Lucas Rosa
03/18/16
l2857275
COP 4600

Assignment 3: Java

According to the professor and the text for the class, a monitor is an object which only allows one of its methods to be executed at once. In Java, a monitor is associated with each object. [1] For the rest of this paper, all monitors mentioned are within in the context of the Java programming language.

The monitor enforces mutually exclusive access to synchronized methods invoked on the associated object. {1] Mutual exclusion is an important concept used for managing contention, of multiple threads, on shared data. When a thread calls a synchronized method on an object there are two possible states the monitor for that object can be in (*unowned or owned*). [1] The JVM is responsible for checking the monitor. [1] If the monitor is 'unowned,' then ownership is given to the calling thread. [1] When the monitor is 'unowned' the thread is allowed to proceed with the method call. If the monitor is owned by another thread, the calling thread will be put on hold until the monitor becomes available. [1] Eventually, when a thread exits a synchronized method, it releases the monitor. [1] Releasing the monitor allows a waiting thread to proceed with its synchronized method call. {1]

The monitors we learned in class were vaguer than the concepts in the Java implementation of a monitor. Although, they can be considered similar on an abstract level. In both cases the monitor is an object in which only one function of a monitor can be executed at the same time, and there are internal condition variables. It is important to note that Java monitors are not like traditional critical sections. [1] Declaring a method synchronized does not imply that only one thread may execute that method at a time. [1] What it really means it that only one thread may invoke that method on a particular object at any given time. [1] This is credited to the fact that Java monitors are associated with objects, not with block of code.

Java also provides some other synchronization functionality. One example being a semaphore. Conceptually, a semaphore maintains a set of permits. [2] Each attempt at acquiring the object is blocked if necessary until a permit is available. [2] Every time the object is released a permit is added, potentially releasing a blocked acquirer. [2] A key difference in this implementation is that no actual permit objects are used; the Semaphore just keeps a count of the number of available permits and acts as it should. [2]

For a class in Java to satisfy the monitor's fundamental mutual exclusion property the declaration of it's methods must contain the statement 'synchronized.' In Java all objects have an associated monitor, so it is more about making sure that the methods that need this property are declared properly. For example:

Lucas Rosa
03/18/16
l2857275
COP 4600

```
class Counter
{
        private int count = 0;
        public void synchronized Increment() {
            int n = count;
            count = n+1;
        }
}
```

[1]

In Java it is possible to mutually exclude critical sections smaller than an entire method without resorting to mutex locks. The statement 'synchronized' creates a mutual exclusion zone (critical section). This statement is not limited to methods and can be used for blocks of code simply by using the statement. [1]

To describe a condition variable, the 'cond' keyword can be used. A condition variable is a kind of queue of processes who are waiting on the same condition. Conditions can be created by use the 'newCondition' method on the lock. A condition is a variable of type 'Condition.' Example:

```
private final Condition notFull = lock.newCondition();
private final Condition notEmpty = lock.newCondition();
```

[3]

References

1. http://www.csc.villanova.edu/~mdamian/threads/javamonitors.html. Villa Nova. Retrieved March 18, 2016.
2. http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/Semaphore.html. Oracle. Retrieved March 18, 2016.
3. http://baptiste-wicht.com/posts/2010/09/java-concurrency-part-5-monitors-locks-and-conditions.html. Retrieved March 18, 2016.