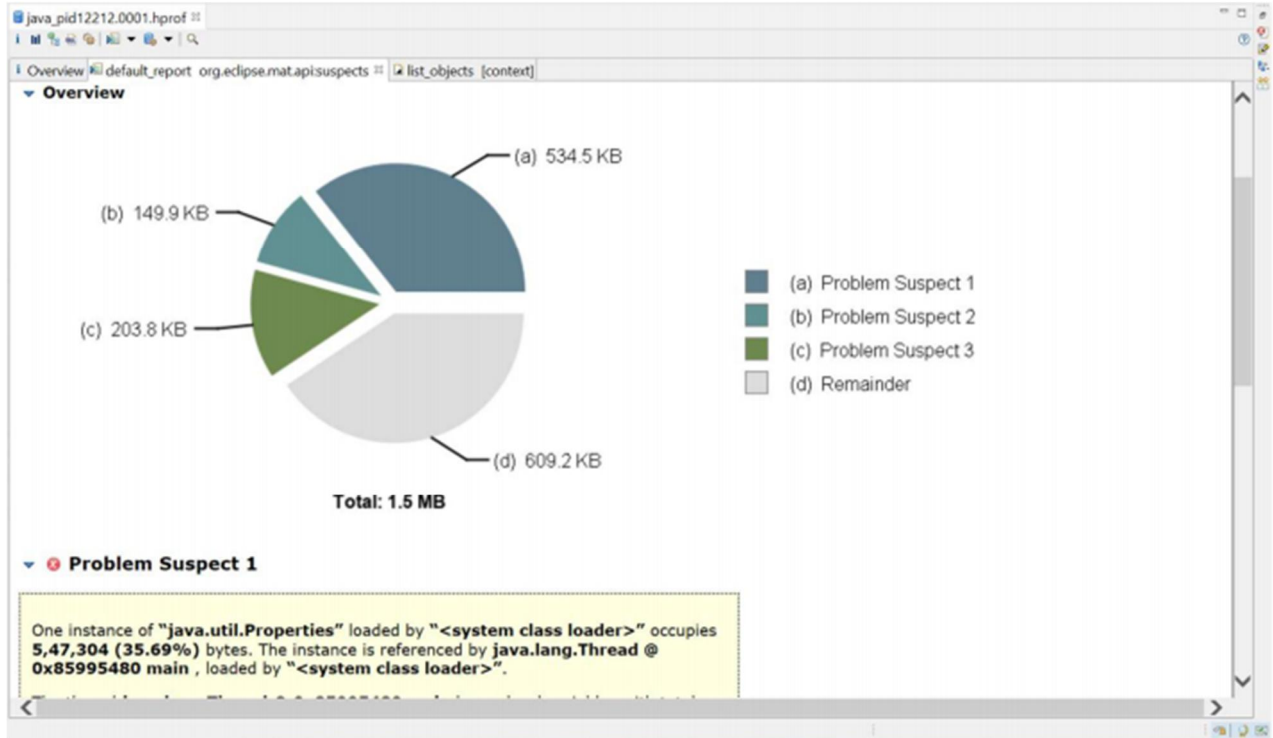
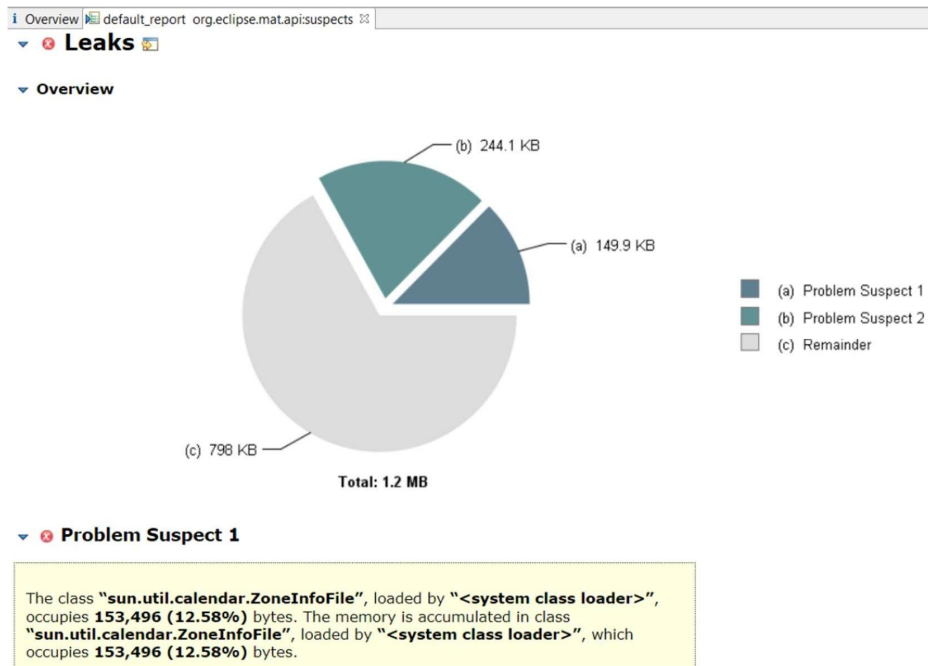


HashMap WithoutEqualAndHash pdf file

Problem: Inefficient memory consumption due to missing implementation of equals and hashCode function in Key.java class, hence using Object class for generating hashcode values for same entries.

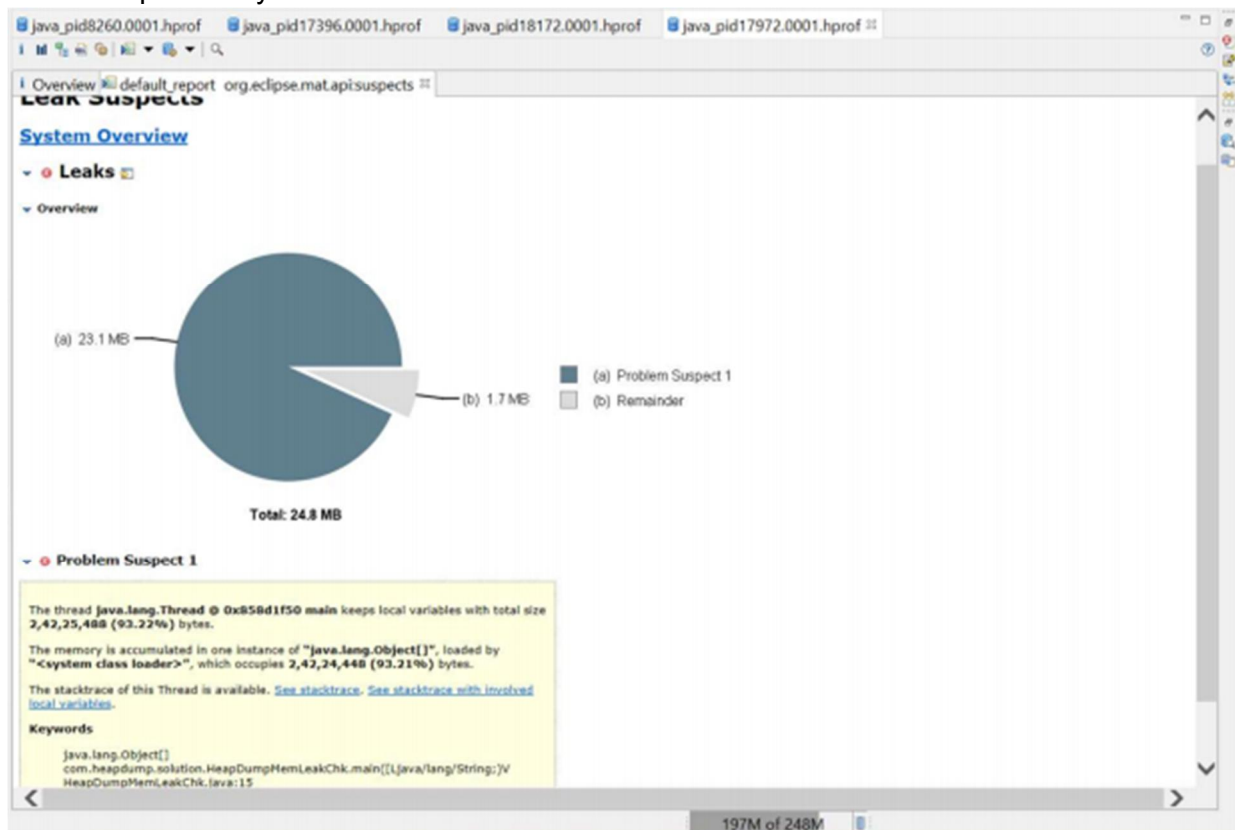


Solution: HashMap WithoutEqualAndHashcode Optimised Version



HeapDumpMemLeakChk pdf file

Problem: Unused ArrayList containing duplicate instances of Leak.java class lying in JVM heap memory.



Solution: Remove unused leak ArrayList instance or Make ObjectForLeak.java class singleton

OutOfMemoryHeapDump pdf file

Problem: Adding new instances of ObjectForLeak.java class in an infinite loop.

The screenshot displays the IntelliJ IDEA interface. The top pane shows a heap dump analysis for the application 'com.heapdump.analysis.OutOfMemoryHeapDump (pid 19852)'. The 'Classes' tab is selected, showing a list of classes and their memory usage. The class 'com.heapdump.solution.ObjectForLeak' is highlighted, showing 70,091,071 instances (99.9%) and a size of 1,121,457,136 (44.4%).

The bottom pane shows the source code of the 'OutOfMemoryHeapDump' class. The code is as follows:

```
public class OutOfMemoryHeapDump {  
    public static void main(String[] args) {  
        List<com.heapdump.solution.ObjectForLeak> leak = new ArrayList<>();  
        try {  
            Thread.sleep(15000);  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
        while(true) {  
            leak.add(new ObjectForLeak());  
        }  
    }  
}
```

The bottom pane also shows the console output, which includes the following error message:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  
Process finished with exit code 1
```

Solution : Put the break in while loop

```
1 package com.heapdump.solution;
2
3 import ...
4
5
6 public class OutOfMemoryHeapDump {
7
8     public static void main(String[] args) {
9         List<com.heapdump.solution.ObjectForLeak> leak = new ArrayList<>();
10
11         while(true) {
12             leak.add(new ObjectForLeak());
13             break;
14         }
15     }
16 }
```

IFMemoryHeapDump (1) ×

vaagent:C:\Apps\JetBrains\IntelliJ IDEA 2021.1.2\lib\idea_rt.jar=61163:C:\Apps\JetBrains\IntelliJ IDEA 2021.1

StaticObjectReference pdf

Problem : Unused double object :

Applications ×

com.heapdump.analysis.StaticObjectReference (pid 14400) ×

Overview Monitor Threads Sampler [HeapDump] 7:01:34 PM ×

com.heapdump.analysis.StaticObjectReference (pid 14400)

Heap Dump

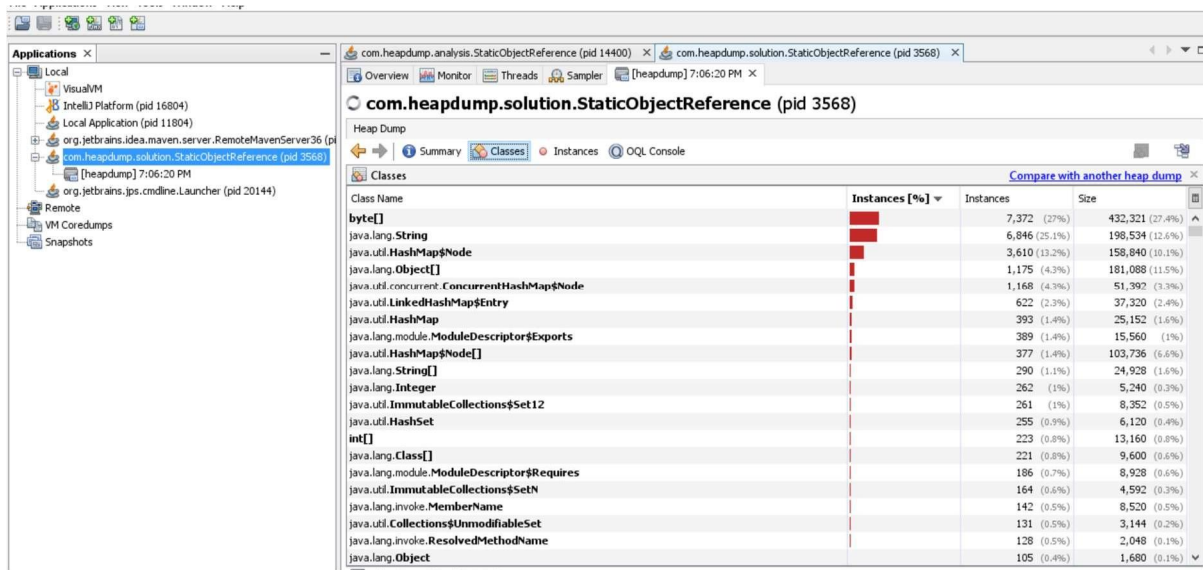
Summary Classes Instances OQL Console

Classes

Compare with another heap dump ×

Class Name	Instances [%]	Instances	Size
java.lang.Double	97.3%	1,000,001	24,000,024 (71.5%)
byte[]	0.7%	7,373	432,374 (1.3%)
java.lang.String	0.7%	6,847	198,563 (0.6%)
java.util.HashMap\$Node	0.4%	3,610	158,840 (0.5%)
java.lang.Object[]	0.1%	1,176	8,181,112 (24.4%)
java.util.concurrent.ConcurrentHashMap\$Node	0.1%	1,168	51,392 (0.2%)
java.util.LinkedHashMap\$Entry	0.1%	622	37,320 (0.1%)
java.util.HashMap	0%	393	25,152 (0.1%)
java.lang.module.ModuleDescriptor\$Exports	0%	389	15,560 (0%)
java.util.HashMap\$Node[]	0%	377	103,736 (0.3%)
java.lang.String[]	0%	290	24,928 (0.1%)
java.lang.Integer	0%	262	5,240 (0%)
java.util.ImmutableCollections\$Set12	0%	261	8,352 (0%)
java.util.HashSet	0%	255	6,120 (0%)
int[]	0%	223	13,160 (0%)
java.lang.Class[]	0%	221	9,600 (0%)
java.lang.module.ModuleDescriptor\$Requires	0%	186	8,928 (0%)
java.util.ImmutableCollections\$SetN	0%	164	4,592 (0%)
java.lang.invoke.MemberName	0%	142	8,520 (0%)
java.util.Collections\$UnmodifiableSet	0%	131	3,144 (0%)
java.lang.invoke.ResolvedMethodName	0%	128	2,048 (0%)

Solution: unreferenced the list object



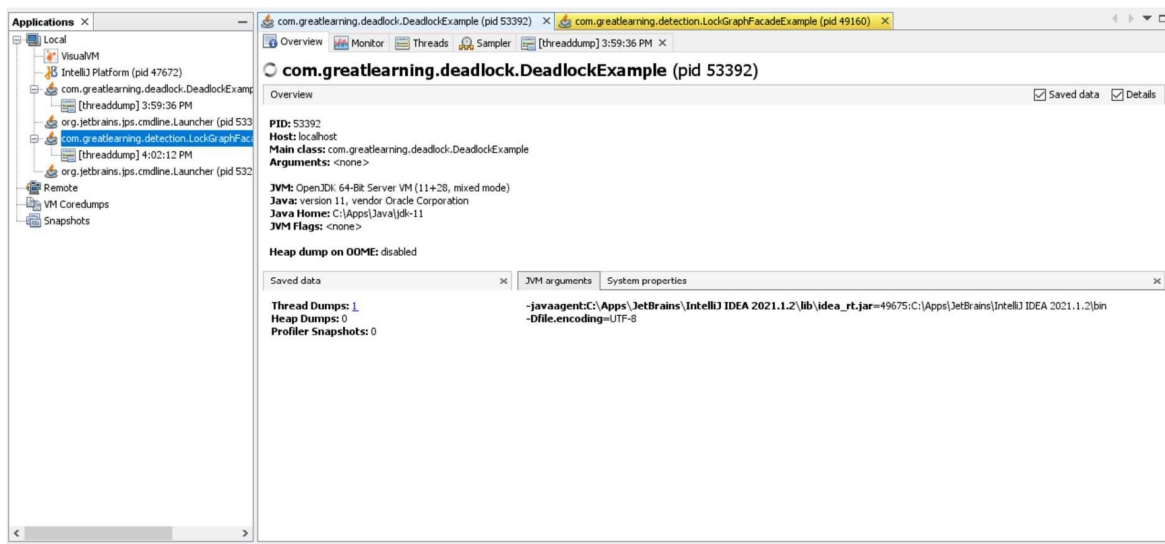
Unclosed DB Connection:

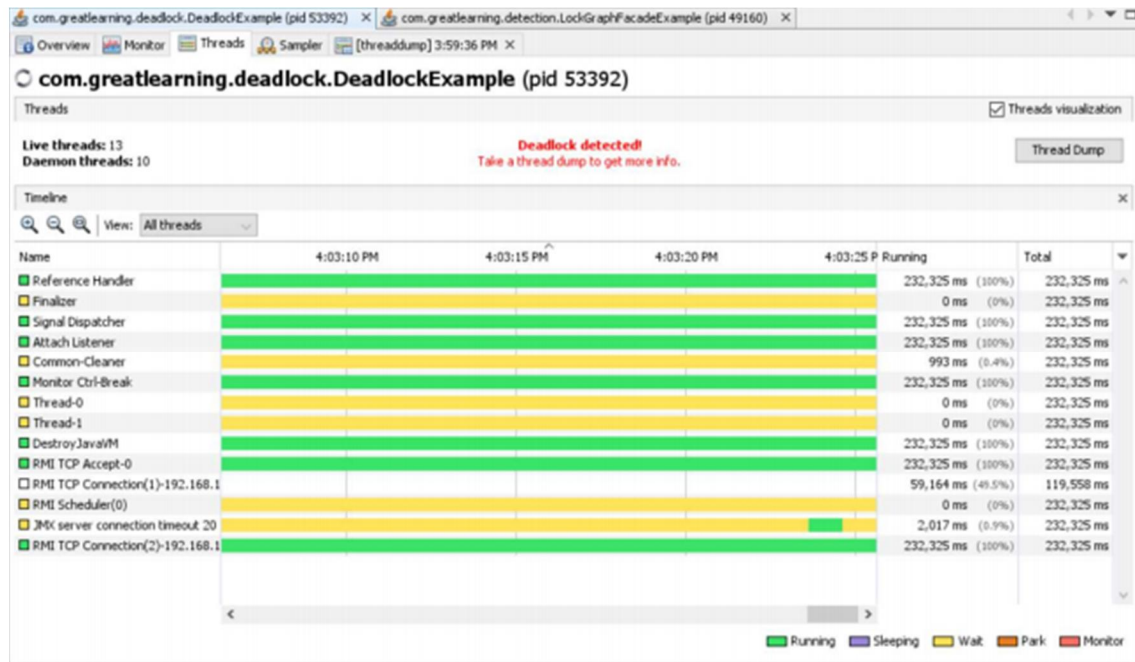
Problem: Connection not closed after

opening **Solution:** Close connection after opening

1. You are required, to generate the deadlock report using JVisualVM and share the proper Screenshots.
2. Refactor the code in order to resolve the Deadlock.

DeadLock screenshot using jvisualVm





com.greatlearning.deadlock.DeadlockExample (pid 53392)

Thread Dump

```

"G1 Main Marker" os_prio=2 cpu=0.00ms elapsed=38.28s tid=0x00000214f1bb8000 nid=0x7984 runnable
"G1 Conc#0" os_prio=2 cpu=0.00ms elapsed=38.28s tid=0x00000214f1bba800 nid=0xbfc8 runnable
"G1 Refine#0" os_prio=2 cpu=0.00ms elapsed=38.27s tid=0x00000214ff17b800 nid=0xcdd4 runnable
"G1 Young RemSet Sampling" os_prio=2 cpu=0.00ms elapsed=38.27s tid=0x00000214ff17c000 nid=0x6618 runnable
"VM Periodic Task Thread" os_prio=2 cpu=15.63ms elapsed=38.01s tid=0x000002149125d000 nid=0xbf5c waiting on condition
JHI global refs: 17, weak refs: 0
Found one Java-level deadlock:
*****
"Thread-0":
  waiting for ownable synchronizer 0x00000008d2dc9c0, (a java.util.concurrent.locks.ReentrantLock$NonfairSync),
  which is held by "Thread-1"
"Thread-1":
  waiting for ownable synchronizer 0x00000008d2dc990, (a java.util.concurrent.locks.ReentrantLock$NonfairSync),
  which is held by "Thread-0"
Java stack information for the threads listed above:
*****
"Thread-0":
  at jdk.internal.misc.Unsafe.park(java.base@11/Native Method)
  - parking to wait for <0x00000008d2dc9c0> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)
  at java.util.concurrent.locks.LockSupport.park(java.base@11/LockSupport.java:194)
  at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(java.base@11/AbstractQueuedSynchronizer.java:885)

```

LockGraphFacadeExample

com.greatlearning.detection.LockGraphFacadeExample (pid 36244)

Overview

Save data: ☒ Saved data ☒ Details

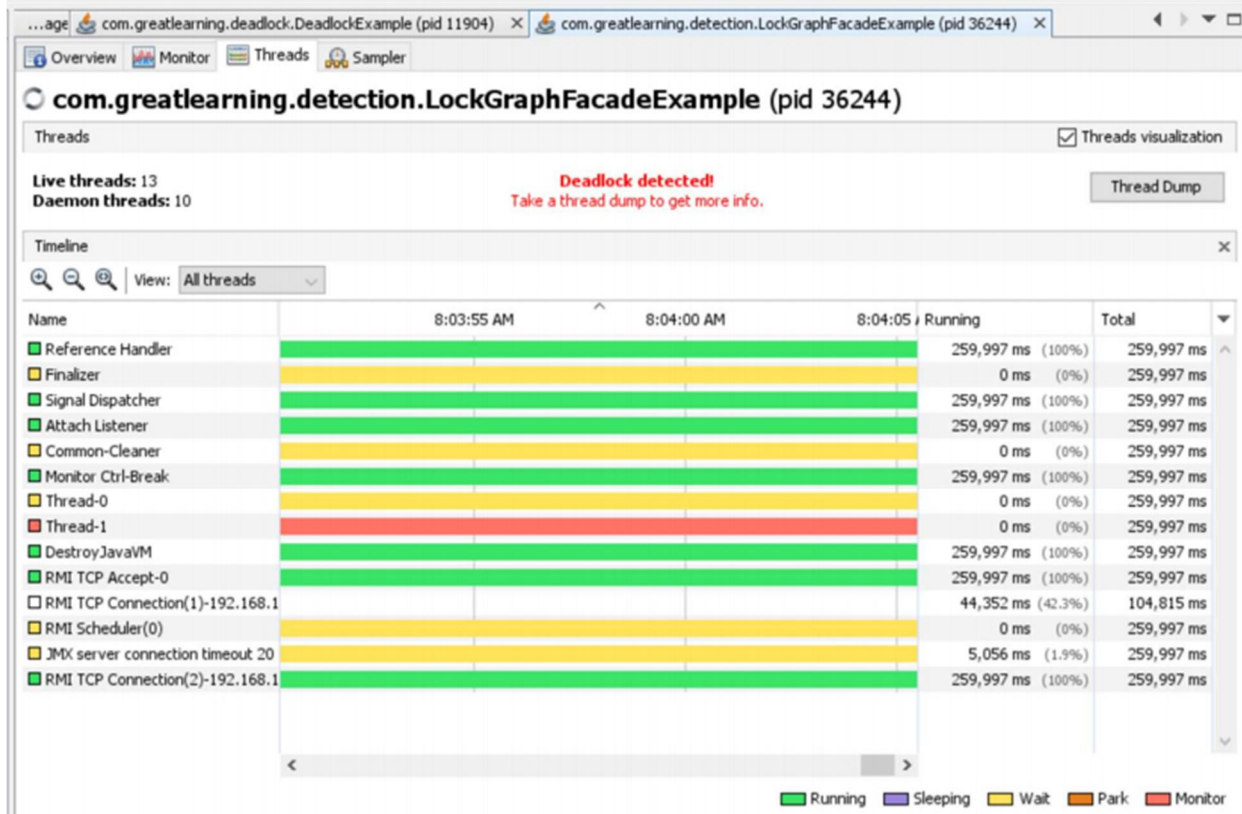
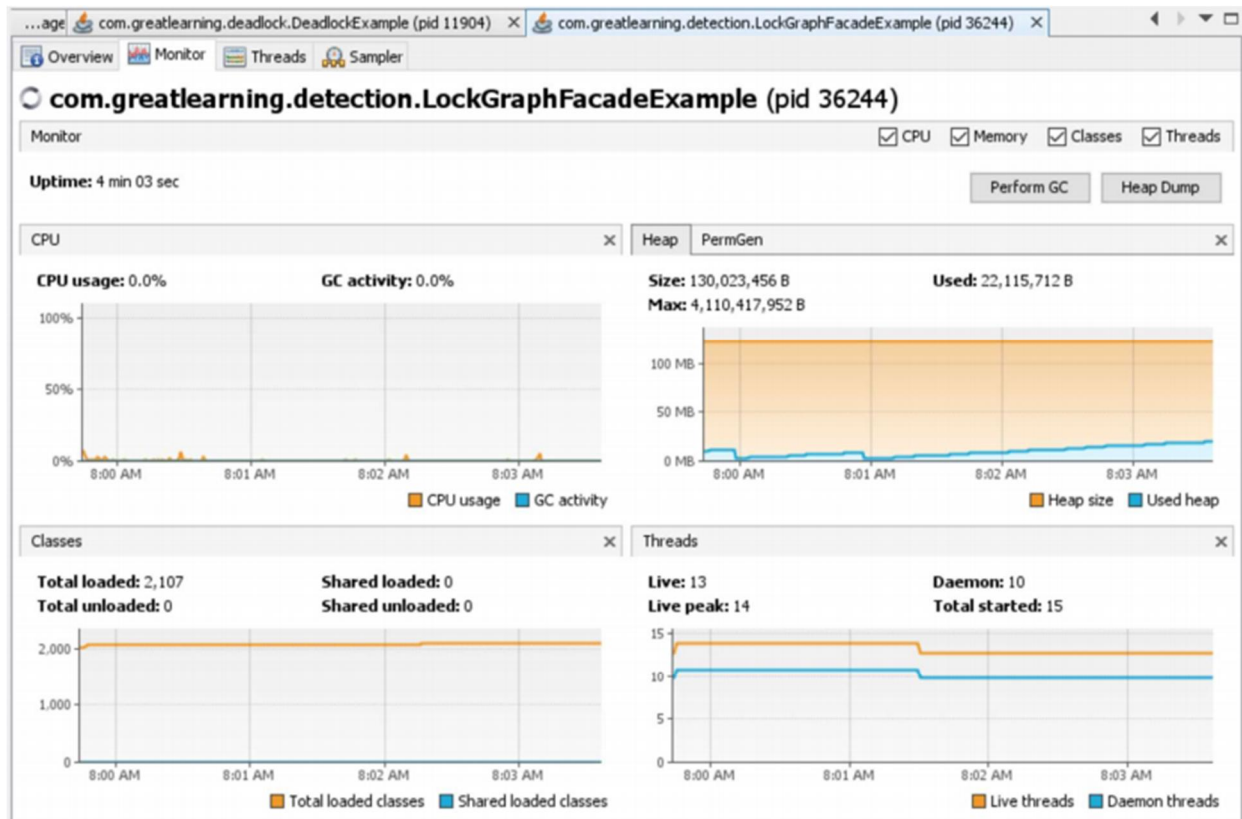
PID: 36244
Host: localhost
Main class: com.greatlearning.detection.LockGraphFacadeExample
Arguments: <none>

JVM: OpenJDK 64-Bit Server VM (11+28, mixed mode)
Java: version 11, vendor Oracle Corporation
Java Home: C:\Apps\Java\jdk-11
JVM Flags: <none>

Heap dump on OOME: disabled

Saved data: ☒ JVM arguments: ☒ System properties: ☒

Thread Dumps: Heap Dumps: 0
Profiler Snapshot: -javaagent:C:\Apps\JetBrains\IntelliJ IDEA 2021.1.2\lib\idea_rt.jar=60743:C:\Apps\JetBrains\IntelliJ IDEA 2021.1.2\bin
-Dfile.encoding=UTF-8



```

Thread Dump
"G1 Conc#0" os_prio=2 cpu=0.00ms elapsed=16.27s tid=0x000001db8dc4c800 nid=0x4048 runnable
"G1 Refine#0" os_prio=2 cpu=0.00ms elapsed=16.27s tid=0x000001db8dcd0000 nid=0x9b70 runnable
"G1 Young RemSet Sampling" os_prio=2 cpu=0.00ms elapsed=16.27s tid=0x000001db8dcd1000 nid=0xd68 runnable
"VM Periodic Task Thread" os_prio=2 cpu=0.00ms elapsed=16.09s tid=0x000001dbad0c2000 nid=0xc2f90 waiting on condition

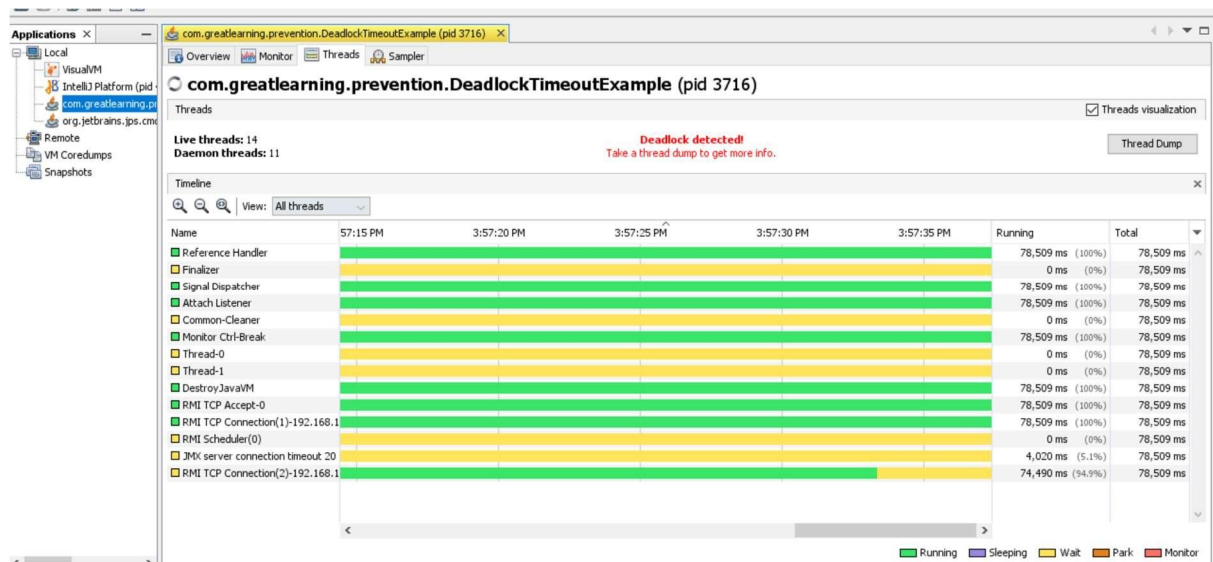
JNI global refs: 17, weak refs: 0

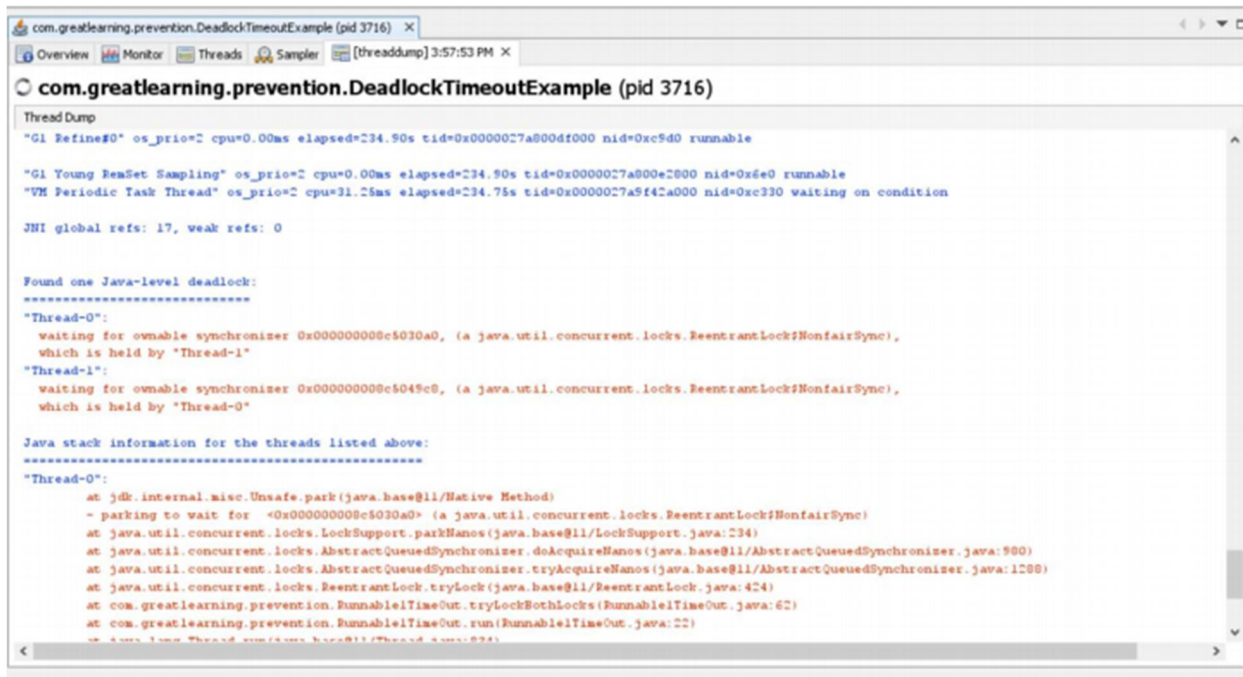
Found one Java-level deadlock:
=====
"Thread-0":
  waiting to lock monitor 0x000001dbac416c00 (object 0x000000008d2dfb70, a com.greatlearning.detection.LockGraphFacade),
  which is held by "Thread-1"
"Thread-1":
  waiting for ownable synchronizer 0x000000008d2e1e38, (a java.util.concurrent.locks.ReentrantLock$NonfairSync),
  which is held by "Thread-0"

Java stack information for the threads listed above:
=====
"Thread-0":
  at com.greatlearning.detection.LockGraphFacade.tryLock(LockGraphFacade.java:15)
  - waiting to lock <0x000000008d2dfb70> (a com.greatlearning.detection.LockGraphFacade)
  at com.greatlearning.detection.RunnableDeadlockDetection.tryLockBothLocks(RunnableDeadlockDetection.java:58)
  at com.greatlearning.detection.RunnableDeadlockDetection.run(RunnableDeadlockDetection.java:24)
  at java.lang.Thread.run(java.base@11/Thread.java:834)
"Thread-1":

```

DeadlockTimeoutExample





HashMapWithoutEqualandHashCode

HeapDumpMemLeakChk pdf file

The screenshot shows a Java heap dump analysis tool displaying a list of objects and their memory usage. The table has the following columns: Class Name, Shallow Heap, Retained Heap, and Percentage.

Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.Properties @ 0x8599e228	56	5,47,304	35.69%
class sun.util.calendar.ZoneInfoFile @ 0x859017c0 System	120	1,53,496	10.01%
java.util.HashSet @ 0x8594c880	16	91,056	5.94%
class sun.util.resources.Bundles @ 0x85966948 System C	24	65,344	4.26%
java.util.concurrent.ConcurrentHashMap @ 0x859e17d0	64	33,968	2.22%
class java.lang.System @ 0x8598ee68 System Class	48	33,632	2.19%
java.io.PrintStream @ 0x859b3610	40	25,112	1.64%
java.lang.Module @ 0x859e7460	48	17,808	1.16%
class java.nio.charset.Charset @ 0x859a1078 System Clas	32	14,296	0.93%
class sun.util.cldr.CLDRLocaleDataMetaInfo @ 0x8599	16	11,936	0.78%
class java.lang.invoke.LambdaForm\$Kind @ 0x8593d580	272	9,904	0.65%
java.util.HashMap @ 0x8592be28	48	8,544	0.56%
jdk.internal.loader.ClassLoaders\$PlatformClassLoader @ (96	8,160	0.53%
class java.lang.invoke.MethodType @ 0x85991158 Syster	48	8,096	0.53%
java.lang.module.ModuleDescriptor @ 0x859c4630	64	7,696	0.50%
class sun.util.resources.cldr.provider.CLDRLocaleDataMeta	8	7,256	0.47%
java.lang.module.Configuration @ 0x859e76a8	40	6,712	0.44%
class sun.util.resources.TimeZoneNames @ 0x8597b858 s	8	5,360	0.35%
jdk.internal.loader.ClassLoaders\$AppClassLoader @ 0x85	96	5,128	0.33%
class java.lang.Integer\$IntegerCache @ 0x8599cbb8 Syst	24	5,104	0.33%
class sun.util.locale.provider.LocaleProviderAdapter @ 0x	24	4,712	0.31%
java.lang.ModuleLayer @ 0x859efb40	40	4,320	0.28%
java.lang.module.ModuleDescriptor @ 0x859cbf28	64	4,256	0.28%
class jdk.internal.module.SystemModules\$default @ 0x85	8	4,064	0.27%
java.lang.Module @ 0x859f1270	48	3,904	0.25%
java.lang.Module @ 0x859eff20	48	3,776	0.25%
class jdk.internal.module.ArchivedModuleGraph @ 0x859	8	3,288	0.21%

173M of 269M