Chapitre 5

Ce chapitre illustre les techniques permettant de développer les applets. Une applet est un programme type *thread* interprété par un navigateur WEB.

Nous allons étudier comment:

- dessiner une image
- organiser le captage des événements générés par le clavier et la souris
- récupérer les arguments externes
- ajouter des boutons
- créer une animation dans un *thread* et jouer des clips audio

L'applet la plus simple affiche une ligne du texte:

```
import java.applet.*;     // package avec les classes pour la creation des applets
import java.awt.*;        // package pour la visualisation (graphique) des applets


public class AppletUn extends Applet {
public void paint(Graphics g) {                 // cette methode affiche applet
g.drawString("Bonjour",20,30); }
}
```

Dans l'applet ci-dessus il n'a y aucun traitement. L'introduction d'un traitement initial et unique passe par la méthode *init()*; la méthode *paint()* peut être exécutée plusieurs fois.

```
import java.applet.*;     // package avec les classes pour la creation des applets
import java.awt.*;        // package pour la visualisation (graphique) des applets
public class AppletDeux extends Applet {
static final String texte = "Bonjour";
private Font font;
public void init() {            // ici demarre l'execution initiale de l'applet
font = new Font("Courier",Font.BOLD, 36);
}
public void paint(Graphics g)                // paint()  affiche applet
{ g.setColor(Color.yellow);
  g.fillOval(20,20,200,200);
  g.setColor(Color.black);
  g.setFont(font);
  g.drawString(texte,40,120); }
}
```

Captage des événements

Les événements générés par la souris peuvent être captés par l'applet grâce aux méthodes pré-définies dans le package *java.awt*. Par exemple les méthodes *mouseDown()* et *mouseDrag()* détectent les

événements associés à l'appui d'un bouton et au déplacement de la souris. Si un événement est effectivement détecté, les méthodes retournent la valeur booléenne *true*.

```
import java.applet.*;     // package avec les classes pour la creation des applets
import java.awt.*;        // package pour la visualisation (graphique) des applets


public class AppletTrois extends Applet {
private int dernier_x = 0;
private int dernier_y = 0;


public void init() {
this.setBackground(Color.yellow);  // initialisation de la couleur du fond
}


public boolean mouseDown(Event e, int x, int y) {
dernier_x = x;
dernier_y = y;
return true;
}


public boolean mouseDrag(Event e, int x, int y) {
Graphics g = getGraphics();
g.setColor(Color.red);
g.drawLine(dernier_x,dernier_y,x,y);
dernier_x = x;
dernier_y = y;
return true;
}
}
```

Paramètres d'une applet

Une applet peut récupérer ses arguments (*parameters*) à partir de la page WEB dans laquelle elle est integrée. Le texte suivant (un fichier *html*) montre comment saisir les paramètres d'une applet.

```
<html>
<head>
<title>Test de l'applet avec parametres</title>
</head>
<body>
Dessiner dans l'applet:
<hr>
<APPLET CODE=AppletQuatre.class WIDTH=400 HEIGHT=400>   // nom et taille de l'applet
<param name="cfond" value="00AAFF">
<param name="csurf" value="000000">
```

```
</APPLET>
</body>
</html>
```

Les paramètres à lire sont récupérés par la méthode *init()*.

```
import java.applet.*;     // package avec les classes pour la creation des applets
import java.awt.*;        // package pour la visualisation (graphique) des applets
public class AppletQuatre extends Applet{
private int dernier_x = 0; private int dernier_y = 0;

public void init() {
Color cfond = getColorPar("cfond");  // appel a  la methode getColorPar ci-dessous
Color csurf = getColorPar("csurf");     // getColorPar transforme une couleur donnee en hexa en un entier
if(csurf != null) this.setForeground(csurf);
if(cfond != null) this.setBackground(cfond);
}
protected Color getColorPar(String nom) {
String valeur=this.getParameter(nom);  // lecture d'un parametre
int intvaleur;
try { intvaleur= Integer.parseInt(valeur,16); }  // conversion hexa => decimal
catch(NumberFormatException e) { return null; }
return new Color(intvaleur);  // creation de la nouvelle couleur
}
public boolean mouseDown(Event e, int x, int y) {
dernier_x = x;
dernier_y = y;
return true;
}
public boolean mouseDrag(Event e, int x, int y) {
Graphics g = getGraphics();
g.drawLine(dernier_x,dernier_y,x,y);
dernier_x = x;
dernier_y = y;
return true;
}
}
```

L'applet suivante est une extension de l'applet précédente. Dans cet exemple nous ajoutons un bouton permettant d'enclencher une action. Cette action efface le contenu graphique de l'applet. Initialement l'applet est vue comme un rectangle de la couleur du fond. L'opération de l'effacement consiste à dessiner sur toute la surface de l'applet la couleur du fond.

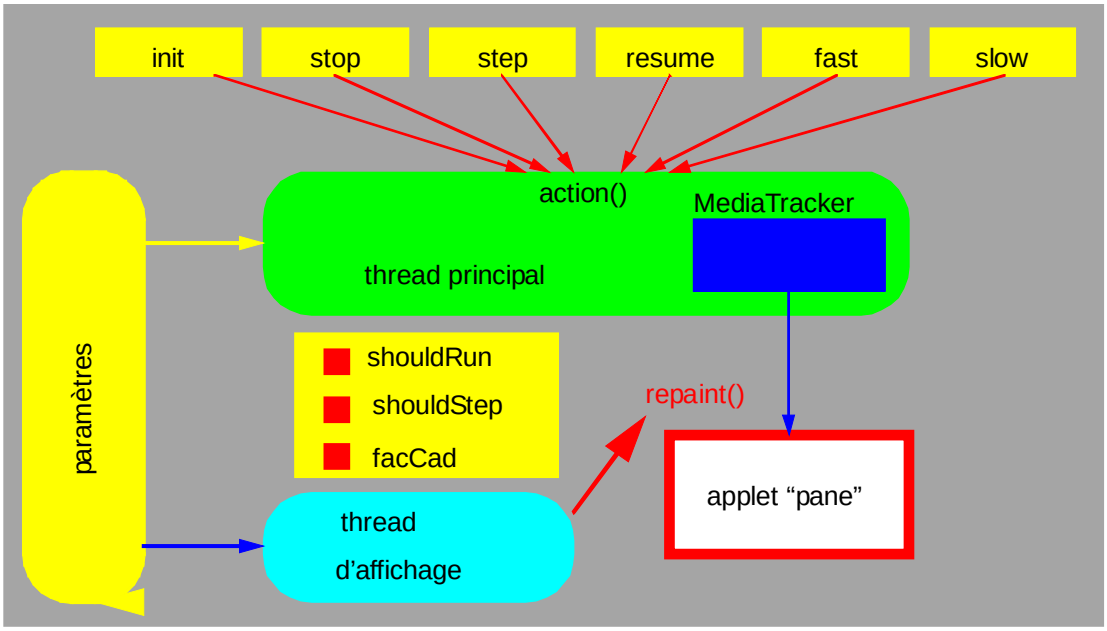
Attention: Cette opération de l'effacement peut durer plusieurs secondes.

```
import java.applet.*;     // package avec les classes pour la creation des applets
import java.awt.*;        // package pour la visualisation (graphique) des applets

public class AppletCinq extends AppletQuatre{
private Button effacer;

public void init() {
super.init();          // integrer la méthode init()  de la classe superieure
effacer = new Button("effacer");
effacer.setForeground(Color.black);
effacer.setBackground(Color.yellow);
this.add(effacer);
}

public boolean action(Event event,Object obj) {     // gestion des evenements
if(event.target == effacer)
 {
 Graphics g = this.getGraphics();
 Rectangle r = this.bounds();
 g.setColor(this.getBackground());
 g.fillRect(r.x,r.y,r.width,r.height);
 g.setColor(this.getForeground());
 return true;
 }
else return super.action(event,obj);
}
}
```

Une applet permet de dessiner des images mais également peut afficher des images, et jouer des fichiers son importés.

Dans l'exemple suivant,nous présentons une applet permettant de gérer plusieurs activités parallèles à l'aide d'un *thread* et d'un gestionnaire d'événements.

Dans la méthode *init()* nous créons un ensemble de boutons, nous lisons les paramètres et instanciations une classe *MediaTracker()*. La méthode *addImage()* de la classe *MediaTracker* permet de charger un ensemble d'images à visualiser dans l'applet. Les images seront affichées ultérieurement par la méthode *drawImage()* de la classe *Graphics*.



Les actions de l’affichage sont exécutées par un *thread TimerThread()*. La cadence de l’affichage est indiquée par un paramètre (*cadence*).

Le *thread* est instancié et initialisé dans la méthode *start()* de l’applet. La méthode *run()* du *thread* ré-affiche les images par l’appel de *repaint()* effectué sur le composant graphique *comp*.

L’applet fonctionne en continu.

```
public void run() {
 while(true) {
  try
  {
  if(fgetStep())                    // mode de fonctionnement pas a pas - si fgetStep() egal true
          {
          comp.repaint();
          fsetStep(false);
          fsetRun(false);
          }
  if(fgetRun()) comp.repaint();  // mode de fonctionnement cyclique - si fgetRun() egal true
  sleep(timediff*fgetCad());
  }
  catch(Exception e) {}
 }
}
```

La méthode *run()* est contrôlée par trois arguments:

- *fgetStep()* - autorise un pas de fonctionnement ,
- *fgetRun()* - autorise l’affichage cyclique ,
- *fgetCad()* - détermine la cadence d’affichage.

Les méthodes *fsetStep()* et *fsetRun()* permettent d'initialiser des indicateurs booléens de contrôle. Les affectations et la lecture des indicateurs de contrôle sont effectués en parallèle avec les actions de l'utilisateur initialisées par le biais des boutons. Les événements captés sur les boutons sont redirigés vers la méthode *action()* du *thread* principal.

```
public boolean action(Event evt, Object arg) {
if(evt.target instanceof Button) {
  String label = (String)arg;
  if(label.equals("init")) count = 0;
  if(label.equals("resume")) timer.fsetRun(true);
  if(label.equals("stop")) timer.fsetRun(false);
  if(label.equals("step")) timer.fsetStep(true);
  if(label.equals("slow")) timer.fsetCad(timer.fgetCad() * 2);
  if(label.equals("fast")) timer.fsetCad(timer.fgetCad() / 2);  }
  return true;
 }
```

La méthode *action()* positionne différents indicateurs en parallèle avec les activités du *thread TimerThread()*. Les conflits d'accès potentiels sont gérés par les moniteurs (méthodes synchronisées).

Par exemple, les méthodes *fsetRun(boolean)* et *fgetRun()* sont des méthodes synchronisées permettant l'accès exclusif à l'indicateur privé *shouldRun*.

```
public synchronized boolean fsetRun(boolean setRun) {
shouldRun = setRun;
return true;
}

public synchronized boolean fgetRun() {
return shouldRun;
}
```

Ci-dessous le programme complet de l'applet:

```
import java.applet.*;
import java.awt.*;
public class MonApplet2 extends Applet {
Button binit; Button arret; Button reprise; Button step; Button slow; Button fast;
int count, lastcount;
int cad;
TimerThread timer;
Image[] pictures;
String numero; String chemin;
String repertoire; // repertoire des images
String nombre;     // nombre d'images
String cadence;    // cadence d'affichage
String codage;     // type d'images .jpg , .gif, ..
```

```
public void init() {
binit = new Button("init"); arret = new Button("stop"); step = new Button("step");
reprise = new Button("resume"); fast = new Button("fast"); slow = new Button("slow");
add(binit); add(arret); add(step); add(reprise); add(fast); add(slow);
count = 0;
nombre = getParameter("nombre");
lastcount = Integer.parseInt(nombre);
repertoire = getParameter("repertoire");
Ci-dessous le programme complet de l'applet:pictures = new Image[lastcount];
codage = getParameter("codage");
MediaTracker tracker = new MediaTracker(this);
for(int a=0; a<lastcount;a++)
 {
 numero = new Integer(a).toString();
 chemin = repertoire + "/" + numero + codage;
 pictures[a] = getImage(getCodeBase(), chemin);
 tracker.addImage(pictures[a],0);
 }
tracker.checkAll(true);
setBackground(Color.white);
}
public void start() {
 cadence = getParameter("cadence");
 cad = Integer.parseInt(cadence);
 timer = new TimerThread(this,cad);
 timer.start();
 }
public void stop() {
 timer.fsetRun(false);
 timer = null;
 }
public void update(Graphics g) {
 paint(g);
 }
public void paint(Graphics g) {
 g.drawImage(pictures[count++],10,25,null);
 if(count == lastcount) count = 0;
 }
      public boolean action(Event evt, Object arg) {
      if(evt.target instanceof Button) {
        String label = (String)arg;
        if(label.equals("init")) count = 0;
        if(label.equals("resume")) timer.fsetRun(true);
```

```
  if(label.equals("stop")) timer.fsetRun(false);
  if(label.equals("step")) timer.fsetStep(true);
  if(label.equals("slow")) timer.fsetCad(timer.fgetCad() * 2);
  if(label.equals("fast")) timer.fsetCad(timer.fgetCad() / 2);  }
  return true;
 }
}
```

Et le programme du *thread*:

```
import java.applet.*;
import java.awt.*;
public class TimerThread extends Thread {
Component comp;
int timediff;
private boolean shouldRun; boolean setRun;
private boolean shouldStep; boolean setStep;
private int facCad=128; int setCad;
public TimerThread(Component comp, int timediff)
{
  this.comp = comp;
  this.timediff = timediff;  this.fsetRun(false);  this.fsetStep(false);
}
public synchronized boolean fsetRun(boolean setRun) { shouldRun = setRun; return true;}
public synchronized boolean fgetRun() { return shouldRun;}
public synchronized boolean fsetStep(boolean setStep) { shouldStep = setStep; return true;}
public synchronized boolean fgetStep() { return shouldStep;}
public synchronized int fsetCad(int setCad) { if(setCad > 1) facCad = setCad; return facCad;}
public synchronized int fgetCad() { return facCad; }
public void run() {
  while(true) {
    try
    {
    if(fgetStep())
              {  comp.repaint();
        fsetStep(false);
              fsetRun(false); }
    if(fgetRun()) comp.repaint();
    sleep(timediff*fgetCad());
    }
    catch(Exception e) {}
  }
 }
}
```

Le dernier exemple illustre comment ajouter un simple *thread* qui apporte un fond sonore. La solution la plus rapide consiste à prendre une applet existant et à l'étendre par un *thread* supplémentaire qui joue le son (un fichier son).

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


public class SoundThread extends Thread {
private Applet applet;
private String faudio;
public SoundThread(Applet app, String fa) {
applet = app;
faudio = fa;
this.start();
}
public void run() {
 applet.play(applet.getDocumentBase(),faudio);
  }
}
```

Evidemment le *thread SoundThread* doit être déclaré et instancié dans la méthode *start()* de l'applet:

```
 Thread sound;
 sound = new SoundThread(this,cad);
```

Remarque: La méthode *play()* lance le fichier audio immédiatement après son chargement. Pour pouvoir écouter le même clip plusieurs fois, il faut utiliser la méthode *getAudioClip()*. Une fois l'objet *AudioClip* chargé dans l'applet, il peut être lancé de façon cyclique par la méthode *play()* ou *loop()*. Dans l'exemple suivant la classe *PlayBeep* récupère un fichier audio et lance son exécution en boucle.

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


public class PlayBeep extends Applet implements Runnable {
private AudioClip beep;
private boolean stopped = false;


public void init() {
beep = this.getAudioClip(this.getDocumentBase(),"sons/beep.wav");
if(beep != null) {
  Thread t = new Thread(this);
  t.start();    // lancement du thread principal
  }
}


public void start()
```

```
{ this.stopped = false; }
public void stop()
{ this.stopped = true; }
public void run() {
Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
while(true) {
 if(!stopped) beep.play();
 try  {  Thread.sleep(4000); }
 catch(InterruptedException e) {}
  }
}
}
```

Résumé

Les applets permettent de développer très rapidement les interfaces utilisateur à intégrer dans les pages WEB. Elle permettent d'effectuer des traitements en série et en parallèle par le biais de plusieurs *threads*. Les applets offrent une excellente opportunité pour le développement des interfaces réseaux côté client.

Dans la deuxième partie de cet ouvrage nous présentons les mécanismes de communication sur le réseau Internet. Nous présentons les classes Java introduites pour la programmation des applications Internet et nous développons quelques applications simples type client-serveur.