### **SER422 Spring 2016 Lab1 Sockets and Threads Review Exercise**

**Assigned 1/19/16, due 1/31/16 at 11:59pm via online submission to Blackboard**

In this lab you will write a client-server calculator program that calculates a running total. You can start from the SockClient[State].java and SockServer[State].java code found in your sample code for the class.

1. (10) Make the server stateful by accepting only a single integer and calculating a running total.
2. (10) Change the protocol by allowing the client to send a "reset" to set the running total back to 0
3. (20) Make the server stateful on a per client basis – meaning a separate running total per distinct client
4. (10) Solve the integer encoding problem
5. (15) Persist the state of your server-side calculator to an XML file and be able to restore it when starting from an XML file. This will add a command-line parameter to your SockServer.
6. (35) Improve server throughput by making it multi-threaded yet threadsafe. Introduce an artificial delay parameter on the command line so I can test this behavior.

1-4 are fairly self-explanatory. #1 says you should be able to run “java SockClient1 100” and return 100, then run it again “java SockClient1 50” and return 150. #2 says you should be able to run the same as #1, but then do “java SockClient2 <reset command>” and return 0, then “java SockClient2 30” and return 30. #3 says change it per client. In this scenario, you will need to distinguish between clients by having a client id on the command line, “java SockClient3 <id1> 100” outputs 100, then “java SockClient3 <id2> 50” outputs 50, then “java SockClient3 <id1> 25” returns 125. #4 is a data marshaling problem – the given code has a bug when integer values get above a certain range, please fix.

#6 is a bit more involved, which is why it is worth more points. You need to do 2 related things. First, you need to not have the server complete its additional work before allowing a new socket connection (you will have to introduce threads to do this). Second, you will have to make sure that if 2 clients with the same id attempt to modify the same running total, that the operations do not get messed up. What I will be looking for with the latter is that you are threadsafe yet maximize throughput (meaning you add safety in the smartest possible way, not just degenerate back to a full blocking case). To test this you will add a command-line parameter to the server program that is an artificial delay in ms and use the Thread.sleep call to introduce such a delay when each command executes on the server. That is “java SockServer6 5000”.

Number 5 means what it says – be sure to save the internal state of your server to an XML file, and be able to restore from it when the server starts up. The XML format is up to you, but you must save the state of all clients and maintain thread safety.

Note I am requiring you to save each solution as SockServer# and SockClient# where # corresponds to the requirement above. Each requirement should only require a few new lines of code.

**Submission:**

You are to implement your solution **by yourself**, no partners or outside help for this lab! Submit via a zipfile named <asurite>lab1.zip on Blackboard by the due date. You should have 13 files, SockServer[Client]1.java to SockServer[Client]6.java, plus a README.txt that explains design decisions and any command line changes (like for #2 and #3).