How JVM handles thread synchronization - 2022

**A lock is like a token or privilege** that only one thread can "possess" at any one time. When a thread wants to lock a particular object or class, it asks the JVM. JVM responds to thread with a lock maybe very soon, maybe later, or never. When the thread no longer needs the lock, it returns it to the JVM.

**A monitor is basically a guardian** **in that it watches over a sequence of code**, making sure only one thread at a time executes the code. **Each monitor is associated with an object reference**. It is the responsibility of monitor to watch an arriving thread must obtain a lock on the referenced object.

**A single thread is allowed to lock the same object multiple times since it is reentrant.** JVM maintains a count of the number of times the object has been locked. An unlocked object has a count of zero. When a thread acquires the lock for the first time, the count is incremented to one. Each time the thread acquires a lock on the same object, a count is incremented. Each time the thread releases the lock, the count is decremented. When the count reaches zero, the lock is released and made available to other threads.

**In Java language terminology, the coordination of multiple threads that must access shared data is called synchronization**. **coordination of multiple threads to access shared data is called synchronization.**

**Two opcodes, monitorenter and monitorexit are used by JVM for accomplishing this task.**

When monitorenter is encountered by the Java virtual machine, it acquires the lock for the object referred to by objectref on the stack. If the thread already owns the lock for that object, a count is incremented. Each time monitorexit is executed for the thread on the object, the count is decremented. When the count reaches zero, the monitor is released.