JVM Performance Optimisation Training Summary (Sept 2024 by [yCrash](https://ycrash.io/java-performance-training))

# Performance KPI (Key Performance Index)

1 – Throughput

* Percentage of work-related tasks done. E.g. in 24 hours, a system spends a total of 5 minutes for GC. ∴throughput = (100 - ( 5 / ( 24 x 60)) x 100)% = 99.652%.

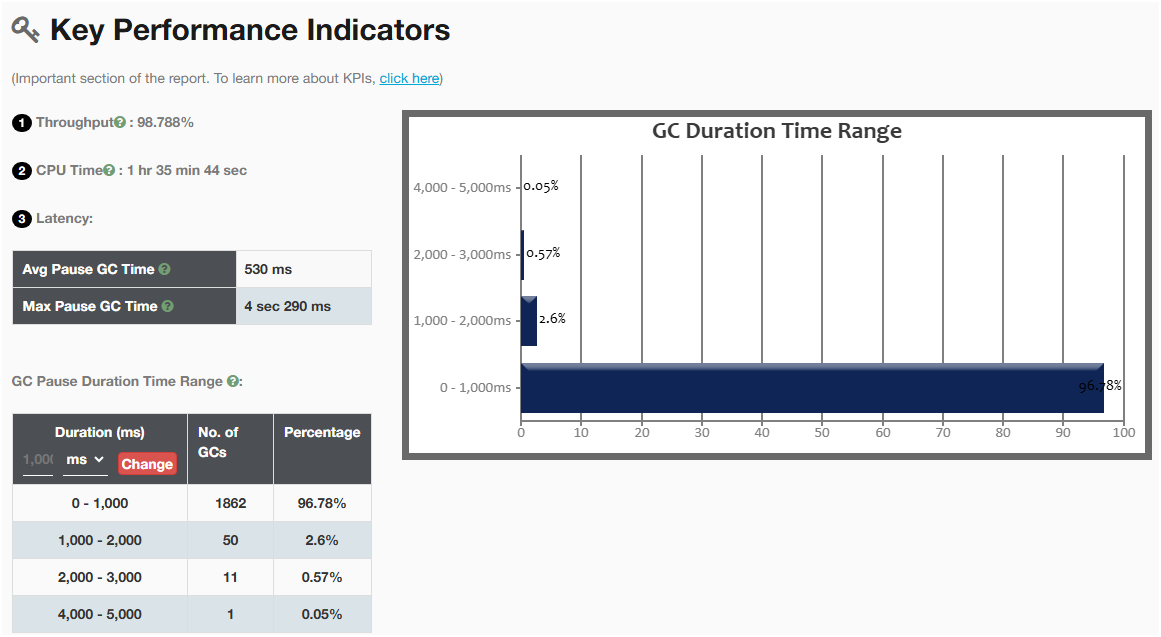
2 – Latency

* Amount of time taken for GC: maximum, average, distribution.

3 – Footprint

* Amount of CPU time taken for GC.

[GCEasy](https://gceasy.io/) can be used to display the information based on GC log as shown below. See [GC log section](#_GC_log:) for more details on GC log).



# Performance Problems

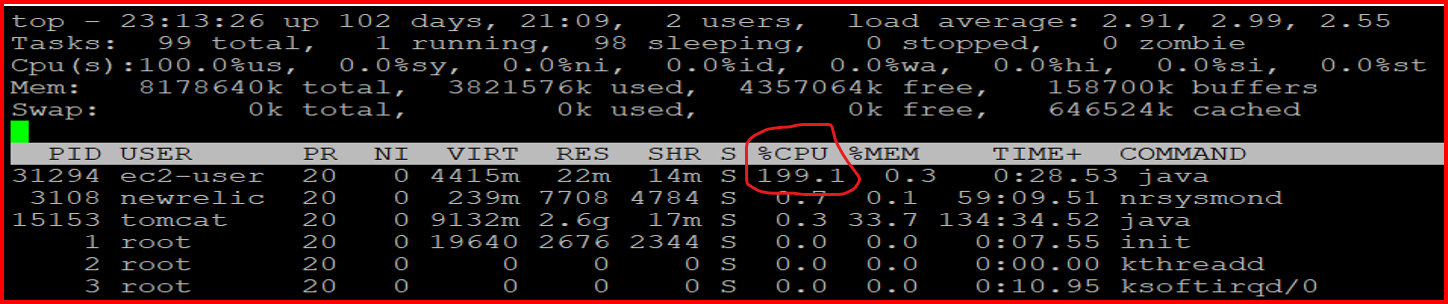
## CPU Spike

### Why

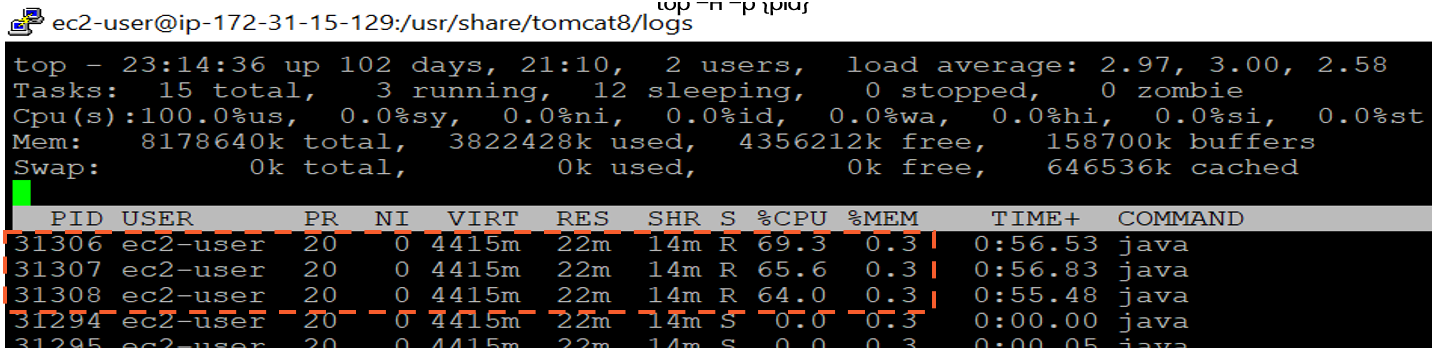
* Blocked thread(s)

### How to Solve

1. Using *top* tool, confirm that there is CPU spike. Command: *# top -H -p <pid>*



1. Identify which threads caused the spike.



1. Collect thread dump and lookup these threads: JMC, jcmd, JVisualVM
2. Identify and resolve lines of code that causes the blocking from stack trace of the identified threads. This can be also done in thread analysis tools: JVisualVM, FastThread.

## OutOfMemoryError

### Why

Not enough certain type of memory specified in the type of the OOME - specified in *java.lang.OutOfMemoryError: <type>*, where *type* can be:

* Java heap space
* GC overhead limit exceeded
* Requested array size exceed VM limit
* Permgen space
* Metaspace
* Unable to create new native thread
* Kill process or sacrifice child
* Reason stack\_trace\_with\_native method

Memory leak can be the main cause.

### How to Solve

1. Capture heap dumps: jmap, jcmd
2. Analyse to look for root cause: HeapHero, JVisualVM

See [Heap Dump section](#_Heap_Dump) for more details.

## StackOverflowError

### Why

Not enough stack size configured.

### How to Solve

Adjust stack size using JVM argument -Xss.

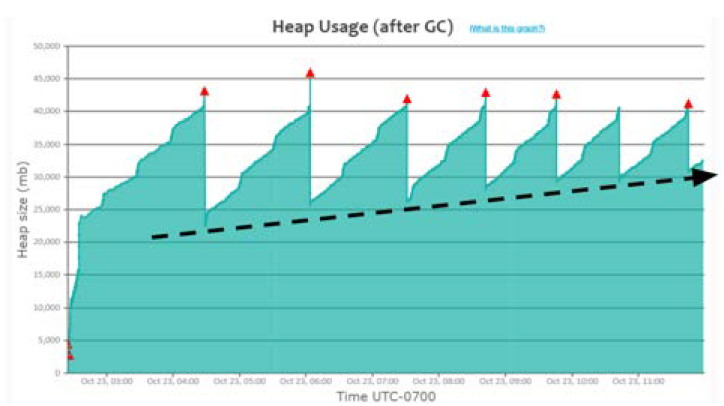
# GC Patterns

## Healthy GC (sawtooth)

A graph showing the amount of air in the air

Description automatically generated with medium confidence

## Acute Memory Leak (uptrend)

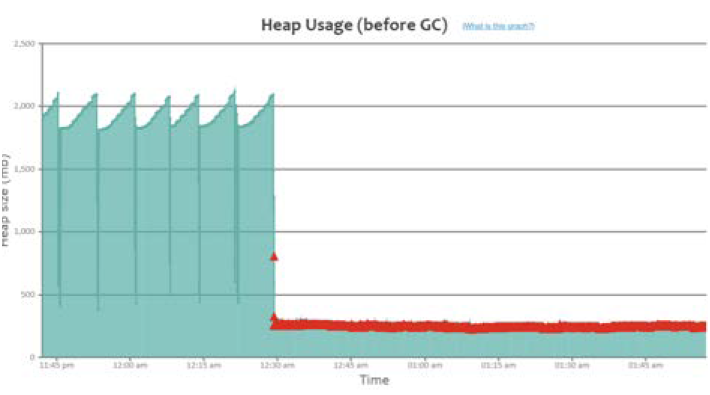


## Heavy Caching

A graph showing a number of blue dots

Description automatically generated with medium confidence

## Metaspace memory problem



# Useful JVM Arguments for Optimisation

## Heap

Use one of the followings to specify your application heap size:

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Xmx | Supported in all versions of JDK.  Example:  # java -Xmx512m -XshowSettings:vm -version |
| -XX:MaxRAMFraction  -XX:MinRAMFraction | Only available in JDK8u131-190.  Recommended for containers. Should be used in conjunction with:  -XX:+UnlockExperimentalVMOptions  -XX:+UseCGroupMemoryLimitForHeap  Example:  # docker run -m 1GB openjdk:8u131 java  -XX:+UnlockExperimentalVMOptions  -XX:+UseCGroupMemoryLimitForHeap  -XX:MaxRAMFraction=2  -XshowSettings:vm -version |
| -XX:MaxRAMPercentage  -XX:MinRAMPercentage | Available in JDK8u191 and above.  Recommended for app deployed in containers.  Example:  # docker run -m 1GB openjdk:10 java  -XX:MaxRAMPercentage=50  -XshowSettings:vm -version |

## Metaspace

* Region where class definitions, method definitions, and other JVM metadata are stored.

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Xx:MaxMetaspaceSize | Example:  -XX:MaxMetaspaceSize=256m |

## Stack:

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Xss | Specifies stack size. Adjust with knowledge of total number of thread as value is per thread.  Example:  -Xss256k |

## GC selection

Use one of the followings:

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -XX:+UseSerialGC | To use Serial GC algorithm. |
| -XX:+UseParallelGC | To use Parallel GC algorithm. |
| -XX:+UseConcMarkSweepGC | To use CMS GC algorithm. |
| -XX:+ UseG1GC | To use G1GC GC algorithm. |
| -XX:+ UseShenandoahGC | To use Shenandoah GC algorithm. |
| -XX:+ UseZGC | To use Z GC algorithm.  Starting JDK23:  To use Z GC Generational, add:  -XX:+ZGenerational  Or, to use non-generational, add:  -XX:-ZGenerational |

### When to Use What

Guide taken from [here](https://www.youtube.com/watch?v=2AZ0KKeXJSo&list=PLYe9-n3XGabh2p95ptzEoIbHuM7jdRO5S&index=5&ab_channel=DefogTech).

A screenshot of a computer program

Description automatically generated

With G1GC, you can specify the soft-target for maximum GC pause time using JVM args:

* ***-XX:MaxGCPauseMillis***
* ***-XX:GCPauseIntervalMillis***

## Timeouts

Use any of the followings if needed:

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| ‑Dsun.net.client.defaultConnectTimeout | Timeout to connect to host.  Example:  ‑Dsun.net.client.defaultConnectTimeout=2000 |
| -Dsun.net.client.defaultReadTimeout | Timeout when reading from input stream.  Example:  -Dsun.net.client.defaultReadTimeout=2000 |

# Useful JVM Arguments for Troubleshooting

## GC log:

* Analyse GC log for period of 24 hours during weekdays for high and low traffic monitoring.
* Can be used to troubleshoot GC-related problems: long GC pauses, irresponsive application, low throughput, memory leak indication GC pattern

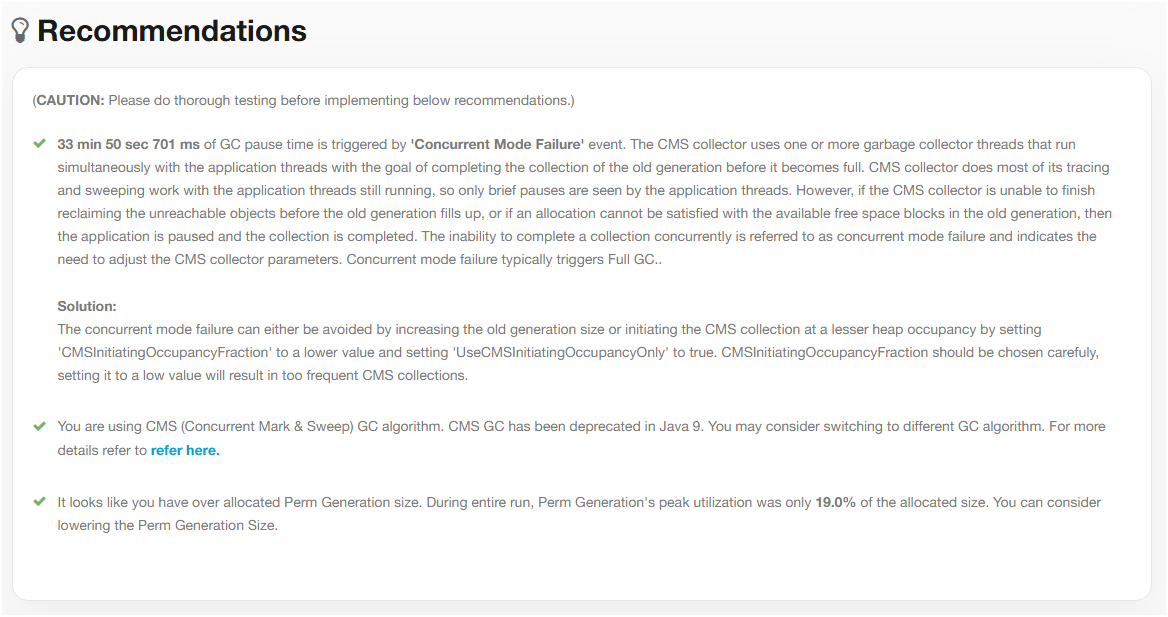
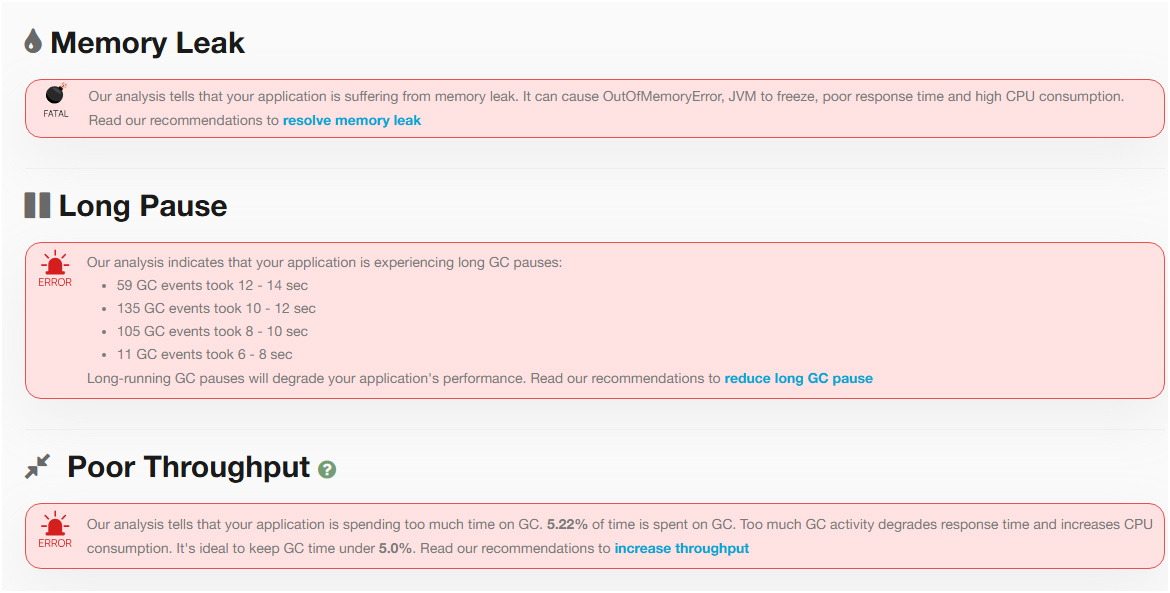
|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -verbose:gc  -Xloggc:<log\_file\_path>  -XX:+PrintGCDetails  -XX:+PrintGCDateStamps | For Java 7 and below  Example:  *java -verbose:gc*  *-Xloggc:/var/log/myapp/gc.log*  *-XX:+PrintGCDetails*  *-XX:+PrintGCDateStamps -jar myapp.jar* |
| -XX:+PrintGC  -XX:+PrintGCDetails  -XX:+PrintGCDateStamps  -Xloggc:<log\_file\_path>  -XX:+UseGCLogFileRotation  -XX:NumberOfGCLogFiles=<number\_of\_files>  -XX:GCLogFileSize=<size>[k|m|g] | For Java 8: it has additional log rotation option.  Example:  *java -XX:+PrintGC –*  *XX:+PrintGCDetails*  *-XX:+PrintGCDateStamps*  *-Xloggc:/var/log/myapp/gc.log*  *-XX:+UseGCLogFileRotation*  *-XX:NumberOfGCLogFiles=5*  *-XX:GCLogFileSize=10m -jar myapp.jar* |
| -Xlog:gc\*:  file=<log\_file\_path>:  time,uptime,level,tags:  filecount=<number\_of\_files>,  filesize=<size>[k|m|g] | For Java 9 and above: it is using unified logging -Xlog  Example:  *java -Xlog:gc\*:*  *file=/var/log/myapp/gc.log:*  *time,uptime,level,tags:*  *filecount=5,*  *filesize=10m -jar myapp.jar* |

### Tools

To analyse GC log, we can use:

* [GCEasy](https://gceasy.io/) by yCrash
* JDK Mission Control
* JVisualVM
* [IBM Health Centre and/or IBM GC and Memory Visualizer](https://www.ibm.com/support/pages/java-sdk-monitoring-and-post-mortem)
* [Garbage Cat](https://github.com/mgm3746/garbagecat)

With [GCEasy](https://gceasy.io/), summary and recommendations are provided like shown below:



## Heap Dump

* Can be used to troubleshoot memory-related problems: slow memory leaks, GC problems, OutOfMemoryError

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -XX:+HeapDumpOnOutOfMemoryError  -XX:HeapDumpPath=<file\_path> | Example:  -XX:+HeapDumpOnOutOfMemoryError  -XX:HeapDumpPath=/opt/tmp/heapdump.hprof |
|  |  |

### Tools

1 - [HeapHero](https://heaphero.io/)

2 – jcmd.exe: available with JDK

* $ jcmd <pid> GC.heap\_dump <file\_path>. E.g.: $ jcmd 37320 GC.heap\_dump /opt/tmp/heapdump.bin

3 – JVisualVM

## Thread Dump

* It’s a snapshot of all the threads running in a Java process.
* Can be used to troubleshoot: CPU spikes, unresponsiveness, poor response time, hung threads, high memory consumption.
* No noticeable overhead in capturing thread dumps on every 5 minutes or 2 minutes interval.

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| - |  |

### Tools

1 – jstack.exe: available with JDK

* $ jstack -l <pid> > <file\_path>. E.g.: jstack -l 37320 > /opt/tmp/threadDump.txt

2 – kill-3 <pid>: available with JRE

* $ kill -3 <pid>. E.g.: $ kill -3 37320

3 – jcmd.exe: available with JDK

* $ jcmd <pid> Thread.print > <file\_path>. E.g.: $ jcmd 37320 Thread.print > /opt/tmp/threadDump.txt

4 - JVisualVM

# Useful JVM Arguments as Reactive Actions

## When OutOfMemoryError

Use any of the followings:

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -XX:+HeapDumpOnOutOfMemoryError  -XX:HeapDumpPath={heap-dump-file-path} | See [Heap Dump section](#_Heap_Dump) for more details. |
| -XX:OnOutOfMemoryError=<script\_path> | Execute a script when OOME occurs.  Example:  -XX:OnOutOfMemoryError=/scripts/restart-myapp.sh |
| -XX:+CrashOnOutOfMemoryError | JVM exits when OOME occurs. Text and binary files are produced before exit. Not recommended. |
| -XX:+ExitOnOutOfMemoryError | Like CrashOnOutOfMemoryError but without text and binary files. Not recommended. |

# Recommended Practice for Optimum Performance

## Set Max Heap Size and Metaspace Size accordingly

Heap size and Metaspace size plays a role in determining the frequency of GC events for your application.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Xmx | Example:  Setting heap size to 2GB:  -Xmx2g |
| -XX:MaxMetaspaceSize | Example:  Setting Metaspace size to 256MB:  -XX:MaxMetaspaceSize=256m |

## Use ZGC for Java 11+ Application

ZGC is known for its sub-millisecond pauses, allowing latency-sensitive systems to thrive.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -XX:+UseZGC |  |

## Always Enable GC Logging

GC Logging has (close to) no-impact to application performance but very useful for troubleshooting purposes. See [GC log section](#_GC_log:) for more details.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -XX:+PrintGCDetails  -XX:+PrintGCDateStamps  -Xloggc: <file\_path> | Up to JDK 8  Example:  -XX:+PrintGCDetails  -XX:+PrintGCDateStamps  -Xloggc:/opt/workspace/myAppgc.log |
| -Xlog:gc\*:file=<file\_path> | JDK 8 +  Example:  -Xlog:gc\*:file=/opt/workspace/myAppgc.log |

## Create Heap Dump on Out of Memory Error

Heap dump is very useful in troubleshooting OutOfMemoryError in application when it happens.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -XX:+HeapDumpOnOutOfMemoryError  -XX:HeapDumpPath=<file\_path> | Example:  -XX:+HeapDumpOnOutOfMemoryError  -XX:HeapDumpPath=/dmp/my-heap-dump.hprof |

## Increase Stack Memory Only When Needed

Each thread will have its own stack. When not enough memory in stack, StackOverflowError is thrown.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Xss | Example:  Setting stack size to 256KB:  -Xss256k |

## Set timeout for connection

This is to avoid unresponsiveness in your application caused by remote applications and safeguard your applications high availability.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Dsun.net.client.defaultConnectTimeout  -Dsun.net.client.defaultReadTimeout | Example:  -Dsun.net.client.defaultConnectTimeout=2000  -Dsun.net.client.defaultReadTimeout=2000 |

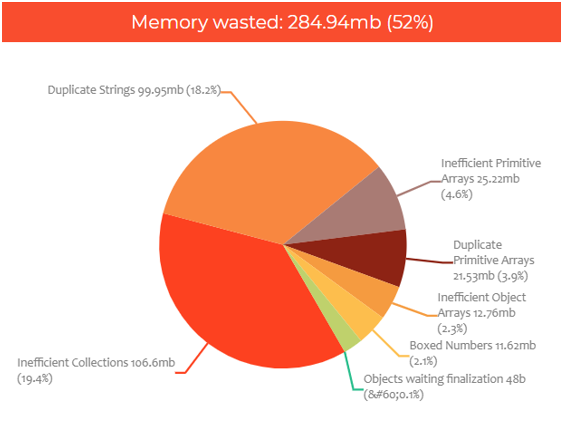
## Set Time Zone for Your Application

This is particularly useful for sensitive business requirements in an application running in a distributed environment.

### Setup

|  |  |
| --- | --- |
| **JVM Arguments** | **Remarks** |
| -Duser.timezone | Example:  -Duser.timezone=”Asia/Kolkata” |

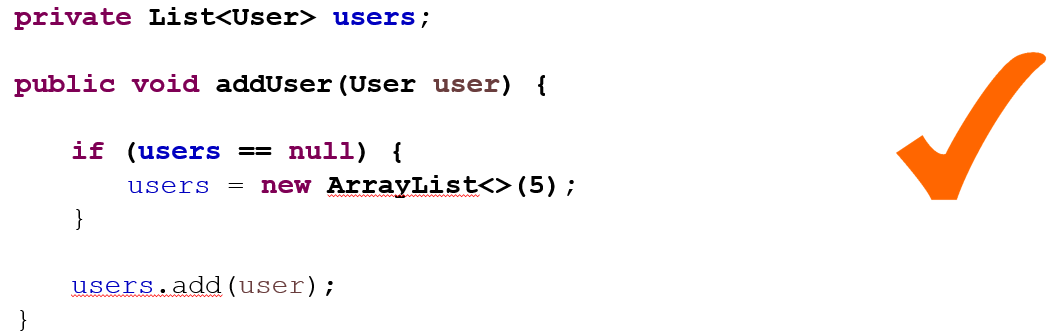
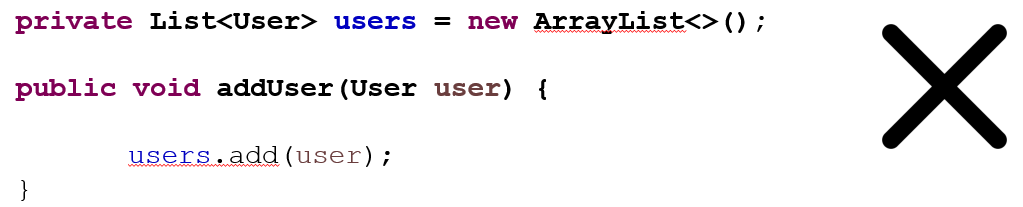
## Stop Wasting Memory in Your Code



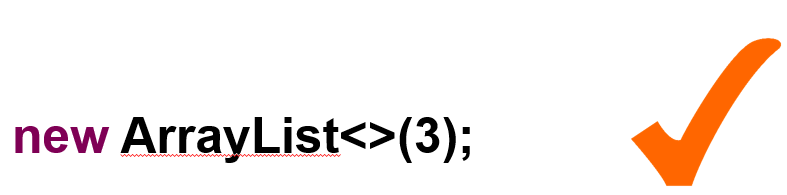
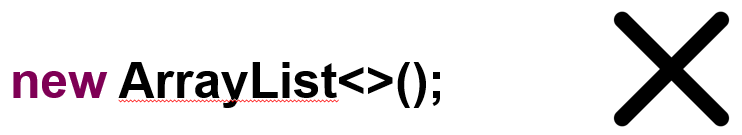
Several ways to optimise object creation and management:

Collections:

1. Lazy initialisation



1. Specify capacity:

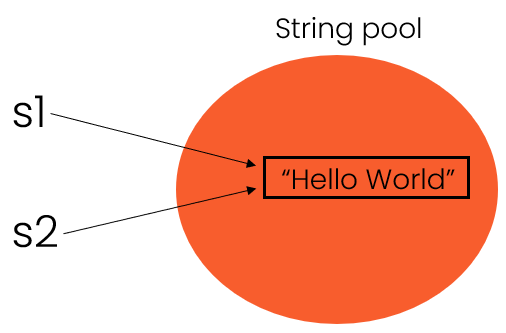


1. Null instead of clear():

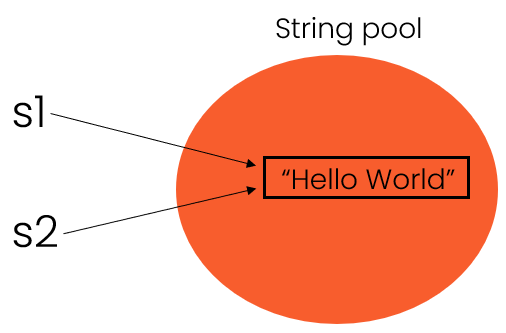
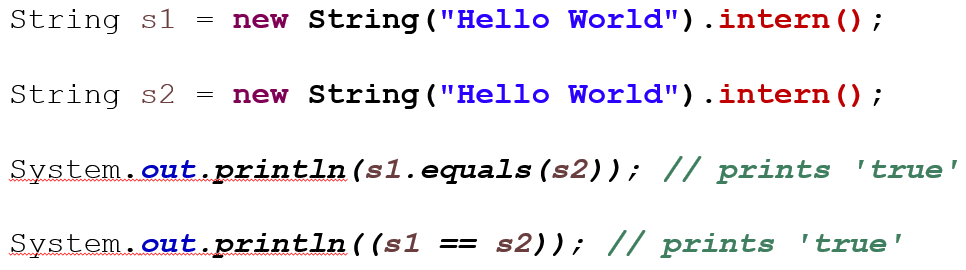


Strings:

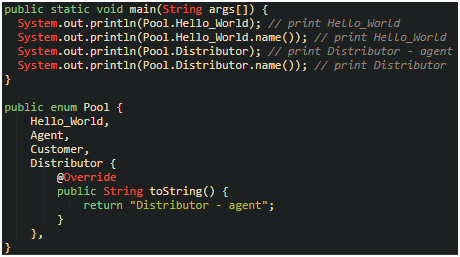
1. Use string literals



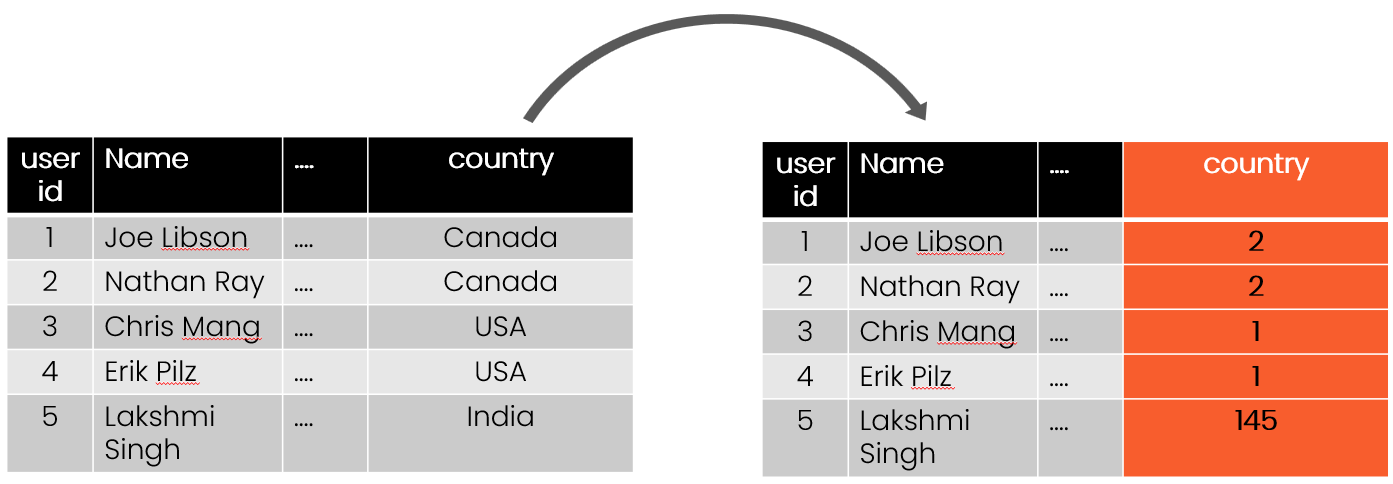
1. Use intern()



1. Alternative:
   1. Use Enum

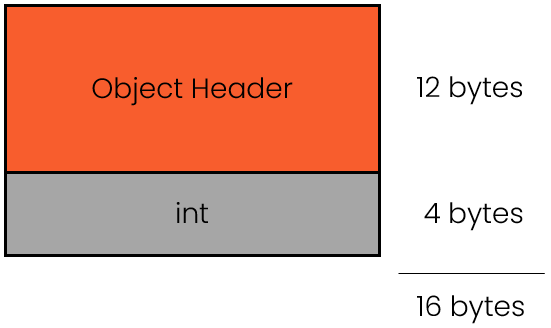


* 1. In db, consider storing data as primitive types



1. G1GC only: -XX:+UseDeduplication

Objects: use primitive types as much as possible and avoid boxed object as each object incurs overhead of 12bytes for header:

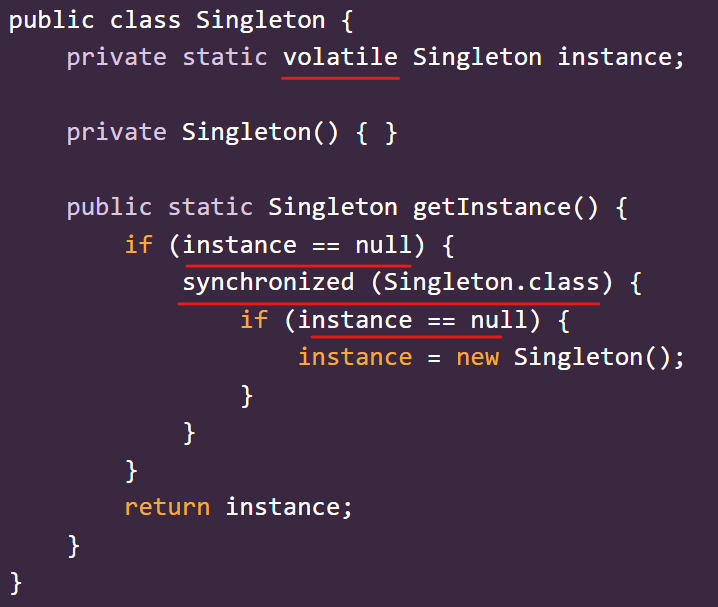


### Heap Monitoring Tools

See [heap dump tools section](#_Tools) for details.

## Use Volatile for Singleton

Inspired by a conversation with Romeu Flores. Source: [here](https://codingtechroom.com/question/singleton-pattern-with-volatile-java).



Volatile Singleton instance

The code above solves the following issues:

* Multiple threads accessing the getInstance method simultaneously can lead to multiple instances being created if proper synchronization is not used.
* The instance may not be fully initialized before another thread gets access due to issues related to caching by the CPU.

Details on the solution:

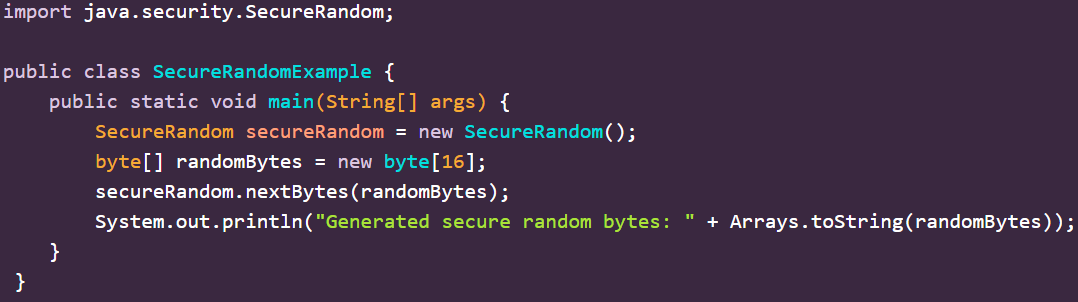
* Use the 'volatile' keyword to declare the instance variable to ensure that its value is always read from main memory, preventing caching issues.
* Implement double-checked locking in the getInstance method, which allows for lazy initialization while minimizing synchronization overhead.

Volatile keyword ensures visibility across threads.

Double-checked locking is implemented to avoid unnecessary synchronization and ensure proper instance initialization.

## Use SecureRandom for Enhanced Security

Inspired by a conversation with Romeu Flores. Source: [here](https://codingtechroom.com/question/provide-entropy-to-jvm).



SecureRandom in action

The code above solves the following issues:

* Insufficient system entropy due to lack of unpredictable events.
* Poor configuration of the random number generator
* Operating in a virtualized environment with limited access to hardware entropy sources

Details on the solution:

* Use a hardware random number generator (HRNG) which can provide high-quality entropy.
* Configure the JVM to use an alternative source of entropy, such as a secure file or device that gathers random data.
* Increase system entropy by monitoring and managing entropy sources, particularly in virtual machines

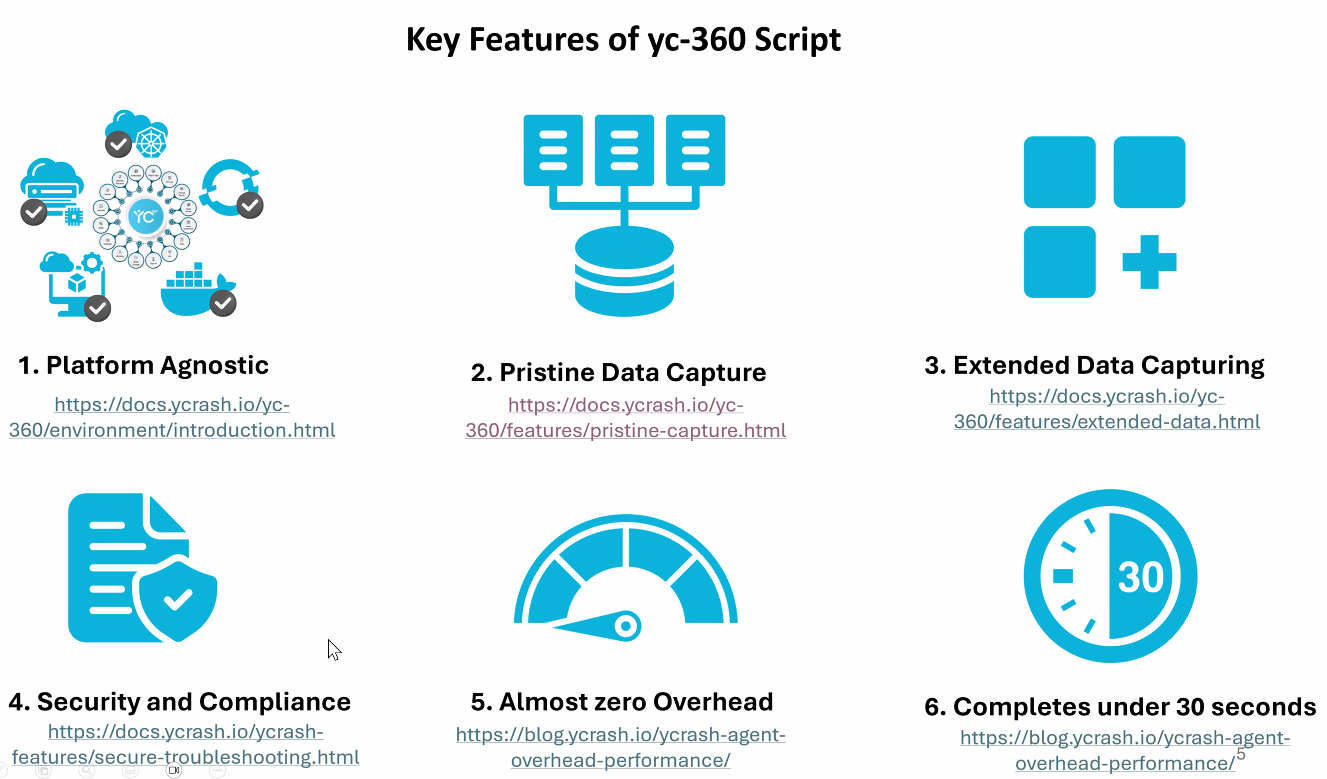
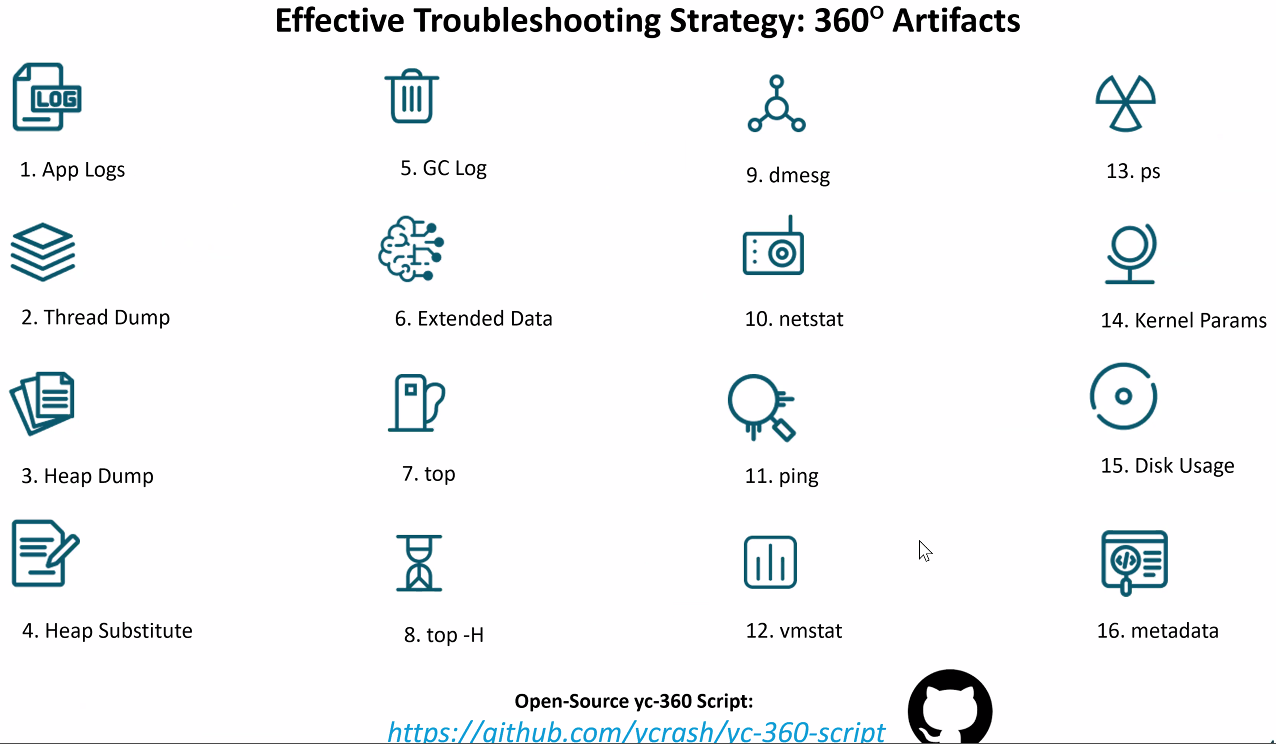
Entropy = a measure of disorder or randomness in a system.

Always use SecureRandom instead for cryptographic applications.

To provide adequate entropy sources in cloud or virtualised environments, implement additional entropy gathering methods like using /dev/random or /dev/urandom on Linux.

## Use yCrash 360-degree Script as Artefact Collection

Location: [here](https://github.com/ycrash/yc-360-script). The script is open-source. Once run, the bundled zip file can be uploaded onto yCrash web app (licence required).



Unless if heapdump is enabled, it will be longer than 30sec

Run the script once problem is detected, e.g. after application crashes.