Synchronizing Shared Resources

public class Counter {

private int count = 0;

// Synchronized method to ensure thread-safe access to the counter

public synchronized void increment() {

count++;

}

public int getCount() {

return count;

}

}

public class SynchronizedExample extends Thread {

private Counter counter;

public SynchronizedExample(Counter counter) {

this.counter = counter;

}

@Override

public void run() {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

}

public static void main(String[] args) throws InterruptedException {

Counter counter = new Counter();

// Create and start multiple threads

Thread thread1 = new SynchronizedExample(counter);

Thread thread2 = new SynchronizedExample(counter);

thread1.start();

thread2.start();

// Wait for threads to finish

thread1.join();

thread2.join();

System.out.println("Final counter value: " + counter.getCount());

}

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Explanation:**

• The Counter class contains a shared variable count that will be incremented by multiple threads.

• The increment() method is synchronized, which means only one thread can execute it at a time, ensuring thread safety.

• In the SynchronizedExample class, each thread will increment the counter 1000 times. We use join() to make sure the main thread waits for the threads to finish before printing the final value of the counter.