Introduction to Java Performance Optimization

Murat Öngüdü

03/2019

About me

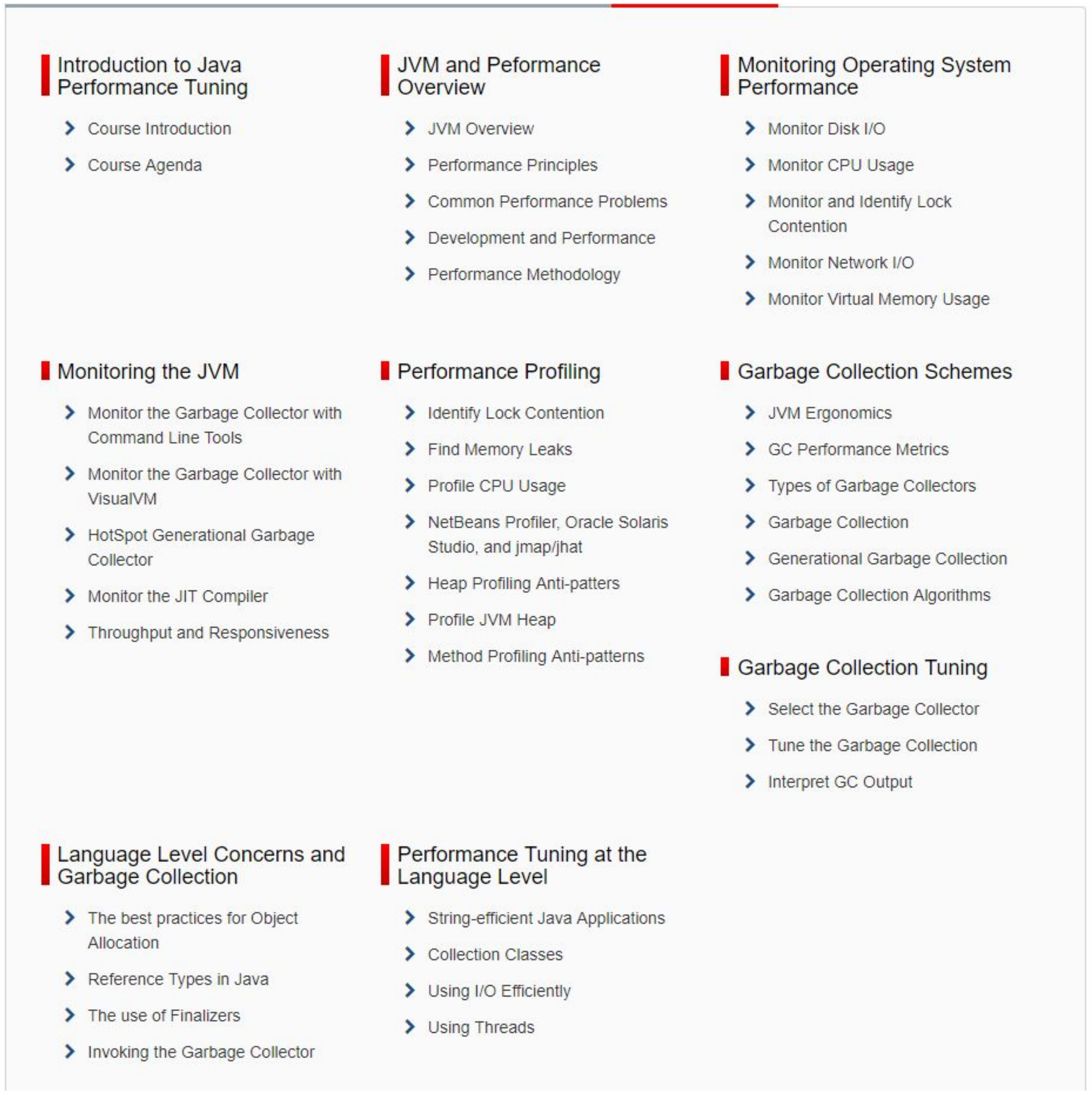
Murat Öngüdü

Software Craftsman, Change Agent

15+ years experience with wearing many hats.

Oracle Course: Java Performance Tuning and Optimization

https://education.oracle.com/java-performance-tuning-and-optimization/courP_1519



Introduction to Java Performance Tuning
- Course Introduction
- Course Agenda

JVM and Peformance Overview
- JVM Overview
- Performance Principles
- Common Performance Problems
- Development and Performance
- Performance Methodology

Monitoring Operating System Performance
- Monitor Disk I/O
- Monitor CPU Usage
- Monitor and Identify Lock Contention
- Monitor Network I/O
- Monitor Virtual Memory Usage

Monitoring the JVM
- Monitor the Garbage Collector with Command Line Tools
- Monitor the Garbage Collector with VisualVM
- HotSpot Generational Garbage Collector
- Monitor the JIT Compiler
- Throughput and Responsiveness

Performance Profiling
- Identify Lock Contention
- Find Memory Leaks
- Profile CPU Usage
- NetBeans Profiler, Oracle Solaris Studio, and jmap/jhat
- Heap Profiling Anti-patters
- Profile JVM Heap
- Method Profiling Anti-patterns

Garbage Collection Schemes
- JVM Ergonomics
- GC Performance Metrics
- Types of Garbage Collectors
- Garbage Collection
- Generational Garbage Collection
- Garbage Collection Algorithms

Garbage Collection Tuning
- Select the Garbage Collector
- Tune the Garbage Collection
- Interpret GC Output

Language Level Concerns and Garbage Collection
- The best practices for Object Allocation
- Reference Types in Java
- The use of Finalizers
- Invoking the Garbage Collector

Performance Tuning at the Language Level
- String-efficient Java Applications
- Collection Classes
- Using I/O Efficiently
- Using Threads

Agenda

- Optimization that matters
- JVM, Heap, Garbage Collector
- Responsiveness & Throughput
- Java Visual VM
- Get Heap Dump
- Get Thread Dump
- IBM Thread and Monitor Analyzer for Java
- Eclipse Memory Analyzer (MAT)
- Sampling, Profiling

Optimization that matters

1. When should I optimize?
2. What should I optimize?
3. When should I stop optimizing?

Premature Optimization

- We should forget about small efficiencies, say about 97% of the time; premature optimization is the root of all evil.

  Donald Knuth

- Don’t let out-of-context dogma from pioneering heroes prevent you from thinking about the code you are writing.

  Java Performance: The Definitive Guide

  Scott Oaks

Optimization that matters

Java 1.1.8 performance was eight times faster than Java 1.0 performance.

Optimize for the common case

01

Write Better Algorithms

(Consider Big O)

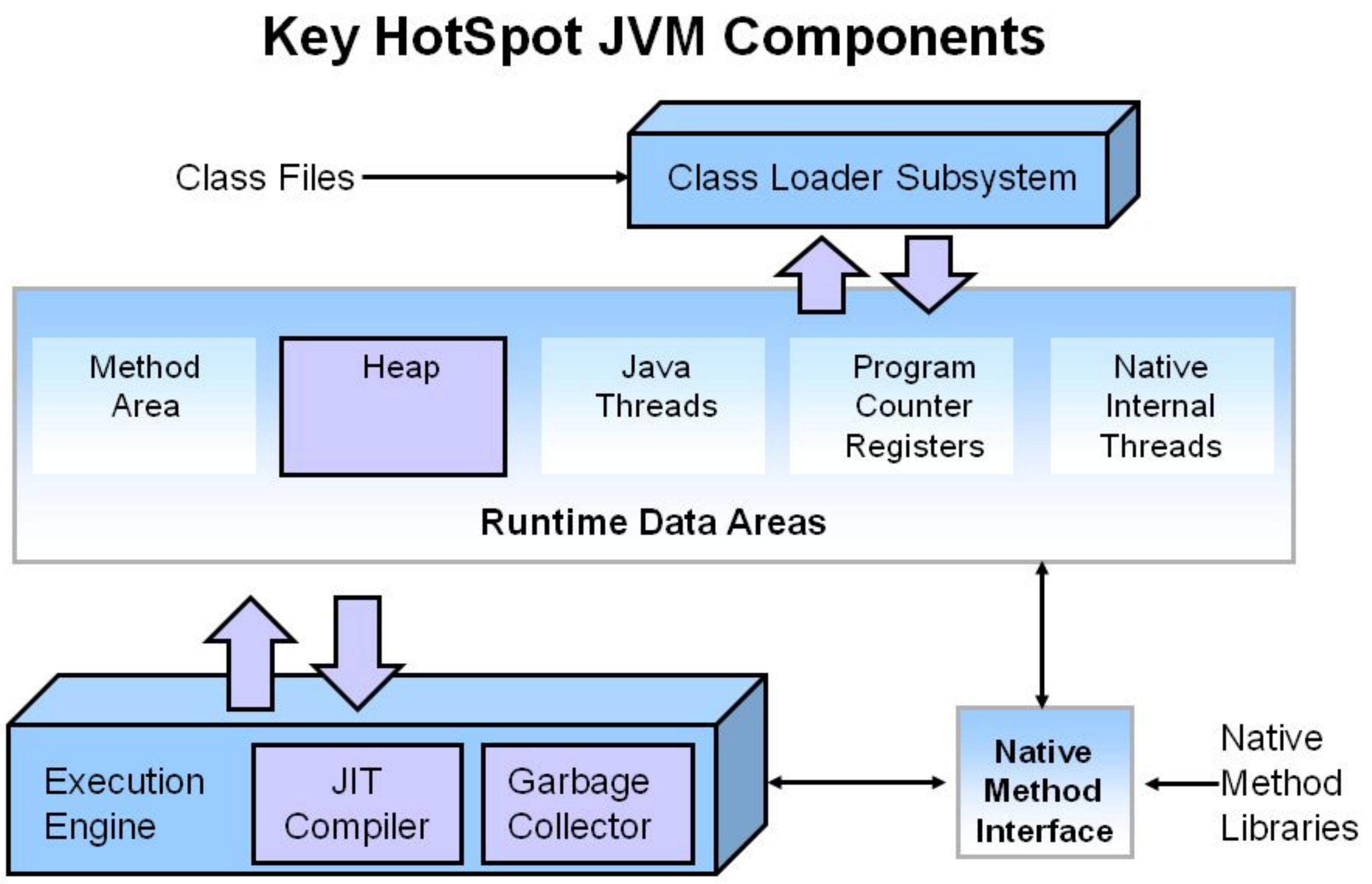
02

Write Less Code

03

Look elsewhere: Database, File IO, Network IO, etc.

Java Performance, The Definitive Guide.



https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Heap

- Memory for all class instances and arrays is allocated.
- May be of a fixed size or may be expanded as required.
- Heap storage for objects is reclaimed by an automatic storage management system (known as a *garbage collector*); objects are never explicitly deallocated.

Manual Memory Management

```
int send_request() {
   size_t n = read_size();
   int *elements = malloc(n * sizeof(int));

   if(read_elements(n, elements) < n) {
      // elements not freed!
      return -1;
   }

   // …

   free(elements)
   return 0;
}
```

https://plumbr.io/handbook/what-is-garbage-collection

Automatic Garbage Collection

Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects.

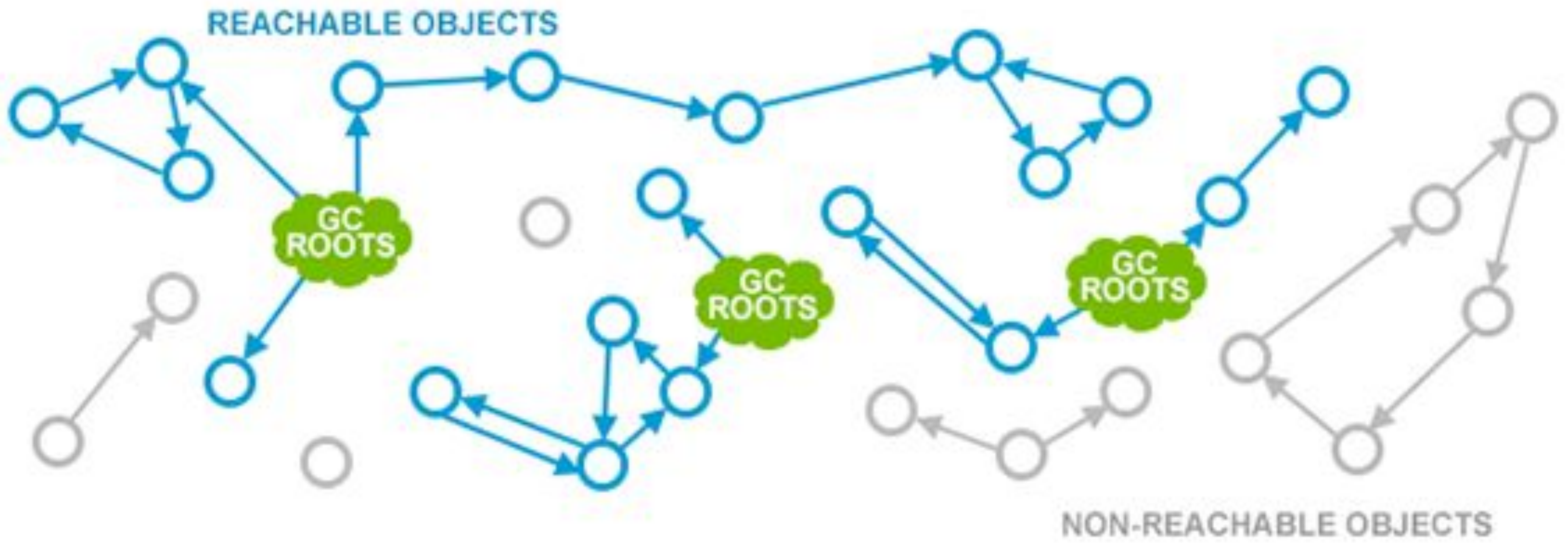
Unused objects = no reference (Soft references doesn’t count)

Stops the world in application (Pause Time)

Types: Minor GC, Major GC

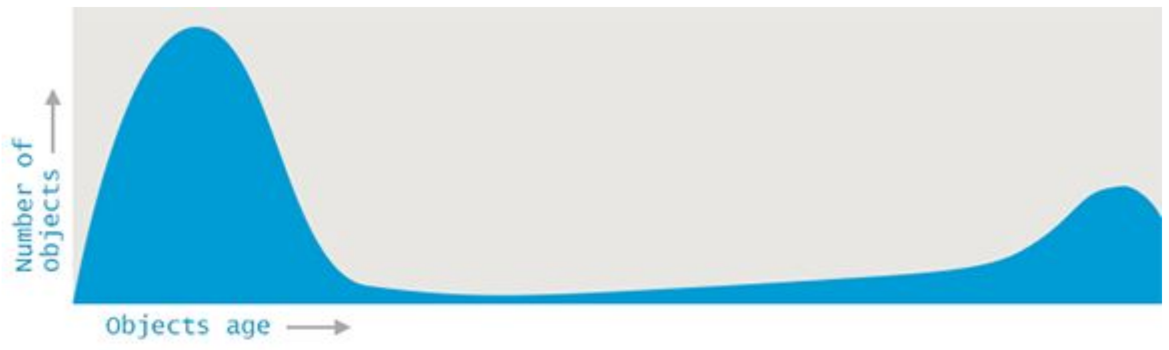
Many Implemantations: …..

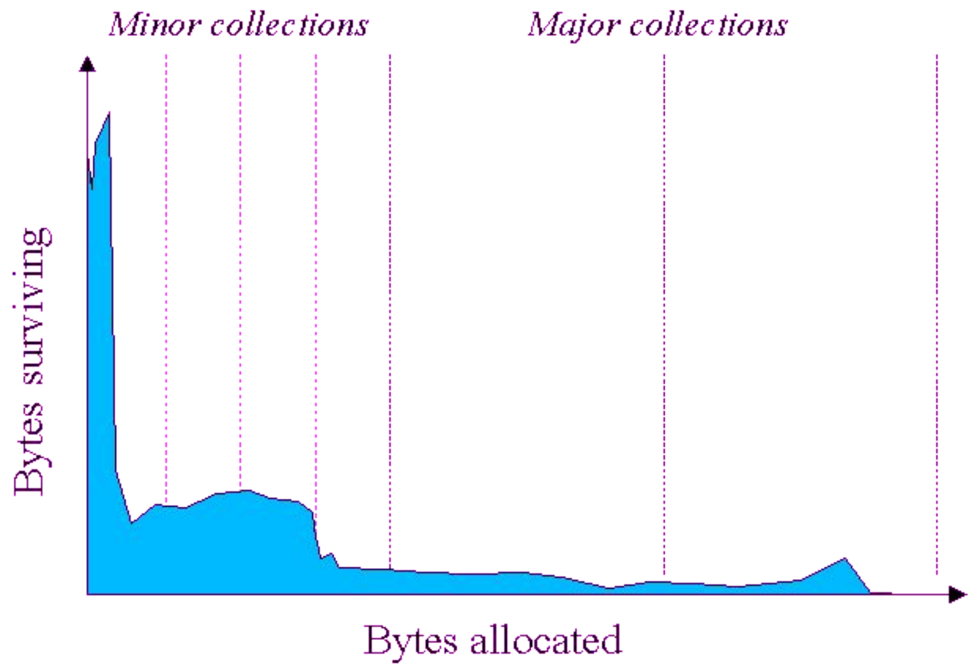
Mark, Sweep



https://plumbr.io/handbook/what-is-garbage-collection

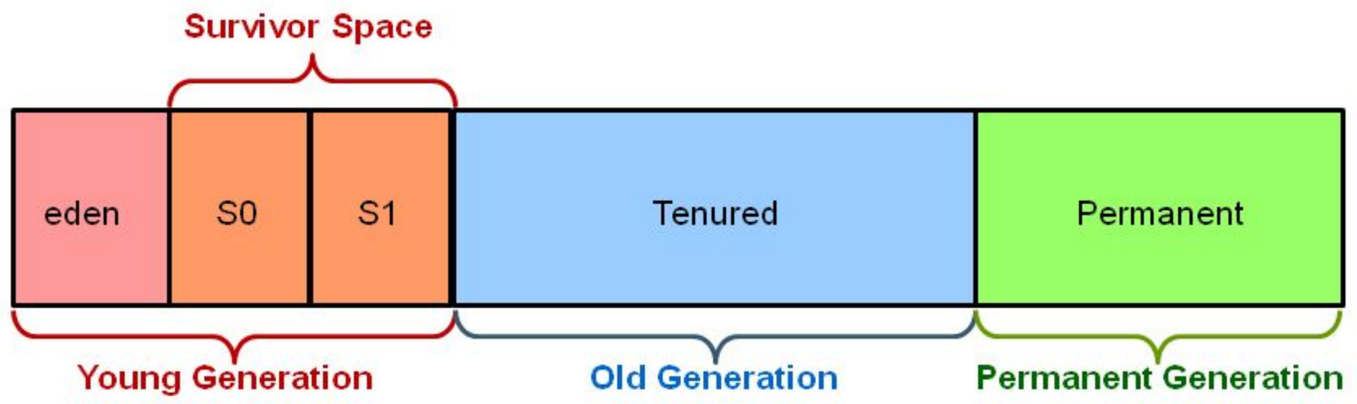
Why Generational Garbage Collection?





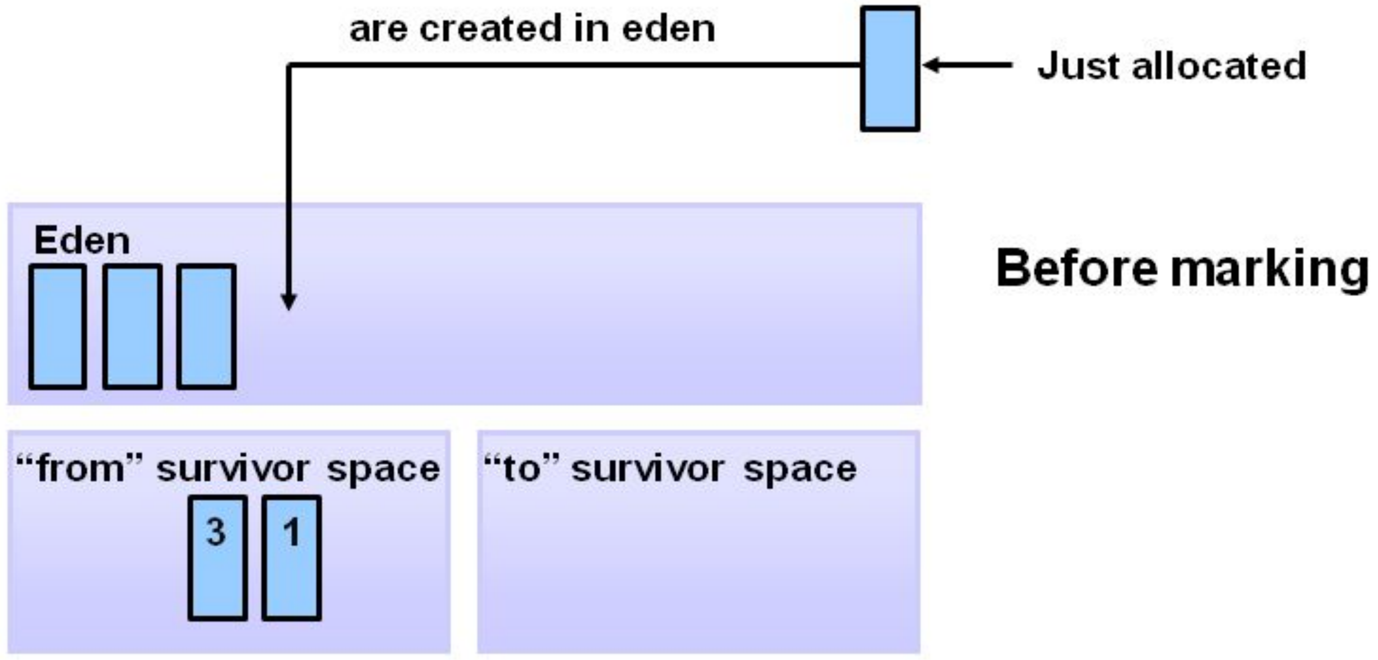
https://plumbr.io/handbook/garbage-collection-in-java
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Hotspot Heap Structure



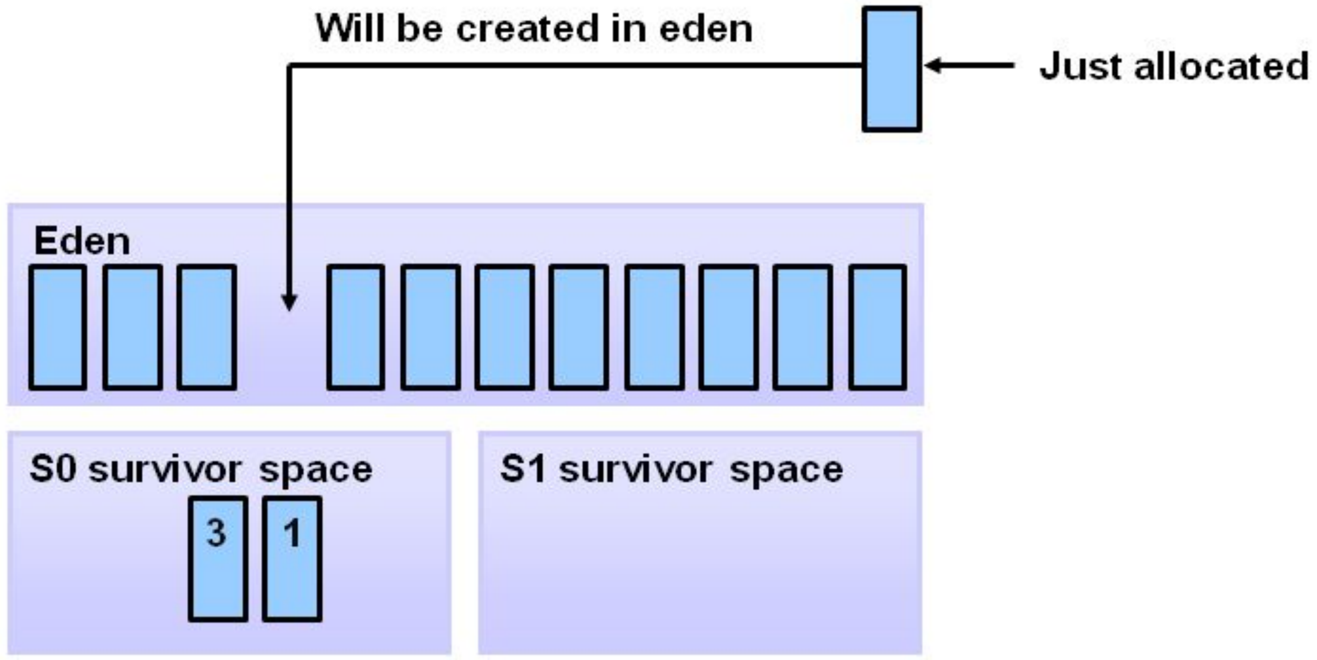
Prior to Java 8

Object Allocation



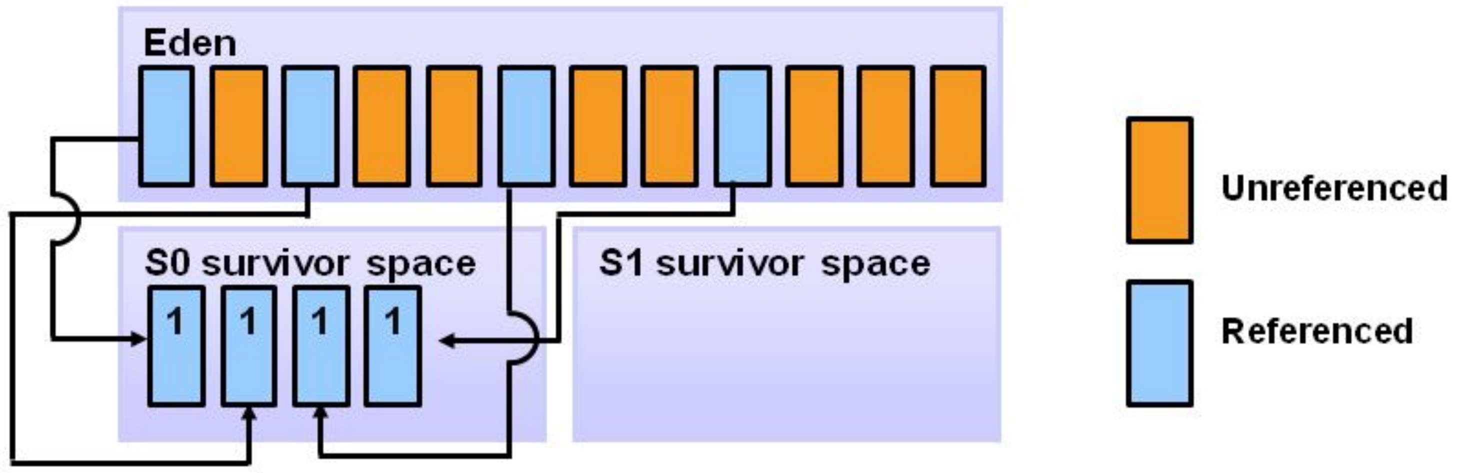
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Filling the Eden Space



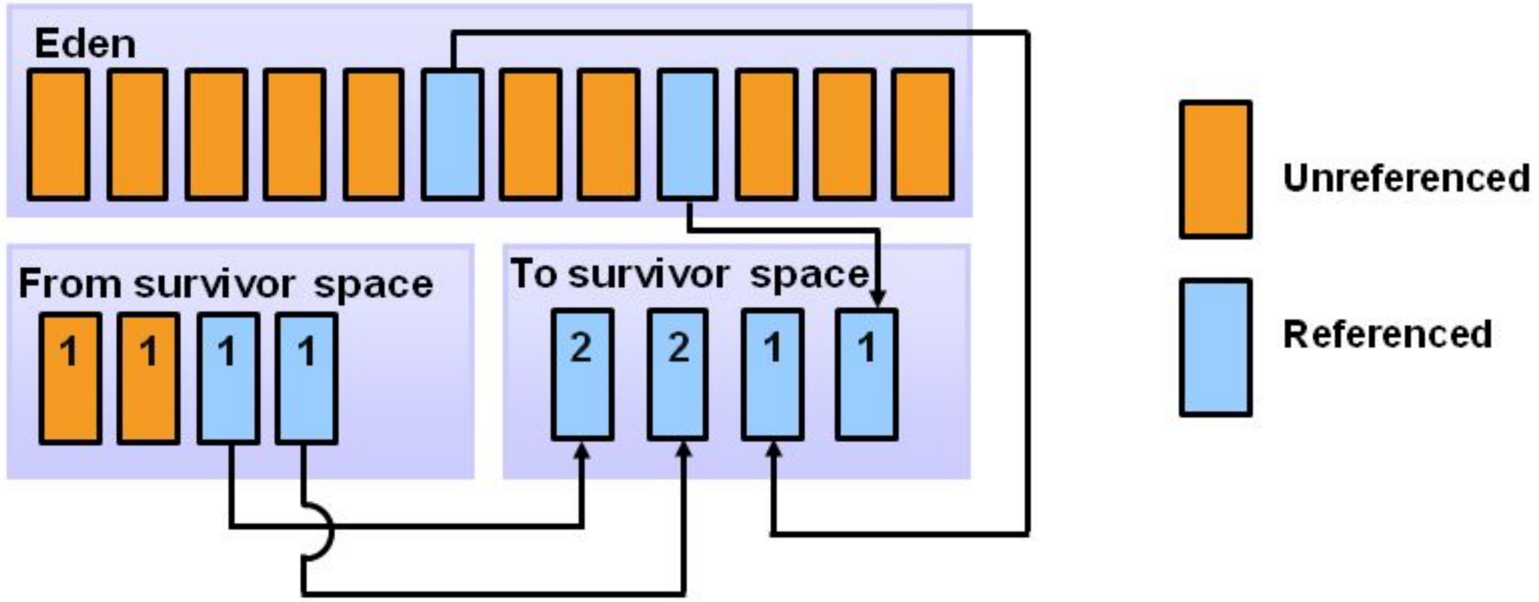
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Copying Referenced Objects



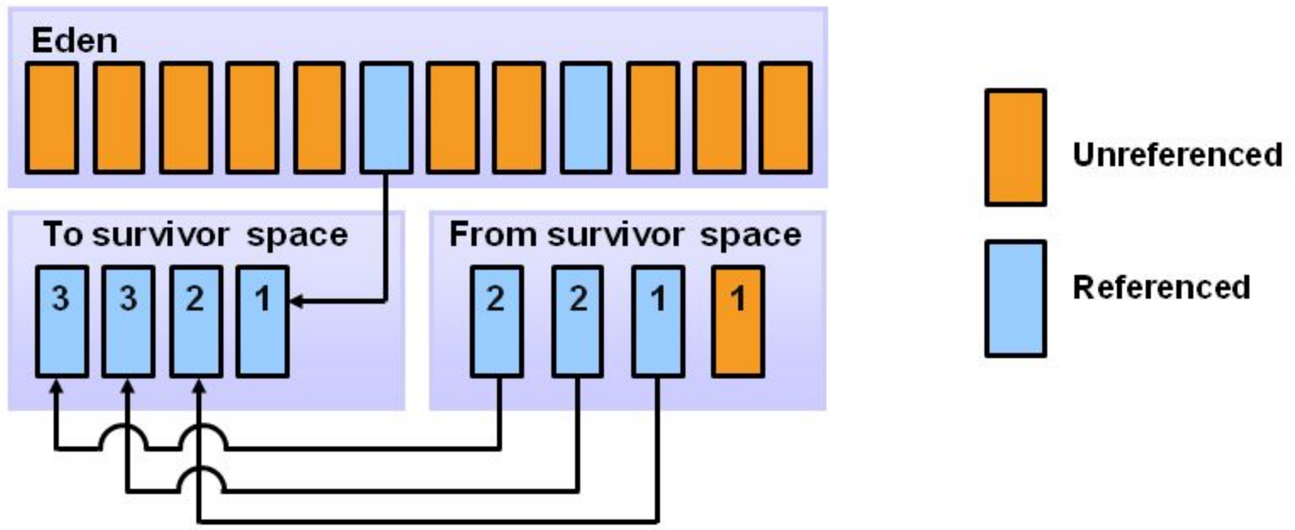
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Object Aging



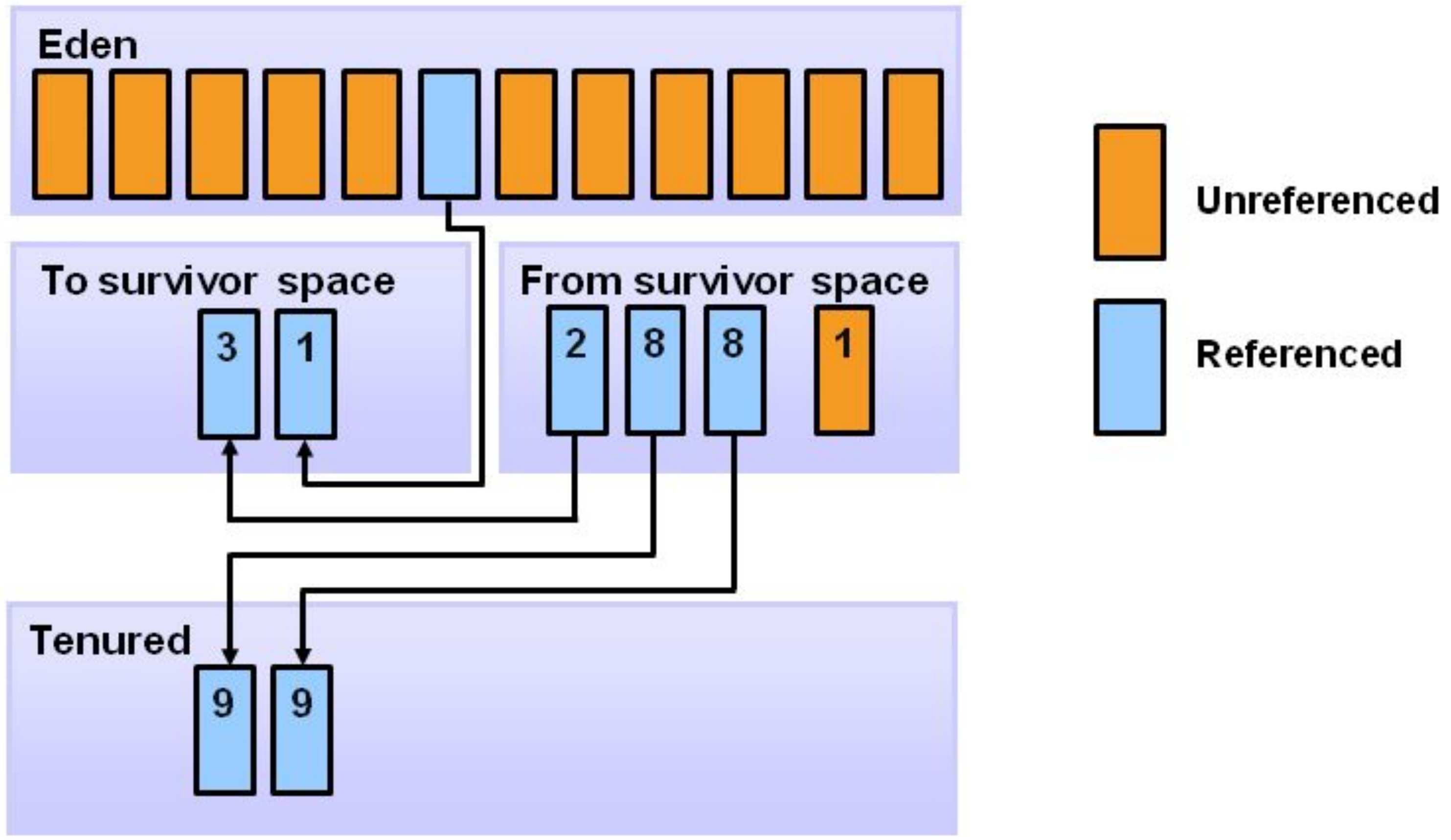
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Additional Aging



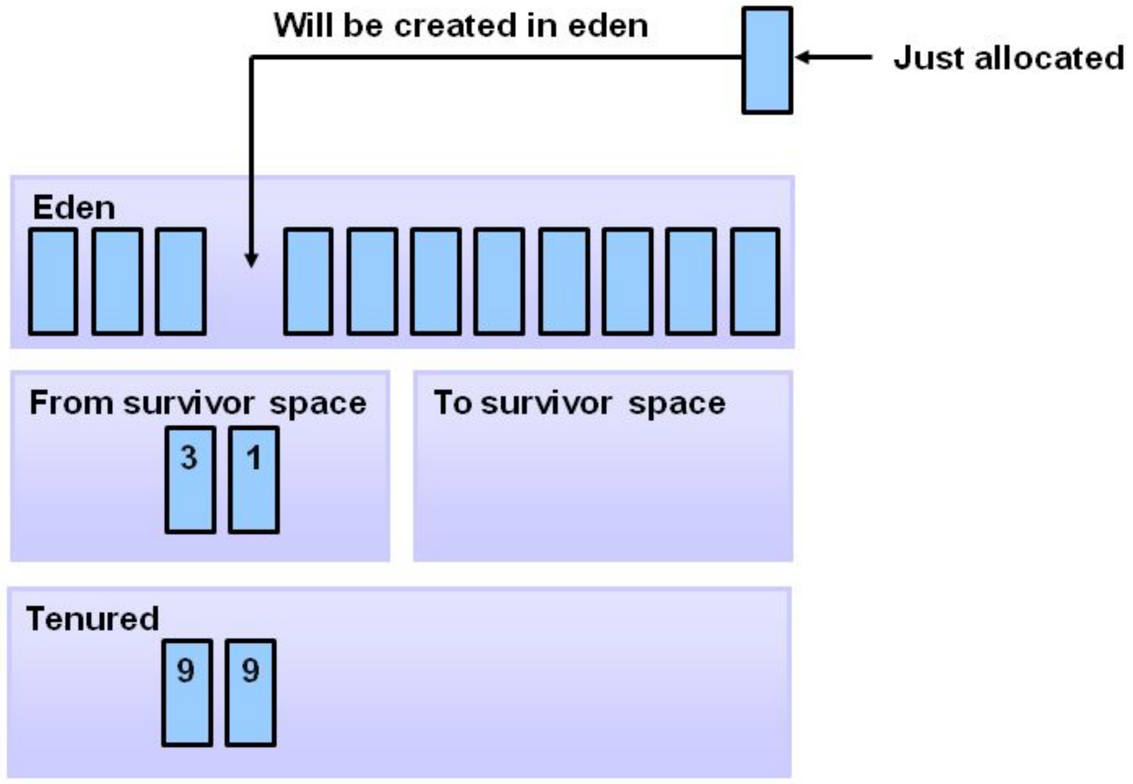
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Promotion



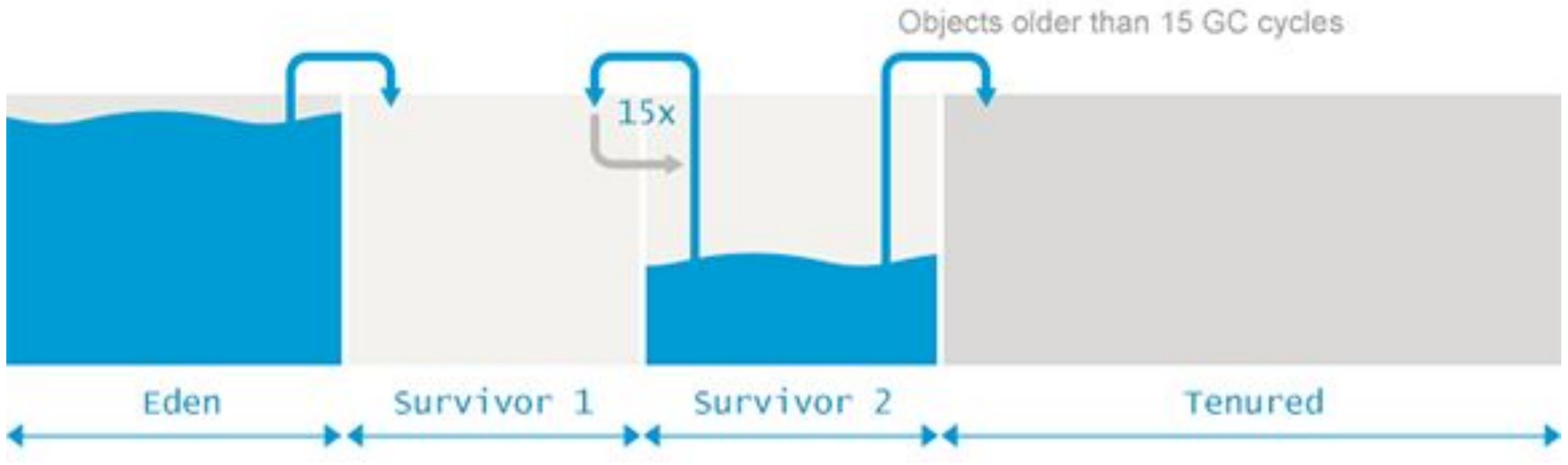
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

GC Process Summary



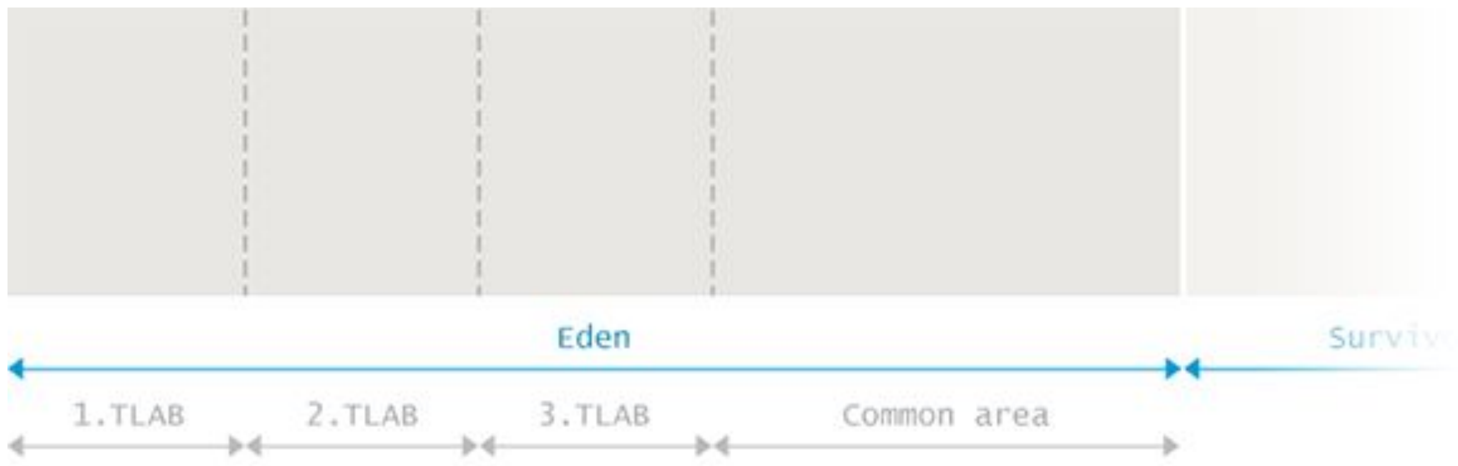
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Object Aging



https://plumbr.io/handbook/garbage-collection-in-java

Thread Local Allocation Buffer (TLAB)

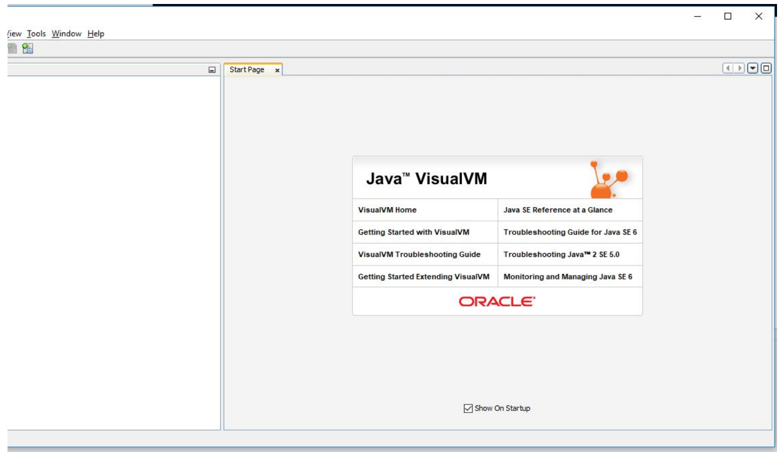


https://plumbr.io/handbook/garbage-collection-in-java

Responsiveness & Throughput

Considering pause time

Java VisualVM


iew Tools Window Help
Start Page
Java™ VisualVM
VisualVM Home
Java SE Reference at a Glance
Getting Started with VisualVM
Troubleshooting Guide for Java SE 6
VisualVM Troubleshooting Guide
Troubleshooting Java™ 2 SE 5.0
Getting Started Extending VisualVM
Monitoring and Managing Java SE 6
ORACLE
Show On Startup

- Looking inside your application
- Useful plugins
- Local & Remote Connection

Opening JMX Port for Remote

- java -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=1617 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -cp target\classes com.murat.memoryproblem.ApplicationProblem

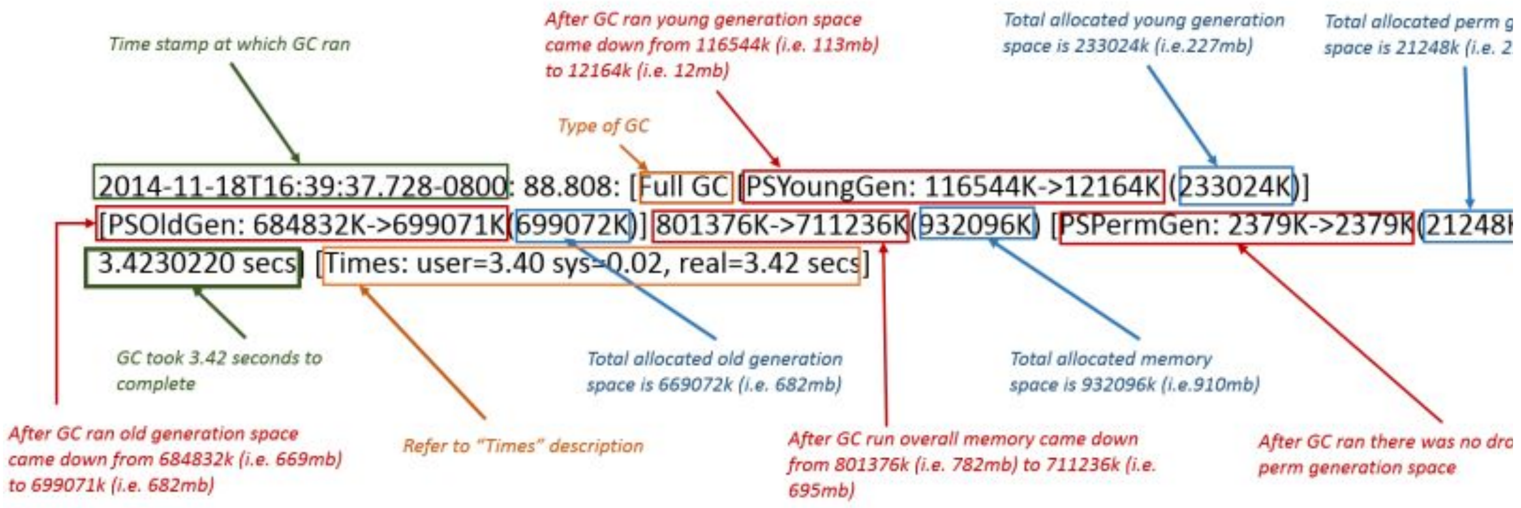
Seeing garbage collector activity in a java app

java -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -Xloggc:<filename> -cp target\classes com.murat.memoryproblem.ApplicationProblem

java -verbose:gc -cp .\classes com.murat.memoryproblem.ApplicationProblem

Get Statistics:

- jstat -gc -t PID 1ss



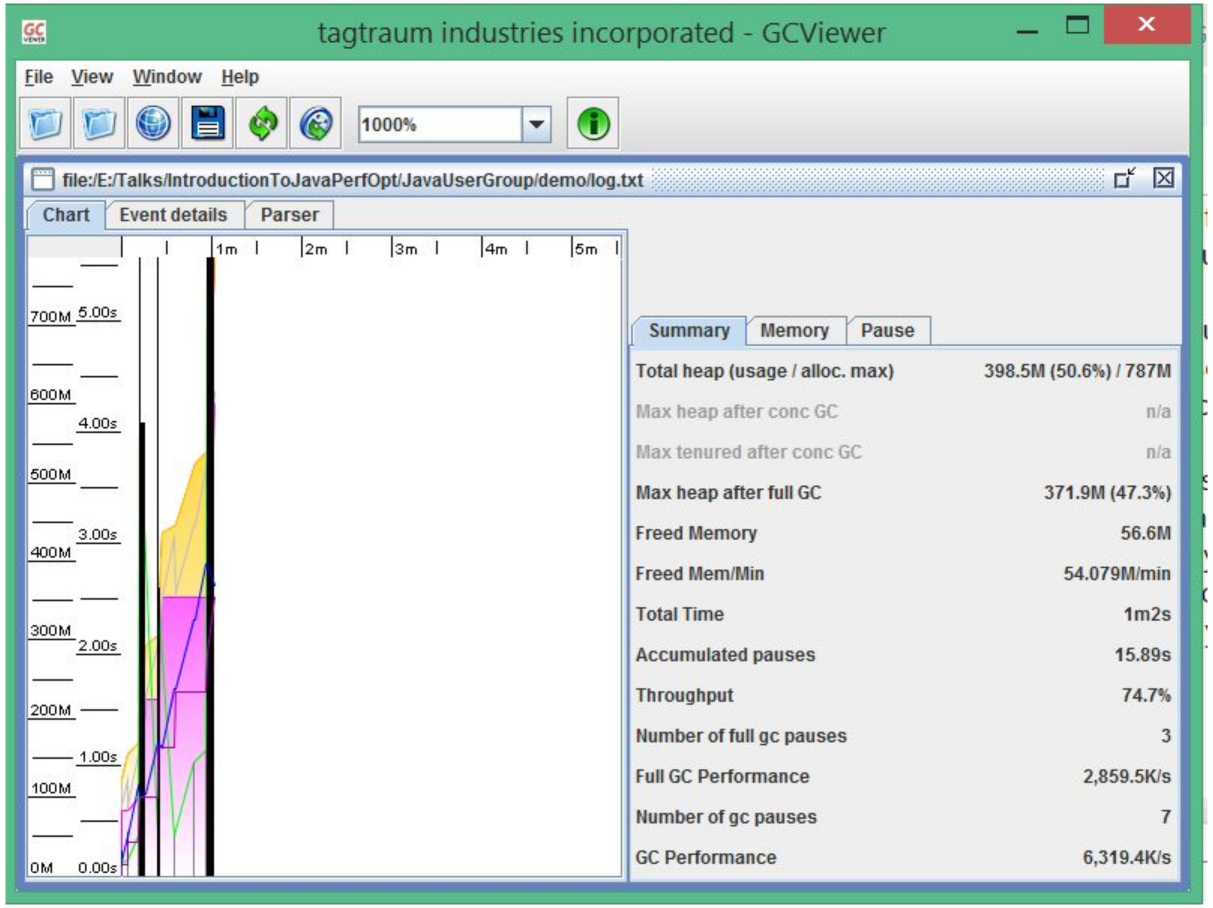
```
2014-11-18T16:39:37.728-0800: 88.808: [Full GC [PSYoungGen: 116544K->12164K(233024K)]
[PSOldGen: 684832K->699071K(699072K)] 801376K->711236K(932096K) [PSPermGen: 2379K->2379K(21248
3.4230220 secs] [Times: user=3.40 sys=0.02, real=3.42 secs]
```

Time stamp at which GC ran
After GC ran young generation space came down from 116544k (i.e. 113mb) to 12164k (i.e. 12mb)
Type of GC
Total allocated young generation space is 233024k (i.e.227mb)
Total allocated perm g space is 21248k (i.e. 2
GC took 3.42 seconds to complete
Total allocated old generation space is 669072k (i.e. 682mb)
Total allocated memory space is 932096k (i.e.910mb)
After GC ran old generation space came down from 684832k (i.e. 669mb) to 699071k (i.e. 682mb)
Refer to "Times" description
After GC run overall memory came down from 801376k (i.e. 782mb) to 711236k (i.e. 695mb)
After GC ran there was no dro perm generation space

https://blog.gceasy.io/2016/02/22/understand-garbage-collection-log/

Visualising garbage collector

GCViewer


tagtraum industries incorporated - GCViewer
File View Window Help
1000%
file:/E:/Talks/IntroductionToJavaPerfOpt/JavaUserGroup/demo/log.txt
Chart Event details Parser
Summary Memory Pause
700M 600M 500M 400M 300M 200M 100M 0M
5.00s 4.00s 3.00s 2.00s 1.00s 0.00s
1m 2m 3m 4m 5m

Total heap (usage / alloc. max)	398.5M (50.6%) / 787M
Max heap after conc GC	n/a
Max tenured after conc GC	n/a
Max heap after full GC	371.9M (47.3%)
Freed Memory	56.6M
Freed Mem/Min	54.079M/min
Total Time	1m2s
Accumulated pauses	15.89s
Throughput	74.7%
Number of full gc pauses	3
Full GC Performance	2,859.5K/s
Number of gc pauses	7
GC Performance	6,319.4K/s

https://github.com/chewiebug/GCViewer

Heap Dump

- Get Java Process Id
  - jps
- Get Heap Dump
  - jmap -dump:format=b,file=<file_path> pid

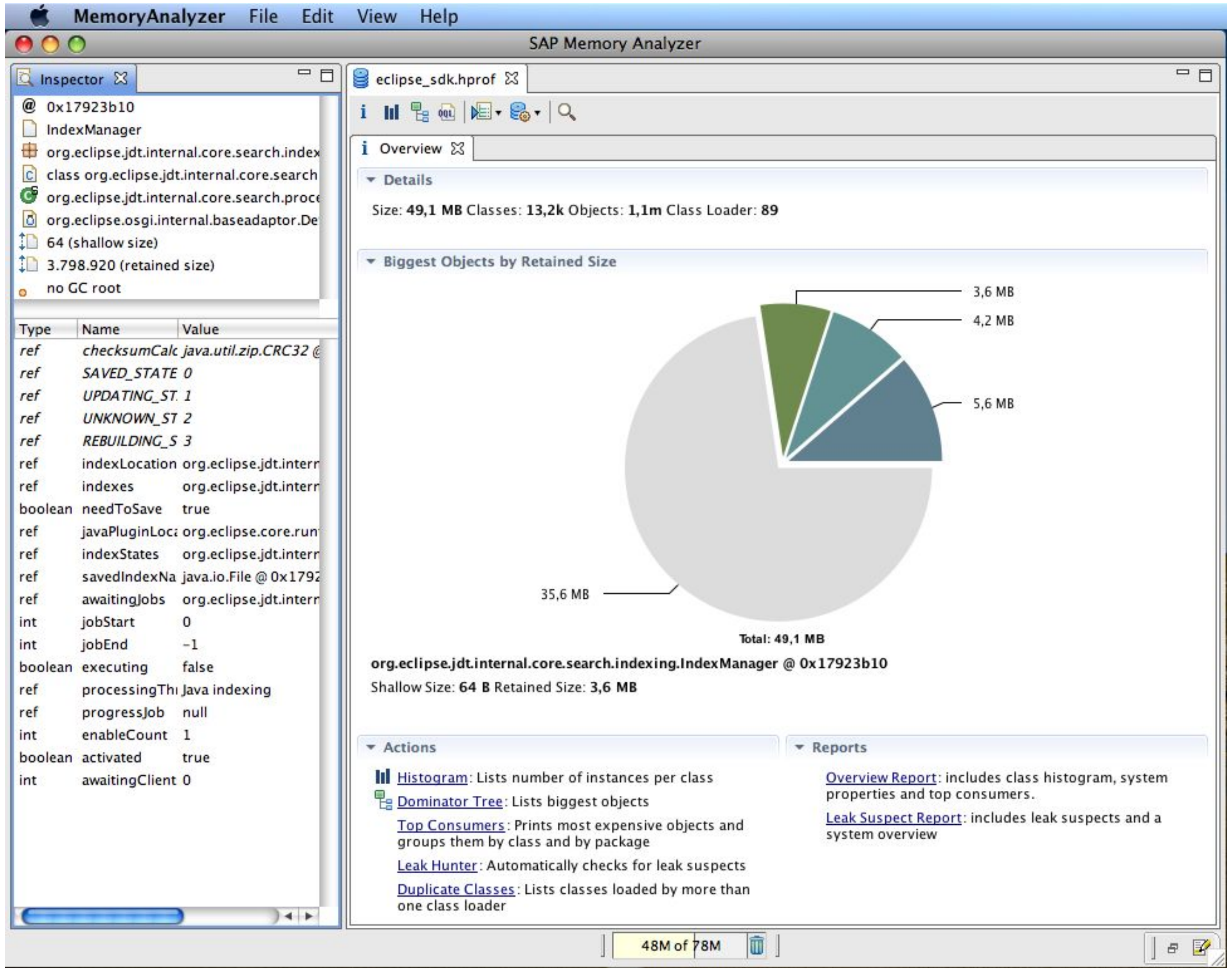
  or
  - jcmd pid GC.heap_dump <file_path>
- Get Heap Dump on out of memory
  - -XX:+HeapDumpOnOutOfMemoryError

    -XX:HeapDumpPath=path

jcmd

- Get Java Process Id
  - jps
- Get Heap Dump
  - jcmd pid GC.heap_dump <file_path>
- Run GC
  - jcmd pid GC.run <file_path>
- Many Useful commands
  - jcmd pid help

Eclipse Memory Analyzer (MAT)



0x17923b10
IndexManager
org.eclipse.jdt.internal.core.search.index
class org.eclipse.jdt.internal.core.search
org.eclipse.jdt.internal.core.search.proce
org.eclipse.osgi.internal.baseadaptor.De
64 (shallow size)
3.798.920 (retained size)
no GC root

Type	Name	Value
ref	*checksumCalc*	*java.util.zip.CRC32 @*
ref	*SAVED_STATE*	*0*
ref	*UPDATING_ST*	*1*
ref	*UNKNOWN_ST*	*2*
ref	*REBUILDING_S*	*3*
ref	indexLocation	org.eclipse.jdt.intern
ref	indexes	org.eclipse.jdt.intern
boolean	needToSave	true
ref	javaPluginLoc	org.eclipse.core.run
ref	indexStates	org.eclipse.jdt.intern
ref	savedIndexNa	java.io.File @ 0x1792
ref	awaitingJobs	org.eclipse.jdt.intern
int	jobStart	0
int	jobEnd	-1
boolean	executing	false
ref	processingThr	Java indexing
ref	progressJob	null
int	enableCount	1
boolean	activated	true
int	awaitingClient	0

Details

Size: **49,1 MB** Classes: **13,2k** Objects: **1,1m** Class Loader: **89**

Biggest Objects by Retained Size

org.eclipse.jdt.internal.core.search.indexing.IndexManager @ 0x17923b10

Shallow Size: **64 B** Retained Size: **3,6 MB**

Actions

- Histogram: Lists number of instances per class
- Dominator Tree: Lists biggest objects
- Top Consumers: Prints most expensive objects and groups them by class and by package
- Leak Hunter: Automatically checks for leak suspects
- Duplicate Classes: Lists classes loaded by more than one class loader

Reports

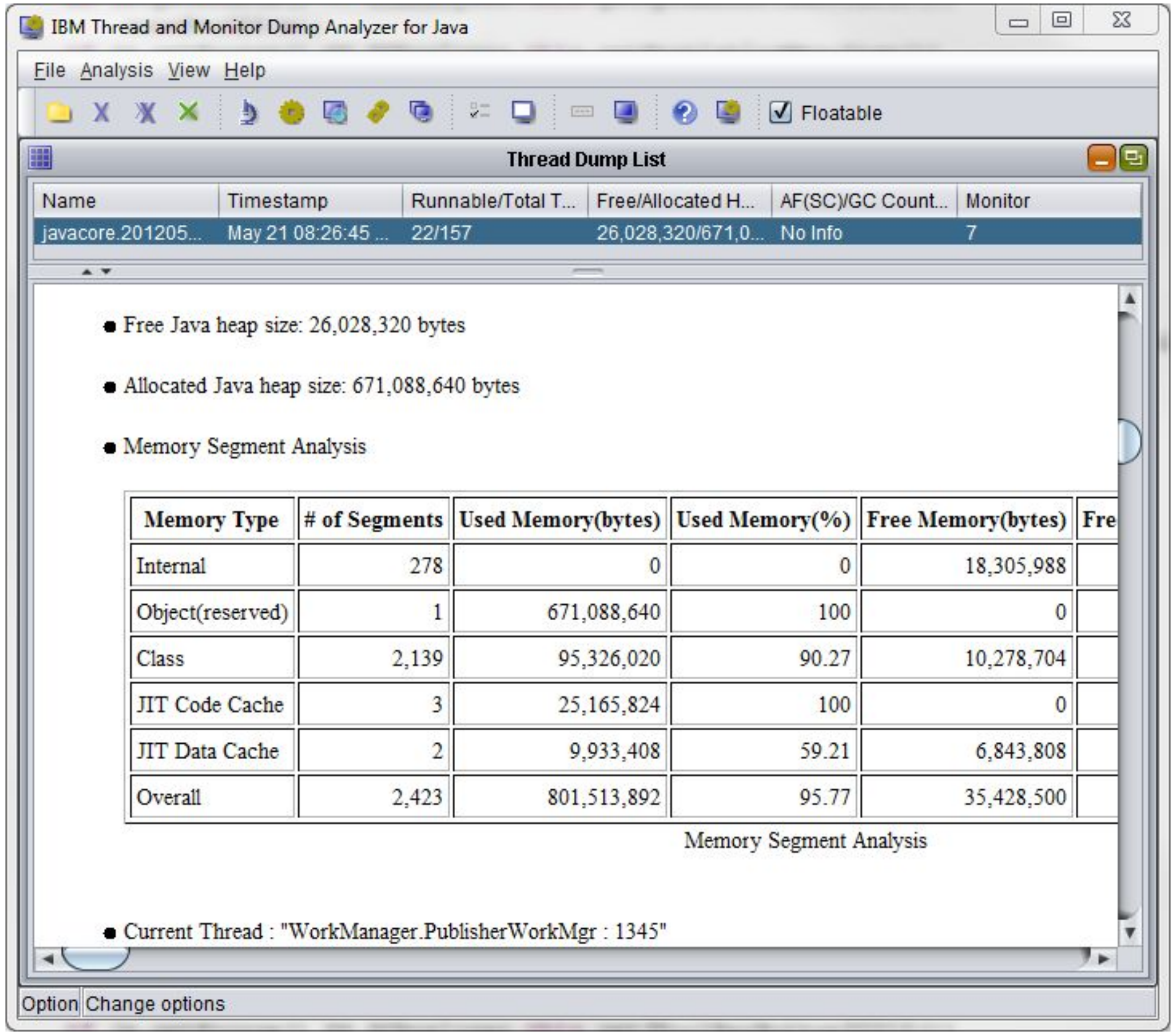
- Overview Report: includes class histogram, system properties and top consumers.
- Leak Suspect Report: includes leak suspects and a system overview

Thread Dump

- Get Java Process Id
  - jps
- Get Thread Dump
  - jstack pid > <file_path>

  or
  - jcmd pid Thread.print <file_path> > <file_path>

IBM Thread and Monitor Dump Analyzer for Java



Name	Timestamp	Runnable/Total T...	Free/Allocated H...	AF(SC)/GC Count...	Monitor
javacore.201205...	May 21 08:26:45 ...	22/157	26,028,320/671,0...	No Info	7

- Free Java heap size: 26,028,320 bytes
- Allocated Java heap size: 671,088,640 bytes
- Memory Segment Analysis

Memory Type	# of Segments	Used Memory(bytes)	Used Memory(%)	Free Memory(bytes)	Fre
Internal	278	0	0	18,305,988	
Object(reserved)	1	671,088,640	100	0	
Class	2,139	95,326,020	90.27	10,278,704	
JIT Code Cache	3	25,165,824	100	0	
JIT Data Cache	2	9,933,408	59.21	6,843,808	
Overall	2,423	801,513,892	95.77	35,428,500	

Memory Segment Analysis

- Current Thread : "WorkManager.PublisherWorkMgr : 1345"

Sampling vs Profiling

- Sampling: periodically statistical data, low overhead
- Profiling: Instrumentation (Probes), overhead and exact numbers.

References

In English:

- Java Performance, The Definitive Guide.
- https://www.oracle.com/technetwork/articles/javase/monitoring-141801.html
- https://plumbr.io/handbook/garbage-collection-in-java

In Turkish:

- http://www.javaturk.org/?s=performans
- http://www.kurumsaljava.com/yazilim-metotlari/performans/

Thank you