Debugging and Profiling

UNIVERSITÀ DELLA CALABRIA

DIMES - Dipartimento di INGEGNERIA INFORMATICA MODELLISTICA, ELETTRONICA E SISTEMISTICA

Ing. Ludovica Sacco
DIMES – UNICAL - 87036 Rende(CS) - Italy
Email: l.sacco@dimes.unical.it

Debugging

Debugging

Debugging is the process of finding errors in the software, regardless of what kind of errors they are.

The developer explores the source code and the variables during software execution and can understand if the behaviour of an individual instruction or groups of instructions is correct or not.

So debugging is useful to identify problematic parts of code, isolating them and fixing bugs.

Debugger

It is software that allows to run an application interactively.

The developer can observe the source code and variables during execution.
It is possible to apply breakpoints at certain lines of code or instance fields to communicate to the debugger when to freeze the execution.

Once the program has been temporarily stopped, the developer can observe the portion of the code currently running and the status of the variables, thus being able to read their value and, if desired, modify it.

Debugging

Eclipse allows you to start a Java program in *Debug mode*.
Eclipse provides a *Debug perspective* which gives you a pre-configured set of *views*. Eclipse allows you to control the execution flow via debug commands.

Operations:
- Setting breakpoints
- Starting the debugger
- Controlling the program execution
- Evaluating variables
- Setting watchpoints

https://www.jetbrains.com/help/idea/debugging-code.html

Controlling the program execution

CMD	IntelliJ	Eclipse	Description
STEP INTO	F7	F5	It xecutes the currently selected line and goes to the next line in your program. If the selected line is a method call the debugger steps into the associated code.
STEP OVER	F8	F6	It steps over the call, i.e. it executes a method without stepping into it in the debugger.
STEP OUT	SHIFT+ F8	F7	It steps out to the caller of the currently executed method. This finishes the execution of the current method and returns to the caller of this method.
RESUME	F9	F8	It tells the Eclipse debugger to resume the execution of the program code until is reaches the next breakpoint or watchpoint.

Profiling

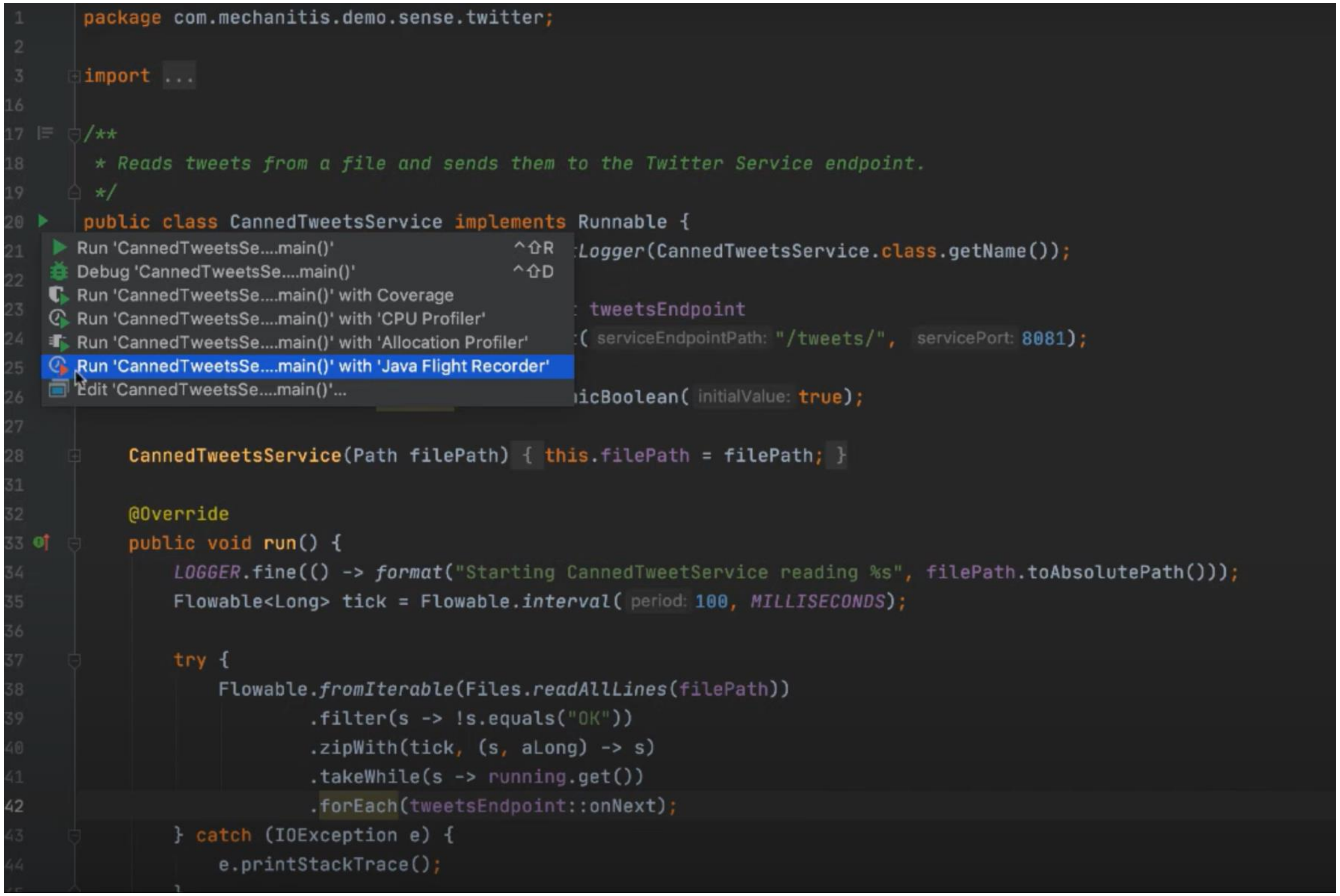
Profiling

It is a software designed to monitor the activity of a program with the aim of analysing its performance, i.e. in terms of processor or memory use.

What is it for?
It is very useful as it allows you to solve performance problems and optimize the program, from a temporal point of view or from the point of view of memory occupation.

Sometimes your app works, but you want to increase performance by boosting its throughput or reducing latency. Other times, you just want to know how code behaves at runtime, determine where the hot spots are, or figure out how a framework operates under the hood.

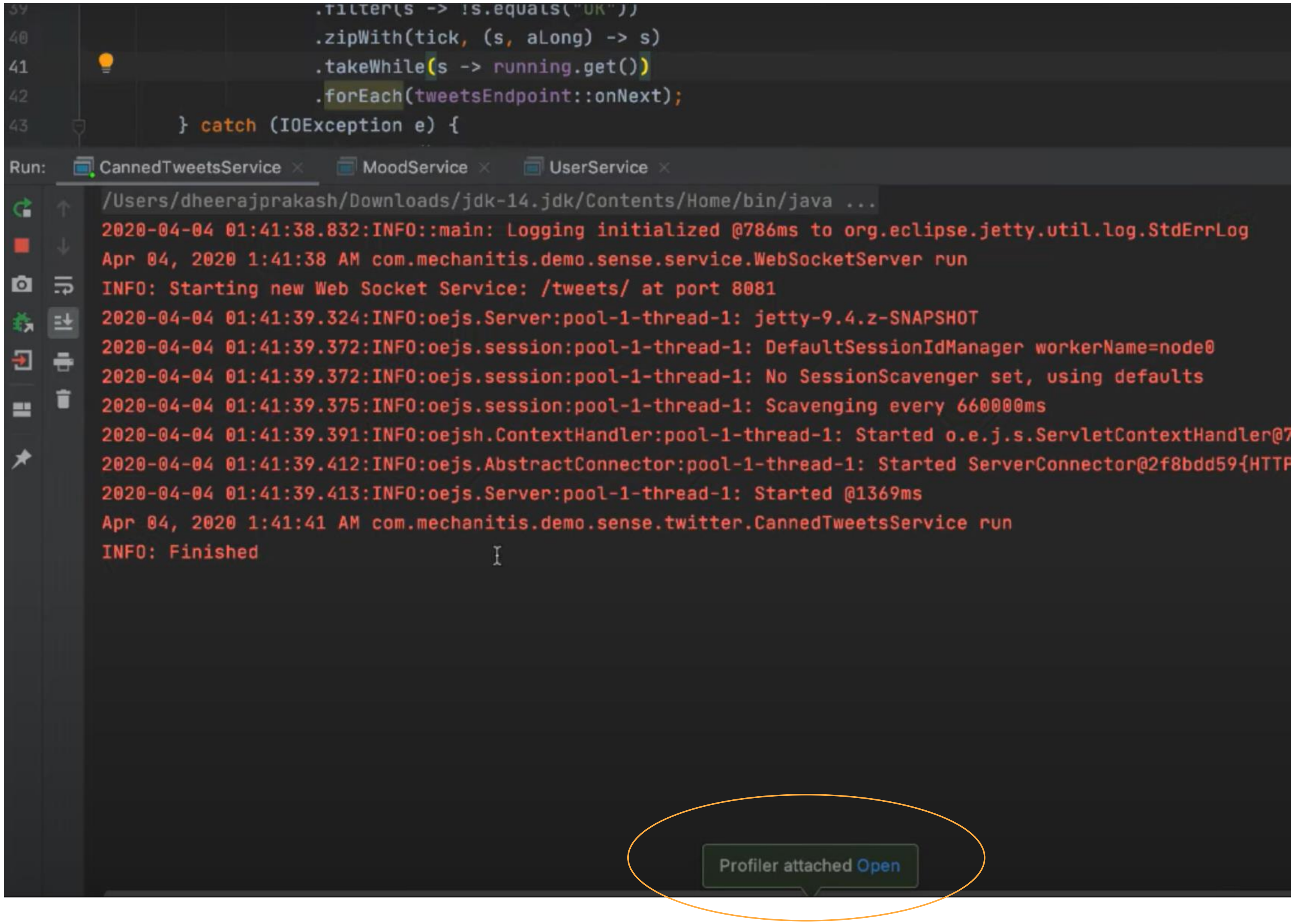
Profiling


Run 'CannedTweetsSe....main()' ^⇧R
Debug 'CannedTweetsSe....main()' ^⇧D
Run 'CannedTweetsSe....main()' with Coverage
Run 'CannedTweetsSe....main()' with 'CPU Profiler'
Run 'CannedTweetsSe....main()' with 'Allocation Profiler'
Run 'CannedTweetsSe....main()' with 'Java Flight Recorder'
Edit 'CannedTweetsSe....main()'...

```
package com.mechanitis.demo.sense.twitter;

import ...

/**
 * Reads tweets from a file and sends them to the Twitter Service endpoint.
 */
public class CannedTweetsService implements Runnable {
                                                        :Logger(CannedTweetsService.class.getName());

                                                         tweetsEndpoint
                                                        ( serviceEndpointPath: "/tweets/",  servicePort: 8081);

                                                        icBoolean( initialValue: true);

    CannedTweetsService(Path filePath) { this.filePath = filePath; }

    @Override
    public void run() {
        LOGGER.fine(() -> format("Starting CannedTweetService reading %s", filePath.toAbsolutePath()));
        Flowable<Long> tick = Flowable.interval( period: 100, MILLISECONDS);

        try {
            Flowable.fromIterable(Files.readAllLines(filePath))
                    .filter(s -> !s.equals("OK"))
                    .zipWith(tick, (s, aLong) -> s)
                    .takeWhile(s -> running.get())
                    .forEach(tweetsEndpoint::onNext);
        } catch (IOException e) {
            e.printStackTrace();
```

https://blog.jetbrains.com/idea/2020/03/profiling-tools-and-intellij-idea-ultimate/

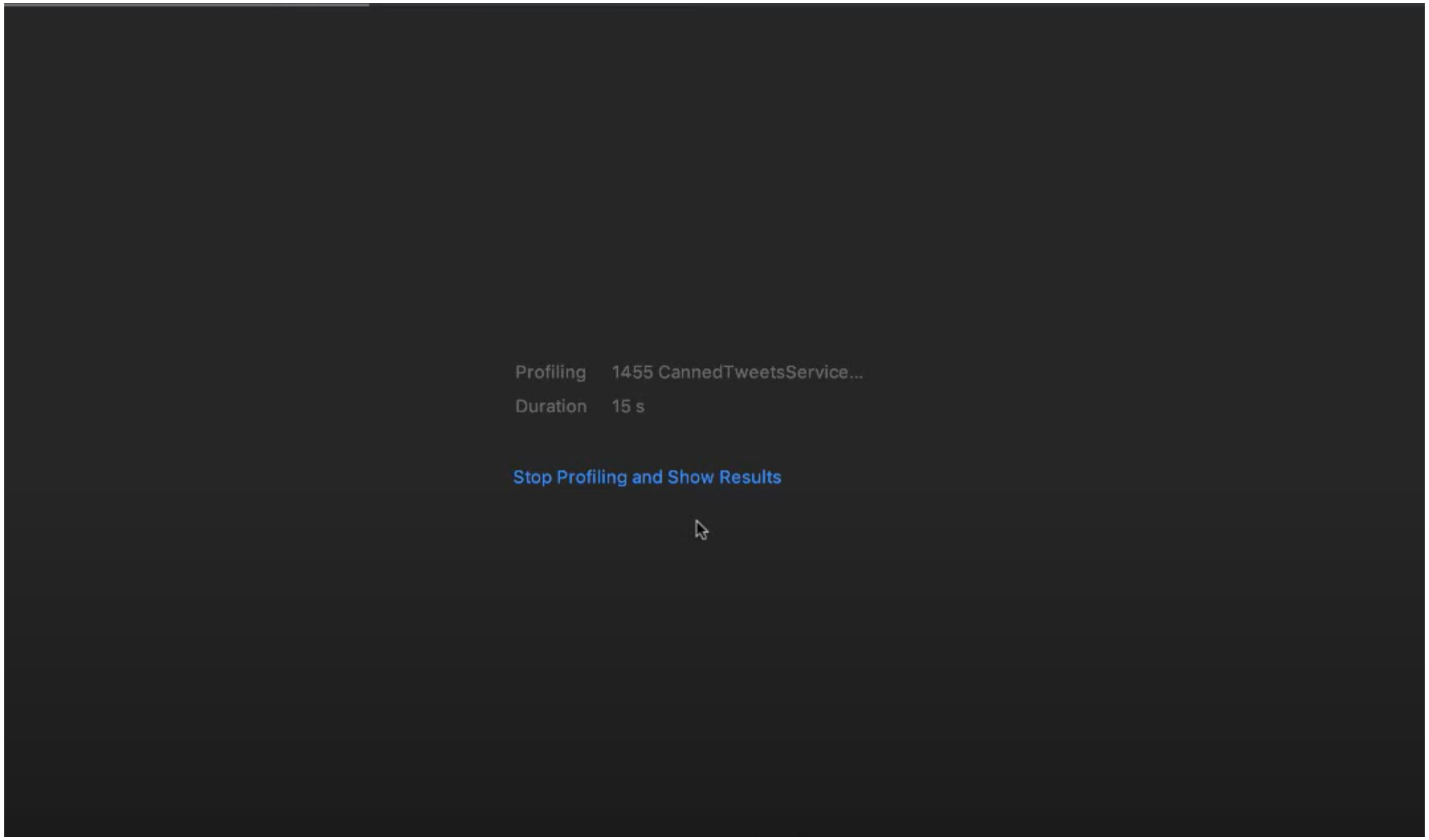
Profiling



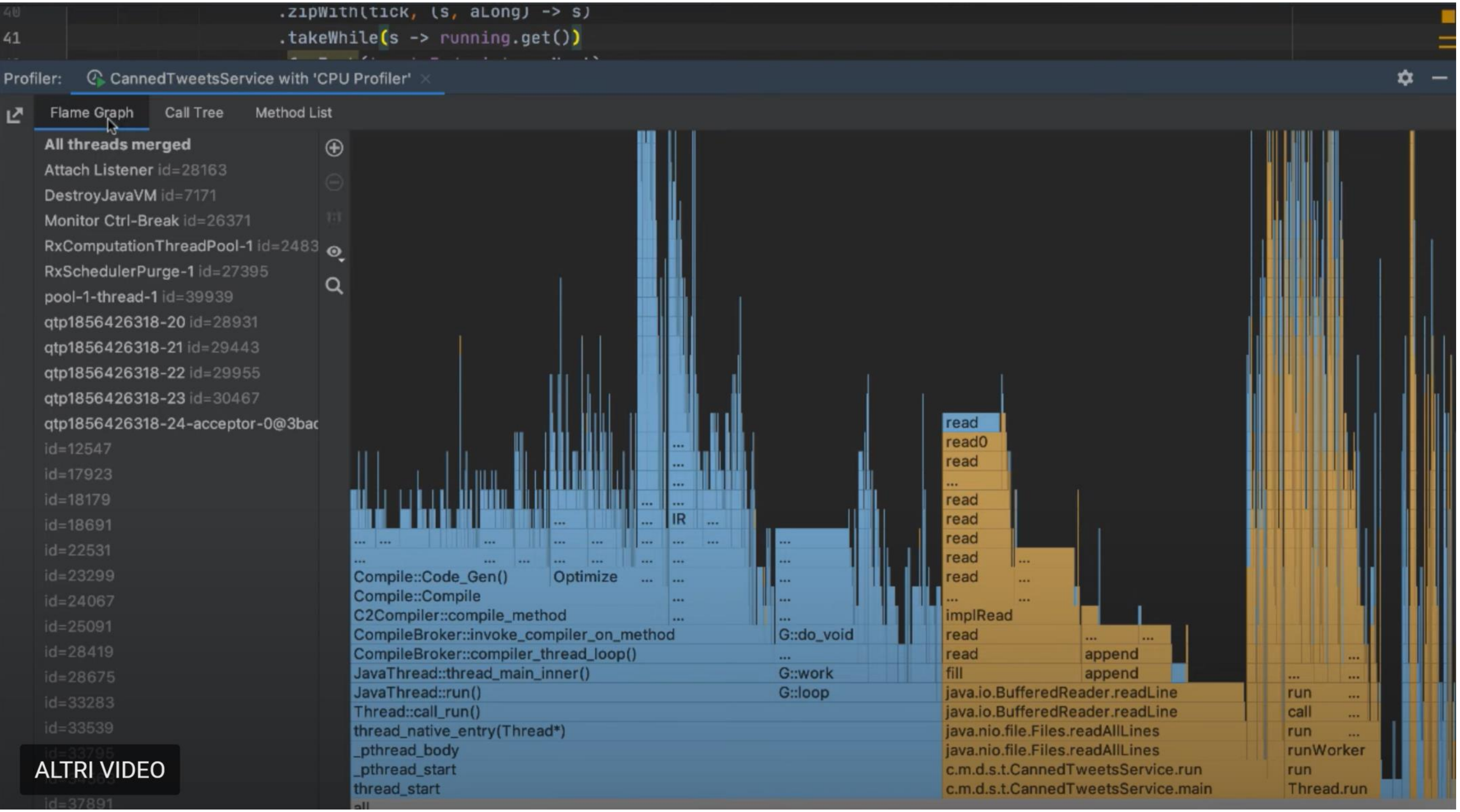
```
                    .filter(s -> !s.equals("OK"))
                    .zipWith(tick, (s, aLong) -> s)
                    .takeWhile(s -> running.get())
                    .forEach(tweetsEndpoint::onNext);
        } catch (IOException e) {
```

```
/Users/dheerajprakash/Downloads/jdk-14.jdk/Contents/Home/bin/java ...
2020-04-04 01:41:38.832:INFO::main: Logging initialized @786ms to org.eclipse.jetty.util.log.StdErrLog
Apr 04, 2020 1:41:38 AM com.mechanitis.demo.sense.service.WebSocketServer run
INFO: Starting new Web Socket Service: /tweets/ at port 8081
2020-04-04 01:41:39.324:INFO:oejs.Server:pool-1-thread-1: jetty-9.4.z-SNAPSHOT
2020-04-04 01:41:39.372:INFO:oejs.session:pool-1-thread-1: DefaultSessionIdManager workerName=node0
2020-04-04 01:41:39.372:INFO:oejs.session:pool-1-thread-1: No SessionScavenger set, using defaults
2020-04-04 01:41:39.375:INFO:oejs.session:pool-1-thread-1: Scavenging every 660000ms
2020-04-04 01:41:39.391:INFO:oejsh.ContextHandler:pool-1-thread-1: Started o.e.j.s.ServletContextHandler@7
2020-04-04 01:41:39.412:INFO:oejs.AbstractConnector:pool-1-thread-1: Started ServerConnector@2f8bdd59{HTTP
2020-04-04 01:41:39.413:INFO:oejs.Server:pool-1-thread-1: Started @1369ms
Apr 04, 2020 1:41:41 AM com.mechanitis.demo.sense.twitter.CannedTweetsService run
INFO: Finished
```

Profiling


Profiling 1455 CannedTweetsService...
Duration 15 s

Profiling: Flame Graph



```
            .zipWith(tick, (s, aLong) -> s)
            .takeWhile(s -> running.get())
```

Profiler: CannedTweetsService with 'CPU Profiler'

Flame Graph | Call Tree | Method List

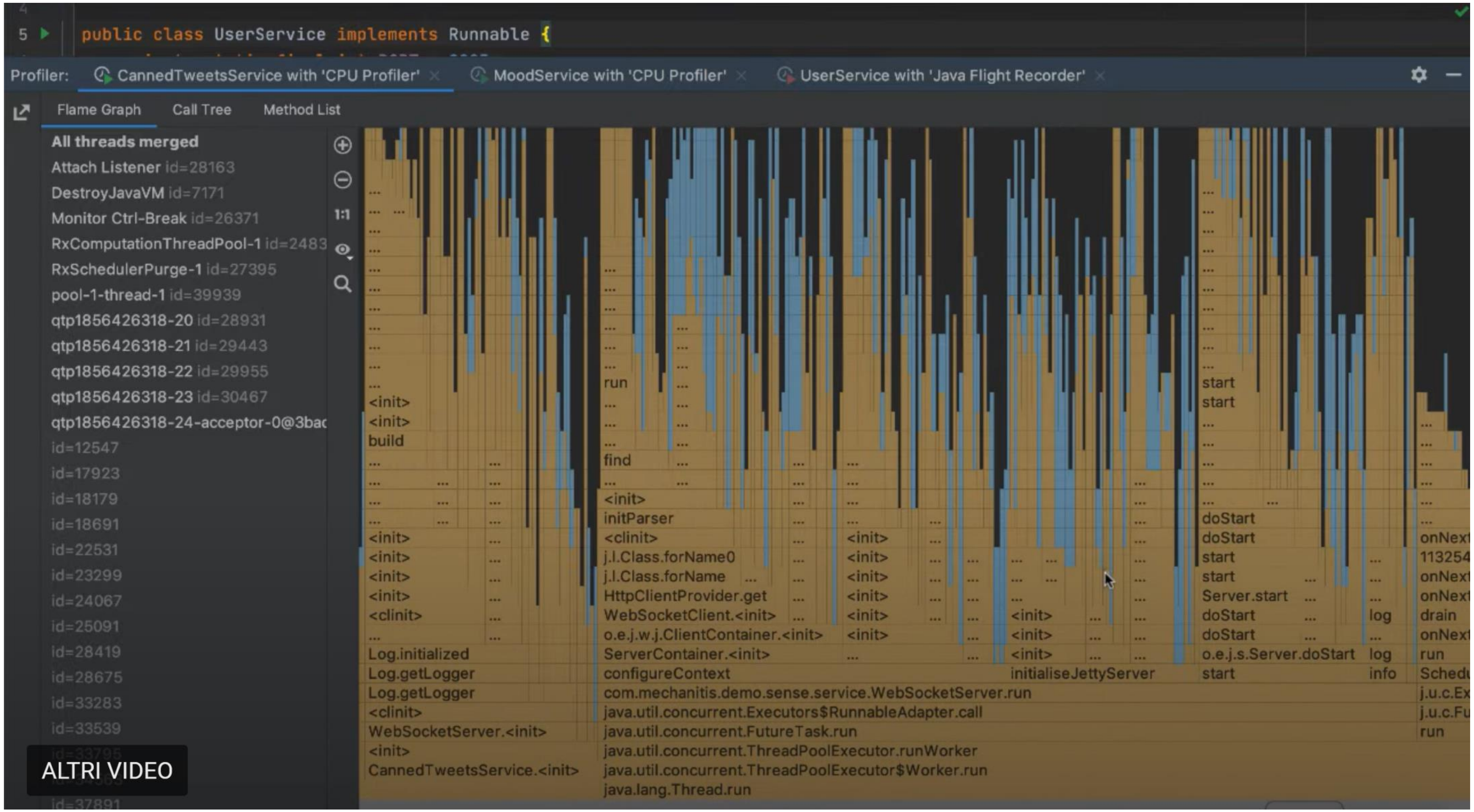
All threads merged
Attach Listener id=28163
DestroyJavaVM id=7171
Monitor Ctrl-Break id=26371
RxComputationThreadPool-1 id=2483
RxSchedulerPurge-1 id=27395
pool-1-thread-1 id=39939
qtp1856426318-20 id=28931
qtp1856426318-21 id=29443
qtp1856426318-22 id=29955
qtp1856426318-23 id=30467
qtp1856426318-24-acceptor-0@3bac
id=12547
id=17923
id=18179
id=18691
id=22531
id=23299
id=24067
id=25091
id=28419
id=28675
id=33283
id=33539
id=37891

ALTRI VIDEO
IR
Compile::Code_Gen()
Optimize
Compile::Compile
C2Compiler::compile_method
CompileBroker::invoke_compiler_on_method
CompileBroker::compiler_thread_loop()
JavaThread::thread_main_inner()
JavaThread::run()
Thread::call_run()
thread_native_entry(Thread*)
_pthread_body
_pthread_start
thread_start
all
G::do_void
G::work
G::loop
read
read0
read
read
read
read
read
read
implRead
read
read
fill
append
append
java.io.BufferedReader.readLine
java.io.BufferedReader.readLine
java.nio.file.Files.readAllLines
java.nio.file.Files.readAllLines
c.m.d.s.t.CannedTweetsService.run
c.m.d.s.t.CannedTweetsService.main
run
call
run
runWorker
run
Thread.run

In blue → native calls

In yellow → java calls

Profiling: Flame Graph



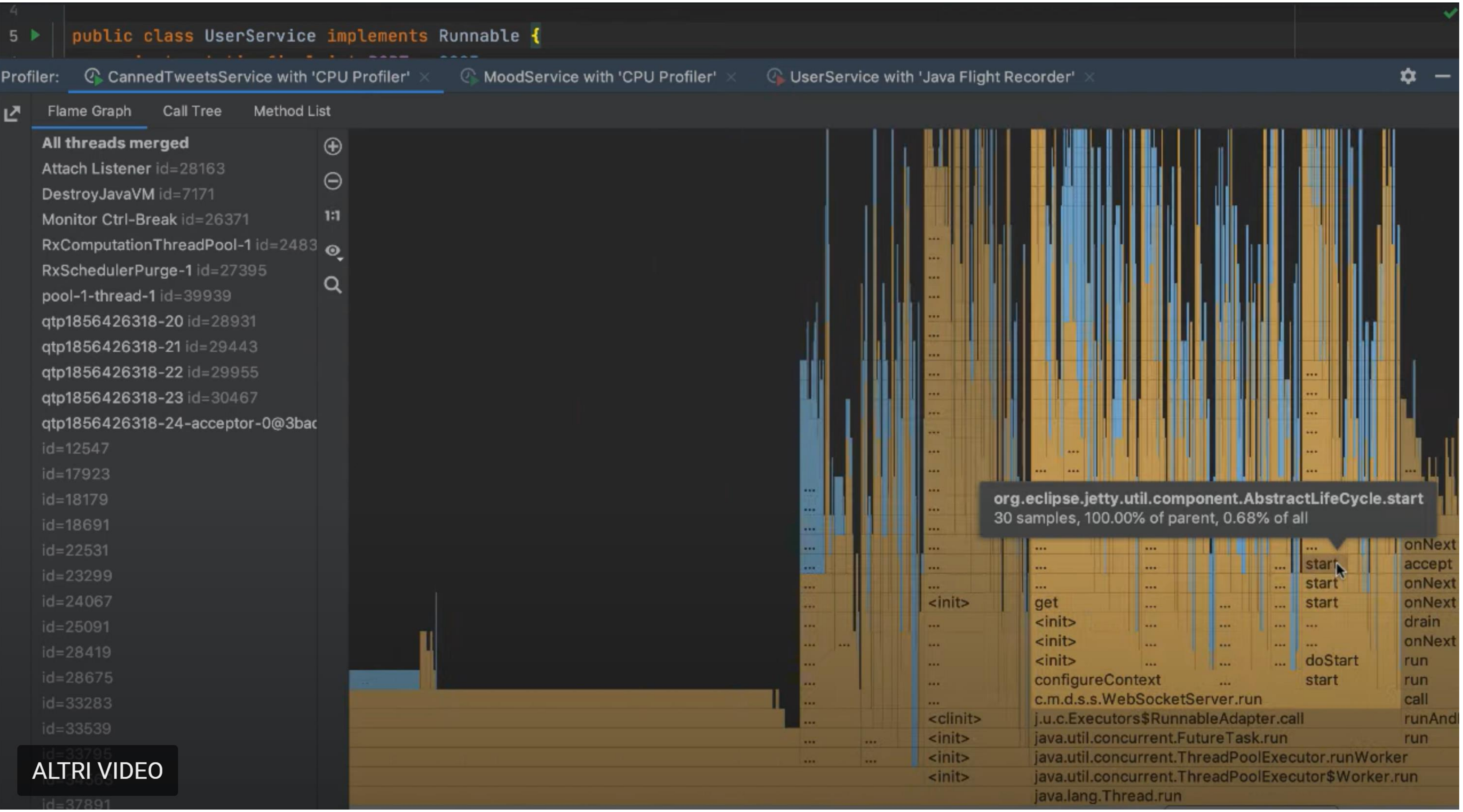
```
public class UserService implements Runnable {
```

Profiler:
CannedTweetsService with 'CPU Profiler'
MoodService with 'CPU Profiler'
UserService with 'Java Flight Recorder'
Flame Graph
Call Tree
Method List
All threads merged
Attach Listener id=28163
DestroyJavaVM id=7171
Monitor Ctrl-Break id=26371
RxComputationThreadPool-1 id=2483
RxSchedulerPurge-1 id=27395
pool-1-thread-1 id=39939
qtp1856426318-20 id=28931
qtp1856426318-21 id=29443
qtp1856426318-22 id=29955
qtp1856426318-23 id=30467
qtp1856426318-24-acceptor-0@3bac
id=12547
id=17923
id=18179
id=18691
id=22531
id=23299
id=24067
id=25091
id=28419
id=28675
id=33283
id=33539
ALTRI VIDEO
CannedTweetsService.<init>
WebSocketServer.<init>
Log.getLogger
Log.initialized
java.lang.Thread.run
java.util.concurrent.ThreadPoolExecutor$Worker.run
java.util.concurrent.ThreadPoolExecutor.runWorker
java.util.concurrent.FutureTask.run
java.util.concurrent.Executors$RunnableAdapter.call
com.mechanitis.demo.sense.service.WebSocketServer.run
configureContext
ServerContainer.<init>
o.e.j.w.j.ClientContainer.<init>
WebSocketClient.<init>
HttpClientProvider.get
j.l.Class.forName
j.l.Class.forName0
<clinit>
initParser
find
run
initialiseJettyServer
start
o.e.j.s.Server.doStart
doStart
Server.start
log
info
onNext
drain
Sched

It does not show the sequence of the calling of methods,
but it shows which methods are calling other methods.
IT IS NOT A TIME SERIES!

It is a snapshot of what is happening in the CPU.

Profiling: Flame Graph



```
public class UserService implements Runnable {
```

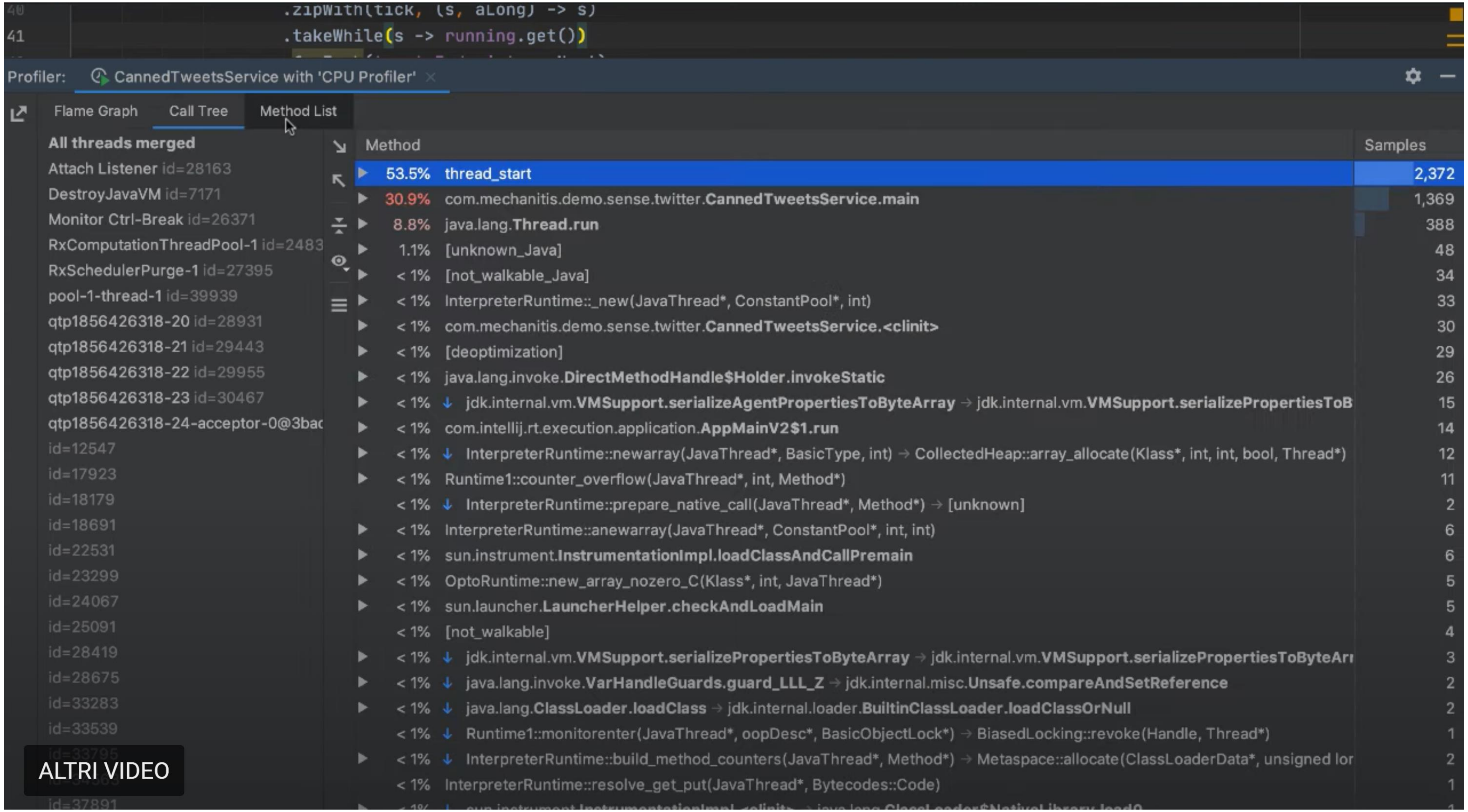
Profiler:
CannedTweetsService with 'CPU Profiler'
MoodService with 'CPU Profiler'
UserService with 'Java Flight Recorder'
Flame Graph
Call Tree
Method List
All threads merged
Attach Listener id=28163
DestroyJavaVM id=7171
Monitor Ctrl-Break id=26371
RxComputationThreadPool-1 id=2483
RxSchedulerPurge-1 id=27395
pool-1-thread-1 id=39939
qtp1856426318-20 id=28931
qtp1856426318-21 id=29443
qtp1856426318-22 id=29955
qtp1856426318-23 id=30467
qtp1856426318-24-acceptor-0@3bac
id=12547
id=17923
id=18179
id=18691
id=22531
id=23299
id=24067
id=25091
id=28419
id=28675
id=33283
id=33539
id=37891
ALTRI VIDEO

org.eclipse.jetty.util.component.AbstractLifeCycle.start
30 samples, 100.00% of parent, 0.68% of all

start
start
start
doStart
start
get
<init>
<init>
<init>
configureContext
c.m.d.s.s.WebSocketServer.run
j.u.c.Executors$RunnableAdapter.call
java.util.concurrent.FutureTask.run
java.util.concurrent.ThreadPoolExecutor.runWorker
java.util.concurrent.ThreadPoolExecutor$Worker.run
java.lang.Thread.run
<clinit>
onNext
accept
onNext
onNext
drain
onNext
run
run
call
runAndl
run

If the method spends a lot of CPU, it does not necessarily mean that is not well coded.
Perhaps, it is called many times.

Profiling: Call Tree



```
                    .zipWith(tick, (s, aLong) -> s)
                    .takeWhile(s -> running.get())
```

Profiler: CannedTweetsService with 'CPU Profiler'

Flame Graph | Call Tree | Method List

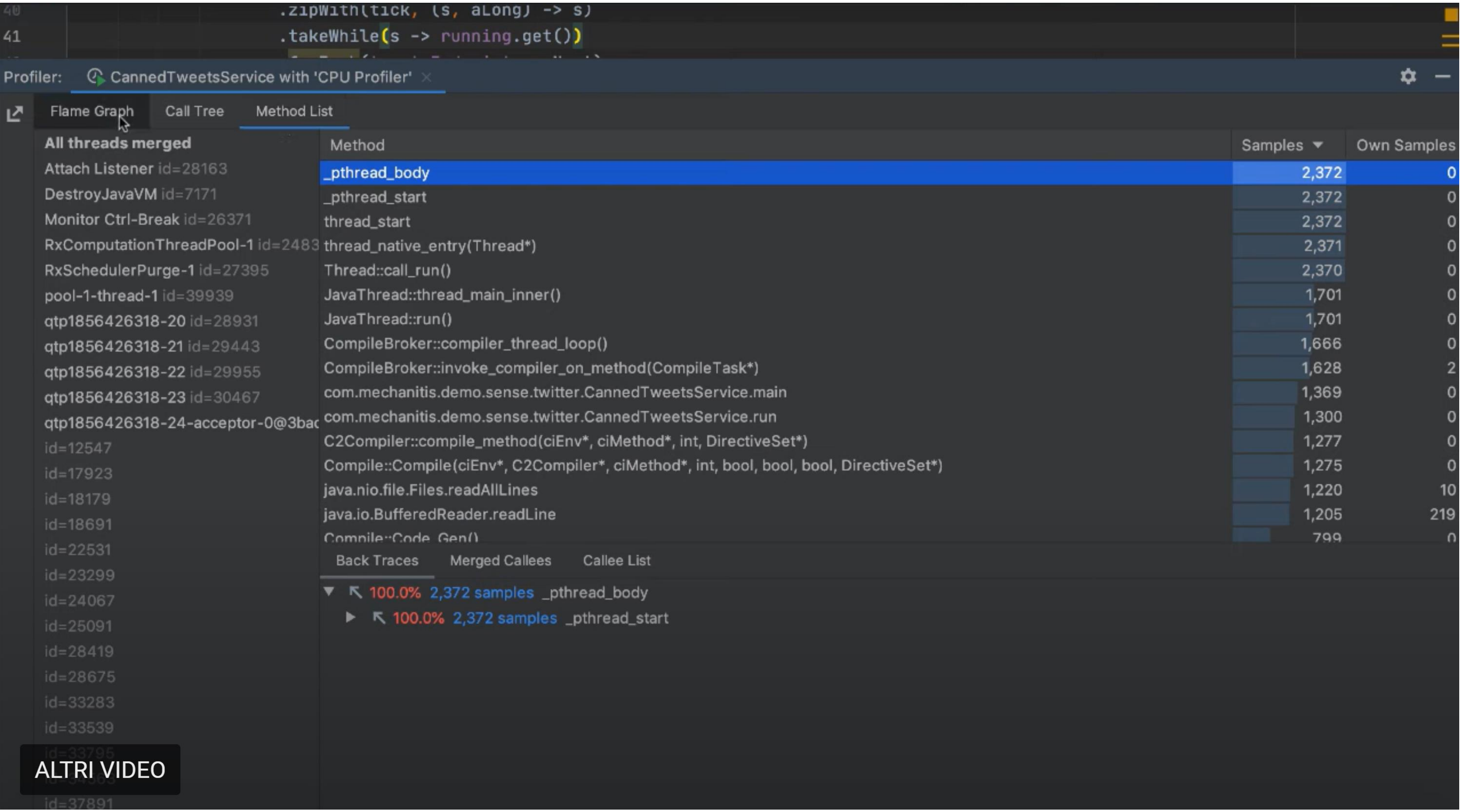
All threads merged
Attach Listener id=28163
DestroyJavaVM id=7171
Monitor Ctrl-Break id=26371
RxComputationThreadPool-1 id=2483
RxSchedulerPurge-1 id=27395
pool-1-thread-1 id=39939
qtp1856426318-20 id=28931
qtp1856426318-21 id=29443
qtp1856426318-22 id=29955
qtp1856426318-23 id=30467
qtp1856426318-24-acceptor-0@3bac
id=12547
id=17923
id=18179
id=18691
id=22531
id=23299
id=24067
id=25091
id=28419
id=28675
id=33283
id=33539

	Method	Samples
53.5%	thread_start	2,372
30.9%	com.mechanitis.demo.sense.twitter.**CannedTweetsService.main**	1,369
8.8%	java.lang.**Thread.run**	388
1.1%	[unknown_Java]	48
< 1%	[not_walkable_Java]	34
< 1%	InterpreterRuntime::_new(JavaThread*, ConstantPool*, int)	33
< 1%	com.mechanitis.demo.sense.twitter.**CannedTweetsService.<clinit>**	30
< 1%	[deoptimization]	29
< 1%	java.lang.invoke.**DirectMethodHandle$Holder.invokeStatic**	26
< 1%	↓ jdk.internal.vm.**VMSupport.serializeAgentPropertiesToByteArray** → jdk.internal.vm.**VMSupport.serializePropertiesToB**	15
< 1%	com.intellij.rt.execution.application.**AppMainV2$1.run**	14
< 1%	↓ InterpreterRuntime::newarray(JavaThread*, BasicType, int) → CollectedHeap::array_allocate(Klass*, int, int, bool, Thread*)	12
< 1%	Runtime1::counter_overflow(JavaThread*, int, Method*)	11
< 1%	↓ InterpreterRuntime::prepare_native_call(JavaThread*, Method*) → [unknown]	2
< 1%	InterpreterRuntime::anewarray(JavaThread*, ConstantPool*, int, int)	6
< 1%	sun.instrument.**InstrumentationImpl.loadClassAndCallPremain**	6
< 1%	OptoRuntime::new_array_nozero_C(Klass*, int, JavaThread*)	5
< 1%	sun.launcher.**LauncherHelper.checkAndLoadMain**	5
< 1%	[not_walkable]	4
< 1%	↓ jdk.internal.vm.**VMSupport.serializePropertiesToByteArray** → jdk.internal.vm.**VMSupport.serializePropertiesToByteArr**	3
< 1%	↓ java.lang.invoke.**VarHandleGuards.guard_LLL_Z** → jdk.internal.misc.**Unsafe.compareAndSetReference**	2
< 1%	↓ java.lang.**ClassLoader.loadClass** → jdk.internal.loader.**BuiltinClassLoader.loadClassOrNull**	2
< 1%	↓ Runtime1::monitorenter(JavaThread*, oopDesc*, BasicObjectLock*) → BiasedLocking::revoke(Handle, Thread*)	1
< 1%	↓ InterpreterRuntime::build_method_counters(JavaThread*, Method*) → Metaspace::allocate(ClassLoaderData*, unsigned lor	2
< 1%	InterpreterRuntime::resolve_get_put(JavaThread*, Bytecodes::Code)	1

ALTRI VIDEO

How methods are called and the percentage of total CPU time used by them.

Profiling: Method List



```
.zipWith(tick, (s, aLong) -> s)
.takeWhile(s -> running.get())
```

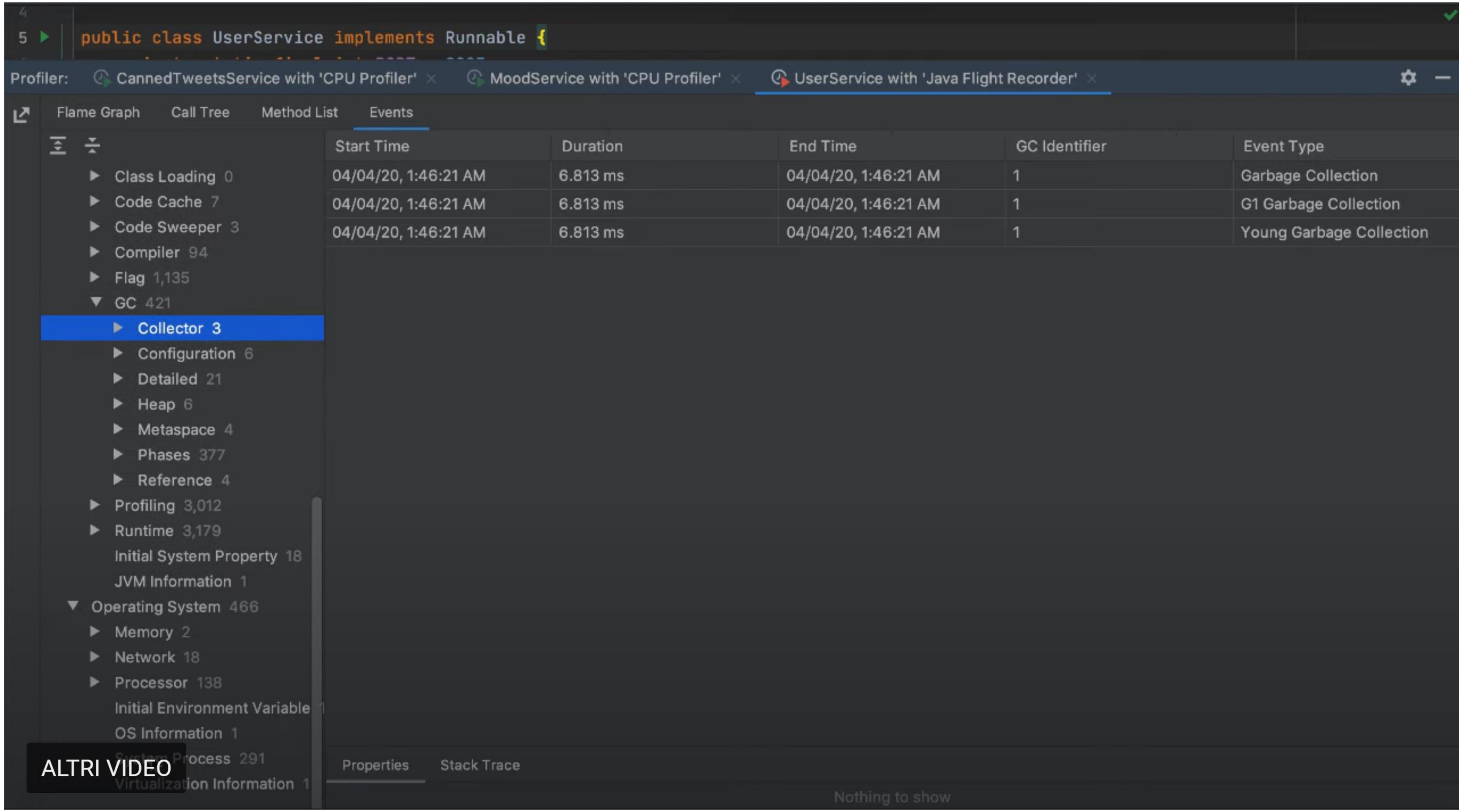
All threads merged
Attach Listener id=28163
DestroyJavaVM id=7171
Monitor Ctrl-Break id=26371
RxComputationThreadPool-1 id=2483
RxSchedulerPurge-1 id=27395
pool-1-thread-1 id=39939
qtp1856426318-20 id=28931
qtp1856426318-21 id=29443
qtp1856426318-22 id=29955
qtp1856426318-23 id=30467
qtp1856426318-24-acceptor-0@3bac
id=12547
id=17923
id=18179
id=18691
id=22531
id=23299
id=24067
id=25091
id=28419
id=28675
id=33283
id=33539
id=37891

Method	Samples	Own Samples
_pthread_body	2,372	0
_pthread_start	2,372	0
thread_start	2,372	0
thread_native_entry(Thread*)	2,371	0
Thread::call_run()	2,370	0
JavaThread::thread_main_inner()	1,701	0
JavaThread::run()	1,701	0
CompileBroker::compiler_thread_loop()	1,666	0
CompileBroker::invoke_compiler_on_method(CompileTask*)	1,628	2
com.mechanitis.demo.sense.twitter.CannedTweetsService.main	1,369	0
com.mechanitis.demo.sense.twitter.CannedTweetsService.run	1,300	0
C2Compiler::compile_method(ciEnv*, ciMethod*, int, DirectiveSet*)	1,277	0
Compile::Compile(ciEnv*, C2Compiler*, ciMethod*, int, bool, bool, bool, DirectiveSet*)	1,275	0
java.nio.file.Files.readAllLines	1,220	10
java.io.BufferedReader.readLine	1,205	219
Compile::Code_Gen()	799	0

```
▼ ↖ 100.0% 2,372 samples _pthread_body
  ▶ ↖ 100.0% 2,372 samples _pthread_start
```

Methods called while profiling.

Profiling: Events



```java
public class UserService implements Runnable {
```

Profiler:
CannedTweetsService with 'CPU Profiler'
MoodService with 'CPU Profiler'
UserService with 'Java Flight Recorder'
Flame Graph
Call Tree
Method List
Events
Properties
Stack Trace
Nothing to show
ALTRI VIDEO

- Class Loading 0
- Code Cache 7
- Code Sweeper 3
- Compiler 94
- Flag 1,135
- GC 421
  - Collector 3
  - Configuration 6
  - Detailed 21
  - Heap 6
  - Metaspace 4
  - Phases 377
  - Reference 4
- Profiling 3,012
- Runtime 3,179
- Initial System Property 18
- JVM Information 1
- Operating System 466
  - Memory 2
  - Network 18
  - Processor 138
  - Initial Environment Variable 1
  - OS Information 1
  - System Process 291
  - Virtualization Information 1

Start Time	Duration	End Time	GC Identifier	Event Type
04/04/20, 1:46:21 AM	6.813 ms	04/04/20, 1:46:21 AM	1	Garbage Collection
04/04/20, 1:46:21 AM	6.813 ms	04/04/20, 1:46:21 AM	1	G1 Garbage Collection
04/04/20, 1:46:21 AM	6.813 ms	04/04/20, 1:46:21 AM	1	Young Garbage Collection

Series of events during the execution (i.e. garbage collector).