Processus

Table des matières

Créer des processus

La classe Thread

La classe Thread (contenue dans le package java.lang) permet de créer des processus. Ceux-ci sont déclenchés par leur méthode **start()** et stoppés par leur méthode **stop()**.
La méthode start() appelle la méthode **run()** que nous devons redéfinir dans les classes dérivées.

Exemple de classe dérivée de Thread

Nous allons créer un processus dont la tâche sera simplment d'afficher son nom. Pour cela il nous suffira de définir un constructeur et la méthode run().

```
public class ThreadAff extends Thread {

 public ThreadAff(String nom) {
  super(nom);
 }

 //afficher 10 fois le nom
 public void run() {
  for (int i=0; i<10; i++) {
   //effectue la tâche prévue
   System.out.println("Je suis le processus "+getName());
   //passe la main au processus suivant
   Thread.yield();
  }
 }

}
```

Noter l'utilisation de la méthode statique yield() qui permet aux autres processus de s'exécuter.

Mise en oeuvre dans un programme

Ecrivons un programme utilisant la classe ThreadAff que nous venons de définir. Pour créer 2 processus de type ThreadAff il suffit d'utiliser deux variables t1 et t2.

```
public class TestThread {

 public static void main(String args[]) {
  ThreadAff t1=new ThreadAff("1");
  ThreadAff t2=new ThreadAff("2");
  t1.start();
  t2.start();
  for (int i=0; i<15; i++) {
   System.out.println("Je suis la tâche principale.");
   Thread.yield();
  }
 }

}
```

En exécutant ce programme on voit trois processus agir : le processus principal de l'application et les deux processus créés.

L'interface Runnable

Les applets peuvent utiliser des processus en implémentant l'interface Runnable qui demande une méthode **run**. Le processus doit être associé à l'applet par son constructeur, sa méthode start appelle alors la méthode run de l'applet.

Squelette d'applet implémentant l'interface Runnable

L'applet gère un processus à travers une variable de type Thread. Sa méthode start, appelée juste après init, crée le processus et le démarre, ce qui provoque l'exécution de la méthode run. Celle-ci est formée d'une boucle dans laquelle se fait le travail du processus, avec à chaque tour un arrêt (méthode **wait**) permettant au processus principal de l'applet de s'exécuter pour la gestion des évènements (affichage, souris, boutons, etc...). Une variable **speed** permet de gérer le temps d'arrêt qui est exprimé en millièmes de secondes. Enfin nous utilisons la méthode stop de l'applet, qui est appelée lors de la fermeture de la page html, pour stopper le processus.

```
import java.awt.*;
import java.applet.*;

public class MaClasse extends Applet
 implements Runnable {

 private Thread processus=null;
 int speed=20;

 public void init() {
 }

 public void start() {
  if (processus==null) {
   processus=new Thread(this);
   processus.start();
  }
 }

 public void stop() {
  processus.stop();
  processus=null;
 }

 public void run() {
  while (processus!=null) {
   repaint();
   //modifications du dessin
   try {
    processus.sleep(speed);
   } catch (InterruptedException e){}
  }
  processus=null;
 }
```

```
}
```

Création d'un chronomètre

A titre d'exemple, créons une applet qui affiche un chronomètre en utilisant le modèle précédent.

Comme le chronomètre affiche les dixièmes de secondes, la variable speed est fixée à 100.

Une variable **ds** compte les dixièmes de secondes. Le travail du processus se limite donc à incrémenter cette variable et à réafficher le chronomètre.

```
import java.awt.*;
import java.applet.*;

public class Chrono1 extends Applet
 implements Runnable {

 private Thread chronometre;
 int speed=100;
 private int ds=0;  //dixième de seconde

 public void init() {
  setBackground(Color.white);
  setForeground(Color.blue);
  setFont(new Font("SansSerif",Font.BOLD, getSize().height));
 }

 public void start () {
  // Au début de l'applet, création et démarrage du chronomètre
  if (chronometre==null) {
   chronometre=new Thread(this);
   chronometre.start();
  }
 }

 public void run () {
  while (chronometre!=null) {
   repaint ();
   ds++;
   try {
    chronometre.sleep(speed);
   } catch (InterruptedException e) {}
  }
 }

 public void stop () {
  // A la fin de l'applet, arrêt du chronometre
  chronometre.stop();
  chronometre=null;
 }

 public void paint (Graphics g) {
  // Dessine le temps écoulé sous forme de h:mm:ss:d
  String S=ds/36000 +":"+(ds/6000)%6+(ds/600)%10
    +":"+(ds/100)%6+(ds/10)%10+ ":" + ds%10;
  //affichage centré
  FontMetrics fm=g.getFontMetrics();
```

```
  int largeur=fm.stringWidth(S);
  int hauteur=fm.getHeight();
  int x=(getSize().width-largeur)/2;
  int y=(getSize().height-hauteur)/2;
  g.drawString(S, x, y+hauteur-fm.getDescent());
 }
}
```

En utilisant le modèle fourni précédemment, il nous a suffi d'écrire les méthodes init (choix des couleurs) et paint (affichage), puis de compléter la méthode run pour incrémenter le nombre de secondes.

Texte défilant

Nous allons créer l'applet DefilText qui fait défiler un texte passé en paramètre dans le fichier html.

Nous utiliserons le double buffering pour obtenir un affichage sans clignotements.

Déclarations

Commençons par déclarer les variables dont nous aurons besoin pour travailler.

```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;

public class DefilText extends Applet
 implements Runnable {

 String msg=null;          // message affiché
 Thread processus = null;  // le processus
 int x_coord=0;            // coordonnées d'affichage
 int y_coord=0;
 int len=0;                // longueur du message
 int speed=20;             // temps de pause
 //pour le double buffering
 Image img=null;           // image en mémoire
 Graphics gi=null;         // contexte graphique pour l'image
```

Méthode init

Nous nous contenterons de récupérer le texte à afficher et de définir les couleurs utilisées.

```
 public void init() {
  msg=getParameter("text");
  if (msg==null) msg="Texte défilant";
  setFont(new Font("TimesRoman",Font.BOLD,36));
  setBackground(Color.white);
  setForeground(Color.red);
 }
```

Gestion du processus

Nous reprenons le modèle vu dans l'activité précédente. La méthode run fera appel à la méthode faireImage que nous écrirons ensuite.

```
public void start() {
 if (processus==null) {
  processus=new Thread(this);
  processus.start();
 }
}

public void stop() {
 if (processus!=null) processus.stop();
 processus=null;
}

public void run() {
 while (processus!=null) {
  faireImage();
  repaint();
  x_coord--;
  if (x_coord<=-len) x_coord=getSize().width;
  try {
   processus.sleep(speed);
  } catch (InterruptedException e){}
 }
 processus=null;
}
```

L'instruction `x_coord--` provoque le défilement en modifiant l'abscisse de la position d'affichage.

La ligne `if (x_coord<=-len) x_coord=getSize().width;` permet de reprendre l'affichage à son départ après un passage complet.

La méthode faireImage

Au premier appel cette méthode initialise l'image et les différentes variables concernant la position du texte. Elle remplit ensuite l'image avec la couleur de fond et affiche le texte avec la couleur d'affichage.

```
public void faireImage() {
 if (len==0) { //calcul des mesures du texte
  img=createImage(getSize().width, getSize().height);
  gi=img.getGraphics();
  FontMetrics fm=gi.getFontMetrics();
  len=fm.stringWidth(msg);
  x_coord=getSize().width;
  y_coord=(getSize().height-fm.getHeight())/2+fm.getAscent();
 }
 gi.setColor(getBackground());
 gi.fillRect(0,0,getSize().width,getSize().height);
 gi.setColor(getForeground());
 gi.drawString(msg,x_coord,y_coord);
}
```

Méthodes paint et update

La méthode paint se contente d'afficher l'image. La méthode update est réécrite pour éviter l'effacement de l'applet qui provoque un scintillement.

```
public void paint(Graphics g) {
 if (img!=null) g.drawImage(img,0,0,this);
}

public void update(Graphics g) {
 paint(g);
}
```

Exercices

1. Chronomètre
   Compléter l'applet chronomètre en provoquant son arrêt et son redémarrage à chaque clic.

2. Image déformée
   Affichage d'une image qui se déforme de façon continue.

3. Effets de couleurs
   Changements continus de couleur.

4. Couleurs et mouvements
   Un texte dont chaque lettre bouge et change de couleur.

5. Gifs animés
   Animations créées avec une succession d'images.