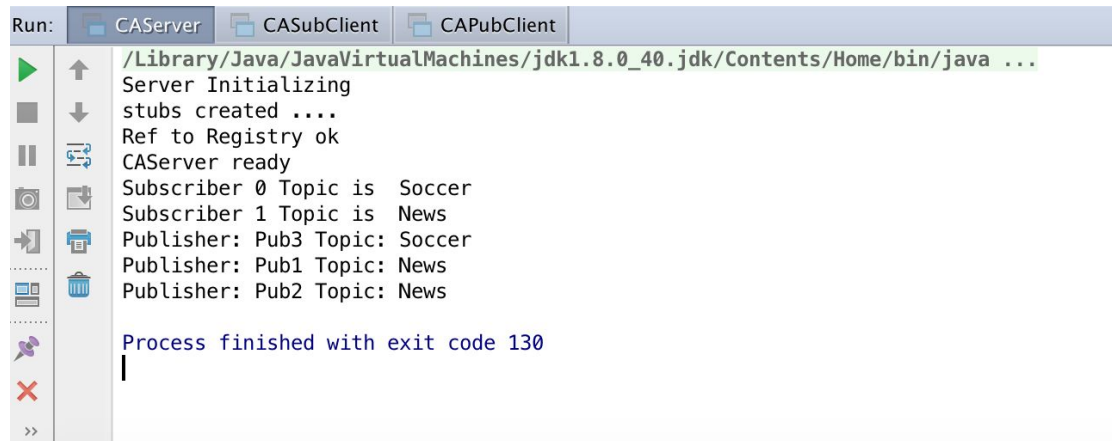


Step 1 – Functional correctness Part 1

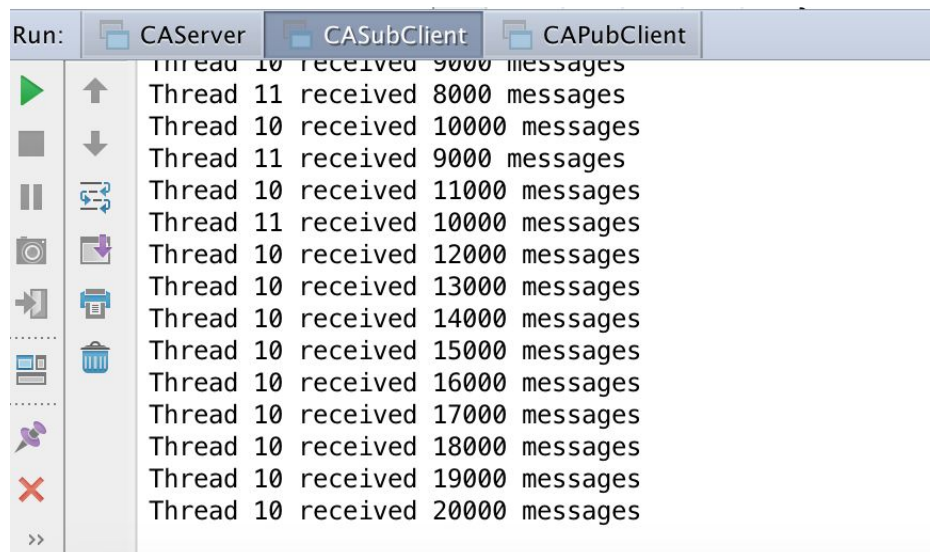
I created 3 content providers whose topics are News, News and Soccer, as well as 2 content subscribers who subscribe to topics News and Soccer. The running result is as below:

Content Aggregator:



```
Run: CAServer CASubClient CAPubClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin/java ...
Server Initializing
stubs created ....
Ref to Registry ok
CAServer ready
Subscriber 0 Topic is Soccer
Subscriber 1 Topic is News
Publisher: Pub3 Topic: Soccer
Publisher: Pub1 Topic: News
Publisher: Pub2 Topic: News
Process finished with exit code 130
```

Content Subscriber:



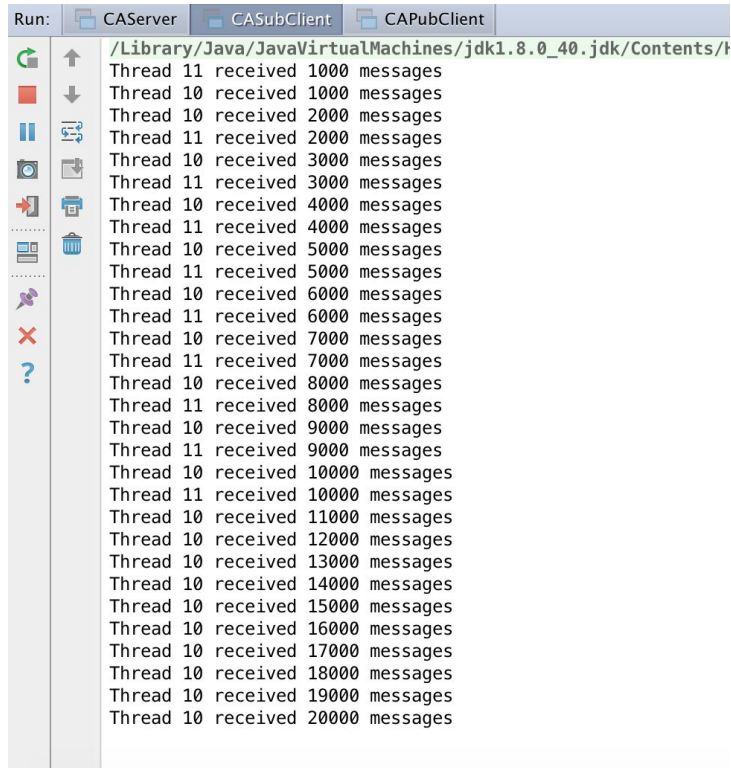
```
Run: CAServer CASubClient CAPubClient
Thread 10 received 9000 messages
Thread 11 received 8000 messages
Thread 10 received 10000 messages
Thread 11 received 9000 messages
Thread 10 received 11000 messages
Thread 11 received 10000 messages
Thread 10 received 12000 messages
Thread 10 received 13000 messages
Thread 10 received 14000 messages
Thread 10 received 15000 messages
Thread 10 received 16000 messages
Thread 10 received 17000 messages
Thread 10 received 18000 messages
Thread 10 received 19000 messages
Thread 10 received 20000 messages
```

Thread 10 is subscriber 1 (subscribe to News), and thread 11 is subscriber 0 (to Soccer). In the end thread 10 received 20000 messages and thread 11 received 10000 messages, which is correct.

Step 2 – Functional correctness part 2

For this time there are 4 content providers, 2 are News, 1 is Soccer and 1 is Junk. The running result is as below:

The output of Content Subscriber is the same:



```
Run: CAsServer CAsSubClient CAPubClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/t
Thread 11 received 1000 messages
Thread 10 received 1000 messages
Thread 10 received 2000 messages
Thread 11 received 2000 messages
Thread 10 received 3000 messages
Thread 11 received 3000 messages
Thread 10 received 4000 messages
Thread 11 received 4000 messages
Thread 10 received 5000 messages
Thread 11 received 5000 messages
Thread 10 received 6000 messages
Thread 11 received 6000 messages
Thread 10 received 7000 messages
Thread 11 received 7000 messages
Thread 10 received 8000 messages
Thread 11 received 8000 messages
Thread 10 received 9000 messages
Thread 11 received 9000 messages
Thread 10 received 10000 messages
Thread 11 received 10000 messages
Thread 10 received 11000 messages
Thread 10 received 12000 messages
Thread 10 received 13000 messages
Thread 10 received 14000 messages
Thread 10 received 15000 messages
Thread 10 received 16000 messages
Thread 10 received 17000 messages
Thread 10 received 18000 messages
Thread 10 received 19000 messages
Thread 10 received 20000 messages
```

For the Content Aggregator, I print the number of messages for each topic every 5 seconds. From the output we can see messages on News, Soccer and Junk first grew, and then stayed at 20000, 10000 and 10000 for several seconds. After the first few messages expired, the number of messages began to decrease.

```
Subscriber 0 Topic is Soccer
Subscriber 1 Topic is News
Current seconds: 5
Publisher: Pub1 Topic: News
Publisher: Pub3 Topic: Soccer
Publisher: Pub2 Topic: News
Publisher: Pub4 Topic: Junk
Current seconds: 10
Topic: Soccer has 3103 messages.
Topic: News has 6263 messages.
Topic: Junk has 3017 messages.
Current seconds: 15
Topic: Soccer has 5614 messages.
Topic: News has 10945 messages.
Topic: Junk has 5889 messages.
Current seconds: 20
Topic: Soccer has 6875 messages.
Topic: News has 13308 messages.
Topic: Junk has 7453 messages.
Current seconds: 25
Topic: Soccer has 7844 messages.
Topic: News has 14878 messages.
Topic: Junk has 8512 messages.
Current seconds: 30
Topic: Soccer has 8677 messages.
Topic: News has 16197 messages.
Topic: Junk has 9405 messages.
Current seconds: 35
Topic: Soccer has 9421 messages.
Topic: News has 17458 messages.
Topic: Junk has 10000 messages.
Current seconds: 40
Topic: Soccer has 10000 messages.
```

Topic: Soccer has 10000 messages.
Topic: News has 19990 messages.
Topic: Junk has 10000 messages.
Current seconds: 50
Topic: Soccer has 10000 messages.
Topic: News has 20000 messages.
Topic: Junk has 10000 messages.
Current seconds: 55
Topic: Soccer has 10000 messages.
Topic: News has 20000 messages.
Topic: Junk has 10000 messages.
Current seconds: 60
Topic: Soccer has 6827 messages.
Topic: News has 13546 messages.
Topic: Junk has 6878 messages.
Current seconds: 65
Topic: Soccer has 4375 messages.
Topic: News has 9035 messages.
Topic: Junk has 4099 messages.
Current seconds: 70
Topic: Soccer has 3118 messages.
Topic: News has 6678 messages.
Topic: Junk has 2539 messages.
Current seconds: 75
Topic: Soccer has 2150 messages.
Topic: News has 5110 messages.
Topic: Junk has 1482 messages.
Current seconds: 80
Topic: Soccer has 1318 messages.
Topic: News has 3793 messages.
Topic: Junk has 590 messages.
Current seconds: 85
Topic: Soccer has 574 messages.

Current seconds: 75
Topic: Soccer has 2150 messages.
Topic: News has 5110 messages.
Topic: Junk has 1482 messages.
Current seconds: 80
Topic: Soccer has 1318 messages.
Topic: News has 3793 messages.
Topic: Junk has 590 messages.
Current seconds: 85
Topic: Soccer has 574 messages.
Topic: News has 2532 messages.
Topic: Junk has 0 messages.
Current seconds: 90
Topic: Soccer has 0 messages.
Topic: News has 1247 messages.
Topic: Junk has 0 messages.
Current seconds: 95
Topic: Soccer has 0 messages.
Topic: News has 6 messages.
Topic: Junk has 0 messages.
Current seconds: 100
Topic: Soccer has 0 messages.
Topic: News has 0 messages.
Topic: Junk has 0 messages.
Current seconds: 105
Topic: Soccer has 0 messages.
Topic: News has 0 messages.
Topic: Junk has 0 messages.
Current seconds: 110
Topic: Soccer has 0 messages.
Topic: News has 0 messages.
Topic: Junk has 0 messages.
Current seconds: 115

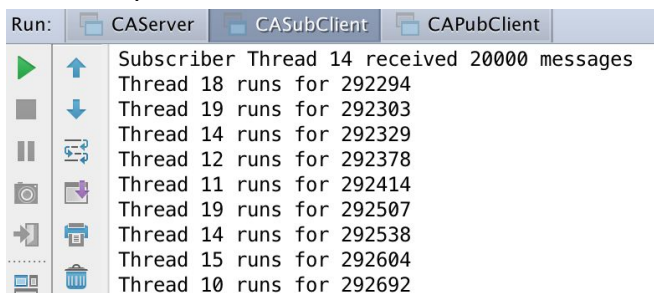
Step 3 – Stress Testing 1

The wall times for each subscriber and publisher are: (in milliseconds)

P.S subscribers are kicked off about 1 seconds prior to publishers

Publisher	Wall Time	Publisher	Wall Time	Subscriber	Wall Time
Pub1	282889	Pub11	288486	Sub1	295333
Pub2	283939	Pub12	288397	Sub2	292692
Pub3	285120	Pub13	288137	Sub3	292604
Pub4	285184	Pub14	288096	Sub4	292538
Pub5	285300	Pub15	287969	Sub5	292507
Pub6	285899	Pub16	287944	Sub6	292414
Pub7	287356	Pub17	287411	Sub7	292378
Pub8	287347	Pub18	287163	Sub8	292329
Pub9	285900	Pub19	285929	Sub9	292303
Pub10	285913	Pub20	285909	Sub10	292294

IntelliJ output:

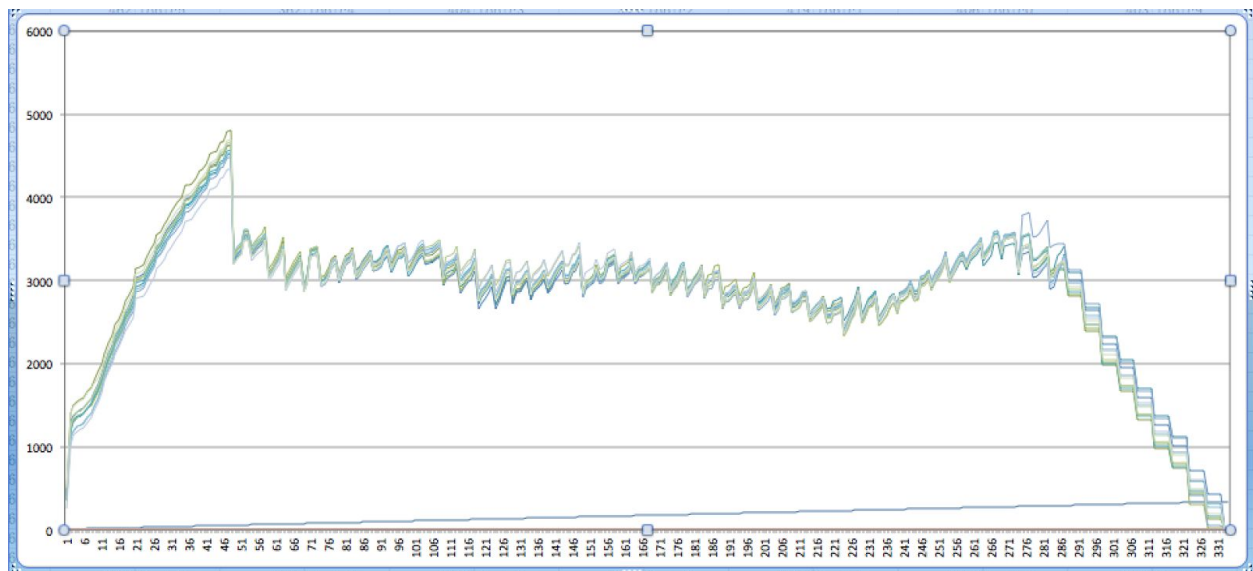


```

Run: CAServer CASubClient CAPubClient
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/
Publisher Thread 29 runs for 282889
Publisher Thread 11 runs for 283939
Publisher Thread 16 runs for 285120
Publisher Thread 13 runs for 285184
Publisher Thread 17 runs for 285300
Publisher Thread 12 runs for 285899
Publisher Thread 23 runs for 285900
Publisher Thread 14 runs for 285909
Publisher Thread 24 runs for 285913
Publisher Thread 22 runs for 285929
Publisher Thread 15 runs for 287163
Publisher Thread 25 runs for 287347
Publisher Thread 10 runs for 287356
Publisher Thread 19 runs for 287411
Publisher Thread 26 runs for 287944
Publisher Thread 20 runs for 287969
Publisher Thread 27 runs for 288096
Publisher Thread 21 runs for 288137
Publisher Thread 28 runs for 288397
Publisher Thread 18 runs for 288486

```

Number of messages for each topic for Content Aggregator:
 (I'm not familiar with Java Diagnostic so I just output the message size of every second to a csv file. I have also included the csv file in the repo.)



The shape shows that at first all the messages grew steadily, and when it reached its peak (at 36s), the number remained stable for quite a few seconds. That's the point when the speed that publishers publish message is roughly equal to speed that messages expired. Then after the first batch of messages expired, the number started to go down, then to 0.

I'm using a **pull** approach for subscribers to retrieve latest contents, and each time when a subscriber wants to retrieve a message, it takes $O(n)$ time, where n is the number of messages of the topic that the subscriber subscribes to. Moreover, for every 5 seconds I'll run a scheduled timer task to check the TTL of the oldest messages, and remove outdated messages, that takes $O(nk)$ time, where k is the total number of topics. In practice the performance of my code (on server side) is terrible, the wall time for both subscriber and publisher clients are quite long. In fact the subscribers cannot even consume all the messages before the default TTL (30 seconds) of the messages. Only when I increase the TTL to 45 seconds, can all the subscribers consume all the messages.

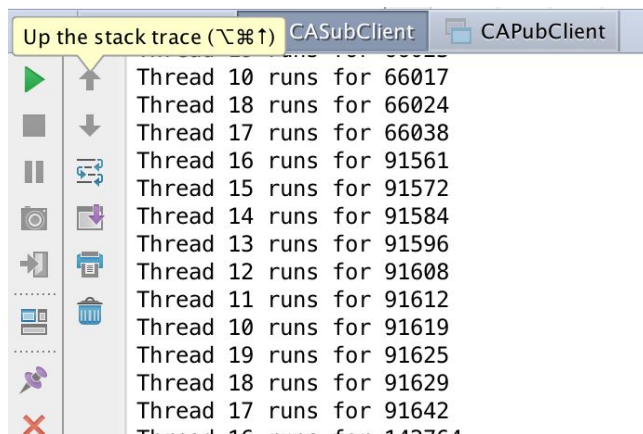
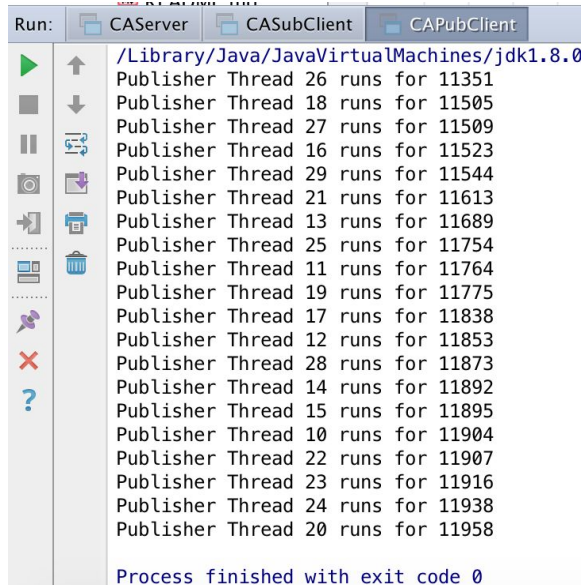
That being said, the correctness of my solution looks good. There is **not a single** failed attempt to send/receive messages for publishers or subscribers.

Step 4 – Stress Testing 2

The wall times for each subscriber and publisher are: (in milliseconds)

Publisher	Wall Time	Publisher	Wall Time	Subscriber	Wall Time
Pub1	11907	Pub11	11916	Sub1	91642
Pub2	11904	Pub12	11938	Sub2	91629
Pub3	11895	Pub13	11958	Sub3	91625
Pub4	11838	Pub14	11892	Sub4	91619
Pub5	11775	Pub15	11873	Sub5	91612
Pub6	11764	Pub16	11853	Sub6	91608
Pub7	11689	Pub17	11754	Sub7	91596
Pub8	11544	Pub18	11613	Sub8	91584
Pub9	11509	Pub19	11523	Sub9	91572
Pub10	11351	Pub20	11505	Sub10	91561

IntelliJ output:



Compare to last time, the wall time of both publishers and subscribers are much much shorter. That's because I store all the messages by each topic in a single HashMap<> object, when we are running publishers and subscribers simultaneously, they both need to obtain the lock for the hashmap so that they can read/write data to it. I enforce those operations to be serial, so running publishers and subscribers simultaneously will be a lot more slower than running subscribers after publishers are done.