Where Is My Memory?

Plumbr

Who am I

- Nikita Salnikov-Tarnovski
- Founder and Master Developer from Plumbr
- @iNikem / @JavaPlumbr

Plumbr

- Application performance monitor with root cause detection
- In case of problem reports you exact details
- Memory leaks, class loader leaks, GC related problems, contented locks, slow JDBC and HTTP requests, OOMs

Agenda

- Quick overview of Java Memory Management
- A word on Garbage Collector
- Reachability and memory leaks
- Different kinds of OutOfMemoryErrors
- Memory usage monitoring
- Heap dump
- Eclipse Memory Analyser Tool

The most important thing

- Ask questions!

JVM process memory

- JVM is just usual process from OS point of view
- And requires some memory for itself:
  - GC
  - JIT
  - JNI
  - threads

JVM application memory

- And then comes your application
- Heap
- Permanent Generation
- Threads
- And native memory
  - Off-heap allocations
  - Metaspace

Default sizes

- Default sizes of these regions depends on the computer
- `java -XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal -version`
- `MaxHeapSize`
- `MaxPermSize/MaxMetaspaceSize`
- `ThreadStackSize`
- http://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/toc.html

How to change them

- You almost always want to go with non-default sizes
- `-Xmx2g`
- `-XX:MaxPermSize=128m`
- `-Xss512k (rarely)`
- `-XX:MaxMetaspaceSize=128m (rarely)`

Home reading

- https://plumbr.eu/blog/memory-leaks/why-does-my-java-process-consume-more-memory-than-xmx

You have to fit

- All that memory HAS to come from RAM
- You do NOT ever let it go to swap

Java memory management

- JVM has automatic memory management
- Developers don’t think about it
- They just create new objects and go on

Garbage Collector

- Subsystem of JVM for reclaiming “unused” memory
- Memory occupied by unused objects
- Not JVM specific, many runtimes have it
- Different algorithms
  - 8 in Oracle HotSpot
  - Plus C4 from Azul
  - Plus Shenandoah from RedHat

GC “wizardry”

- GC is not mind reading magician
- It always works by very specific and strict algorithm
- “No references, thus garbage”

GC roots

- Special objects, always considered to be alive
- Often heap objects referenced from outside the heap

GC roots

- System Classes
- JNI references
- Running Threads
- Local variables or parameters
- Native stack
- Used monitors
- Other :)

References

- From an object to the value of one of its instance fields
- From an array to one of its elements
- From an object to its class
- From a class to its class loader
- From a class to the value of one of its static fields
- From a class to its superclass

Not that hard

- Those are so called “hard references”
- There are also weak, soft and phantom references
- Subclasses of java.lang.ref.Reference

Weak Reference

- You can wrap an object into a weak reference
- If object is only *weakly reachable,* GC can discard it

Soft Reference

- GC can discard object wrapped into soft reference at any time.
- ANY time.
- But guarantees it happens before OOM

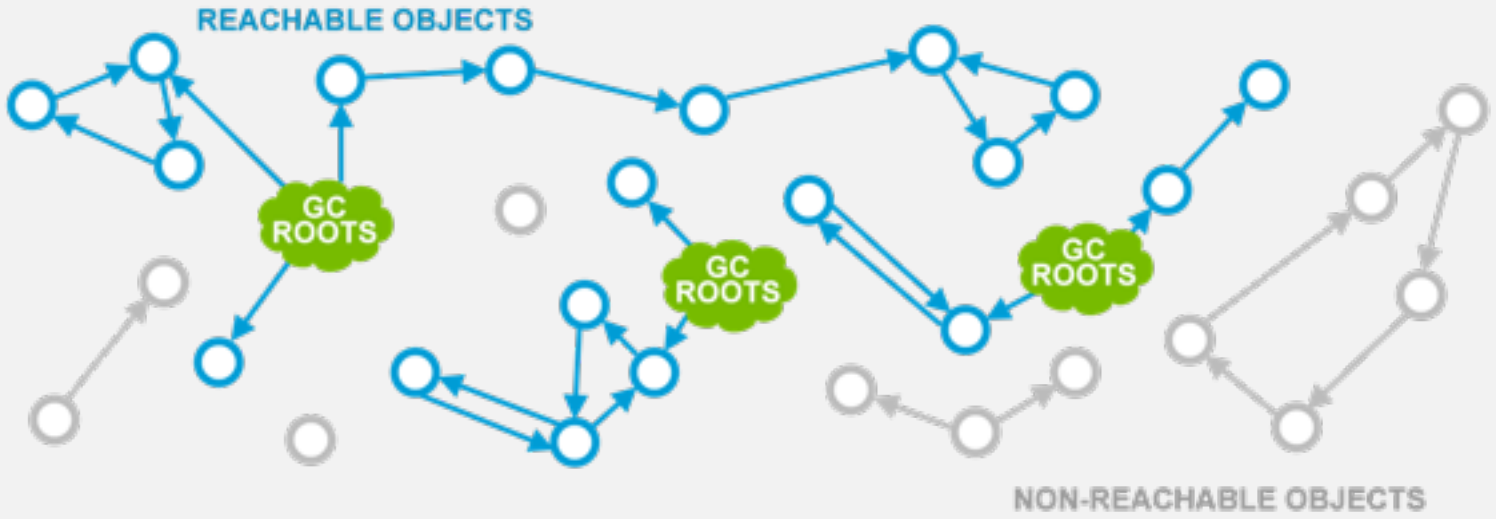
Phantom Reference

- If you find a legitimate use for them - call me
- I have seen 2 in my 15 years in Java.

Reachability

- Mark all GC roots as “reachable”
- Mark all objects referenced from “reachable” objects as “reachable” too
- Repeat until all reachable objects found
- Everything else is garbage and can be thrown away

The basis of GC



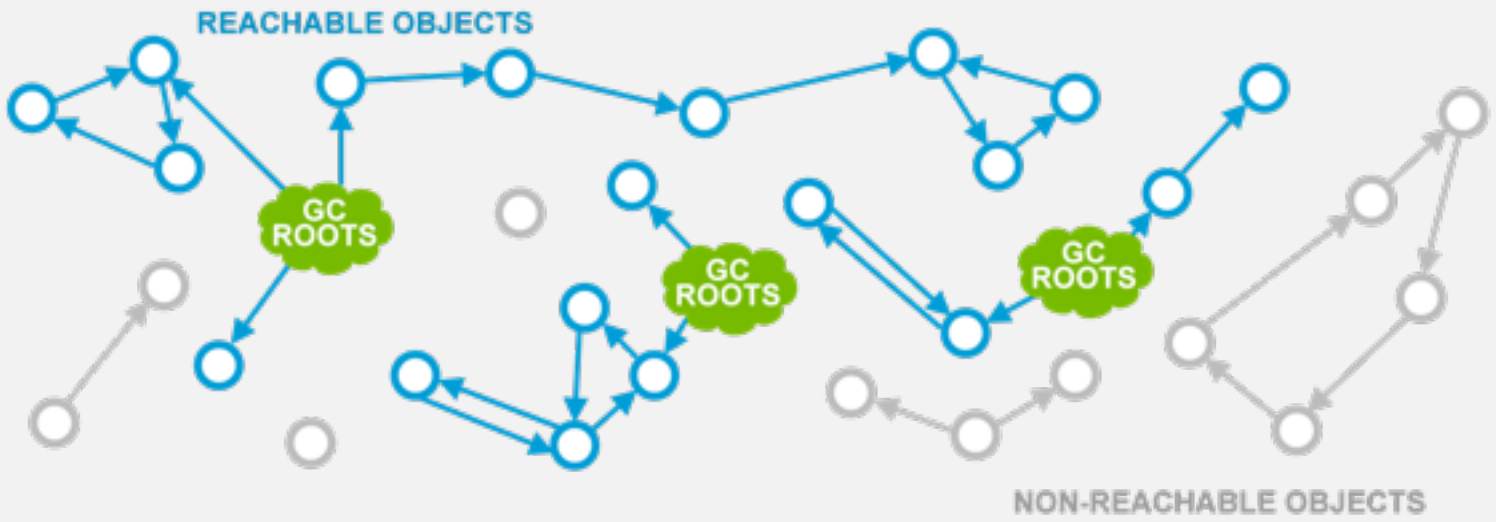
Home reading

- https://plumbr.eu/java-garbage-collection-handbook

Memory leak

- Reachable object(s), that will never be used by application
- Repetitive creation of such objects

Can you find one?



Of course you cannot

The notion of memory leak is 100% application specific

Examples

- Caches without look-ups and eviction
- String.substring
- Immortal threads
- Unclosed IO streams
- Storages with longer lifespan than stored values

So what?

- That memory cannot be reclaimed by GC
- Decreases the amount available to the application
- If leak is growing, soon there is nothing left

Symptoms

- OutOfMemoryError: XXX
- Application is very slow due to excessive GC

OOM

- Not all of them signals of memory leaks
  - Unable to create new native thread
  - Out of swap space
  - Requested array size exceeds VM limit

Home reading

- https://plumbr.eu/outofmemoryerror

Not a memory leak

- Too high allocation rate
- Cache with wrong size
- Trying to load too much data at once
- Fat data structures

Memory monitoring

- VisualVM/Java Mission Control
- jstat

GC logs

- `-XX:+PrintGCDetails`
- `-XX:+PrintGCTimeStamps`
- `-Xloggc:file.log`
- `-XX:+UseGCLogFileRotation`
- `-XX:NumberOfGClogFiles=N`

GC logs analyzers

- http://www.fasterj.com/tools/gcloganalysers.shtml
- https://github.com/chewiebug/GCViewer

Problem confirmed

- Reduce memory usage
- Tune GC
- Increase Xmx/PermGen

Memory dump

- One of the best ways to find out what consumes memory
- Binary representation of objects graph written to a file
- NOT an accurate representation

How to get memory dump

- `jmap -dump:format=b,file=heap.hprof`
- `-XX:+HeapDumpOnOutOfMemoryError`
  - `-XX:HeapDumpPath=./java_pid<pid>.hprof`

When to get memory dump

- As late as possible!
- You want to let that needle grow and fill the whole hey sack

What to do with it

- Get it to a computer with lot of memory.
- Add memory to that computer
- MAT

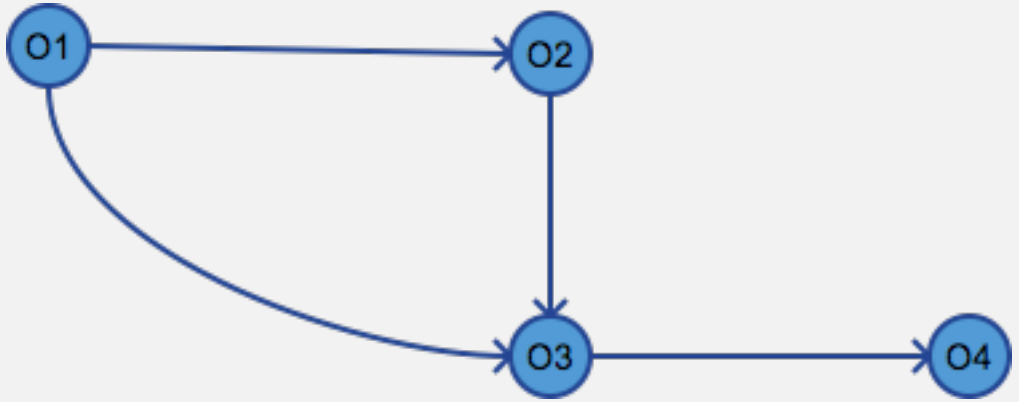
Shallow vs Deep

- You can measure *shallow* size of the object
- Or *deep* size of the subgraph starting with the object
- Or *retained* size of the subgraph *dominated* by the object

Shallow object size

- Size of the object itself
- With object header and all fields
- But without fields' values

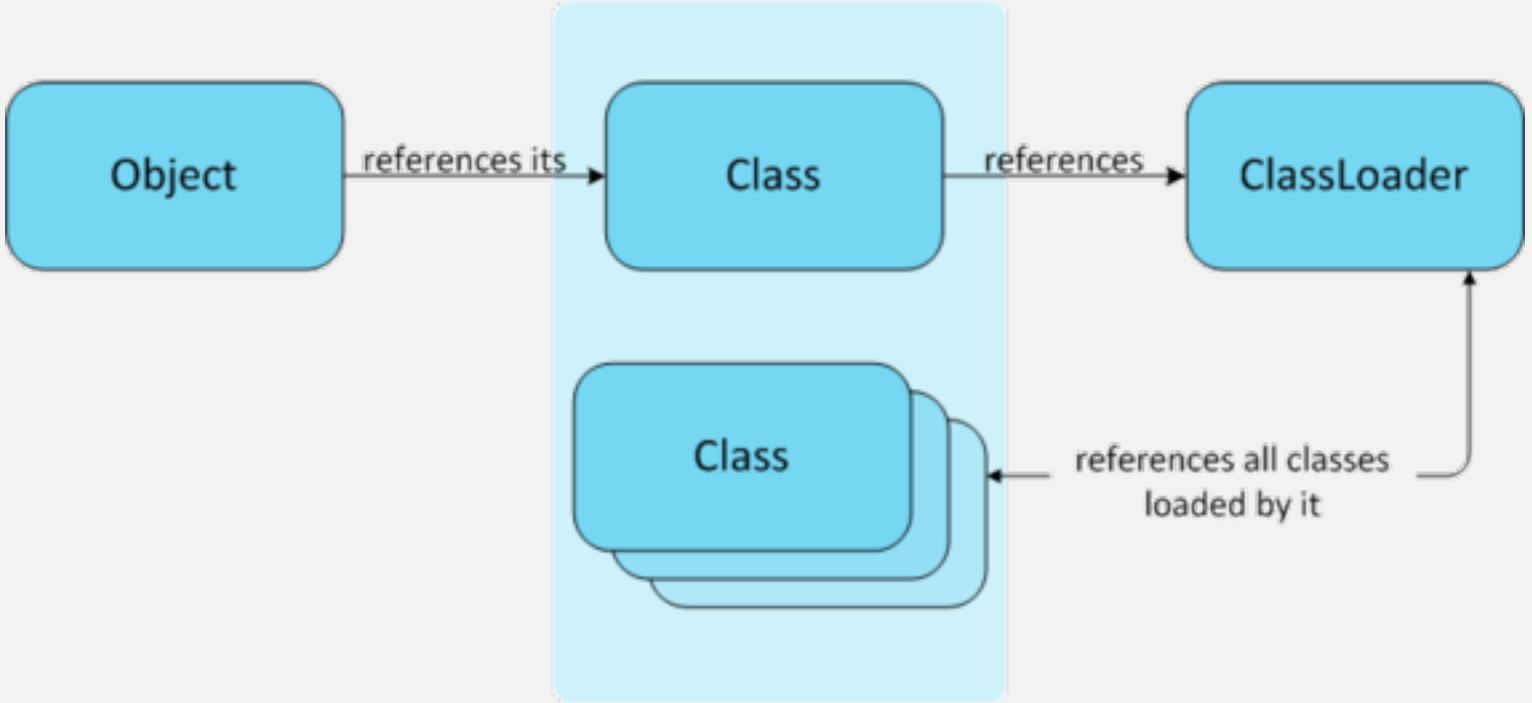
Retained size



- r(O1)=O1+O2+O3+O4
- r(O2)=O2
- r(O3)=O3+O4
- r(O4)=O4

- d(O1)=O1+O2+O3+O4
- d(O2)=O2+O3+O4
- d(O3)=O3+O4
- d(O4)=O4

Classloader leak



It’s not your fault

- Most of the classloader leaks you will ever encounter are not your fault
- Double-edge sword of reuse and modular development
- You have no idea what do you use in your application

Home reading

- https://plumbr.eu/blog/what-is-a-permgen-leak

Other tools

- Do NOT use profilers
- https://plumbr.eu/blog/solving-outofmemoryerror-memory-profilers

Solving performance problems is hard.
We don’t think it needs to be.

Plumbr