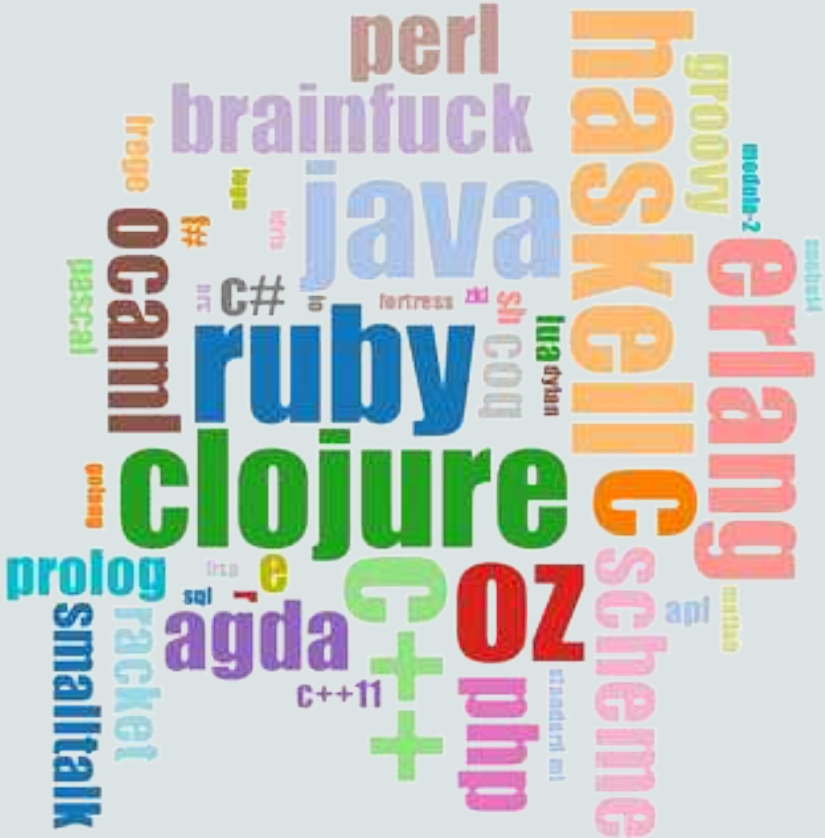

ORACLE

Understanding How Graal Works

a Java JIT Compiler Written in Java

@JaroslavTulach
Oracle Labs



ORACLE

Jaroslav Tulach

NetBeans Founder – 20 years ago

NetBeans Initial Architect – up to 2001

Practical API Design book – published 2008

Java/JavaScript/co. Interop – 2012 - now

Oracle Labs: Graal/Truffle – 2015 - now

Program Agenda

Short & mostly incorrect history

What is a JIT compiler?

Why write JIT compiler in Java?

Inevitable polyglot destiny

It all begins in MatFyz

- MatFyz 1992-1998 & Xelfi 1995-1997
- NetBeans & Roman Staněk
  - IDE vs. Platform
- Sun Microsystems 1999
  - Open Source – May 2000
- Oracle 2010
  - JDeveloper & OSGi

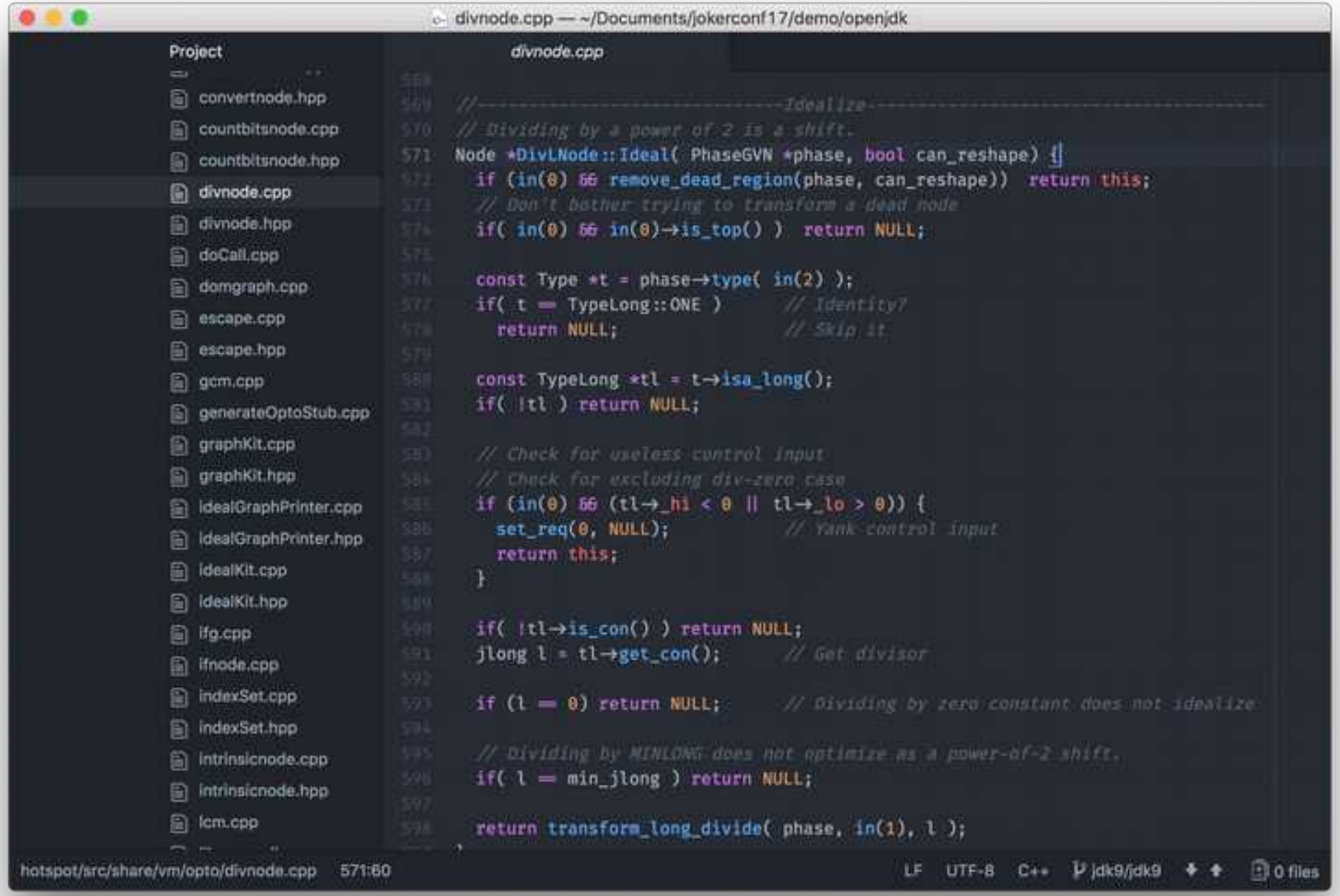
Safe Harbor Statement

The preceding and following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

http://chrisseaton.com/rubytruffle/jokerconf17/

```
$ git clone https://github.com/dmlloyd/openjdk.git
```

```
openjdk/hotspot/src/share/vm/opto
```



https://www.youtube.com/watch?v=Hqw57GJSrac

Things I won't do again...

- Write a VM in C/C++
  - Java plenty fast now
  - Mixing OOPS in a non-GC language a total pain
  - Forgetting 'this' is an OOP
    - Across a GC-allowable call
  - Roll-your-own malloc pointless now

https://www.youtube.com/watch?v=Hqw57GJSrac

Setting up Graal

```
$ export JAVA_HOME=/bin/jdk-11.0.5/
$ export PATH=$JAVA_HOME/bin:$PATH
$ java -version
java version "11.0.5" 2019-10-15 LTS
```

```
$ git clone https://github.com/graalvm/mx.git
$ (cd mx; git checkout 5.252.3)
$ export PATH=`pwd`/mx:$PATH
```

```
$ git clone https://github.com/oracle/graal.git --branch vm-20.0.0
```

```
$ cd graal/compiler
$ mx --java-home=$JAVA_HOME build
```

Setting up IDE

```
$ unzip netbeans-11.3-bin.zip
$ export PATH=`pwd`/netbeans/bin:$PATH
```

```
$ cd graal/compiler
$ mx --java-home=$JAVA_HOME netbeansinit
$ netbeans –open src/org.graalvm.*
```

Demo

Graal Sources in NetBeans

```
class Demo {
  public static void main(String[] args) {
    while (true) {
      workload(14, 2);
    }
  }

  private static int workload(int a, int b) {
    return a + b;
  }
}
```

```
$ javac Demo.java
$ java -XX:+PrintCompilation -XX:CompileOnly=Demo::workload -cp . Demo
...
     78    1       3       Demo::workload (4 bytes)
     78    2       1       Demo::workload (4 bytes)
     78    1       3       Demo::workload (4 bytes)   made not entrant
...
```

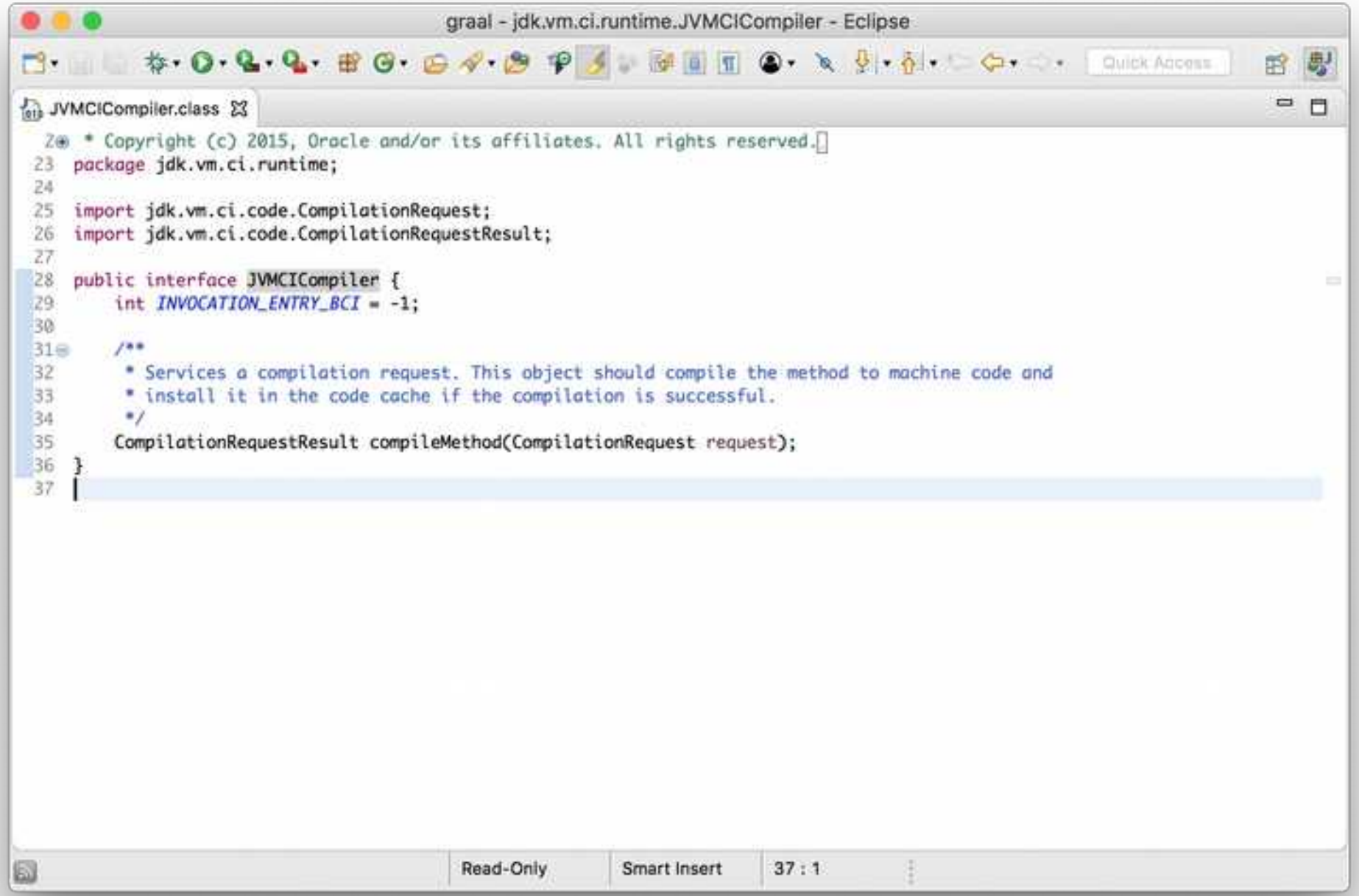
```
$ cd graal/compiler
$ mx build && mx unittest GraalAPITest
Updating/creating ./mxbuild/linux-amd64/graaljdks/jdk11-cmp
$ /mxbuild/linux-amd64/graaljdks/jdk11-cmp/bin/java \
  -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler -XX:-TieredCompilation  -XX:+PrintCompilation
  -XX:CompileOnly=Demo::workload \
  -agentlib:jdwp=transport=dt_socket,server=y,address=8000,suspend=y \
  -cp . Demo
...
  8642   17             Demo::workload (4 bytes)
...
```

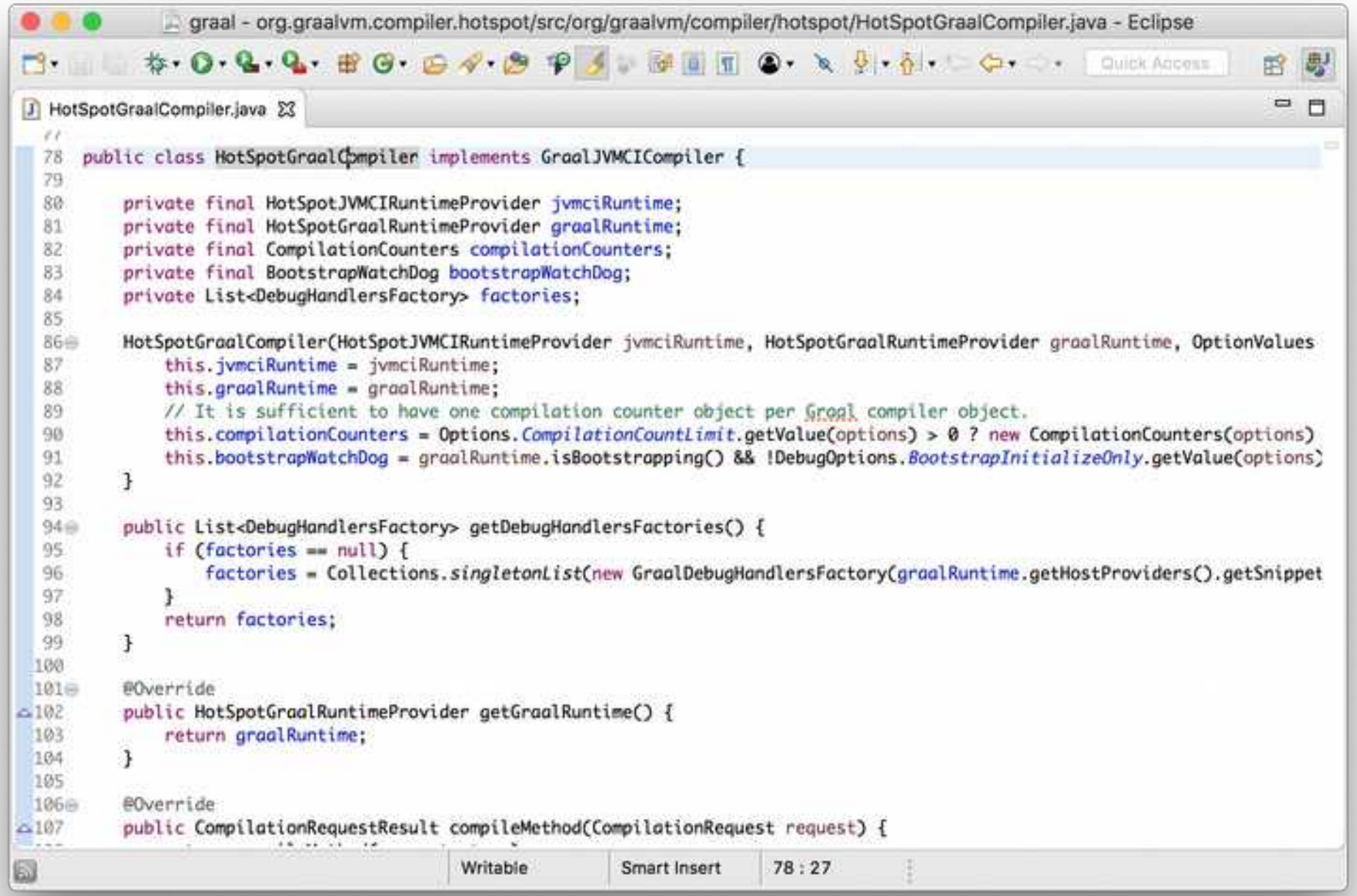
The JVM compiler interface

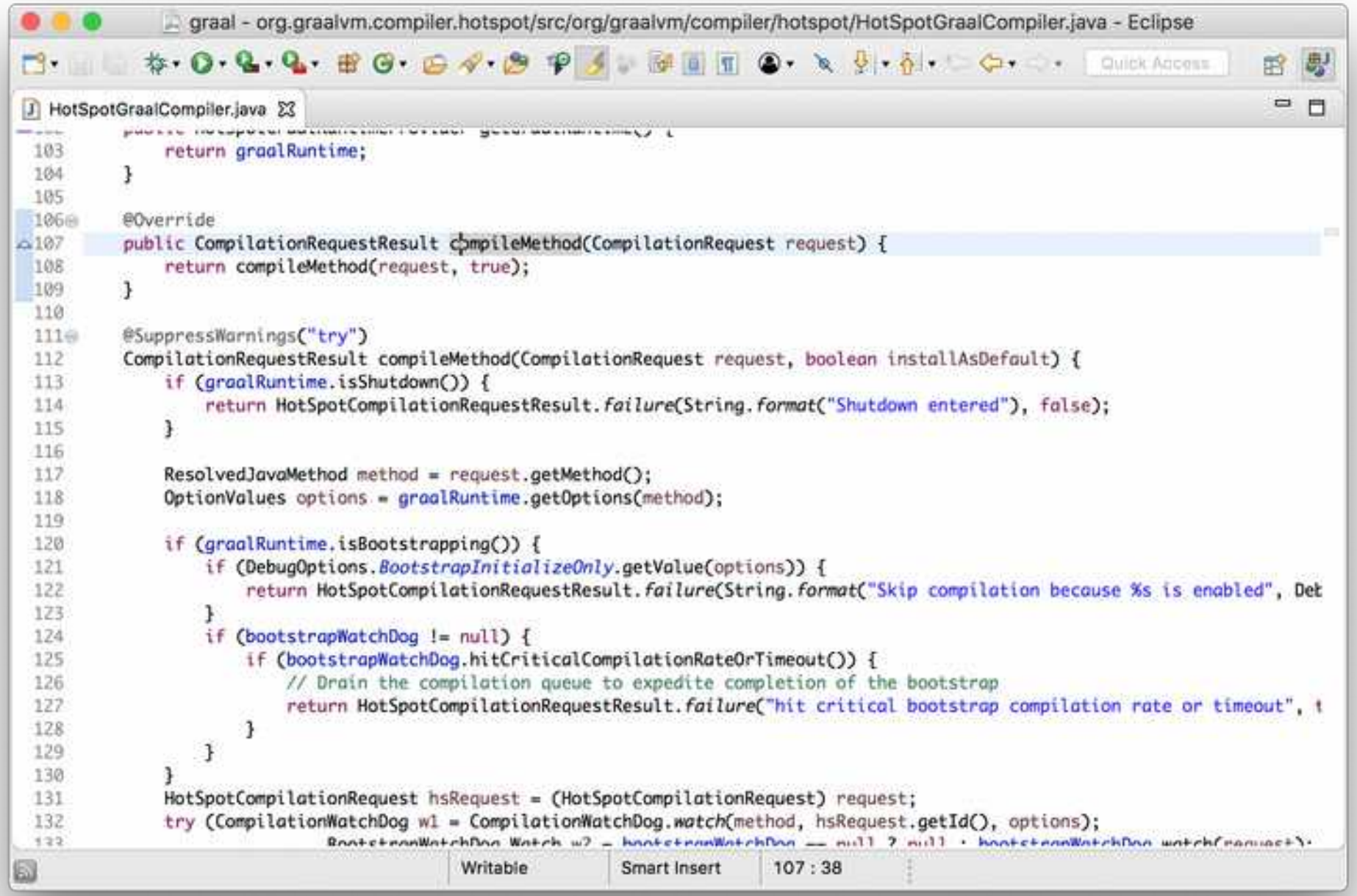
```
interface JVMCICompiler {
    byte[] compileMethod(byte[] bytecode);
}
```

```
interface JVMCICompiler {
  void compileMethod(CompilationRequest request);
}

interface CompilationRequest {
    JavaMethod getMethod();
}

interface JavaMethod {
    byte[] getCode();
    int getMaxLocals();
    int getMaxStackSize();
    ProfilingInfo getProfilingInfo();
    ...
}
```

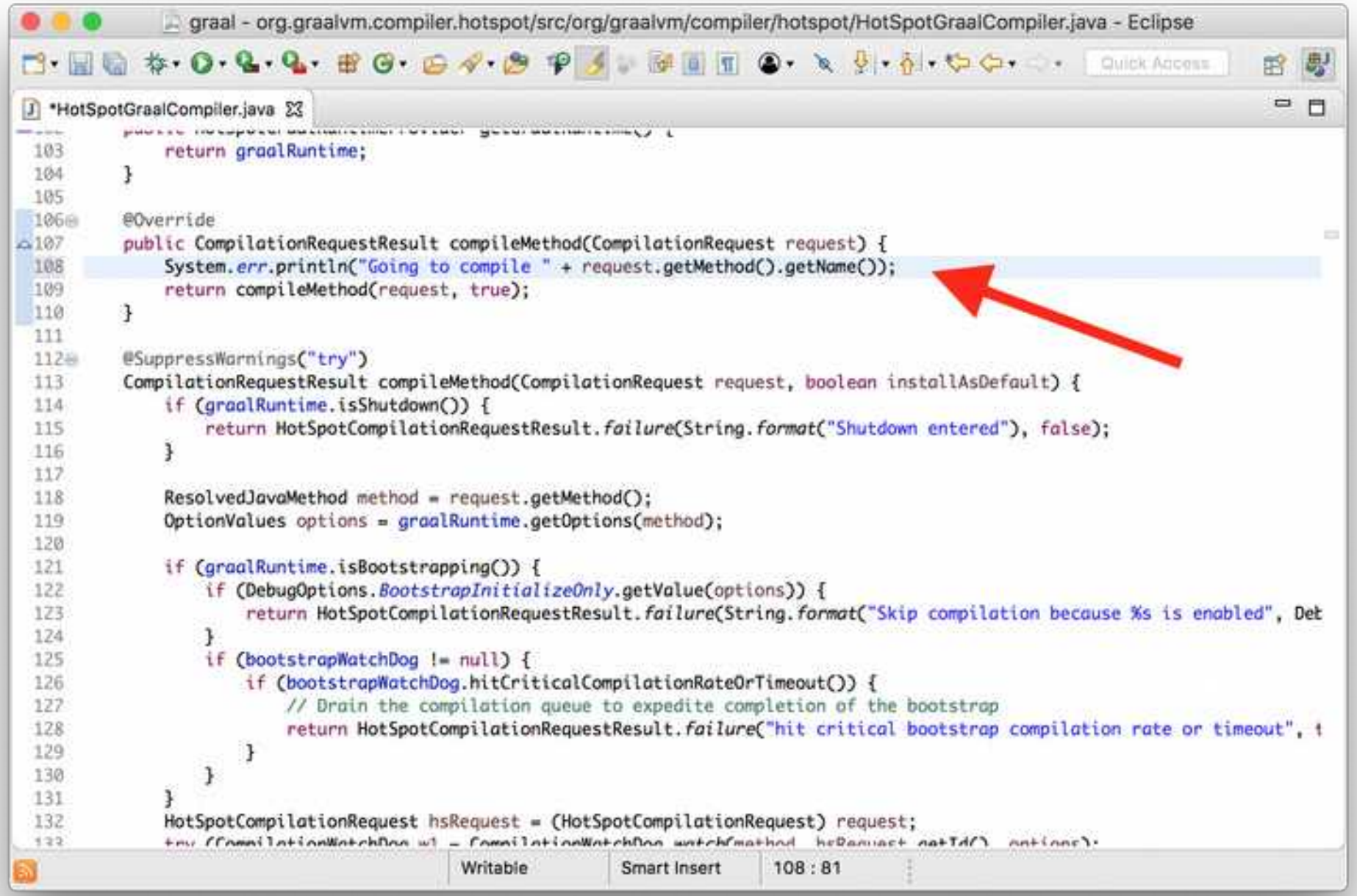
```
HotSpot.installCode(targetCode);
```







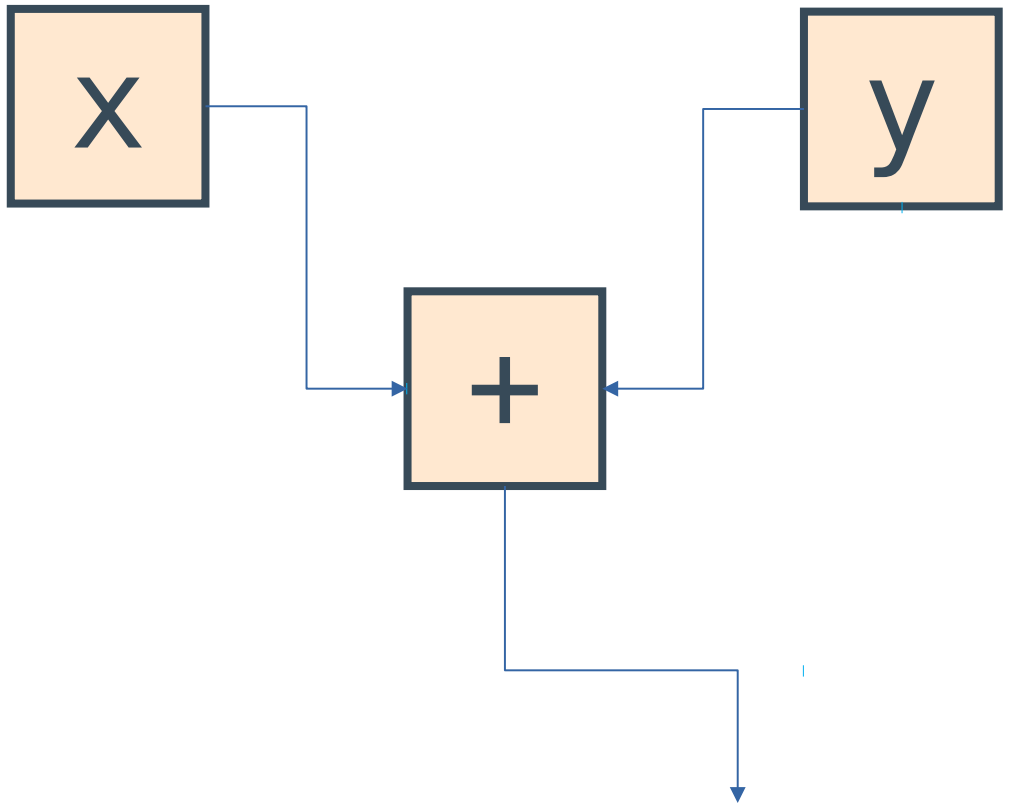
```
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationRequestResult compileMethod(CompilationRequest request) {
    System.err.println("Going to compile " + request.getMethod().getName());
    ...
  }
}
```



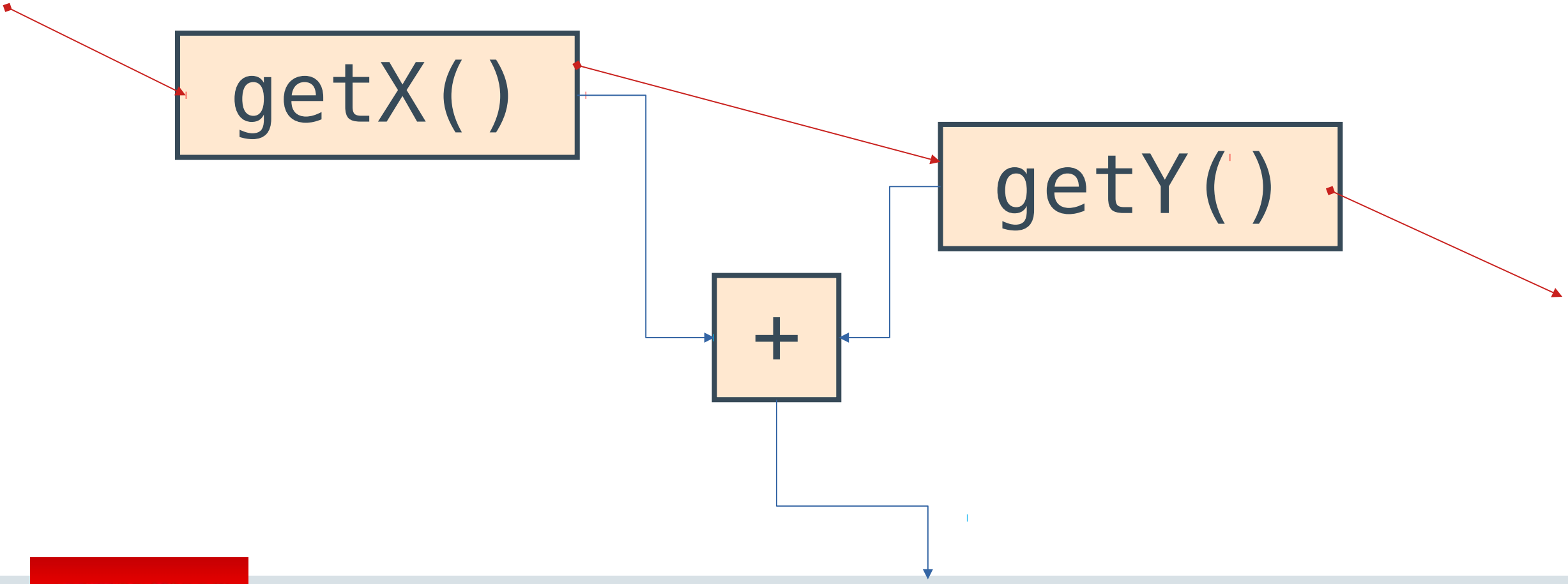
```
$ cd graal/compiler
$ mx build && mx unittest GraalAPITest
Updating/creating ./mxbuild/linux-amd64/graaljdks/jdk11-cmp
$ /mxbuild/linux-amd64/graaljdks/jdk11-cmp/bin/java \
  -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler -XX:-TieredCompilation  -XX:+PrintCompilation
  -XX:CompileOnly=Demo::workload \
  -agentlib:jdwp=transport=dt_socket,server=y,address=8000,suspend=y \
  -cp . Demo
…
Going to compile workload
  8642   17             Demo::workload (4 bytes)
...
```

The Graal graph

x + y



getX() + getY()

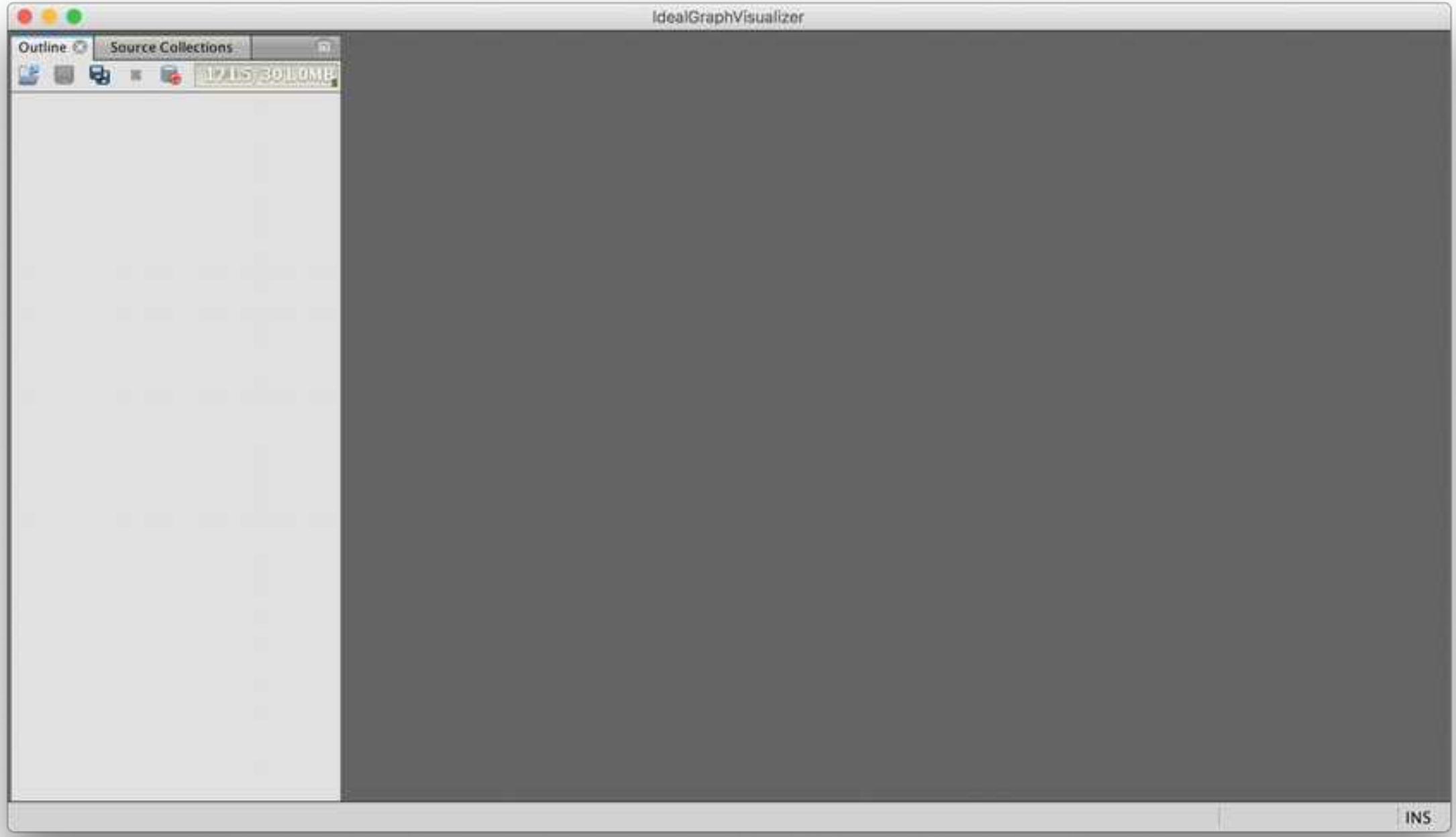


Setting IGV

https://www.oracle.com/downloads/graalvm-downloads.html

```
$ unzip /Downloads/idealgraphvisualizer-20.0.0-all.zip
$ export PATH=`pwd`/idealgraphvisualizer/bin:$PATH
```

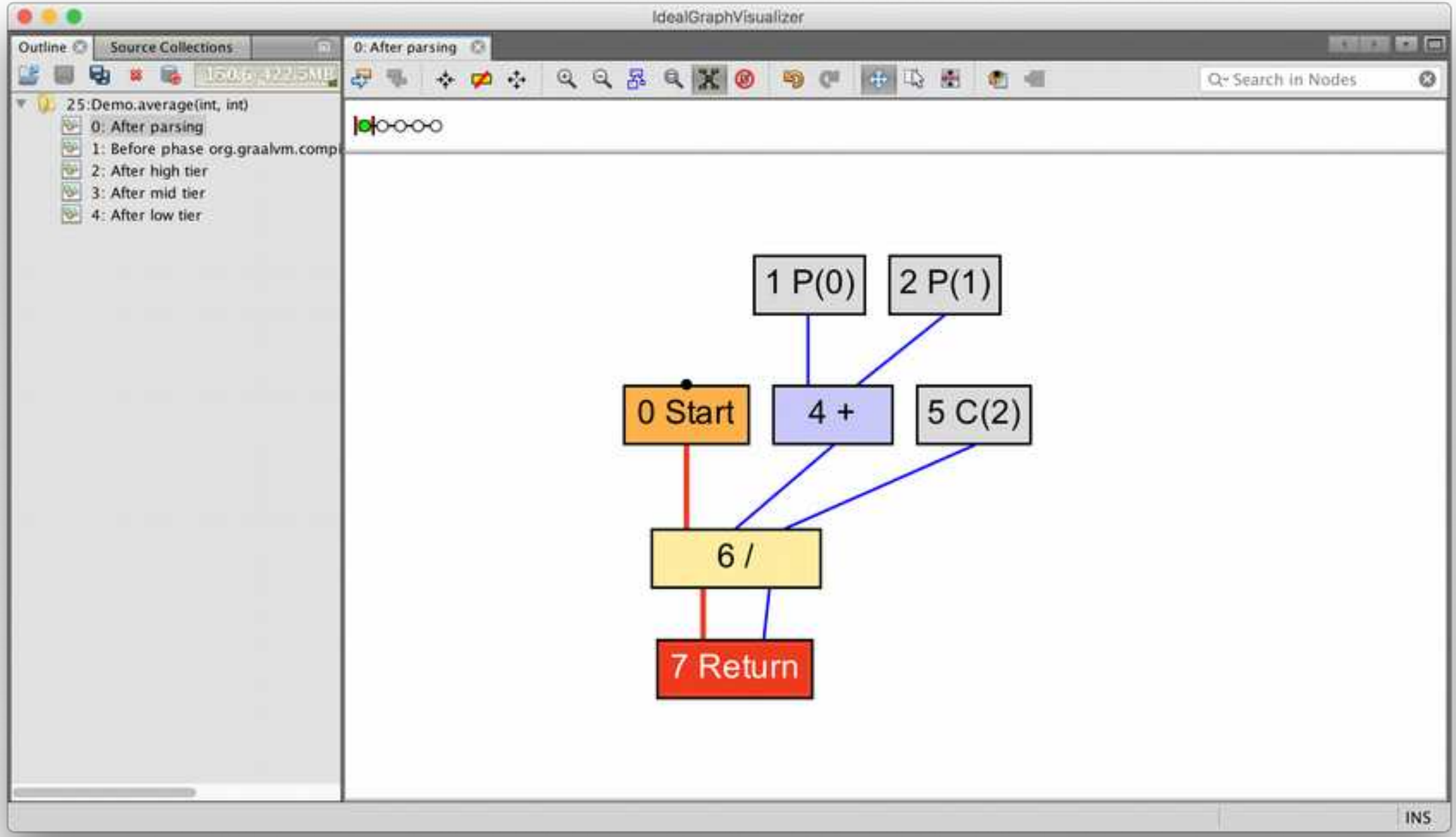
```
$ cd graal/compiler
$ idealgraphvisualizer -open . # right click & open project
```



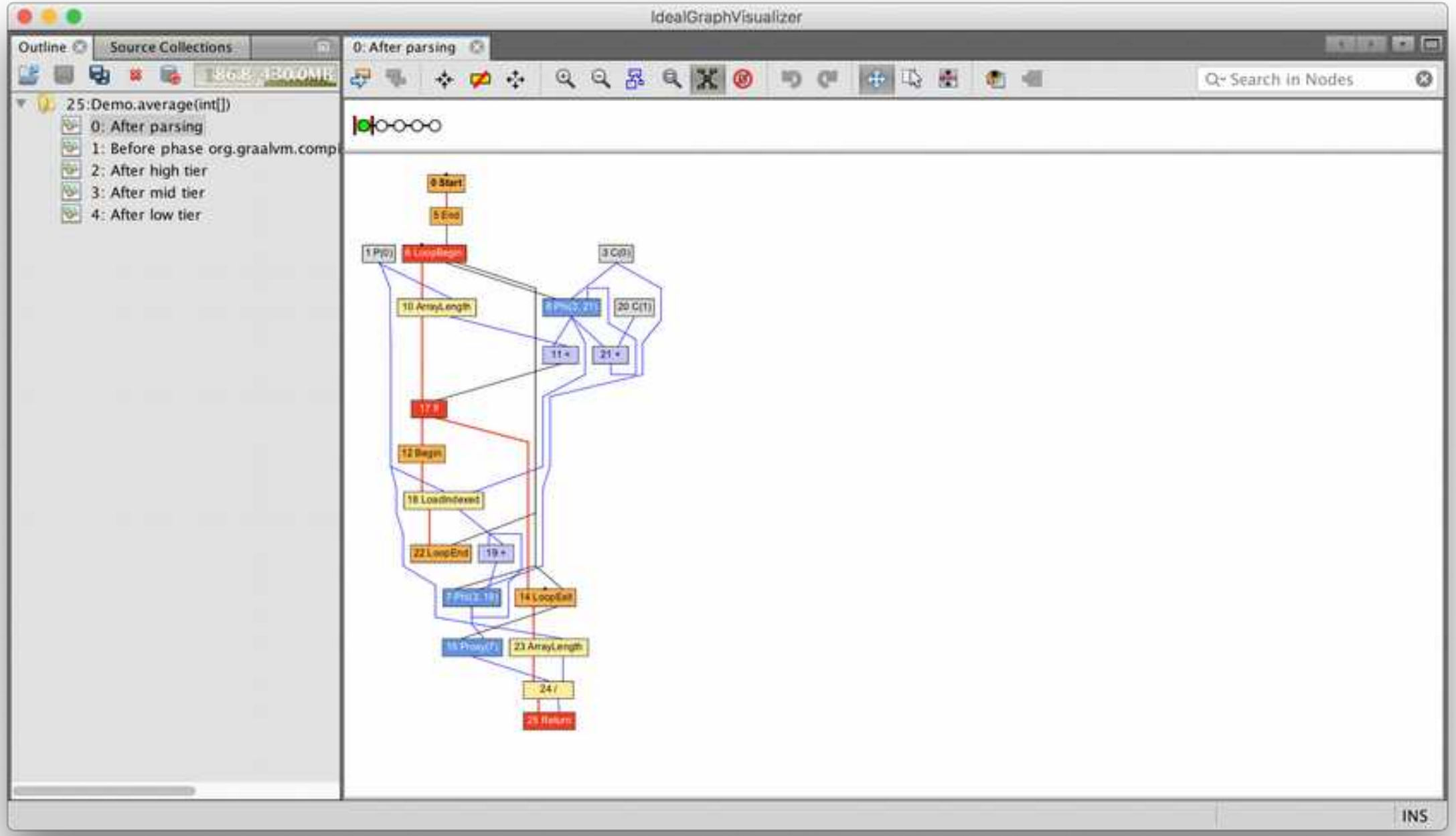
-Dgraal.Dump

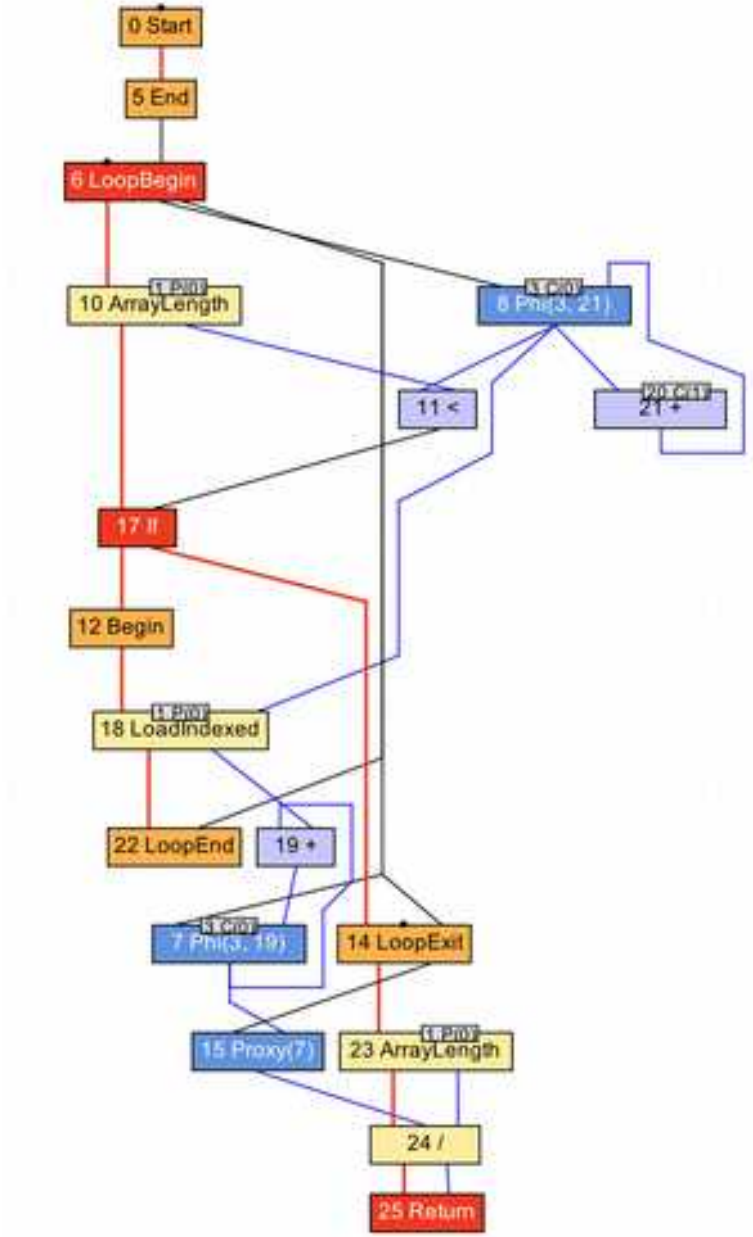
```
$ ./mxbuild/linux-amd64/graaljdks/jdk11-cmp/bin/java \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+UseJVMCICompiler -XX:-TieredCompilation \
  -XX:CompileOnly=Demo::workload \
  -Dgraal.Dump=:1 -Dgraal.PrintGraph=Network \
  -cp . Demo
```

```
int average(int a, int b) {
   return (a + b) / 2;
}
```



```
int average(int[] values) {
  int sum = 0;
  for (int n = 0; n < values.length; n++) {
    sum += values[n];
  }
  return sum / values.length;
}
```





From bytecode to machine code

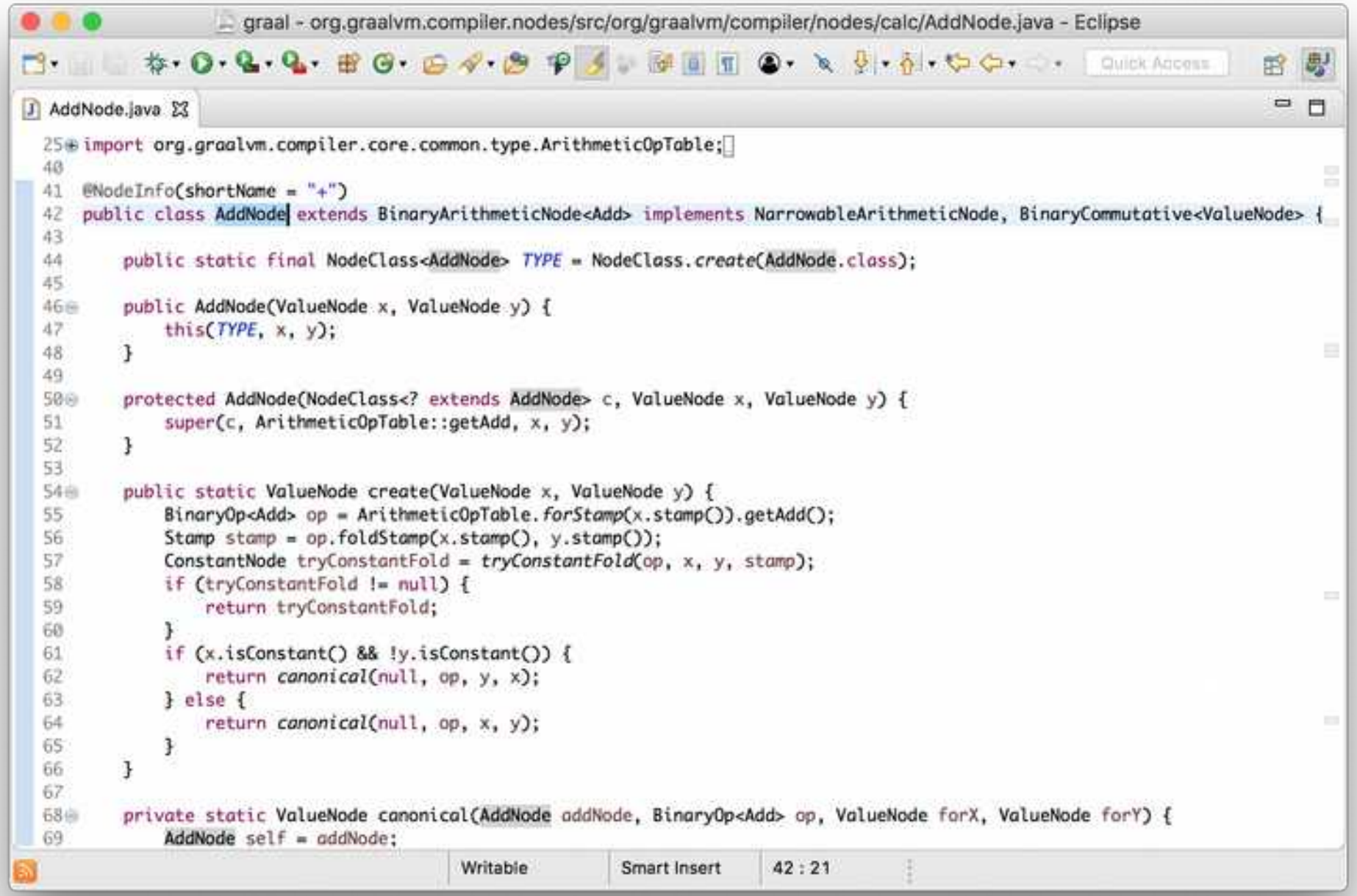
Bytecode in…

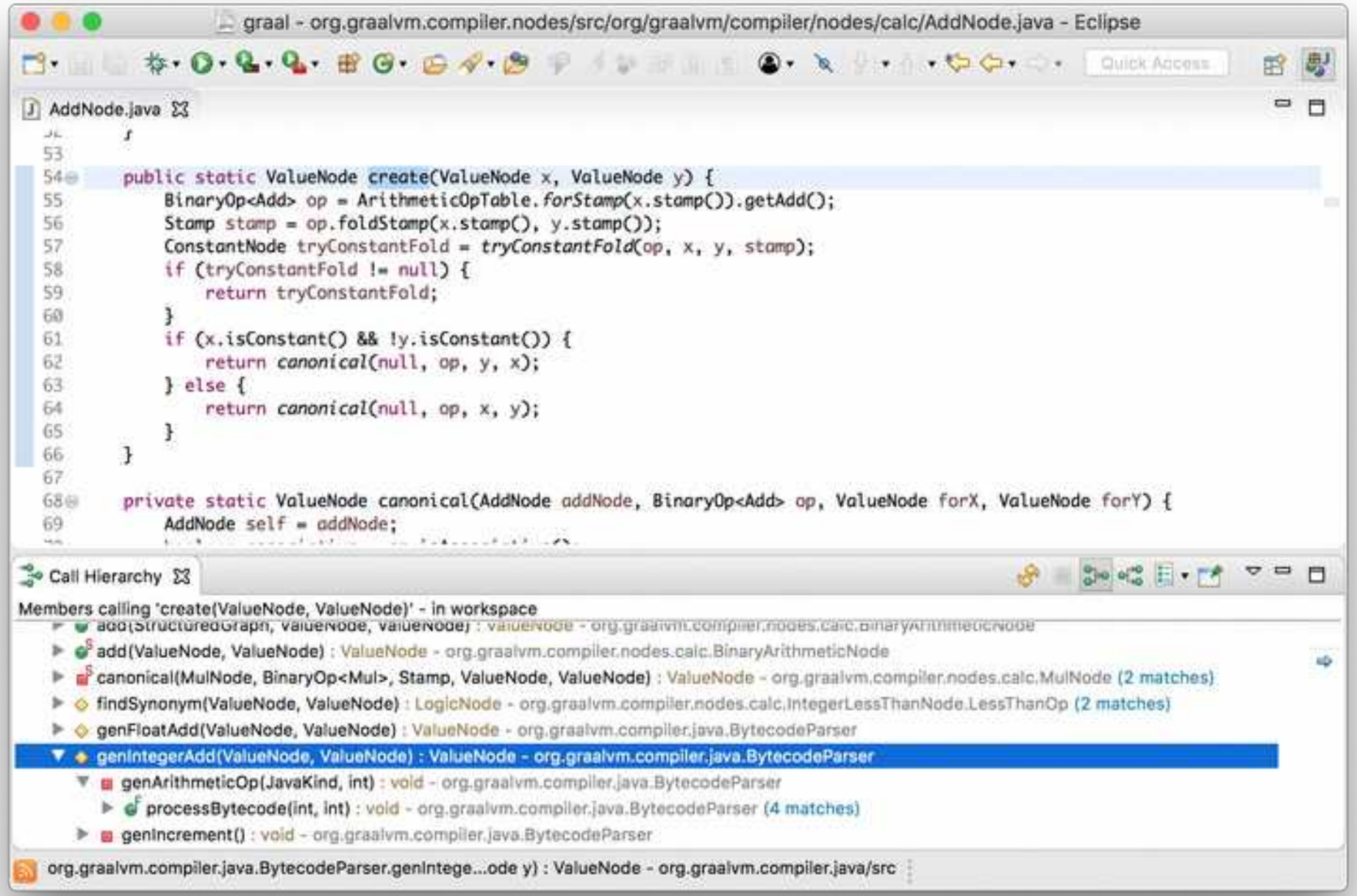
```
int workload(int a, int b) {
  return a + b;
}
```

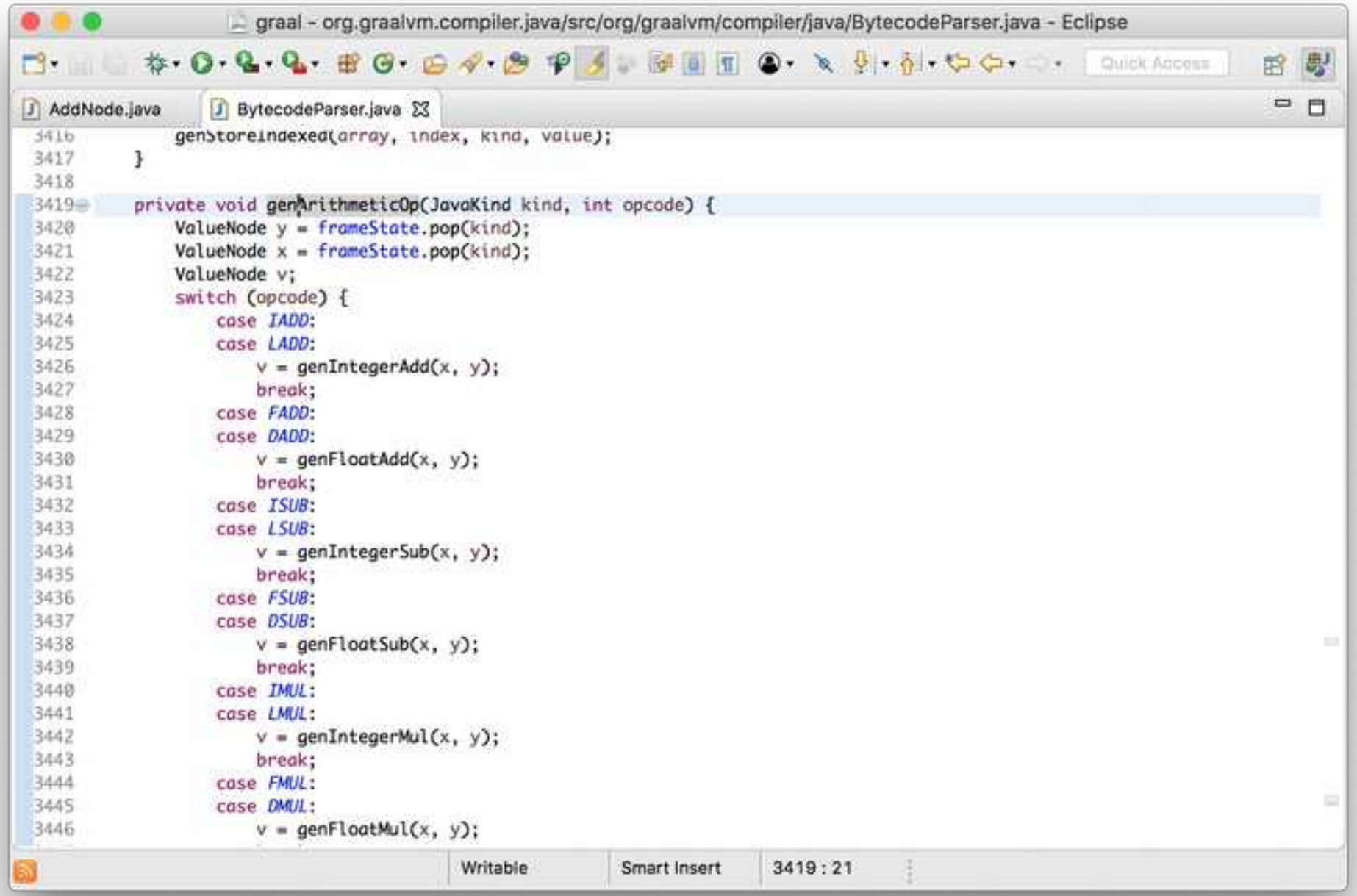
```
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationRequestResult compileMethod(CompilationRequest request) {
    System.err.println(request.getMethod().getName() + " bytecode: "
      + Arrays.toString(request.getMethod().getCode()));
    ...
  }
}
```

```
workload bytecode: [26, 27, 96, -84]
```

The bytecode parser…







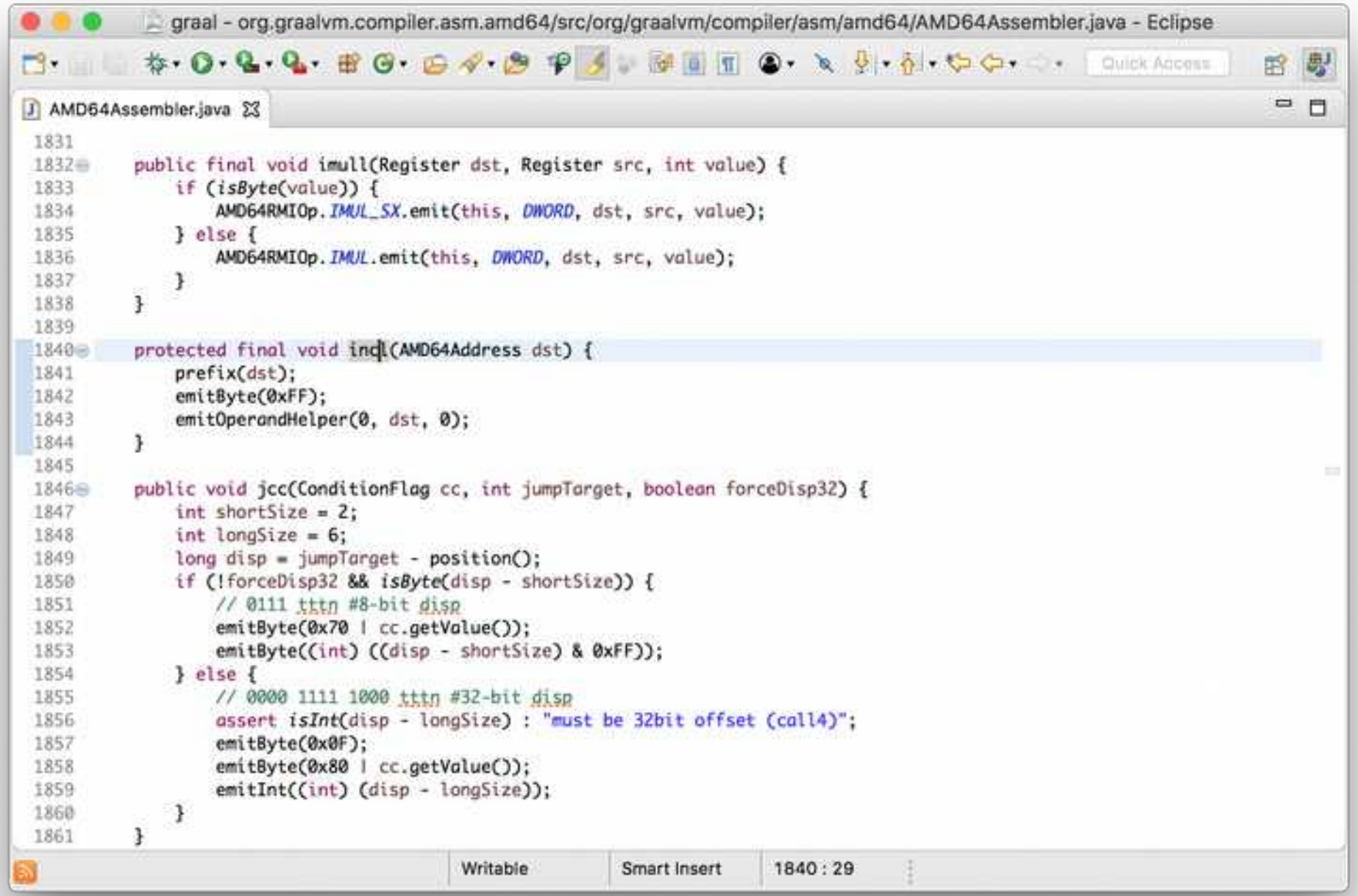
```
private void genArithmeticOp(JavaKind kind, int opcode) {
    ValueNode y = frameState.pop(kind);
    ValueNode x = frameState.pop(kind);
    ValueNode v;
    switch (opcode) {
        ...
        case LADD:
            v = genIntegerAdd(x, y);
            break;
        ...
    }
    frameState.push(kind, append(v));
}
```

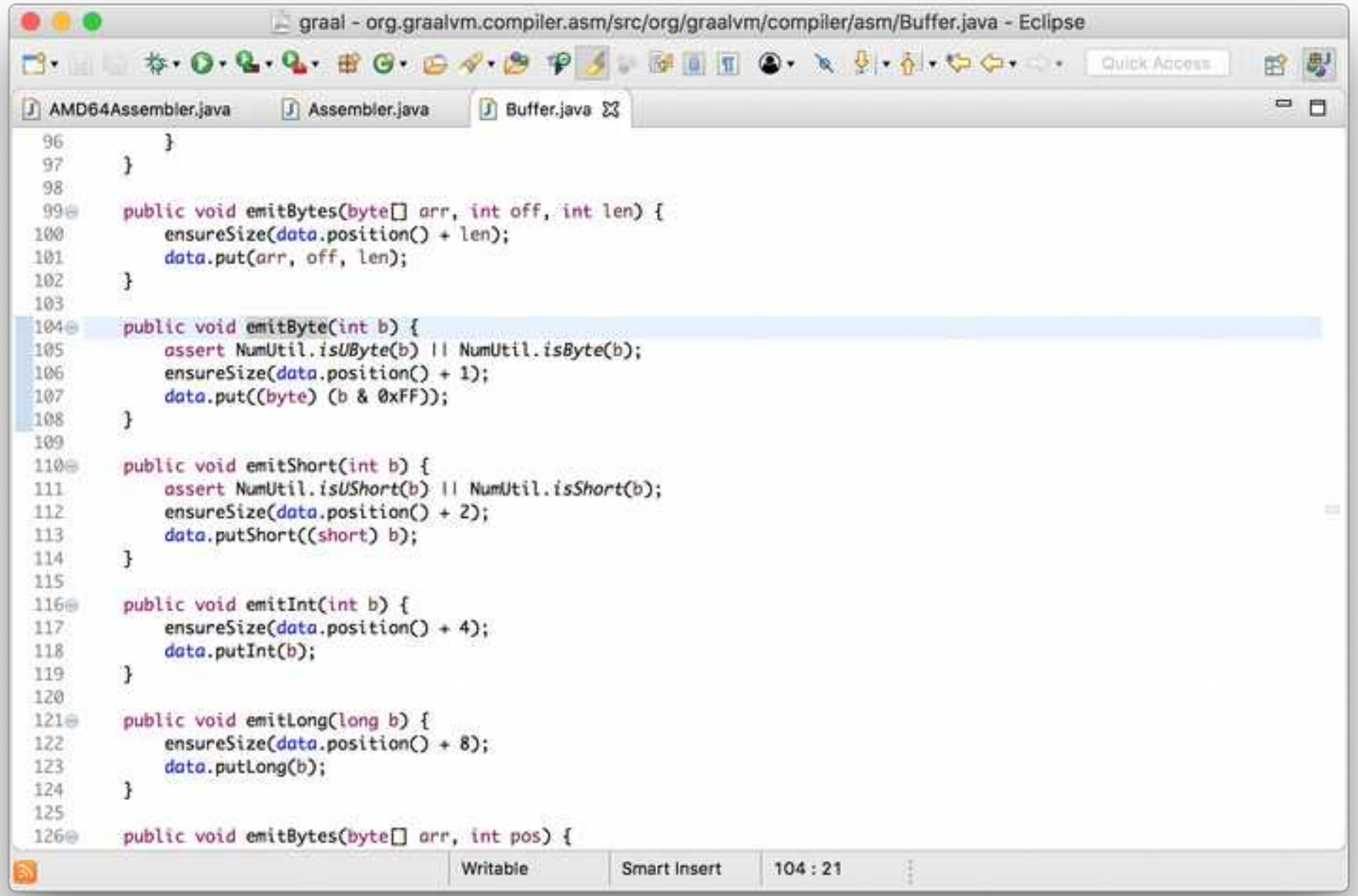
Emitting assembly…

```
void generate(Generator gen) {
    gen.emitAdd(a, b);
}
```

```
int workload(int a) {
  return a + 1;
}
```

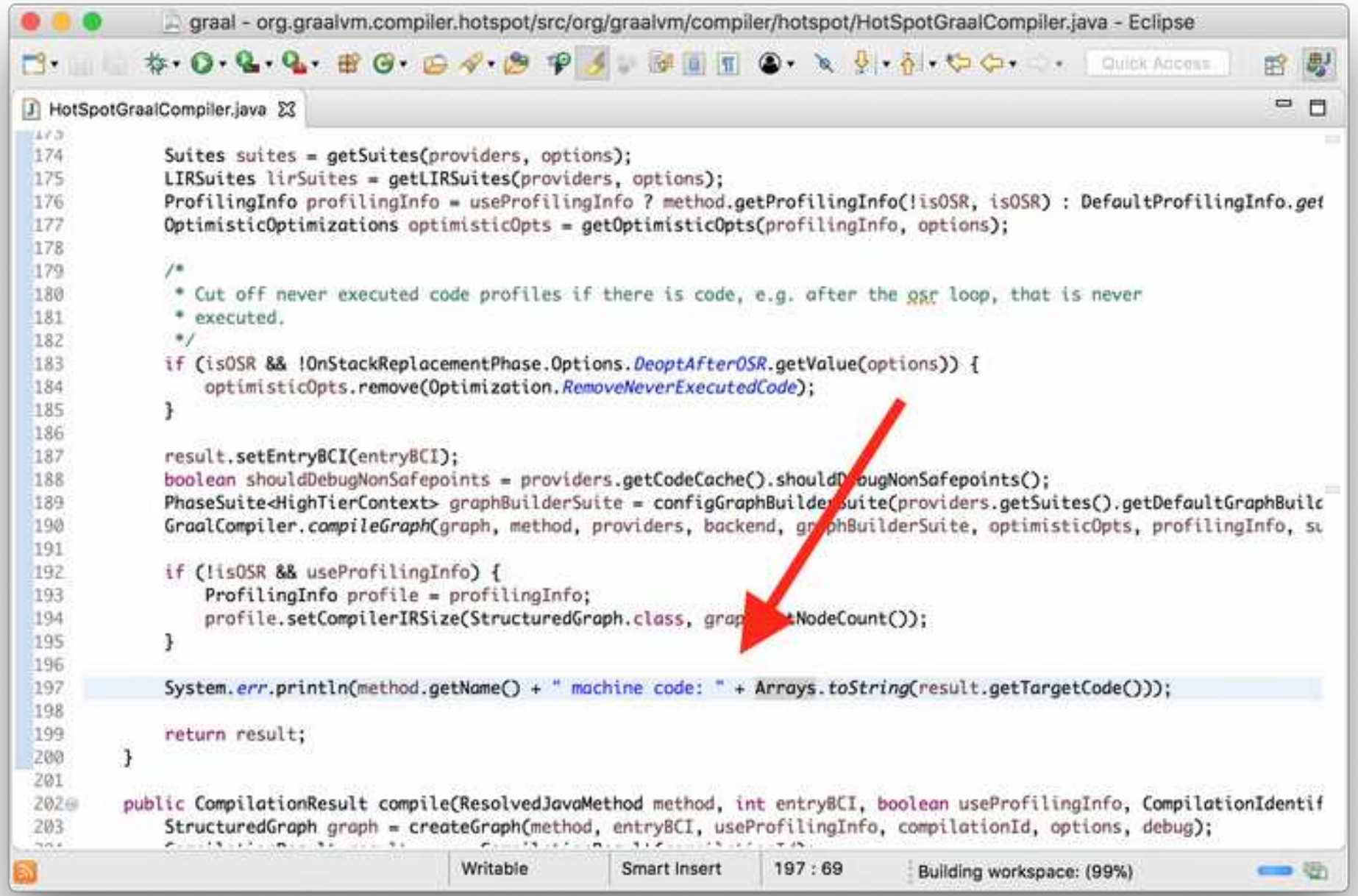
```
void incl(Register dst) {
    int encode = prefixAndEncode(dst.encoding);
    emitByte(0xFF);
    emitByte(0xC0 | encode);
}

void emitByte(int b) {
    data.put((byte) (b & 0xFF));
}
```





Machine code out…

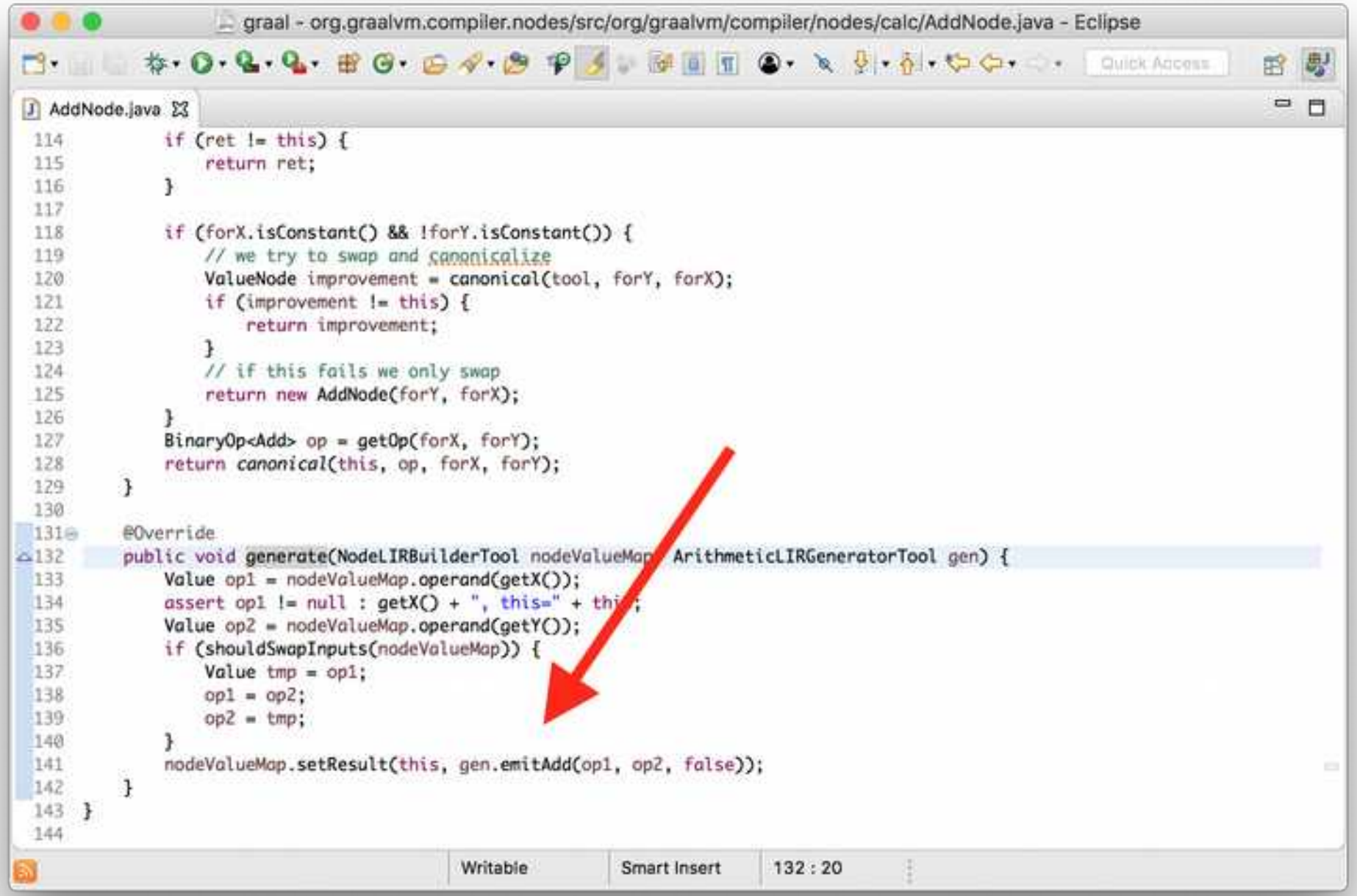
```
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationResult compileHelper(...) {
    ...
    System.err.println(method.getName() + " machine code: "
      + Arrays.toString(result.getTargetCode()));
    ...
  }
}
```




```
174         Suites suites = getSuites(providers, options);
175         LIRSuites lirSuites = getLIRSuites(providers, options);
176         ProfilingInfo profilingInfo = useProfilingInfo ? method.getProfilingInfo(!isOSR, isOSR) : DefaultProfilingInfo.get
177         OptimisticOptimizations optimisticOpts = getOptimisticOpts(profilingInfo, options);
178
179         /*
180          * Cut off never executed code profiles if there is code, e.g. after the osr loop, that is never
181          * executed.
182          */
183         if (isOSR && !OnStackReplacementPhase.Options.DeoptAfterOSR.getValue(options)) {
184             optimisticOpts.remove(Optimization.RemoveNeverExecutedCode);
185         }
186
187         result.setEntryBCI(entryBCI);
188         boolean shouldDebugNonSafepoints = providers.getCodeCache().shouldD ugNonSafepoints();
189         PhaseSuite<HighTierContext> graphBuilderSuite = configGraphBuilde uite(providers.getSuites().getDefaultGraphBuild
190         GraalCompiler.compileGraph(graph, method, providers, backend, g phBuilderSuite, optimisticOpts, profilingInfo, su
191
192         if (!isOSR && useProfilingInfo) {
193             ProfilingInfo profile = profilingInfo;
194             profile.setCompilerIRSize(StructuredGraph.class, grap tNodeCount());
195         }
196
197         System.err.println(method.getName() + " machine code: " + Arrays.toString(result.getTargetCode()));
198
199         return result;
200     }
201
202     public CompilationResult compile(ResolvedJavaMethod method, int entryBCI, boolean useProfilingInfo, CompilationIdentif
203         StructuredGraph graph = createGraph(method, entryBCI, useProfilingInfo, compilationId, options, debug);
```

```
$ ./mxbuild/linux-amd64/graaljdks/jdk11-cmp/bin/java \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+UnlockDiagnosticVMOptions \
  -XX:+UseJVMCICompiler -XX:-TieredCompilation \
  -XX:CompileOnly=Demo::workload -XX:+PrintAssembly \
  -cp . Demo
```

```
workload machine code: [15, 31, 68, 0, 0, 3, -14, -117, -58, -123, 5, ...]
...
0x00000010f71cda0: nopl   0x0(%rax,%rax,1)
0x00000010f71cda5: add    %edx,%esi           ;*iadd {reexecute=0 rethrow=0 return_oop=0}
                                              ; - Demo::workload@2 (line 10)

0x00000010f71cda7: mov    %esi,%eax           ;*ireturn {reexecute=0 rethrow=0 return_oop=0}
                                              ; - Demo::workload@3 (line 10)

0x00000010f71cda9: test   %eax,-0xcba8da9(%rip)        # 0x0000000102b74006
                                              ;   {poll_return}
0x00000010f71cdaf: vzeroupper
0x00000010f71cdb2: retq
```



```
workload mechine code: [15, 31, 68, 0, 0, 43, -14, -117, -58, -123, 5, ...]
0x0000000107f451a0: nopl   0x0(%rax,%rax,1)
0x0000000107f451a5: sub    %edx,%esi           ;*iadd {reexecute=0 rethrow=0 return_oop=0}
                                               ; - Demo::workload@2 (line 10)

0x0000000107f451a7: mov    %esi,%eax           ;*ireturn {reexecute=0 rethrow=0 return_oop=0}
                                               ; - Demo::workload@3 (line 10)

0x0000000107f451a9: test   %eax,-0x1db81a9(%rip)        # 0x000000010618d006
                                               ;   {poll_return}
0x0000000107f451af: vzeroupper
0x0000000107f451b2: retq
```

```
[26, 27, 96, -84] → [15, 31, 68, 0, 0, 43, -14, -117, -58, -123, 5, ...]
```

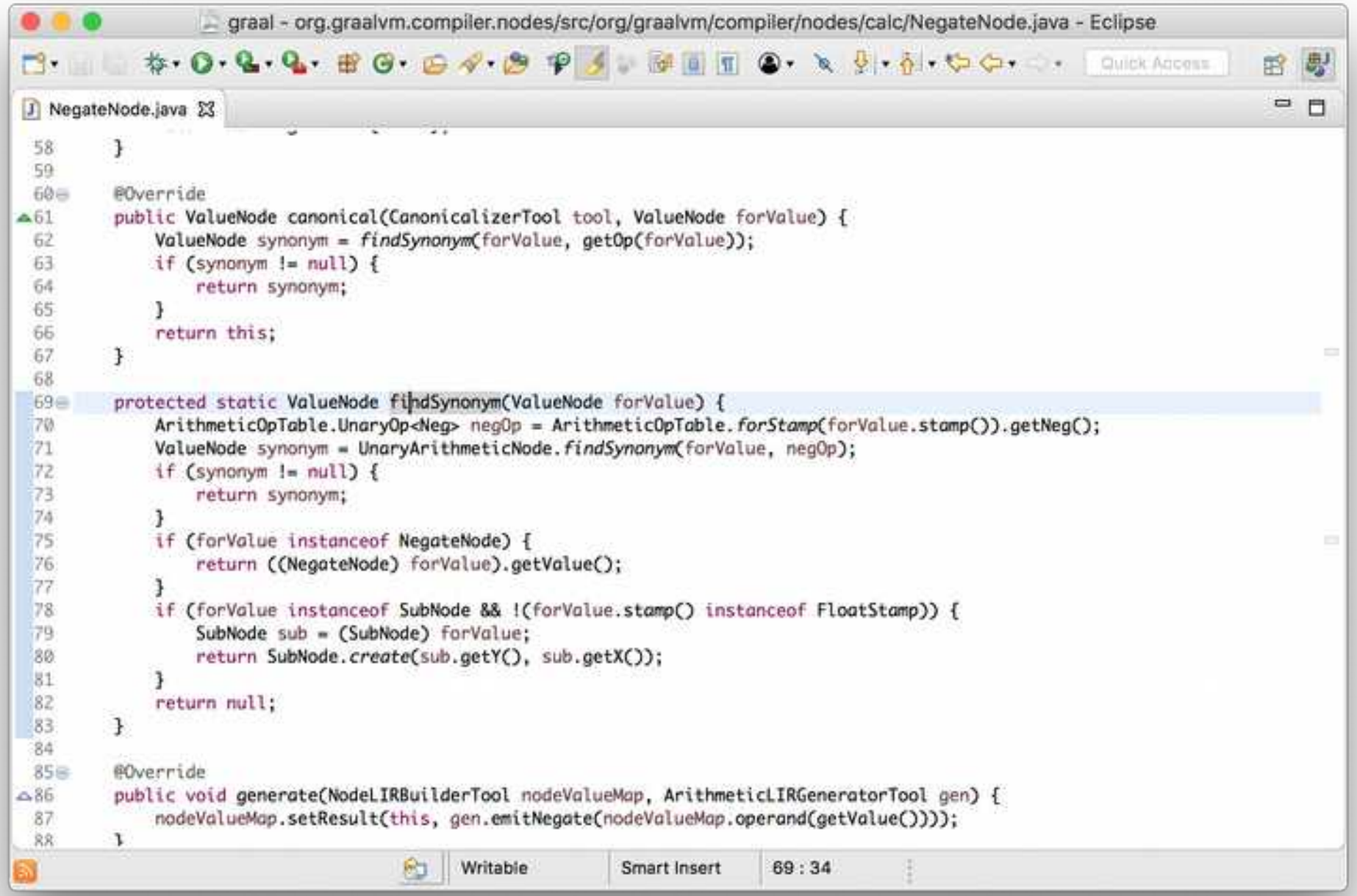
Optimisations

Canonicalisation

```
interface Phase {
  void run(Graph graph);
}
```

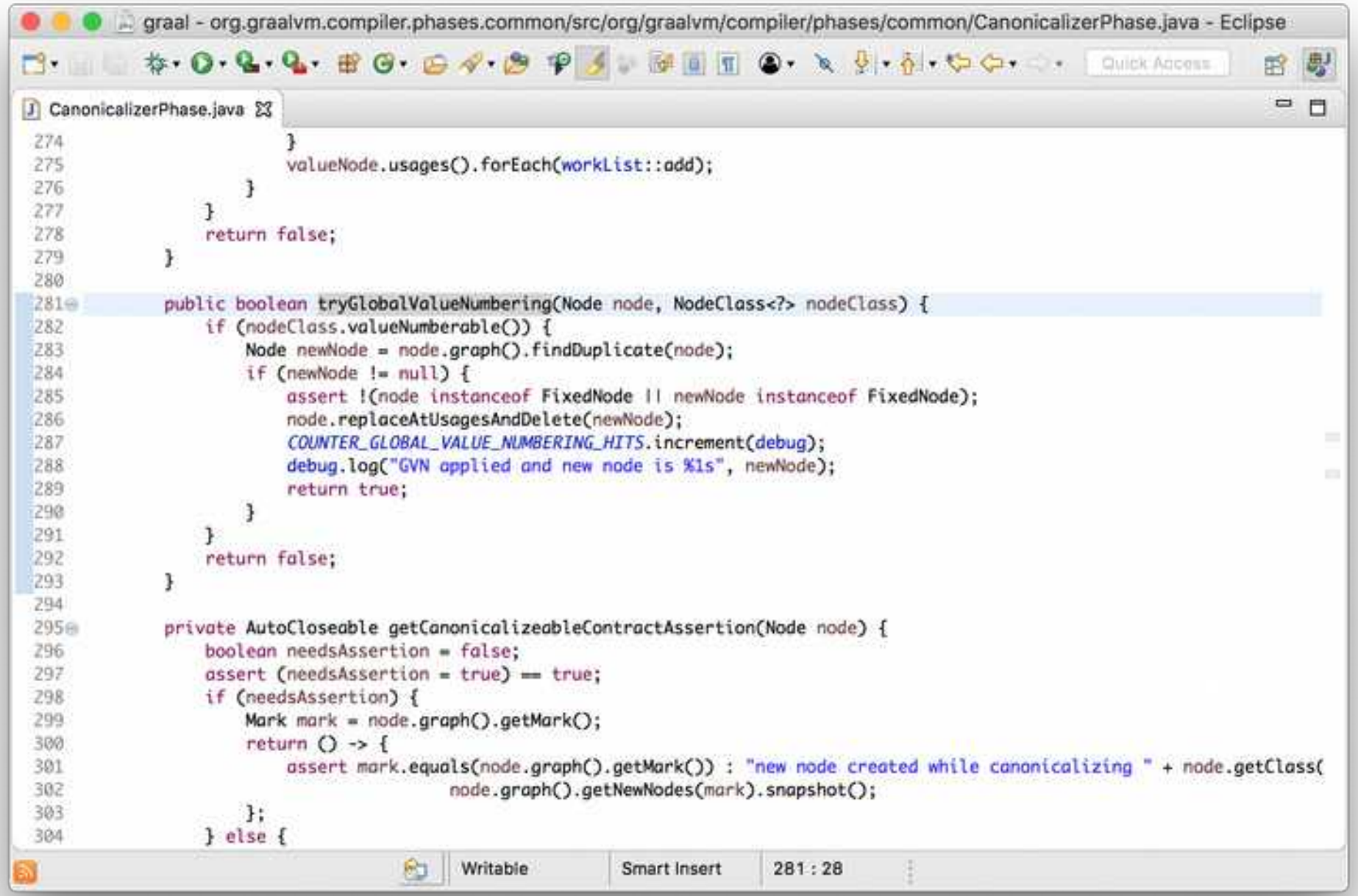
```
interface Node {
  Node canonical();
}
```

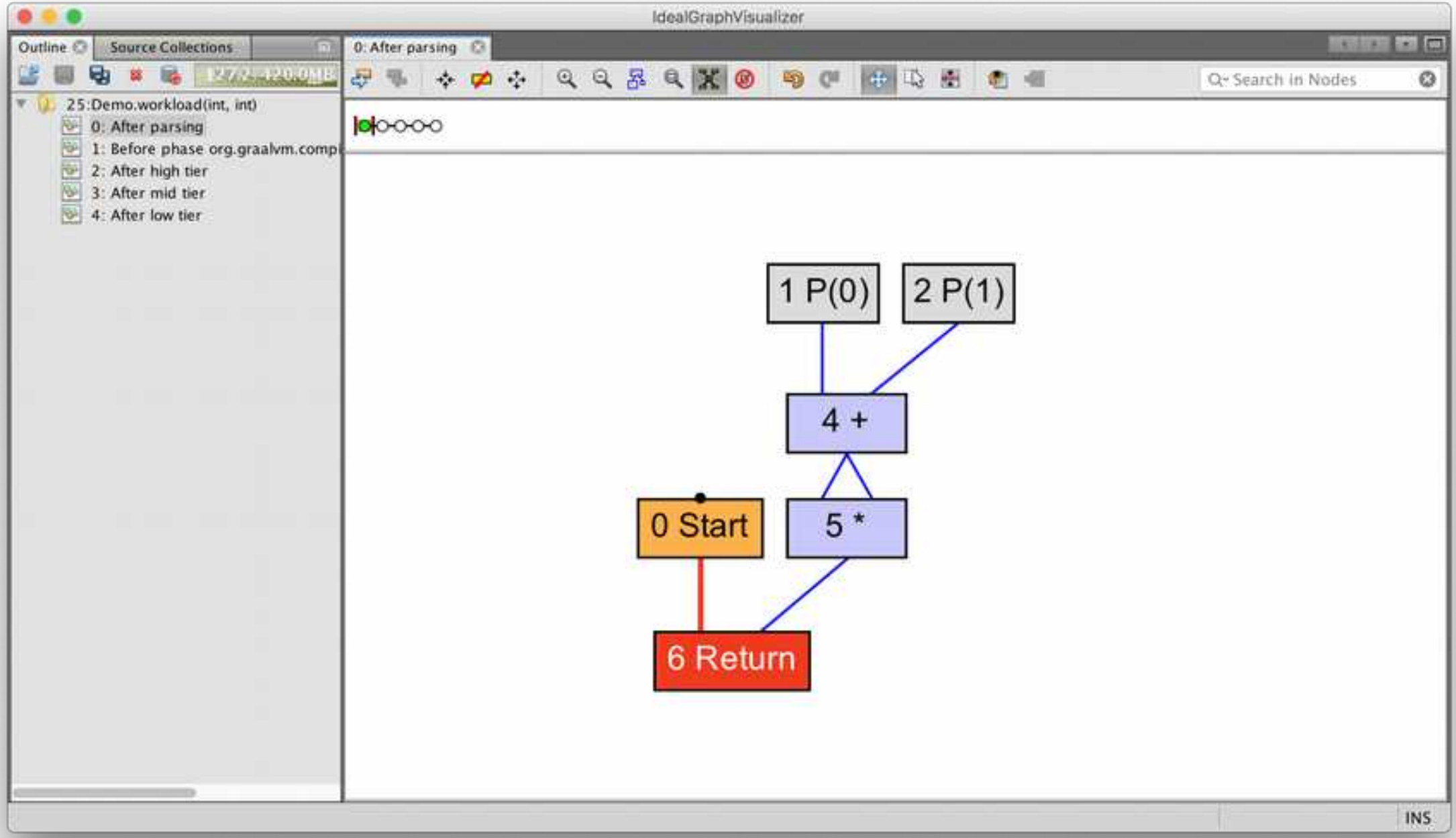
```
class NegateNode implements Node {
  Node canonical() {
    if (value instanceof NegateNode) {
      return ((NegateNode) value).getValue();
    } else {
      return this;
    }
  }
}
```



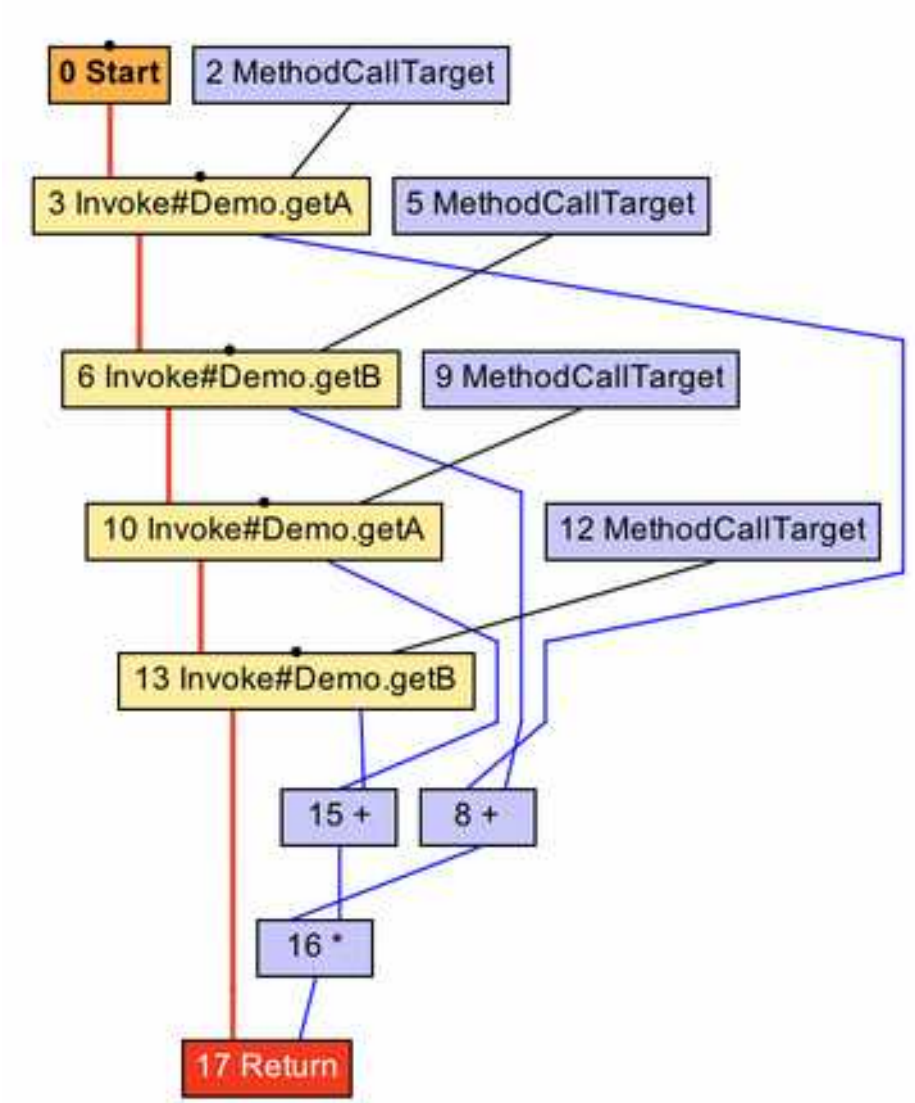
Global value numbering

```
int workload(int a, int b) {
  return (a + b) * (a + b);
}
```





```
int workload() {
  return (getA() + getB()) * (getA() + getB());
}
```



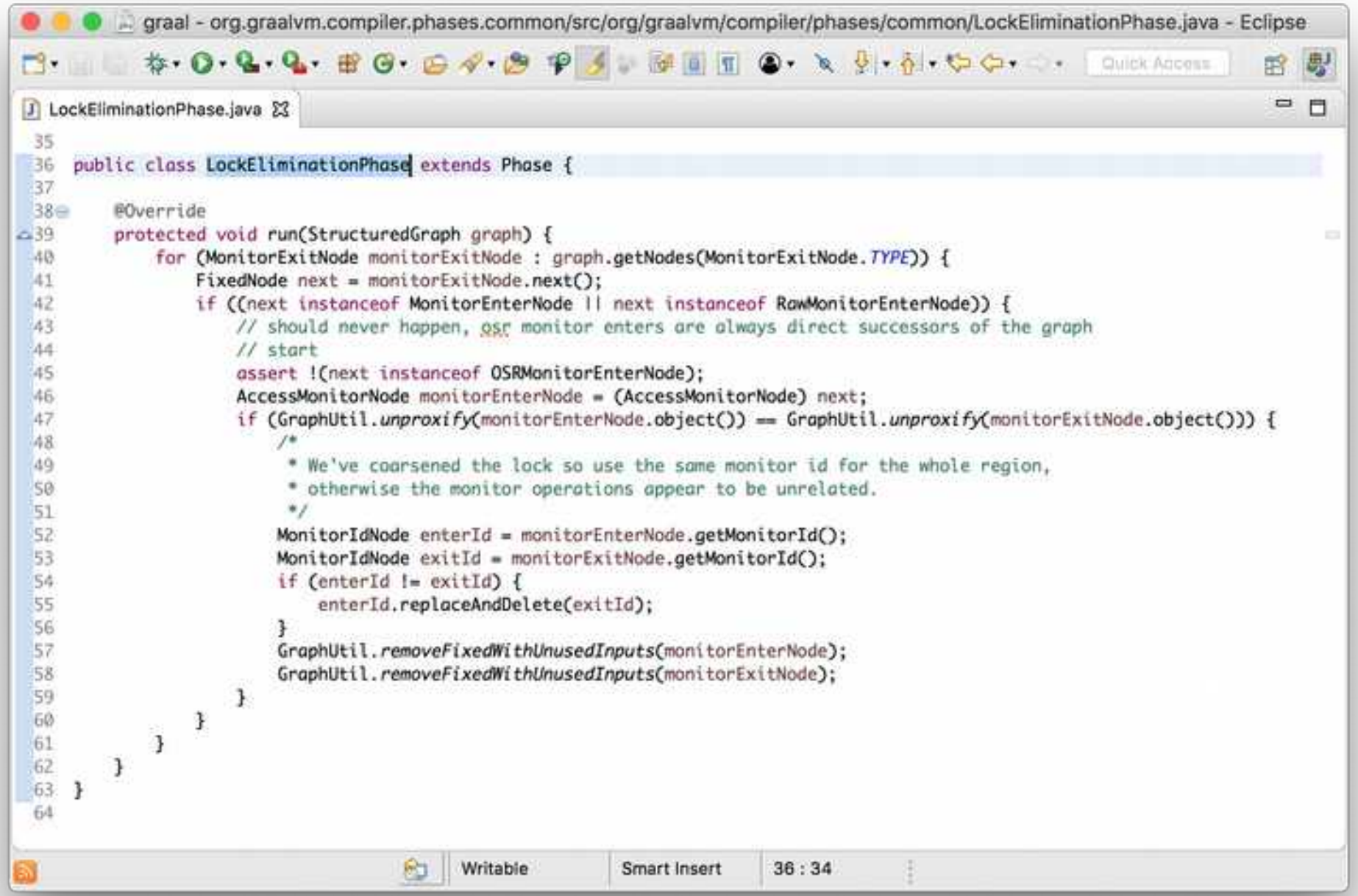
Lock coarsening

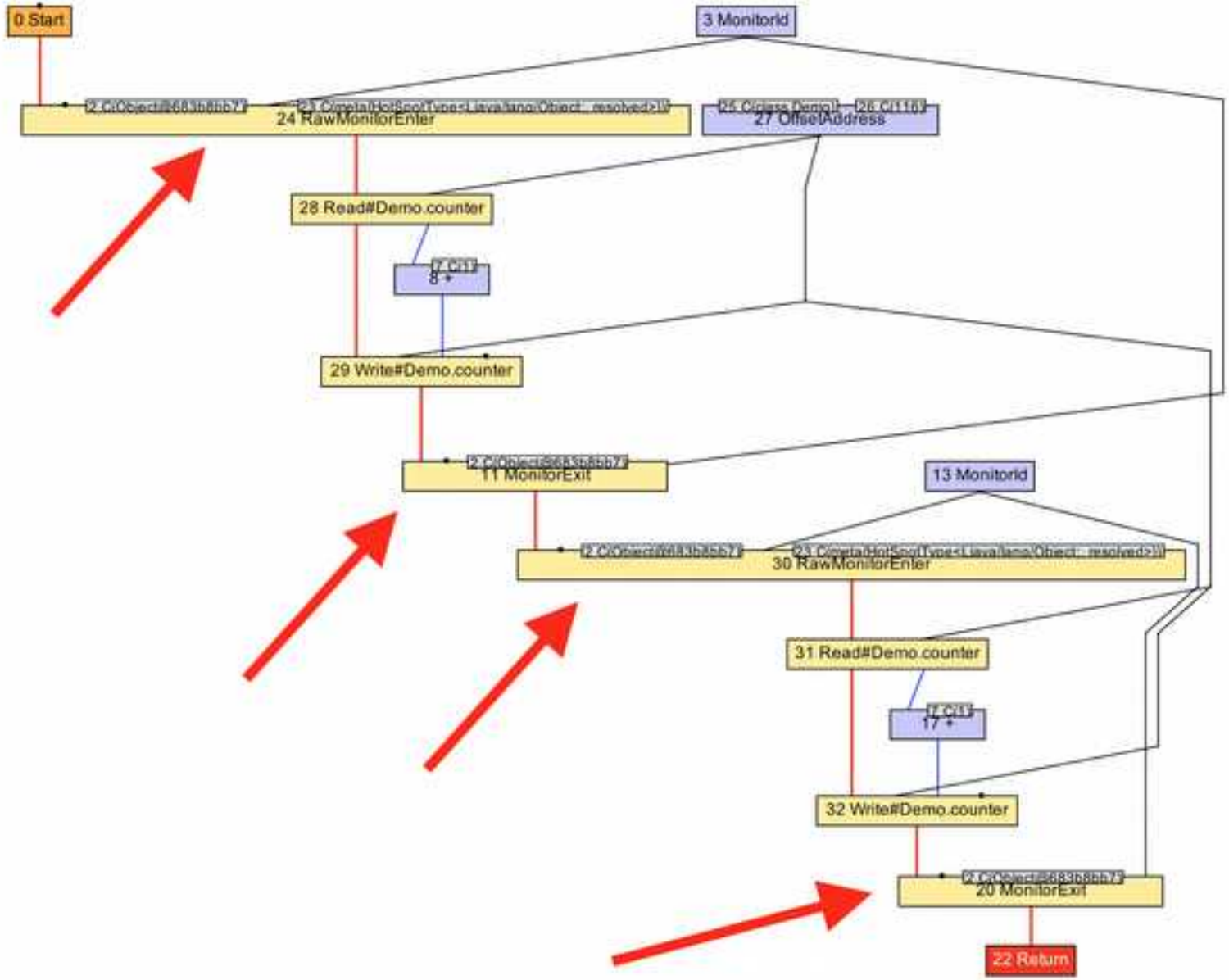
```
void workload() {
  synchronized (monitor) {
    counter++;
  }
  synchronized (monitor) {
    counter++;
  }
}
```

```
void workload() {
  monitor.enter();
  counter++;
  monitor.exit();
  monitor.enter();
  counter++;
  monitor.exit();
}
```

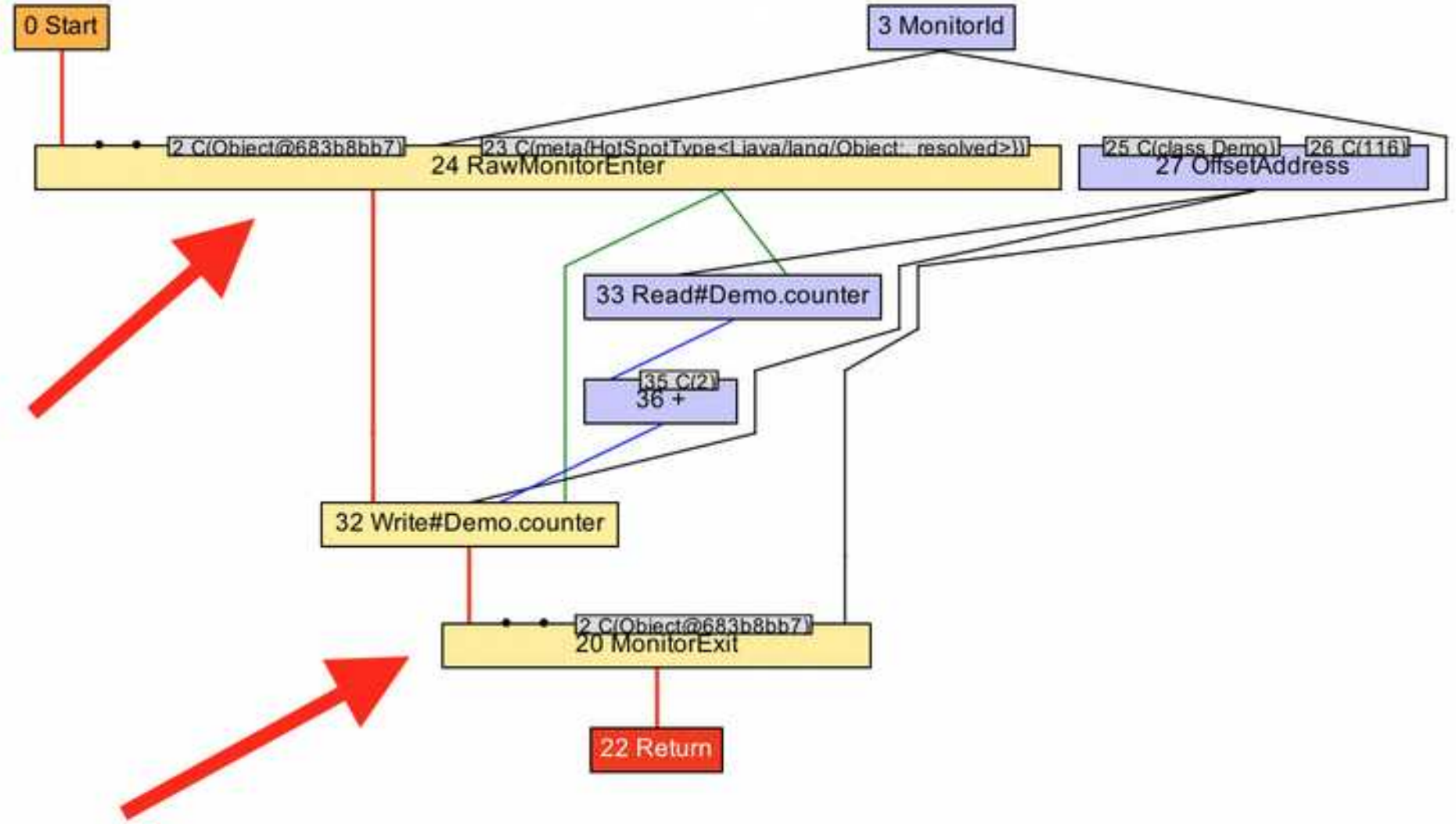
```
void workload() {
  monitor.enter();
  counter++;
  counter++;
  monitor.exit();
}
```

```
void run(StructuredGraph graph) {
  for (monitorExitNode monitorExitNode : graph.getNodes(monitorExitNode.class)) {
    FixedNode next = monitorExitNode.next();
    if (next instanceof monitorEnterNode) {
      AccessmonitorNode monitorEnterNode = (AccessmonitorNode) next;
      if (monitorEnterNode.object() == monitorExitNode.object()) {
        monitorExitNode.remove();
        monitorEnterNode.remove();
      }
    }
  }
}
```



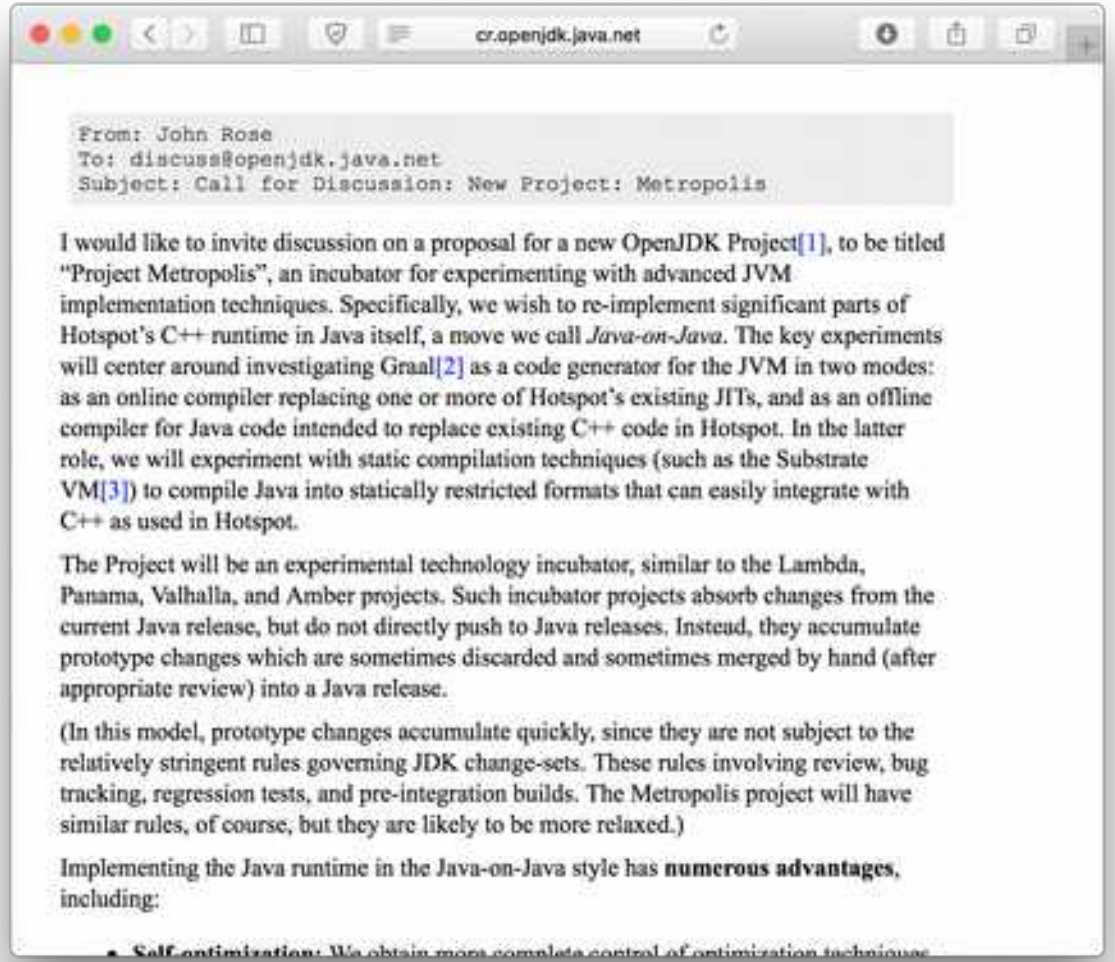


```
void workload() {
  monitor.enter();
  counter += 2;
  monitor.exit();
}
```



A final tier compiler

```
-XX:+UseJVMCICompiler
```



http://cr.openjdk.java.net/~jrose/metropolis/Metropolis-Proposal.html

Your own specific optimisations

Ahead of time compilation

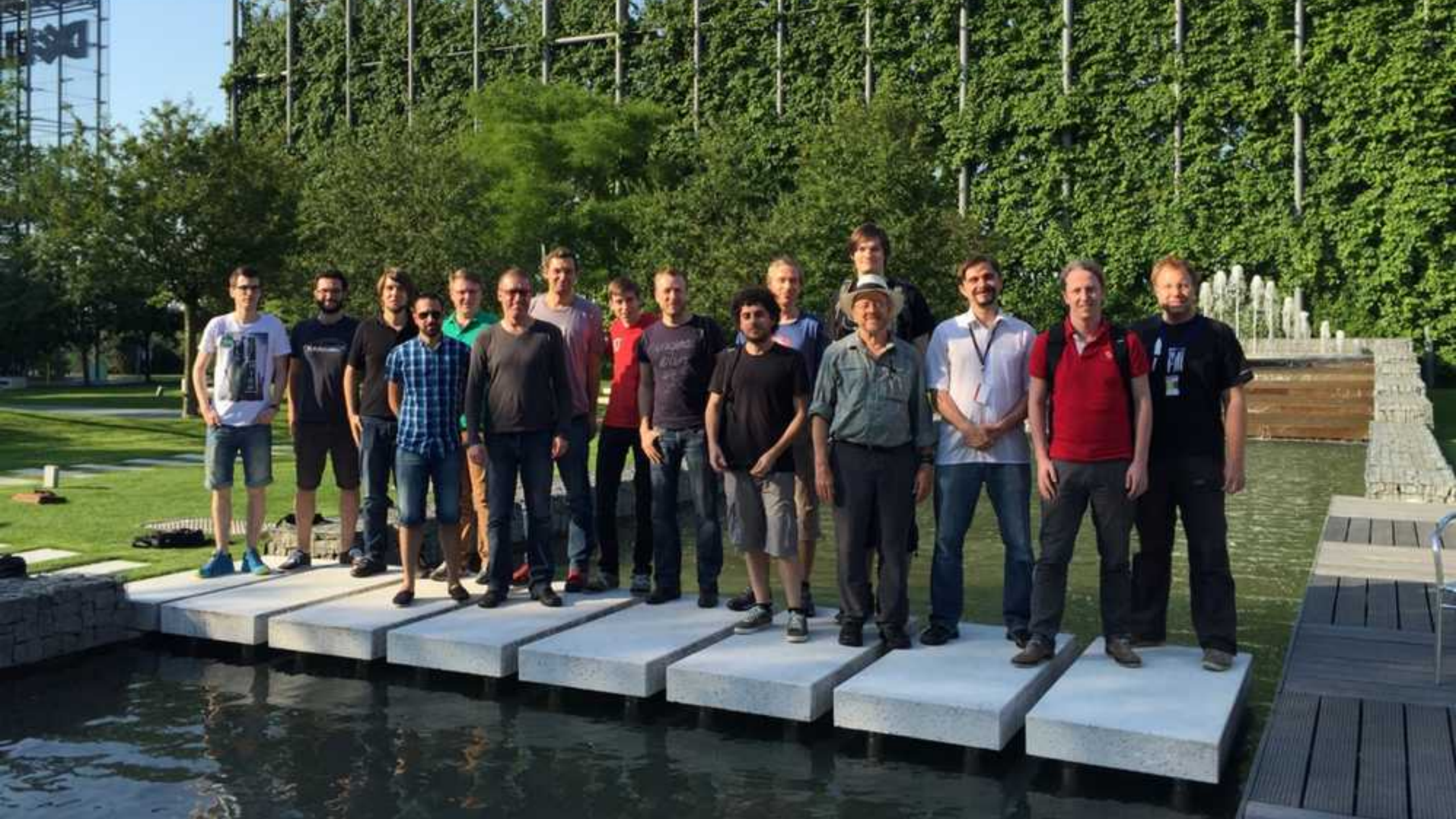
```
$ javac Hello.java

$ graalvm-0.28.2/bin/native-image Hello
    classlist:     966.44 ms
        (cap):     804.46 ms
        setup:   1,514.31 ms
  (typeflow):   2,580.70 ms
   (objects):     719.04 ms
  (features):      16.27 ms
    analysis:   3,422.58 ms
    universe:     262.09 ms
     (parse):     528.44 ms
    (inline):   1,259.94 ms
   (compile):   6,716.20 ms
     compile:   8,817.97 ms
       image:   1,070.29 ms
   debuginfo:     672.64 ms
       write:   1,797.45 ms
     [total]:  17,907.56 ms
```

ORACLE

```
$ ls -lh hello
-rwxr-xr-x  1 chrisseaton  staff   6.6M  4 Oct 18:35 hello

$ file ./hello
./hellojava: Mach-O 64-bit executable x86_64

$ time ./hello
Hello!

real    0m0.010s
user    0m0.003s
sys 0m0.003s
```

Truffle

Summary



Team

Oracle
Florian Angerer
Danilo Ansaloni
Stefan Anzinger
Martin Balin
Cosmin Basca
Daniele Bonetta
Dušan Bálek
Matthias Brantner
Lucas Braun
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Svatopluk Dědic
Martin Entlicher
Pit Fender
Francois Farquet
Brandon Fish
Matthias Grimmer
Christian Häubl
Peter Hofer
Bastian Hossbach
Christian Humer
Tomáš Hůrka
Mick Jordan

Oracle (continued)
Vojin Jovanovic
Anantha Kandukuri
Harshad Kasture
Cansu Kaynak
Peter Kessler
Duncan MacGregor
Jiří Maršík
Kevin Menard
Miloslav Metelka
Tomáš Myšík
Petr Pišl
Oleg Pliss
Jakub Podlešák
Aleksandar Prokopec
Tom Rodriguez
Roland Schatz
Benjamin Schlegel
Chris Seaton
Jiří Sedláček
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Boris Spasojevic
Lukas Stadler
Codrut Stancu

Oracle (continued)
Jan Štola
Tomáš Stupka
Farhan Tauheed
Jaroslav Tulach
Alexander Ulrich
Michael Van De Vanter
Aleksandar Vitorovic
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger
Tomáš Zezula
Yudi Zheng

Red Hat
Andrew Dinn
Andrew Haley

Intel
Michael Berg

Twitter
Chris Thalinger

Oracle Interns
Brian Belleville
Ondrej Douda
Juan Fumero
Miguel Garcia
Hugo Guiroux
Shams Imam
Berkin Ilbeyi
Hugo Kapp
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
Tristan Overney
Aleksandar Pejovic
David Piorkowski
Philipp Riedmann
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Oracle Alumni
Erik Eckstein
Michael Haupt
Christos Kotselidis
David Leibs
Adam Welc
Till Westmann

JKU Linz
Hanspeter Mössenböck
Benoit Daloze
Josef Eisl
Thomas Feichtinger
Josef Haider
Christian Huber
David Leopoldseder
Stefan Marr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

TU Berlin:
Volker Markl
Andreas Kunft
Jens Meiners
Tilmann Rabl

University of Edinburgh
Christophe Dubach
Juan José Fumero Alfonso
Ranjeet Singh
Toomas Remmelg

LaBRI
Floréal Morandat

University of California, Irvine
Michael Franz
Yeoul Na
Mohaned Qunaibit
Gulfem Savrun Yeniceri
Wei Zhang

Purdue University
Jan Vitek
Tomas Kalibera
Petr Maj
Lei Zhao

T. U. Dortmund
Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis
Duncan Temple Lang
Nicholas Ulle

University of Lugano, Switzerland
Walter Binder
Sun Haiyang

Safe Harbor Statement

The preceding is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Q&A

@JaroslavTulach
Oracle Labs

Integrated Cloud

Applications & Platform Services


ORACLE