Assignment 1 - Introduction to Java RMI

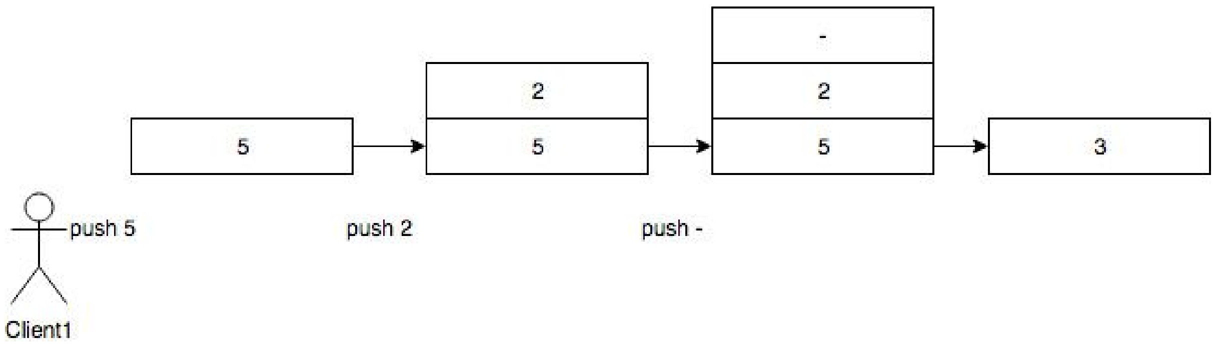
Hideki Yoshinaga | a1658945

One stack one client

Related source files :

- Calculator.java
- CalculatorImplementation.java
- CalculatorServer.java
- CalculatorClient.java

The one stack one client model is simply implemented on top of the Oracle RMI tutorial model. On the server side, there is a stack represented by an ArrayList. Client can remotely calling the methods through the stub. Since there is no way to identify the client, one client could pop other client's stack frame.



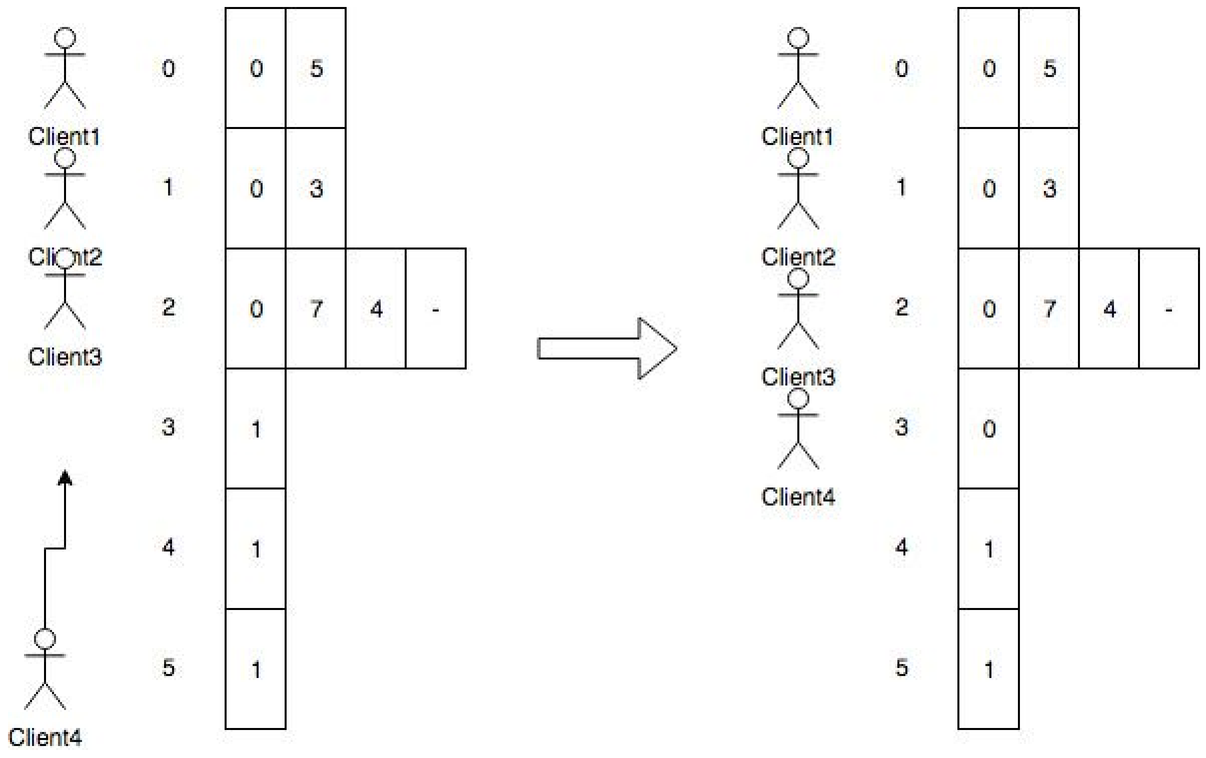
Individual stack multi-threaded client (Bonus)

Related source files :

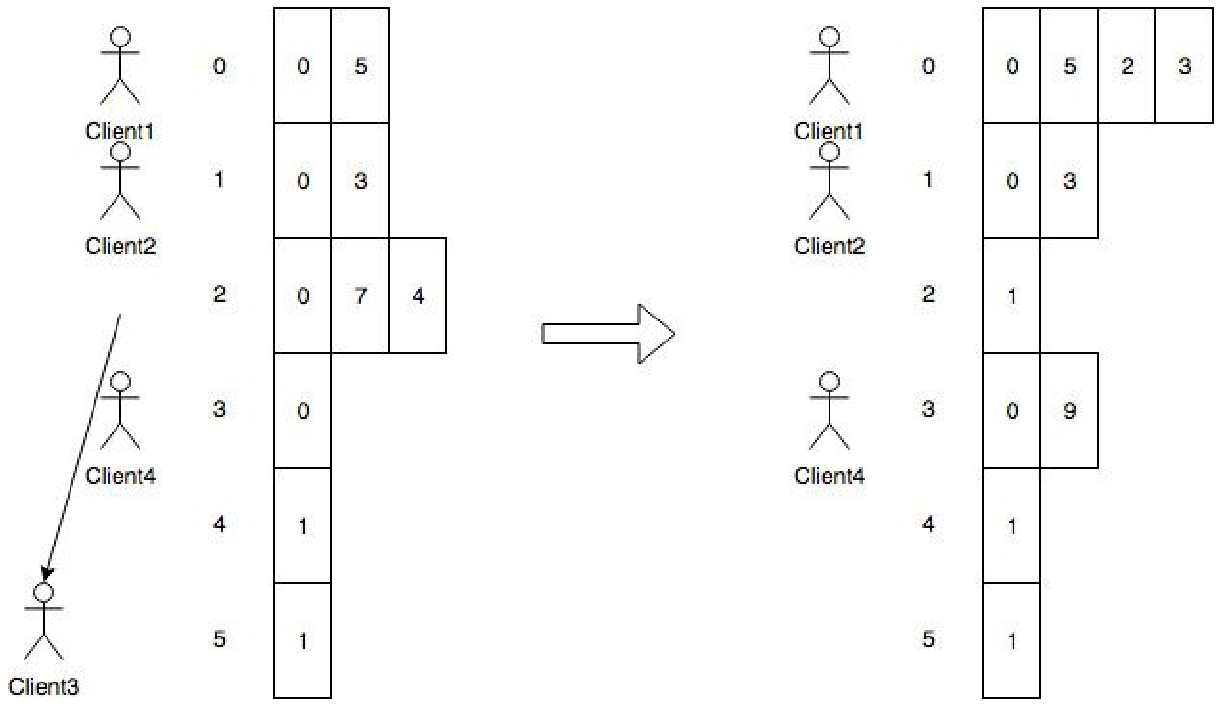
- Calculator1.java
- CalculatorImplementation1.java
- CalculatorServer.java
- CalculatorMultiClient1.java

The individual stack multi-threaded client model is implemented on top of one stack one client model. This time, the stack is a scalable 2 dimensional ArrayList. The ArrayList is initialized to have 100 stacks(ArrayList), the first stack frame is set to 1 to represent the stack is available.

Client needs to call getIndex() method to get an ID number to use the stack. The ID number is actually the index number of the stack. getIndex() method will iteratively check the ArrayList to find an available stack and return its index number. Once the client has the ID, then they can do all the operations now. Since each client(thread) will have their own stack, the operation will not interact with other clients. After the client has done all the jobs, the client can call freeIndex() method to make the ID and the stack available again for other clients.



When new client join



When client leave

One shared stack multi-threaded client

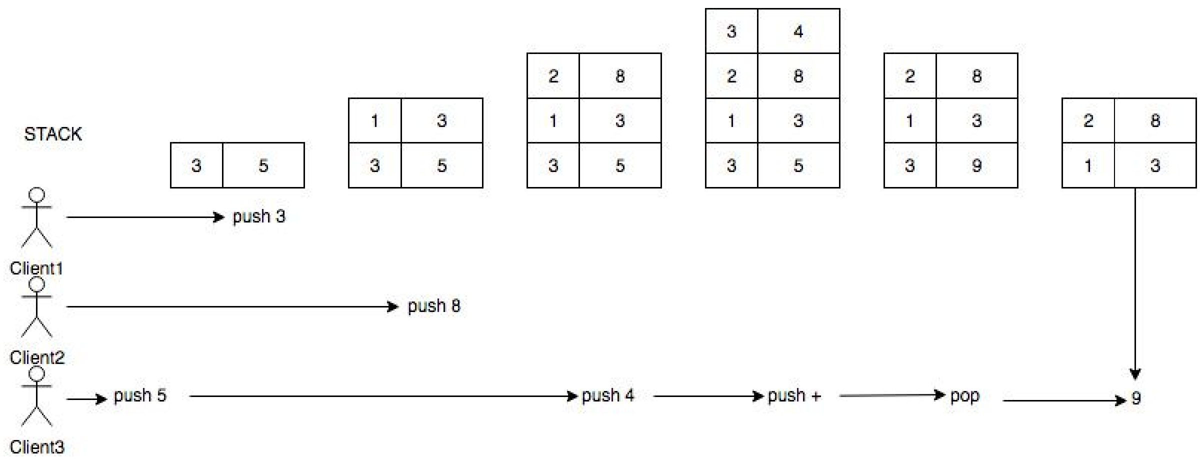
Related source files :

- Calculator2.java
- CalculatorImplementation2.java
- CalculatorServer.java
- CalculatorMultiClient2.java

The one shared stack multi-threaded client model is implemented on top of individual stack multi-threaded client model. However, all client will share the one same stack with other clients. To identify each client, user needs to call genToken() method to get a 8 digits random number and alphabet mixed token to be able to access the stack. The server can identify each client by using these tokens. Tokens are stored in a hashMap as keys with its int value(because I thought it might be easier to deal with int rather than 8 digit String).

Client needs to use the token to do all the operations. This time, the stack has to be synchronized when pushing or popping because its shared with other clients. The stack is designed to have many stack frames on the same stack, each stack frame has the user's hashed token value at the front for identifying which stack frame belongs to which client. When a client want to pop a value for example, the pop method will back travers the shared stack and find the first stack frame that is belong to the client, then return it.

user	token	hash value
client1	JD45G5JK	1
client2	RT2HW5RT	2
client3	3G23FRTM	3



How to test

- **Test cases and answers**

All test cases are in a folder called test and all answers are in a folder called ans. Test case can be edit but need to make sure to edit answer as well. Test cases and answers can also be randomly generated by command `./gen_test.sh` within the assignment1 directory. This will generate 50 test cases with random numbers.

- **How to compile**

In the assigenment1 directory, use the command `javac -d calculator *.java` to compile all java file and put the class files in calculator directory.

- **How to start the RMI registry**

In the assigenment1 directory, type `export CLASSPATH=calculator` to set the CLASSPATH to the directory where keeps the class files. Use the command `rmiregistry &` to run the RMI registry in the background.

- **How to start the Calculator Server**

In the assigenment1 directory, wait a few seconds until the RMI registry is on, then use the command `java -classpath calculator CalculatorServer &` to start the calculator server in the background.

- **To test the one stack one client model**

In the assigenment1 directory, use the command `java  -classpath calculator CalculatorClient` to test the one stack one client model. This command can take an argument (int debug > 0) to show more information for example `java  -classpath calculator CalculatorClient 2`
The test is done by calculating all 50 test cases and automatically compare with the answers.

- **To test the individual stack multi-threaded client model**

In the assigenment1 directory, use the command `java  -classpath calculator CalculatorMultiClient1 [number of threads] ([debug level])` to test the individual stack multi-threaded client model. For example test with 4 threads, type command `java  -classpath calculator CalculatorMultiClient1 4`
The debug level is set to 0 by default, can be set to larger number to display more information.

- **To test the one shared stack multi-threaded client model**

In the assigenment1 directory, use the command `java  -classpath calculator CalculatorMultiClient2 [number of threads] ([debug level])` to test the individual stack multi-threaded client model. For example test with 4 threads, type command `java  -classpath calculator CalculatorMultiClient2 4`
The debug level is set to 0 by default, can be set to larger number to display more information.