

DAY 23:

ASSIGNMENT 2:

Task 2: States and Transitions

Create a Java class that simulates a thread going through different lifecycle states: NEW, RUNNABLE, WAITING, TIMED_WAITING, BLOCKED, and TERMINATED. Use methods like sleep(), wait(), notify(), and join() to demonstrate these states..

```
public class ThreadLifecycleDemo2 {  
    private static final Object lock = new Object();  
  
    public static void main(String[] args) {  
        Thread thread = new Thread(new Runnable() {  
            @Override  
            public void run() {  
                try {  
                    // Thread in RUNNABLE state  
                    System.out.println("Thread state: " +  
Thread.currentThread().getState() + " (RUNNABLE)");  
  
                    // Thread in TIMED_WAITING state  
                    System.out.println("Thread going to sleep...");  
                    Thread.sleep(3000);  
                }  
            }  
        });  
        thread.start();  
    }  
}
```

```
        System.out.println("Thread state: " +  
Thread.currentThread().getState() + " (TIMED_WAITING)");
```

```
        synchronized (lock) {  
            // Notify main thread to proceed  
            lock.notify();  
  
            // Thread in WAITING state  
            System.out.println("Thread waiting for lock...");  
            lock.wait();  
            System.out.println("Thread state: " +  
Thread.currentThread().getState() + " (WAITING)");  
        }  
  
        // Simulating some work  
        System.out.println("Thread resumed and completing...");  
  
    } catch (InterruptedException e) {  
        System.out.println("Thread interrupted.");  
    }  
}  
});
```

```
// Thread in NEW state  
System.out.println("Thread state: " + thread.getState() + " (NEW)");
```

```
synchronized (lock) {
```

```
thread.start();

// Main thread waits to ensure the other thread gets to TIMED_WAITING
state
try {
    Thread.sleep(1000);
    System.out.println("Main thread waiting for notification...");
    lock.wait();
} catch (InterruptedException e) {
    e.printStackTrace();
}

// Thread in BLOCKED state
System.out.println("Thread state: " + thread.getState() + " (BLOCKED)");

// Notify the other thread to wake up from WAITING state
lock.notify();
}

// Wait for the thread to finish
try {
    thread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}

// Thread in TERMINATED state
```

```
        System.out.println("Thread state: " + thread.getState() + "  
(TERMINATED)");  
    }  
}
```

Explanation:

1. ***NEW State***: The thread is created but not yet started. We print the state before calling start().
2. ***RUNNABLE State***: The thread is started and is executing. We print the state after calling start().
3. ***TIMED_WAITING State***: The thread calls Thread.sleep(3000) to sleep for 3 seconds, demonstrating the timed waiting state.
4. ***WAITING State***: The thread waits on an object monitor using lock.wait(), putting it in the waiting state.
5. ***BLOCKED State***: This state is demonstrated by having the main thread acquire the lock while the other thread tries to reacquire it, making the other thread go into the blocked state.

example shows how to coordinate the main thread and the demo thread to observe various states effectively. 6.

TERMINATED State: After the thread completes execution, we print the state again, which will be terminated.

Running the Program:

When you run this program, you will see the thread transitioning through the different states as described. The synchronized block and wait()/notify() calls ensure that the thread moves through the WAITING and BLOCKED states. The join() call ensures the main thread waits for the demo thread to complete before printing the terminated state