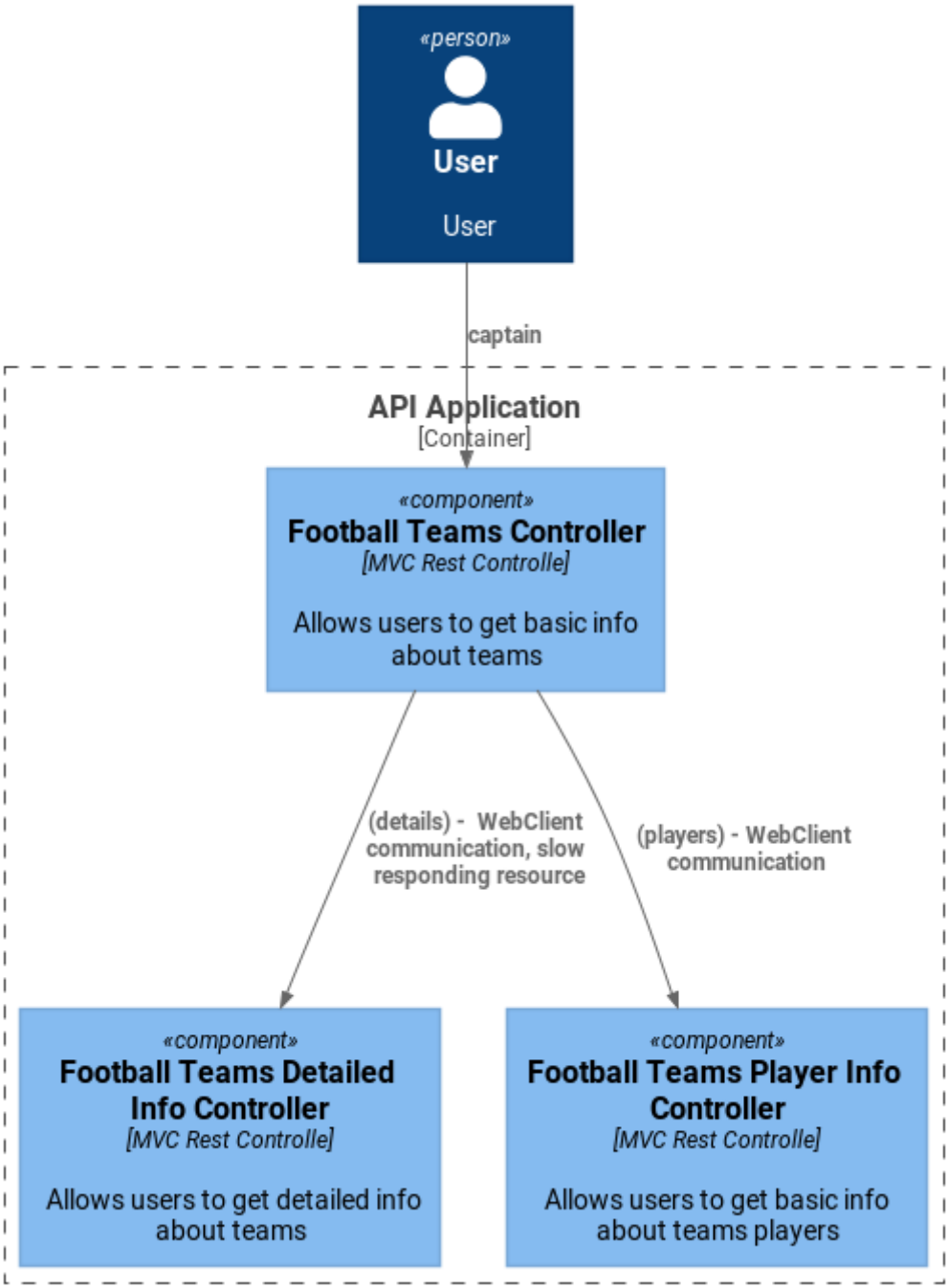
Test construction

The test is structured as follows. The client communicates with the controller and can perform two actions "captain" and "score". Each of them communicates with a different controller (via WebClient). After the "captain" action, TeamController communicates with TeamPlayersController, and after the "score" action, TeamController communicates with TemDetailsController.

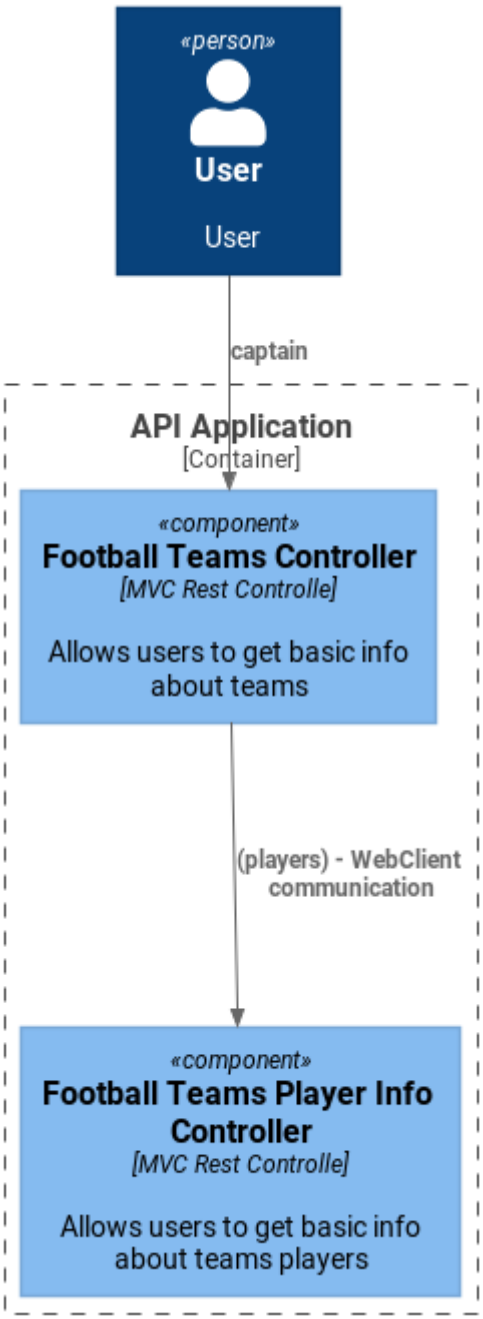
One of these resources is slow responding.

Test common structure

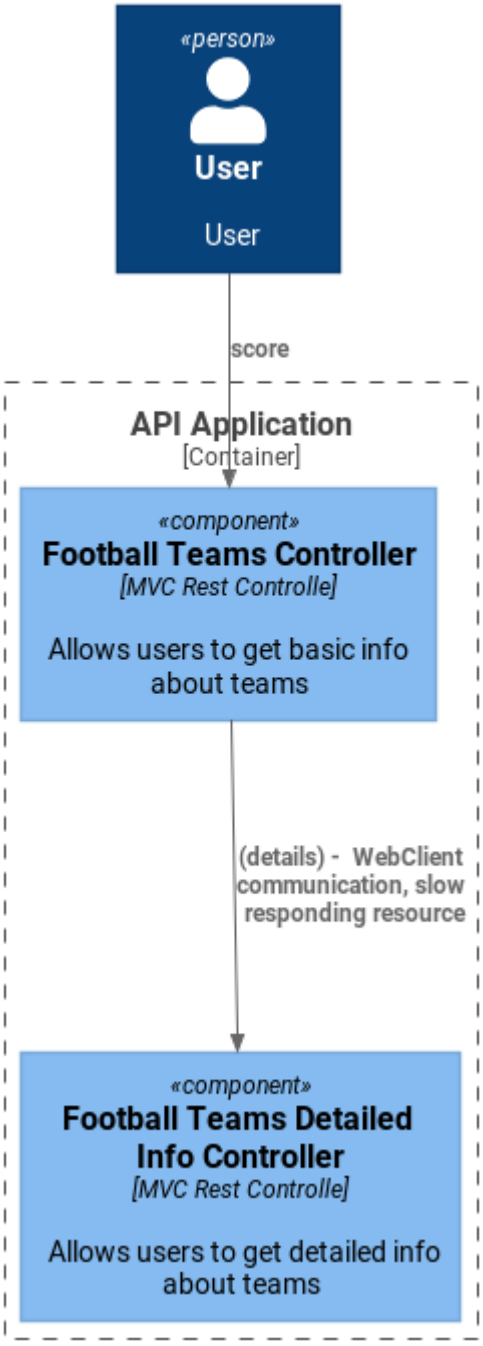


Test "captain" path communication

-



Test "score" path communication



Environment configuration

Property	Value
Environment	Spring`boot
Web server	Tomcat
Maximum amount of worker threads (server.tomcat.threads.max)	200
Maximum queue length for incoming connection requests when all possible request processing threads are in use. (server.tomcat.accept-count)	100

https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.web

Tool	Role
h2load,ab,hey,wrk	Benchmark tools
Micrometer,Prometheus,Grafana	Monitoring
http://gclogs.com/	GC logs analyser

Test - scenario 1

Traffic with the following parameters was generated:

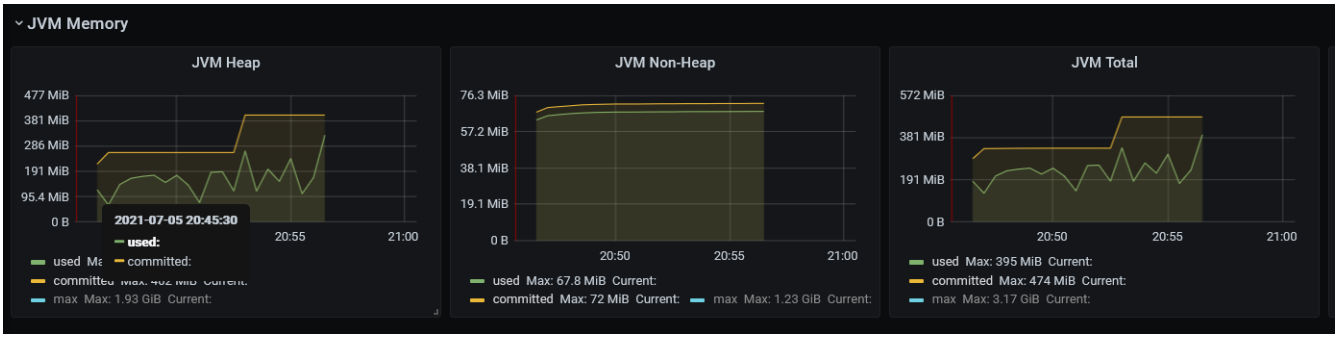
```
hey -z 10m -c 100 http://localhost:8080/team/PL/score
```

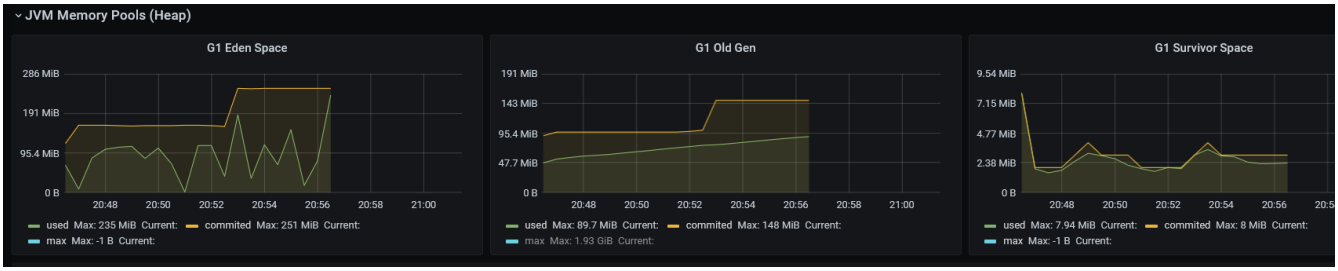
```
hey -z 10m -c 100 http://localhost:8080/team/PL/captain
```

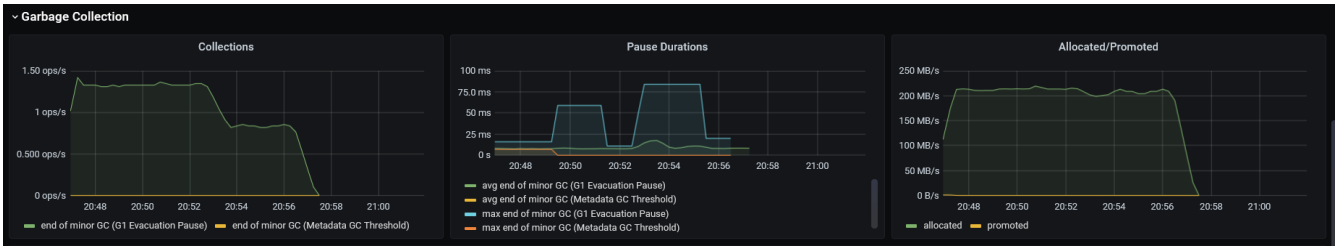
Results

```
mw@localhost:~> hey -z 10m -c 100 http://localhost:8080/team/PL/score

Summary:
  Total:        600.2506 secs
  Slowest:      1.0942 secs
  Fastest:      0.2520 secs
  Average:      0.4023 secs
  Requests/sec: 248.4312


Response time histogram:
  0.252 [1]     |
  0.336 [1521]  |■
  0.420 [113217]        |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.505 [33811] |■■■■■■■■■■■■
  0.589 [459]   |
  0.673 [49]    |
  0.757 [10]    |
  0.842 [11]    |
  0.926 [28]    |
  1.010 [9]     |
  1.094 [5]     |


Latency distribution:
  10% in 0.3702 secs
  25% in 0.3849 secs
  50% in 0.4009 secs
  75% in 0.4187 secs
  90% in 0.4373 secs
  95% in 0.4495 secs
  99% in 0.4800 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0000 secs, 0.2520 secs, 1.0942 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0039 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0242 secs
  resp wait:    0.4021 secs, 0.2519 secs, 1.0735 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0397 secs

Status code distribution:
  [200] 149121 responses



mw@localhost:~>
```

```
mw@localhost:/usr/bin> hey -z 10m -c 100 http://localhost:8080/team/PL/captain

Summary:
  Total:        600.0984 secs
  Slowest:      1.6296 secs
  Fastest:      0.0401 secs
  Average:      0.1596 secs
  Requests/sec: 626.6822


Response time histogram:
  0.040 [1]     |
  0.199 [340465]        |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.358 [35238] |■■■■
  0.517 [233]   |
  0.676 [34]    |
  0.835 [0]     |
  0.994 [0]     |
  1.153 [0]     |
  1.312 [10]    |
  1.471 [76]    |
  1.630 [14]    |


Latency distribution:
  10% in 0.1224 secs
  25% in 0.1386 secs
  50% in 0.1567 secs
  75% in 0.1771 secs
  90% in 0.1979 secs
  95% in 0.2123 secs
  99% in 0.2481 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0000 secs, 0.0401 secs, 1.6296 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0068 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0508 secs
  resp wait:    0.1594 secs, 0.0400 secs, 1.6195 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0794 secs

Status code distribution:
  [200] 376071 responses



mw@localhost:/usr/bin>
```



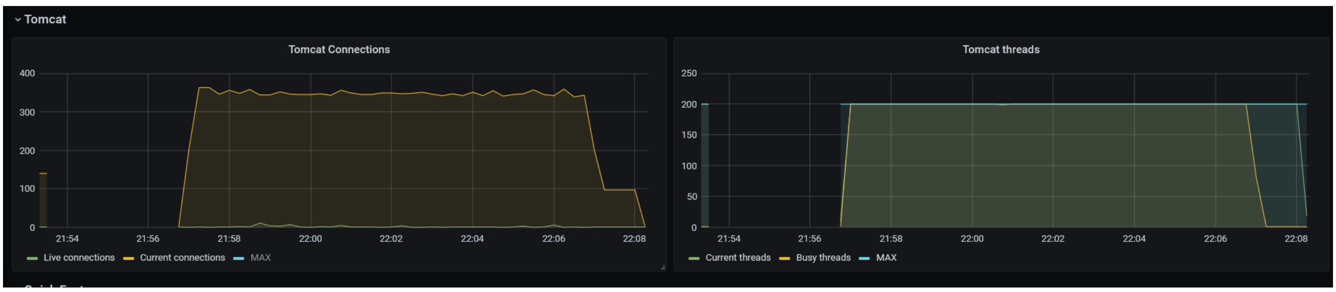


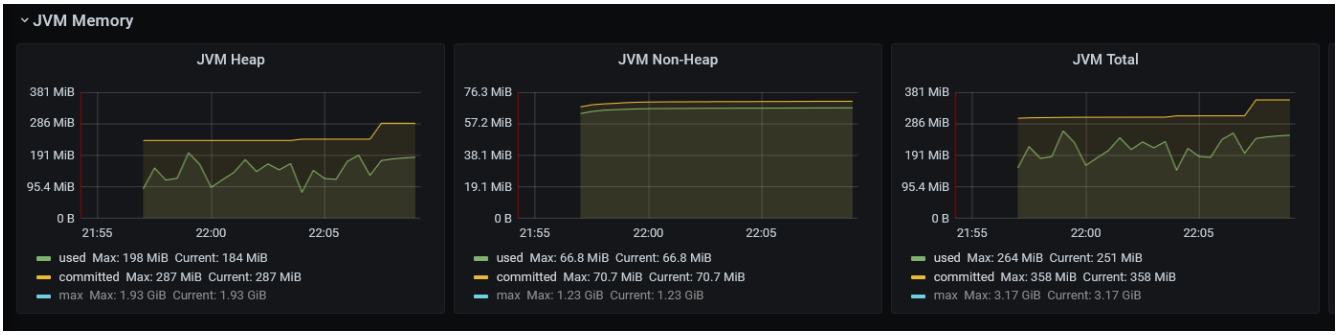


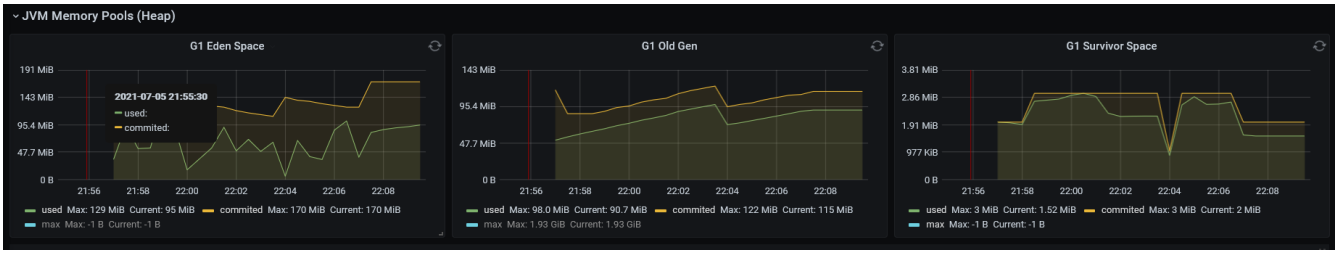
Test - scenario 2 - memory leak (static collection)

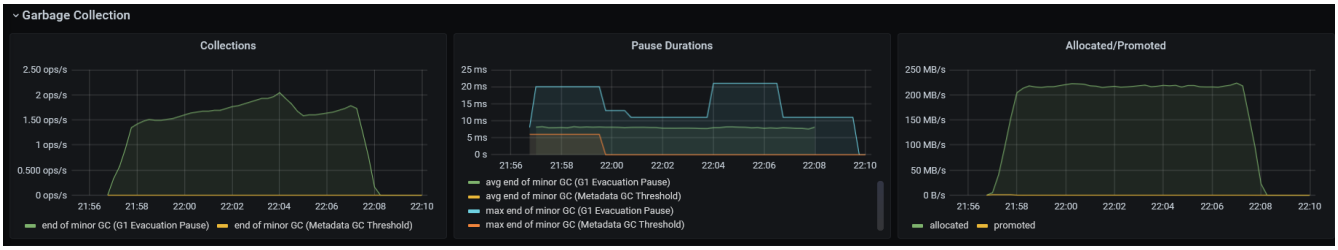
```
mw@localhost:/usr/bin> hey -z 10m -c 100 http://localhost:8080/team/PL/score

Summary:
  Total:        600.4155 secs
  Slowest:      1.4293 secs
  Fastest:      0.2521 secs
  Average:      0.3977 secs
  Requests/sec: 251.3709


Response time histogram:
  0.252 [1]     |
  0.370 [19080] |■■■■■■
  0.488 [131249]        |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.605 [478]   |
  0.723 [19]    |
  0.841 [0]     |
  0.958 [0]     |
  1.076 [0]     |
  1.194 [0]     |
  1.312 [57]    |
  1.429 [43]    |


Latency distribution:
  10% in 0.3659 secs
  25% in 0.3819 secs
  50% in 0.3985 secs
  75% in 0.4162 secs
  90% in 0.4338 secs
  95% in 0.4450 secs
  99% in 0.4703 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0000 secs, 0.2521 secs, 1.4293 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0008 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0174 secs
  resp wait:    0.3975 secs, 0.2520 secs, 1.4202 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0360 secs

Status code distribution:
  [200] 150927 responses



mw@localhost:/usr/bin>
```

```
mw@localhost:~> hey -z 10m -c 100 http://localhost:8080/team/PL/captain
^[[A
Summary:
  Total:        600.0460 secs
  Slowest:      0.5058 secs
  Fastest:      0.0079 secs
  Average:      0.1534 secs
  Requests/sec: 651.7350


Response time histogram:
  0.008 [1]     |
  0.058 [4999]  |■
  0.107 [22063] |■■■■■
  0.157 [189182]        |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.207 [154530]        |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
  0.257 [19026] |■■■■
  0.307 [1175]  |
  0.356 [83]    |
  0.406 [10]    |
  0.456 [1]     |
  0.506 [1]     |


Latency distribution:
  10% in 0.1149 secs
  25% in 0.1338 secs
  50% in 0.1534 secs
  75% in 0.1738 secs
  90% in 0.1944 secs
  95% in 0.2078 secs
  99% in 0.2374 secs

Details (average, fastest, slowest):
  DNS+dialup:   0.0000 secs, 0.0079 secs, 0.5058 secs
  DNS-lookup:   0.0000 secs, 0.0000 secs, 0.0064 secs
  req write:    0.0000 secs, 0.0000 secs, 0.0517 secs
  resp wait:    0.1533 secs, 0.0077 secs, 0.5058 secs
  resp read:    0.0001 secs, 0.0000 secs, 0.0553 secs

Status code distribution:
  [200] 391071 responses



mw@localhost:~> hey -z 10m -c 100 http://localhost:8080/team/PL/captain
```









Appendix - additional tests - TODO

Additionally, the application creates 6000 sleeping threads at startup. 5 minutes after it starts, threads activate and fill their object arrays while monitoring is in progress. This places additional strain and resource utilization.

```
@PostConstruct
   public void init(){
       for (int i = 0; i < 6000; i++) {
           (new MyThread()).start();
           log.info("Thread created=>"+i);
       }
   }
```

```
class MyThread extends Thread{
    List<MyObject> list = new ArrayList<>(200);

    @Override
    public void run() {
        try {
            Thread.sleep(500*1000);
        } catch (InterruptedException e) {
            throw new IllegalArgumentException(e);
        }


    }

    private void mwstart() {
        for (int i = 0; i < 600; i++) {
            list.add(new MyObject(Thread.currentThread().getName(),i));
            System.out.println("New Object added!
Thread=>"+Thread.currentThread().getName());
        }
    }
}

@AllArgsConstructor
class MyObject{
    String name;
    Integer account;
}
```