



Programmer's Guide to the JDK Flight Recorder

Joakim Nordström

Sustaining Engineer

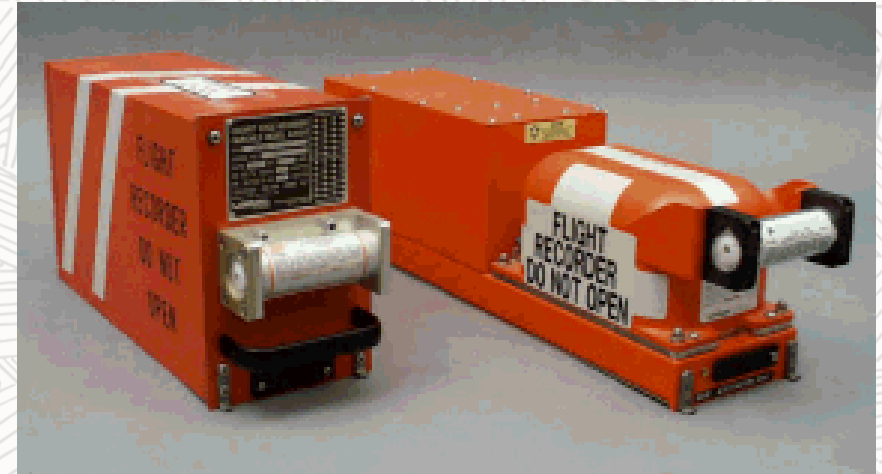
Java Product Group, Oracle

February 2023



JDK Flight Recorder

- Records events from the JVM, the JDK and applications
 - unified way to inspect the entire software stack
 - correlate events across different software layers and components
- Part of the JVM
- Low overhead
- Designed to be "always on"



Agenda

1

**Start/stop
recordings**



2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

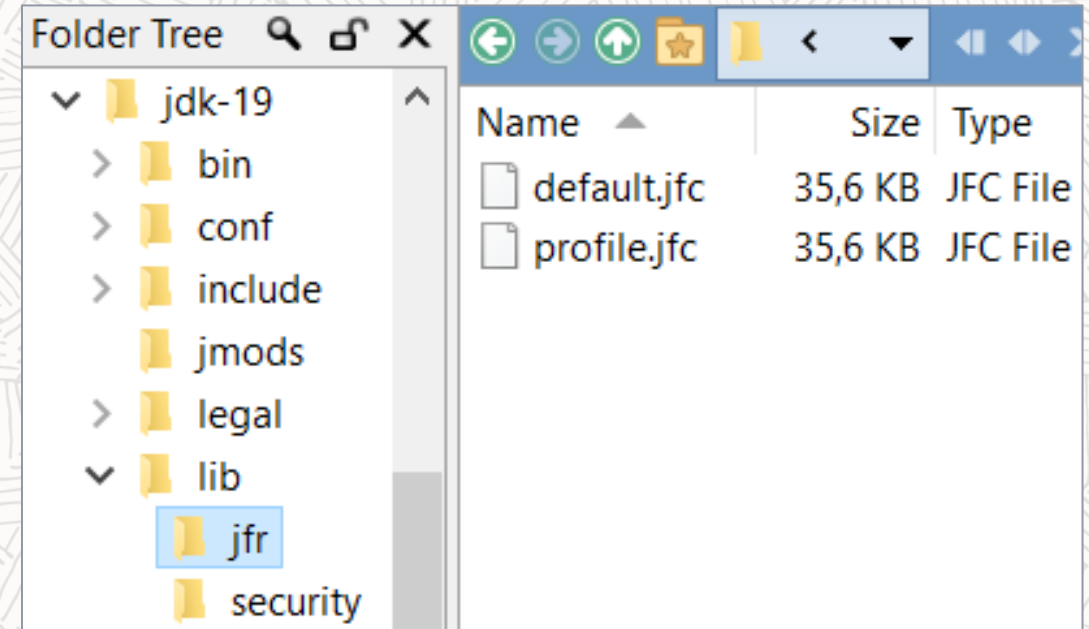
**Consuming
events**

5

**Using events
in unit-test**

Designed to be always on

- Two profiles shipped with the JVM
 - **default.jfc**
 - less than 1% overhead
 - designed for continuous recording, "always on"
 - **profile.jfc**
 - can have slightly more overhead (<2%)
 - slightly changed thresholds
 - a few more stack traces



Tip: don't alter these files!

- Make copies if needed
 - Use `jfr` commandline tool!
- Store custom settings together with app settings

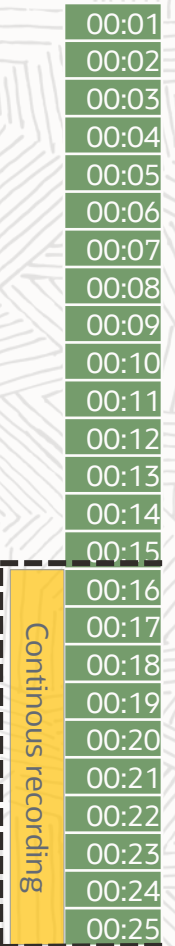
Designed to be always on

- Always start your JVM with Flight Recording!
 - Default profile has less than 1 % overhead

```
java -XX:StartFlightRecording:dumponexit=true,\
      filename=/var/log/rec_%p_%t.jfr
```

- **dumponexit=true**
 - will write JFR on JVM exit
- default name: **hotspot_<pid>_<timestamp>.jfr**
 - **filename** can be
 - directory
 - filename
 - **%p** for Process ID
 - **%t** for timestamp

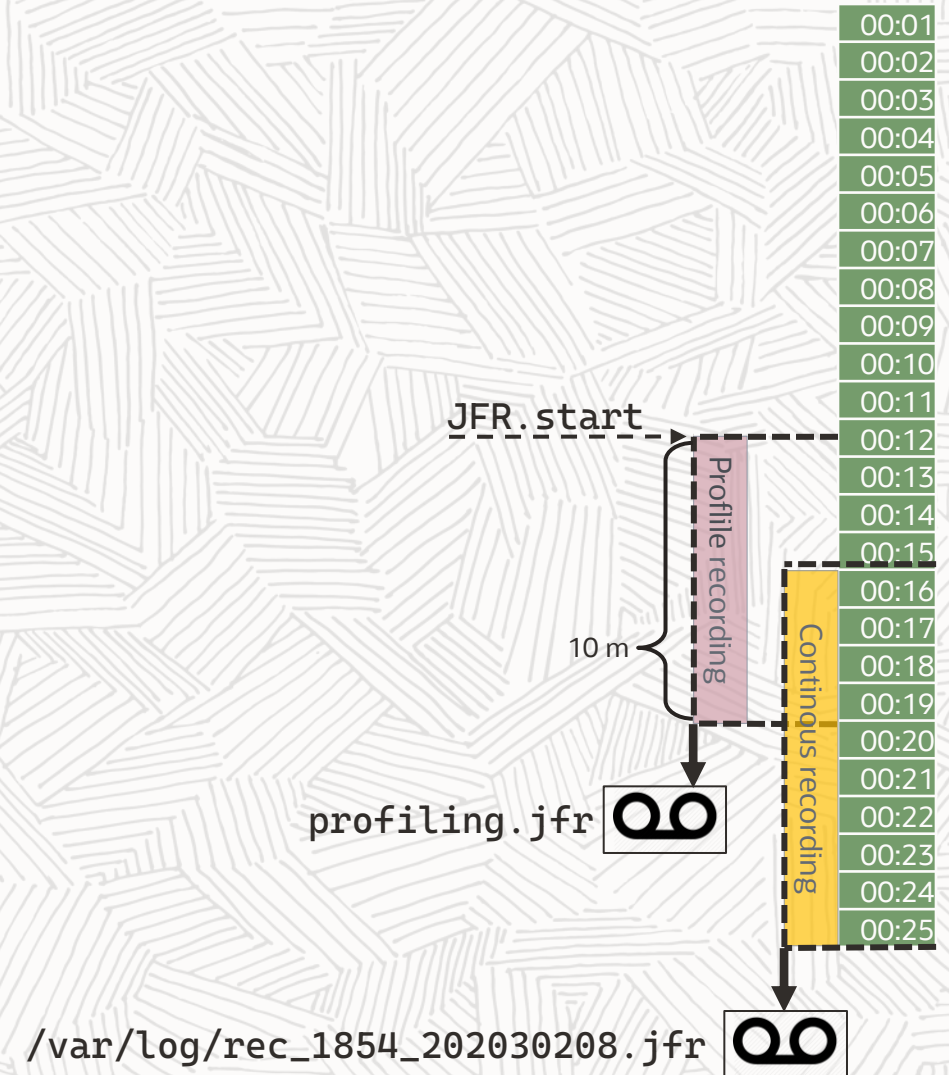
/var/log/rec_1854_202030208.jfr



Start a recording during runtime

- Start recording using the “profile” setting:

```
$ jcmd <pid> JFR.start settings=profile \  
    filename=profiling.jfr \  
    duration=10m
```



Using JCMD to control JFR recordings

- Start recording

```
$ jcmd <pid> JFR.start
```

- Write recording to disk

```
$ jcmd <pid> JFR.dump
```

- Stop recording

```
$ jcmd <pid> JFR.stop
```

- Check active recordings

```
$ jcmd <pid> JFR.check
```

30496:

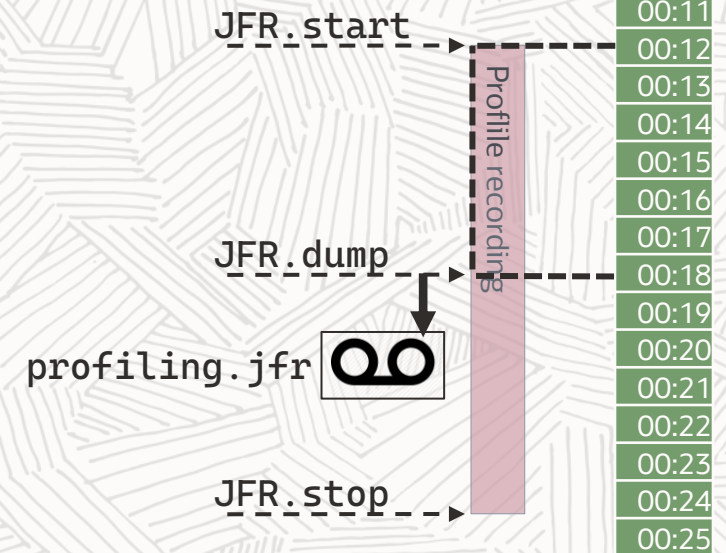
Recording 1: name=1 maxsize=250,0MB (running)

List all running JVMs pids:

```
$ jcmd
```

List all jcmd commands:

```
$ jcmd <pid> help
```



Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

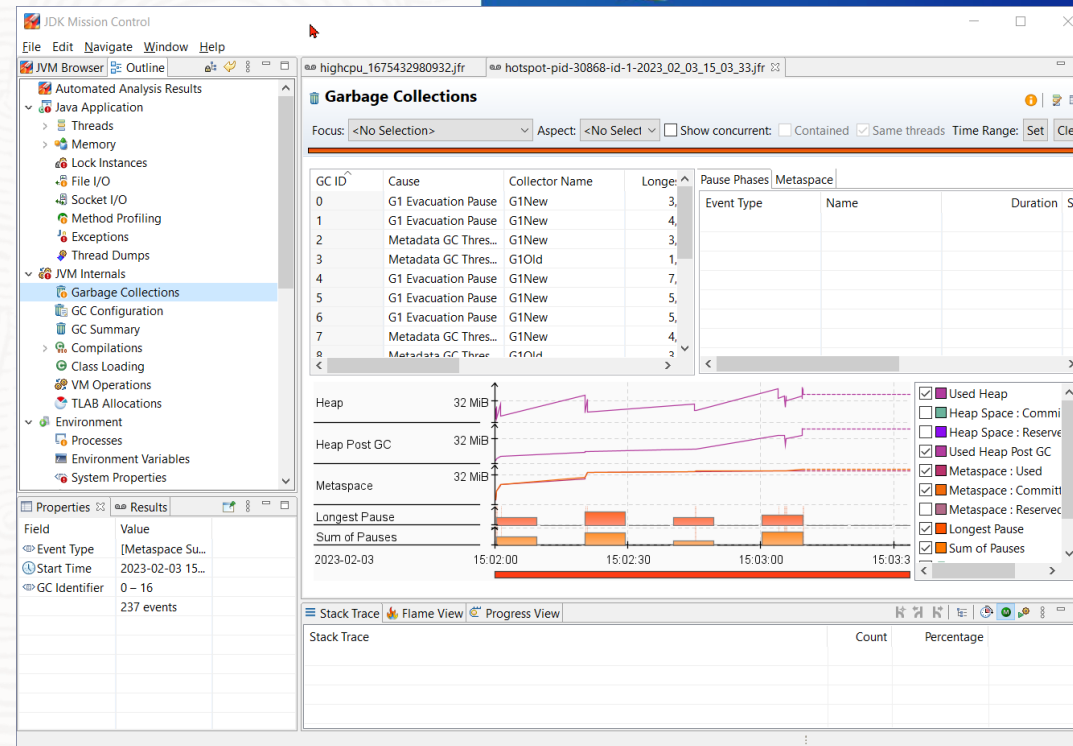
**Using events
in unit-test**



Examining the JFR recording

JDK Mission Control 8, “JMC”

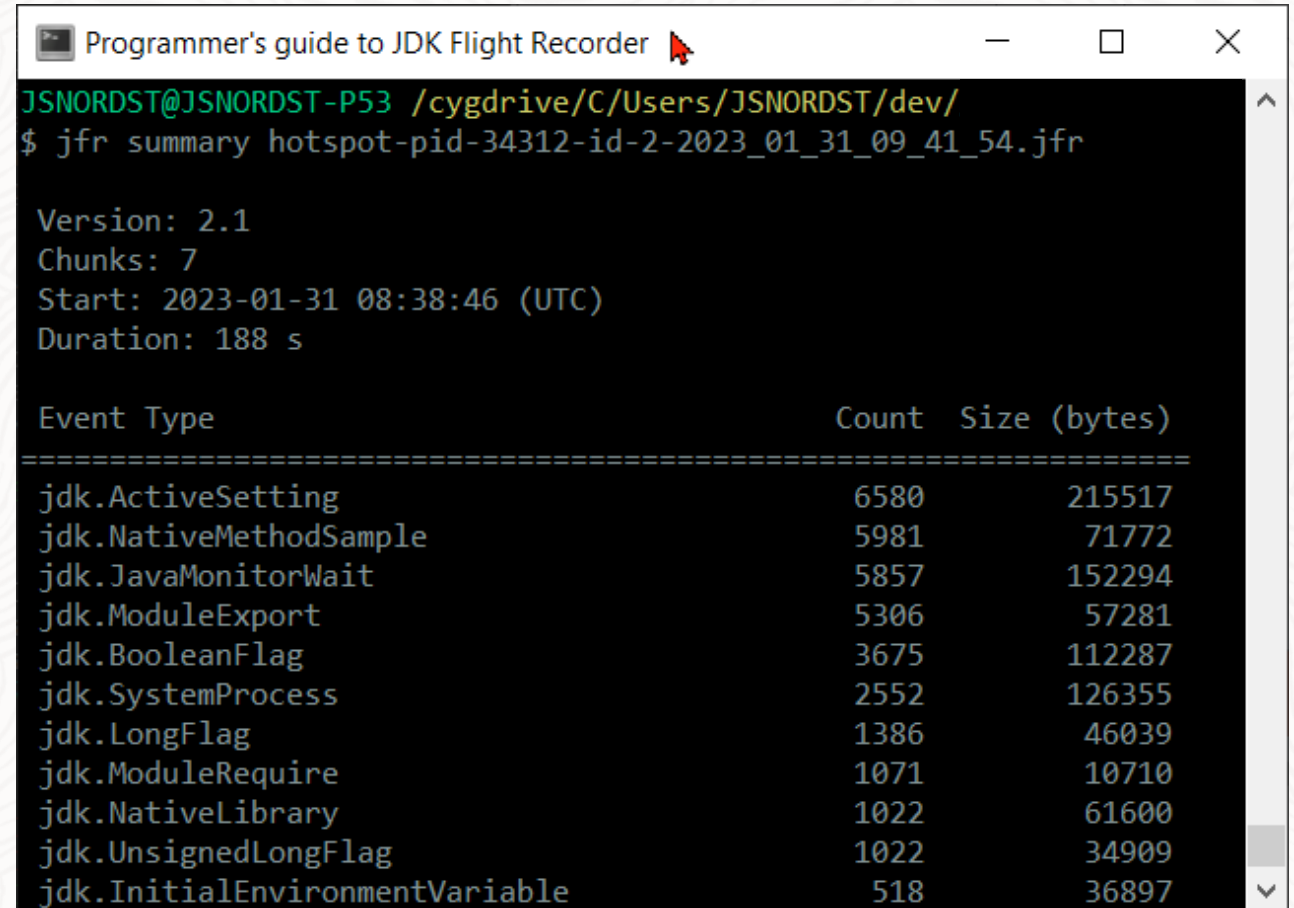
- Graphical interface to analyze recordings
- Separate download:
oracle.com/java/technologies/jdk-mission-control.html
- Eclipse Plugins



Examining the JFR recording

JFR commandline tool

- Available in the JDK
- List events, output as text/JSON
- Create .jfc configuration files
- Disassemble chunks from recording
 - If you have very large JFR recording
- Assemble leftover JFR chunks
- Scrub events from recording



```
Programmer's guide to JDK Flight Recorder
JSNORDST@JSNORDST-P53 /cygdrive/C/Users/JSNORDST/dev/
$ jfr summary hotspot-pid-34312-id-2-2023_01_31_09_41_54.jfr

Version: 2.1
Chunks: 7
Start: 2023-01-31 08:38:46 (UTC)
Duration: 188 s

Event Type                                     Count  Size (bytes)
=====
jdk.ActiveSetting                             6580    215517
jdk.NativeMethodSample                       5981     71772
jdk.JavaMonitorWait                          5857   152294
jdk.ModuleExport                             5306     57281
jdk.BooleanFlag                              3675    112287
jdk.SystemProcess                            2552   126355
jdk.LongFlag                                 1386     46039
jdk.ModuleRequire                             1071     10710
jdk.NativeLibrary                             1022     61600
jdk.UnsignedLongFlag                          1022     34909
jdk.InitialEnvironmentVariable                 518     36897
```

Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

**Using events
in unit-test**



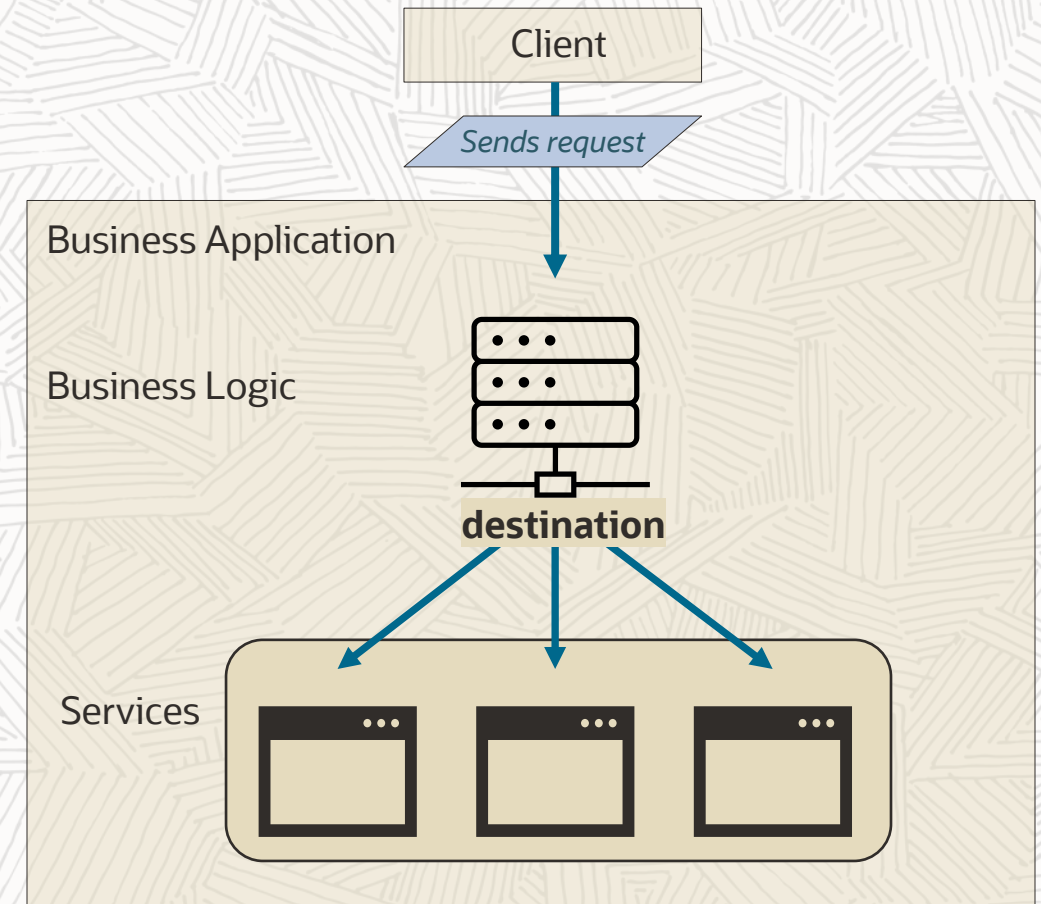
Our Business Application -- Inside the Java Dumpster

Mimics a real world system

- Server application
 - (Jetty, Micronaut, Tomcat, WebLogic, etc)
- JDK 19

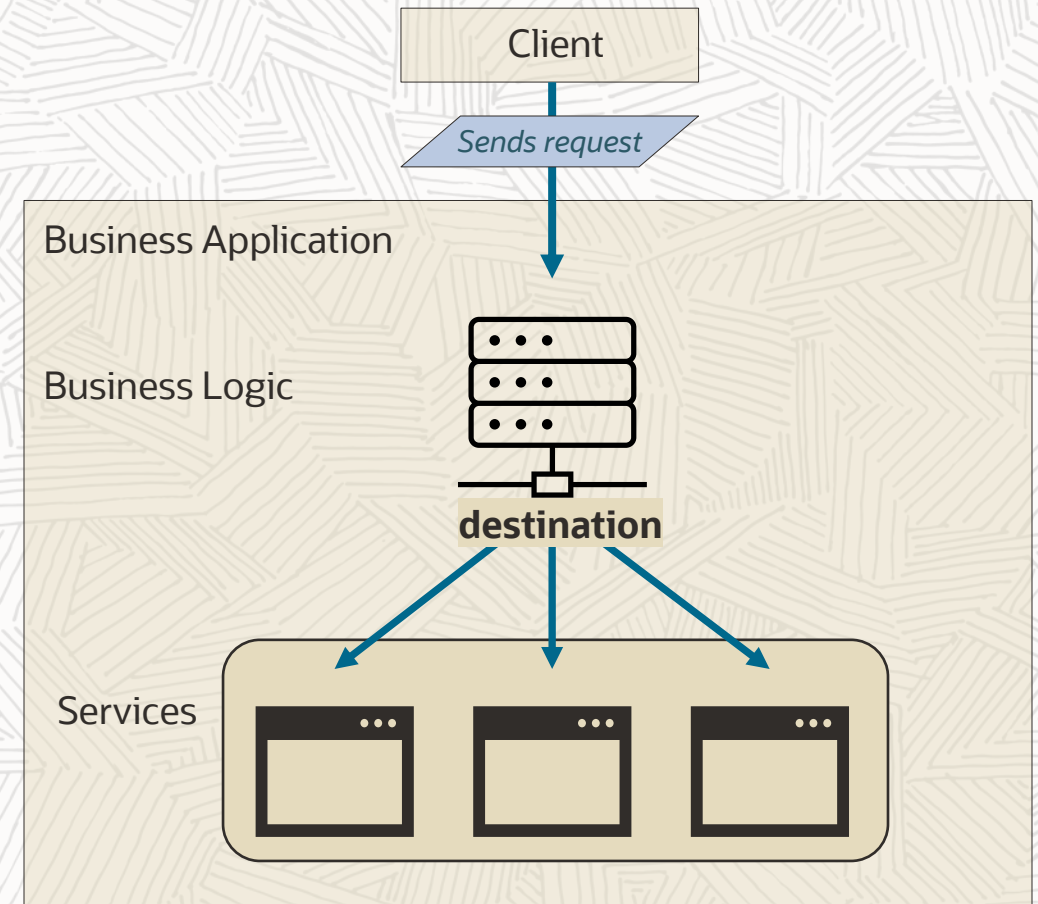
Client sends request to Business Application

Business Logic decides based on **destination**
which service should handle the request



Our Business Application -- Inside the Java Dumpster

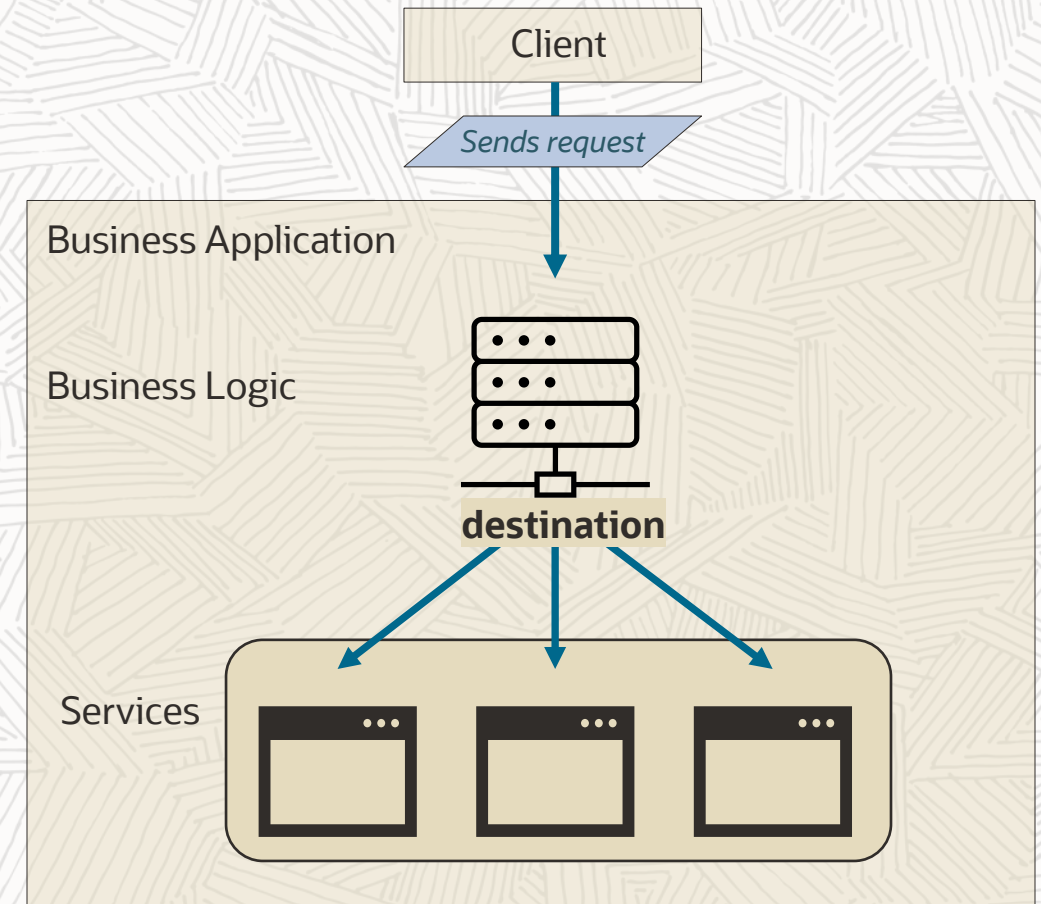
```
Service lookupService(Destination destination) {  
  
    switch(destination) {  
        case Comp3:  
            return new UploadImageService();  
  
        case Comp4:  
            return new EarnCreditsService();  
  
        ...  
    }  
}
```



Our Business Application -- Inside the Java Dumpster

Requirements

- Monitor service calls
- Catch unhandled destinations



```
package inside.dumpster.monitoring.event;
```

```
public class ServiceCallEvent {
```

```
    public String destination;
```

```
    public Class serviceImplClass;  
}
```

```

package inside.dumpster.monitoring.event;

@Name("inside.dumpster.ServiceCall")
@Label("Service Call")
@Category({"Business Application", "Services"})
public class ServiceCallEvent extends jdk.jfr.Event {
    @Label("Service Destination")
    public String destination;

    @Name("serviceClass")
    @Label("Service Implementation Class")
    public Class serviceImplClass;
}

```

@Name

- unique
- use reverse domain-name notation
 - com.acme.Application

@Label

- capitalized
- human-readable

@Name (for a field)

- start with lowerCase

@Category

- capitalized
- descriptive
- unique
- create a logic hierarchy

```
public BusinessLogicService lookupService(Destination destination) {  
    final BusinessLogicService service;  
  
    switch(destination) {  
        case Comp3:  
            service = new UploadImageService();  
            break;  
        ...  
    }  
  
    return service;  
}
```

```
public BusinessLogicService lookupService(Destination destination) {  
    final BusinessLogicService service;  
  
    ServiceCallEvent serviceCallEvent = new ServiceCallEvent();  
    serviceCallEvent.destination = destination.name();  
    serviceCallEvent.begin();  
    switch(destination) {  
        case Comp3:  
            service = new UploadImageService();  
            break;  
        ...  
    }  
    serviceCallEvent.serviceImplClass = service.getClass();  
    serviceCallEvent.commit();  
  
    return service;  
}
```

```
public BusinessLogicService lookupService(Destination destination) {  
    final BusinessLogicService service;  
  
    ServiceCallEvent serviceCallEvent = new ServiceCallEvent();  
    serviceCallEvent.destination = destination.name();  
    serviceCallEvent.begin();  
    switch(destination) {  
        case Comp3:  
            service = new UploadImageService();  
            break;  
        ...  
        case Unknown:  
        default:  
            UnhandledServiceCallEvent unhandledServiceEvent = new UnhandledServiceCallEvent();  
            unhandledServiceEvent.destination = destination.name();  
            unhandledServiceEvent.commit();  
  
            return null;  
        }  
    serviceCallEvent.serviceImplClass = service.getClass();  
    serviceCallEvent.commit();  
  
    return service;  
}
```

```
public BusinessLogicService lookupService(Destination destination) {  
    final BusinessLogicService service;  
  
    ServiceCallEvent serviceCallEvent = new ServiceCallEvent();  
    serviceCallEvent.destination = destination.name();  
    serviceCallEvent.begin();  
    switch(destination) {  
        case Comp3:  
            service = new UploadImageService();  
            break;  
        ...  
        case Unknown:  
        default:  
            UnhandledServiceCallEvent unhandledServiceEvent = new UnhandledServiceCallEvent();  
            unhandledServiceEvent.destination = destination.name();  
            unhandledServiceEvent.commit();  
  
            return null; ←  
    }  
    serviceCallEvent.serviceImplClass = service.getClass();  
    serviceCallEvent.commit();  
  
    return service;  
}
```

Other metadata

```
@Label("Bytes Read")
@Category({"Backend", "IO"})
@Enabled(true )
public class BytesRead extends Event {
    @Label("Bytes Read")
    @DataAmount
    public long bytes;
}
```

Event used in tight loop:

```
for (int pos : backend.getNext()) {
    var event = new BytesRead();
    event.begin();
    byte b[] = backend.read(pos);
    event.end();
    event.bytes = b.length();
    event.commit();
}
```

Other metadata

```
@Label("Bytes Read")
@Category({"Backend", "IO"})
@Enabled(false)
public class BytesRead extends Event {
    @Label("Bytes Read")
    @DataAmount
    public long bytes;
}
```

Event used in tight loop:

```
for (int pos : backend.getNext()) {
    var event = new BytesRead();
    event.begin();
    byte b[] = backend.read(pos);
    event.end();
    event.bytes = b.length();
    event.commit();
}
```

Other metadata

```
@Label("Bytes Read")
@Category({"Backend", "IO"})
@Enabled(false)
public class BytesRead extends Event {
    @Label("Bytes Read")
    @DataAmount
    public long bytes;
}
```

Some other annotations:
@MemoryAddress
@Frequency
@Percentage

Event used in tight loop:

```
for (int pos : backend.getNext()) {
    var event = new BytesRead();
    event.begin();
    byte b[] = backend.read(pos);
    event.end();
    event.bytes = b.length();
    event.commit();
}
```

Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

**Using events
in unit-test**



Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

**Using events
in unit-test**



JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - Compilations
 - Class Loading
 - VM Operations
 - TLAB Allocations
- Environment
 - Processes
 - Environment Variables
 - System Properties
 - Native Libraries
 - Recording
- Event Browser

diagnosis_2022_10_11_10_02_AM.jfr

Event Browser

<No Selection> Aspect: <No Selection> ☐ Show concurrent: ☐ Contained ☒ Same threads

Event Types Tree

Search the tree

- Business Application 525
 - Data 155
 - Services 370
 - Invocation 181
 - Service Call 185
 - Unhandled Service Call 4
 - Flight Recorder 1 725
 - Java Application 3 731
 - Java Development Kit 0
 - Java Virtual Machine 8 271

Start Time	Event Type
2022-10-11 10:02:36	Object Allocation Sample
2022-10-11 10:03:00	Object Allocation Sample
2022-10-11 10:03:05	Object Allocation Sample

Properties Results

0 Biased Locking Revocation +

N/A DMS Incidents +

Stack Trace

Stack Trace

- void java.lang.Thread
- void io.netty.util.conc
- void io.netty.util.inter
- void io.netty.util.concurrent.SingleThreadEventExecutor\$4.run():986
- void io.netty.channel.nio.NioEventLoop.run():460
- int io.netty.channel.nio.NioEventLoop.select(long):813
- int io.netty.channel.nio.SelectedSelectionKeySetSelector.select():68
- int sun.nio.ch.SelectorImpl.select():146
- int sun.nio.ch.SelectorImpl.lockAndDoSelect(Consumer, long):129
- int sun.nio.ch.WEPollSelectorImpl.doSelect(Consumer, long):111
- int sun.nio.ch.WEPoll.wait(long, long, int, int):-1

3215
2300
1967
1967
1967
1967
1967
1967
1967



JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - Compilations
 - Class Loading
 - VM Operations
 - TLAB Allocations
- Environment
 - Processes
 - Environment Variables
 - System Properties
 - Native Libraries
 - Recording
 - Event Browser

diagnosis_2022_10_11_10_02_AM.jfr

Event Browser

<No Selection> Aspect: <No Selection> ☐ Show concurrent: ☐ Contained ☒ Same threads

Event Types Tree

Search the tree

- > Business Application 525
 - > Data 155
 - > Services 370
 - > Invocation 181
 - Service Call 485
 - Unhandled Service Call 4
- > Flight Recorder 1 725
- > Java Application
- > Java Development
- > Java Virtual Machine
- > Operating System

Start Time	Event Type
2022-10-11 10:02:36	Object Allocation Sample
2022-10-11 10:03:00	Object Allocation Sample
2022-10-11 10:03:05	Object Allocation Sample

```
package inside.dumpster.monitoring.event;  
  
@Category({"Business Application", "Services"})  
@Name("inside.dumpster.ServiceCall")  
@Label("Service Call")  
public class ServiceCallEvent extends Event {  
    @Label("Service Destination")  
    public String destination;  
}
```

1 725
on 3 731
ent Kit 0
chine 8 271

3215
2300
1967
1967
1967
1967
1967
1967

JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - Compilations
 - Class Loading
 - VM Operations

diagnosis_2022_10_11_10_02_AM.jfr

Filtered Events

4: File I/O Histogram Selection Aspect: Selected events (5 File Write) ☒ Show concurrent: ☐ Contained ☒ Same threads Time Range: Set Clear

OR

- Type is inside.dumpster.ServiceInvocation
- Type is inside.dumpster.UnhandledServiceCall
- Type is jdk.FileWrite
- Type is inside.dumpster.ProcessData

List Chart

Start Time	Event Type	Duration	ServiceClass	Size of up...	Source Device of ...	Event Thread
2023-02-03 11:23:31	Data Upload	36,119 s		581 KiB	Comp955877	default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	Service Invoked	36,119 s	inside...			default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	File Write	255,627 ms	inside.dumpster.uploadtext.UploadTextService			default-nioEventLoopGroup-1-17

Start Time	Event Type	Duration	ServiceClass	Size of up...	Source Device of ...	Event Thread
2023-02-03 11:23:31	Data Upload	36,119 s		581 KiB	Comp955877	default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	Service Invoked	36,119 s	inside...			default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	File Write	255,627 ms	inside.dumpster.uploadtext.UploadTextService			default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	File Write	16,755 ms				default-nioEventLoopGroup-1-17

Properties Results

0 Biased Locking Revocation +

N/A DMS Incidents +

Stack Trace

Stack Trace	Count
Object io.micronaut.web.router.RouteMatch.execute():111	1
Object io.micronaut.web.router.AbstractRouteMatch.execute(Map):303	1
Object io.micronaut.context.DefaultBeanContext\$4.invoke(Object[]):583	1
Object io.micronaut.context.AbstractExecutableMethodsDefinition\$DispatchedExecutableMethod.invoke(Object, Object[]):351	1
Object inside.dumpster.micronaut.\$MicronautController\$Definition\$Exec.dispatch(int, Object, Object[]):-1	1
Result inside.dumpster.micronaut.MicronautController.destination(byte[], String, String, String, String, String, String, Str...	1
Result inside.dumpster.bl.BusinessLogicServiceWrapper.invoke(Payload):48	1
Result inside.dumpster.uploadtext.UploadTextService.invoke(Payload):25	1
UploadTextResult inside.dumpster.uploadtext.UploadTextService.invoke(UploadTextPayload):50	1
StoredData inside.dumpster.backend.repository.TextRepository.storeData(Text):45	1
void java.io.FileOutputStream.write(int):61	1



Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

**Using events
in unit-test**



Consuming events programmatically

`jdk.jfr.consumer.EventStream`

- Stream events from recording

`jdk.jfr.Recording`

- Configure, start, stop, dump recording

`jdk.jfr.consumer.RecordingStream`

- Stream events

`jdk.management.jfr.RemoteRecordingStream`

- Stream events from remote source

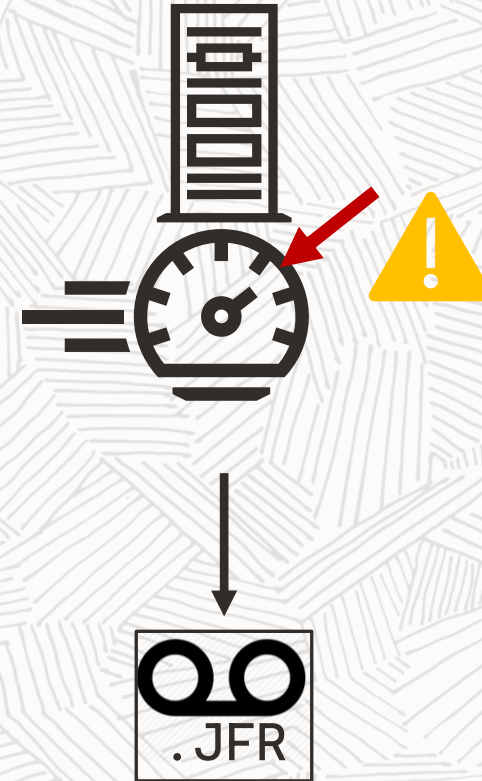
Monitoring CPU level using RecordingStream

Problem

- Our JVM takes up a lot of CPU, and we don't know why

Action

- When the CPU load is high, dump a JFR recording to analyze why this happens



Monitoring CPU level using RecordingStream

```
1.
2.
3.
4.
5.
6. void startCPULoadMonitor(Path dumpPath) throws Exception {
7.
8.     try (var stream = new RecordingStream()) {
9.
10.         stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.         stream.onEvent("jdk.CPULoad", event -> {
12.             float cpuLoad = event.getFloat("jvmUser");
13.             if (cpuLoad > 0.8) {
14.                 stream.dump(dumpPath);
15.             }
16.         });
17.
18.     }
19. }
```

RecordingStream has no events enabled without Configuration

- **enable events explicitly**, or
- add Configuration

Monitoring CPU level using RecordingStream

```
1.
2.
3.
4.
5.
6. void startCPULoadMonitor(Path dumpPath) throws Exception {
7.     Configuration conf = Configuration.getConfiguration("default");
8.     try (var stream = new RecordingStream( conf )) {
9.
10. //         stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.         stream.onEvent("jdk.CPULoad", event -> {
12.             float cpuLoad = event.getFloat("jvmUser");
13.             if (cpuLoad > 0.8) {
14.                 stream.dump(dumpPath);
15.             }
16.         });
17.
18.     }
19. }
```

RecordingStream has no events enabled without Configuration

- enable events explicitly, or
- **add Configuration**

Monitoring CPU level using RecordingStream

```
1.
2.
3.
4.
5.
6. void startCPULoadMonitor(Path dumpPath) throws Exception {
7.     Configuration conf = Configuration.getConfiguration("default");
8.     try (var stream = new RecordingStream( conf )) {
9.
10.         stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.         stream.onEvent("jdk.CPULoad", event -> {
12.             float cpuLoad = event.getFloat("jvmUser");
13.             if (cpuLoad > 0.8) {
14.                 stream.dump(dumpPath);
15.             }
16.         });
17.
18.     }
19. }
```

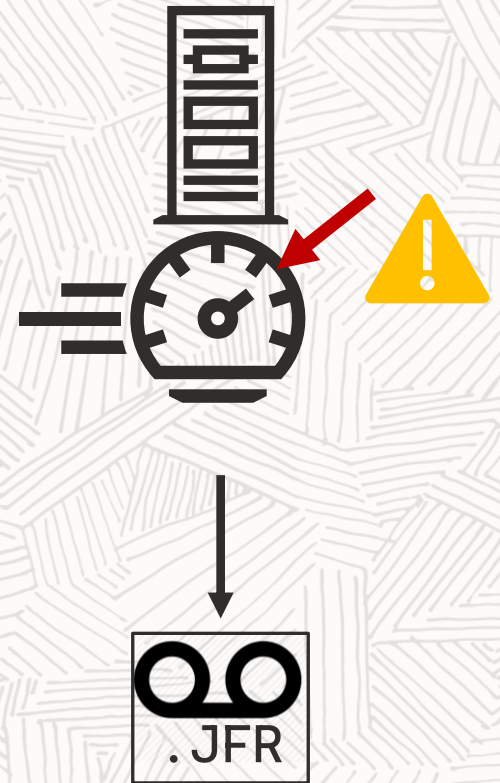
RecordingStream has no events enabled without Configuration

- enable events explicitly, or
- add Configuration

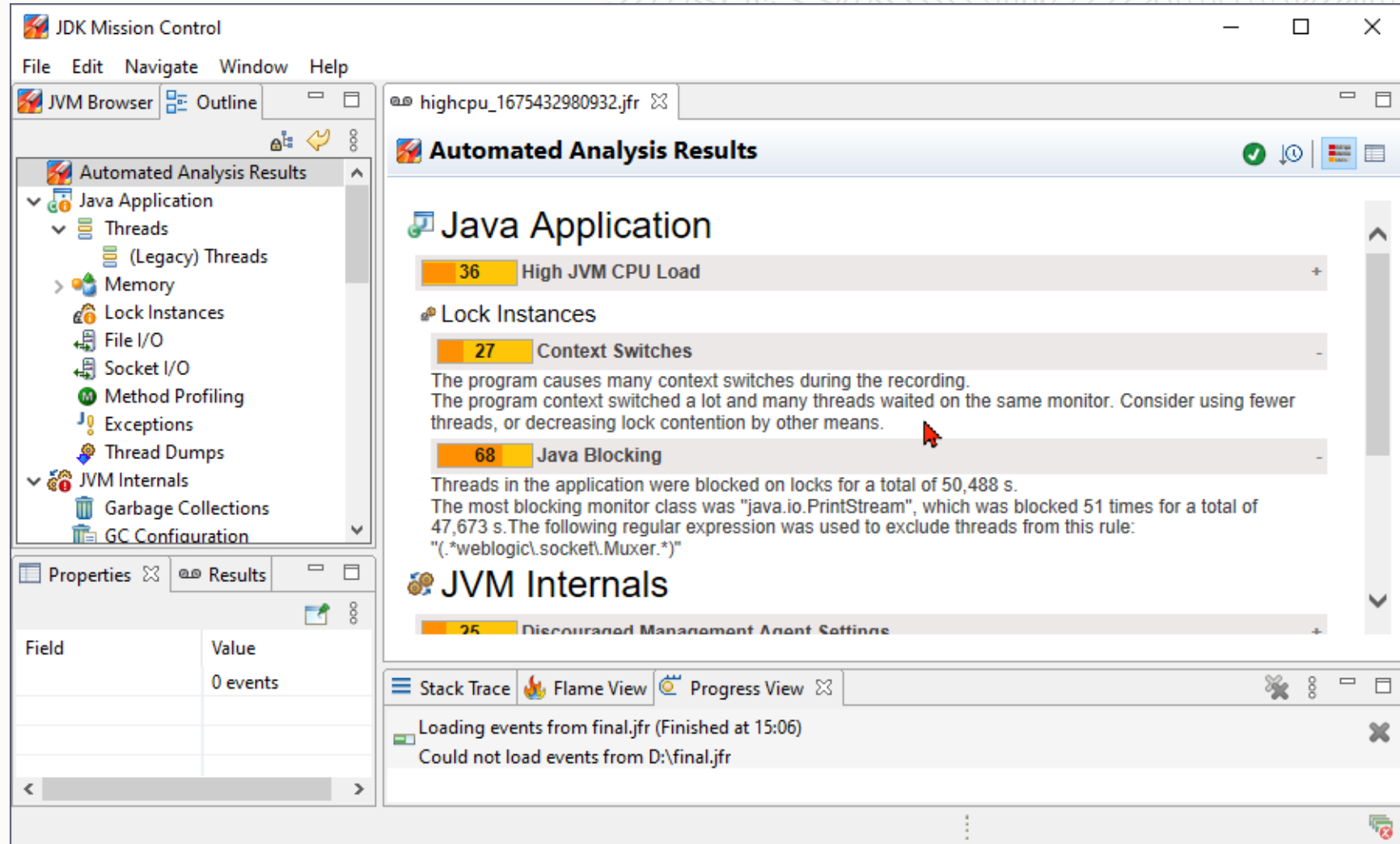
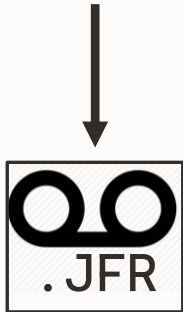
Since we want a JFR file to analyze, we want as many events as possible – use Configuration

Monitoring CPU level using RecordingStream

```
1.
2.
3.
4.
5.
6. void startCPULoadMonitor(Path dumpPath) throws Exception {
7.     Configuration conf = Configuration.getConfiguration("default");
8.     try (var stream = new RecordingStream( conf )) {
9.
10.         stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.         stream.onEvent("jdk.CPULoad", event -> {
12.             float cpuLoad = event.getFloat("jvmUser");
13.             if (cpuLoad > 0.8) {
14.                 stream.dump(dumpPath);
15.             }
16.         });
17.         stream.start();
18.     }
19. }
```



Monitoring CPU level using RecordingStream

The screenshot displays the JDK Mission Control (JDK MC) interface. The left sidebar shows a tree view of analysis categories: Java Application, Threads, (Legacy) Threads, Memory, Lock Instances, File I/O, Socket I/O, Method Profiling, Exceptions, Thread Dumps, JVM Internals, Garbage Collections, and GC Configuration. The main pane is titled "Automated Analysis Results" and shows a summary of performance issues. The "Java Application" section highlights three key metrics: High JVM CPU Load (36), Context Switches (27), and Java Blocking (68). The "JVM Internals" section shows Discouraged Management Agent Settings (25). The bottom pane shows a message: "Loading events from final.jfr (Finished at 15:06) Could not load events from D:\final.jfr".

JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

Automated Analysis Results

- Java Application
 - Threads
 - (Legacy) Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration

highcpu_1675432980932.jfr

Automated Analysis Results

Java Application

- 36** High JVM CPU Load
- 27** Context Switches
 - The program causes many context switches during the recording. The program context switched a lot and many threads waited on the same monitor. Consider using fewer threads, or decreasing lock contention by other means.
- 68** Java Blocking
 - Threads in the application were blocked on locks for a total of 50,488 s. The most blocking monitor class was "java.io.PrintStream", which was blocked 51 times for a total of 47,673 s. The following regular expression was used to exclude threads from this rule: "(.*weblogic\.socket\.Muxer.*)"

JVM Internals

- 25** Discouraged Management Agent Settings

Stack Trace Flame View Progress View

Loading events from final.jfr (Finished at 15:06)
Could not load events from D:\final.jfr

Monitoring CPU level using RecordingStream

The screenshot displays the JDK Mission Control interface for monitoring CPU levels. The left sidebar shows the navigation tree with 'Event Browser' selected. The main window shows the 'Event Browser' for the recording 'highcpu_1675432980932.jfr'. The 'Event Types Tree' on the left lists various CPU-related events, with 'CPU Load 7' selected. The 'Filter the table' section on the right shows a table of events with columns: Start Time, Event Thread, System ..., and User Mode... The table lists several events from CreditCrunch threads. The bottom status bar shows the 'Stack Trace' for the selected event, displaying 'int java.util.Random.next(int)'.

JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

- GC Configuration
- GC Summary
- Compilations
- Class Loading
- VM Operations
- TLAB Allocations
- Environment
 - Processes
 - Environment Variables
 - System Properties
 - Native Libraries
 - Recording
 - Event Browser**
 - WebLogic

highcpu_1675432980932.jfr

Event Browser

Focus: <No Selection> Aspect: <No Selection> ☒ Show concurrent: ☐ Contained ☐

Event Types Tree

Search the tree

- CPU Information 1
- CPU Load 7**
- CPU Throttling 0
- CPU Time Stamp Co
- CPU Usage 0
- Thread CPU Load 73
- Thread Context Swit
- Container Configuratio

Filter the table

Start Time	Event Thread	System ...	User Mode...
2023-02-03 15:02:57.881	CreditCrunch 17	0 %	9,34 %
2023-02-03 15:02:57.881	CreditCrunch 16	0 %	3,15 %
2023-02-03 15:02:57.881	CreditCrunch 3	0 %	3,13 %
2023-02-03 15:02:57.881	CreditCrunch 11	0 %	3,13 %
2023-02-03 15:02:57.881	CreditCrunch 8	0 %	3,1 %
2023-02-03 15:02:57.881	CreditCrunch 4	0 %	3,09 %
2023-02-03 15:02:57.881	CreditCrunch 5	0 %	3,08 %

Stack Trace

```
int java.util.Random.next(int)
```



Monitoring CPU level using RecordingStream

The screenshot displays the JDK Mission Control interface for monitoring CPU usage. The left sidebar shows a tree of monitoring categories, with 'Event Browser' selected under 'Recording'. The main window shows the 'Event Browser' for the recording 'highcpu_1675432980932.jfr'. The 'Event Types Tree' on the left lists various CPU-related events, with 'CPU Load 7' selected. The 'Filter the table' dropdown menu is open, showing options like 'Sort Columns', 'Visible Columns', 'Copy', 'Clipboard Settings', 'Store Selection', 'Store and Set As Focused Selection' (highlighted), 'Show Filter', and 'Show Search'. The table below shows the following data:

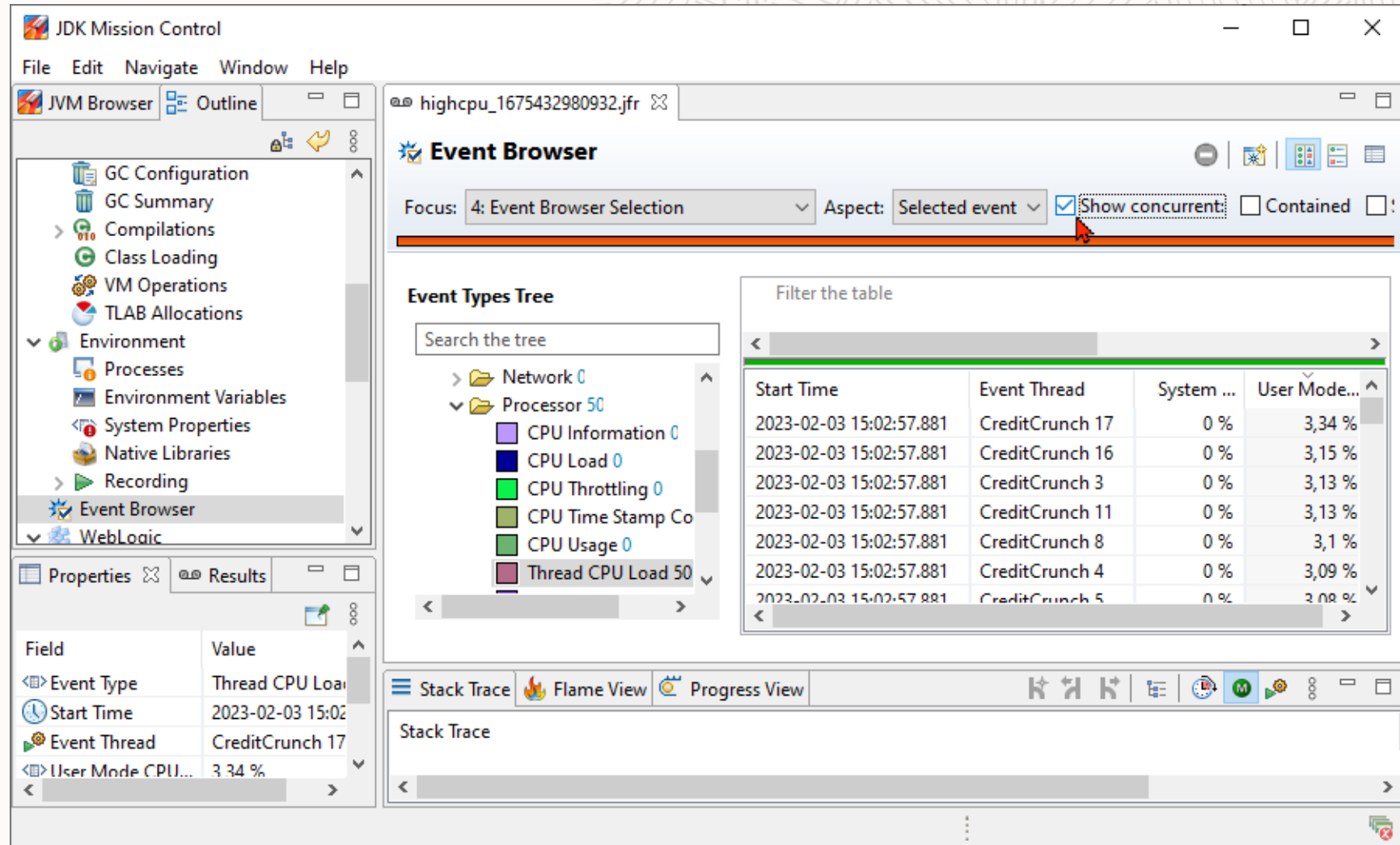
Start Time	Event Thread	System ...	User Mode...
2023-02-03 15:02:57.881	CreditCrunch 17	0 %	3,34 %
2	Sort Columns	>	3,15 %
2	Visible Columns	>	3,13 %
2	Copy	Ctrl+C	3,13 %
2	Clipboard Settings	>	3,1 %
2	Store Selection		3,09 %
2	Store and Set As Focused Selection		3,08 %
2	Show Filter		
2	Show Search		

The bottom of the window shows the 'Properties' tab with the following details:

Field	Value
Event Type	Thread CPU Load
Start Time	2023-02-03 15:02:57.881
Event Thread	CreditCrunch 17
User Mode CPU...	3,34 %



Monitoring CPU level using RecordingStream



JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

- GC Configuration
- GC Summary
- Compilations
- Class Loading
- VM Operations
- TLAB Allocations
- Environment
 - Processes
 - Environment Variables
 - System Properties
 - Native Libraries
 - Recording
 - Event Browser
- WebLogic

highcpu_1675432980932.jfr

Event Browser

Focus: 4: Event Browser Selection Aspect: Selected event ☒ Show concurrent: ☐ Contained ☐

Event Types Tree

Search the tree

- Network 0
- Processor 50
 - CPU Information 0
 - CPU Load 0
 - CPU Throttling 0
 - CPU Time Stamp Co
 - CPU Usage 0
 - Thread CPU Load 50

Filter the table

Start Time	Event Thread	System ...	User Mode...
2023-02-03 15:02:57.881	CreditCrunch 17	0 %	3,34 %
2023-02-03 15:02:57.881	CreditCrunch 16	0 %	3,15 %
2023-02-03 15:02:57.881	CreditCrunch 3	0 %	3,13 %
2023-02-03 15:02:57.881	CreditCrunch 11	0 %	3,13 %
2023-02-03 15:02:57.881	CreditCrunch 8	0 %	3,1 %
2023-02-03 15:02:57.881	CreditCrunch 4	0 %	3,09 %
2023-02-03 15:02:57.881	CreditCrunch 5	0 %	3,08 %

Properties Results

Field	Value
Event Type	Thread CPU Load
Start Time	2023-02-03 15:02:57.881
Event Thread	CreditCrunch 17
User Mode CPU...	3.34 %

Stack Trace Flame View Progress View

Stack Trace

Monitoring CPU level using RecordingStream

JDK Mission Control

File Edit Navigate Window Help

JVM Browser Outline

Automated Analysis Results

- Java Application
 - Threads
 - Memory
 - Lock Instances
 - File I/O
 - Socket I/O
 - Method Profiling
 - Exceptions
 - Thread Dumps
- JVM Internals
 - Garbage Collections
 - GC Configuration
 - GC Summary

Properties Results

Field	Value
Event Type	Thread Dump
Start Time	2023-02-03 15:02
Thread Dump	[2023-02-03 15:02]
	2 events

highcpu_1675432980932.jfr

Thread Dumps

Focus: 4: Event Browser Selection Aspect: Thread Name Show concurrent: Contained

Search the tree

- 2023-02-03 15:02:52.024
 - CreditCrunch 17
 - CreditCrunch 17
 - CreditCrunch 17
- 2023-02-03 15:03:01.318
 - CreditCrunch 17
 - CreditCrunch 17

"CreditCrunch 17" #187 prio=5 os_prio=0 cpu=0.00ms elapsed=0.41s tid=0x000001fdcaf69f50 nid=0x7d68 waiting for monitor entry [0x000000d5d5cfe000]
java.lang.Thread.State: BLOCKED (on object monitor)
at java.io.PrintStream.writeln
at java.io.PrintStream.println
(java.base@17.0.5/PrintStream.java:718)
- waiting to lock <0x000000060416efc0> (a java.io.PrintStream)
at java.io.PrintStream.println
(java.base@17.0.5/PrintStream.java:1028)
at inside.dumpster.credits.EarnCreditsService
SCPUConsumer.run(EarnCreditsService.java:50)
at java.lang.Thread.run(java.base@17.0.5/Thread.java:833)

Stack Trace Flame View Progress View

Stack Trace

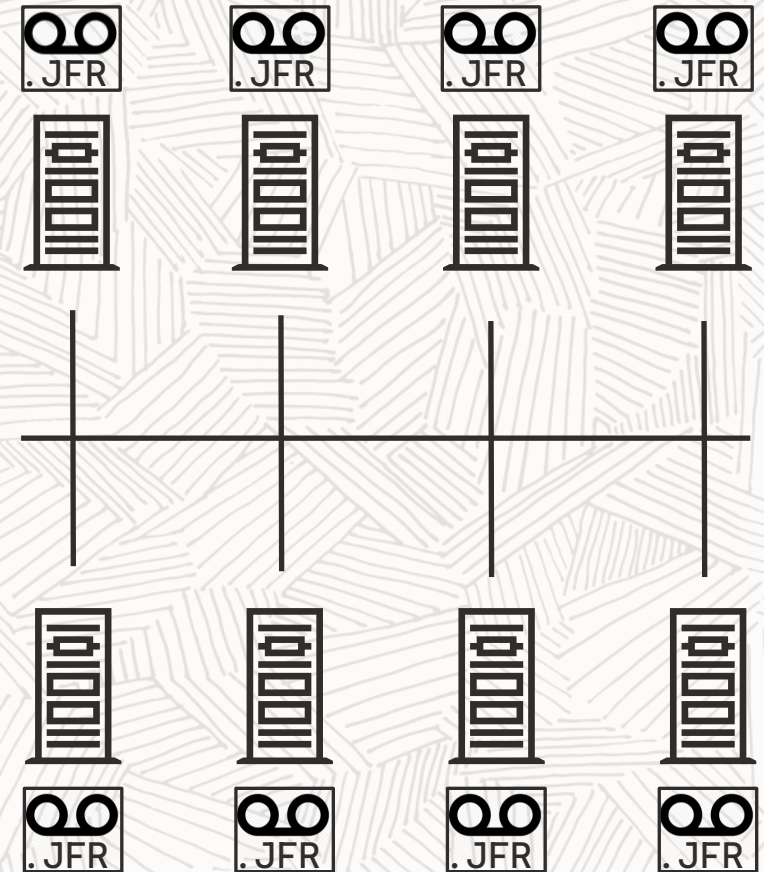
Remotely monitoring CPU level using RemoteRecordingStream

Problem

- We're saving JFR recordings on high CPU, but our server nodes are shortlived, and automatically removed

Solution

- Dump JFR recording on a more persistent node



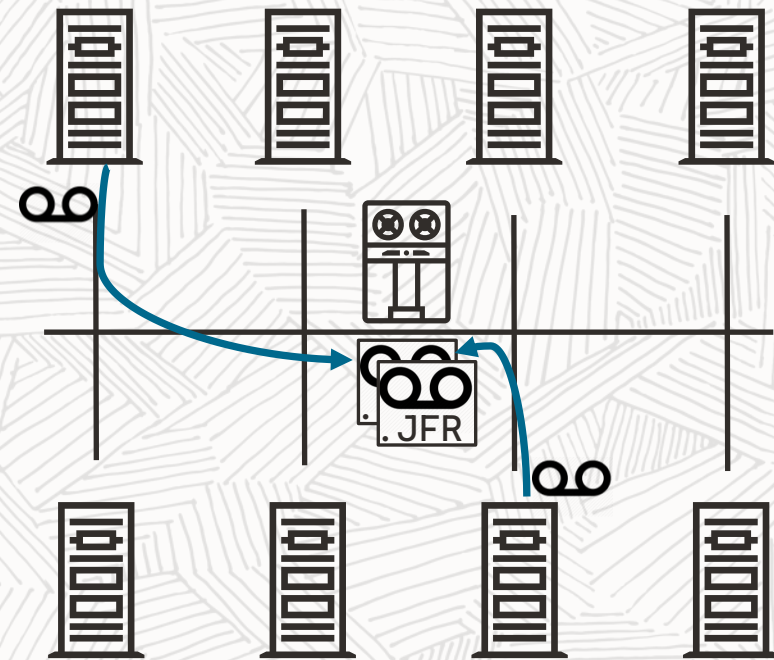
Remotely monitoring CPU level using RemoteRecordingStream

Problem

- We're saving JFR recordings on high CPU, but our server nodes are shortlived, and automatically removed

Solution

- Dump JFR recording on a more persistent node



Remotely monitoring CPU level using RemoteRecordingStream

```
1. void startCPULoadMonitor(String host, Path dumpPath) throws Exception {
2.
3.
4.
5.
6.
7.
8.     try (var stream = new RemoteRecordingStream()) {
9.
10.         stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.         stream.onEvent("jdk.CPULoad", event -> {
12.             float cpuLoad = event.getFloat("jvmUser");
13.             if (cpuLoad > 0.8) {
14.                 stream.dump(dumpPath);
15.             }
16.         });
17.         stream.start();
18.     }
19. }
```

Remotely monitoring CPU level using RemoteRecordingStream

```
1. void startCPULoadMonitor(String host, Path dumpPath) throws Exception {
2.     JMXServiceURL u = new JMXServiceURL(
3.         "service:jmx:rmi:///jndi/rmi://" + host + "/jmxrmi");
4.     JMXConnector c = JMXConnectorFactory.connect(u,
5.         Map.of("jmx.remote.credentials", new String[]{user, pwd}));
6.     MBeanServerConnection conn = c.getMBeanServerConnection();
7.
8.     try (var stream = new RemoteRecordingStream( conn )) {
9.
10.        stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.        stream.onEvent("jdk.CPULoad", event -> {
12.            float cpuLoad = event.getFloat("jvmUser");
13.            if (cpuLoad > 0.8) {
14.                stream.dump(dumpPath);
15.            }
16.        });
17.        stream.start();
18.    }
19. }
```

Remotely monitoring CPU level using RemoteRecordingStream

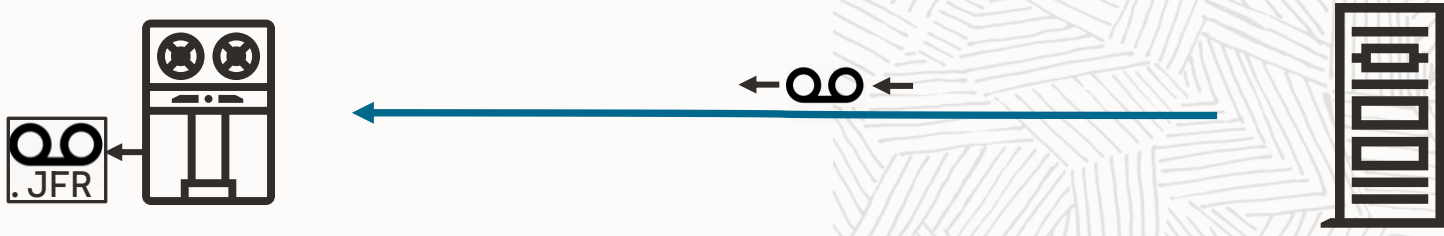
```
1. void startCPULoadMonitor(String host, Path dumpPath) throws Exception {
2.     JMXServiceURL u = new JMXServiceURL(
3.         "service:jmx:rmi:///jndi/rmi://" + host + "/jmxrmi");
4.     JMXConnector c = JMXConnectorFactory.connect(u,
5.         Map.of("jmx.remote.credentials", new String[]{user, pwd}));
6.     MBeanServerConnection conn = c.getMBeanServerConnection();
7.
8.     try (var stream = new RemoteRecordingStream( conn )) {
9.         stream.setMaxAge(Duration.ofMinutes(10));
10.        stream.enable("jdk.CPULoad").withPeriod(Duration.ofSeconds(1));
11.        stream.onEvent("jdk.CPULoad", event -> {
12.            float cpuLoad = event.getFloat("jvmUser");
13.            if (cpuLoad > 0.8) {
14.                stream.dump(dumpPath);
15.            }
16.        });
17.        stream.start();
18.    }
19. }
```

Javadoc:

It's highly recommended that a max age or max size is set before starting the stream.

This means we'll keep 10 minutes worth of data on the client.

Remotely monitoring CPU level using RemoteRecordingStream



Client side

```
java -jar RemoteMonitor.jar server12:12345
```

Server side

```
java -XX:StartFlightRecording
-Dcom.sun.management.jmxremote.port=12345
-Dcom.sun.management.jmxremote.authenticate=true
-Dcom.sun.management.jmxremote.password.file=jmx.pwd
-Dcom.sun.management.jmxremote.access.file=jmx.acs
-Dcom.sun.management.jmxremote.ssl=false
-Djava.rmi.server.hostname=127.0.0.1
-jar server.jar
```

JMX setup link:

<https://docs.oracle.com/en/java/javase/19/management/monitoring-and-management-using-jmx-technology.html>

Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

**Using events
in unit-test**



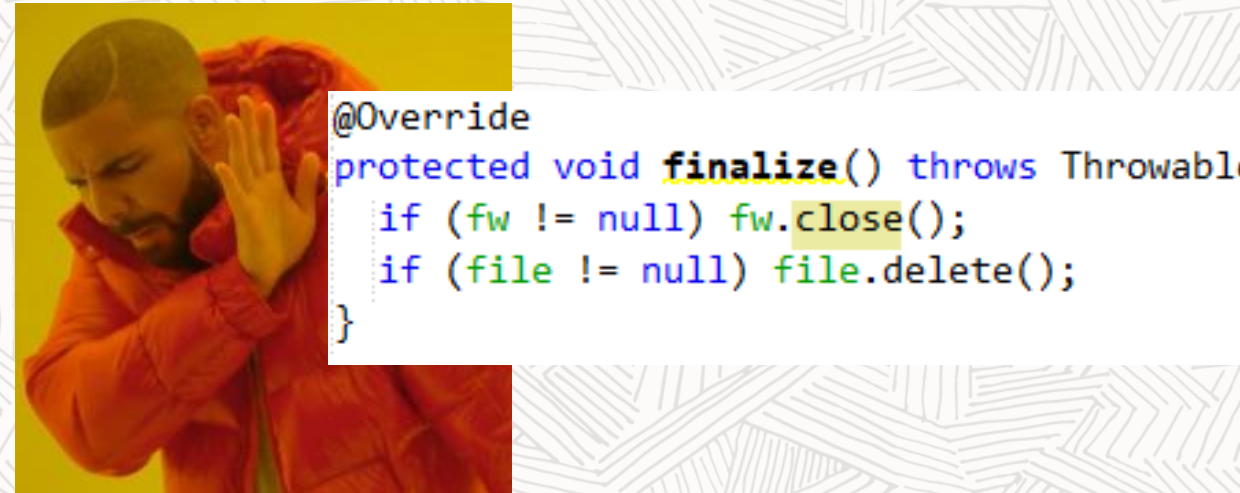
Using JFR events in unit tests

JEP 421: Deprecate Finalization for Removal

Finalizers are deprecated for removal

New JFR event:

- `jdk.FinalizerStatistics` events are emitted for each instantiated class with a non-empty `finalize()` method



Using JFR events in unit tests

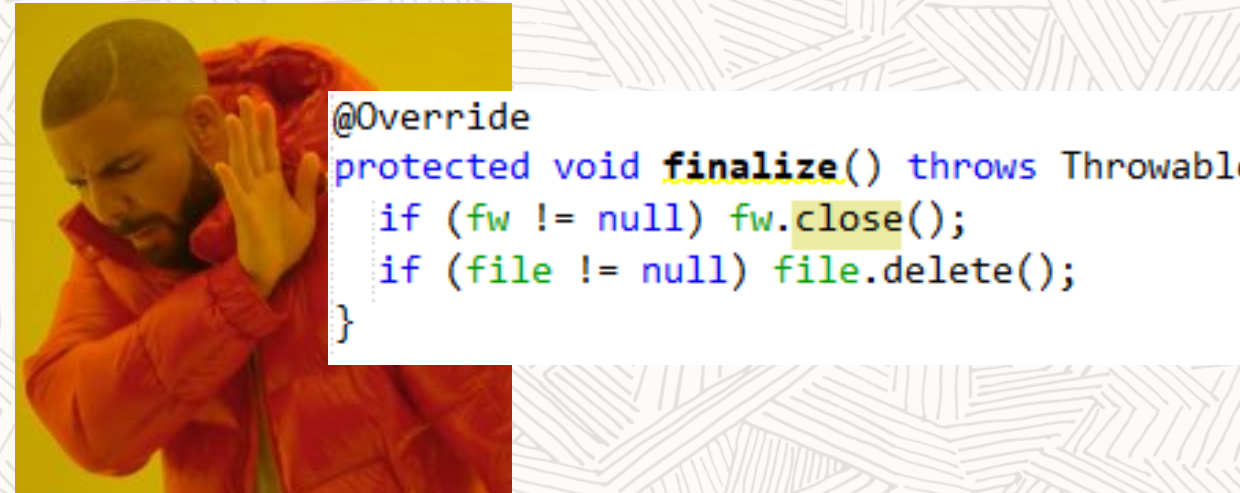
Finalizers are deprecated for removal

Problem

- Developers keep adding finalizers in their services.

Solution

- Add a testcase that captures all `jdk.FinalizerStatistics` events



Using JFR events in unit tests -- Recording

```
1. public void testNoServicesUsesFinalizers() throws Exception {
2.     try (Recording r = new Recording()) {
3.         r.enable("jdk.FinalizerStatistics");
4.         r.start();
5.
6.         callAllServices(); // performs a call to each registered service
7.
8.         r.stop();
9.         r.dump(tempFile);
10.
11.        try (EventStream stream = EventStream.openFile(tempFile)) {
12.            stream.onEvent("jdk.FinalizerStatistics", event -> {
13.                classesWithFinalizers.add(event.getClass("finalizableClass").getName());
14.            });
15.            stream.start();
16.        }
17.        Files.delete(tempFile);
18.    }
19.    assertTrue(classesWithFinalizers.isEmpty());
20. }
```

Using JFR events in unit tests -- Recording

```
1. public void testNoServicesUsesFinalizers() throws Exception {
2.     try (Recording r = new Recording()) {
3.         r.enable("jdk.FinalizerStatistics");
4.         r.start();
5.
6.         callAllServices(); // performs a call to each registered service
7.
8.         r.stop();
9.         r.dump(tempFile);
10.
11.         try (EventStream stream
12.             stream.onEvent("jdk.Fi
13.                 classesWithFinalizer
14.             });
15.             stream.start();
16.         }
17.         Files.delete(tempFile);
18.     }
19.     assertTrue(classesWithFinal
20. }
```

53

Test Results	Bookmarks	Usages	Output	Sources	Notifications	Sessions
inside.dumpster.businesslogic:BusinessLogic:jar:1.0-SNAPSHOT (Unit) X						
Tests passed: 100,00 %						
The test passed. (1,324 s)						
inside.dumpster.bl.NoFinalizersTest passed						
testNoServicesUsesFinalizers passed (1,109 s)						



Using JFR events in unit tests

Recommended to use `jdk.jfr.Recording`

- `RecordingStream` *can* be used in unit-testing, however:
 - cumbersome to end the stream;
have to wait for all events to be consumed
- Until now!
- [JDK-8295487](#) JFR: Add stop methods for recording streams
- NB: Will come in JDK 20

Using JFR events in unit tests -- RecordingStream



JDK 20

```
1.
2. @Test public void testNoServicesUsesFinalizers throws Exception {
3.     try (var stream = new RecordingStream()) {
4.         stream.enable("jdk.FinalizerStatistics");
5.         stream.onEvent("jdk.FinalizerStatistics", (event) -> {
6.             classesWithFinalizers.add(event.getClass("finalizableClass").getName());
7.         });
8.         stream.startAsync();
9.
10.        callAllServices();
11.
12.        stream.stop();           // new in JDK 20
13.    }
14.    assertTrue(classesWithFinalizers.isEmpty());
15. }
```

Using JFR events in unit tests -- RecordingStream

JDK 20

```
1.
2. @Test public void testNoServicesUsesFinalizers throws Exception {
3.     try (var stream = new RecordingStream()) {
4.         stream.enable("jdk.FinalizerStatistics");
5.         stream.onEvent("jdk.FinalizerStatistics", (event) -> {
6.             classesWithFinalizers.add(event.getClass("finalizableClass").getName());
7.         });
8.         stream.startAsync();
9.
10.        callAllServices();
11.
12.        stream.stop(); // ne
13.    }
14.    assertTrue(classesWithFinaliz
15. }
```

The screenshot shows the 'Test Results' tab in an IDE. The test runner is 'inside.dumpster.businesslogic:BusinessLogic:jar:1.0-SNAPSHOT (Unit)'. A green progress bar indicates 'Tests passed: 100,00 %'. Below this, a message states 'The test passed. (1,324 s)'. The test results list shows two items: 'inside.dumpster.bl.NoFinalizersTest passed' and 'testNoServicesUsesFinalizers passed (1,109 s)', both marked with green checkmarks. A vertical toolbar on the left contains icons for running, stopping, and other test actions. A mouse cursor is visible at the bottom right.

Agenda

1

**Start/stop
recordings**

2

**Analyzing
recordings**

3

**Create and
use custom
events**

4

**Consuming
events**

5

**Using events
in unit-test**



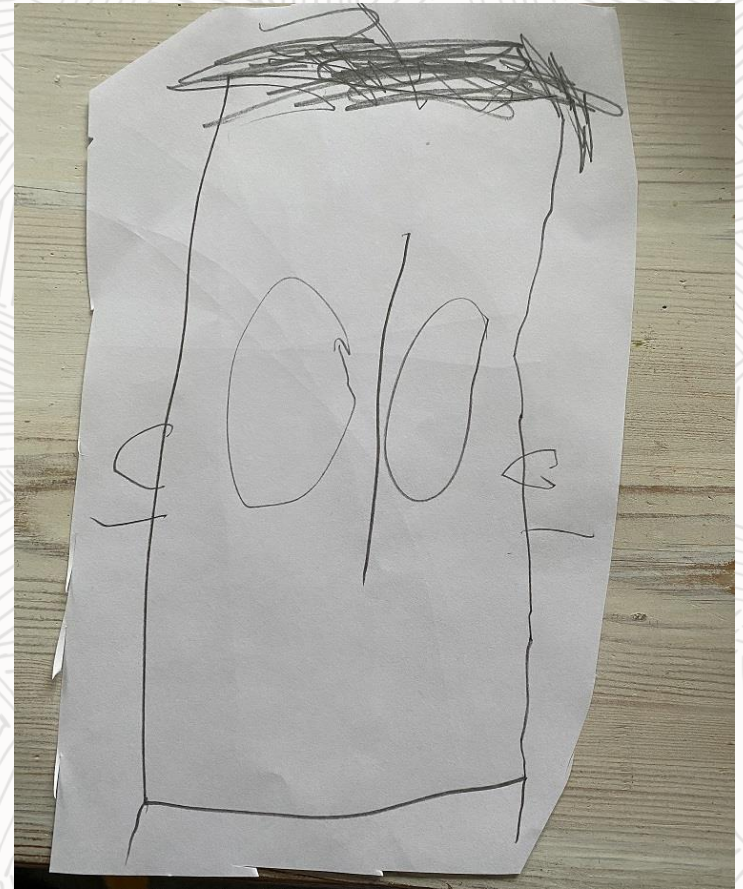
Questions?

Blog: jaokim.github.io

Github: github.com/jaokim/inside-java-dumpster

Twitter: @jaokim

Mastodon: @jaokim@techhub.social



eof
—