**Carátula para entrega de prácticas**

|  |  |
| --- | --- |
| Facultad de Ingeniería | Laboratorio de docencia |
|  |  |

Laboratorios de computación salas A y B

*Profesor:* M.I. Edgar Tista García

image.jpeg

*Asignatura:*

*Grupo:*

*No de Práctica(s):*

*Integrante(s):*

*No. de Equipo de cómputo empleado: 31-32*

Programación Orientada a Objetos

3

12

Velasco Vanegas Ricardo Alonso

Félix Flores Paul Jaime

*Semestre:* 2019-2

*Fecha de entrega: 18/Mayo/2019*

*Observaciones:*

CALIFICACIÓN: \_\_\_\_\_\_\_\_\_\_

Objetivo

Comprender el concepto y uso de hilos en programación orientada a objetos y entender su comportamiento/sintaxis a base de ejemplos y ejercicios prácticos.

Desarrollo y análisis

Ejercicios de laboratorio

Ejemplo de laboratorio 1:

En este ejemplo se crea una clase que hereda de “Thread”. Después se crea un constructor el cual recibe como parámetro una variable “nombre” de tipo String, y en el constructor se manda a llamar a “super(nombre)” para asignar tal valor recibido.

De igual forma, se crea un método “run” el cual con un ciclo for que se repite 5 veces, se manda a imprimir en pantalla el número de iteración acompañado del hilo al que pertenece. Tras terminar tal ciclo, se imprime en pantalla un mensaje que menciona cuando el hilo termina.

Ya dentro del main, se crean 2 instancias de “Hilo”, cada una recibiendo como parámetros una oración “String” la cual menciona cuando es el primer o segundo hilo, acompañado del método “.start()”.

Después de tales operaciones se imprime en pantalla el fin del hilo principal.

Ya durante la ejecución del programa, aparece primero en pantalla el fin del hilo principal, y a continuación empieza el ciclo for el cual imprime en pantalla primero la iteración 1 del segundo hilo, y después la iteración 1 del primer hilo. Repite este proceso hasta llegar a la 5 iteración y finalmente imprime en pantalla que se han terminado la segunda y la primera iteración (En ese orden).

Esto denota como los hilos empiezan desde el último creado hasta el primero en cuanto a prioridad, y de cómo ambos realizan acciones de forma paralela durante el mismo ciclo for.

Ejemplo de laboratorio 2:

En este ejemplo se maneja de forma similar al anterior, solo que ahora en la clase principal se le agrega “implements Runnable”, la cual funciona como interface y se le agregan los métodos de tal.

A continuación, se crea un método que no devuelve ningún valor llamado “run”, y dentro de este se corre un ciclo for 5 veces el cual menciona el número de iteración del hilo llamado.

Ya en el programa main se crean nuevas variables “Thread” las cuales tienen como nombre “Primer hilo” y “Segundo Hilo” respectivamente, los cuales son mandados a llamar y a “correr” con “.start”, el cual usa el método de run anteriormente creado para hacer los ciclos continuamente.

Ejemplo de laboratorio 3:

En la clase inicial se define como hija de la clase “Thread” por medio de extends.

Tras esto, en el constructor inicial de la clase, se reciben como parámetros “g” de tipo ThreadGroup y “n” de tipo String, los cuales asignan tales valores de la variable en la clase padre por medio de super(g,n).

Ahora, al definir el método “run”, hacemos un ciclo for que se repite 10 veces e imprime en pantalla el resultado del método “getName()”.

Ahora, un nuevo método de nombre listarHilos es creado, el cual es estático y recibe como parámetro una variable de nombre “grupoActual” de tipo ThreadGroup.

En tal método, se declara una variable llamada “numHilos” de tipo int, y una lista de hilos.

Ahora, a la variable de numHilos se le asigna el valor de la cuenta activa de hilos actual por medio del método “activeCount”, y después la listaDeHilos es creada, con un Thread el cual recibe como parámetro la variable de numHilos.

Ahora se manda a llamar al método .enumerate el cual recibe como parámetros a la lista de hilos pasada.

Despues, se manda a imprimir en pantalla el número de hilos activos, el cual es la cantidad de la variable numHilos, y finalmente se crea un ciclo for el cual se repite un número de veces igual al número de hilos activos. Tal ciclo for lo que hace es imprimir en pantalla el hilo activo junto con su nombre ubicado dentro de la lista de hilos.

Empezando con el main, se crea un grupo de hilos con el método ThreadGroup, el cual se implementa 5 veces con 5 diferentes hilos, en donde se menciona que el hilo 1 es el que tiene la máxima prioridad. Esto ocurre gracias al método setPriority. Tal método es igual usado para colocar a la prioridad del grupo en general a la normal.

A continuación, solo se imprime en pantalla la prioridad del grupo y la de cada hilo por separado, además de empezar el runnable de cada hilo.

Ejemplo de laboratorio 4:

En este ejemplo se muestra una aplicación práctica del uso de hilos en una situación real.

Antes de empezar con el programa, la clase principal se convierte en subclase de Thread, además de declarar una variable estatica de nombre “saldo”, la cual es de tipo entero.

Tras declarar al constructor inicial de la clase, se define ahora el método de “run”. En tal método, con una variable if se busca saber si el nombre del hilo recibido es Deposito 1 o Deposito 2. Si se encuentra que el nombre es alguno de estos dos, se manda a llamar al método depositar dinero con un parámetro de 100. En caso de no ser ese el nombre, se manda a llamar al método extraer dinero.

En el método para depositar dinero, se agrega la cantidad recibida en la variable estática de “saldo”, y se menciona el depósito. En el método para retirar dinero, primero se verifica que el saldo sea mayor a 0. En caso de que esta verificación ocurra satisfactoriamente, se le resta al saldo la cantidad de dinero sacada y se imprime en pantalla la cantidad retirada más el saldo restante.

Ya al empezar en el main, se empiezan los hilos para realizar las operaciones acorde a su nombre.

Ejercicio 1

En este programa se hace un ejemplo de 3 diferentes hilos los cuales corren de forma contigua. En la clase “Hilo2”, la cual tiene como interface a Runnable, se empieza creando una variable “hilo” de tipo Thread.

En el método crearYComenzar, el cual recibe como parámetro un nombre, se crea una variable “miHilo” de tipo MiHilo2, la cual gracias al constructor inicial se le asigna el nombre recibido en el parámetro anterior, y empieza a correr por el método start.

En el método “run”, primero se imprime en pantalla como empiezan a correr los hilos. Después, en un ciclo for (El cual esta dentro de un try-catch por si se interrumpe) se inicia recursivamente 10 veces en donde se mencionan los hilos, con apoyo del método “sleep” para que se pueda ver como el programa espera cierto tiempo para aparecer en pantalla donde se imprime el valor del ciclo for en cada uno de los hilos.

Tras terminar el ciclo for, conforme van terminando los hilos, se imprime en pantalla que estos han terminado. Cabe denotar que la velocidad de estos hilos no depende enteramente del programa, ya que al usar múltiples recursos distintos del sistema estos pueden imprimirse en diferentes partes de tiempo, como puede apreciarse en la prueba de compilación donde en el momento que terminó el hilo #3, el hilo #1 todavía no había terminado.

Si modificamos el valor dentro de sleep(), lo que el programa hace es que el ciclo de los hilos decida esperar cierta cantidad de tiempo, lo cual provoca que el resultado final al esperar cada hilo varié dependiendo de la velocidad de cada hilo.

En el inicio de la clase MiHilo2, esta permite correr hilos por medio de “implements Runnable”, sin embargo existe una segunda manera de permitir correr los hilos, y es haciendo que la clase sea una subclase de Thread, lo cual daría a lugar que sea “extends Thread”.

Tambien cabe añadir que en el constructor, no es necesario que reciba un valor de tipo String. También es posible que reciba un valor entero el cual será asignado como el valor de un hilo.

Ejercicio 2

En este ejercicio primero se empieza con la creación de 5 hilos distintos. Uno con prioridad alta, otro con prioridad baja y otros 3 con prioridad normal. Esto se logra por medio de una segunda clase de nombre “PrioridadHilos” la cual tiene como interface a Runnable y tiene un constructor en el cual al recibir un nombre de un hilo este es creado.

Continuando en el main, se colocan las prioridades a los hilos creados por medio de “setPriority”, en donde se recibe como parámetro el “NORM\_PRIORITY” con una suma de valores enteros en la cual se obtienen unos como de mayor prioridad y otros como menor dependiendo de su valor como tal.

A continuación, se mandan a llamar los hilos con apoyo de “join”. Este método lo que hace es que permite a los hilos a ponerse en espera hasta que otro hilo termina su ejecución. Esto se puede notar en el ciclo do-while del método run, en donde cada hilo cuenta de forma indefinida dependiendo de su prioridad.

Esto se nota en el final del programa, en donde aparece en pantalla cuantas veces el ciclo fue repetido para cada hilo. Se puede notar que el hilo de máxima prioridad llegó hasta la cantidad máxima, mientras que los otros hilos no llegaron a tal cantidad.

Si el método “join” no se encontrara en el programa, los hilos correrían al mismo tiempo, lo cual en este caso puede que provoque una saturación en la memoria puesto que muchos de los hilos trabajan al mismo tiempo, lo cual provocaría que estos no funcionen correctamente.

Al eliminar las líneas de código en donde se coloca la prioridad de los hilos, se puede notar como el valor de estos cambia drásticamente, ya que al no existir prioridad entre hilos, estos funcionan como si fueran todos iguales, lo cual da resultados más balanceados para el ciclo contador.

Ejercicio 3

El programa empieza con una lista “a” en donde se colocan los números del 1 al 13. Después, en una clase aparte llamada “MiHilo” se define el constructor el cual recibe una variable String y una variable numérica, y se crean dos hilos los cuales empiezan gracias al constructor. Estos dos hilos son #1 y #2, y a su vez ambos reciben como parámetro la lista numérica.

Después de esto, se manda un try catch en donde se intentan correr tales hilos con el comando “join” para que lo hagan por separado. Si ahora nos vamos a la clase “MiHilo”, dentro del método “run”, notamos que está la instrucción de “synchronized(sumArray)”, la cual es un método de la clase “Suma”.

Tal método lo que hace es iniciar un ciclo for el cual se repite un número de veces igual al largo de la lista de números pasada. Dentro de tal ciclo, se va aumentando de forma constante el valor de una variable “sum” y se imprime en pantalla, lo cual hace que vaya aumentando el valor dependiendo del valor de la lista (Primero se suma 1, después 2, después 3, etc).

Tal valor tras regresar del método, imprime en pantalla la suma para el segundo hilo, y ahora se vuelve a repetir la operación para el segundo hilo en el main.

En este ejemplo se puede denotar como la sincronización es importante ya que ambos hilos usan los mismos datos como recurso (En este caso, la lista) y si no son sincronizados puede que existan problemas en la suma y se den resultados diferentes para ambos hilos (Aun cuando las operaciones hechas son las mismas).

Tal sincronización puede ser usada de igual manera en programas que usen los mismos objetos, métodos, bases de datos, listas, etc. Conviene que se sincronicen ya que esto garantiza una salida de datos más precisa en donde no se corre el grande riesgo de que los recursos usados por la computadora causen errores al ambos leer la misma información en momentos diferentes.

Ejercicio 4

En programación, el problema de productor-consumidor consiste básicamente de dos procesos, los cuales son conocidos como “productor” y “consumidor” respectivamente. El proceso de “productor” genera un producto y lo almacena, mientras que el consumidor toma tal producto para su uso. Estos dos procesos al ocurrir al “mismo tiempo”, provocan que puedan existir problemas en los hilos en caso de que no exista la sincronización, ya que ambos procesos están usando un mismo objeto al mismo tiempo. El productor no puede añadir más productos de lo que la capacidad le permita y el consumidor no puede obtener el producto en caso de que este no exista.

Tal problema puede ser resuelto y aplicado en diversos ámbitos de la vida cotidiana. Algunos ejemplos de aplicación de esta metodología son:

* Una fábrica de bienes los cuales son creados y después “tomados” por un exportador
* Un sistema de agricultura el cual exporta una lista de alimentos los cuales son tomados y llevados a otro lado
* En una empresa el trabajo de un empleado el cual tras ser creado es tomado por un “supervisor” el cual corrige y aprueba/deniega al producto.

Estos tres ejemplos dados pueden ser aplicados en un programa de aplicación de programación. Existen muchos otros casos en donde puede ser aplicado, y en el ámbito de programación la gran mayoría de estos pueden ser planteados y resueltos.

En el ejercicio práctico anexado se puede notar como existen diferentes clases de hilos los cuales manejan el mismo producto. Existe un productor (El granjero), varios consumidores (Las empresas), y el producto como tal (El cual se almacena en el contenedor).

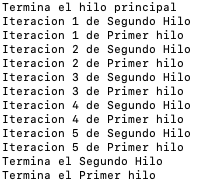
Mientras el productor crea e ingresa en la variable estática diferentes valores al azar de números de verduras, un hilo de una empresa elegida al azar va a manejar la misma información que fue producida por el granjero, usando de igual forma la clase de “Contenedor” como base para decir que compró las verduras.

Este es un ejemplo de “Productor-consumidor” ya que el granjero crea información de datos los cuales son usados en consiguiente por un hilo distinto de forma sincronizada (Esto quiere decir que tal información no puede ser usada hasta que el hilo que la está usando termine sus operaciones con esta).

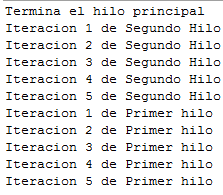
De esta forma, cuando el granjero termina de manejar la información del contenedor de verduras, empieza un hilo de la empresa el cual realiza operaciones para comprarlas. Tras terminar tal hilo del consumidor, se regresa al granjero el cual produce de nuevo verduras, y se regresa de nuevo al consumidor de forma continua.

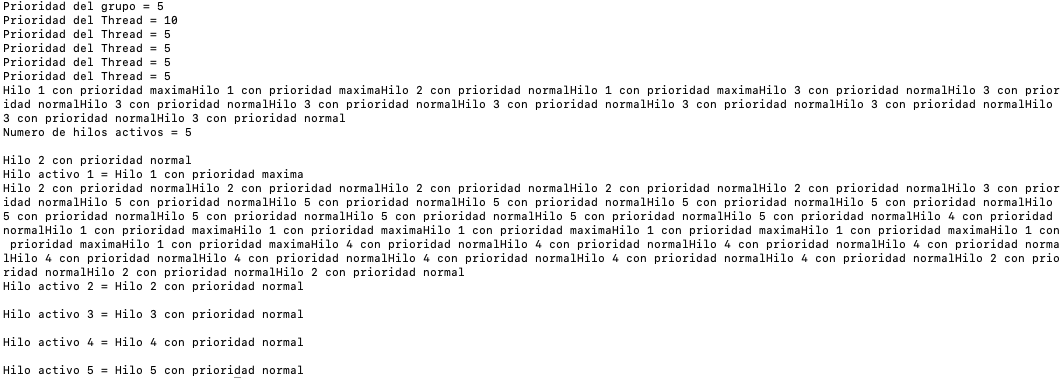
Evidencia de implementación

Ejemplo del laboratorio 1:

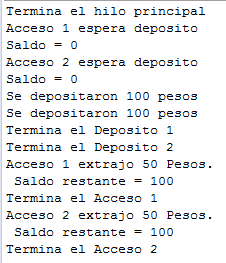


Ejemplo del laboratorio 2:

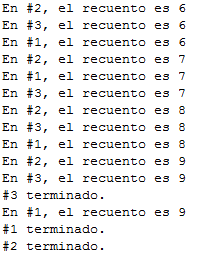
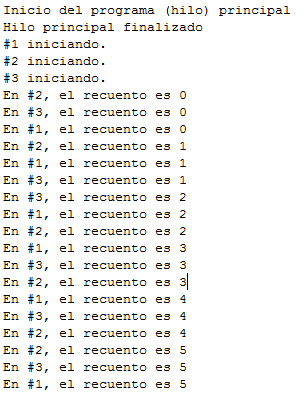


Ejemplo del laboratorio 3:

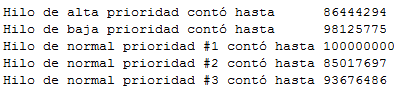
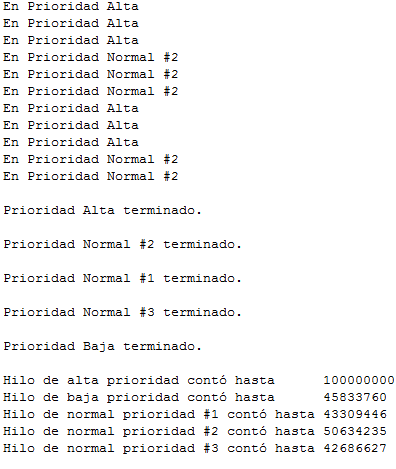
Ejemplo del laboratorio 4:



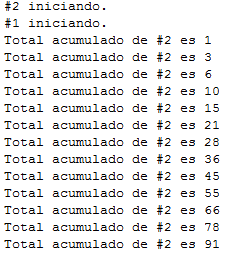
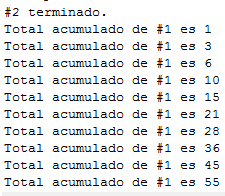
Ejercicio 1



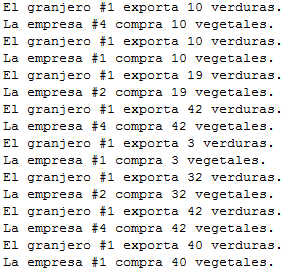
Ejercicio 2



Ejercicio 3

Ejercicio 4



Conclusiones

Referencias Usadas

**- Información teórica de la metodología de “Productor-Consumidor”:**

<https://es.wikipedia.org/wiki/Problema_productor-consumidor>

**- Información teórica de la metodología de “Productor-Consumidor” y base del ejercicio 4:**

<https://programaressencillo.wordpress.com/2014/11/25/java-monitores-ejemplo-productor-consumidor/>