**Coral S. Schmidt #148830**

**public** **class** TestAndSetSemaphore {

**private** **volatile** **boolean** lock;

**public** TestAndSetSemaphore() {

        lock = false; *// Semaphore is initially unlocked*

    }

**public** **void** acquire() {

        while (testAndSet(lock)) {

*// Busy waiting until the lock becomes false*

        }

    }

**public** **void** release() {

        lock = false; *// Release the lock*

    }

*// Atomic test-and-set operation*

**private** **synchronized** **boolean** testAndSet(**boolean** value) {

**boolean** previous = value;

        value = true; *// Set to true to indicate lock attempt*

        return previous;

    }

}

This Java pseudocode demonstrates the implementation of binary semaphore operations using the test-and-set instruction. The `TestAndSetSemaphore` class encapsulates the functionality of a binary semaphore, providing methods for acquiring and releasing the semaphore. The `lock` boolean variable serves as the semaphore's state, where `false` indicates that the semaphore is available, and `true` signifies that it is currently locked.

In the `acquire()` method, the pseudocode utilizes a busy-waiting loop to attempt to acquire the semaphore repeatedly. Within this loop, the `testAndSet()` method is called to atomically set the `lock` variable to `true,` indicating an attempted lock acquisition. If the `testAndSet()` operation returns `true`, indicating that the lock was already held, the loop continues until the lock becomes available. Conversely, the `release()` method sets the `lock` variable back to `false,` signaling the release of the semaphore. This pseudocode effectively demonstrates how test-and-set instructions can implement mutual exclusion through a binary semaphore in Java, ensuring that only one thread can access the critical section simultaneously.