TP
Traitement d'image bmp

Bitmap, connu aussi sous son abréviation *BMP*, est un format d'image développé par Microsoft et IBM. C'est un des formats d'images les plus simples à développer et à utiliser pour programmer. Il est lisible par quasiment tous les éditeurs d'images.

A – Structure d'un fichier bmp

La structure du fichier est découpée en trois zones

- L'entête
- La palette de couleur
- Des données relatives à l'image

A-1 L'entête du fichier

Elle contient des données relatives au fichier :

Offset#	Taille	Valeur
0x0000	2 octets	• **BM** - Windows 3.1x, 95, NT, etc. • **BA** - OS/2 *Bitmap Array* • **CI** - OS/2 Icône Couleur (*Color Icon*) • **CP** - OS/2 Pointeur Couleur (*Color Pointer*) • **IC** - OS/2 Icône (*Icon*) • **PT** - OS/2 Pointeur (*Pointer*)
0x0002	4 octets	la taille du fichier BMP en octets
0x0006	2 octets	réservé pour l'identifiant de l'application qui a créé le fichier
0x0008	2 octets	réservé pour l'identifiant de l'application qui a créé le fichier
0x000A	4 octets	adresse de départ du contenu du BMP

Les données de l'entête qui nous intéresse ici sont la taille du fichier et l'adresse du départ du contenu BMP.

Dans ce TP, nous allons travailler sur des images bmp 24 bits. Pour ce type d'image, il n'y a pas de palettes de couleurs. On passe donc directement au contenu du fichier.

A-2 Le contenu du fichier BMP

Pour une image en BMP 24 bits, pour chaque pixels on a un octet bleu, un octet vert et un octet rouge.

Chaque pixels est donc codé sur 24 bits ou encore 3 octets.

Le contenu commence généralement dans ce cas à l'octet 54 (le premier octet étant l'octet 0)

Contrairement à la plupart des formats d'images, les pixels de l'image sont codés en partant de la ligne inférieure de l'image. Chaque ligne (codée de gauche à droite) doit toujours occuper un nombre d'octets multiple de 4. Si la ligne ne possède pas un nombre d'octets multiple de 4, on remplit la ligne en ajoutant des zéros.

A-3 Exercice

Ouvrir l'image « imageConteneur.bmp » à l'aide d'un éditeur hexadécimal. Cette image et retrouver les éléments suivants :

- La taille du fichier
- Le codage du premier pixel de l'image.
- Chaque ligne est-elle composée par un nombre de pixels multiple de 4

B – Charger une image bmp

L'objectif est de charger une image bmp 24 bits et de stocker les valeurs de chaque pixel dans un tableau que l'on pourra ensuite modifier..

Dans ce TP, on va utiliser plusieurs fois le chargement d'une image dans un tableau, on va donc créer une class qui s'en chargera des qu'on l'appellera.

Cette *class* est appelée ***FichierToTableauOctet*** et elle construit un *Objet* dont les attributs sont les suivants :

- *tabDonnees* : Tableau d'octet du fichier que la méthode va créer à partir du fichier.
- *nbOctetFichier* : Taille du fichier ainsi que du tableau d'octet.

Cette class a aussi 5 méthodes :

- *FichierToTableauOctet (String adresse)* : Elles à le même nom que la *class*, c'est le constructeur qui va fabriquer les attributs de cet *Objet*. Pour construire *tabDonnees*, elle va utiliser la méthode *ChargerTabDonnee(String adresse).* Celle-ci va créer le tableau d'octets à partir de l'adresse d'un fichier. *FichierToTableauOctet(String adresse)* doit aussi construire le réel nbOctetFichier. C'est ce qu'elle fait à l'aide de la méthode *NbOctetFichier*.
- Il reste deux méthodes, getTabDonnees() qui permet de récupérer le tableau de données via une autre class et getTailleTableau() qui renverra la taille du tableau.

Cette Class est donnée ci-dessous.

```
Import java.io.File ;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FichierToTableauOctet {

	private int[] tabDonnees;
	double nbOctetFichier;

	//Constructeur
	public FichierToTableauOctet(String adresse){
		tabDonnees = ChargerTabDonnees(adresse);
		nbOctetFichier = NbOctetFichier(adresse);
	}

	public int[] getTabDonnees(){
		return tabDonnees;
	}

	public double getTailleTableau(){
		return nbOctetFichier;
	}

	private int[] ChargerTabDonnees(String adresse){
		File fichier = new File(adresse);
		int nbOctetFichier = (int) fichier.length();
		int rgOctet = 0;
		int[] tab = new int[nbOctetFichier];
		try{
			FileInputStream source = new FileInputStream(adresse);
			try{
				for(rgOctet=0;rgOctet<nbOctetFichier;rgOctet++){
					tab[rgOctet]=source.read();
				}
			}
			finally{
				source.close();
			}
		}
		catch(FileNotFoundException ef){
			System.out.println("le fichier introuvable");
		}
		catch(IOException e){
			System.out.println(e+"erreur lors de la lecture du fichier");
		}
		return tab;
	}

	private double NbOctetFichier(String adresse){
		File fichier = new File(adresse);
		double nbOctetFichier = fichier.length();
		return nbOctetFichier;
	}
}
```

Pour pouvoir tester ce programme, on va devoir créer une autre class de type main pour lancer ce programme.

```
public class TraitementImage {

	public static void main(String[] args) {
		FichierToTableauOctet tableau = new FichierToTableauOctet("imageConteneur.bmp");
		int[] tableauImageConteneur = tableau.getTabDonnees();
		double tailleFichierConteneur = tableau.getTailleTableau();
		System.out.println("Programme terminé");
	}
}
```

Après avoir tape ce programme, vérifier s'il marche via un debug ou à l'aide de la commande System.out.println() pour voir si *tableauImageConteneur* contient bien les octets de l'image « imageConteneur.bmp »

Cette image devra se trouver à la racine du projet.

C – Créer une image bmp à partir d'un tableau d'octet

L'objectif est de créer une image bmp 24 bits à partir d'un tableau d'octets donnant les valeurs des couleur Rouge, bleu et vert de chaque pixel. La encore on va créer une class qui s'en chargera des qu'on l'appellera.

Cette *class* est appelée ***TableauOctetToFichier*** et elle crée et enregistre le fichier à l'adresse que l'on souhaite. En l'appelant, on devra lui envoyer une adresse et un tableau complet contenant l'entête du fichier ainsi que les couleurs de chaque pixel.

Cette class contient 1 seule méthode :

FichierToTableauOctet (int[] tabDonnees,String adresse)

Elle a le même nom que la *class*. c'est elle qui va fabriquer le fichier image à partir du tableau *tabDonnees*. Elle enregistrera ce fichier à l'adresse *adresse*.

Cette Class est donnée ci-dessous.

```
import java.io.FileOutputStream;
import java.io.IOException;

public class TableauOctetToFichier {

   public TableauOctetToFichier(int[] tabDonnees, String nouvelleAdresse){
             int tailleFichierModifier = tabDonnees.length;
       try {
             FileOutputStream fos=new FileOutputStream(nouvelleAdresse);
             try {
              int rgTab =0;
              while(rgTab<tailleFichierModifier){
                     fos.write(tabDonnees[rgTab]);
                     rgTab++;
              }
             }
             finally{
                 fos.close();
             }
        }
        catch(IOException e){
             System.out.println(e+"Erreur dans la création de la nouvelle image à
partir d'un tableau");
        }
   }
}
```

Pour pouvoir tester ce programme, on va devoir modifier la class de type main *CacherImage*.

On va donc charger un tableau d'octet à partir d'une image, on va ensuite modifier ce tableau d'octet puis créer un fichier bmp à partir de ce tableau.

```
public class TraitementImage {
        public static void main(String[] args) {
                FichierToTableauOctet tableau = new FichierToTableauOctet("imageConteneur.bmp");
                int[] tableauImageConteneur = tableau.getTabDonnees();
                double tailleFichierConteneur = tableau.getTailleTableau();
                int[] tableauImageNouvelle = new int[(int) tailleFichierConteneur];
            for( int i=0; i < 54; i++ ) {
                     tableauImageNouvelle[i] = tableauImageConteneur[i];
                }

            //On modifie le tableau contenant les octets de l'image conteneur
             for( int i=54; i < (int) tailleFichierConteneur; i+=3 ) {
               tableauImageNouvelle[i] = tableauImageConteneur[i] & 0x00;
               tableauImageNouvelle[i+1] = tableauImageConteneur[i+1] & 0xFF;
               tableauImageNouvelle[i+2] = tableauImageConteneur[i+2] & 0x00;
             }

             new TableauOctetToFichier(tableauImageNouvelle,"imageNouvelle.bmp");
             System.out.println("Programme terminé");
        }
}
```

Exercice

Pour vérifier si ce programme fonctionne, lancé le. Il doit vous créer un nouveau fichier à la racine du projet au nom « imageNouvelle.bmp » qui est différente de l'image.

Vous pouvez tester ce programme test avec d'autres types de modification.

Que ce passe-t-il si chaque couleur de chaque pixel de l'image *imageConteneur* est remplacée par la moyenne des trois couleurs.

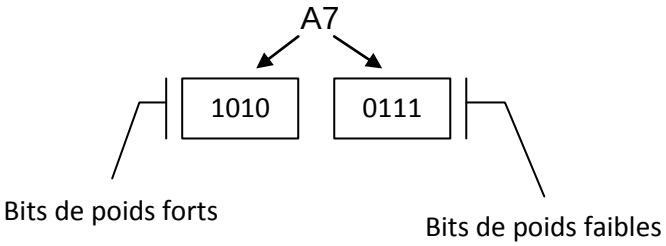
Modifier la Class *TraitementImage* pour voir le résultat.

D – Stéganographie

La stéganographie consiste à cacher une image dans une autre image.

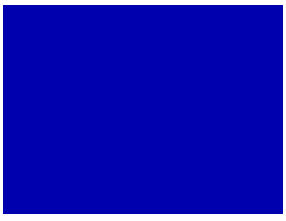
Dans une image bmp 24 bits, chaque couleur de base de chaque pixel est représentée par un octet. Cet octet a 4 bits de poids fort et quatre bits de poids faible.

En hexadécimal, A7 représente donc un octet.



Le nombre représenté en Hexadécimal par AF est donc proche de A0, et a l'œil nu, la différence de couleur est faible. En effet AF correspond au nombre entier 10x16+15=175 et A0 au nombre entier 10x16=160.

Bleu AF



Bleu A0

L'idée de la stéganographie est donc de créer une nouvelle image tells que les bits de poids forts de chaque pixels seront ceux de l'image conteneur, et les bits de poids faible seront les bits de poids forts de l'image à Cachée.

Image Conteneur

Image à cachée

Octet **A** 8

Octet **C** F

Octet **A C**

Image Nouvelle

L'image nouvelle ressemblera donc beaucoup à l'image conteneur, mais on verra en transparence une image cachée. Si l'image conteneur contient beaucoup de nuances de couleurs et de contraste, il sera plus difficile de voir l'image cachée.

On pourra récupérer l'image d'origine en récupérant les bits de poids faibles de l'image et en les décalant de 4 bits vers la gauche pour les placer dans les bits de poids forts.

Cela donnera l'image suivante :

Octet C0

On peut modifier la class TraitementImage ou en créer une nouvelle CacherImage comme suit pour cacher une image dans une autre.

La Class suivantes cache donc l'image « imageACacher.bmp » dans l'image « imageACacher.bmp »

```
public class CacherImage {

	public static void main(String[] args) {

		FichierToTableauOctet tableau = new FichierToTableauOctet("imageConteneur.bmp");
		int[] tableauImageConteneur = tableau.getTabDonnees();
		double tailleFichierConteneur = tableau.getTailleTableau();

		tableau = new FichierToTableauOctet("imageACacher.bmp");
		int[] tableauImageCachee = tableau.getTabDonnees();
		double tailleFichierImageCachee = tableau.getTailleTableau();

		int[] tableauImageNouvelle = new int[(int) tailleFichierConteneur];

		for( int i=0; i < 54; i++ ) {
			tableauImageNouvelle[i] = tableauImageConteneur[i];
		}
		for( int i=54; i < (int) tailleFichierConteneur; i++ ) {
		tableauImageConteneur[i] = tableauImageConteneur[i] & 0xF0;
		if(i<(int)tailleFichierImageCachee){
			tableauImageCachee[i] = tableauImageCachee[i] >> 4;
			tableauImageNouvelle[i] = tableauImageCachee[i] & 0x0F;
		}
		else{
			tableauImageNouvelle[i] = 0;
		}

		tableauImageNouvelle[i] = tableauImageConteneur[i] | tableauImageNouvelle[i];

	    }
	new TableauOctetToFichier(tableauImageNouvelle,"imageNouvelle.bmp");

	System.out.println("Programme terminé");
	}

}
```

Faire fonctionner le programme précédent et l'essayer avec d'autres image que celle du dossier projet. Attention, ces images devront être dans le format bmp 24 bits.

Exercice

Créer une Class *VoirImageCachee* qui va créer un fichier dans lequel on retrouvera l'image cachée dans les bits de poids faible d'une autre image.

Exercice complémentaire : Comment pourrait-on mieux cacher l'image de telle sorte qu'elle n'apparaisse plus en transparence ? Mettre en application cette idée.