Production profiling with JDK Flight Recorder & JDK Mission Control

Jean-Philippe Bempel, Datadog
@jpbempel

Production Time Profiling and Diagnostics

"The big challenge is no longer really performance. The big challenge is profiling, and especially profiling in production."

- Tony Printezis, JVM engineer, Twitter
(Devoxx 2015, "Life of a Twitter JVM Engineer", 49:49)

DATADOG

Production Profiling and Diagnostics?

- Minimal Observer Effect
  - Low overhead
  - Not affect the behaviour of the application
  - Not undo optimizations

- Safe to use in production
  - Well tested
  - Widely used

Data Flight Recorder for the JVM

- Built into the JVM
- Records information about the JVM and the application
- Low overhead / High performance
- APIs available to record custom information
- Can be used to solve a range of different problems
- Open sourced by Oracle in OpenJDK 11
- Backported to OpenJDK 8 since 8u262/8u272!

Flight Recorder Helps You...

Resolve problems faster

- Data can always be recorded - no need for a reproducer
- Recordings can be captured and shared
- Find real bottlenecks in your applications

Designed for production systems - low observer effect

- Profiling in JFR will not undo optimizations like scalarization
- Profiling in JFR will not be skewed by safe points (like Async Profiler)

Comprehensive Tool Chain

- Control the Flight Recorder
  - Command line parameters
  - POJO API
  - JMX API
  - `jcmd` (tool in the JDK)
  - JDK Mission Control

Comprehensive Tool Chain

- Add custom data
  - Java API (in the JDK)
  - JDK Mission Control (agent)
  - Third party integrations (Open Tracing, Brave etc)

- Use the data
  - JDK Mission Control (application)
  - JDK Mission Control (core libraries)
  - `jfr` (tool in the JDK)

Flight Recorder Data

- Data recorded as events
- Events are data points in time
  - Event types have fields (a.k.a. attributes)
  - Fields are self describing (typed / content types e.g. long / epoch ms)

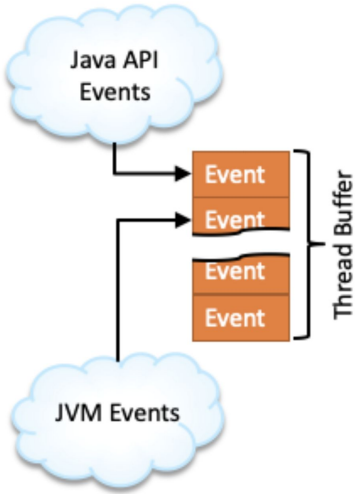
Different Kinds of Data

- CPU Profiling
- Allocation Profiling
- Thread Latency Profiling
- GC
- Compiler
- Memory Leak Profiler
- File & Socket IOs
- ...and much more

Size is typically about 2 megs per minute (app. 100k events)

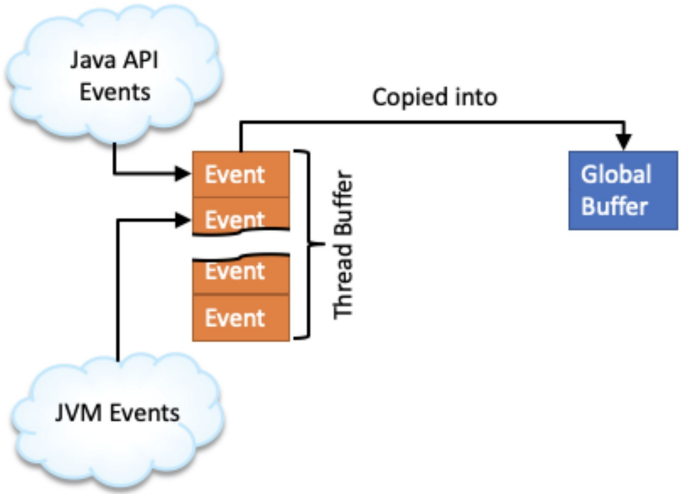
Flight Recorder Innards

- High performance recording engine
- Events recorded into thread buffers



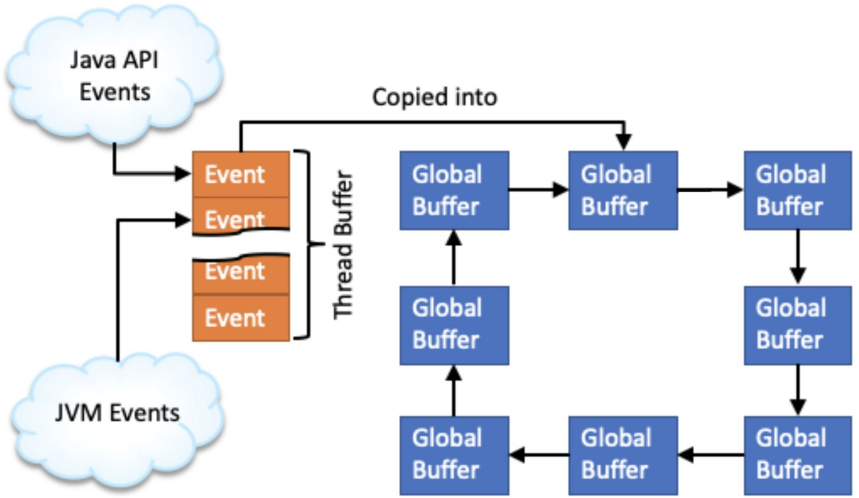
Flight Recorder Innards

- High performance recording engine
- Events recorded into thread buffers
- When full, copied into global buffer



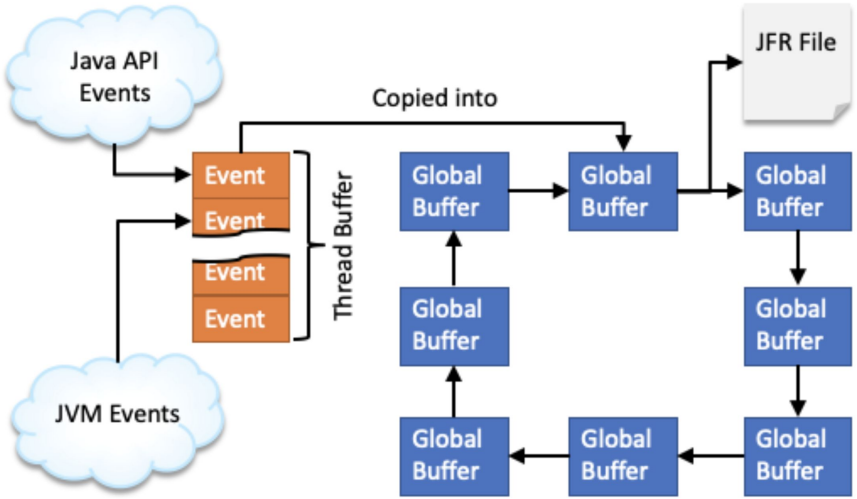
Flight Recorder Innards

- High performance recording engine
- Events recorded into thread buffers
- When full, copied into global buffer
- Can be configured to keep on overwriting/reusing the buffer



Flight Recorder Innards

- High performance recording engine
- Events recorded into thread buffers
- When full, copied into global buffer
- Can be configured to keep on overwriting/reusing the buffer
- …or emit to disk



Adding Custom Events

- Easy to correlate with events from the runtime
  - E.g. record a distributed tracing span, with trace id, span id and parent span id, then see what happened during the processing of that particular span
- Piggy back on the whole JFR infrastructure
  (jcmd, jmc, command line flags)
- High performance
  - High precision, cheap timestamping
  - Cheap stack traces
  - Binary, compact data
- Self describing, easy to consume

Examples: github.com/thegreystone/java-svc/tree/master/jfr

Simple JFR Event Generation Example

```
public class HelloJfr {
    @Label("Hello World")
    static class HelloWorldEvent extends Event {
        @Label("Message")
        String message;
    }
    public static void main(String [] args) {
        HelloWorldEvent event = new HelloWorldEvent();
        event.message = "Hello World!";
        event.commit();
    }
}
```

Metadata Example

```
@Label("Native Library Load")
@Name("org.example.process.NativeLibraryLoad")
@Description("Emitted on the loading of a native library")
@Category("Process")
public final class LibraryLoadEvent extends Event {
        @Label("File Name")
        String fileName;

        @Label("Start Address")
        @MemoryAddress
        long startAddress;

        @Label("Bytes Loaded")
        @DataAmount(DataAmount.BYTES)
        long bytesLoaded;

        @Label("Library file creation time")
        @Timestamp(Timestamp.MILLISECONDS_SINCE_EPOCH)
        long creationTime;
}
```

Controlling the Flight Recorder

- JDK Mission Control
  - Via JMX
- JCMD
  - Command line tool to talk to running JVMs
- Command line flags
  - `-XX:StartFlightRecording=delay=20s,duration=60s,name=MyRecording,filename=/tmp/myrecording.jfr,settings=profile`
- Programmatically
  - JMX API
  - Pojo API

Flight Recorder Templates (.jfc files)

Contains information about what and how to record, e.g.:

- What event types to enable
- What thresholds to use for events with durations
- What periodicity to sample requestable events

There are two templates by default:

- default.jfc – less than 1% overhead
- profiling.jfc – less than 2% overhead

Templates are located in the lib/jfr folder of the JDK.

Templates can be edited and exported from JMC.

Looking at Flight Recordings

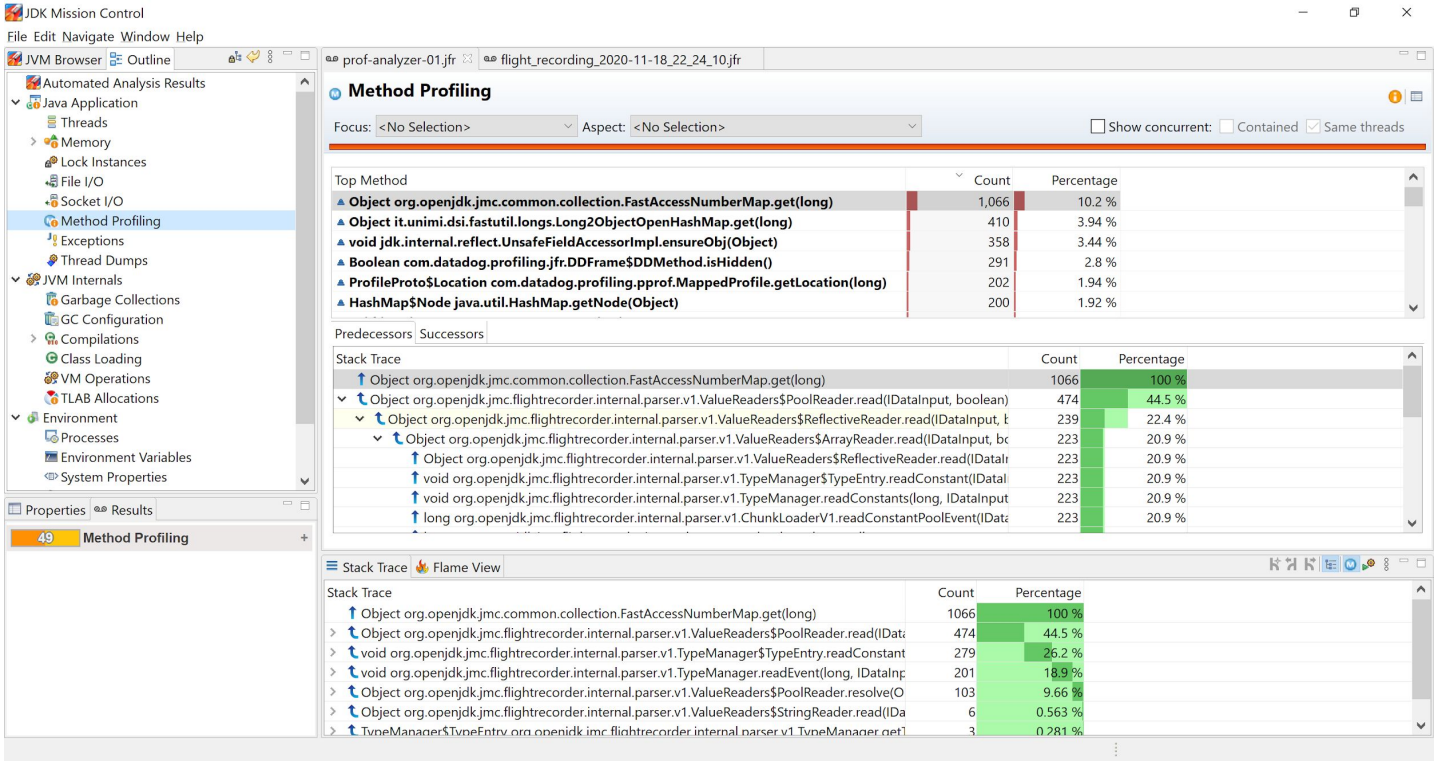
- JDK `jfr` tool
  - Simple command-line tool to look at recordings
- JDK parser
  - Supports recordings with the JDK version the recording was created
  - Included in the JDK
  - External iteration

Looking at Flight Recordings

- JMC core parser
  - Supports recordings from all versions of JFR
  - Compiles and runs on JDK 8+
  - Analysis Rules
  - Declarative / Internal iteration
  - Available as Maven artifacts

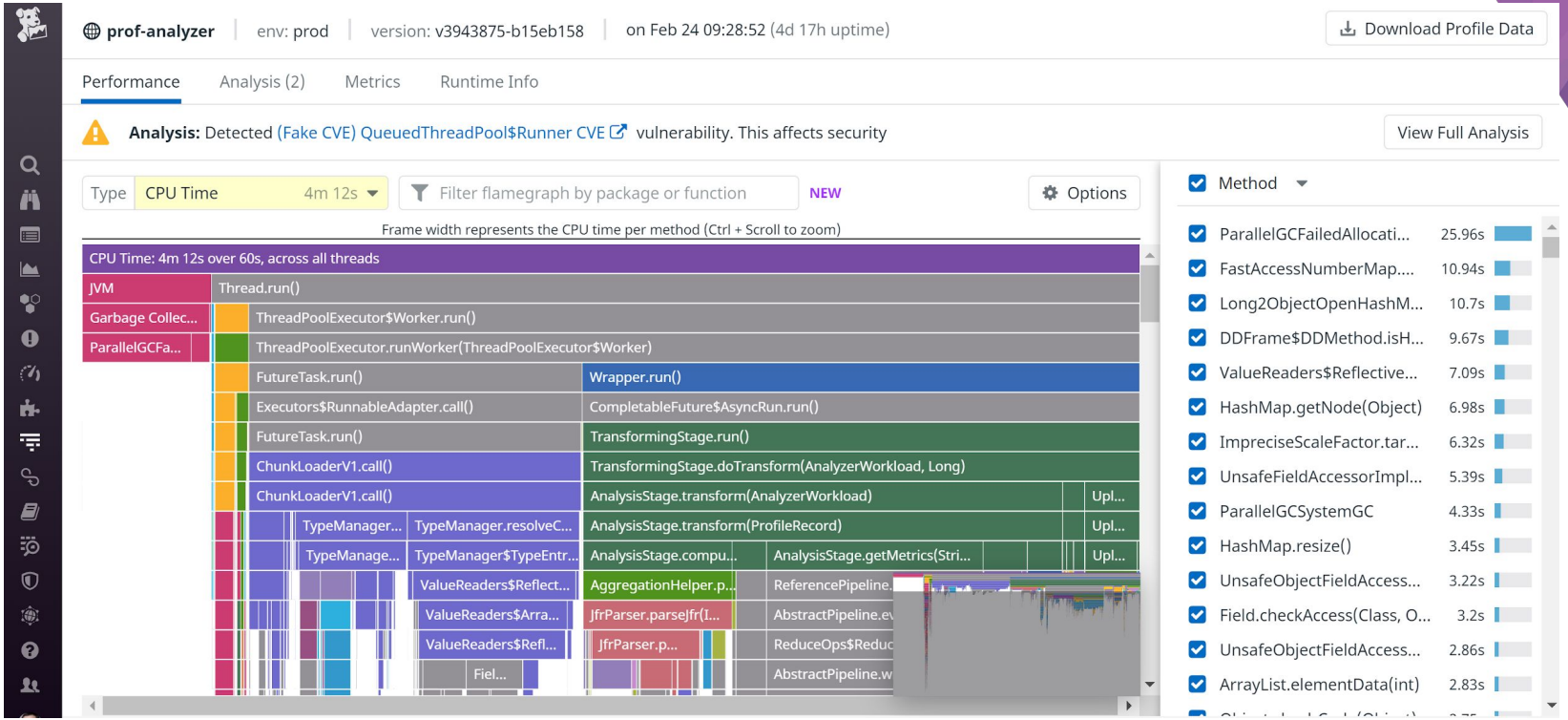
Looking at Flight Recordings

- JMC



Looking at Flight Recordings

- Datadog Profiling!



The profiler will always lie to you...

- Every profiler in existence will lie to you
  - Change runtime behaviour (e.g. undo optimizations)
  - Biases (safe point bias)

- Talk: Profilers Are Lying Hobbits (and we hate them!) from Nitsan Wakart

The profiler will always lie to you...

- Know your profiler and it’s chosen trade-offs
  - JFR Execution Samples (CPU profiling) choses low runtime impact and constant overhead. Will _not_ include native samples.
  - JFR allocation profiling is sampling for nursery allocations - not exact counts but a good estimation, especially over time.
  - JFR provides contextual information, e.g. for locks (monitor enter):
    - Thread monitor class
    - Which thread was holding the monitor
    - Monitor address
    - …

    There is always a price, in this case thresholds...

JFR Demos

JDK Mission Control

JDK Mission Control - JMC

Tools suite

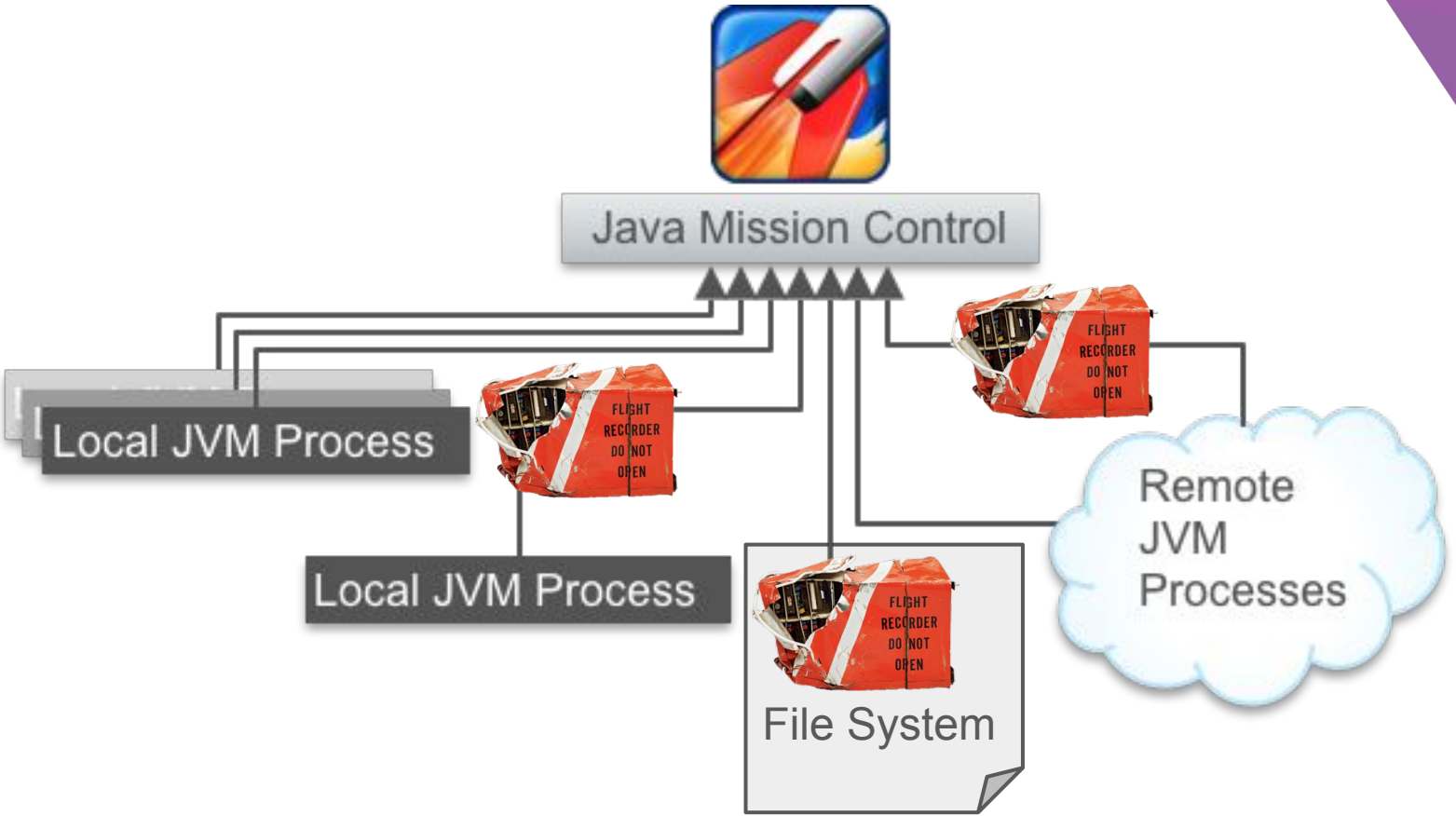
- JFR
  - Create
  - Analyze
- JMX Console
  - Real time monitoring
- Additional plug-ins
  - JOverflow
  - Moar JFR
  - Plug-in plug-in

Open Source

- OpenJDK project
- Actively worked on by Oracle, RedHat, DataDog, individual contributors…

- Several distributions of the application… and Eclipse plug-ins

Where to get Flight Recordings



Mission Control + Flight Recorder UI

- JFR Wizard for creating recordings

- Rules Overview
- Prebuilt pages
  - Method Profiling
  - Locks
  - Memory
  - ...
- Event Browser
  - All the events
- Custom pages
  - For JDK events or your custom events

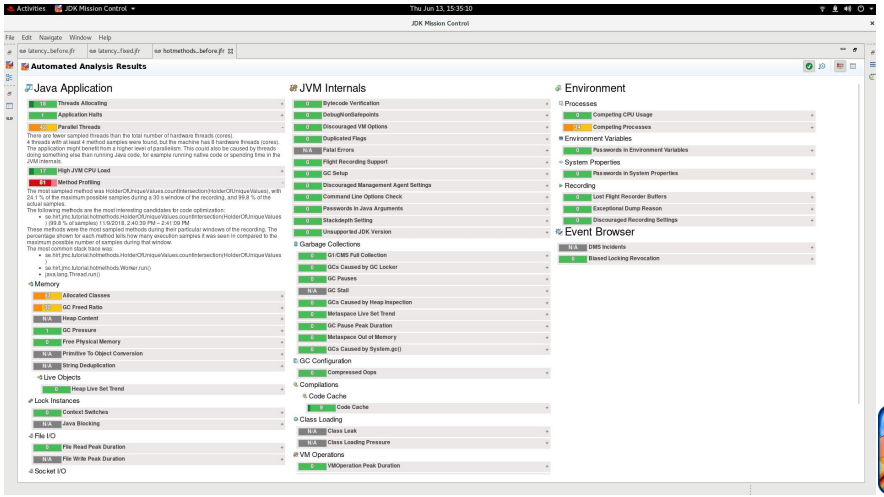
JMC Demos

Extensive API

```
public static void main(String[] args) throws Exception {
        var file = new File(args[0]);

        IItemCollection events = JfrLoaderToolkit.loadEvents(file);
        IItemCollection monitorEnterEvents = events.apply(JdkFilters.MONITOR_ENTER);

        IQuantity eventCount = monitorEnterEvents.getAggregate(Aggregators.count());
        IQuantity avg = monitorEnterEvents.getAggregate(Aggregators.avg((JfrAttributes.DURATION));
        IQuantity stddev = monitorEnterEvents.getAggregate(Aggregators.stddev(JfrAttributes.DURATION));

        System.out.println(String.format("# of events: %d, avg: %s, stdddev: %s\n",
                                eventCount.longValue(),
                                avg.displayUsing(IDisplayable.AUTO),
                                stddev.displayUsing(IDisplayable.AUTO)));
}
```

HTML Report Example

```
public static void main(String[] args) throws Exception {
    var file = new File(args[0]);
    var recording = JfrLoaderToolkit.loadEvents(file);
    var report = JfrHtmlRulesReport.createReport(recording);

    System.out.println(report);
}
```



JMC core Demos

JMC Agent

- Incubation project in JMC
  - Not published (yet)
  - Build from source
- Declaratively insert JFR events anywhere

```
<event id="demo.jfr.MyEvent" >
      <name>My Awesome Event</ name>
      <description>This is the best event ever.</ description>
      <path>demo/jfr</path>
      <stacktrace>true</stacktrace>
      <class>org.openjdk.jmc.agent.test.InstrumentMe</ class>
      <method>
            <name>myInstrumentedMethod</ name>
            <descriptor>(Lorg/openjdk/jmc/bciagent/test/Gurka;)V</ descriptor>
            <parameter index="0">
                  <name>Gurka Attribute</ name>
                  <description>The one and only Gurk-parameter</ description>
                  <contenttype>None</contenttype>
            </parameter>
      </method>
</event>
```

Summary

Resources

Join the project mailing list and start coding right now!

Repo:	https://github.com/openjdk/jmc
Mailing list:	http://mail.openjdk.java.net/mailman/listinfo/jmc-dev
Slack:	https://jdkmissioncontrol.slack.com
Download JMC:	Your distribution of choice (e.g. AdoptOpenJDK)
JMC Tutorial:	https://github.com/thegreystone/jmc-tutorial

Q & A

@jpbempel