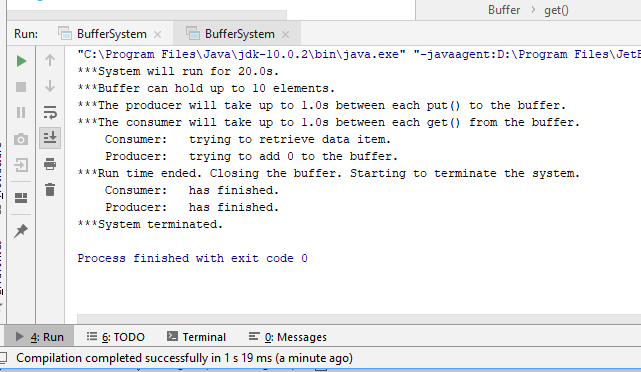
**In the lecture it was said that in the implementation of bounded buffers using semaphores (see figure 1 on page 3) the order of the criticalSection.P() and noOfElements.P() in the Buffer class's get method, was essential, but that the order of criticalSection.V() noOfElements.V() in the same class's put(item) method, was not. Identify the corresponding piece of code in the Buffer class provided and make the change. Can you produce an error situation?**

*The code after changing critiicalSection.P() and noOfElements.P() opposite way around.*

*/\*\*  
 \* Provide protected access to the buffer to allow an item to be retrieved.  
 \** ***@return*** *the data item that has been in the buffer longest or null if the retrieval failed because, e.g.., the  
 \* buffer has closed.  
 \** ***@throws*** *BufferError if it was not possible to retrieve an item from the buffer due to an interrupted exception.  
 \** ***@throws*** *SemaphoreLimitError if any of the semaphores exceeds their limit. This exception would probably  
 \* arise as a consequence of an error in the implementation of semaphores.  
 \*/***public** T get() **throws** BufferError, SemaphoreLimitError {  
 T item;  
 **try** {  
 **criticalSection**.poll(); *// is the buffer available?* **noOfElements**.poll(); *// is there at least one data item in the buffer?* item = getItem(); *// add the data item* **criticalSection**.vote(); *// make the buffer available again* **noOfSpaces**.vote(); *// there is now one more space in the buffer* } **catch** (InterruptedException ie) {  
 **throw new** BufferError(**"Buffer: Data item could not be retrieved from the buffer.\n"** +  
 **"\t"** + ie.getMessage());  
 }  
 **return** item;  
}

**The Error: Deadlock.** System terminates after the time (20 seconds) pass, however Consumer and Producer are both stuck at deadlock and can not execute the program the way expected.



**Why does the error situation arise when the code is changed as described in question 1? Why does it not arise in the original code?**

Swapping **criticalSection**.poll(); and **noOfElements**.poll(); methods stops program from executing. The reason is calling **criticalSection**.poll(); at get() method inside the buffer first, rather than **noOfElements**.poll(); causes an error, as it enters critical section before checking if there are any elements produced. As there are no items produced, and the Consumer is in Critical Section the system freezes. It could unfreeze only if it would release the critical section itself, however, it could only be done once the next task - **noOfElements**.poll(); finish doing it its own task. However, this is not possible, as there are no elements in buffer at all, and this method waits until item appears. As get() method is stuck in there and waits for producer, in the meantime, produce() is stuck, because producer cannot access Critical Section to access shared variable and produce an data item, which then could be accessed by get() method.

Brief Scenario:

* Consumer enters Critical Section when buffer is empty
* Producer checks if there are space available
* Producer can not access Critical Section as it is taken. It waits continuously until Critical Section will be free
* Consumer checks how many elements are available in buffer. (none)
* Consumer waits for elements to appear in buffer, before releasing Critical Section
* Producer waits for Critical Section to be released, before producing an item.

**Deadlock!**

**Is the order of the calls of P() in the Buffer class's put method also essential?**

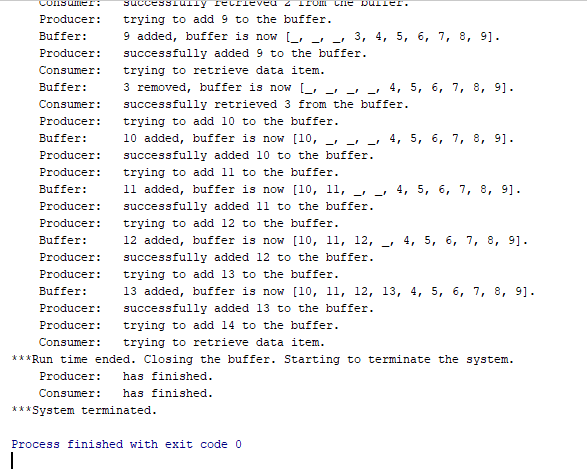
Yes. Changing the order of P() in the Buffer class’s put method is also essential, as entering Critical Section first, rather then checking if there are any available space, may possibly create an deadlock, as in question above.

Scenario:

* Producer enters critical section
* Consumer checks there are some items available to consume
* Producer checks if there is space available on full buffer and gets an answer **– none**
* Producer needs to wait, until some space will be available, so it waits();
* Consumer cannot access the critical section and release the space to allow producer keep working, as critical section is busy. Consumer waits for critical section to be released.
* Producer waits for Consumer to consume a product, so it can not exit the Critical Section.

**Deadlock!**

As shown in the test below, the Producer enters Critical Section and tries to add 14 to the buffer, however the buffer is full. It stays in Critical Section, waits for Consumer to consume the product, however Consumer can not do this, as in order to do the previous, consumer needs to enter critical section which will not be released by producer.



**Self evaluation:**

**Detailed explanation for both get() and put() methods**

**Identified error (deadlock) as an error**

**Providing sufficient amount of testing/ multiple runs evidence.**