attendu de certaines des méthodes de `ImageOperations`. Notez que l'implémentation de ces méthodes doit avoir une complexité asymptotique de  $O(n^2)$  au pire des cas.

Méthode	Description	Exemple
<code>resize(...)</code>	Redimensionne l'image aux tailles <code>h</code> et <code>w</code> . Les paramètres doivent être strictement positifs.	

		
inset(...)	Insère l'image donnée en paramètre (pm) à la position (row0, col0). Les pixels de l'image source qui dépassent dans l'image de destination ne sont pas recopierés.	 $x =$ 
rotate(...)	<p>Tourne l'image d'un certain angle autour d'un point. L'image tourne dans le sens des aiguilles d'une montre lorsque l'angle est positif et dans le sens inverse lorsque l'angle est négatif. Les positions vides doivent devenir des pixels blancs.</p> <p>Rappel: La rotation de <math>(x, y)</math> d'un angle <math>\theta</math> autour de <math>(a, b)</math> donne <math>(x', y')</math>:</p> $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & -\cos(\theta) \cdot a + \sin(\theta) \cdot b + a \\ \sin(\theta) & \cos(\theta) & -\sin(\theta) \cdot a - \cos(\theta) \cdot b + b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & -\cos(\theta) \cdot a - \sin(\theta) \cdot b + a \\ -\sin(\theta) & \cos(\theta) & \sin(\theta) \cdot a - \cos(\theta) \cdot b + b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ <p>Afin d'avoir le meilleur effet possible, parcourez tous les pixels de l'image cible et trouvez le pixel équivalent dans l'image source (en utilisant la 2<sup>ème</sup> matrice).</p>	 
zoomIn(...)	<p>Effectue un zoom uniforme autour d'un pixel <math>(x, y)</math> selon un certain facteur de zoom.</p> <p>Exemple : pour effectuer un zoom autour du point <math>(x,y)</math> avec un facteur de zoom <math>zoomFactor</math>, on élargie la partie de l'image centrée en <math>(x,y)</math> et qui est de largeur <math>width/zoomFactor</math> et de hauteur <math>height/zoomFactor</math>.</p> <p>Dans l'exemple à droite, le pixel autour duquel on effectue le zoom est <math>(x,y)=(width/2,height/4)</math> et <math>zoomFactor=2</math>.</p> <p>Au cas où le centre est l'un des bords de l'image, il faut changer le centre en sorte que l'image à zoomer est à l'intérieur de l'image originale.</p>	

crop(...)	<p>Découpe l'image pour qu'elle soit de hauteur <math>h</math> et de largeur <math>w</math>. Les paramètres doivent être strictement positifs. Si la nouvelle dimension est plus petite, les pixels dépassant sont perdus. Si au contraire la nouvelle dimension est plus grande, les nouveaux pixels sont blancs.</p>	
translate(...)	<p>Déplace l'image en <math>x</math> et <math>y</math>. La translation peut être négative. Les pixels non couverts sont blancs. Les pixels qui dépassent ne sont pas recopierés.</p>	
Inverser	<p>Inverse une image en mettant les pixels qui sont en haut en bas, et placer les pixels qui sont en bas de l'image en haut.</p>	

<pre>replaceColor(...)</pre>	<p>Remplace les pixels dont la couleur est entre « min » et « max », avec le nouveau pixel « newPixel ». Utilisez la méthode « compareTo » implémentée dans les classes qui héritent de « AbstractPixel ».</p> <p>Par exemple : On remplace tout les pixels rouges avec des pixels blancs.</p>	
------------------------------	--	---

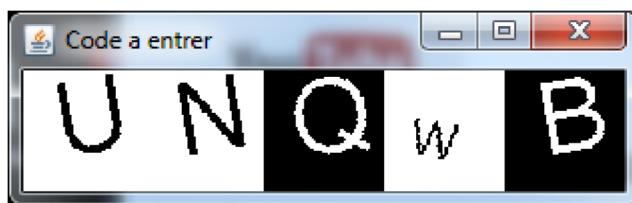
À la fin de ce travail, vous devriez être en mesure d'exécuter la partie **Exercice 2** du fichier principal `Main.java` et obtenir ceci :



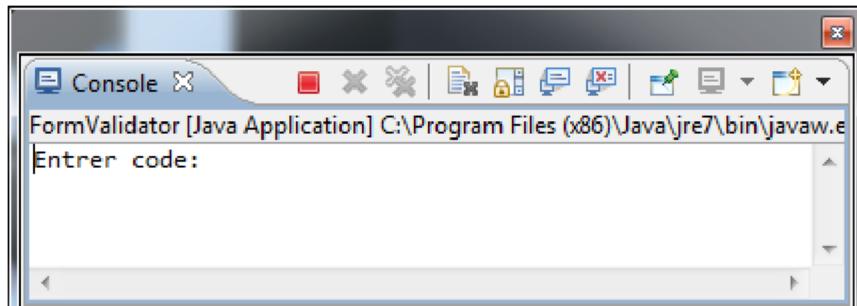
### Exercice 3 : Héritage et interfaces (1 pts)

La dernière partie de ce travail utilise votre librairie et une base de données de fichiers images pour chacune des lettres A à Z. Il vous est demandé d'implémenter deux fonctions aléatoires, l'une générant un code alphabétique (lettres en majuscules) d'une longueur maximale de 10 (la fonction « generateCode » dans la classe « FormValidator »). La deuxième fonction (« generateTransform » de la même classe « FormValidator ») génère un vecteur d'entiers entre 0 et 9 de la même longueur identifiant les transformations (que vous avez implanté dans l'exercice 2) à effectuer sur les lettres. À la fin de ce travail, vous devriez être en mesure d'exécuter correctement `FormValidator.java`. Ce qui suit illustre le fonctionnement attendu :

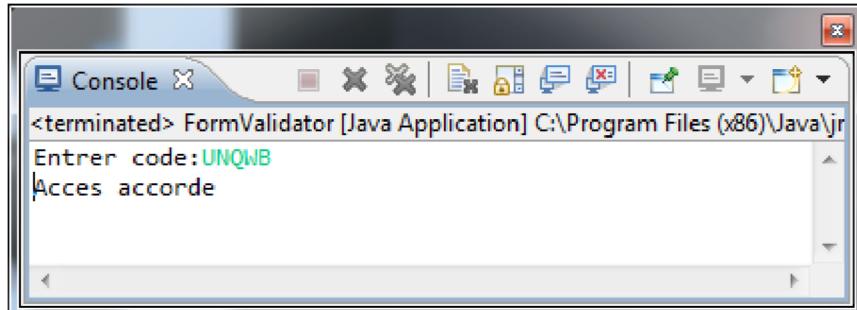
Étape 1 :



Étape 2 :



Étape 3 :



### Instructions pour la remise :

Le travail doit être fait par équipe de 2 personnes et doit être remis via Moodle au plus tard le :

- 18 Septembre avant 23h59 pour le groupe Mardi (B2).
- 24 Septembre avant 23h59 pour le groupe Lundi (B2).
- 25 Septembre avant 23h59 pour le groupe Mardi (B1).
- 01 Octobre avant 23h59 pour le groupe Lundi (B1).

Veuillez envoyer vos fichiers .java **seulement**, dans **un seul répertoire**, le tout dans une archive de type **\*.zip** (et seulement **zip**, pas de **rar**, **7z**, etc) qui portera le nom :

**inf2010\_lab1\_MatriculeX\_MatriculeY.zip**, où MatriculeX < MatriculeY.

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.