Java Troubleshooting and Diagnostic

Roman Makarevich

<epam> Delivering Excellence in Software Engineering

Dealing with Errors

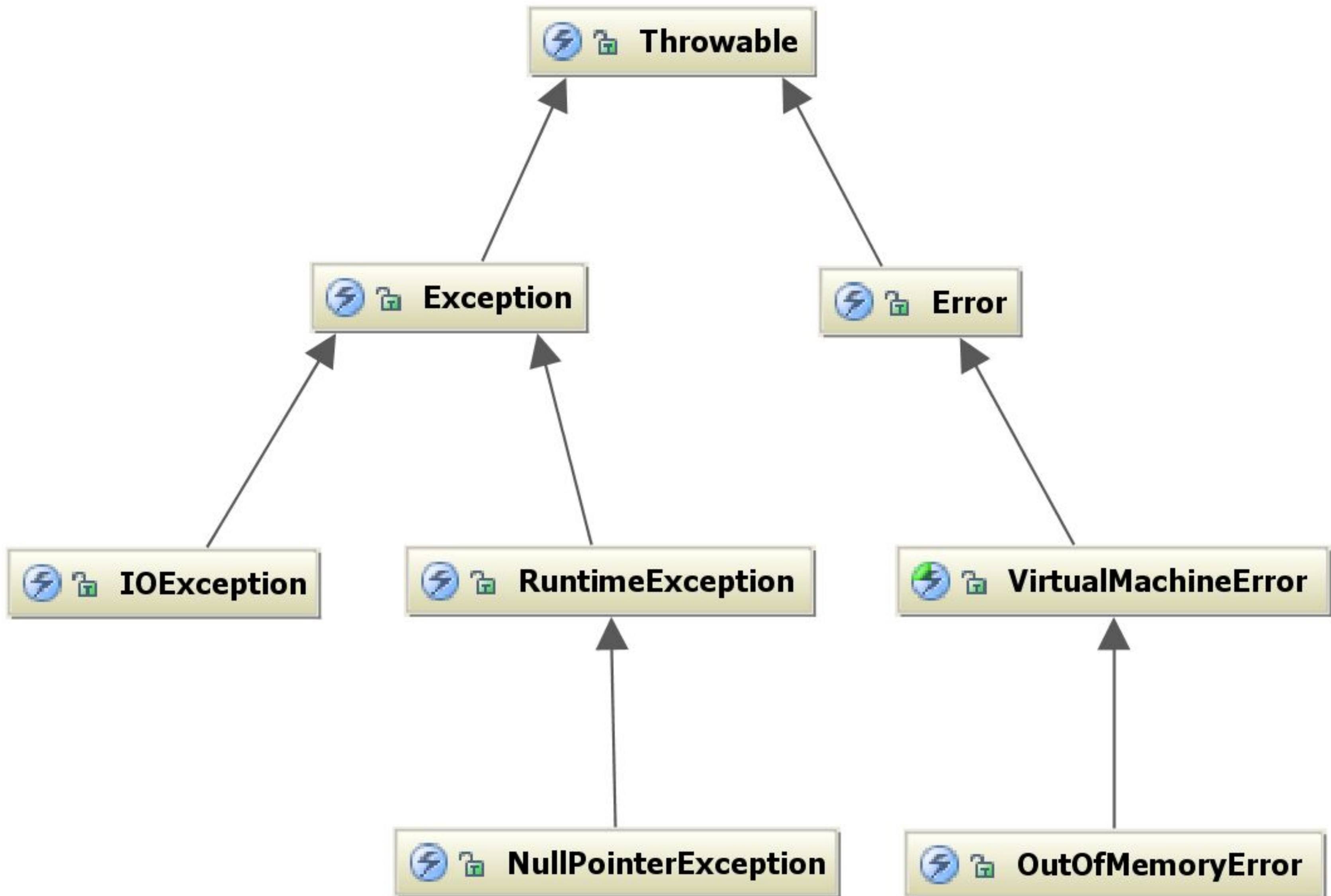
User input errors: In addition to the inevitable typos, some users like to blaze their own trail instead of following directions. Suppose, for example, that a user asks to connect to a URL that is syntactically wrong. Your code should check the syntax, but suppose it does not. Then the network package will complain.

Device errors: Hardware does not always do what you want it to. The printer may be turned off. A web page may be temporarily unavailable. Devices will often fail in the middle of a task. For example, a printer may run out of paper in the middle of a printout.

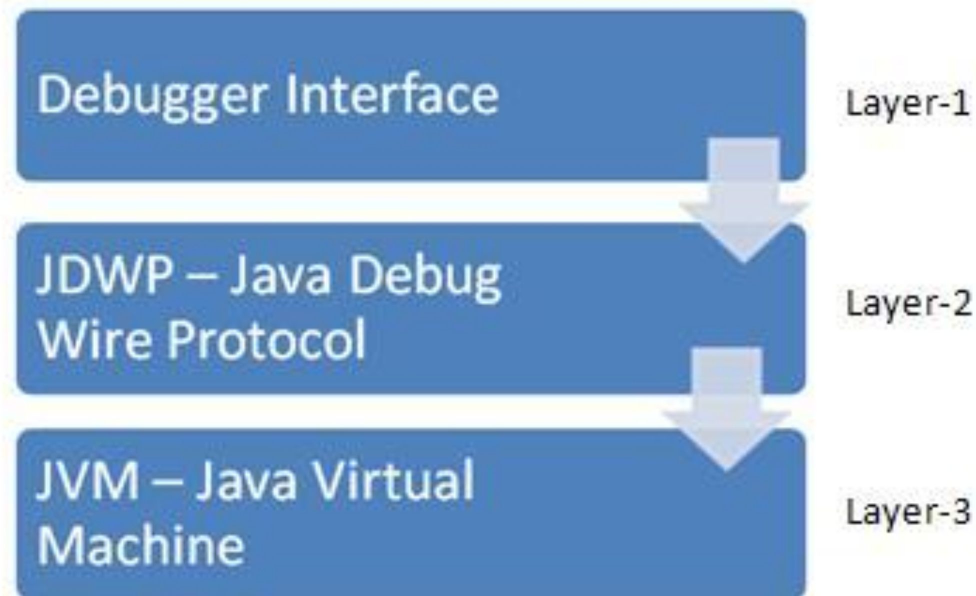
Physical limitations: Disks can fill up; you can run out of available memory.

Code errors: A method may not perform correctly. For example, it could deliver wrong answers or use other methods incorrectly. Computing an invalid array index, trying to find a nonexistent entry in a hash table, and trying to pop an empty stack are all examples of a code error.

The Classification of Exceptions



Java Platform Debugger Architecture



Java Platform Debugger Architecture

JVM Debug Parameters

Modern JVMs

-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=5005

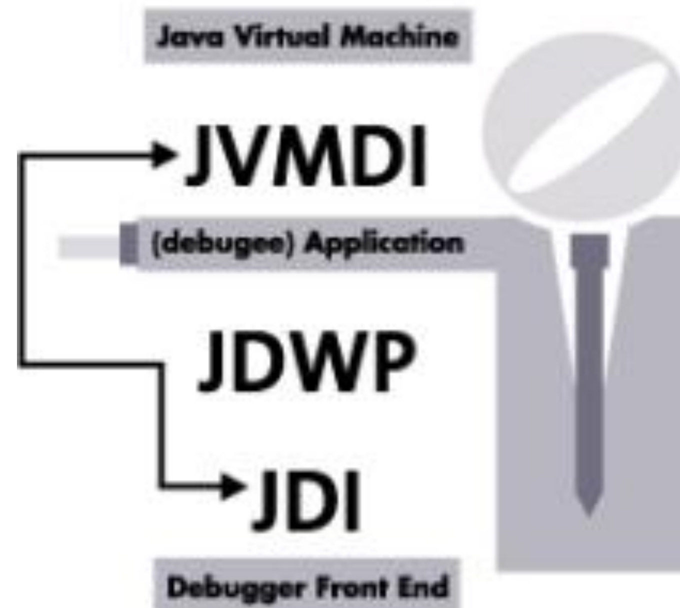
For JDK 1.4.x

-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005

For JDK 1.3.x or earlier

-Xnoagent -Djava.compiler=NONE -Xdebug
-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005

Java Platform Debugger Architecture



Exceptions and Performance

```
Stack s = new Stack();

for (int i = 0; i < 1000000; i++) {
    if (!s.isEmpty()) {
        s.pop();
    }
}
```

15 millisecond

```
for (int i = 0; i < 1000000; i++) {
    try {
        s.pop();
    } catch (EmptyStackException ex) {}
}
```

3000 millisecond

HotSpot Compilers

Note:
Check your production environment VM options !
Are you running HotSpot VM using client or server compilers !?

- HotSpot Client VM:

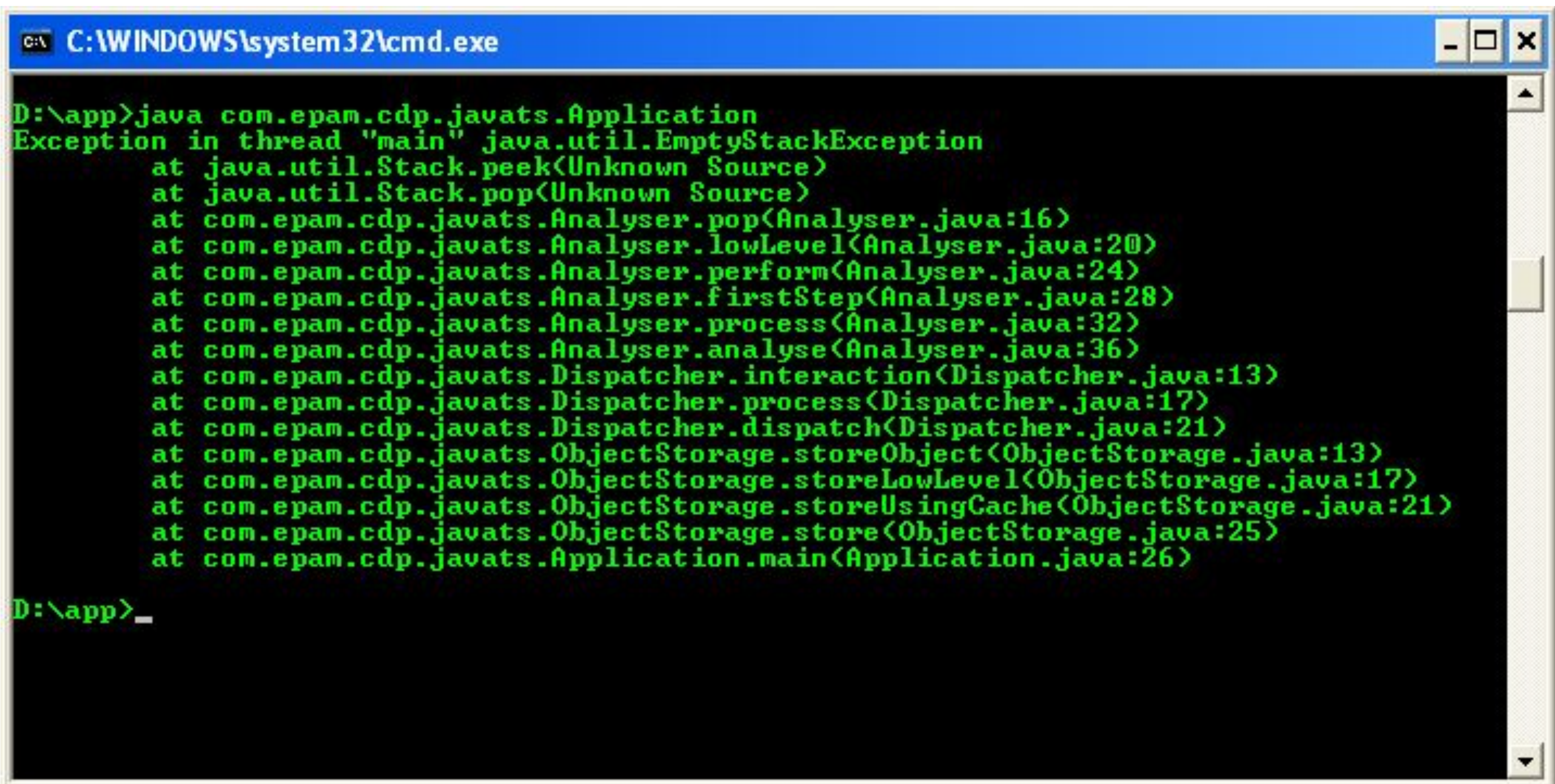
```
java version "1.5.0_11"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_11-b03)
Java HotSpot(TM) Client VM (build 1.5.0_11-b03, mixed mode, sharing)
```

- HotSpot Server VM:

```
java version "1.5.0_11"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_11-b03)
Java HotSpot(TM) Server VM (build 1.5.0_11-b03, mixed mode)
```

What is a Java stack trace?

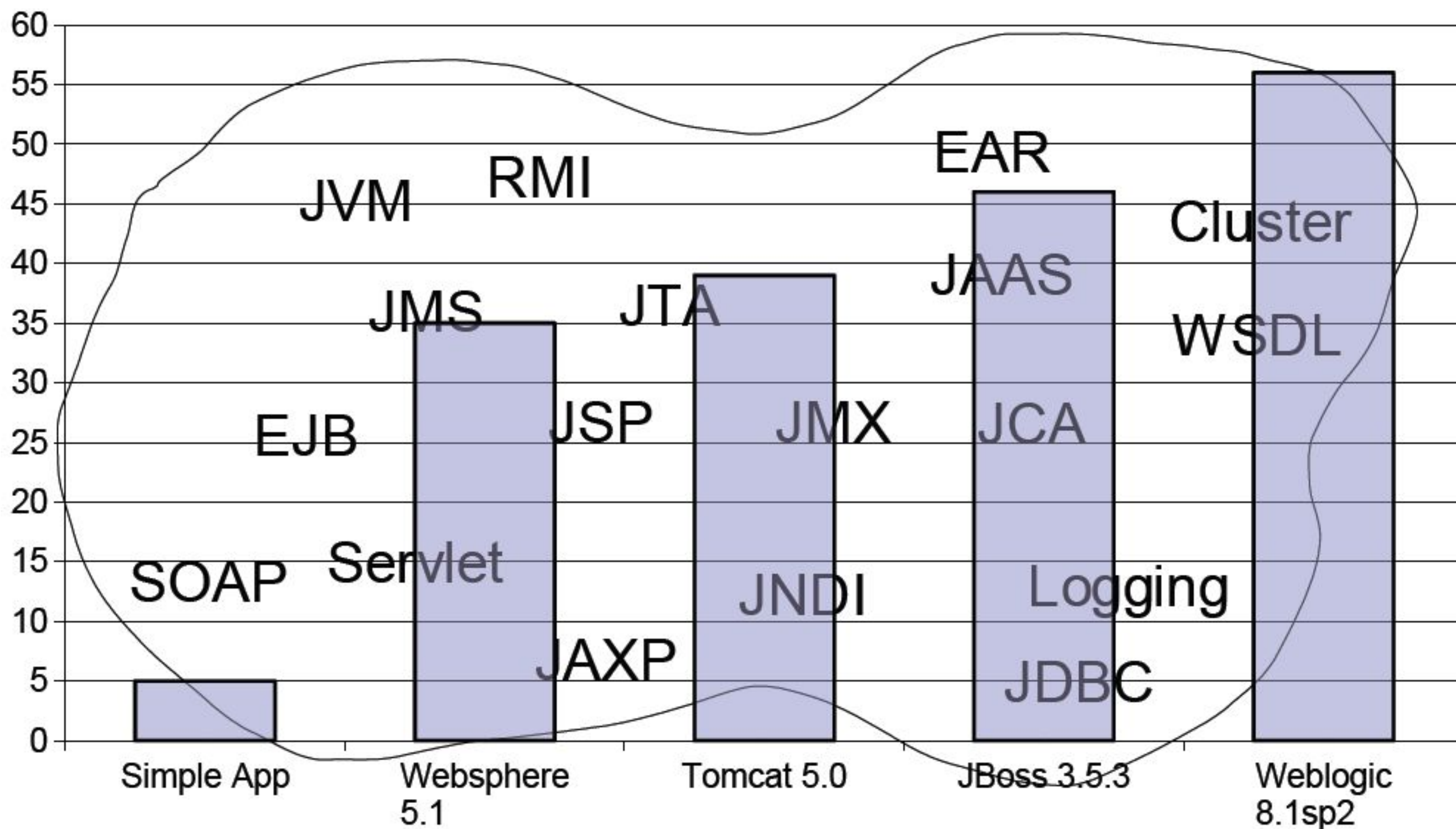
Java stack trace is a user-friendly snapshot of the Java thread.



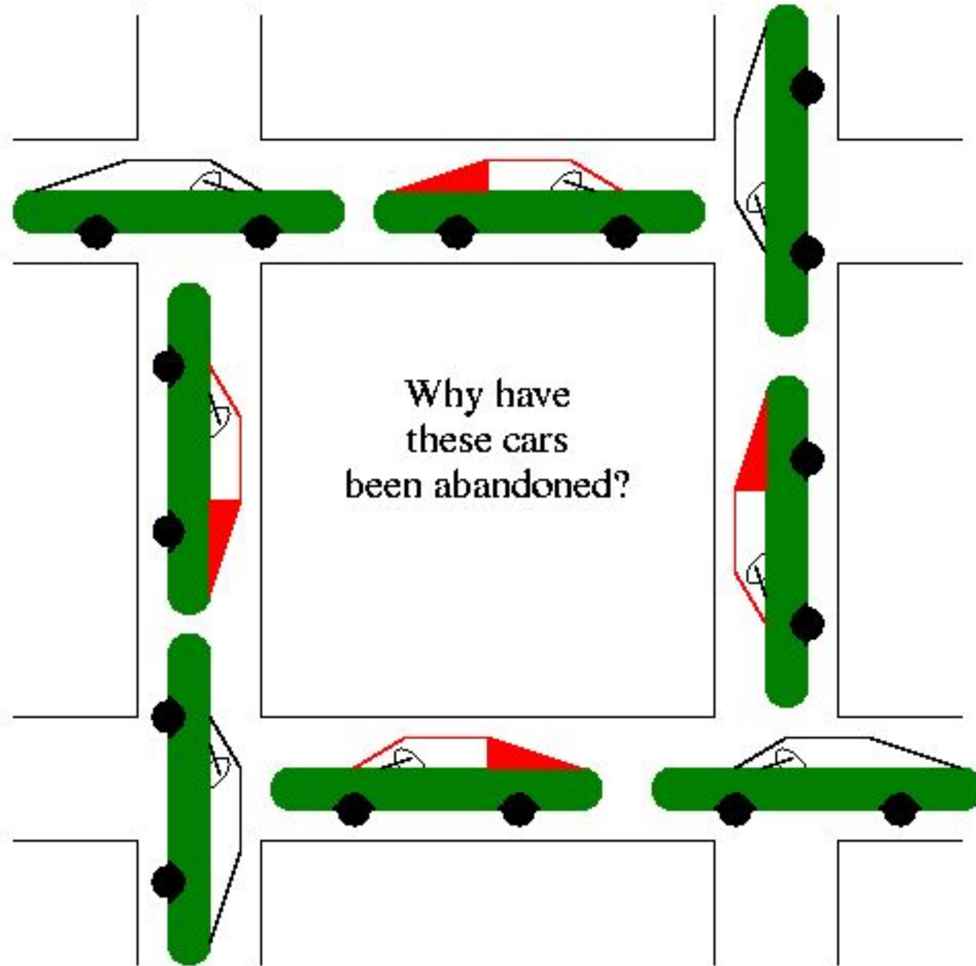
Java Bytecode Debugging Information

source	Source file debugging information
lines	Line number debugging information
vars	Local variable debugging information

Thread Count (Default Configuration)



Deadlock



Expert's Checklist

For hanging, deadlocked or frozen programs: If you think your program is hanging, generate a stack trace and examine the threads in states MW or CW. If the program is deadlocked then some of the system threads will probably show up as the current threads, because there is nothing else for the JVM to do.

For crashed, aborted programs: On UNIX look for a core file. You can analyze this file in a native debugging tool such as gdb or dbx. Look for threads that have called native methods. Because Java technology uses a safe memory model, any corruption probably occurred in the native code. Remember that the JVM also uses native code, so it may not necessarily be a bug in your application.

For busy programs: The best course of action you can take for busy programs is to generate frequent stack traces. This will narrow down the code path that is causing the errors, and you can then start your investigation from there.

Where Is My Stacktrace?

How is Java Thread Dump Generated?

- By sending a signal to JVM (ctrl+break)
- Using JDK 5/6 tools (jps, jstack)
- Using debugging tools (jdb, IDEs)
- Using Java API calls
- Other ad hoc tools (e.g. adaptj StackTrace)

Thread Dump By Sending a Signal to JVM

UNIX:

- Ctrl+\
- kill -QUIT process_id

Windows:

- Ctrl+Break
- SendSignal process_id

Notes:

- No -Xrs in Java command line!
- SendSignal is a homemade program!

Thread Dump Using JDK 5/6 tools

C:\WINDOWS\system32\cmd.exe

jps

```
D:\app>jps
3248 Application
800
2436 Jps
```

jstack

```
D:\app>jstack 3248
2009-03-01 17:34:06
Full thread dump Java HotSpot(TM) Client VM (1.6.0_01-b06 mixed mode, sharing):

"Low Memory Detector" daemon prio=6 tid=0x02b23000 nid=0x708 runnable [0x00000000..0x00000000]
   java.lang.Thread.State: RUNNABLE

"CompilerThread0" daemon prio=10 tid=0x02b1e400 nid=0xa4 waiting on condition [0x00000000..0x02d9f81c]
   java.lang.Thread.State: RUNNABLE

"Attach Listener" daemon prio=10 tid=0x02b1d000 nid=0x118 runnable [0x00000000..0x00000000]
   java.lang.Thread.State: RUNNABLE

"Signal Dispatcher" daemon prio=10 tid=0x02b1c400 nid=0x740 runnable [0x00000000..0x00000000]
   java.lang.Thread.State: RUNNABLE

"Finalizer" daemon prio=8 tid=0x02ade000 nid=0x100 in Object.wait() [0x02caf000..0x02cafc94]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x22edb1d8> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:116)
        - locked <0x22edb1d8> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:132)
        at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:159)

"Reference Handler" daemon prio=10 tid=0x02add000 nid=0x408 in Object.wait() [0x02c5f000..0x02c5fd14]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x22edb268> (a java.lang.ref.Reference$Lock)
        at java.lang.Object.wait(Object.java:485)
```

Thread Dump Using Debugging Tools

Командная строка - java -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,ad...

java with debug

```
C:\app>java -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=543
2 com.epam.cdp.javats.Application
Listening for transport dt_socket at address: 5432
```

Командная строка - jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=5432

jdb

suspend all threads

get thread dump

```
C:\>jdb -connect com.sun.jdi.SocketAttach:hostname=localhost,port=5432
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
> suspend
All threads suspended.
> where all
Attach Listener:
Signal Dispatcher:
Finalizer:
  [1] java.lang.Object.wait (native method)
  [2] java.lang.ref.ReferenceQueue.remove (ReferenceQueue.java:116)
  [3] java.lang.ref.ReferenceQueue.remove (ReferenceQueue.java:132)
  [4] java.lang.ref.Finalizer$FinalizerThread.run (Finalizer.java:159)
Reference Handler:
  [1] java.lang.Object.wait (native method)
  [2] java.lang.Object.wait (Object.java:485)
  [3] java.lang.ref.Reference$ReferenceHandler.run (Reference.java:116)
main:
  [1] java.util.Stack.peek (Stack.java:85)
  [2] java.util.Stack.pop (Stack.java:67)
  [3] com.epam.cdp.javats.Application.main (Application.java:31)
> _
```

Using a Debugger

Command	Description
threads [tHReadgroup]	Lists threads
thread tHRead_id	Sets default thread
suspend [tHRead_id(s)]	Suspends threads (default: all)
resume [tHRead_id(s)]	Resumes threads (default: all)
where [thread_id] or all	Dumps a thread's stack
wherei [tHRead_id] or all	Dumps a thread's stack and program counter info
tHReadgroups	Lists thread groups
tHReadgroup name	Sets current thread group
print name(s)	Prints object or field
dump name(s)	Prints all object information
locals	Prints all current local variables
classes	Lists currently known classes
methods class	Lists a class's methods
stop in class.method	Sets a breakpoint in a method
stop at class:line	Sets a breakpoint at a line
up [n]	Moves up a thread's stack
down [n]	Moves down a thread's stack

Command	Description
clear class:line	Clears a breakpoint
step	Executes the current line, stepping inside calls
stepi	Executes the current instruction
step up	Executes until the end of the current method
next	Executes the current line, stepping over calls
cont	Continues execution from breakpoint
catch class	Breaks for the specified exception
ignore class	Ignores the specified exception
list [line]	Prints source code
use [path]	Displays or changes the source path
memory	Reports memory usage
gc	Frees unused objects
load class	Loads Java class to be debugged
run [class [args]]	Starts execution of a loaded Java class
!!	Repeats last command
help (or ?)	Lists commands
exit (or quit)	Exits debugger

Using Java API calls

- **Throwable.printStackTrace()**
- **Thread.dumpStack()**
- Since Java 1.5: **Thread.getState()**
- Since Java 1.5: **Thread.getStackTrace()**
- Since Java 1.5: **Thread.getAllStackTraces()**

Thread Dump Analyser

TDA - Thread Dumps of D:\Downloads\dining-philosophers-threaddump.txt

File View Tools Help

- D:\Downloads\dining-philosophers-threaddump.txt
  - Logfile
  - Dump No. 1 at line 1
    - Threads (19 Threads overall)
    - Threads sleeping on Monitors (12 Threads sleeping)
    - Threads locking Monitors (12 Threads locking)
    - Monitors (12 Monitors)

Name	Type	Prio	Thread-ID	Native-ID	State	Address Range
Thread-7	Task	4	189276704	5488	in Object.wait()	[0x0bbcf000..0x0bbcfae8]
Thread-6	Task	4	189274120	2692	in Object.wait()	[0x0bb8f000..0x0bb8fb68]
Thread-5	Task	4	189260560	5756	in Object.wait()	[0x0bb4f000..0x0bb4fbe8]
Thread-4	Task	4	189257736	5936	in Object.wait()	[0x0bb0f000..0x0bb0fc68]
Thread-3	Task	4	189271256	4548	in Object.wait()	[0x0bacf000..0x0bacfce8]
AWT-EventQueue-1	Task	4	189194704	5832	in Object.wait()	[0x0ba4f000..0x0ba4fa68]
DestroyJavaVM	Task	6	2518464	4376	waiting on condition	[0x00000000..0x0006fae8]
AWT-EventQueue-0	Task	6	189079392	4684	in Object.wait()	[0x0b82f000..0x0b82fbe8]
thread applet-concurrency/diners/Diners.class	Task	4	188525376	4256	in Object.wait()	[0x0b7ef000..0x0b7efb68]
AWT-Windows	Daemon	6	180947768	4388	runnable	[0x0af0f000..0x0af0fce8]
AWT-Shutdown	Task	6	180946816	2012	in Object.wait()	[0x0aecf000..0x0aecfd68]
Java2D Disposer	Daemon	10	180890280	4116	in Object.wait()	[0x0ae8f000..0x0ae8f9e8]
Low Memory Detector	Daemon	6	11095664	4152	runnable	[0x00000000..0x00000000]
CompilerThread0	Daemon	10	11090544	4808	waiting on condition	[0x00000000..0x0abcf8c8]
Signal Dispatcher	Daemon	10	11087400	5820	waiting on condition	[0x00000000..0x00000000]
Finalizer	Daemon	8	11050192	4164	in Object.wait()	[0x0ab4f000..0x0ab4fc68]
Reference Handler	Daemon	10	11044960	4204	in Object.wait()	[0x0ab0f000..0x0ab0fce8]
VM Thread	Task	10	11034008	4144	runnable	<no address range>
VM Periodic Task Thread	Task	10	11100360	5300	waiting on condition	<no address range>

```
"AWT-EventQueue-0" prio=6 tid=0x0b451f60 nid=0x124c in Object.wait() [0x0b82f000..0x0b82fbe8]
        at java.lang.Object.wait(Native Method)
        - waiting on <0x0300a858> (a java.awt.EventQueue)
        at java.lang.Object.wait(Object.java:474)
        at java.awt.EventQueue.getNextEvent(EventQueue.java:345)
        - locked <0x0300a858> (a java.awt.EventQueue)
        at java.awt.EventDispatchThread.pumpOneEventForHierarchy(EventDispatchThread.java:189)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:163)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:157)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:149)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:110)
```

TDA - Thread Dump Analyzer 2.1 | 3,8MB/4,5MB

IBM Thread & Monitor Dump Analyser

IBM Thread and Monitor Dump Analyzer for Java

File Analysis View Help

Floatable

Thread Dump List

Name	Timestamp	Runnable/Total Threads	Free/Allocated Heap(Free%)	AF(SC)/GC Counter	Monitor
dining-philosophers-thread...	No Info	3/19	unknown/unknown(unknown)	No Info	No Info

- File name : D:\Work\EPAM\CDP\Modules\Java Troubleshooting and Diagnostic\info\dining-philosophers-threaddump.txt
- Thread Status Analysis

Status	Number of Threads : 19	Percentage
Deadlock	0	0 (%)
Runnable	3	16 (%)
Waiting on condition	4	21 (%)
Waiting on monitor	0	0 (%)
Suspended	0	0 (%)
Object.wait()	12	63 (%)
Blocked	0	0 (%)
Parked	0	0 (%)

- Thread Method Analysis

Method Name	Number of Threads : 19	Percentage
java.lang.Object.wait(Native Method)	12	63 (%)
NO JAVA STACK	6	32 (%)
sun.awt.windows.WToolkit.eventLoop(Native Method)	1	5 (%)

- Thread Aggregation Analysis

Thread Type	Number of Threads : 19	Percentage
Thread	8	42 (%)

IBM Thread & Monitor Dump Analyser

IBM Thread and Monitor Dump Analyzer for Java

File Analysis View Help

Floatable

Thread Detail : dining-philosophers-threaddump.txt_1

Name	State	NativeID	Method
AWT-EventQueue-0	in Object.wait()	0x124c	java.lang.Object.wait(Native Method)
AWT-EventQueue-1	in Object.wait()	0x16c8	java.lang.Object.wait(Native Method)
AWT-Shutdown	in Object.wait()	0x7dc	java.lang.Object.wait(Native Method)
AWT-Windows	Runnable	0x1124	sun.awt.windows.WToolkit.eventLoop...
CompilerThread0	Waiting on condition	0x12c8	NO JAVA STACK
DestroyJavaVM	Waiting on condition	0x1118	NO JAVA STACK
Finalizer	in Object.wait()	0x1044	java.lang.Object.wait(Native Method)
Java2D Disposer	in Object.wait()	0x1014	java.lang.Object.wait(Native Method)
Low Memory Detector	Runnable	0x1038	NO JAVA STACK
Reference Handler	in Object.wait()	0x106c	java.lang.Object.wait(Native Method)
Signal Dispatcher	Waiting on condition	0x16bc	NO JAVA STACK
Thread-3	in Object.wait()	0x11c4	java.lang.Object.wait(Native Method)
Thread-4	in Object.wait()	0x1730	java.lang.Object.wait(Native Method)
Thread-5	in Object.wait()	0x167c	java.lang.Object.wait(Native Method)
Thread-6	in Object.wait()	0xa84	java.lang.Object.wait(Native Method)
Thread-7	in Object.wait()	0x1570	java.lang.Object.wait(Native Method)
VM Periodic Task Thread	Waiting on condition	0x14b4	NO JAVA STACK
VM Thread	Runnable	0x1030	NO JAVA STACK
thread applet-concurren...	in Object.wait()	0x10a0	java.lang.Object.wait(Native Method)

Waiting Threads : 0

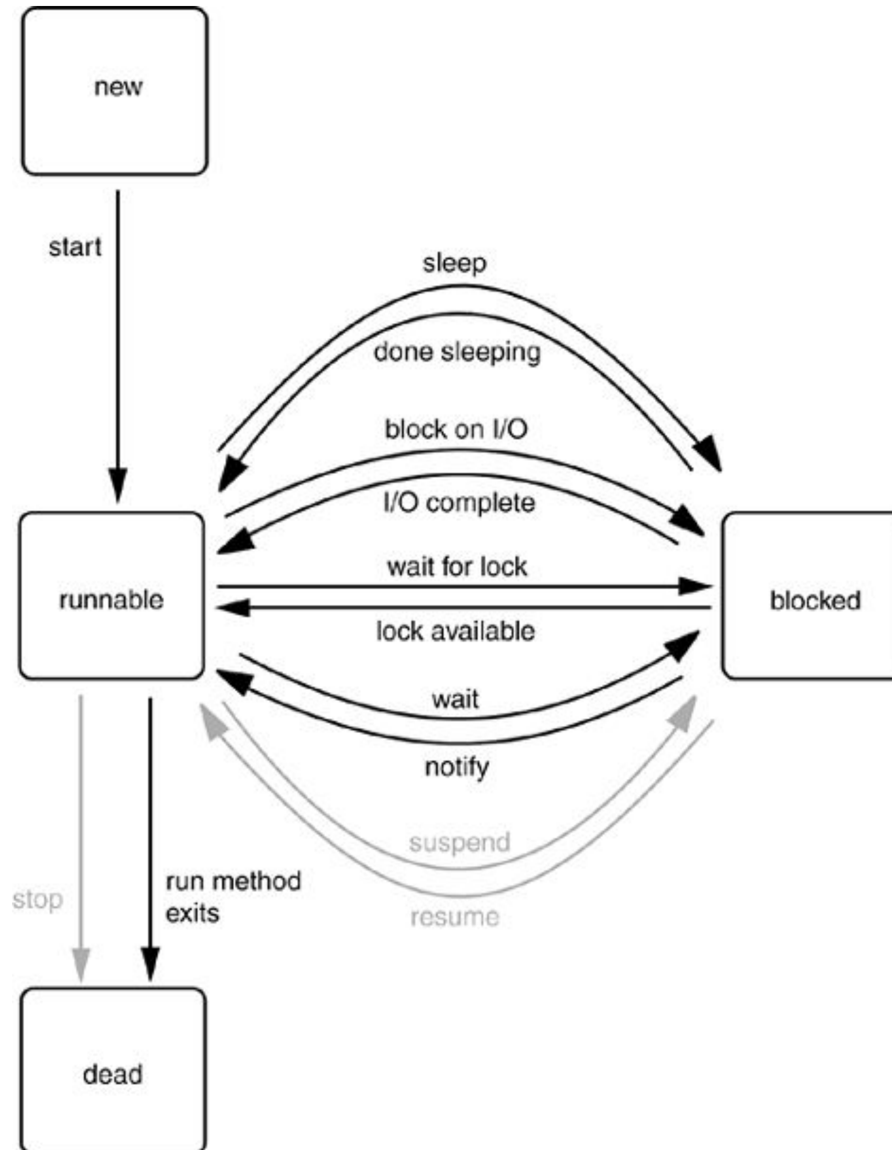
Blocked by : 0

Thread Name	AWT-Windows
State	Runnable
Java Stack	at sun.awt.windows.WToolkit.eventLoop(Native Method) at sun.awt.windows.WToolkit.run(WToolkit.java:269) at java.lang.Thread.run(Thread.java:595)
Native Stack	No Native stack trace available

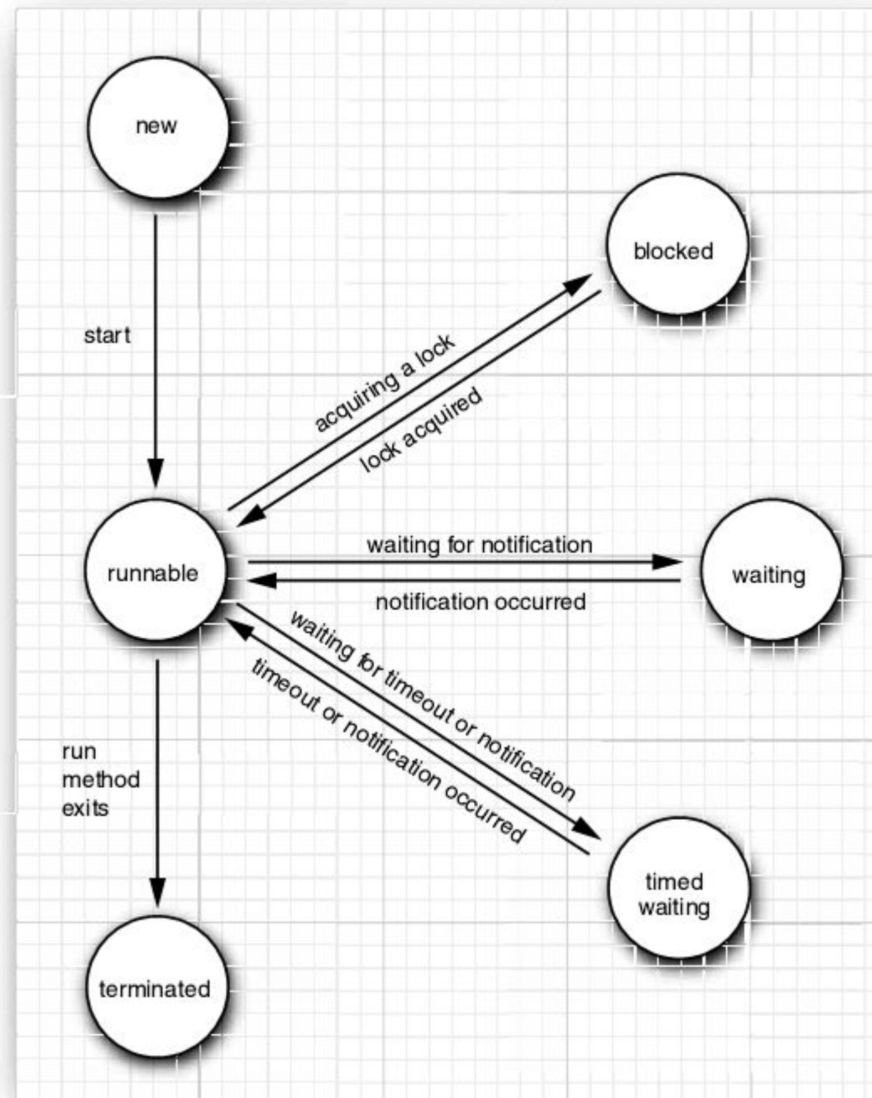
Determining the Thread States

R	Running or runnable thread
S	Suspended thread
CW	Thread waiting on a condition variable
MW	Thread waiting on a monitor lock
MS	Thread suspended waiting on a monitor lock

Thread States



Thread States



Java 1.4 tools

jinfo			
jmap			
jstack			
jconsole			
jps			
jstat			
jdb			

Java 1.5 tools

jinfo			
jmap			
jstack			
jconsole			
jps			
jstat			
jdb			

Java 1.6 tools

jinfo			
jmap			
jstack			
jconsole			
jps			
jstat			
jdb			

Debugging Performance Issues (1)

Symptom: High CPU consumption and poor response time

Thread dump profile: Most of the dumps show the same thread in the same method or same class

Solution: The method/class is the one which is definitely taking a lot of CPU. See if you can optimize these calls. Some of the REALLY easy kills we have had in this category is using a Collection.remove(Object) where the backend collection is a List. Change the backed collection to be a HashSet. A word of caution though: There have been times when the runnable threads are innocent and the GC is the one consuming the CPU.

Debugging Performance Issues (2)

Symptom: Low CPU consumption most of which is kernel time and poor response time

Thread dump profile: Most thread dumps have the runnable threads performing some IO operations

Solution: Most likely your application is IO bound. If you are reading a lot of files from the disc, see if you can implement Producer-Consumer pattern. The Producer can perform the IO operations and Consumers do the processing on the data which has been read by the producer. If you notice that most IO operations are from the data base driver, see if you can reduce the number of queries to the database or see if you can cache the results of the query locally.

Debugging Performance Issues (3)

Symptom: Medium/Low CPU consumption in a highly multithreaded application

Thread dump profile: Most threads in most thread dumps are waiting for a monitor on same object

Solution: The thread dump profile says it all. See if you can: eliminate the need for synchronization [using ThreadLocal/Session-scopeobjects] or reduce the amount of code being executed within the synchronized block.

Debugging Performance Issues (4)

Symptom: Medium/Low CPU consumption in a highly multithreaded application

Thread dump profile: Most threads in most thread dumps are waiting for a resource

Solution: If all the threads are choked for resources, say waiting on the pool to create EJB-bean objects/DB Connection objects, see if you can increase the pool size.

Example 1: Deadlock

org.apache.log4j.Category.callAppenders():

```
    public void callAppenders(LoggingEvent event)
    {
181   int writes = 0;

183   for (Category c = this; c != null; c = c.parent)
      {
185     synchronized (c) {
186       if (c.aai != null)
187         writes += c.aai.appendLoopOnAppenders(event);

189       if (c.additive) break label43;
190       label43: break label66:
        }

      }

195   if (writes == 0)
196     label66: this.repository.emitNoAppenderWarning(this);
    }
```

Example 2: Performance Issue

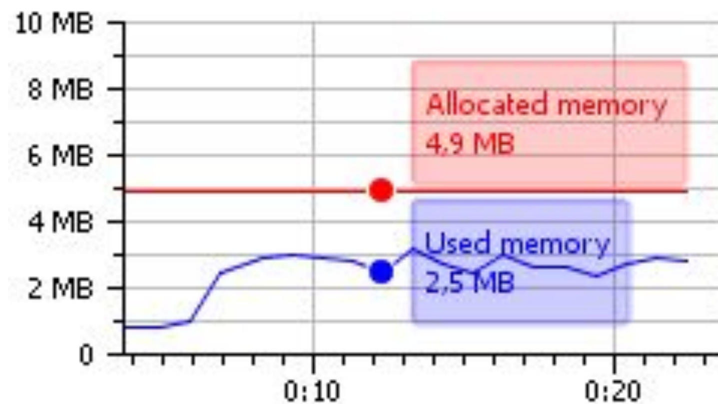
...
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1019)
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1065)
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1065)
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1065)
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1065)
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1065)
at org.apache.tools.ant.DirectoryScanner.scandir(DirectoryScanner.java:1065)
at
org.apache.tools.ant.DirectoryScanner.checkIncludePatterns(DirectoryScanner.java:836)
at org.apache.tools.ant.DirectoryScanner.scan(DirectoryScanner.java:808)
...

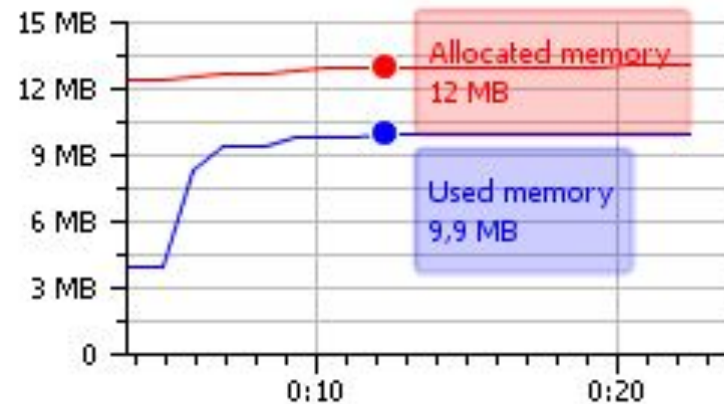
JVM Memory Structure

- Eden Space
- Survivor Space
- Tenured Generation

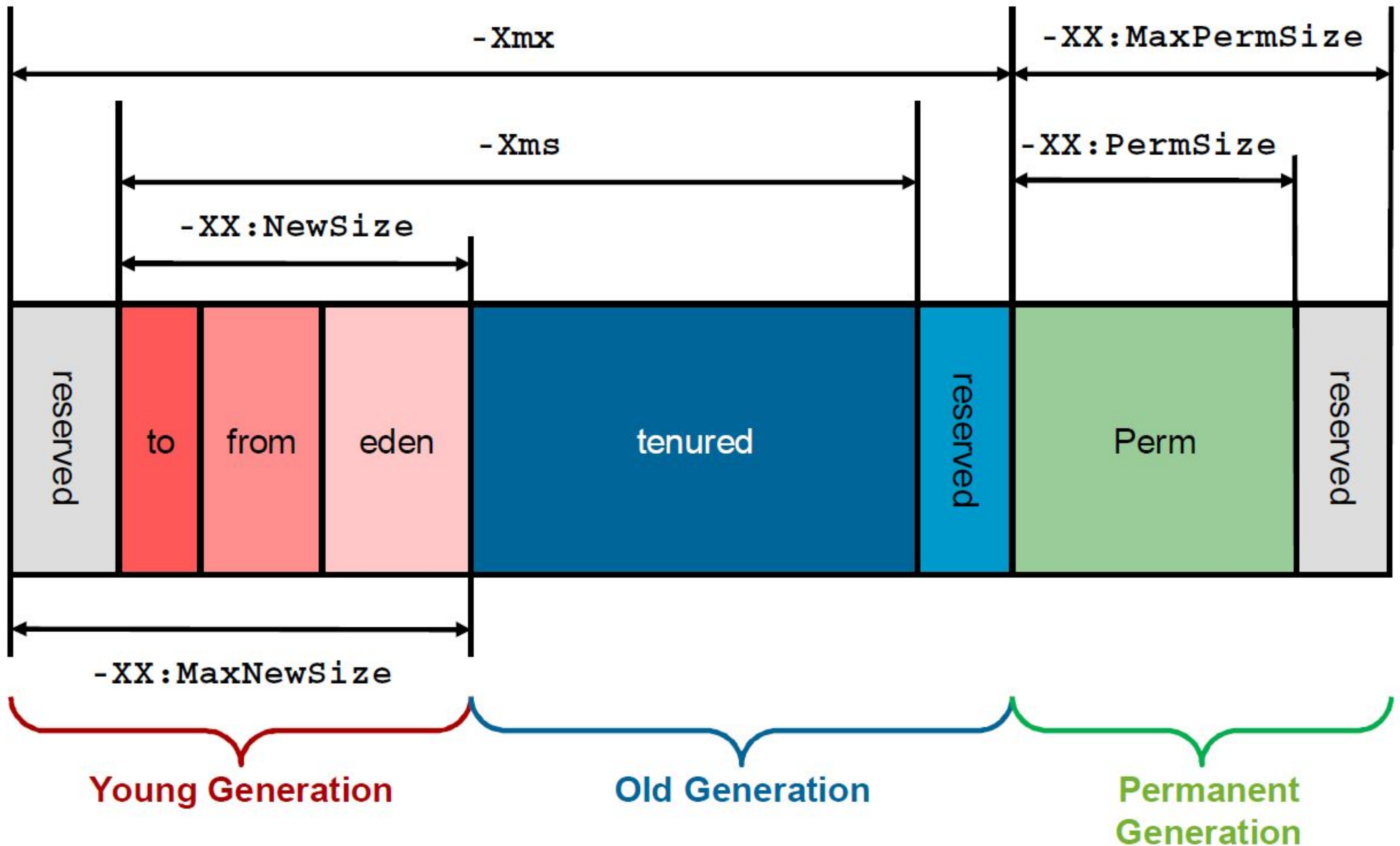
- Permanent Generation
- Code Cache

Allocated and Used Memory





Java Heap Memory & Tuning Options



Heap Dump

Typical information which can be found in heap dumps (depending on the heap dump type) is:

All Objects
Class, fields, primitive values and references

All Classes
Classloader, name, super class, static fields

Garbage Collection Roots
Objects defined to be reachable by the JVM

Thread Stacks and Local Variables
The call-stacks of threads at the moment of the snapshot, and per-frame information about local objects

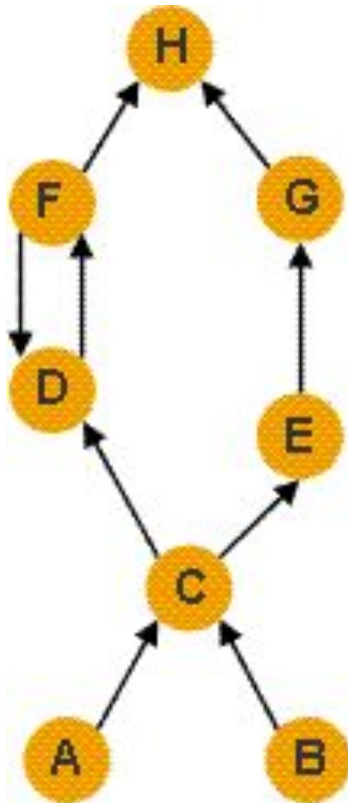
Shallow vs. Retained Heap

Shallow heap is the memory consumed by one object. 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 garbage collected.

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.

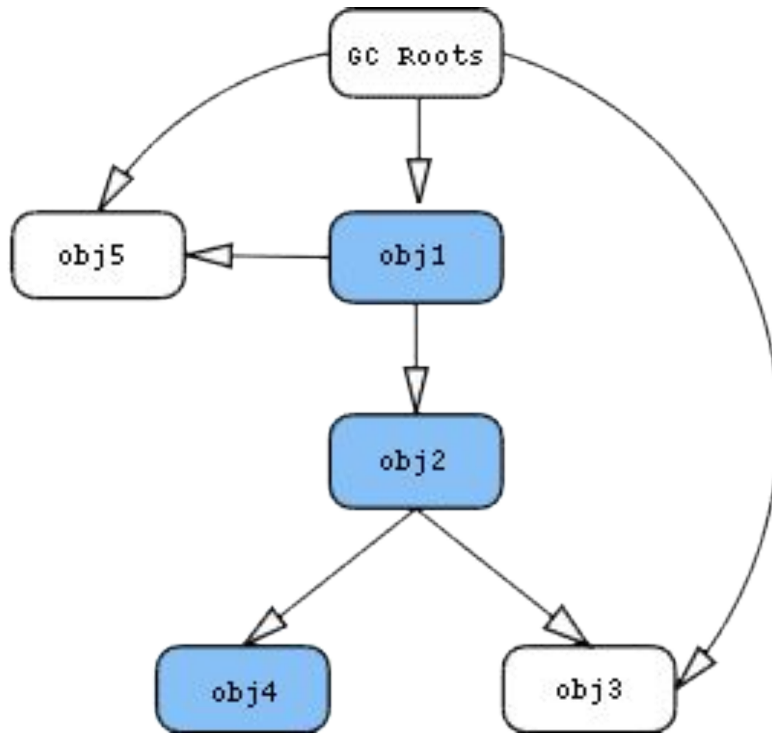
Shallow vs. Retained Heap

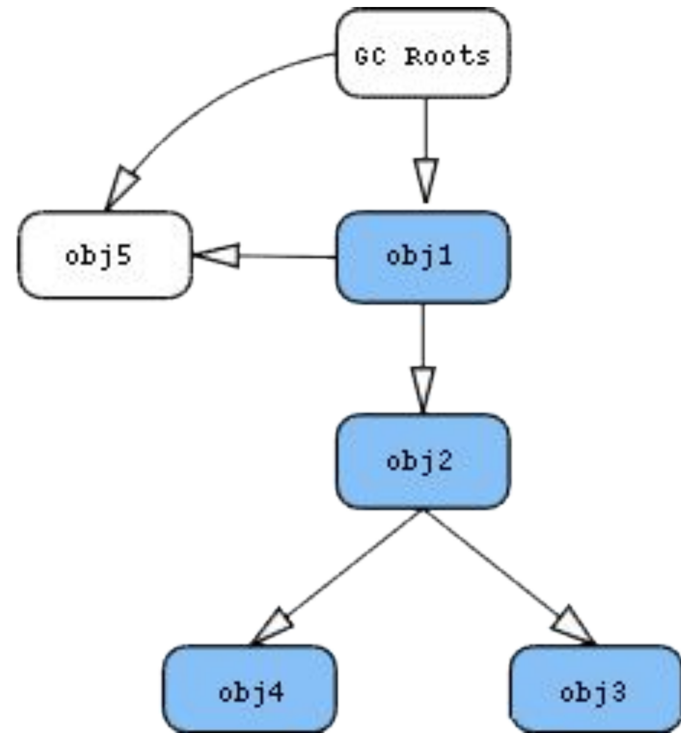


A and **B** are **garbage collection roots**, e.g. method parameters, locally created objects, objects used for wait(), notify() or synchronized(), etc.

Leading Set	**Retained Set**
E	E,G
C	C,D,E,F,G,H
A,B	A,B,C,D,E,F,G,H

Shallow and retained sizes



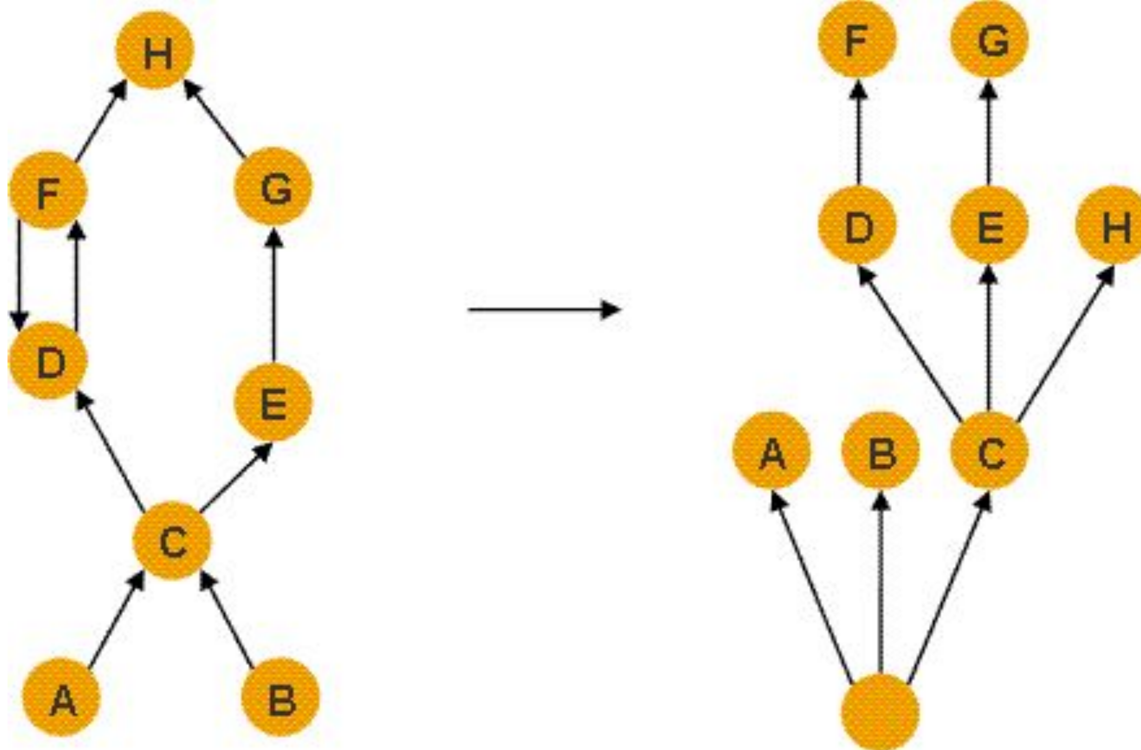


Dominator Tree

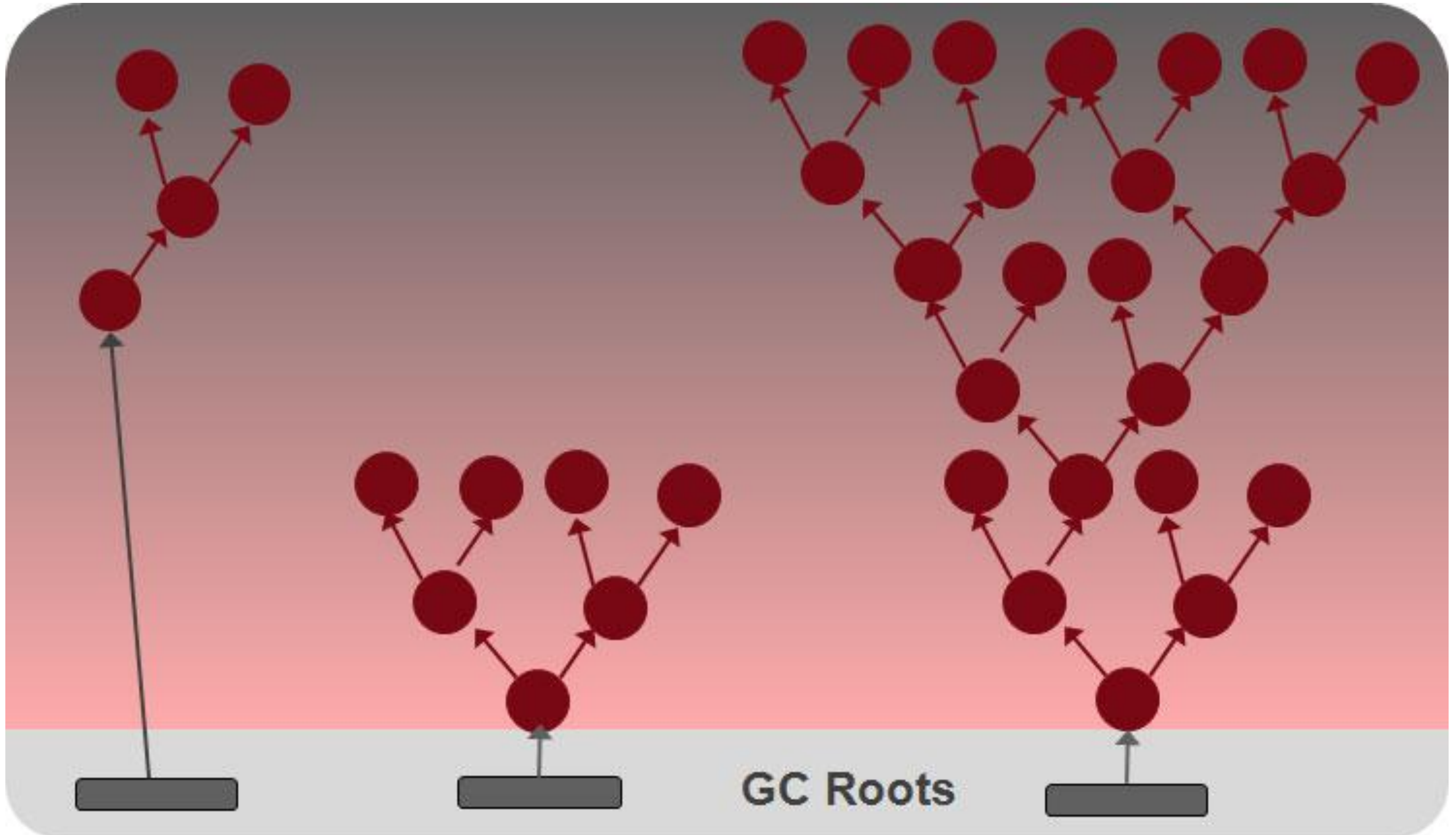
An object x **dominates** an object y if every path in the object graph from the start (or the root) node to y must go through x.

The **immediate dominator** x of some object y is the dominator closest to the object y.

A **dominator tree** is built out of the object graph.



Garbage Collection Roots



Garbage Collection Roots

System Class

JNI Local

JNI Global

Thread Block

Thread

Busy Monitor

Java Local

Native Stack

Finalizer

Unfinalized

Unreachable

Unknown

Garbage Collection Roots

Class

Thread

Stack Local

JNI Local

JNI Global

Monitor Used

Held by JVM

How is Java Heap Dump Generated?

1. Get Heap Dump on an OutOfMemoryError

2. Interactively Trigger a Heap Dump:

- By sending a signal to JVM (ctrl+break)
- Using JDK 5/6 tools (jps, jmap)
- Using JConsole
- Other ad hoc tools (e.g. Eclipse MAT)

Heap Dump on an OutOfMemoryError

```
java -XX:+HeapDumpOnOutOfMemoryError MainClass
```

Heap Dump By Sending a Signal to JVM

java -XX:+HeapDumpOnCtrlBreak MainClass

+

See Thread Dump By Sending a Signal to JVM

Heap Dump Using JDK 5/6 tools

jmap -dump:format=b,file=<filename.hprof> <pid>

Heap Dump Using JConsole

Java Monitoring & Management Console

Connection Window Help

pid: 3808

Overview | Memory | Threads | Classes | VM Summary | MBeans

- JMImplementation
- com.sun.management
  - HotSpotDiagnostic
    - Attributes
    - Operations
      - dumpHeap
      - getVMOption
      - setVMOption
- java.lang
- java.util.logging

Operation invocation

void dumpHeap (p0 ,blog\base.hprof , p1 true)

MBeanOperationInfo

Name	Value
Operation:	
Name	dumpHeap
Description	dumpHeap
Impact	UNKNOWN
ReturnType	void
Parameter-0:	

Descriptor

Name	Value
Operation:	
openType	javax.management.openmbean.SimpleType(n...
originalType	void
Parameter-0:	
openType	javax.management.openmbean.SimpleType(n...
originalType	java.lang.String

Eclipse Memory Analyser

Eclipse Memory Analyzer

File Edit Window Help

Inspector

@ 0x7dcb040
Worker
org.eclipse.core.internal.jobs
class org.eclipse.core.internal.jo...
java.lang.Thread
org.eclipse.osgi.internal.basead...
112 (shallow size)
168.653.960 (retained size)
GC root: Thread

Statics Attributes

Type	Name	Value
ref	pool	org.eclipse.c
ref	currentJob	org.eclipse.r
ref	uncaughtE...	null
ref	throwableF...	null
bool...	stopBefore...	false
ref	blockerLock	java.lang.Ol
ref	blocker	null
ref	parkBlocker	null
int	threadStatus	5
long	tid	25
long	nativePark...	0
long	stackSize	0
ref	inheritableT...	java.lang.Th
ref	threadLocals	java.lang.Th
ref	inheritedAc...	java.securit
ref	contextCla...	org.eclipse.c
ref	group	main
ref	target	null
bool...	stillborn	false
bool...	daemon	false
bool...	single_step	false
long	eetop	1230009344
ref	threadQ	null
int	priority	5
ref	name	Worker-2

mat_eclipse35.hprof

Overview

Details

Size: **186,9 MB** Classes: **6,9k** Objects: **1,3m** Class Loader: **157**

Biggest Objects by Retained Size

160,8 MB
9,5 MB
843,6 KB
15,7 MB

Total: 186,9 MB

org.eclipse.core.internal.jobs.Worker @ 0x7dcb040 Worker-2

Shallow Size: **112 B** Retained Size: **160,8 MB**

Actions

Histogram: Lists number of instances per class

Dominator Tree: List the **biggest objects** and what they keep alive.

Top Consumers: Print the most **expensive objects** grouped by

Reports

Leak Suspects: includes leak suspects and a system overview

Top Components: list reports for components bigger than 1 percent of the total heap.

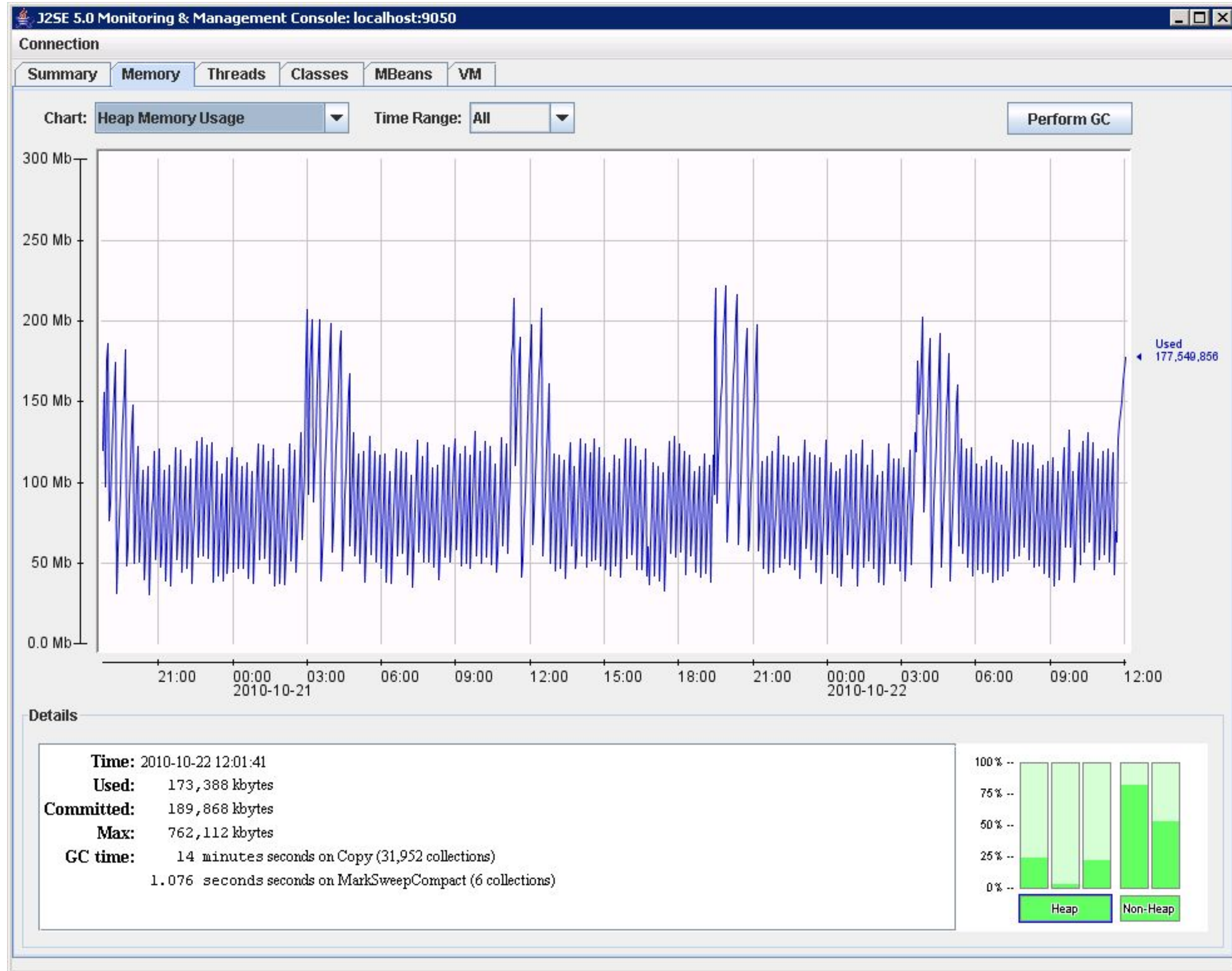
Step By Step

Component Report: Analyze objects which belong to a **common root package** or **class loader**.

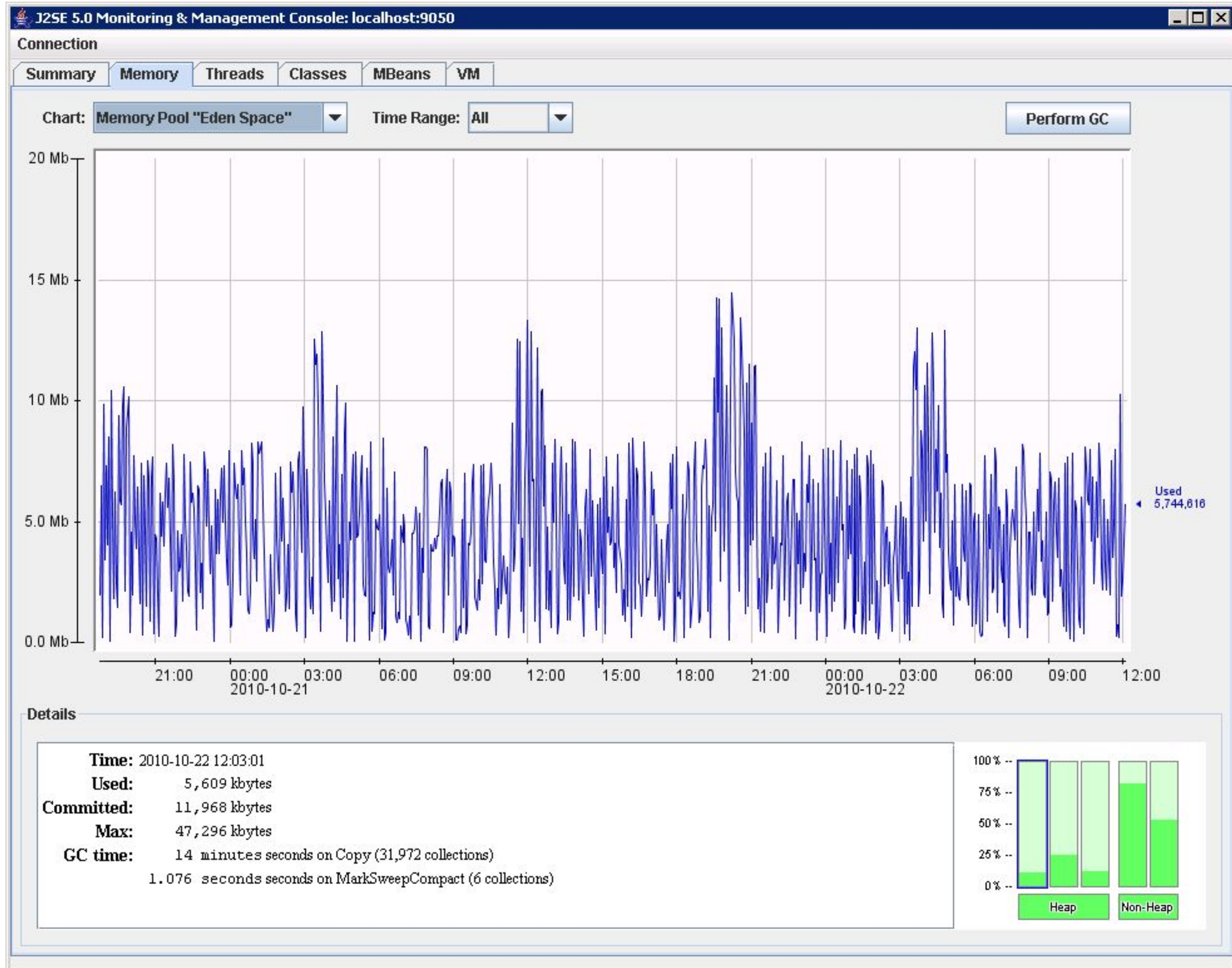
36M of 73M

Example 3: Memory Leak

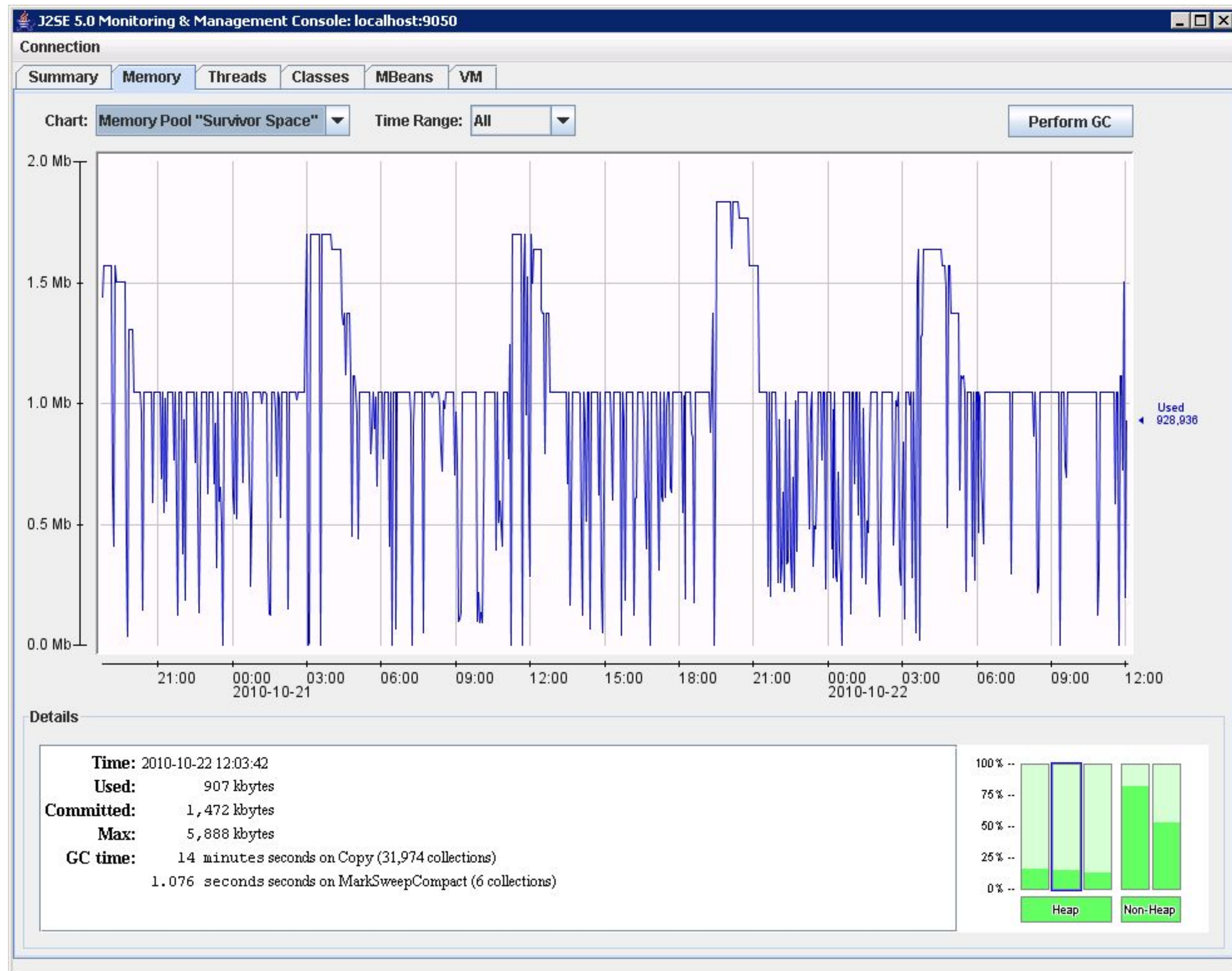
Heap Memory Usage



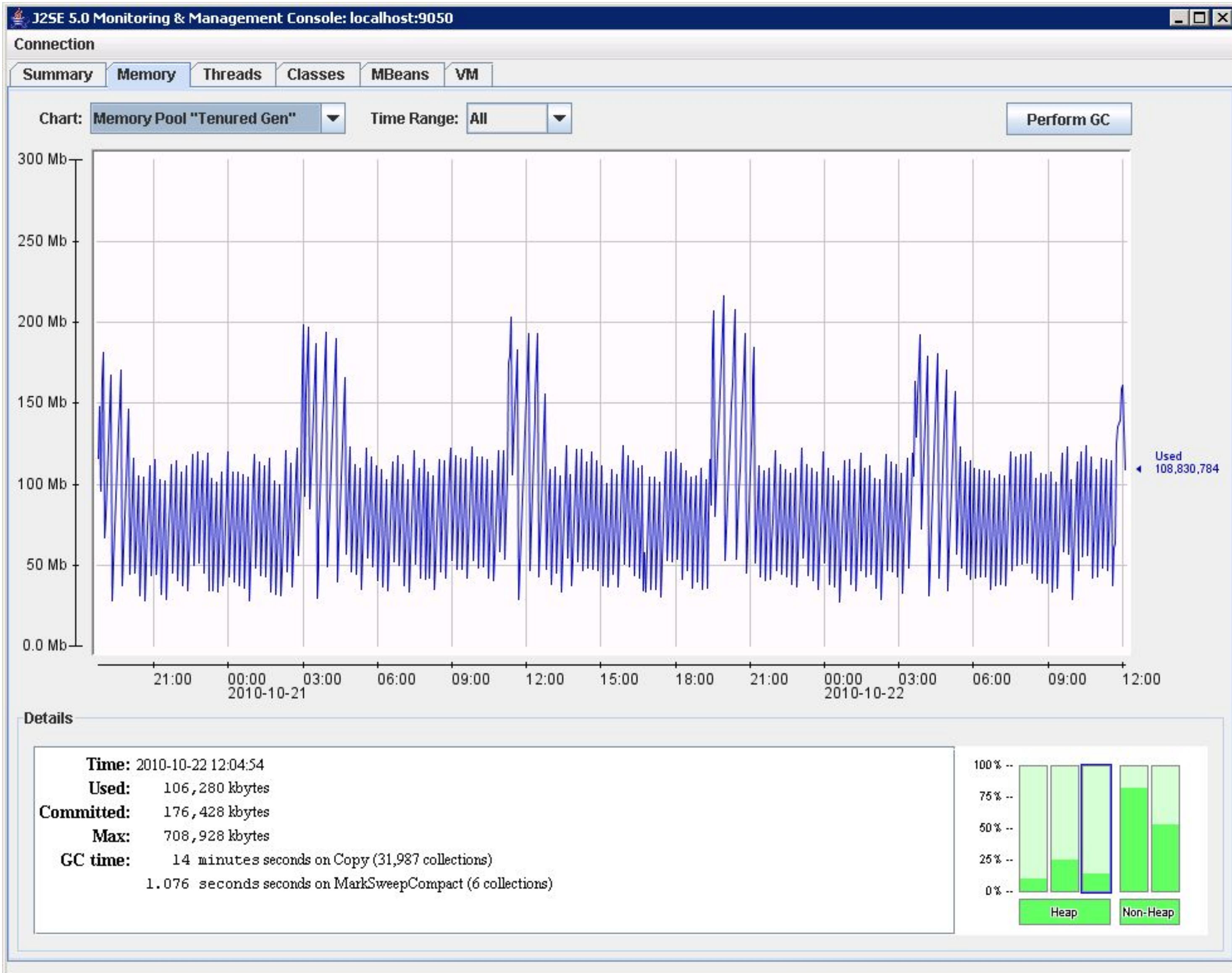
Memory Pool “Eden Space”



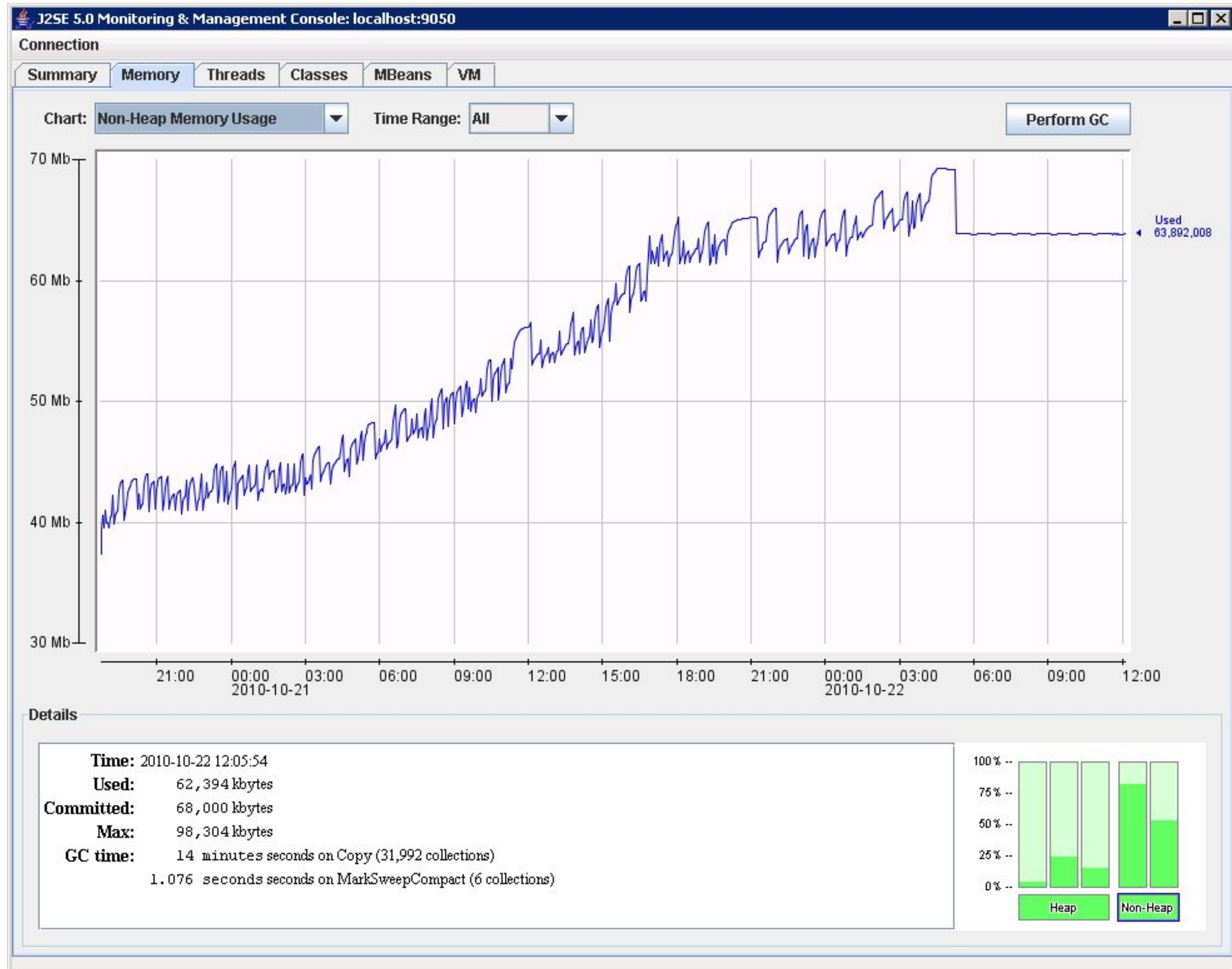
Memory Pool "Survivor Space"



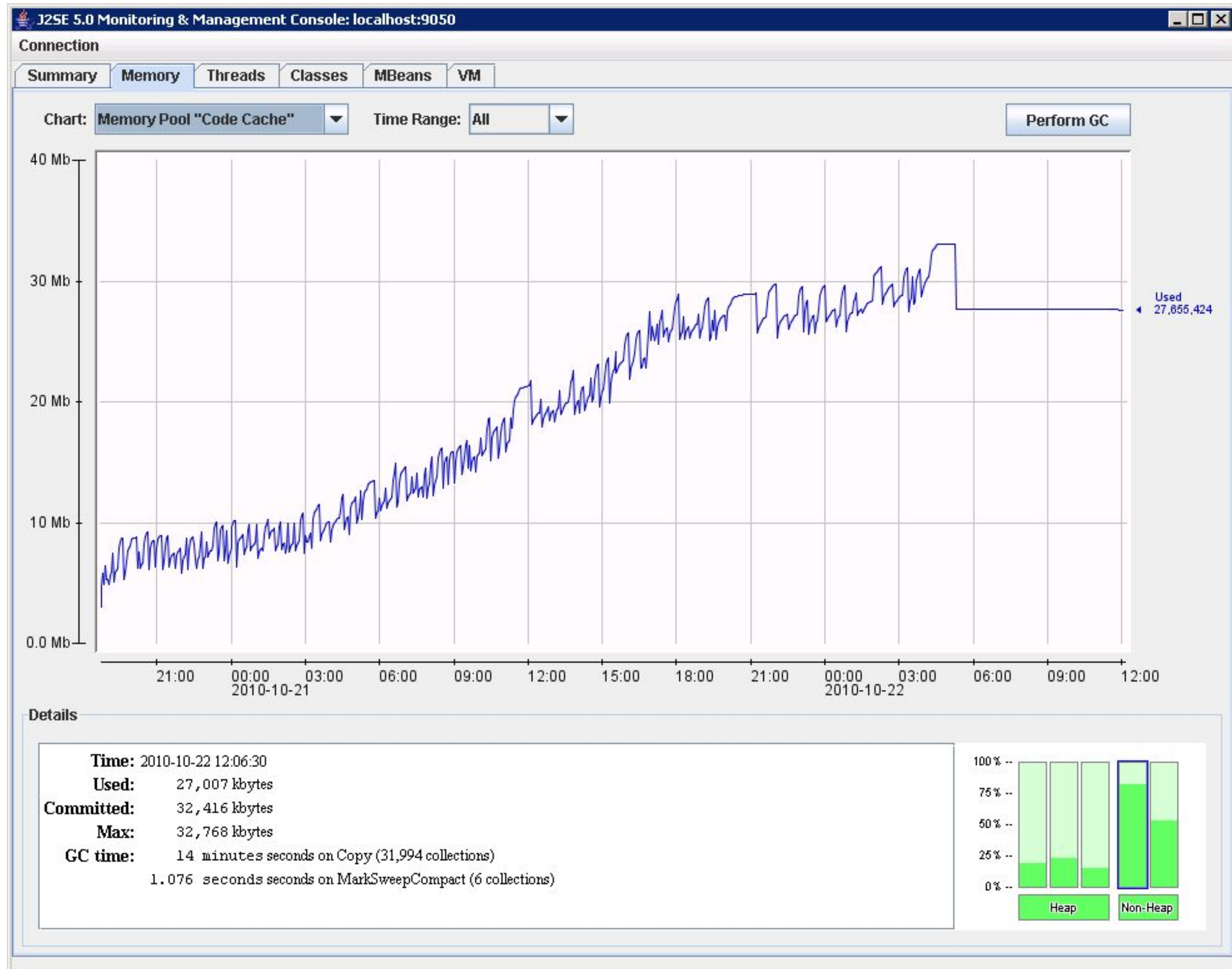
Memory Pool "Tenured Gen"



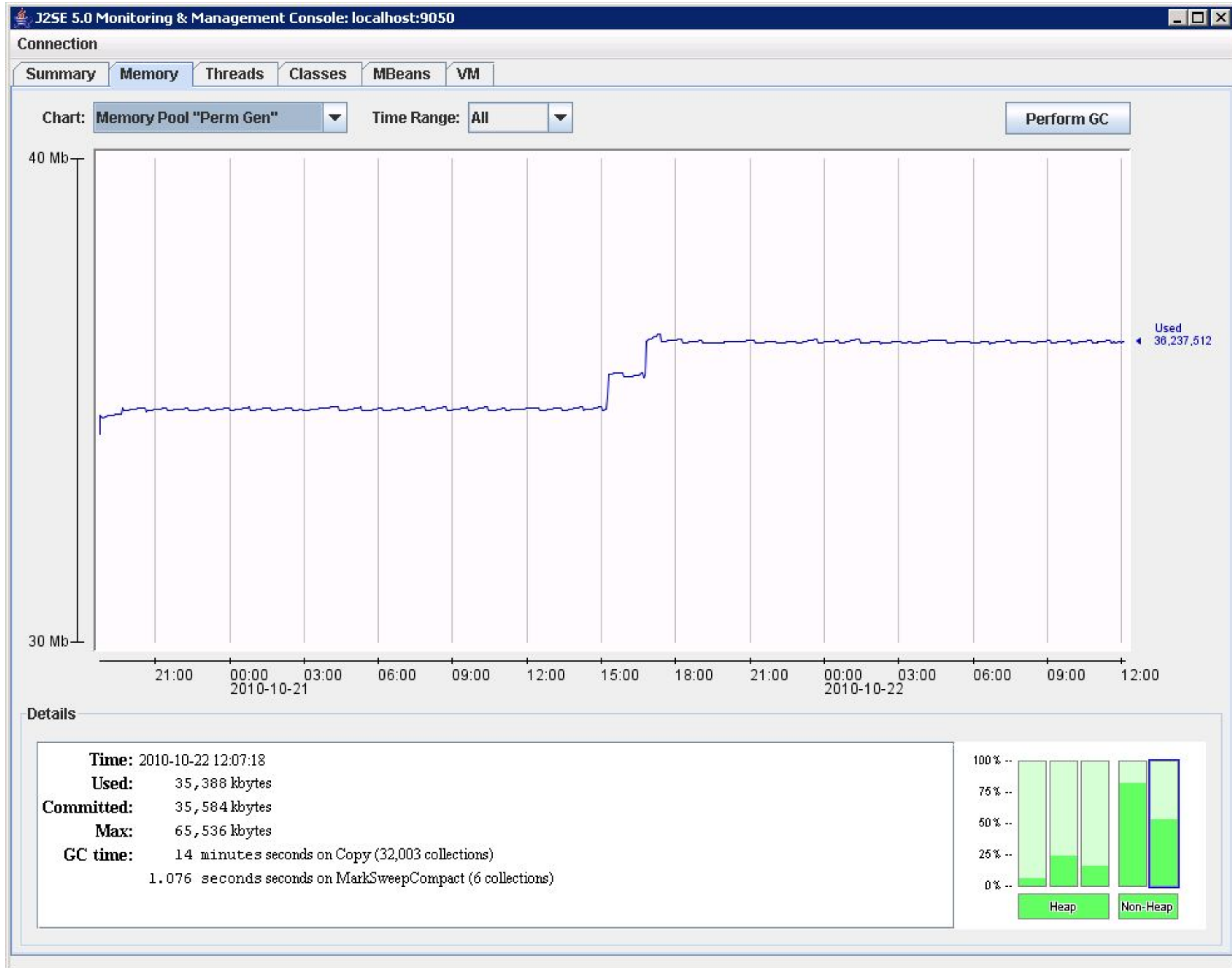
Non-Heap Memory Usage



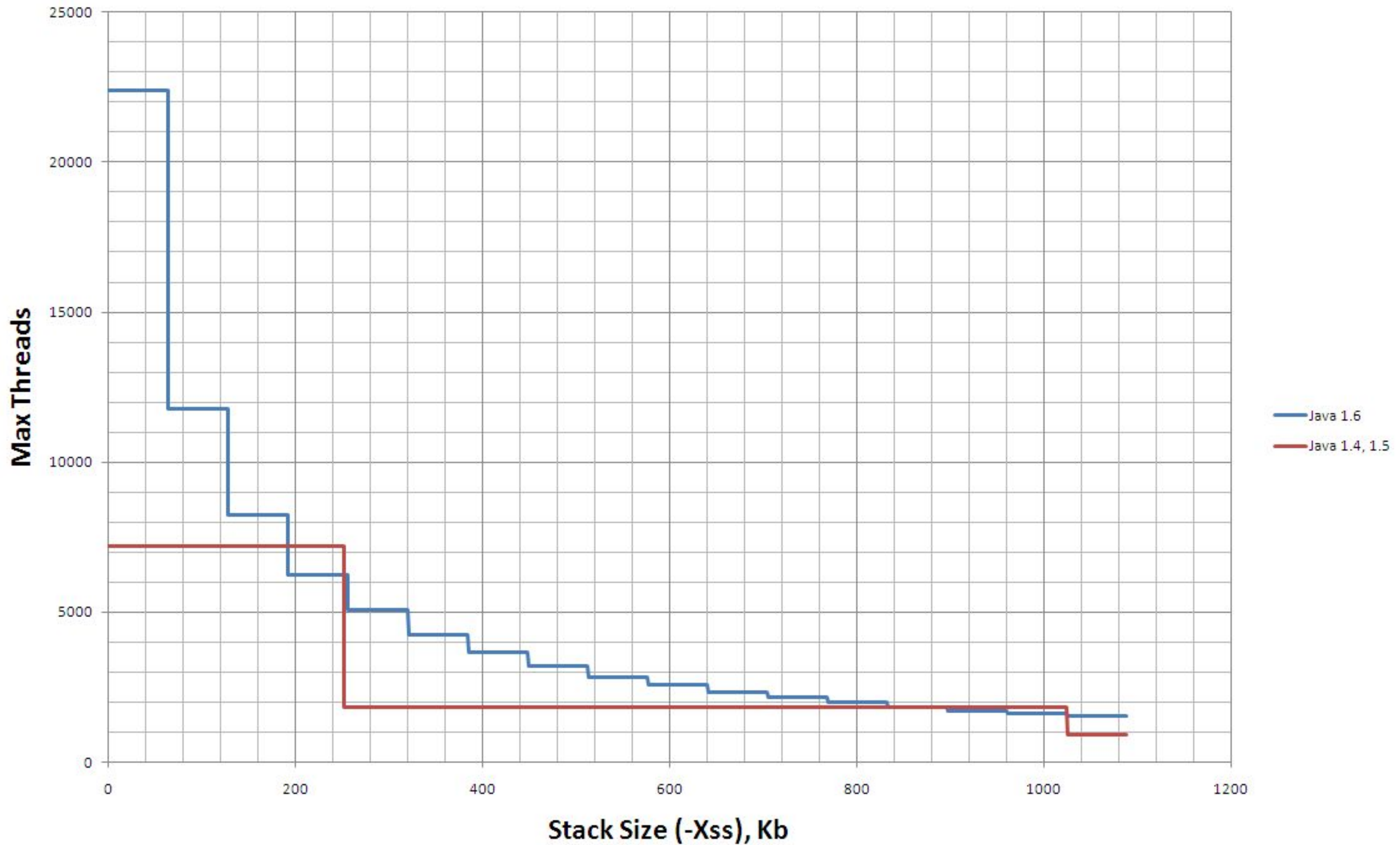
Memory Pool “Code Cache”



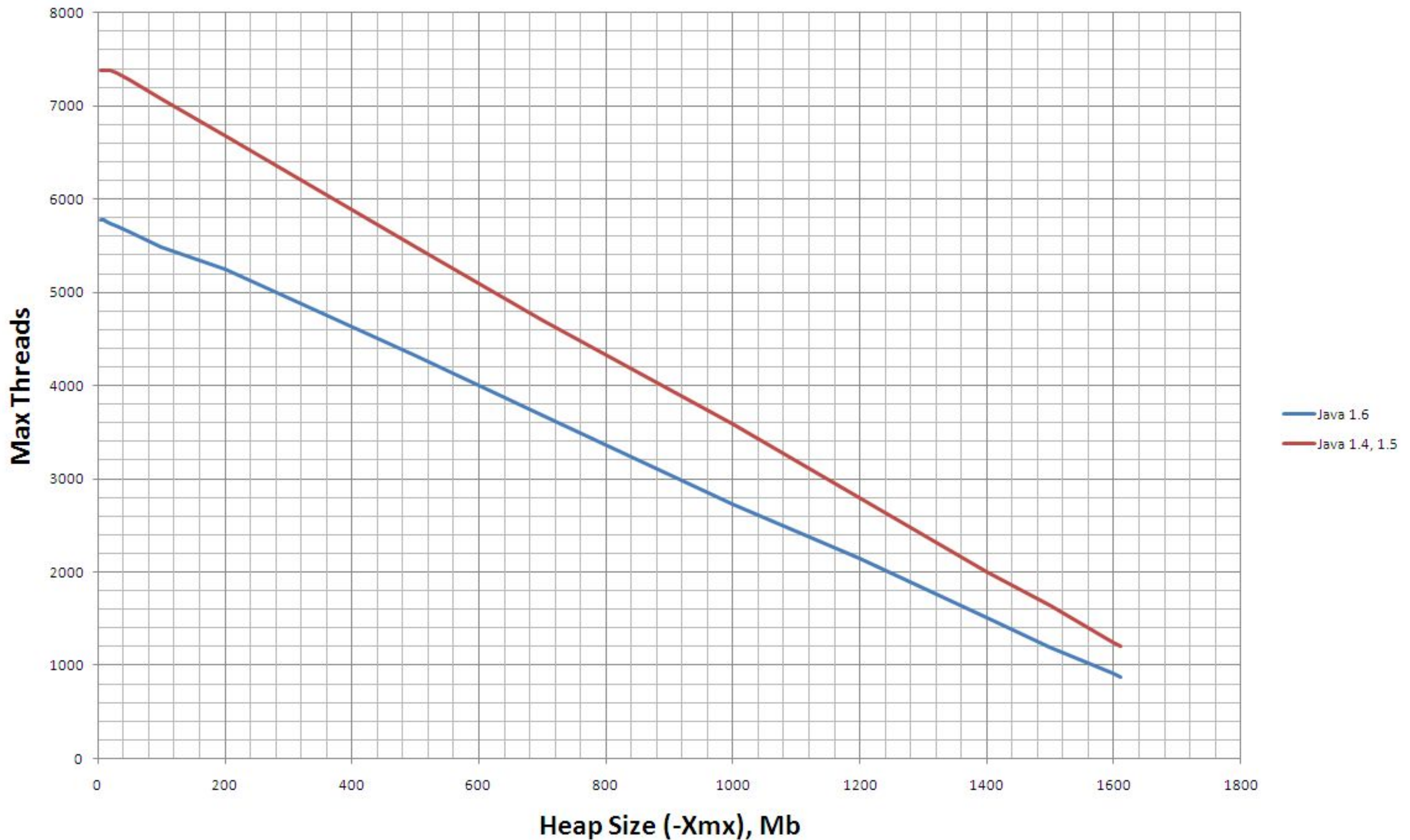
Memory Pool “Perm Gen”



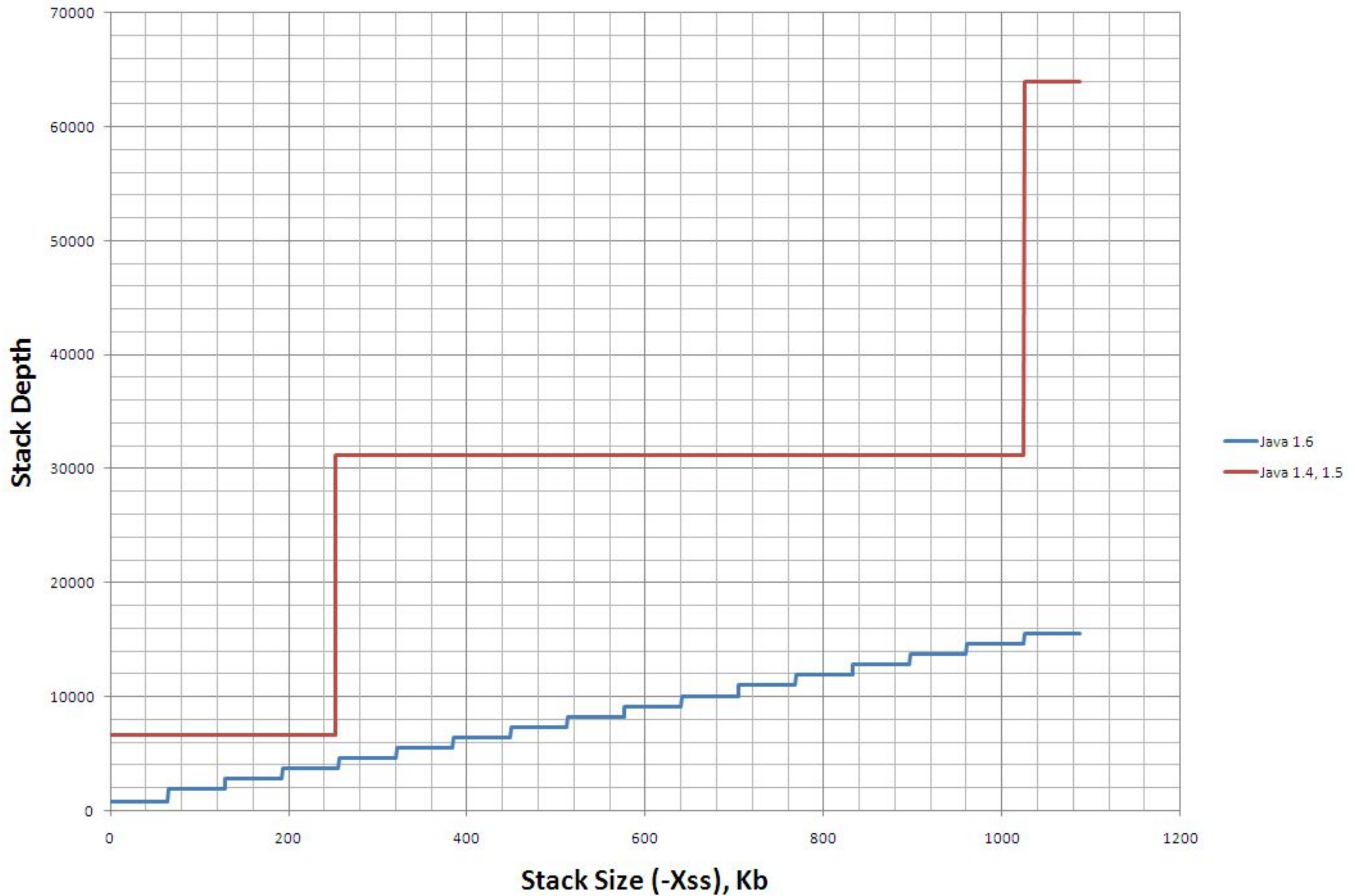
Max Thread Count Depends on Stack Size



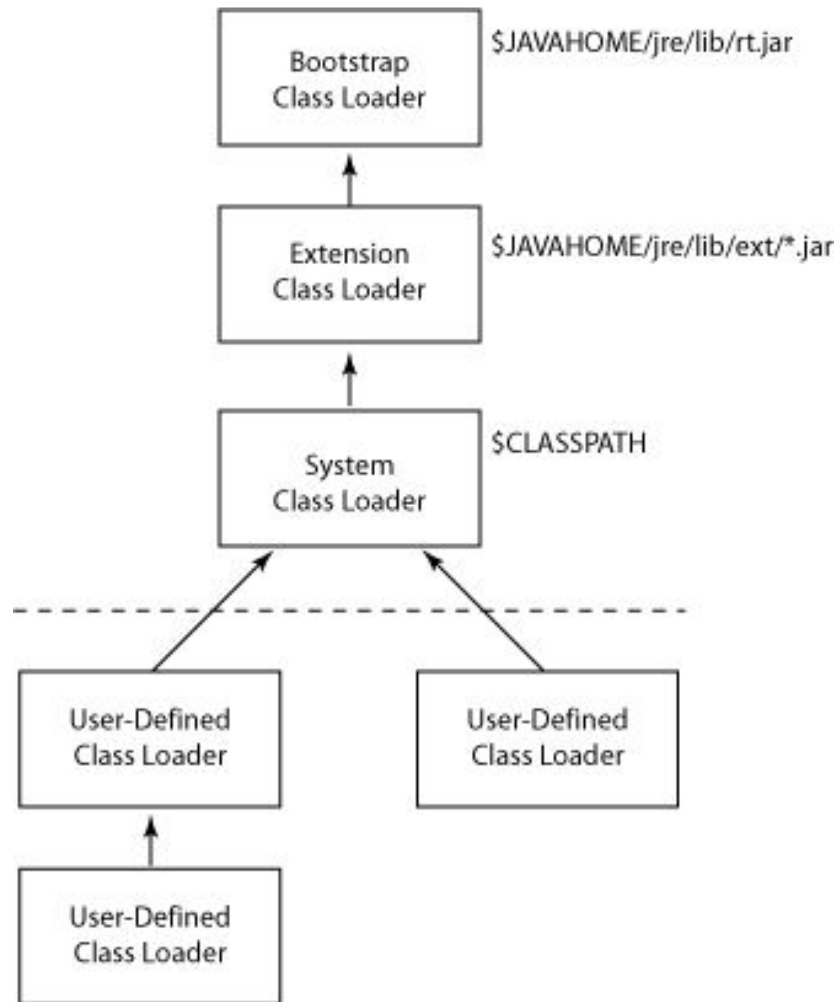
Max Thread Count Depends on Heap Size



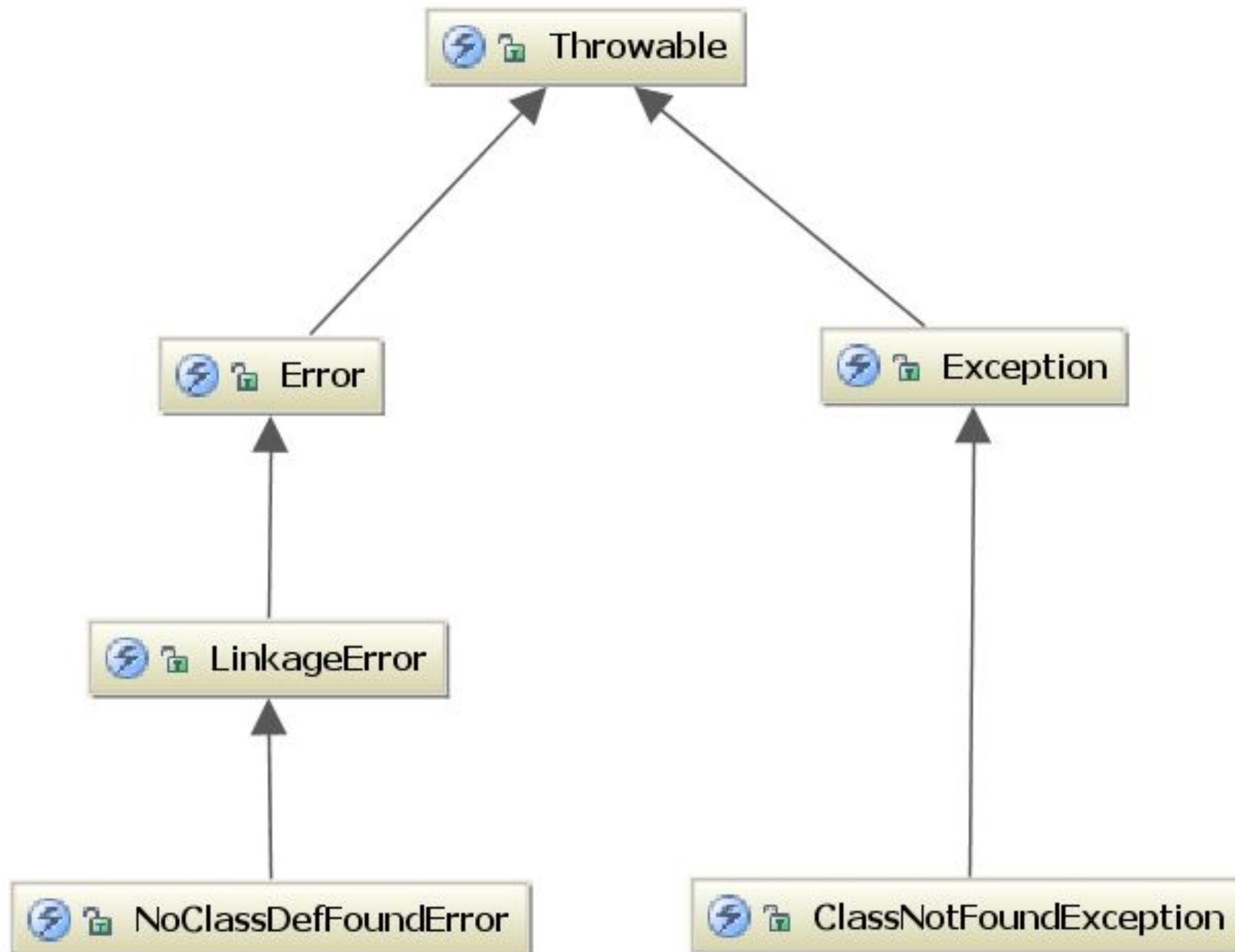
Stack Depth Depends on Stack Size



Java Class Loading



NoClassDefFoundError vs ClassNotFoundException



The End