



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE OAXACA

ASIGNATURA: TOPICOS AVANZADOS DE PROGRAMACIÓN

CATEDRÁTICO: HERNANDEZ ABREGO ANAYANSI CRISTINA

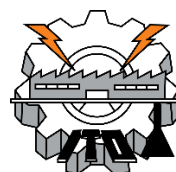
ALUMNOS:

- GARCÍA GARCÍA JOSÉ ÁNGEL
- SANCHEZ LOPEZ LAURA YESSSENIA

UNIDAD: 3

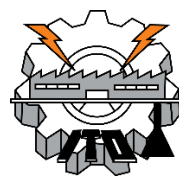
PRACTICA 3.1- HILOS

OAXACA DE JÚAREZ, OAX, 20/ABRIL/2020



Índice

BITACORA	3
CÓDIGO.....	6
PRUEBAS.....	13











BITACORA

El trabajo se comenzó a realizar desde el día sábado 11 de abril del 2020. Se utilizó como bitácora a GitHub.

History for [Temas](#) / [Unidad 3](#) / [practica_06](#)

Commits on Apr 11, 2020

PruebaExecutor	 c72a0df	
ChepeAicrag12 committed 4 hours ago		
ConSincronia	 f6607f3	
ChepeAicrag12 committed 4 hours ago		
SinSincronia	 4f761b1	
ChepeAicrag12 committed 4 hours ago		
InicioPractica3-1Hilos	 e116fba	
ChepeAicrag12 committed 6 hours ago		

- 1) Las inconsistencias presentadas en el programa sin sincronización como se muestra en la siguiente imagen.

La forma de agregar y quitar no se da de forma continua, es decir, así como se agrega se debe quitar y esto no pasa, puede que se agregue un elemento y después otros y hasta un lapso posterior se llegue a quitar el primer elemento agregado. Por ejemplo, al agregar el elemento M, este se consume mucho después de que se agrega a diferencia de X que es retirado una vez que ha sido agregado.

También noto la inconsistencia en que ningún momento el productor 1 agrega algo a la pila, aunque esto es error del código de productor.

De igual forma se observa que un mismo consumidor retira las letras recién agregadas y no da el acceso a los demás.

```
Output - practica_06 (run)
run:
Pila vacia, intentó retirar Thread-2
Productor 2 agregó X en hilo Thread-0
Productor 2 agregó M en hilo Thread-1
Pila vacia, intentó retirar Thread-4
Pila vacia, intentó retirar Thread-3
Hilo: Thread-3 Consumidor 3 : X
Productor 2 agregó N en hilo Thread-1
Hilo: Thread-3 Consumidor 3 : N
Hilo: Thread-4 Consumidor 3 : M
Pila vacia, intentó retirar Thread-2
Pila vacia, intentó retirar Thread-2
Pila vacia, intentó retirar Thread-3
Productor 2 agregó S en hilo Thread-1
Productor 2 agregó U en hilo Thread-0
Hilo: Thread-4 Consumidor 3 : U
Productor 2 agregó Q en hilo Thread-0
Hilo: Thread-3 Consumidor 3 : Q
Productor 2 agregó H en hilo Thread-1
Hilo: Thread-2 Consumidor 3 : H
Hilo: Thread-3 Consumidor 3 : S
Pila vacia, intentó retirar Thread-4
Productor 2 agregó F en hilo Thread-0
Hilo: Thread-3 Consumidor 3 : F
Pila vacia, intentó retirar Thread-3
```



```
Hilo: Thread-2 Consumidor 3 : Y
Hilo: Thread-3 Consumidor 3 : G
Pila vacia, intentó retirar Thread-4
Pila vacia, intentó retirar Thread-2
Pila vacia, intentó retirar Thread-3
Pila vacia, intentó retirar Thread-4
Productor 2 agregó a V en hilo Thread-1
Productor 2 agregó a Y en hilo Thread-0
Hilo: Thread-4 Consumidor 3 : Y
Hilo: Thread-4 Consumidor 3 : V
Productor 2 agregó a J en hilo Thread-0
```

a) Describe textualmente que hace la cláusula “synchronized”

Indica una zona restringida para el uso de los Threads, es como si tuviera un candado, esto permite que un solo hilo puede estar ejecutando lo que contiene esa parte restringida por un momento determinado y los otros hilos tendrán que esperar a que se termine la ejecución de esa parte. Con parte me refiero a un método completo o parte interna de este método.

b) Escribe otra forma de usar la cláusula “notify”

notifyAll() te permite notificar a todos los hilos de que ya ha terminado el proceso del Thread que invoca dicho método libereando así el método para ser usado por otro hilo. Funciona igual que notify(), la diferencia es que notify notifica arbitrariamente a un hilo y en cambio notifyAll lo hace para todos los hilos que se encuentren disponibles.

c) ¿Por qué crees que es necesario usar la cláusula “notify” en el método poner?

Es necesario para indicarle al hilo de consumidor de que ya se puede empezar a quitar elementos de la pila, sin esto el hilo de consumidor siempre estará en espera y no cambiará su estado. Aunque en este caso se notifica al otro hilo productor de que puede proceder a colocar otro elemento o en su caso el mismo hilo que acabo de colocar el elemento lo puede volver a hacer.

Además, que se usa en el método que tiene la cláusula de synchronized porque así permite al hilo que abandona dejarle su lugar a otro para continuar con su ejecución, esto porque solo un hilo puede realizar lo que contiene el método poner, esto funciona con ayuda de los métodos wait() y notify()/notifyAll()

d) ¿Por qué crees que se llega a una situación como la mostrada abajo (se muestra la parte final) ?, donde el programa sigue corriendo en forma indefinida

Porque se terminan las iteraciones del for, es decir, se termina de ejecutar los métodos de poner y quitar. Sabemos que va una iteración adelante los productores, por lo que en su última iteración los productores colocan los últimos elementos en la pila pero estos elementos son consumidos por una interacción anterior de los consumidores, entonces quedando así más iteraciones para los consumidores donde estos ejecutan al método quitar y la pila se encuentra ya sin elementos y los productores no vuelven a ejecutar el método poner, por lo que estos hilos se quedan en espera de que el productor coloque más elementos pero este ya



no lo podrá hacer porque no entrará al for y esto ocasiona que no se ejecuta otra vez el método poner y así estará el programa.

Se concluye con que si aumentamos el valor de iteraciones en el for se tendrá un programa más extenso pero que de igual forma llegará a ese punto, ya que está limitado.

e) Ventajas y desventajas de manejar los hilos con “ExecutorService” en lugar de manejarlos directamente.

Ventajas:

- La reutilización de hilos evita la necesidad de crear más hilos, porque esto es proceso altamente costoso además de la memoria que emplea cada hilo, con esto obtenemos un grado de mejoría.
- Los ejecutores se encargan de la gestión del procesamiento de las tareas (creación de hilos, el uso de start y join).
- El api se encarga de iniciar los Threads de la forma más correcta y con beneficio para nosotros.

Desventajas:

- No resultan necesarios cuando nuestro número de hilos creados se ajusta al número de núcleos de nuestra máquina, en este caso un ejecutor sería más dañino que beneficioso.

En nuestro caso como hacemos uso de `newCachedThreadPool()`, este crea tantos hilos nuevos como sea necesario, reutilizando los ya creados a medida que quedan libres y destruyendo los que están un tiempo sin actividad (`ThreadPoolExecutor`), que es muy cómodo para utilizar sin preocuparnos de la cantidad de hilos a utilizar.



CÓDIGO

CLASE CONSUMIDOR

```
/**
 * @author Sanchez Lopez Laura Yessenia
 * @author García García José Ángel
 */
public class Consumidor implements Runnable {

    private Pila pila;
    private static int numCons = 0;
    private int numC;

    public Consumidor(Pila p) {
        pila = p;
        numC = ++numCons;
    }

    @Override
    public void run() {
        char c;
        for (int i = 0; i < 20; i++) {
            c = pila.quitar();
            System.out.println("Hilo: " + Thread.currentThread().getName() + "
Consumidor " + numC + " : " + c);
            try {
                Thread.sleep((int) (Math.random() * 777));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```



CLASE PRODUCTOR

```
/**
 * @author Sanchez Lopez Laura Yessenia
 * @author García García José Ángel
 */
public class Productor implements Runnable {

    private Pila pila;
    private static int numProd = 0;
    private int numP;

    public Productor(Pila p) {
        pila = p;
        numP = ++numProd;
    }

    @Override
    public void run() {
        char c;
        for (int i = 0; i < 20; i++) {
            c = (char) (Math.random() * 26 + 65);
            pila.poner(c);
            System.out.println(" Productor " + numP + " agregó " + c + " en hilo "
+ Thread.currentThread().getName());
            try {
                Thread.sleep((int) (Math.random() * 777));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```



CLASE PILA

```
/**
 * @author Sanchez Lopez Laura Yessenia
 * @author García García José Ángel
 */
public class Pila {

    private int tope;
    private char[] datos;

    public Pila(int nd){
        datos = new char[nd];
        tope = -1;
    }

    public boolean llena(){
        return tope == datos.length-1;
    }

    public boolean vacia(){
        return tope < 0;
    }

    public synchronized void poner(char c){
        if (llena())
            System.out.println("Pila llena, intentó colocar " +
Thread.currentThread().getName());
        while(llena())
            try {
                this.wait();
            }
        }
    }
}
```




```

        } catch (InterruptedException e) {
            System.out.println("Error al poner");
        }
        tope++;
        datos[tope] = c;
        this.notify();
    }

    public synchronized char quitar(){
        char d = ' ';
        if (vacía())
            System.out.println("Pila vacía, en espera el hilo " +
Thread.currentThread().getName());
        while (vacía())
            try {
                this.wait();
            } catch (InterruptedException e) {
                System.out.println("Error al quitar xd");
            }
        d = datos[tope];
        tope--;
        this.notify();
        return d;
    }

    public char ver(){
        return !vacía() ? datos[tope] : ' ';
    }
}

```



CLASE PRUEBA

```
/**
 * @author Sanchez Lopez Laura Yessenia
 * @author García García José Ángel
 */
public class Prueba {

    public static void main(String[] args) {
        Pila pila = new Pila(10);
        Productor p1 = new Productor(pila);
        Thread prodT1 = new Thread(p1, "Hilo_1-productor");
        prodT1.start();
        Thread c1 = new Thread(new Consumidor(pila), "Hilo_1-consumidor");
        Thread c2 = new Thread(new Consumidor(pila), "Hilo_2-consumidor");
        Thread c3 = new Thread(new Consumidor(pila), "Hilo_3-consumidor");
        Productor p2 = new Productor(pila);
        Thread prodT2 = new Thread(p2, "Hilo_2-productor");
        prodT2.start();
        c1.start();
        c2.start();
        c3.start();
    }
}
```



CLASE PRUEBA EXECUTOR

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
/**
 * @author Sanchez Lopez Laura Yessenia
 * @author García García José Ángel
 */
public class PruebaExecutorService {
    public static void main(String[] args) {
        Pila pila = new Pila(10);
        Productor p1 = new Productor(pila);
        Productor p2 = new Productor(pila);
        Consumidor c1 = new Consumidor(pila);
        Consumidor c2 = new Consumidor(pila);
        Consumidor c3 = new Consumidor(pila);
        ExecutorService ejecutor = Executors.newCachedThreadPool();
        ejecutor.execute(p1);
        ejecutor.execute(p2);
        ejecutor.execute(c2);
        ejecutor.execute(c3);
        ejecutor.execute(c1);
        ejecutor.shutdown();
        try {
            boolean tareasTerminaron = ejecutor.awaitTermination(1,
TimeUnit.MINUTES);
            if (tareasTerminaron) {
                System.out.println("Todas las tareas terminaron");
            }else{
                System.out.println("Se agotoó el tiempo esperando a que las tareas
terminaran.");
            }
        }
    }
}
```



```
        System.exit(1);
    }
} catch (InterruptedException e) {
    System.out.println("Hubo una interrupcion mientras se esperaba a que
terminaran las tareas");
}
}
```



PRUEBAS

Ejecución sin sincronía.

```
Output - practica_06 (run)

run:
  Productor 2 agregó a O en hilo Thread-1
  Productor 1 agregó a I en hilo Thread-0
  Hilo: Thread-3 Consumidor 2 : I
  Pila vacia, intentó retirar Thread-4
  Hilo: Thread-2 Consumidor 1 : O
  Pila vacia, intentó retirar Thread-4
  Pila vacia, intentó retirar Thread-2
  Pila vacia, intentó retirar Thread-2
  Pila vacia, intentó retirar Thread-4
  Pila vacia, intentó retirar Thread-4
  Pila vacia, intentó retirar Thread-2
  Productor 1 agregó a O en hilo Thread-0
  Productor 2 agregó a H en hilo Thread-1
  Productor 1 agregó a B en hilo Thread-0
  Hilo: Thread-3 Consumidor 2 : B
  Hilo: Thread-4 Consumidor 3 : H
  Hilo: Thread-4 Consumidor 3 : O
  Pila vacia, intentó retirar Thread-2
  Pila vacia, intentó retirar Thread-4
  Productor 2 agregó a Y en hilo Thread-1
  Hilo: Thread-2 Consumidor 1 : Y
  Pila vacia, intentó retirar Thread-4
  Productor 1 agregó a H en hilo Thread-0
  Hilo: Thread-3 Consumidor 2 : H
  Pila vacia, intentó retirar Thread-3
```

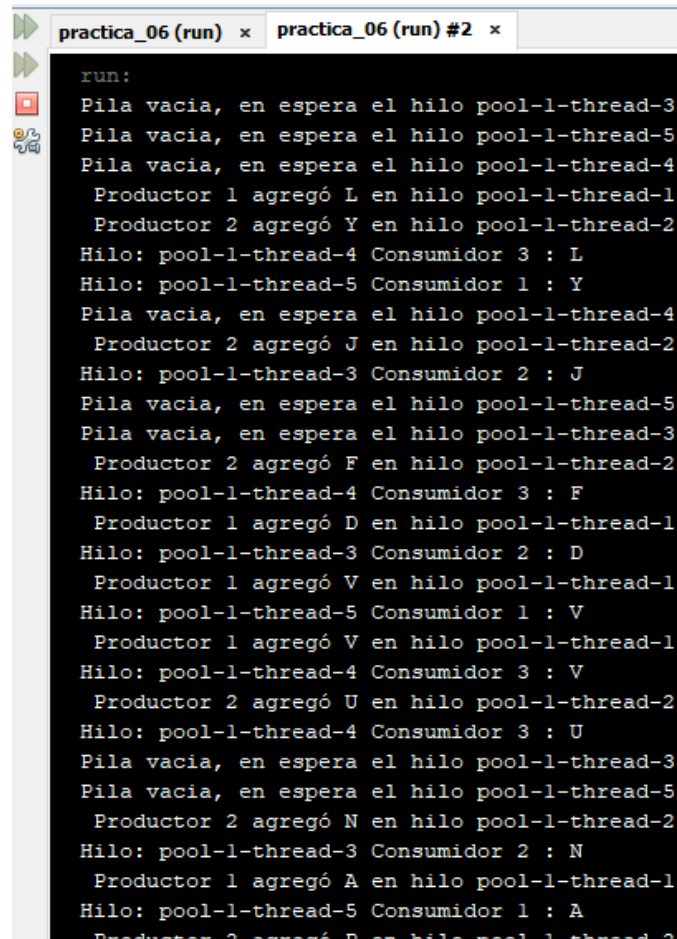
Ejecución con sincronía.

```
Output - practica_06 (run)

run:
  Pila vacia, en espera el hilo Hilo_1-consumidor
  Pila vacia, en espera el hilo Hilo_2-consumidor
  Productor 1 agregó X en hilo Hilo_1-productor
  Hilo: Hilo_3-consumidor Consumidor 3 : X
  Productor 2 agregó B en hilo Hilo_2-productor
  Hilo: Hilo_2-consumidor Consumidor 2 : B
  Pila vacia, en espera el hilo Hilo_2-consumidor
  Productor 2 agregó G en hilo Hilo_2-productor
  Hilo: Hilo_1-consumidor Consumidor 1 : G
  Pila vacia, en espera el hilo Hilo_3-consumidor
  Productor 1 agregó S en hilo Hilo_1-productor
  Hilo: Hilo_2-consumidor Consumidor 2 : S
  Productor 2 agregó F en hilo Hilo_2-productor
  Hilo: Hilo_3-consumidor Consumidor 3 : F
  Pila vacia, en espera el hilo Hilo_1-consumidor
  Productor 1 agregó G en hilo Hilo_1-productor
  Hilo: Hilo_1-consumidor Consumidor 1 : G
  Productor 2 agregó P en hilo Hilo_2-productor
  Hilo: Hilo_3-consumidor Consumidor 3 : P
  Productor 1 agregó Y en hilo Hilo_1-productor
  Productor 1 agregó U en hilo Hilo_1-productor
  Hilo: Hilo_3-consumidor Consumidor 3 : U
  Productor 2 agregó M en hilo Hilo_2-productor
  Productor 2 agregó G en hilo Hilo_2-productor
  Hilo: Hilo_2-consumidor Consumidor 2 : G
```



Ejecución con Executor Service



```
run:
Pila vacia, en espera el hilo pool-1-thread-3
Pila vacia, en espera el hilo pool-1-thread-5
Pila vacia, en espera el hilo pool-1-thread-4
  Productor 1 agregó L en hilo pool-1-thread-1
  Productor 2 agregó Y en hilo pool-1-thread-2
Hilo: pool-1-thread-4 Consumidor 3 : L
Hilo: pool-1-thread-5 Consumidor 1 : Y
Pila vacia, en espera el hilo pool-1-thread-4
  Productor 2 agregó J en hilo pool-1-thread-2
Hilo: pool-1-thread-3 Consumidor 2 : J
Pila vacia, en espera el hilo pool-1-thread-5
Pila vacia, en espera el hilo pool-1-thread-3
  Productor 2 agregó F en hilo pool-1-thread-2
Hilo: pool-1-thread-4 Consumidor 3 : F
  Productor 1 agregó D en hilo pool-1-thread-1
Hilo: pool-1-thread-3 Consumidor 2 : D
  Productor 1 agregó V en hilo pool-1-thread-1
Hilo: pool-1-thread-5 Consumidor 1 : V
  Productor 1 agregó V en hilo pool-1-thread-1
Hilo: pool-1-thread-4 Consumidor 3 : V
  Productor 2 agregó U en hilo pool-1-thread-2
Hilo: pool-1-thread-4 Consumidor 3 : U
Pila vacia, en espera el hilo pool-1-thread-3
Pila vacia, en espera el hilo pool-1-thread-5
  Productor 2 agregó N en hilo pool-1-thread-2
Hilo: pool-1-thread-3 Consumidor 2 : N
  Productor 1 agregó A en hilo pool-1-thread-1
Hilo: pool-1-thread-5 Consumidor 1 : A
  Productor 2 agregó D en hilo pool-1-thread-2
```

Terminando este ultimo de la forma como se muestra en la imagen, esto porque el consumidor se queda esperando a que coloque más elementos en la pila pero esto no va a pasar.

```
Pila vacia, en espera el hilo pool-1-thread-3
  Productor 1 agregó A en hilo pool-1-thread-1
Hilo: pool-1-thread-4 Consumidor 3 : A
Pila vacia, en espera el hilo pool-1-thread-4
Se agotoó el tiempo esperando a que las tareas terminaran.
C:\Users\lucy\AppData\Local\NetBeans\Cache\8.1\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 minute 0 seconds)
```

