


CyclicBarrier

Concepto

- Mecanismo que permite sincronizar la ejecución de un grupo de threads
 - Objeto que permite que un conjunto de hilos puedan esperarse unos a otros en un determinado punto.
 - Cuando todos los hilos del grupo alcanzan la barrera, se desbloquean y continúan su ejecución.
- 

Creación

- Se instancian como objetos de la clase `CyclicBarrier` de `java.util.concurrent`
- Dispone de dos constructores:
 - `CyclicBarrier(int hilos)`. Crea un `CyclicBarrier` que permite sincronizar el número de hilos indicados en el constructor
 - `CyclicBarrier(int hilos, Runnable action)`. Cuando todos los hilos alcanzan la barrera, se ejecuta la acción indicada en el segundo parámetro del constructor

Sincronización de los hilos

- La sincronización de los hilos se realiza a través del método `await()` de `CyclicBarrier`.
- Cuando un hilo llama a este método, queda en espera en ese punto.
- Cuando el total de hilos que ha realizado la llamada a `await()` es igual al valor indicado en el constructor, se liberan todos los hilos y se ejecuta el `Runnable`.
- El contador vuelve a ponerse a 0 tras alcanzar todos los hilos la barrera.

Ejemplo

- Realización de un programa que recupere los números almacenados en tres ficheros distintos y después muestre la suma de los mismos.

Solución

Thread

```
public class Lector implements Runnable {
    CyclicBarrier barrier;
    List<Integer> lista;
    String url;
    public Lector(CyclicBarrier barrier, List<Integer> lista, String url) {
        super();
        this.barrier = barrier;
        this.lista = lista;
        this.url = url;
    }
    @Override
    public void run() {
        try {
            Files.lines(Path.of(url))
                .forEach(n-> lista.add(Integer.parseInt(n)));
            barrier.await(); // tras la lectura esperan en la barrera
        } catch (IOException | InterruptedException | BrokenBarrierException e) {
            e.printStackTrace();
        }
    }
}
```

Principal

```
public class Lanzador {
    public static void main(String[] args) {
        List<Integer> lista=new CopyOnWriteArrayList<>();
        CyclicBarrier barrier=new CyclicBarrier(3,0->{
            System.out.println(lista.stream().mapToInt(n->n).sum());
});
        for(int i=1;i<=3;i++){
            new Thread(new Lector(barrier,lista,"numbers"+i+".txt")).start();
        }
    }
}
```

Revisión conceptos



Dado el siguiente bloque de código definido en el método run() de una implementación de Runnable:

```
System.out.println("hello"); //line 1  
System.out.println("printing..."); //line 2
```

Si queremos que todos los hilos muestren el mensaje "hello", antes de que aparezca el primer mensaje "printing...":

- a. Encerramos las instrucciones en un bloque sincronizado
- b. Realizamos la llamada al método await() de un CyclicBarrier entre las líneas 1 y 2
- c. Realizamos la llamada al método await() de un CyclicBarrier antes de la línea 1
- d. Realizamos la llamada al método await() de un CyclicBarrier en el constructor de la clase

Respuesta

La respuesta es la B. Tras la ejecución de la primera instrucción, los hilos quedarán en espera hasta que todos hayan pasado por ese punto.