

## **Práctica 6. Creación de una aplicación con hilos que comparten procesos y datos.**

### **Competencia a desarrollar**

Crea subprogramas para resolver problemas concurrentes utilizando Multihilos.

### **Introducción**

Una de las aplicaciones sobre manejo del 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

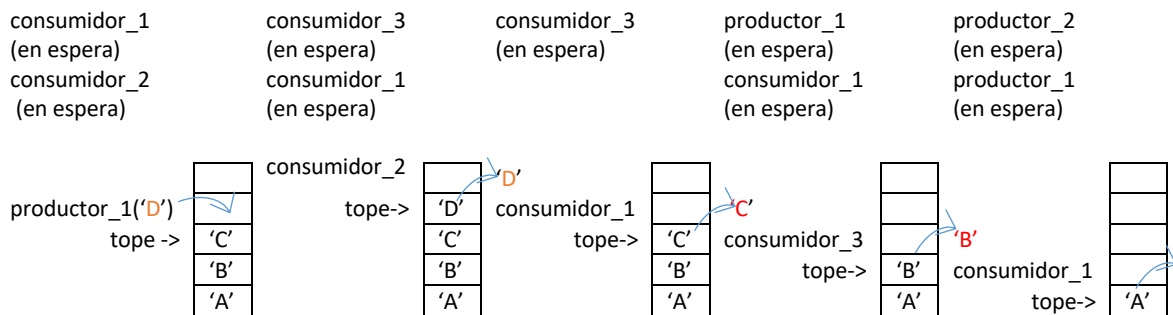
Software: Cualquier IDE de java con una versión del jdk 1.7 o superior

### **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. Donde aparece el signo “?” indica que el estudiante debe completar código o responder preguntas que debe anotar en su bitácora de seguimiento.

### Plantamiento del caso

Dado un conjunto de datos en memoria de tipo carácter organizados en una estructura de pila implementada con un arreglo, se tienen varios procesos que acceden a ellos en forma concurrente (el acceso es aleatorio) unos para agregar datos en ella y otros para retirarlos, los que agregan son designados como productores y los que retiran designados como consumidores, en el esquema siguiente se muestra un posible escenario esperado con una pila para cinco elementos, suponiendo que la pila ya cuenta con los datos 'A', 'B' y 'C', se tienen varios hilos en ejecución que se encuentran en espera o realizando alguna operación como se aprecia en el siguiente esquema. Los procesos que aparecen en la primera línea encima de los esquemas de la pila son los más recientes generados, cada esquema (columna) ilustra un estado de la pila:



La pila de caracteres corresponde a los datos compartidos y su acceso a ellos es mediante las operaciones clásicas de esta estructura:

- 'poner(carácter)' agrega un elemento de tipo carácter encima del tope de la pila.
- 'quitar()' retira el elemento que está en el tope de la pila y lo regresa.
- 'ver()' regresa el elemento que está en el tope si retirarlo.

La operación compartida depende del tipo de flujo o hilo, los hilos que producen comparten la operación de poner() y los hilos que consumen comparten la operación quitar().

Lo deseable al compartir los datos entre varios flujos o hilos, es que los hilos encargados de producir elementos los coloquen en la posición correcta del tope y los que consumen datos retiren el elemento que les corresponde. También se debe hacer que se cumpla lo siguiente:

- Evitar que un hilo en espera se quede sin ser atendido en el tiempo asignado de procesamiento, esto es evitar la inanición de un proceso.
- Evitar que se produzca un interbloqueo (dead-lock).
- Hacer que cuando la pila esté vacía y ésta llegue a ser accedida por un hilo de tipo consumidor, dicho proceso del hilo se quede en espera hasta que un proceso de tipo productor coloque un nuevo elemento, notificándose al proceso en espera (y a todos los que estén en esta circunstancia).
- Hacer que cuando la pila esté llena y ésta sea accedida por un proceso de un hilo de tipo productor, dicho proceso quede en espera hasta que un proceso de tipo consumidor retire

un elemento, notificándolo al proceso en espera (y a todos los que estén es esta circunstancia).

A continuación se muestra un fragmento de una ejecución de un programa en java cumpliendo con lo estipulado arriba. Se han creado dos hilos que ejecutan cada uno a un proceso productor y tres para ejecutar al proceso consumidor. Los hilos están identificados por un nombre que inicia con el texto `Thread` y un número consecutivo (se le puede asignar otro, si se desea). Dentro de cada proceso hay un ciclo encargado de agregar o retirar según el tipo de proceso varios elementos de tipo caracteres alfabéticos. Cuando la pila llega a estar vacía, los proceso de tipo consumidor se quedan en espera hasta que un productor agregue un elemento, entonces el consumidor en orden de espera obtiene el elemento recién agregado, tal como se aprecia en seguida:

```

run:
  Productor 2 agregó a Z en hilo Thread-1
  Hilo: Thread-2 Consumidor 3 :Z
  Productor 1 agregó a W en hilo Thread-0
  Hilo: Thread-4 Consumidor 3 :W
  Pila vacia, en espera el hilo Thread-3
  Pila vacia, en espera el hilo Thread-4
  Productor 1 agregó a B en hilo Thread-0
  Hilo: Thread-3 Consumidor 3 :B
  Pila vacia, en espera el hilo Thread-3
  Productor 2 agregó a S en hilo Thread-1
  Hilo: Thread-4 Consumidor 3 :S
  Pila vacia, en espera el hilo Thread-2
  Pila vacia, en espera el hilo Thread-4
  Productor 2 agregó a P en hilo Thread-1
  Hilo: Thread-3 Consumidor 3 :P
  Productor 1 agregó a Q en hilo Thread-0
  Hilo: Thread-2 Consumidor 3 :Q
  Pila vacia, en espera el hilo Thread-3
  Productor 2 agregó a O en hilo Thread-1
  Hilo: Thread-4 Consumidor 3 :O
  Pila vacia, en espera el hilo Thread-2
  Productor 1 agregó a D en hilo Thread-0
  Hilo: Thread-3 Consumidor 3 :D
  
```

## 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)

```
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(vacia())

        {
            System.out.println("Pila vacia, intentó retirar "+Thread.currentThread().getName() );
        }

        else {
            d =datos[tope];
            tope--;
        }
        return d;
    }
    public char ver()
    { if(!vacia())
        return datos[tope];
        return ' ';
    }
}
```

2. Creación de la clase productor

```
public class Productor implements Runnable{
    private Pilas pila;
    private static int numProd=1;
    public Productor(Pilas p)
    { pila=p;
      numProd++;
    }
    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 "+numProd +" agregó a "+c+ " en hilo "+ Thread.currentThread().getName());
            try {
                Thread.sleep((int)(Math.random()×777));}
            catch(InterruptedException e){}
        } //for
    }
}
```

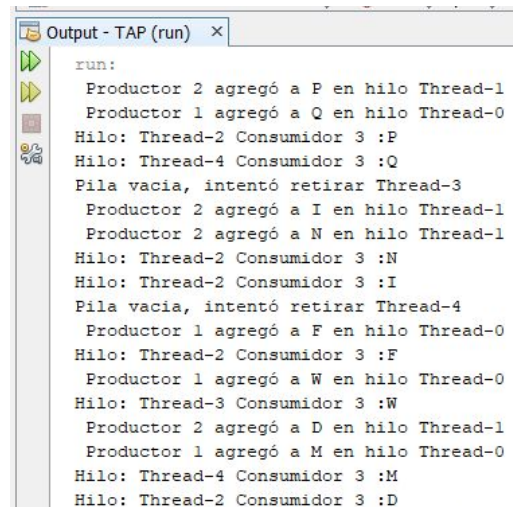
3. Creación de la clase consumidor

```
public class Consumidor implements Runnable{
    private Pilas pila;
    private static int numCons=0;
    public Consumidor(Pilas p)
    { pila=p;
      numCons++;
    }
    public void run(){
        char c;
        for(int i=0;i<20; i++){
            c= pila.quitar();
            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
    } // run
}
```

#### 4. Creación de la clase prueba

```
public class PruebaProductorConsumidor {
    public static void main(String[] args) {
        Pilas pila = new Pilas(10);
        Productor p1 = new Productor(pila);
        Productor p2 = new Productor(pila);
        Thread prodT1 = new Thread (p1);//
        prodT1.start();//
    }
}
```

5. ¿Crea el otro productor y los tres consumidores.
6. Haz una prueba de ejecución donde se aprecie inconsistencias tal como se muestra a continuación:



#### 7. Contesta en tu bitácora de seguimiento

- a) Señala los casos de inconsistencia

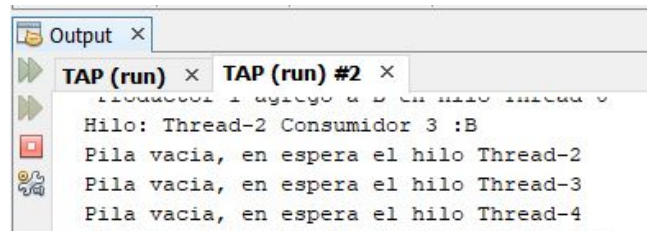
## II. Versión con sincronía

1. Crea la clase pila en el paquete correspondiente, ahora se llamará Pila

```
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;}
    // ? a Ti te corresponde hacer el metodo poner
    public synchronized char quitar() { //
    char d=' ';
    if(vacia())
        System.out.println("Pila vacia, en espera el hilo "+Thread.currentThread().getName() );
    while(vacia())
    try{
        this.wait();
    }
    catch(InterruptedException e){}
    d =datos[tope];
    tope--;
    this.notify();
    return d;
    }
    public char ver()
    { if(!vacia())
      return datos[tope];
    return ' ';
    }
}
```

2. ¿ Te corresponde crear el método poner(). Tienes que seguir las siguientes consideraciones:
  - 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"
  - b) Escribe otra forma de usar la cláusula "notify"
  - b) ¿Por qué crees que es necesario usar la cláusula "notify" en el método poner?

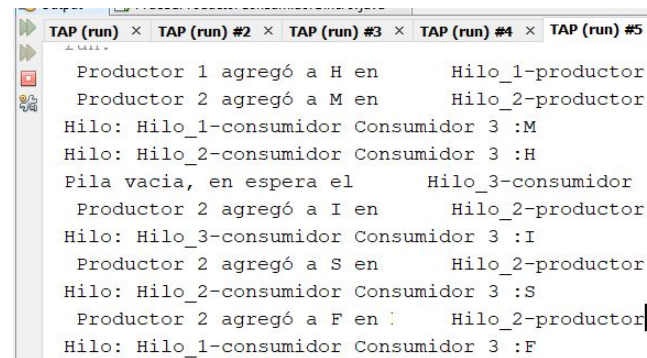
c) ¿Por qué crees que se llega a una situación como la mostrada abajo(se muestra la parte final)?, donde el programa sigue corriendo en forma indefinida. Puedes explicarlo en forma textual o esquemáticamente. Puedes usar también tus propias ejecuciones de prueba.



```

Output x
TAP (run) x TAP (run) #2 x
Hilo: Thread-2 Consumidor 3 :B
Pila vacia, en espera el hilo Thread-2
Pila vacia, en espera el hilo Thread-3
Pila vacia, en espera el hilo Thread-4
  
```

6. Cambia los nombres de los hilos a tu manera como se indica, haz una prueba para verificarlo.



```

TAP (run) x TAP (run) #2 x TAP (run) #3 x TAP (run) #4 x TAP (run) #5 :
Productor 1 agregó a H en Hilo_1-productor
Productor 2 agregó a M en Hilo_2-productor
Hilo: Hilo_1-consumidor Consumidor 3 :M
Hilo: Hilo_2-consumidor Consumidor 3 :H
Pila vacia, en espera el Hilo_3-consumidor
Productor 2 agregó a I en Hilo_2-productor
Hilo: Hilo_3-consumidor Consumidor 3 :I
Productor 2 agregó a S en Hilo_2-productor
Hilo: Hilo_2-consumidor Consumidor 3 :S
Productor 2 agregó a F en Hilo_2-productor
Hilo: Hilo_1-consumidor Consumidor 3 :F
  
```

7. Una alternativa de manejo de hilos es mediante un administrador de hilos disponible en java llamado *"ExecutorService"*.

? Adapta la clase prueba utilizando a *"ExecutorService"* , donde p1,p2 son productores y c1,c2,c3 son cosumidores

```

ExecutorService ejecutor = Executors.newCachedThreadPool();
ejecutor.execute(p1);
ejecutor.execute(p2);
ejecutor.execute(c2);
ejecutor.execute(c3);
ejecutor.execute(c1);
ejecutor.shutdown();

try
{
//espera de 1 minuto de los consumidores-productores en su ejecucion
boolean tareasTerminaron = ejecutor.awaitTermination(
1, TimeUnit.MINUTES );
if ( tareasTerminaron )
System.out.println( "todas las tareas teminaron" );// imprime el contenido
else{
System.out.println(
"Se agoto el tiempo esperando a que las tareas terminaran." );
System.exit(1);
}
} // fin de try
catch ( InterruptedException ex )
{
System.out.println(
"Hubo una interrupcion mientras esperaba a que terminaran las tareas." );
}

```

8. ¿ En tu bitácora de seguimiento anota las ventajas y desventajas de manejar los hilos con “*ExecutorService*” en lugar de manejarlos directamente.

### Sugerencias didácticas

Documenta tu código

Realiza la práctica en grupos de trabajo de 2 a 4 estudiantes.

Lleva una bitácora de seguimiento de lo solicitado durante la práctica

Comparte tu experiencia con otros integrantes mediante un foro del grupo

### Reporte del estudiante

Código fuente de las clases involucradas cuando se usa la sincronía

Imágenes de resultados de ejecución sin usar sincronía y aplicando la sincronía.

Entrega de la bitácora de seguimiento

### Bibliografía

1. Deitel y Deitel. Como programar en Java. Prentice Hall. Septima Edición, 2008
2. Oracle Documentación, The Java™ Tutorials.  
<https://docs.oracle.com/javase/tutorial/uiswing/components/index.html>. Último acceso 16/11/18