Chapter 5

This chapter illustrates techniques to develop applets. An applet is a program of type thread interpreted by a web browser.

We will discuss how:

- Draw a picture
- organize the collection of events generated by the keyboard and mouse
- retrieve external arguments
- add buttons
- create an animation in a thread and play sound clips

The simplest applet displays a line of text:

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


public class AppletUn extends Applet {
public void paint(Graphics g) {                 // this method displays applet
g.drawString(“Bonjour”,20,30); }
}
```

In the applet above there is no data processing. The initial processing requires the method init (); the method paint () may be executed multiple times.

```
import java.applet.*;
import java.awt
public class AppletDeux extends Applet {
static final String texte = "Bonjour";
private Font font;
public void init() {            // initialization
font = new Font("Courier",Font.BOLD, 36);
}
public void paint(Graphics g)               // paint()  display
{ g.setColor(Color.yellow);
  g.fillOval(20,20,200,200);
  g.setColor(Color.black);
  g.setFont(font);
  g.drawString(texte,40,120); }
}
```

Capture of events

The events generated by the mouse can be captured by the applet using the methods pre-defined in the *java.awt* package. For example, methods *mouseDown()* and *mouseDrag()* detect events associated with the touch of a button and the movement of the mouse. If an event is actually detected, the methods return Boolean value *true*.

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

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

public void init() {
this.setBackground(Color.yellow);
}

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;
}
}
```

Settings of applet

An applet can retrieve its arguments (parameters) from the web page in which it is integrated. The following text (html file) shows how to seize the parameters of an 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>   // name and size of the applet
<param name="cfond" value="00AAFF">
<param name="csurf" value="000000">
</APPLET>
</body>
</html>
```

The parameters to read are recovered by the method init ().

```
import java.applet.*;
import java.awt.*;
public class AppletQuatre extends Applet{
private int dernier_x = 0; private int dernier_y = 0;


public void init() {
Color cfond = getColorPar("cfond");
Color csurf = getColorPar("csurf");
if(csurf != null) this.setForeground(csurf);
if(cfond != null) this.setBackground(cfond);
}
protected Color getColorPar(String nom) {
String valeur=this.getParameter(nom);
int intvaleur;
try { intvaleur= Integer.parseInt(valeur,16); }  // conversion hexa => decimal
catch(NumberFormatException e) { return null; }
return new Color(intvaleur);
}
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;
}
}
```

The following applet is an extension of the previous applet. In this example we add a button to initiate action. This action clears the graphic content of the applet. Initially the applet is seen as a rectangle of background color. The process of clearing needs to draw the background color on the entire surface of the applet.

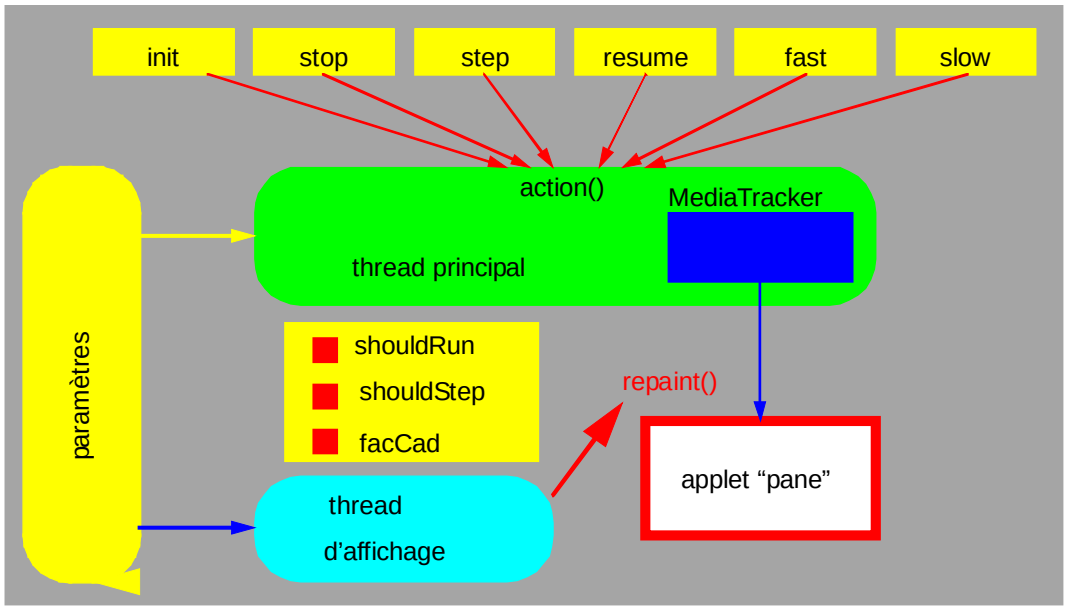
Warning: This process of erasing may take several seconds

```
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();          // integration of the  init() method of the superclass
        effacer = new Button("effacer");
        effacer.setForeground(Color.black);
        effacer.setBackground(Color.yellow);
        this.add(effacer);
        }

public boolean action(Event event,Object obj) {     // event management
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);
}
}
```

An applet can draw pictures but can also display images and play imported sound files.

In the following example, we present an applet to manage multiple activities in parallel using a thread and an event handler. In the *init()* method we create a set of buttons, we read the parameters and instantiate a class *MediaTracker*. The method *addImage()* of class *MediaTracker* can load a set of images to be displayed in the applet. The images will be displayed later by the method *drawImage()* method of the *Graphics* class.



The actions of the display are executed by a thread *TimerThread()*. The speed of the display is indicated by a parameter (*speed*).

The thread is instantiated and initialized in the *start()* method of the applet. The *run()* method of thread re-displays the images by calling *repaint()* performed on the graphical component *comp*.

The applet runs continuously.

```
public void run() {
 while(true) {
  try
  {
  if(fgetStep())                    // mode step by step - if fgetStep() equal true
            {
            comp.repaint();
            fsetStep(false);
            fsetRun(false);
            }
  if(fgetRun()) comp.repaint();  // mode cyclique - if fgetRun() equal true
  sleep(timediff*fgetCad());
  }
  catch(Exception e) {}
 }
}
```

The run () method is controlled by three arguments:

- fgetStep () - allows a step function,
- fgetRun () - allows the cyclical display
- fgetCad () - determines the display speed.

Methods fsetStep () and fsetRun () can initialize control indicators based on Boolean values. The assignments and monitoring indicators are performed in parallel with the actions of the user initialized through the buttons. The events captured on the buttons are redirected to the *action()* method of the main thread.

```
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;
 }
```

The method *action()* sets various indicators in parallel with the activities of the thread *TimerThread()*. The potential access conflicts are managed by the monitors (synchronized methods). For example, methods *fsetRun(boolean)* and *fgetRun()* are synchronized allowing exclusive access to the private *shouldRun* indicator.

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

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

Below the full program of the 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");
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;
         }
        }
```

And the program of the 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) {}   }
  }
 }
```

The last example illustrates how to add a single thread that provides the background. The quickest solution is to take an existing applet and extend it by an additional thread that plays the sound (a sound file).

```
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);
  }
}
```

Obviously the thread *SoundThread* must be declared and instantiated in the *start()* method of the applet:

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

Note: The play () method starts the audio file immediately after loading. To listen to the same clip several times, use the method *getAudioClip()*. Once AudioClip be loaded into the applet, it can be started cyclically by the method *play()* or *loop()*. In the following example the class *PlayBeep* retrieves an audio file and starts running in a loop.

```
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) {}
  }
} }
```

Summary

Applets allows us to develop user interfaces to integrate into web pages. They can perform serial or parallel processing through the use of multiple threads. The applets offer an excellent opportunity for the development of client-side network interfaces.

In the second part of this work we present the mechanisms of communication on the Internet. We present the Java classes used to program with Internet protocols and give some examples of simple client-server applications.