Eclipse Memory Analyzer Tool (MAT)

Sang Shin
JPassion.com
"Learn with Passion!"

Topics

- Basic concepts
  - Heap dump
  - Shallow vs. Retained heap
  - Dominator tree
  - GC (Garbage Collection) Roots
  - Incoming & outgoing references
  - Accumulation point
- How to detect memory leak
- Class loader memory leak example

Heapdump

What is a Heap Dump?

- A heap dump is a snapshot of the memory of a Java process at a certain moment of time
- Heap dump format
  - HPROF binary format (most common)
  - IBM system dumps (after pre-processing them)
  - IBM portable heap dumps (PHD)
- Usually a full GC is triggered before the heap dump is written so it contains information about the remaining objects

What Does a HeapDump Contain?

- All Objects
  - Class, field, primitive values and references
- All Classes
  - Classloader, name, super class, static fields
- Thread stacks and local variables
  - The call-stacks of threads at the moment of the snapshot, and per-frame information about local objects
- Garbage Collection(GC) roots

A Heap Dump Does NOT Tell You..

- Where an object was allocated
- When an object was created
- How many objects were garbage collected

- It is indeed just a snapshot

A Heap Dump Can Help You

- Analyze the reason for an *OutOfMemoryError*
- Analyze the memory footprint of an application
- Debug non-memory related problems too
  - Why an application is non-responsive? (Through threads analysis)

How to Get a Heap Dump

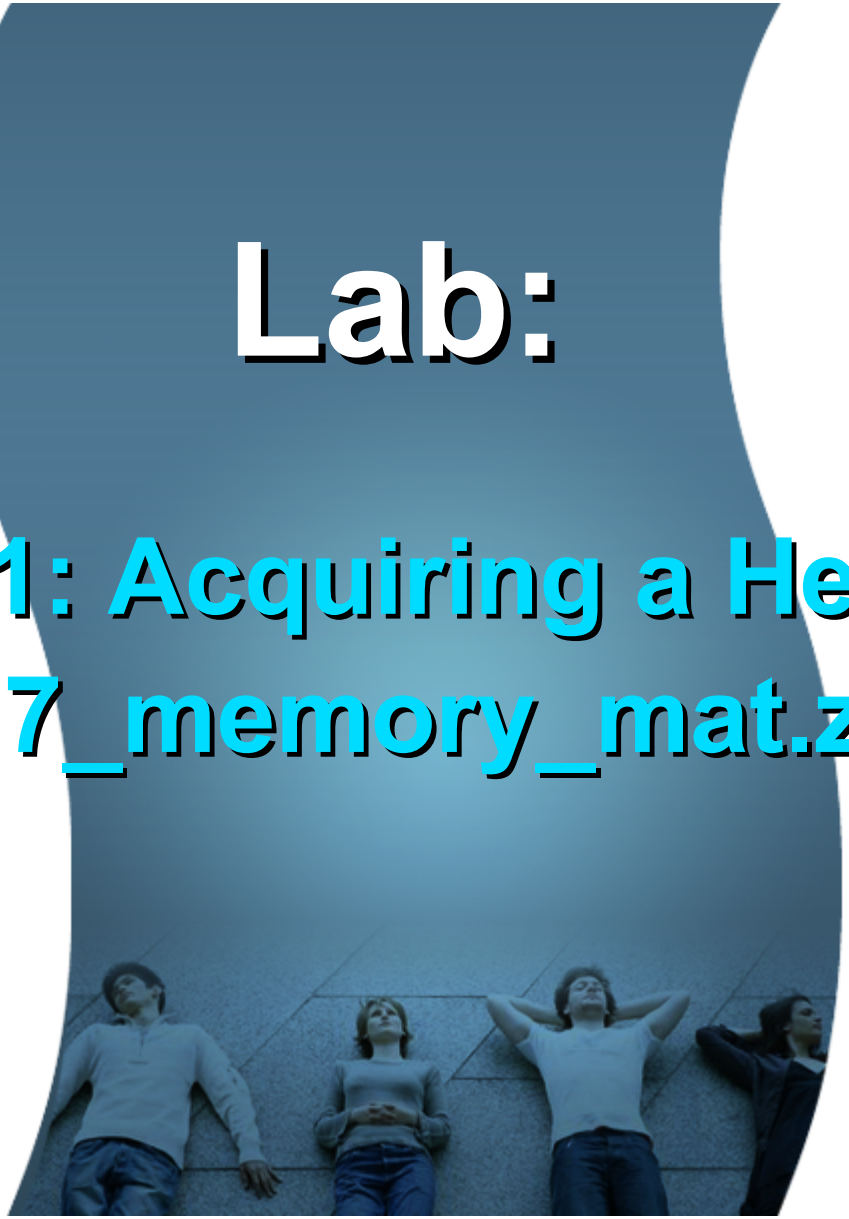
- You can trigger a heap dump (on-demand heap dumping)
  - Within a tool (jconsole, Eclipse Memory Analyzer, NetBeans, Eclipse, JMC, etc)
  - jmap -dump:format=b,file=<filename.hprof> <pid>
- Application started with following JVM option creates a Heap dump when OutOfMemoryError occurs
  - *-XX:+HeapDumpOnOutOfMemoryError*
  - There is no negative performance impact on the VM
- Application started with following JVM option creates a Heap dump when CTRL+BREAK is pressed
  - *-XX:+HeapDumpOnCtrlBreak*

How to Get a “Good” Heap Dump

- When memory is exhausted, the leak will occupy the most of the heap space
- Ensure big enough heap space, this will make the leak easier to find
  - The memory leak pattern looks move obvious

Lab:

Exercise 1: Acquiring a Heapdump
5117_memory_mat.zip

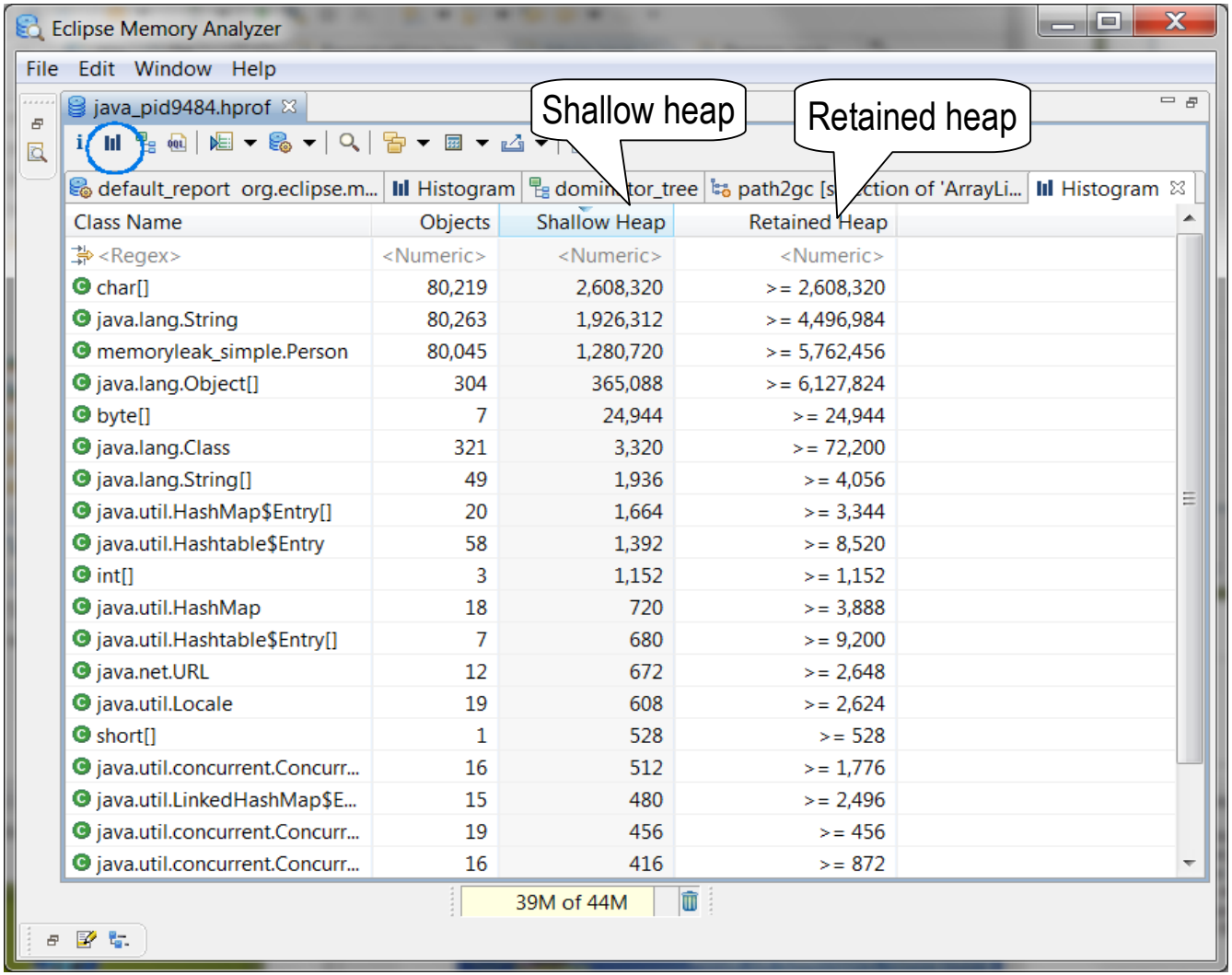


Shallow Heap
vs.
Retained Heap

Shallow Heap vs. Retained Heap

- Shallow heap is the memory consumed by one object
  - Its size in the heap
  - An object needs 32 or 64 bits (depending on the OS architecture) per reference, 4 bytes per Integer, 8 bytes per Long, etc.
  - Depending on the heap dump format, the size may be adjusted (e.g. aligned to 8, etc...) to model better the real consumption of the VM
- Retained set of X is the set of objects which would be removed by GC when X is GC'ed
  - Retained heap of X is the sum of shallow sizes of all objects in the retained set of X, i.e. memory kept alive by X
  - Amount of heap memory that will be freed when X is garbage collected

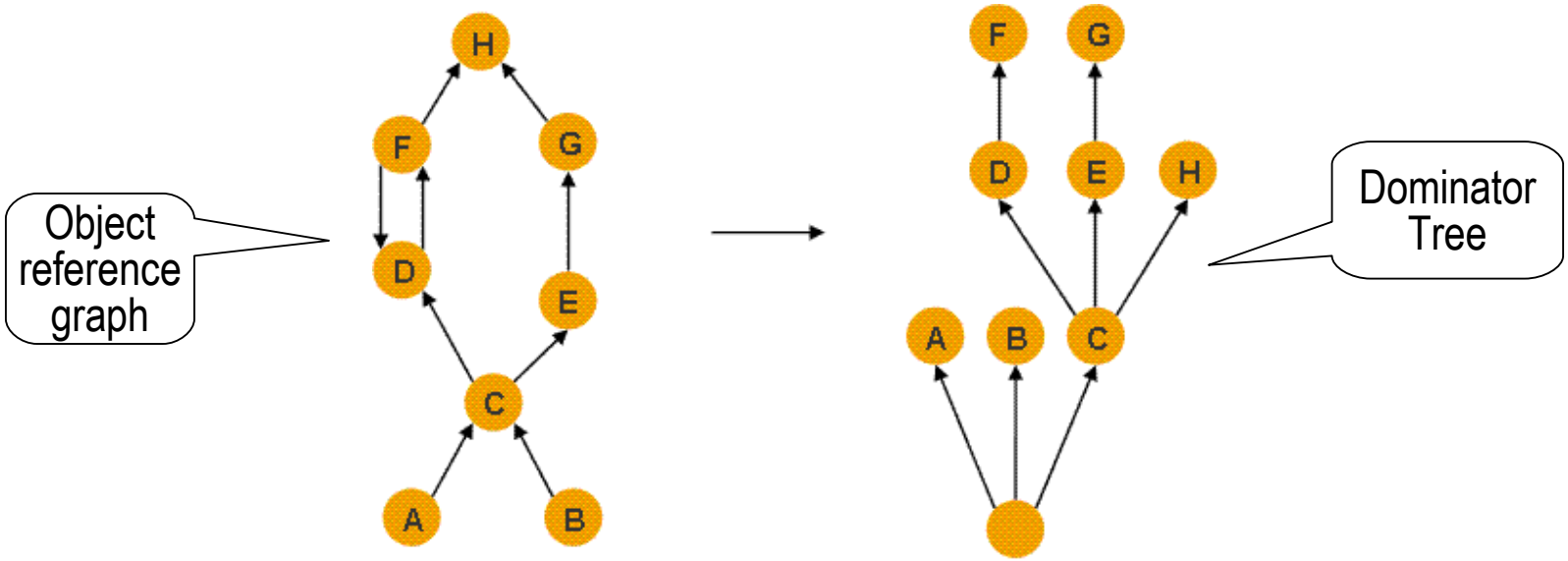
Shallow Heap vs. Retained Heap



Dominator Tree

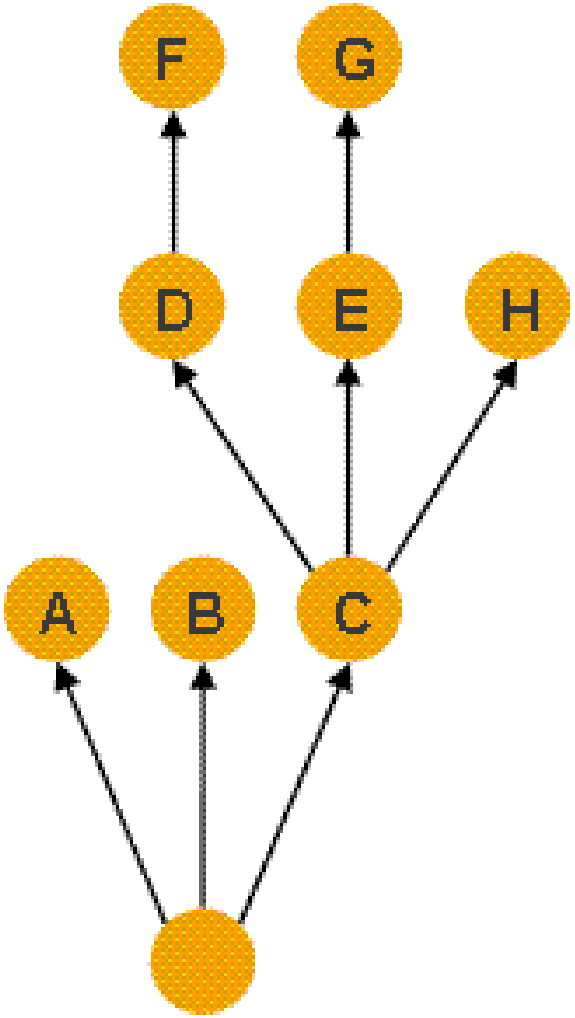
What is and Why Dominator Tree?

- A dominator tree is built out of the object graph.
- The transformation of the “Object reference graph” into a “Dominator tree” allows you to easily identify the biggest chunks of “retained memory” and the keep-alive dependencies among objects.



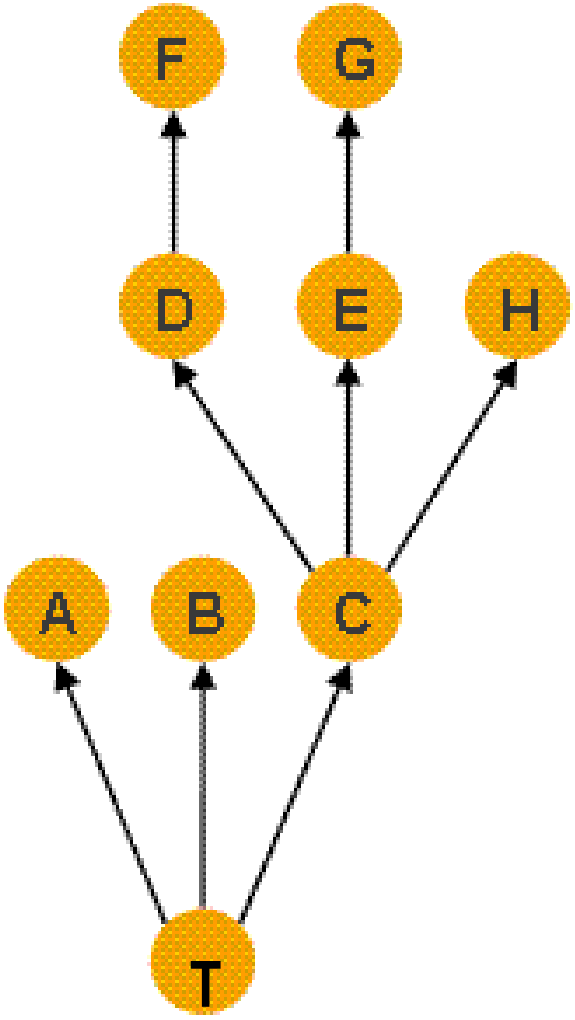
Objects in Dominator Tree

- Each object is the immediate dominator of its children, so dependencies between the objects are easily identified.
- An object x dominates an object y if every path in the object graph from the start (or the root) node to y must go through x
- “C” is immediate dominator of “D”, “E”, and “H”
- “C” is dominator of “D”, “E”, “H”, “F”, “G”



Dominator Tree & Retained Set & Heap

- The objects belonging to the sub-tree of x (i.e. the objects dominated by x) represent the retained set of x
- If “C” is GC'ed, the all the retained heap space of it will be also GC'ed
- The retained heap space of “C” equals the collection of all shallow heap spaces of its chidren - “D”, “E”, “H”, “F”, “G”

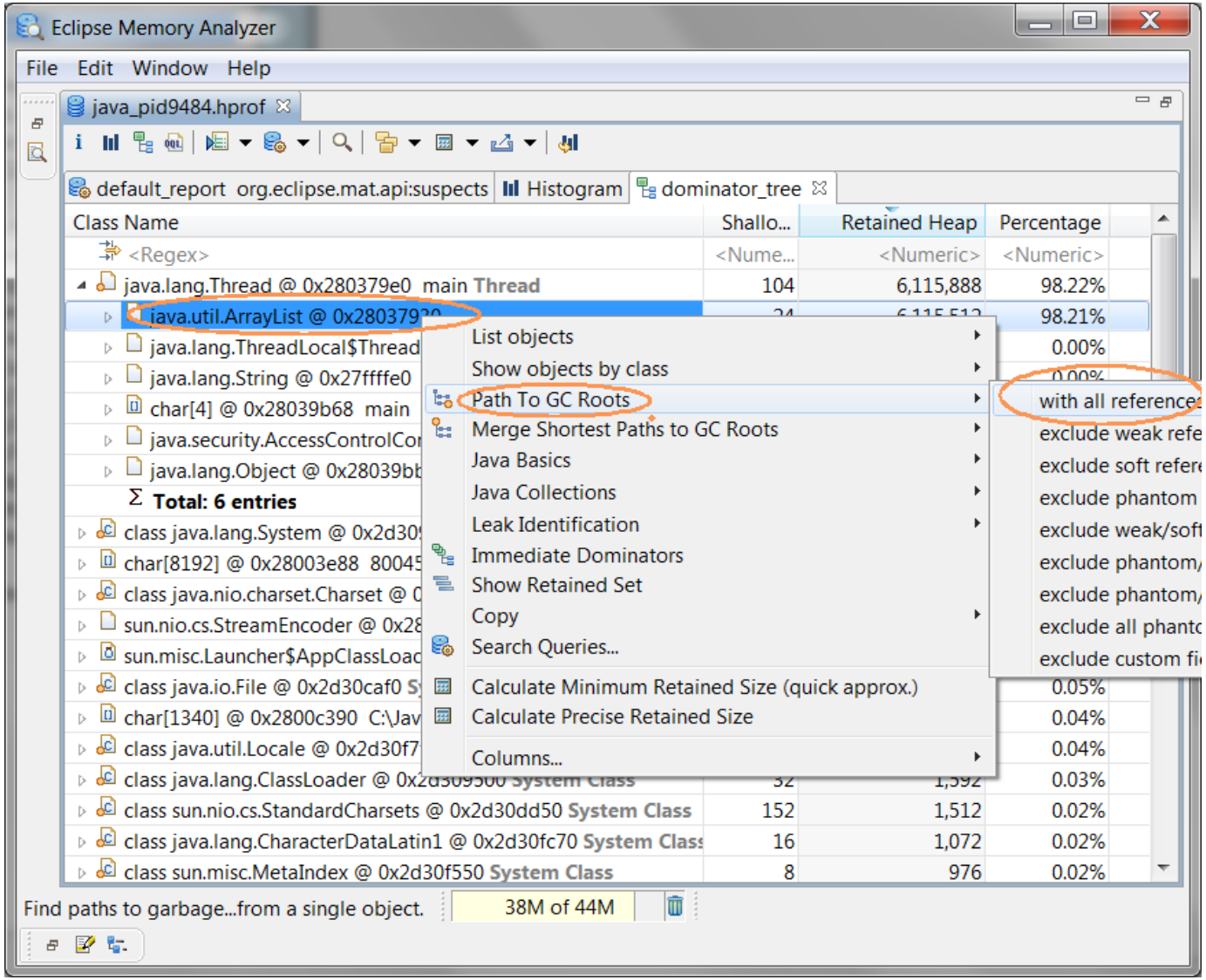


GC (Garbage Collection) Roots

What is & Why Garbage Collection Roots?

- A Garbage Collection root (GC root) is an object that is accessible from outside the heap
  - They are root owner (root dominator) of other objects in the heap
- The *Find Nearest GC Root* feature can help you track down memory leaks by showing the owner chain of the references that prevents an object from being garbage collected.
- Example scenarios where an object is a GC root:
  - Thread – A started, but not stopped, thread
  - System class - Class loaded by bootstrap/system class loader. For example, everything from the rt.jar like java.util.*
  - ...

Path to GC Root

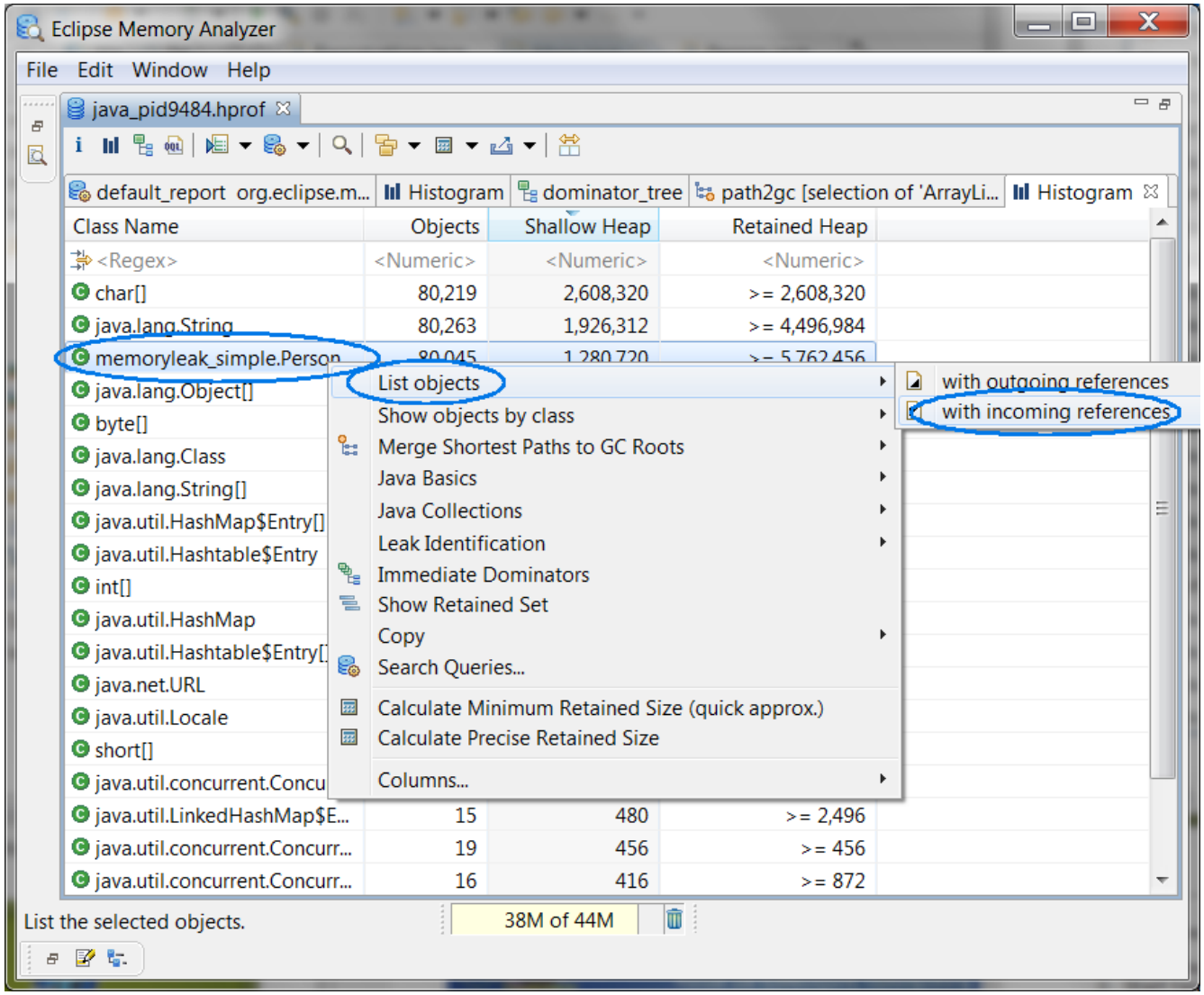


Other Misc. Concepts

Incoming & Outgoing References

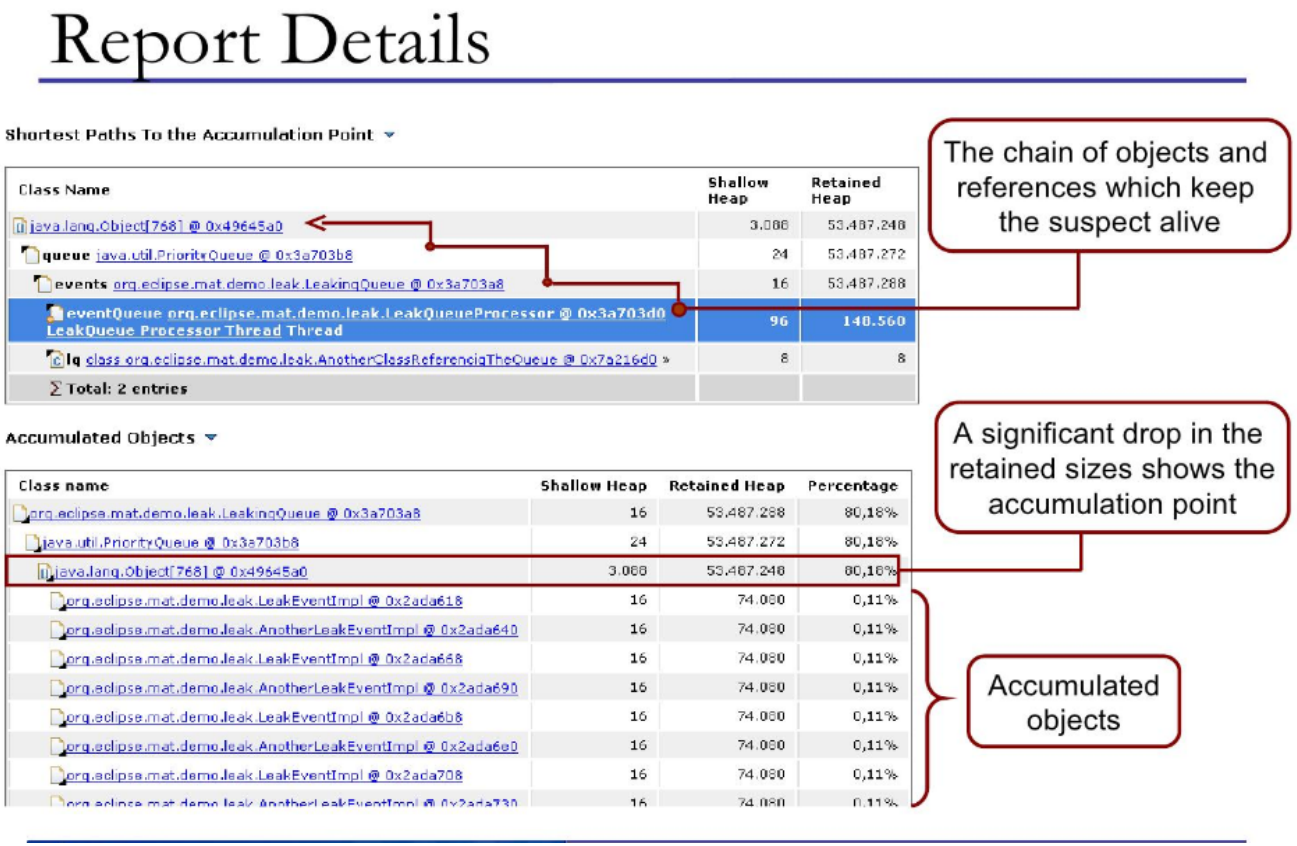
- Outgoing references
  - Show what objects the current object is making references to
- Incoming references
  - Shows what objects are making references to the current object
  - Starts from GC Root

Incoming References



Accumulation Point

- Shows significant drop in the retained size – good candidate where memory leak starts to occur



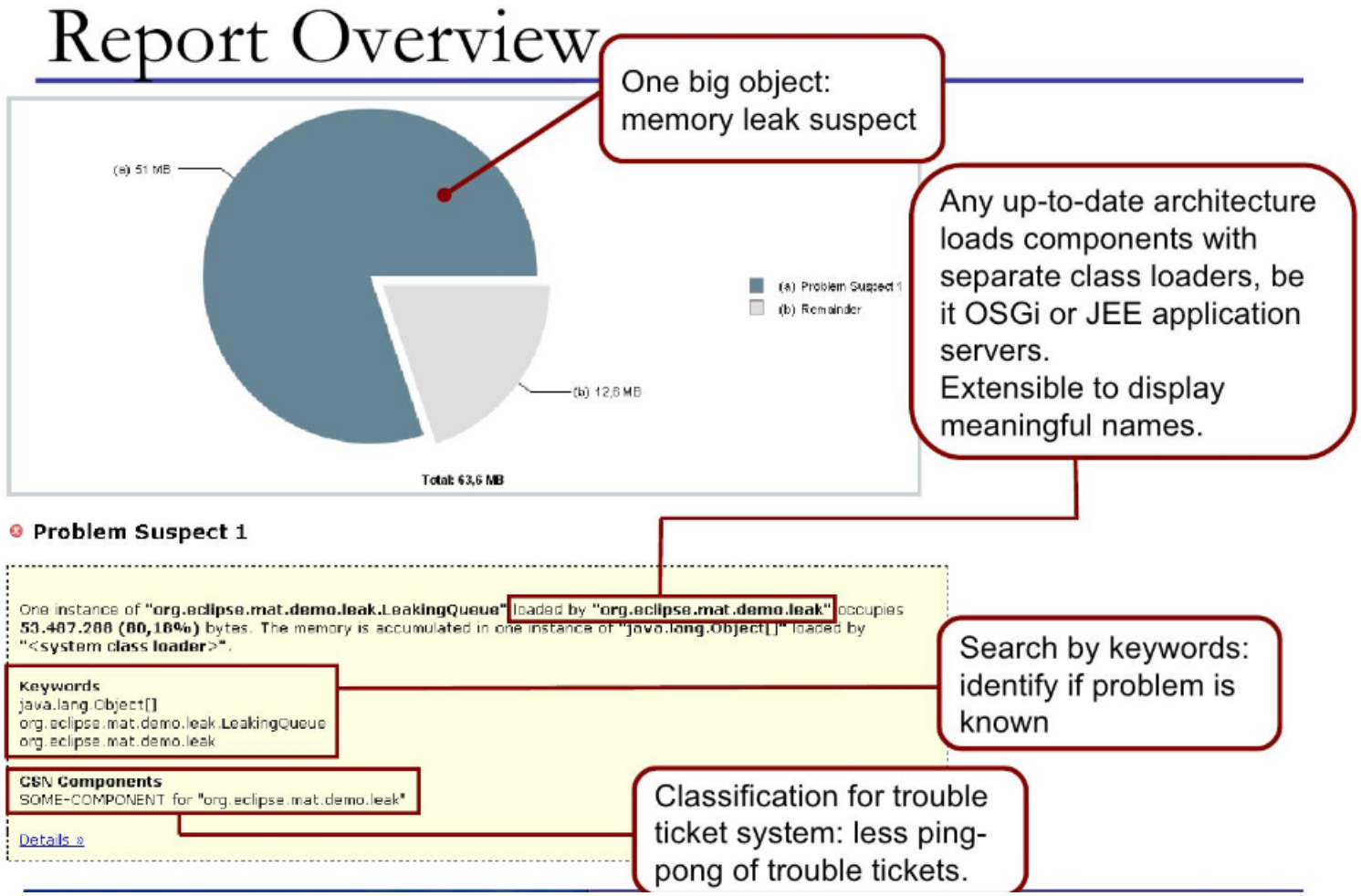
How to Analyze a Heapdump? (How to find Memory Leak?)

Schemes of Analyzing Heap Dump

- Find the biggest objects
  - Good starting point
- Analyze why they are kept in memory
  - Someone has a reference to the objects
  - Incoming references, GC Root
- Analyze what makes them big
  - Check retained heap
  - Accumulation point

Memory Analyzer performs the above and suggests “Problem Suspect”

One Big Object, Problem Suspect



Chain of Incoming References, Accumulation Point

Report Details

Shortest Paths To the Accumulation Point

Class Name	Shallow Heap	Retained Heap
java.lang.Object[768] @ 0x49645a0	3.088	53.487.248
queue java.util.PriorityQueue @ 0x3a703b8	24	53.487.272
events org.eclipse.mat.demo.leak.LeakingQueue @ 0x3a703a8	16	53.487.288
eventQueue org.eclipse.mat.demo.leak.LeakQueueProcessor @ 0x3a703d0 LeakQueue Processor Thread Thread	96	140.560
lq class org.eclipse.mat.demo.leak.AnotherClassReferencingTheQueue @ 0x7a216d0 »	8	8
Σ Total: 2 entries		

The chain of objects and references which keep the suspect alive

Accumulated Objects

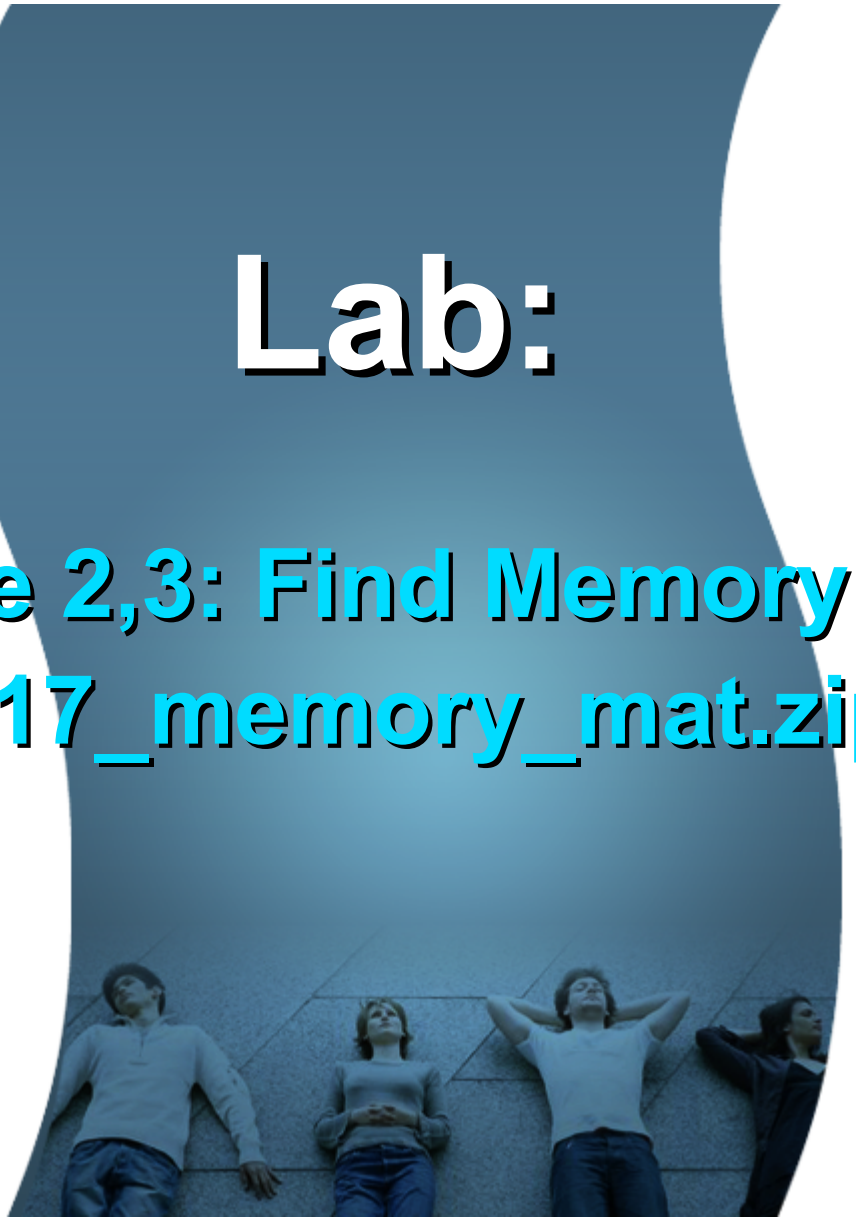
Class name	Shallow Heap	Retained Heap	Percentage
org.eclipse.mat.demo.leak.LeakingQueue @ 0x3a703a8	16	53.487.288	80,18%
java.util.PriorityQueue @ 0x3a703b8	24	53.487.272	80,18%
java.lang.Object[768] @ 0x49645a0	3.088	53.487.248	80,18%
org.eclipse.mat.demo.leak.LeakEventImpl @ 0x2ada618	16	74.080	0,11%
org.eclipse.mat.demo.leak.AnotherLeakEventImpl @ 0x2ada640	16	74.080	0,11%
org.eclipse.mat.demo.leak.LeakEventImpl @ 0x2ada668	16	74.080	0,11%
org.eclipse.mat.demo.leak.AnotherLeakEventImpl @ 0x2ada690	16	74.080	0,11%
org.eclipse.mat.demo.leak.LeakEventImpl @ 0x2ada6b8	16	74.080	0,11%
org.eclipse.mat.demo.leak.AnotherLeakEventImpl @ 0x2ada6e0	16	74.080	0,11%
org.eclipse.mat.demo.leak.LeakEventImpl @ 0x2ada708	16	74.080	0,11%
org.eclipse.mat.demo.leak.AnotherLeakEventImpl @ 0x2ada730	16	74.080	0,11%

A significant drop in the retained sizes shows the accumulation point

Accumulated objects

Lab:

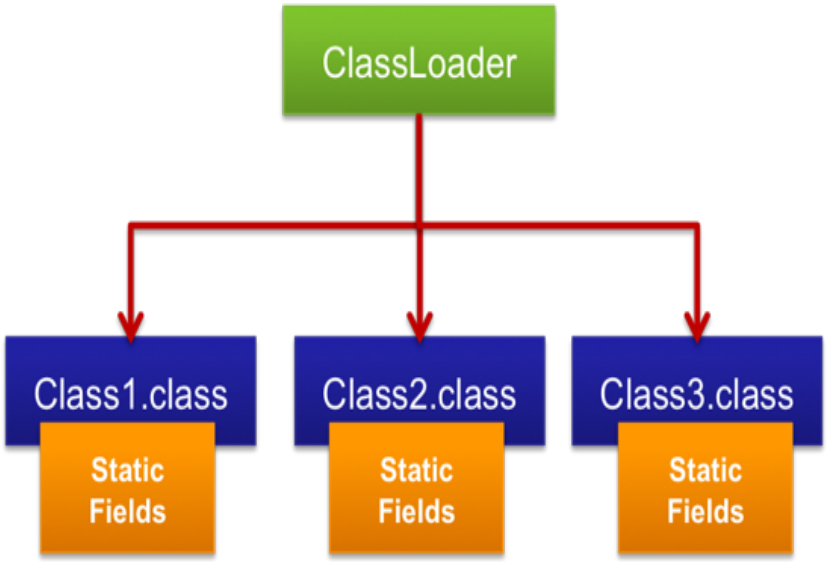
Exercise 2,3: Find Memory Leaks
5117_memory_mat.zip



ClassLoader Memory Leak

ClassLoader and Classes it Loaded

- Every object has a reference to its class object
- Every class object has a reference to its classloader
- Every classloader in turn has a reference to each of the classes it has loaded, each of which might hold some static fields defined in the class: (This is the killer!!)



Why ClassLoader Leak is so Common?

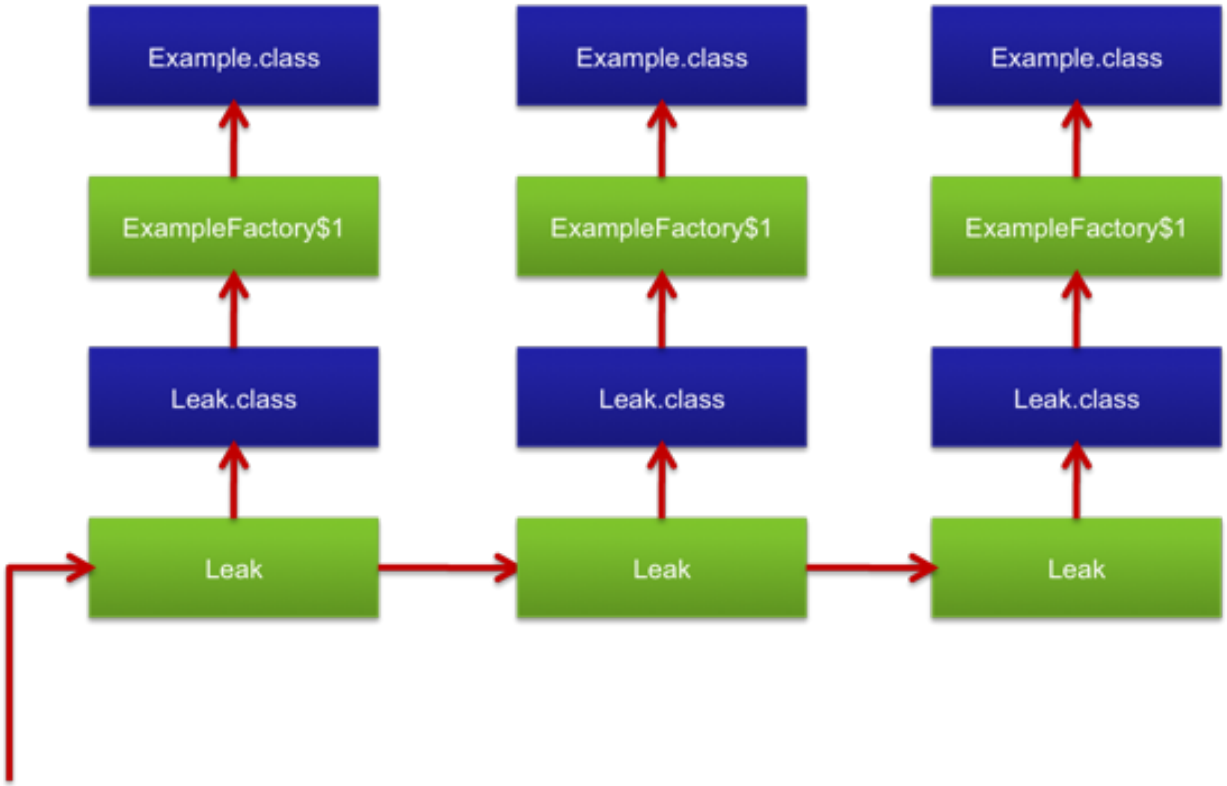
- To leak a classloader, it's enough to leave a reference to any object, created from a class, loaded by that classloader
  - Even if that object seems completely harmless (e.g. doesn't have a single field), it will still hold on to its classloader

Why ClassLoader Leak is so Bad?

- If a classloader is leaked, then it will hold on to all its classes and all their static fields
  - Even if your application doesn't have any large static caches, it doesn't mean that the framework you use doesn't hold them for you (e.g. Log4J is a common culprit)
- Major cause of OutOfMemoryException

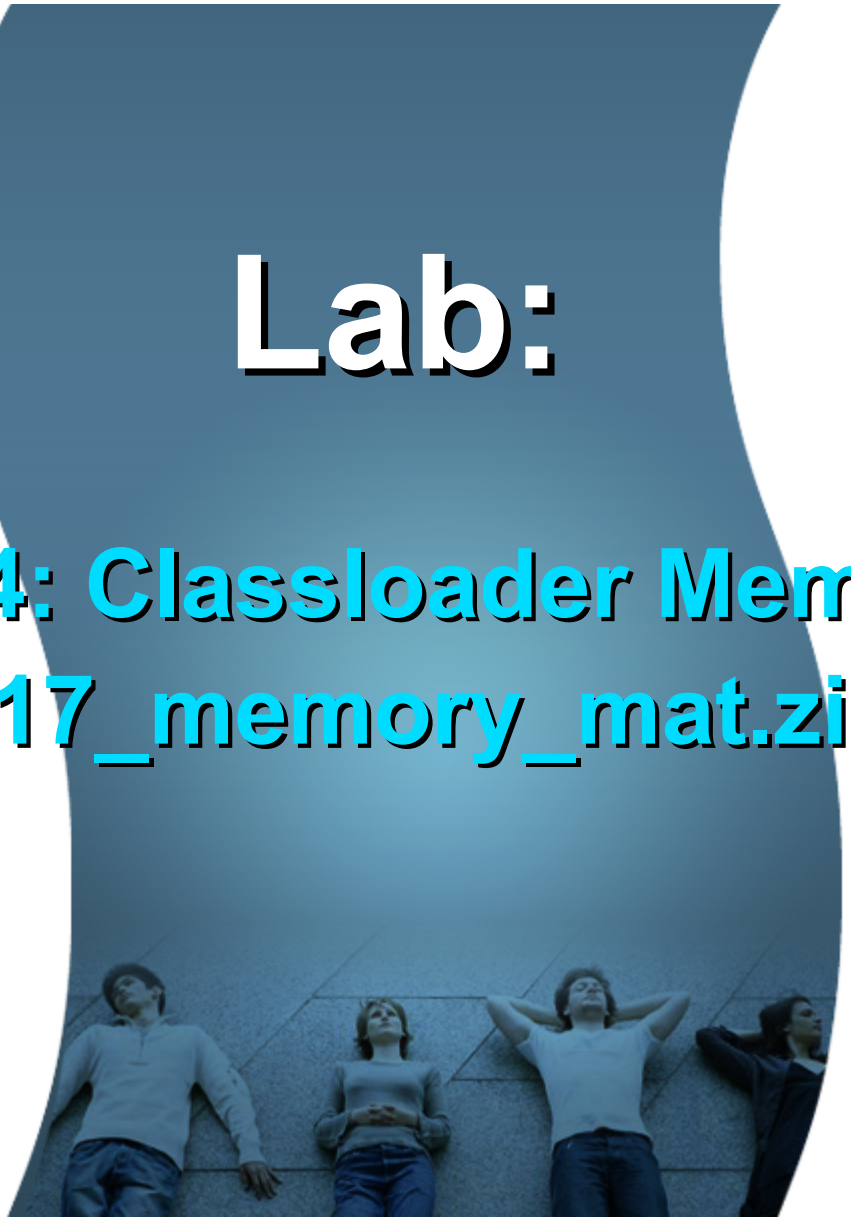
ClassLoader Leak Example

- Each *Leak* object and it class object are leaking. They are holding on to their classloaders
- The classloaders are holding onto the *Example* class object (including the static fields) they have loaded



Lab:

Exercise 4: Classloader Memory leak
5117_memory_mat.zip



Learn with Passion!
JPassion.com