**wait():**

* Este método se llama en un objeto dentro de un bloque sincronizado para liberar el monitor del objeto y poner el hilo actual en estado de espera hasta que otro hilo llame a **notify()** o **notifyAll()** en el mismo objeto.
* La invocación de **wait()** debe estar dentro de un bloque **synchronized** para evitar problemas de concurrencia.

synchronized (objetoMonitor) {

while (condicionNoCumplida) {

objetoMonitor.wait();

}

// Realizar operaciones cuando la condición se cumple

}

**notify():**

* Este método se llama en un objeto dentro de un bloque sincronizado para notificar a un hilo que está en estado de espera (debido a una llamada previa a **wait()**) que puede continuar su ejecución.
* El hilo notificado competirá con otros hilos por el acceso al monitor.

synchronized (objetoMonitor) {

// Realizar operaciones

objetoMonitor.notify();

}

**notifyAll():**

* Similar a **notify()**, pero notifica a todos los hilos que están en espera en el objeto monitor. Esto puede ser útil en situaciones donde varios hilos pueden competir por un recurso compartido y todos deben ser notificados para reevaluar la condición.

synchronized (objetoMonitor) {

// Realizar operaciones

objetoMonitor.notifyAll();

}

**wait():**

* **wait()** se utiliza para poner un hilo en espera hasta que otra parte del código notifique que cierta condición se ha cumplido.

synchronized (objetoMonitor) {

while (!condicionCumplida) {

try {

objetoMonitor.wait(); // El hilo entra en estado de espera

} catch (InterruptedException e) {

e.printStackTrace();

}

}

// Realizar operaciones cuando la condición se cumple

}

* + Aquí, el hilo se bloquea en **wait()** hasta que otro hilo notifica al objeto monitor mediante **notify()** o **notifyAll()**.

1. **notify():**
   * **notify()** se utiliza para despertar un solo hilo que esté esperando en el objeto monitor.

synchronized (objetoMonitor) {

// Realizar operaciones

objetoMonitor.notify(); // Despierta un hilo en espera

}

* + Es importante tener en cuenta que **notify()** no garantiza cuál hilo específico se despertará. Puede ser cualquiera de los hilos en espera.

1. **notifyAll():**
   * **notifyAll()** se utiliza para despertar a todos los hilos que están en espera en el objeto monitor.

synchronized (objetoMonitor) {

// Realizar operaciones

objetoMonitor.notifyAll(); // Despierta todos los hilos en espera

}

* + **notifyAll()** es más seguro en situaciones donde varios hilos pueden estar esperando para evitar que algún hilo quede permanentemente bloqueado.

**Ejemplo Completo:**

Considere un caso donde hay un productor y varios consumidores compartiendo un recurso (por ejemplo, una cola). El productor agrega elementos a la cola, y los consumidores los retiran. **wait()** y **notify()** se utilizan para sincronizar el acceso a la cola.

import java.util.LinkedList;

import java.util.Queue;

class ColaCompartida {

private Queue<Integer> cola = new LinkedList<>();

public synchronized void agregarElemento(int elemento) {

cola.add(elemento);

notify(); // Notifica a cualquier hilo que esté esperando

}

public synchronized int retirarElemento() {

while (cola.isEmpty()) {

try {

wait(); // Espera hasta que haya elementos en la cola

} catch (InterruptedException e) {

e.printStackTrace();

}

}

return cola.poll();

}

}

class Productor implements Runnable {

private ColaCompartida cola;

public Productor(ColaCompartida cola) {

this.cola = cola;

}

public void run() {

for (int i = 0; i < 5; i++) {

cola.agregarElemento(i);

System.out.println("Productor agregó: " + i);

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Consumidor implements Runnable {

private ColaCompartida cola;

public Consumidor(ColaCompartida cola) {

this.cola = cola;

}

public void run() {

for (int i = 0; i < 5; i++) {

int elemento = cola.retirarElemento();

System.out.println("Consumidor retiró: " + elemento);

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class Main {

public static void main(String[] args) {

ColaCompartida cola = new ColaCompartida();

Thread productorThread = new Thread(new Productor(cola));

Thread consumidorThread = new Thread(new Consumidor(cola));

productorThread.start();

consumidorThread.start();

}

}