2023

Práctica 2

ÁLVARO MARTÍNEZ GARCÍA
48150407-E

PROGRAMACIÓN CONCURRENTE

Contenido

Hilos en Java

Debemos implementar en Java un hilo que lance la impresión de una palabra 10 veces. En el programa principal crearemos dos hilos de ese tipo inicializados con dos palabras diferentes y los lanzaremos para que se ejecuten a la vez.

Los hilos se deben implementar de las dos formas vistas en clase de teoría:

Cuestión 1

Creando una clase que herede de la clase *Thread* y que ejecute los hilos desde el programa principal. Para crear un hilo el programador define una clase que extiende la clase *Thread* que es parte del paquete *java.lang*. Comenta los resultados obtenidos al utilizar este método.

En el archivo *javaHilos1.java*, se puede observar el programa desarrollado para esta cuestión. Es similar al ejemplo que se puede ver en la teoría del tema 2. Está compuesto del constructor de la clase, que asignará el string que pasemos para que imprima posteriormente, el método *run()* que se encargará de cumplir la función de método del hilo y mostrará en pantalla la palabra elegida 10 veces y, por último, el *main* que ejecutaremos para comprobar el funcionamiento del programa. Primero crearemos instancias de la clase que hemos creado, siendo estas de tipo *Thread* y ejecutaremos el método *start()* para que ambos hilos comiencen con su ejecución concurrente. A continuación en nuestro código, necesitaremos el método *join()* para esperar a ambos hilos y evitar que el programa acabase antes de que termine la ejecución de los hilos. Esta ejecución, sin el *join()*, podría verse así:

```
user@user-VirtualBox:~/git/PConcurrente/p2$ java ThreadConHerencia
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
Fin del hilo principal
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
user@user-VirtualBox:~/git/PConcurrente/p2$
```

Como podemos observar, la línea de finalización del programa se muestra antes de la segunda palabra que queremos mostrar. Para eso, utilizamos el método *join()*, el cual lanza una excepción que tendremos que tratar, y después se terminaría nuestro programa:

```java
a.start();
b.start();

try{
    a.join();
    b.join();
}catch(InterruptedException e){
    System.out.println("Error esperando al hilo");
}

System.out.println("Fin del hilo principal");
```

De esta forma, obtenemos lo deseado, un programa que llama a dos hilos que mostrarán en la consola dos palabras distintas. La salida de este programa es la siguiente:

```
user@user-VirtualBox:~/git/PConcurrente/p2$ java ThreadConHerencia
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
Fin del hilo principal
user@user-VirtualBox:~/git/PConcurrente/p2$
```

Cuestión 2

Crear el hilo implementando la interfaz *Runnable*. Comenta los resultados obtenidos al ejecutar este método.

Este programa, que se encuentra en el archivo *javaHilos2.java*, tiene una estructura similar al de la cuestión anterior. La principal diferencia con el otro código es que en este utilizaremos los hilos implementando la interfaz *Runnable*. Implementar los hilos de esta manera nos permite que nuestra clase pueda heredar de otras clases al no tener que heredar de T*hread*. Esto hará que la creación de los hilos tenga que ser diferente, instanciando primero a nuestra clase y luego creado variables de tipo *Thread* con estas instancias, de la siguiente manera:

```java
public static void main(String[]args){
    ThreadConRunnable a=new ThreadConRunnable("hilo1");
    ThreadConRunnable b=new ThreadConRunnable("hilo2");

    Thread t1=new Thread (a);
    Thread t2=new Thread (b);

    t1.start();
    t2.start();
```

A partir de ahí, el código es igual a la cuestión anterior. La salida también es igual ya que el funcionamiento es el mismo.

```
user@user-VirtualBox:~/git/PConcurrente/p2$ java ThreadConRunnable
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
Fin del hilo principal
user@user-VirtualBox:~/git/PConcurrente/p2$
```

Cuestión 3

Compara ambos métodos y determina cual consideras que es más adecuado y cuáles son las razones que hacen un método más adecuado que el otro.

Como se ha mencionado en la cuestión anterior, la diferencia principal en las dos formas de utilizar hilos en Java es la herencia. Java no permite herencia múltiple, por lo que nuestra clase solo puede heredar de una. En clases que ya hereden de una previamente, estas no serían capaces de heredar de la clase *Thread*, por lo que en esos casos es necesario implementar la interfaz *Runnable*. Sin embargo, utilizar hilos en Java de esta forma puede resultar algo confusa para el programador. Es por ello que, salvo que la clase herede ya de otra y no pueda heredar de *Thread*, se pueden usar ambos métodos, siendo la herencia de *Thread* el más claro.

Cuestión 4

Modifica el programa realizado en el apartado *2*, para que el primer hilo creado en el main tenga prioridad *1*. ¿Qué ocurre con la ejecución de los hilos? ¿Qué puedes decir acerca de la gestión de prioridades por parte de la máquina virtual de tu instalación de Java?

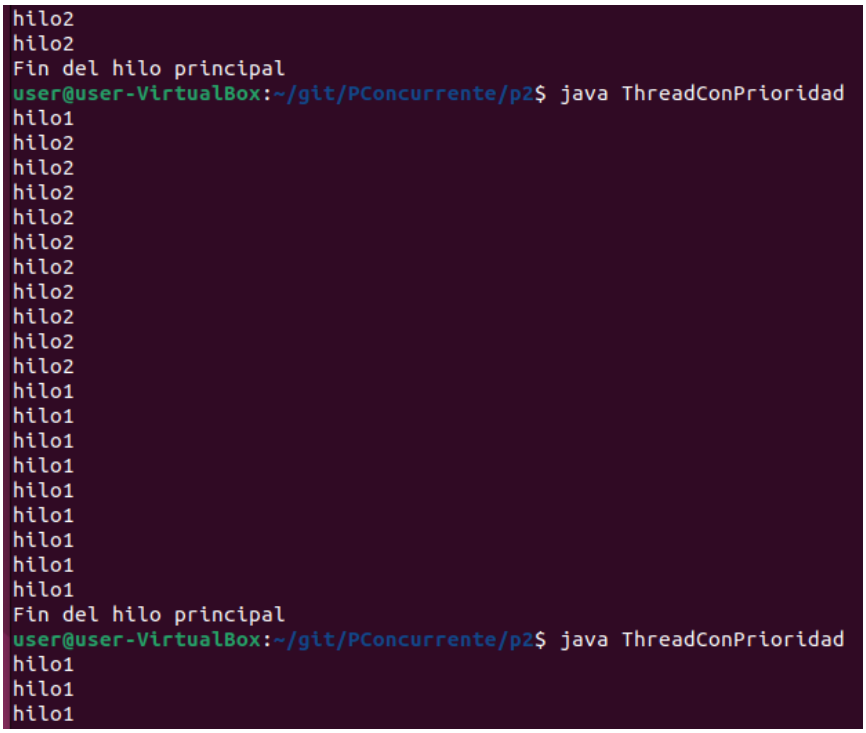
El programa correspondiente a esta cuestión se encuentra en el archivo *javaHilos4.java*. Este código es igual que el de la cuestión 2 salvo por una línea para cambiar la prioridad del primer hilo creado a 1:

```
Thread t1=new Thread (a);
Thread t2=new Thread (b);

t1.setPriority(Thread.MIN_PRIORITY);

t1.start();
t2.start();
```

La prioridad de los hilos en Java tiene que ver con el tiempo que el planificador le ceda al hilo para su ejecución. Es por ello que, en este ejemplo, la ejecución no varía respecto a la salida de la cuestión 2, pues es un programa simple donde los hilos no tienen que hacer cálculos complejos que tomen mucho tiempo:

```
user@user-VirtualBox:~/git/PConcurrente/p2$ java ThreadConPrioridad
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo1
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
hilo2
Fin del hilo principal
user@user-VirtualBox:~/git/PConcurrente/p2$
```

Sin embargo, si ejecutamos el programa varias veces (o de alguna otra forma que dificulte una ejecución sencilla y rápida del programa), podemos observar como el planificador sí puede dar prioridad al segundo hilo, que tiene una prioridad mayor que el primer hilo:



Aunque esta propiedad de los hilos en Java nos permita más libertad para manejar el planificador, la ejecución de los hilos con una prioridad menor no está garantizada, por lo que utilizar estas prioridades puede ser más complejo o puede darnos errores difíciles de solucionar.

Hilos en Python

Implementa en Python un programa que lance *5* hilos que se encarguen de actualizar una variable global compartida *50,000* veces cada uno. Si el programa funciona correctamente la variable (inicializada a *0*) debería acabar valiendo *250,000*. Ejecuta el programa varias veces y comenta los resultados que observes.

Para utilizar hilos en Python, utilizamos el módulo *threading*. Además, tendremos que hacer una función, en este caso llamada *thread()*, para indicarle a cada hilo que la ejecute, de forma similar a Java o C. Esta función aumentará el contador 50,000 veces, como se indica en el enunciado. Luego, ya en el método *main*, crearemos cada uno de los 5 hilos, cuyo objetivo será ejecutar la función *thread()* y los guardaremos en un vector. Después de crearlos y llamarlos, tendremos que esperar a cada uno de ellos con *join()* y mostrar el resultado final de la variable *contador*. Este código se encuentra en el archivo *pythonHilos.*py y la salida que obtenemos sería la siguiente:

```
user@user-VirtualBox:~/git/PConcurrente/p2$ python3 pythonHilos.py
Thread Thread-1 (thread)
Thread Thread-2 (thread)
Thread Thread-3 (thread)
Thread Thread-4 (thread)
Thread Thread-5 (thread)
Counter value: 250000 Expected: 250000

user@user-VirtualBox:~/git/PConcurrente/p2$
```

Como podemos observar en la captura de la salida, el resultado obtenido es el mismo que el que buscábamos, 250,000. Además, si ejecutamos varias veces el programa, podremos observar que el resultado obtenido siempre es correcto. Sin embargo, nosotros como programadores no hemos tomado ninguna medida para evitar que haya problemas de solapamiento entre los hilos al tratar con la variable *contador*. Haber obtenido un resultado correcto a pesar de esto es porque Python hace por defecto un lock a las variables globales, evitando así el solapamiento y obteniendo siempre una ejecución con un resultado correcto.

Hilos en Rust

Escribe un programa en Rust que haga algo similar al caso anterior (python). Ejecuta el programa varias veces y comenta los resultados que observes.

El caso de Rust, cuyo código podemos ver en el archivo *rustHilos.rs*, es distinto al de Python porque Rust no hace lock de variables globales, lo que dificulta que los resultados mientras tratamos variables globales con hilos sean correctos. Es por eso que utilizamos Arc, o *Atomicaly Reference Counted*. Esto nos permitirá crear varios punteros (y mantener la cuenta de estos punteros) los cuales apuntarán hacia un Mutex que, en este caso, inicializaremos a 0. Gracias a esto, cada vez que creemos un hilo para que aumente el valor de contador, clonaremos este puntero, el cual se "moverá" a la ejecución de cada uno de los 5 hilos que crearemos. De esta forma, cada uno de los hilos podrá apuntar a la variable *counter* (si no utilizásemos Arc, moveríamos la variable al primer hilo y entonces no estaría disponible para los siguientes) y esta variable no tendrá problemas de solapamiento al ser un Mutex. Por último, esperaremos a cada uno de los hilos con *join().unwrap()* y mostraremos el resultado, que igual que en el ejercicio de Python, debería ser siempre 250,000.

```
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$ ./rustHilos
Result: 250000
user@user-VirtualBox:~/git/PConcurrente/p2$
```