Java Troubleshooting and Diagnostic

Roman Makarevich

<epam> Delivering Excellence in Software Engineering

Dealing with Errors

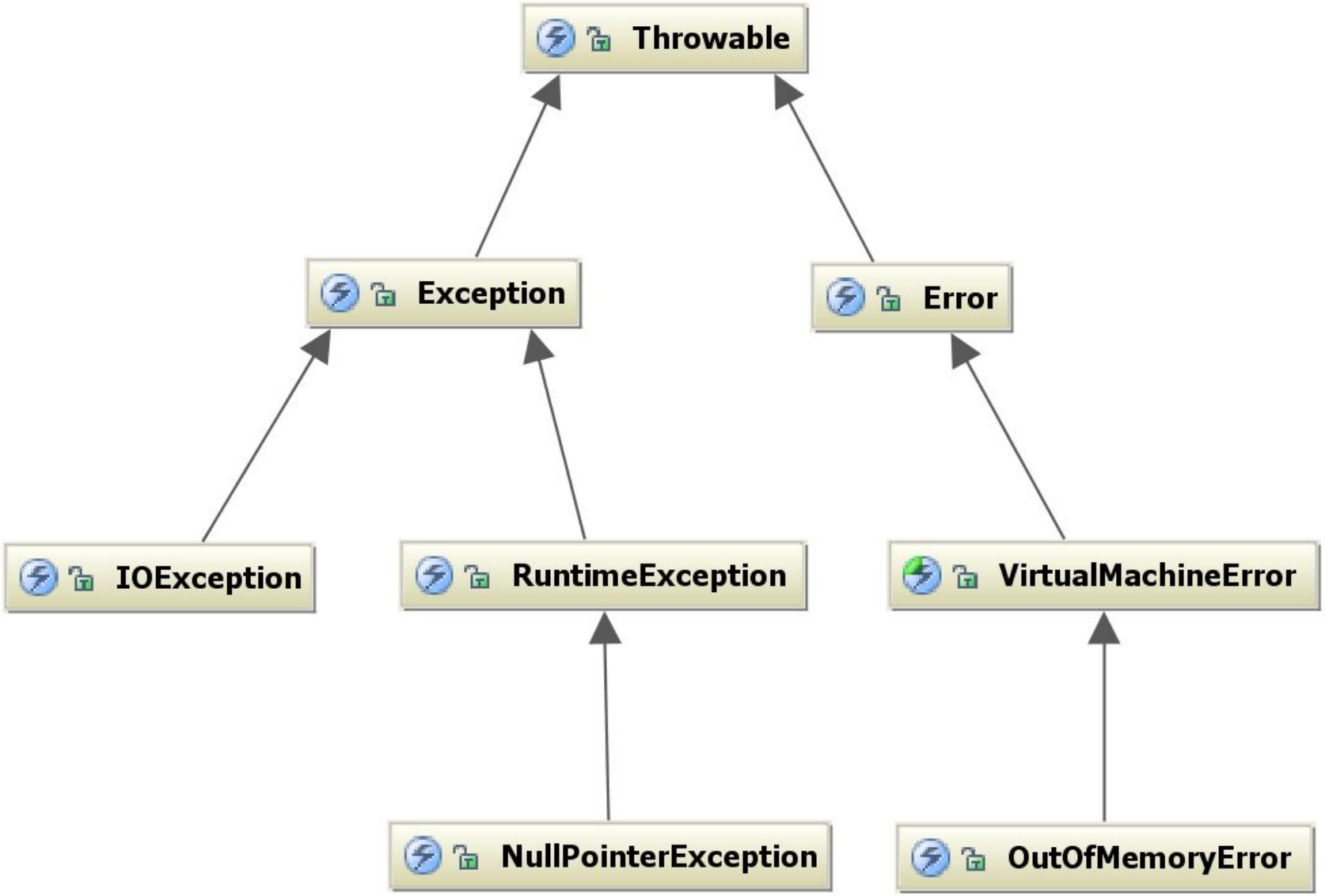
User input errors: In addition to the inevitable typos, some users like to blaze their own trail instead of following directions. Suppose, for example, that a user asks to connect to a URL that is syntactically wrong. Your code should check the syntax, but suppose it does not. Then the network package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

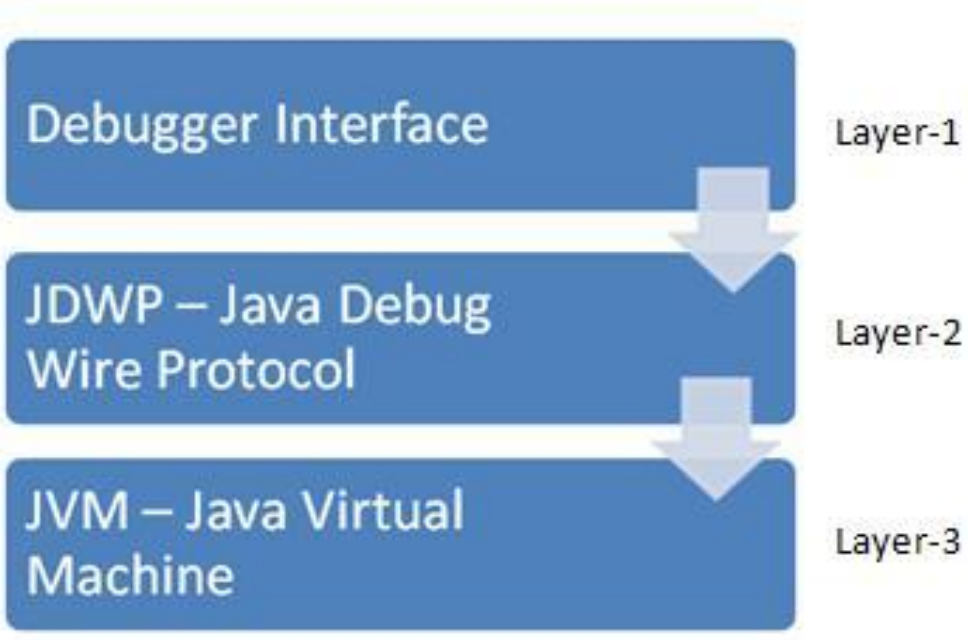
Physical limitations: Disks can fill up; you can run out of available memory.

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

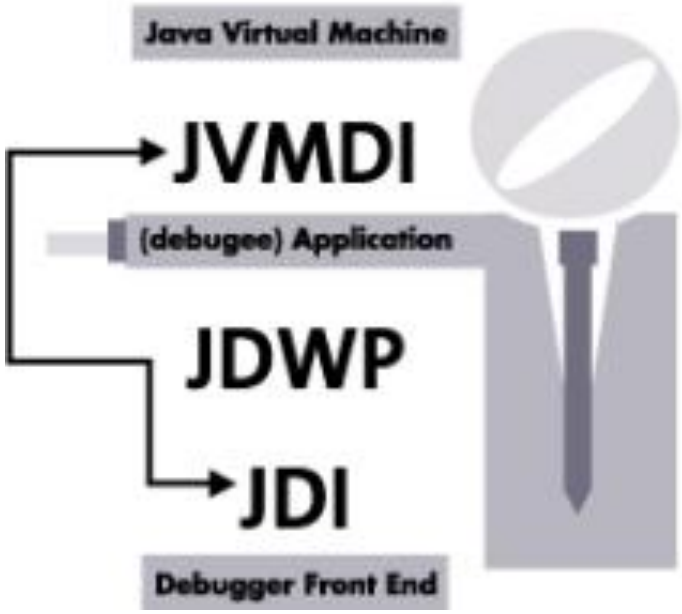
For JDK 1.4.x

-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:

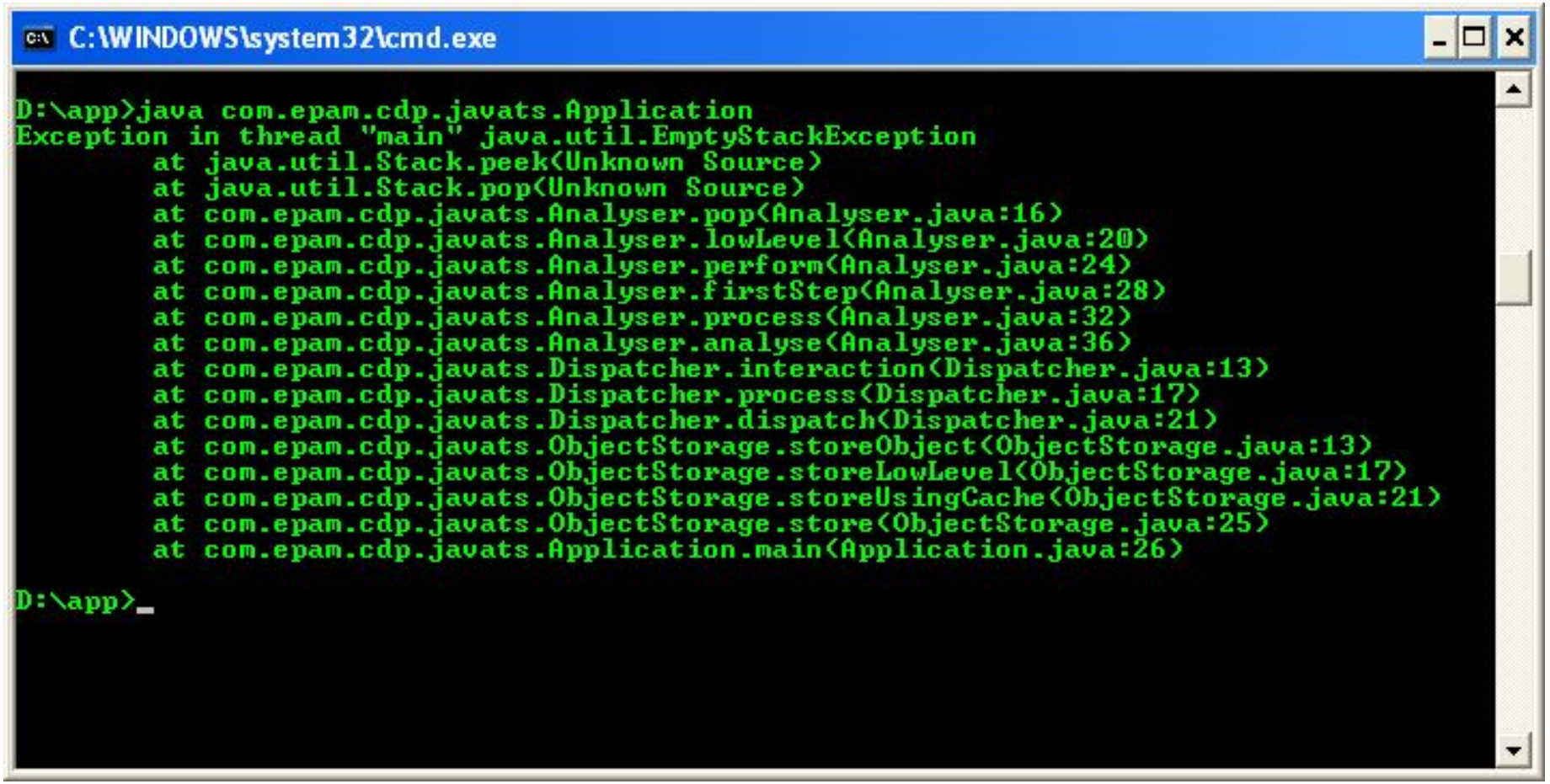
```
java version "1.5.0_11"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_11-b03)
Java HotSpot(TM) Client VM (build 1.5.0_11-b03, mixed mode, sharing)
```

- 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?

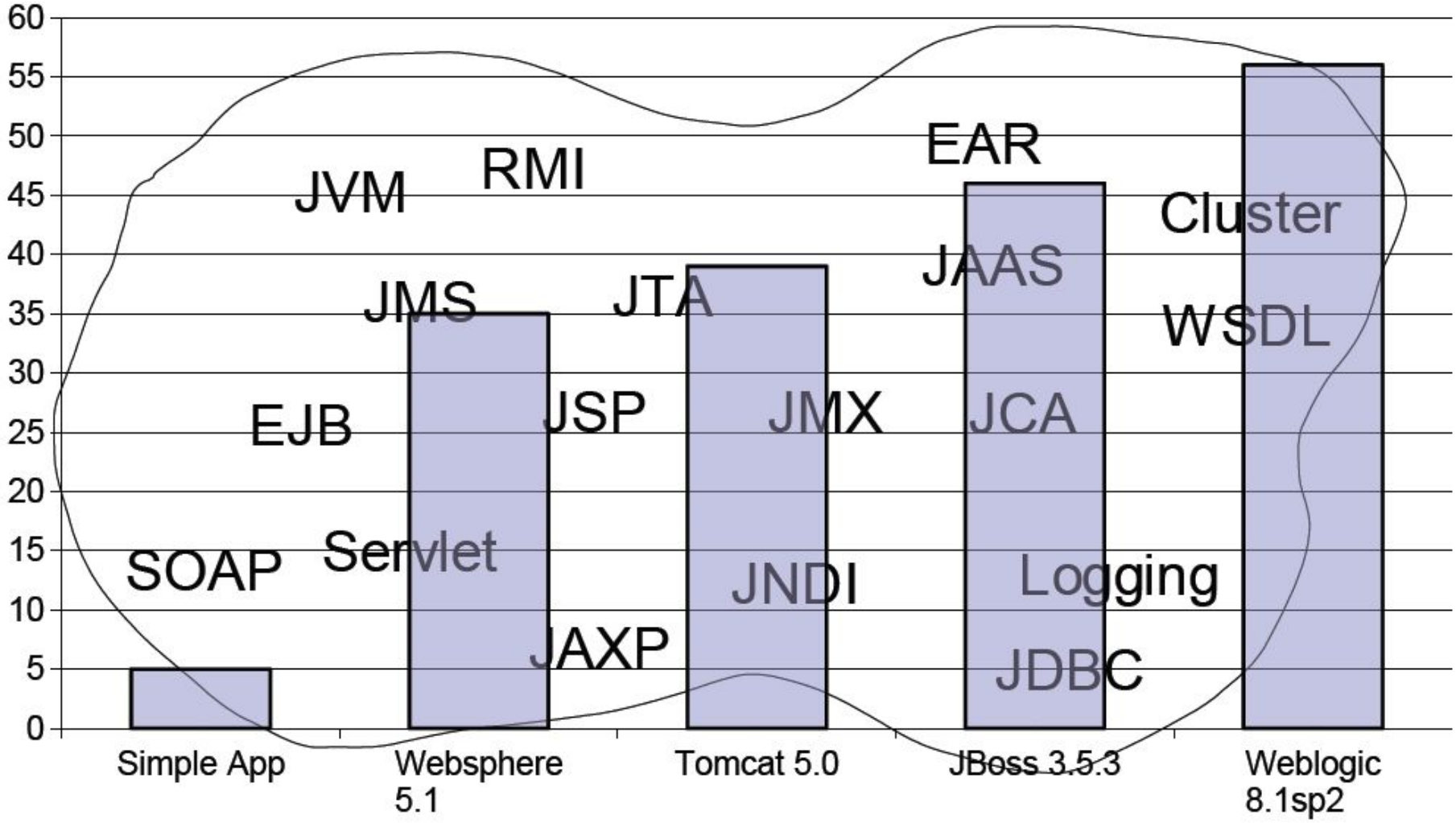
Java stack trace is a user-friendly snapshot of the Java thread.



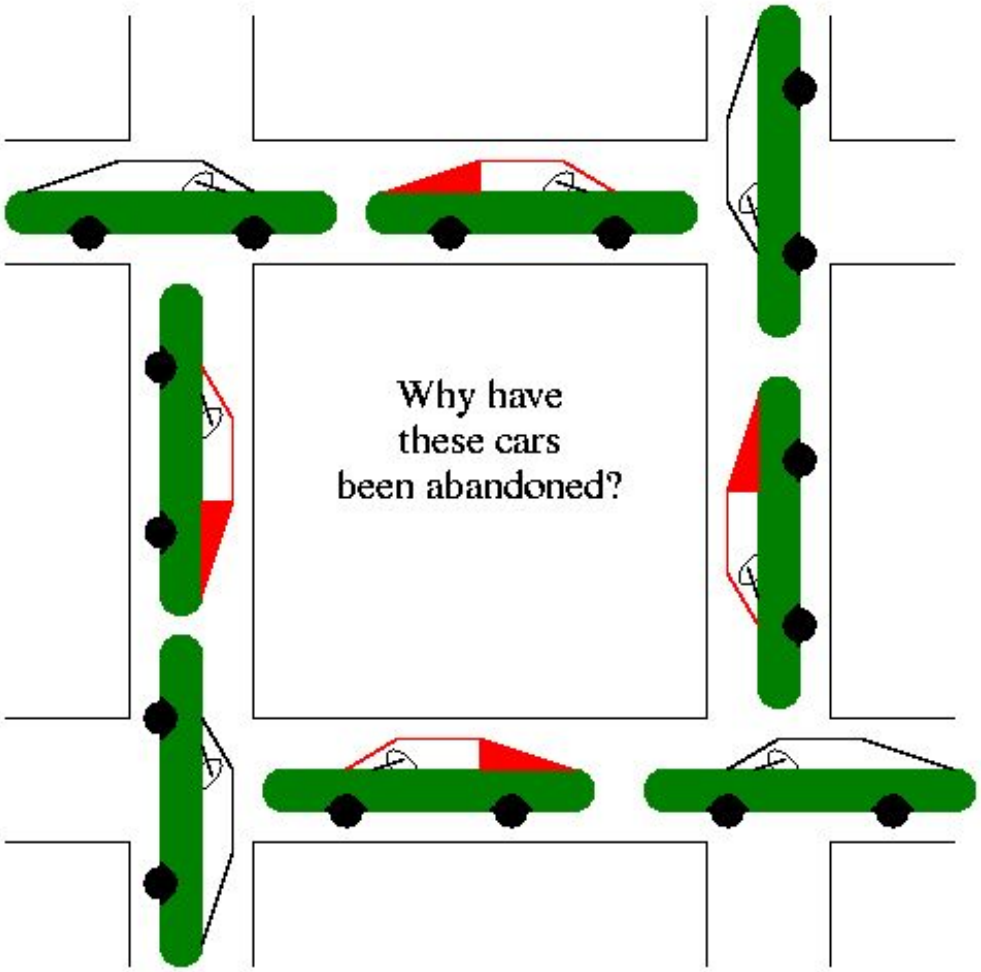
Java Bytecode Debugging Information

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

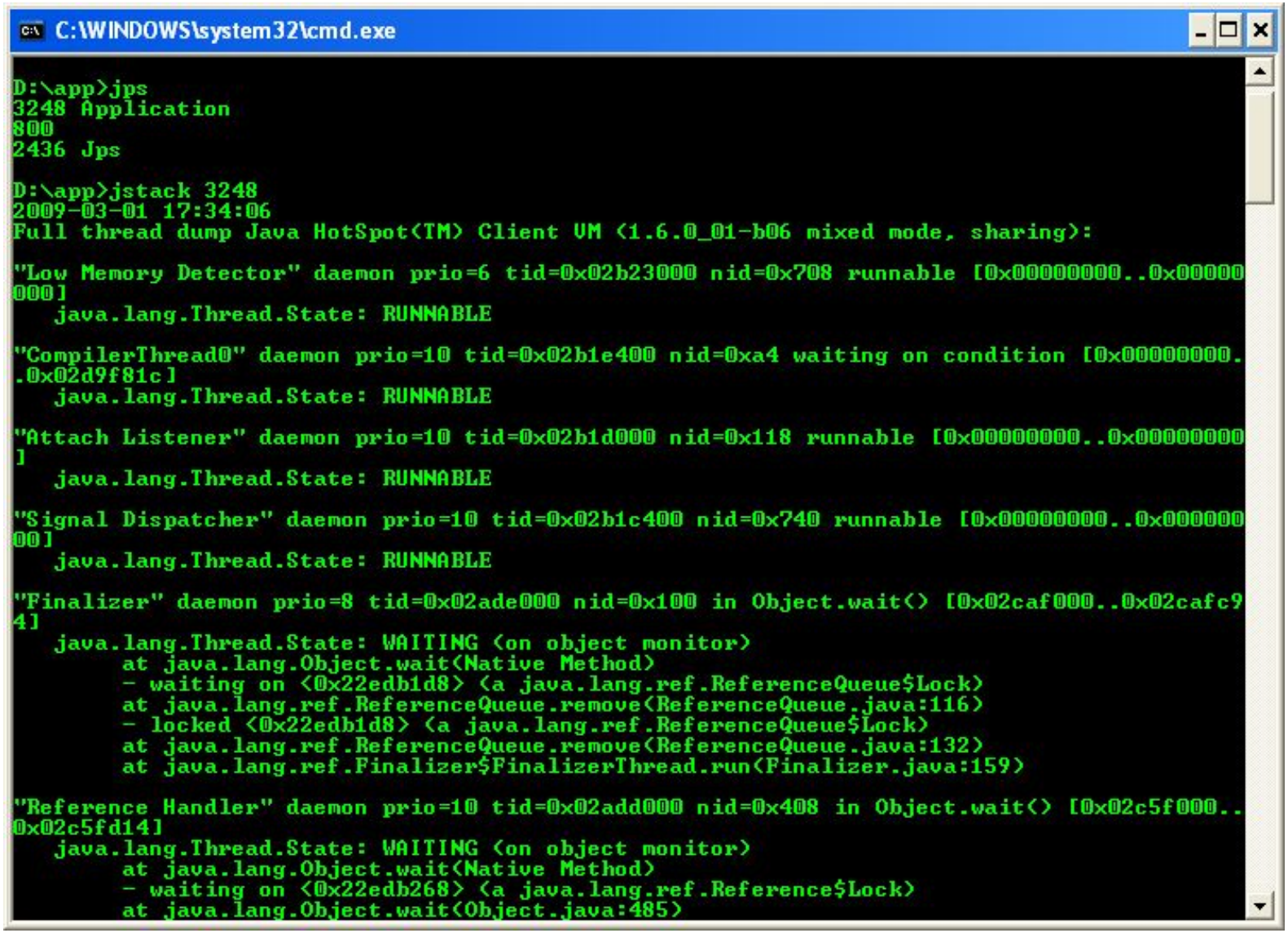
Notes:

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

Thread Dump Using JDK 5/6 tools

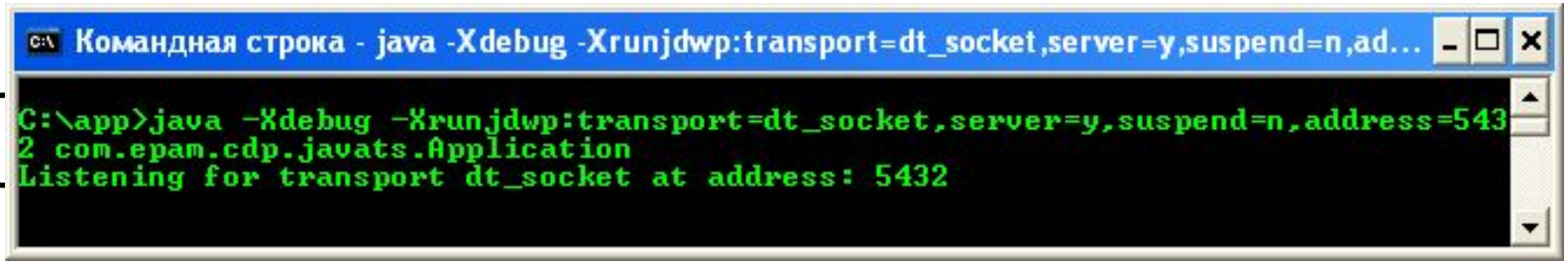
jps

jstack



Thread Dump Using Debugging Tools

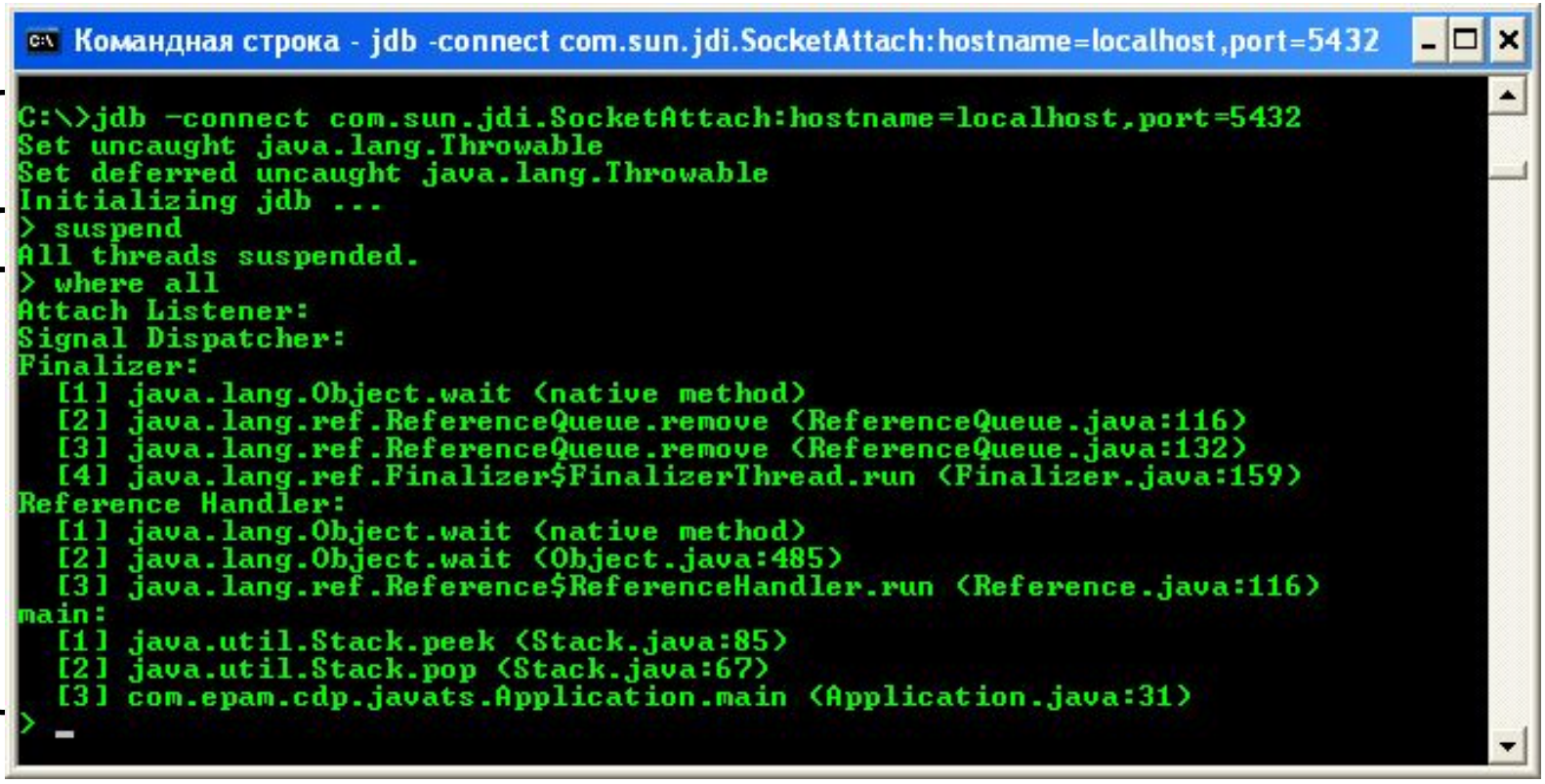
java with debug



jdb

suspend all threads

get thread dump



Using a Debugger

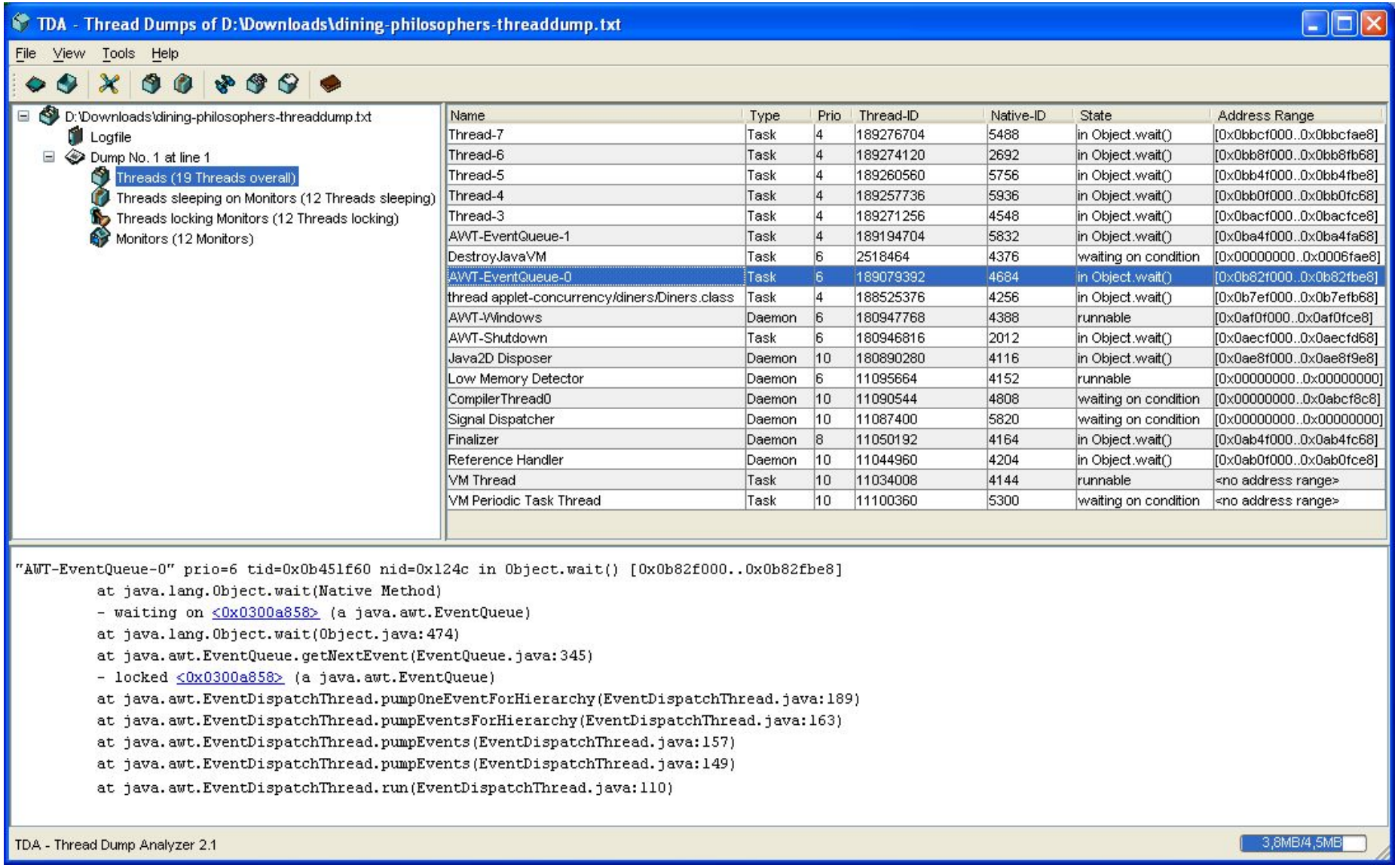
threads [tHReadgroup]	Lists threads
thread tHRead_id	Sets default thread
suspend [tHRead_id(s)]	Suspends threads (default: all)
resume [tHRead_id(s)]	Resumes threads (default: all)
where [thread_id] or all	Dumps a thread's stack
wherei [tHRead_id] or all	Dumps a thread's stack and program counter info
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

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

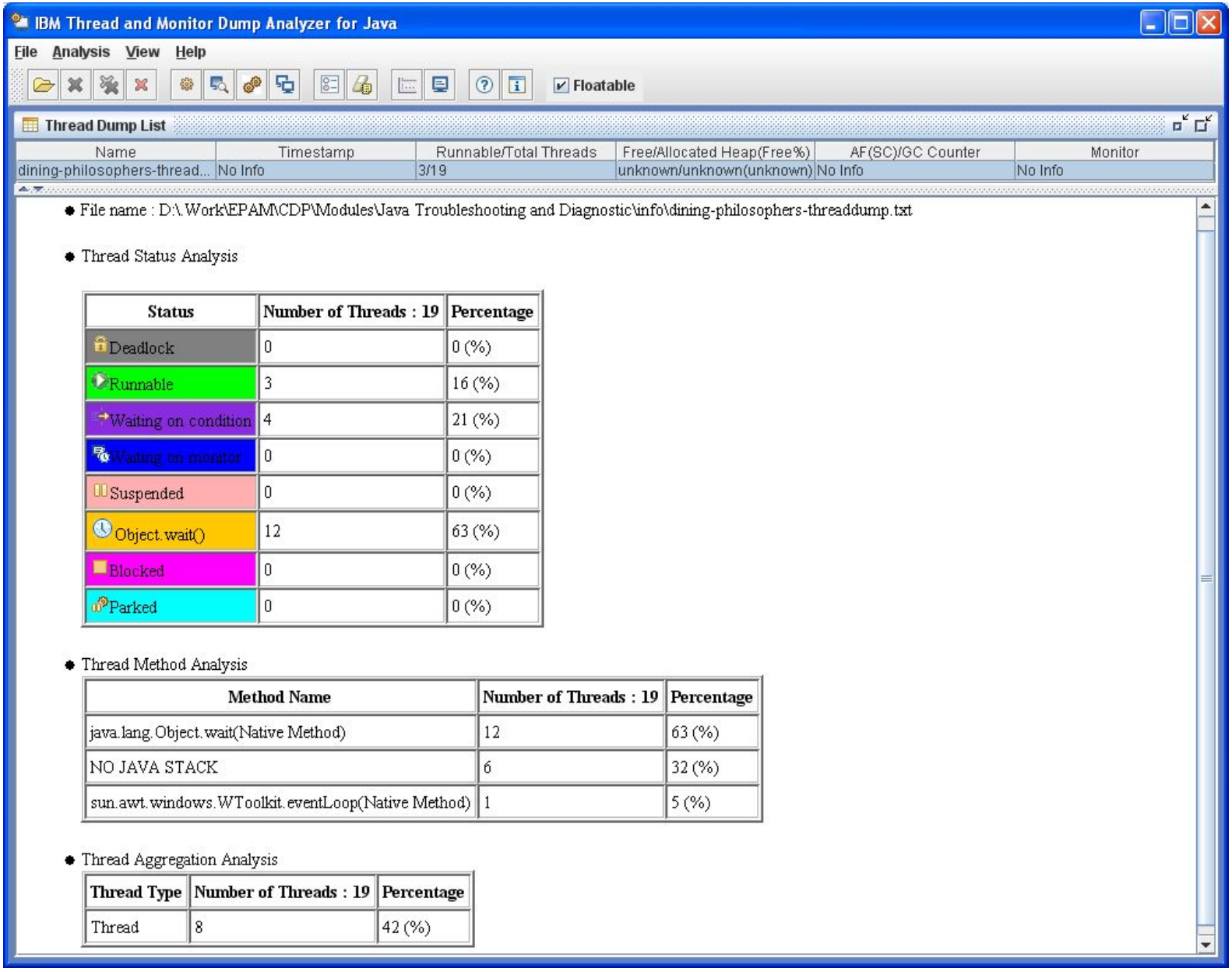
Using Java API calls

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

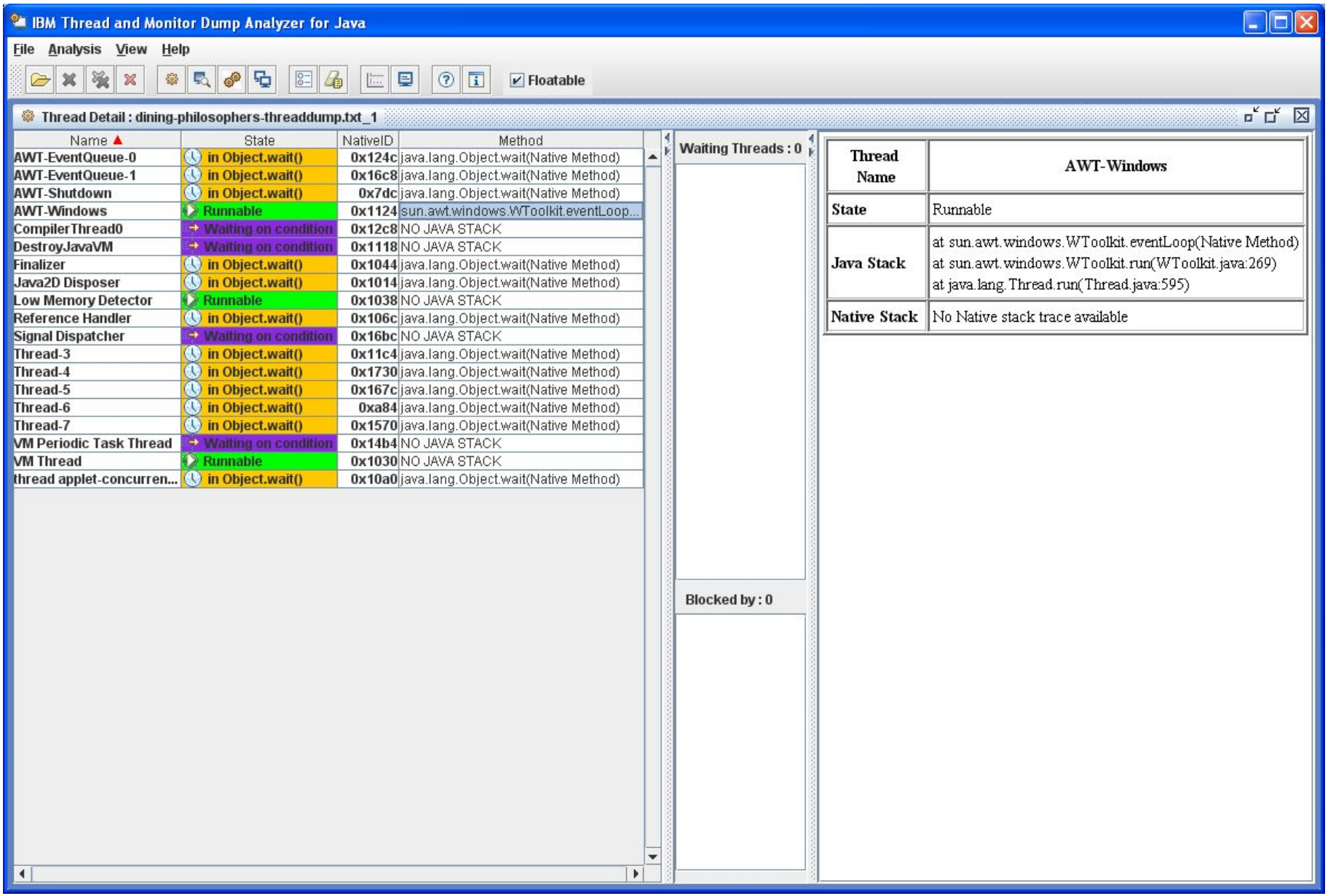
Thread Dump Analyser



IBM Thread & Monitor Dump Analyser



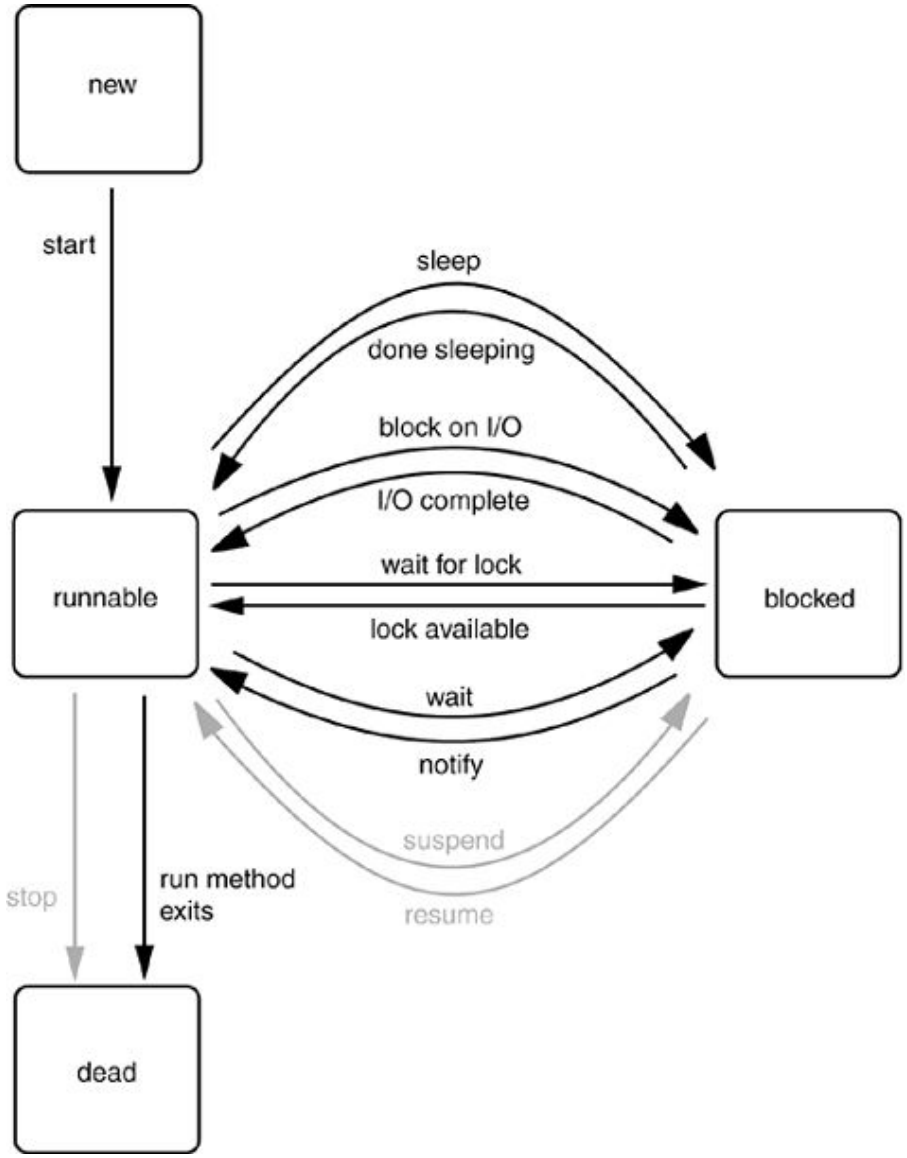
IBM Thread & Monitor Dump Analyser



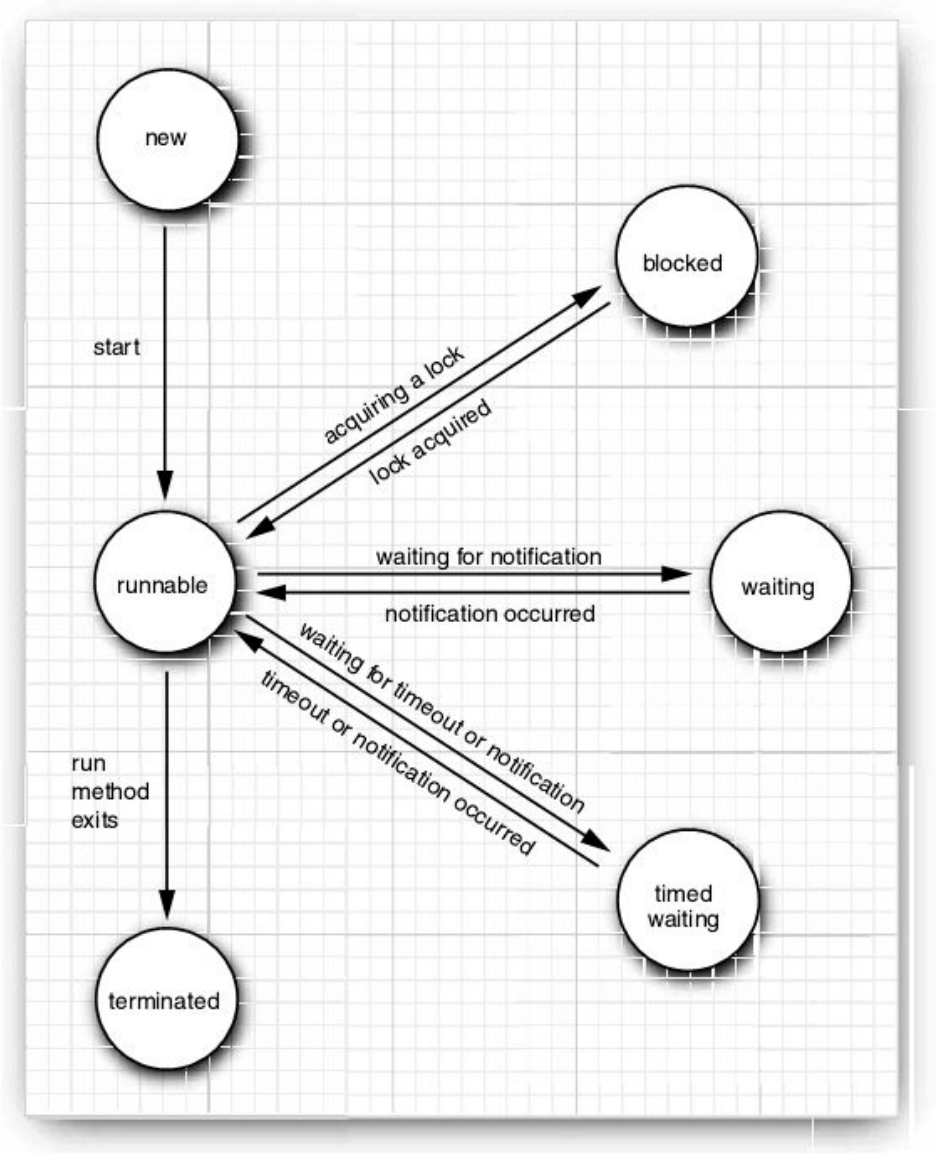
Determining the Thread States

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

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

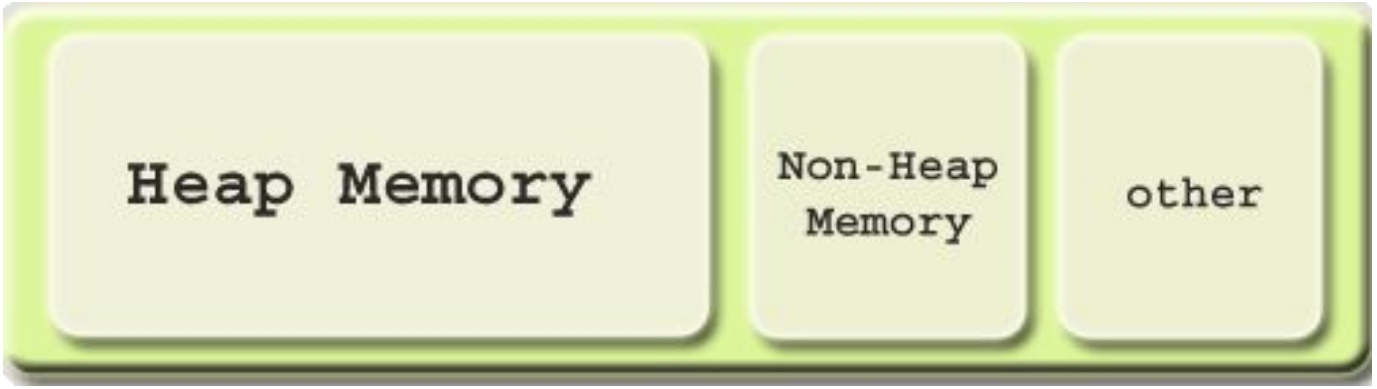
org.apache.log4j.Category.callAppenders():

```
    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

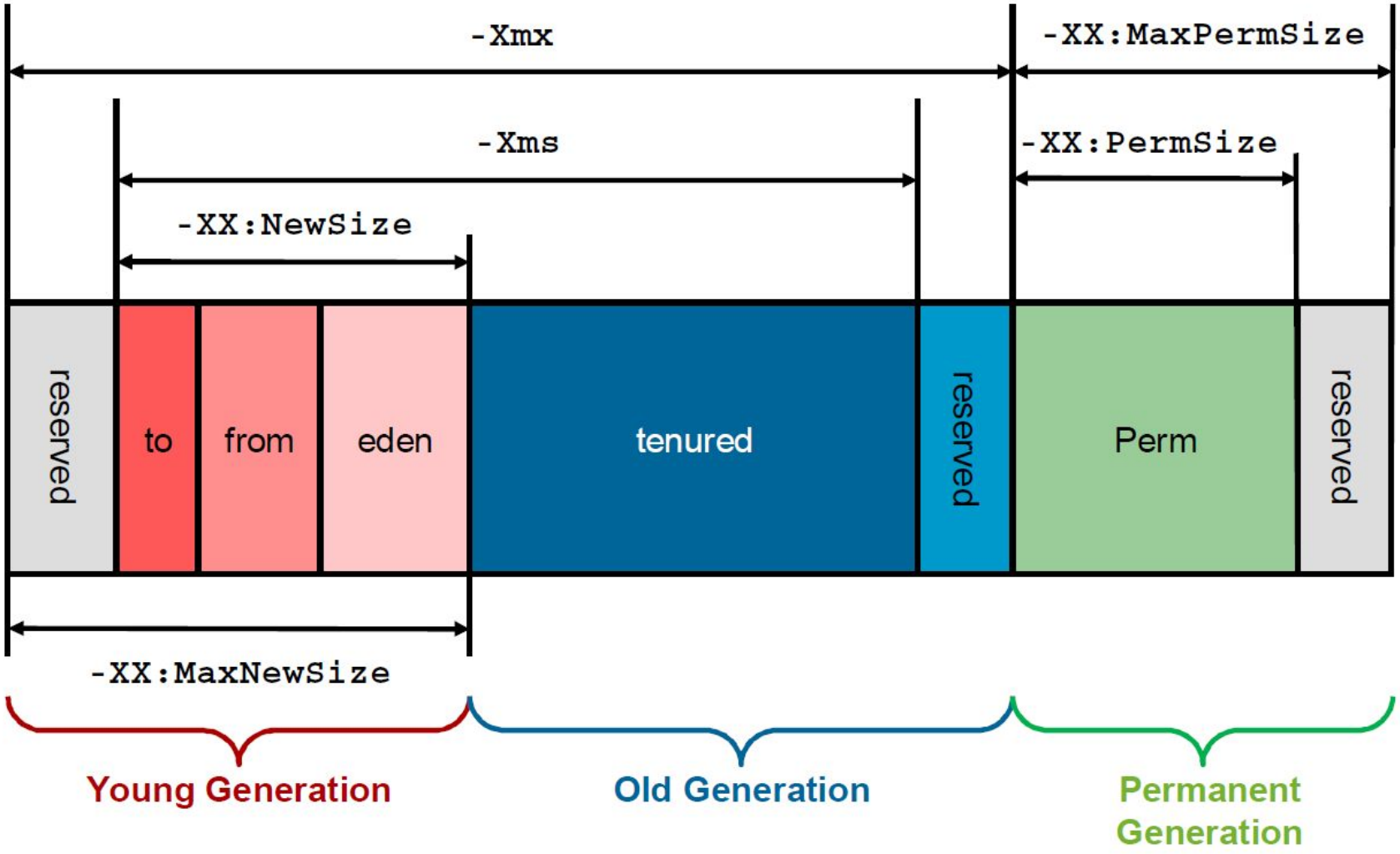


- Eden Space
- Survivor Space
- Tenured Generation

- 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

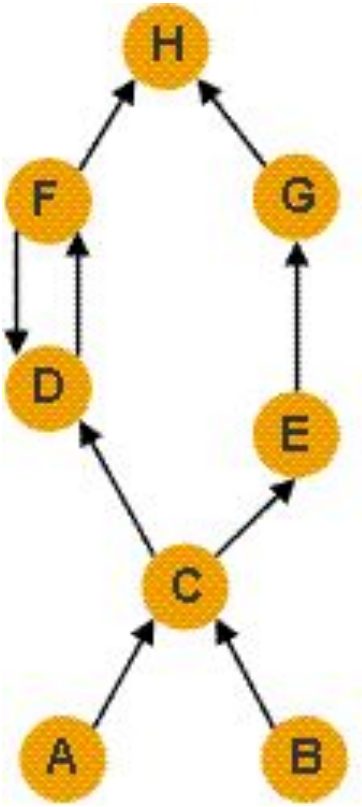
Shallow vs. Retained Heap

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.

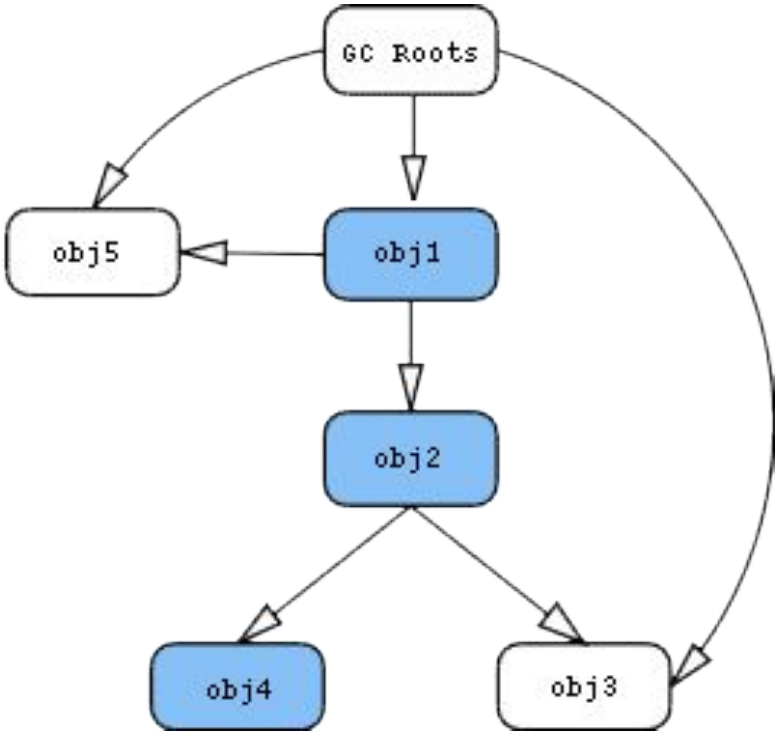
Shallow vs. Retained Heap

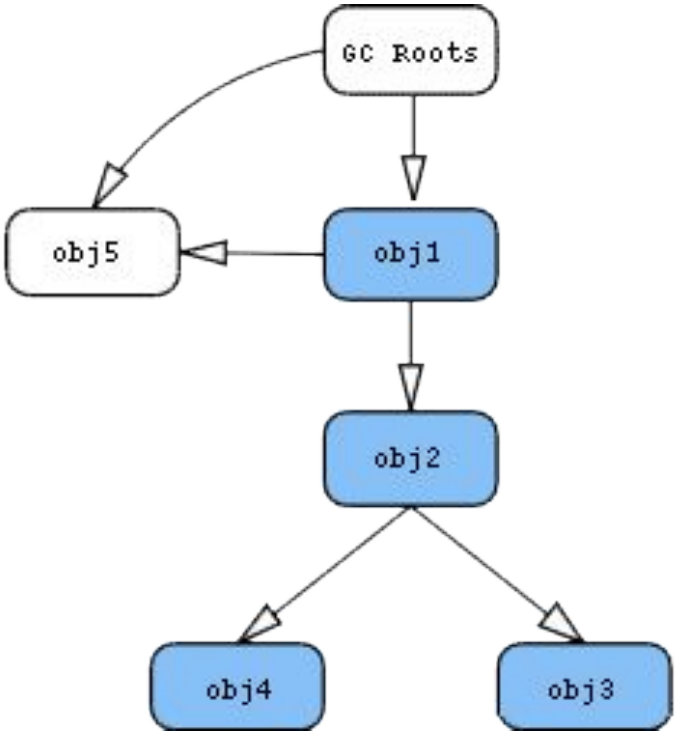


A and **B** are **garbage collection roots**, e.g. method parameters, locally created objects, objects used for wait(), notify() or synchronized(), etc.

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



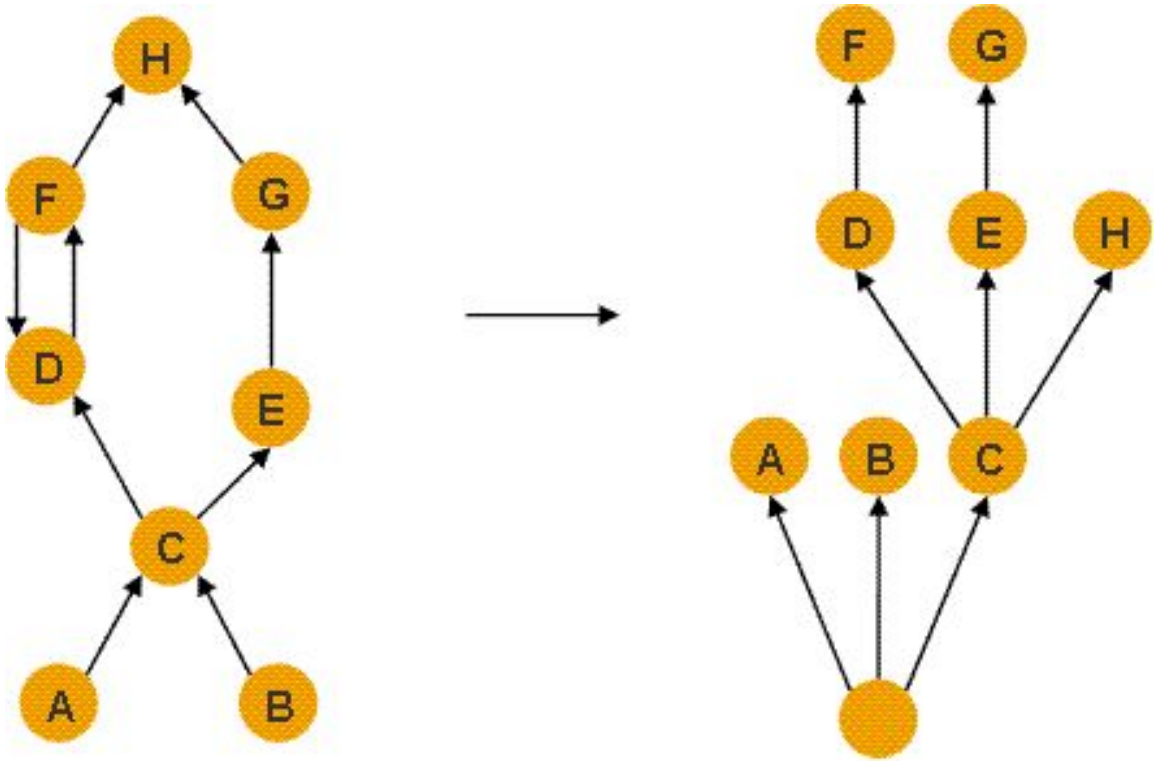


Dominator Tree

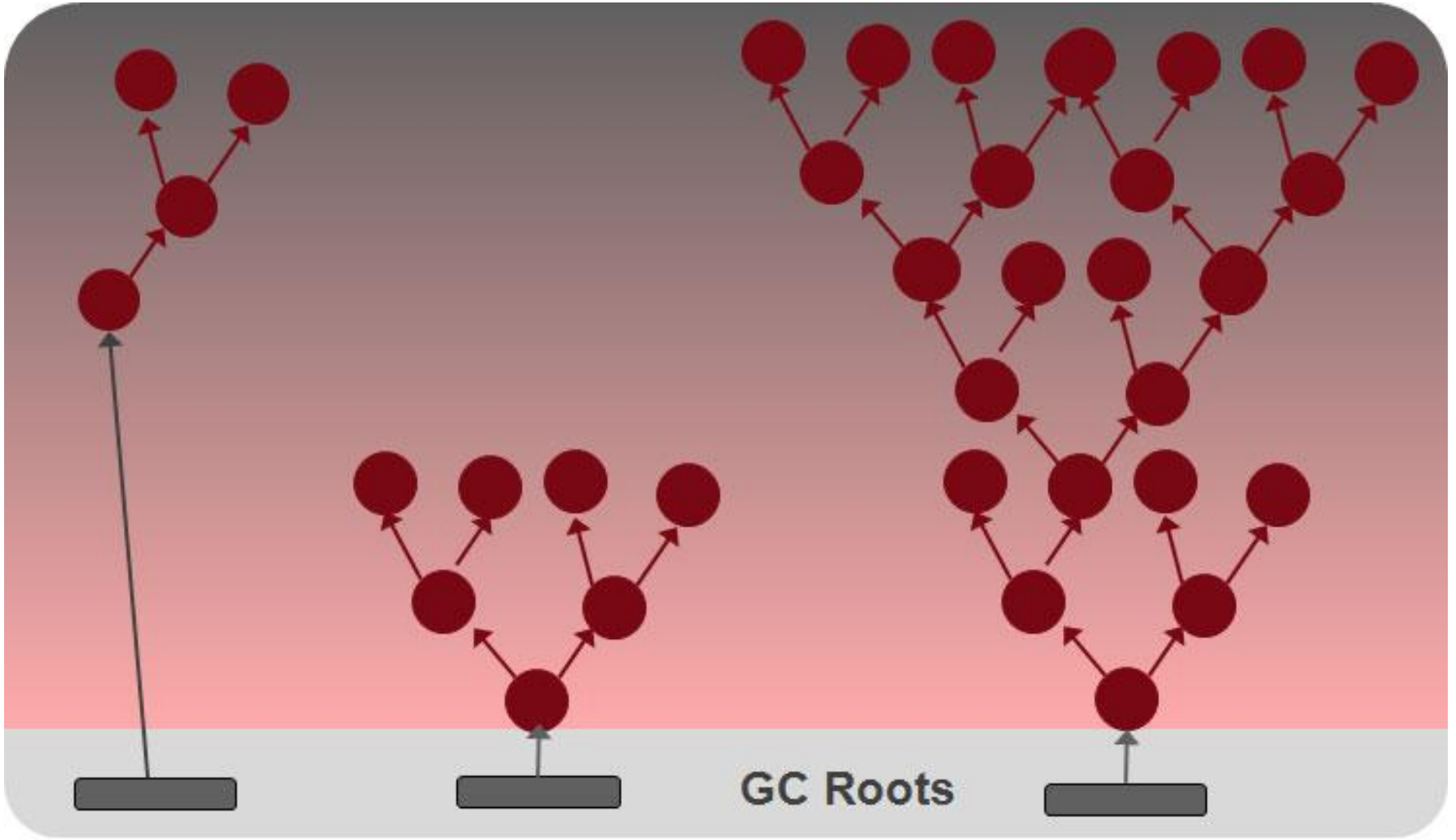
An object x **dominates** an object y if every path in the object graph from the start (or the root) node to y must go through x.

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

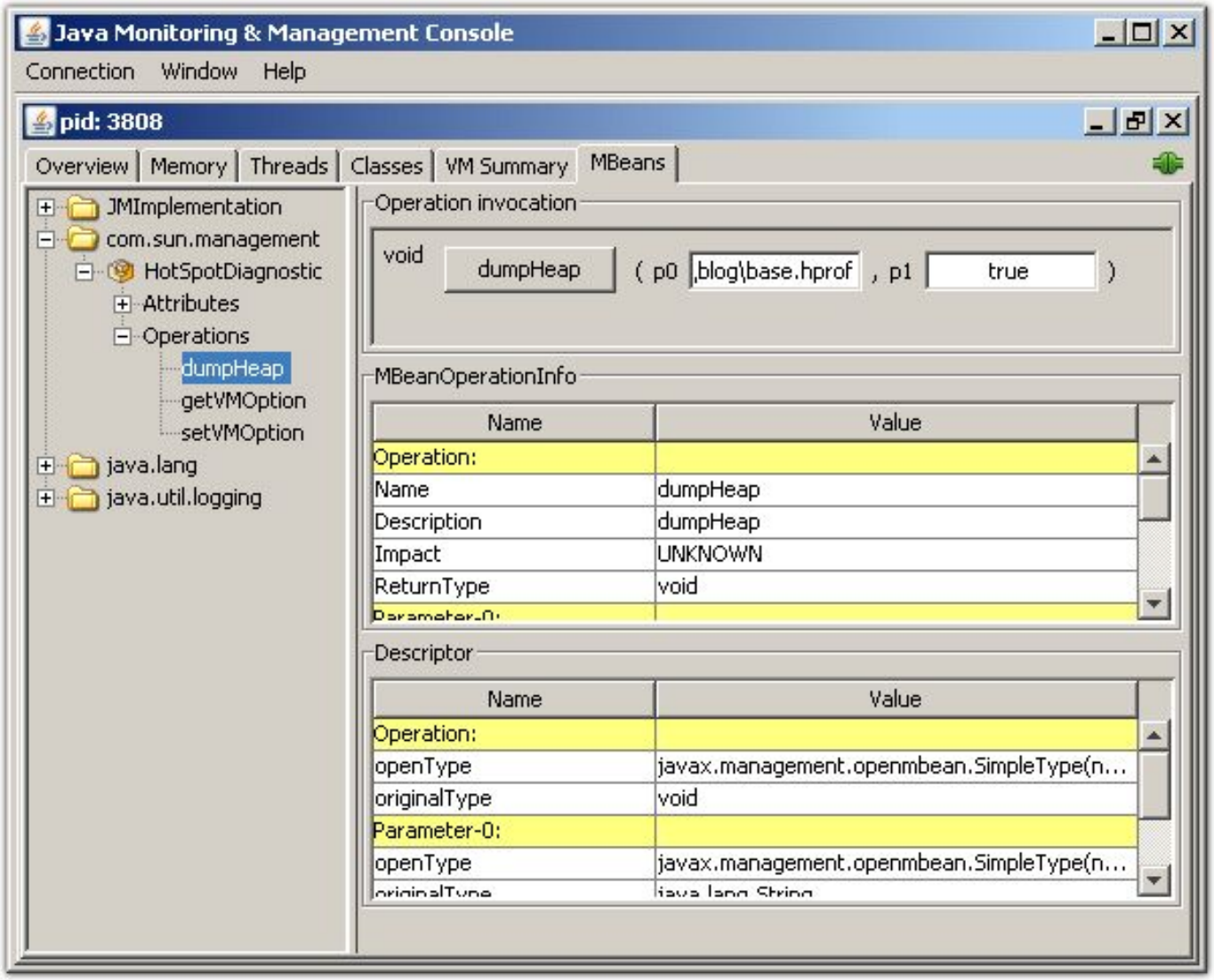
+

See Thread Dump By Sending a Signal to JVM

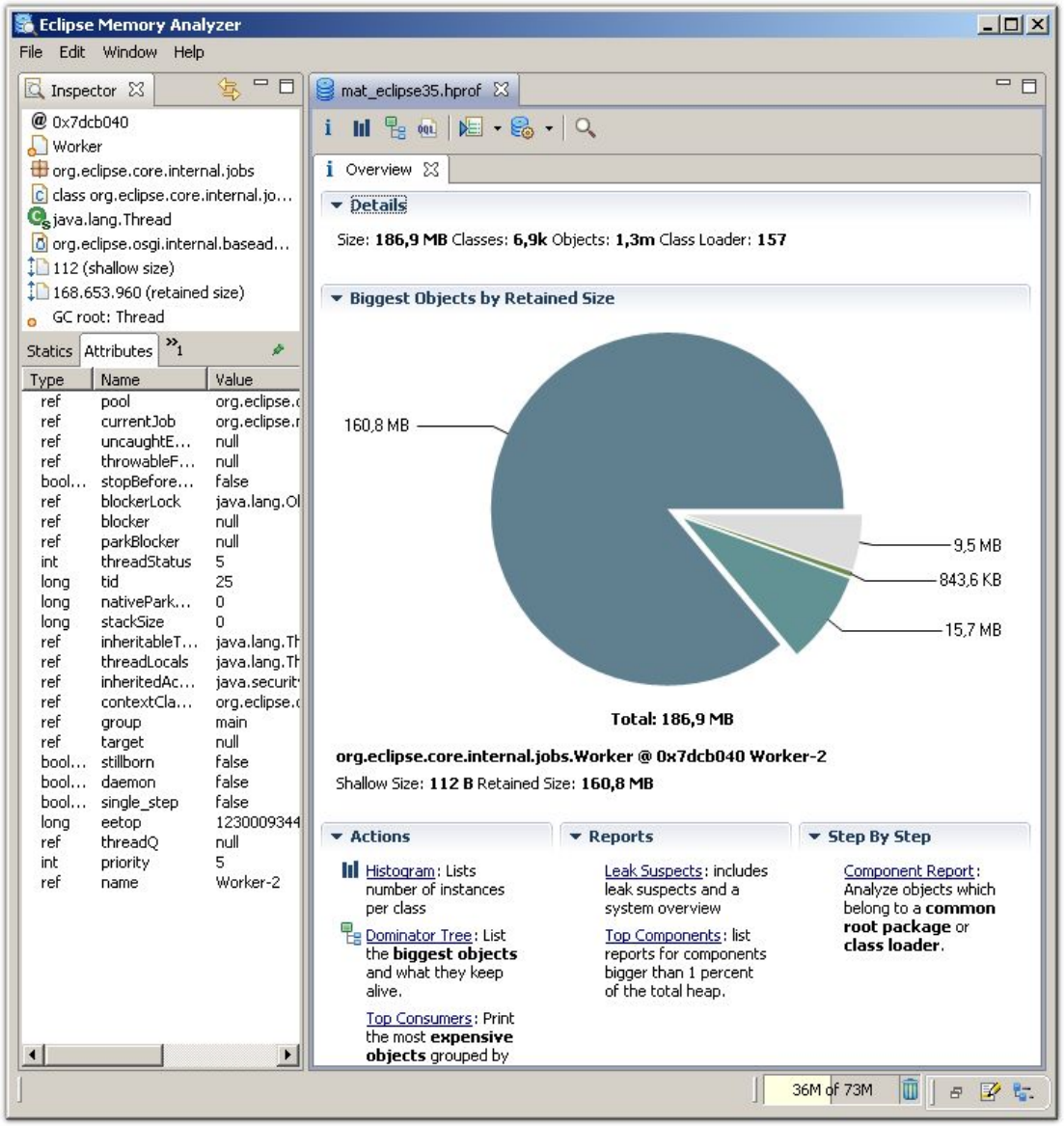
Heap Dump Using JDK 5/6 tools

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

Heap Dump Using JConsole

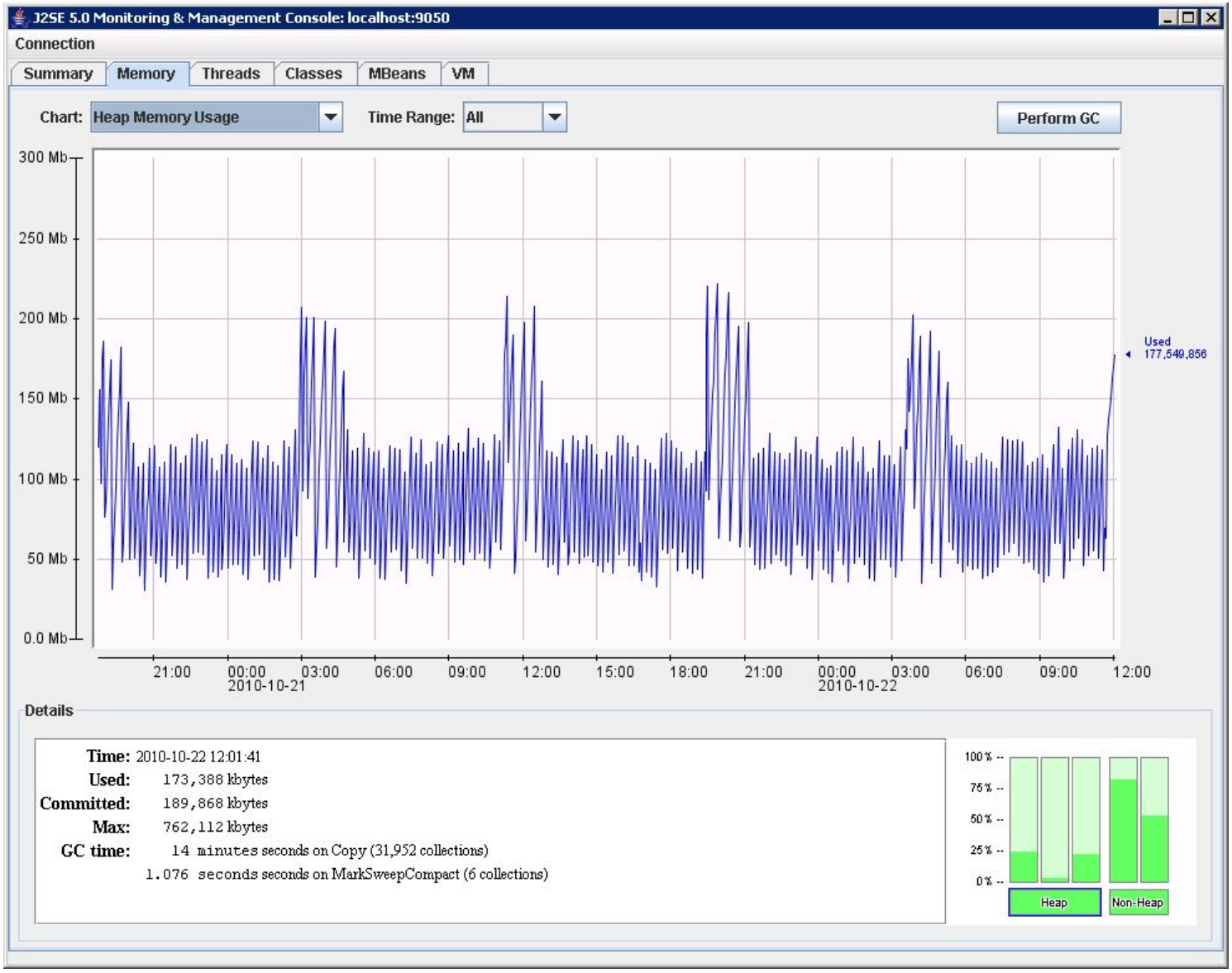


Eclipse Memory Analyser

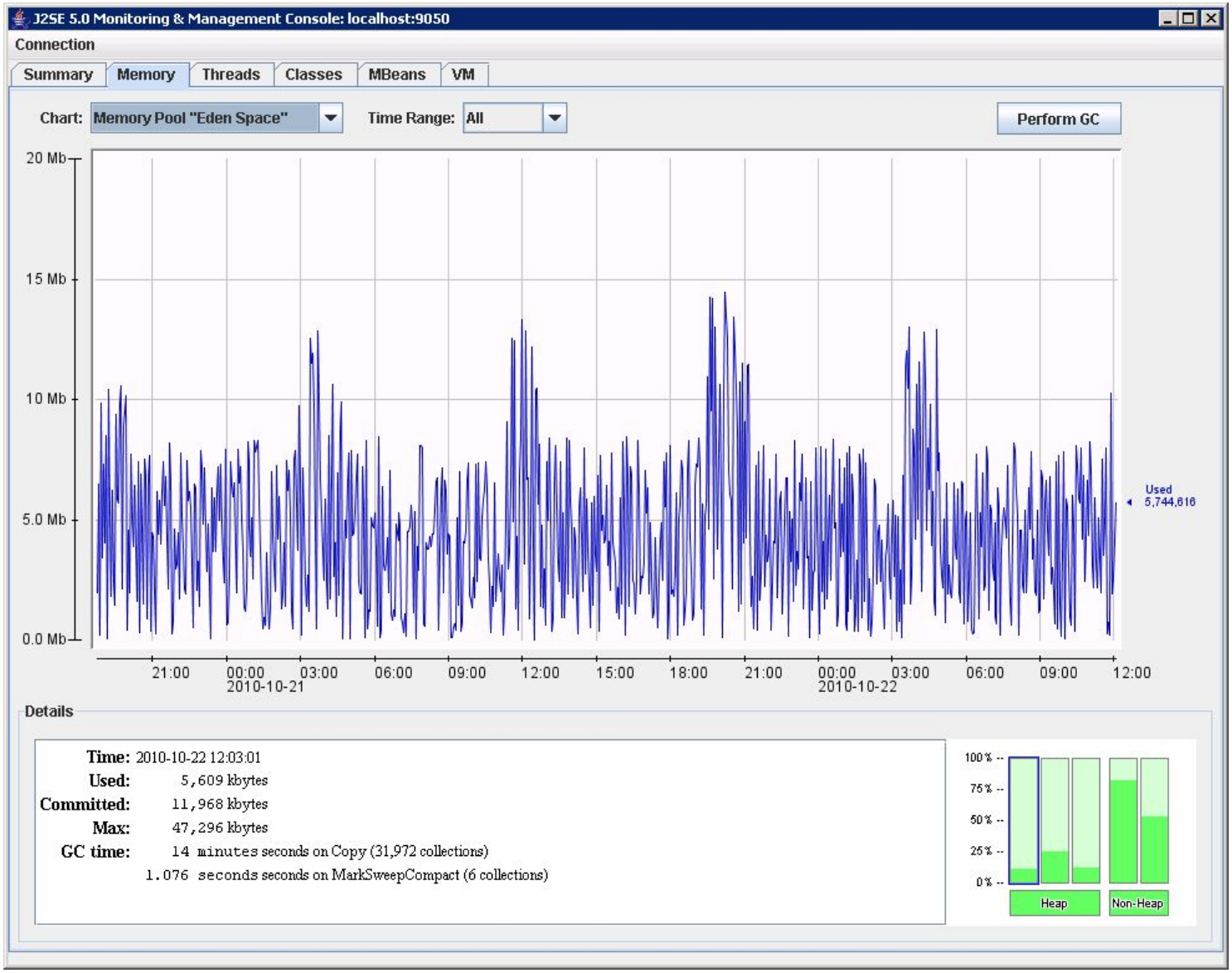


Example 3: Memory Leak

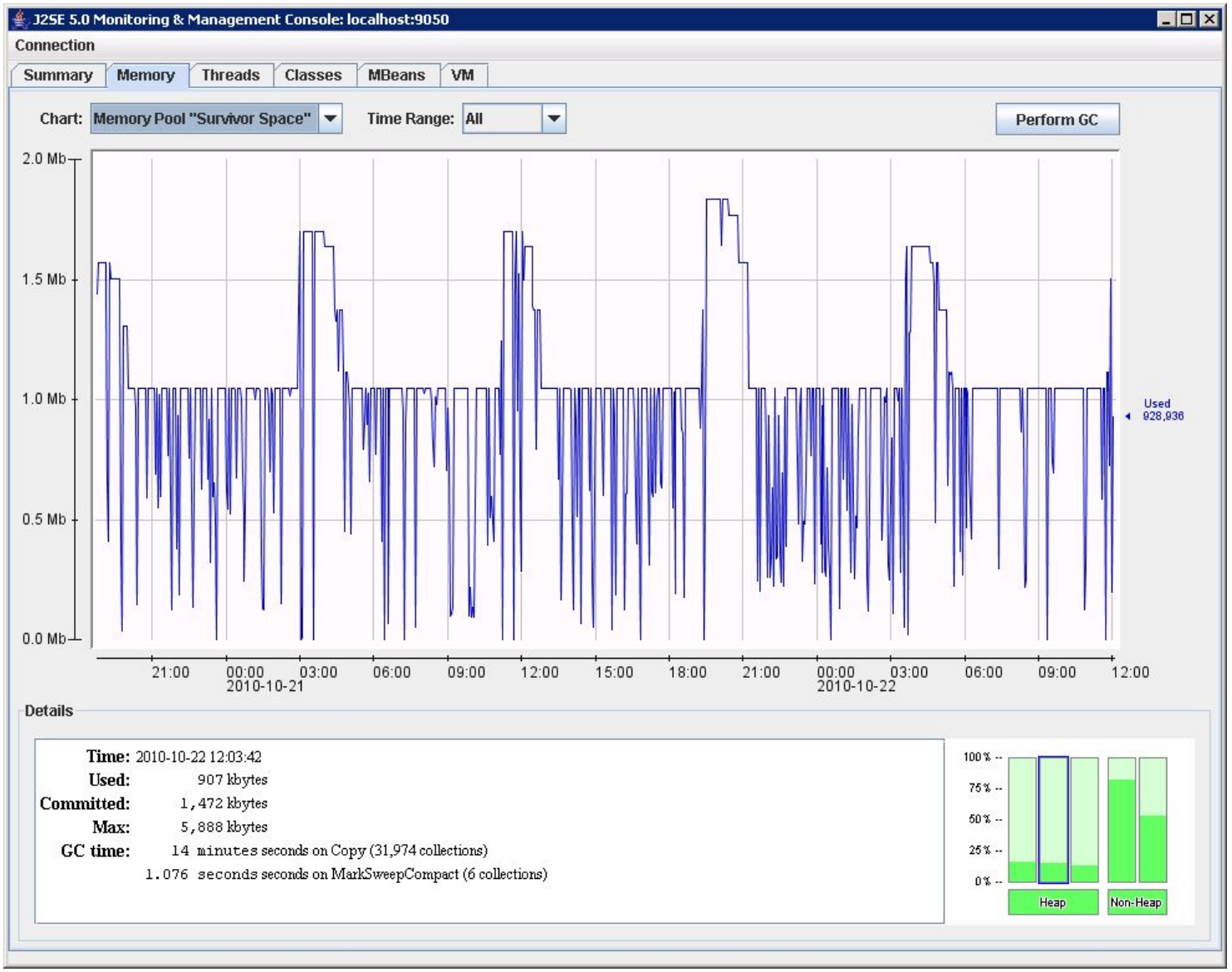
Heap Memory Usage



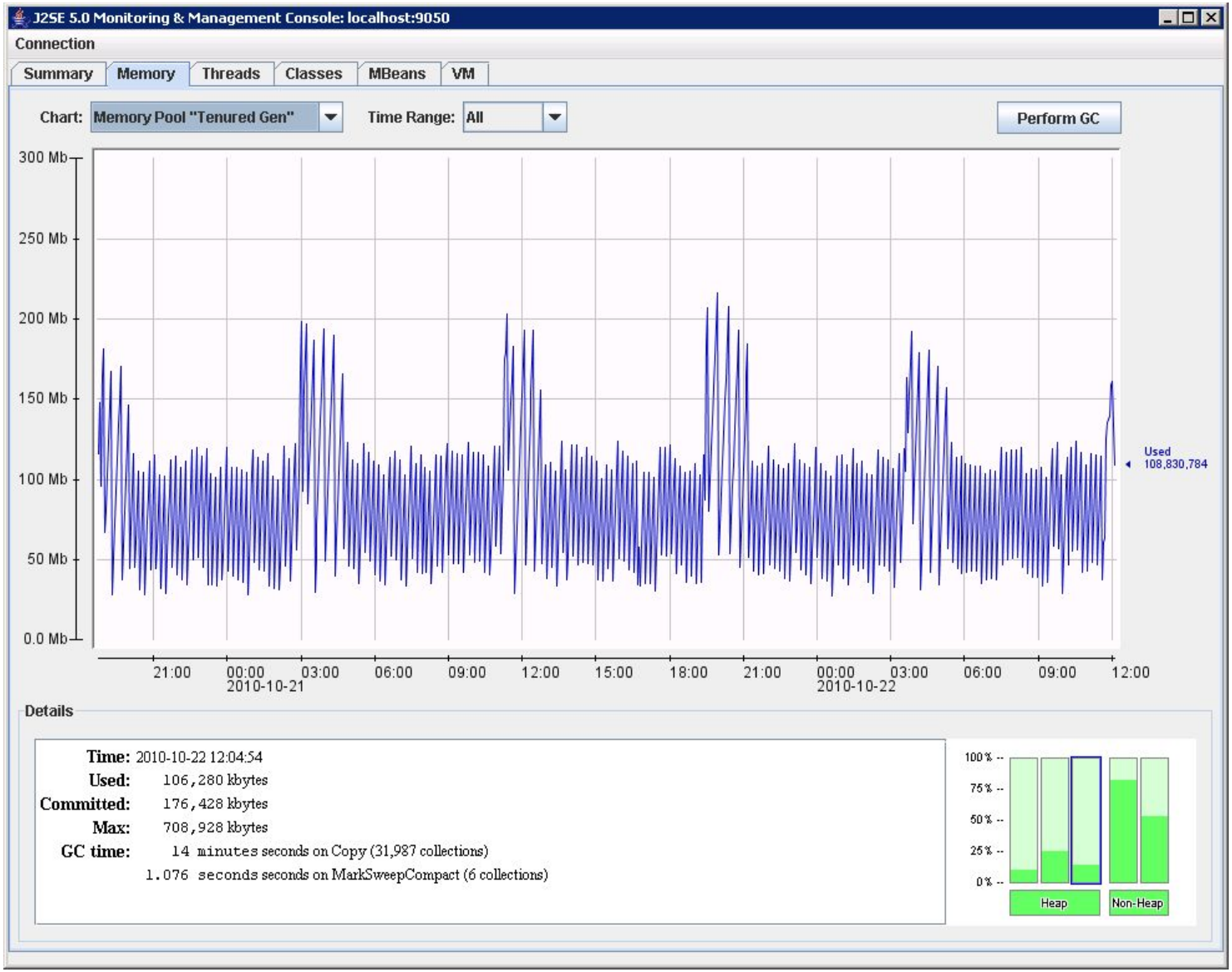
Memory Pool “Eden Space”



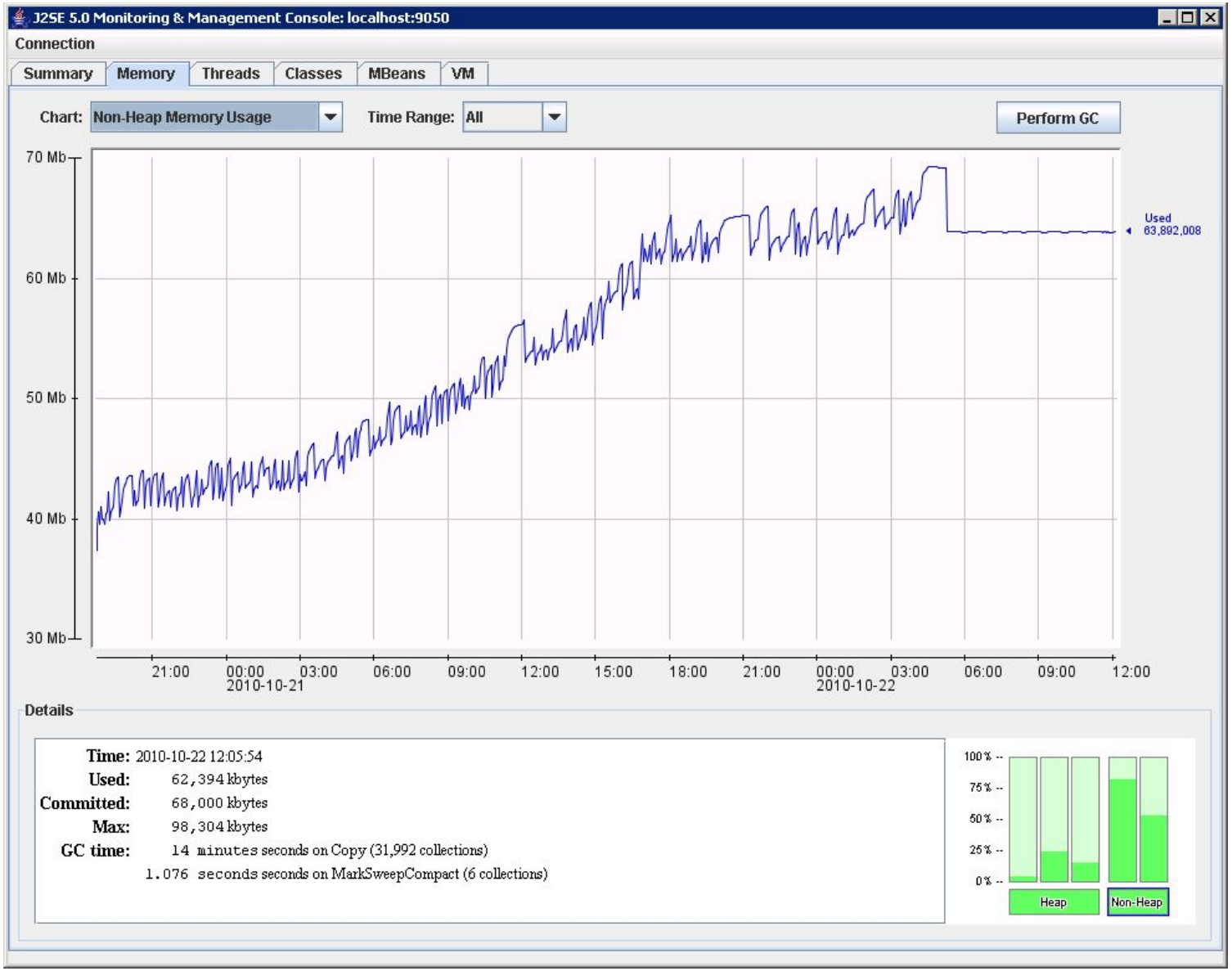
Memory Pool “Survivor Space”



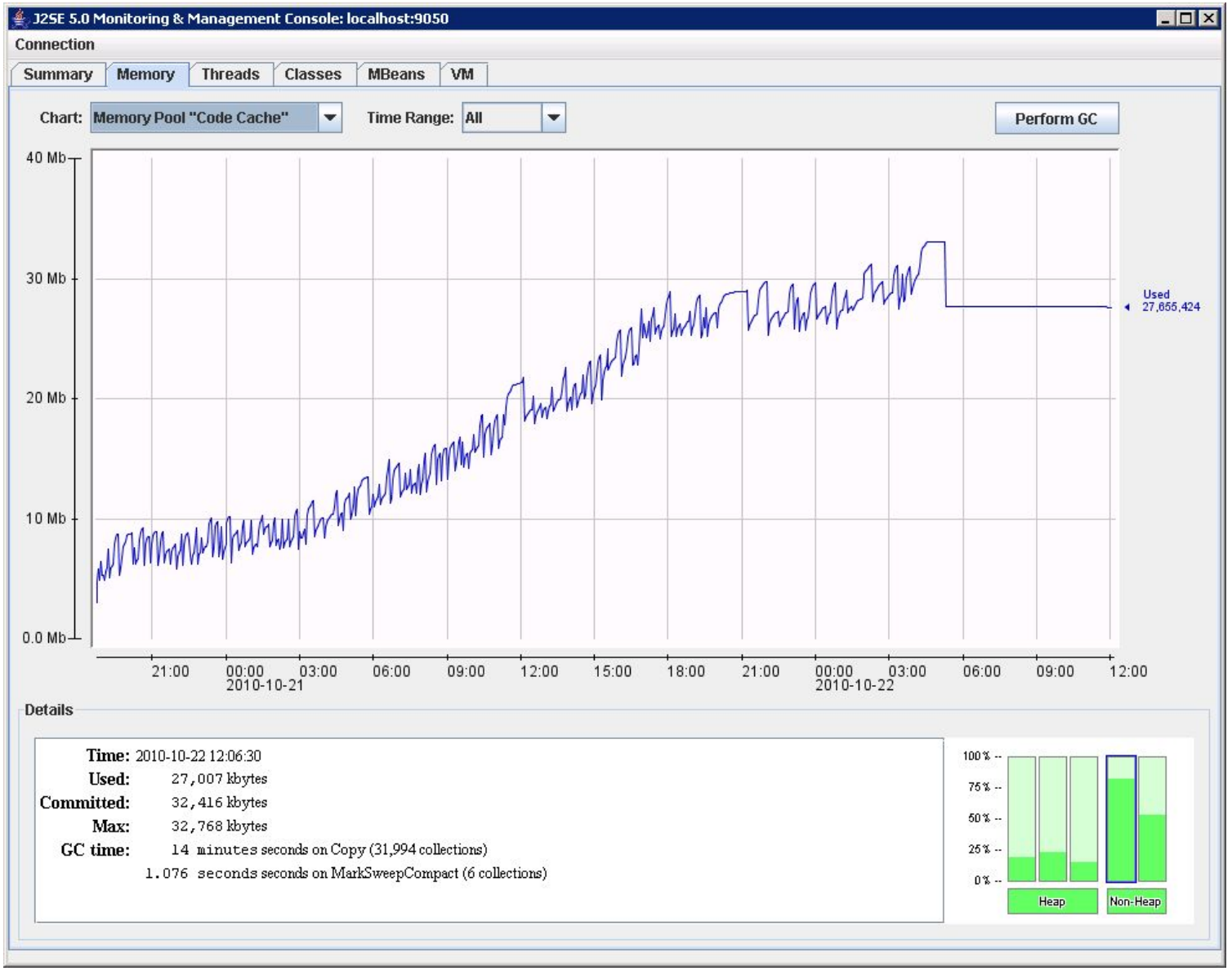
Memory Pool “Tenured Gen”



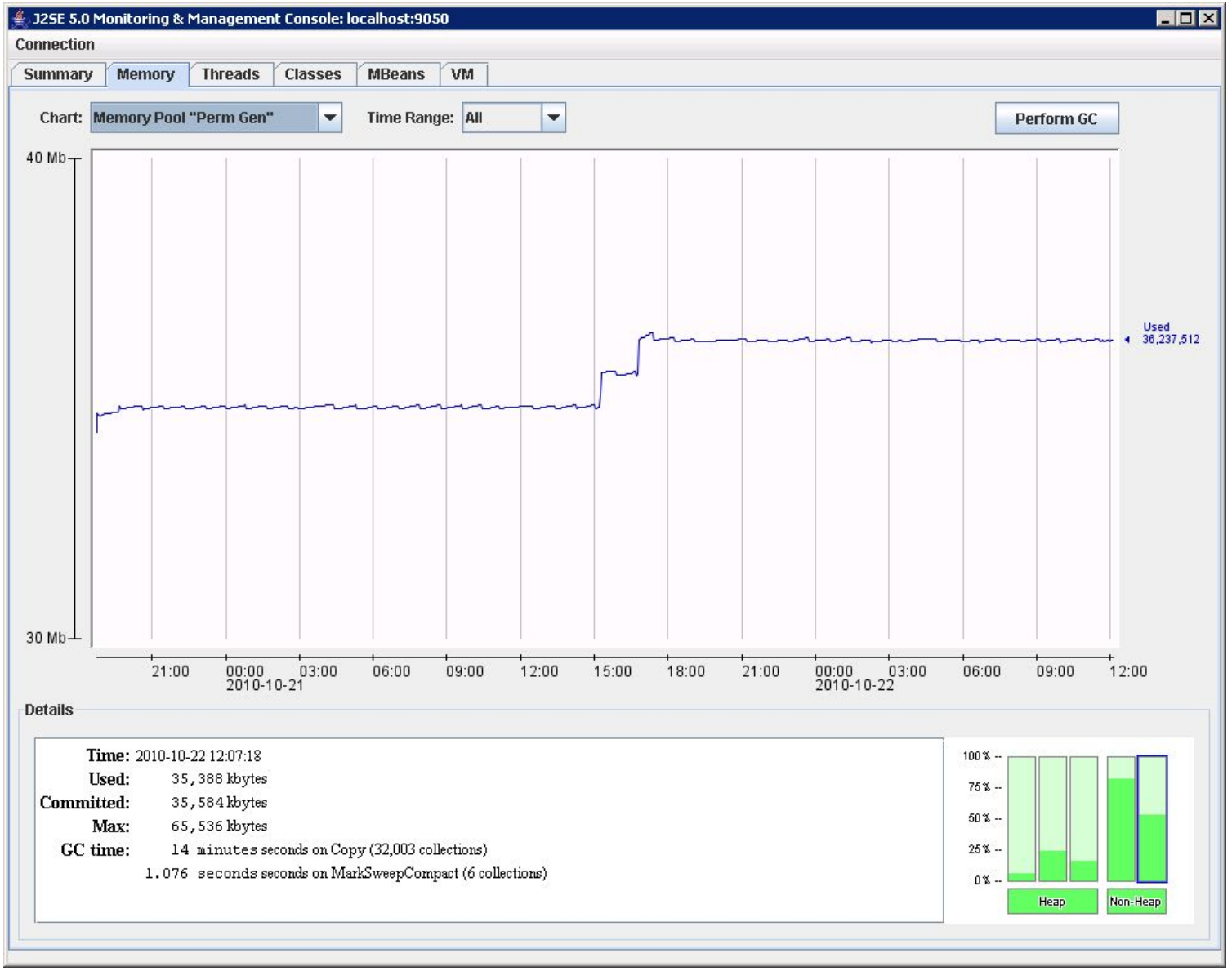
Non-Heap Memory Usage



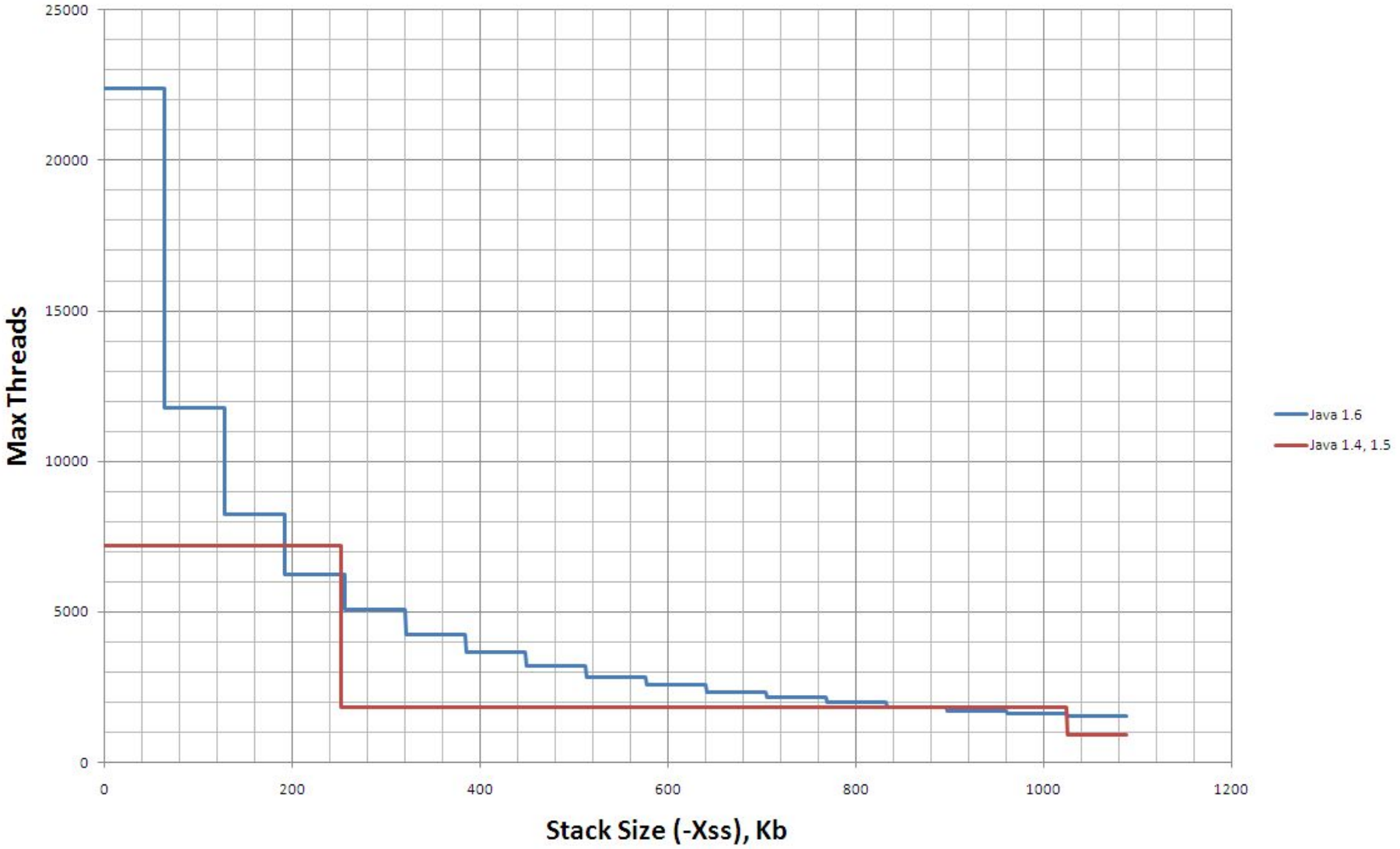
Memory Pool “Code Cache”



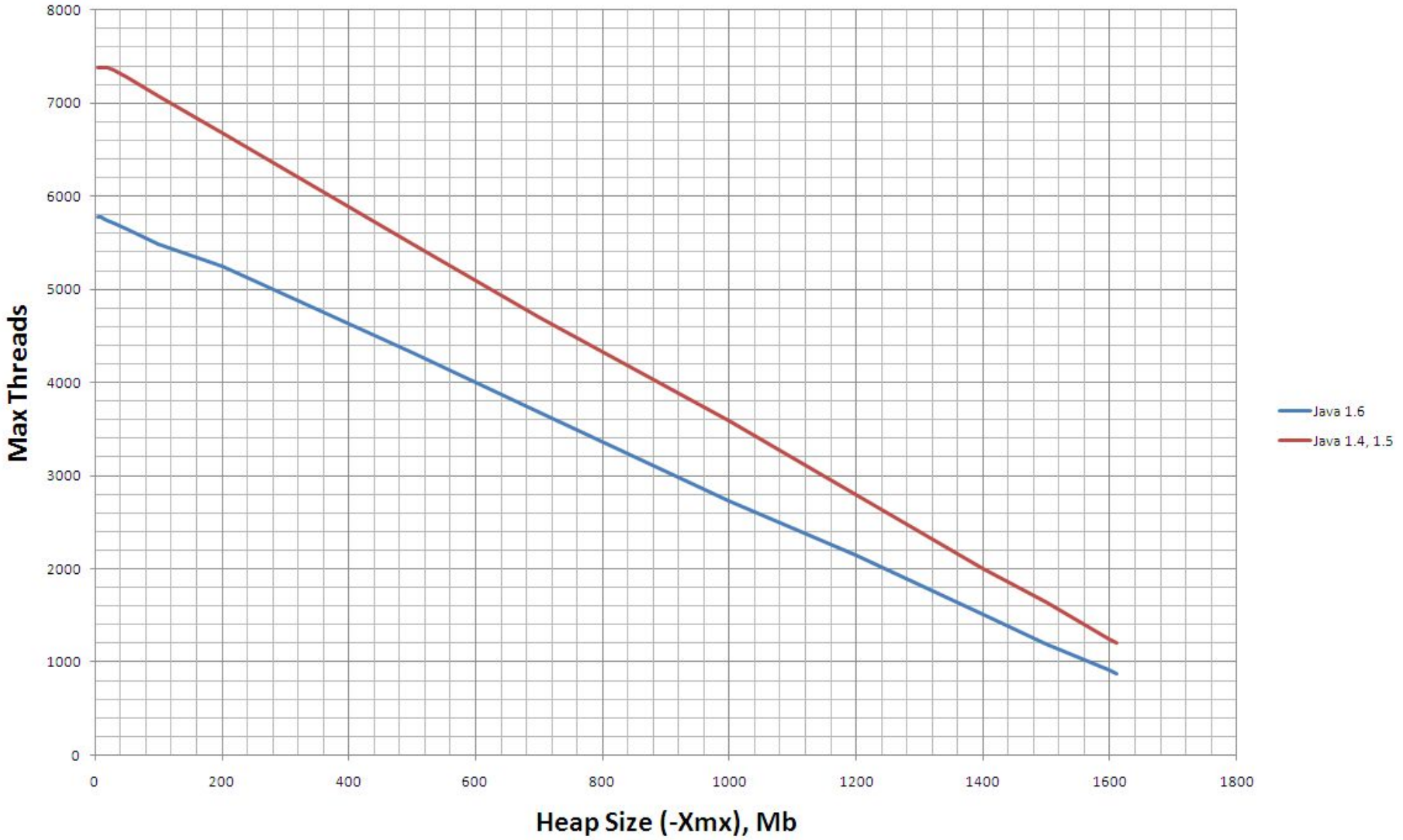
Memory Pool “Perm Gen”



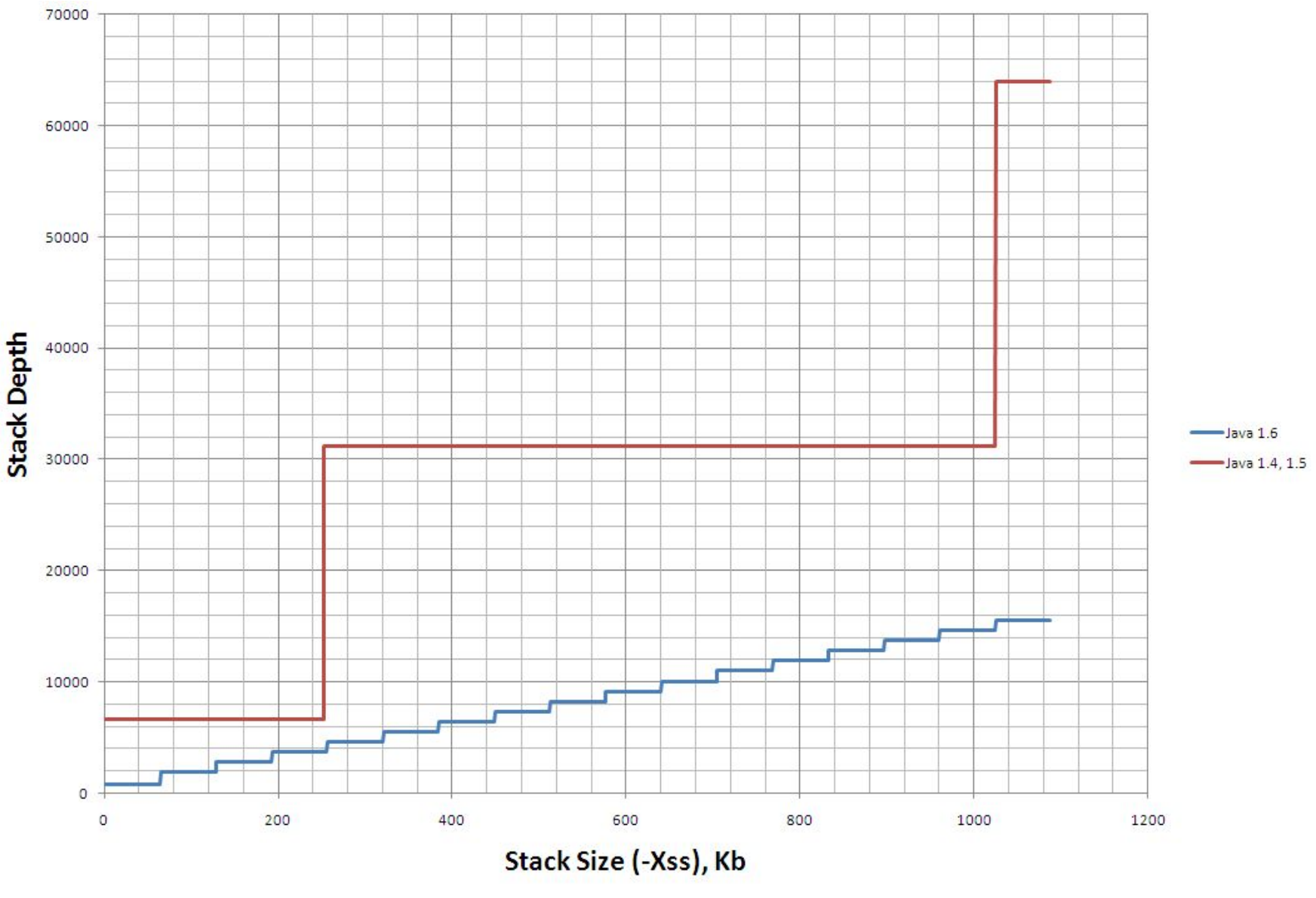
Max Thread Count Depends on Stack Size



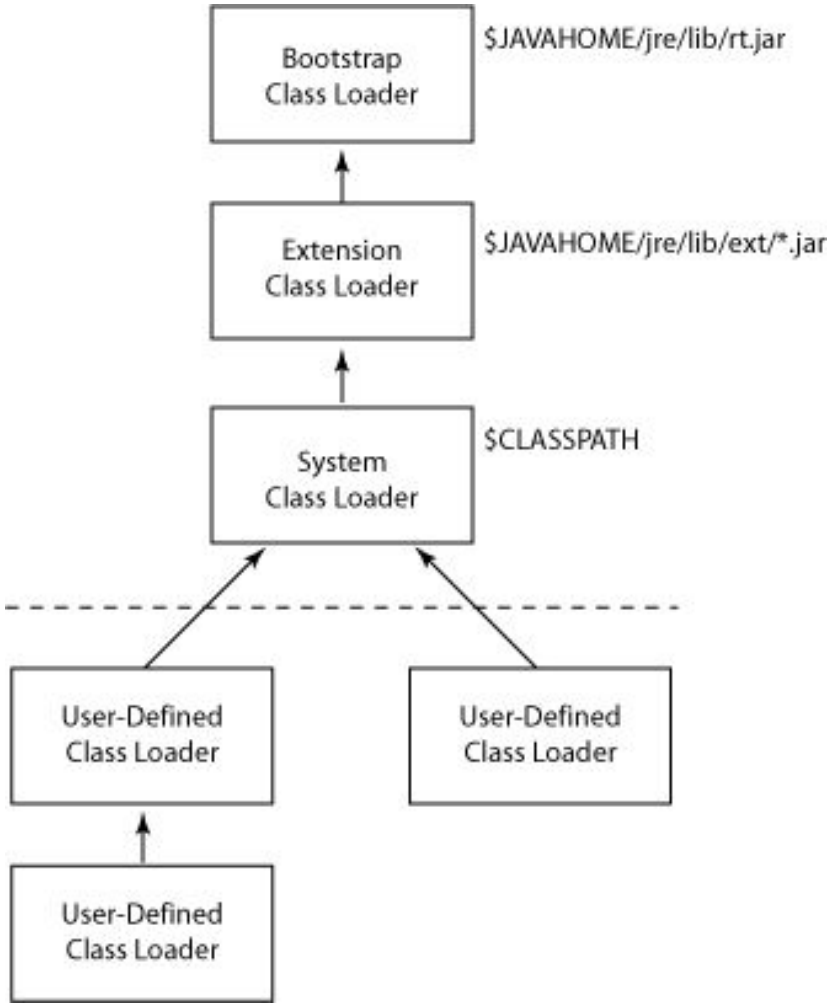
Max Thread Count Depends on Heap Size



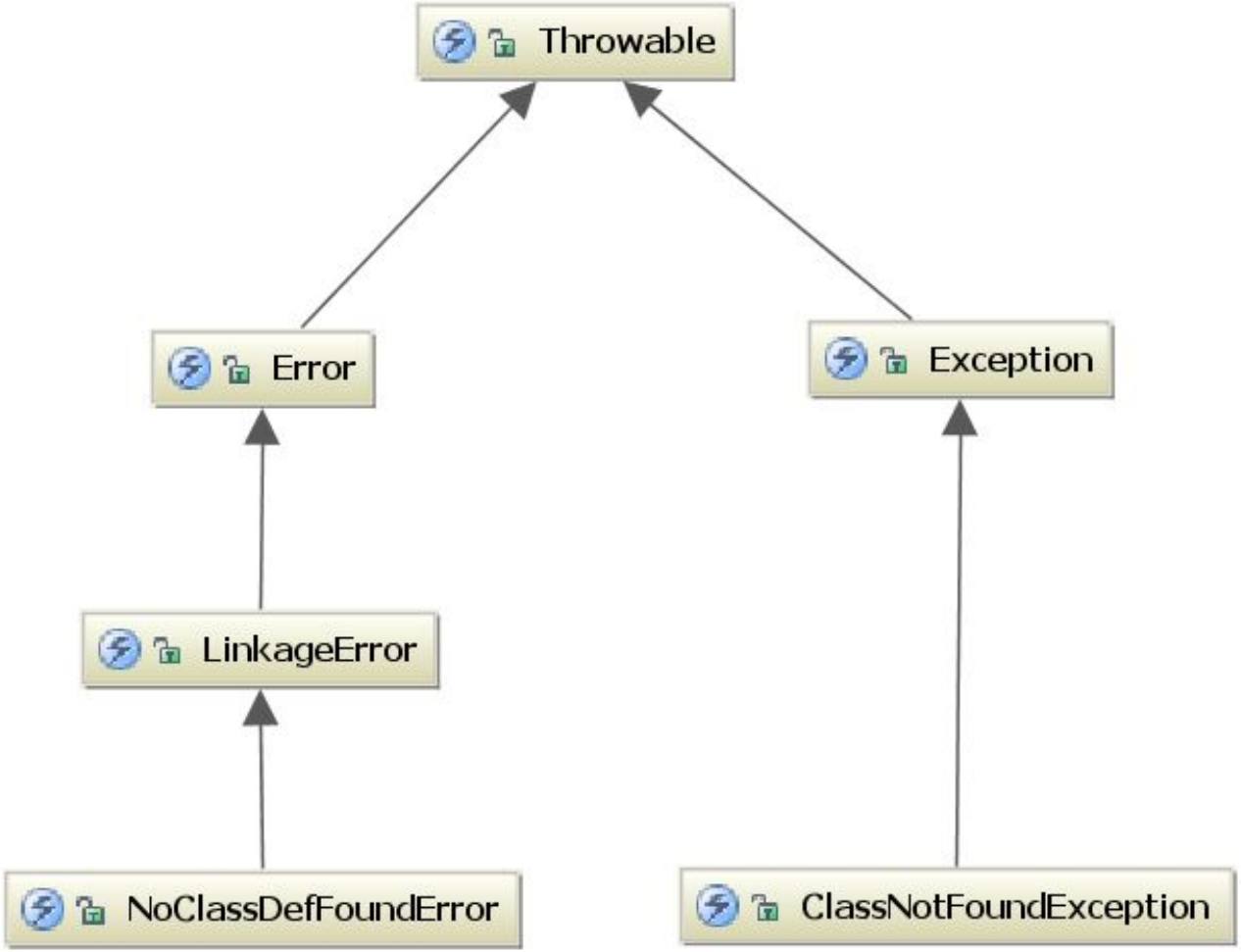
Stack Depth Depends on Stack Size



Java Class Loading



NoClassDefFoundError vs ClassNotFoundException



The End