

CONCURRENCIA EN JAVA

wait() y notify()

PRODUCTOR Y CONSUMIDOR

Para entender el problema de la sincronización de hilos vamos a explicar un problema típico como es el del productor y el consumidor.



En nuestro problema existen uno o varios hilos productores, encargados de llenar un recipiente. Un productor llenará el recipiente siempre y cuando esté vacío; si el recipiente está lleno, debe bloquearse. El hilo productor saldrá del estado de bloqueo cuando se le notifique que el recipiente está vacío. Por otra parte, el hilo **productor** debe notificar a los hilos bloqueados que esperan a que se llene el recipiente que éste se ha llenado.

```
SI recipiente_Lleno ENTONCES
    Esperar_Vaciar_Recipiente
EN OTRO CASO
    Llenar_Recipiente
    Notificar_Llenado_Recipiente
```

el hilo **consumidor** vaciará el recipiente si está lleno. En el caso de que no esté lleno, se bloqueará a la espera de que se le notifique que ha sido llenado. Una vez vaciado el recipiente, deberá notificar a los hilos que estaban esperando para llenar el recipiente que éste ha sido vaciado.

```
SI recipiente_Vacío ENTONCES
    Esperar_Llenar_Recipiente
EN OTRO CASO
    Vaciar_Recipiente
    Notificar_Vaciado_Recipiente
```

Hay que tener en cuenta que el acceso al objeto compartido (el recipiente) ha de hacerse de forma sincronizada; por lo tanto, debemos hacer uso de las regiones críticas como en prácticas anteriores (synchronized).

En primer lugar vamos a definir la clase que implementa el objeto compartido que tiene que ser sincronizado (será el monitor).

MONITOR

Esta clase tiene un atributo contenido donde se almacena un valor entero. Debe contar con dos métodos para almacenar y descargar el contenido de forma sincronizada. Si el recipiente no se ha vaciado, habrá que esperar a que un hilo consumidor lo vacíe. Una vez vaciado, se debe notificar que está vacío, por si hay algún hilo productor bloqueado a la espera de llenarlo. Si el recipiente no se ha llenado, un hilo consumidor bloqueado para poder vaciar el contenido del recipiente.

```

package Boletin9;

public class Recipiente {
    private int contenido;
    private boolean lleno = false;

    public synchronized int vaciar() {
        while (lleno == false) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("Interrupcion del hilo ... ");
            }
        }
        lleno = false;
        notifyAll();
        return contenido;
    }

    public synchronized void llenar (int valor) {
        while (lleno == true) {
            try {
                wait();
            } catch (InterruptedException e) {
                System.out.println("Interrupcion del hilo ... ");
            }
        }
        contenido = valor;
        lleno = true;
        notifyAll();
    }
}

```

El monitor tiene dos atributos:

- **contenido:** objeto donde vamos a almacenar el contenido del recipiente.
- **lleno:** valor lógico que nos indica el estado del recipiente.

Además, posee dos métodos:

- **vaciar:** es el método utilizado por los hilos consumidores. Si el recipiente no está lleno, el hilo se bloquea a la espera de que lo llene un hilo productor. Cuando le notifican que está lleno, salde del estado de bloqueo, indica que va a coger el contenido del recipiente y lo va a dejar vacío, y notifica que el recipiente está vacío a los procesos que puedan estar bloqueados esperando para llenarlo.
- **llenar:** es el método utilizado por los hilos productores. Si el recipiente está lleno, se queda bloqueado esperando la notificación de que se ha vaciado. Cuando se le comunica este evento y el recipiente está vacío, lo llena y notifica que está lleno a los hilos que puedan estar bloqueados a la espera de que se llenara.

PRODUCTOR

Esta clase sólo tendrá el método `run()` donde invocará a la variable compartida. Después de llenar el recipiente, se bloquea durante un tiempo aleatorio.

```
package Boletin9;

public class Productor extends Thread {
    private Recipiente reci;
    private int numero; //Si hay varios productores cada
                        //cada uno llevará un numero

    //Constructor
    public Productor(Recipiente recipi, int num) {
        reci=recipi;
        numero=num;
    }

    public void run() {
        for (int i=1; i<=5; i++) {
            reci.llenar(i);
            System.out.println("Productor "+numero+" pone el valor "+i);

            //espera un tiempo antes de volver a llenar
            try {
                sleep((int) (Math.random()*100));
            } catch (InterruptedException e)
            {
                System.out.println("Interrupcion del hilo...");
            }
        }
    }
}
```

Esta clase tiene dos atributos:

- **numero:** nos permite identificar al hilo en el caso de que haya más de un hilo productor. Su valor se lo asigno en el constructor de la clase
- **reci:** almacena una referencia al recipiente común. Esta referencia se la transfiero a la clase a través del constructor.

El método `run()` del hilo sólo invoca al método `llenar()` del recipiente para llenarlo y espera un tiempo determinado antes de volver a llevar el recipiente.

CONSUMIDOR

Esta clase tiene los mismos atributos que el hilo productor. Ahora en el método *run()* el hilo invocará al método *vaciar()* del recipiente para consumir el valor que contiene, luego se bloquea durante unos segundos antes de consumir otro valor.

```
package Boletin9;
```

```
public class Consumidor extends Thread {
```

```
    private Recipiente reci;
```

```
    private int numero;
```

```
    //constructor
```

```
    public Consumidor(Recipiente recipi, int nume) {
```

```
        reci=recipi;
```

```
        numero=nume;
```

```
    }
```

```
    public void run() {
```

```
        int valor=0;
```

```
        for (int i=1; i<=5; i++) {
```

```
            valor = reci.vaciar();
```

```
            System.out.println("Consumidor "+numero+" toma el valor "+i);
```

```
            //espera un tiempo antes de volver a consumir
```

```
            try {
```

```
                sleep((int) (Math.random()*100));
```

```
            } catch (InterruptedException e)
```

```
            {
```

```
                System.out.println("Interrupcion del hilo...");
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

PROGRAMA PRINCIPAL

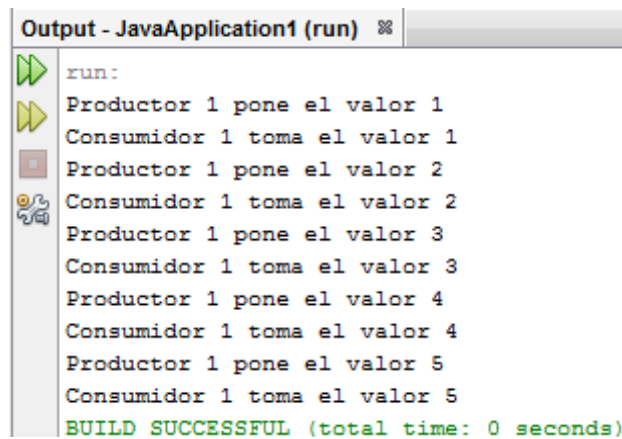
Para terminar, el programa principal en el que creo un objeto de la clase Recipiente y los hilos Productor y Consumidor y lanzo su ejecución.

```
package Boletin9;

public class MainProduConsu {
    public static void main(String args[]) {
        Recipiente reci = new Recipiente();

        Productor produ1 = new Productor(reci,1);
        Consumidor consu1 = new Consumidor(reci,1);

        produ1.start();
        consu1.start();
    }
}
```



```
Output - JavaApplication1 (run) %
run:
Productor 1 pone el valor 1
Consumidor 1 toma el valor 1
Productor 1 pone el valor 2
Consumidor 1 toma el valor 2
Productor 1 pone el valor 3
Consumidor 1 toma el valor 3
Productor 1 pone el valor 4
Consumidor 1 toma el valor 4
Productor 1 pone el valor 5
Consumidor 1 toma el valor 5
BUILD SUCCESSFUL (total time: 0 seconds)
```