README

SER422 Spring 2016 Lab1

A. Command Line Usage

#	Client	Server
1	`java SockClient1 <int>`	`java SockServer1`
2	`java SockClient2 'r'/<int>`	`java SockServer2`
3	`java SockClient3 <client_id> 'r'(reset)/ 't'(total) / <int>`	`java SockServer3`
4	`java SockClient4 <client_id> 'r'(reset)/ 't'(total) / <int>`	`java SockServer4`
5	`java SockClient5 <client_id> 'r'(reset)/ 't'(total) / <int>`	`java SockServer5 <file_name>.xml`
6	`java SockClient6 <client_id> 'r'(reset)/ 't'(total) / <int>`	`java SockServer6 <file_name>.xml <delay>`

B. Design Decisions

#1

- Server was made stateful by keeping the running total in-memory as an Integer variable.

#2

- The client sent data to the server as 'r' or 'i123', where 123 is the end-user input. The input prefixed with 'i' made it easier for the server to distinguish between reset and integer inputs.

#3

- Server was made stateful per client basis by keeping a in-memory HashMap, where each client_id was mapped to its running total.

#4

- The integer encoding problem is solved by sending array of bytes from client side.
- At the server side, a string is built from the byte-array with the help of *inputStream.available()* method. This string was then parsed to an Integer to perform further addition operations.

#5

- XMLWrapper class handles all XML related operations.
- When the server starts up, XMLWrapper object immediately checks whether the XML file is existing or not. If not it creates a new file and returns the document object for it. Otherwise it just returns the document object and keeps it in-memory.

- Method *performOperation()* handles mapping of client_id to appropriate element and then executing the operation and then writing back the results to the file.
- It is essential that dirty reads and writes are prevented, and hence the method *performOperation()* is a synchronized method. Thread safety is achieved by this.
- It is important to note, the method *writeXML()* is called from both synchronized and asynchronized methods. It inherits the synchronous properties of its caller method.
- Design of XML:

```
<root>
  <client>
      <id> 1 </id>
      <running_total> 100 </running_total>
  </client>
</root>
```

#6

- Multi-threading has been implemented, by executing each request from SockClient6 inside a new thread.
- Another major improvement in throughput has been achieved by implementing synchronization per Client_Id.
- Execution of requests with different client_id happens in parallel, while execution of requests with same client_id happens serially (synchronized fashion).
- Dirty reads and writes can happen only when same resources are being shared. Different client_ids will access different resources in the XML and hence parallel execution in this case will not create messed up data.
- This is achieved by using a HashMap where each ClientId is mapped to its own XMLWrapper Object. *{ClientId -> Wrapper Object}*.
- Method *performOperation()* is synchronized as in #5, making it thread safe.
- Threadpool is used to further improve the throughput. A fixed thread pool with 10 workers have been pre-spawned, and hence eliminating the overhead of creating and destroying a new thread every time.