Java2D-TD4

Images: introduction -Applets

1- Les images en Java

Le package **java.awt.image** contient les classes utilisées en java pour manipuler des images (allez voir les classes et interfaces de ce package : ils sont nombreux). Nous allons commencer par créer une image très simple : un dégradé linéaire du bleu au rouge. Voici la classe **GPix.java** qui réalise ceci :

```
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;
import java.lang.*;
import java.util.*;

public class GPix extends JFrame {

    private static GPix mon_objet;
    private static Image img;

    public static void main(String[] args ) {

        mon_objet = new GPix();
        mon_objet.init();

    }

    public void paint(Graphics g ) {

        Graphics2D g2D = (Graphics2D) g;
        g2D.drawImage(img, 0, 0, this);

    }

    public void init() {

        int w = 700;
        int h = 700;
        int pix[] = new int[w*h];
        int index =0;
        for( int y = 0; y < h; y++) {
        int red = (y*255)/(h-1);
        for( int x = 0; x < w; x++ ) {
        int blue = (x*255)/(w-1);
        pix[index++]= (255 << 24) | ( red << 16) | blue;
        }
        }
```

```
        img = createImage( new MemoryImageSource(w,h,pix,0,w));
        Toolkit theKit = mon_objet.getToolkit();
        Dimension wndSize = theKit.getScreenSize();
        mon_objet.setBounds(wndSize.width/6, wndSize.height/6, 700, 700);
        mon_objet.setVisible(true);
        }

}
```

La classe usuelle pour définir des images est **Image**. On utilise sa méthode **createImage()** pour définir une image. Ici nous utilisons la classe **MemoryImageSource** pour définir une image simple, faite d'un tableau de pixels. Chaque pixel est un entier, donc correspond à un mot de 32 bits, et il y a donc assez de place pour stocker sur 8 bits chaque composante rouge, verte et bleue d'un pixel. Dans ce programme nous définissons nous-même les valeurs des pixels. Chaque pixel est composé de 4 valeurs de 8 bits : l'octet le plus à gauche représente le canal alpha, ou transparence (quand il est à 0 le pixel est totalement opaque). Ensuite viennent dans l'ordre le rouge, le vert et le bleu. Que signifie l'instruction **(255 << 24)** dans le programme ci-dessus ? Que se passe-t-il si vous l'enlevez ? Remarquez que, pour afficher une image, vous faites appel à la méthode **drawImage()** du contexte graphique **g2D**. Consultez la documentation pour comprendre la signification des arguments.

Exercice : modifier le programme ci-dessus pour réaliser un dégradé linéaire du vert au bleu.

2- URLs : identification des sources sur le réseau.

Bien souvent, on va chercher des images (ou bien des fichiers) situées quelque part sur le net. Java est un langage qui vous permet d'accéder à des données distribuées sur le net. Ces sources de données sont identifiées par un **URL** : **Uniform Resource Locator**. Une source identifiée par un **URL** peut être un fichier situé sur un ordinateur distant, ou bien l'adresse d'une application, par exemple un moteur de recherche. Un **URL** identifie les données distantes en généralisant le mécanisme bien connu des noms de fichiers sur une machine locale. Un **URL** identifie l'ordinateur sur lequel se trouve la donnée, et il précise le type de protocole utilisé pour communiquer avec l'ordinateur distant: par exemple **ftp** ou **http**. Ce protocole est **http** quand vous téléchargez une page web depuis un serveur, **ftp** si vous téléchargez un simple fichier sur un serveur.

Un URL est fait de 4 champs, dont certains peuvent être omis parcequ'ils ont une valeur par défaut :

Protocole	Nom de domaine	Numéro de port	Chemin d'accès et nom de fichier
http://	**www.yahia.com**	**:80**	**index.html**
ftp://	**java.sun.com**	**:21**	

Le **nom de domaine** identifie l'ordinateur de façon unique sur le net. Le **numéro de port** désigne un type de service. Souvent, il n'est pas nécessaire de le préciser, car il est sous-entendu par défaut. Par exemple le port par défaut pour le protocole **http** est 80.

Nous pourrions aller telecharger depuis une classe java une image directement, mais il faut que l'image soit accessible sur un serveur et déclarée comme telle. Nous allons donc plutôt ramener sur notre disque local une image située quelque part sur le net, puis nous la lirons depuis notre disque local. Allez sur **www.yahia.com/shot.jpg** et téléchargez l'image sur votre disque, au meme endroit que les classes que vous écrivez dans ce TD. Vou pouvez également télécharger n'importe quelle image si vous en avez une en préférence.

3- Applets.

Il existe beaucoup de façons de créer des images en java, et une méthode, pariculièrement bien adaptée au téléchargement des images sur le net, consiste à utiliser les applets. Une applet est un programme java qui tourne dans le runtime environnement d'un serveur web. Pour exécuter une applet, il faut donc la charger dans un browser web et l'exécuter depuis le browser. Il y a une autre façon, fort utile lors du développement des applets, et qui consiste à utiliser le petit programme **appletviewer** fourni en standard avec le JDK java.

Une applet est définie par son cycle de vie : dès qu'une applet est chargée, elle exécute dans l'ordre ses méthodes suivantes:

init(), start(), paint(). Et chacune de ces méthodes est appelée quand il le faut. Par exemple, si vous changer la taille de la fenêtre du browser, les méthodes **start()** et **paint()** sont automatiquement appelées par le runtime environnement du browser sans que vous n'ayez rien à faire.

Une applet est un programme java qui répond à des critères de sécurité : par exemple une applet ne pourra jamais lire un fichier sur votre disque. La seule façon pour elle d'accéder à des données est d'utiliser des **URL**, et donc d'accéder à des fichiers déclarés accessibles par un **serveur** web.

Une applet se définit de la façon suivante : vous écrivez un programme java dont la classe principale étend **Applet** ou **JApplet**, vous redéfinissez celle des méthodes **init()**, **start()** et **paint()** qui vous intéresse, et vous écrivez un fichier html du même nom que la classe comme précisé ci-dessous.

L'avantage d'utikiser une applet pour afficher des images est que vous disposez de la méthode **getImage()**, dont l'argument peut d'ailleurs être une URL. Signalons que le constructeur d'une URL peut utiliser la méthode **getCodeBase()** comme indiqué ci-dessous, et qui retourne l'URL de la source où se trouve le fichier **.class** de l'Applet.

Le programme suivant (voir ici) définit une applet (grâce à l'implémentation swing des applets -**JApplet**-) qui ouvre une fenêtre, et affiche l'image **shot.jpg** téléchargée auparavant.

```
import java.awt.*;
import javax.swing.*;
import java.lang.*;
import java.util.*;
import java.net.*;
import java.awt.image.*;


public class GAffiche extends JApplet {

        private static GAffiche mon_objet;
        private static Image img;

        public void paint(Graphics g ) {

                Graphics2D g2D = (Graphics2D) g;
                g2D.drawImage(img, 0, 0, this);

        }

        public void init() {
```

```
        mon_objet = new GAffiche();

        try {
        img = getImage( new URL(getCodeBase(),"shot.jpg"));
        }

        catch( MalformedURLException e ){ System.out.println("Probleme
        avec l'image");}

        }

}
```

Ensuite, écrivez, au même endroit que la classe un fichier **GAffiche.html** qui contient la ligne suivante :

<applet code="GAffiche.class", width=400 height=400></applet>

Compilez la classe java puis exécutez en utilisant **appletviewer** de la façon suivante :

appletviewer GAffiche.html. Comparez avec les classes java que nous avons écrit jusqu'à présent. Notez qu'il n'y a pas de méthode **main().**