Homework Assignment: THREAD

As explained in class, your homework assignment is to modify the Monte Carlo simulation that was previously presented in class to run in a multi-threaded environment. You can use Eran's version instead of your own.

All of the notes you need were uploaded to the group site on Friday night.

The class that you will have to write will be called PooledMonteCarlManager. Inside it, there will be a thread pool that will create and start 10 threads. Initially, these threads will be in a waiting state. (See wait/notify functionality or blocking queues in the multithreading lecture pdf.)

When the PooledMonteCarloManager is asked to perform a calculation, it will assign up to 3 threads to the problem. In other words, you will have to find a way to distribute the problem over at most 3 but possibly fewer threads. How you do this is up to you as long as you keep your design flexible and elegant and provide the appropriate unit tests.

When threads finish their calculations, they should be put back into a waiting state and returned to the list of free threads that is maintained by the PooledMonteCarloManager. This is the typical functionality of a thread pool and there is plenty of literature in computer science explaining how to write one.

Here is the way a problem is passed to the PooledMonteCarloManager:

```
public void aPooledMonteCarloManager.solve (
     I_PricingProblemSpecification spec,
     I_SolutionCollector solutionCollector
) {
     // …
}
```

The problem specification will be sent to the solve method using the spec variable. When a solution is found, the solution will be sent to solutionCollecto as follows.

```
     solutionCollector.receiveSolution( aSolution );
```

The reason that the solve method does not return an answer directly is that it should not block the solver from receiving other requests that it can parcel out to free threads.