Thread Control and Deadlocks Lab Documentation

Author: Ndayambaje Virgile

1. Thread Interruption

Concept:

Thread interruption is a mechanism that allows a thread to be stopped gracefully by signaling it to stop, using the interrupt() method.

Implementation:

In the ThreadInterruptionDemo class, we simulate a task that checks if it has been interrupted using Thread.interrupted() and exits when it is. This method allows the thread to exit during sleep or any other blocking operation.

Code snippet:

```
Thread task = new Thread(() -> {
    for (int i = 0; i < 10; i++) {
        if (Thread.interrupted()) {
            return;  // Exit gracefully
        }
        Thread.sleep(1000);
    }
});
```

2. Fork/Join Framework

Concept:

The Fork/Join framework in Java allows parallel processing by recursively splitting a task into smaller chunks and then processing them in parallel.

Implementation:

In ForkJoinTaskDemo, a large array of integers is split into smaller tasks using the RecursiveTask class. The task processes the chunks in parallel, and the result is the sum of the array.

Code snippet:

```
SumTask leftTask = new SumTask(array, start, mid);
SumTask rightTask = new SumTask(array, mid, end);
invokeAll(leftTask, rightTask);
return leftTask.join() + rightTask.join();
```

3. Deadlock Scenario

Concept:

A deadlock occurs when two or more threads block each other by holding a lock and waiting for the other thread to release its lock.

Implementation:

In DeadlockScenario, two threads acquire two locks (lock1 and lock2) in opposite order, creating a deadlock. Both threads block each other, and the program cannot proceed.

Code snippet:

```
synchronized (lock1) {
    Thread.sleep(100);
    synchronized (lock2) { ... }
}
```

4. Deadlock Prevention

Concept:

One way to prevent deadlocks is to use ordered locking. By ensuring that all threads acquire locks in the same order, we can avoid circular wait conditions.

Implementation:

In DeadlockPrevention, both threads acquire the locks in the same order (lock1 first, then lock2), preventing deadlocks from occurring.

Code snippet:

```
synchronized (lock1) {
    synchronized (lock2) { ... }
}
```

Conclusion

This lab demonstrates important multithreading concepts such as thread interruption, parallel processing with Fork/Join, deadlock scenarios, and techniques to prevent deadlocks. Understanding and implementing these techniques is crucial for building robust and concurrent applications.