Miguel Ángel Ramírez Pérez mramper0604@g.educaand.es

Explicación - Ejercicio02 - Productor_Consumidor

Para este ejercicio se han utilizado dos formas (***wait/notify*** y ***BlockingQueue***). Las clases **ProductorEnteros** y **ConsumidorEnteros** son las dos únicas que utilizare para las dos versiones.

Para empezar con el ejercicio he creado una interfaz llamada **BufferInterface**, lo que hago para utilizarla es implementarla en las clases **Buffer** y **BufferBQ**. Gracias a la interfaz las clases **ProductorEnteros** y **ConsumidorEnteros** no piden una clase concreta, sino que piden cualquier objeto que esté en **BufferInterface.**

Explicación de las Clases

1. **ProductorEnteros.java**

    a. **Constructor:** Recibe el **BufferInterface** y cuántos elementos tiene que fabricar.
    b. **run():**
        - Tiene un bucle for que se repite **numElementos** veces.
        - **buffer.put(elemento):** El productor simplemente "pone" el elemento. Si el buffer está lleno, este método **put()** se quedará bloqueado él solo hasta que haya sitio. El productor no se preocupa de nada.
        - **Thread.sleep(...):** Duerme un tiempo aleatorio (0-3 seg) para simular que tarda en producir.
        - **catch (InterruptedException e):** Si el hilo es interrumpido mientras duerme o espera en el put, lo capturamos, imprimimos un error y volvemos a marcar el hilo como interrumpido con **Thread.currentThread().interrupt()**.

2. **ConsumidorEnteros.java**
    a. **Constructor:** Solo necesita el **BufferInterface** para saber de dónde coger elementos.
    b. **run():**
        - **while (!Thread.currentThread().isInterrupted()):** ¡Esta es la línea más importante! Es un bucle "infinito" que se traduce como: "Sigue ejecutándote mientras no te hayan interrumpido".
        - **Integer elemento = buffer.get():** Igual que el productor, el consumidor solo "coge". Si el buffer está vacío, este método se quedará bloqueado hasta que un productor ponga algo.
        - **Thread.sleep(...):** Duerme un poco (0-1 seg) para simular que tarda en procesar el elemento.
        - **catch (InterruptedException e):** Esta es la forma de parar el hilo. Cuando el hilo main llame a **consumidor.interrupt()**, el **sleep()** (o el get() si estaba esperando) lanzará esta excepción. El **catch** lo captura, imprimimos el mensaje de que hemos finalizado, y salimos del **run()**. Al salir del **run()**, el hilo muere de forma ordenada.

3. **Buffer.java (wait/notify)**
    a. **put(E elemento):**
        - **synchronized:** Declaro el método synchronized para que solo un hilo pueda entrar a la vez. Esto me da el "candado" (monitor lock).
        - **while (buffer.size() == SIZE):** Comprueba el tamaño del buffer para saber si está lleno.
        - **wait():** El hilo suelta el candado (para que otro hilo, pueda entrar a get()) y se va a "dormir" (estado WAITING).
        - **notifyAll():** Cuando por fin pone el elemento, "despierta" a todos los hilos que estén durmiendo (**wait()**).
    b. **get():**
        - **synchronized:** Coge el candado.
        - **while (buffer.isEmpty()):** Si está vacío.
        - **wait():** Suelta el candado y a dormir.
        - **notifyAll():** Cuando coge un elemento, despierta a todos.

4. **BufferBQ (BlockingQueue):**
    a. Simplemente uso un **ArrayBlockingQueue**, que es una clase que ya hace todo lo de **wait/notify** por mi.
    b. **put(E elemento):** Solo llamo a **queue.put(elemento)**. Si la cola está llena, **put()** se bloquea solo hasta que haya sitio.
    c. **get():** Solo llamo a **queue.take()**. Si la cola está vacía, **take()** se bloquea solo hasta que haya un elemento.
    d. Es mucho más limpia, segura y más rápida.

5. **Main.java y MainBQ.java**
    Ambos main son casi idénticos gracias a la interfaz.
    a. **Creo el Buffer:** new Buffer(...) o new BufferBQ(...).
    b. **Creo los Hilos:** Creo los **ProductorEnteros** y **ConsumidorEnteros** y les paso el buffer.
    c. **Los Inicio: start()** a todos.
    d. **for (Thread p : hilosProductores) { p.join(); }:** Esta es la segunda parte clave. El hilo main se une (join) solo a los productores. Esto significa que main se queda parado en esta línea hasta que el último productor haya terminado su bucle for y muera.
    e. **consumidor1.interrupt();:** En cuanto el join de los productores termina, main sigue ejecutándose y llama a **interrupt()** a los consumidores.
    f. Esto hace que los consumidores salten a su bloque **catch**, terminen su **run()** y mueran limpiamente.
    g. Por último, hago **join()** a los consumidores para asegurarse de que han terminado antes de imprimir el mensaje final.