Animations : applet Neko

Introduction

Ce chapitre a pour but de montrer comment créer des séquences animées et surtout comment réduire l'effet de vacillement présent dans certaines animations. Pour créer une animation, on affichera une suite d'images à grande vitesse (méthode de création des GIFS animés). Pour cela, il faudra utiliser les threads (voir chapitre sur les Threads pour plus de renseignements), les méthodes de dessin Graphics de l'applet, les méthodes de l'applet (init(), update()…) et la mise en mémoire tampon.

Réduire des vacillements

Pour créer une animation, il suffit donc d'afficher une première image, de l'effacer, d'afficher une seconde, de l'effacer, d'en afficher une troisième et de l'effacer et ainsi de suite à très grande vitesse. Se pose alors le problème du rafraîchissement de l'écran : Java va redessiner toute la zone de l'applet, ce qui nous donne une impression d'affichage saccadé. Pour résoudre ce problème, on peut avoir recours à deux méthodes : utiliser la méthode Repaint() (synthèse de paint() et de update()) en le redéfinissant pour qu'elle n'efface que les zones concernées par le changement. Cette solution est lourde et incommode. La seconde solution consiste à redéfinir paint() et update() en utilisant une double mise en mémoire tampon (Double-buffering), les images et seulement les images seront stockées en mémoire tampon, ensuite on affichera cette mémoire tampon (donc seulement les images et pas le reste du cadre).

1° méthode

A tout hasard, voici une redéfinition de update() :

```
public void update(Graphics g) {
    g.setColor(getBackground()) ;
    g.fillRect(0,0,size().width, size().height);
    g.setColor(getForeground());
    paint(g);
}

public void update(Graphics screen) { // utilisation de la méthode précédemment définie
    paint(screen);
}
```

2° méthode

1. *Créer des variables d'instance pour stocker les images*
2. *Créer les images et le contexte graphique au moment d'initialisation de l'applet*
3. *Effectuer les opérations de dessin dans le tampon*
4. *Dessiner le tampon à l'écran à la fin de la méthode paint()*

```
// 1
Images offScrenImage ;
Graphics offScreen ;
// 2
offScreenImage = createImage(size().width, size().height);
offscreen = offscreenImage.getGraphics();
// 3
offscreen.drawImage(bug, 10, 10, this);
screen.drawImage(offscrenImage, 0, 0, this);
// 4
public void update(Graphics g) {
    paint(g) ;
}
```

Application : applet Neko

L'applet Neko est issue de JAVA2 le programmeur par Roger Cadenhead et Laura Lemay aux éditions Campus Press, elle représente un petit chat parcourant la fenêtre de l'applet. La technique utilisée est celle des threads. La technique du programme est simple : pour chaque action du chat on écrit une méthode qui appelle l'image correspondante un certain nombre de fois. On spécifier ensuite ce nombre en fonction de la durée de l'animation.

```
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Color;

public class Neko extends java.applet.Applet
    implements Runnable { // necessaire pour les threads

    Image nekoPics[] = new Image[9]; // matrice d'images
    Image currentImg; // image courante
    Thread runner; // déclaration du thread
    int x; // abscisse de l'image
    int y = 50; // son ordonnée

    public void init() { // initialisation de l'applet
        String nekoSrc[] = { "right1.gif", "right2.gif",
            "stop.gif", "yawn.gif", "scratch1.gif",
            "scratch2.gif","sleep1.gif", "sleep2.gif",
            "awake.gif" }; // on remplie la matrice avec les images

        for (int i=0; i < nekoPics.length; i++) {
            nekoPics[i] = getImage(getCodeBase(),
                "images/" + nekoSrc[i]);
        }
    }

    public void start() { //méthode start de l'applet, opérations sur les threads
        if (runner == null) {
            runner = new Thread(this);
            runner.start();
        }
    }

    public void stop() { // comment arreter l'animation
        runner = null;
    }

    public void run() { // lancer l'applet
        setBackground(Color.white);
        // part d'un côté de l'écran pour la milieu
        nekoRun(0, size().width / 2);
        // pause
        currentImg = nekoPics[2];
        repaint();
        pause(1000);
        // baille 3 fois
        currentImg = nekoPics[3];
        repaint();
        pause(1000);
        // se gratte 4 fois
        nekoScratch(4);
        // dort 5 "temps"
        nekoSleep(5);
        // se réveille et s'en va
        currentImg = nekoPics[8];
        repaint();
        pause(500);
        nekoRun(x, size().width + 10);
    }

    void nekoRun(int start, int end) { // méthode qui permet d'exécuter les
//mouvements
                                       // définis précédemment ici: courir et bailler)
        for (int i = start; i < end; i += 10) {
            x = i;
```

```
            // images pour le bâillement
            if (currentImg == nekoPics[0])
                currentImg = nekoPics[1];
            else currentImg = nekoPics[0];
            repaint();
            pause(150);
        }
    }

    void nekoScratch(int numTimes) { // pour se gratter...
        for (int i = numTimes; i > 0; i--) {
            currentImg = nekoPics[4];
            repaint();
            pause(150);
            currentImg = nekoPics[5];
            repaint();
            pause(150);
        }
    }

    void nekoSleep(int numTimes) { // ... dormir
        for (int i = numTimes; i > 0; i--) {
            currentImg = nekoPics[6];
            repaint();
            pause(250);
            currentImg = nekoPics[7];
            repaint();
            pause(250);
        }
    }

    void pause(int time) { // une pause dans l'animation
        try {
            Thread.sleep(time);
        } catch (InterruptedException e) { }
    }

    public void paint(Graphics screen) { // affichage de l'animation
        if (currentImg != null)
            screen.drawImage(currentImg, x, y, this);
    }
}
```

```
<applet code="Neko.class" width=300 height=200>
</applet>
```