**Design**

In order to calculate the average inflow rate λ and the average outflow rate µ, a class name *“Count.java”* was created. This count class keep track of the inflow and outflow counts incrementing an atomic long value in this multithread environment. An instance of Count is created for the thread pool of clients (used the same instance) and other one for the thread pool of workers. Therefore, each time the clients send request to the server the request count is incremented, same thing happen with the workers each time they successfully served a request from the pool the count is incremented.

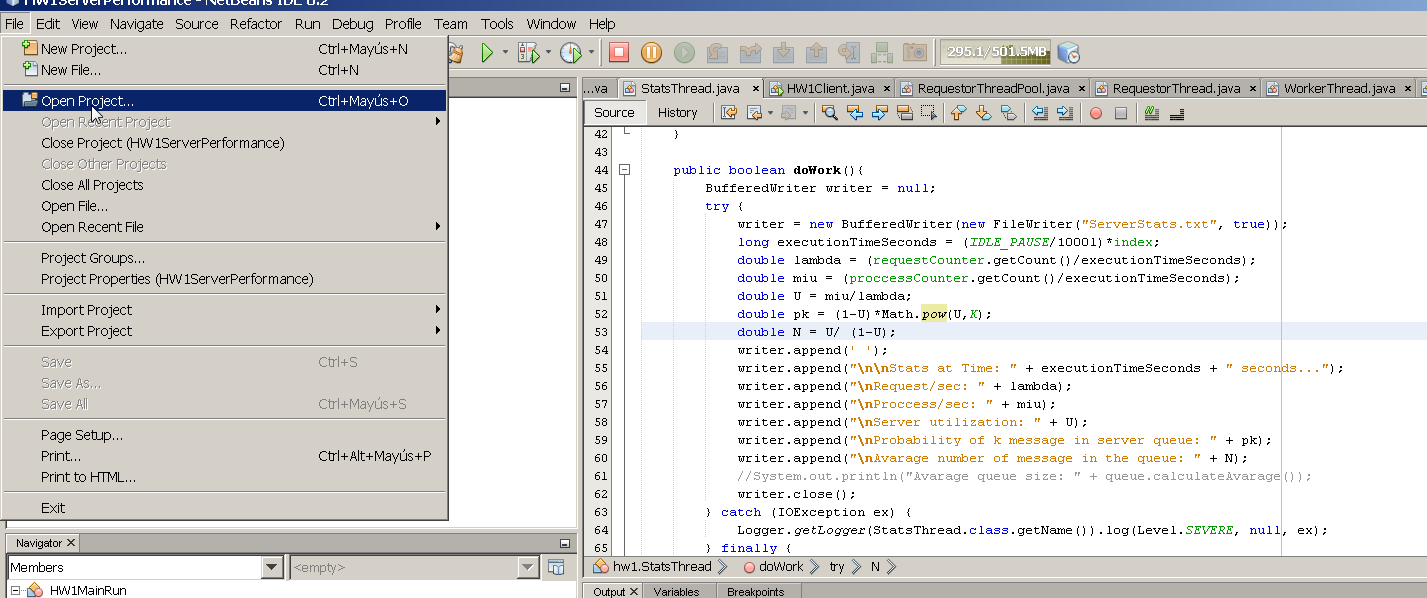
A new thread class named *“StatsThread.java”* is in charge of calculating the statics every 10 seconds and print them to a text file "ServerStats.txt":

* average inflow rate **λ**
* average outflow rate **µ**
* Server utilization **U**
* Probability of k message in server queue ***pk***
* Average number of message in the queue **N**

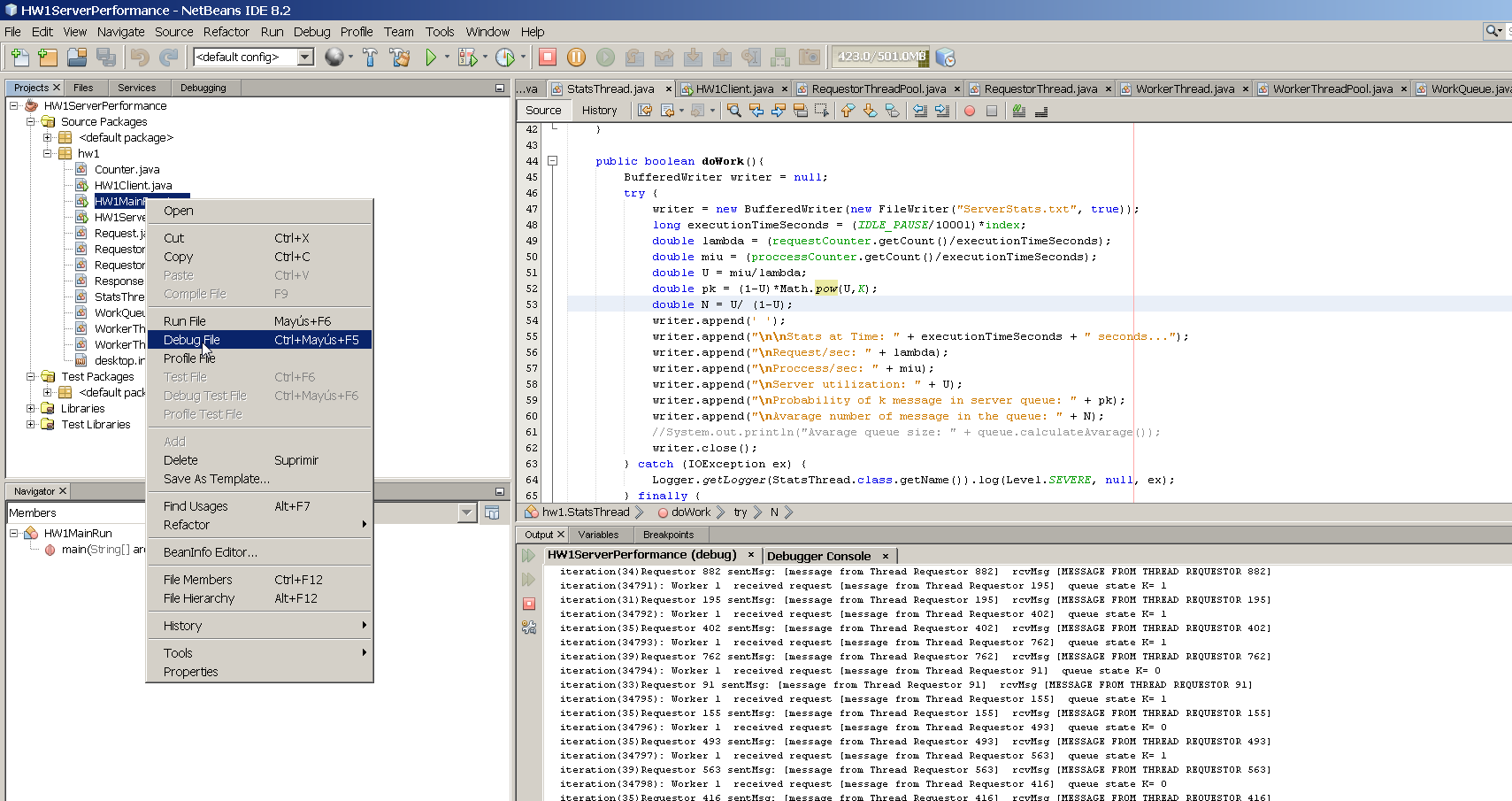
The class “*HW1MainRun.java”* run the test for client, server and stats at the same time.

**Execution**

1. Unzip file and open project in NetBeans

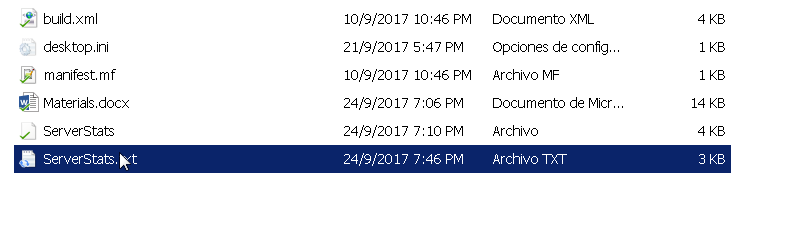


1. Select class “*HW1MainRun.java”* and right click, then select Run File



**Reporting of results**

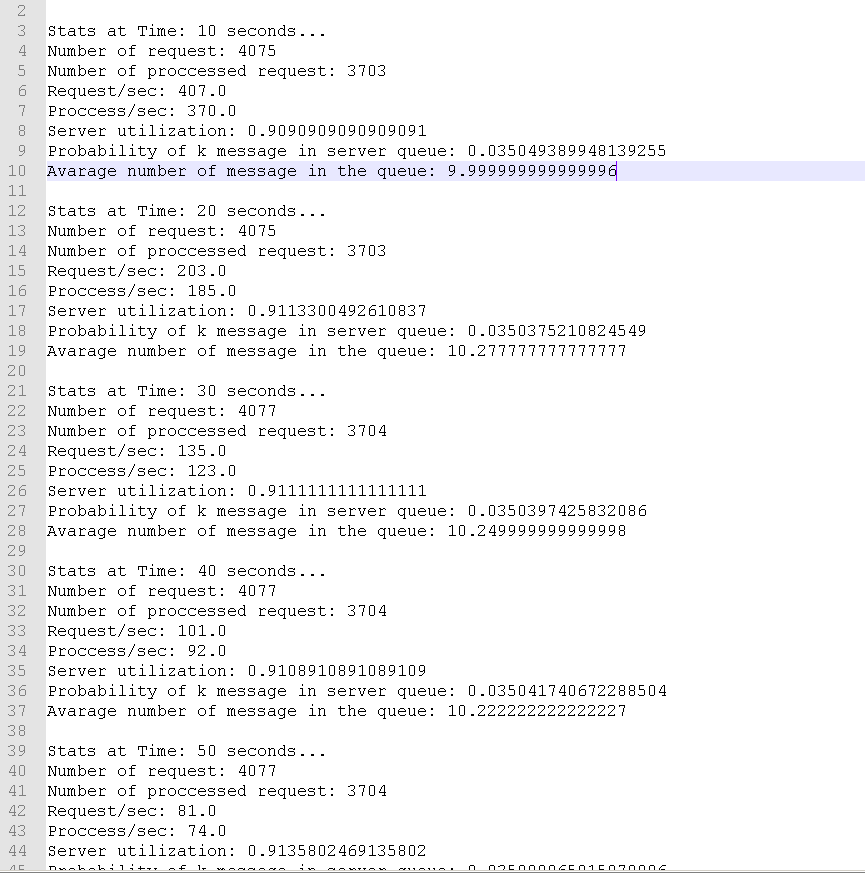
The results for the run can be find in a text file in the project directory. The text file contains the calculation of the variables every 10 seconds.



Four test were performed using different values for number of workers, resulting in this output:

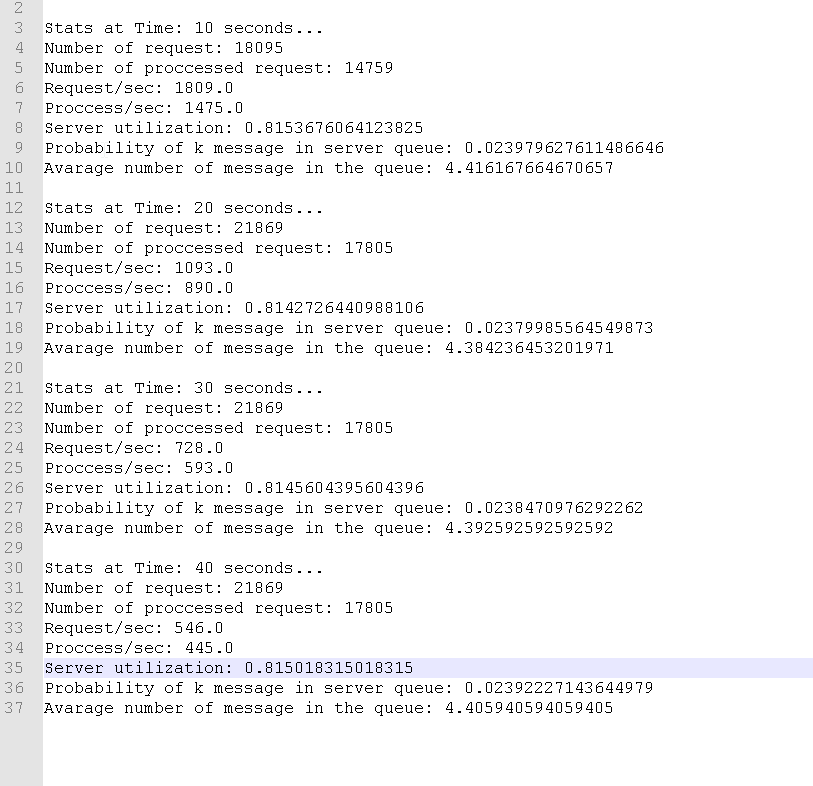
***NUM\_REQUESTERS= 10***

***NUM\_WORKERS= 1***



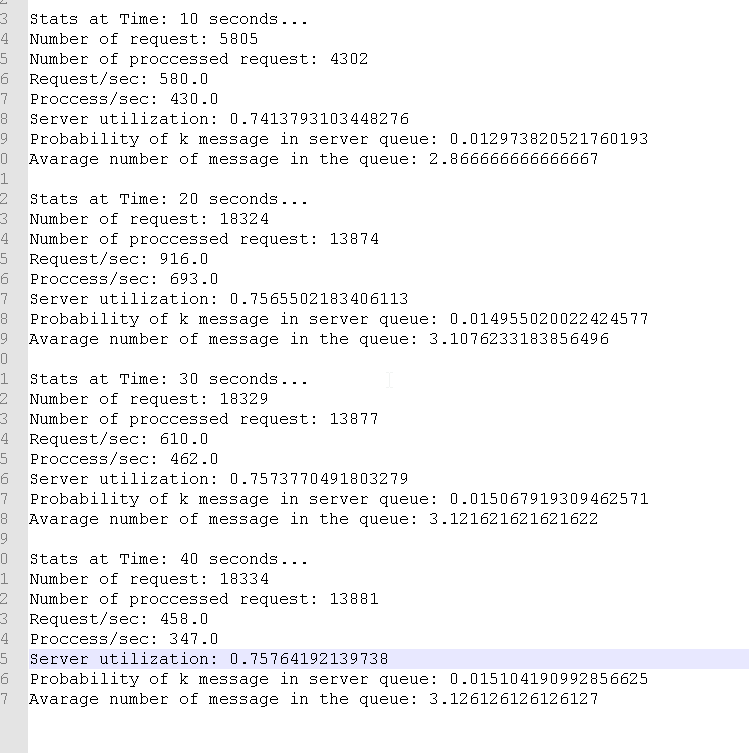
***NUM\_REQUESTERS= 10***

***NUM\_WORKERS= 3***



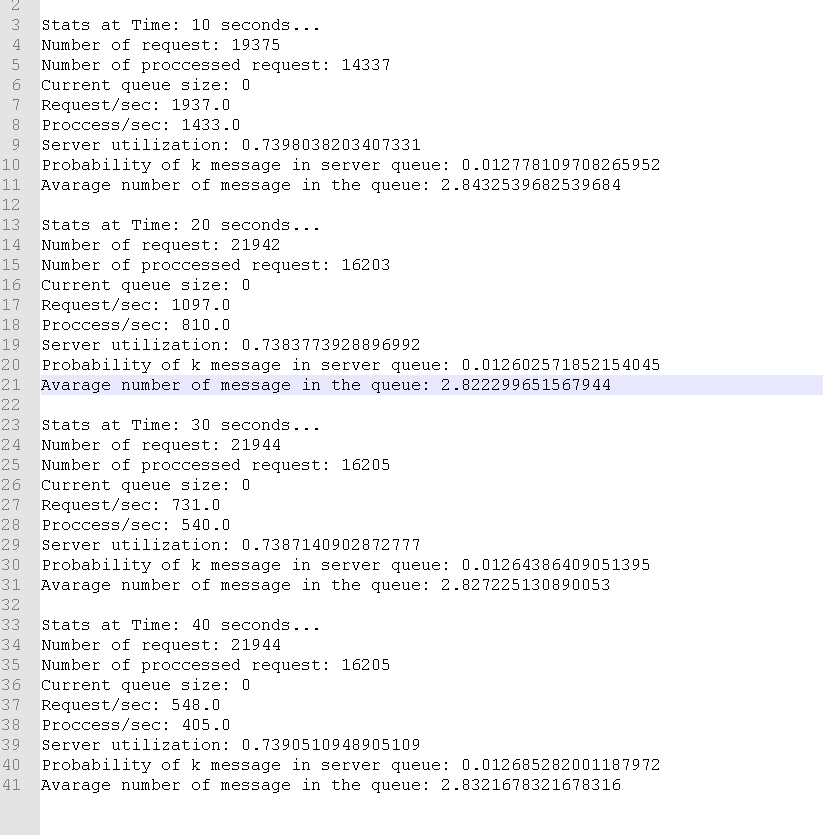
***NUM\_REQUESTERS= 10***

***NUM\_WORKERS= 6***



***NUM\_REQUESTERS= 10***

***NUM\_WORKERS= 9***



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***NUM\_REQUESTERS= 10***  ***NUM\_WORKERS= 1*** | ***NUM\_REQUESTERS= 10***  ***NUM\_WORKERS= 3*** | ***NUM\_REQUESTERS= 10***  ***NUM\_WORKERS= 6*** | ***NUM\_REQUESTERS= 10***  ***NUM\_WORKERS= 9*** |
| Avarage number of message in the queue (N) | 9.99 | 4.41 | 2.86 | 2.84 |

**Observations**

Running several times the test with different client request pool size and workers pool size, when request are increase maintaining less worker, workers can’t manage to process all incoming request and the work queue increments its size what can generate an overflow of memory and lose of information. When server increase the number of workers maintaining the number of request, the work queue starts to decrease and the average number of message in the queue as well. The other case is when there is a balanced level of incoming request and workers dispatching those requests, so the work queue remains in a balanced average size.