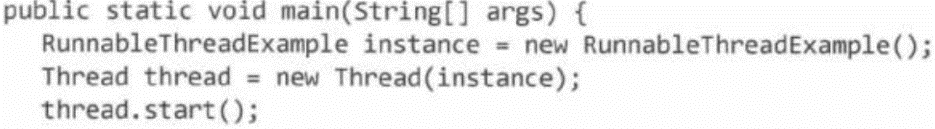
**Knowledge and Theory: Threads and Locks**

**Two Ways to Implement a Thread:**

1. Extending the thread class.
2. Implementing the runnable interface. Once you create an object that extends the “Runnable” interface, you pass it in the constructor of a thread object. This allows you to have threads like “Connection Thread” that can also inherit other classes.



1. There is no particular advantage besides the latter allowing you to inherit other classes. Keep in mind that if you pass in the same object between two threads, you must worry about synchronization issues.

* INCLUDE AGENTS, SYNCHRONIZED BLOCK, WAIT INFO!!

|  |
| --- |
| **The Four Conditions for Deadlock to Occur:**   1. **Mutual Exclusion:**Only one process can access a resource at a given time. *(Or, more accurately, there is limited access to a resource. A deadlock could also occur if a resource has limited quantity.)* The resource could be a printer (where only one can access it at a time), or it could be a stove (where there is only four so only four can access it at a time). 2. **Hold and Wait:**Processes already holding a resource can request and wait for additional resources, without relinquishing their current resources. For example, suppose there is a synchronized method. If a process can get the lock, enter this method, and then wait for something else (while still holding the lock) then deadlock can occur. If however, the process must either wait and release the lock or continue (i.e. your unable to get the resource so move on) with the lock, then deadlock cannot occur. 3. **No Preemption**: One process cannot forcibly remove another process'resource. If a process can force another process let go of the resource, deadlock cannot happen. 4. **Circular Wait:**Two or more processes form a circular wait. For example, process X cannot start activity 1 until process Y relinquishes resource 1. Process Y cannot relinquish resource 1 until process X is done activity 1. In addition, for circular wait to occur, the above must occur. Process X cannot pre-empt process Y. Only one process can use resource 1 at a time. Process Y can hold and wait.  * In order to remove deadlock, you must remove one of these four conditions. Usually, # 4 is the one that gets removed. To do this you must programmatically ensure that a circular wait never occurs. * You could also include pre-emption. This gives process the ability to force someone out of a resource after a given time when you have an idea that deadlock is ocurring. * You could also not allow processes to hold and wait at the same time. For example, imagine someone’s clothes are done in the washer and they are waiting for the drier to be free. Imagine the person’s clothes in the drier are being stored there as a bin and are waiting for the washer to be free. The real problem here is people who won’t hold and resource and then wait at the same time. |