



---

---

**TECNOLÓGICO NACIONAL DE MÉXICO  
INSTITUTO TECNOLÓGICO DE OAXACA**

**TÓPICOS AVANZADOS DE PROGRAMACIÓN**

**PRÁCTICA TAP 3-1 HILOS**

**PRESENTA ESTUDIANTE DE LA CARRERA INGENIERÍA EN SISTEMAS  
COMPUTACIONALES:**

**LUIS MARTÍNEZ JOSÉ DANIEL  
ZARATE CARREÑO JOSÉ VALENTÍN**

**DOCENTE: HERNANDEZ ABREGO ANAYANSI**

**GRUPO: 4SA**

**HORA: 9:00- 10:00**

**Oaxaca de Juárez, Oax, 21 de abril de 2020.**

## **Introducción**

Una de las aplicaciones sobre manejo de hilos o subprocesos a nivel de programación es cuando se comparten datos y se realizan operaciones sobre ellos con diferentes procesos concurrentes, unos que intentan hacer una operación para agregar nuevos datos y otros que intentan retirarlos. En esta práctica se incluye un caso sobre este tipo de aplicaciones para apreciar y ejercitar con los elementos que intervienen en los procesos concurrentes que permiten controlar comportamientos anómalos como la inconsistencia, o espera indefinida para lograr los resultados correctos esperados.

## **Correlación con los temas y aplicación en el contexto**

Aquí se ponen en práctica los temas 3.3 y 3.4, que son utilizados en la aplicación a desarrollar, lo cual requiere de la comprensión del tema 3.1 donde se ve el concepto de hilo y respecto al temas 3.2 aunque no se ve explícitamente, se asume que se cuenta con la experiencia de trabajar con un solo flujo y se puede hacer la comparación al trabajar con varios flujos durante el desarrollo de esta práctica. El contexto a donde se aplica es en el ambiente de concurrencia a nivel de programación a través de java, utilizando una estructura de datos en memoria para su acceso y manipulación.

## **Material y equipo necesario**

- Equipo de cómputo: Laptop o PC
- Software: Cualquier IDE de java con una versión del jdk 1.7 o superior

## **Metodología**

Se parte de la exposición de un caso donde intervienen varios hilos que tienen dos tipos de procesos que comparten un conjunto de datos en memoria. Luego durante el desarrollo se plantea crear la aplicación en java. Se realizan dos versiones de la aplicación para que el estudiante vaya experimentando las diferencias cuando no se usa sincronía durante la concurrencia que cuando ésta es utilizada.

## DESARROLLO DE LA PRÁCTICA

Preparativos: Crea un proyecto simple java, puedes llamarlo practica\_06, dentro de él crea dos paquetes fuente, uno es para la versión sin sincronía y el otro para la versión con sincronía.

- I. Versión sin sincronía
  1. En el paquete fuente correspondiente crea la clase pila con el código mostrado ( aquí se le llama 'Pilas', donde la 's' indica que es sin sincronía)
  2. Creación de la clase productor
  3. Creación de la clase consumidor
  4. Creación de la clase prueba

## PRUEBA

```

: Output - Practica_06 (run)
run:
    Productor 2 agrego a W en el hilo Thread-1
    Productor 1 agrego a E en el hilo Thread-0
Hilo: Thread-3 Consumidor 3 :E
Hilo: Thread-2 Consumidor 3 :W
Pila vacia, intentó retirar Thread-4
Pila vacia, intentó retirar Thread-4
Pila vacia, intentó retirar Thread-3
Pila vacia, intentó retirar Thread-2
    Productor 2 agrego a Q en el hilo Thread-1
    Productor 1 agrego a I en el hilo Thread-0
Hilo: Thread-2 Consumidor 3 :I
Hilo: Thread-3 Consumidor 3 :Q
Pila vacia, intentó retirar Thread-4
    Productor 2 agrego a K en el hilo Thread-1
    Productor 1 agrego a J en el hilo Thread-0
Hilo: Thread-2 Consumidor 3 :J
Hilo: Thread-3 Consumidor 3 :K
    Productor 2 agrego a K en el hilo Thread-1
    Productor 1 agrego a Y en el hilo Thread-0
Hilo: Thread-4 Consumidor 3 :Y
Hilo: Thread-3 Consumidor 3 :K
Pila vacia, intentó retirar Thread-2
Pila vacia, intentó retirar Thread-3
Pila vacia, intentó retirar Thread-3
    Productor 2 agrego a J en el hilo Thread-1
Hilo: Thread-4 Consumidor 3 :J
    Productor 1 agrego a R en el hilo Thread-0
Hilo: Thread-3 Consumidor 3 :R
Pila vacia, intentó retirar Thread-2
Pila vacia, intentó retirar Thread-3
Pila vacia, intentó retirar Thread-4

```

## Código (Sin sincronía)

### Clase Pilas

```
package Version_1;

public class Pilas {

    private int tope;
    private char[] datos;

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

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

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

    public void poner(char c) {
        if (llena()) {
            System.out.println("Pila llena,intentó colocar " + Thread.currentThread().getName());
        } else {
            tope++;
            datos[tope] = c;
        }
    }

    public char quitar() {
        char d = ' ';
        if (vacía()) {
            System.out.println("Pila vacia, intentó retirar " + Thread.currentThread().getName());
        } else {
            d = datos[tope];
            tope--;
        }
        return d;
    }
}
```

```

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

```

## Clase Productor

```

package Version_1;
public class Productor implements Runnable {
    private Pilas pila;
    private int numProd;

    public Productor(Pilas p){
        pila=p;
    }
    public void setnumProd(int nu){
        numProd = nu;
    }
    public int getnumProd(){
        return 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 "+getnumProd()+" agrego a "+c+"
en el hilo "+Thread.currentThread().getName());
            try{
                Thread.sleep((int)(Math.random()*777));
            }catch(InterruptedException e){

            }
        }
    }
}

```

## Clase Consumidor

```
package Version_1;
public class Consumidor implements Runnable{
    private Pilas pila;
    private static int numCons=0;

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

    @Override
    public void run() {
        char c;
        for (int i = 0; i < 20; i++) {
            c = pila.uitar();
            if (Character.isAlphabetic(c)) {
                System.out.println("Hilo: " + Thread.currentThread().getName
() + " Consumidor " + numCons + " :" + c);
            }
            try {
                Thread.sleep((int) (Math.random() * 777));
            } catch (InterruptedException e) {
            }
        } //for
    }
}
```

## Clase Prueba

```
package Version_1;
public class PruebaproductorConsumidor {

    public static void main(String[] args) {
        Pilas pila = new Pilas(10);

        Productor p1 = new Productor(pila);
        p1.setnumProd(1);
        Thread prodT1 = new Thread(p1);
        prodT1.start();
    }
}
```

```

    Productor p2 = new Productor(pila);
    p2.setnumProd(2);
    Thread prodT2 = new Thread(p2);
    prodT2.start();

    Consumidor c1 = new Consumidor(pila);
    Consumidor c2 = new Consumidor(pila);
    Consumidor c3 = new Consumidor(pila);
    Thread c_1 = new Thread(c1);
    Thread c_2 = new Thread(c2);
    Thread c_3 = new Thread(c3);

    c_1.start();
    c_2.start();
    c_3.start();

}
}

```

## II. Versión con sincronía

1. Crea la clase pila en el paquete correspondiente, ahora se llamará Pila
2. Se crea el Método poner con las siguientes condiciones:
  - a) Tomar en cuenta que la pila puede estar llena, en ese caso el hilo se quedará en espera hasta que un consumidor retire un elemento de la pila.
  - b) Al agregar un elemento se debe hacer una notificación de que la pila tiene elementos (por lo menos uno) por si hubiera algún consumidor en espera.
3. . Crea la clase productor que es muy semejante al código del caso sin sincronía, solo que ahora se utiliza a un objeto de la clase 'Pila'.
4. Crea la clase consumidor que es muy semejante al código realizado sin sincronía, solo que este caso no se requiere usar la cláusula "if", pero sí lo que está dentro de ella (System.out.....).
5. En tu bitácora de seguimiento contesta lo siguiente:
  - a) Describe textualmente que hace la cláusula "synchronized"

Todos los bloques sincronizados en el mismo objeto solo pueden tener un hilo ejecutándose dentro de ellos a la vez. Todos los otros hilos que intentan ingresar al bloque sincronizado se bloquean hasta que el hilo dentro del bloque sincronizado sale del bloque.

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

Utilizar los métodos notify y notifyAll para informarles de que la condición cambió y se puede despertar. Tanto notify() y el método notifyAll() notifican, pero notify() envía una sola notificación sin garantizar que hilo será notificado y notifyAll() envía notificaciones a todos los hilos. Así que si sólo un hilo está esperando en un objeto bloqueado, también conocido como monitor, entonces tanto notify como notifyAll notificarán a dicho hilo.

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

Al agregar un elemento se hace una notificación de que la pila tiene elementos (por lo menos uno) por si hubiera algún consumidor en espera.

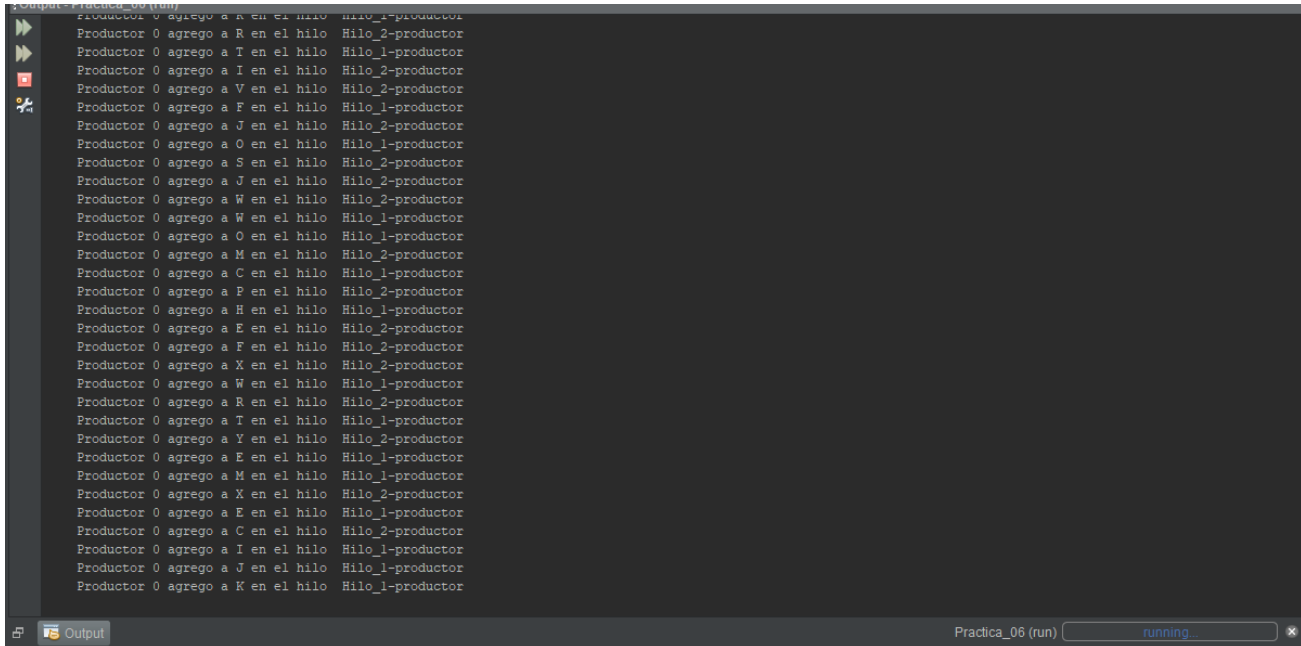
6. Cambia los nombres de los hilos a tu manera como se indica, haz una prueba para verificarlo.
7. Una alternativa de manejo de hilos es mediante un administrador de hilos disponible en java llamado “ExecutorService”.



## Prueba (con sincronía)

Sin utilizar “ExecutorService”

Se logra observar que el programa sigue corriendo en forma indefinida

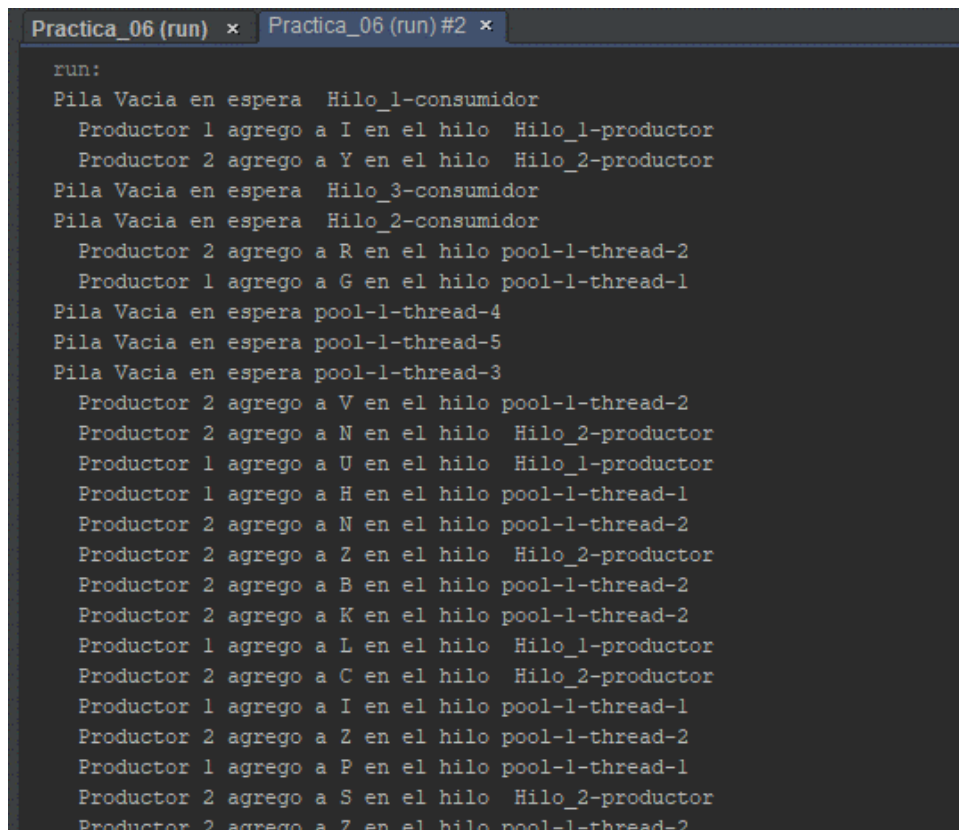


```
Practica_06 (run)
Productor 0 agrego a K en el hilo Hilo_1-productor
Productor 0 agrego a R en el hilo Hilo_2-productor
Productor 0 agrego a T en el hilo Hilo_1-productor
Productor 0 agrego a I en el hilo Hilo_2-productor
Productor 0 agrego a V en el hilo Hilo_2-productor
Productor 0 agrego a F en el hilo Hilo_1-productor
Productor 0 agrego a J en el hilo Hilo_2-productor
Productor 0 agrego a O en el hilo Hilo_1-productor
Productor 0 agrego a S en el hilo Hilo_2-productor
Productor 0 agrego a J en el hilo Hilo_2-productor
Productor 0 agrego a W en el hilo Hilo_2-productor
Productor 0 agrego a W en el hilo Hilo_1-productor
Productor 0 agrego a O en el hilo Hilo_1-productor
Productor 0 agrego a M en el hilo Hilo_2-productor
Productor 0 agrego a C en el hilo Hilo_1-productor
Productor 0 agrego a P en el hilo Hilo_2-productor
Productor 0 agrego a H en el hilo Hilo_1-productor
Productor 0 agrego a E en el hilo Hilo_2-productor
Productor 0 agrego a F en el hilo Hilo_2-productor
Productor 0 agrego a X en el hilo Hilo_2-productor
Productor 0 agrego a W en el hilo Hilo_1-productor
Productor 0 agrego a R en el hilo Hilo_2-productor
Productor 0 agrego a T en el hilo Hilo_1-productor
Productor 0 agrego a Y en el hilo Hilo_2-productor
Productor 0 agrego a E en el hilo Hilo_1-productor
Productor 0 agrego a M en el hilo Hilo_1-productor
Productor 0 agrego a X en el hilo Hilo_2-productor
Productor 0 agrego a E en el hilo Hilo_1-productor
Productor 0 agrego a C en el hilo Hilo_2-productor
Productor 0 agrego a I en el hilo Hilo_1-productor
Productor 0 agrego a J en el hilo Hilo_1-productor
Productor 0 agrego a K en el hilo Hilo_1-productor
Practica_06 (run) running...
```

## Prueba 2 (con sincronía)

Con “ExecutorService”

En la siguiente prueba en el caso de usar el administrador de hilos ExecutorService en este caso se administra el tiempo de ejecución del hilo así evitaremos un loop infinito como se mostró anteriormente.



```
Practica_06 (run) x Practica_06 (run) #2 x
run:
Pila Vacía en espera Hilo_1-consumidor
Productor 1 agrego a I en el hilo Hilo_1-productor
Productor 2 agrego a Y en el hilo Hilo_2-productor
Pila Vacía en espera Hilo_3-consumidor
Pila Vacía en espera Hilo_2-consumidor
Productor 2 agrego a R en el hilo pool-1-thread-2
Productor 1 agrego a G en el hilo pool-1-thread-1
Pila Vacía en espera pool-1-thread-4
Pila Vacía en espera pool-1-thread-5
Pila Vacía en espera pool-1-thread-3
Productor 2 agrego a V en el hilo pool-1-thread-2
Productor 2 agrego a N en el hilo Hilo_2-productor
Productor 1 agrego a U en el hilo Hilo_1-productor
Productor 1 agrego a H en el hilo pool-1-thread-1
Productor 2 agrego a N en el hilo pool-1-thread-2
Productor 2 agrego a Z en el hilo Hilo_2-productor
Productor 2 agrego a B en el hilo pool-1-thread-2
Productor 2 agrego a K en el hilo pool-1-thread-2
Productor 1 agrego a L en el hilo Hilo_1-productor
Productor 2 agrego a C en el hilo Hilo_2-productor
Productor 1 agrego a I en el hilo pool-1-thread-1
Productor 2 agrego a Z en el hilo pool-1-thread-2
Productor 1 agrego a P en el hilo pool-1-thread-1
Productor 2 agrego a S en el hilo Hilo_2-productor
Productor 2 agrego a Z en el hilo pool-1-thread-2
```

## Código (con sincronía)

### Clase Pila

```
package version_2;
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 void poner(char c) {
        if (llena()) {
            System.out.println("Pila llena, intento colocar " + Thread.currentThread().getName());
        }
        while (llena()) {
            try {
                this.wait();
            } catch (InterruptedException e) {
            }
            tope++;
            c = datos[tope];
            this.notify();
        }
    }

    public synchronized char quitar() {
        char d = ' ';
        if (vacía()) {
            System.out.println("Pila Vacía en espera " + Thread.currentThread().getName());
        }
    }
}
```

```

    }
    while (vacía()) {
        try {
            this.wait();
        } catch (InterruptedException e) {

        }
        d = datos[tope];
        tope--;
        this.notify();
    }
    return d;
}

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

```

## Clase Productor

```

package version_2;
public class Productor implements Runnable {
    private Pila pila;
    private int numProd;

    public Productor(Pila p){
        pila=p;
    }
    public void setnumProd(int nu){
        numProd = nu;
    }
    public int getnumProd(){
        return 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 "+getnumProd()+" agrego a "+c+"
en el hilo "+Thread.currentThread().getName());
        try{
            Thread.sleep((int)(Math.random()*777));
        }catch(InterruptedException e){

        }
    }
}
}
}
}

```

## Clase Consumidor

```

package version_2;
public class Consumidor implements Runnable{
private Pila pila;
private static int numCons=0;

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

    }

    @Override
    public void run() {
        char c;
        for (int i = 0; i < 20; i++) {
            c = pila.uitar();

            System.out.println("Hilo: " + Thread.currentThread().getName
() + " Consumidor " + numCons + " : " + c);

            try {
                Thread.sleep((int) (Math.random() * 777));
            } catch (InterruptedException e) {
            }
        } //for
    }
}
}

```

## Clase prueba con sincronía

```
public class PruebaconSincronia {

    public static void main(String[] args) throws InterruptedException {
        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);

        Thread prodT1 = new Thread(p1);
        prodT1.setName(" Hilo_1-productor");
        prodT1.start();
        Thread prodT2 = new Thread(p2);
        prodT2.setName(" Hilo_2-productor");
        prodT2.start();
        Thread prodT3 = new Thread(c1);
        prodT3.setName(" Hilo_1-consumidor");
        prodT3.start();
        Thread prodT4 = new Thread(c2);
        prodT4.setName(" Hilo_2-consumidor");
        prodT4.start();
        Thread prodT5 = new Thread(c3);
        prodT5.setName(" Hilo_3-consumidor");
        prodT5.start();
    }
}
```

## Clase Prueba (ExecutorService)

```
package version_2;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class PruebaconSincroniaExecuteService {

    public static void main(String[] args) throws InterruptedException {
        Pila pila = new Pila(10);
        Productor p1 = new Productor(pila);
        p1.setnumProd(1);
        Productor p2 = new Productor(pila);
```

```

        p2.setnumProd(2);
        Consumidor c1 = new Consumidor(pila);
        Consumidor c2 = new Consumidor(pila);
        Consumidor c3 = new Consumidor(pila);

        Thread prodT1 = new Thread(p1);
        prodT1.setName(" Hilo_1-productor");
        prodT1.start();
        Thread prodT2 = new Thread(p2);
        prodT2.setName(" Hilo_2-productor");
        prodT2.start();
        Thread prodT3 = new Thread(c1);
        prodT3.setName(" Hilo_1-consumidor");
        prodT3.start();
        Thread prodT4 = new Thread(c2);
        prodT4.setName(" Hilo_2-consumidor");
        prodT4.start();
        Thread prodT5 = new Thread(c3);
        prodT5.setName(" Hilo_3-consumidor");
        prodT5.start();

        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 agoto el tiempo esperando a que las tarea
s terminaran.");
                System.exit(1);
            }
        }
        catch (InterruptedException ex){
            System.out.println("Hubo una interrupcion mientras esperaba a que te
rminaran las tareas");}
    }
}

```