UA

ProgramaciónConcurrente

DanielAsensiRochDNI : **48776120C**

11 de octubre de 2022

Índice

1. Ejercicio 1: Java

Las formas vistas en clase han sido dadas en clase para la implementación de hilos y la creación de los mismos en Java han sido heredando *.extends Thread"* y implementando Runnable *ïmplements runnable"*

Ejercicio: 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.

1.1. Thread con Herencia

Se ha implementado el siguiente código para la realización del ejercicio implementando herencia:

```
public class ThreadConHerencia extends Thread{
    String palabra;
    int veces = 10;

    public ThreadConHerencia(String palabra){
        this.palabra = palabra;
    }

    public void run(){
        for(int i = 0; i < veces; i++){
            System.out.println(palabra);
        }
    }

    public static void main(String[] args){
        ThreadConHerencia hilo1 = new ThreadConHerencia("Hilo1");
        ThreadConHerencia hilo2 = new ThreadConHerencia("Hilo2");
        hilo1.start();
        hilo2.start();
    }

}
```

1.2. Thread con Runnable

Se ha implementado el siguiente código para la realización del ejercicio implementando runnable:

```
public class ThreadConRunnable implements Runnable{
    String palabra;
    int veces = 10;


    public ThreadConRunnable(String p){
        palabra = p;
    }

    public void run(){
        for(int i = 0; i < veces; i++)
        {
            System.out.println(palabra);
        }
    }

    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();

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

1.3. Comparación

Al llamar al método Start los hilos comenzarán a ejecutarse con su método Run, dando igual la implementación que realicemos ya que ambas implementan el mismo método.

El método de implementación más adecuado sería el Runnable ya que al java no aceptar herencia múltiple pero si múltiple implementación de interfaces, si implementamos la herencia con Thread no podremos realizar herencia con otra clase ya implementada la cual sea necesaria.

1.4. SetPriority()

Utilizando la función setPriority() con prioridad 1 al hilo1 le estamos diciendo que su prioridad sea menor a la del hilo2,por lo que en teoría el hilo2 se ejecutaría antes. Sin embargo, realizando varias ejecuciones se puede apreciar que no se cumple siempre.

```
public class ThreadConRunnable implements Runnable{
    String palabra;
    int veces = 10;


    public ThreadConRunnable(String p){
        palabra = p;
    }

    public void run(){
        for(int i = 0; i < veces; i++)
        {
            System.out.println(palabra);
        }
    }

    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.setPriority(1);

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

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

2. 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.

```
import threading


THREADS = 5
```

```
MAX_COUNT = 50000
EXPECTED = 250000

counter = 0

def thread():
    #Contador globa
    global counter
    print(format(threading.current_thread().name))

    for i in range(MAX_COUNT):
        counter += 1

def main():
    #Vector para almacenar los hilos
    threads = []

    #Bucle para clear los hilos
    for i in range(THREADS):
    #Creamos un nuevo hilo y lo metemos al vector
        t = threading.Thread(target=thread)
        threads.append(t)

        #Comienza la ejecuci n del hilo
        t.start()


    #Espera a que todos los se terminen
    for t in threads:
        t.join()


    #Imprimimos los valores de los contadores de cada hilo
    print("Valor del contador: {} Esperado: {}\n".format(counter, EXPECTED))

if __name__ == "__main__":
    main()
```

Tras varias ejecuciones se observan resultados incorrectos debido a que existe un problema de sincronización al intentar acceder al mismo recurso (variable global).

3. Rust

Implementa en Rust 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

Para trabajar con hilos en Rust podemos recurrir a una combinación de Mutex y Arc

```
use std::thread;
use std::sync::{Mutex,Arc};

fn main(){
    //Valores esperados para el computo final
    let expected = 250000;
    let counter = Arc::new(Mutex::new(0));
    let mut handles = vec![];

    for i in 0..5 {
        let counter = counter.clone();
        let handle = thread::spawn(move || {

            //Necesitamos hacer un lock para acceder a los datos
            let mut num = counter.lock().unwrap();
            for _ in 0..50000 {
                *num += 1;
            }
```

```
19              println!("Hilo {}: Contador: {}", i, *num);
20          });
21          handles.push(handle);
22      }
23
24      for handle in handles {
25          handle.join().unwrap();
26      }
27      println!();
28      println!("Valor contador: {}\nEsperado: {}", *counter.lock().unwrap(), expected);
29  }
```

Al contrario que en python, rust si que obtiene los resultados esperados en todo momento.