ORACLE

Programmer's Guide to the JDK Flight Recorder

Joakim Nordström
Sustaining Engineer
Java Product Group, Oracle
February 2023

JDK Flight Recorder

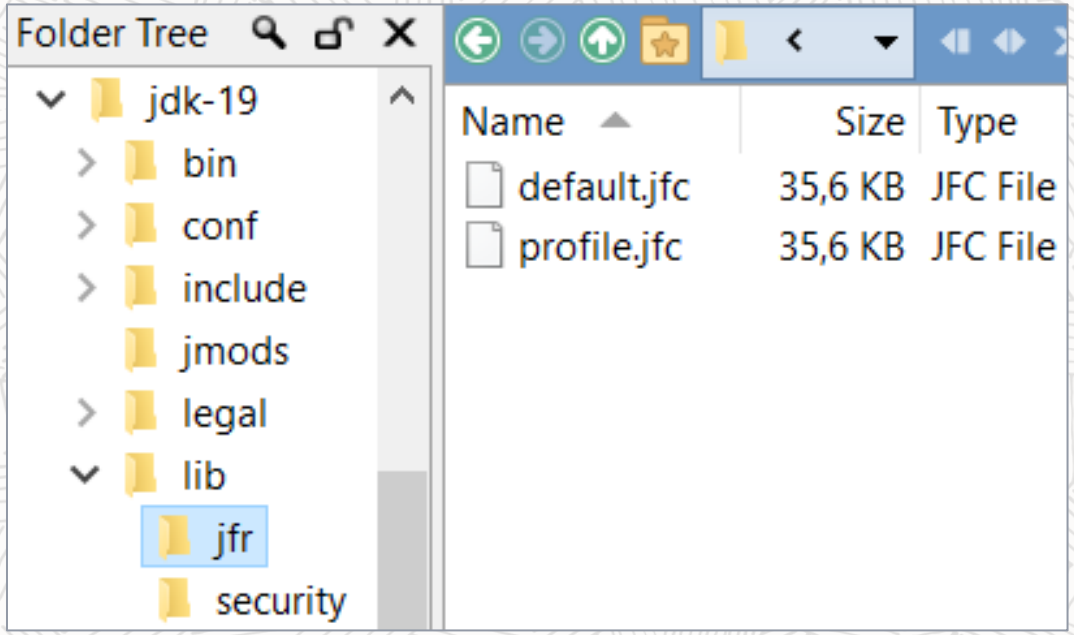
- Records events from the JVM, the JDK and applications
  - unified way to inspect the entire software stack
  - correlate events across different software layers and components
- Part of the JVM
- Low overhead
- Designed to be ”always on”

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 continous 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!

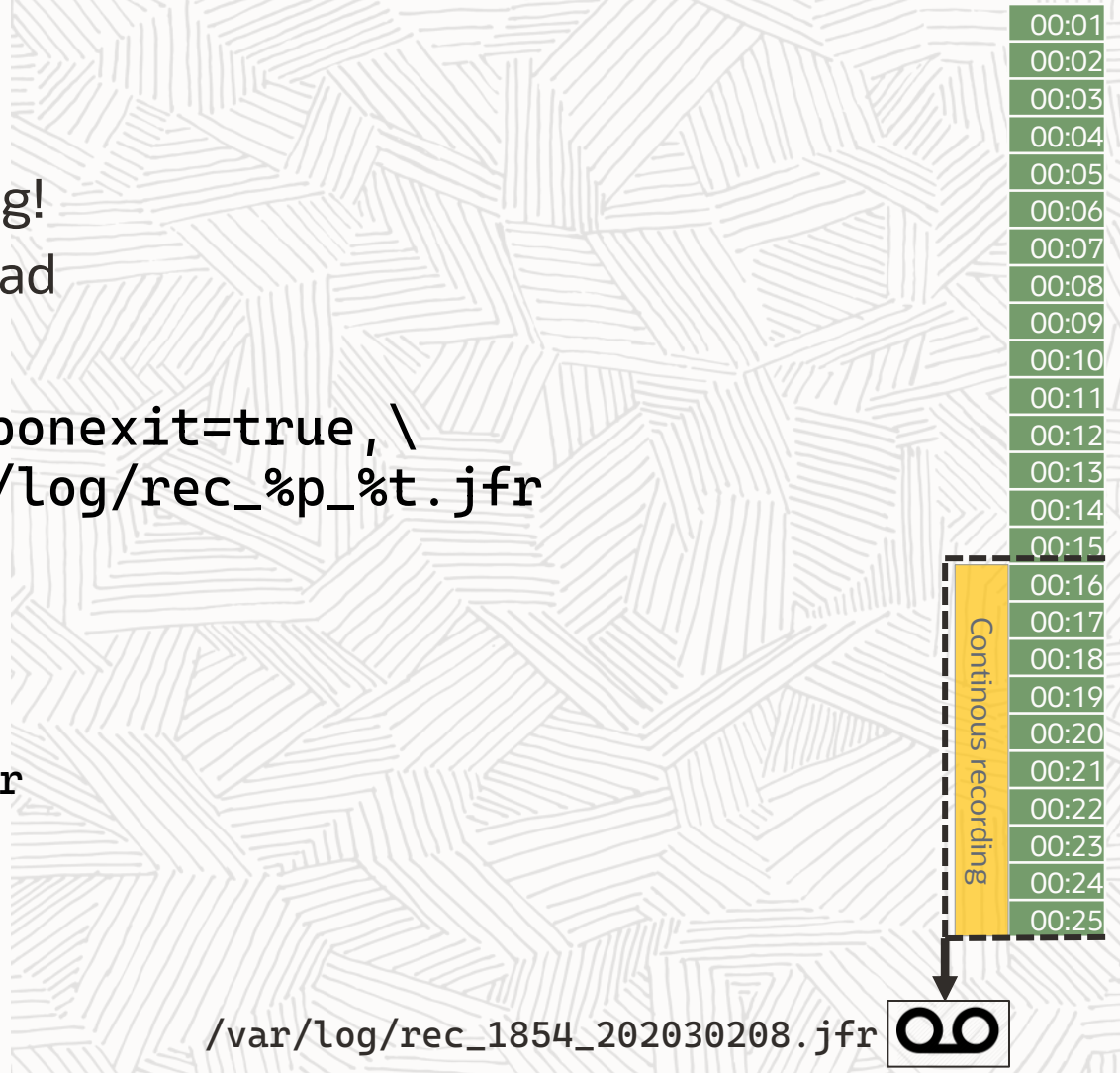
- Make copies if needed
  - Use `jfr` commmandline tool!
- Store custom settings together with app settings

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
```

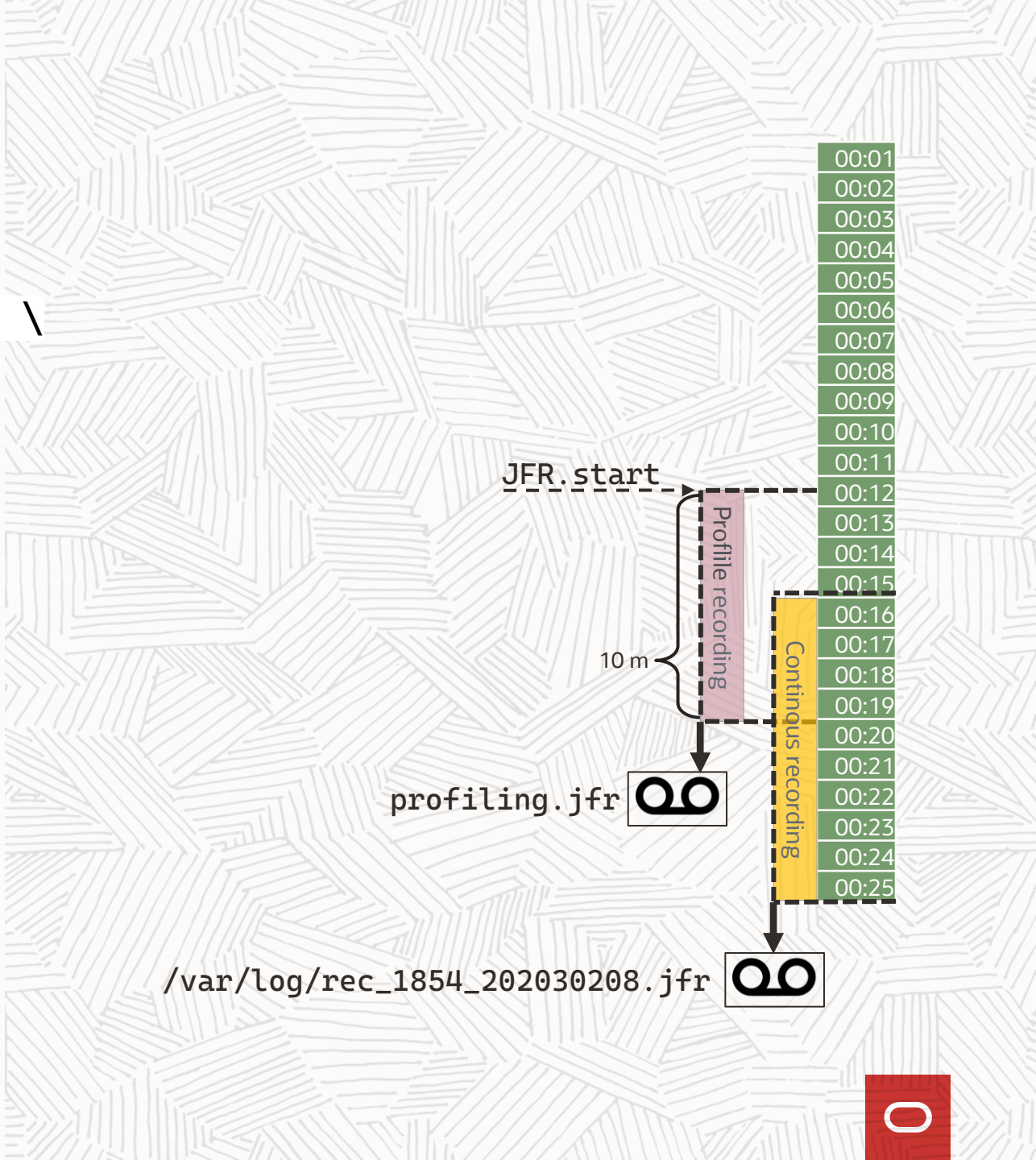
- **`dumponexit=true`**
  - will write JFR on JVM exit
- default name: **`hotspot_<pid>_<timestamp>.jfr`**
  - **`filename`** can be
    - directory
    - filename
    - **`%p`** for Process ID
    - **`%t`** for timestamp



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

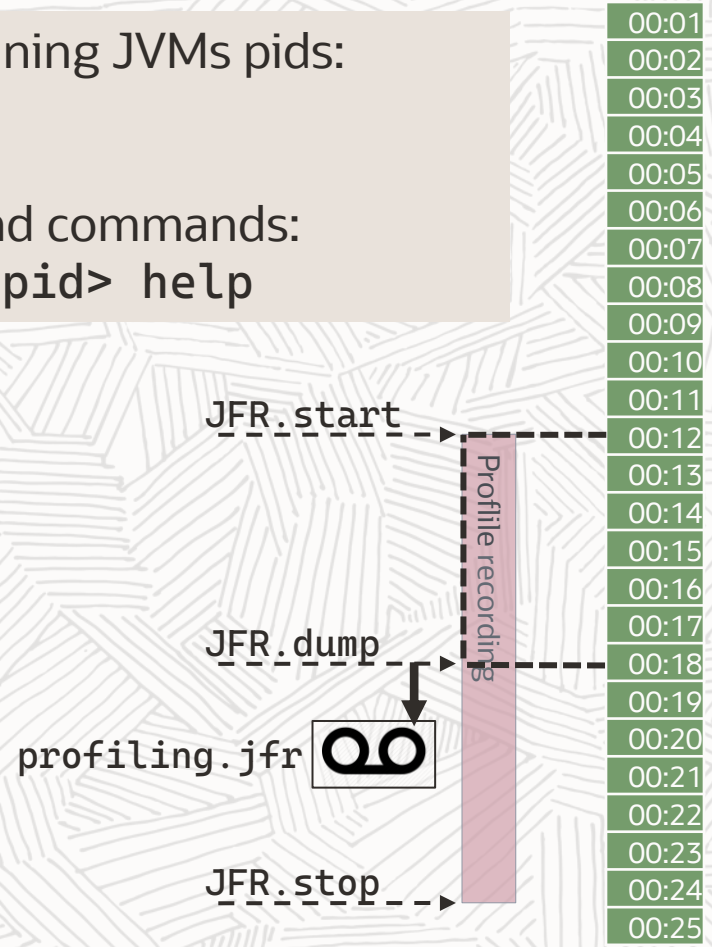
```
$ jcmd <pid> JFR.check
30496:
Recording 1: name=1 maxsize=250,0MB (running)
```

List all running JVMs pids:

```
$ jcmd
```

List all jcmd commands:

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



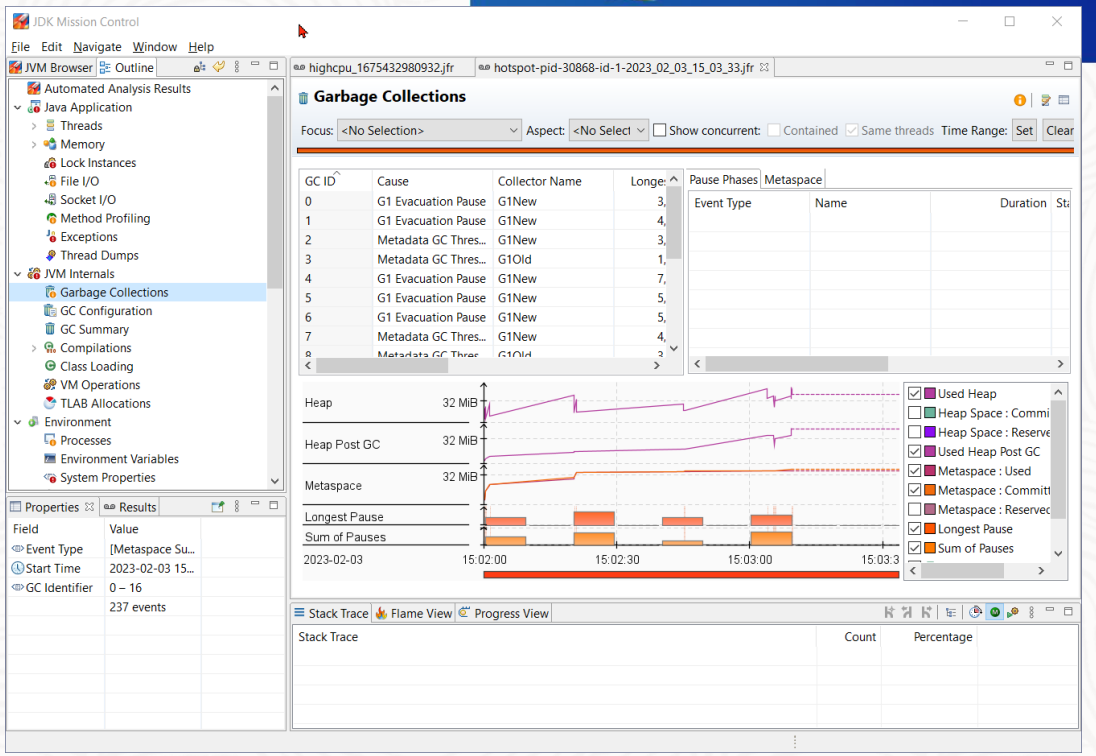
Agenda

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”

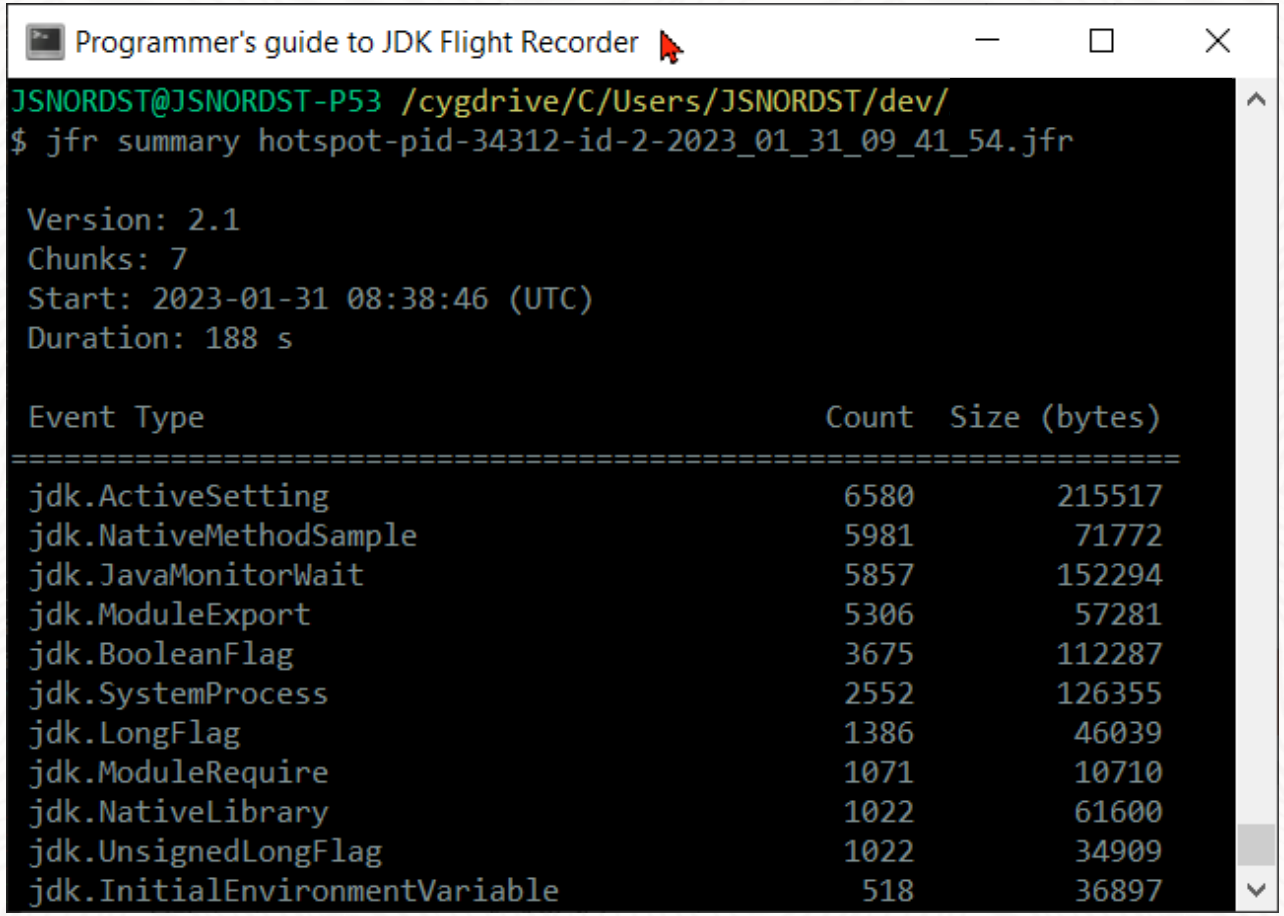
- Graphical interface to analyze recordings
- Separate download: oracle.com/java/technologies/jdk-mission-control.html
- Eclipse Plugins



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



Agenda

1. Start/stop recordings
2. Analyzing recordings
3. Create and use custom events
4. Consuming events
5. Using events in unit-test

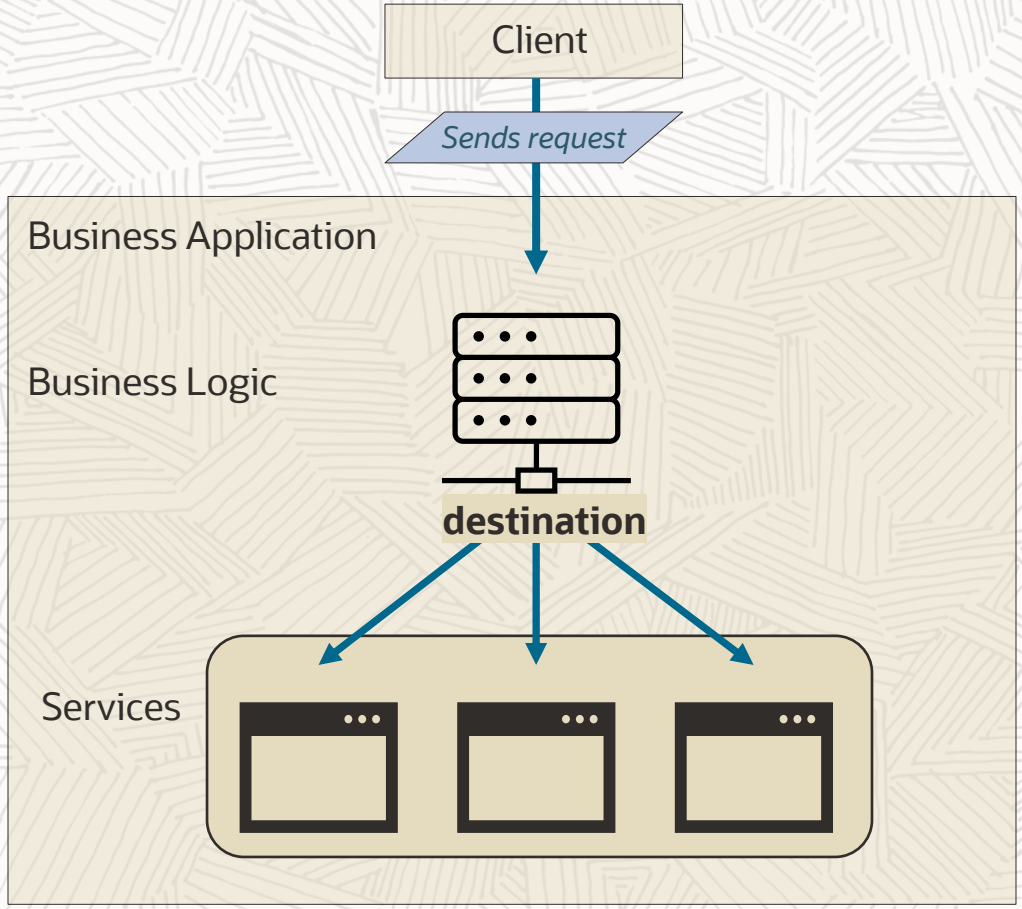
Our Business Application -- Inside the Java Dumpster

Mimics a real world system

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

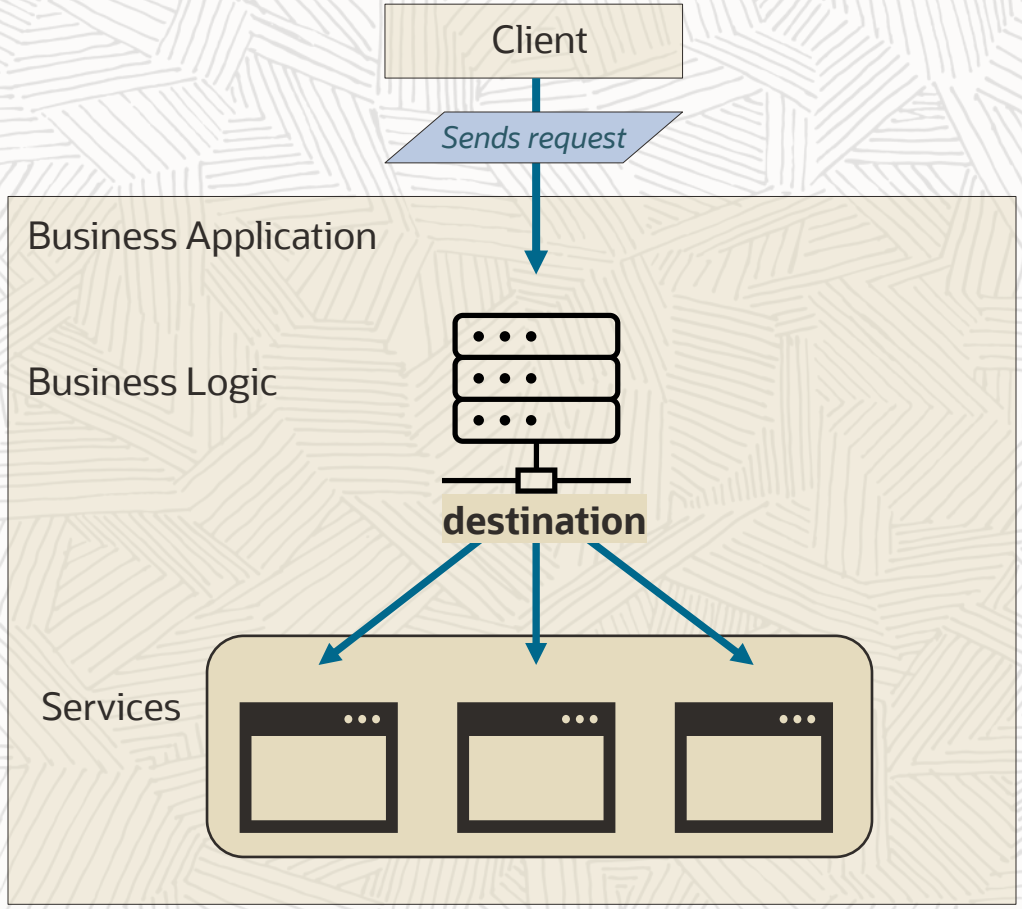
Client sends request to Business Application

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



Our Business Application -- Inside the Java Dumpster

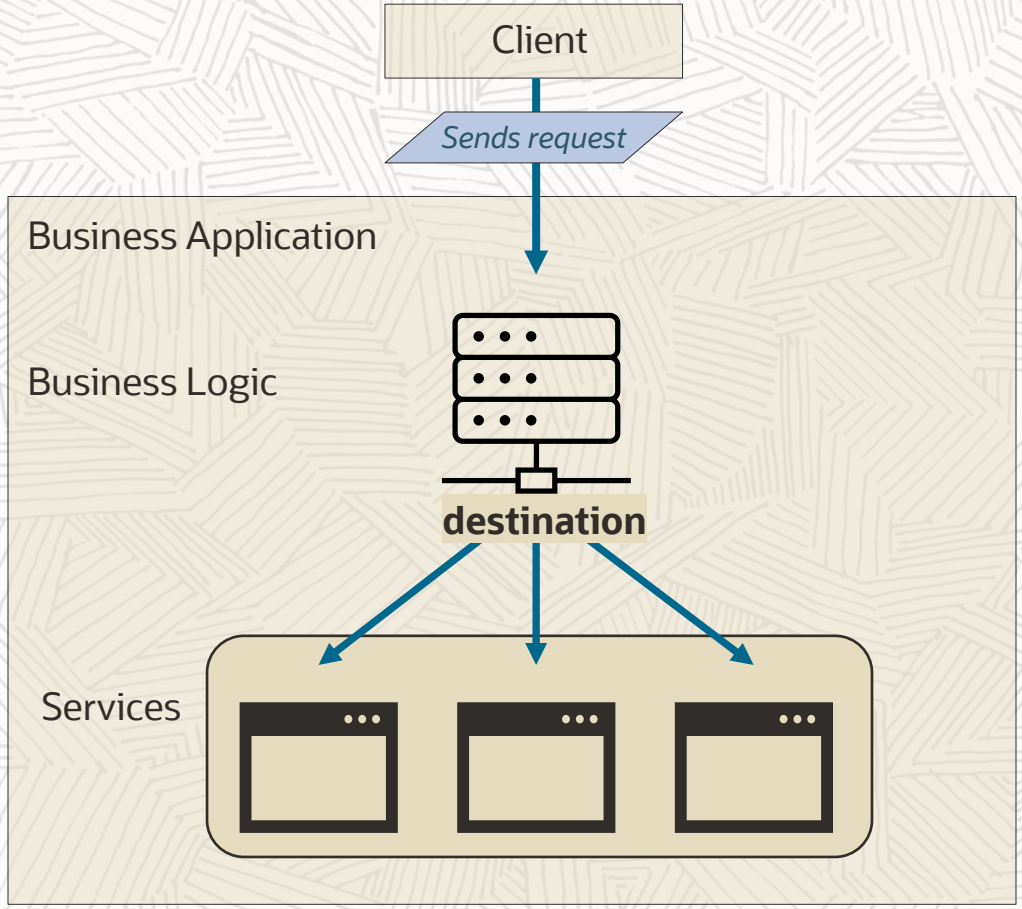
```
Service lookupService(Destination destination) {

  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;
}
```


```java
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:

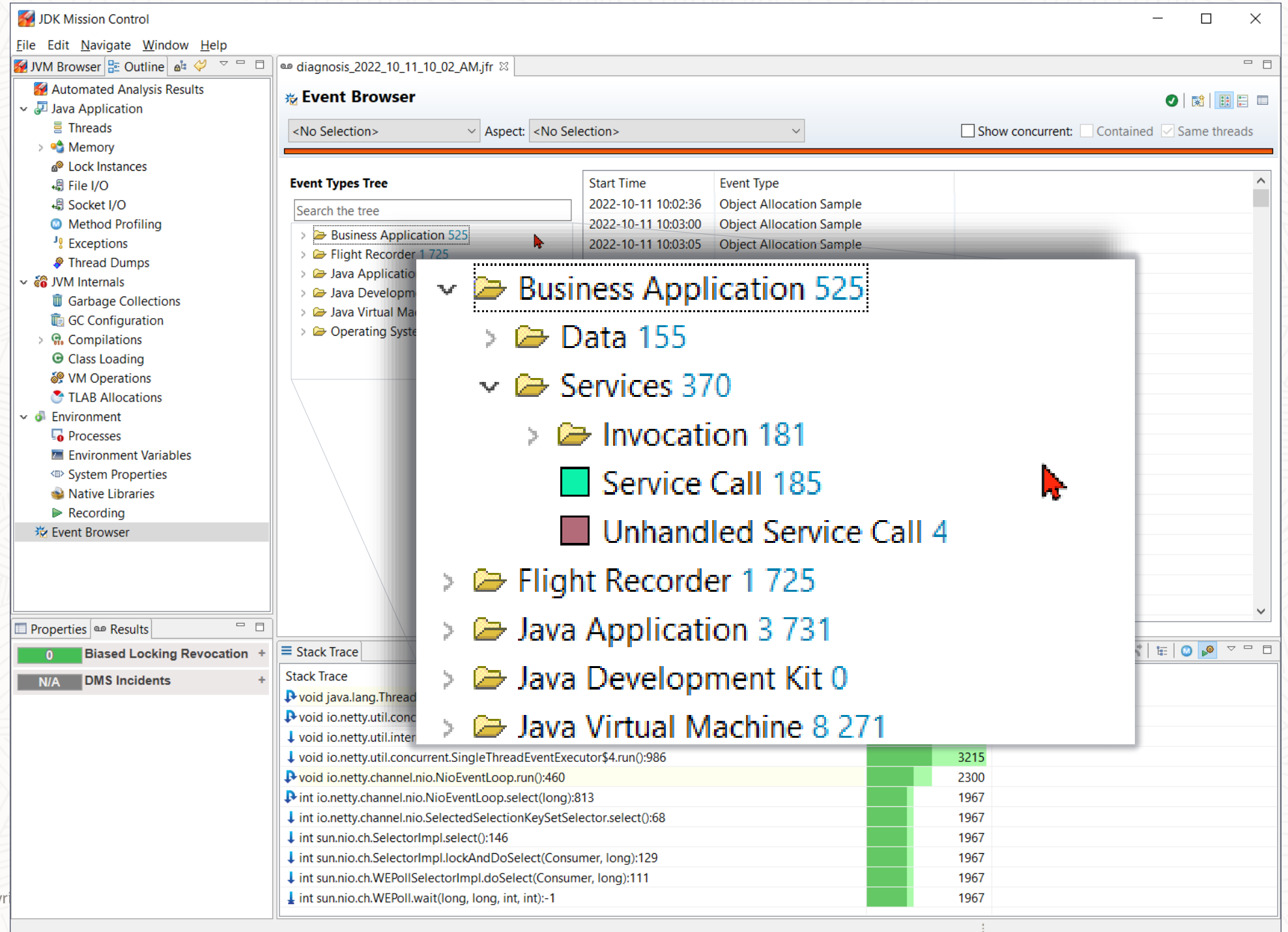
```
for (int pos : backend.getNext()) {
    var event = new BytesRead();
    event.begin();
    byte b[] = backend.read(pos);
    event.end();
    event.bytes = b.length();
    event.commit();
}
```

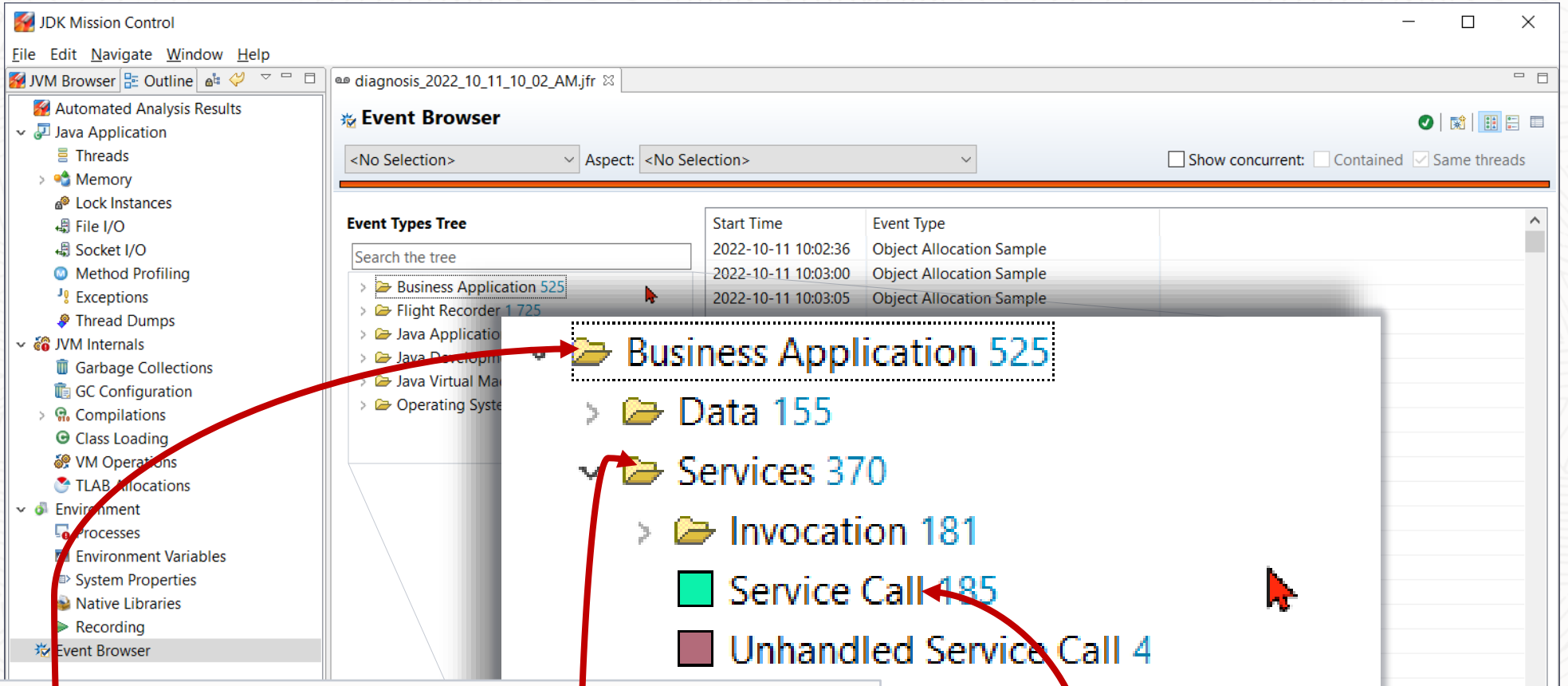

Agenda

1. Start/stop recordings
2. Analyzing recordings
3. Create and use custom events
4. Consuming events
5. Using events in unit-test

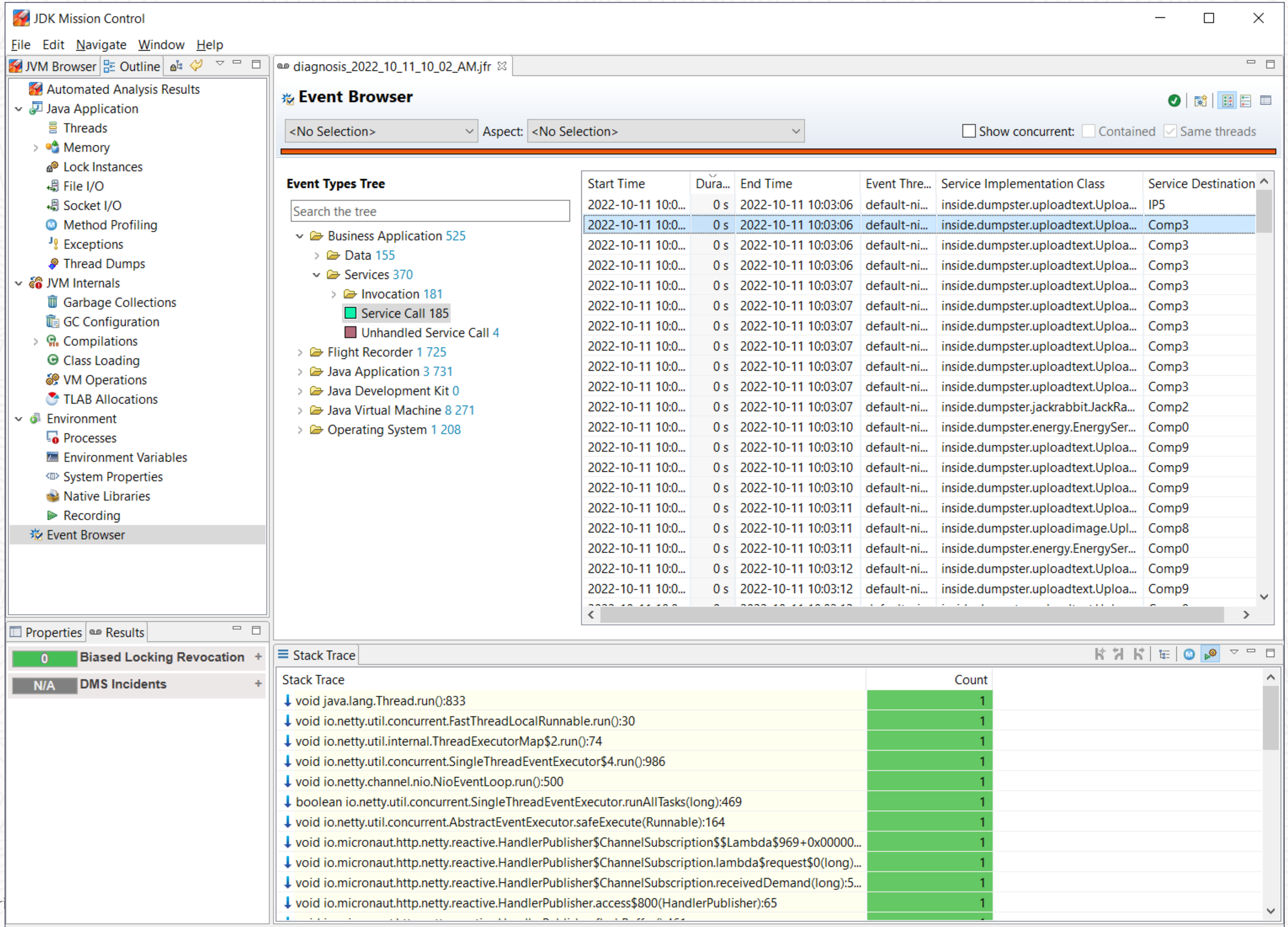
Agenda

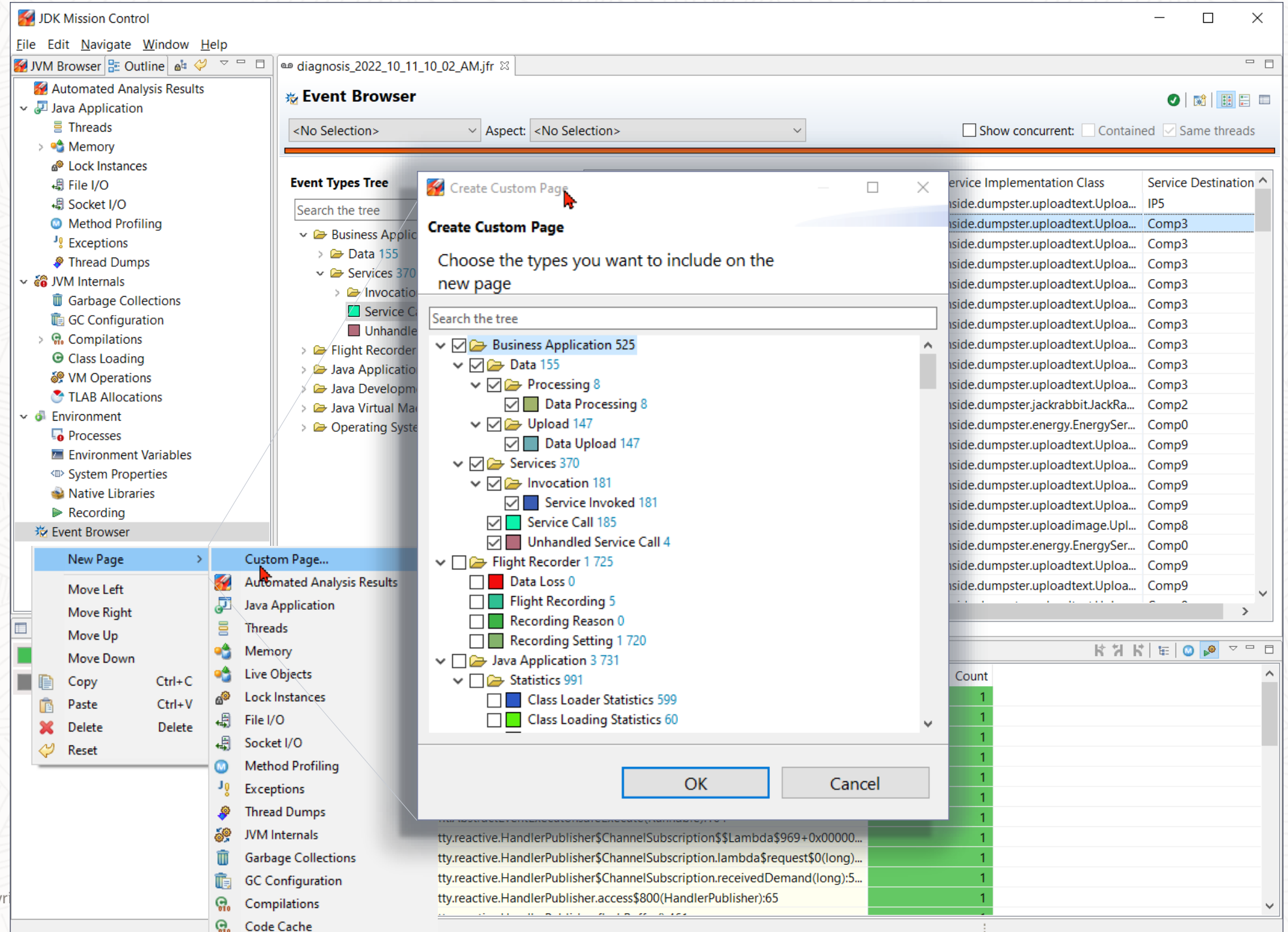
1. Start/stop recordings
2. Analyzing recordings
3. Create and use custom events
4. Consuming events
5. Using events in unit-test

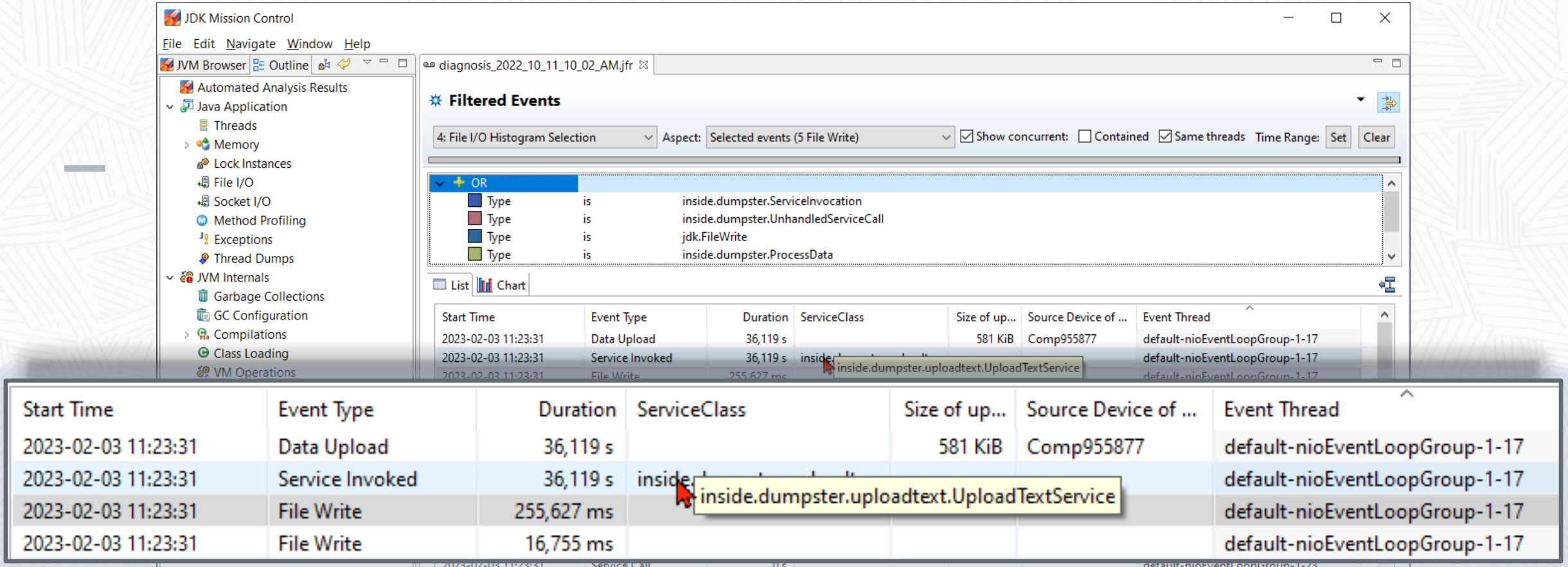

JDK Mission Control
File Edit Navigate Window Help
JVM Browser Outline
Automated Analysis Results
Java Application
Threads
Memory
Lock Instances
File I/O
Socket I/O
Method Profiling
Exceptions
Thread Dumps
JVM Internals
Garbage Collections
GC Configuration
Compilations
Class Loading
VM Operations
TLAB 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
Flight Recorder 1 725
Java Application
Java Development
Java Virtual Ma
Operating Syste
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
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
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
3215
void io.netty.channel.nio.NioEventLoop.run():460
2300
int io.netty.channel.nio.NioEventLoop.select(long):813
1967
int io.netty.channel.nio.SelectedSelectionKeySetSelector.select():68
1967
int sun.nio.ch.SelectorImpl.select():146
1967
int sun.nio.ch.SelectorImpl.lockAndDoSelect(Consumer, long):129
1967
int sun.nio.ch.WEPollSelectorImpl.doSelect(Consumer, long):111
1967
int sun.nio.ch.WEPoll.wait(long, long, int, int):-1
1967




```
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;
```


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
Properties
Results
0 Biased Locking Revocation
N/A DMS Incidents
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
Operating System 1 208
Start Time
Dura...
End Time
Event Thre...
Service Implementation Class
Service Destination
2022-10-11 10:0... 0 s 2022-10-11 10:03:06 default-ni... inside.dumpster.uploadtext.Uploa... IP5
2022-10-11 10:0... 0 s 2022-10-11 10:03:06 default-ni... inside.dumpster.uploadtext.Uploa... Comp3
2022-10-11 10:0... 0 s 2022-10-11 10:03:07 default-ni... inside.dumpster.uploadtext.Uploa... Comp3
2022-10-11 10:0... 0 s 2022-10-11 10:03:07 default-ni... inside.dumpster.jackrabbit.JackRa... Comp2
2022-10-11 10:0... 0 s 2022-10-11 10:03:10 default-ni... inside.dumpster.energy.EnergySer... Comp0
2022-10-11 10:0... 0 s 2022-10-11 10:03:10 default-ni... inside.dumpster.uploadtext.Uploa... Comp9
2022-10-11 10:0... 0 s 2022-10-11 10:03:11 default-ni... inside.dumpster.uploadimage.Upl... Comp8
2022-10-11 10:0... 0 s 2022-10-11 10:03:12 default-ni... inside.dumpster.uploadtext.Uploa... Comp9
Stack Trace
Stack Trace
Count
void java.lang.Thread.run():833
void io.netty.util.concurrent.FastThreadLocalRunnable.run():30
void io.netty.util.internal.ThreadExecutorMap$2.run():74
void io.netty.util.concurrent.SingleThreadEventExecutor$4.run():986
void io.netty.channel.nio.NioEventLoop.run():500
boolean io.netty.util.concurrent.SingleThreadEventExecutor.runAllTasks(long):469
void io.netty.util.concurrent.AbstractEventExecutor.safeExecute(Runnable):164
void io.micronaut.http.netty.reactive.HandlerPublisher$ChannelSubscription$$Lambda$969+0x00000...
void io.micronaut.http.netty.reactive.HandlerPublisher$ChannelSubscription.lambda$request$0(long)...
void io.micronaut.http.netty.reactive.HandlerPublisher$ChannelSubscription.receivedDemand(long):5...
void io.micronaut.http.netty.reactive.HandlerPublisher.access$800(HandlerPublisher):65


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 Applic
Data 155
Services 370
Invocatio
Service C
Unhandle
Flight Recorder
Java Applicatio
Java Developm
Java Virtual Ma
Operating Syste
Service Implementation Class
Service Destination
IP5
Comp3
Comp2
Comp0
Comp9
Comp8
New Page
Move Left
Move Right
Move Up
Move Down
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Reset
Custom Page...
Automated Analysis Results
Java Application
Threads
Memory
Live Objects
Lock Instances
File I/O
Socket I/O
Method Profiling
Exceptions
Thread Dumps
JVM Internals
Garbage Collections
GC Configuration
Compilations
Code Cache
Create Custom Page
Create Custom Page
Choose the types you want to include on the new page
Search the tree
Business Application 525
Data 155
Processing 8
Data Processing 8
Upload 147
Data Upload 147
Services 370
Invocation 181
Service Invoked 181
Service Call 185
Unhandled Service Call 4
Flight Recorder 1 725
Data Loss 0
Flight Recording 5
Recording Reason 0
Recording Setting 1 720
Java Application 3 731
Statistics 991
Class Loader Statistics 599
Class Loading Statistics 60
OK
Cancel
Count
tty.reactive.HandlerPublisher$ChannelSubscription$$Lambda$969+0x00000...
tty.reactive.HandlerPublisher$ChannelSubscription.lambda$request$0(long)...
tty.reactive.HandlerPublisher$ChannelSubscription.receivedDemand(long):5...
tty.reactive.HandlerPublisher.access$800(HandlerPublisher):65



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.dumpster.uploadtext.UploadTextService			default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	File Write	255,627 ms				default-nioEventLoopGroup-1-17
2023-02-03 11:23:31	File Write	16,755 ms				default-nioEventLoopGroup-1-17

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

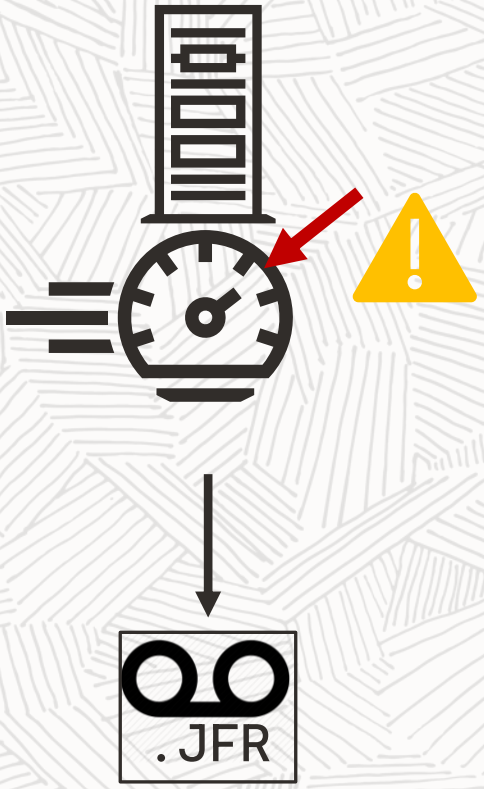
Monitoring CPU level using RecordingStream

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.   }
```

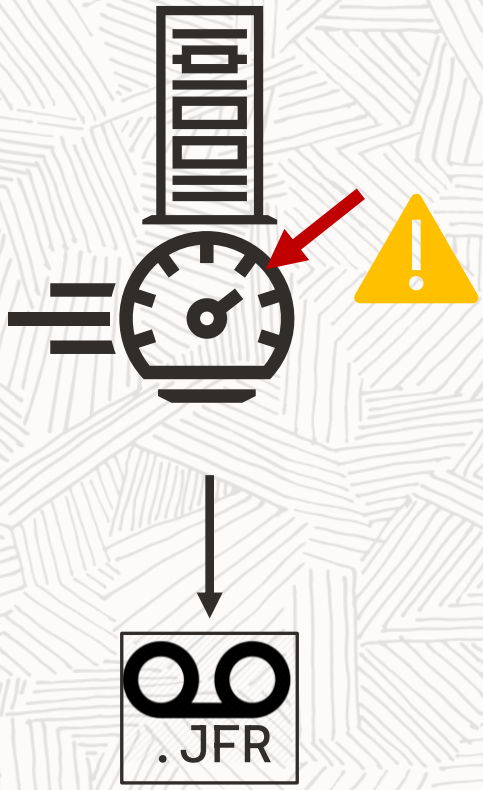
RecordingStream has no events enabled without Configuration

- enable events explicitly, or
- add Configuration

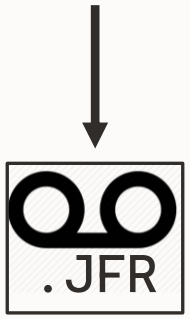
Since we want a JFR file to analyze, we want as many events as possible – use Configuration

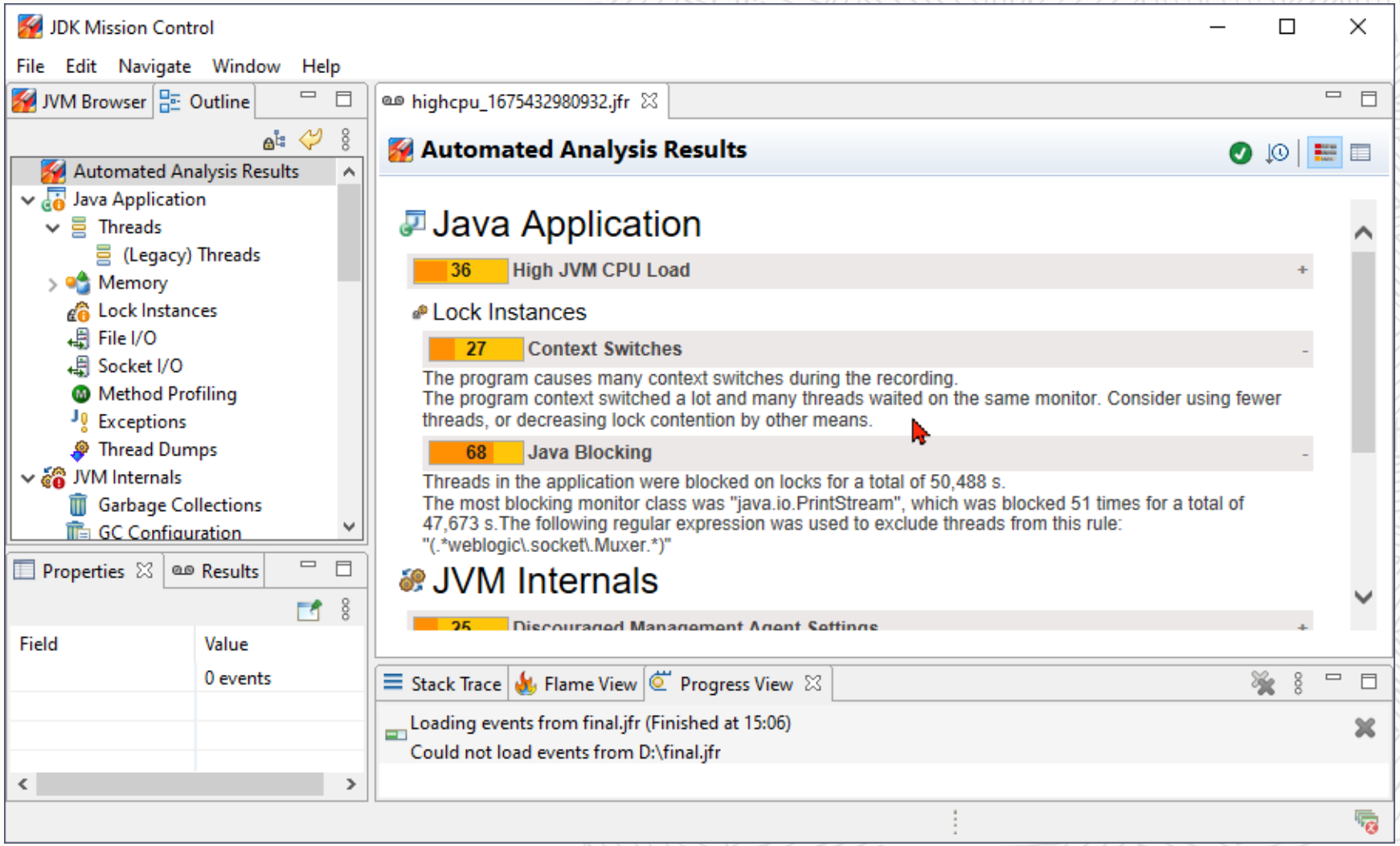
Monitoring CPU level using RecordingStream

```
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.   }
```

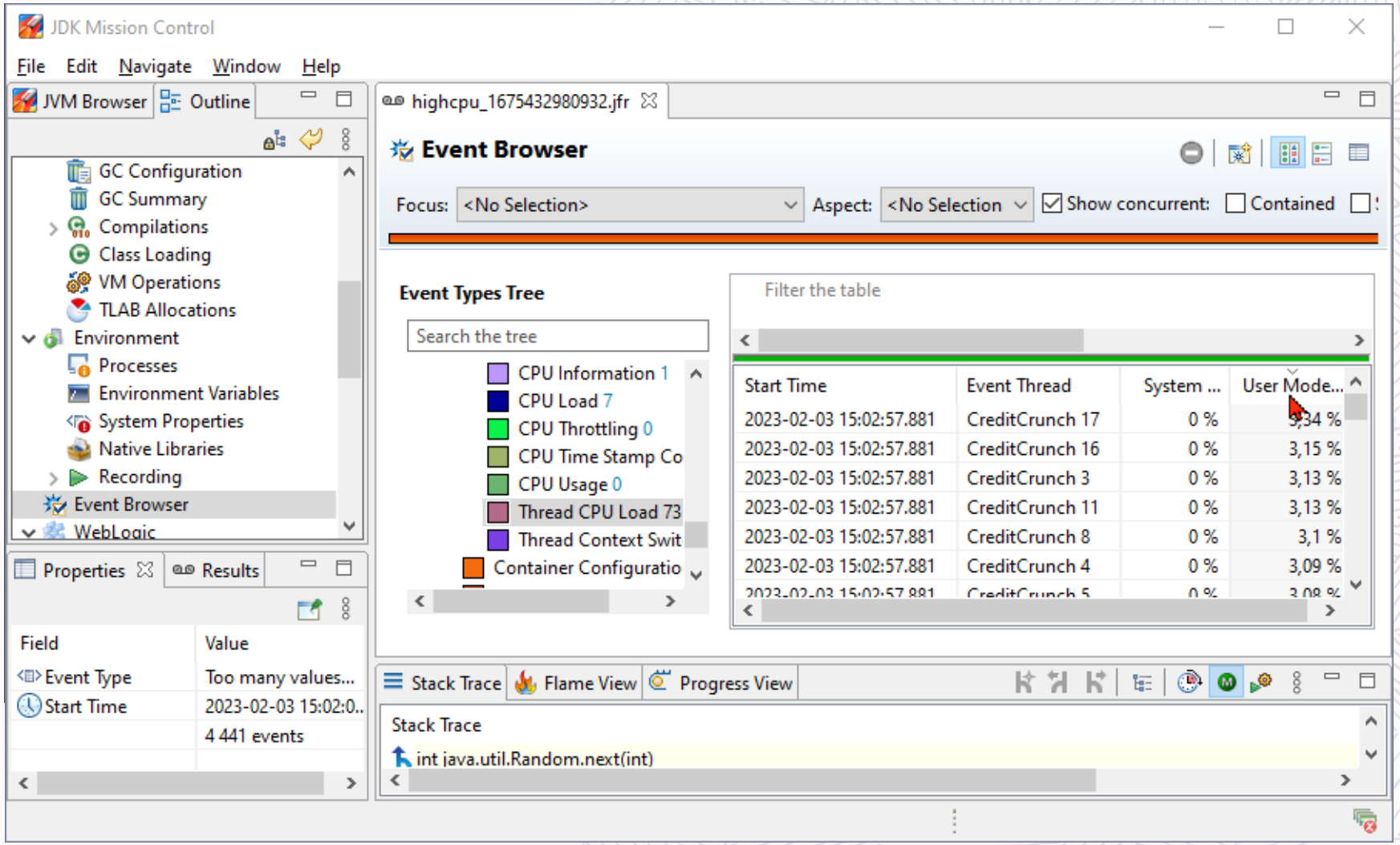


Monitoring CPU level using RecordingStream

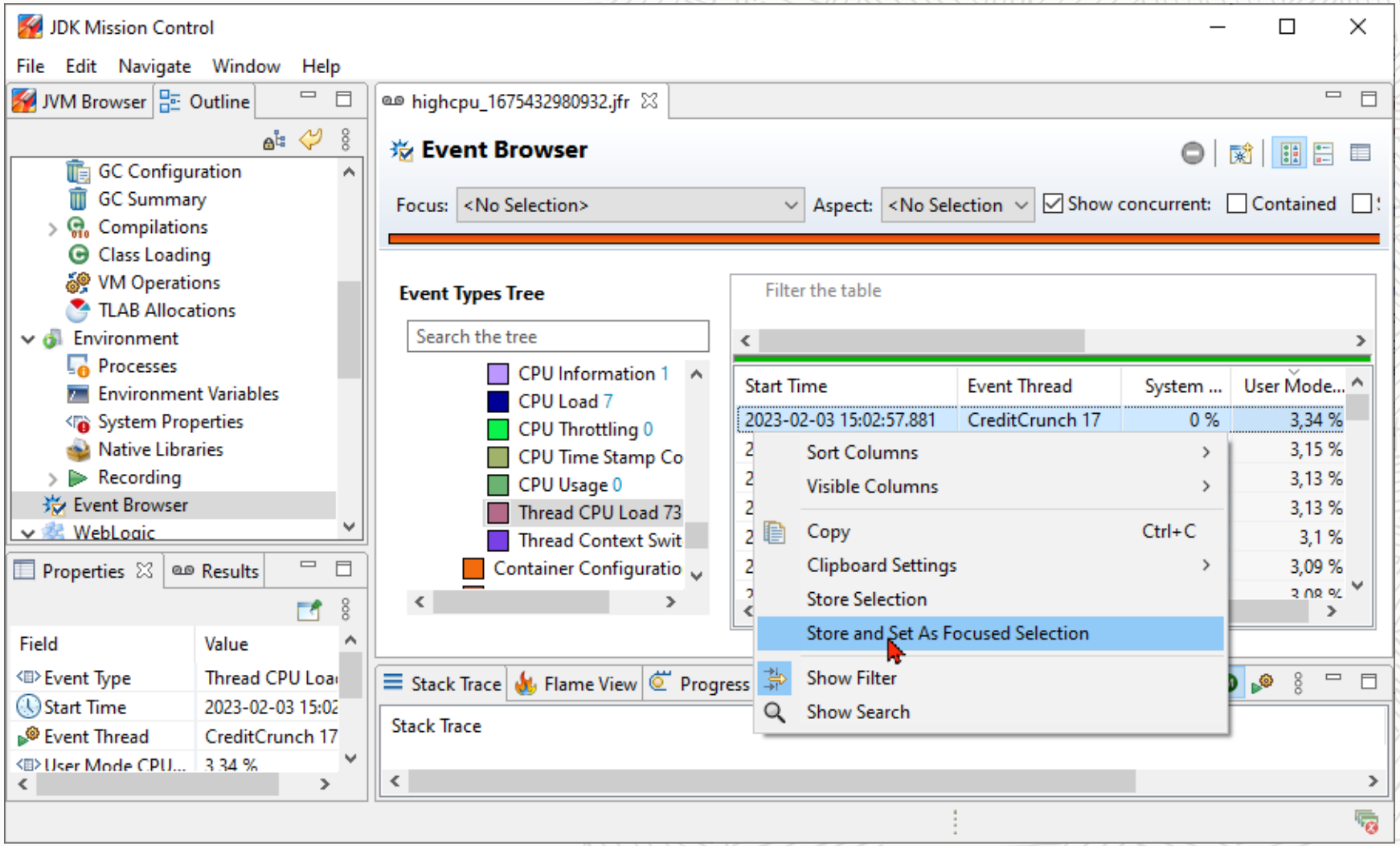




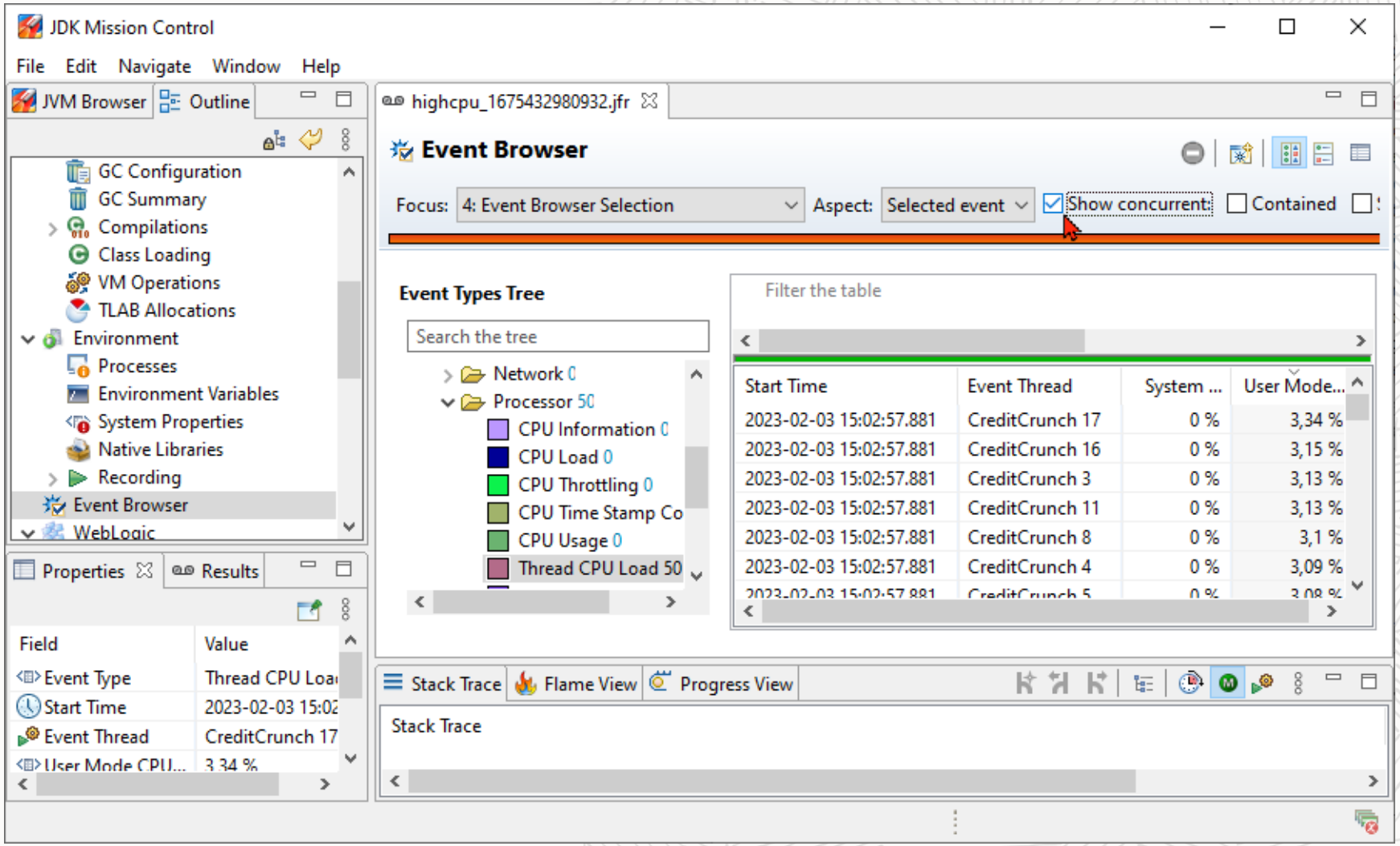
Monitoring CPU level using RecordingStream



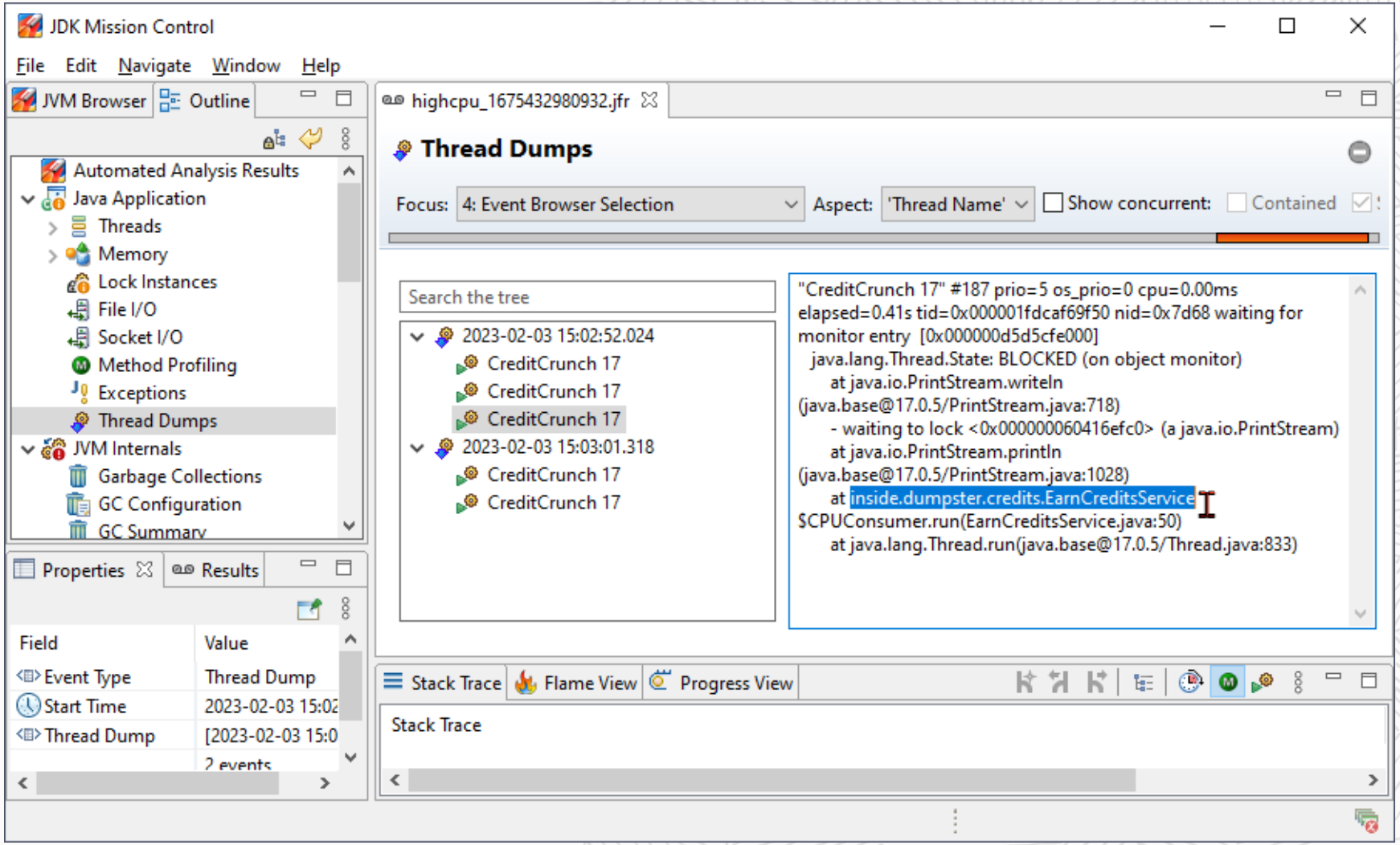
Monitoring CPU level using RecordingStream



Monitoring CPU level using RecordingStream



Monitoring CPU level using RecordingStream



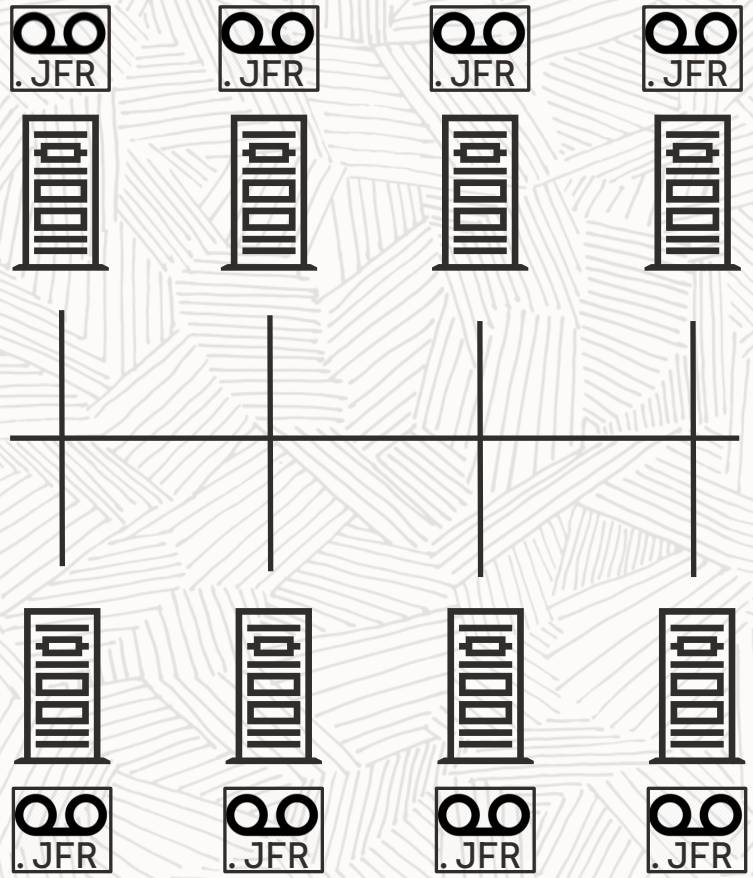
Remotely monitoring CPU level using RemoteRecordingStream

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



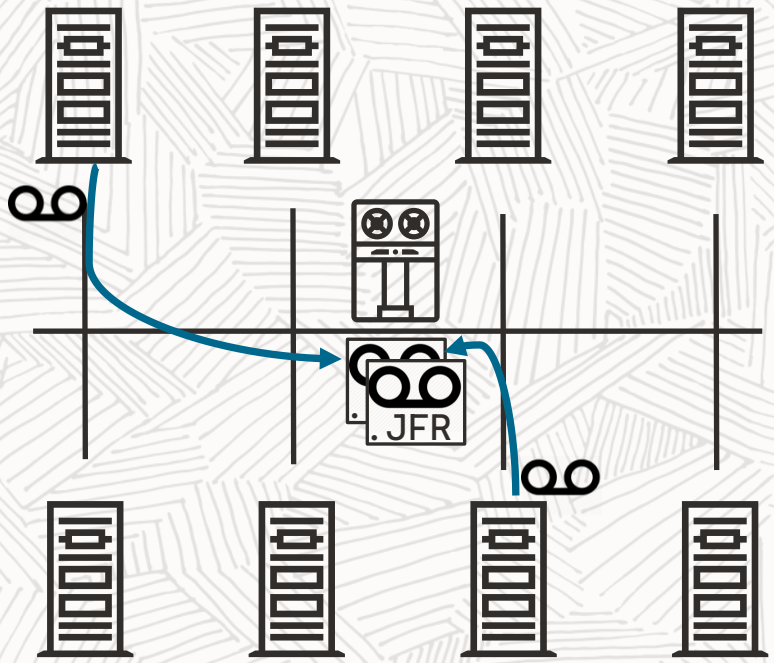
Remotely monitoring CPU level using RemoteRecordingStream

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.   }
```

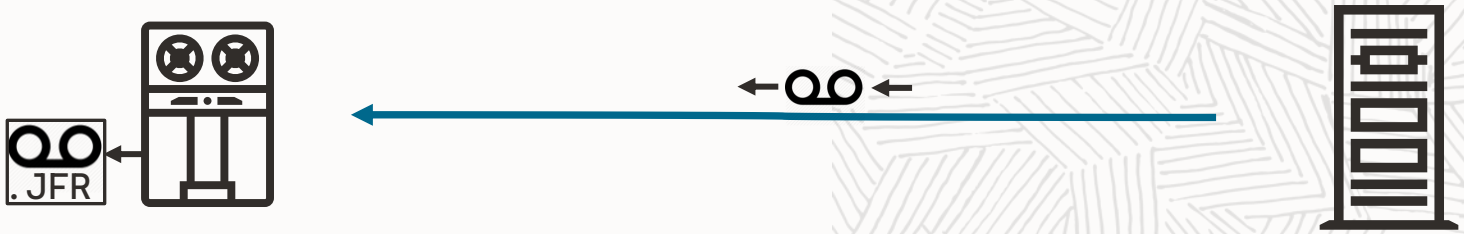

Remotely monitoring CPU level using RemoteRecordingStream

```
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

```
@Override
protected void finalize() throws Throwable
    if (fw != null) fw.close();
    if (file != null) file.delete();
}
```

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

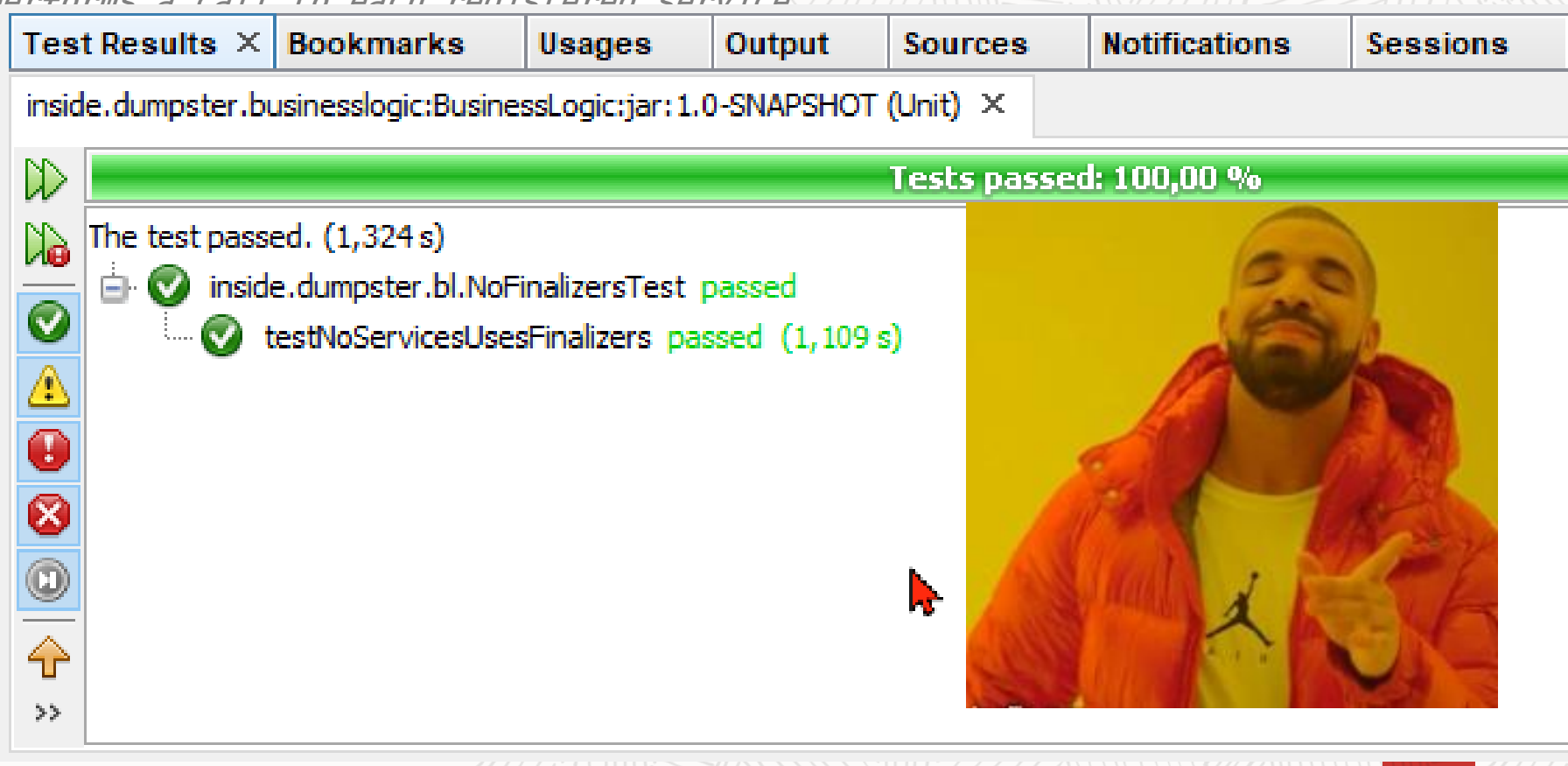
```
@Override
protected void finalize() throws Throwabl
    if (fw != null) fw.close();
    if (file != null) file.delete();
}
```

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. }
```



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

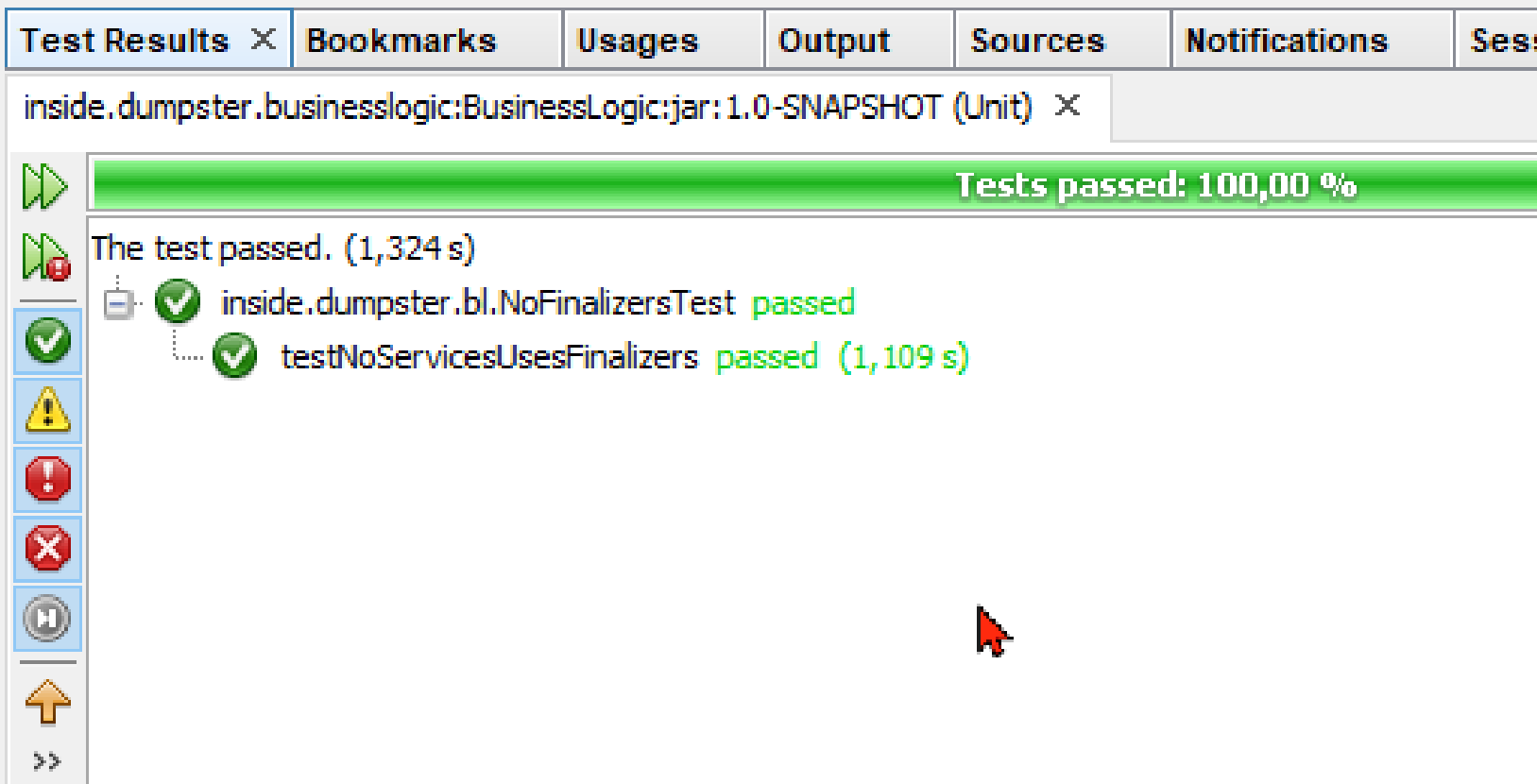
  - Until now!
  - JDK-8295487 JFR: Add stop methods for recording streams
  - NB: Will come in JDK 20

Using JFR events in unit tests -- RecordingStream


```
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


```
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. }
```

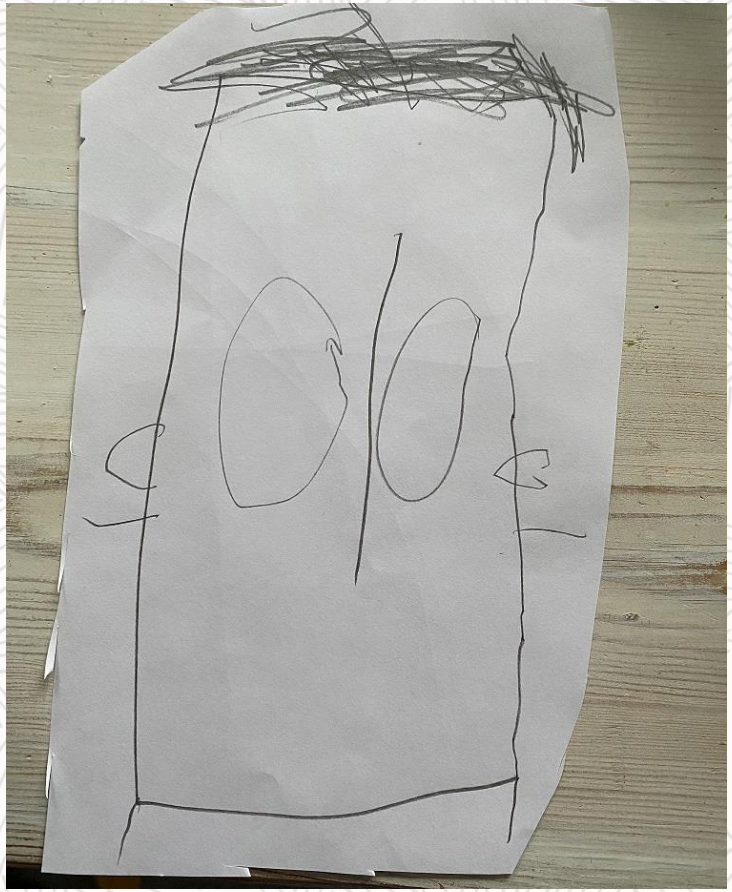


Agenda

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