Table of Contents

[Monitoring and troubleshooting Practical Task 1](#_Toc77084953)

[OutOfMemory (OOM) error troubleshooting 1](#_Toc77084954)

[Deadlock troubleshooting 3](#_Toc77084955)

[Remote JVM profiling 5](#_Toc77084956)

[Inspect a Flight Recording 5](#_Toc77084957)

[jinfo 6](#_Toc77084958)

[Practical task evaluation rules: 6](#_Toc77084959)

[References 7](#_Toc77084960)

# Monitoring and troubleshooting Practical Task

## OutOfMemory (OOM) error troubleshooting

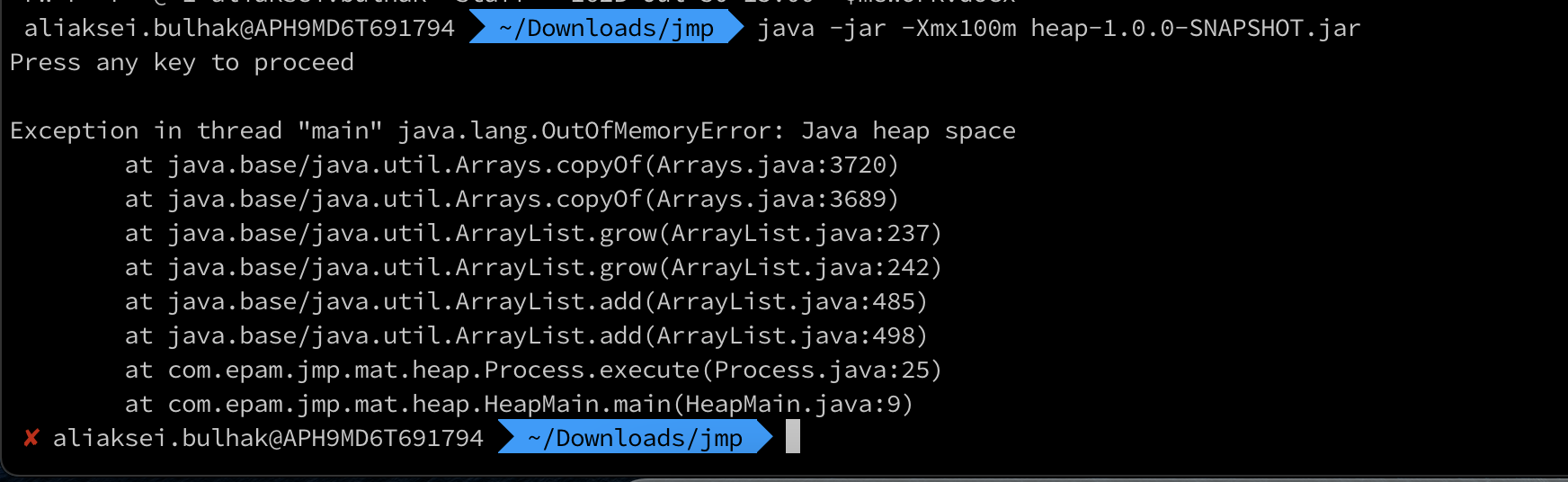
#### Get OOM error

Execute and press any key:

```

java -jar -Xmx100m heap-1.0.0-SNAPSHOT.jar

```



#### Use jvisualvm to observe OOM

- Execute:

```

java -jar -Xmx100m heap-1.0.0-SNAPSHOT.jar

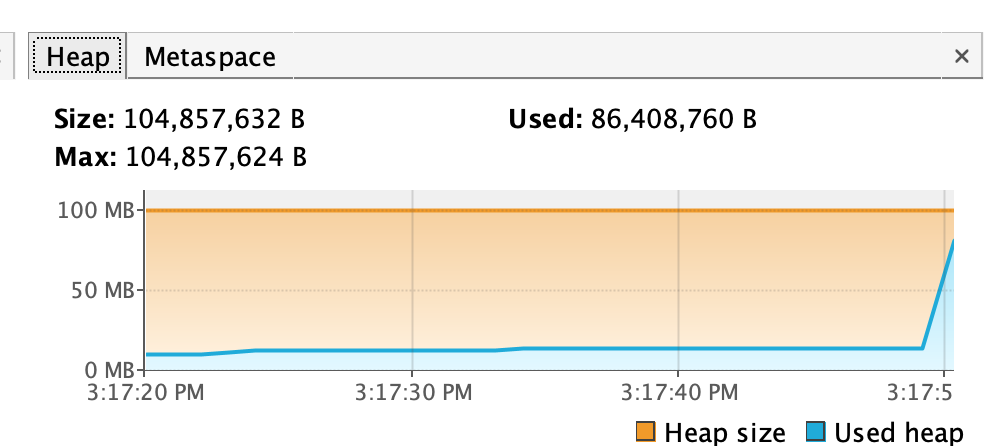
```

- In jvisualvm connect to our java process

- Go to "Monitor" tab

- Press any key in our application

- Observe how heap grows



#### Get heap dump

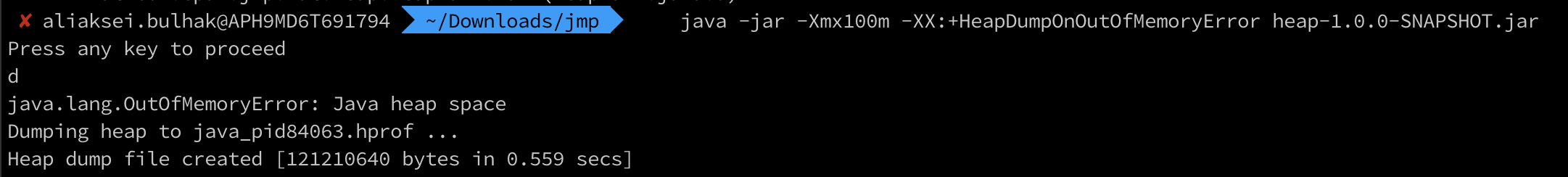
Using -XX:+HeapDumpOnOutOfMemoryError option

- Execute and press any key:

```

java -jar -Xmx100m -XX:+HeapDumpOnOutOfMemoryError heap-1.0.0-SNAPSHOT.jar

```



#### [Optional] Using jcmd

Get pid using `jps` here and further through this document:

```

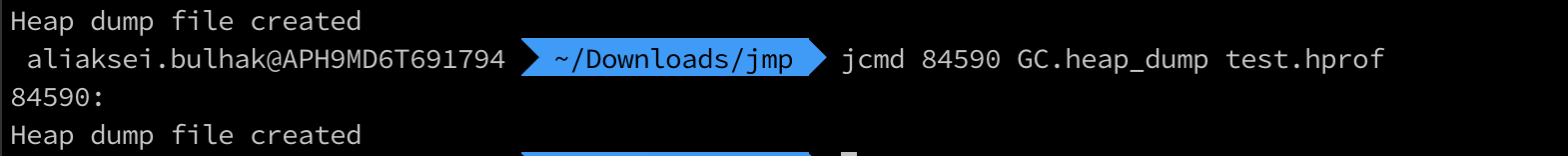
jps -lvm

```

```

jcmd <pid> GC.heap\_dump <filename>

```

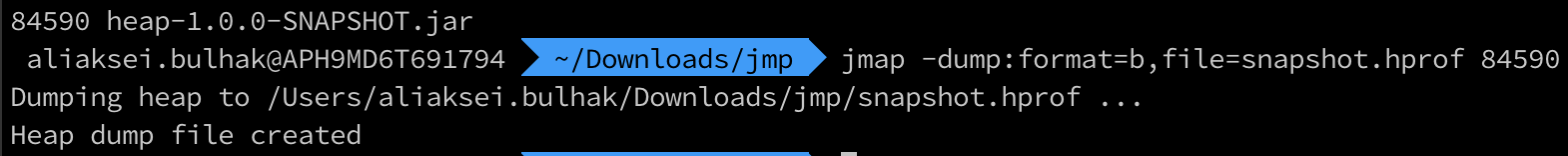


#### [Optional] Using jmap

```

jmap -dump:format=b,file=snapshot.hprof <pid>

```



#### Get heap histogram

Using jcmd

```

jcmd <pid> GC.class\_histogram

```

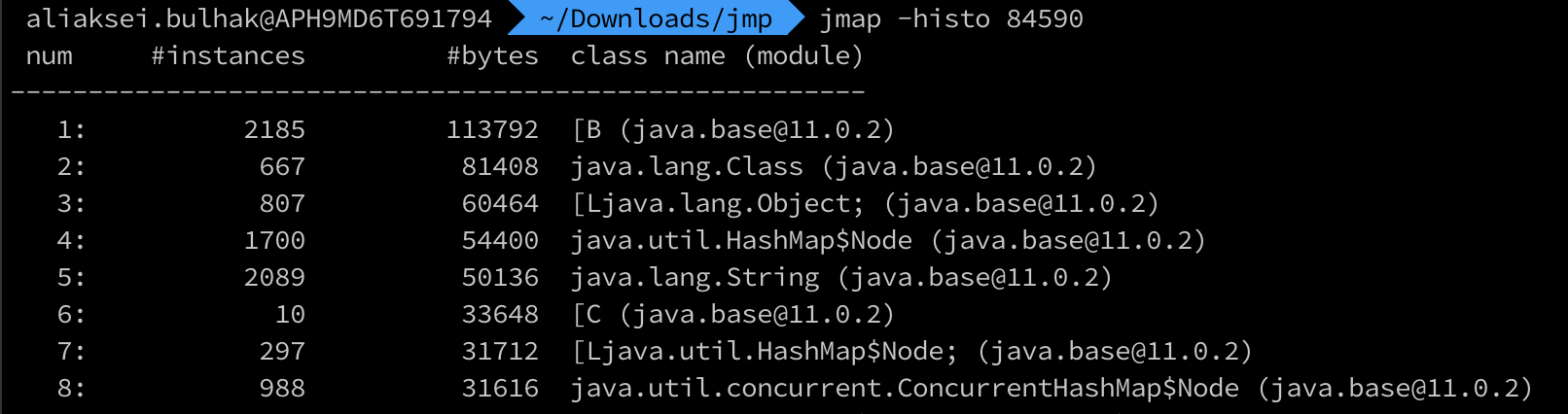


Using jmap

```

jmap -histo <pid>

```

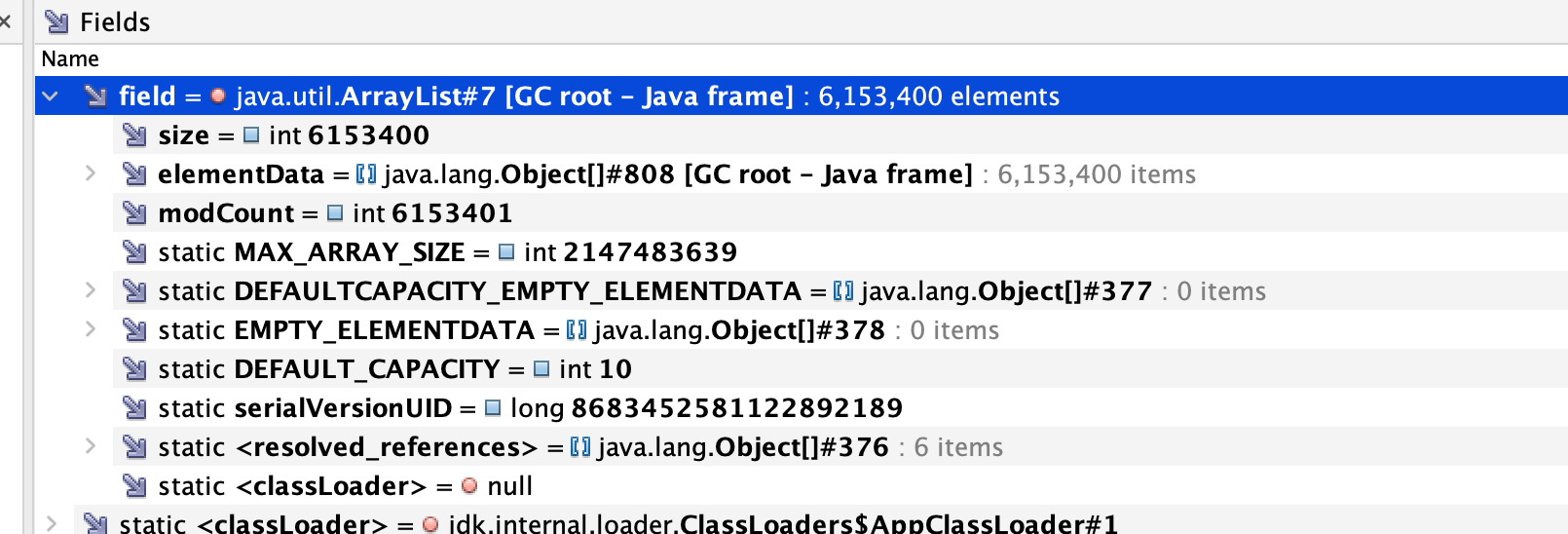


#### Analyze heap dump

Using Java Visual VM

- Open retrieved heap dump in jvisualvm

- Identify memory leak



OQL

Execute OQL in jvisualvm:

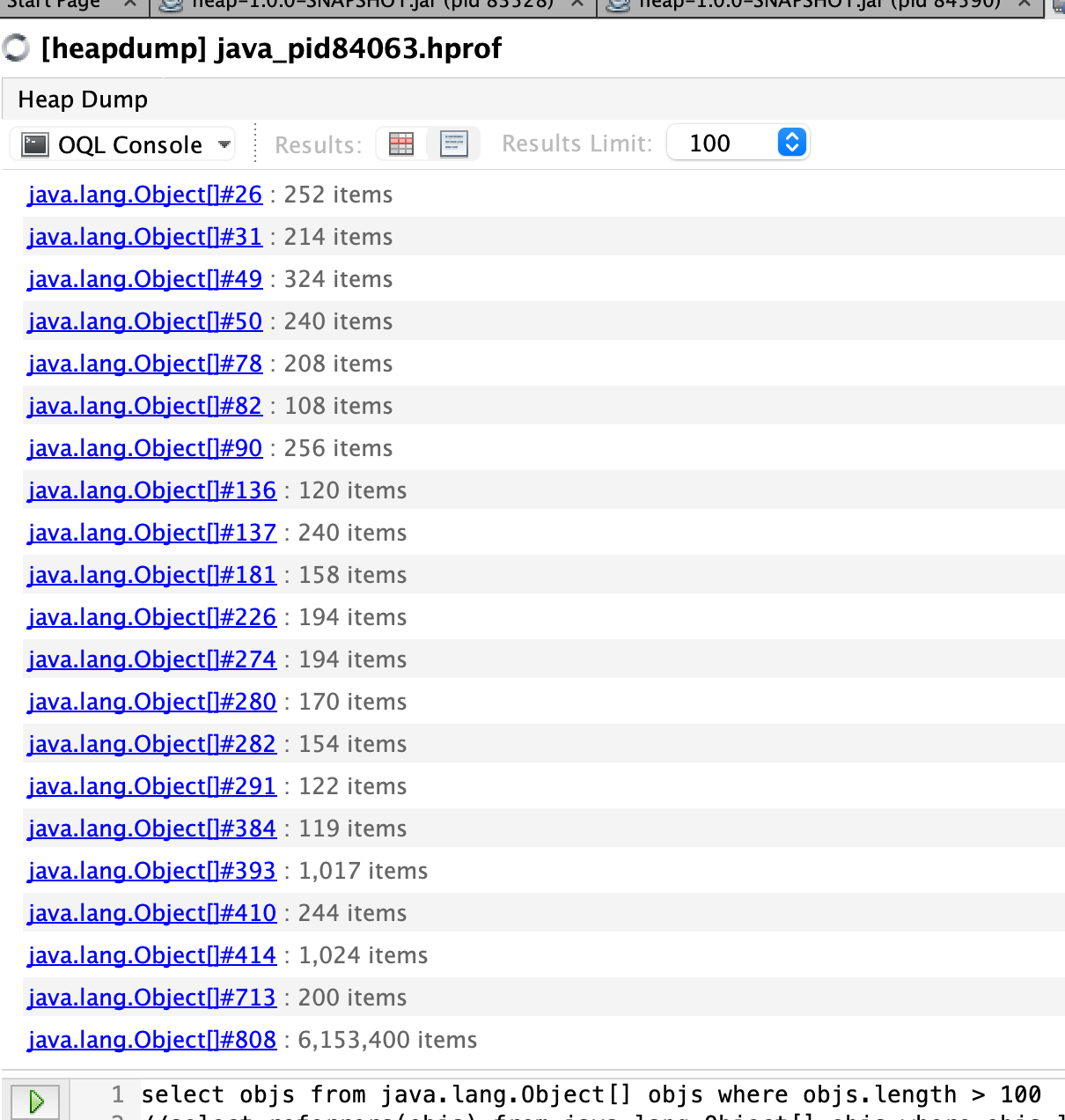
```

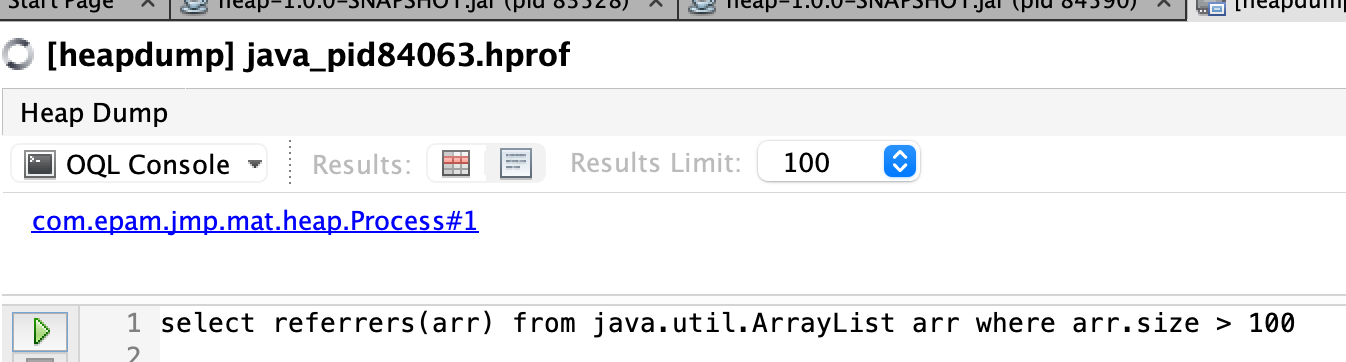
select objs from java.lang.Object[] objs where objs.length > 100

select referrers(objs) from java.lang.Object[] objs where objs.length > 100

select referrers(arr) from java.util.ArrayList arr where arr.size > 100

```



Startup `jhat` (note: `jhat` was decommissioned in JDK 9)

```

jhat <head\_dump.hprof>

```

Execute OQL in jhat

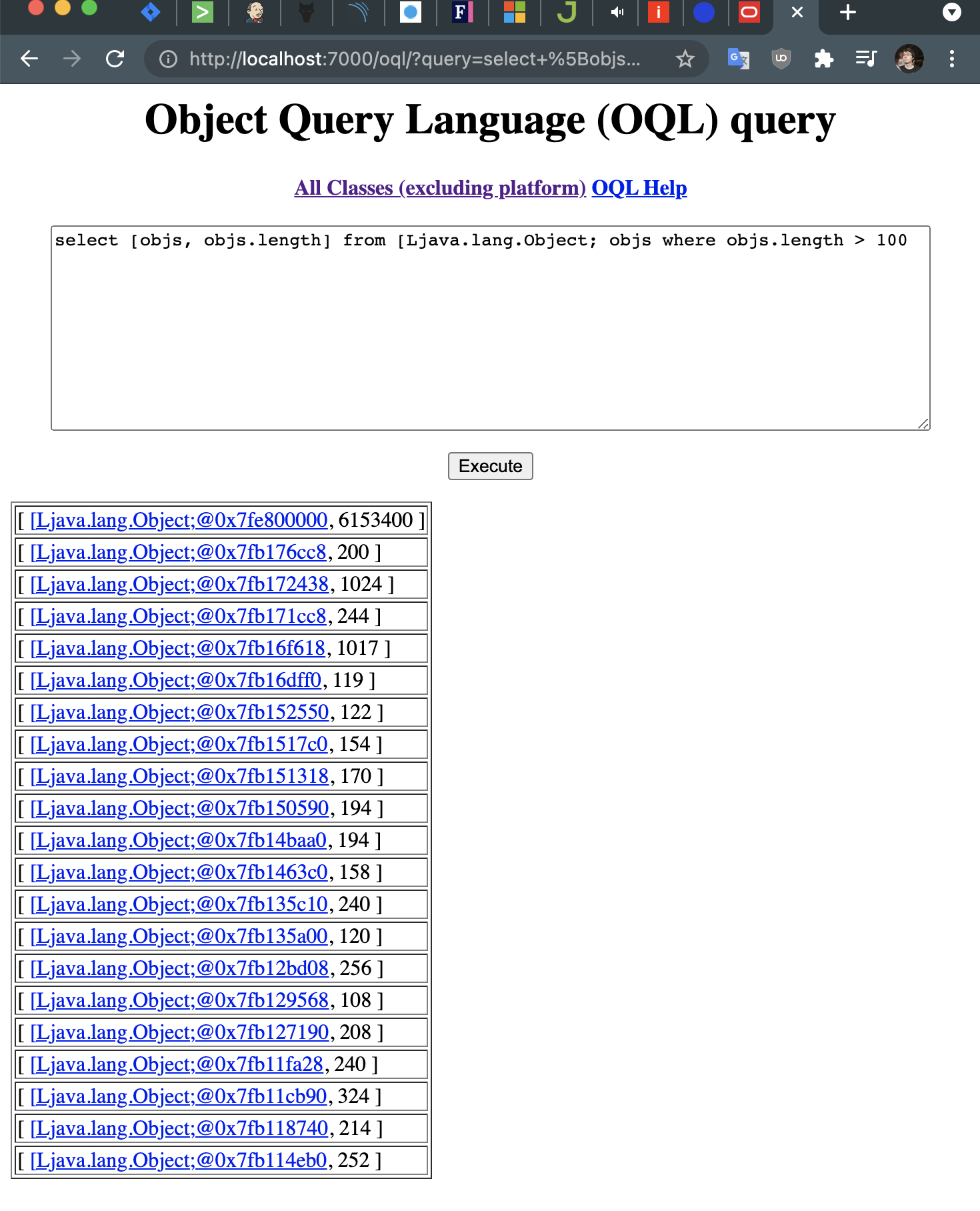
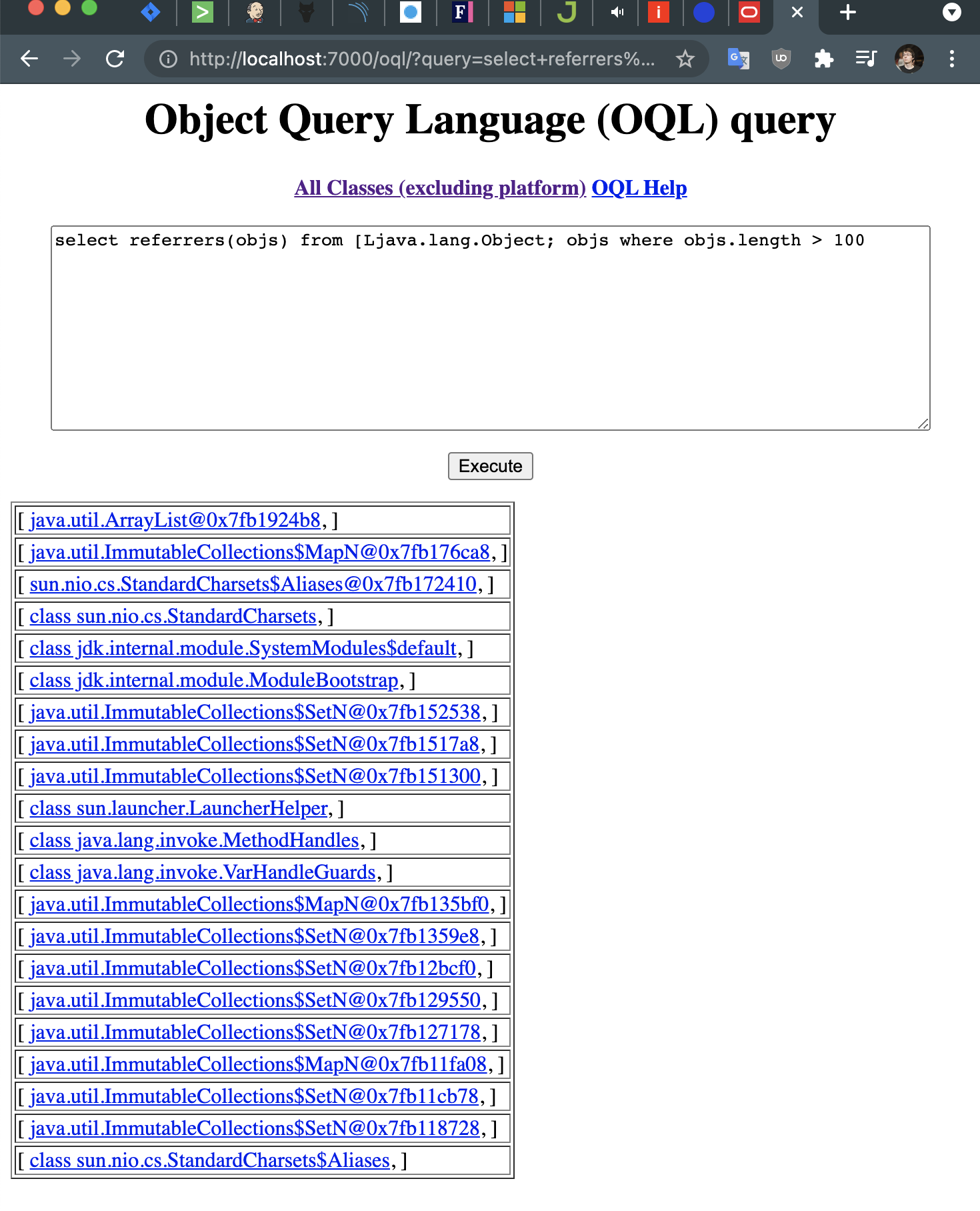
```

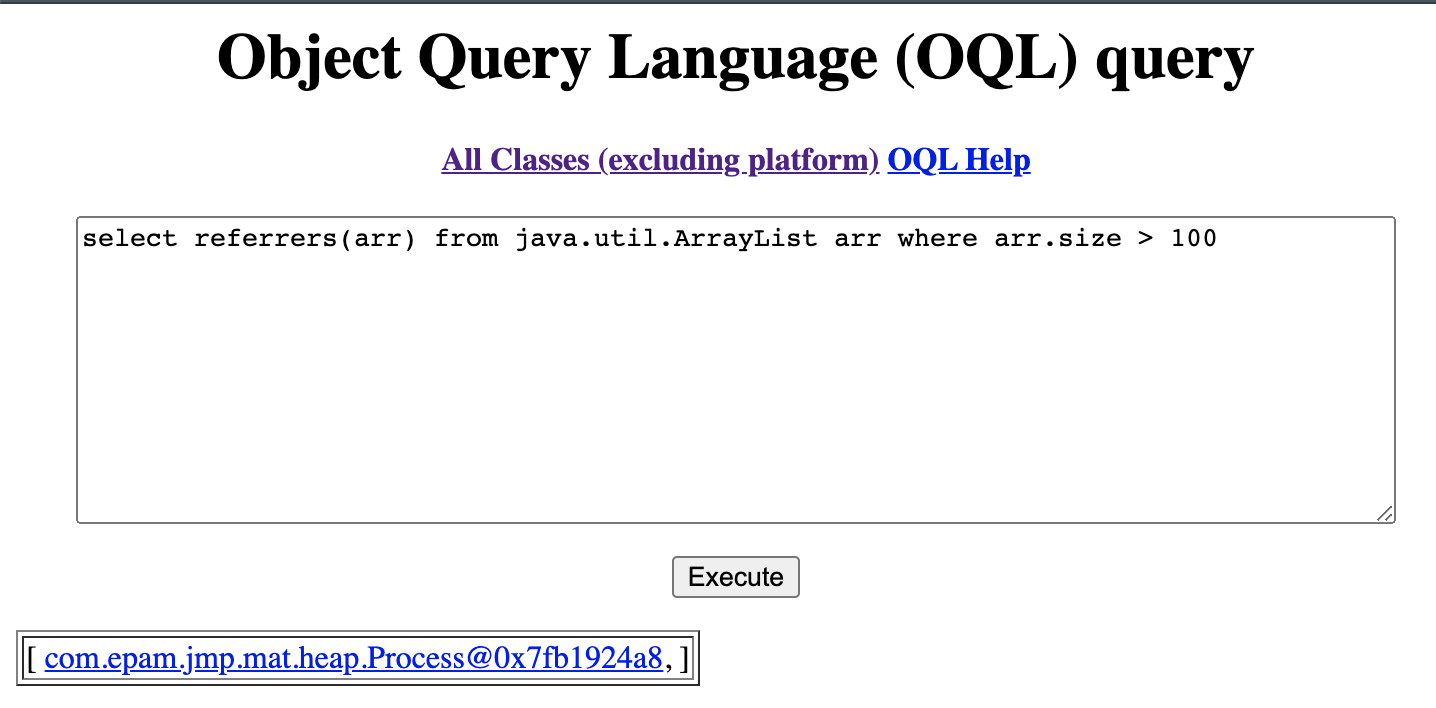
select [objs, objs.length] from [Ljava.lang.Object; objs where objs.length > 100

select referrers(objs) from [Ljava.lang.Object; objs where objs.length > 100

select referrers(arr) from java.util.ArrayList arr where arr.size > 100

```



Please note small OQL syntax difference in jhat and jvisualvm.

## Deadlock troubleshooting

#### Get deadlock

- Execute java application that simulates deadlock:

```

java -jar deadlock-1.0.0-SNAPSHOT.jar

```

- Get thread dump and locate lines similar to:

```

Found one Java-level deadlock:

=============================

"Thread 2":

waiting to lock monitor 0x000000001bf40b68 (object 0x000000076b7777c8, a java.lang.Object),

which is held by "Thread 1"

"Thread 1":

waiting to lock monitor 0x000000001bf43608 (object 0x000000076b7777d8, a java.lang.Object),

which is held by "Thread 2"

Java stack information for the threads listed above:

===================================================

"Thread 2":

at com.epam.jmp.mat.deadlock.SimulateDeadLock.method2(SimulateDeadLock.java:44)

- waiting to lock <0x000000076b7777c8> (a java.lang.Object)

- locked <0x000000076b7777d8> (a java.lang.Object)

at com.epam.jmp.mat.deadlock.DeadLockMain$2.run(DeadLockMain.java:18)

"Thread 1":

at com.epam.jmp.mat.deadlock.SimulateDeadLock.method1(SimulateDeadLock.java:24)

- waiting to lock <0x000000076b7777d8> (a java.lang.Object)

- locked <0x000000076b7777c8> (a java.lang.Object)

at com.epam.jmp.mat.deadlock.DeadLockMain$1.run(DeadLockMain.java:11)

Found 1 deadlock.

```

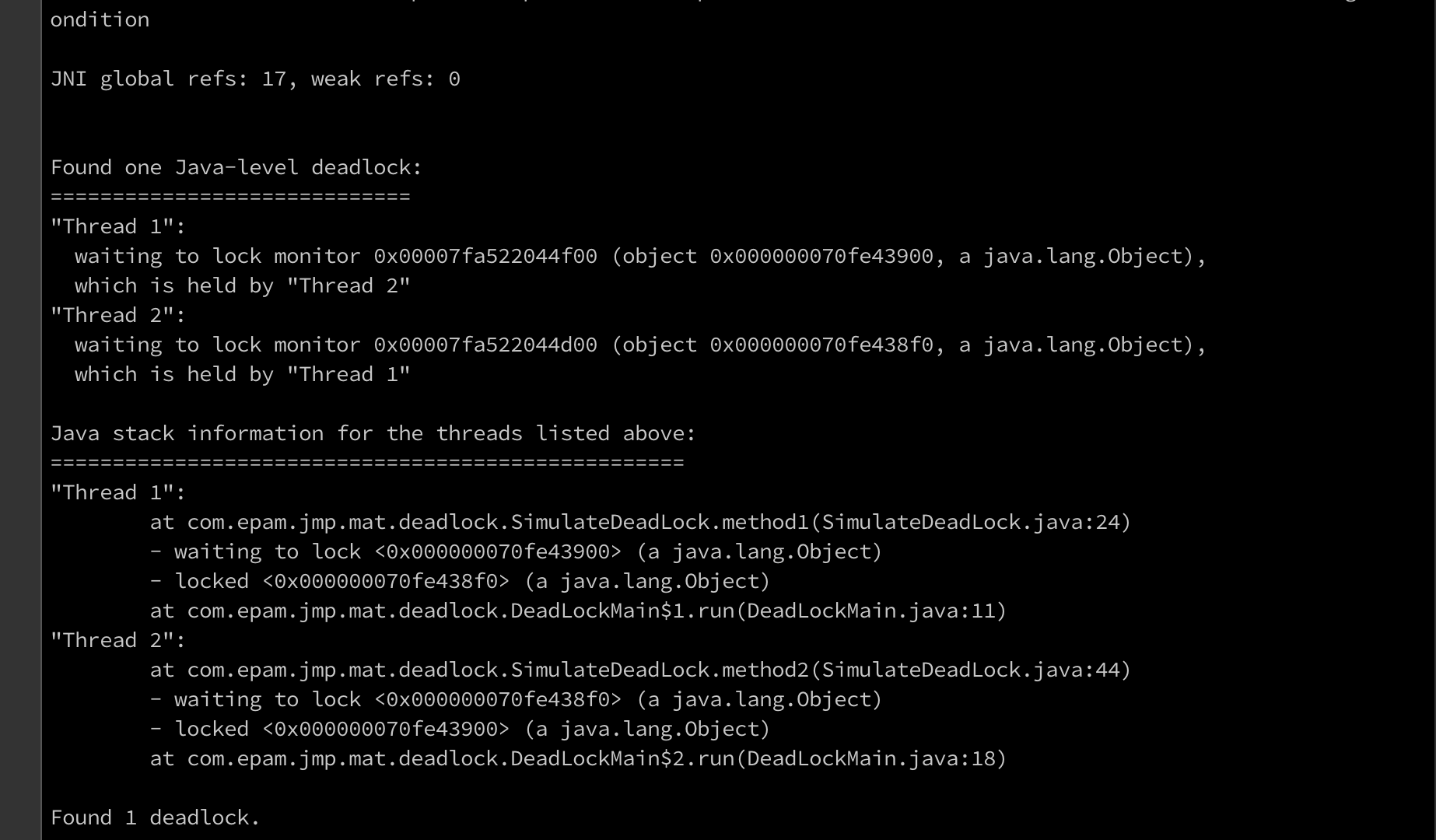
#### Get thread dump

1} jstack

```

jstack -l <pid>

```



2} kill -3

```

kill -3 <pid>

```

3} jvisualvm



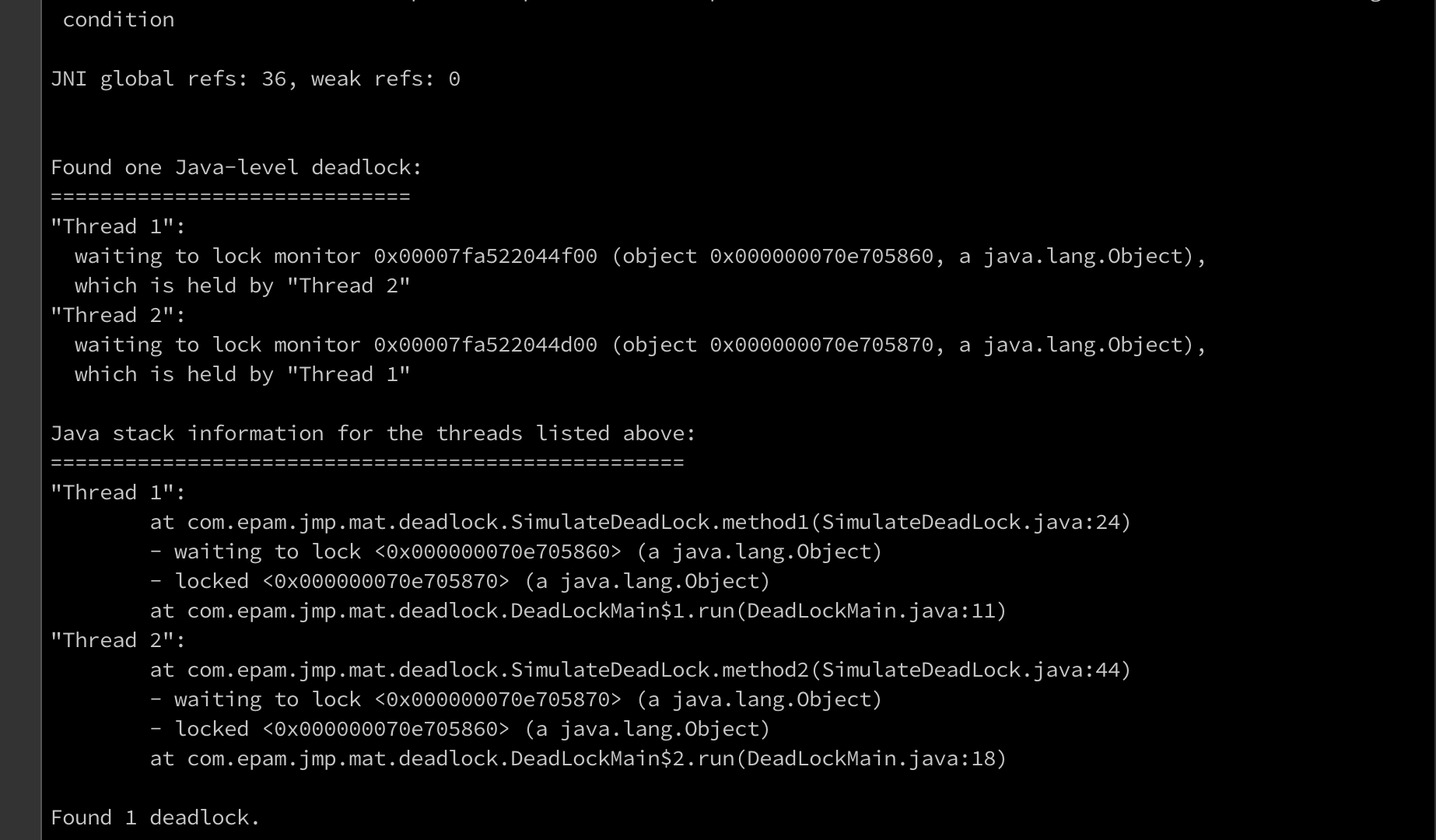
4} Windows (Ctrl + Break)

5} jcmd

```

jcmd <pid> Thread.print

```



## Remote JVM profilingRemote JVM profiling

Using [JMX Technology](https://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html)

For insecure remote connection use parameters:

```

-Dcom.sun.management.jmxremote

-Dcom.sun.management.jmxremote.port=7890

-Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.ssl=false

```

```

java -jar -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=7890 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false simple-1.0.0-SNAPSHOT.jar

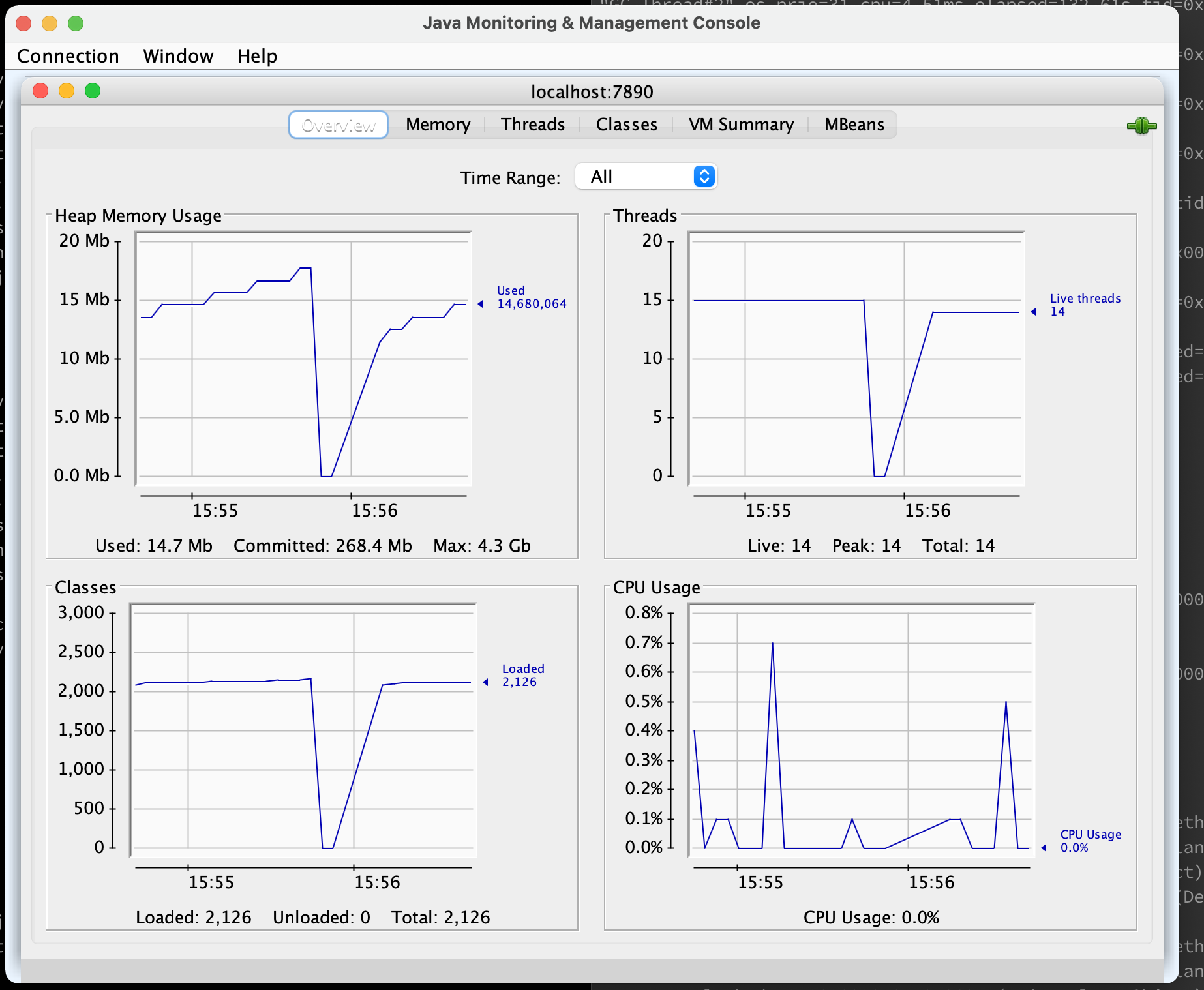
```

Connect to JVM using jconsole:

```

jconsole localhost:7890

```



You see this drop because I pressed enter in terminal to simulate OOM and check how it will be displayed on graphic and then reconnected to application after restart

## Inspect a Flight Recording

Execute JVM with two special parameters:

```

-XX:+UnlockCommercialFeatures

-XX:+FlightRecorder

```

```

java -jar -Xmx100m -XX:+UnlockCommercialFeatures -XX:+FlightRecorder -XX:StartFlightRecording=dumponexit=true,filename=flight.jfr heap-1.0.0-SNAPSHOT.jar

```

Enable Flight Recording on JVM without these parameters:

```

java -jar -Xmx100m -XX:+UnlockCommercialFeatures heap-1.0.0-SNAPSHOT.jar

jps -lvm

jcmd <pid> JFR.start name=heap\_recording filename=flight.jfr dumponexit=true

```

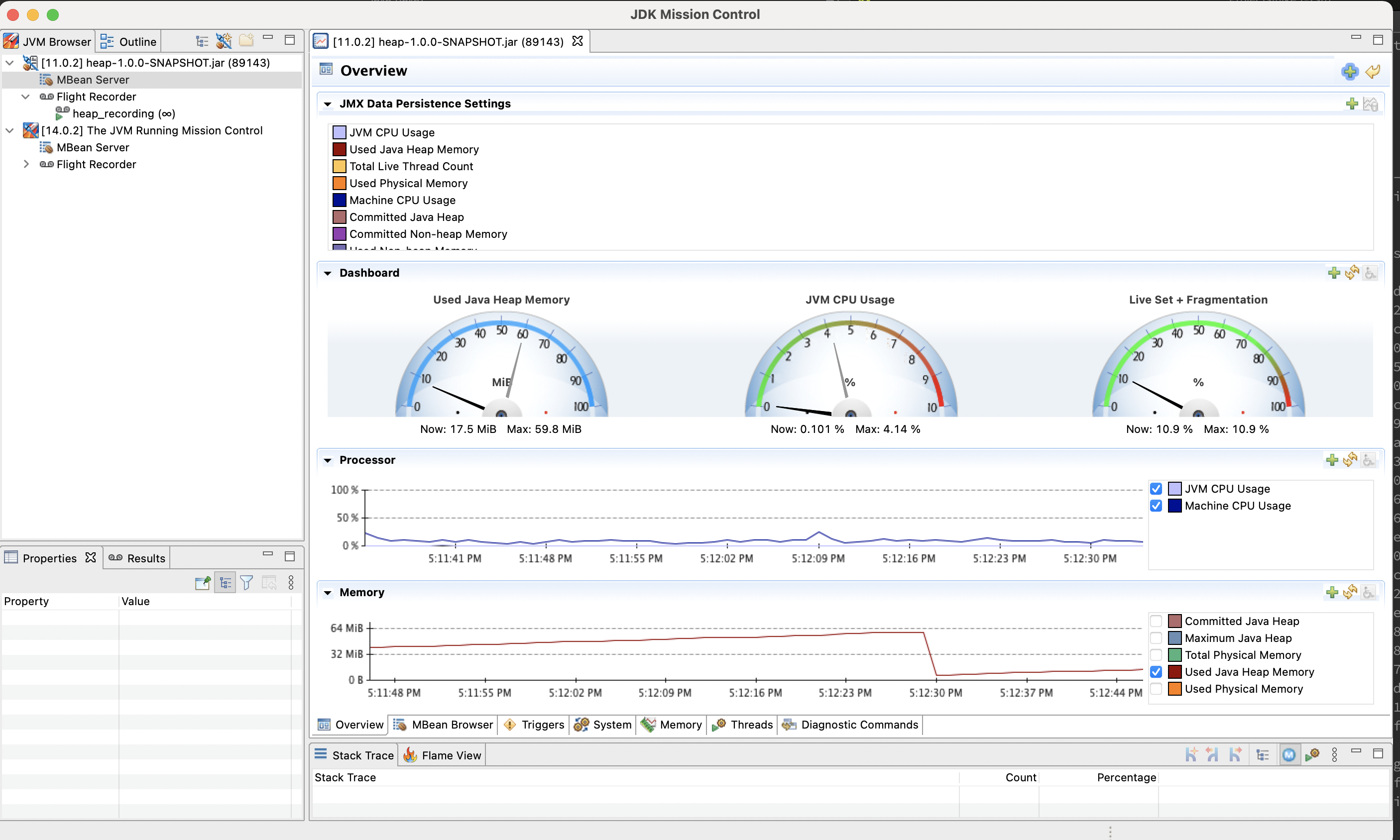
Open Java Mission Control and connect to default HotSpot of our JVM:

```

jmc

```

On Mac if you specify this argument -XX:+UnlockCommercialFeatures then application will fail



## jinfo

Print system properties and command-line flags that were used to start the JVM.

```

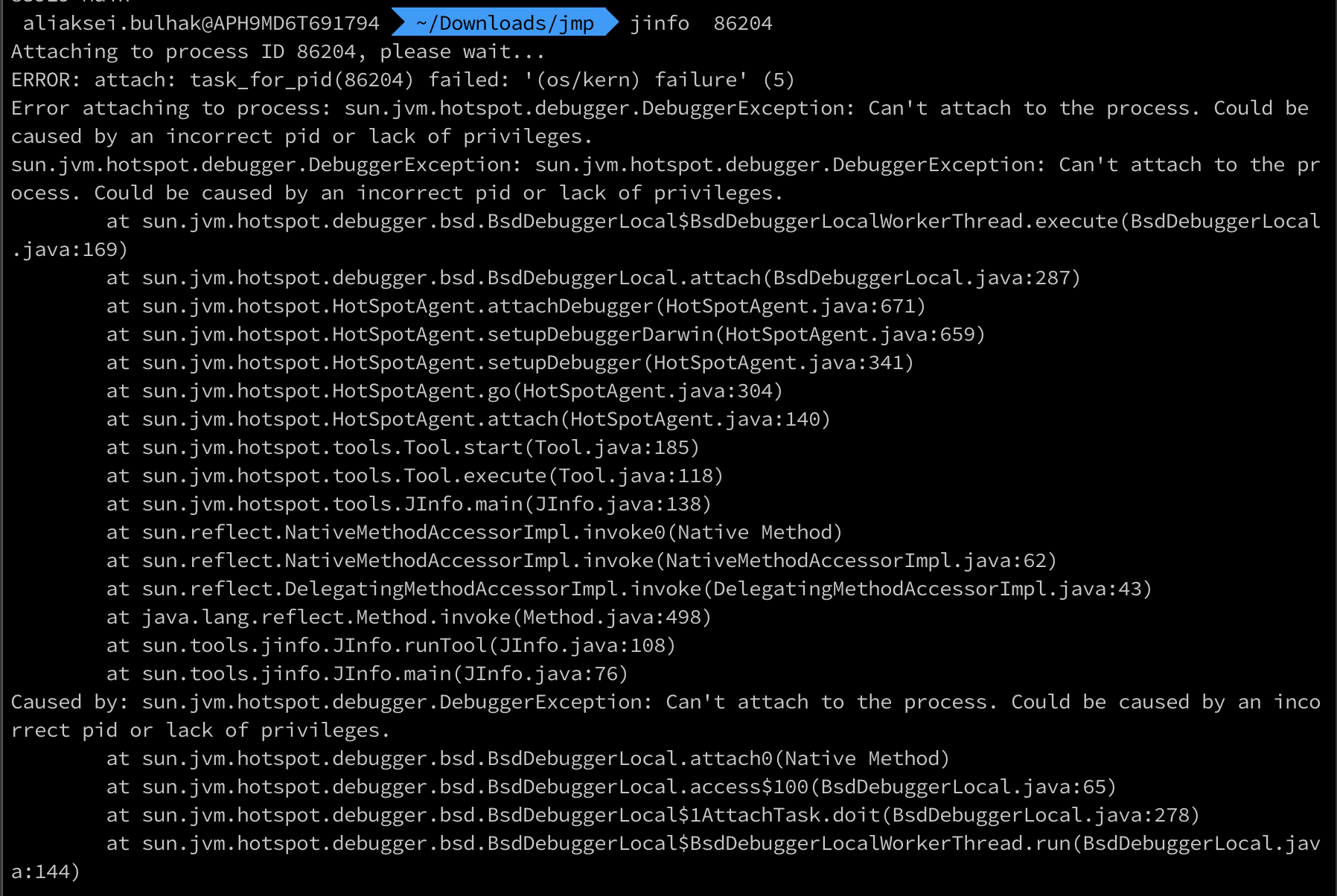
java -jar simple-1.0.0-SNAPSHOT.jar

jps

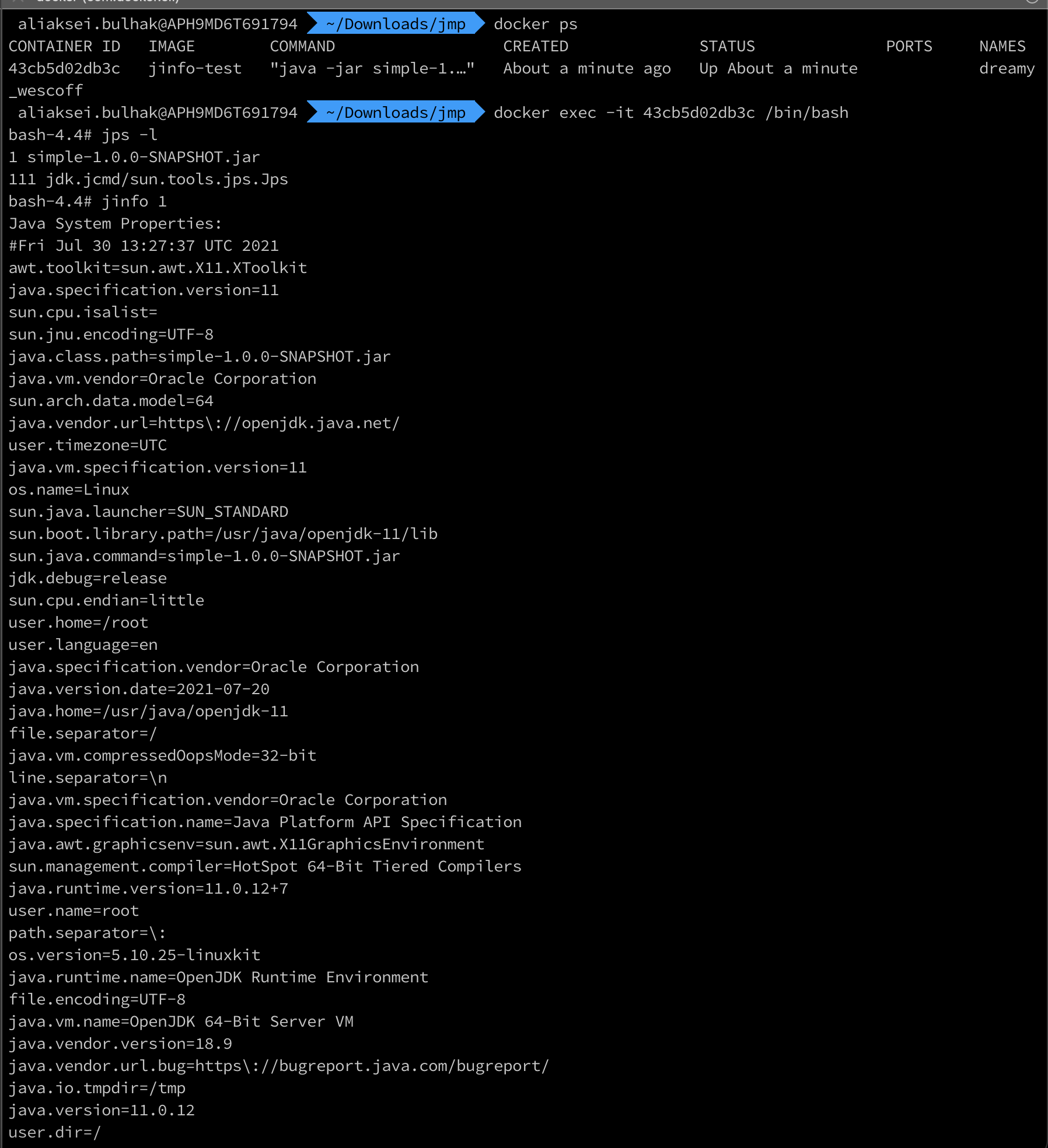
jinfo <pid>

```

On MacOs jinfo does not work and fails with exception



So I packed jar file into docker and executed jinfo inside container



Dockerfile itself is pretty simple:

FROM openjdk:11.0.12-oraclelinux8

COPY simple-1.0.0-SNAPSHOT.jar .

CMD [ "java", "-jar", "simple-1.0.0-SNAPSHOT.jar"]

## Practical task evaluation rules:

\* OOM errors troubleshooting : 1.5 points

\* Deadlock troubleshooting : 1.5 points

\* Remote JVM profiling : 1 point

\* FlightRecorder : 1 point

## References

* [Java 8 Troubleshoot Guide](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/)
* [Monitoring Tools](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr025.html)
* [Java VisualVM](https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/intro.html)
* [JConsole](https://docs.oracle.com/javase/8/docs/technotes/guides/management/jconsole.html)
* [The jcmd Utility](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr006.html)
* [The jmap Utility](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr014.html)
* [The jhat Utility](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr012.html)
* [The jstack Utility](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr016.html)
* [The jps Utility](https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr015.html)
* [JVM Tool Interface (JVM TI)](https://docs.oracle.com/javase/8/docs/technotes/guides/jvmti/)
* [VisualVM Standalone](https://visualvm.github.io/)
* [Analyzing a Heap Dump Using OQL](https://visualvm.github.io/documentation.html)
* [Java Mission Control](https://github.com/openjdk/jmc)