Semana 7 Productor/Consumidor

P3

El esta segunda entrega de la semana 7 se reutiliza el código hecho en la P4 para la realización de 2 nuevos ejercicios.

En el primer apartado del primer ejercicio se cambia la sincronización del *Buffer* para usar *loks* en lugar de los métodos *synchronized*.

En el segundo apartado se pide que el productor no ejecute un *write* 2 veces consecutivas. Para ello se añade una condición en el método de escritura que garantice que el ultimo hilo ejecutado es diferente al actual.

Para una mejor representación se rellena el *Buffer* en su constructor, obteniendo la siguiente salida:

Consumidor is reading, Product size is: 4
Consumidor is reading, Product size is: 3
Productor is writting, Product size is: 4
Consumidor is reading, Product size is: 3
Consumidor is reading, Product size is: 2
Productor is writting, Product size is: 3
Consumidor is reading, Product size is: 2
Productor is writting, Product size is: 3
Consumidor is reading, Product size is: 2
Productor is writting, Product size is: 3
Consumidor is reading, Product size is: 2
Consumidor is reading, Product size is: 1
Consumidor is reading, Product size is: 0

Como se puede ver, es habitual que el escritor escriba 2 veces consecutivas mientras que se garantiza que el lector nunca lee 2 veces de forma consecutiva.

Otra forma más sencilla de garantizar esta condición sería poner un *buffer* con 1 de capacidad, lo que implicaría que nunca se pudiese escribir 2 veces seguidas sin superar dicha capacidad.

El el tercer ejercicio de la entrega se pide que se sustituya la clase *Buffer* por una *BlockingQueue*. *BlockingQueue* es una clase que extienda a *Queue* añadiendo operaciones bloqueantes.

Para realizar este cambio se hace que *Buffer<E> extends ArrayBlockingQueue<E>*. Tras esto se modifican los métodos *write* y *read* para que usen los métodos heredados *put()* y *take()* e impriman el mismo mensaje que en el ejercicio anterior.

Usar clase *BlockingQueue* simplifica mucho el ejercicio, pues dicha clase ya tiene una capacidad máxima que no permite sobrepasar (definida en el constructor de la clase) y una cantidad minima (0) que impide que los hilos hagan *take()*.