Explicación de `synchronized` en Java

El uso de la palabra clave `synchronized` en Java es fundamental para el manejo de la concurrencia y la sincronización de hilos. Permite controlar el acceso a recursos compartidos, asegurando que solo un hilo a la vez pueda ejecutar una sección crítica de código. Esto previene problemas como la corrupción de datos o condiciones de carrera.

PROGRAMA 1: `synchronized(t1)`

En este programa, la sección `synchronized(t1)` significa que el hilo que ejecuta este código debe obtener el monitor (o "candado") del objeto `t1` antes de poder proceder. El objeto `t1` es una instancia de `Thread` que se acaba de crear y se ha iniciado (`t1.start()`).

Cuando el hilo principal llega a `synchronized(t1)` , intenta adquirir el monitor de `t1` . Una vez que lo obtiene, ejecuta el código dentro del bloque `synchronized` , que incluye `t1.wait(5000)` . El método `wait()` hace que el hilo actual (el hilo principal en este caso) libere el monitor de `t1` y espere durante 5000 milisegundos (o hasta que otro hilo lo notifique a través de `notify()` o `notifyAll()`).

Puntos clave:

- El `synchronized` se aplica a un objeto específico (`t1`).
- El hilo principal adquiere el monitor de `t1` .
- `t1.wait(5000)` hace que el hilo principal libere el monitor de `t1` y espere.
- El `Thread t1` creado no tiene ninguna tarea asignada, por lo que su `run()` método está vacío y termina inmediatamente. Esto significa que `t1` no tiene la oportunidad de notificar al hilo principal.
- Por lo tanto, el hilo principal esperará los 5000 milisegundos completos antes de continuar.

PROGRAMA 2: `synchronized void go()`

En este segundo programa, la palabra clave `synchronized` se aplica directamente al método `go()` . Esto significa que el monitor que se adquiere es el del objeto `this` , es decir, la instancia actual de la clase `Bees` .

Cuando el hilo principal llama a `new Bees().go()` , antes de que el método `go()` pueda ejecutarse, el hilo principal debe adquirir el monitor del objeto `Bees` . Una vez que lo obtiene, el código dentro del método `go()` se ejecuta. Dentro de este método, se llama a `this.wait(5000)` .

El método `this.wait(5000)` hace que el hilo actual (el hilo principal) libere el monitor del objeto `Bees` y espere durante 5000 milisegundos (o hasta que otro hilo lo notifique).

Puntos clave:

- El `synchronized` se aplica al método `go()`, lo que implica que el monitor es el de la instancia de la clase (`this`).
- El hilo principal adquiere el monitor de la instancia de `Bees`.
- `this.wait(5000)` hace que el hilo principal libere el monitor de la instancia de `Bees` y espere.
- Al igual que en el PROGRAMA 1, el `Thread t1` creado no tiene ninguna tarea asignada y termina inmediatamente, por lo que no hay ningún hilo que pueda notificar al hilo principal.
- Por lo tanto, el hilo principal esperará los 5000 milisegundos completos antes de continuar.

Diferencias clave y propósito de `synchronized`

La principal diferencia entre ambos programas radica en **qué objeto se utiliza como monitor** para la sincronización:

- **PROGRAMA 1:** El monitor es el objeto `t1` (la instancia de `Thread` creada dentro del método `go()`). Esto significa que el `synchronized` solo protege el bloque de código específico que lo envuelve, y el candado es exclusivo de ese objeto `t1`.
- **PROGRAMA 2:** El monitor es el objeto `this` (la instancia de `Bees` sobre la que se llama al método `go()`). Esto significa que el `synchronized` protege todo el método `go()`, y cualquier otro hilo que intente llamar a un método `synchronized` en la *misma instancia* de `Bees` tendrá que esperar a que el hilo actual libere el monitor.

En ambos casos, el propósito de `synchronized` es asegurar que solo un hilo pueda ejecutar la sección de código protegida en un momento dado. El método `wait()` se utiliza para que un hilo libere el monitor y espere, permitiendo que otros hilos intenten adquirirlo. Sin embargo, en estos ejemplos específicos, dado que el hilo `t1` no realiza ninguna acción que pueda notificar al hilo principal, el `wait()` simplemente actúa como una pausa forzada.

Analogía para niños: El Juguete Especial y la Sala de Juegos

Imagina que tienes un juguete muy especial, como un coche de carreras teledirigido, que solo una persona puede usar a la vez para que no se rompa. La sala de juegos es donde están todos los juguetes.

El Juguete Especial (el objeto monitor):

Cuando un niño quiere jugar con el coche de carreras, primero tiene que **pedir el coche** (adquirir el monitor). Si otro niño ya lo tiene, tiene que **esperar su turno** (el hilo espera a que el monitor esté disponible).

Una vez que el niño tiene el coche, puede jugar con él. Mientras juega, **nadie más puede tocar ese coche** (la sección `synchronized` está protegida). Cuando termina de jugar, **suelta el coche** para que otro niño pueda usarlo (libera el monitor).

La Sala de Juegos Entera (el método `synchronized`):

Ahora, imagina que la regla no es solo para el coche, sino para **toda la sala de juegos**. Si un niño entra a la sala de juegos para jugar con *cualquier* juguete, la puerta de la sala se **cierra con llave** (el método `synchronized` bloquea el objeto `this`).

Mientras ese niño está dentro, **nadie más puede entrar a la sala de juegos** (otros hilos no pueden llamar a métodos `synchronized` en la misma instancia). Cuando el niño sale de la sala, **la puerta se abre** y otro niño puede entrar.

¿Y el `wait()`?

Imagina que el niño que tiene el coche de carreras se cansa un poco y decide **sentarse a descansar un momento** (llama a `wait()`). Mientras descansa, **suelta el coche** para que otro niño pueda usarlo. Después de un rato, o si alguien le dice "¡Ya puedes seguir jugando!" (un `notify()`), el niño se levanta y **vuelve a intentar coger el coche** para seguir jugando.

Así, `synchronized` es como tener una regla para que solo una persona a la vez pueda usar algo especial (un objeto) o estar en un lugar especial (un método), para que todo funcione bien y no haya líos.