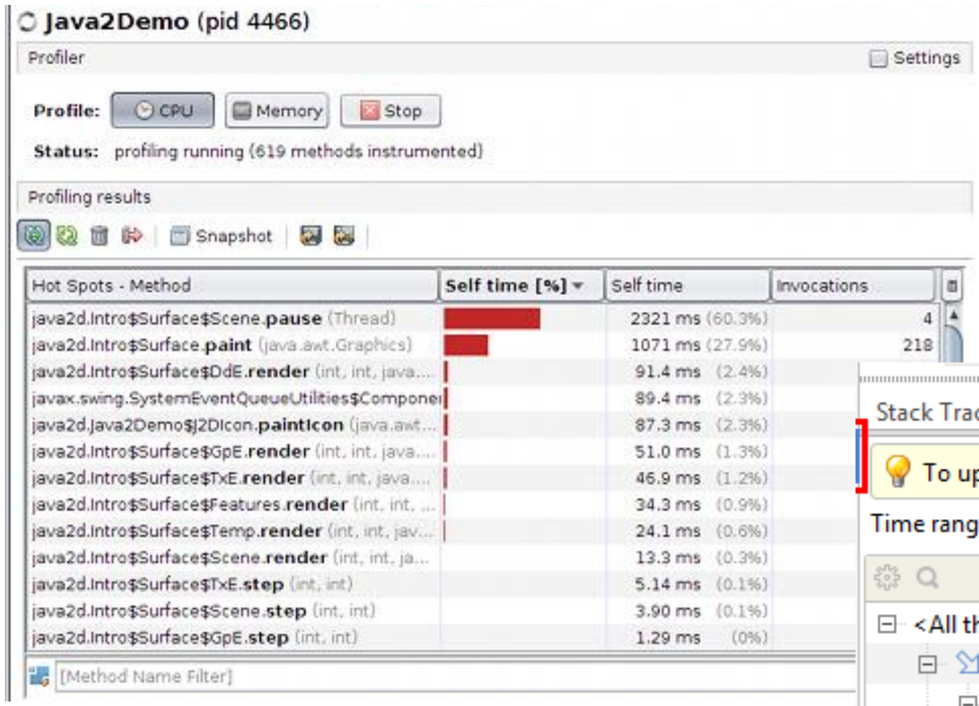
In-situ Visualization of Profiling Data

Sebastian Baltes
University of Trier, Germany
@s_baltes
research@sbaltes.com

Oliver Moseler
University of Trier, Germany
olivermoseler@gmx.de

Visual Performance Analysis Tools

- **Profiling tools** record program runs and assign measured performance values to code entities (e.g. runtime or memory consumption)
- We focus on **runtime consumption** and **Java** programs
- Standard user interface: **Lists**



Java2Demo (pid 4466)

Profiler | Settings

Profile: CPU | Memory | Stop

Status: profiling running (619 methods instrumented)

Profiling results

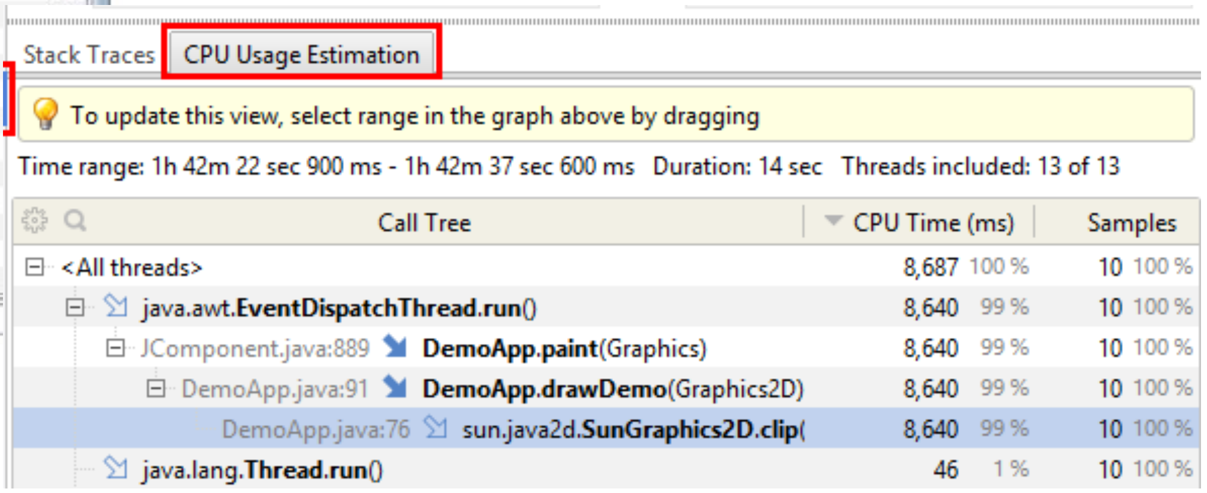
Snapshot

Hot Spots - Method	Self time [%]	Self time	Invocations
java2d.Intro$Surface$Scene.**pause** (Thread)		2321 ms (60.3%)	4
java2d.Intro$Surface.**paint** (java.awt.Graphics)		1071 ms (27.9%)	218
java2d.Intro$Surface$DdE.**render** (int, int, java...		91.4 ms (2.4%)	
javax.swing.SystemEventQueueUtilities$Compone		89.4 ms (2.3%)	
java2d.Java2Demo$J2DIcon.**paintIcon** (java.awt...		87.3 ms (2.3%)	
java2d.Intro$Surface$GpE.**render** (int, int, java....		51.0 ms (1.3%)	
java2d.Intro$Surface$TxE.**render** (int, int, java....		46.9 ms (1.2%)	
java2d.Intro$Surface$Features.**render** (int, int, ...		34.3 ms (0.9%)	
java2d.Intro$Surface$Temp.**render** (int, int, jav...		24.1 ms (0.6%)	
java2d.Intro$Surface$Scene.**render** (int, int, ja...		13.3 ms (0.3%)	
java2d.Intro$Surface$TxE.**step** (int, int)		5.14 ms (0.1%)	
java2d.Intro$Surface$Scene.**step** (int, int)		3.90 ms (0.1%)	
java2d.Intro$Surface$GpE.**step** (int, int)		1.29 ms (0%)	

[Method Name Filter]

VisualVM

YourKit



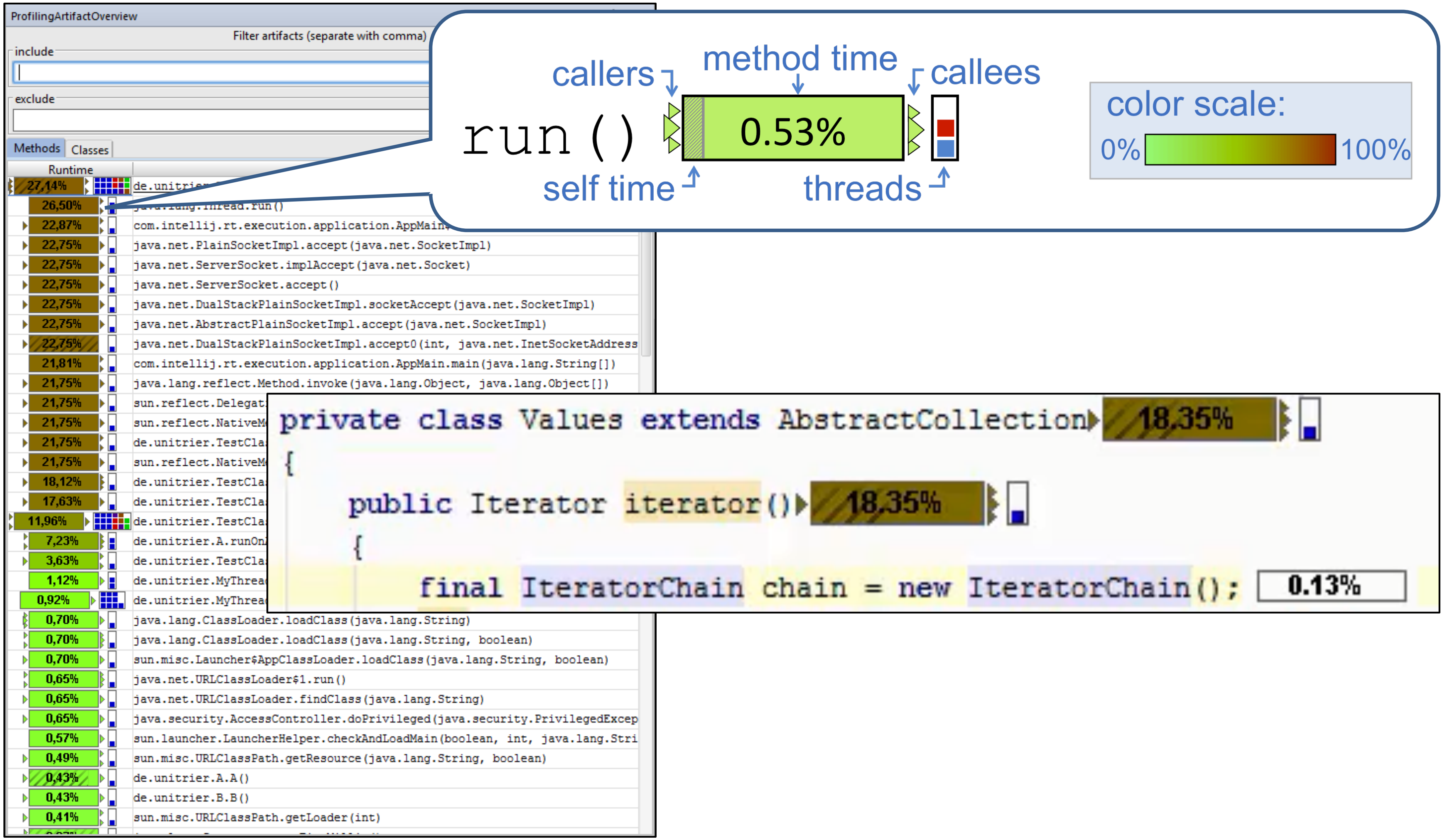
Stack Traces | CPU Usage Estimation

To update this view, select range in the graph above by dragging

Time range: 1h 42m 22 sec 900 ms - 1h 42m 37 sec 600 ms   Duration: 14 sec   Threads included: 13 of 13

Call Tree	CPU Time (ms)		Samples	
<All threads>	8,687	100 %	10	100 %
java.awt.**EventDispatchThread.run**()	8,640	99 %	10	100 %
JComponent.java:889 **DemoApp.paint**(Graphics)	8,640	99 %	10	100 %
DemoApp.java:91 **DemoApp.drawDemo**(Graphics2D)	8,640	99 %	10	100 %
DemoApp.java:76 sun.java2d.**SunGraphics2D.clip**(	8,640	99 %	10	100 %
java.lang.**Thread.run**()	46	1 %	10	100 %

Our Tool



ProfilingArtifactOverview

Filter artifacts (separate with comma)

include

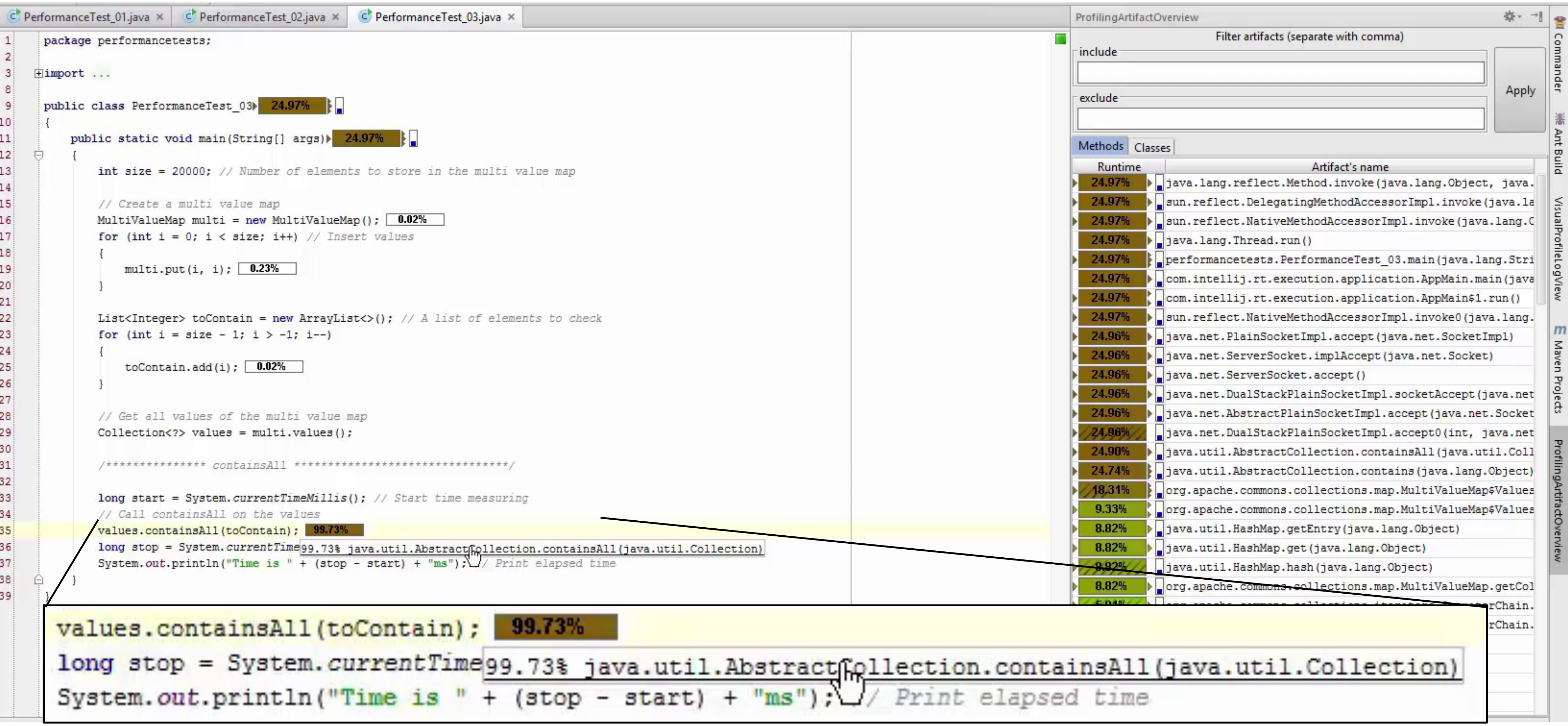
exclude

Methods | Classes

Runtime	
27,14%	de.unitrier
26,50%	java.lang.Thread.run()
22,87%	com.intellij.rt.execution.application.AppMain
22,75%	java.net.PlainSocketImpl.accept(java.net.SocketImpl)
22,75%	java.net.ServerSocket.implAccept(java.net.Socket)
22,75%	java.net.ServerSocket.accept()
22,75%	java.net.DualStackPlainSocketImpl.socketAccept(java.net.SocketImpl)
22,75%	java.net.AbstractPlainSocketImpl.accept(java.net.SocketImpl)
22,75%	java.net.DualStackPlainSocketImpl.accept0(int, java.net.InetSocketAddress
21,81%	com.intellij.rt.execution.application.AppMain.main(java.lang.String[])
21,75%	java.lang.reflect.Method.invoke(java.lang.Object, java.lang.Object[])
21,75%	sun.reflect.Delegat
21,75%	sun.reflect.NativeM
21,75%	de.unitrier.TestCla
21,75%	sun.reflect.NativeM
18,12%	de.unitrier.TestCla
17,63%	de.unitrier.TestCla
11,96%	de.unitrier.TestCla
7,23%	de.unitrier.A.runOn
3,63%	de.unitrier.TestCla
1,12%	de.unitrier.MyThrea
0,92%	de.unitrier.MyThrea
0,70%	java.lang.ClassLoader.loadClass(java.lang.String)
0,70%	java.lang.ClassLoader.loadClass(java.lang.String, boolean)
0,70%	sun.misc.Launcher$AppClassLoader.loadClass(java.lang.String, boolean)
0,65%	java.net.URLClassLoader$1.run()
0,65%	java.net.URLClassLoader.findClass(java.lang.String)
0,65%	java.security.AccessController.doPrivileged(java.security.PrivilegedExcep
0,57%	sun.launcher.LauncherHelper.checkAndLoadMain(boolean, int, java.lang.Stri
0,49%	sun.misc.URLClassPath.getResource(java.lang.String, boolean)
0,43%	de.unitrier.A.A()
0,43%	de.unitrier.B.B()
0,41%	sun.misc.URLClassPath.getLoader(int)

```
private class Values extends AbstractCollection 18,35%
{
    public Iterator iterator() 18,35%
    {
        final IteratorChain chain = new IteratorChain(); 0.13%
```

Navigation



```
package performancetests;

import ...

public class PerformanceTest_03 24.97%
{
    public static void main(String[] args) 24.97%
    {
        int size = 20000; // Number of elements to store in the multi value map

        // Create a multi value map
        MultiValueMap multi = new MultiValueMap(); 0.02%
        for (int i = 0; i < size; i++) // Insert values
        {
            multi.put(i, i); 0.23%
        }

        List<Integer> toContain = new ArrayList<>(); // A list of elements to check
        for (int i = size - 1; i > -1; i--)
        {
            toContain.add(i); 0.02%
        }

        // Get all values of the multi value map
        Collection<?> values = multi.values();

        /*************** containsAll ********************************/

        long start = System.currentTimeMillis(); // Start time measuring
        // Call containsAll on the values
        values.containsAll(toContain); 99.73%
        long stop = System.currentTime 99.73% java.util.AbstractCollection.containsAll(java.util.Collection)
        System.out.println("Time is " + (stop - start) + "ms"); // Print elapsed time
    }
}
```

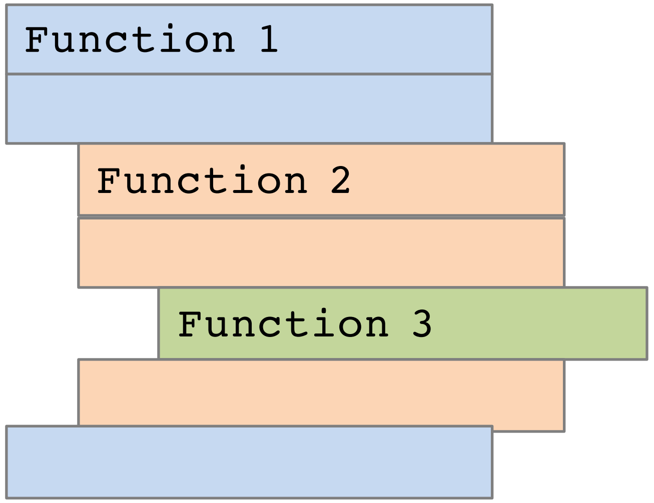
Runtime	Artifact's name
24.97%	java.lang.reflect.Method.invoke(java.lang.Object, java.
24.97%	sun.reflect.DelegatingMethodAccessorImpl.invoke(java.la
24.97%	sun.reflect.NativeMethodAccessorImpl.invoke(java.lang.C
24.97%	java.lang.Thread.run()
24.97%	performancetests.PerformanceTest_03.main(java.lang.Stri
24.97%	com.intellij.rt.execution.application.AppMain.main(java
24.97%	com.intellij.rt.execution.application.AppMain$1.run()
24.97%	sun.reflect.NativeMethodAccessorImpl.invoke0(java.lang.
24.96%	java.net.PlainSocketImpl.accept(java.net.SocketImpl)
24.96%	java.net.ServerSocket.implAccept(java.net.Socket)
24.96%	java.net.ServerSocket.accept()
24.96%	java.net.DualStackPlainSocketImpl.socketAccept(java.net
24.96%	java.net.AbstractPlainSocketImpl.accept(java.net.Socket
24.96%	java.net.DualStackPlainSocketImpl.accept0(int, java.net
24.90%	java.util.AbstractCollection.containsAll(java.util.Coll
24.74%	java.util.AbstractCollection.contains(java.lang.Object)
18.31%	org.apache.commons.collections.map.MultiValueMap$Values
9.33%	org.apache.commons.collections.map.MultiValueMap$Values
8.82%	java.util.HashMap.getEntry(java.lang.Object)
8.82%	java.util.HashMap.get(java.lang.Object)
8.82%	java.util.HashMap.hash(java.lang.Object)
8.82%	org.apache.commons.collections.map.MultiValueMap.getCol
[illegible]	[illegible]rChain.
[illegible]	[illegible]rChain.

```
values.containsAll(toContain); 99.73%
long stop = System.currentTime 99.73% java.util.AbstractCollection.containsAll(java.util.Collection)
System.out.println("Time is " + (stop - start) + "ms"); // Print elapsed time
```

Stack Sampling

- **Profiler:**
  - Analysis tool
  - Measure runtime consumption or memory usage of a program
  - Identify performance bugs
  - Optimize programs

- **Sampling approach:**
  - Heuristic methodology
  - Estimate runtime consumption
  - Stop target program periodically
  - Record a sample of the current state of the stack traces from all threads
  - Target program runs slower



Stack Sampling

- **Post mortem analysis:**
  - *Method time:*
    Method was found within a stack
  - *Self time:*
    Method was found on top of a stack
  - *Caller and callee runtime:*
    Time spent in called methods

- **The approach doesn't track every single stack trace:**
  - Results can vary
  - Run multiple samplings to get more reliable propositions

More Information

debugging.sbaltes.com

Sebastian Baltes
University of Trier, Germany

@s_baltes

research@sbaltes.com

Oliver Moseler
University of Trier, Germany

olivermoseler@gmx.de