1.- Introducción

Métodos para la interrupción de hilos

- void interrupt()
- boolean isInterrupted()
- Static boolean interrupted() (método de clase)
- stop() (este método está obsoleto)

Partimos del ejemplo anterior.

Proyecto Rebotando

Ejercicio 1

Vamos a detener uno de los hilos

Para observar el movimiento de la pelota, introducimos una pausa con el método "static void sleep()", este método lanza la excepción " InterruptedException"

¿Qué sucede cuando aplicamos el método sleep() a un hilo en ejecución ?

Ese hilo en ejecución queda automáticamente bloqueado mientras se encuentra en pausa , que significa que esté bloqueado, pues que en ese hilo no se puede hacer nada, ni siquiera solicitar su interrupción.

¿Qué sucede si solicitamos la interrupción (void interrupt()) de un hilo bloqueado?

En este caso, el método sleep() lanzará la excepción InterruptedException y por lo tanto debemos gestionarla.

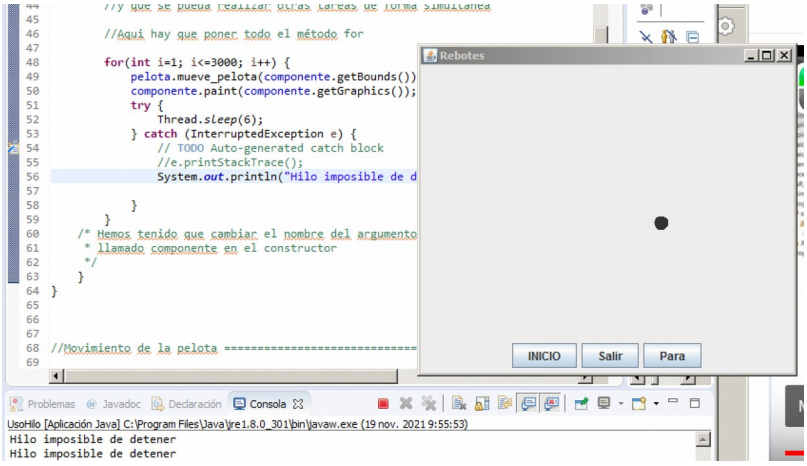
SOLUCIÓN

Agregamos un botón más "Parada" , que detenga la ejecución del hilo en uso. Este botón llamará a un nuevo método que lo llamaremos "detener()"

El metodo comienza_el_juego lanza el hilo con start() esto lo que hace es llamar al método run() correspondiente al hilo de la clase Thread , por lo tanto, en el método detener debemos usar la misma instancia (t) en nuestro caso , para detener la ejecución del hilo.
La instancia debe ser accesible desde el método detener() como del método comienza_el_juego().

La instancia está declarada en el método comienza_el_juego() por lo que no se podrá acceder desde el método detener(), debemos de hacer esta instancia global, a nivel de clase, por lo que hay que sacar su declaración fuera del método comienza_el_juego()

Si usamos el método interrupt(), cuando usamos sleep() , el hilo no se detendrá y lanzará la excepción InterruptedException, esto no sucedería si no usamos sleep().



Si usamos interrupted() o bien isinterrupted().

Interrupted chequea si el hilo ha sido interrumpido (este método lo aplicamos a cualquier hilo)

isInterrupted chequea si este hilo ha sido interrumpido (este método se aplica a un hilo en concreto)

Sustituir el bucle for donde la pelota rebota por un bucle while donde pregunta si el hilo ha sido interrumpido.

while (!Thread.interrupted()) **(este método es static por lo que es un método de clase)**

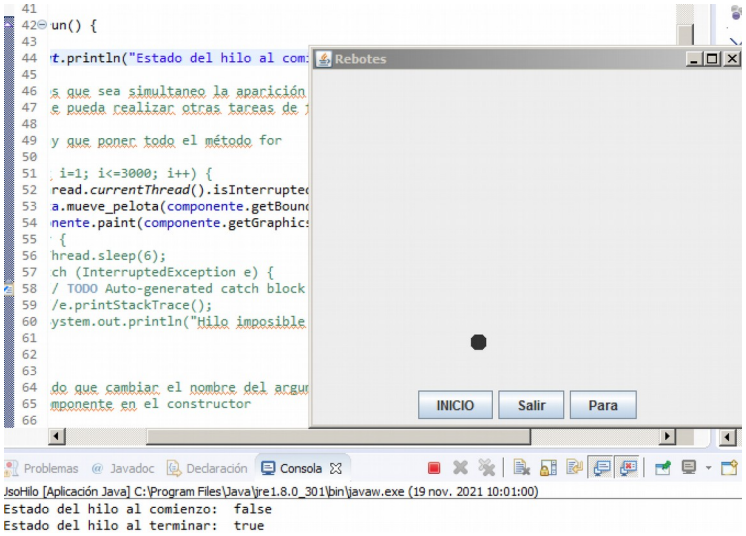
Con **isInterupted()** no es un método estático por lo que debemos indicar el hilo en concreto que se está ejecutando.

while(!Thread.currentThread().isInterrupted())

Imprimir el estado del hilo

Lo hacemos al comienzo del método run()

System.out.println("Estado del hilo al comenzar"+ Thread.currentThread().isInterrupted());



Ejercicio 2
PARAR VARIOS THREAD

En el ejercicio anterior solo se detenía el último hilo . En este ejercicio se trata de detener el hilo de forma individual y a voluntad.

¿Por qué se detiene solo el último hilo?

```
public void comienza_el_juego () {

		Pelota pelota=new Pelota();
		lamina.add(pelota);
		//Creamos una instancia de la clase que implementa Runnable y la almacenamos en una variable de tipo Runnable
		Runnable r = new PelotaHilos(pelota, lamina);
		//Creamos una instancia de la clase Thread y le pasamos al constructor el objeto Runnable
				//Thread t=new Thread(r);
		t=new Thread(r);
		//inicializamos el hilo de ejecución
		t.start();
	}
```

En el método comienza el juego tenemos una instancia de la clase Thread que se llama “t” y esa instancia comienza toda la tarea.

Sucede que al darle al botón de inicio por primera vez, lee el código de la clase comienza el juego, si pulsamos una segunda vez, se vuelve a ejecutar el código por lo que vuelve a crear otra instancia de la clase Thread llamada “t” y no puede haber dos instancias con el mismo nombre. La segunda instancia machaca a la anterior y el primer hilo sigue en ejecución indefinidamente, ahora bien , no se llama “t”, porque ese nombre a sido asignado a un nuevo hilo

Si pulsamos en el botón PARAR , llama al método detener() que tiene la instrucción (t.interrupt()), detiene la instancia “t” por lo que la primera que no se llama “t” se queda corriendo indefinidamente.

Bajamos la velocidad de rebote salvando la excepción.

Si pulsamos el botón de PARAR como estamos usando el metod sleep() lanza la excepción , capturamos la excepción y le decimos que detenga el hilo.

Ejercicio 3

TRABAJAR CON LOS HILOS DE FORMA INDEPENDIENTE

Esto lo solucionamos creando varias instancias con diferentes nombres de la clase Thread.

- Creamos un botón para cada hilo

Elimino los botones que teníamos creado . Elimino el método poner botón y los tres botones creado anteriormente.

En los campos de clase creo tres hilos

Creamos tres botones para iniciar los hilos y otros tres para detenerlos.

Creamos los botones en el constructor de MarcoRebote

En el método "comienza_el_juego" es necesario que diferencia el botón pulsado. Esto se hace introduciendo el parámetro evento en la llamada al método comienza_el_juego.

```
inicia1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent evento) {
                                comienza_el_juego(evento);
                        }
```

Modificamos el método para que reciba un parámetro de este tipo.

```
public void comienza_el_juego (ActionEvent evento) {
                //El objeto evento recibido permite diferenciar el botón pulsado.
```