

La programación del ratón (I)

El ratón es uno de los dispositivos estándares en un interfaz gráfico, que facilita la interacción del usuario con el programa. Las acciones de pulsar sobre un botón, seleccionar un elemento del menú, o una herramienta en una caja de herramientas, activar o desactivar una casilla de verificación, dibujar una figura en el canvas del programa Paint de Windows, seleccionar una palabra en el procesador de textos, etc, se realizan frecuentemente cuando se trabaja con algún programa.

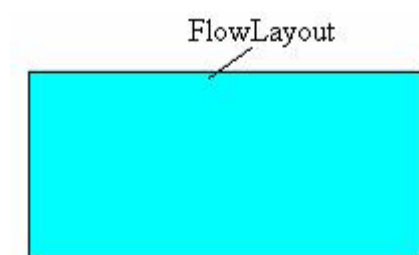
No hace demasiado tiempo, la programación del ratón era complicada y se realizaba a bajo nivel, con instrucciones cercanas al lenguaje de la máquina. La creación de un interfaz gráfico que hiciese uso del ratón era una tarea muy complicada hasta la aparición de las librerías específicas, primero de funciones y luego de clases, como TurboVision, ObjectVision, etc. Dichas librerías definen elementos estándares, como ventanas, marcos, controles y menús, dejando al usuario la tarea de crear el interfaz con dichos elementos y de definir su comportamiento, liberándole de los detalles de la programación a bajo nivel.

Pulsando el botón izquierdo del ratón

Propósito

Vamos a dibujar pequeños círculos de color rojo sobre el área de trabajo del applet cada vez que se pulsa el botón izquierdo del ratón. Los puntos son círculos centrados en la posición que señala el puntero del ratón cuando se pulsa el botón izquierdo.

Diseño



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

En la función respuesta a la acción de pulsar el botón izquierdo del ratón, dibujar un círculo centrado en la posición en el que se ha pulsado el botón izquierdo del ratón. Las coordenadas *x* e *y* de dicho punto se obtienen a partir del objeto *ev* de la clase *MouseEvent* mediante *getX* y *getY*. A continuación, dibujar el círculo en el contexto gráfico del applet.

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

```
}
```

Respuesta a las acciones del usuario

Ya vimos como podíamos manejar los sucesos provenientes de un botón haciendo que la clase que describe el applet implemente el interface [*ActionListener*](#). En este ejemplo vamos a emplear el mismo modelo.

La clase *RatonApplet1* solamente puede derivar de *Applet*, no puede derivar de más de una clase, pero puede implementar más de un interface. La clase *RatonApplet1* al implementar el interface *MouseListener* debe definir todas las funciones declaradas en dicho interface, aunque solamente estemos interesados en una de ellas.

```
public class RatonApplet1 extends Applet implements MouseListener {
    //....
    public void mousePressed(MouseEvent event) {
        //código ...
    }
    public void mouseExited(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
}
```

Asociamos el componente que genera los sucesos, el applet (**this**), y el objeto de la clase que implementa el interfaz *MouseListener*, es decir, que maneja dichos sucesos, que es también el applet (**this**).

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

o abreviadamente,

```
addMouseListener(this);
```

Cuando se pulsa el botón izquierdo del ratón se llama a *mousePressed*, aquí tenemos que situar el código correspondiente a la tarea a realizar, que no es otra que dibujar pequeños círculos de color rojo.

En primer lugar, se obtiene el contexto gráfico *g*, (un objeto de la clase *Graphics*) mediante *getGraphics*. Una vez obtenido el contexto gráfico, se pueden pintar en ella llamando a distintas funciones definidas en la clase *Graphics*. Finalmente, no debemos de olvidarnos de liberar los recursos asociados al contexto gráfico *g*, mediante la llamada a *dispose*.

La función respuesta *mousePressed* nos suministra un objeto *ev* de la clase *MouseEvent* que nos proporciona información acerca del suceso (event). Las funciones miembro *getX*, *getY* de la clase *MouseEvent*, nos suministra las coordenadas del punto donde estaba situado el puntero del ratón en el momento de pulsar su botón izquierdo. Alternativamente, *getPoint* nos suministra estas dos coordenadas encapsuladas en un objeto de la clase *Point*. Una vez que tenemos las coordenadas del punto podemos dibujar mediante *fillOval* un círculo centrado en dicho punto de un determinado radio.

```
public void mousePressed(MouseEvent ev) {
```

```

    Graphics g = getGraphics();
    g.setColor(Color.red);
    g.fillOval(ev.getX()-radio, ev.getY()-radio, 2*radio, 2*radio);
    g.dispose();
}

```

El código completo de este ejemplo, es el siguiente

```

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

public class RatonApplet1 extends Applet implements MouseListener {
    private final int radio = 8;
    public void init() {
        addMouseListener(this);
    }
    public void mousePressed(MouseEvent ev) {
        Graphics g = getGraphics();
        g.setColor(Color.red);
        g.fillOval(ev.getX()-radio, ev.getY()-radio, 2*radio, 2*radio);
        g.dispose();
    }
    public void mouseExited(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
}

```

Dibujar los puntos y guardarlos en memoria

Propósito

En los programas anteriores se dibujan los puntos en la ventana del applet, y se oculta por otra ventana, al descubrir el applet, los puntos han desaparecido. Vamos a crear un programa de manera que las coordenadas de los puntos sean guardados en memoria y que vuelvan a pintarse cuando la ventana del applet se descubre después de haber estado oculta parcial o totalmente por otra ventana. Para probarlo, se dibujan algunos puntos rojos en la superficie del applet, luego se oculta parcialmente el applet subiendo o bajando esta página web, actuando sobre la barra de desplazamiento vertical de la ventana del navegador.

Debemos de recordar, que la función miembro *paint* se redefine en la clase derivada de *Applet* para pintar algo en su superficie, en las siguientes circunstancias:

- Cuando aparece por primera vez el applet
- Cuando se llama a la función *repaint* desde otro método, véase el ejemplo hacer doble-clic sobre un elemento de la lista.
- Cuando la ventana del applet ha sido ocultada por otra y se descubre.

Diseño

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

Crear un array de objetos de la clase *Point*, cuya dimensión venga dada por la constante *MAXPUNTOS*

Guardar los puntos que se dibujan con el ratón en el array. Los objetos de la clase *Point* se obtienen a partir del objeto *ev* de la clase *MouseEvent*, mediante la función *getPoint*.

```
void dibujaPuntos(MouseEvent ev) {  
    Point p=ev.getPoint();  
}
```

Redefinir la función *paint*, para dibujar todos los puntos guardados en el array.

Respuesta a las acciones del usuario

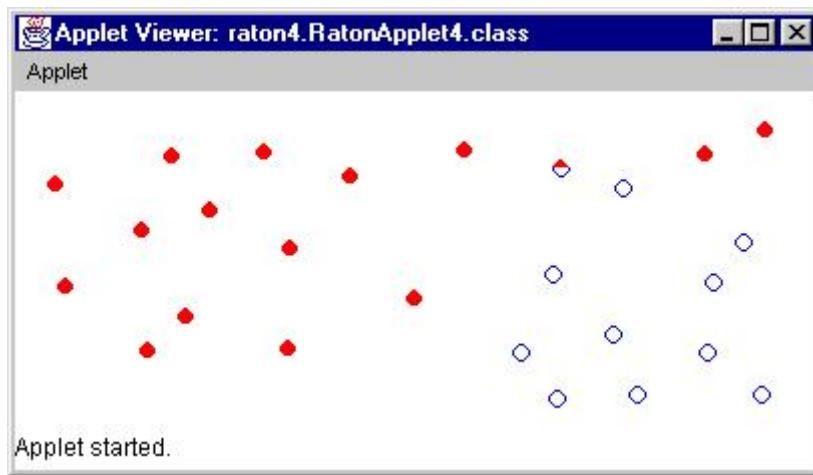
Partimos del programa anterior guardando los puntos en una array de objetos del tipo *Point*, hasta un máximo de *MAXPUNTOS*. A la definición de la función respuesta *dibujaPuntos* le añadimos el código que guarda los puntos en el array *puntos*, y previene que no se sobrepase la dimensión de dicho array

```
void dibujaPuntos(MouseEvent ev) {  
    Graphics g=getGraphics();  
    g.setColor(Color.red);  
    if (nPuntos<MAXPUNTOS) {  
        puntos[nPuntos]=ev.getPoint();  
        nPuntos++;  
    }  
    g.fillOval(ev.getX()-4, ev.getY()-4, 8, 8);  
    g.dispose();  
}
```

Redefinimos la función *paint* para que dibuje en forma de circunferencias de color azul todos los puntos guardados en memoria. De este modo, diferenciamos los puntos que se dibujan al pulsar el botón izquierdo del ratón (círculos de color rojo), de los puntos cuyas coordenadas se guardan en memoria y se dibujan (circunferencias de color azul) cuando se llama a la función *paint*.

```
public void paint(Graphics g) {  
    g.setColor(Color.blue);  
    for (int i=0; i<nPuntos; i++) {  
        g.drawOval(puntos[i].x-4, puntos[i].y-4, 8, 8);  
    }  
}
```

¿Qué ocurre cuando se oculta parcialmente la ventana del applet y se vuelve a descubrir. Nos daremos cuenta como se muestra en la figura adjunta, que solamente se vuelve a dibujar aquella zona que ha estado ocultada. Como podemos ver en la figura se vuelven a dibujar (en color azul) los puntos e incluso (aquí está la sorpresa) la parte de un punto que ha estado parcialmente oculto, los demás permanecen inalterados



El código completo de este ejemplo, es el siguiente

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

public class RatonApplet4 extends Applet {
    final int MAXPUNTOS=20;
    Point puntos[]=new Point[MAXPUNTOS];
    int nPuntos=0;
    public void init() {
        this.addMouseListener(new RatonEvent(this));
    }
    void dibujaPuntos(MouseEvent ev) {
        Graphics g=getGraphics();
        g.setColor(Color.red);
        if(nPuntos<MAXPUNTOS){
            puntos[nPuntos]=ev.getPoint();
            nPuntos++;
        }
        g.fillOval(ev.getX()-4, ev.getY()-4, 8, 8);
        g.dispose();
    }

    public void paint(Graphics g){
        g.setColor(Color.blue);
        for(int i=0; i<nPuntos; i++){
            g.drawOval(puntos[i].x-4, puntos[i].y-4, 8, 8);
        }
    }
}

class RatonEvent implements MouseListener {
    RatonApplet4 applet;
    RatonEvent(RatonApplet4 applet){
        this.applet=applet;
    }
    public void mousePressed(MouseEvent ev) {
        applet.dibujaPuntos(ev);
    }
    public void mouseExited(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
}
```

}
