Threading Class for Exercise D1

The java source code is available on the class web site

```java
public class BusyBody extends Thread {
    static long start;
    static long end;
    static long time;
    private int id;
    public BusyBody(int id) {
        this.id = id;
    }
    public double getBusy() {
        int max = 500000;
        double total = 1.0;
        for (int i=0; i<max; i++) {
            total = total + (1.0 + Math.random() * Math.random() * Math.random());
        }
        System.out.print( "[Worker " + id + ": " + (int)total + "] " );
        return total;
    }
    public void run() {
        getBusy();
    }
    public static void main(String[] args) {
        int maxThreads = 20;
        System.out.println("Number of workers & threads = " + maxThreads);
        BusyBody[] busy = new BusyBody[maxThreads];
        for (int i=0; i<maxThreads; i++) {
            busy[i] = new BusyBody(i);
        }
        double ans = 0.0;
        // GO Sequential
        start = System.currentTimeMillis();
        for (int i=0; i<maxThreads; i++) {
            ans= busy[i].getBusy();
        }
        end = System.currentTimeMillis();
        time = end - start;
    }
```
System.out.println("\nSequential Time.. = " + time);

// get the time so we can measure same task run as threads
start = System.currentTimeMillis();

// DO: Launch each worker ( busy[i] ) as a separate thread

// DO: use join() to wait for these threads to complete

// measure the amount of time it took
end = System.currentTimeMillis();
time = end - start;

System.out.println("\nMulti threaded Time.. = " + time);
DataManager

/**************************
 * DataManager maintains an internal array in order to keep
 * the array uniform - all values must always be the same.
 * In a single threaded world this code works fine but it is not
 * thread safe. See the main method in DataWorker for this code
 * run with multi-threading.
 */

public class DataManager {

private int [] data;

public DataManager (int size) {
    data = new int[size];
}

public int getSize () {
    return data.length;
}

public void uniformUpdate (int val) {
    // compute a random value for yielding
    int loopYield = 1 + (int)(10 * Math.random());

    // set all array values to a single value
    for (int i=0; i<data.length; i++) {
        data[i]=val;
        System.out.print(val);
        if (i % loopYield == 0) Thread.yield();
    }
    System.out.println();
}

public boolean isDataUniform () {
    int val = data[0];  // grab first elt
    for (int i=1; i<data.length; i++) {
        if (data[i] != val) return false;
    }
    return true;  // checked all values
}
```java
public int getTotal () {
    int total = 0;
    for (int i=0; i<data.length; i++) {
        total+=data[i];
    }
    return total;
}

DataWorker

/* A DataWorker instance knows its DataManager and has a 
* doWork() method that gets its work done. This work can 
* be run as a separate thread when another thread calls start() 
* which will trigger run -- and then doWork() 
* 
* Every instance also has an id. 
* 
* The main() method creates some workers and launches them as 
* threads. It prints the state of the data - uniform or not. 
* 
* The DataManager class is defined so it should always have ONE value 
* in its internal array buffer. 
* 
* todo: modify either DataWorker or DataManager so that 
* isDataUniform returns true in a multithreaded environment */

public class DataWorker extends Thread {
    private int id;
    private DataManager dm;

    public DataWorker (int id, DataManager dm) {
        this.id = id;
        this.dm = dm;
    }

    public void doWork() {
        dm.uniformUpdate(this.id);
    }

    public void run() {
        doWork();
    }

    public static void main(String[] args) {
        int size = 50;
        DataManager dm = new DataManager(size);

        int nThreads = 2;
        DataWorker [] workers = new DataWorker[nThreads];

        // create the workers
        for (int i=0; i<nThreads; i++) {
            workers[i] = new DataWorker(i+1, dm);
        }
    }
}
```
get workers to earn their keep
for (int i=0; i<nThreads; i++) {
    // sequential : workers[i].doWork();
    // multithreaded: workers[i].start();
    workers[i].start();
}

main thread must wait if multithreads -
// and since join may throw InterruptedException
// we must use a try..catch block
try {
    for (int i=0; i<nThreads; i++) {
        workers[i].join();
    }
} catch (InterruptedException e) {}