Week 4. Lab exercise
Remote Calculator

Felipe Orihuela-Espina

Contents

1 Problem statement

In this exercise you are going to program a simple calculator over a TCP/IP client-server architecture. The problem is only a bit more elaborate than the UDP based client-server architecture example that you have seen in the lecture, but also more interactive; as the user using the client program will be able to type the service command at execution. Although the solution to be developed is meant to be for $n:1$, we shall only simulate 1 client and 1 server.

Our server offers a very simple calculator service. The calculator can only interpret 2 binary operations; addition and multiplication and return to the client the corresponding sum or product to the requests. Our server understands the commands in prefixed notation e.g. $+$ 3 4. The spaces here are important. The server will operate in `double` precision. The server will be a multithreaded server analogous to the example you saw in the lecture, but differently, here the implementation will be using TCP and streams for the passing of the messages over the socket. Also, we shall make the server loop infinite, but we shall make sure that we can remotely kill the server remotely after a certain time period (e.g. 30 secs for practical purposes).

Server will be hosted in the localhost 127.0.0.1 and will be accesible through port 6868.

The client will collect the calculator command in the aformentioned prefixed notation and send the request to the calculator service through a socket.

Both the client and the server messages will be terminated using a # symbol; that is, although the human user, upon being prompted, will type;

```
+ 3 4
```

The client will send the message:

```
+ 3 4\#
```

Analogously, although the service outcome message will be:

```
7.0
```

The server output message will be:

```
7.0\#
```

In total, you will be programming 4 classes:

- The client `Client.java`, responsible for interacting with the user, connecting to the socket and requesting the service.
- The server main class `CalculatorServer.java`, responsible for creating the socket and executing the service loop where the service threads are created.
- The server's service thread class `CalculatorService.java`, responsible for dealing with one client's request, effectively implementing the calculator service.
- A main class `RemoteCalculator.java` capable of launching the server, one client and also of killing the server process after some time.

In *all* class, you are requested to treat exceptions in the method that first can raise them i.e. do not rely in `throws` clauses to deal with exceptions at a later stage. Some exceptions will force you to stop the execution of either the server, the client or the main class without a correct resolution of the request. Make sure that in those cases you exit the program with code 1. Exit with code 0 if program execution is correct. The killing of the server after the time out by the main class is considered a correct execution in this case.

2 What to submit?

This is a formative assignment. It is not assessed. You do NOT need to submit anything!

3 Synchronization

Synchronization here happens in several places:

- In the `RemoteCalculator.java`, the main thread should await for a given amount of time (e.g. 30 secs) for the server and then kill the process.
- In the service loop in `CalculatorServer.java`, the server must wait passively to receive requests before launching a new thread.
- In the client, after seding a request, it has to wait for the outcome to arrive back before reporting it to the user.

4 Class Client

Class `Client` requests the user command, send the message to the server, waits for the outcome to arrive and report the outcome to the user. Complete the missing methods.

```
/*
 * Client.java
 * A calculator client
 * Request an input from a user representing a
    simple addition or product operation,
 * then connects to the server to request the
    service of the operation, and
 * return the user the outcome of the operation.
 */

//Import session
//TO BE COMPLETED

public class Client {

    private Socket clientSocket = null;
    private String userCommand = null; //The user
       command
    private String serviceOutcome = null; //The
       service outcome

    //Class Constructor
    public Client(){
        //Initialize socket and connect to server
        //TO BE COMPLETED
    }

    public void requestService() {
        //TO BE COMPLETED
        //Add terminator character '\#' to the
           service request message
    }

    public void reportServiceOutcome() {
        //TO BE COMPLETED
    }

    //Execute client
    public void execute() {
        //Prompt the user for an operation:
        //TO BE COMPLETED

        //Request service
        try{
            //Request service
            this.requestService();
            //Report user outcome of service
            this.reportServiceOutcome();
            //Close the connection with the server
            this.clientSocket.close();
        }catch(Exception e)
        {// Raised if connection is refused or
           other technical issue
            System.out.println("Client: Exception "
               + e);
        }
    }

    public static void main (String[] args) {
        //TO BE COMPLETED
        System.out.println("Client: Finished.");
        System.exit(0);
    }
}
```

5 Class CalculatorServer

Class `CalculatorServer` is the server main class; it initializes the socket and runs the service loop. Complete the missing methods.

```
/*
 * RemoteCalculator.java
 * The server main class.
 * This server provides a calculator service.
 */

//Import session
//TO BE COMPLETED

public class CalculatorServer{

    private int thePort = 0;
    private String theIPAddress = null;
    private ServerSocket serverSocket =  null;

    //Class constructor
    public CalculatorServer(){
        //Initialize the TCP socket
        //TO BE COMPLETED

        //Initialize the socket and runs the
           service loop
        //TO BE COMPLETED
    }

    //Runs the service loop
    public void executeServiceLoop()
    {
        try {
            //Service loop
            while (true) {
                //Listening for incoming client
                   requests.
                        //TO BE COMPLETED

                //A new request has now arrived;
                   create a service thread to
                   attend the request
                //The service thread will be
                   responsible for sending back the
                   outcome, so
                //this service can now "forget"
                   about that request and move on.
                        //TO BE COMPLETED

            }
        } catch (Exception e){
            //The creation of the server socket can
               cause several exceptions;
            //See
               https://docs.oracle.com/javase/7/docs/api/java/net/Serve
            System.out.println(e);
        }
        System.out.println("Server: Finished.");
        System.exit(0);
    }

    public static void main(String[] args){
        //Run the server
        //TO BE COMPLETED
    }
}
```

6 Class CalculatorService

Class `CalculatorService` is the server service thread class; it implements the calculator service and is also responsible for parsing the incoming user command. Complete the missing methods.

```
/*
 * CalculatorService.java
 * The service threads for the calculator server.
 * This class implements the calculator.
 * Request are made in prefix notation e.g.
 *  "+ 4 3"
 */

//Import session
//TO BE COMPLETED

public class CalculatorService extends Thread{

    private Socket serverSocket =  null;
    private String requestStr = null;
    private String outcome = null;

    //Class constructor
    public CalculatorService(Socket aSocket)  {
        //Launch the calculator service thread
        //TO BE COMPLETED
    }

    //Retrieve the request from the socket
    public String retrieveRequest()  {
        //TO BE COMPLETED
        return this.requestStr;
    }

    //Parse the request command and execute the
       calculation
    public boolean attendRequest()  {
       //TO BE COMPLETED
       //Hint: Use StringTokenizer for parsing the
          user command and a switch statement to
          deal with the different operations
        return flagRequestAttended;
    }

    //Wrap and return service outcome
    public void returnServiceOutcome()  {
       //TO BE COMPLETED
    }

    //The service thread run() method
    public void run() {
        try {
            //Retrieve the service request from the
               socket
            this.retrieveRequest();
            System.out.println("Service thread " +
               this.getId() + ": Request retrieved:
               " + this.requestStr);
            //Attend the request
            boolean tmp = this.attendRequest();
            //Send back the outcome of the request
            if (!tmp)
                System.out.println("Service thread
                   " + this.getId() + ": Unable to
                   provide service.");
            this.returnServiceOutcome();
        }catch (Exception e){
            System.out.println("Service thread " +
               this.getId() + ": " + e);
        }
        //Terminate service thread (by exiting
           run() method)
        System.out.println("Service thread " +
           this.getId() + ": Finished service.");
    }
}
```

7 Class RemoteCalculator

Class `RemoteCalculator` launches independent command windows for the server and the client, and kills the server process after a certain amount of time. Complete the missing methods.

```
/*
 * RemoteCalculator.java
 * A class to hold the main method and launch the
    server and clients.
 */

//Import session
//TO BE COMPLETED

public class RemoteCalculator {
    public static void main (String[] args)
    {
        //TO BE COMPLETED
        //Hint: Runtime.getRuntime().exec() can be
           used to launch a command window where
           the server or the client can be run.
        System.out.println("MAIN: Finished
           simulation.");
        System.exit(0);
    }
}
```

8 Exemplary program output

The following is an exemplary valid output of the program.
Server side

```
Server at 127.0.0.1 is listening on port : 6868
Service thread 19: Request retrieved: * 2 3
Service thread 19: Product operation found.
Service thread 19: Service outcome returned; 6.0
Service thread 19: Finished service.
```

Client side; note the user command ∗23 was typed using the keyboard.

```
Client: Please type your operation in prefix notation e.g. + 4 3:
* 2 3
Client: Requesting calculator service for command '* 2 3'
Client: Service outcome: 6.0
Client: Finished.
```

Main side

```
MAIN: Launching Calculator server. Server will be alive for about 1 minute.
MAIN: Launching clients.
MAIN: Server timed out. Killing server.
MAIN: Finished simulation.

Process finished with exit code 0
```