Programación de Objetos Distribuidos

Trabajo Práctico 1: Threads

Cuerpo docente

Benítez, Mariano Miguel Guillermo

Goldberg, Daniel

Alumno

Romarión, Germán Rodrigo (Legajo: 51296)

Organización general de carpetas y archivos

Los ejercicios de este trabajo práctico fueron organizados en una jerarquía de paquetes cuya raíz es *ar.edu.itba*.
Cada ejercicio cuenta con un paquete padre denominado *.ejn*, donde *n* es el número de ejercicio. Dentro de cada uno de estos paquetes se encuentran clases que se usarán en común para el desarrollo de los ítems del ejercicio. A su vez, este último paquete nombrado posee paquetes que contienen las clases con los métodos *main* para ejecutar cada uno de los ítems del ejercicio en cuestión.

Ejercicio 1

c. Funcionalmente no se aprecia diferencia alguna entre extender la clase *Thread* e implementar la interfaz *Runnable*. Sin embargo, se requiere de una entidad extra que corra el código de la implementación de *Runnable*. En mi caso, lo que hice fue crear un thread, al cual le paso como parámetro en el constructor al runnable, para que al llamar al método *start()* del thread ejecute el método *run()* en un hilo de ejecución separado.

En términos del lenguaje, si extiendo la clase thread solamente para sobrescribir el método *run()* entonces probablemente sea mejor idea implementar runnable. Además con dicha interfaz, puedo correr la tarea desde distintos medios, ya sea pasando la implementación como parámetro a un thread, o a través de un *Executor*, con lo cual ofrece mucha más flexibilidad.

Ejercicio 2

a. Si cada una de las tareas imprimiera el resultado de la variable compartida al final de su ejecución, entonces el resultado esperado sería:

Thread 1: 1000
Thread 2: 2000
Thread 3: 3000
Thread 4: 4000

Sin embargo, ni el orden de los hilos, ni los valores finales coinciden en la mayor parte de los casos con estos.

Cada uno de los hilos de ejecución no se instancia en el orden en el que fueron convocados en el código *Java*, si no cuando la *Virtual Machine* lo cree apropiado. Además, por tratarse de threads, los cuatro están ejecutándose "en paralelo", por lo tanto, acceden a la variable compartida a la vez. Si uno de los thread imprime el resultado final luego de haber incrementado *n* veces el valor de la variable, entonces podría pasar que, si *x* era el valor antes de que el thread la alterara, el resultado no fuera *x + n*, si no algo distinto (generalmente mayor a *x + n*), dado que mientras la tarea modificaba a la variable, había otra que hacía lo mismo en simultáneo.

Ejemplos de resultados obtenidos:

i)

Thread 1: 1000
Thread 2: 3000
Thread 3: 2000
Thread 4: 4000

ii)

Thread 1: 1000
Thread 2: 2823
Thread 3: 2000
Thread 4: 3000

Ejercicio 3

c. Cada una de las tareas que consumen cadenas de caracteres y cuentan las vocales requieren de cierto procesamiento de la información, con lo cual, la tarea que produce cadenas demora menos en producirlas respecto de lo que demoran en consumirlas los otros tipos de tareas.

Habiendo dicho esto, si hay muchas tareas que producen cadenas de caracteres, y muchas que las consumen, entonces las cadenas se irán acumulando en grandes cantidades, puesto que se producen a una velocidad mayor de lo que se consumen.

Ejercicio 4

b. El *deadlock* que generé en el ítem **a** de este ejercicio fue obtenido a partir de bloquear el acceso a dos objetos distintos en bloques *synchronized* anidados en un thread, y tratar de obtener acceso a dichos objetos desde otro thread pero pidiendo bloqueo para esos objetos en distinto orden, con lo cual los threads se quedan esperando uno al otro para que liberen al objeto de interés.

Para evitar esta situación lo que hice fue en ambos threads sincronizar las tareas bloqueando los objetos en el mismo orden, con lo cual cuando el primero quiera acceder al segundo objeto dentro del bloque syncrhonized anidado podrá hacerlo, ya que el segundo thread está esperando a que el primero libere al primer objeto para poder bloquear al segundo.

Ejercicio 5

a. Opté por usar el *ExecutorService* que cuenta con un pool de threads fijo para ejecutar las tareas asignadas. He decidido tomar esta decisión dado que en todos los ejercicios de este trabajo práctico se solicitan realizar los distintos ítems con un número fijo de hilos (Ej. 2_a: 4 threads; ej 3_b: 6 threads; etc…), con lo cual le especifico a la clase *Executors* cuál será la cantidad de hilos que correrán, y automáticamente se creará el pool con esa cantidad.