

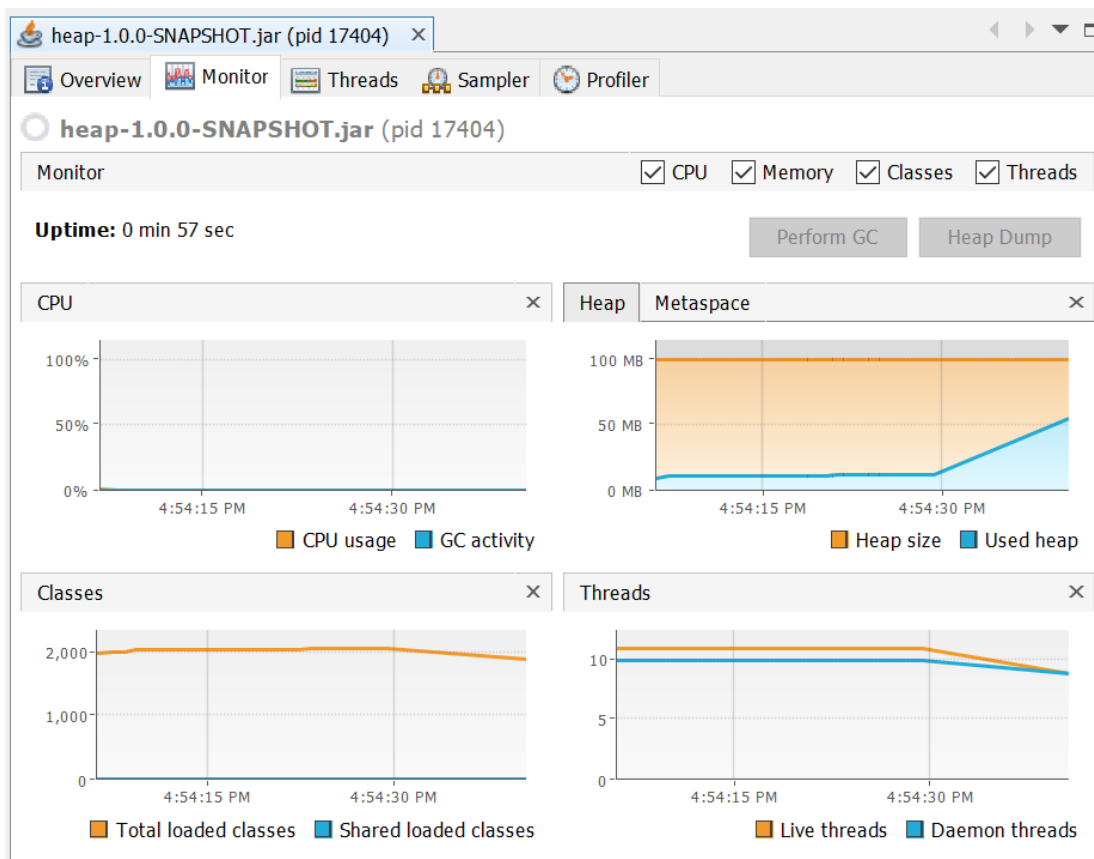
## Task 1:

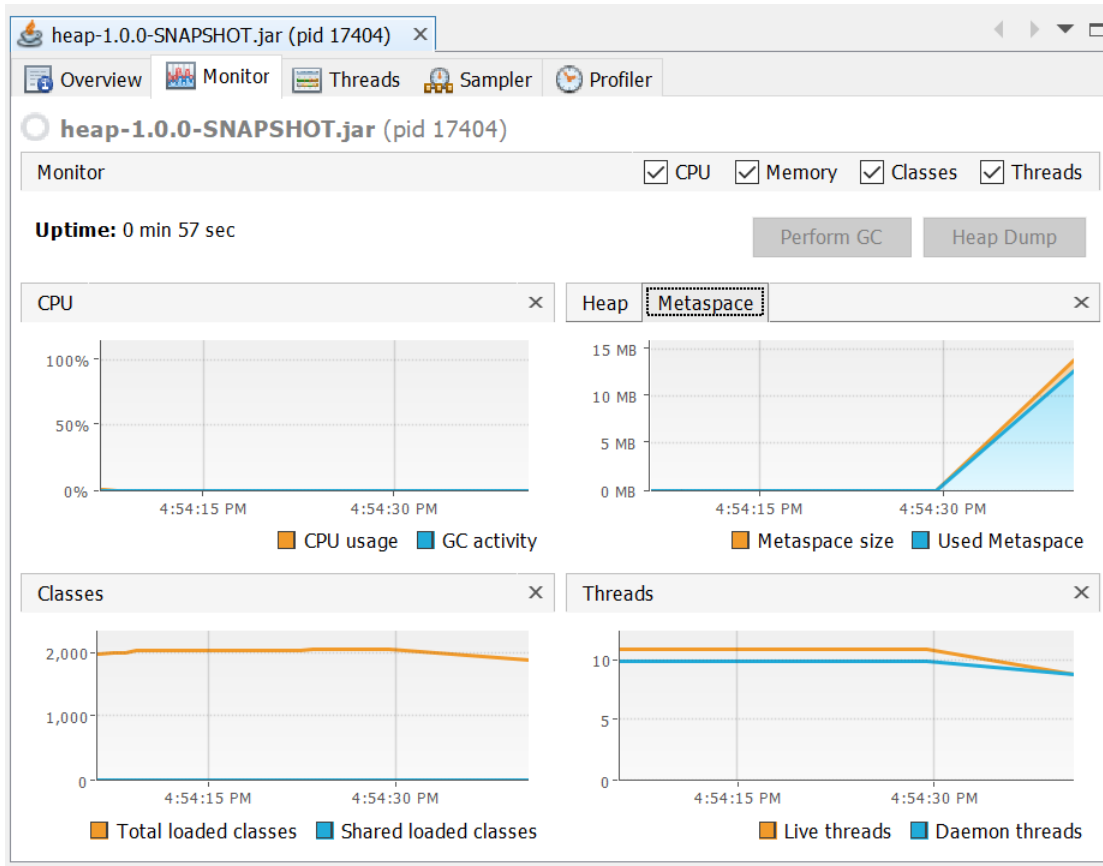
### 1.1 Get OOM error

```
C:\Users\Uliana_Shafatova\Downloads>java -jar -Xmx100m heap-1.0.0-SNAPSHOT.jar
Press any key to proceed
d
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.util.Arrays.copyOf(Arrays.java:3720)
    at java.base/java.util.Arrays.copyOf(Arrays.java:3689)
    at java.base/java.util.ArrayList.grow(ArrayList.java:238)
    at java.base/java.util.ArrayList.grow(ArrayList.java:243)
    at java.base/java.util.ArrayList.add(ArrayList.java:486)
    at java.base/java.util.ArrayList.add(ArrayList.java:499)
    at com.epam.jmp.mat.heap.Process.execute(Process.java:25)
    at com.epam.jmp.mat.heap.HeapMain.main(HeapMain.java:9)
```

На рисунке выше произошла смерть хипы. У хипы слишком маленький максимальный размер, который быстро переполнился, а GC не справился, поэтому она умерла.

### 1.2 Use jvisualvm to observe OOM





Программа насоздавала объектов, очередной оказался слишком большим, чтобы влезть в хипу и она упала с OOM Error.

VisualVM и другие тулы (jconsole, JMC) используют Java Management Beans (MBeans) в JVM, которые публикуют всю эту информацию (когда и как происходила сборка мусора, сколько каких объектов было создано и сколько памяти они съели и т.д.). MBeans поставляются JVM через JMX-агента: он создает реестр таких бинов (просто классы с суффиксом MBean, согласно спецификации) и предоставляет к нему интерфейс, используемый клиентами.

## 1.3 Get heap dump

### 1.3.1 Using -XX:+HeapDumpOnOutOfMemoryError option

```
C:\Users\Uliana_Shafatova\Desktop>java -jar -Xmx100m -XX:+HeapDumpOnOutOfMemoryError heap-1.0.0-SNAPSHOT.jar
Press any key to proceed
pomogite
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid22116.hprof ...
Heap dump file created [121257728 bytes in 0.436 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.util.Arrays.copyOf(Arrays.java:3720)
    at java.base/java.util.Arrays.copyOf(Arrays.java:3689)
    at java.base/java.util.ArrayList.grow(ArrayList.java:238)
    at java.base/java.util.ArrayList.grow(ArrayList.java:243)
    at java.base/java.util.ArrayList.add(ArrayList.java:486)
    at java.base/java.util.ArrayList.add(ArrayList.java:499)
    at com.epam.jsp.mat.heap.Process.execute(Process.java:25)
    at com.epam.jsp.mat.heap.HeapMain.main(HeapMain.java:9)
```

-XX:+HeapDumpOnOutOfMemoryError заставила JVM сбросить всю доступную ей информацию о хипе в файл, который она назовет в честь павшего процесса программы от OOM.

### 1.3.2 [Optional] Using jcmd

```
C:\Users\Uliana_Shafatova\Desktop>jcmd 5392 GC.heap_dump myHeapaDoesNotFeelSoGood
5392:
Heap dump file created
```

### 1.3.3 [Optional] Using jmap

```
C:\Users\Uliana_Shafatova\Desktop>jmap -dump:format=b,file=poorBigHeapa.hprof 19468
Heap dump file created
```

jcmd и jmap делают дампы хипы в момент их вызова: jmap делает дампы всех объектов в хипе, а jcmd — только живых, вызывая перед этим полную сборку мусора. jcmd по сути включает в себя функционал jmap и других утилит (jps, jinfo, jstack). Если я правильно поняла, то оно пользуется JVM Invocation API через нативный указатель, после чего существует в JVM как самостоятельный поток и тыркает ее.

## 1.4 Get heap histogram

### 1.4.1 Using jcmd

```
C:\Users\Uliana_Shafatova\Desktop>jcmd 19468 GC.class_histogram
19468:
num      #instances      #bytes  class name (module)
-----
  1:         2393      122360  [B (java.base@11.0.12)
  2:          679       83080  java.lang.Class (java.base@11.0.12)
  3:          924       65696  [Ljava.lang.Object; (java.base@11.0.12)
  4:         2274       54576  java.lang.String (java.base@11.0.12)
  5:         1681       53792  java.util.HashMap$Node (java.base@11.0.12)
  6:           17       42504  [C (java.base@11.0.12)
  7:          311       32392  [Ljava.util.HashMap$Node; (java.base@11.0.12)
  8:         1005       32160  java.util.concurrent.ConcurrentHashMap$Node (java.base@11.0.12)
  9:          324       15552  java.util.HashMap (java.base@11.0.12)
 10:          20       14720  [Ljava.util.concurrent.ConcurrentHashMap$Node; (java.base@11.0.12)
 11:          361       8664   java.lang.module.ModuleDescriptor$Exports (java.base@11.0.12)
 12:          290       6960   java.util.ImmutableCollections$Set12 (java.base@11.0.12)
 13:           62       4960   java.net.URI (java.base@11.0.12)
 14:          135       4320   java.lang.module.ModuleDescriptor$Requires (java.base@11.0.12)
 15:          256       4096   java.lang.Integer (java.base@11.0.12)
 16:          169       4056   java.util.ImmutableCollections$SetN (java.base@11.0.12)
 17:           63       4032   java.net.URL (java.base@11.0.12)
 18:          127       4000   [I (java.base@11.0.12)
 19:           62       3968   java.lang.module.ModuleDescriptor (java.base@11.0.12)
 20:           68       3808   java.lang.Module (java.base@11.0.12)
 21:           62       3472   jdk.internal.module.ModuleReferenceImpl (java.base@11.0.12)
 22:            9       3384   java.lang.Thread (java.base@11.0.12)
 23:          198       3168   java.util.HashSet (java.base@11.0.12)
 24:           33       2904   java.lang.reflect.Method (java.base@11.0.12)
 25:           88       2544   [Ljava.lang.Class; (java.base@11.0.12)
 26:          125       2000   java.util.Collections$UnmodifiableSet (java.base@11.0.12)
```

### 1.4.2 Using jmap

```
C:\Users\Uliana_Shafatova\Desktop>jmap -histo 19468
num      #instances      #bytes  class name (module)
-----
  1:         2393      122360  [B (java.base@11.0.12)
  2:          679       83080  java.lang.Class (java.base@11.0.12)
  3:          924       65696  [Ljava.lang.Object; (java.base@11.0.12)
  4:         2274       54576  java.lang.String (java.base@11.0.12)
  5:         1681       53792  java.util.HashMap$Node (java.base@11.0.12)
  6:           17       42504  [C (java.base@11.0.12)
  7:          311       32392  [Ljava.util.HashMap$Node; (java.base@11.0.12)
  8:         1005       32160  java.util.concurrent.ConcurrentHashMap$Node (java.base@11.0.12)
  9:          324       15552  java.util.HashMap (java.base@11.0.12)
 10:          20       14720  [Ljava.util.concurrent.ConcurrentHashMap$Node; (java.base@11.0.12)
 11:          361       8664   java.lang.module.ModuleDescriptor$Exports (java.base@11.0.12)
 12:          290       6960   java.util.ImmutableCollections$Set12 (java.base@11.0.12)
 13:           62       4960   java.net.URI (java.base@11.0.12)
 14:          135       4320   java.lang.module.ModuleDescriptor$Requires (java.base@11.0.12)
 15:          256       4096   java.lang.Integer (java.base@11.0.12)
 16:          169       4056   java.util.ImmutableCollections$SetN (java.base@11.0.12)
 17:           63       4032   java.net.URL (java.base@11.0.12)
 18:          127       4000   [I (java.base@11.0.12)
 19:           62       3968   java.lang.module.ModuleDescriptor (java.base@11.0.12)
 20:           68       3808   java.lang.Module (java.base@11.0.12)
 21:           62       3472   jdk.internal.module.ModuleReferenceImpl (java.base@11.0.12)
 22:            9       3384   java.lang.Thread (java.base@11.0.12)
 23:          198       3168   java.util.HashSet (java.base@11.0.12)
 24:           33       2904   java.lang.reflect.Method (java.base@11.0.12)
 25:           88       2544   [Ljava.lang.Class; (java.base@11.0.12)
 26:          125       2000   java.util.Collections$UnmodifiableSet (java.base@11.0.12)
 27:           30       1920   java.util.concurrent.ConcurrentHashMap (java.base@11.0.12)
```

JVM пробегает по хипе и считает, сколько памяти занимают экземпляры каждого класса. Далее выводит гистограмму, сортированную по размеру классов.

jcmd при таком вызове произведет полную сборку мусора перед сбором данных для гистограммы класса.

## 1.5 Analyze heap dump

### 1.5.1 Using Java Visual VM

Open retrieved heap dump in jvisualvm

Identify memory leak

The screenshot shows the Java Visual VM interface with a heap dump loaded. The title bar indicates the file is `java_pid22116.hprof`. The main window has a tab labeled `[heapdump] java_pid22116.hprof` and a sub-tab `Summary`.

**Heap Dump Summary:**

Heap		Environment	
Size:	116,458,185 B	System	Windows 10 (10.0)
Classes:	676	Architecture:	amd64 64bit
Instances:	2,474,550	Java Home:	C:\Program Files\Java\jdk-11.0.12
Classloaders:	3	Java Version:	11.0.12
GC Roots:	664	Java Name:	Java HotSpot(TM) 64-Bit Server VM (11.0.12+8-LTS-237, mixed mode)
Objects Pending for Finalization:	0	Java Vendor:	Oracle Corporation
		JVM Uptime:	n/a

**Enabled Modules** [ show ]

**System Properties** [ show ]

**OutOfMemoryError Thread** [ view all ]

This heap dump has been created automatically on an OutOfMemoryError thrown in this thread:

**"main" prio=5 tid=1 RUNNABLE**

**Classes by Number of Instances** [ view all ]

Class	Count	Percentage
<code>byte[]</code>	1,233,065	(49.8%)
<code>java.lang.String</code>	1,232,977	(49.8%)
<code>java.util.HashMap\$Node</code>	1,681	(0.1%)
<code>java.util.concurrent.ConcurrentHashMap\$Node</code>	1,010	(0%)
<code>java.lang.Object[]</code>	925	(0%)

**Classes by Size of Instances** [ view all ]

Class	Size (B)	Percentage
<code>java.lang.Object[]</code>	49,349,264 B	(42.4%)
<code>java.lang.String</code>	35,756,333 B	(30.7%)
<code>byte[]</code>	30,900,311 B	(26.5%)
<code>java.util.HashMap\$Node</code>	73,964 B	(0.1%)
<code>java.util.HashMap\$Node[]</code>	62,144 B	(0.1%)

**Instances by Size** [ view all ]

**Dominators by Retained Size** [ view all ]

Name	Count	Size	Retained
java.lang.Object[]	1 (0%)	49,227,224 B (42.3%)	115,683,944 B (99.3%)
java.lang.Class	600 (0%)	79,200 B (0.1%)	366,742 B (0.3%)
java.lang.Thread	7 (0%)	1,134 B (0%)	3,056 B (0%)
java.lang.OutOfMemoryError	7 (0%)	420 B (0%)	1,007 B (0%)
java.lang.ref.Finalizer\$FinalizerThread	1 (0%)	163 B (0%)	247 B (0%)
jdk.internal.misc.InnocuousThread	1 (0%)	163 B (0%)	179 B (0%)
java.lang.ref.Reference\$ReferenceHandler	1 (0%)	162 B (0%)	246 B (0%)
java.lang.ThreadGroup	2 (0%)	132 B (0%)	357 B (0%)
java.lang.VirtualMachineError	2 (0%)	120 B (0%)	120 B (0%)
java.lang.String	3 (0%)	87 B (0%)	189 B (0%)
java.lang.ref.ReferenceQueue	2 (0%)	80 B (0%)	80 B (0%)
java.lang.NullPointerException	1 (0%)	60 B (0%)	60 B (0%)
java.lang.ArithmeticException	1 (0%)	60 B (0%)	122 B (0%)
java.lang.OutOfMemoryError[]	1 (0%)	56 B (0%)	3,752 B (0%)
jdk.internal.ref.CleanerImpl	1 (0%)	48 B (0%)	216 B (0%)
java.util.ArrayList	1 (0%)	32 B (0%)	32 B (0%)
java.lang.ref.ReferenceQueue\$Lock	2 (0%)	32 B (0%)	32 B (0%)
java.lang.Class[]	1 (0%)	24 B (0%)	24 B (0%)
java.lang.Long	1 (0%)	24 B (0%)	24 B (0%)
com.epam.jmp.mat.heapProcess	1 (0%)	24 B (0%)	24 B (0%)
java.lang.String[]	1 (0%)	24 B (0%)	24 B (0%)

Ну, можно попробовать посмотреть объекты, которые насоздавала программа. Из забавного тут есть ReferenceQueue: очередь фантомных ссылок объектов на удаление. А еще очень большой массив объектов, который не удаляется потому что используется исполняющимся джава-потокком, а потому считается живым и «немусором»:

Name	Count	Size	Retained
java.lang.Object[]	1 (0%)	49,227,224 B (42.3%)	115,683,944 B (99.3%)
java.lang.Object[]#925 [GC root - Java frame] : 6,153,400 items		49,227,224 B (42.3%)	115,683,944 B (99.3%)
<items>			
<references>			
elementData in java.util.ArrayList#1 [GC root - Java frame] : 6,153,400 elements		32 B (0%)	32 B (0%)

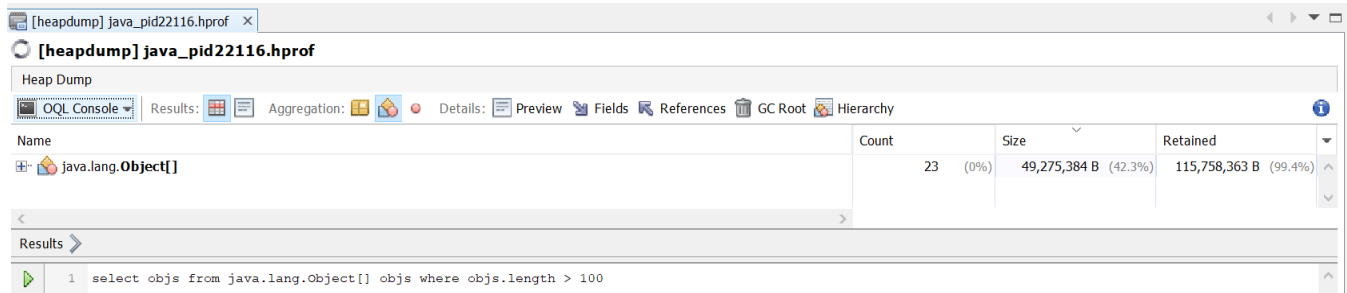
Падение программы вызвало очередное увеличение ArrayList в 1,5 раза после захламления его новой порцией объектов.

## 1.6 OQL

### 1.6.1 Execute OQL in jvisualvm:

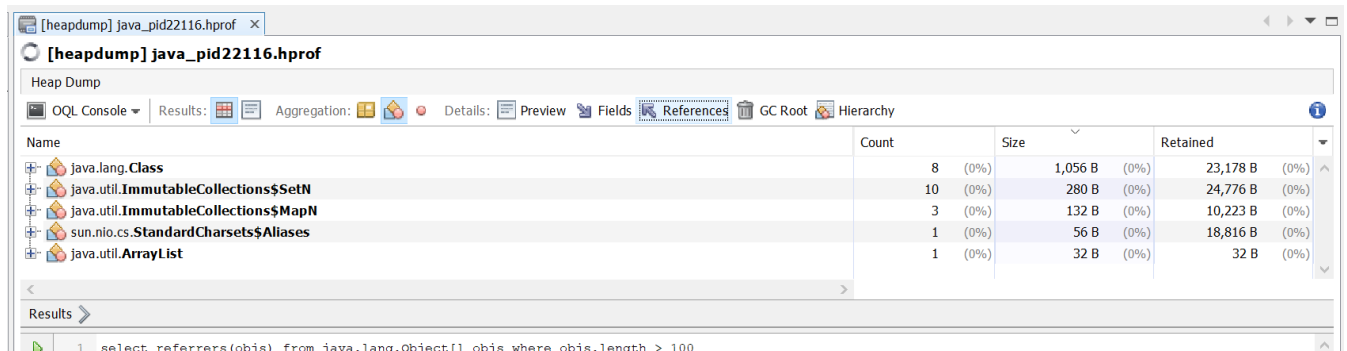
OQL (Object Query Language) – SQL-подобный язык для декларативного поиска объектов по хипе.

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



Тут оно попросило выбрать такие objs типа Object[], что в них будет больше 100 ячеек. Всего было 23 таких массива, один больше другого, так как все пересоздавались под капотом ArrayList, когда ему нужно было разрастись.

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



referrers – объекты, которые хранят в себе ссылку на кого-нибудь из objs типа Object[], у которого больше 100 ячеек. Таким был тот самый ArrayList.

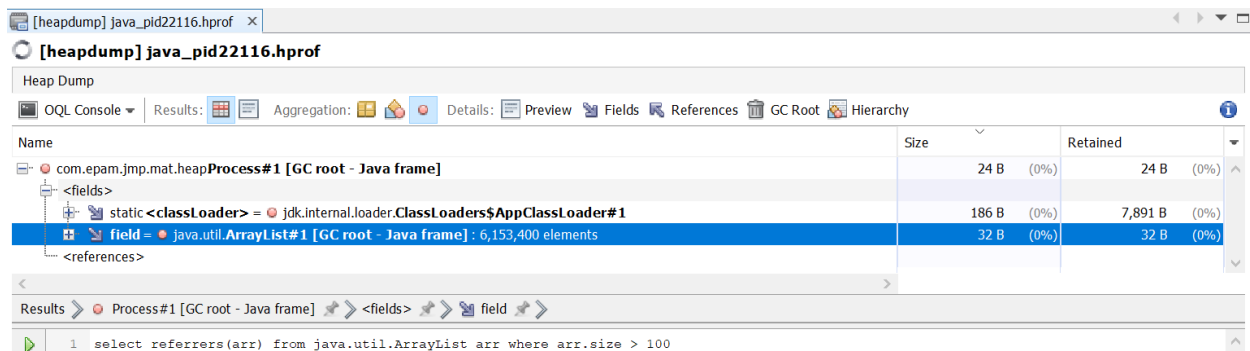
Внутри java.lang.Class есть поле Objects[] signers:

```
public Object[] getSigners()
```

Gets the signers of this class.

Вроде эти ребята представляют информацию о сертификатах, использованных для «подписи» этого кода, чтобы другие пользователи тоже могли использовать этот Class у себя (*это очень дословный перевод стековерфлоува, но вроде поняла*)

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



На один единственный огромный ArrayList ссылался один единственный объект класса Process, который что-то в него записывал.

## 1.6.2 Execute OQL in jhat

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

jhat <head\_dump.hprof>

- 1) select [objs, objs.length] from [Ljava.lang.Object; objs where objs.length > 100
- 2) select referrers(objs) from [Ljava.lang.Object; objs where objs.length > 100
- 3) select referrers(arr) from java.util.ArrayList arr where arr.size > 100

VPN не помог обойти Oracle, чтобы скачать 8 JDK :C

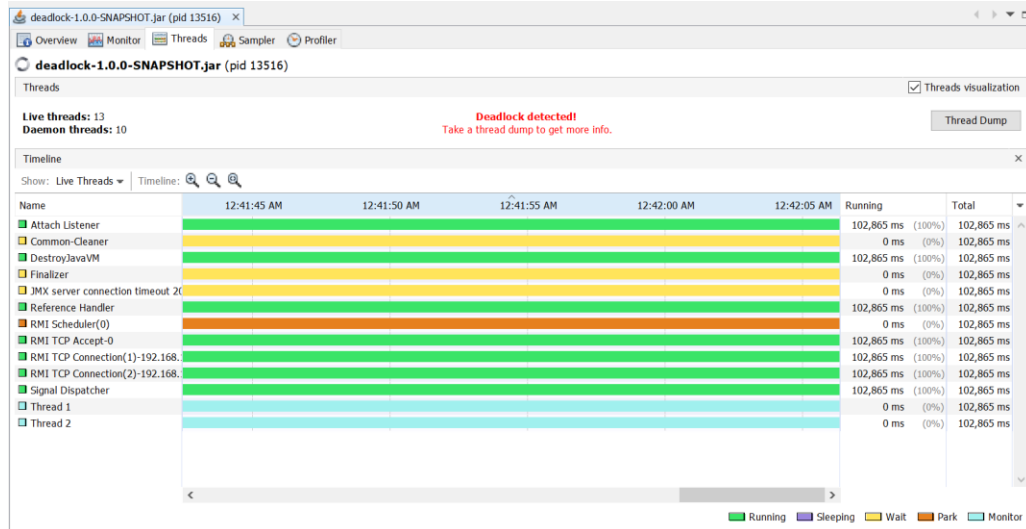
Оно будет делать то же самое, но с другим синтаксисом: в (1) оно просит явно указать в выборке, что мне нужно, массивы непримитивов здесь начинаются с [L.



## Task 2:

### 2.1 Get deadlock

java -jar deadlock-1.0.0-SNAPSHOT.jar



Running – поток в рабочем состоянии

Sleeping – поток в состоянии Thread.sleep()

Wait – поток ожидает монитора объекта

Park – если правильно поняла, то поток завязан на синхронизирующий объект и ожидает срабатывания его условия, когда тот допускает его к действию (стопится через LockSupport.parkUntil() или LockSupport.parkTimeout())

Monitor – поток захватил монитор какого-то ресурса

## 2.2 Get thread dump

### 2.2.1 jstack

jstack -l <pid>

```
Full thread dump Java HotSpot(TM) 64-Bit Server VM (11.0.12+8-LTS-237 mixed mode):

Threads class SMR info:
 java_thread_list=0x000002577fe084b0, length=12, elements={
 0x000002577fbd5000, 0x000002577fbd8800, 0x000002577fc36000, 0x000002577fc37000,
 0x000002577fc38000, 0x000002577fc39800, 0x000002577fc40800, 0x000002577fc44800,
 0x000002577fde3000, 0x000002577fe65800, 0x000002577fe6a800, 0x000002577f691b1800
 }

"Reference Handler" #2 daemon prio=10 os_prio=2 cpu=0.00ms elapsed=23.91s tid=0x000002577fbd5000 nid=0x5eb0 waiting on condition [0x0000006737eff000]
 java.lang.Thread.State: RUNNABLE
   at java.lang.ref.Reference.waitForReferencePendingList(java.base@11.0.12/Native Method)
   at java.lang.ref.Reference.processPendingReferences(java.base@11.0.12/Reference.java:241)
   at java.lang.ref.Reference$ReferenceHandler.run(java.base@11.0.12/Reference.java:213)

  Locked ownable synchronizers:
    - None

"Finalizer" #3 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=23.91s tid=0x000002577fbd8800 nid=0x54c0 in Object.wait() [0x0000006737fff000]
 java.lang.Thread.State: WAITING (on object monitor)
   at java.lang.Object.wait(java.base@11.0.12/Native Method)
   - waiting on <0x0000000627608fa8> (a java.lang.ref.ReferenceQueue$Lock)
   at java.lang.ref.ReferenceQueue.remove(java.base@11.0.12/ReferenceQueue.java:155)
   - waiting to re-lock in wait() <0x0000000627608fa8> (a java.lang.ref.ReferenceQueue$Lock)
   at java.lang.ref.ReferenceQueue.remove(java.base@11.0.12/ReferenceQueue.java:176)
   at java.lang.ref.Finalizer$FinalizerThread.run(java.base@11.0.12/Finalizer.java:170)

  Locked ownable synchronizers:
    - None

"Signal Dispatcher" #4 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=23.89s tid=0x000002577fc36000 nid=0x1af0 runnable [0x0000000000000000]
 java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None

"Attach Listener" #5 daemon prio=5 os_prio=2 cpu=0.00ms elapsed=23.89s tid=0x000002577fc37000 nid=0x5a0c waiting on condition [0x0000000000000000]
 java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None

"Service Thread" #6 daemon prio=9 os_prio=0 cpu=0.00ms elapsed=23.89s tid=0x000002577fc38000 nid=0x1920 runnable [0x0000000000000000]
 java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None

"C2 CompilerThread0" #7 daemon prio=9 os_prio=2 cpu=31.25ms elapsed=23.89s tid=0x000002577fc39800 nid=0xcd0 waiting on condition [0x0000000000000000]
 java.lang.Thread.State: RUNNABLE
  No compile task

  Locked ownable synchronizers:
    - None

"C1 CompilerThread0" #10 daemon prio=9 os_prio=2 cpu=15.63ms elapsed=23.89s tid=0x000002577fc40800 nid=0x44a8 waiting on condition [0x0000000000000000]
 java.lang.Thread.State: RUNNABLE
  No compile task

  Locked ownable synchronizers:
    - None

"Sweeper thread" #11 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=23.89s tid=0x000002577fc44800 nid=0x5be0 runnable [0x0000000000000000]
 java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None
```

```

"Common-Cleaner" #12 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=23.85s tid=0x000002577fde3000 nid=0x60ec in Object.wait() [0x00000067387fe000]
  java.lang.Thread.State: TIMED_WAITING (on object monitor)
    at java.lang.Object.wait(java.base@11.0.12/Native Method)
    - waiting on <0x0000000627796c50> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(java.base@11.0.12/ReferenceQueue.java:155)
    - waiting to re-lock in wait() <0x0000000627796c50> (a java.lang.ref.ReferenceQueue$Lock)
    at jdk.internal.ref.CleanerImpl.run(java.base@11.0.12/CleanerImpl.java:148)
    at java.lang.Thread.run(java.base@11.0.12/Thread.java:834)
    at jdk.internal.misc.InnocuousThread.run(java.base@11.0.12/InnocuousThread.java:134)

  Locked ownable synchronizers:
    - None

"Thread 1" #13 prio=5 os_prio=0 cpu=0.00ms elapsed=23.82s tid=0x000002577fe65800 nid=0x5994 waiting for monitor entry [0x00000067388ff000]
  java.lang.Thread.State: BLOCKED (on object monitor)
    at com.epam.jsp.mat.deadlock.SimulateDeadLock.method1(SimulateDeadLock.java:24)
    - waiting to lock <0x00000006277b9198> (a java.lang.Object)
    - locked <0x00000006277b9198> (a java.lang.Object)
    at com.epam.jsp.mat.deadlock.DeadLockMain$1.run(DeadLockMain.java:11)

  Locked ownable synchronizers:
    - None

"Thread 2" #14 prio=5 os_prio=0 cpu=0.00ms elapsed=23.82s tid=0x000002577fe6a800 nid=0x66dc waiting for monitor entry [0x00000067389ff000]
  java.lang.Thread.State: BLOCKED (on object monitor)
    at com.epam.jsp.mat.deadlock.SimulateDeadLock.method2(SimulateDeadLock.java:44)
    - waiting to lock <0x00000006277b9198> (a java.lang.Object)
    - locked <0x00000006277b9198> (a java.lang.Object)
    at com.epam.jsp.mat.deadlock.DeadLockMain$2.run(DeadLockMain.java:18)

  Locked ownable synchronizers:
    - None

```

```

DestroyJavaVM" #15 prio=5 os_prio=0 cpu=125.00ms elapsed=23.82s tid=0x00000257691b1800 nid=0x528c waiting on condition [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None

VM Thread" os_prio=2 cpu=0.00ms elapsed=23.92s tid=0x000002577fbc1800 nid=0x4e94 runnable

GC Thread#0" os_prio=2 cpu=0.00ms elapsed=23.94s tid=0x00000257691c9800 nid=0x68ec runnable

G1 Main Marker" os_prio=2 cpu=0.00ms elapsed=23.94s tid=0x0000025769246800 nid=0x58d4 runnable

G1 Conc#0" os_prio=2 cpu=0.00ms elapsed=23.94s tid=0x0000025769248800 nid=0x6510 runnable

G1 Refine#0" os_prio=2 cpu=0.00ms elapsed=23.93s tid=0x000002577f20b000 nid=0x6bc8 runnable

G1 Young RemSet Sampling" os_prio=2 cpu=0.00ms elapsed=23.93s tid=0x000002577f20c000 nid=0x47d0 runnable
VM Periodic Task Thread" os_prio=2 cpu=0.00ms elapsed=23.85s tid=0x000002577fde0000 nid=0x6be0 waiting on condition

NI global refs: 8, weak refs: 0

```

```

Found one Java-level deadlock:
=====
"Thread 1":
  waiting to lock monitor 0x000002577fbc9c80 (object 0x00000006277b9198, a java.lang.Object),
  which is held by "Thread 2"
"Thread 2":
  waiting to lock monitor 0x000002577fbc7b80 (object 0x00000006277b9188, a java.lang.Object),
  which is held by "Thread 1"

```

Java stack information for the threads listed above:

```

=====
"Thread 1":
  at com.epam.jsp.mat.deadlock.SimulateDeadLock.method1(SimulateDeadLock.java:24)
  - waiting to lock <0x00000006277b9198> (a java.lang.Object)
  - locked <0x00000006277b9188> (a java.lang.Object)
  at com.epam.jsp.mat.deadlock.DeadLockMain$1.run(DeadLockMain.java:11)
"Thread 2":
  at com.epam.jsp.mat.deadlock.SimulateDeadLock.method2(SimulateDeadLock.java:44)
  - waiting to lock <0x00000006277b9188> (a java.lang.Object)
  - locked <0x00000006277b9198> (a java.lang.Object)
  at com.epam.jsp.mat.deadlock.DeadLockMain$2.run(DeadLockMain.java:18)

```

Found 1 deadlock.

jstack цепляется к Java-процессу по его pid и показывает информацию о 12 обнаруженных потоках: имя (name), приоритет выполнения (prior), уникальный идентификатор потока в JVM (tid), уникальный идентификатор потока в ОС (nid), текущее состояние (state) потока и его стектрейс.

Если правильно понимаю, то поток ReferenceHandler подчищает фантомные ссылки объектов, оставшиеся после потока Finalizer; SignalDispatcher нужен для обработки нажатия определенных клавиш ОС; AttachListener – предоставляет динамическую привязку к процессу в JVM (например, тот же jmap им должен пользоваться); Service Thread ждет каких-то событий JVM для обработки (когда остается мало памяти, уведомления по процессу сборки мусора и т.д.); Sweeper Thread вроде удаляет байт-код полностью скомпилированных методов после C2? ; VM Thread запускает те самые операции по созданию дампов хипы или потоков, которые в этот момент должны быть в «безопасной позиции» (приостановлены, синхронизированы локом или монитором).

C1 и C2 – потоки разных JIT-компиляторов байт-кода JVM в нативный, C1 работает быстрее, но выдает менее оптимизированный код, C2 – наоборот, работает медленнее, но выдает более быстрый код.

В конце дампа видно отдельные потоки сборщика мусора G1, который копается пока что в одном регионе памяти.

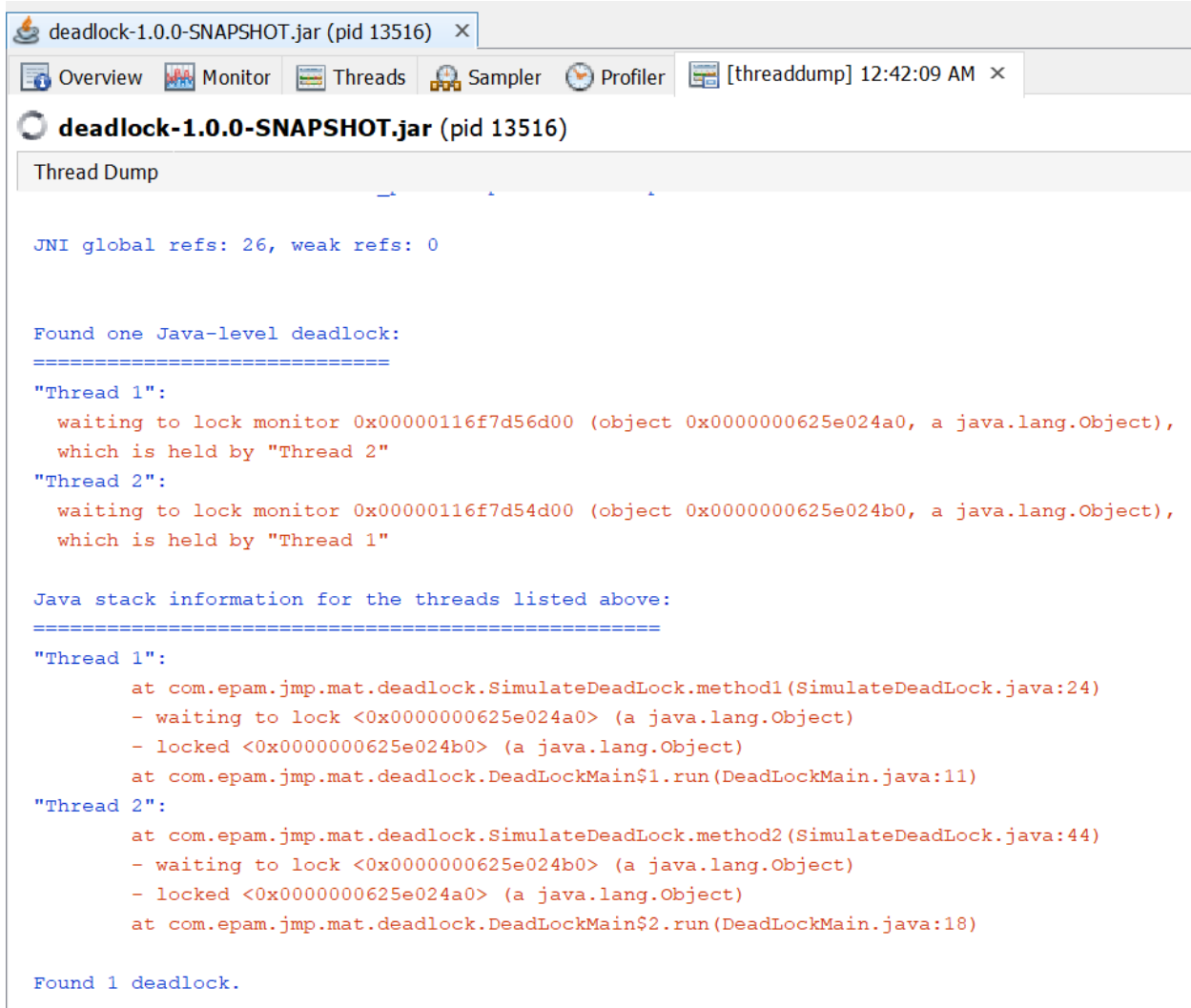
jstack нашло один дедлок и объяснило его в конце: какие потоки сцепились за ресурсы между собой и информацию о них.

### **2.2.1 kill -3**

kill -3 <pid>

**У меня не Linux :<**

## 2.2.2 jvisualvm



```
deadlock-1.0.0-SNAPSHOT.jar (pid 13516) x
Overview Monitor Threads Sampler Profiler [threaddump] 12:42:09 AM x

deadlock-1.0.0-SNAPSHOT.jar (pid 13516)
Thread Dump

JNI global refs: 26, weak refs: 0

Found one Java-level deadlock:
=====
"Thread 1":
  waiting to lock monitor 0x00000116f7d56d00 (object 0x0000000625e024a0, a java.lang.Object),
  which is held by "Thread 2"
"Thread 2":
  waiting to lock monitor 0x00000116f7d54d00 (object 0x0000000625e024b0, a java.lang.Object),
  which is held by "Thread 1"

Java stack information for the threads listed above:
=====
"Thread 1":
  at com.epam.jump.mat.deadlock.SimulateDeadLock.method1(SimulateDeadLock.java:24)
  - waiting to lock <0x0000000625e024a0> (a java.lang.Object)
  - locked <0x0000000625e024b0> (a java.lang.Object)
  at com.epam.jump.mat.deadlock.DeadLockMain$1.run(DeadLockMain.java:11)
"Thread 2":
  at com.epam.jump.mat.deadlock.SimulateDeadLock.method2(SimulateDeadLock.java:44)
  - waiting to lock <0x0000000625e024b0> (a java.lang.Object)
  - locked <0x0000000625e024a0> (a java.lang.Object)
  at com.epam.jump.mat.deadlock.DeadLockMain$2.run(DeadLockMain.java:18)

Found 1 deadlock.
```

Все то же, но цветное

## 2.2.3 Windows (Ctrl + Break)

```
Found one Java-level deadlock:
=====
"Thread 1":
  waiting to lock monitor 0x00000205ee498580 (object 0x00000006277b8ff8, a java.lang.Object),
  which is held by "Thread 2"
"Thread 2":
  waiting to lock monitor 0x00000205ee496580 (object 0x00000006277b8fe8, a java.lang.Object),
  which is held by "Thread 1"

Java stack information for the threads listed above:
=====
"Thread 1":
  at com.epam.jsp.mat.deadlock.SimulateDeadLock.method1(SimulateDeadLock.java:24)
  - waiting to lock <0x00000006277b8ff8> (a java.lang.Object)
  - locked <0x00000006277b8fe8> (a java.lang.Object)
  at com.epam.jsp.mat.deadlock.DeadLockMain$1.run(DeadLockMain.java:11)
"Thread 2":
  at com.epam.jsp.mat.deadlock.SimulateDeadLock.method2(SimulateDeadLock.java:44)
  - waiting to lock <0x00000006277b8fe8> (a java.lang.Object)
  - locked <0x00000006277b8ff8> (a java.lang.Object)
  at com.epam.jsp.mat.deadlock.DeadLockMain$2.run(DeadLockMain.java:18)

Found 1 deadlock.

Heap
garbage-first heap   total 518144K, used 2048K [0x0000000607e00000, 0x0000000800000000)
region size 2048K, 2 young (4096K), 0 survivors (0K)
Metaspace            used 4705K, capacity 4740K, committed 4864K, reserved 1056768K
class space          used 418K, capacity 428K, committed 512K, reserved 1048576K
```

## 2.2.4 jcmd

jcmd <pid> Thread.print

```
Found one Java-level deadlock:
=====
"Thread 1":
  waiting to lock monitor 0x0000014a7dd45680 (object 0x00000006277b9070, a java.lang.Object),
  which is held by "Thread 2"
"Thread 2":
  waiting to lock monitor 0x0000014a7dd43580 (object 0x00000006277b9060, a java.lang.Object),
  which is held by "Thread 1"

Java stack information for the threads listed above:
=====
"Thread 1":
  at com.epam.jsp.mat.deadlock.SimulateDeadLock.method1(SimulateDeadLock.java:24)
  - waiting to lock <0x00000006277b9070> (a java.lang.Object)
  - locked <0x00000006277b9060> (a java.lang.Object)
  at com.epam.jsp.mat.deadlock.DeadLockMain$1.run(DeadLockMain.java:11)
"Thread 2":
  at com.epam.jsp.mat.deadlock.SimulateDeadLock.method2(SimulateDeadLock.java:44)
  - waiting to lock <0x00000006277b9060> (a java.lang.Object)
  - locked <0x00000006277b9070> (a java.lang.Object)
  at com.epam.jsp.mat.deadlock.DeadLockMain$2.run(DeadLockMain.java:18)

Found 1 deadlock.
```

### Task 3:

## Remote JVM profiling

### 3.1 Using JMX Technology

For insecure remote connection use parameters:

java -jar

-Dcom.sun.management.jmxremote (1)

-Dcom.sun.management.jmxremote.port=7890 (2)

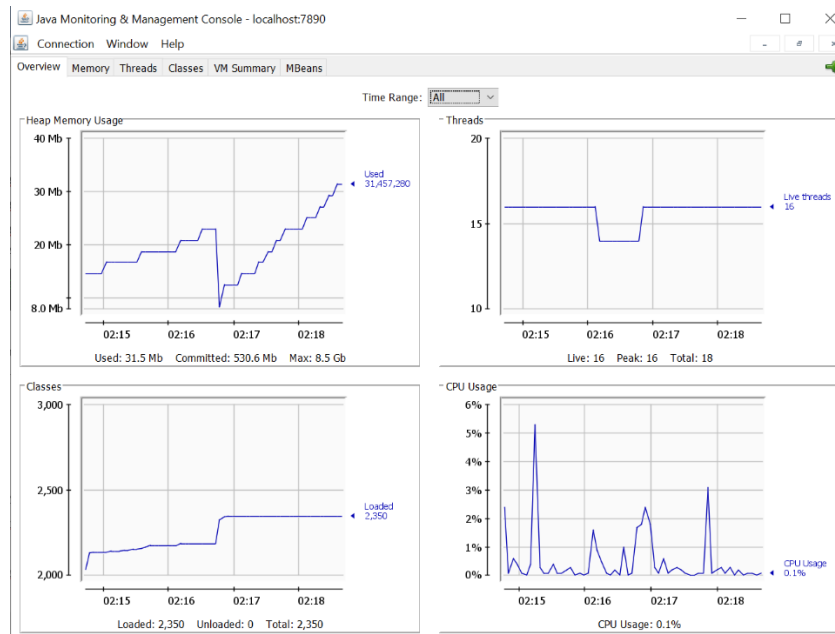
-Dcom.sun.management.jmxremote.authenticate=false (3)

-Dcom.sun.management.jmxremote.ssl=false (4)

simple-1.0.0-SNAPSHOT.jar

```
C:\Users\Uliana_Shafatova\Desktop>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
Press any key to proceed
```

Приложение запустилось с JMX RMI (Remote Method Invocation protocol) на указанном порту (2) без защиты (3 и 4). RMI позволяет использовать процессу на одном хосте взаимодействовать с другим процессом с другого хоста. RMI должно поддерживаться каждой имплементацией JMX клиента. «Удаленным» клиентом будет выступать JConsole, запущенный как «jconsole localhost:7890». Унифицированного порта RMI нет по умолчанию по соображениям безопасности. Вообще и приложение, и JConsole в данном случае запущены на одном хосте, поэтому он не специализируется, но, если правильно поняла, то это можно задать через -Djava.rmi.server.hostname=<ip хоста> для приложения. После подключения оно работает по такому же принципу с MBeans, что и VisualVM.



## Task 4:

### Inspect a Flight Recording

#### 4.1 Execute JVM with two special parameters:

java

-jar

-Xmx100m

-XX:+UnlockCommercialFeatures ← Оно мне не нужно, у меня 11 JDK

-XX:+FlightRecorder

**-XX:StartFlightRecording=dumpsonexit=true,filename=flight.jfr**

heap-1.0.0-SNAPSHOT.jar

```
C:\Users\Uliana_Shafatova\Desktop>java -jar -Xmx100m -XX:+FlightRecorder -XX:StartFlightRecording=dumpsonexit=true,filename=flight.jfr heap-1.0.0-SNAPSHOT.jar
Started recording 1. No limit specified, using maxsize=250MB as default.
```

```
Use jcmd 9492 JFR.dump name=1 to copy recording data to file.
Press any key to proceed
```

```
C:\Windows\system32\cmd.exe
```

```
C:\Users\Uliana_Shafatova\Desktop>jcmd 9492 JFR.dump name=1
```

```
9492:
```

```
Dumped recording "1", 334.3 kB written to:
```

```
C:\Users\Uliana_Shafatova\Desktop\flight.jfr
```

```
C:\Users\Uliana_Shafatova\Desktop>jcmd 9492 JFR.dump name=1
```

```
9492:
```

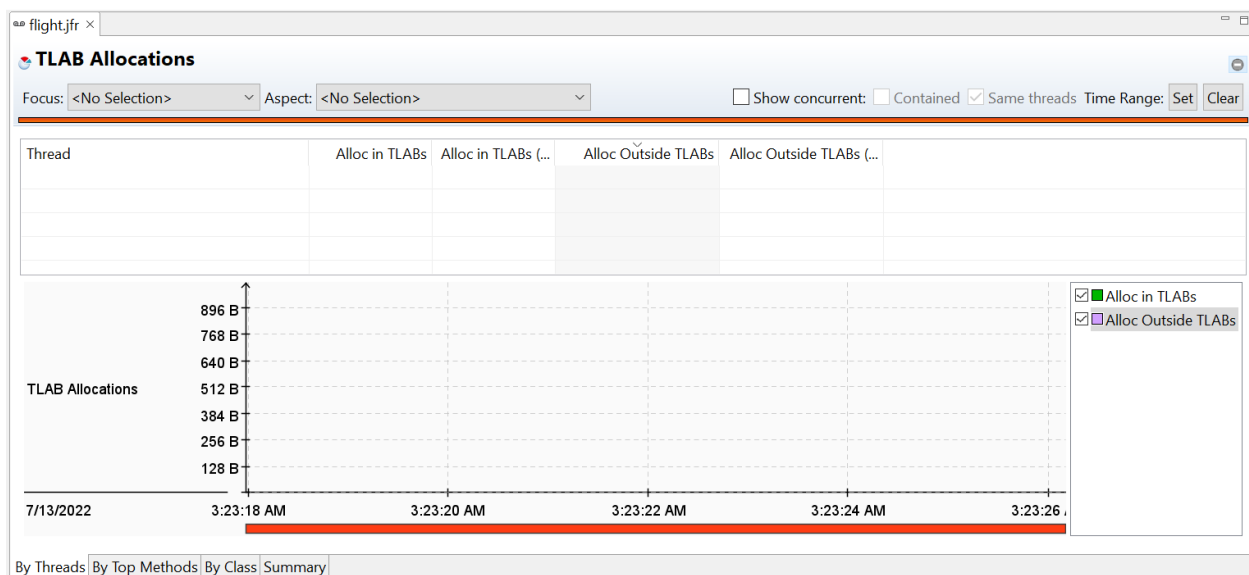
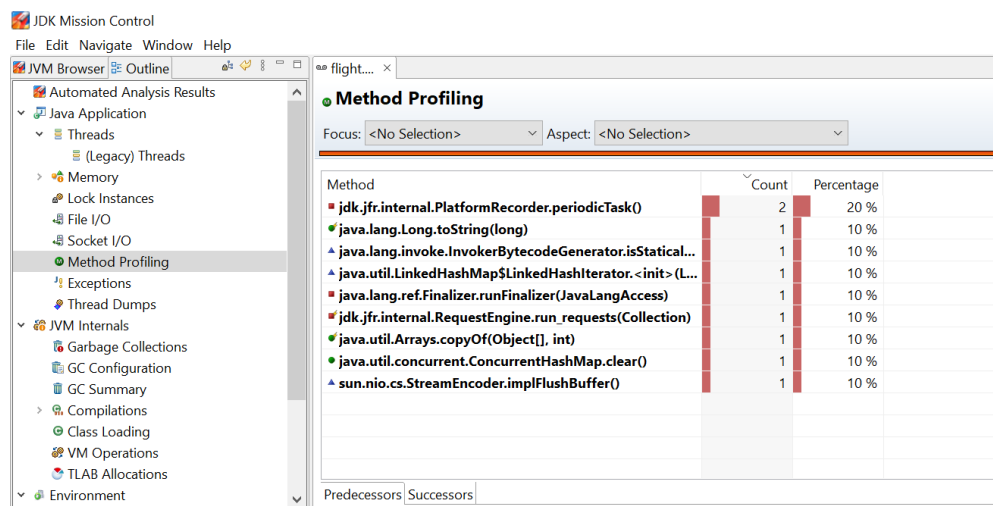
```
Dumped recording "1", 578.6 kB written to:
```

```
C:\Users\Uliana_Shafatova\Desktop\flight.jfr
```



Java Flight Recorder нужно для диагностики и профилирования работающего приложения.

Указанные при запуске приложения настройки (должны были включить фичи коммерческой лицензии для использования JFR, актуально до 12 Джавы, потом перешло в 11, так как она LTS), саму возможность JFR и параметры его работы вместе с приложением, по истечении которой он запишет все в файл flight.jfr, который позже можно будет посмотреть в, например, Java Mission Control.



TLAB (Thread Local Allocated Buffer) – кусочки хипы, на которые JVM ее порезала и отдала потокам, чтобы они локально закидывали туда новые объекты. JFR не пишет информацию о создании каждого объекта в процессе, но когда заканчивается место в каком-либо TLAB'е, оно фиксирует, при создании какого объекта класса это происходит, но это позволяет собрать картину при длительной работе приложения, наверное, тут оно ничего не успело понять тоже.

**Event Browser**

Focus: <No Selection> Aspect: <No Selection> ☐ Show concurrent: ☐

**Event Types Tree**

- Search the tree
- Flight Recorder 285
  - Data Loss 0
  - Flight Recording 1
  - Recording Reason 0
  - Recording Setting 284
- Java Application 361
  - Statistics 85
    - Class Loader Statistics 31
    - Class Loading Statistics 8

Event Thread	Parent Java Thread	New Java Thread
main		main
C2 CompilerThread1	C1 CompilerThread0	C2 CompilerThread1
C2 CompilerThread2	C1 CompilerThread0	C2 CompilerThread2
RMI TCP Accept-0	Attach Listener	RMI TCP Accept-0
C2 CompilerThread1	C2 CompilerThread0	C2 CompilerThread1
DestroyJavaVM		DestroyJavaVM
Logging-Cleaner	DestroyJavaVM	Logging-Cleaner
JFR: Shutdown Hook	DestroyJavaVM	JFR: Shutdown Hook

**Stack Trace**

Stack Trace	Count	Percentage
void java.lang.ApplicationShutdownHooks.runHooks()	2	66.7 %
void java.lang.ApplicationShutdownHooks\$1.run()	2	66.7 %
void java.lang.Shutdown.runHooks()	2	66.7 %
void java.lang.Shutdown.shutdown()	2	66.7 %

## 4.2 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
```

```
22348 heap-1.0.0-SNAPSHOT.jar -Xmx100m
C:\Users\Uliana_Shafatova\Desktop>jcmd 22348 JFR.start name=heap_recording filename=flight-1111.jfr dumponexit=true
22348:
Started recording 1. No limit specified, using maxsize=250MB as default.
Use jcmd 22348 JFR.dump name=heap_recording to copy recording data to file.
```

А так можно запускать JFR, управляя им через jcmd, если, например, приложение уже в рабочем состоянии.

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

The screenshot displays the JDK Mission Control (JMC) application. The left sidebar shows a tree view with the following structure:

- JVM Browser
- Outline
- [11.0.12] The JVM Running Mission Control
- [11.0.12] VisualVM (17052)
- [11.0.12] heap-1.0.0-SNAPSHOT.jar (4020)
  - MBean Server
  - JMC Agent
  - Flight Recorder

The main panel shows the 'System' tab for the selected JVM. It contains three sections:

- Server Information**

Category	Value
Connection	[11.0.12] heap-1.0.0-SNAPSHOT.jar (4020) on J...
Operating System	Windows 10 10.0
OS Architecture	amd64
Number of Processors	0
- JVM Statistics**

Name	Value	Description	Object Name
Currently Loaded	2,539	The number of classes...	java.lang:type=...
Uptime	1 min ...	The uptime of the JVM.	java.lang:type=...
- System Properties**

Key	Value
sun.desktop	windows
awt.toolkit	sun.awt.windows.WToolkit
java.specification.versi...	11
sun.arch.data.model	amd64

At the bottom, there is a tabbed interface with the following tabs: Overview, MBean Browser, Triggers, System (selected), Memory, Threads, and Diagnostic Commands.

(Я не совсем поняла, что от меня тут требовалось, я просто потыкала и посмотрела :д)

```
jinfo <pid>
```

20