Production Debugging @ 100mph

About Me

Co-founder – Takipi (God mode in Production Code).

Co-founder – VisualTao (acquired by Autodesk).

Director, AutoCAD Web & Mobile.

Software Architect at IAI Aerospace.

Coding for the past 16 years - C++, Delphi, .NET, Java.

Focus on real-time, scalable systems.

Blogs at takipiblog.com

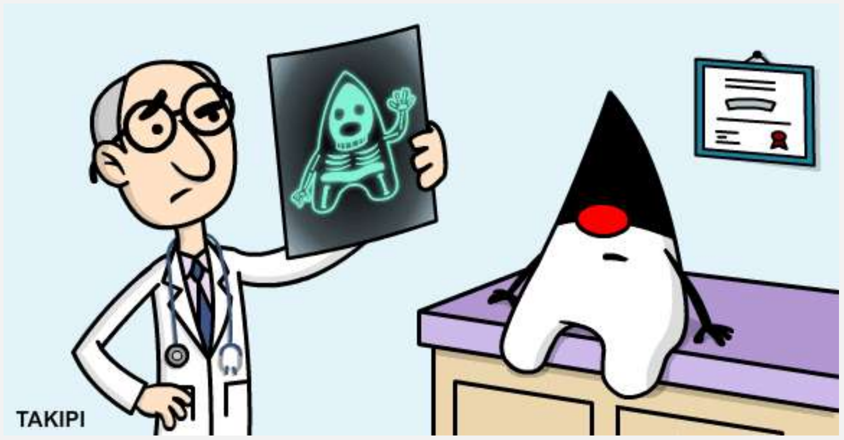
Overview

Dev-stage debugging is forward-tracing.

Production debugging is focused on backtracing.

Modern production debugging poses two challenges: state **isolation** and data **distribution**.

Direct correlation between quality of data to MTTR.



Agenda

1. Distributed logging – best practices.

1. Preemptive jstacks

2. Java 8 – state of the stack

3. Inspecting state with Btrace

1. Extracting state with custom Java agents.

Solid Logging Practices

Make sure these are baked into your logging context –

1. Code context.
2. Time + duration.
3. Thread ID (preferably name).
4. Transaction ID (for async & distributed debugging).

Transaction ID

- Logging is usually a multi–threaded / process affair.
- Generate a UUID at every thread entry point into your app – the transaction ID.
- Append the ID into each log entry.
- Try to maintain it across machines – critical for **distributed / async debugging**.

Thread Names

- Thread *name* is a mutable property.
- Can be set to hold transaction specific state.
- Some frameworks (e.g. EJB) don't like that.
- Can be super helpful when debugging in tandem with **jstack**.

Thread Names (2)

- Transaction ID
- Servlet parameters, Queue message ID
- Start time

Thread.currentThread().setName(Context, TID, Params, Time,..)

```
"pool-1-thread-1" #17 prio=5 os_prio=31 tid=0x00007f9d620c9800 nid=0x6d03
in Object.wait() [0x000000013ebcc000]

”MsgID: AB5CAD, type: Analyze, queue: ACTIVE_PROD, TID: 5678956, TS:
11/8/20014 18:34  "
#17 prio=5 os_prio=31 tid=0x00007f9d620c9800 nid=0x6d03 in Object.wait()
[0x000000013ebcc000]
```

Global Exception Handlers

Your **last line of defense** - critical to pick up on unhandled exceptions.

Setting the callback:

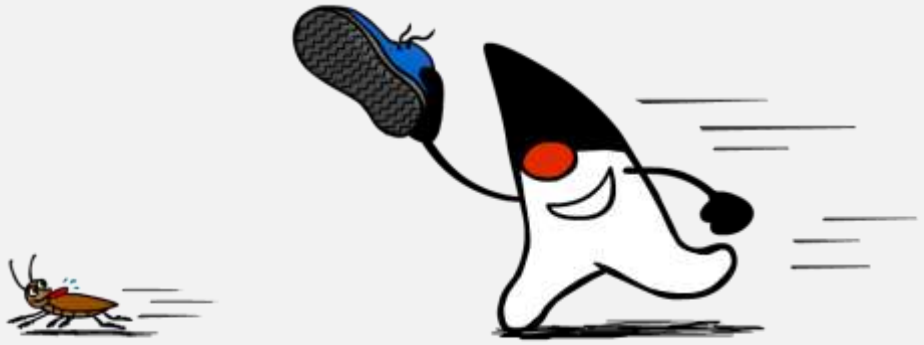
```
public static void Thread.setDefaultUncaughtExceptionHandler(UncaughtExceptionHandler eh)
```

```
void UncaughtExceptionHandler.uncaughtException(Thread t, Throwable e) {
        logger.error("Uncaught error in thread " + t, e);
}
```

This is where thread **Name + TLS** are critical as the only surviving state.

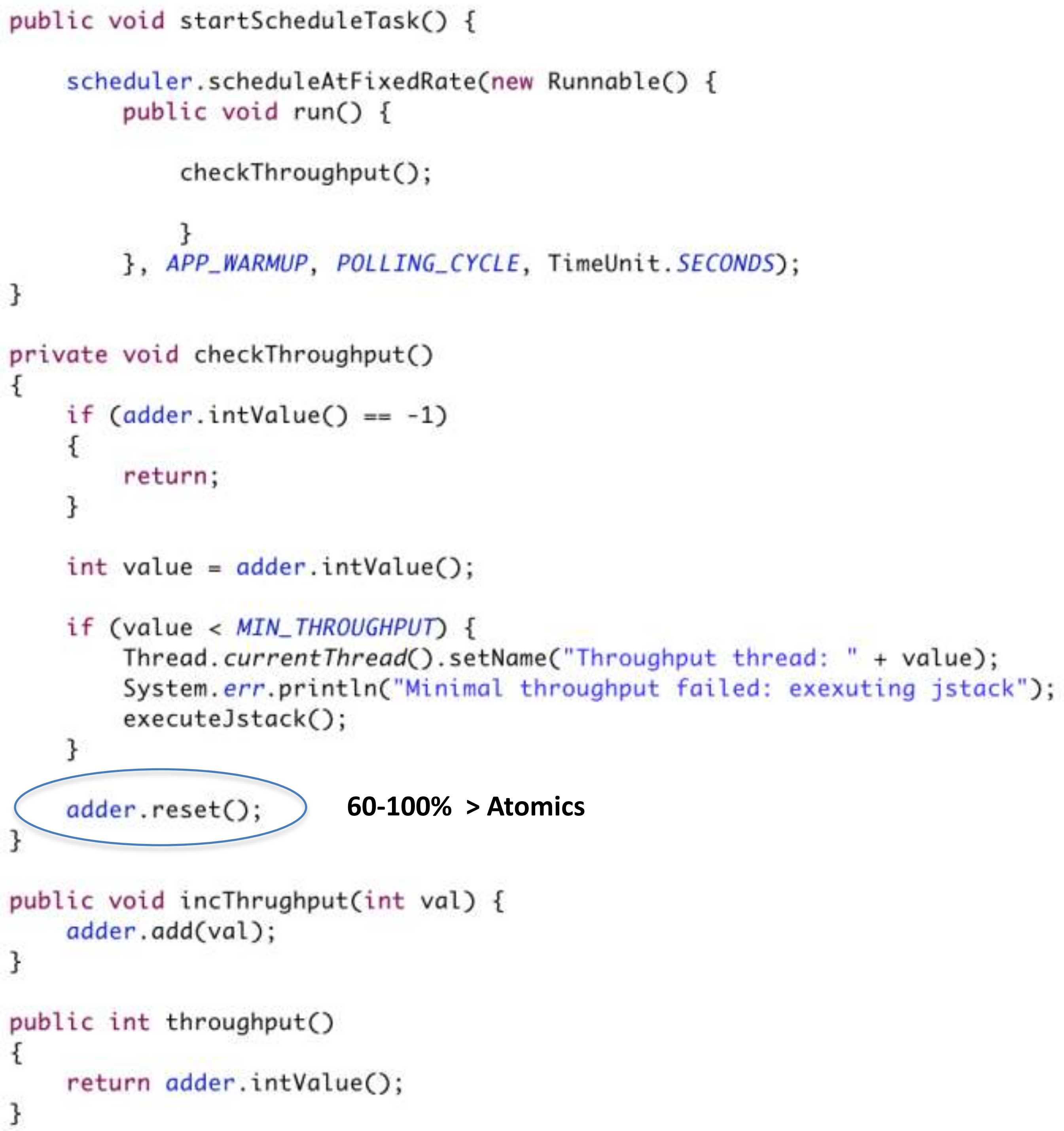
Preemptive jstack

- A production debugging foundation.
- Presents two issues –
  - Activated only in retrospect.
  - **No state:** does not provide any variable state.
- Let's see how we can overcome these with preemptive jstacks.



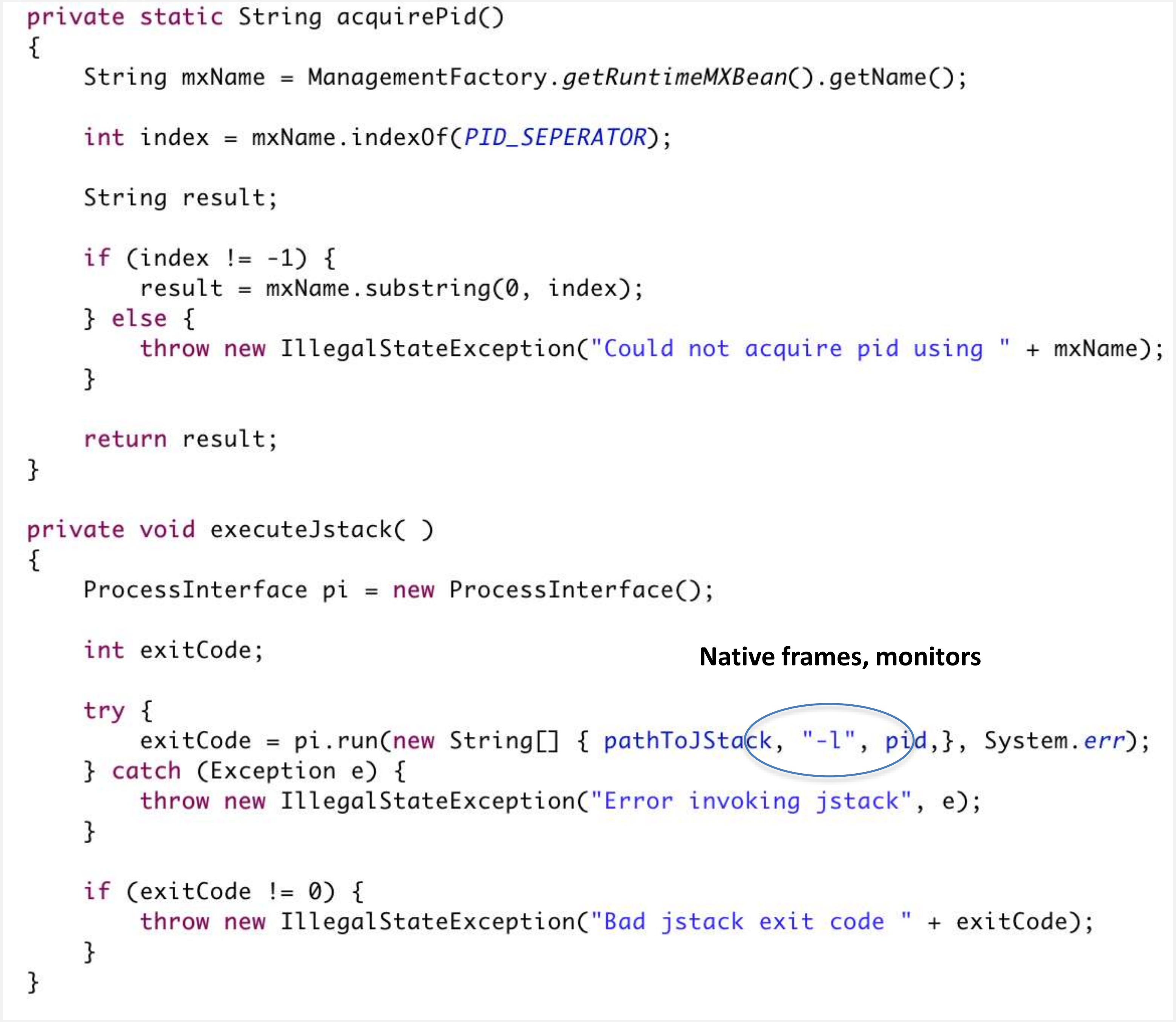
Preemptive jstack - Demo

github.com/takipi/jstack



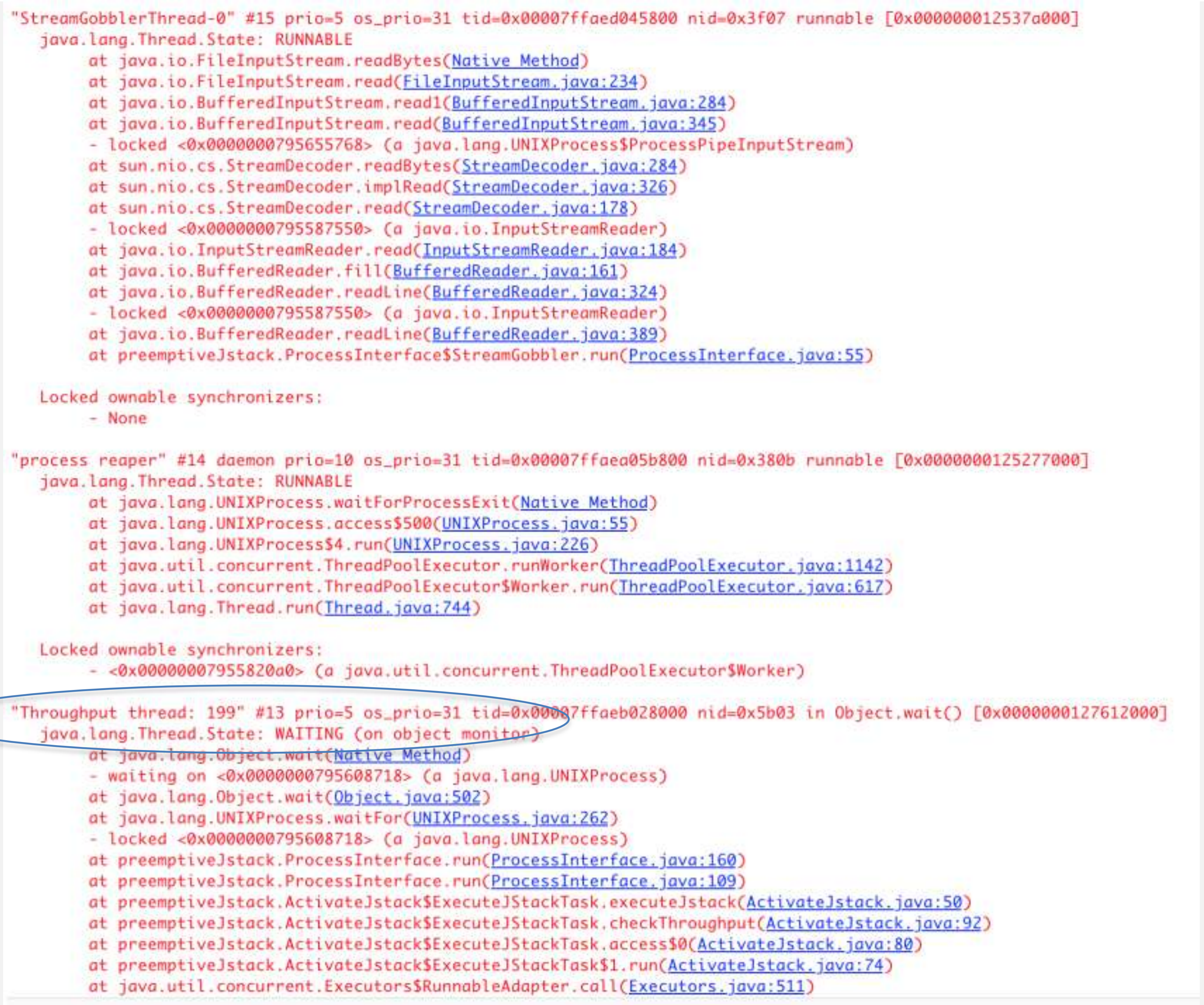
```
public void startScheduleTask() {

    scheduler.scheduleAtFixedRate(new Runnable() {
        public void run() {

            checkThroughput();

            }
        }, APP_WARMUP, POLLING_CYCLE, TimeUnit.SECONDS);
}

private void checkThroughput()
{
    if (adder.intValue() == -1)
    {
        return;
    }

    int value = adder.intValue();

    if (value < MIN_THROUGHPUT) {
        Thread.currentThread().setName("Throughput thread: " + value);
        System.err.println("Minimal throughput failed: exexuting jstack");
        executeJstack();
    }

    adder.reset();
}

public void incThrughput(int val) {
    adder.add(val);
}

public int throughput()
{
    return adder.intValue();
}
```

60-100% > Atomics



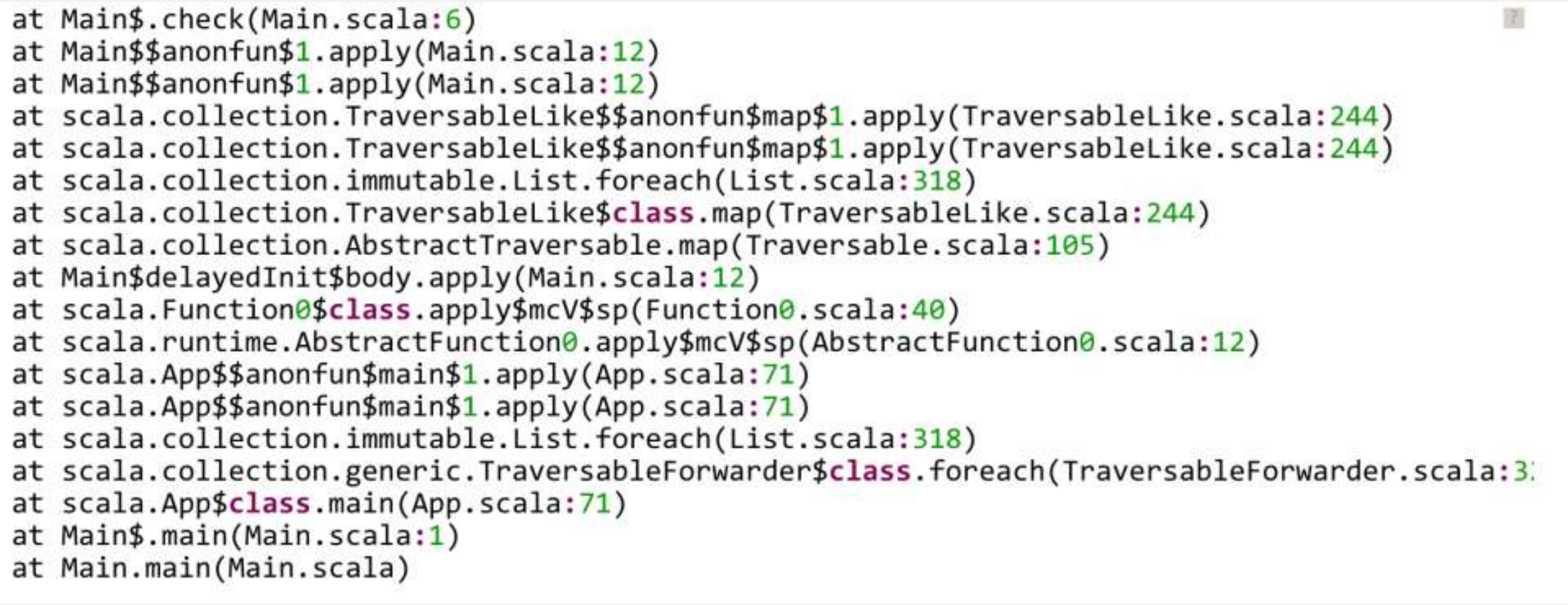
```java
private static String acquirePid()
{
    String mxName = ManagementFactory.getRuntimeMXBean().getName();

    int index = mxName.indexOf(PID_SEPERATOR);

    String result;

    if (index != -1) {
        result = mxName.substring(0, index);
    } else {
        throw new IllegalStateException("Could not acquire pid using " + mxName);
    }

    return result;
}

private void executeJstack( )
{
    ProcessInterface pi = new ProcessInterface();

    int exitCode;

    try {
        exitCode = pi.run(new String[] { pathToJStack, "-l", pid,}, System.err);
    } catch (Exception e) {
        throw new IllegalStateException("Error invoking jstack", e);
    }

    if (exitCode != 0) {
        throw new IllegalStateException("Bad jstack exit code " + exitCode);
    }
}
```

Native frames, monitors


```
"StreamGobblerThread-0" #15 prio=5 os_prio=31 tid=0x00007ffaed045800 nid=0x3f07 runnable [0x000000012537a000]
   java.lang.Thread.State: RUNNABLE
        at java.io.FileInputStream.readBytes(Native Method)
        at java.io.FileInputStream.read(FileInputStream.java:234)
        at java.io.BufferedInputStream.read1(BufferedInputStream.java:284)
        at java.io.BufferedInputStream.read(BufferedInputStream.java:345)
        - locked <0x0000000795655768> (a java.lang.UNIXProcess$ProcessPipeInputStream)
        at sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:284)
        at sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:326)
        at sun.nio.cs.StreamDecoder.read(StreamDecoder.java:178)
        - locked <0x0000000795587550> (a java.io.InputStreamReader)
        at java.io.InputStreamReader.read(InputStreamReader.java:184)
        at java.io.BufferedReader.fill(BufferedReader.java:161)
        at java.io.BufferedReader.readLine(BufferedReader.java:324)
        - locked <0x0000000795587550> (a java.io.InputStreamReader)
        at java.io.BufferedReader.readLine(BufferedReader.java:389)
        at preemptiveJstack.ProcessInterface$StreamGobbler.run(ProcessInterface.java:55)

   Locked ownable synchronizers:
        - None

"process reaper" #14 daemon prio=10 os_prio=31 tid=0x00007ffaea05b800 nid=0x380b runnable [0x0000000125277000]
   java.lang.Thread.State: RUNNABLE
        at java.lang.UNIXProcess.waitForProcessExit(Native Method)
        at java.lang.UNIXProcess.access$500(UNIXProcess.java:55)
        at java.lang.UNIXProcess$4.run(UNIXProcess.java:226)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:744)

   Locked ownable synchronizers:
        - <0x00000007955820a0> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"Throughput thread: 199" #13 prio=5 os_prio=31 tid=0x00007ffaeb028000 nid=0x5b03 in Object.wait() [0x0000000127612000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x0000000795608718> (a java.lang.UNIXProcess)
        at java.lang.Object.wait(Object.java:502)
        at java.lang.UNIXProcess.waitFor(UNIXProcess.java:262)
        - locked <0x0000000795608718> (a java.lang.UNIXProcess)
        at preemptiveJstack.ProcessInterface.run(ProcessInterface.java:160)
        at preemptiveJstack.ProcessInterface.run(ProcessInterface.java:109)
        at preemptiveJstack.ActivateJstack$ExecuteJStackTask.executeJstack(ActivateJstack.java:50)
        at preemptiveJstack.ActivateJstack$ExecuteJStackTask.checkThroughput(ActivateJstack.java:92)
        at preemptiveJstack.ActivateJstack$ExecuteJStackTask.access$0(ActivateJstack.java:80)
        at preemptiveJstack.ActivateJstack$ExecuteJStackTask$1.run(ActivateJstack.java:74)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
```

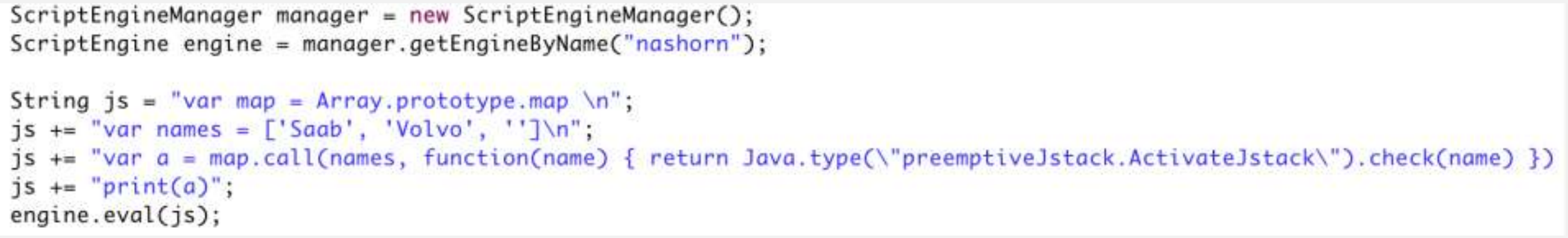
Java 8 stack traces

```
val lengths = names.map(name => check(name.length))
```

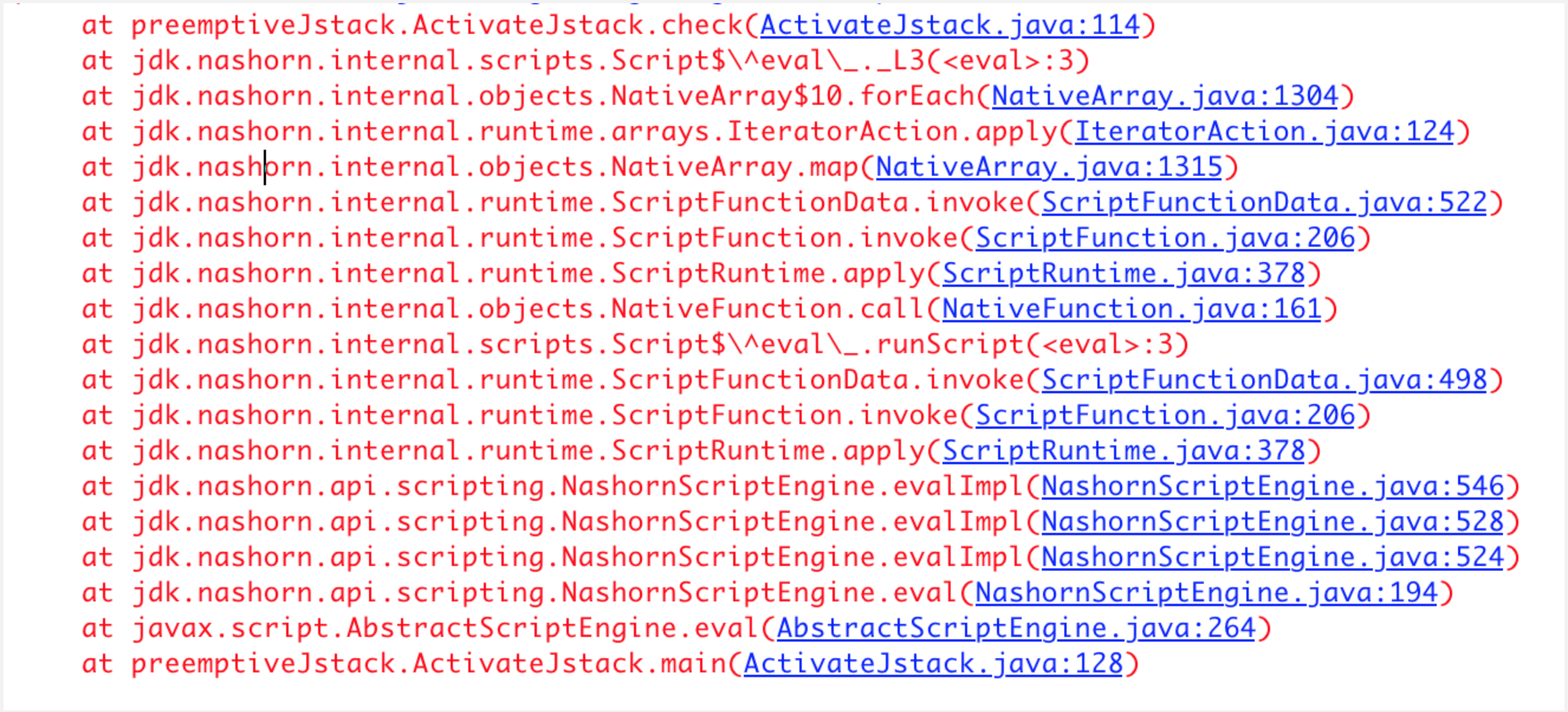


```
at Main$.check(Main.scala:6)
at Main$$anonfun$1.apply(Main.scala:12)
at Main$$anonfun$1.apply(Main.scala:12)
at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:244)
at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:244)
at scala.collection.immutable.List.foreach(List.scala:318)
at scala.collection.TraversableLike$class.map(TraversableLike.scala:244)
at scala.collection.AbstractTraversable.map(Traversable.scala:105)
at Main$delayedInit$body.apply(Main.scala:12)
at scala.Function0$class.apply$mcV$sp(Function0.scala:40)
at scala.runtime.AbstractFunction0.apply$mcV$sp(AbstractFunction0.scala:12)
at scala.App$$anonfun$main$1.apply(App.scala:71)
at scala.App$$anonfun$main$1.apply(App.scala:71)
at scala.collection.immutable.List.foreach(List.scala:318)
at scala.collection.generic.TraversableForwarder$class.foreach(TraversableForwarder.scala:3
at scala.App$class.main(App.scala:71)
at Main$.main(Main.scala:1)
at Main.main(Main.scala)
```

```
Stream lengths = names.stream().map(name -> check(name));

at LmbdaMain.check(LmbdaMain.java:19)
at LmbdaMain.lambda$0(LmbdaMain.java:37)
at LmbdaMain$$Lambda$1/821270929.apply(Unknown Source)
at java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:193)
at java.util.Spliterators$ArraySpliterator.forEachRemaining(Spliterators.java:948)
at java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:512)
at java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:502)
at java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:708)
at java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)
at java.util.stream.LongPipeline.reduce(LongPipeline.java:438)
at java.util.stream.LongPipeline.sum(LongPipeline.java:396)
at java.util.stream.ReferencePipeline.count(ReferencePipeline.java:526)
at LmbdaMain.main(LmbdaMain.java:39)
```



```
ScriptEngineManager manager = new ScriptEngineManager();
ScriptEngine engine = manager.getEngineByName("nashorn");

String js = "var map = Array.prototype.map \n";
js += "var names = ['Saab', 'Volvo', '']\n";
js += "var a = map.call(names, function(name) { return Java.type(\"preemptiveJstack.ActivateJstack\").check(name) })
js += "print(a)";
engine.eval(js);
```



```
    at preemptiveJstack.ActivateJstack.check(ActivateJstack.java:114)
    at jdk.nashorn.internal.scripts.Script$\^eval\_._L3(<eval>:3)
    at jdk.nashorn.internal.objects.NativeArray$10.forEach(NativeArray.java:1304)
    at jdk.nashorn.internal.runtime.arrays.IteratorAction.apply(IteratorAction.java:124)
    at jdk.nashorn.internal.objects.NativeArray.map(NativeArray.java:1315)
    at jdk.nashorn.internal.runtime.ScriptFunctionData.invoke(ScriptFunctionData.java:522)
    at jdk.nashorn.internal.runtime.ScriptFunction.invoke(ScriptFunction.java:206)
    at jdk.nashorn.internal.runtime.ScriptRuntime.apply(ScriptRuntime.java:378)
    at jdk.nashorn.internal.objects.NativeFunction.call(NativeFunction.java:161)
    at jdk.nashorn.internal.scripts.Script$\^eval\_.runScript(<eval>:3)
    at jdk.nashorn.internal.runtime.ScriptFunctionData.invoke(ScriptFunctionData.java:498)
    at jdk.nashorn.internal.runtime.ScriptFunction.invoke(ScriptFunction.java:206)
    at jdk.nashorn.internal.runtime.ScriptRuntime.apply(ScriptRuntime.java:378)
    at jdk.nashorn.api.scripting.NashornScriptEngine.evalImpl(NashornScriptEngine.java:546)
    at jdk.nashorn.api.scripting.NashornScriptEngine.evalImpl(NashornScriptEngine.java:528)
    at jdk.nashorn.api.scripting.NashornScriptEngine.evalImpl(NashornScriptEngine.java:524)
    at jdk.nashorn.api.scripting.NashornScriptEngine.eval(NashornScriptEngine.java:194)
    at javax.script.AbstractScriptEngine.eval(AbstractScriptEngine.java:264)
    at preemptiveJstack.ActivateJstack.main(ActivateJstack.java:128)
```

BTrace

- An advanced open-source tool for extracting state from a live JVM.
- Uses a *Java agent* and a meta-scripting language to capture state.
- **Pros**: Lets you probe variable state without modifying / restarting the JVM.
- **Cons**: read-only querying using a custom syntax and libraries.

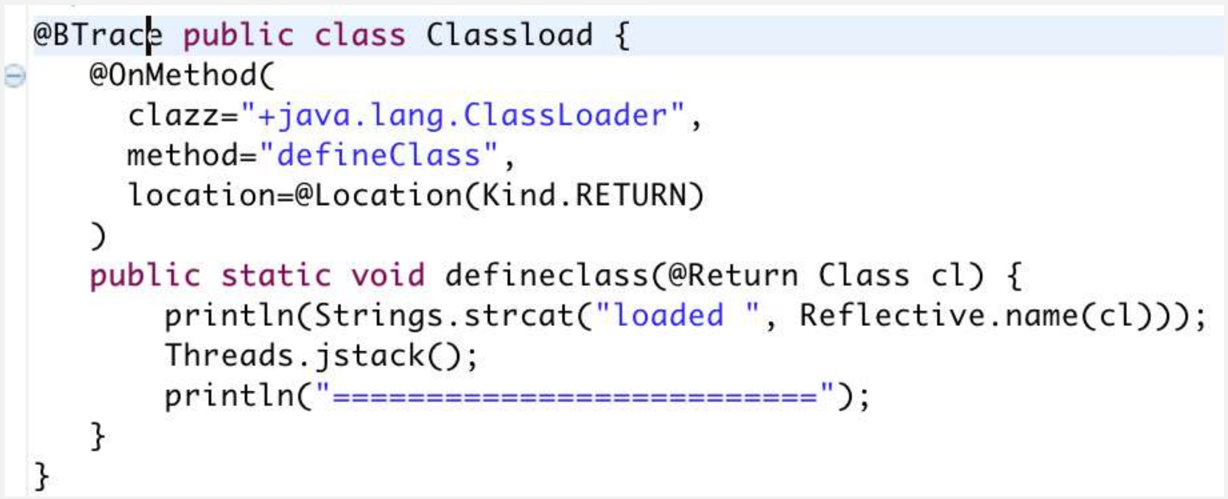
BTrace - Restrictions

- Can not create new objects.
- Can not create new arrays.
- Can not throw exceptions.
- Can not catch exceptions.
- Can not make arbitrary instance or static method calls - only the public static methods of com.sun.btrace.BTraceUtils class may be called from a BTrace program.
- Can not assign to static or instance fields of target program's classes and objects. But, BTrace class can assign to it's own static fields ("trace state" can be mutated).
- Can not have instance fields and methods. Only static public void returning methods are allowed for a BTrace class. And all fields have to be static.
- Can not have outer, inner, nested or local classes.
- Can not have synchronized blocks or synchronized methods.
- can not have loops (for, while, do..while)
- Can not extend arbitrary class (super class has to be java.lang.Object)
- Can not implement interfaces.
- Can not contains assert statements.
- Can not use class literals.

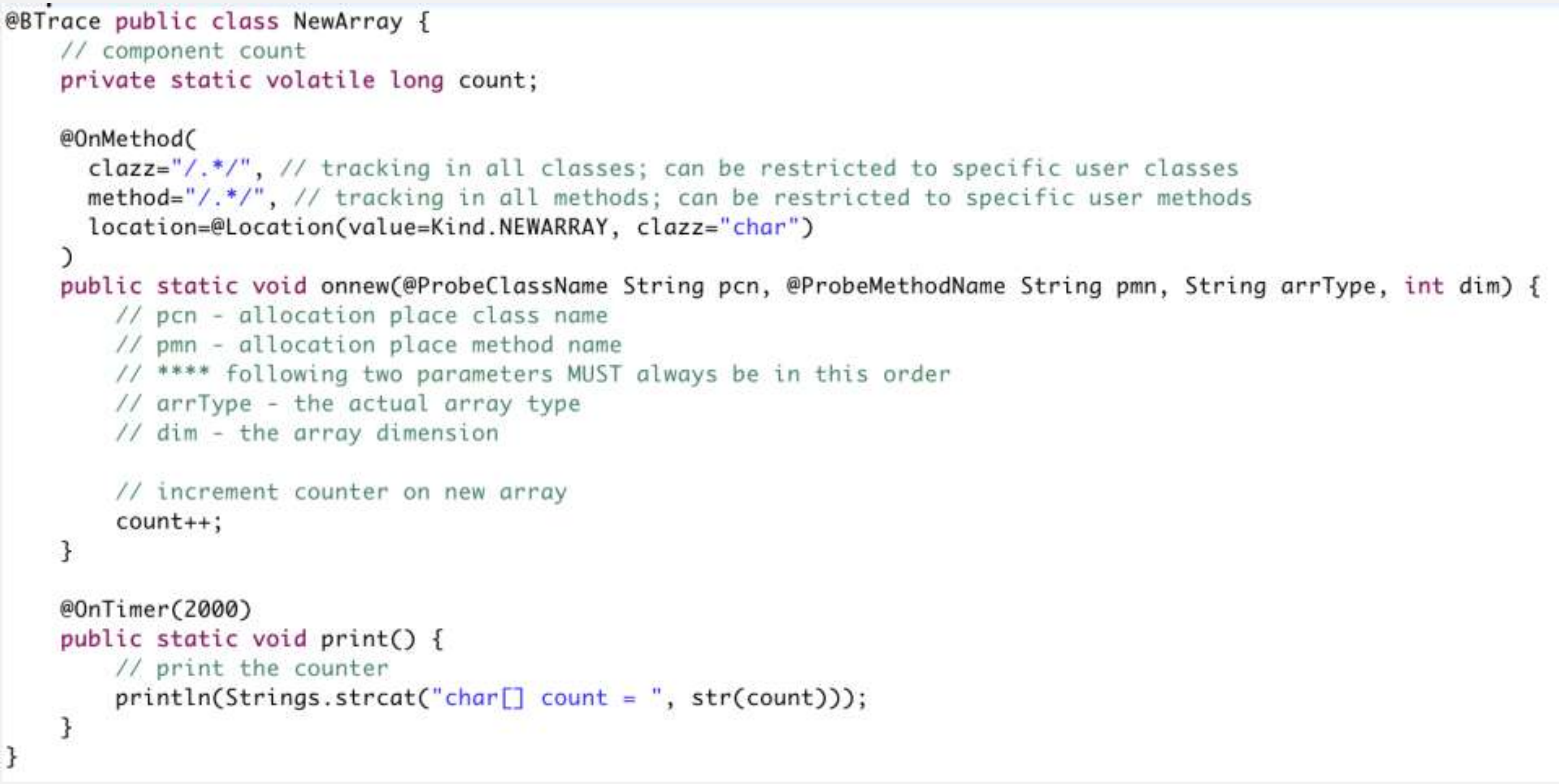
BTrace - Demo

kenai.com/projects/btrace

```java
@BTrace public class FileTracker {
    @TLS private static String name;

    @OnMethod(
        clazz="java.io.FileInputStream",
        method="<init>"
    )
    public static void onNewFileInputStream(@Self FileInputStream self, File f) {
        name = Strings.str(f);
    }

    @OnMethod(
        clazz="java.io.FileInputStream",
        method="<init>",
        type="void (java.io.File)",
        location=@Location(Kind.RETURN)
    )
    public static void onNewFileInputStreamReturn() {
        if (name != null) {
            println(Strings.strcat("opened for read ", name));
            name = null;
        }
    }

    @OnMethod(
        clazz="java.io.FileOutputStream",
        method="<init>"
    )
    public static void onNewFileOutputStream(@Self FileOutputStream self, File f, boolean b) {
        name = str(f);
    }

    @OnMethod(
        clazz="java.io.FileOutputStream",
        method="<init>",
        type="void (java.io.File, boolean)",
        location=@Location(Kind.RETURN)
    )
    public static void OnNewFileOutputStreamReturn() {
        if (name != null) {
            println(Strings.strcat("opened for write ", name));
            name = null;
        }
    }
```



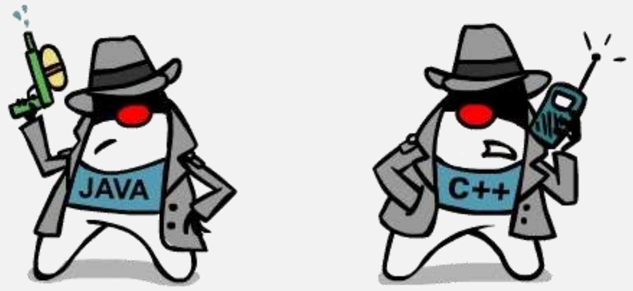
```java
@BTrace public class Classload {
    @OnMethod(
      clazz="+java.lang.ClassLoader",
      method="defineClass",
      location=@Location(Kind.RETURN)
    )
    public static void defineclass(@Return Class cl) {
        println(Strings.strcat("loaded ", Reflective.name(cl)));
        Threads.jstack();
        println("==========================");
    }
}
```



```java
@BTrace public class NewArray {
    // component count
    private static volatile long count;

    @OnMethod(
      clazz="/.*/", // tracking in all classes; can be restricted to specific user classes
      method="/.*/", // tracking in all methods; can be restricted to specific user methods
      location=@Location(value=Kind.NEWARRAY, clazz="char")
    )
    public static void onnew(@ProbeClassName String pcn, @ProbeMethodName String pmn, String arrType, int dim) {
        // pcn - allocation place class name
        // pmn - allocation place method name
        // **** following two parameters MUST always be in this order
        // arrType - the actual array type
        // dim - the array dimension

        // increment counter on new array
        count++;
    }

    @OnTimer(2000)
    public static void print() {
        // print the counter
        println(Strings.strcat("char[] count = ", str(count)));
    }
}
```

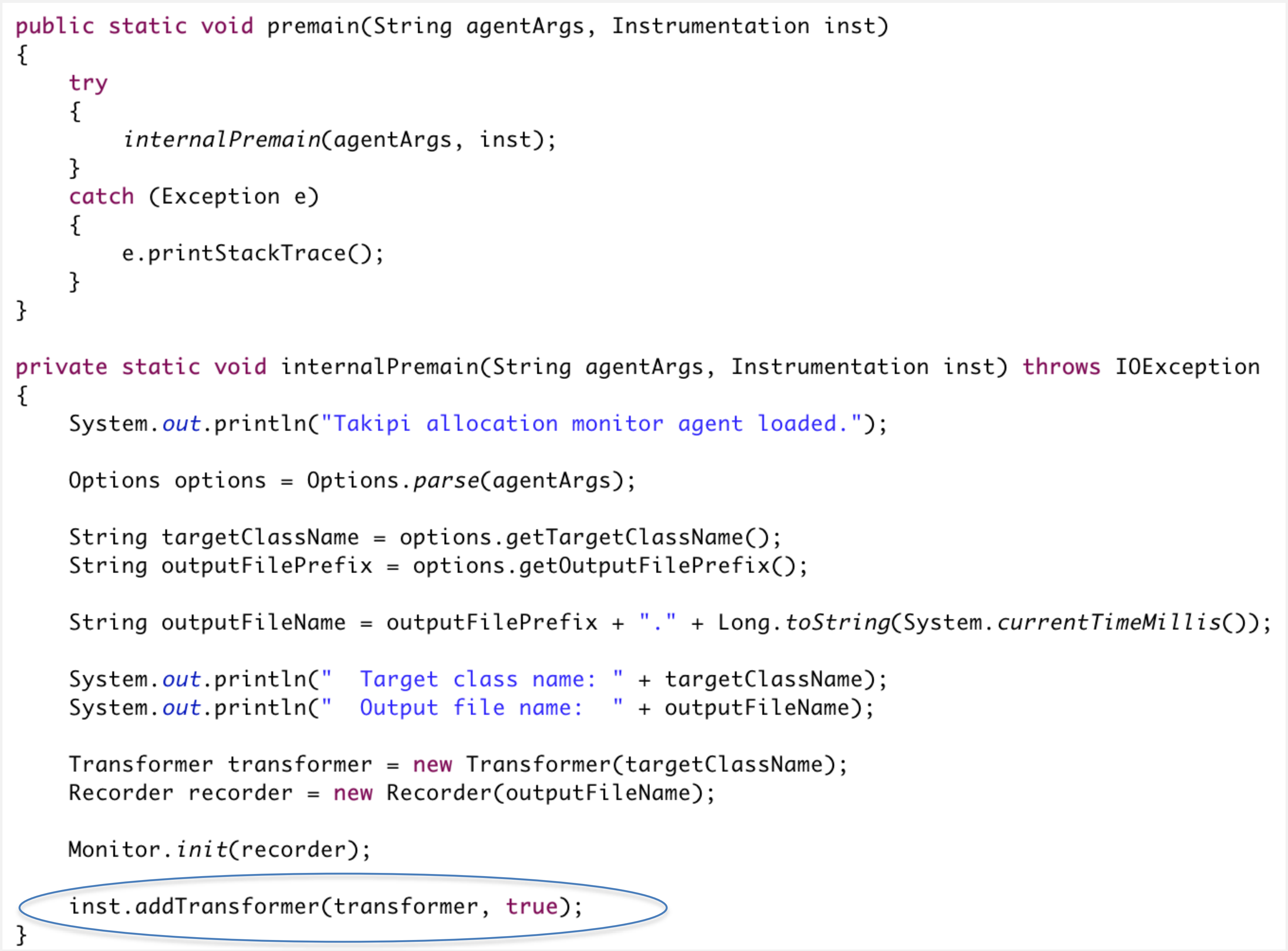
Custom Java Agents

- An advanced technique for instrumenting code dynamically.
- The foundation for most profiling / debugging tools.
- Two types of agents: Java and Native.
- **Pros**: extremely powerful technique to collect state from a live app.
- **Cons**: requires knowledge of creating *verifiable* bytecode.



Custom Agent - Demo

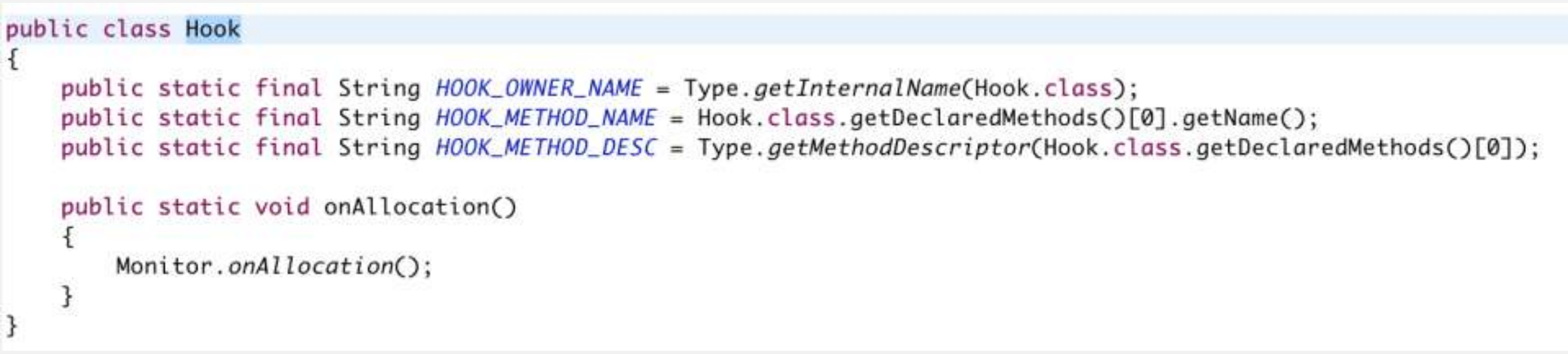
github.com/takipi/debugAgent



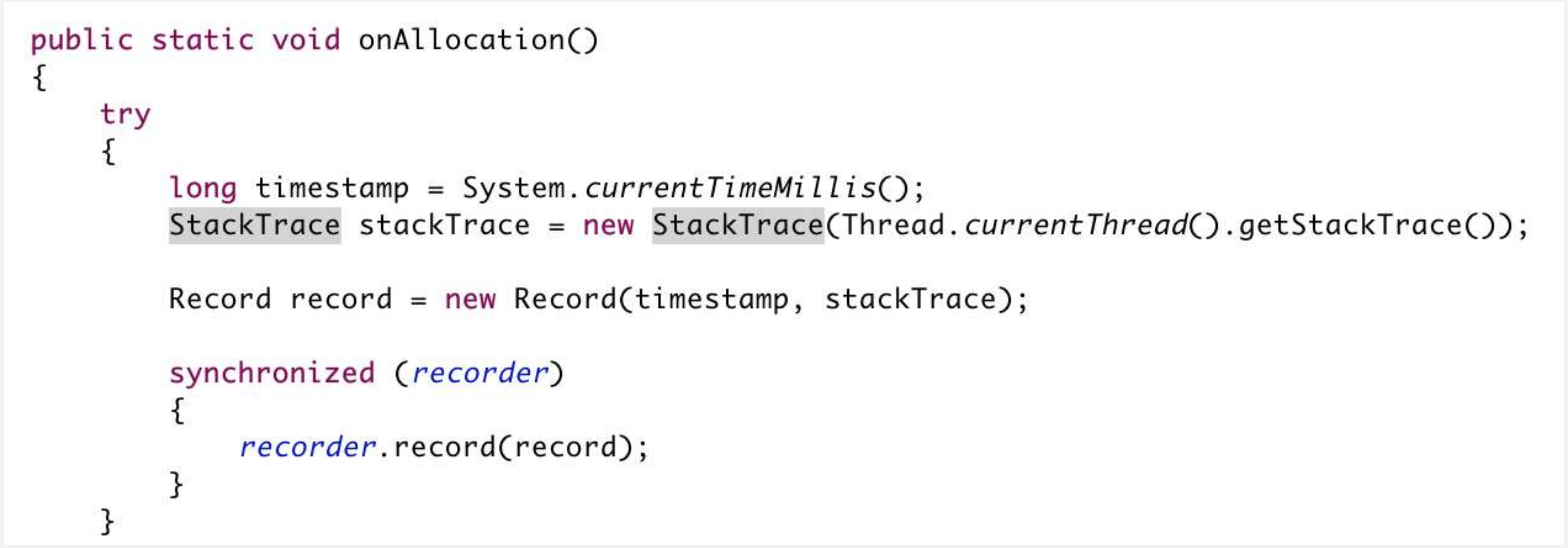
```
public static void premain(String agentArgs, Instrumentation inst)
{
    try
    {
        internalPremain(agentArgs, inst);
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

private static void internalPremain(String agentArgs, Instrumentation inst) throws IOException
{
    System.out.println("Takipi allocation monitor agent loaded.");

    Options options = Options.parse(agentArgs);

    String targetClassName = options.getTargetClassName();
    String outputFilePrefix = options.getOutputFilePrefix();

    String outputFileName = outputFilePrefix + "." + Long.toString(System.currentTimeMillis());

    System.out.println("  Target class name: " + targetClassName);
    System.out.println("  Output file name:  " + outputFileName);

    Transformer transformer = new Transformer(targetClassName);
    Recorder recorder = new Recorder(outputFileName);

    Monitor.init(recorder);

    inst.addTransformer(transformer, true);
}
```

```java
public class Transformer implements ClassFileTransformer
{
    private static final String INIT_METHOD_NAME    = "<init>";

    private final String targetClassName;

    public Transformer(String targetClassName)
    {
        this.targetClassName = targetClassName;
    }

    @Override
    public byte[] transform(ClassLoader loader, String className,
            Class<?> classBeingRedefined,
            ProtectionDomain protectionDomain, byte[] classfileBuffer)
            throws IllegalClassFormatException
    {
        if (!className.equals(targetClassName))
        {
            return null;
        }

        ClassReader cr = new ClassReader(classf
        ClassWriter cw = new ClassWriter(cr, Cl

        AllocationMonitorClassVisitor cv = new

        cr.accept(cv, 0);

        return cw.toByteArray();
    }
```

```java
@Override
public void visitCode()
{
    super.visitCode();

    super.visitMethodInsn(Opcodes.INVOKESTATIC,
            Hook.HOOK_OWNER_NAME,
            Hook.HOOK_METHOD_NAME,
            Hook.HOOK_METHOD_DESC, false);
}
```

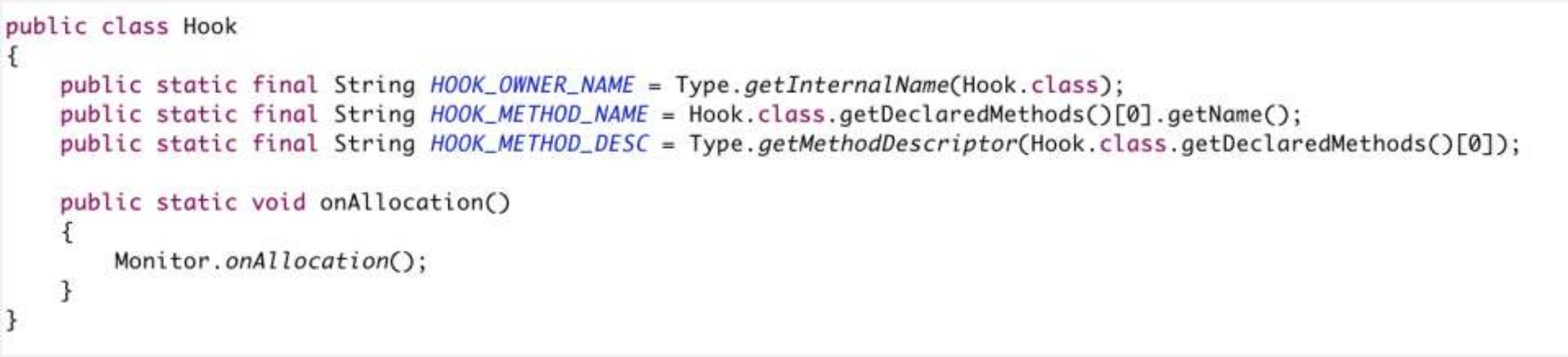


```
public class Hook
{
    public static final String HOOK_OWNER_NAME = Type.getInternalName(Hook.class);
    public static final String HOOK_METHOD_NAME = Hook.class.getDeclaredMethods()[0].getName();
    public static final String HOOK_METHOD_DESC = Type.getMethodDescriptor(Hook.class.getDeclaredMethods()[0]);

    public static void onAllocation()
    {
        Monitor.onAllocation();
    }
}
```

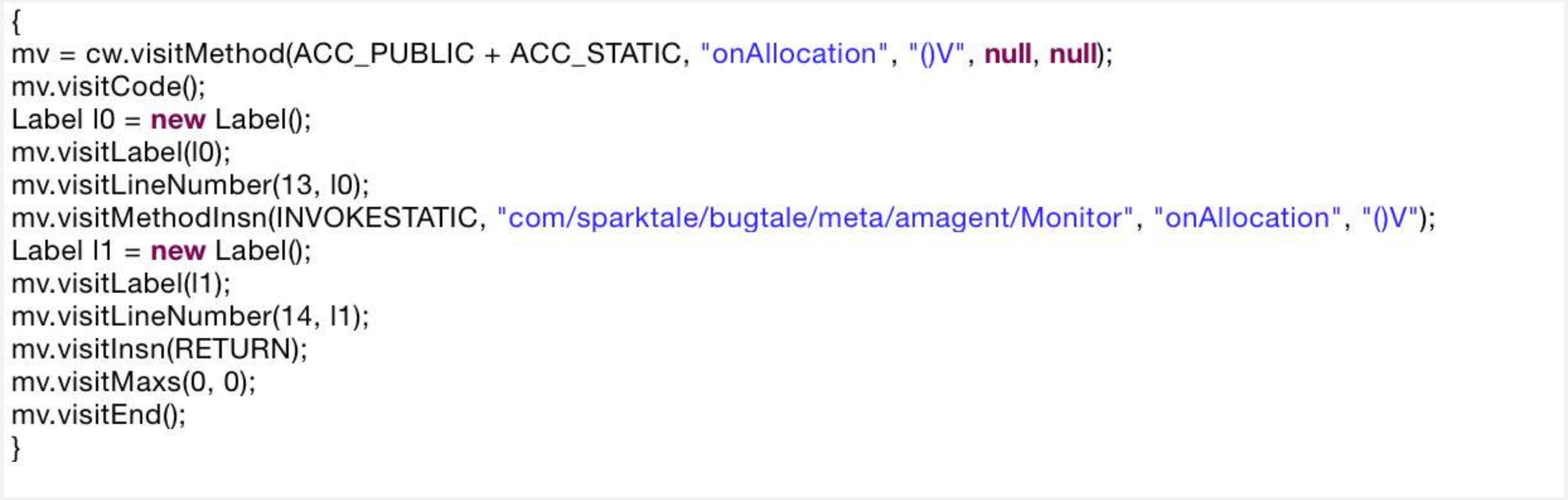


```
public static void onAllocation()
{
    try
    {
        long timestamp = System.currentTimeMillis();
        StackTrace stackTrace = new StackTrace(Thread.currentThread().getStackTrace());

        Record record = new Record(timestamp, stackTrace);

        synchronized (recorder)
        {
            recorder.record(record);
        }
    }
```

Auto generating bytecode (ASMifier)



```java
public class Hook
{
    public static final String HOOK_OWNER_NAME = Type.getInternalName(Hook.class);
    public static final String HOOK_METHOD_NAME = Hook.class.getDeclaredMethods()[0].getName();
    public static final String HOOK_METHOD_DESC = Type.getMethodDescriptor(Hook.class.getDeclaredMethods()[0]);

    public static void onAllocation()
    {
        Monitor.onAllocation();
    }
}
```



```java
{
mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "onAllocation", "()V", null, null);
mv.visitCode();
Label l0 = new Label();
mv.visitLabel(l0);
mv.visitLineNumber(13, l0);
mv.visitMethodInsn(INVOKESTATIC, "com/sparktale/bugtale/meta/amagent/Monitor", "onAllocation", "()V");
Label l1 = new Label();
mv.visitLabel(l1);
mv.visitLineNumber(14, l1);
mv.visitInsn(RETURN);
mv.visitMaxs(0, 0);
mv.visitEnd();
}
```

Native Agents

- Java agents are written in Java. Have access to the *Instrumentation* API.
- Native agents – written in C++.
- Have access to JVMTI – the JVM's low-level set of APIs and capabilities.
  - JIT compilation, GC, Monitor, Exception, breakpoints, ..
- More complex to write. Capability performance impact.
- Platform dependent.

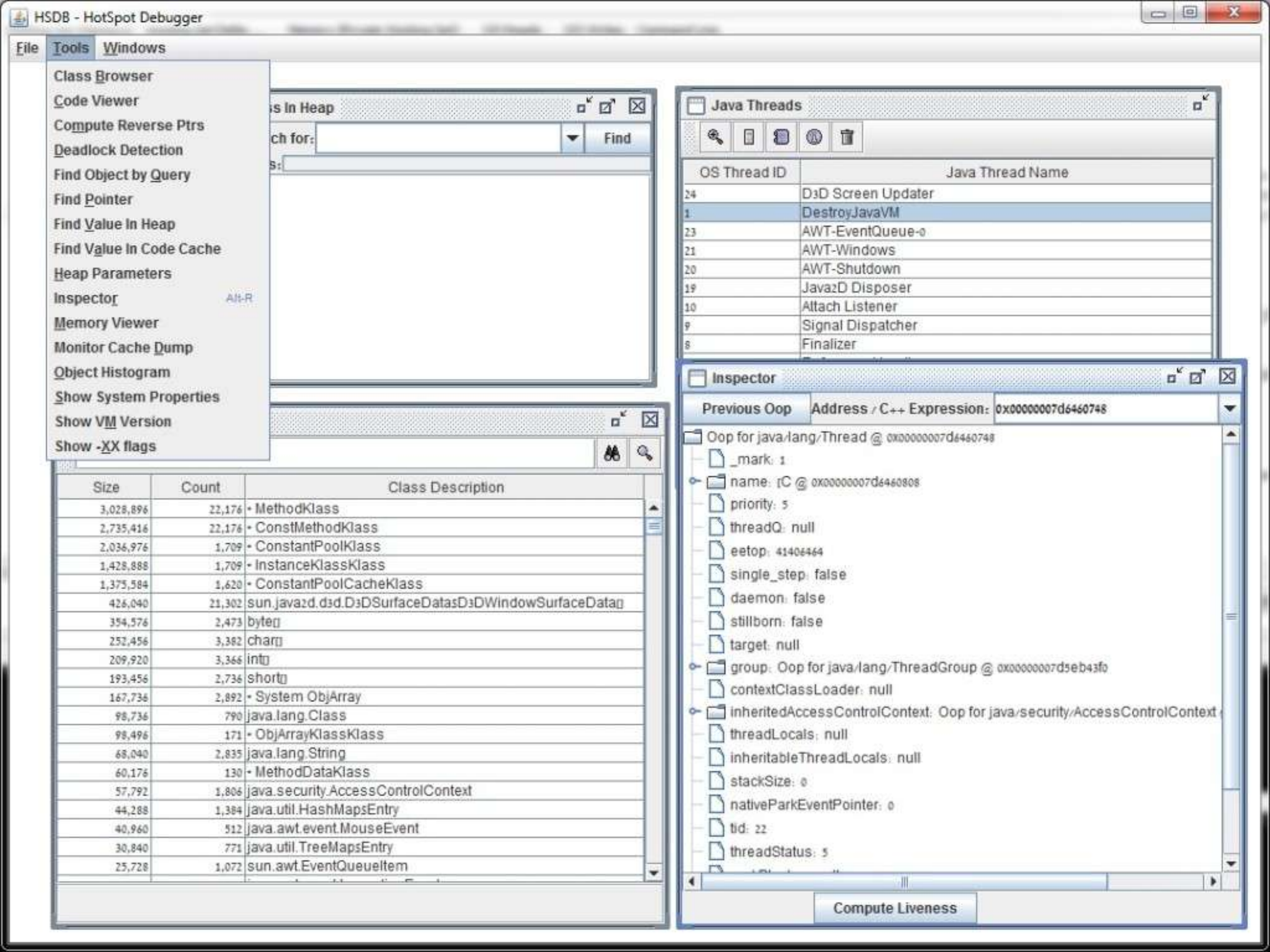
Thanks!

Takipi - Detect, priotitize and debug bugs at high-scale.

tal.weiss@takipi.com

@takipid

takipiblog.com

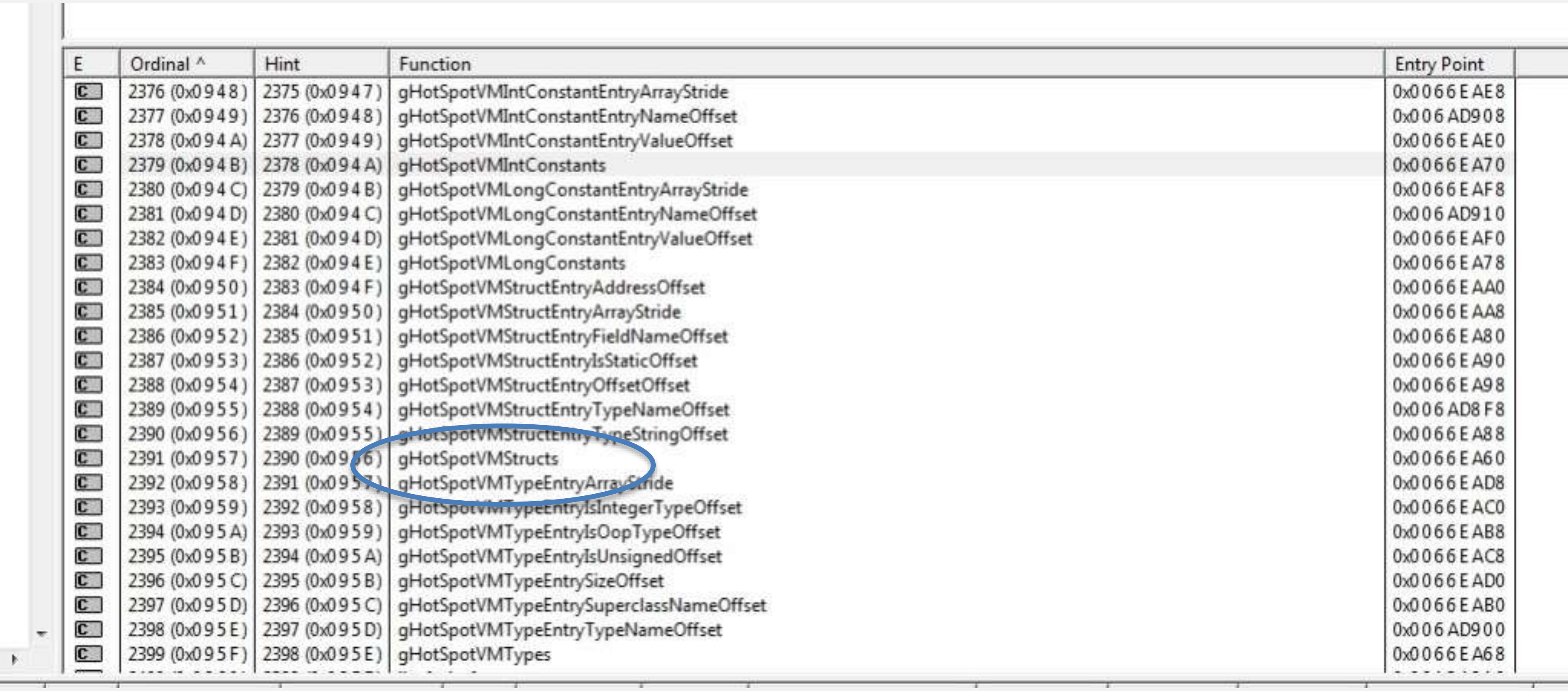


OS Thread ID	Java Thread Name
24	D3D Screen Updater
1	DestroyJavaVM
23	AWT-EventQueue-0
21	AWT-Windows
20	AWT-Shutdown
19	Java2D Disposer
10	Attach Listener
9	Signal Dispatcher
8	Finalizer

Address / C++ Expression: 0x000000007d6460748

```
Oop for java/lang/Thread @ 0x000000007d6460748
  _mark: 1
  name: [C @ 0x000000007d6460808
  priority: 5
  threadQ: null
  eetop: 41406464
  single_step: false
  daemon: false
  stillborn: false
  target: null
  group: Oop for java/lang/ThreadGroup @ 0x000000007d5eb43f0
  contextClassLoader: null
  inheritedAccessControlContext: Oop for java/security/AccessControlContext
  threadLocals: null
  inheritableThreadLocals: null
  stackSize: 0
  nativeParkEventPointer: 0
  tid: 22
  threadStatus: 5
```

Size	Count	Class Description
3,028,896	22,176	• MethodKlass
2,735,416	22,176	• ConstMethodKlass
2,036,976	1,709	• ConstantPoolKlass
1,428,888	1,709	• InstanceKlassKlass
1,375,584	1,620	• ConstantPoolCacheKlass
426,040	21,302	sun.java2d.d3d.D3DSurfaceData$D3DWindowSurfaceData[]
354,576	2,473	byte[]
252,456	3,382	char[]
209,920	3,366	int[]
193,456	2,736	short[]
167,736	2,892	• System ObjArray
98,736	790	java.lang.Class
98,496	171	• ObjArrayKlassKlass
68,040	2,835	java.lang.String
60,176	130	• MethodDataKlass
57,792	1,806	java.security.AccessControlContext
44,288	1,384	java.util.HashMap$Entry
40,960	512	java.awt.event.MouseEvent
30,840	771	java.util.TreeMap$Entry
25,728	1,072	sun.awt.EventQueueItem



E	Ordinal ^	Hint	Function	Entry Point
C	2376 (0x0948)	2375 (0x0947)	gHotSpotVMIntConstantEntryArrayStride	0x0066EAE8
C	2377 (0x0949)	2376 (0x0948)	gHotSpotVMIntConstantEntryNameOffset	0x006AD908
C	2378 (0x094A)	2377 (0x0949)	gHotSpotVMIntConstantEntryValueOffset	0x0066EAE0
C	2379 (0x094B)	2378 (0x094A)	gHotSpotVMIntConstants	0x0066EA70
C	2380 (0x094C)	2379 (0x094B)	gHotSpotVMLongConstantEntryArrayStride	0x0066EAF8
C	2381 (0x094D)	2380 (0x094C)	gHotSpotVMLongConstantEntryNameOffset	0x006AD910
C	2382 (0x094E)	2381 (0x094D)	gHotSpotVMLongConstantEntryValueOffset	0x0066EAF0
C	2383 (0x094F)	2382 (0x094E)	gHotSpotVMLongConstants	0x0066EA78
C	2384 (0x0950)	2383 (0x094F)	gHotSpotVMStructEntryAddressOffset	0x0066EAA0
C	2385 (0x0951)	2384 (0x0950)	gHotSpotVMStructEntryArrayStride	0x0066EAA8
C	2386 (0x0952)	2385 (0x0951)	gHotSpotVMStructEntryFieldNameOffset	0x0066EA80
C	2387 (0x0953)	2386 (0x0952)	gHotSpotVMStructEntryIsStaticOffset	0x0066EA90
C	2388 (0x0954)	2387 (0x0953)	gHotSpotVMStructEntryOffsetOffset	0x0066EA98
C	2389 (0x0955)	2388 (0x0954)	gHotSpotVMStructEntryTypeNameOffset	0x006AD8F8
C	2390 (0x0956)	2389 (0x0955)	gHotSpotVMStructEntryTypeStringOffset	0x0066EA88
C	2391 (0x0957)	2390 (0x0956)	gHotSpotVMStructs	0x0066EA60
C	2392 (0x0958)	2391 (0x0957)	gHotSpotVMTypeEntryArrayStride	0x0066EAD8
C	2393 (0x0959)	2392 (0x0958)	gHotSpotVMTypeEntryIsIntegerTypeOffset	0x0066EAC0
C	2394 (0x095A)	2393 (0x0959)	gHotSpotVMTypeEntryIsOopTypeOffset	0x0066EAB8
C	2395 (0x095B)	2394 (0x095A)	gHotSpotVMTypeEntryIsUnsignedOffset	0x0066EAC8
C	2396 (0x095C)	2395 (0x095B)	gHotSpotVMTypeEntrySizeOffset	0x0066EAD0
C	2397 (0x095D)	2396 (0x095C)	gHotSpotVMTypeEntrySuperclassNameOffset	0x0066EAB0
C	2398 (0x095E)	2397 (0x095D)	gHotSpotVMTypeEntryTypeNameOffset	0x006AD900
C	2399 (0x095F)	2398 (0x095E)	gHotSpotVMTypes	0x0066EA68