Relatório: Exemplo de Mutex em Java

Este relatório documenta o exemplo de implementação de um mutex simples em Java, onde duas threads incrementam uma variável compartilhada de forma segura. A solução utiliza métodos lock() e unlock() para exclusão mútua.

1. Código Fonte

```
public class MutexCounterExample {

    public static class SimpleMutex {
        private boolean locked = false;
        public synchronized void lock() throws InterruptedException {
            while (locked) {
                wait();
            }
            locked = true;
        }
        public synchronized void unlock() {
            locked = false;
            notify();
        }
    }

    private static int sharedCounter = 0;
    private static final SimpleMutex mutex = new SimpleMutex();
    private static final int ITERATIONS = 1_000_000;

    public static void main(String[] args) throws InterruptedException {
        Runnable counterTask = () -> {
            try {
                for (int i = 0; i < ITERATIONS; i++) {
                    mutex.lock();
                    try {
                        sharedCounter++;
                    } finally {
                        mutex.unlock();
                    }
                }
                System.out.println(Thread.currentThread().getName() + " finalizou.");
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        };

        Thread t1 = new Thread(counterTask, "Contador-1");
        Thread t2 = new Thread(counterTask, "Contador-2");

        t1.start();
        t2.start();
        t1.join();
        t2.join();

        System.out.printf("Valor final = %d (esperado %d)%n", sharedCounter, 2 * ITERATIONS);
    }
}
```

2. Explicação Detalhada

SimpleMutex: implementa um mutex bloqueante usando *synchronized*, com métodos *lock()* e *unlock()*. O método *lock()* usa um loop **while** para esperar até que *locked* seja falso, chamando *wait()*. Já o *unlock()* define *locked* como falso e chama *notify()* para acordar uma thread em espera.

3. Conclusão

Com esta implementação, garantimos que somente uma thread por vez execute a seção crítica onde o contador compartilhado é incrementado, evitando race conditions e garantindo o valor final esperado.