#BaselOne17

CSS Versicherung

Adobe

Basel basel.ch

adesso business. people. technology.

JAVA USER GROUP CH

bluesky CONSIDER IT DONE

canoo [delivering end-user happiness]

Basler Versicherungen

OpenJDK tools for fun and profit

by Dmitry Vyazelenko @ BaselOne 2017

About me

- Senior Software Engineer @ Canoo Engineering AG, Switzerland

- Disorganizer at JCrete Unconference, www.jcrete.org

- Contacts: vyazelenko.com, @DVyazelenko

Agenda

- JOL
- JMH
- JCStress
- JITWatch

JOL

JOL (Java Object Layout) is the tiny toolbox to analyze object layout schemes in JVMs. These tools are using Unsafe, JVMTI, and Serviceability Agent (SA) heavily to decoder the *actual* object layout, footprint, and references. This makes JOL much more accurate than other tools relying on heap dumps, specification assumptions, etc.

http://openjdk.java.net/projects/code-tools/jol/

Getting started

- Maven dependency:

```
<dependency>
    <groupId>org.openjdk.jol</groupId>
    <artifactId>jol-core</artifactId>
    <version>put-the-version-here</version>
</dependency>
```

- Build from source:

hg clone http://hg.openjdk.java.net/code-tools/jol/ jol

- Java version: 6+

Command Line Usage

```
java -jar jol-cli/target/jol-cli.jar help

Usage: jol-cli.jar <mode> [optional arguments]*

Available modes:
             estimates: Simulate the class layout in different VM modes.
             externals: Show the object externals: the objects reachable from a given instance.
             footprint: Estimate the footprint of all objects reachable from a given instance
              heapdump: Consume the heap dump and estimate the savings in different layout strategies.
         heapdumpstats: Consume the heap dump and print the most frequent instances.
             idealpack: Compute the object footprint under different field layout strategies.
             internals: Show the object internals: field layout and default contents, object header
                shapes: Dump the object shapes present in JAR files or heap dumps.
       string-compress: Consume the heap dumps and figures out the savings attainable with compressed
strings.
```

```
java -jar jol-cli/target/jol-cli.jar internals java.lang.Boolean

# Running 64-bit HotSpot VM.
# Using compressed oop with 3-bit shift.
# Using compressed klass with 3-bit shift.
# Objects are 8 bytes aligned.
# Field sizes by type: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]
# Array element sizes: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]

Instantiated the sample instance via public java.lang.Boolean(boolean)

java.lang.Boolean object internals:
 OFFSET  SIZE      TYPE DESCRIPTION                               VALUE
      0     4           (object header)                           01 00 00 00 (00000001 00000000 00000000 00000000) (1)
      4     4           (object header)                           00 00 00 00 (00000000 00000000 00000000 00000000) (0)
      8     4           (object header)                           85 20 00 f8 (10000101 00100000 00000000 11111000) (-134209403)
     12     1   boolean Boolean.value                             false
     13     3           (loss due to the next object alignment)
Instance size: 16 bytes
Space losses: 0 bytes internal + 3 bytes external = 3 bytes total
```

Alignment

```
package org.openjdk.jol.samples;

import org.openjdk.jol.info.ClassLayout;
import org.openjdk.jol.vm.VM;

import static java.lang.System.out;

public class JOLSample_02_Alignment {

    public static void main(String[] args) throws Exception {
        out.println(VM.current().details());
        out.println(ClassLayout.parseClass(A.class).toPrintable());
    }

    public static class A {
        long f;
    }

}
```

Alignment

```
# Running 64-bit HotSpot VM.
# Using compressed oop with 3-bit shift.
# Using compressed klass with 3-bit shift.
# Objects are 8 bytes aligned.
# Field sizes by type: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]
# Array element sizes: 4, 1, 1, 2, 2, 4, 4, 8, 8 [bytes]

org.openjdk.jol.samples.JOLSample_02_Alignment$A object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                               VALUE
      0    12        (object header)                           N/A
     12     4        (alignment/padding gap)
     16     8   long A.f                                       N/A
Instance size: 24 bytes
Space losses: 4 bytes internal + 0 bytes external = 4 bytes total
```

Throwable

```
package org.openjdk.jol.samples;

import org.openjdk.jol.info.ClassLayout;
import org.openjdk.jol.vm.VM;

import static java.lang.System.out;

public class JOLSample_07_Exceptions {

    public static void main(String[] args) throws Exception {
        out.println(VM.current().details());
        out.println(ClassLayout.parseClass(Throwable.class).toPrintable());
    }

}
```

Throwable

```
java.lang.Throwable object internals:
 OFFSET  SIZE                               TYPE DESCRIPTION                               VALUE
      0    12                                    (object header)                           N/A
     12     4                                    (alignment/padding gap)
     16     4                   java.lang.String Throwable.detailMessage                   N/A
     20     4              java.lang.Throwable Throwable.cause                             N/A
     24     4   java.lang.StackTraceElement[] Throwable.stackTrace                         N/A
     28     4                  java.util.List Throwable.suppressedExceptions               N/A
Instance size: 32 bytes
Space losses: 4 bytes internal + 0 bytes external = 4 bytes total
```

Class

```
package org.openjdk.jol.samples;

import org.openjdk.jol.info.ClassLayout;
import org.openjdk.jol.vm.VM;

import static java.lang.System.out;

public class JOLSample_08_Class {

    public static void main(String[] args) throws Exception {
        out.println(VM.current().details());
        out.println(ClassLayout.parseClass(Class.class).toPrintable());
    }

}
```

Class

```
java.lang.Class object internals:
 OFFSET  SIZE                                               TYPE DESCRIPTION                                          VALUE
      0    12                                                    (object header)                                      N/A
     12     4                        java.lang.reflect.Constructor Class.cachedConstructor                            N/A
     16     4                                java.lang.Class Class.newInstanceCallerCache                             N/A
     20     4                               java.lang.String Class.name                                               N/A
     24     4                                                    (alignment/padding gap)
     28     4                          java.lang.ref.SoftReference Class.reflectionData                               N/A
     32     4   sun.reflect.generics.repository.ClassRepository Class.genericInfo                                    N/A
     36     4                                java.lang.Object[] Class.enumConstants                                   N/A
     40     4                                     java.util.Map Class.enumConstantDirectory                           N/A
     44     4                      java.lang.Class.AnnotationData Class.annotationData                               N/A
     48     4              sun.reflect.annotation.AnnotationType Class.annotationType                                N/A
     52     4                 java.lang.ClassValue.ClassValueMap Class.classValueMap                                 N/A
     56    32                                                    (alignment/padding gap)
     88     4                                                int Class.classRedefinedCount                         N/A
     92     4                                                    (loss due to the next object alignment)
Instance size: 96 bytes
Space losses: 36 bytes internal + 4 bytes external = 40 bytes total
```

Collections

```
public class CollectionLayout {
    public static void main(String[] args) {
        int[] array = new int[1_000_000];
        ArrayList<Integer> arrayList = new ArrayList<>(array.length);
        LinkedList<Integer> linkedList = new LinkedList<>();
        HashSet<Integer> hashSet = new HashSet<>();
        for (int i = 0; i < array.length; i++) {
            array[i] = i;
            Integer value = new Integer(i);
            arrayList.add(value);
            linkedList.add(value);
            hashSet.add(value);
        }
        System.out.println(GraphLayout.parseInstance(array).toFootprint());
        System.out.println(GraphLayout.parseInstance(arrayList).toFootprint());
        System.out.println(GraphLayout.parseInstance(linkedList).toFootprint());
        System.out.println(GraphLayout.parseInstance(hashSet).toFootprint());
    }
}
```

Collections

```
[I@39ed3c8dd footprint:
     COUNT       AVG       SUM   DESCRIPTION
         1   4000016   4000016   [I
         1             4000016   (total)

java.util.ArrayList@71dac704d footprint:
     COUNT       AVG       SUM   DESCRIPTION
         1   4000016   4000016   [Ljava.lang.Object;
   1000000        16  16000000   java.lang.Integer
         1        24        24   java.util.ArrayList
   1000002             20000040   (total)
```

Collection	Overhead in bytes
ArrayList	4
LinkedList	24
HashMap	40

```
java.util.LinkedList@6a213a91d footprint:
     COUNT       AVG       SUM   DESCRIPTION
   1000000        16  16000000   java.lang.Integer
         1        32        32   java.util.LinkedList
   1000000        24  24000000   java.util.LinkedList$Node
   2000001             40000032   (total)

java.util.HashSet@1b5b7e1bd footprint:
     COUNT       AVG       SUM   DESCRIPTION
         1   8388624   8388624   [Ljava.util.HashMap$Node;
   1000000        16  16000000   java.lang.Integer
         1        16        16   java.lang.Object
         1        48        48   java.util.HashMap
   1000000        32  32000000   java.util.HashMap$Node
         1        16        16   java.util.HashSet
   2000004             56388704   (total)
```

Object header

```
public class JOLSample_15_IdentityHashCode {
    public static void main(String[] args) throws Exception {
        out.println(VM.current().details());

        final A a = new A();

        ClassLayout layout = ClassLayout.parseInstance(a);

        out.println("**** Fresh object");
        out.println(layout.toPrintable());

        out.println("hashCode: " + Integer.toHexString(a.hashCode()));
        out.println();

        out.println("**** After identityHashCode()");
        out.println(layout.toPrintable());
    }

    public static class A {
   }
}
```

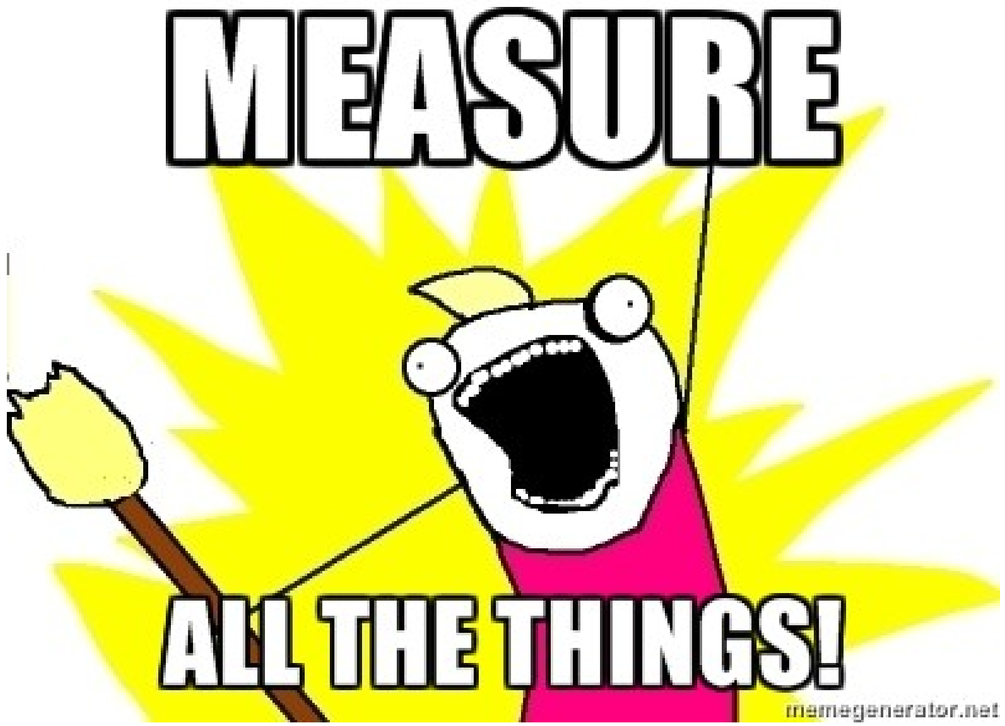
Object header

```
**** Fresh object
org.openjdk.jol.samples.JOLSample_15_IdentityHashCode$A object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                               VALUE
      0     4        (object header)                           01 00 00 00 (00000001 00000000 00000000 00000000) (1)
      4     4        (object header)                           00 00 00 00 (00000000 00000000 00000000 00000000) (0)
      8     4        (object header)                           80 12 01 f8 (10000000 00010010 00000001 11111000) (-134147456)
     12     4        (loss due to the next object alignment)
Instance size: 16 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total

hashCode: 4459eb14

**** After identityHashCode()
org.openjdk.jol.samples.JOLSample_15_IdentityHashCode$A object internals:
 OFFSET  SIZE   TYPE DESCRIPTION                               VALUE
      0     4        (object header)                           01 14 eb 59 (00000001 00010100 11101011 01011001) (1508578305)
      4     4        (object header)                           44 00 00 00 (01000100 00000000 00000000 00000000) (68)
      8     4        (object header)                           80 12 01 f8 (10000000 00010010 00000001 11111000) (-134147456)
     12     4        (loss due to the next object alignment)
Instance size: 16 bytes
Space losses: 0 bytes internal + 4 bytes external = 4 bytes total
```

JMH

Why benchmark?

- Assess performance of a software system or a single component
- Compare performance of different algorithms/implementations
- Investigate impact of configuration and/or environment changes on a performance of your system



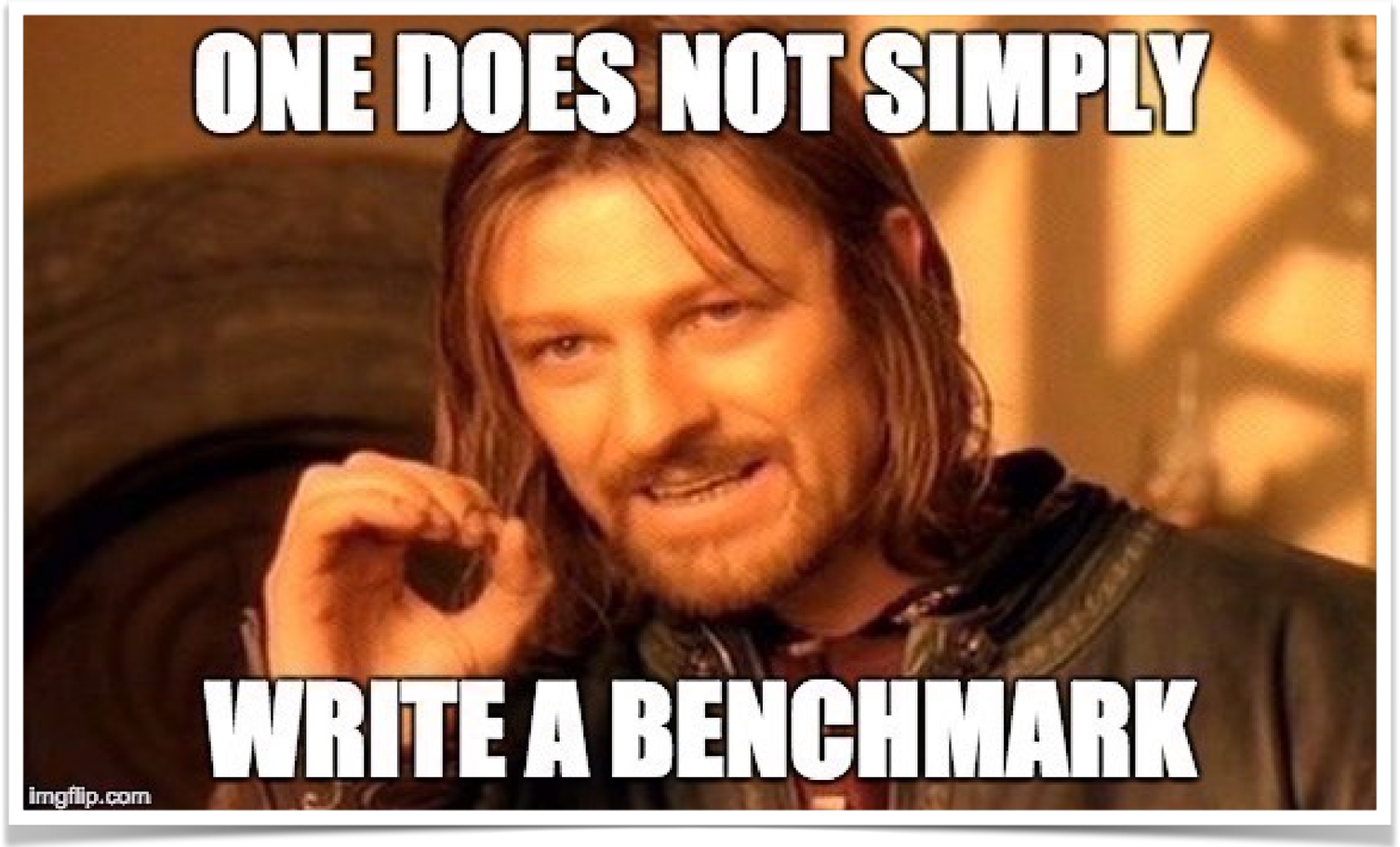
“Without exception every microbenchmark I've seen has had serious flaws”

–Dr. Cliff Click

http://www.azulsystems.com/presentations/art-of-java-benchmarking

```
public class ListGetBenchmarkBroken {
    private static final int ITERATIONS = 1_000_000_000;

    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        long startTime = System.nanoTime();
        for (int i = 0; i < ITERATIONS; i++) {
            list.get(3);
        }
        long endTime = System.nanoTime();
        long duration = endTime - startTime;
        System.out.println("Executed in "
                + TimeUnit.NANOSECONDS.toMillis(duration) + "ms, " +
                +((double) duration / ITERATIONS) + " ns/op");
    }
}
```

Executed in 6ms, 0.006736898 ns/op


ONE DOES NOT SIMPLY
WRITE A BENCHMARK
imgflip.com

JMH

- OpenJDK project: http://openjdk.java.net/projects/code-tools/jmh/
- Java harness for building, running, and analyzing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM
- De-facto standard for writing benchmarks on JVM
- JEP 230: Microbenchmark Suite (*Status: Candidate*)

JMH features

- Benchmark code generation and runtime:
  - Annotation-based benchmarks
  - Parameters and state
  - Support for multi-threaded benchmarks
  - Blackhole, compiler control etc…
- Command line and API for running benchmarks
- Built-in profilers

```
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@State(Scope.Benchmark)
public class ListGetBenchmarkJMH {
    private List<Integer> list;
    private int index;

    @Setup
    public void setUp() {
        list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        index = 3;
    }

    @Benchmark
    public void baseline() {
    }

    @Benchmark
    public Integer get() {
        return list.get(index);
    }
}
```

```
Benchmark                      Mode  Cnt  Score   Error  Units
ListGetBenchmarkJMH.baseline  avgt  200  0.282 ± 0.003  ns/op
ListGetBenchmarkJMH.get       avgt  200  3.985 ± 0.024  ns/op
```

Profilers

```
java -jar target/benchmarks.jar -lprof

Supported profilers:
         cl: Classloader profiling via standard MBeans
       comp: JIT compiler profiling via standard MBeans
         gc: GC profiling via standard MBeans
      hs_cl: HotSpot (tm) classloader profiling via implementation-specific MBeans
    hs_comp: HotSpot (tm) JIT compiler profiling via implementation-specific MBeans
      hs_gc: HotSpot (tm) memory manager (GC) profiling via implementation-specific MBeans
      hs_rt: HotSpot (tm) runtime profiling via implementation-specific MBeans
     hs_thr: HotSpot (tm) threading subsystem via implementation-specific MBeans
     pauses: Pauses profiler
       perf: Linux perf Statistics
    perfasm: Linux perf + PrintAssembly Profiler
   perfnorm: Linux perf statistics, normalized by operation count
 safepoints: Safepoints profiler
      stack: Simple and naive Java stack profiler

Unsupported profilers:
   xperfasm: <none>
```

```
public class JMHSample_36_BranchPrediction {
    @Benchmark
    @OperationsPerInvocation(COUNT)
    public void sorted(Blackhole bh1, Blackhole bh2) {
        for (byte v : sorted) {
            if (v > 0) {
                bh1.consume(v);
            } else {
                bh2.consume(v);
            }
        }
    }

    @Benchmark
    @OperationsPerInvocation(COUNT)
    public void unsorted(Blackhole bh1, Blackhole bh2) {
        for (byte v : unsorted) {
            if (v > 0) {
                bh1.consume(v);
            } else {
                bh2.consume(v);
            }
        }
    }
}
```

	sorted	unsorted	Unit
	2.403 ± 0.008	7.007 ± 0.034	ns/op
CPI	0.266 ± 0.005	0.770 ± 0.004	#/op
L1-dcache-load-misses	0.016 ± 0.002	0.017 ± 0.003	#/op
L1-dcache-loads	10.977 ± 0.263	11.033 ± 0.184	#/op
L1-dcache-stores	4.997 ± 0.114	4.993 ± 0.043	#/op
L1-icache-load-misses	0.001 ± 0.001	0.002 ± 0.004	#/op
LLC-load-misses	$\approx 10^{-5}$	$\approx 10^{-4}$	#/op
LLC-loads	0.001 ± 0.001	0.001 ± 0.001	#/op
LLC-store-misses	$\approx 10^{-5}$	$\approx 10^{-5}$	#/op
LLC-stores	$\approx 10^{-4}$	$\approx 10^{-4}$	#/op
branch-misses	$\approx 10^{-4}$	0.502 ± 0.015	#/op
branches	6.474 ± 0.116	6.503 ± 0.084	#/op
cycles	8.226 ± 0.163	23.881 ± 0.279	#/op
dTLB-load-misses	$\approx 10^{-5}$	$\approx 10^{-5}$	#/op
dTLB-loads	11.007 ± 0.042	10.991 ± 0.128	#/op
dTLB-store-misses	$\approx 10^{-6}$	$\approx 10^{-5}$	#/op
dTLB-stores	5.006 ± 0.029	4.994 ± 0.050	#/op
iTLB-load-misses	$\approx 10^{-5}$	$\approx 10^{-5}$	#/op

HashMap lookup

```
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@State(Scope.Benchmark)
public class LongKeyMap {
    private java.util.concurrent.ConcurrentHashMap<Long, Double> chm;
    private org.jctools.maps.NonBlockingHashMapLong<Double> nbhm;
    private long key;

    @Benchmark public Double get_ConcurrentHashMap() {
        return chm.get(key);
    }

    @Benchmark public Double get_NonBlockingHashMapLong() {
        return nbhm.get(key);
    }
}
```

HashMap lookup

Benchmark	1.7.0_80	1.8.0_144	9+181	Eclipse OpenJ9 (J9VM - cea1ed7)*
LongKeyMap.get_ConcurrentHashMap	11.112 ± 0.107	6.645 ± 0.055	6.685 ± 0.076	59.421 ± 0.999
LongKeyMap.get_NonBlockingHashMapLong	4.562 ± 0.016	4.597 ± 0.078	4.854 ± 0.058	7.747 ± 0.032

* *Using* `-Xgcpolicy:metronome` *(see https://github.com/eclipse/openj9/issues/42)*

HashMap lookup ignore-case

```
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@State(Scope.Benchmark)
public class MapGet {
    @Benchmark public Integer HashMap_toLowerCase() {
        return hashMap.get(key.toLowerCase());
    }

    @Benchmark public Integer TreeMap() {
        return treeMap.get(key);
    }

    @Benchmark public Integer IgnoreCaseMap() {
        Integer val = ignoreCaseMap.get(key);
        if (val == null) {
            val = ignoreCaseMap.get(key.toLowerCase());
            if (val != null) {
                ignoreCaseMap.put(key, val);
            }
        }
        return val;
    }
}
```

HashMap lookup ignore-case

Benchmark	1.7.0_80	1.8.0_144	9+181	Unit
MapGet.HashMap_toLowerCase	53.327 ± 0.516	51.948 ± 0.260	33.133 ± 0.232	ns/op
MapGet.HashMap_toLowerCase: gc.alloc.rate	1049.153 ± 10.037	1076.685 ± 5.367	920.655 ± 6.411	MB/sec
MapGet.HashMap_toLowerCase: gc.alloc.rate.norm	88.000 ± 0.001	88.000 ± 0.001	48.000 ± 0.001	B/op
MapGet.TreeMap	76.531 ± 1.267	68.769 ± 0.645	95.223 ± 1.106	ns/op
MapGet.TreeMap: gc.alloc.rate	0.001 ± 0.001	0.001 ± 0.001	0.001 ± 0.001	MB/sec
MapGet.TreeMap: gc.alloc.rate.norm	≈ 10^{-4}	≈ 10^{-4}	≈ 10^{-4}	B/op
MapGet.IgnoreCaseMap	12.142 ± 0.387	10.688 ± 0.101	11.081 ± 0.174	ns/op
MapGet.IgnoreCaseMap: gc.alloc.rate	0.001 ± 0.001	0.001 ± 0.001	0.001 ± 0.001	MB/sec
MapGet.IgnoreCaseMap: gc.alloc.rate.norm	≈ 10^{-5}	≈ 10^{-5}	≈ 10^{-5}	B/op

JEP 254: Compact Strings

New vs reflection

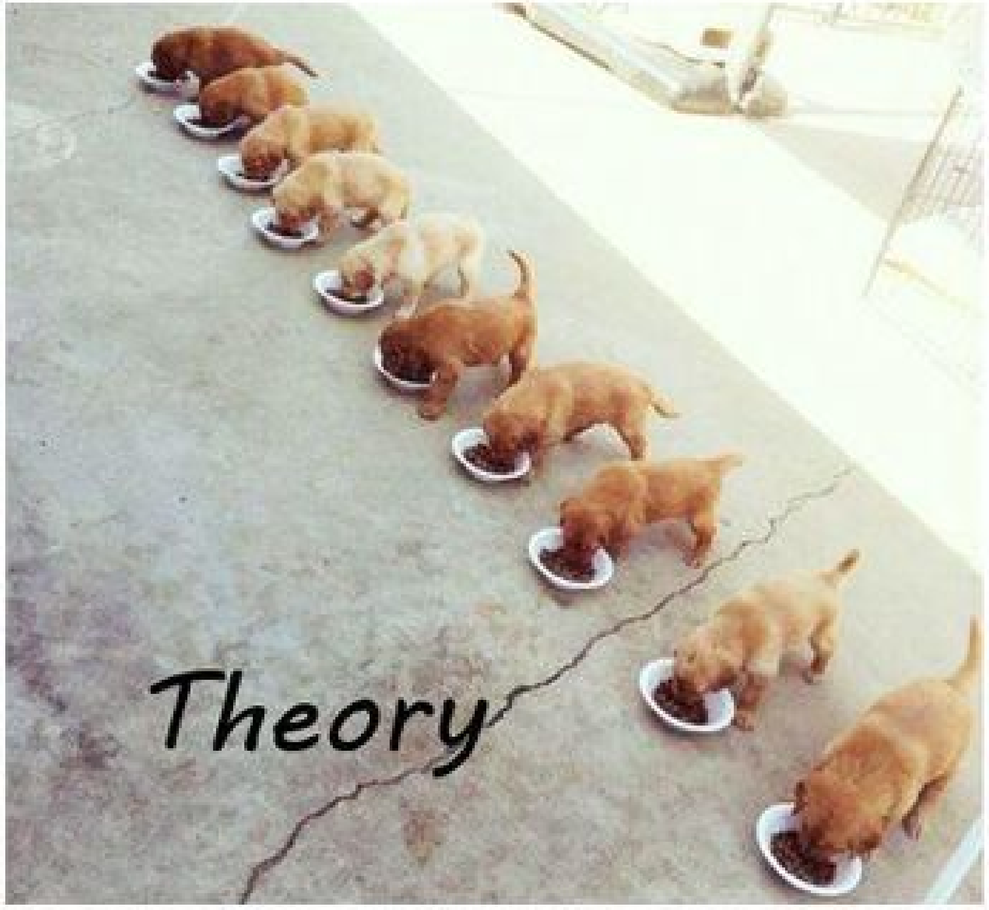
```
public class CreateNewInstance {
    @Benchmark public Object find_constructor_allocate() throws Exception {
        Constructor<?> ctr = klass.getConstructor(Date.class, String.class);
        return ctr.newInstance(arg1, arg2);
    }

    @Benchmark public Object conctructor_allocate() throws Exception {
        return contructor.newInstance(arg1, arg2);
    }

    @Benchmark public Object new_allocate() throws Exception {
        return new C(arg1, arg2);
    }
}
```

New vs reflection

Benchmark	1.7.0_80	1.8.0_144	9+181
CreateNewInstance.find_constructor_allocate	404.127 ± 2.373	86.418 ± 0.360	53.048 ± 0.652
CreateNewInstance.conctructor_allocate	159.547 ± 0.825	18.163 ± 0.164	19.472 ± 0.204
CreateNewInstance.new_allocate	13.713 ± 0.043	12.565 ± 0.192	12.255 ± 0.024

JCStress

Multithreaded programming




CONCURRENCY ISSUES?
I AM THE ONLY THREAD
memegenerator.net

The Java Concurrency Stress tests (`jcstress`) is an experimental harness and a suite of tests to aid the research in the correctness of concurrency support in the JVM, class libraries, and hardware.

http://openjdk.java.net/projects/code-tools/jcstress/

Getting started

- Maven archetype:

```
mvn archetype:generate \
 -DinteractiveMode=false \
 -DarchetypeGroupId=org.openjdk.jcstress \
 -DarchetypeArtifactId=jcstress-java-test-archetype \
 -DarchetypeVersion=0.4 \
 -DgroupId=org.sample \
 -DartifactId=jcstress-test \
 -Dversion=1.0
```

- Or build from source:

```
hg clone http://hg.openjdk.java.net/code-tools/jcstress/ jcstress
```

- Java version: JDK 9 to build/JDK 8 to run

Counter

```
public interface Counter {
    int increment();
}

public class BrokenCounter implements Counter {
    private int value;

    @Override
    public int increment() {
        return ++value;
    }
}

public class AtomicCounter implements Counter {
    private final AtomicInteger value = new AtomicInteger();

    @Override
    public int increment() {
        return value.incrementAndGet();
    }
}
```

Counter

```
public class NaiveCounterTest {
    private static ExecutorService executor = Executors.newFixedThreadPool(2);

    private static void test(Counter counter) throws InterruptedException {
        CountDownLatch startGate = new CountDownLatch(3);
        CountDownLatch endGate = new CountDownLatch(2);
        for (int i = 0; i < 2; i++) {
            executor.submit(() -> {
                startGate.countDown();
                startGate.await();
                System.out.println(counter.increment());
                endGate.countDown();
                return null;
            });
        }
        startGate.countDown();
        endGate.await();
    }

    public static void main(String[] args) throws InterruptedException {
        test(new BrokenCounter());
        test(new AtomicCounter());
    }
}
```

Counter

```
import org.openjdk.jcstress.annotations.*;
import org.openjdk.jcstress.infra.results.II_Result;

import static org.openjdk.jcstress.annotations.Expect.*;

@JCStressTest
@Outcome(id = "1, 2", expect = ACCEPTABLE, desc = "actor1 incremented, then actor2.")
@Outcome(id = "2, 1", expect = ACCEPTABLE, desc = "actor2 incremented, then actor1.")
@Outcome(id = "1, 1", expect = ACCEPTABLE_INTERESTING, desc = "a race on increment")
@State
public class BrokenCounterTest {
    Counter counter = new BrokenCounter();

    @Actor
    public void actor1(II_Result r) {
        r.r1 = counter.increment();
    }

    @Actor
    public void actor2(II_Result r) {
        r.r2 = counter.increment();
    }
}
```

Counter

```
import org.openjdk.jcstress.annotations.*;
import org.openjdk.jcstress.infra.results.II_Result;

import static org.openjdk.jcstress.annotations.Expect.*;

@JCStressTest
@Outcome(id = "1, 2", expect = ACCEPTABLE, desc = "actor1 incremented, then actor2.")
@Outcome(id = "2, 1", expect = ACCEPTABLE, desc = "actor2 incremented, then actor1.")
@Outcome(expect = FORBIDDEN, desc = "Other cases are forbidden.")
@State
public class AtomicCounterTest {
    Counter counter = new AtomicCounter();

    @Actor
    public void actor1(II_Result r) {
        r.r1 = counter.increment();
    }

    @Actor
    public void actor2(II_Result r) {
        r.r2 = counter.increment();
    }
}
```

Counter

```
java -jar target/jcstress.jar -t BrokenCounterTest
```

Test configurations

TC 1 JVM options: [] Iterations: 5 Time: 1000
TC 2 JVM options: [-XX:-TieredCompilation] Iterations: 5 Time: 1000
TC 3 JVM options: [-XX:TieredStopAtLevel=1] Iterations: 5 Time: 1000
TC 4 JVM options: [-Xint] Iterations: 5 Time: 1000

Observed states

Observed state	TC 1	TC 2	TC 3	TC 4	Expectation
1, 1	6965801	7314671	2103154	75921	ACCEPTABLE_INTERESTING
1, 2	41057571	49570071	33009310	3280906	ACCEPTABLE
2, 1	19880616	11678196	19012344	2175711	ACCEPTABLE
	OK	OK	OK	OK	

Counter

```
java -jar target/jcstress.jar -t AtomicCounterTest
```

Test configurations

TC 1 JVM options: [] Iterations: 5 Time: 1000
TC 2 JVM options: [-XX:-TieredCompilation] Iterations: 5 Time: 1000
TC 3 JVM options: [-XX:TieredStopAtLevel=1] Iterations: 5 Time: 1000
TC 4 JVM options: [-Xint] Iterations: 5 Time: 1000

Observed states

Observed state	TC 1	TC 2	TC 3	TC 4	Expectation
1, 2	34333179	51501733	46293126	1863779	ACCEPTABLE
2, 1	13702589	37883185	37296502	1750039	ACCEPTABLE
	OK	OK	OK	OK	

JITWatch

Log analyser / visualiser for Java HotSpot JIT compiler. Inspect inlining decisions, hot methods, bytecode, and assembly. View results in the JavaFX user interface.

https://github.com/AdoptOpenJDK/jitwatch

Getting started

- Build from source:

```
mvn clean compile test exec:java
```

```
gradlew run
```

- Java version: JDK 7+

How to get JIT output

- Basic output

  `-XX:+PrintCompilation`

- JIT log

  `-XX:+UnlockDiagnosticVMOptions -XX:+LogCompilation -XX:LogFile=$LogFilePath`

- Get the assembly

  `-XX:+PrintAssembly`

  Requires the hsdis (HotSpot disassembler) binary to be added to your JRE.

  See https://github.com/AdoptOpenJDK/jitwatch/wiki/Building-hsdis

DEMO

Recap

- **JOL** – toolbox to analyze object layout schemes in JVMs
- **JMH** – Swiss army knife of benchmarking. Gives you powerful tools to write correct benchmarks and investigate performance problems
- **JCStress** – harness for writing concurrency tests and a suit of such tests for JVM (JMM, core libraries, Hotspot, hardware)
- **JITWatch** – log analyser and visualiser for the HotSpot JIT compiler

References

- Code samples: https://github.com/vyazelenko/BaselOne2017
- JOL: http://openjdk.java.net/projects/code-tools/jol/
- JMH: http://openjdk.java.net/projects/code-tools/jmh/
- JCStress: http://openjdk.java.net/projects/code-tools/jcstress/
- JITWatch: https://github.com/AdoptOpenJDK/jitwatch
- Collection of JMH resources by Nitsan Wakart (@nitsanw)
- Aleksey Shipilëv's (@shipilev) blog: shipilev.net/

Questions?

@DVyazelenko