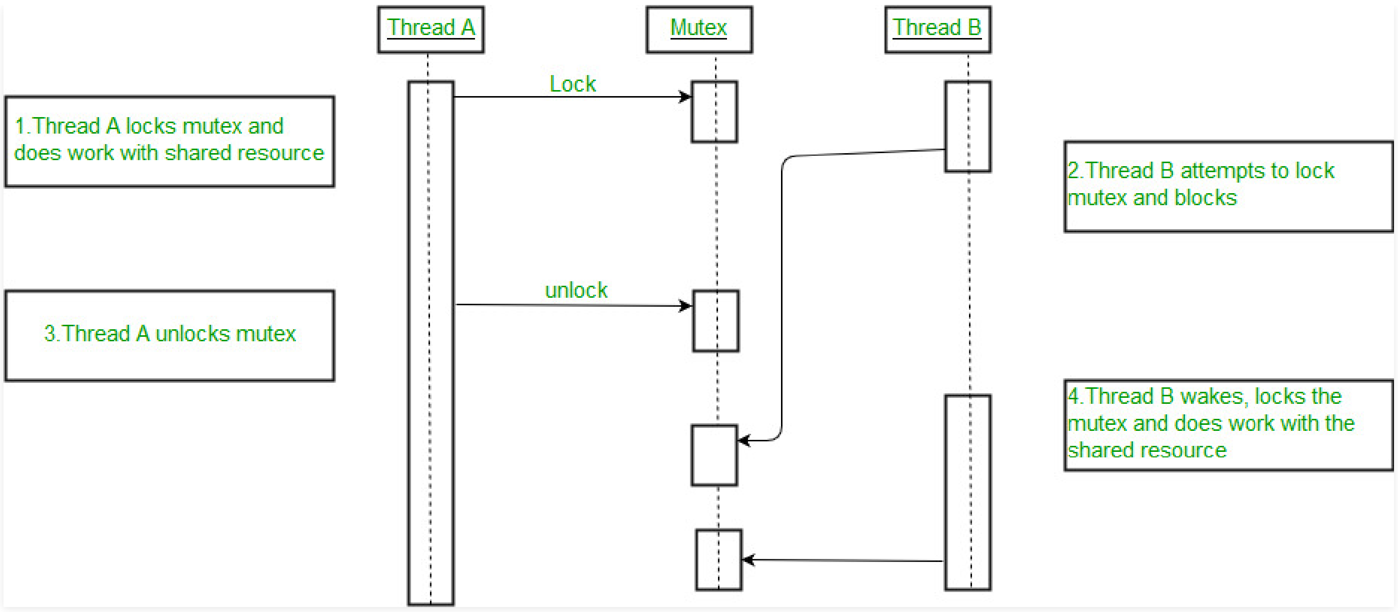
Difference Between Lock and Monitor in Java Concurrency

Difficulty Level : Easy Last Updated : 18 Dec, 2021

Java Concurrency basically deals with concepts like multithreading and other concurrent operations. This is done to ensure maximum and efficient utilization of CPU performance thereby reducing its idle time in general. Locks have been in existence to implement multithreading much before the monitors have come to usage. Back then locks (or mutex) were parts of threads inside the program that worked on flag mechanisms to synchronize with other threads. They have always been working as a tool to provide synchronous access control over resources and shared objects. With further advancements, the use of monitors started as a mechanism to handle access and coordinate threads which proved to be more efficient, error-free, and compatible in object-oriented programs. Before we move on to find the differences between the two let’s have a closer look into each of them.

Overview of Lock (or Mutex)

Lock originally has been used in the logical section of the threads that were used to provide synchronized access control between the threads. Threads checked the availability of access control over shared objects through flags attached to the object that indicated whether or not the shared resource is free (unlocked) or busy (locked). Now the concurrency API provides support of using locks explicitly using Lock Interface in java. The explicit method has a finer control mechanism as compared to the implicit implementation of locks using monitors. Before we move on to discuss monitors let us look at an illustration that demonstrates the functioning of basic locks.


Thread A
Mutex
Thread B
Lock
1.Thread A locks mutex and does work with shared resource
2.Thread B attempts to lock mutex and blocks
unlock
3.Thread A unlocks mutex
4.Thread B wakes, locks the mutex and does work with the shared resource

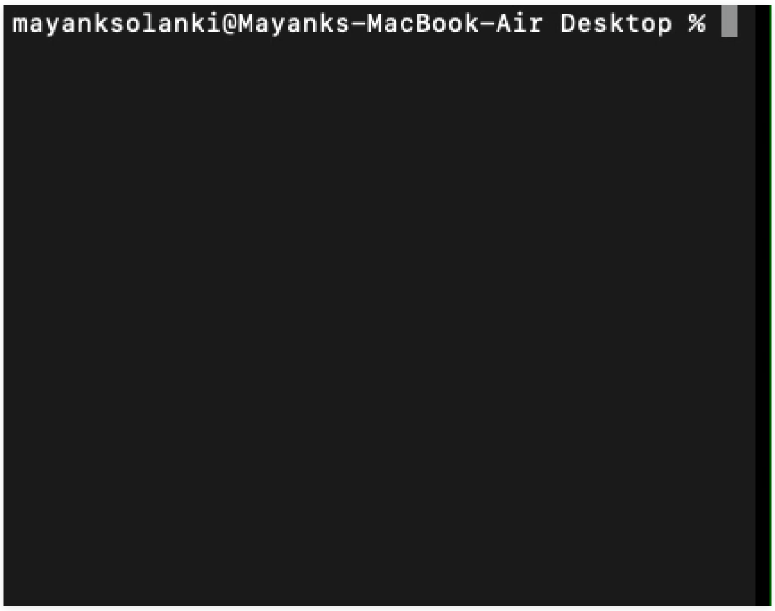
Monitor – Overview

Monitor in Java Concurrency is a synchronization mechanism that provides the fundamental requirements of multithreading namely mutual exclusion between various threads and cooperation among threads working at common tasks. Monitors basically 'monitor' the access control of shared resources and objects among threads. Using this construct only one thread at a time gets access control over the critical section at the resource while other threads are blocked and made to wait until certain conditions. In Java, monitors are implemented using synchronized keyword (synchronized blocks, synchronized methods or classes). For example, let's see how two threads t1 and t2 are synchronized to use a shared data printer object.

```java
// Java Program to Illustrate Monitoe in Java Concurrency

// Importing input output classes
import java.io.*;

// Class 1
// Helepr class
class SharedDataPrinter {

    // Monitor implementation is carried on by
    // Using synchronous method

    // Method (synchronised)
    synchronized public void display(String str)
    {
        for (int i = 0; i < str.length(); i++) {
            System.out.print(str.charAt(i));

            // Try-catch bloc kfor exceptions as we are
            // using sleep() method
            try {

                // Making thread to sleep for very
                // nanoseconds as passed in the arguments
                Thread.sleep(100);
            }
            catch (Exception e) {
            }
        }
    }
}

// Class 2
// Helper class extending the Thread class
class Thread1 extends Thread {

    SharedDataPrinter p;

    // Thread
    public Thread1(SharedDataPrinter p)
    {

        // This keyword refers to current instance itself
        this.p = p;
    }

    // run() method for this thread invoked as
    // start() method is called in the main() method
    public void run()
    {

        // Print statement
        p.display("Geeks");
    }
}

// Class 2 (similar to class 1)
// Helper class extending the Thread class
class Thread2 extends Thread {

    SharedDataPrinter p;

    public Thread2(SharedDataPrinter p) { this.p = p; }

    public void run()
    {

        // Print statement
        p.display(" for Geeks");
    }
}
```

```
// Class 3
// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Instance of a shared resource used to print
        // strings (single character at a time)
        SharedDataPrinter printer = new SharedDataPrinter();

        // Thread objects sharing data printer
        Thread1 t1 = new Thread1(printer);
        Thread2 t2 = new Thread2(printer);

        // Calling start methods for both threads
        // using the start() method
        t1.start();
        t2.start();
    }
}
```

Output:



```
mayanksolanki@Mayanks-MacBook-Air Desktop %
```

Finally wrapping off with the article let us discuss the major differences between Lock and Monitor in concurrency in java that is pictorially depicted in the image below shown as follows:

Lock (Mutex)	Monitor
Have been used since the coining of Multithreading concepts.	Came into existence with later developments in the field.
Usually in the form of a data field or flag that helps implement coordination.	Synchronicity is implemented via a construct mechanism.A similar
Critical Section (the lock/unlock functions and other operations on the shared object) is a part of the thread itself.	Similar mechanism of lock/unlock for synchronization along with operational functions (such as read/write) is present with the shared object only.
Implementation of mutual exclusion (execution of one thread preventing others' execution) and cooperation (threads working on a common task) is the responsibility of the threads.	Mutual Exclusion between different set of threads and cooperation (if needed) is all handled by the shared resource itself.
Loosely linked mechanism as all the threads are independent and handle their synchronization in access control themselves.	The mechanism is quite robust and reliable as everything is managed at the resource side only.
This method is highly prone to errors when locking time and the constructed mechanism use thread synchronization operation time slice are comparable. There is a good chance that while a thread puts a lock its time slice gets over and the other thread starts working on the resource.	The monitors are well designed to work with small thread pools and perform very efficiently unless inter thread communication becomes a necessity.
Ready queue or thread pools are either not present or else handled by the operating system.	Threads wait in queues managed by the shared object they all are trying to access control over.
Locks independently are not much in use and are implemented much less widely.	Monitors intrinsically use inter-thread locks only and are much more in usage.

Note:

As we see monitors themselves are implemented with the necessary support of lock s, it is often said that they are not different but complementary in the nature of their existence are operating