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..

ANS:

```
package com.Day23;
public class Task2 {
  private static final Object monitor = new Object();
  public static void main(String[] args) {
    Thread thread = new Thread(new LifecycleTask());
    System.out.println("Thread state after creation: " + thread.getState()); // NEW
    thread.start();
    System.out.println("Thread state after calling start(): " + thread.getState()); // RUNNABLE
    try {
      Thread.sleep(100); // Ensuring the thread has time to start and possibly wait.
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println("Thread state after starting: " + thread.getState()); // RUNNABLE or WAITING
    synchronized (monitor) {
      monitor.notify();
    }
    try {
      thread.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    System.out.println("Thread state after termination: " + thread.getState()); // TERMINATED
  }
  static class LifecycleTask implements Runnable {
    @Override
    public void run() {
      try {
        synchronized (monitor) {
          System.out.println("Thread state inside synchronized block: " +
Thread.currentThread().getState()); // BLOCKED if another thread holds the lock, otherwise
RUNNABLE
          monitor.wait(); // WAITING
        }
        Thread.sleep(100); // TIMED_WAITING
        synchronized (monitor) {
          // simulate some work
        }
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
      System.out.println("Thread state before termination: " + Thread.currentThread().getState()); //
RUNNABLE
    }
  }
}
```

OUTPUT:

```
Thread state after creation: NEW
Thread state after calling start(): RUNNABLE
Thread state inside synchronized block: RUNNABLE
Thread state after starting: WAITING
Thread state before termination: RUNNABLE
Thread state after termination: TERMINATED
```