2.3.2. La interfaz RUNNABLE

Para añadir la funcionalidad de hilo a una clase que deriva de otra clase (por ejemplo, un applet), siendo esta distinta de Thread, se utiliza la interfaz Runnable. Esta interfaz glade la funcionalidad de hilo a una clase con solo implementarla. Por ejemplo, para dadir la funcionalidad de hilo a un applet definimos la clase como:

public class Reloj extends Applet implements Runnable {}

La interfaz Runnable proporciona un único método, el método run(). Este es ejecutado por el objeto hilo asociado. La forma general de declarar un hilo implementando la interfaz Runnable es la siguiente:

```
class NombreHilo implements Runnable {
  //propiedades, constructores y metodos de la clase
public void run() {
  //acciones que lleva a cabo el hilo
   }
}
```

Para crear un objeto hilo con el comportamiento de NombreHilo escribo lo siguiente:

```
NombreHilo h = new NombreHilo();
```

Y para iniciar su ejecución utilizamos el método start():

```
new Thread (h).start ( ) ;
```

O bien para lanzar el hilo escribimos lo anterior en dos pasos:

```
Thread hl = new Thread (h).
h1.start ();
```

O en un paso todo:

```
new Thread (new NombreHilo ( ) ).start ( );
```

El siguiente ejemplo declara la clase PrimerHiloR que implementa la interfaz Runnable, en el método run() se indica la funcionalidad del hilo, en este caso es pintar un mensaje y visualizar el identificador del hilo actualmente en ejecución:

```
public class PrimerHiloR implements Runnable {
      public void run(){
          System.out.println("Hola desde el Hilo! "                                    +
                              Thread.currentThread().getId());
       }
}//PrimerHiloR
```

A continuación, se muestra la clase *UsaPrimerHiloR.java* donde se lanzan varios hilos del tipo anterior de distintas formas:

```
public class UsaPrimerHiloR {
   public static void main(String[] args) {
      //Primer hilo
      PrimerHiloR hilo1 = new PrimerHiloR();
       new Thread(hilo1).start();

      //Segundo hilo
        PrimerHiloR hilo2 = new PrimerHiloR( );
         Thread hilo= new Thread(hilo2);
       hijo.start();

        //Tercer Hilo
            new Thread(new PrimerHiloR()).start( );
```

```
}//UsaPrimerHiloR
```

ACTIVIDAD 2.2

Crea una clase que implemente la interfaz Runnable cuya única funcionalidad sea visualizer el mensaje “hola mundo” seguido de una cadena que se recibirá en el constructor y seguii del indentificador del hilo. Crea un programa Java que visualice el mensaje anterior creando para ello 5 hilos diferentes usando la clase creada anteriormente.

Seguidamente vamos a ver como usar un hilo en un applet para realizar una tarea repetitiva, en el ejemplo la tarea sera mostrar la hora con los minutos y segundos: HH:MM:SS; vease Figura 2.2; normalmente la tarea repetitiva se encierra en un bucle infinito. Un applet es una aplicacion Java que se puede insertar en una pagina web; cuando el navegador carga la pagina, el applet se carga y se ejecuta. Nuestro applet implementara la interfaz Runnable, por tanto debe incluir el metodo run() con la tarea repetitiva.

Hemos de tener en cuenta que al utilizar applet en versiones de Java 10 y superiores se muestra una advertencia indicando que la API de applet y AppletViewer estan en desuso.

En un applet se definen varios métodos:

- init(): con instrucciones para inicializar el applet. este metodo es llamado una vez cuando se carga el applet.
- start(): parecido a ink) pero con la diferencia de que es llamado cuando se reinicia el applet.
- paint(): que se encarga de mostrar el contenido del applet; se ejecuta cada vez que se tenga que redibujar.
- stop(): es invocado al ocultar el applet, se utiliza para detener hilos.

El navegador web llama primero al metodo init(), luego a paint() y a continuación al método start(). El hilo lo crearemos dentro del metodo start() usamos la siguiente expresión:

hilo = new Thread(this);

Al especificar this en la sentencia new Thread() se indica que el applet proporciona el cuerpo del hilo.

La estructura general de un applet que implementa Runnable es la siguiente:

```
import java.awt.*;
import java.applet.*;
public class AppletThread extends Applet implements Runnable{
    private Thread hilo = null;
   public void init(){
    }
   public void start(){
                if(hilo == null)    {
          // crea el hilo
          hilo = new Thread(this);
          hilo.start(); // lanza el hilo
                }
      }
public void run() {
        Thread hiloActual = Thread.currentThread();
            while (hilo == hiloActual)
          // tarea repetitiva
              }
```

```
}
public  void  stop() {
      hilo =  null;
}
public void paint(Graphics  g) {
}
}
```

Cuando el applet necesita matar el hilo le asigna el valor null, esta acción se realiza en el método stop() del applet (se recomienda en los applets que implementan Runnable). Es una forma mas suave de detener el hilo que utilizar el método stop() del hilo (hilo.stop0), ya que este puede resultar peligroso. El código del método run() es el siguiente:

```
public  void run(){
                Thread   hiloActual  = Thread.currentThread();
                While  (hilo  == hiloActual)
              //  tarea   repetitiva
}
}
```

Donde se comprueba cual es el hilo actual con la expresión Thread.currentThread(); el proceso continúa o no dependiendo del valor de la variable del hilo; si la variable apunta al mismo hilo que está actualmente en ejecución el proceso continúa; si es null el proceso formaliza y si la variable hace referencia a otro hilo es que ha ocurrido una extraña situación, la tarea repetitiva no se ejecutará. Aplicarnos la estructura anterior a nuestro applet, Reloj.java, que muestra la hora:

```
import   java.applet.*;
import   java.awt.*;
import   java.text.SimpleDateFormat;
import   java.util.*;

public  class  Reloj  extends  Applet  implements  Runnable  {
    private  Thread  hilo =  null;        //hilo
    private  Font  fuente; //tipo  de  Tetra  para  la hora
    private  String  horaActual          =  "";

    public  void  init() {
           fuente =  new  Font("Verdana",    Font.BOLD,26);
            setBackground(Color.yellow);        //color  de  fondo
            setFont(fuente);//fuente

    public  void start(){
```

```
                    if(hilo ==null){
                hilo= new Thread(this);
               hilo.start();
                  }
}
    public void run(){
          Thread hiloActual = Thread.currentThread();
           while(hilo==hiloActual){
            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
           Calendar cal = Calendar.getInstance();
           horaActual = sdf.format(cal.getTime());
           repaint();//actualizar contenido del applet
                 try{
              Thread.sleep(1000);
              }catch(InterruptedException e){}
     }
}
public void paint(Graphics g){
       g.clearRect(1,1,getSize().width, getSize().height);
       g.drawString(horaActual,20,50);//muestra la hora
}
public void stop(){
        hilo= null;
}
}
```

El metodo repaint() se utiliza para actualizar el applet cuando cambian las imágenes contenidas en el. Los applets no tienen método main(), para ejecutarlos necesitamos crear un fichero HTML, por ejemplo creamos el fichero Reloj.html con el siguiente contenido:

```
<html>
<applet code="Reloj.class" width="200" height="100"› </applet>
</html>
```

Desde un entorno gráfico como Eclipse no sería necesario crear el HTML. Para compilarlo y ejecutarlo desde la línea de comandos usamos el comando appletviewer, se visualizara una ventanita similar a la de la Figura 2.2:

```
D:\CAPIT2>javac Reloj.java
D:\CAPIT2>appletviewer                                    Reloj.html
'I
```

Se usan las clases Calendar y SimpleDateFormat para obtener la hora y darle formato. En el método Paint() del applet se han utilizado los siguientes métodos:

■ clearRect (int x, int y, int ancho, int alto): borra el rectángulo especificado
rellenandolo con el color de fondo de la superficie de dibujo actual.
II

■ setBackground(Color c): establece el color de fondo.

■ setFont(Font fuente): especifica la fuente.

■ drawString(String texto, int x, int y): pinta el texto en las posiciones x e y.

En el siguiente ejemplo se crea un hilo que irá incrementando en 1 un contador, el contador se inicializa en 0. Se definen dos botones. El primero, Iniciar contador, crea el hilo, al pulsarle cambia el texto a Continuar y empieza a ejecutarse el hilo. El botón Parar contador hace que el hilo se detenga y deje de contar, finaliza el metodo run(). Al pulsar de nuevo en Continuar el contador continúa incrementándose a partir del ultimo valor, se lanza un nuevo hilo. La Figura 2.3 muestra un momento de la ejecución.

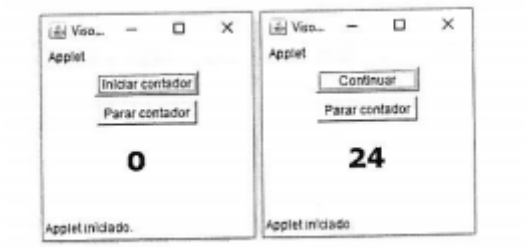

Figura 2.3. Applet ContadorApplet.java.

El applet implementa las interfaces Runnable y ActionListener. Esta última se usa para detectar y manejar eventos de acción, como par ejemplo hacer clic con el ratón en un botón. Posee un solo metodo actionPerformed(ActionEvent e) que es necesario implementar.

En el método init() del applet se añaden los botones con el método add(): add(bl = new Button("Iniciar contador")); y con el método addActionListener() añadimos el listener para que detecte cuando se hace clic sobre el botón: b1.addActionListener(this); se usa this, ya que es la clase la que implementa la interfaz.

Al pulsar uno de los botones se invocara al método actionPerformed(ActionEvent e) donde se analizará el evento que ocurre, la pulsación de un botón o del otro. Dentro del método se comprueba el botón que se ha pulsado. Si se ha pulsado el botón b1 se cambia la etiqueta del botón y se comprueba si el hilo es distinto de nulo y está corriendo, en este caso no se hace nada; y si el hilo es nulo entonces se crea y se inicia su ejecución. Si se pulsa el segundo botón, b2, se utiliza la variable booleana parar asignándole valor true para controlar que no se incremente el contador:

```
public void actionPerformed(ActionEvent e)

if(e.getSource() == bl) //Pulso boton Iniciar contador o Continuar


bl.setLabel("Continuar");

if(h != null && h.isAlive()) II //Si el hilo está corriendo

//no hago nada.

else {
```

```
//creo hilo la primera vez y cuando finaliza el metodo run
h = new Thread(this);
h.start();

} else if(e.getSource() == b2) //Pulso Parar contador
parar = true; //para que finalice el while en el metodo run

I//actionPerformed
```

En el metodo run() se escribe la funcionalidad del hilo. Se ira incrementando el contador siempre y cuando la variable parar sea false. El codigo completo es el siguiente:

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

public class ContadorApplet extends Applet
implements Runnable, ActionListener f private Thread h;
long CONTADOR = 0;
private boolean parar; private Font fuente;
private Button bl,b2; //botones del Applet

public void start() {}

public void init()
setBackground(Color.yellow); //color de fondo add(bl=new Button("Iniciar contador"));
bl.addActionListener(this);
add(b2=new Button("Parar contador")); b2.addActionListener(this);
fuente = new Font("Verdana", Font.BOLD, 26);//tipo letra
public void run() f
parar = false;
Thread hiloActual = Thread.currentThread();
while (h == hiloActual && !parar) {
try
Thread.sleep(300);
catch (InterruptedException e)
e.printStackTrace();
```

```
repaint();
CONTADOR++;



public void paint(Graphics g) {
g.clearRect(0, 0, 400, 400);
g.setFont(fuente); //fuente
g.drawString(Long.toString((long)CONTADOR),80,100);



//para controlar que se pulsan los botones
public void actionPerformed(ActionEvent e) {
if(e.getSource() == 3°1) //Pulso Iniciar contador o Continuar
{
bl.setLabel("Continuar");
if(h != null && h.isAlive()) //Si el hilo est& corriendo
//no hago nada.
else {
//creo hilo la primera vez y cuando finaliza el metodo run h = new Thread(this);
h.start();

} else if(e.getSource() == b2) //Pulso Parar contador
parar=true; //para que finalice el while en el metodo run

I//actionPerformed

public void stop(){
h = null; }
1//fin ContadorApplet
```

ACTIVIDAD 2.3

Partiendo del ejemplo anterior separa el hilo en una clase aparte dentro del applet que extienda Thread. El applet ahora no implementara Runnable, debe quedar asi:

```
public class actividad2_3 extends Applet implements ActionListener
{
class HiloContador extends Thread {
```

```
//atributos  y metodos

}//fin clase
//atributos  y metodos

}//fin actividad2_3
```