

* Package group consists of moving package on each end of routing station (so max of 2?)

what is the order in which the packages are moved?

**Important: To prevent deadlock, ensure that each routing station acquires locks on the conveyors it needs in increasing numerical order.**

Note that a blocked threat can be blocked due to waiting on IO, sleep(),or wait(), or attempting to lock an object that is already locked.Also note, that even though a thread might be in a “runnable” state, it doesn’t mean its running (it might be waiting in the thread queue).

Remember interrupt/signal, wait()/notify()

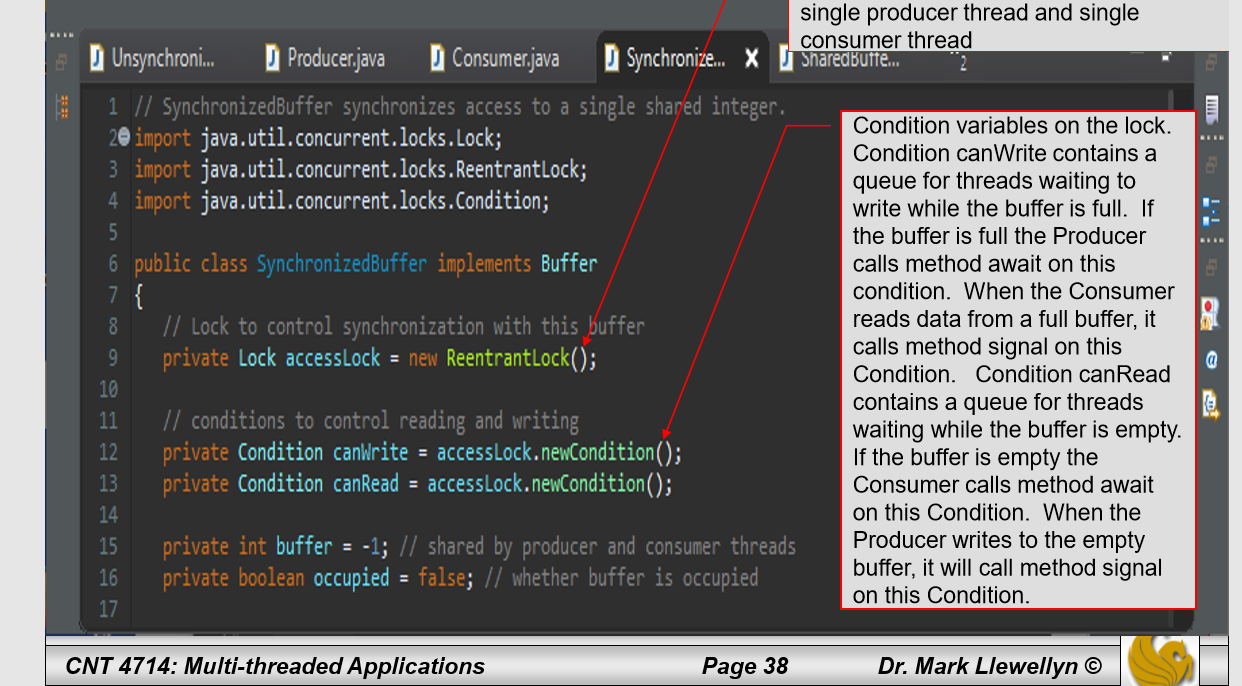
**Random online notes**

*The wait() and join() methods are used to pause the current thread. The* ***wait() is used in with notify() and notifyAll() methods****, but join() is used in Java to wait until one thread finishes its execution.****wait() is mainly used for shared resources****, a thread notifies other waiting thread when a resource becomes free. On the other hand join() is used for waiting a thread to die.*

One most important difference between wait() and join() that is **wait() must be called from synchronized context i.e. synchronized block or method otherwise it will throw IllegalMonitorStateException** but On the other hand, we can call join() method with and without synchronized context in Java.

There are two types of synchronization between threads- mutual exclusion and signal-wait

If a thread has a lock on an object but is now waiting for some condition, it will release the lock and go into a waiting state. To wait on a condition variable, the thread can call the condition’s await() method, which releases the associated lock and places the thread into a wait state for the condition, and other threads can try to obtain the lock.



* Here there are two conditions which are both queues. One ensures that the consumer doesn’t try to read if the buffer is empty, the other ensures that the producer doesn’t try to write if the buffer is full. If either case is true, await() is called from either the producer/consumer.

<https://stackoverflow.com/questions/10395571/condition-vs-wait-notify-mechanism>

<https://stackoverflow.com/questions/15680422/difference-between-wait-and-blocked-thread-states>

Hi Dylan!  
  
An Executor is essentially a "factory" for producing threads and services associated with threads (think asynchronous tasks).  In  Java an Executor is essentially an implementation of a  thread pool with services associated with the pool.  This allows an application to create as many threads as required during the lifetime of the application without the need to create new threads manually.  Implementing your threads for this project using the Runnable interface is the correct way to go (the other way is to extend the Thread class as we discussed in class).  Since we need threads in our application, we need to go with either the Runnable interface or extending the Thread class...this is independent of how the threads are managed.  Yes, you could create and start the threads manually for the project application.  However, since we are implementing a simulation model and the input requirements are not known in advance of execution (i.e., the number of stations required by the simulation), manually creating and starting the threads would be a little bit clumsy.  You could do it in a loop, but this is where the Executor comes in very handy.  You'll create an Executor to manage a thread pool (containing the maximum number of threads that you would ever need) that will allocate and start just the number of threads needed for each run of your simulation.  This is a very typical thing to see in threaded enterprise applications. Basically the Executor provides the application flexibility in thread deployment and low overhead in thread creation and management. Yes, I would definitely recommend using an Executor object for this application to gain some familiarity with the technique.  I'll explain a bit in class as you work on the project as well.  I hope this helps a bit.  
  
Thanks for kind words as well.  
  
Regards,  
  
  
Mark  
  
Dr. Mark Llewellyn  
Department of Computer Science  
University of Central Florida  
4000 Central Florida Blvd.  
Bldg. 116  
Orlando, Florida 32816-2362  
  
Voice: 407-823-2790  
Email: markl@cs.ucf.edu



**DR**

