**Entregable #8.2 [ejercicios] [semana 16 Abr 2018]**

* **Objetivos específicos** que persigue la realización del trabajo.
  + - Practicar con el entorno de Scala sobre eclipse
    - Utilizar algunas de las características del lenguaje Scala diferentes de Java
    - Practicar con el modelo de threads de Scala y su sincronización básica (interbloqueos)
    - Monitor
* **Competencias**: CC14, CT7
* **Resultados de aprendizaje**: RA310, RA312, RA317

**Apellidos y nombre (los grupos son de dos estudiantes)**

|  |
| --- |
| Gutierrez Ospina Anderson |
| Altamirano Cargua Fabricio |

1. **Cree un nuevo Scala App** **llamado** SynchronizedNesting. Escriba el siguiente código:

**import** de.\_

**object** SynchronizedNesting **extends** App {

**import** scala.collection.\_

**private** **val** transfers = mutable.ArrayBuffer[*String*]()

**def** logTransfer(name: *String*, n: Int) = transfers.synchronized {

transfers += s"transfer to account '$name' = $n"

}

}

Un ArrayBuffer (lea la sección 17.1) es como cualquier array excepto que se pueden añadir/eliminar elementos. En el programa anterior se declara un mutable.ArrayBuffer cuyos elementos serán de tipo *String* y sin especificar longitud. El operador += añade un elemento al final del array. Observe que la variable transfers está declarada con **val** lo que implica que siempre apuntará al mismo array, aunque el array puede cambiar sus elementos, como lo denota la palabra clave mutable. Scala tiene colecciones mutables y colecciones inmutables. El método logTransfer establece un bloqueo sobre el objeto transfers para ganar acceso exclusivo a él y anotar la transferencia.

Dentro del objeto SynchronizedNesting, y después de la definición del método logTransfer, escriba las siguientes sentencias:

**class** Account(**val** name: *String*, **var** money: Int)

**def** add(account: Account, n: Int) = account.synchronized {

account.money += n

**if** (n > 10) logTransfer(account.name, n)

}

**val** jane = **new** Account("Jane", 100)

**val** john = **new** Account("John", 200)

**val** t1 = thread { add(jane, 5) }

**val** t2 = thread { add(john, 50) }

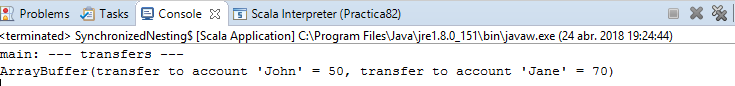
**val** t3 = thread { add(jane, 70) }

t1.join(); t2.join(); t3.join()

log(s"--- transfers ---\n$transfers")

Observe que la clase Account no tiene cuerpo. El compilador cogerá los parámetros de clase (name, money) como parámetros al constructor primario de la clase. Observe, también, que se crean tres threads que invocan la operación add. Esta operación establece un bloqueo sobre una instancia de la clase Account. El método add invoca al método logTransfer que a su vez establece un bloqueo sobre el objeto transfers. Hay anidamiento de bloqueos.

8.2.1Ejecute el programa y escriba el resultado



**2.- Cree un nuevo Scala App llamado** SynchronizedDeadlock.Escriba el siguiente código, que permite las transferencias entre dos cuentas dadas:

**import** de.\_

**object** SynchronizedDeadlock **extends** App {

**import** SynchronizedNesting.Account

**def** send(a: Account, b: Account, n: Int) = a.synchronized {

b.synchronized {

a.money -= n

b.money += n

}

}

**val** a = **new** Account("Jack", 1000)

**val** b = **new** Account("Jill", 2000)

**val** t1 = thread { **for** (i<- 0 until 100) {send(a, b, 1); log (s"send a to b: $i")}}

**val** t2 = thread { **for** (i<- 0 until 100) {send(b, a, 1); log (s"send b to a: $i")}}

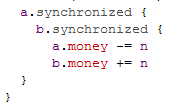
t1.join(); t2.join()

log(s"a = ${a.money}, b = ${b.money}")

}

8.2.2 Ejecute el programa varias veces hasta conseguir que el programa se quede bloqueado o usted se canse de intentarlo. Escriba a continuación la explicación de por qué el programa no se bloquea nunca o por qué se puede quedar bloqueado alguna vez.

Nunca se interbloque ya que el anaidamiento, de ocurrir siempre procede en el mismo orden.



**3.- Cree un nuevo Scala App llamado** SynchronizedPool.Escriba el siguiente código:

**import** de.\_

**import** scala.collection.\_

**object** SynchronizedPool **extends** App {

**private** **val** tasks = mutable.Queue[() => Unit]()

**object** Worker **extends** Thread {

setDaemon(**true**)

**def** poll() = tasks.synchronized {

**while** (tasks.isEmpty) tasks.wait()

tasks.dequeue()

}

**override** **def** run() = **while** (**true**) {

**val** task = poll()

task()

}

}

Worker.start()

**def** asynchronous(body: =>Unit) = tasks.synchronized {

tasks.enqueue(() => body)

tasks.notify()

}

asynchronous { log("Hello ") }

asynchronous { log("World!") }

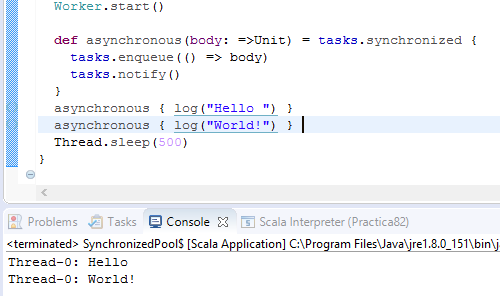
Thread.sleep(500)

}

La declaración tasks = mutable.Queue[() => Unit]() indica que tasks es una colección (Queue) mutable sin tamaño predeterminado y cuyos elementos son bloques de código (() => Unit) que no tienen parámetros de entrada y que no devuelven nada. La sentencia setDaemon(**true**)tiene que ver con la forma en la que la JVM se detiene. No tiene interés para la práctica actual, pero es necesaria para que el programa acabe.

Los objetos Scala (y objetos JVM, en general) soportan un par de métodos especiales llamados wait y notify que permiten esperar y despertar, respectivamente a threads. Solo se puede invocar a estos métodos en un objeto x si el thread actual está en posesión del monitor del objeto x. Cuando un thread T llama a wait en un objeto, libera el monitor y se queda bloqueado hasta que otro thread S llame a notify en el mismo objeto.

8.2.3 Describa con sus propias palabras qué hace este programa. No enumere las sentencias una a una. Indique cuál es el comportamiento global del programa. Por ejemplo, si fuera un programa de ordenación habría que indicar: “el programa realiza la ordenación de los números contenidos en la estructura e; ordena ascendentemente y deja el resultado en x. El algoritmo de ordenación es la burbuja”.



Lo está haciendo el programa es:

Crea una colección y define un método para encolar métodos, que es pasado como un parámetro en la función **asynchronous()** que este a su vez encola el método y cada vez que encola notifica que ha habido un cambio para que cualquier operación (método poll()) que esté esperando se despierte y desencole.