Práctica 7

Ejercicio 1.

Implemente en Kotlin una aplicación que muestre la hora actual en consola y la actualice cada 1 segundo. Evalúe distintos mecanismos para hacerlo.

Ejercicio 2.

Analice el siguiente código Kotlin y responda:

```
class TestSynchronized(id: String) : Thread(id) {
    var frase: Array<String> = arrayOf("UNLP", "PÚBLICA", "AHORA", "Y", "SIEMPRE")
    override fun run() {
        synchronized(System.out) {
            for (palabra in frase)
                println("${this.name} : ${palabra} ")
        }
    }


}
fun main(args: Array<String>) {
    val t1 = TestSynchronized("Thread 1")
    val t2 = TestSynchronized("Thread 2")
    val t3 = TestSynchronized("Thread 3")
    t1.start()
    t2.start()
    t3.start()
}
```

a) ¿Cuál es el efecto del synchronized(System.out)?

El bloque synchronized (System.out) {..} hace un lock sobre la salida de la consola, esto quiere decir mientras un hilo la mantenga bloqueada, los otros hilos no podrán imprimir en pantalla. Una vez que un hilo finaliza de imprimir en pantalla, otro hilo adquiere el lockeo. El contenido del bloque es el código que se debe ejecutar de manera sincrónica, es decir, sin verse interrumpida por otro hilo.

b) ¿Qué tipo de lock hace el código dado?

Es un lockeo a nivel de objeto. El objeto contiene un lock único llamado monitor. Cuando se usa synchronized, el thread adquiere el monitor de System.out, que evita que otros threads entren al bloque sincronizado hasta que se libere el monitor.

synchronized:

Executes the given function block while holding the monitor of the given objectl ock.

Ejercicio 3.

Implemente una aplicación que simule una carrera de 100 metros, donde cada participante está representado por un objeto thread. Para ello, cree un programa que muestre por consola la cantidad de metros recorrida por cada corredor.

a) Use un ejecutor con un pool de tamaño 5 para ejecutar. Luego cambie el tamaño del pool a 3 y observe la ejecución de los threads.

   Cuando el tamaño es 3, los hilos avanzan de forma más intercalada.

b) Supongamos que se quiere saber si un corredor abandona la carrera, retornando algún valor predefinido o en el peor de los casos, disparando una excepción. Analice la interface Callable, usando la documentación de la API y observe sus ventajas.

   Es una tarea que devuelve un resultado y puede lanzar una excepción. Los implementadores definen un único método sin argumentos llamado call. La interfaz Callable es similar a Runnable, ya que ambas están diseñadas para clases cuyas instancias pueden ser ejecutadas por otro hilo. Sin embargo, un Runnable no devuelve un resultado y no puede lanzar una excepción verificada.

   En el Execute Service el método submit extiende el método base java.util.concurrent.Executor#execute(java.lang.Runnable) creando y devolviendo

un Future que puede ser usado para cancelar la ejecución y/o esperar a que finalice.

El ejecutor soporta una clase de tareas llamada Callable, que devuelve un resultado y puede lanzar una excepción. Esta interfaz funcional pertenece al paquete java.util.concurrent. Kotlin convierte automáticamente las interfaces funcionales a lambdas.

Cuando se envia un Callable al executor service, este no puede devolver el resultado directamente, ya que el método submit() no espera hasta que la tarea se complete. En su lugar, el ejecutor devuelve un objeto especial llamado Future, que envuelve el resultado real que puede que aún no exista. Este objeto representa el resultado de una computación asincrónica (tarea). Hasta que la tarea se complete, el resultado real no está presente en el future. Para comprobarlo, se puede usar la propiedad isDone. Es muy probable que devuelva false si la consultas inmediatamente después de obtener un nuevo future.
Se puede recuperar el resultado de un future solo usando el método get(). Este devuelve el resultado cuando la computación ha finalizado; en otras palabras, bloquea el hilo actual y espera el resultado. Este método puede lanzar dos excepciones: ExecutionException e InterruptedException.

https://hyperskill.org/learn/step/21787
https://developer.android.com/reference/kotlin/java/util/concurrent/ExecutorService