
Multicore proj 2



| | |
|------------|-----------------|
| subject | multicore |
| professor | 손봉수 교수님 |
| date | 2023.05.10 |
| department | 소프트웨어대학 소프트웨어학부 |
| name | 20200641 임수현 |

***How to compile**

I use eclipse and also it can work at VScode

(i)-a. Explain the interface/class BlockingQueue and ArrayBlockingQueue with your own English sentences. (DO NOT copy&paste)

-BlockingQueue is an interface that does not cause problems when multiple threads simultaneously insert or remove elements. When there is no element to import from Queue or there is no space to put it in, block it so that the problem does not occur. Unlike add() remove(), put() and take() block without creating an exception if there is no element to import from the queue or if there is no space to put it in.

-ArrayBlockingQueue serves to implement the BlockingQueue interface with class, and the presence of an upper bound allows up to a predetermined number of elements. Store and exit internally in the order of FIFO.

(i)-b. Create and include (in your document) your own example of multithreaded JAVA code (ex1.java) that is simple and executable. (DO NOT copy&paste) Your example code should use put() and take() methods. Also, include example execution results (i.e. output) in your document.

```
<terminated> ex1 [Java Application] C:\Users\Wlimsoo\p2\pool\plugins\Worg.eclipse
Thread-4: trying to put in numlist
Thread-4: just put
Thread-1: trying to put in numlist
Thread-1: just put
Thread-5: trying to put in numlist
Thread-5: just put
Thread-3: trying to put in numlist
Thread-3: just put
Thread-9: trying to put in numlist
Thread-9: just put
Thread-8: trying to put in numlist
Thread-6: trying to put in numlist
Thread-7: trying to put in numlist
Thread-0: trying to put in numlist
Thread-2: trying to put in numlist
Thread-3: about to take out
Thread-3: have been taken
Thread-8: just put
Thread-4: about to take out
Thread-4: have been taken
Thread-6: just put
Thread-1: about to take out
Thread-1: have been taken
Thread-7: just put
Thread-7: about to take out
Thread-7: have been taken
Thread-0: just put
Thread-3: trying to put in numlist
Thread-9: about to take out
Thread-9: have been taken
Thread-2: just put
Thread-8: about to take out
Thread-8: have been taken
Thread-3: just put
Thread-4: trying to put in numlist
Thread-5: about to take out
Thread-5: have been taken
Thread-4: just put
Thread-0: about to take out
Thread-0: have been taken
Thread-1: trying to put in numlist
Thread-1: just put
Thread-9: trying to put in numlist
Thread-2: about to take out
Thread-2: have been taken
Thread-9: just put
Thread-7: trying to put in numlist
Thread-1: about to take out
Thread-1: have been taken
Thread-7: just put
```

(ii)-a. Do the things similar to (i)-a for the class ReadWriteLock.

Multiple threads can read at once, but not write. Therefore, multiple threads can be read from shared resources without causing concurrency errors. Readlock is to lock multiple threads to read at once, and writelock is to lock one thread at a time to write.

(ii)-b. Do the things similar to (i)-b for lock(), unlock(), readLock() and writeLock() of ReadWriteLock. (ex2.java)

```
ex2 [Java Application] C:\Users\Wlimsoo\p2\Wpool
Reader 1 read value: 0
Reader 4 read value: 0
Reader 2 read value: 0
Reader 0 read value: 0
Reader 3 read value: 0
Writer 0 write value: 1
Writer 1 write value: 2
Writer 2 write value: 3
Reader 1 read value: 3
Reader 4 read value: 3
Writer 2 write value: 4
Writer 1 write value: 5
Writer 0 write value: 6
Reader 0 read value: 6
Reader 3 read value: 6
Reader 2 read value: 6
Reader 1 read value: 6
Reader 4 read value: 6
Reader 0 read value: 6
Writer 0 write value: 7
Reader 2 read value: 7
Reader 3 read value: 7
Writer 1 write value: 8
Writer 2 write value: 9
Reader 1 read value: 9
Reader 0 read value: 9
Reader 4 read value: 9
Writer 0 write value: 10
Reader 3 read value: 10
Reader 2 read value: 10
Writer 1 write value: 11
Writer 2 write value: 12
Reader 1 read value: 12
Reader 0 read value: 12
Reader 4 read value: 12
Writer 1 write value: 13
Reader 2 read value: 13
Writer 2 write value: 14
Writer 0 write value: 15
```

(iii)-a. Do the things similar to (i)-a for the class `AtomicInteger`.

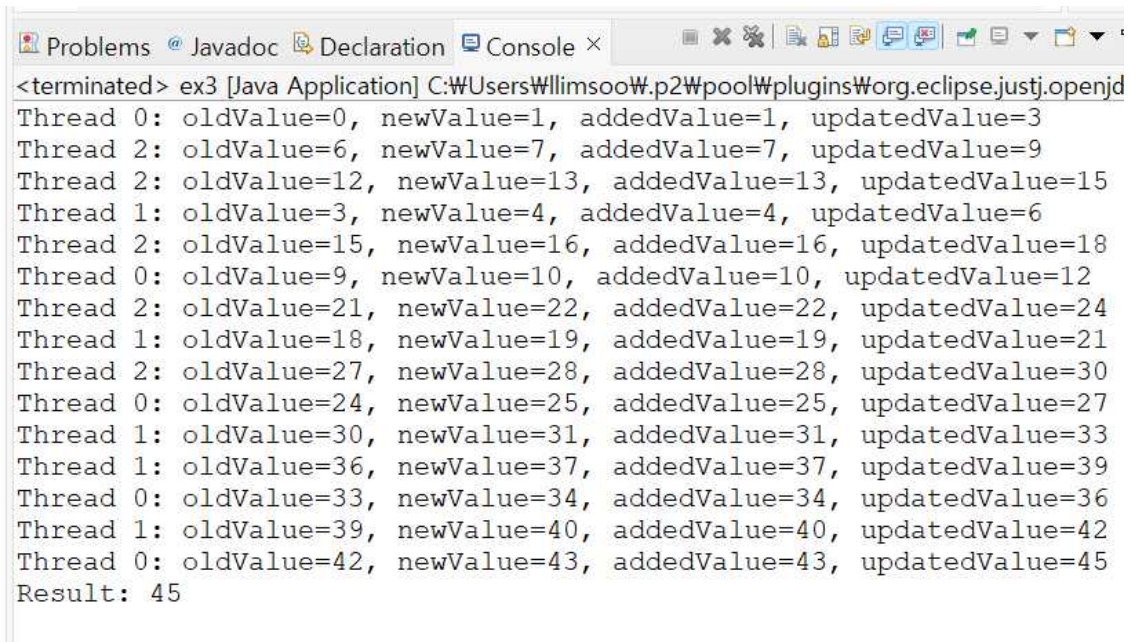
`Atomic Integer` is a class used to safely increase integers in a multi-threaded environment.

Basic int variables can cause problems when accessed simultaneously in a multi-threaded environment. If multiple threads read and write to the variable at the same time, thread contention can cause unexpected results. To prevent this, `Atomic Integer` is implemented to safely grow in a multi-threaded environment using synchronization.

Therefore, in order to safely increase integers in a multi-threaded environment, we recommend using the `Atomic Integer` class.

'`getAndAdd()`' increases the value and gets the value before the increase. '`addAndGet()`' is the difference between getting an increased value.

(iii)-b. Do the things similar to (i)-b for `get()`, `set()`, `getAndAdd()`, and `addAndGet()` methods of `AtomicInteger`. (ex3.java)

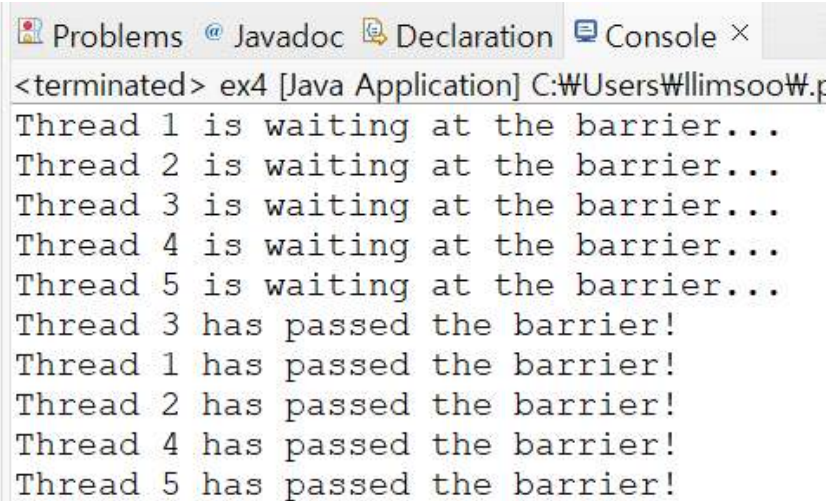


```
<terminated> ex3 [Java Application] C:\Users\Wlimsoo\p2\workspace\plugins\Worg.eclipse.justj.openjdk
Thread 0: oldValue=0, newValue=1, addedValue=1, updatedValue=3
Thread 2: oldValue=6, newValue=7, addedValue=7, updatedValue=9
Thread 2: oldValue=12, newValue=13, addedValue=13, updatedValue=15
Thread 1: oldValue=3, newValue=4, addedValue=4, updatedValue=6
Thread 2: oldValue=15, newValue=16, addedValue=16, updatedValue=18
Thread 0: oldValue=9, newValue=10, addedValue=10, updatedValue=12
Thread 2: oldValue=21, newValue=22, addedValue=22, updatedValue=24
Thread 1: oldValue=18, newValue=19, addedValue=19, updatedValue=21
Thread 2: oldValue=27, newValue=28, addedValue=28, updatedValue=30
Thread 0: oldValue=24, newValue=25, addedValue=25, updatedValue=27
Thread 1: oldValue=30, newValue=31, addedValue=31, updatedValue=33
Thread 1: oldValue=36, newValue=37, addedValue=37, updatedValue=39
Thread 0: oldValue=33, newValue=34, addedValue=34, updatedValue=36
Thread 1: oldValue=39, newValue=40, addedValue=40, updatedValue=42
Thread 0: oldValue=42, newValue=43, addedValue=43, updatedValue=45
Result: 45
```

(iv)-a. Do the things similar to (i)-a for the class `CyclicBarrier`.

When multi-threading, it is a barrier that other threads have to wait for when all threads reach. So, wait using the `wait()` method and release only when all threads reach.

(iv)-b. Do the things similar to (i)-b for `await()` methods of `CyclicBarrier`. (ex4.java)



```
<terminated> ex4 [Java Application] C:\Users\Wllimsoo\W.p
Thread 1 is waiting at the barrier...
Thread 2 is waiting at the barrier...
Thread 3 is waiting at the barrier...
Thread 4 is waiting at the barrier...
Thread 5 is waiting at the barrier...
Thread 3 has passed the barrier!
Thread 1 has passed the barrier!
Thread 2 has passed the barrier!
Thread 4 has passed the barrier!
Thread 5 has passed the barrier!
```

3-2. Submit the source files `ex1.java`, `ex2.java`, `ex3.java` and `ex4.java` as well as the document described above.

ex1)

This code is similar to parkinglot problem. It is to put the thread into the number list. There are 10 threads in total, and the blockingqueue size is 5. I go into blockingqueue one by one, and if the list contains 5 threads, I wait until it is taken. It is a code that repeats these tasks.

ex2)

In this code, the integer value called 'value' is added with a witter of 1, and the reader is responsible for outputting the value. So if the writer gets a writelock, it can add a value, and if the reader gets a lock while the writer is 'sleeping' after unlocking, it can read the value. If the reader 'sleep' for a certain period of time and then unlocks, the writer who has been waiting for the reader to unlock gets the 'writelock' and adds the value.

ex3)

This code is to add a value using semaphoreInteger. There are 3 threads in total and add 3 for each thread. Repeat this 5 times for each thread. In newValue, the value increased through set is stored through get. In the addedValue, 1 is increased through 'getAndAdd()' and the value before the increase is stored. In the updated Value, 1 is increased through 'addAndGet()' and the value after the increase is stored. So the result should be 45 stored, and the atomic Integer can generate race conditions, so the value is different for each run.

ex4)

Wait through 'await()' until all 5 threads reach the barrrier. Outputs a waiting thread. And when all five threads touch the barrier, they are 'passed'. Outputs the passed thread when it is 'pass'.