

# PROGRAMACIÓN AVANZADA

**Alumno:** Máximo Abel Paute Jumbo

**Fecha:** 8 de julio del 2020

## HILOS

La forma más directa para hacer un programa multihilo es extender la clase Thread, y redefinir el método run(). Este método es invocado cuando se inicia el hilo (mediante una llamada al método start() de la clase Thread). El hilo se inicia con la llamada al método run() y termina cuando termina éste.

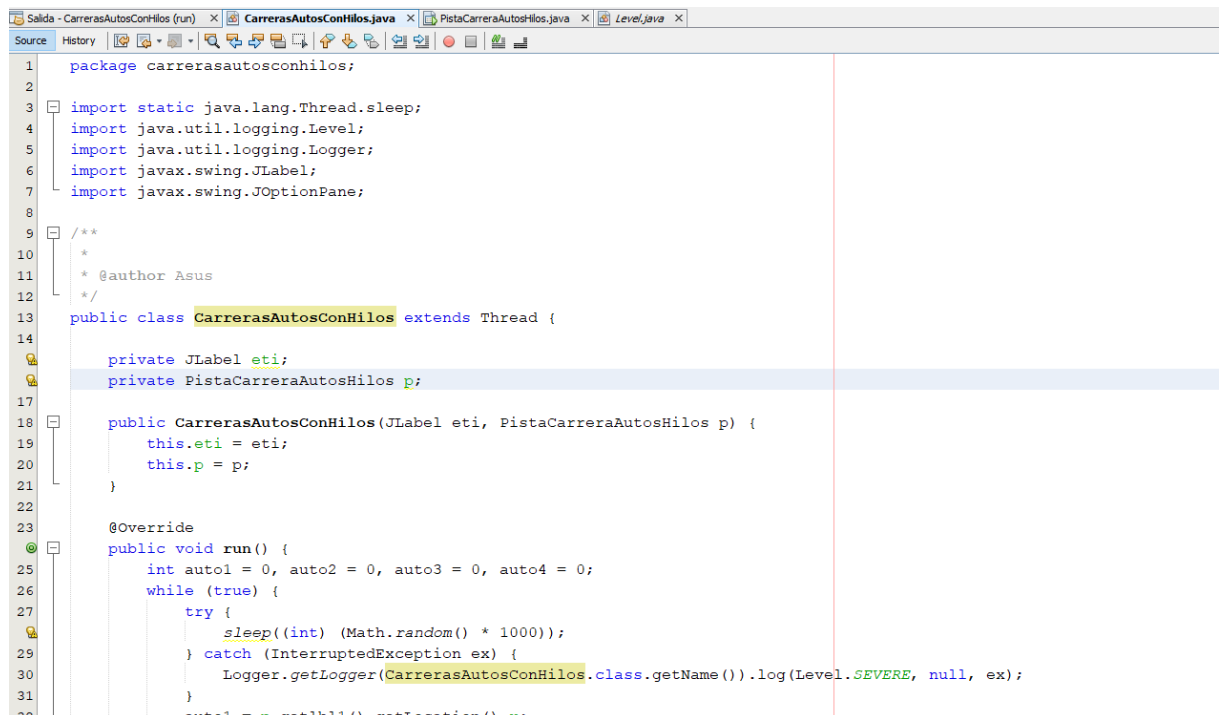
### Simulación de carrera de autos en pista

- La simulación consiste la competencia de 4 carros ejecutados a la misma vez, mediante hilos, obteniendo ganador o empate.

### Proceso

#### 1. Clase CarreraAutosConHilos

Se declara dos atributos de tipo JLabel y PistaCarrerasAutosHilos



```
1 package carrerasautosconhilos;
2
3 import static java.lang.Thread.sleep;
4 import java.util.logging.Level;
5 import java.util.logging.Logger;
6 import javax.swing.JLabel;
7 import javax.swing.JOptionPane;
8
9 /**
10  *
11  * @author Asus
12  */
13 public class CarrerasAutosConHilos extends Thread {
14
15     private JLabel eti;
16     private PistaCarreraAutosHilos p;
17
18     public CarrerasAutosConHilos(JLabel eti, PistaCarreraAutosHilos p) {
19         this.eti = eti;
20         this.p = p;
21     }
22
23     @Override
24     public void run() {
25         int auto1 = 0, auto2 = 0, auto3 = 0, auto4 = 0;
26         while (true) {
27             try {
28                 sleep((int) (Math.random() * 1000));
29             } catch (InterruptedException ex) {
30                 Logger.getLogger(CarrerasAutosConHilos.class.getName()).log(Level.SEVERE, null, ex);
31             }
32             auto1 = p.getk1().getLocation().x;
```

2. **Método run():** El hilo se inicia con la llamada al método run() y termina cuando termina éste.

- Se declara variables para los carros inicializadas en 0.
- Para calcular el tiempo que tarda en ejecutar cada hilo utilizamos el método **Math.random**, multiplicado por 1000 que es el tiempo que representa 1s en milisegundos.
- Se utiliza **getLocation()** Se utiliza para ubicar en el JFrame cada carrito
- **repaint():** Metodo para avisar a la máquina virtual que ese componente necesita repintado. El método en si mismo no borra ni dibuja nada.
- En el if cada auto se va posicionando hacia la derecha de 10 en el eje x, se va acercando a la meta.

```
Salida - CarrerasAutosConHilos (run) x CarrerasAutosConHilos.java x PistaCarreraAutosHilos.java x Level.java x
Source History
22
23 @Override
24 public void run() {
25     int auto1 = 0, auto2 = 0, auto3 = 0, auto4 = 0;
26     while (true) {
27         try {
28             sleep((int) (Math.random() * 1000));
29         } catch (InterruptedException ex) {
30             Logger.getLogger(CarrerasAutosConHilos.class.getName()).log(Level.SEVERE, null, ex);
31         }
32         auto1 = p.getlbl1().getLocation().x;
33         auto2 = p.getlbl2().getLocation().x;
34         auto3 = p.getlbl3().getLocation().x;
35         auto4 = p.getlbl3().getLocation().x;
36         if (auto1 < p.getlbl_Llegada().getLocation().x - 10 && auto2 < p.getlbl_Llegada().getLocation().x - 10
37             && auto3 < p.getlbl_Llegada().getLocation().x - 10 && auto4 < p.getlbl_Llegada().getLocation().x - 10) {
38
39             eti.setLocation(eti.getLocation().x + 10, eti.getLocation().y);
40             p.repaint();
41         } else {
42             break;
43         }
44     }
45
46     if (eti.getLocation().x >= p.getlbl_Llegada().getLocation().x - 10) {
47         if (auto1 > auto2 && auto1 > auto3) {
48             JOptionPane.showMessageDialog(null, "Gano el auto 1");
49         } else if (auto2 > auto1 && auto2 > auto3) {
50             JOptionPane.showMessageDialog(null, "Gano el auto 2");
51         } else if (auto3 > auto1 && auto3 > auto2) {
52             JOptionPane.showMessageDialog(null, "Gano el auto 3");
53         } else if (auto4 > auto1 && auto4 > auto3) {
54             JOptionPane.showMessageDialog(null, "Gano el auto 4");
55         } else {
56             JOptionPane.showMessageDialog(null, "Empate");
57         }
58     }
59 }
60 }
61
```

3. **Botón Iniciar**

- Se crea una instancia de la clase CarreraAutosConHilos con new, luego llamamos al método start() de la thread para hacer que ejecute el método run().

```

146 private void btnIniciarJuegoActionPerformed(java.awt.event.ActionEvent evt) {
147     // TODO add your handling code here:
148
149     lblAuto1.setLocation(0, lblAuto1.getLocation().y);
150     lblAuto2.setLocation(0, lblAuto2.getLocation().y);
151     lblAuto3.setLocation(0, lblAuto3.getLocation().y);
152     lblAuto4.setLocation(0, lblAuto4.getLocation().y);
153     CarrerasAutosConHilos hilo1 = new CarrerasAutosConHilos(lblAuto1, this);
154     CarrerasAutosConHilos hilo2 = new CarrerasAutosConHilos(lblAuto2, this);
155     CarrerasAutosConHilos hilo3 = new CarrerasAutosConHilos(lblAuto3, this);
156     CarrerasAutosConHilos hilo4 = new CarrerasAutosConHilos(lblAuto4, this);
157     hilo1.start();
158     hilo2.start();
159     hilo3.start();
160     hilo4.start();
161
162 }

```

#### 4. Main

- En este caso solo será necesario implementar el método "**run()**" para que los procesos implementados en ese método se ejecuten en un hilo diferente.

```

161
162 }
163
164 /**
165  * @param args the command line arguments
166  */
167 public static void main(String args[]) {
168     /* Set the Nimbus look and feel */
169     Look and feel setting code (optional)
170
171     /* Create and display the form */
172     java.awt.EventQueue.invokeLater(new Runnable() {
173         public void run() {
174             new PistaCarreraAutosHilos().setVisible(true);
175         }
176     });
177
178 }

```

## 5. Resultados



La carrera finaliza con un ganador o un empate, demostrando que se puede realizar varios procesos utilizando hilos que básicamente son multitarea ayudando a optimizar tiempo de ejecución del programa.