New Relic - Technical test

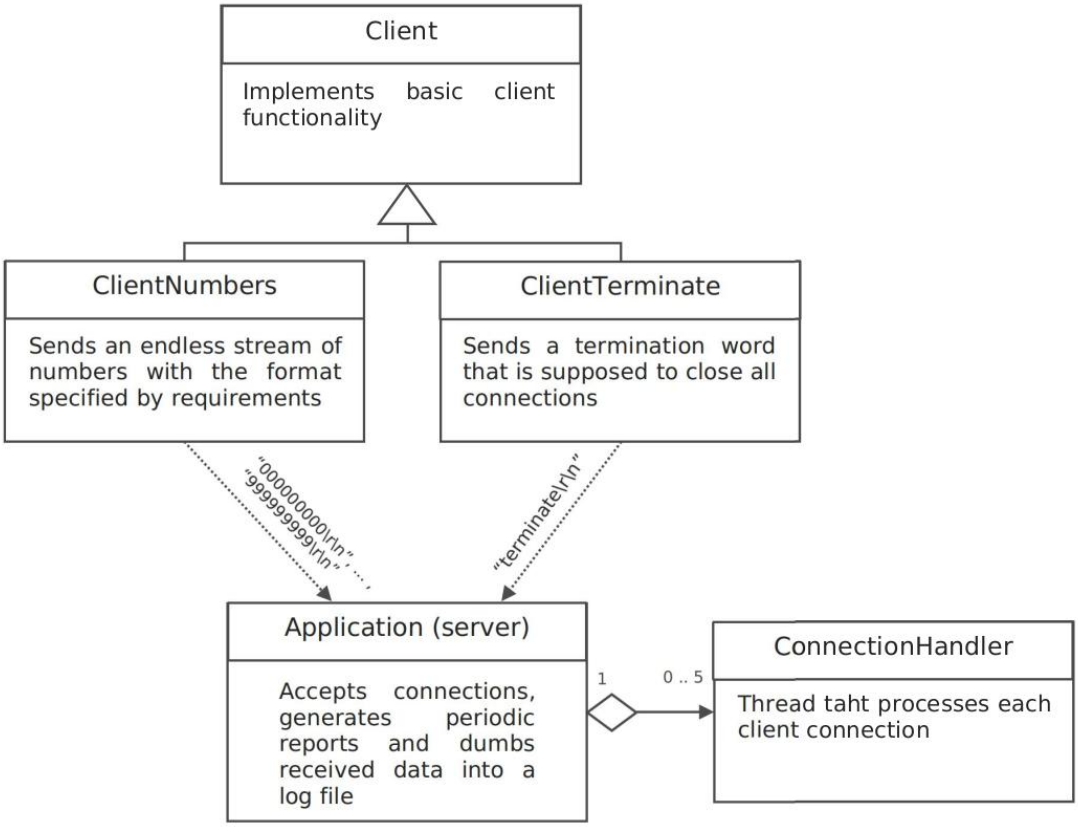
Basic functionality of the program

Application opens a socket and restricts input to at most 5 concurrent clients. Clients will connect to the Application and write any number of 9 digit numbers, and then close the connection. The Application must write a de-duplicated list of these numbers to a log file in no particular order.

Requirements

1. The Application must accept input from at most 5 concurrent clients on TCP/IP port 4000.
2. Input lines presented to the Application via its socket must either be composed of exactly nine decimal digits (e.g.: 314159265 or 007007009) immediately followed by a server-native newline sequence; or a termination sequence as detailed below.
3. Numbers presented to the Application must include leading zeros as necessary to ensure they are each 9 decimal digits.
4. The log file, to be named "numbers.log", must be created anew and/or cleared when the Application starts.
5. Only numbers may be written to the log file. Each number must be followed by a server-native newline sequence.
6. No duplicate numbers may be written to the log file.
7. Any data that does not conform to a valid line of input should be discarded and the client connection terminated immediately and without comment.
8. Every 10 seconds, the Application must print a report to standard output: o The difference since the last report of the count of new unique numbers that have been received. o The difference since the last report of the count of new duplicate numbers that have been received. o The total number of unique numbers received for this run of the Application. o Example text: Received 50 unique numbers, 2 duplicates. Unique total: 567231
9. If any connected client writes a single line with only the word "terminate" followed by a server-native newline sequence, the Application must disconnect all clients and perform a clean shut down as quickly as possible.

Project structure



This Java program is made of 3 parts:

1. Application: It plays the role of server or back-end service of the system.
2. ClientNumbers: Client that connects to Application.
3. ClientTerminate: Client that connects to Application.

Client is an abstract class with basic client functionality. ClientNumbers and ClientTerminate extend such class, implementing the connect() abstract method and adding extra specific functionality.

Application class is the server side which manages all client connections through sockets. Every time a connection is established, the server delegates the task of handling such connection to one of the five threads (if available) present in the thread pool. That way, it focuses on receiving connections only. It is also responsible for generating 10 seconds periodic reports and to dump all unique numbers received from clients into a log file.

Program execution flow

When Application is run, it opens a server socket in a localhost and port 4000. It waits for 20 seconds for someone to connect (default time-out). It also starts a timer and creates a blank log file named numbers.log.

User can then run ClientNumbers, which will create an infinite number of clients sequentially that will try to connect to the server by sending nine decimal digit numbers followed by a server native new line (\r\n for Windows, \n for Linux and \r for MAC). Each client sends one number and disconnects.

On the server side, Application will manage such connections creating a thread for each one. Each thread (ConnectionHandler class) has the task of parsing input data, determining if it is valid and if so, storing it in a ConcurrentHashMap.

The aforementioned map registers all input numbers as keys, which must be unique. If a duplicated number already present in the map is received, the value of such key is incremented by one.

Every 10 seconds, the timer created by Application is responsible of generating and printing a report that gives information about how many unique and duplicate numbers have been received since the last 10 seconds period and the total amount of unique numbers received during the whole execution of the system. If the value of a certain key is bigger than one, that means that a duplicate has been received for that key in the last 10 seconds period. ConcurrentHashMap values are reset after the report is generated. However, keys are maintained in order to be able to write all de-duplicated numbers received into the log file.

If the user wants to stop the program, he must run ClientTerminate, which will send a terminate command. Such command shuts down gracefully the server Application as soon as possible.

Right before server Application is closed, it writes all unique numbers stored in ConcurrentHashMap into the log file created at the beginning.

ClientNumbers detects when Application server is closed and also stops executing.

Assumptions

1. Application server only writes numbers into the generated log file. However, such file includes other data such as date and time automatically included by Logger. These additional information is kept.
2. A report must be generated every 10 seconds. There is no delay on the timer. Therefore, an initial report is generated at the beginning of the application.
3. New line characters are sent by clients. Server reads input stream data using BufferedReader's built-in nextLine(). Such method reads the whole input stream until a new line character is found automatically.
4. Static method System.lineSeparator() has been used to add new line characters in order to make the program platform independent.
5. Nine digit numbers generated by ClientNumbers are generated randomly.
6. Log file is created at the beginning of each Application run and stored within the root project directory.

Build and run instructions

Linux / MAC

From terminal, using Git

Type the following commands from the command line in order:

1. *foo@bar:~$ cd ~*
2. *foo@bar:~$ git clone https://github.com/michael-good/multithreaded-client-server-application.git*
3. *foo@bar:~$ cd multithreaded-client-server-application*
4. *foo@bar:~$ mkdir out*
5. *foo@bar:~$ javac -cp lib/commons-lang3-3.9.jar -d out src/miguel/angel/bueno/sanchez/main/*.java*

Downloading .zip file

If you have downloaded a ZIP file in a folder, skip steps 1, 2 and 3. Open up a terminal in the directory where the files are located and execute the instruction specified in steps 4 and 5.

After completing steps 1 to 5, all project files are compiled into out/ directory. In order to properly execute the program, one must open three different terminals up and navigate to ~/multithreaded-client-server-application. Following commands will execute the server Application and clients. Use one of them for each terminal in the corresponding order:

foo@bar:~$ java -cp lib/commons-lang3-3.9.jar:out miguel.angel.bueno.sanchez.main.Application
foo@bar:~$ java -cp lib/commons-lang3-3.9.jar:out miguel.angel.bueno.sanchez.main.ClientNumbers
foo@bar:~$ java -cp lib/commons-lang3-3.9.jar:out miguel.angel.bueno.sanchez.main.ClientTerminate

NOTE: If you downloaded the project as a ZIP file, the location ~/multithreaded-client-server-application may vary. Move to the correct path where files are stored before proceeding with the running instructions.

Dependencies and other considerations

This project has been written using Java 8 OpenSDK (Java 1.8 / OpenJDK8). It makes use of some methods from org.apache.commons.commons-lang3-3.9 from Apache Commons Lang library.

Make sure that such dependencies are specified to the the Java compiler (javac) when building the project. Just follow instructions above for a correct compilation.