

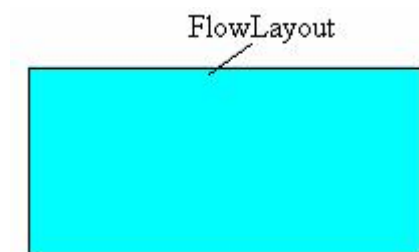
La programación del ratón (II)

Un programa simple de dibujo "a mano alzada"

Propósito

Este ejercicio final va a consistir en un programa de dibujo "a mano alzada". Para trazar una línea irregular con el ratón necesitamos de un punto inicial, es decir, aquél que se obtiene al pulsar el botón izquierdo del ratón. Luego, mantenemos pulsado el botón izquierdo del ratón y lo arrastramos dibujándose de este modo una curva irregular que une las posiciones por las que ha pasado el puntero del ratón.

Diseño



Crear un applet mínimo, solamente con el método *init*

En la función respuesta a la acción de pulsar el botón izquierdo del ratón y de arrastrar el ratón se obtienen las coordenadas *x* e *y* de la posición que señala el puntero, a partir del objeto *ev* de la clase *MouseEvent* mediante *getX* y *getY*.

```
void funcionRespuesta(MouseEvent ev) {  
    int x=ev.getX();  
    int y=ev.getY();  
}
```

Para dibujar una polilínea se precisa de un punto inicial que se obtiene al pulsar el botón izquierdo del ratón. El punto final, se obtiene al arrastrar el ratón luego, se une mediante una línea el punto inicial y final. El punto final se convierte en el inicial para el siguiente movimiento del ratón luego, se traza la segunda línea entre dichos puntos, y así sucesivamente.

Respuesta a las acciones del usuario

Para hacer este programa necesitamos implementar dos interfaces *MouseListener* y *MouseMotionListener* que definan las funciones *mousePressed* y *mouseDragged*, respectivamente. Para ello, vamos a seguir dos aproximaciones.

1.-La clase que describe el applet implementa los interfaces

La clase derivada que describe el applet implementa el interface [*MouseListener*](#) y [*MouseMotionListener*](#). Recuérdese que una clase solamente puede derivar de otra clase pero puede implementar varios interfaces, y tiene que definir todas las funciones declaradas en dichos interfaces.

```
public class RatonApplet6 extends Applet
    implements MouseListener, MouseMotionListener {
    //funciones del interface MouseListener
    public void mousePressed(MouseEvent ev) {
        //código de la función respuesta
    }

    public void mouseExited(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}

    //funciones del interface MouseMotionListener
    public void mouseDragged(MouseEvent ev) {
        //código de la función respuesta
    }
    public void mouseMoved(MouseEvent event) {}
}
```

El siguiente paso, es el de asociar la fuente de los sucesos asociados al ratón el applet (**this**) con el objeto de la clase que implementa los interfaces *MouseListener* y *MouseMotionListener* y que registra dichos sucesos, que también es el applet (**this**).

```
this.addMouseListener(this);
this.addMouseMotionListener(this);
```

Finalmente, queda definir las funciones respuesta *mousePressed* y *mouseDragged*. En la primera, se obtiene el punto inicial, cuando se pulsa el botón izquierdo del ratón.

```
public void mousePressed(MouseEvent ev) {
    uX=ev.getX();
    uY=ev.getY();
}
```

En la llamada a la función *mouseDragged*, se traza una línea entre dos puntos muy próximos. El punto previo de coordenadas *uX* y *uY*, y el punto actual que los suministra el parámetro *ev* de la clase *MouseEvent* de dicha función. Se dibuja una línea en un contexto gráfico *g*, que une los puntos actual y previo. Finalmente, el punto actual se convierte en el punto previo. El contexto gráfico *g* se obtiene mediante la función *getGraphics*.

La función *mouseDragged* se llama muchísimas veces, por lo que se crean muchos objetos *g*, que deberían ser liberados por el recolector de basura (garbage collector) al finalizar su ámbito de existencia, desde el momento en que fue creado hasta el final de la definición de dicha función. En vez de confiar en que dicho recolector libere la memoria ocupada por cada objeto *g* creado cuando se sale de la función *mouseDragged*, podemos hacerlo manualmente mediante la llamada a *dispose*.

Cuando el objeto gráfico *g* lo suministran las funciones *paint* o *update* en su único parámetro no es necesario esta llamada, la liberación de memoria la realiza automáticamente el sistema cuando dichas funciones retornan.

```
public void mouseDragged(MouseEvent ev) {
    int x = ev.getX();
    int y = ev.getY();
    Graphics g=getGraphics();
    g.drawLine(uX, uY, x, y);
    uX=x;    uY=y;
    g.dispose();
}
```

El código completo de este ejemplo, es el siguiente

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

public class RatonApplet6 extends Applet
    implements MouseListener, MouseMotionListener {
    int uX=0, uY=0;

    public void init() {
        this.addMouseListener(this);
        this.addMouseMotionListener(this);
    }
    //interface MouseListener
    public void mousePressed(MouseEvent ev) {
        uX=ev.getX();
        uY=ev.getY();
    }
    public void mouseExited(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}

    //interface MouseMotionListener
    public void mouseDragged(MouseEvent ev) {
        int x = ev.getX();
        int y = ev.getY();
        Graphics g=getGraphics();
        g.drawLine(uX, uY, x, y);
        uX=x;    uY=y;
        g.dispose();
    }
    public void mouseMoved(MouseEvent event) {}
}
```