**Building Scalable Distributed Systems – CS6650**

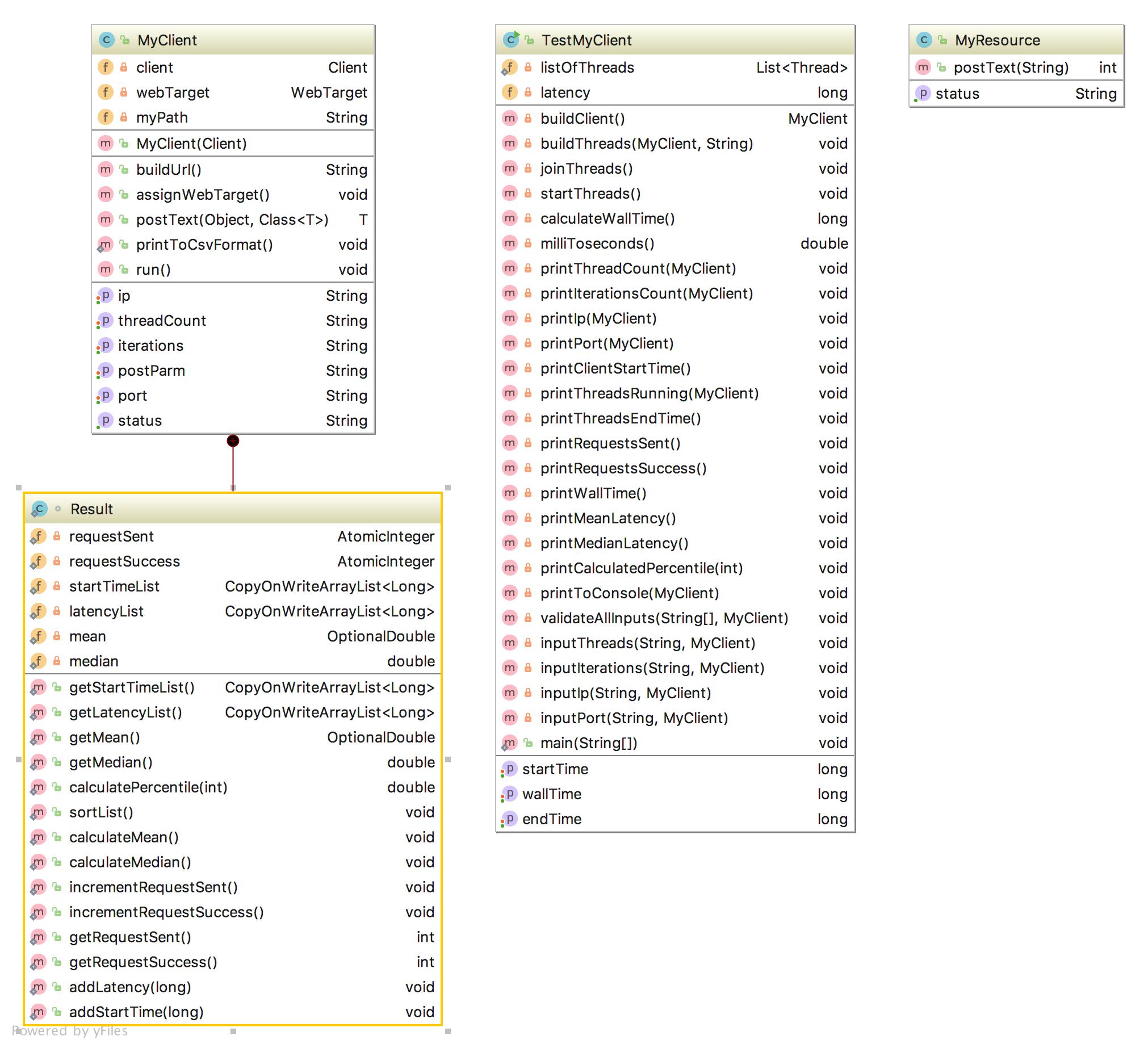
**Assignment – 1**

<https://github.com/kinshukjuneja/BSDS-CS6650>

**Name: Kinshuk Juneja**

1. A 1 page overview of your design (a simple block diagram would suffice). The aim is to quickly summarize your design so emphasize important components and abstractions.  **(1** **point)**

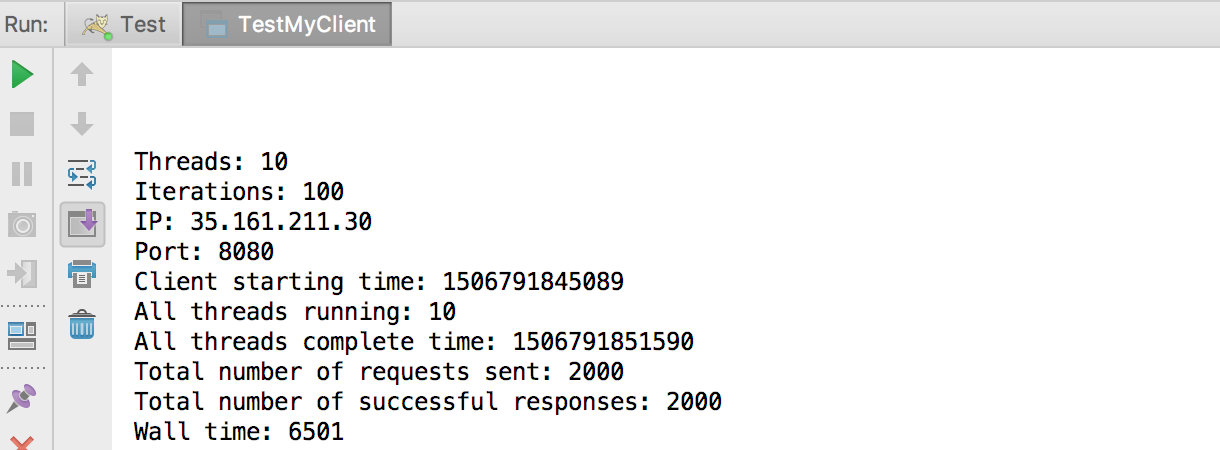
* I have 4 classes in total where Result is an inner class of MyClient which in turn implements Runnable.
* On execution of TestMyClient containing main() method it first builds a client and then creates a list of all threads taken from input or default to 10 and joins those thread so that code following main() is not executed. On starting all the threads, Client’s overridden run() method is executed where requests for GET/POST till number of iterations(default: 100) is run for each GET and POST and returns the value from MyResource class.
* The inner class Result is helpful to calculate number of requests sent, number of successful requests, wall time, mean, median, percentile calculation, so that TestMyClient class can display those in output.
* To plot graphs, I also included list of start time for each request in GET/POST along with latency which is calculated as (end time of request – start time of request).
* Please refer to the block diagram below for an overview:



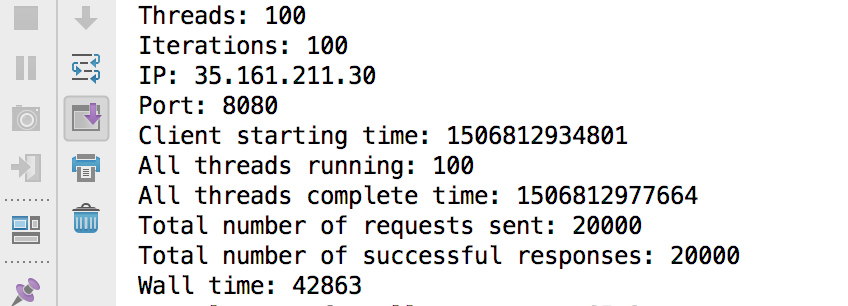
* 2)  URL to your git repo. (3 points)
* <https://github.com/kinshukjuneja/BSDS-CS6650>

3)  Two screenshots for step 4 showing correct execution and completion of the two  specified tests **(3** **points** **for** **each)**

* **GET/POST – 10/100:**

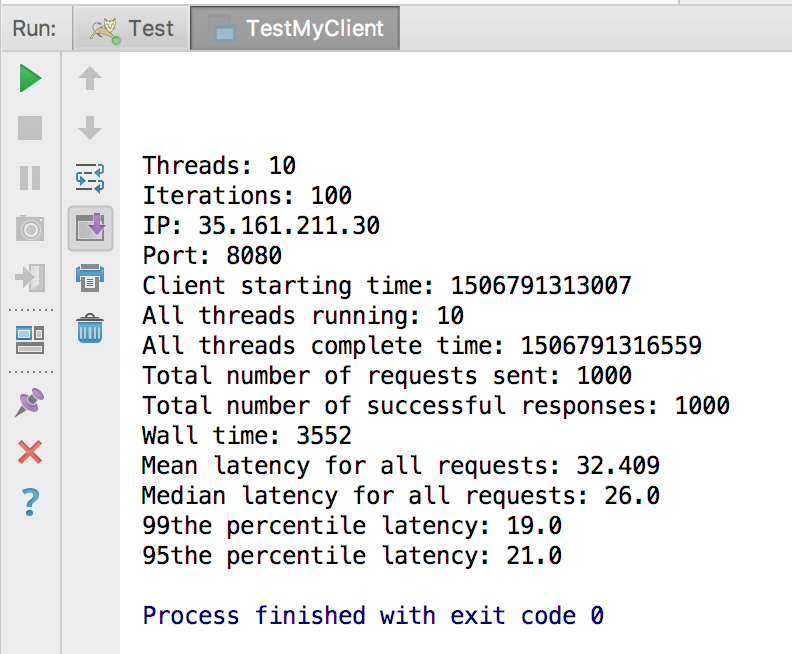
****

* **GET/POST – 100/100**

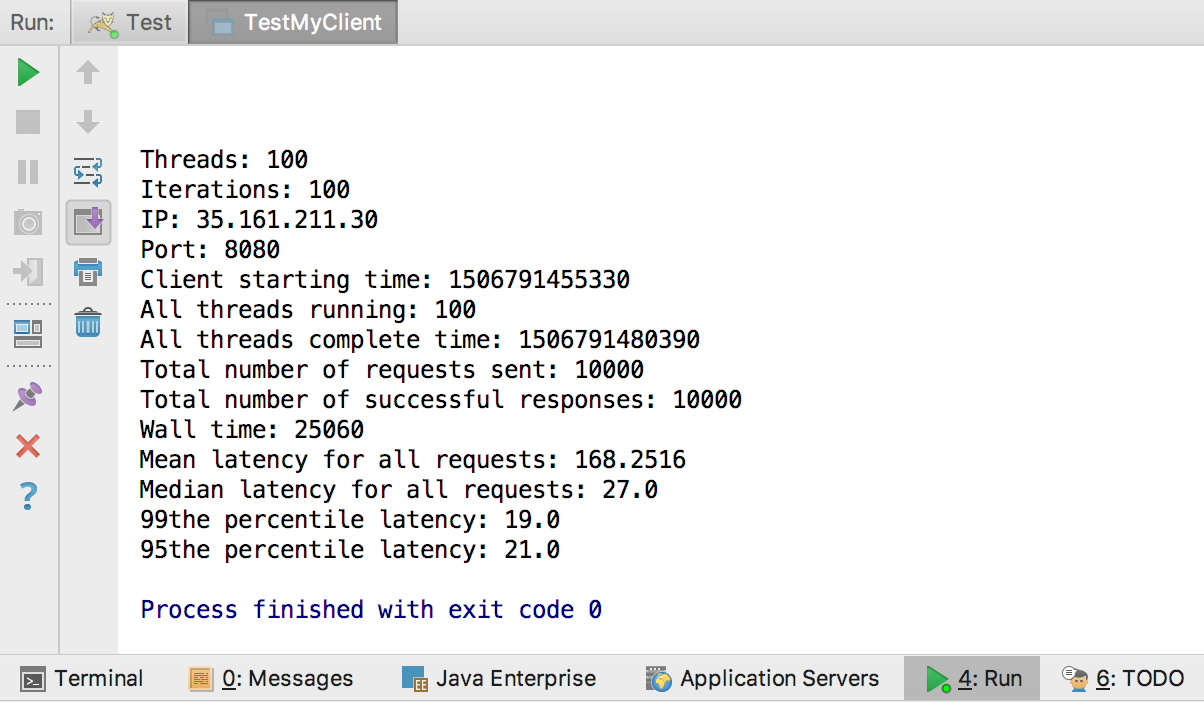
****

4)  Two screenshots for step 5 showing correct execution and completion of the two  specified tests. If you use an additional tool like a spreadsheet, show the results in this in  addition to the two screenshots showing the test running **(** **5** **points** **for** **each)**

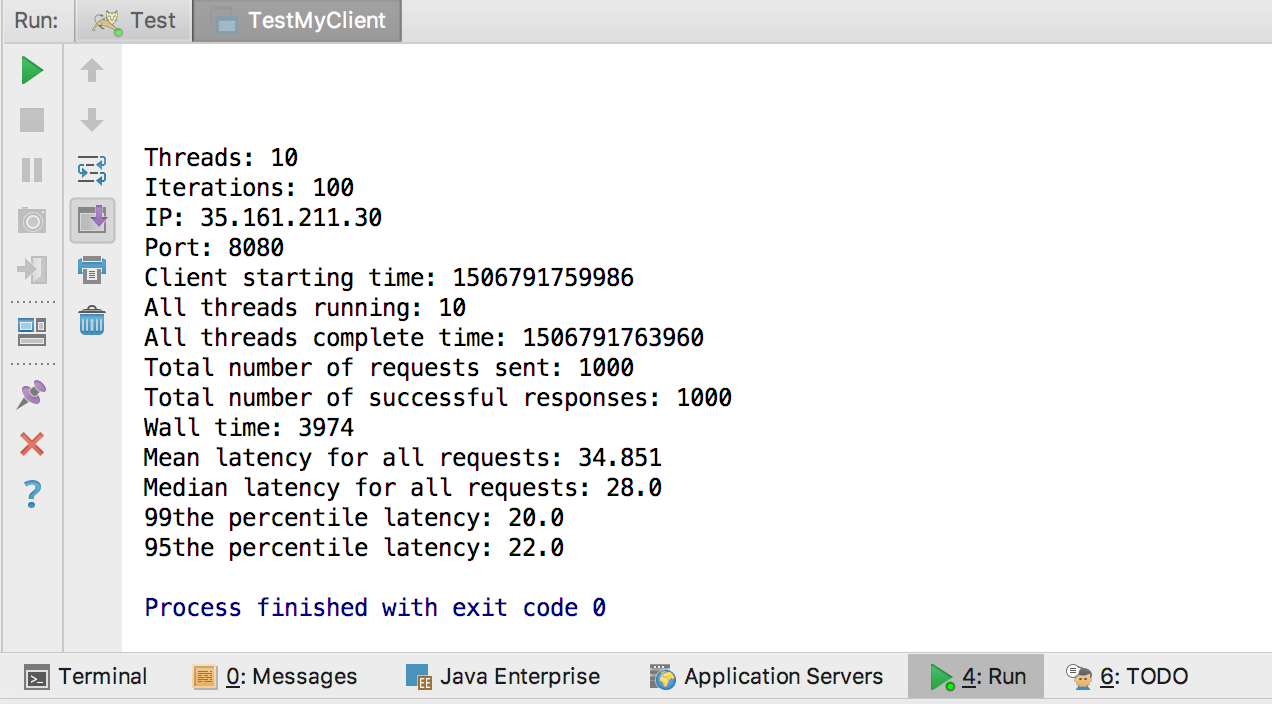
* **GET – 10/100:**

****

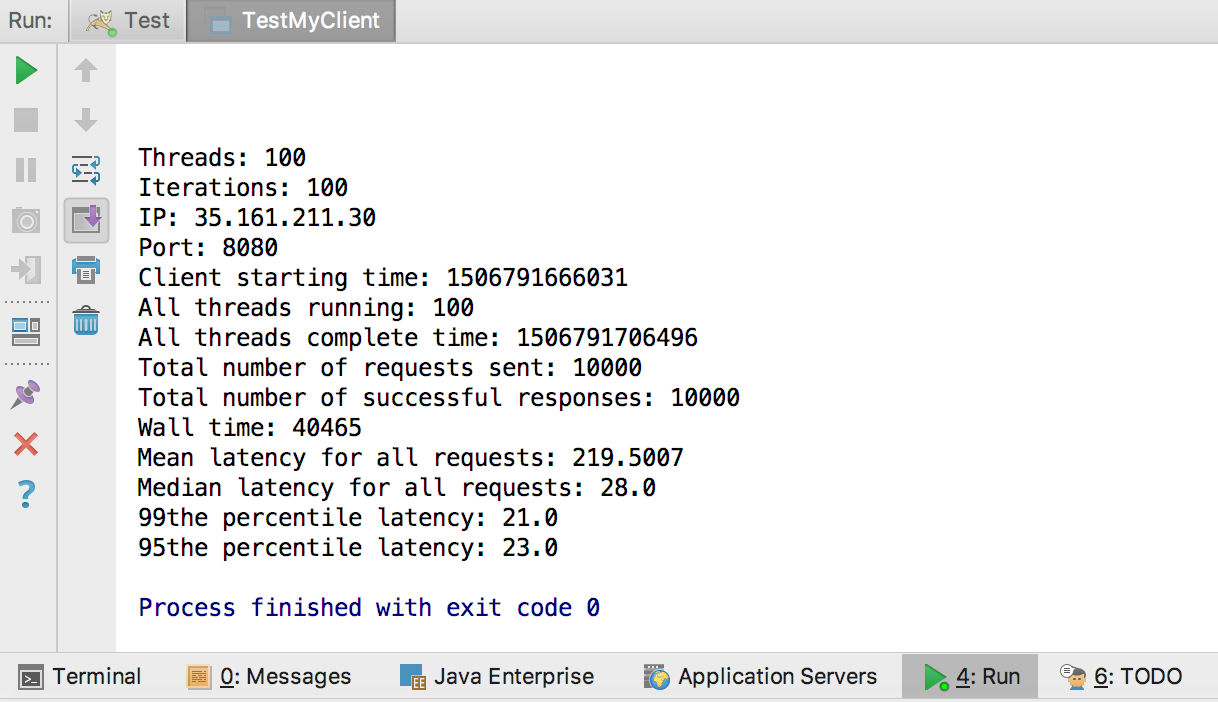
* **GET – 100/100**

****

* **POST – 10/100**

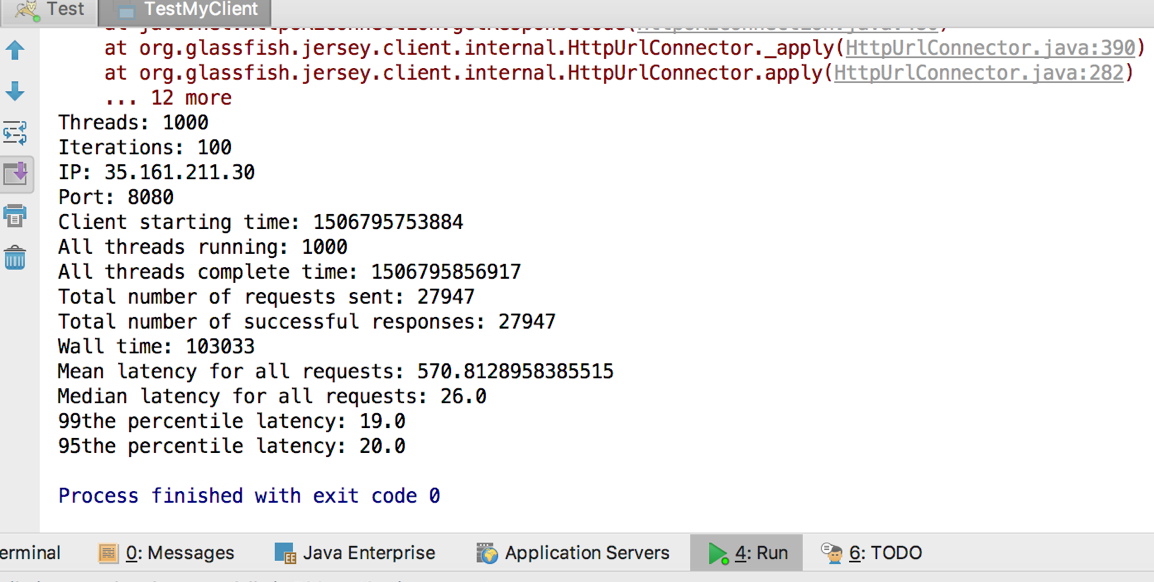
****

* **POST – 100/100**

****

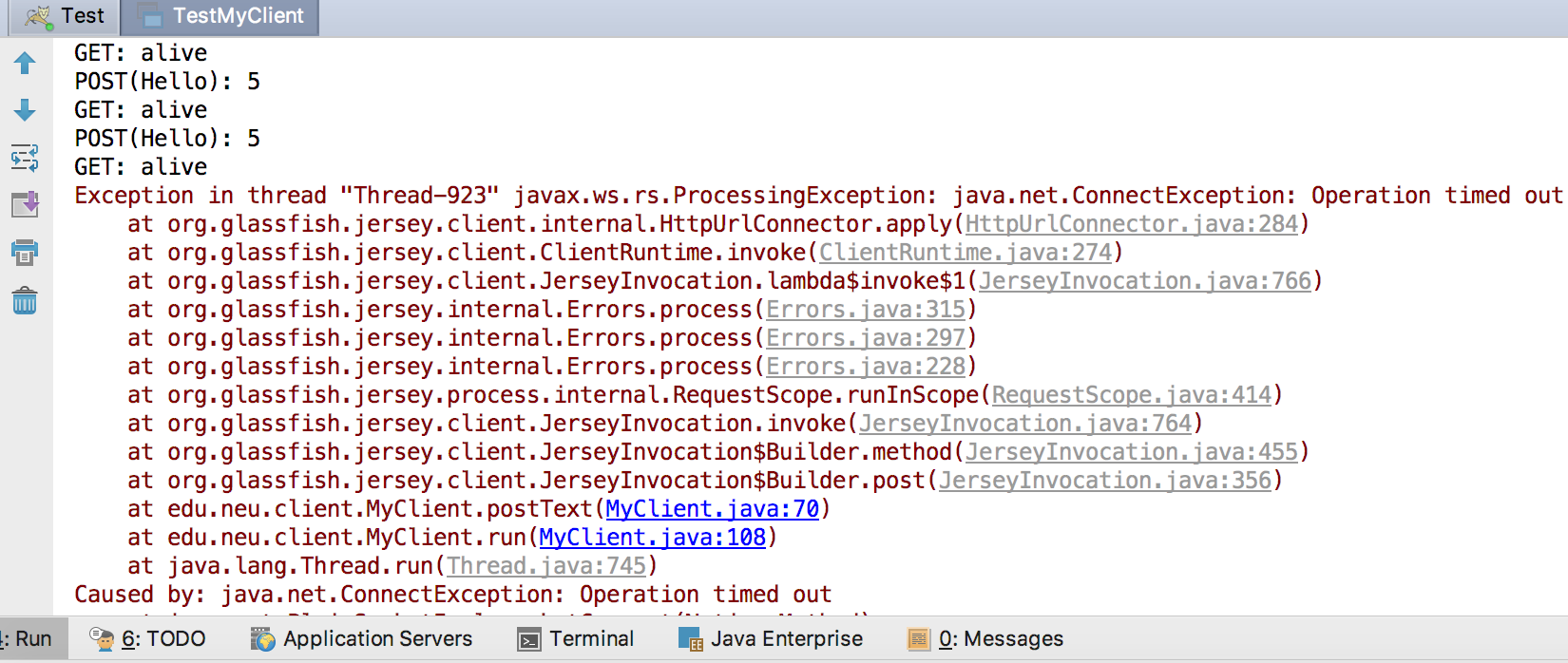
5)  Step 6 Stress testing: Submit a short (1 page?) report describing what you did to explore the tolerances of your application, what broke it, and how. (**1 point**)

* **Check Tolerance**: To explore the tolerance I gradually increased my number of threads for 100 iterations(requests) and vice-verse. The following result was obtained:
* **What broke it - Get – 1000/100:** On increasing the number of threads to 1000 for 100 iterations(requests) just for GET, the total number of requests should be 100000, but only 27947 requests were sent and successfully completely.

****

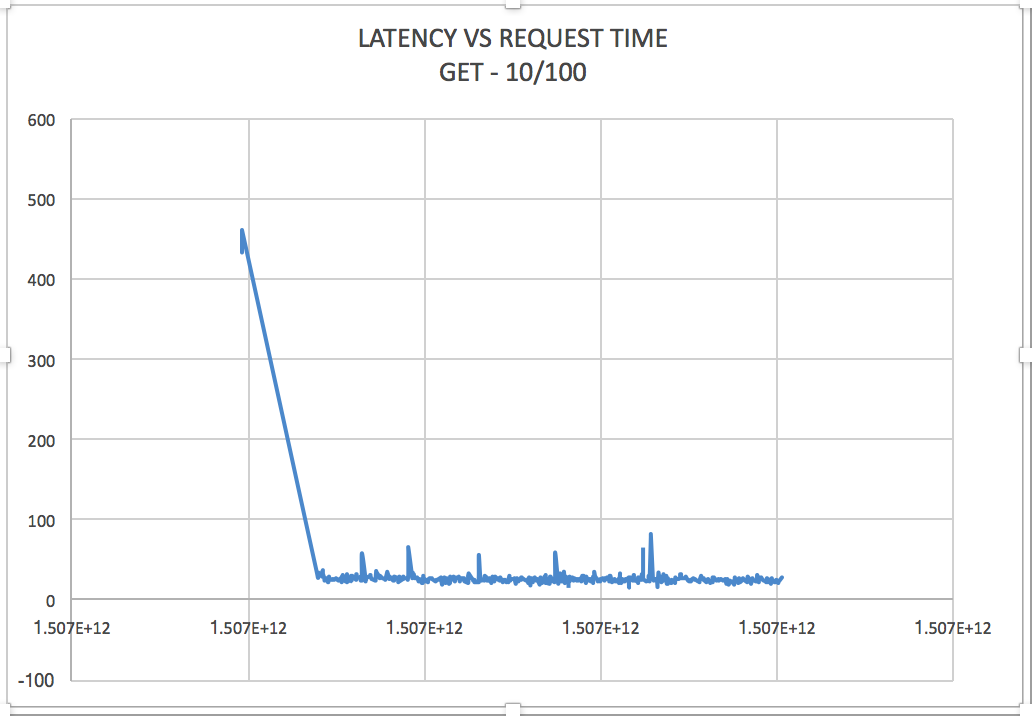
* **How - GET/POST – 1000/100:** On testing for both GET and POST requests simultaneously for 1000 threads and 100 iterations (requests), a total of 200000 requests should have been made, but that did not happen because an error was **first** thrown on making just 17218 requests, after which requests continued but errors appeared in between.
* This was eventually, because in my program **Operation timed out.**

i.e. Caused by: java.net.ConnectException: Operation timed out

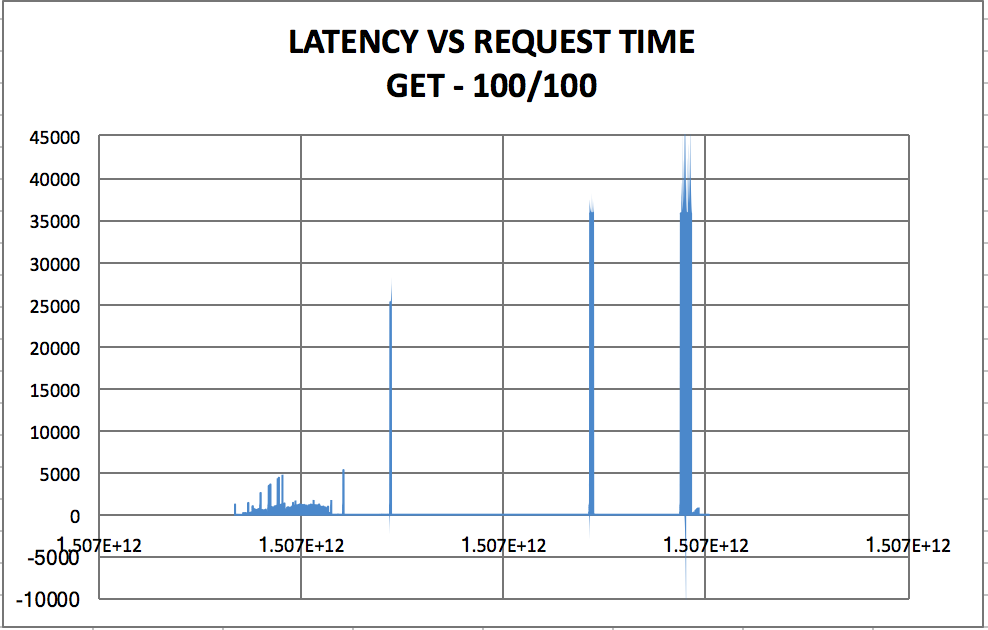
****

6)  Step 6 charting: Submit a 1 page report detailing your test run, method of calculation, and chart showing latencies. (1 point)

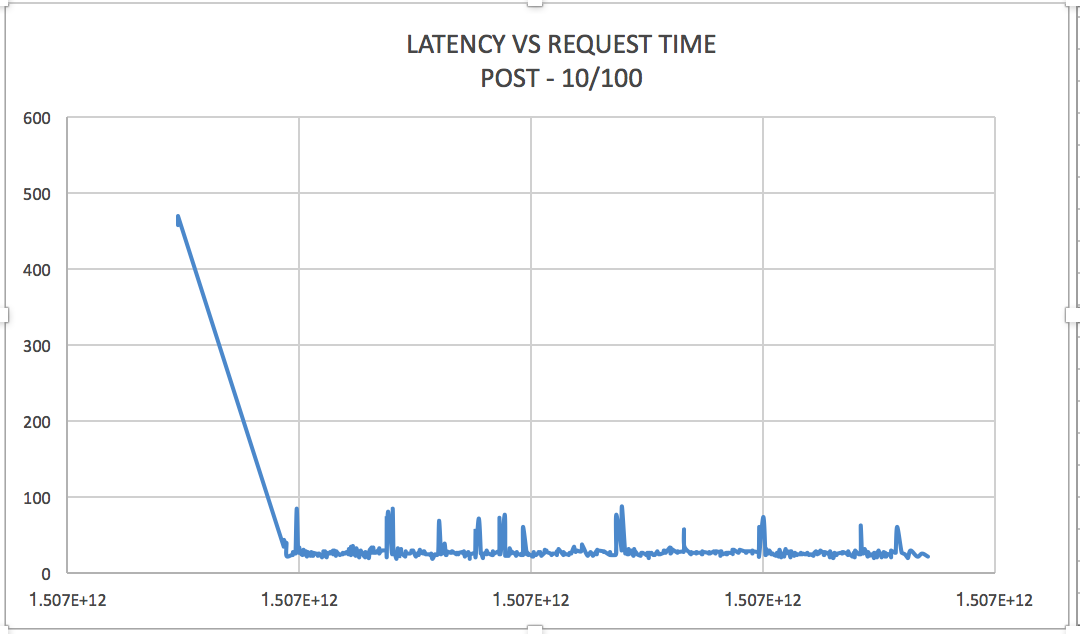
* To plot graphs, I included an ArrayList of start time for each request in GET/POST along with latency which is calculated as (end time of request – start time of request).
* As soon as the threads override Client’s run() method, I calculate the start time in milliseconds(System.currentTimeMillis()) and assign it to a variable startTime for each GET/POST tested individually and add this to my list of start times.
* After this, when a GET/POST request is made and completed, I generate an endTime for GET/POST respectively and subtract endTime – startTime to get the latency which is added to an ArrayList of latency.
* For all the graphs listed below: X-axis is my start time of request made and Y-axis is my Latency.
* **GET – 10/100:**

****

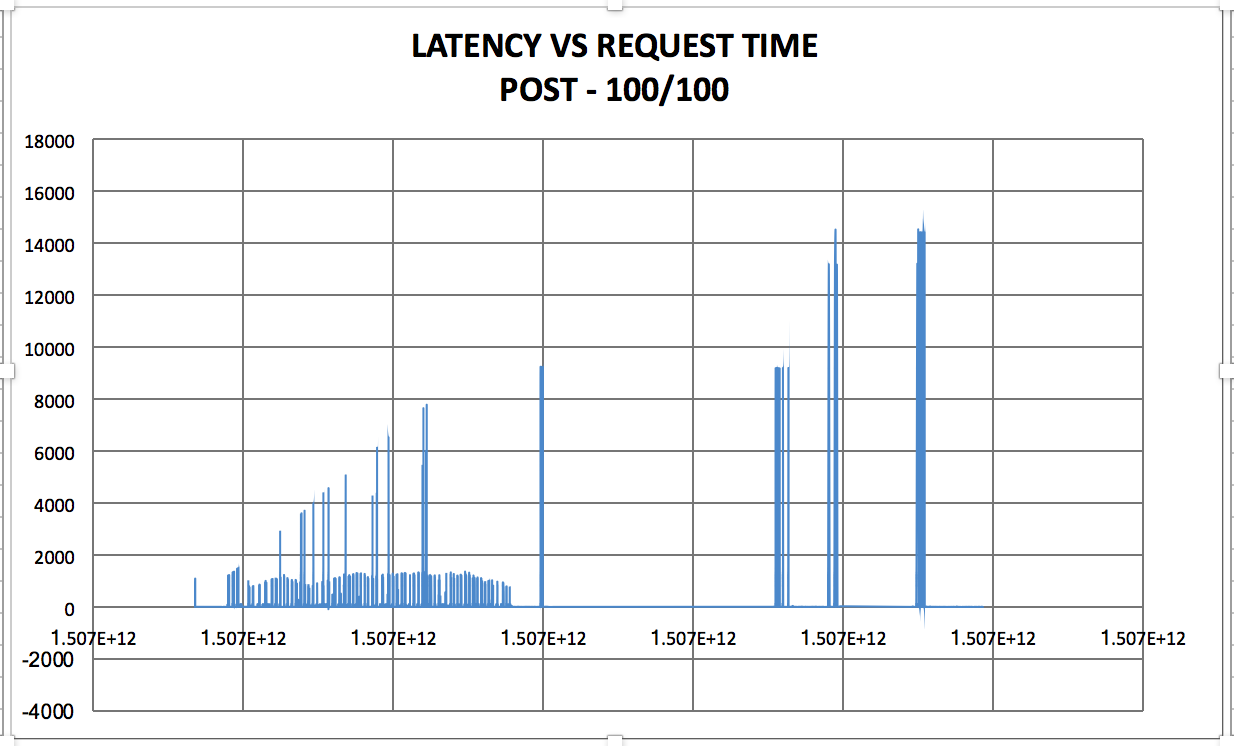
* **GET – 100/100**

****

* **POST – 10/100**

****

* **POST – 100/100**

****