CS108, Stanford
Fall, 2008-09

Handout #23
Osvaldo Jiménez

Thread Array Example

Thanks to Nick Parlante for much of this handout

Here's an example that takes an array of ints and forks off multiple threads to add up all the `ints` in parallel. Uses a `Semaphore` to notice when all the workers are done.

```
import java.util.*;
import java.util.concurrent.*;

/*
 * This is a thread example launches multiple threads to
 * sum up all the elements in an array concurrently.
 * Uses worker threads and a semaphore.
 */
class ArrayThreading  {
    private int[] array;
    private Semaphore allDone;

    // The array contains the values
    // 0, 1, 2... len-1
    public ArrayThreading(int len) {
        array = new int[len];
        for (int i=0; i<len; i++) {
            array[i] = i%10;
        }
    }

    // Worker inner class to add up a section of the array.
    private class Worker extends Thread {
        int start;
        int end;
        long sum;

        // Note the start and end indexes for this worker
        // in the array. Goes up to but not including end index.
        Worker(int start, int end) {
            this.start = start;
            this.end = end;
            sum = 0;
        }

        // Computes the sum for our start..end section
        // in the array (client should call getSum() later).
        public void run() {
            for (int i=start; i<end; i++) {
                sum += array[i];
            }
            allDone.release();
        }

        public long getSum() {
            return sum;
        }
    }
    // This is the key method -- launch all the workers,
    // wait for them to finish.
    public void runParallel() {
        int numWorkers = 10;
        allDone = new Semaphore(0);

        // Make and start all the workers, keeping them in a list.
        List<Worker> workers = new ArrayList<Worker>();
        int lenOneWorker = array.length / numWorkers;
        for (int i=0; i<numWorkers; i++) {
            int start = i * lenOneWorker;
            int end = (i+1) * lenOneWorker;
            // Special case: make the last worker take up all the excess.
            if (i==numWorkers-1) end = array.length;
            Worker worker = new Worker(start, end);
            workers.add(worker);
            worker.start();
        }

        // Wait to finish (this strategy is an alternative to join())
        try {
            allDone.acquire(numWorkers);
            // Note: here use acquire(N) ..
            // could instead init semaphore with -9 and use
            // regular acquire() here
        } catch (InterruptedException ignored) {
        }

        // Gather sums from workers (yay foreach!)
        int sum = 0;
        for (Worker w: workers) sum += w.getSum();

        System.out.println("len:" + array.length + " sum:" + sum + " workers:" + numWorkers);
    }

    /***
     Example command line calls:
     $ java ArrayThreading 10000000     ## 10M -- fine
     len:10000000 sum:45000000 workers:10
     $ java ArrayThreading 100000000    ## 100M too big
        Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at ArrayThreading.<init>(ArrayThreading.java:16)
        at ArrayThreading.main(ArrayThreading.java:92)
     $ java -Xmx500m ArrayThreading 100000000  ## specify 500M memory -- fine
     len:100000000 sum:450000000 workers:10
    ***/
    public static void main(String[] args) {
        // command line argument: array_length
        int len = Integer.parseInt(args[0]);

        ArrayThreading at = new ArrayThreading(len);
        at.runParallel();
    }
}
```