ORACLE®

Understanding How Graal Works

a Java JIT Compiler Written in Java

Chris Seaton
Research Manager
Oracle Labs

chris.seaton@oracle.com
@ChrisGSeaton

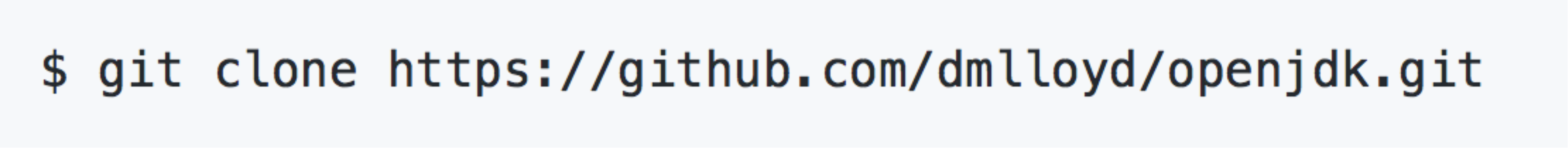
Safe Harbor Statement

The following 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.

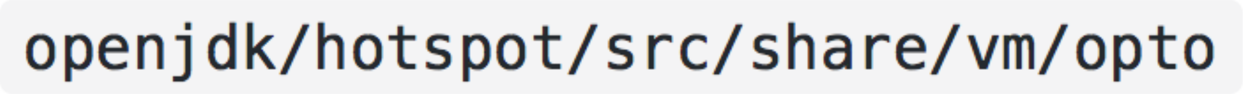
http://chrisseaton.com/rubytruffle/jokerconf17/

What is a JIT compiler?

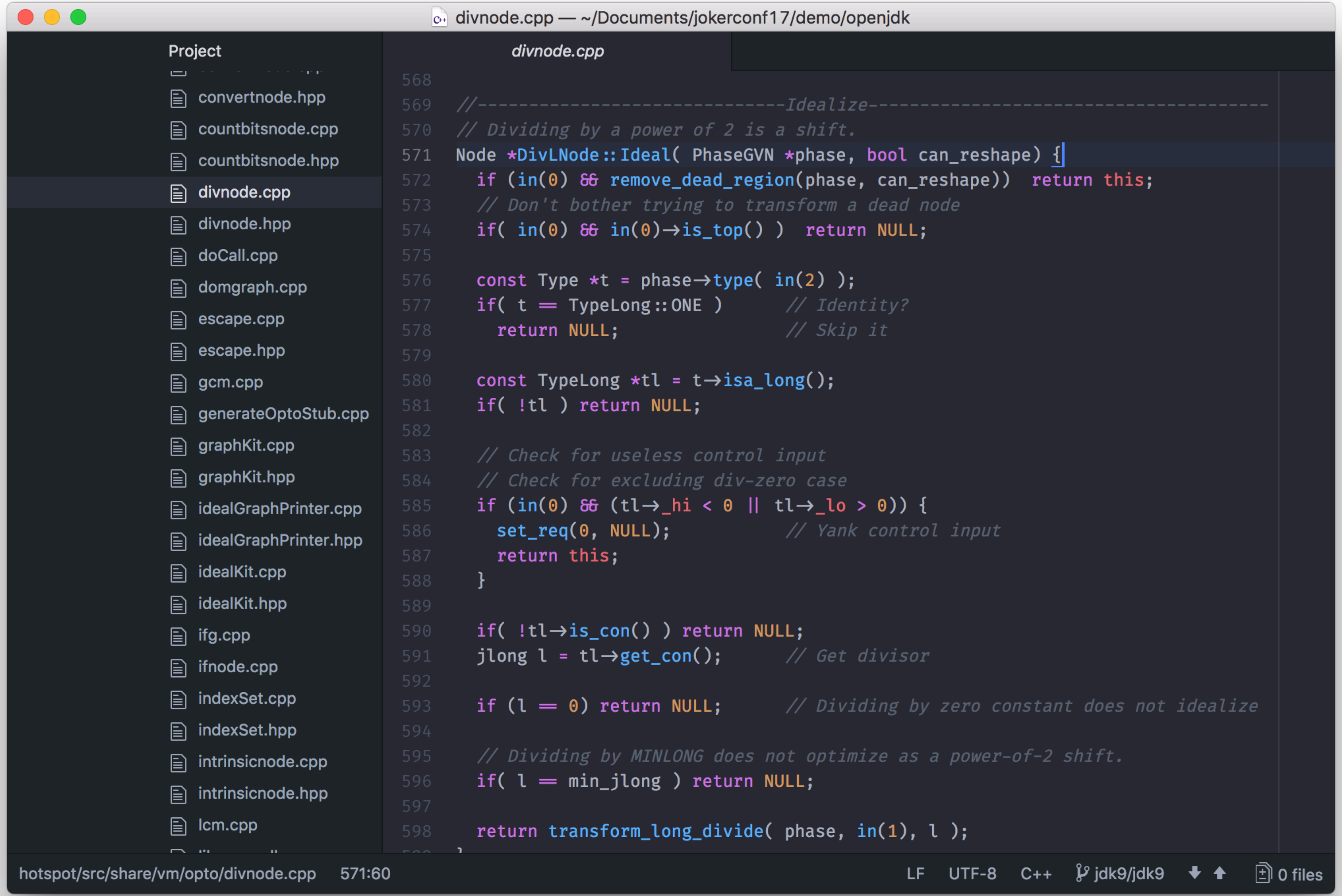
Why write a JIT compiler in Java?



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



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

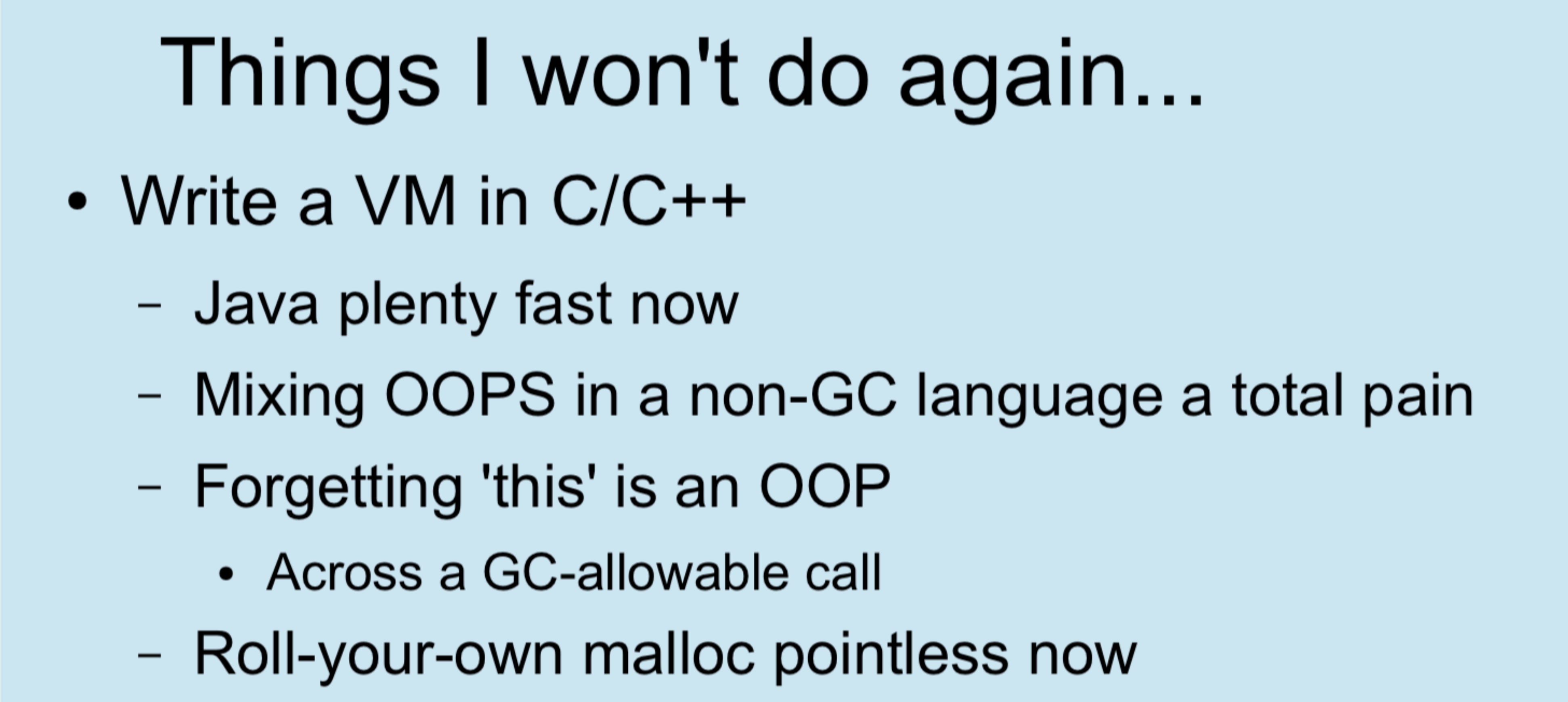


```cpp

//-------------------------------Idealize---------------------------------------
// Dividing by a power of 2 is a shift.
Node *DivLNode::Ideal( PhaseGVN *phase, bool can_reshape) {
  if (in(0) && remove_dead_region(phase, can_reshape))  return this;
  // Don't bother trying to transform a dead node
  if( in(0) && in(0)->is_top() )  return NULL;

  const Type *t = phase->type( in(2) );
  if( t == TypeLong::ONE )      // Identity?
    return NULL;                // Skip it

  const TypeLong *tl = t->isa_long();
  if( !tl ) return NULL;

  // Check for useless control input
  // Check for excluding div-zero case
  if (in(0) && (tl->_hi < 0 || tl->_lo > 0)) {
    set_req(0, NULL);           // Yank control input
    return this;
  }

  if( !tl->is_con() ) return NULL;
  jlong l = tl->get_con();      // Get divisor

  if (l == 0) return NULL;      // Dividing by zero constant does not idealize

  // Dividing by MINLONG does not optimize as a power-of-2 shift.
  if( l == min_jlong ) return NULL;

  return transform_long_divide( phase, in(1), l );
}
```

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-allocatable call
  - Roll-your-own malloc pointless now
- C2's BURS patterning matching
  - Harkens back to VAX days
  - Never needed on RISC
  - Not needed on X86 for a long time now
  - Adds an extra indirection in the JIT engineering

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

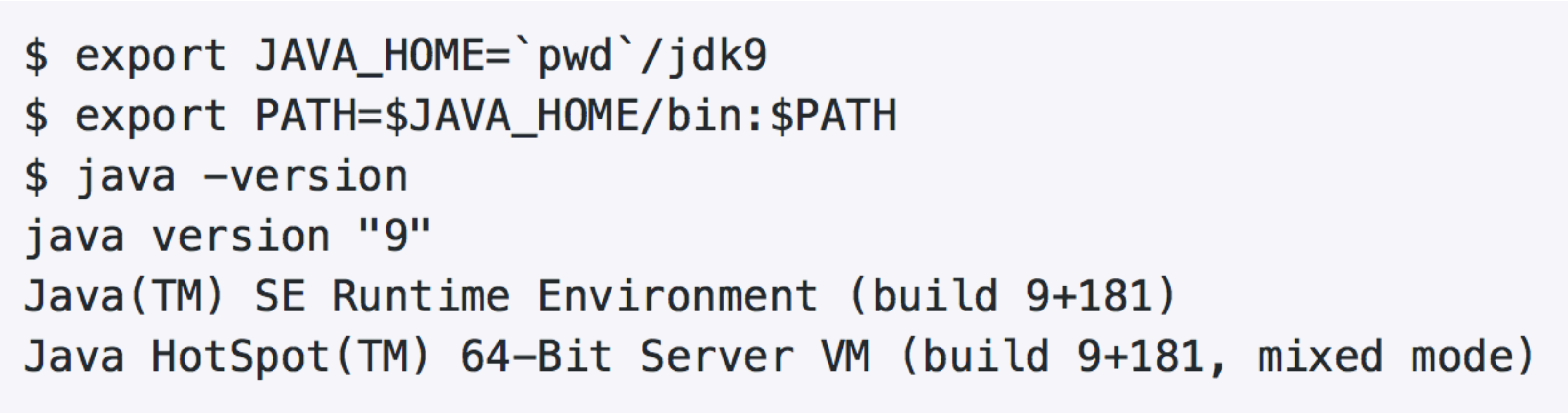


Things I won't do again...

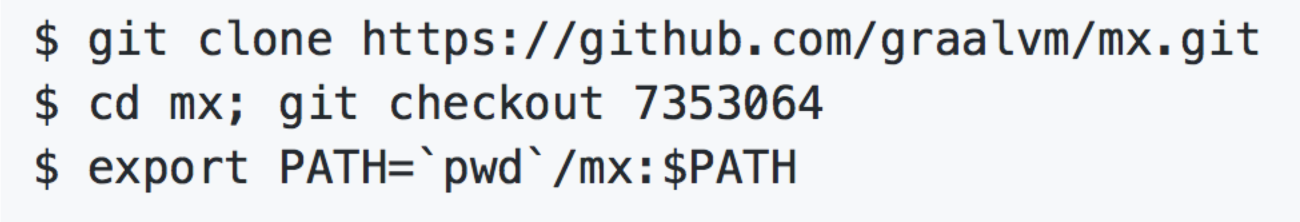
- Write a VM in C/C++
  - Java plenty fast now
  - Mixing OOPS in a non-GC language a total pain
  - Forgetting 'this' is an OOP
    - Across a GC-allowable call
  - Roll-your-own malloc pointless now

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

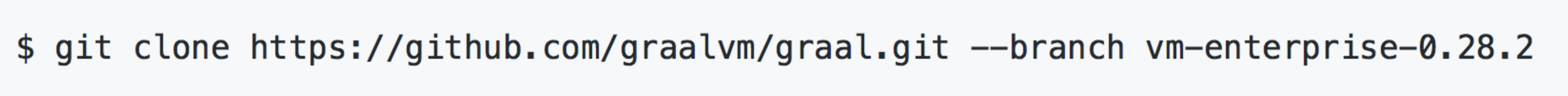
Setting up Graal



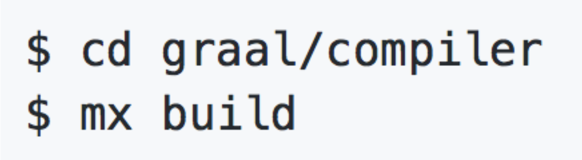
```
$ export JAVA_HOME=`pwd`/jdk9
$ export PATH=$JAVA_HOME/bin:$PATH
$ java -version
java version "9"
Java(TM) SE Runtime Environment (build 9+181)
Java HotSpot(TM) 64-Bit Server VM (build 9+181, mixed mode)
```



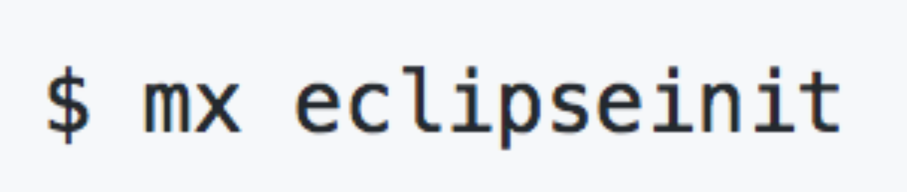
```
$ git clone https://github.com/graalvm/mx.git
$ cd mx; git checkout 7353064
$ export PATH=`pwd`/mx:$PATH
```



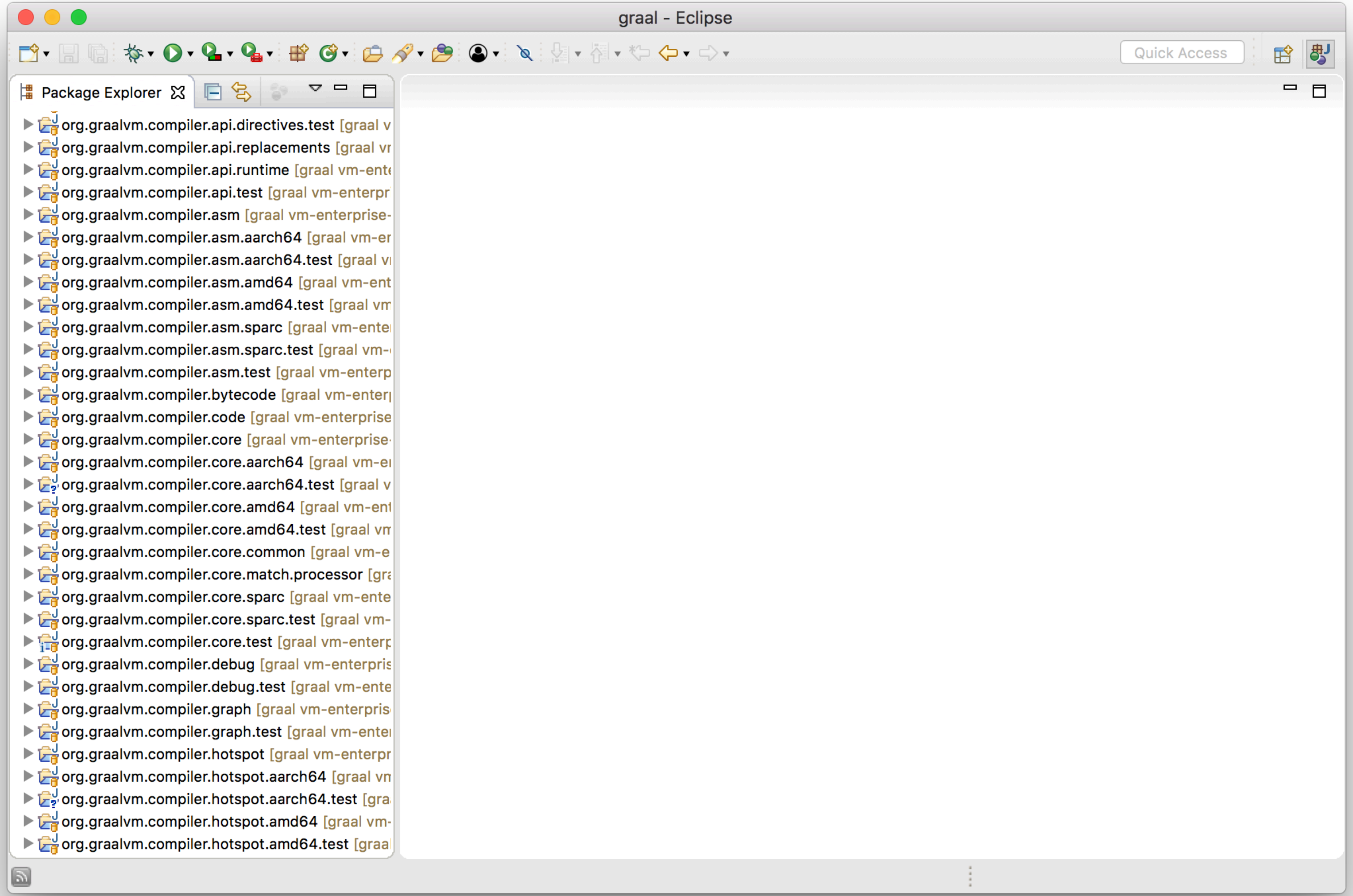
```
$ git clone https://github.com/graalvm/graal.git --branch vm-enterprise-0.28.2
```



```
$ cd graal/compiler
$ mx build
```



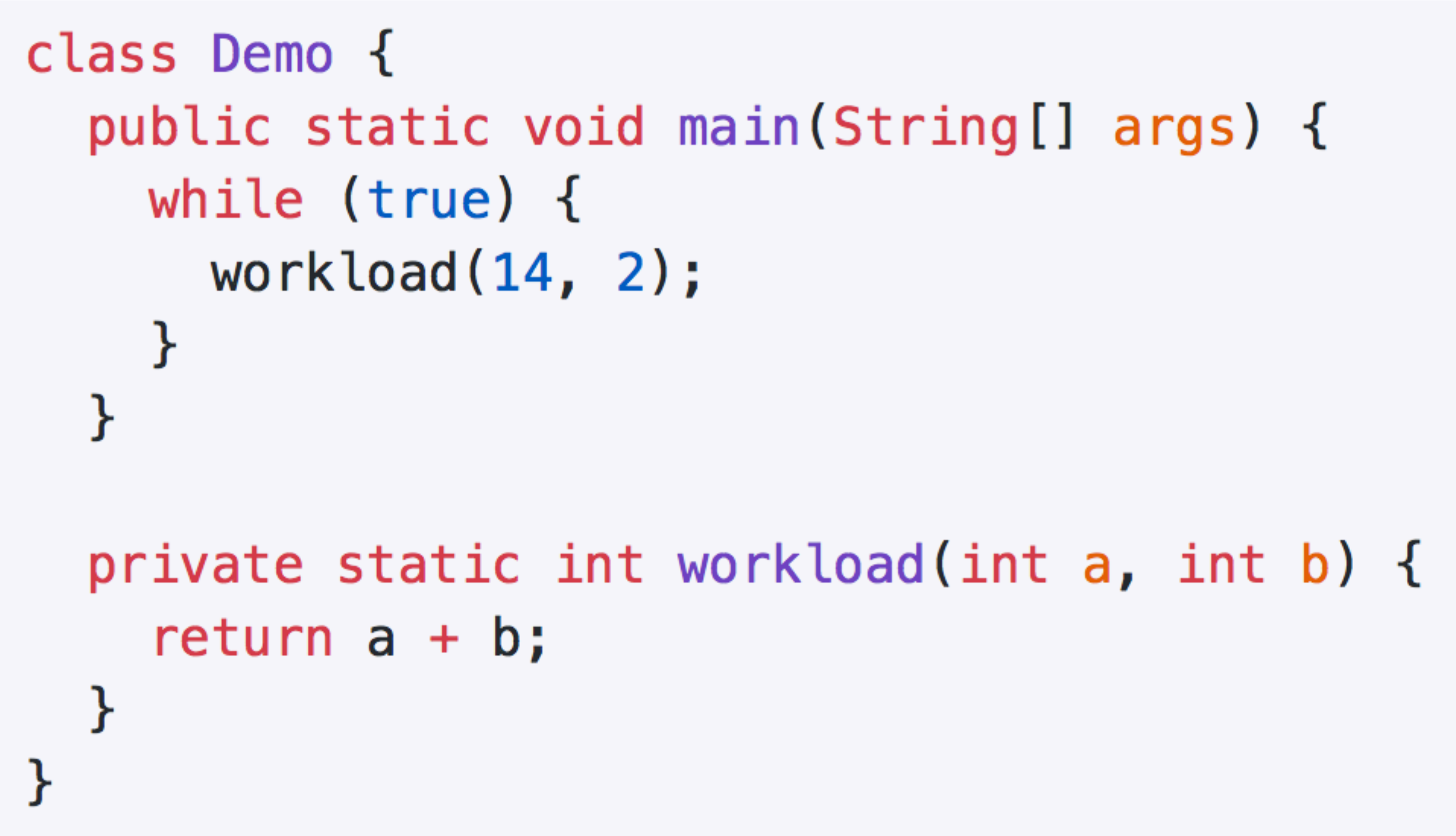
```
$ mx eclipseinit
```



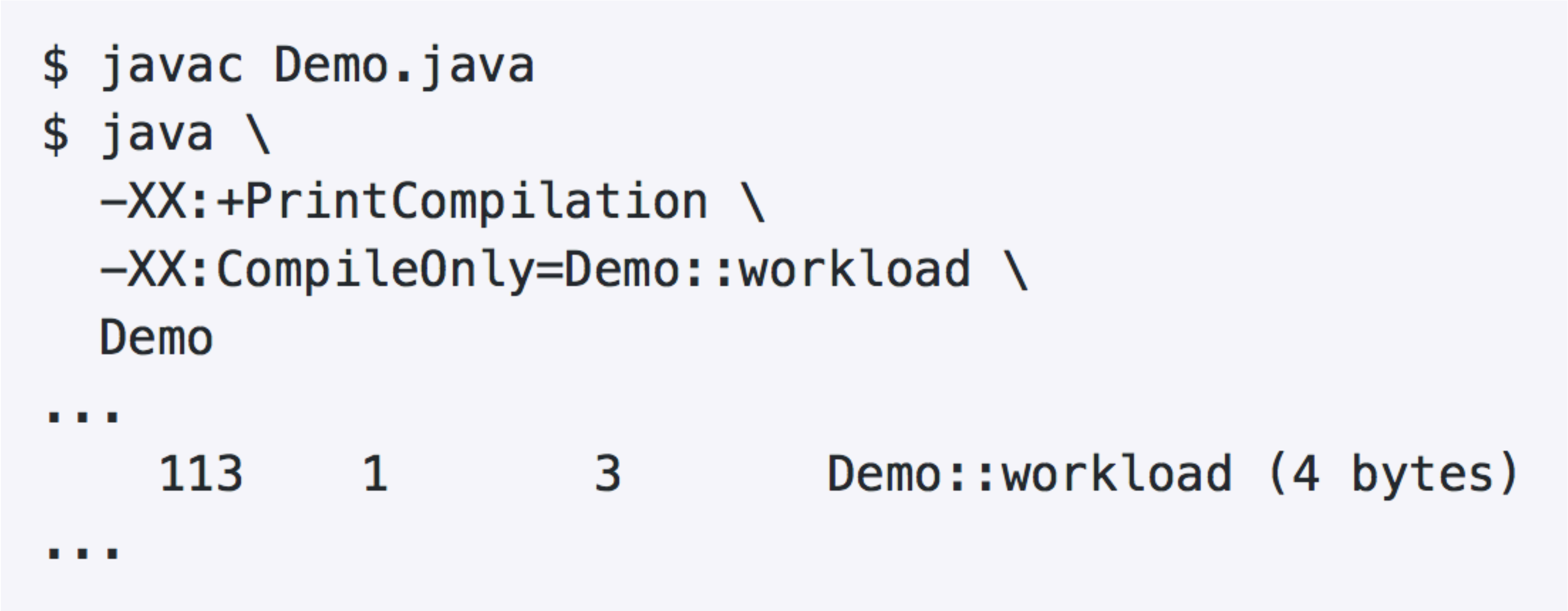
graal - Eclipse

Package Explorer

- org.graalvm.compiler.api.directives.test [graal v
- org.graalvm.compiler.api.replacements [graal vr
- org.graalvm.compiler.api.runtime [graal vm-ente
- org.graalvm.compiler.api.test [graal vm-enterpr
- org.graalvm.compiler.asm [graal vm-enterprise-
- org.graalvm.compiler.asm.aarch64 [graal vm-er
- org.graalvm.compiler.asm.aarch64.test [graal vi
- org.graalvm.compiler.asm.amd64 [graal vm-ent
- org.graalvm.compiler.asm.amd64.test [graal vm
- org.graalvm.compiler.asm.sparc [graal vm-ente
- org.graalvm.compiler.asm.sparc.test [graal vm-
- org.graalvm.compiler.asm.test [graal vm-enterp
- org.graalvm.compiler.bytecode [graal vm-enter
- org.graalvm.compiler.code [graal vm-enterprise
- org.graalvm.compiler.core [graal vm-enterprise-
- org.graalvm.compiler.core.aarch64 [graal vm-er
- org.graalvm.compiler.core.aarch64.test [graal v
- org.graalvm.compiler.core.amd64 [graal vm-ent
- org.graalvm.compiler.core.amd64.test [graal vm
- org.graalvm.compiler.core.common [graal vm-e
- org.graalvm.compiler.core.match.processor [gra
- org.graalvm.compiler.core.sparc [graal vm-ente
- org.graalvm.compiler.core.sparc.test [graal vm-
- org.graalvm.compiler.core.test [graal vm-enterp
- org.graalvm.compiler.debug [graal vm-enterpris
- org.graalvm.compiler.debug.test [graal vm-ente
- org.graalvm.compiler.graph [graal vm-enterpris
- org.graalvm.compiler.graph.test [graal vm-ente
- org.graalvm.compiler.hotspot [graal vm-enterpr
- org.graalvm.compiler.hotspot.aarch64 [graal vm
- org.graalvm.compiler.hotspot.aarch64.test [gra
- org.graalvm.compiler.hotspot.amd64 [graal vm-
- org.graalvm.compiler.hotspot.amd64.test [graa



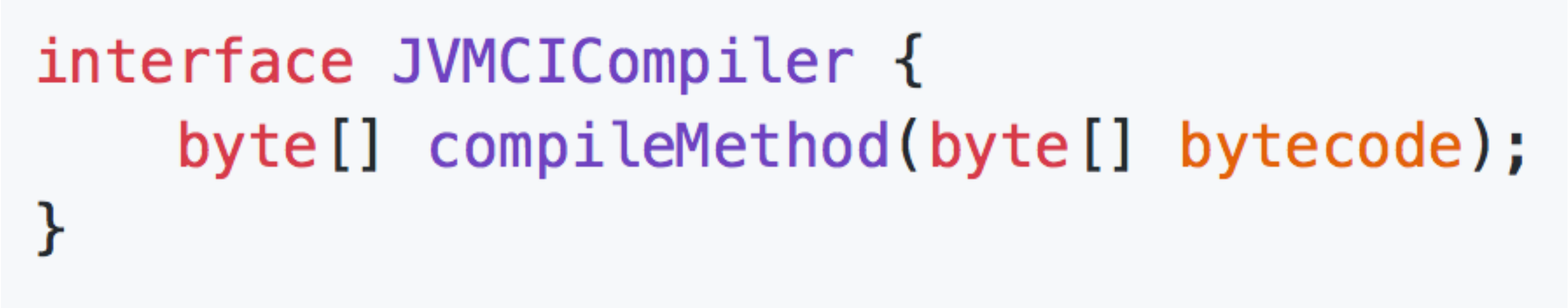
```java
class Demo {
  public static void main(String[] args) {
    while (true) {
      workload(14, 2);
    }
  }

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



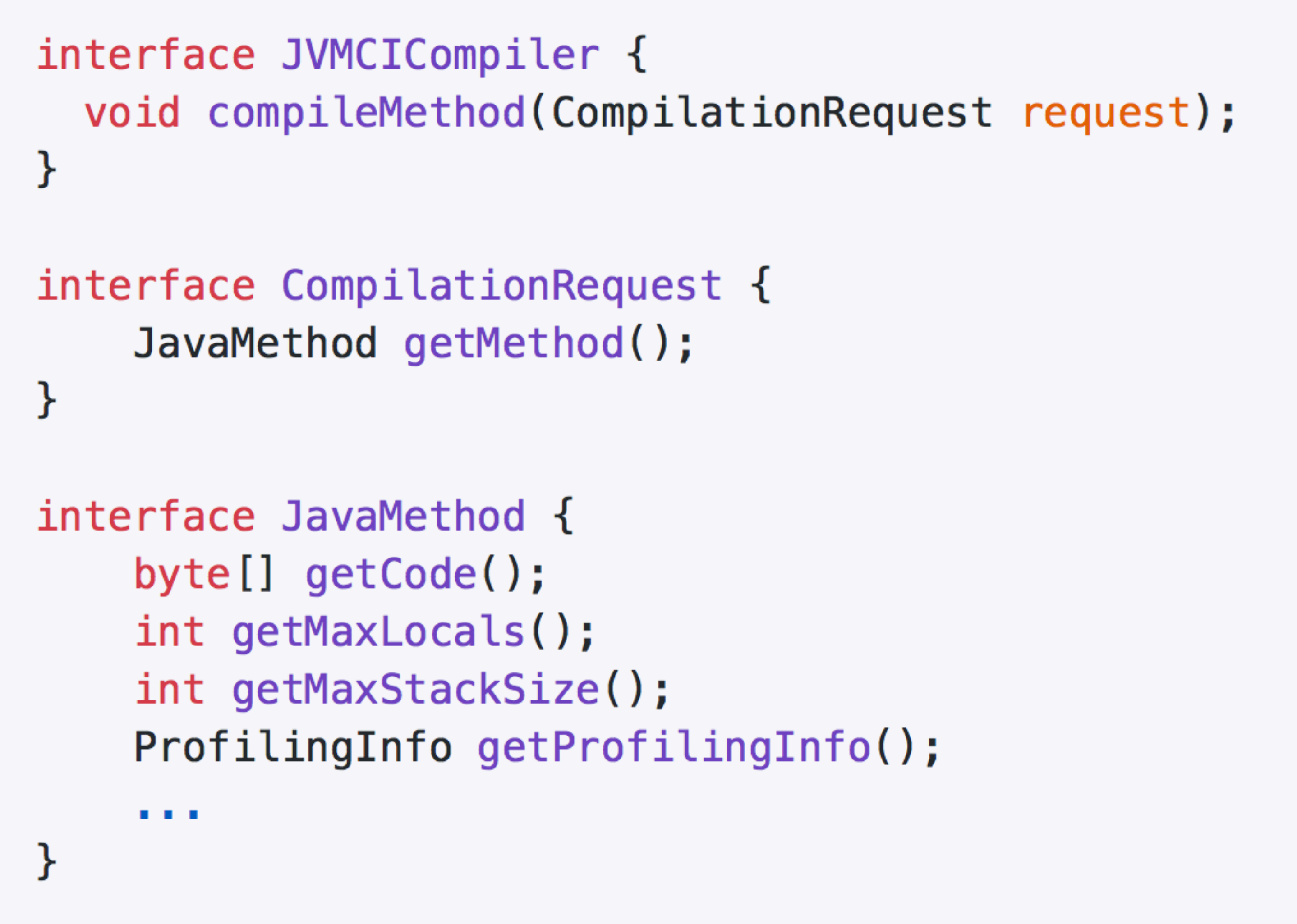
```
$ javac Demo.java
$ java \
  -XX:+PrintCompilation \
  -XX:CompileOnly=Demo::workload \
  Demo
...
    113    1       3       Demo::workload (4 bytes)
...
```

```
$ java \
  --module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/...... \
  --upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler \
  -XX:-TieredCompilation \
  -XX:+PrintCompilation \
  -XX:CompileOnly=Demo::workload \
  Demo
...
    583   25             Demo::workload (4 bytes)
...
```

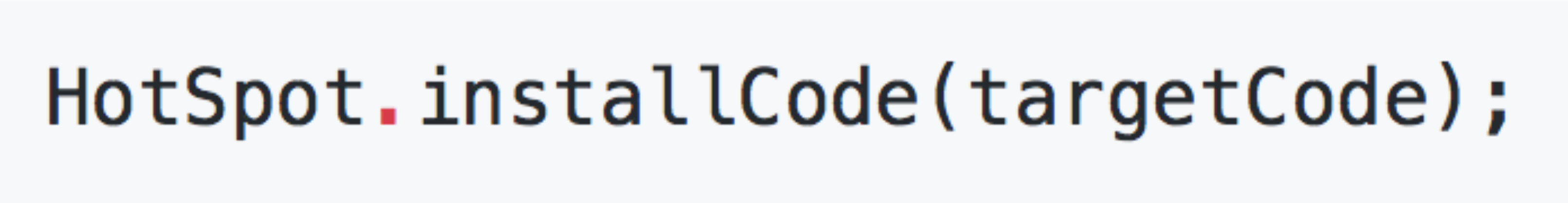
The JVM compiler interface



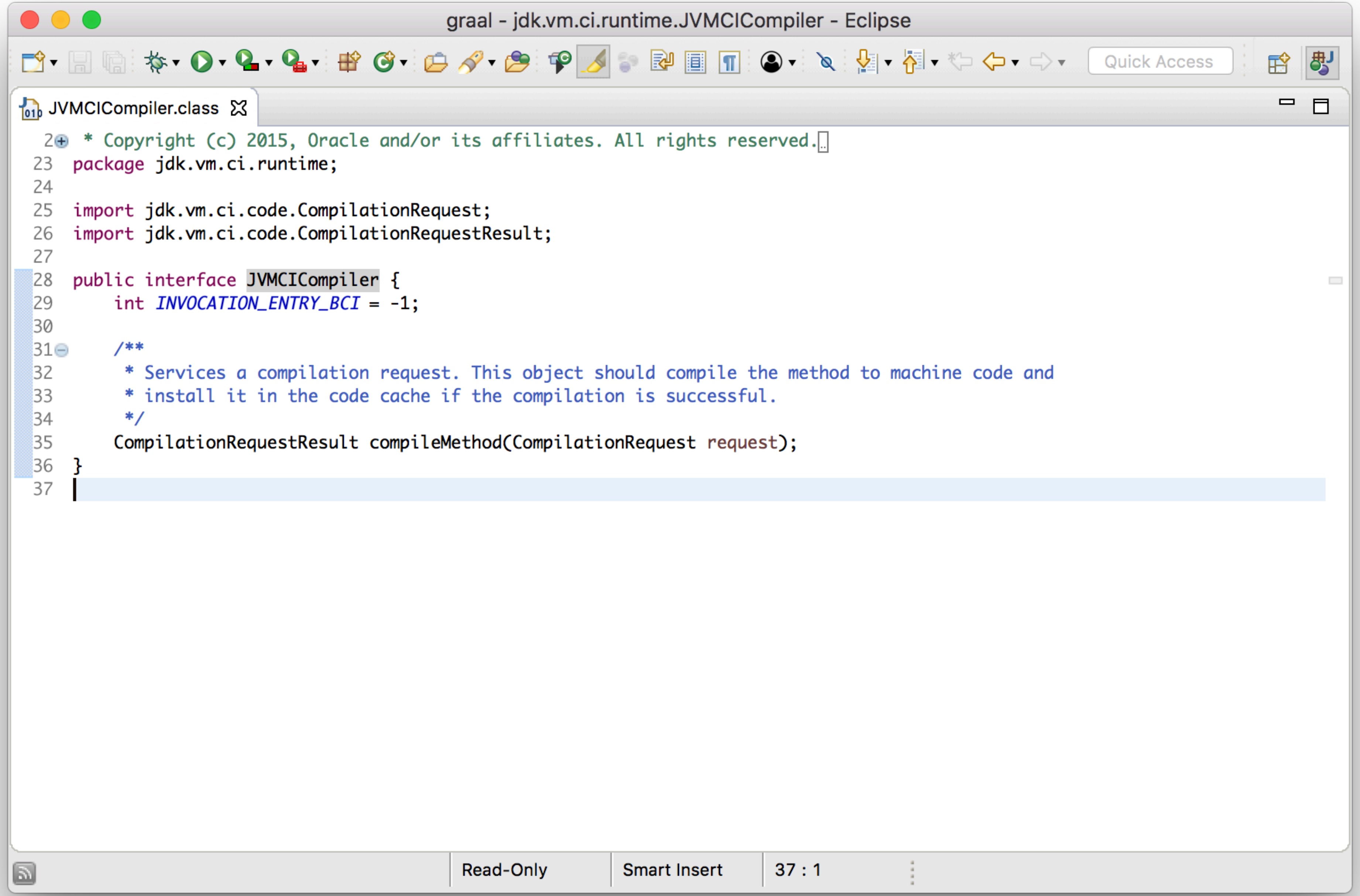
```
interface JVMCICompiler {
    byte[] compileMethod(byte[] bytecode);
}
```



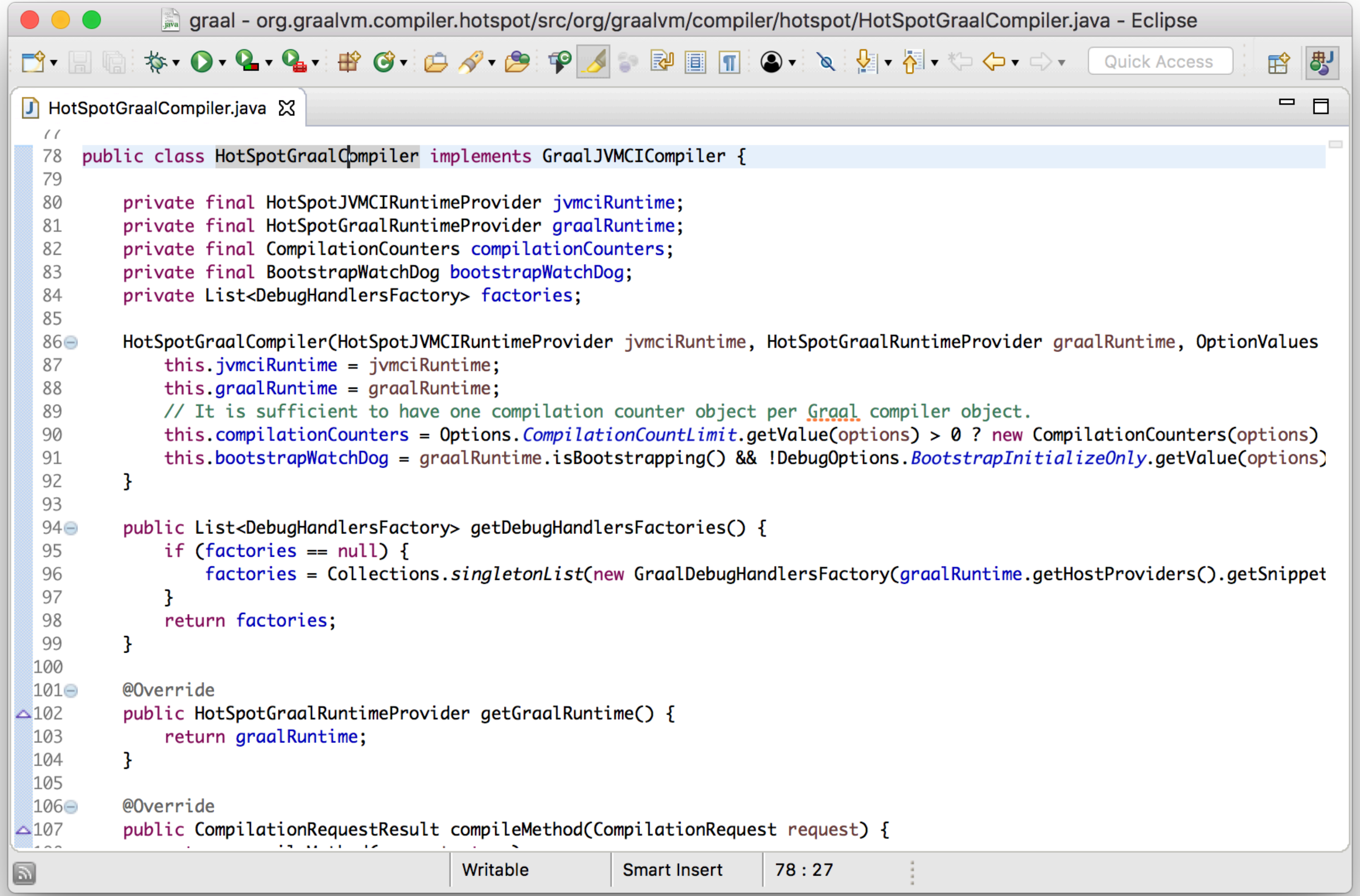
```java
interface JVMCICompiler {
  void compileMethod(CompilationRequest request);
}

interface CompilationRequest {
    JavaMethod getMethod();
}

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



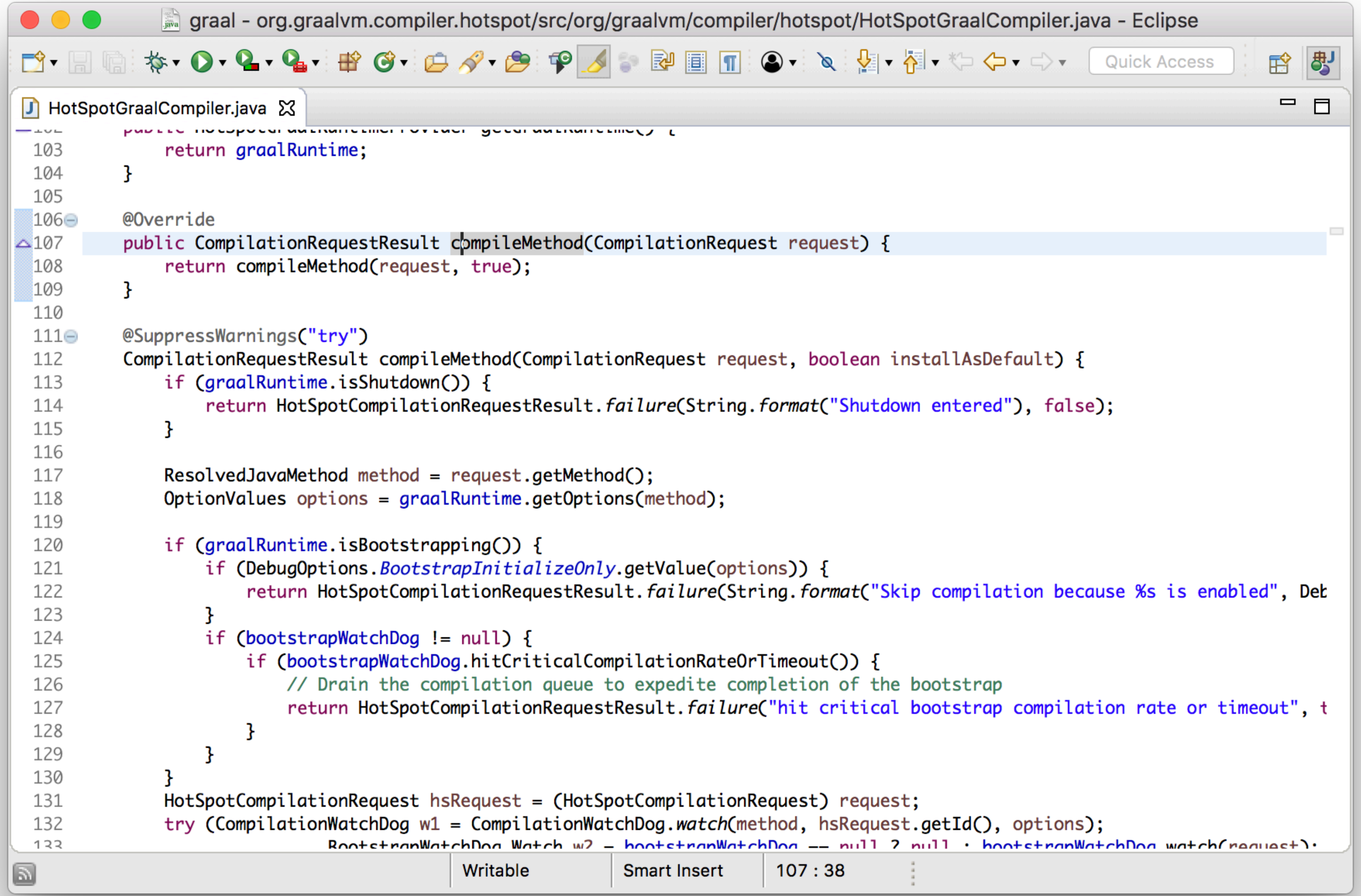
```
HotSpot.installCode(targetCode);
```



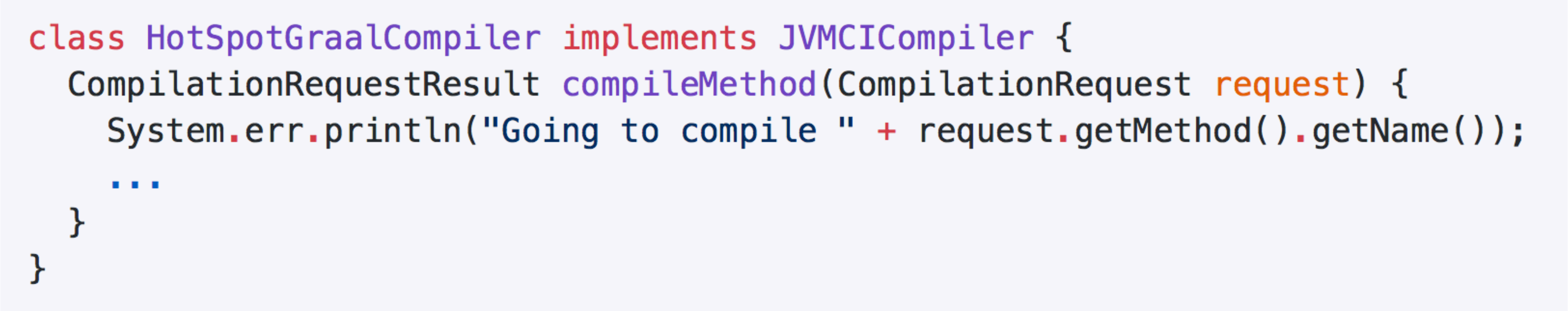
```
 * Copyright (c) 2015, Oracle and/or its affiliates. All rights reserved.
package jdk.vm.ci.runtime;

import jdk.vm.ci.code.CompilationRequest;
import jdk.vm.ci.code.CompilationRequestResult;

public interface JVMCICompiler {
    int INVOCATION_ENTRY_BCI = -1;

    /**
     * Services a compilation request. This object should compile the method to machine code and
     * install it in the code cache if the compilation is successful.
     */
    CompilationRequestResult compileMethod(CompilationRequest request);
}

```


graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse
Quick Access
HotSpotGraalCompiler.java
Writable
Smart Insert
78 : 27

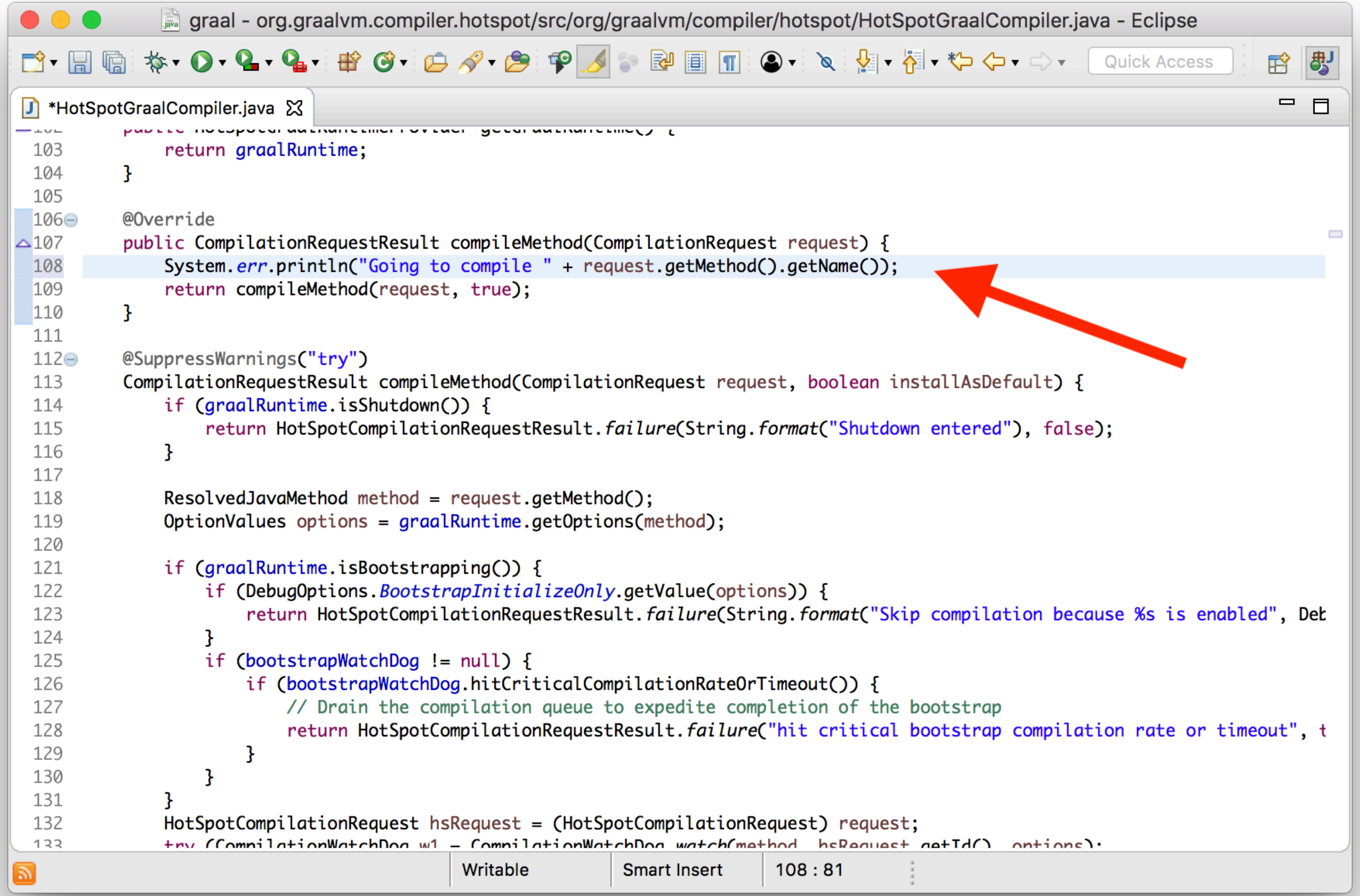
```java
public class HotSpotGraalCompiler implements GraalJVMCICompiler {

    private final HotSpotJVMCIRuntimeProvider jvmciRuntime;
    private final HotSpotGraalRuntimeProvider graalRuntime;
    private final CompilationCounters compilationCounters;
    private final BootstrapWatchDog bootstrapWatchDog;
    private List<DebugHandlersFactory> factories;

    HotSpotGraalCompiler(HotSpotJVMCIRuntimeProvider jvmciRuntime, HotSpotGraalRuntimeProvider graalRuntime, OptionValues
        this.jvmciRuntime = jvmciRuntime;
        this.graalRuntime = graalRuntime;
        // It is sufficient to have one compilation counter object per Graal compiler object.
        this.compilationCounters = Options.CompilationCountLimit.getValue(options) > 0 ? new CompilationCounters(options)
        this.bootstrapWatchDog = graalRuntime.isBootstrapping() && !DebugOptions.BootstrapInitializeOnly.getValue(options)
    }

    public List<DebugHandlersFactory> getDebugHandlersFactories() {
        if (factories == null) {
            factories = Collections.singletonList(new GraalDebugHandlersFactory(graalRuntime.getHostProviders().getSnippet
        }
        return factories;
    }

    @Override
    public HotSpotGraalRuntimeProvider getGraalRuntime() {
        return graalRuntime;
    }

    @Override
    public CompilationRequestResult compileMethod(CompilationRequest request) {
```


graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse
Quick Access
HotSpotGraalCompiler.java
Writable
Smart Insert
107 : 38

```
    public HotSpotGraalRuntimeProvider getGraalRuntime() {
        return graalRuntime;
    }

    @Override
    public CompilationRequestResult compileMethod(CompilationRequest request) {
        return compileMethod(request, true);
    }

    @SuppressWarnings("try")
    CompilationRequestResult compileMethod(CompilationRequest request, boolean installAsDefault) {
        if (graalRuntime.isShutdown()) {
            return HotSpotCompilationRequestResult.failure(String.format("Shutdown entered"), false);
        }

        ResolvedJavaMethod method = request.getMethod();
        OptionValues options = graalRuntime.getOptions(method);

        if (graalRuntime.isBootstrapping()) {
            if (DebugOptions.BootstrapInitializeOnly.getValue(options)) {
                return HotSpotCompilationRequestResult.failure(String.format("Skip compilation because %s is enabled", Deb
            }
            if (bootstrapWatchDog != null) {
                if (bootstrapWatchDog.hitCriticalCompilationRateOrTimeout()) {
                    // Drain the compilation queue to expedite completion of the bootstrap
                    return HotSpotCompilationRequestResult.failure("hit critical bootstrap compilation rate or timeout", t
                }
            }
        }
        HotSpotCompilationRequest hsRequest = (HotSpotCompilationRequest) request;
        try (CompilationWatchDog w1 = CompilationWatchDog.watch(method, hsRequest.getId(), options);
                        BootstrapWatchDog.Watch w2 = bootstrapWatchDog == null ? null : bootstrapWatchDog.watch(request);
```



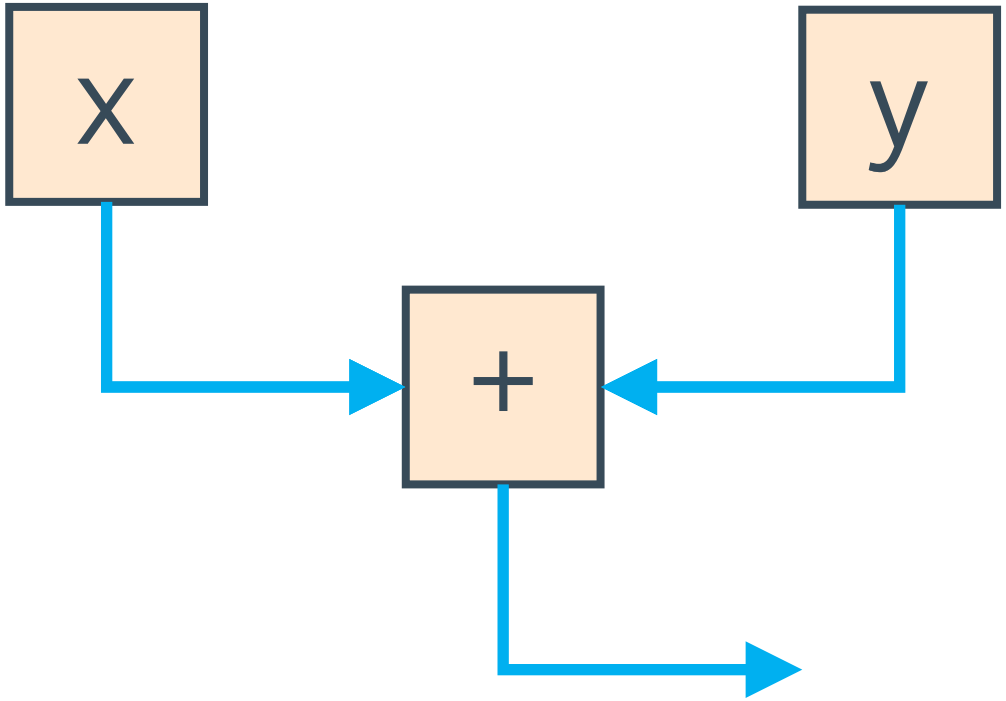
```java
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationRequestResult compileMethod(CompilationRequest request) {
    System.err.println("Going to compile " + request.getMethod().getName());
    ...
  }
}
```


graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse
Quick Access
*HotSpotGraalCompiler.java
Writable
Smart Insert
108 : 81

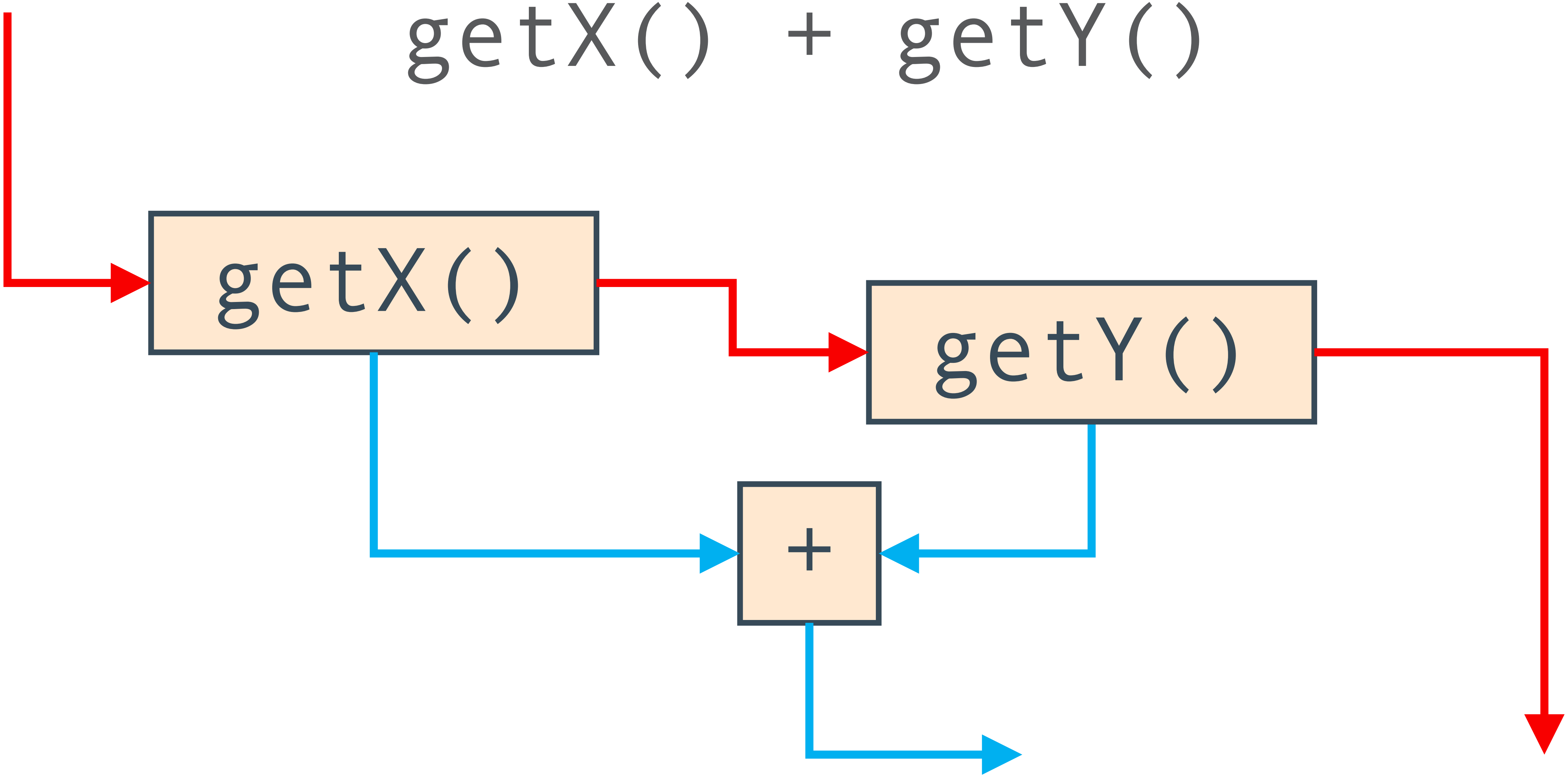
```java
    public HotSpotGraalRuntimeProvider getGraalRuntime() {
        return graalRuntime;
    }

    @Override
    public CompilationRequestResult compileMethod(CompilationRequest request) {
        System.err.println("Going to compile " + request.getMethod().getName());
        return compileMethod(request, true);
    }

    @SuppressWarnings("try")
    CompilationRequestResult compileMethod(CompilationRequest request, boolean installAsDefault) {
        if (graalRuntime.isShutdown()) {
            return HotSpotCompilationRequestResult.failure(String.format("Shutdown entered"), false);
        }

        ResolvedJavaMethod method = request.getMethod();
        OptionValues options = graalRuntime.getOptions(method);

        if (graalRuntime.isBootstrapping()) {
            if (DebugOptions.BootstrapInitializeOnly.getValue(options)) {
                return HotSpotCompilationRequestResult.failure(String.format("Skip compilation because %s is enabled", Deb
            }
            if (bootstrapWatchDog != null) {
                if (bootstrapWatchDog.hitCriticalCompilationRateOrTimeout()) {
                    // Drain the compilation queue to expedite completion of the bootstrap
                    return HotSpotCompilationRequestResult.failure("hit critical bootstrap compilation rate or timeout", t
                }
            }
        }
        HotSpotCompilationRequest hsRequest = (HotSpotCompilationRequest) request;
        try (CompilationWatchDog w1 = CompilationWatchDog.watch(method, hsRequest.getId(), options);
$ java \
  --module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/modules/..... \
  --upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler \
  -XX:-TieredCompilation \
  -XX:CompileOnly=Demo::workload \
  Demo
Going to compile workload
```

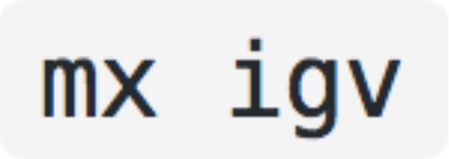
The Graal graph

x + y

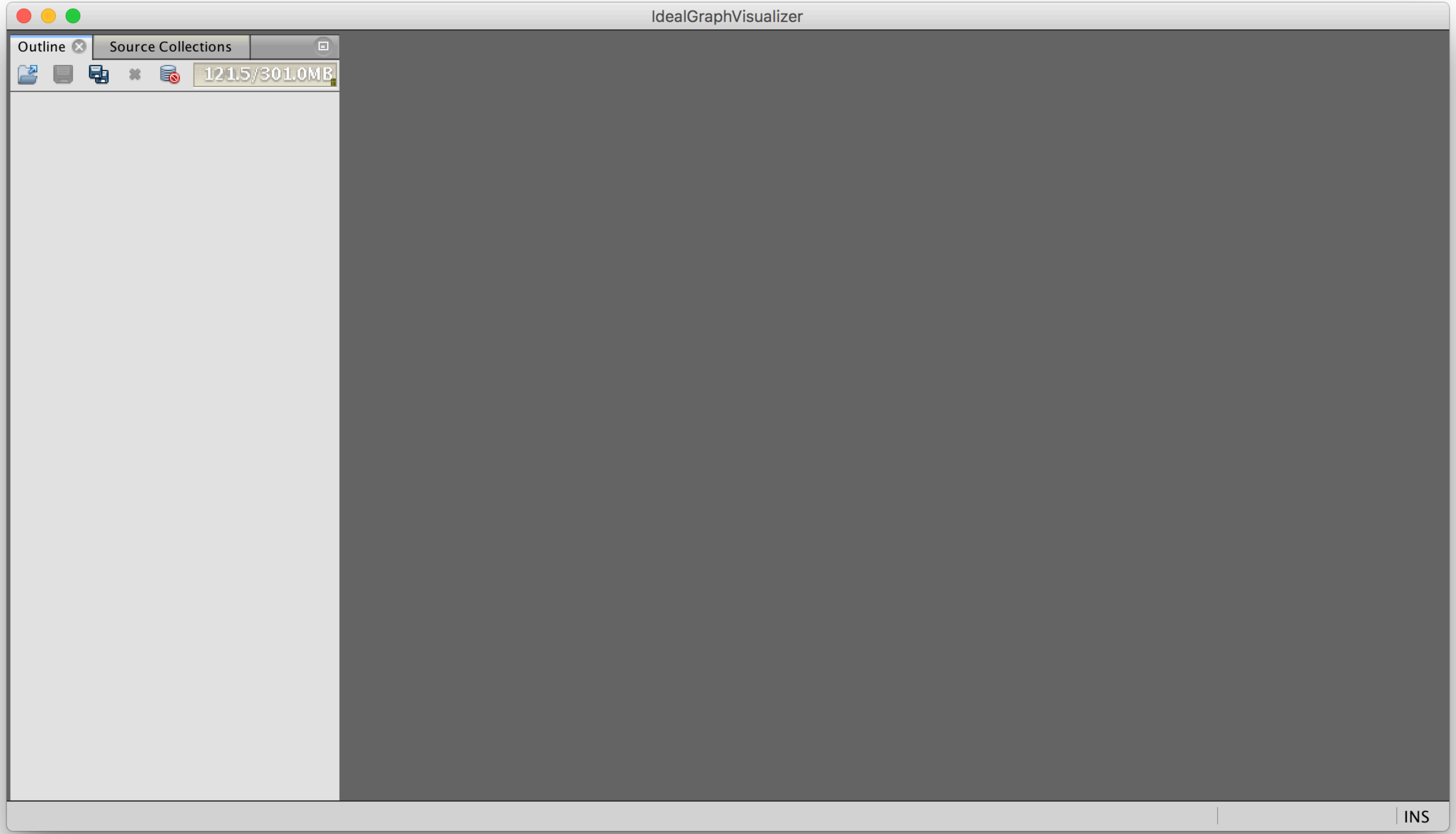


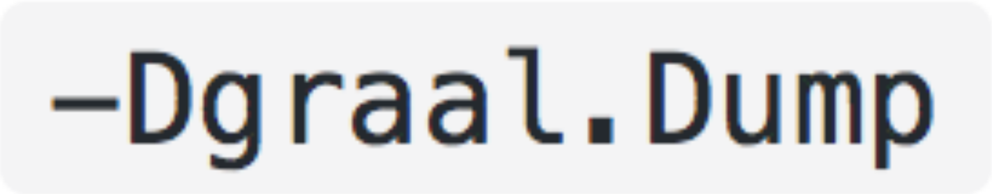
getX() + getY()



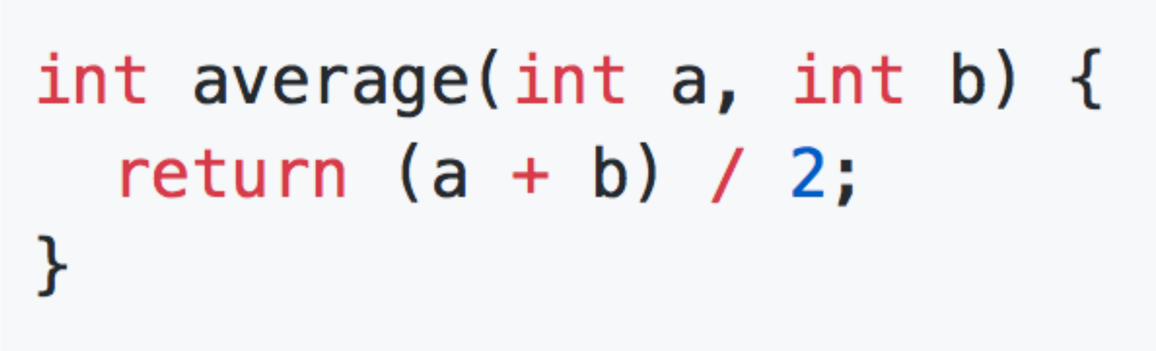


```
mx igv
```

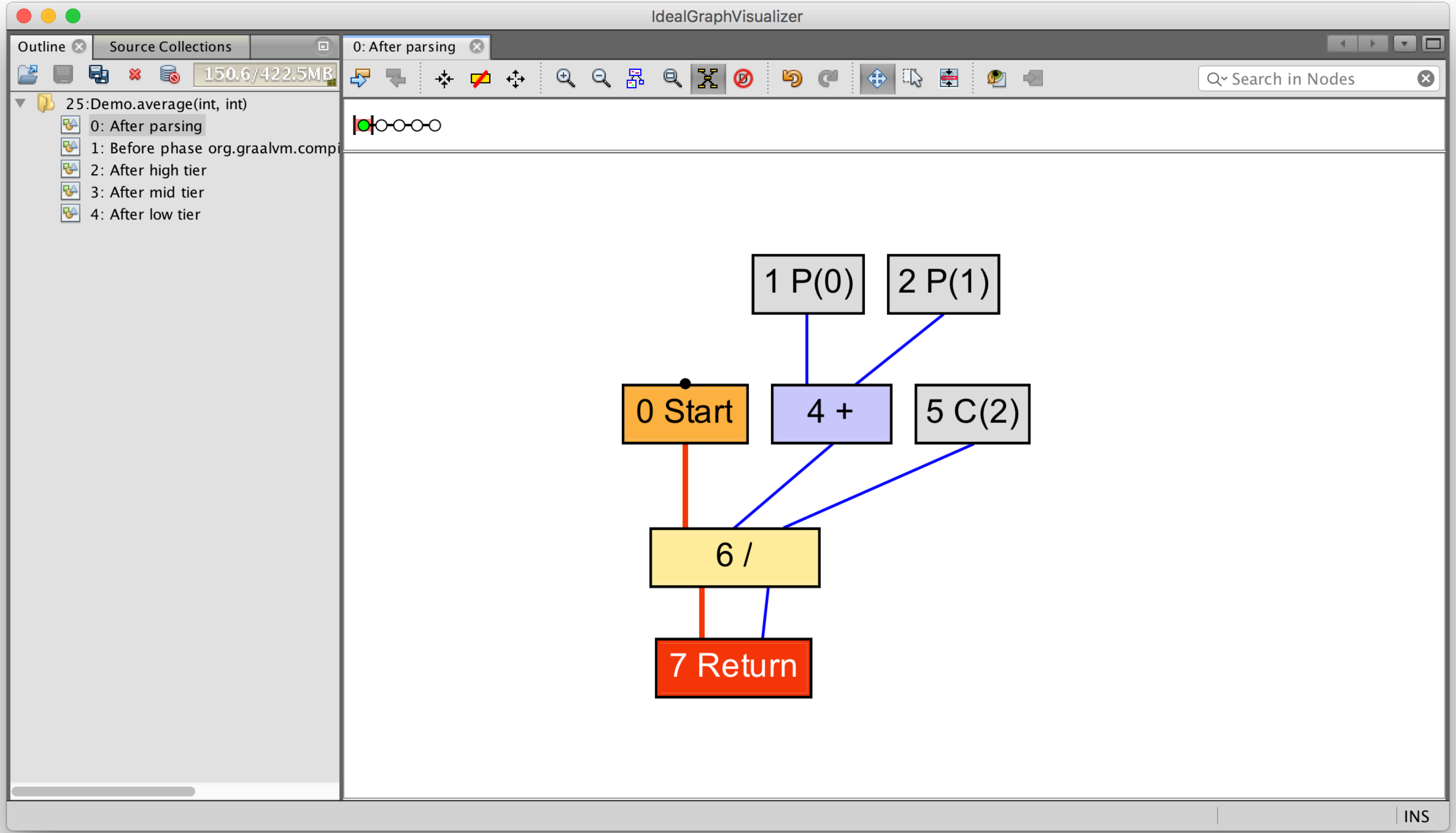

IdealGraphVisualizer
Outline
Source Collections
121.5/301.0MB
INS

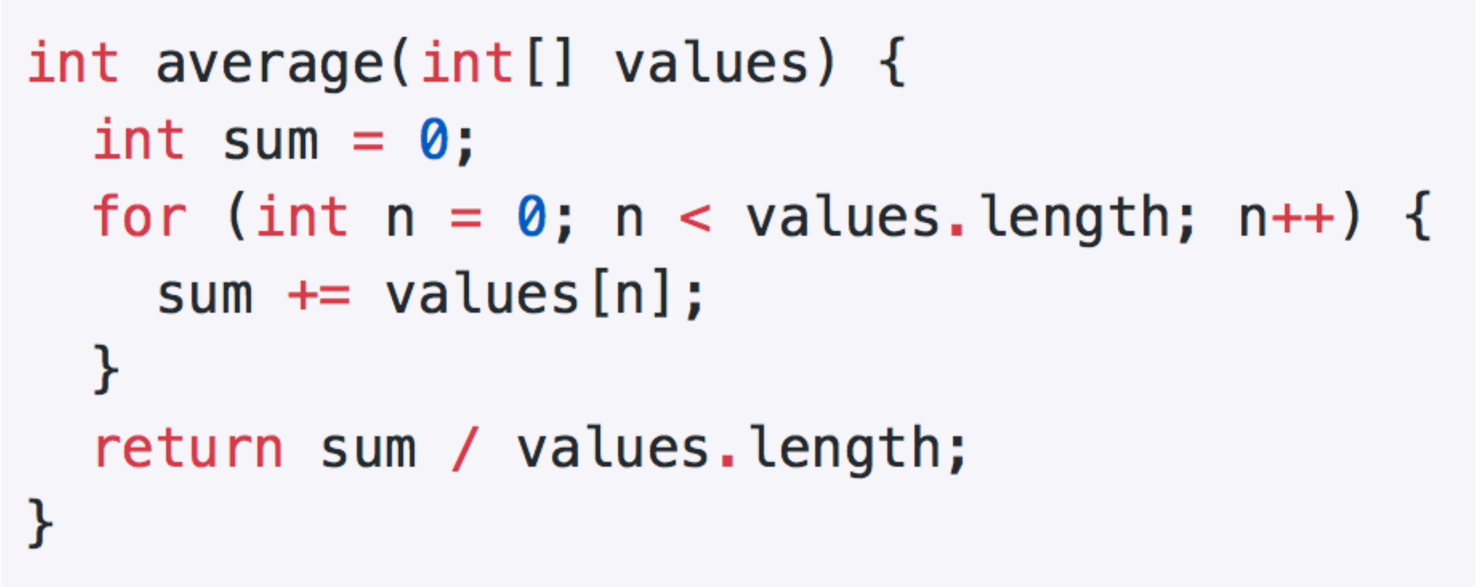


```
-Dgraal.Dump
```

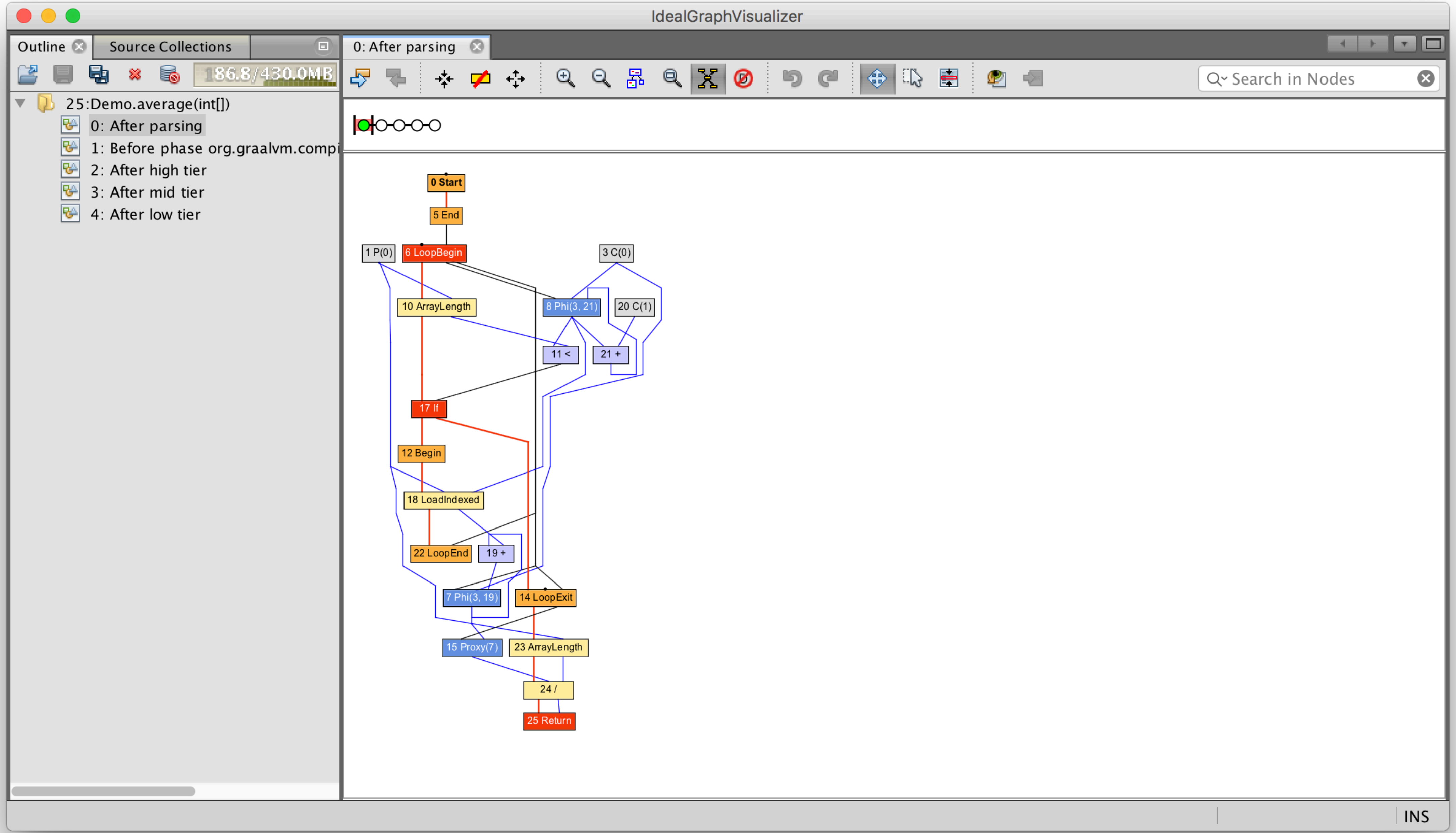


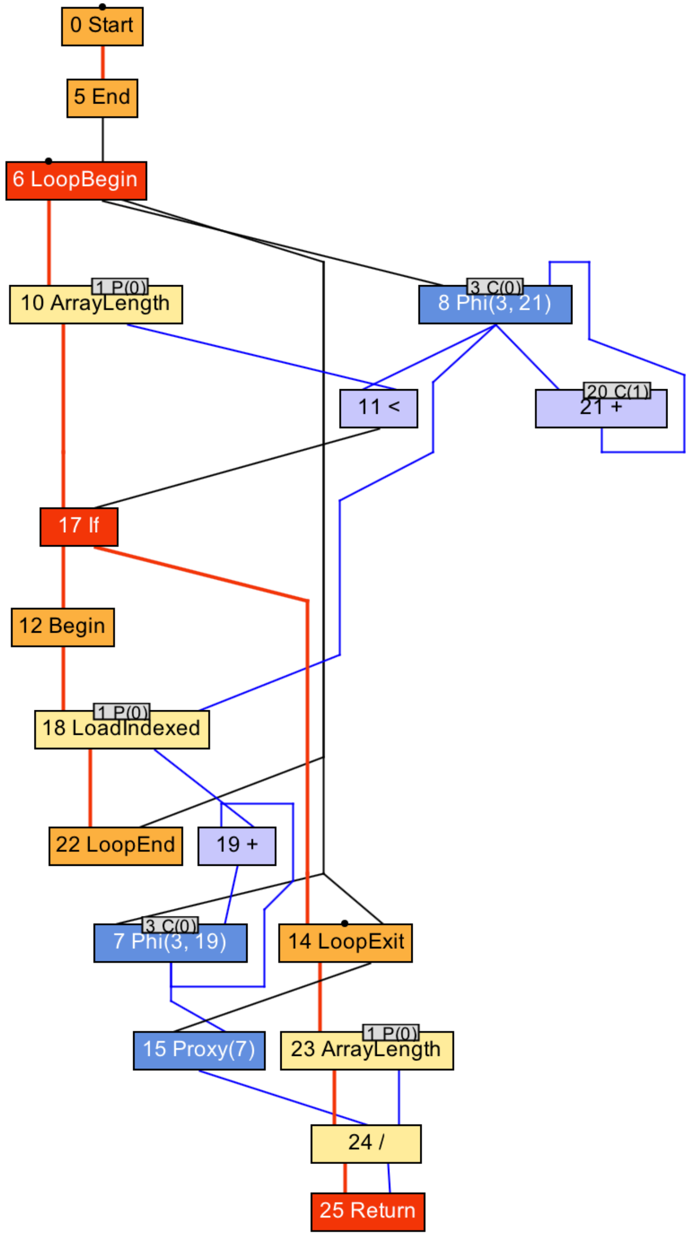
```
int average(int a, int b) {
  return (a + b) / 2;
}
```


IdealGraphVisualizer
Outline
Source Collections
150.6/422.5MB
25:Demo.average(int, int)
0: After parsing
1: Before phase org.graalvm.compi
2: After high tier
3: After mid tier
4: After low tier
0: After parsing
Search in Nodes
1 P(0)
2 P(1)
0 Start
4 +
5 C(2)
6 /
7 Return
INS



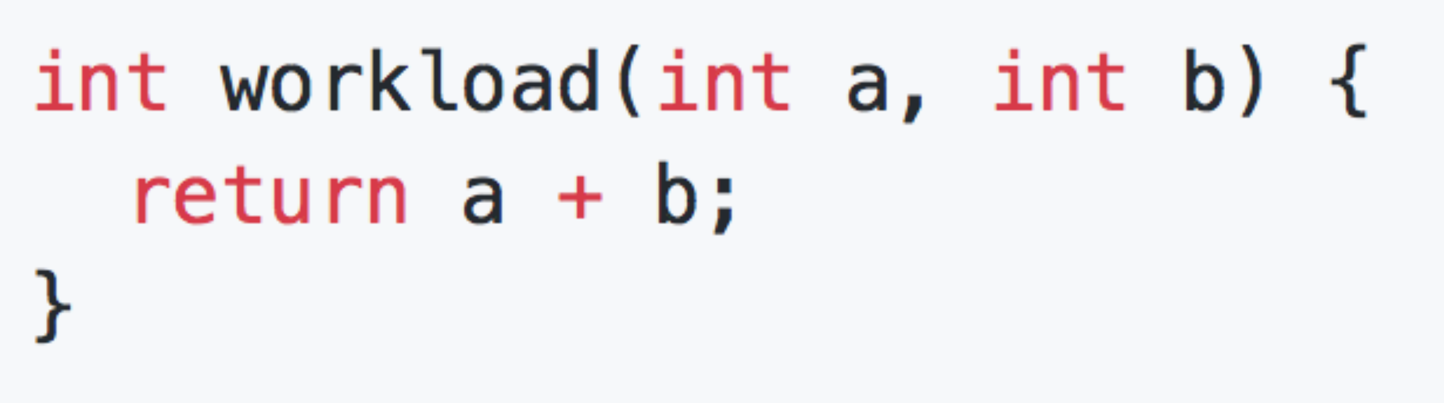
```java
int average(int[] values) {
  int sum = 0;
  for (int n = 0; n < values.length; n++) {
    sum += values[n];
  }
  return sum / values.length;
}
```


IdealGraphVisualizer
Outline
Source Collections
186.8/430.0MB
25:Demo.average(int[])
0: After parsing
1: Before phase org.graalvm.compi
2: After high tier
3: After mid tier
4: After low tier
0: After parsing
Search in Nodes
0 Start
5 End
1 P(0)
6 LoopBegin
3 C(0)
10 ArrayLength
8 Phi(3, 21)
20 C(1)
11 <
21 +
17 If
12 Begin
18 LoadIndexed
22 LoopEnd
19 +
7 Phi(3, 19)
14 LoopExit
15 Proxy(7)
23 ArrayLength
24 /
25 Return
INS

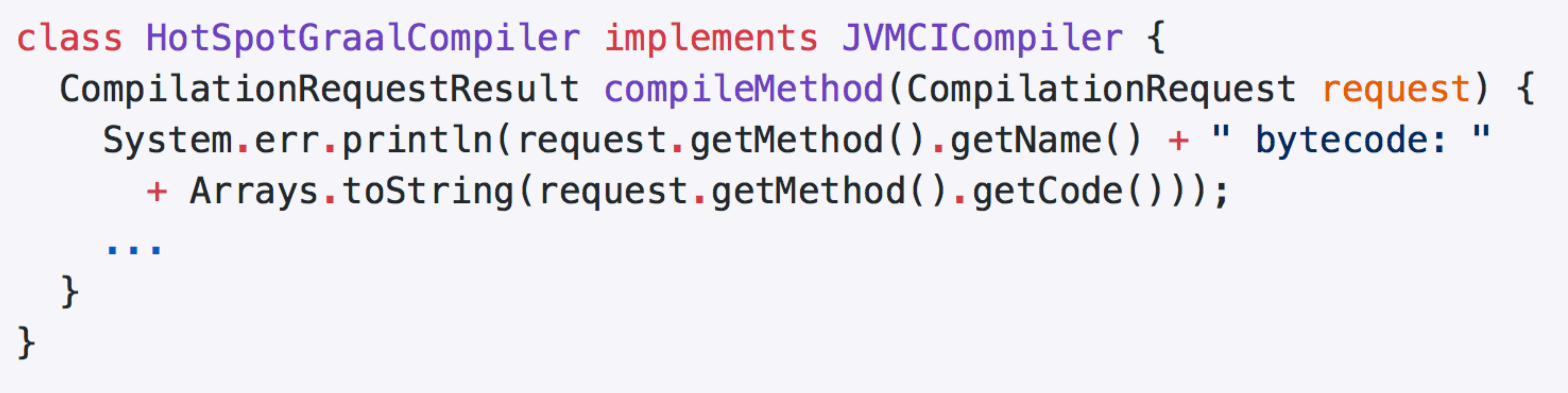

0 Start
5 End
6 LoopBegin
1 P(0)
10 ArrayLength
3 C(0)
8 Phi(3, 21)
11 <
20 C(1)
21 +
17 If
12 Begin
1 P(0)
18 LoadIndexed
22 LoopEnd
19 +
3 C(0)
7 Phi(3, 19)
14 LoopExit
15 Proxy(7)
1 P(0)
23 ArrayLength
24 /
25 Return

From bytecode to machine code

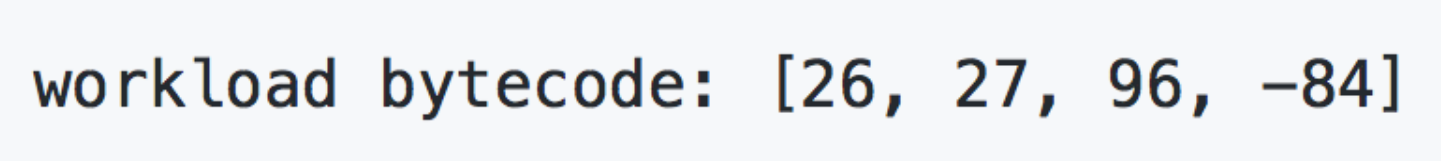
Bytecode in…



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

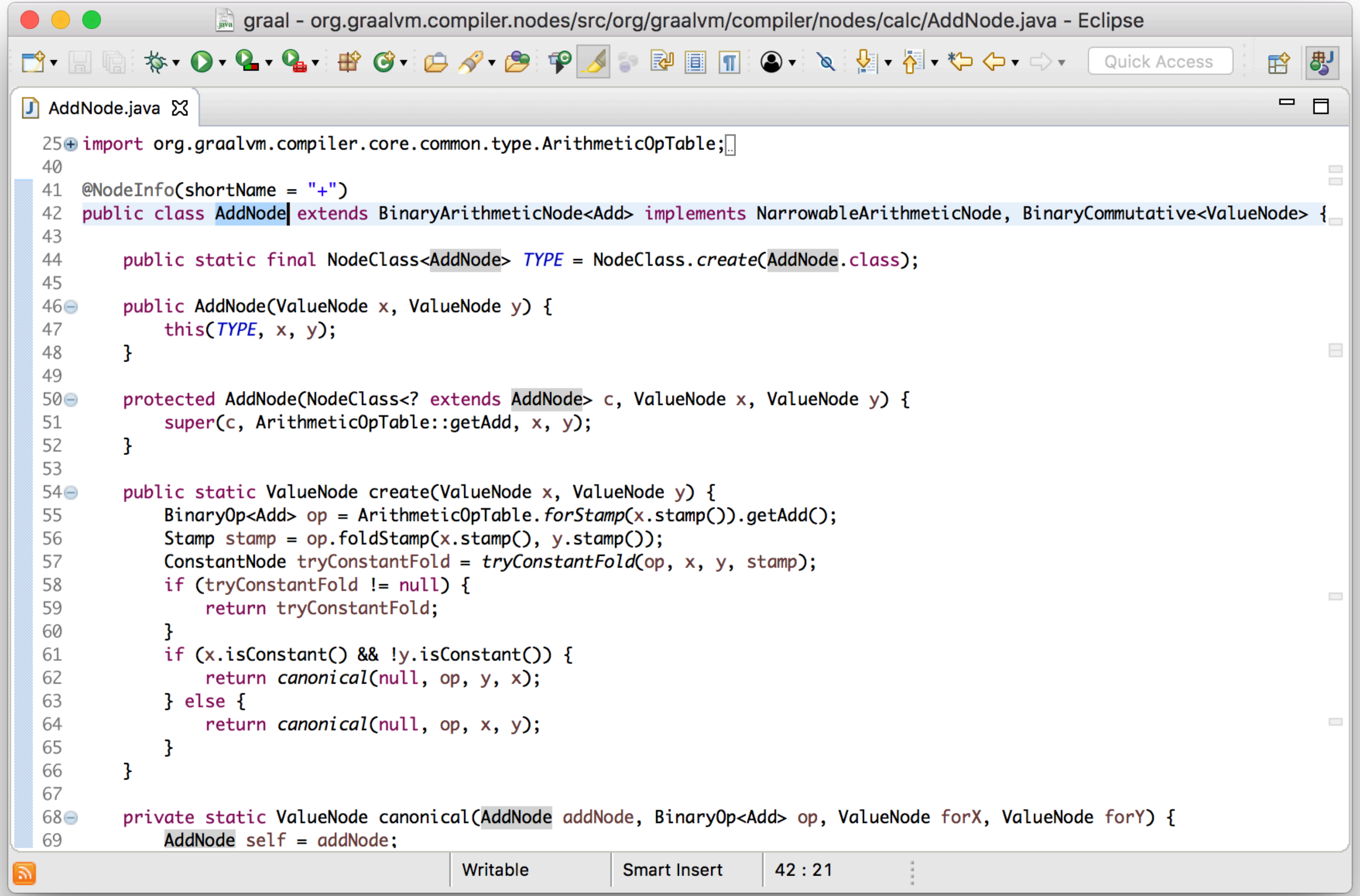


```java
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationRequestResult compileMethod(CompilationRequest request) {
    System.err.println(request.getMethod().getName() + " bytecode: "
      + Arrays.toString(request.getMethod().getCode()));
    ...
  }
}
```



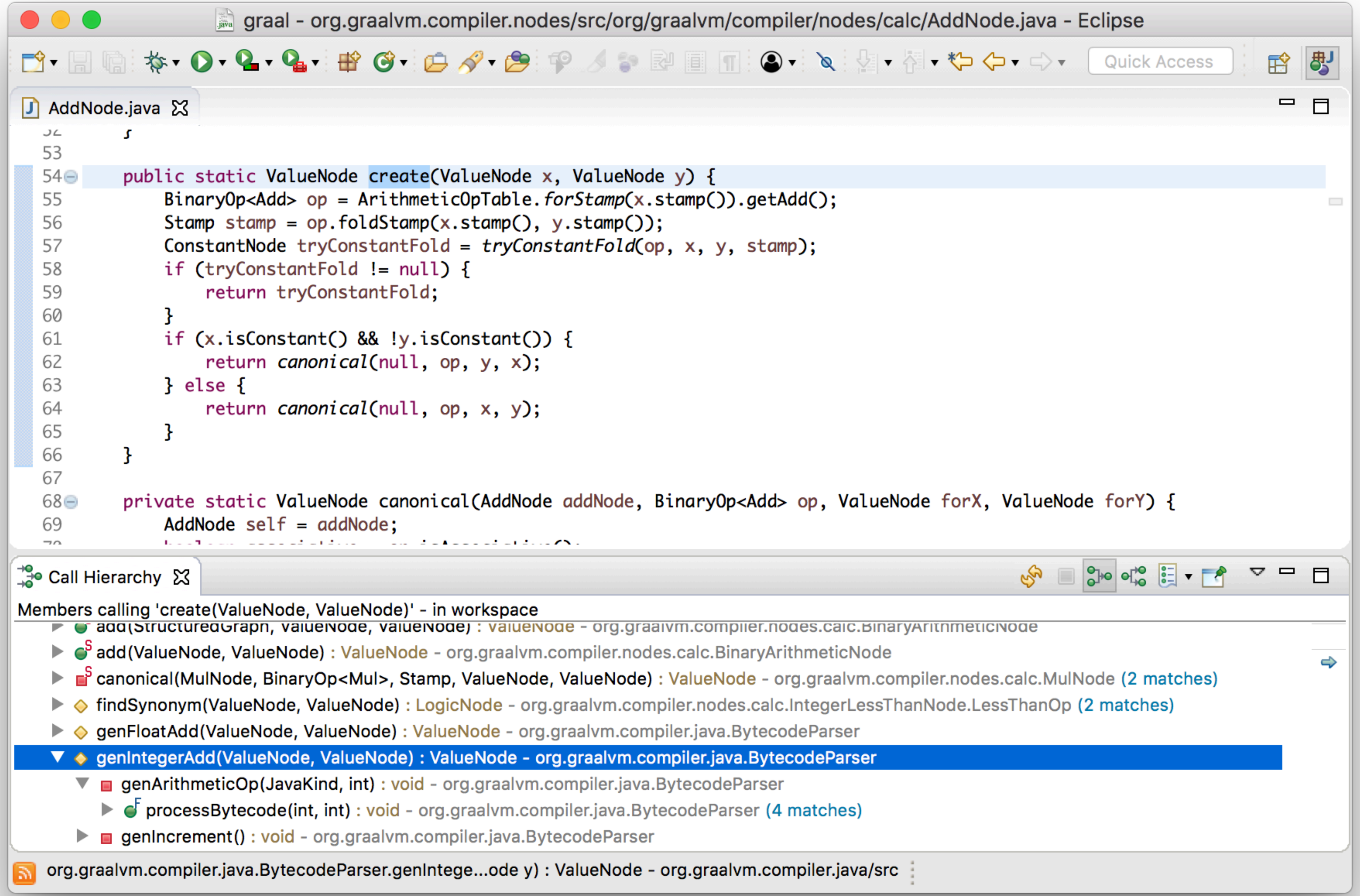
```
workload bytecode: [26, 27, 96, -84]
```

The bytecode parser…



graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/AddNode.java - Eclipse
Quick Access
AddNode.java
Writable
Smart Insert
42 : 21

```java
import org.graalvm.compiler.core.common.type.ArithmeticOpTable;

@NodeInfo(shortName = "+")
public class AddNode extends BinaryArithmeticNode<Add> implements NarrowableArithmeticNode, BinaryCommutative<ValueNode> {

    public static final NodeClass<AddNode> TYPE = NodeClass.create(AddNode.class);

    public AddNode(ValueNode x, ValueNode y) {
        this(TYPE, x, y);
    }

    protected AddNode(NodeClass<? extends AddNode> c, ValueNode x, ValueNode y) {
        super(c, ArithmeticOpTable::getAdd, x, y);
    }

    public static ValueNode create(ValueNode x, ValueNode y) {
        BinaryOp<Add> op = ArithmeticOpTable.forStamp(x.stamp()).getAdd();
        Stamp stamp = op.foldStamp(x.stamp(), y.stamp());
        ConstantNode tryConstantFold = tryConstantFold(op, x, y, stamp);
        if (tryConstantFold != null) {
            return tryConstantFold;
        }
        if (x.isConstant() && !y.isConstant()) {
            return canonical(null, op, y, x);
        } else {
            return canonical(null, op, x, y);
        }
    }

    private static ValueNode canonical(AddNode addNode, BinaryOp<Add> op, ValueNode forX, ValueNode forY) {
        AddNode self = addNode;
```

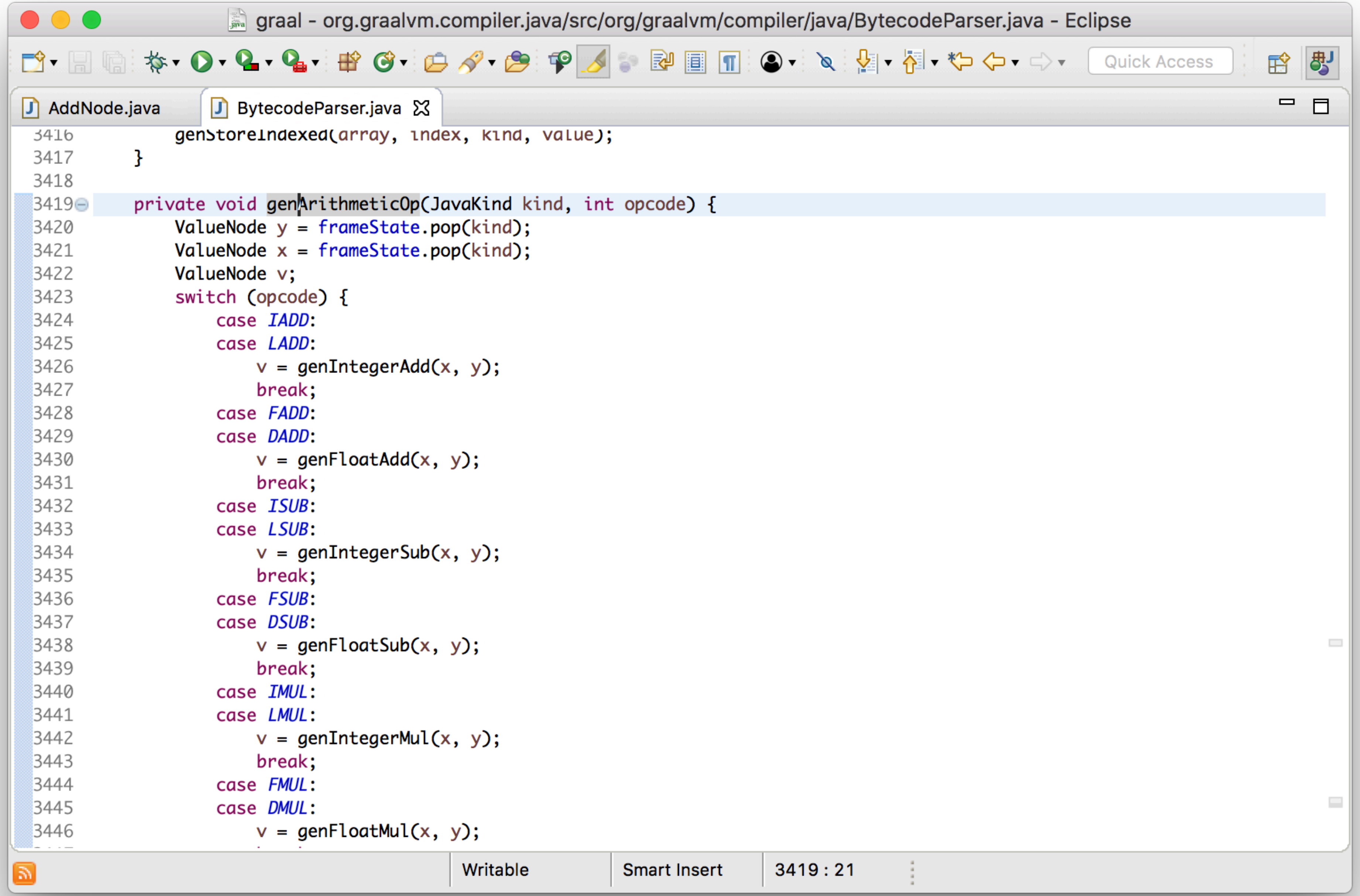

graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/AddNode.java - Eclipse
Quick Access
AddNode.java
Call Hierarchy

```java
    }

    public static ValueNode create(ValueNode x, ValueNode y) {
        BinaryOp<Add> op = ArithmeticOpTable.forStamp(x.stamp()).getAdd();
        Stamp stamp = op.foldStamp(x.stamp(), y.stamp());
        ConstantNode tryConstantFold = tryConstantFold(op, x, y, stamp);
        if (tryConstantFold != null) {
            return tryConstantFold;
        }
        if (x.isConstant() && !y.isConstant()) {
            return canonical(null, op, y, x);
        } else {
            return canonical(null, op, x, y);
        }
    }

    private static ValueNode canonical(AddNode addNode, BinaryOp<Add> op, ValueNode forX, ValueNode forY) {
        AddNode self = addNode;
        boolean associative = op.isAssociative();
```

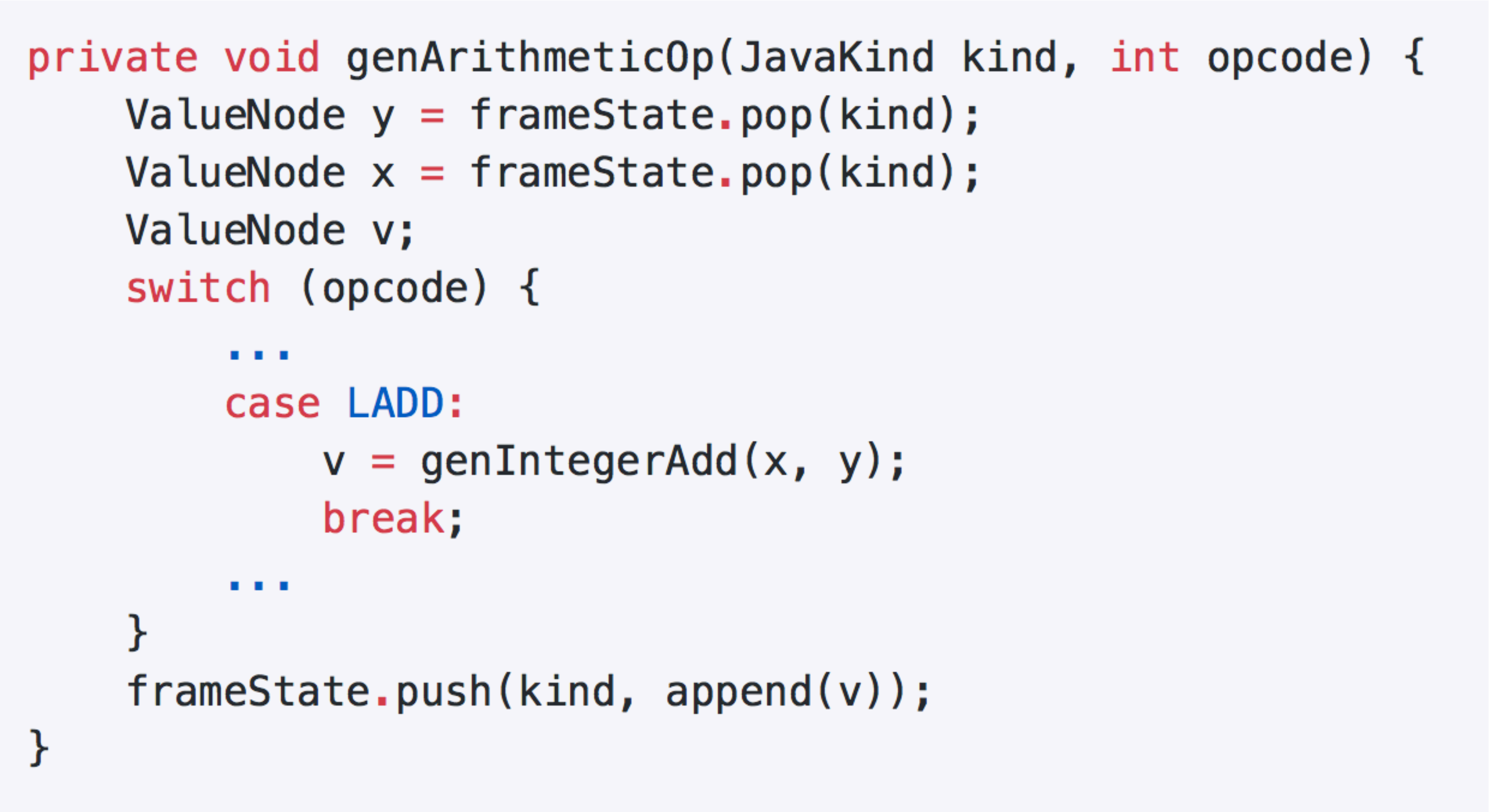
Members calling 'create(ValueNode, ValueNode)' - in workspace

```
▶ add(StructuredGraph, ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.nodes.calc.BinaryArithmeticNode
▶ add(ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.nodes.calc.BinaryArithmeticNode
▶ canonical(MulNode, BinaryOp<Mul>, Stamp, ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.nodes.calc.MulNode (2 matches)
▶ findSynonym(ValueNode, ValueNode) : LogicNode - org.graalvm.compiler.nodes.calc.IntegerLessThanNode.LessThanOp (2 matches)
▶ genFloatAdd(ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.java.BytecodeParser
▼ genIntegerAdd(ValueNode, ValueNode) : ValueNode - org.graalvm.compiler.java.BytecodeParser
    ▼ genArithmeticOp(JavaKind, int) : void - org.graalvm.compiler.java.BytecodeParser
        ▶ processBytecode(int, int) : void - org.graalvm.compiler.java.BytecodeParser (4 matches)
    ▶ genIncrement() : void - org.graalvm.compiler.java.BytecodeParser
```

org.graalvm.compiler.java.BytecodeParser.genIntege...ode y) : ValueNode - org.graalvm.compiler.java/src

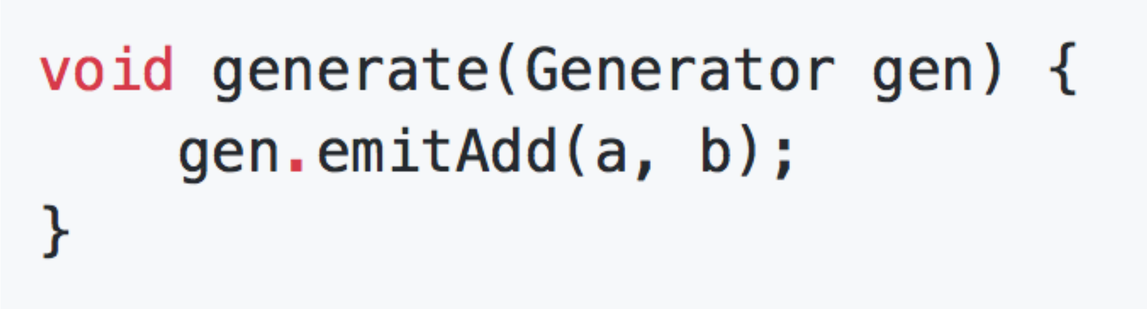

graal - org.graalvm.compiler.java/src/org/graalvm/compiler/java/BytecodeParser.java - Eclipse
Quick Access
AddNode.java
BytecodeParser.java
Writable
Smart Insert
3419 : 21

```java
            genStoreIndexed(array, index, kind, value);
        }

    private void genArithmeticOp(JavaKind kind, int opcode) {
        ValueNode y = frameState.pop(kind);
        ValueNode x = frameState.pop(kind);
        ValueNode v;
        switch (opcode) {
            case IADD:
            case LADD:
                v = genIntegerAdd(x, y);
                break;
            case FADD:
            case DADD:
                v = genFloatAdd(x, y);
                break;
            case ISUB:
            case LSUB:
                v = genIntegerSub(x, y);
                break;
            case FSUB:
            case DSUB:
                v = genFloatSub(x, y);
                break;
            case IMUL:
            case LMUL:
                v = genIntegerMul(x, y);
                break;
            case FMUL:
            case DMUL:
                v = genFloatMul(x, y);
```

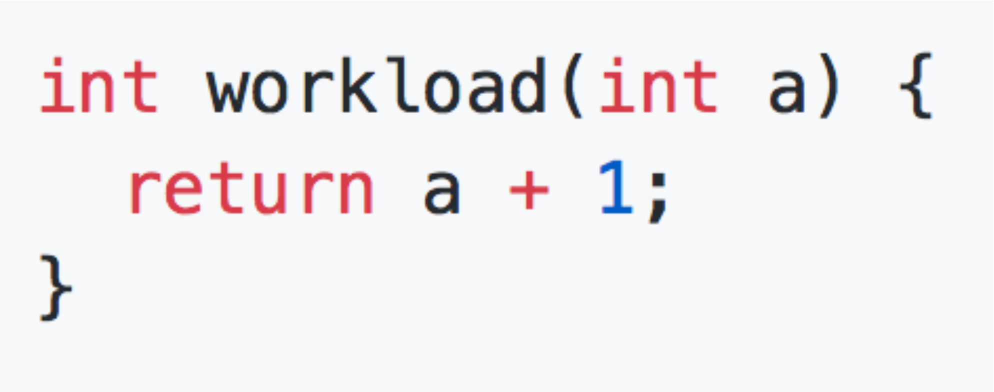


```java
private void genArithmeticOp(JavaKind kind, int opcode) {
    ValueNode y = frameState.pop(kind);
    ValueNode x = frameState.pop(kind);
    ValueNode v;
    switch (opcode) {
        ...
        case LADD:
            v = genIntegerAdd(x, y);
            break;
        ...
    }
    frameState.push(kind,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

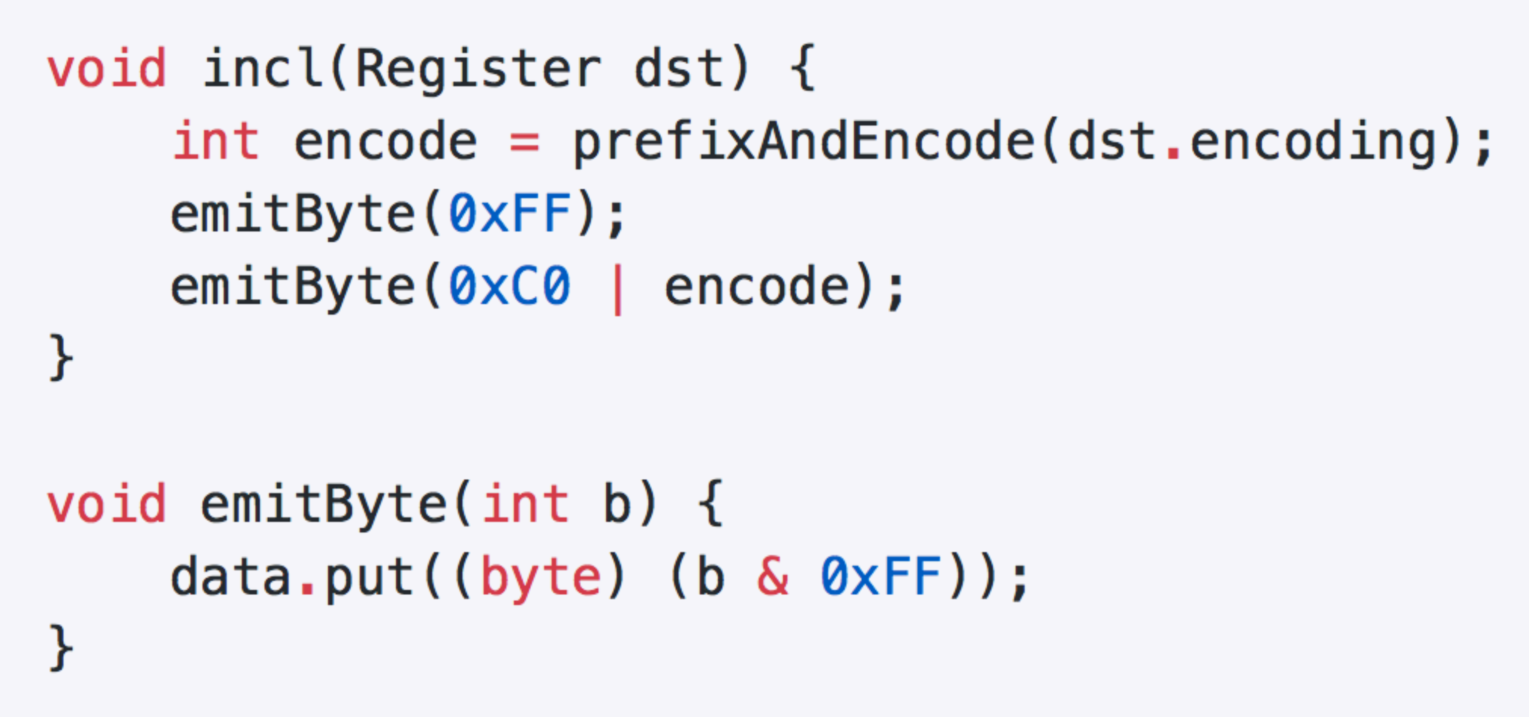
Emitting assembly…



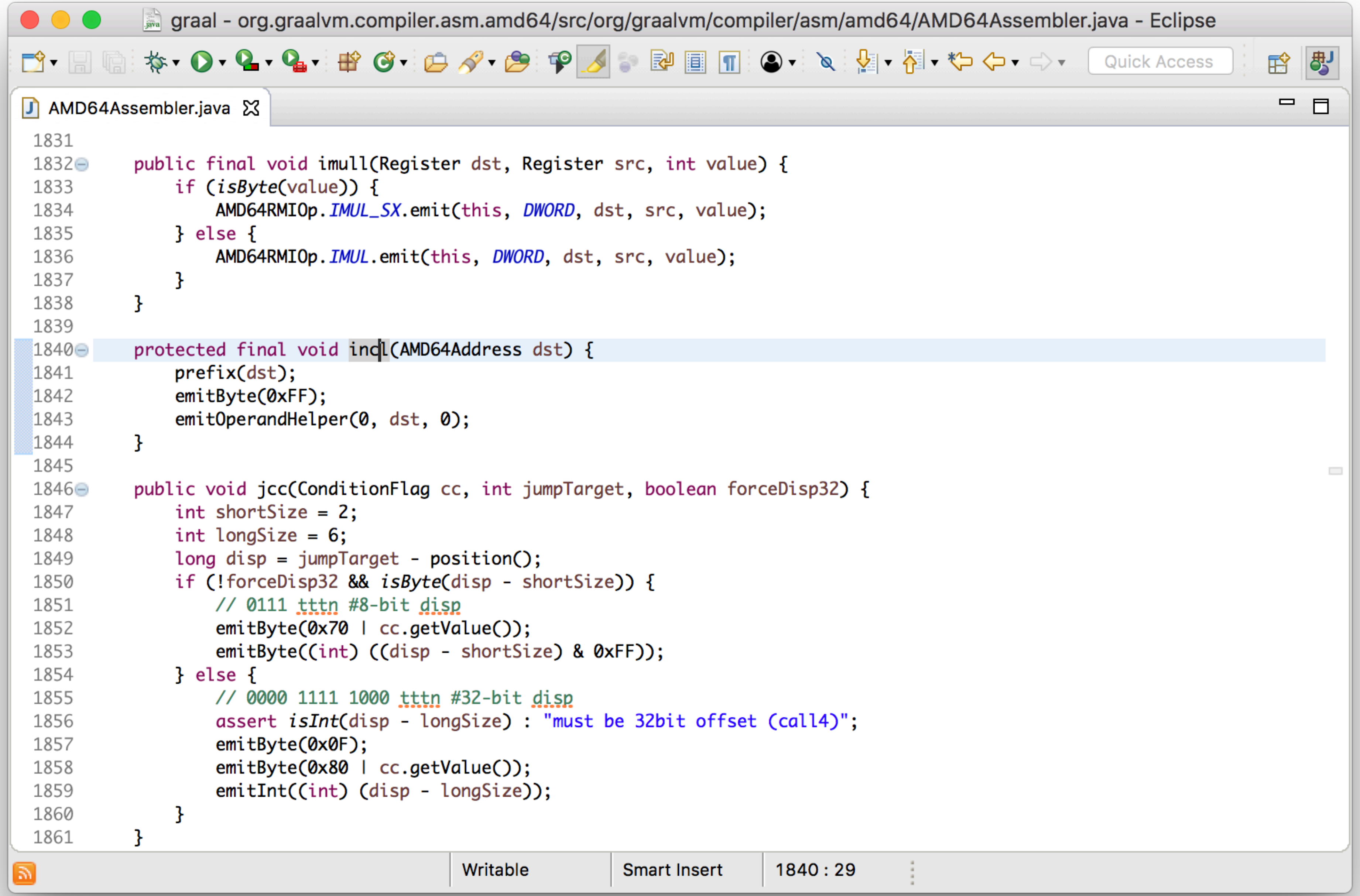
```
void generate(Generator gen) {
    gen.emitAdd(a, b);
}
```



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

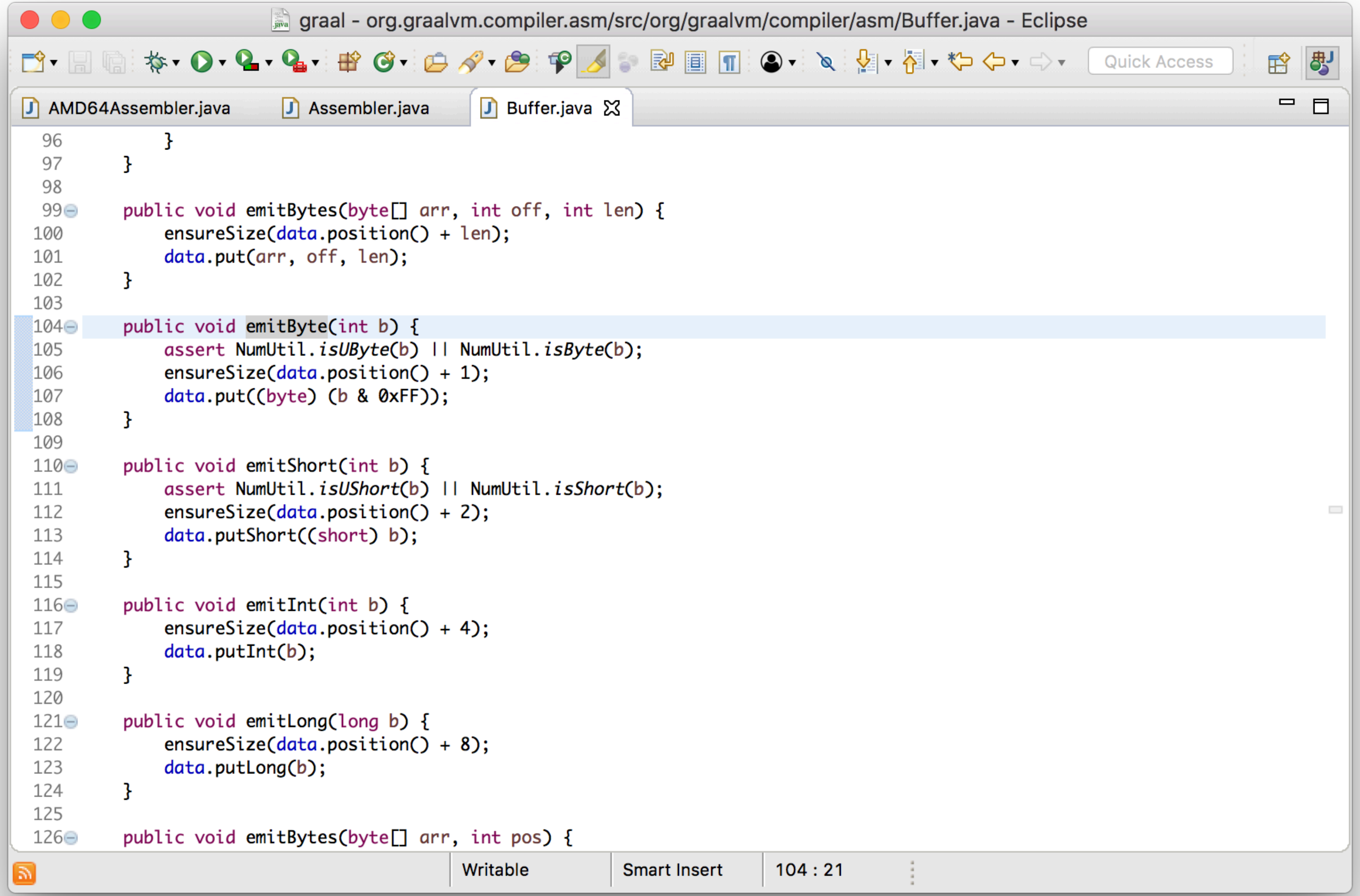


```
void incl(Register dst) {
    int encode = prefixAndEncode(dst.encoding);
    emitByte(0xFF);
    emitByte(0xC0 | encode);
}

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



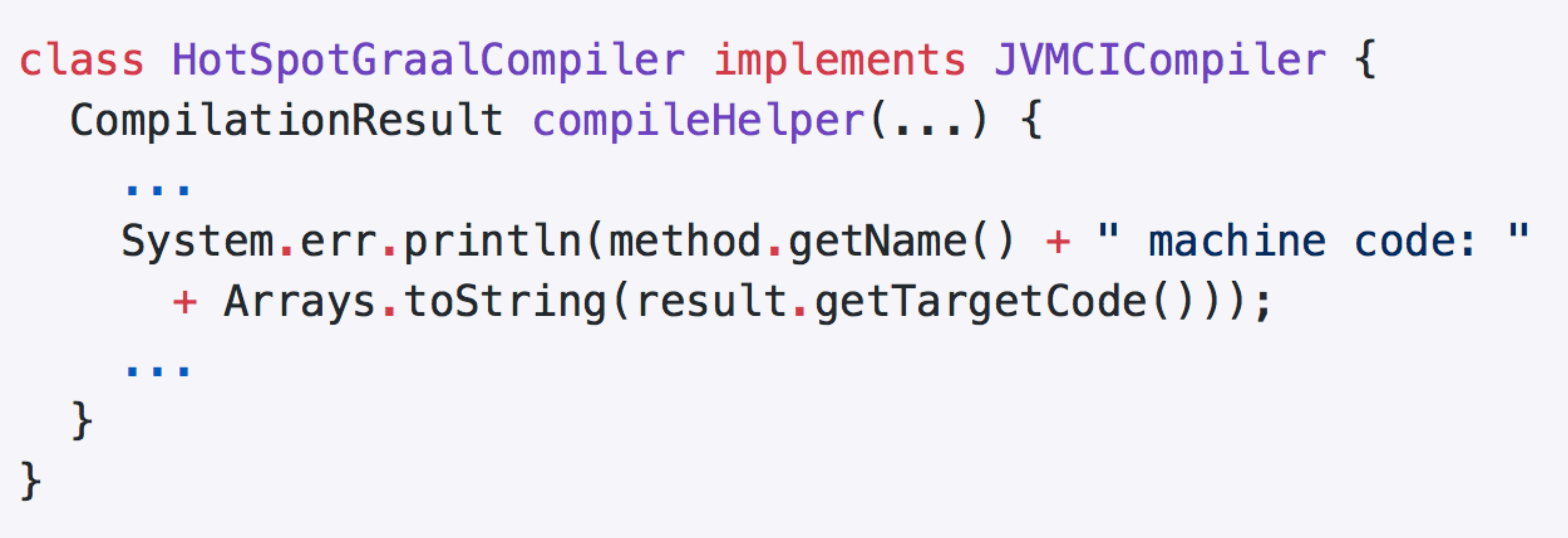
graal - org.graalvm.compiler.asm.amd64/src/org/graalvm/compiler/asm/amd64/AMD64Assembler.java - Eclipse
Quick Access
AMD64Assembler.java
Writable
Smart Insert
1840 : 29

```java

    public final void imull(Register dst, Register src, int value) {
        if (isByte(value)) {
            AMD64RMIOp.IMUL_SX.emit(this, DWORD, dst, src, value);
        } else {
            AMD64RMIOp.IMUL.emit(this, DWORD, dst, src, value);
        }
    }

    protected final void incl(AMD64Address dst) {
        prefix(dst);
        emitByte(0xFF);
        emitOperandHelper(0, dst, 0);
    }

    public void jcc(ConditionFlag cc, int jumpTarget, boolean forceDisp32) {
        int shortSize = 2;
        int longSize = 6;
        long disp = jumpTarget - position();
        if (!forceDisp32 && isByte(disp - shortSize)) {
            // 0111 tttn #8-bit disp
            emitByte(0x70 | cc.getValue());
            emitByte((int) ((disp - shortSize) & 0xFF));
        } else {
            // 0000 1111 1000 tttn #32-bit disp
            assert isInt(disp - longSize) : "must be 32bit offset (call4)";
            emitByte(0x0F);
            emitByte(0x80 | cc.getValue());
            emitInt((int) (disp - longSize));
        }
    }
```

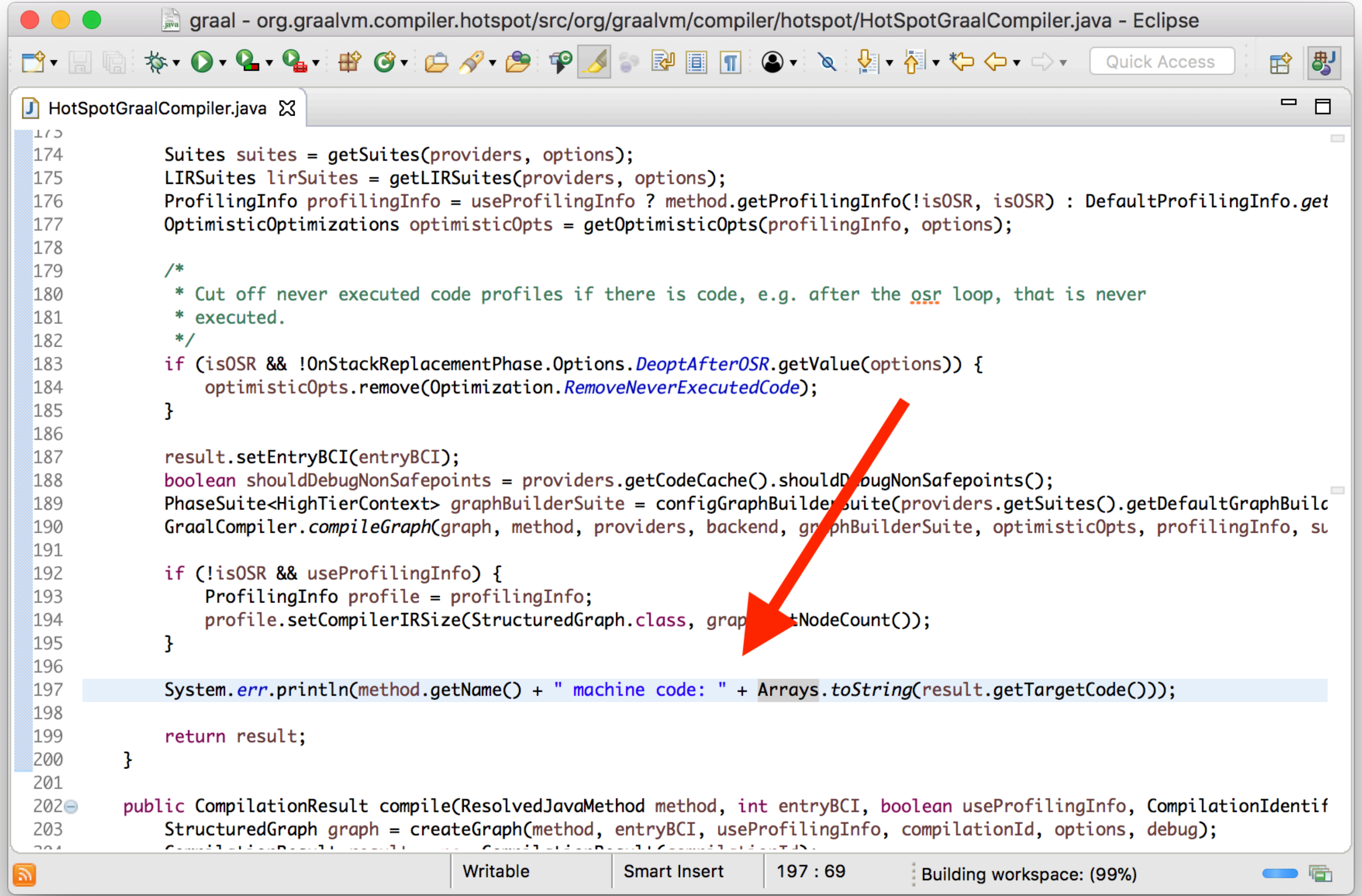


```
        }
    }

    public void emitBytes(byte[] arr, int off, int len) {
        ensureSize(data.position() + len);
        data.put(arr, off, len);
    }

    public void emitByte(int b) {
        assert NumUtil.isUByte(b) || NumUtil.isByte(b);
        ensureSize(data.position() + 1);
        data.put((byte) (b & 0xFF));
    }

    public void emitShort(int b) {
        assert NumUtil.isUShort(b) || NumUtil.isShort(b);
        ensureSize(data.position() + 2);
        data.putShort((short) b);
    }

    public void emitInt(int b) {
        ensureSize(data.position() + 4);
        data.putInt(b);
    }

    public void emitLong(long b) {
        ensureSize(data.position() + 8);
        data.putLong(b);
    }

    public void emitBytes(byte[] arr, int pos) {
```

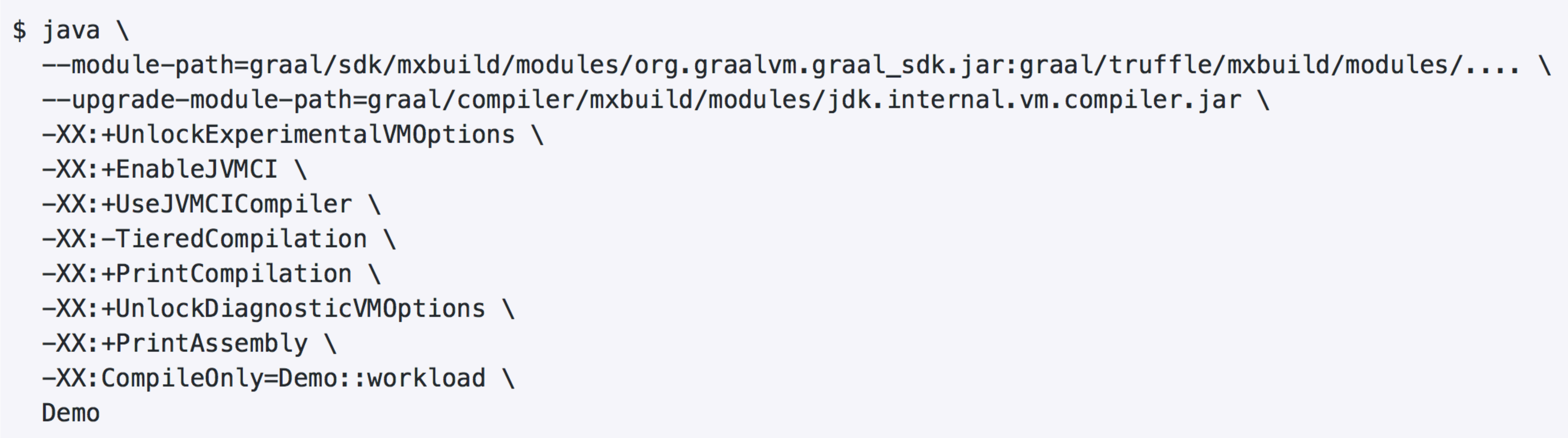
Machine code out…



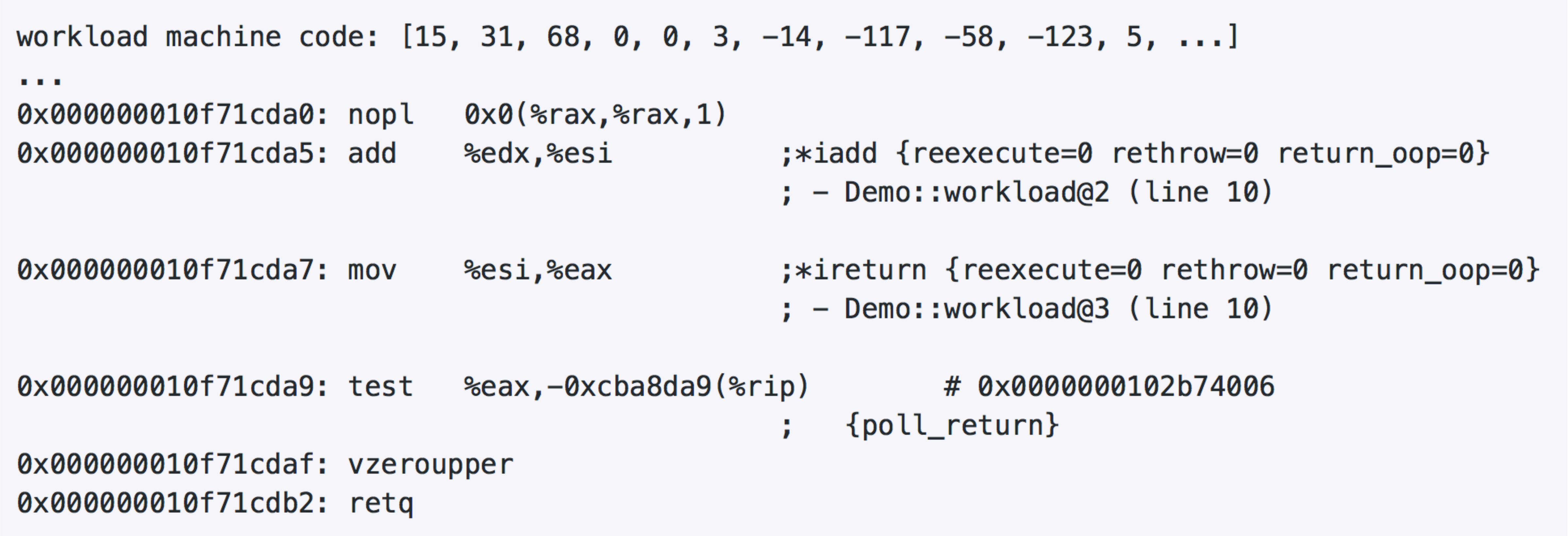
```
class HotSpotGraalCompiler implements JVMCICompiler {
  CompilationResult compileHelper(...) {
    ...
    System.err.println(method.getName() + " machine code: "
      + Arrays.toString(result.getTargetCode()));
    ...
  }
}
```


graal - org.graalvm.compiler.hotspot/src/org/graalvm/compiler/hotspot/HotSpotGraalCompiler.java - Eclipse
Quick Access
HotSpotGraalCompiler.java
Writable
Smart Insert
197 : 69
Building workspace: (99%)

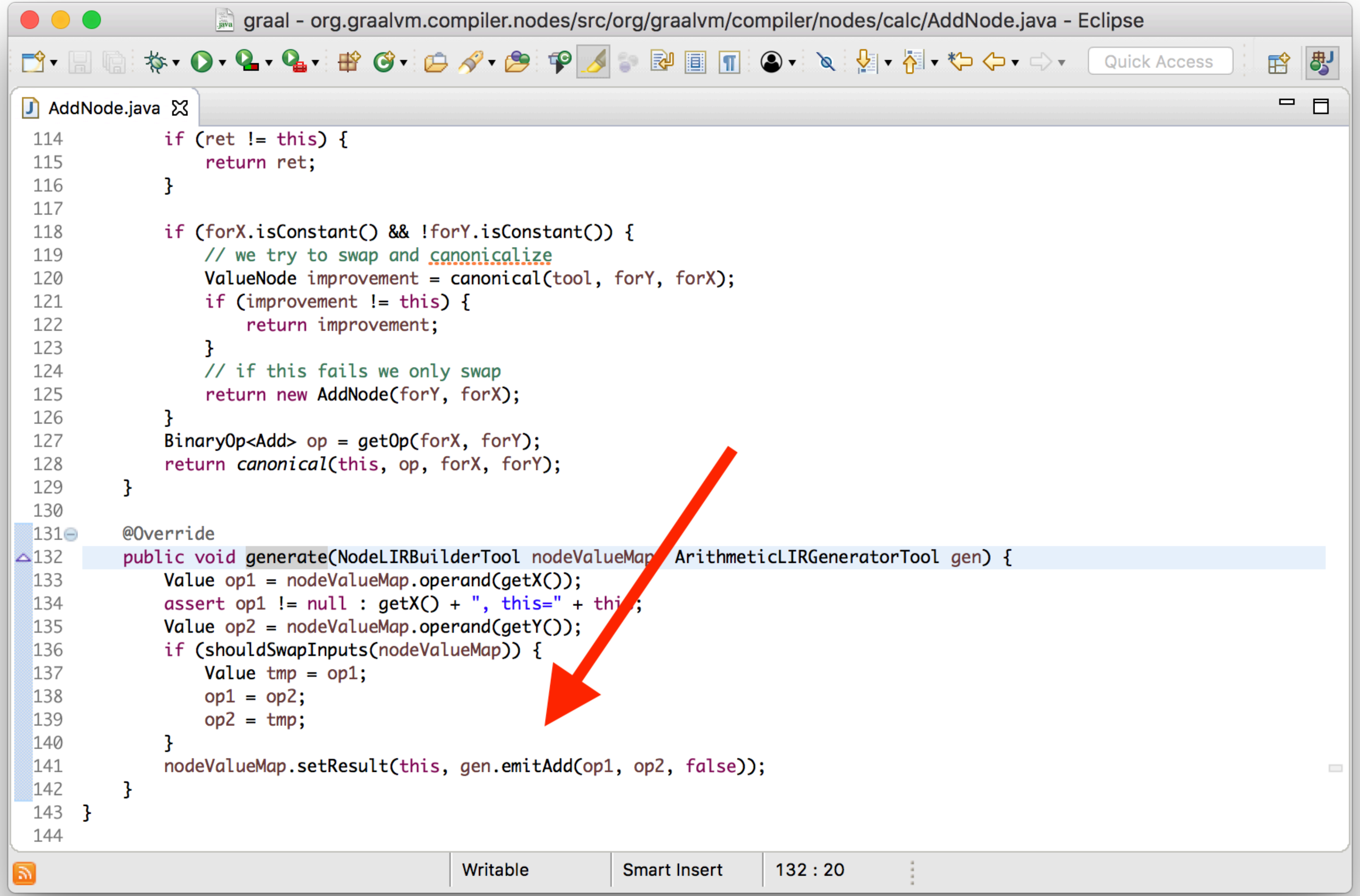
```java
        Suites suites = getSuites(providers, options);
        LIRSuites lirSuites = getLIRSuites(providers, options);
        ProfilingInfo profilingInfo = useProfilingInfo ? method.getProfilingInfo(!isOSR, isOSR) : DefaultProfilingInfo.get
        OptimisticOptimizations optimisticOpts = getOptimisticOpts(profilingInfo, options);

        /*
         * Cut off never executed code profiles if there is code, e.g. after the osr loop, that is never
         * executed.
         */
        if (isOSR && !OnStackReplacementPhase.Options.DeoptAfterOSR.getValue(options)) {
            optimisticOpts.remove(Optimization.RemoveNeverExecutedCode);
        }

        result.setEntryBCI(entryBCI);
        boolean shouldDebugNonSafepoints = providers.getCodeCache().shouldDebugNonSafepoints();
        PhaseSuite<HighTierContext> graphBuilderSuite = configGraphBuilderSuite(providers.getSuites().getDefaultGraphBuilc
        GraalCompiler.compileGraph(graph, method, providers, backend, graphBuilderSuite, optimisticOpts, profilingInfo, su

        if (!isOSR && useProfilingInfo) {
            ProfilingInfo profile = profilingInfo;
            profile.setCompilerIRSize(StructuredGraph.class, graph.getNodeCount());
        }

        System.err.println(method.getName() + " machine code: " + Arrays.toString(result.getTargetCode()));

        return result;
    }

    public CompilationResult compile(ResolvedJavaMethod method, int entryBCI, boolean useProfilingInfo, CompilationIdentif
        StructuredGraph graph = createGraph(method, entryBCI, useProfilingInfo, compilationId, options, debug);
```



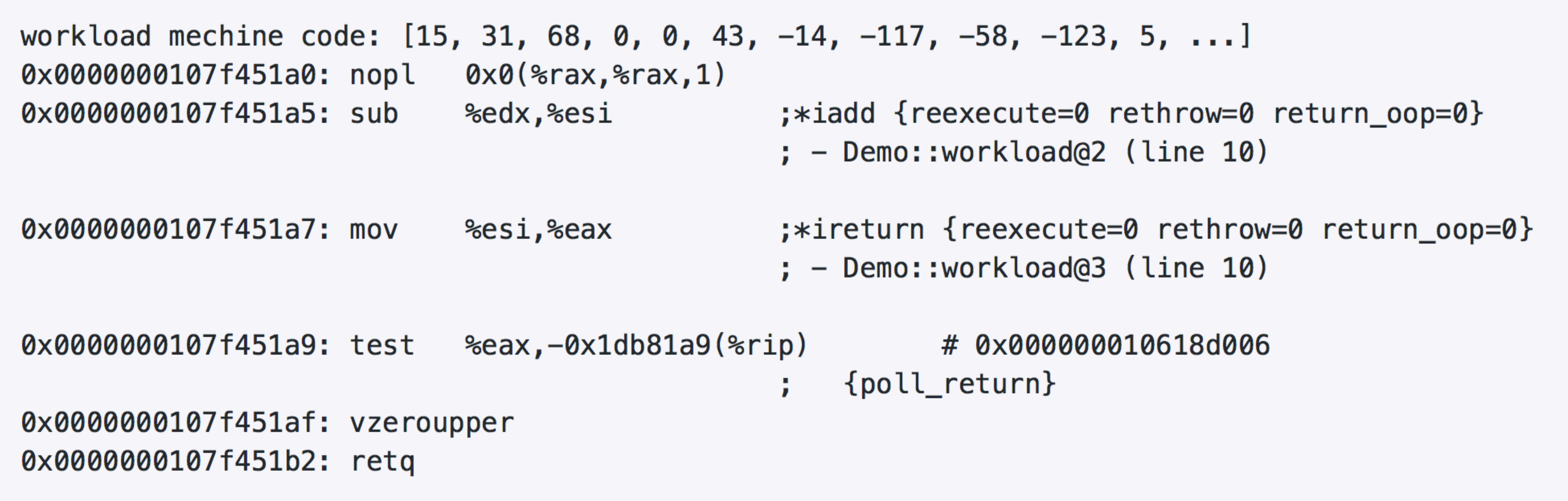
```
$ java \
  --module-path=graal/sdk/mxbuild/modules/org.graalvm.graal_sdk.jar:graal/truffle/mxbuild/modules/.... \
  --upgrade-module-path=graal/compiler/mxbuild/modules/jdk.internal.vm.compiler.jar \
  -XX:+UnlockExperimentalVMOptions \
  -XX:+EnableJVMCI \
  -XX:+UseJVMCICompiler \
  -XX:-TieredCompilation \
  -XX:+PrintCompilation \
  -XX:+UnlockDiagnosticVMOptions \
  -XX:+PrintAssembly \
  -XX:CompileOnly=Demo::workload \
  Demo
```



```
workload machine code: [15, 31, 68, 0, 0, 3, -14, -117, -58, -123, 5, ...]
...
0x000000010f71cda0: nopl   0x0(%rax,%rax,1)
0x000000010f71cda5: add    %edx,%esi              ;*iadd {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - Demo::workload@2 (line 10)

0x000000010f71cda7: mov    %esi,%eax              ;*ireturn {reexecute=0 rethrow=0 return_oop=0}
                                                  ; - Demo::workload@3 (line 10)

0x000000010f71cda9: test   %eax,-0xcba8da9(%rip)        # 0x000000102b74006
                                                  ;   {poll_return}
0x000000010f71cdaf: vzeroupper
0x000000010f71cdb2: retq
```


graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/AddNode.java - Eclipse
Quick Access
AddNode.java
Writable
Smart Insert
132 : 20

```java
        if (ret != this) {
            return ret;
        }

        if (forX.isConstant() && !forY.isConstant()) {
            // we try to swap and canonicalize
            ValueNode improvement = canonical(tool, forY, forX);
            if (improvement != this) {
                return improvement;
            }
            // if this fails we only swap
            return new AddNode(forY, forX);
        }
        BinaryOp<Add> op = getOp(forX, forY);
        return canonical(this, op, forX, forY);
    }

    @Override
    public void generate(NodeLIRBuilderTool nodeValueMap, ArithmeticLIRGeneratorTool gen) {
        Value op1 = nodeValueMap.operand(getX());
        assert op1 != null : getX() + ", this=" + this;
        Value op2 = nodeValueMap.operand(getY());
        if (shouldSwapInputs(nodeValueMap)) {
            Value tmp = op1;
            op1 = op2;
            op2 = tmp;
        }
        nodeValueMap.setResult(this, gen.emitAdd(op1, op2, false));
    }
}
```

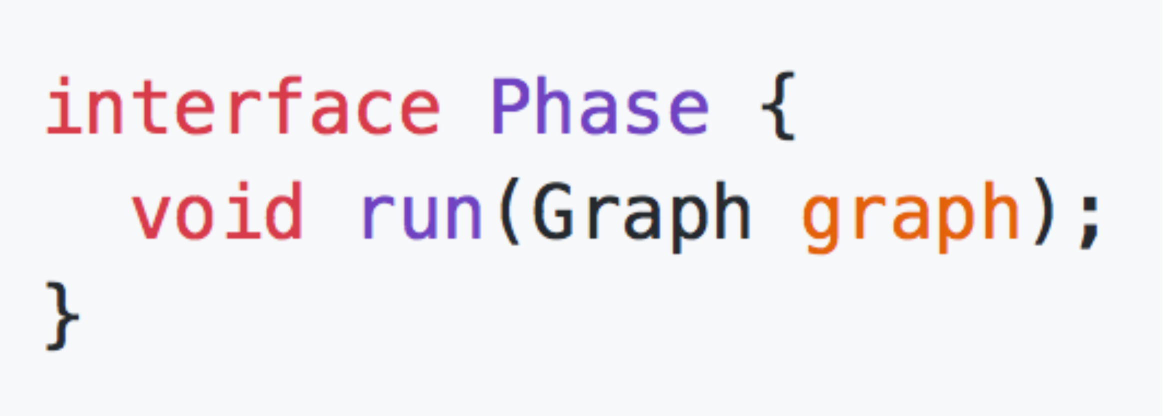


```
workload mechine code: [15, 31, 68, 0, 0, 43, -14, -117, -58, -123, 5, ...]
0x0000000107f451a0: nopl   0x0(%rax,%rax,1)
0x0000000107f451a5: sub    %edx,%esi          ;*iadd {reexecute=0 rethrow=0 return_oop=0}
                                              ; - Demo::workload@2 (line 10)

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
```

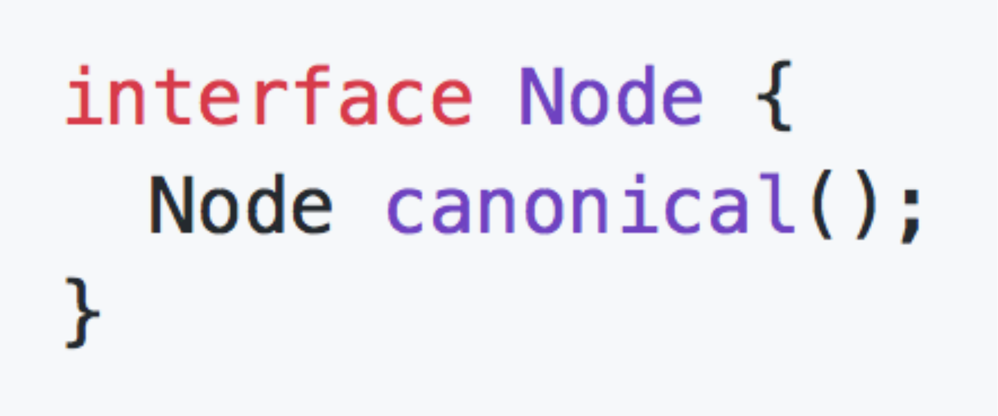
`[26, 27, 96, -84]` → `[15, 31, 68, 0, 0, 43, -14, -117, -58, -123, 5, ...]`

Optimisations

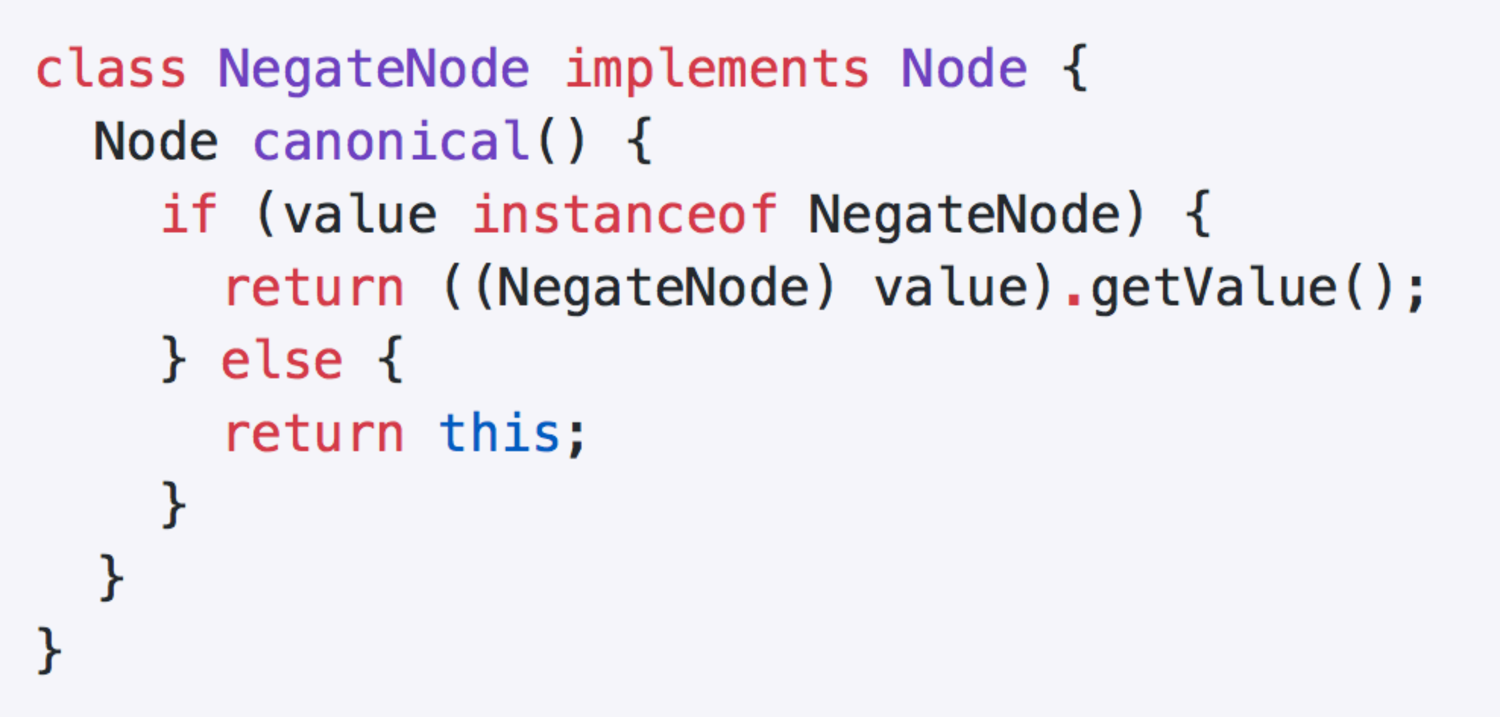
Canonicalisation



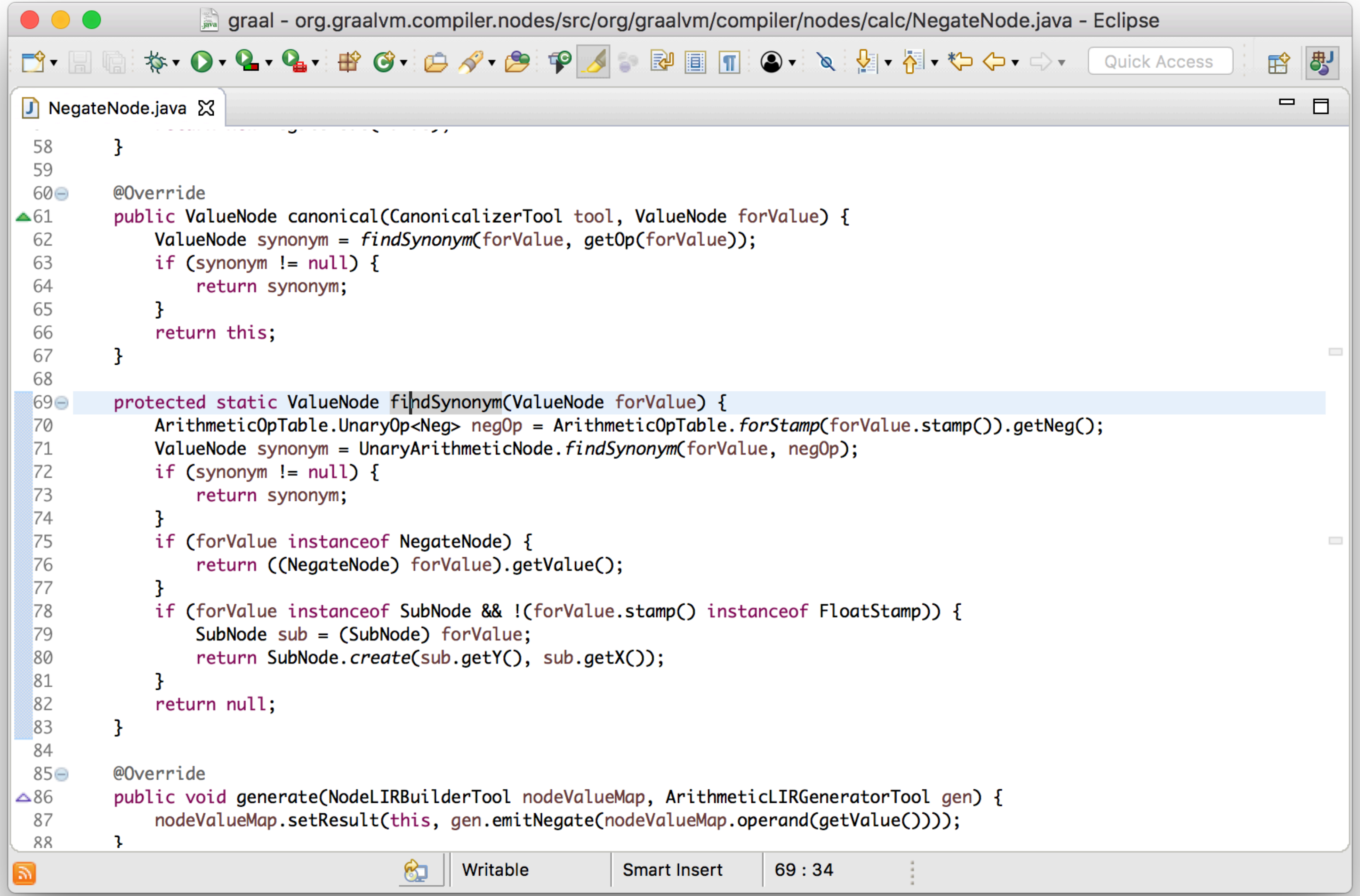
```
interface Phase {
  void run(Graph graph);
}
```



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

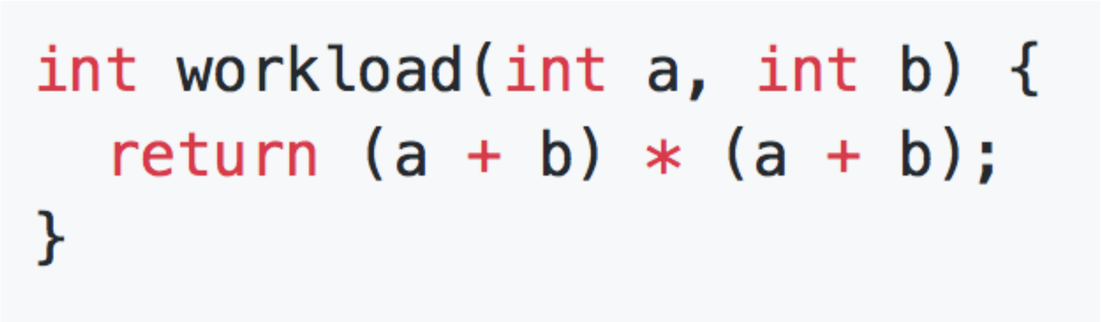


```
class NegateNode implements Node {
  Node canonical() {
    if (value instanceof NegateNode) {
      return ((NegateNode) value).getValue();
    } else {
      return this;
    }
  }
}
```

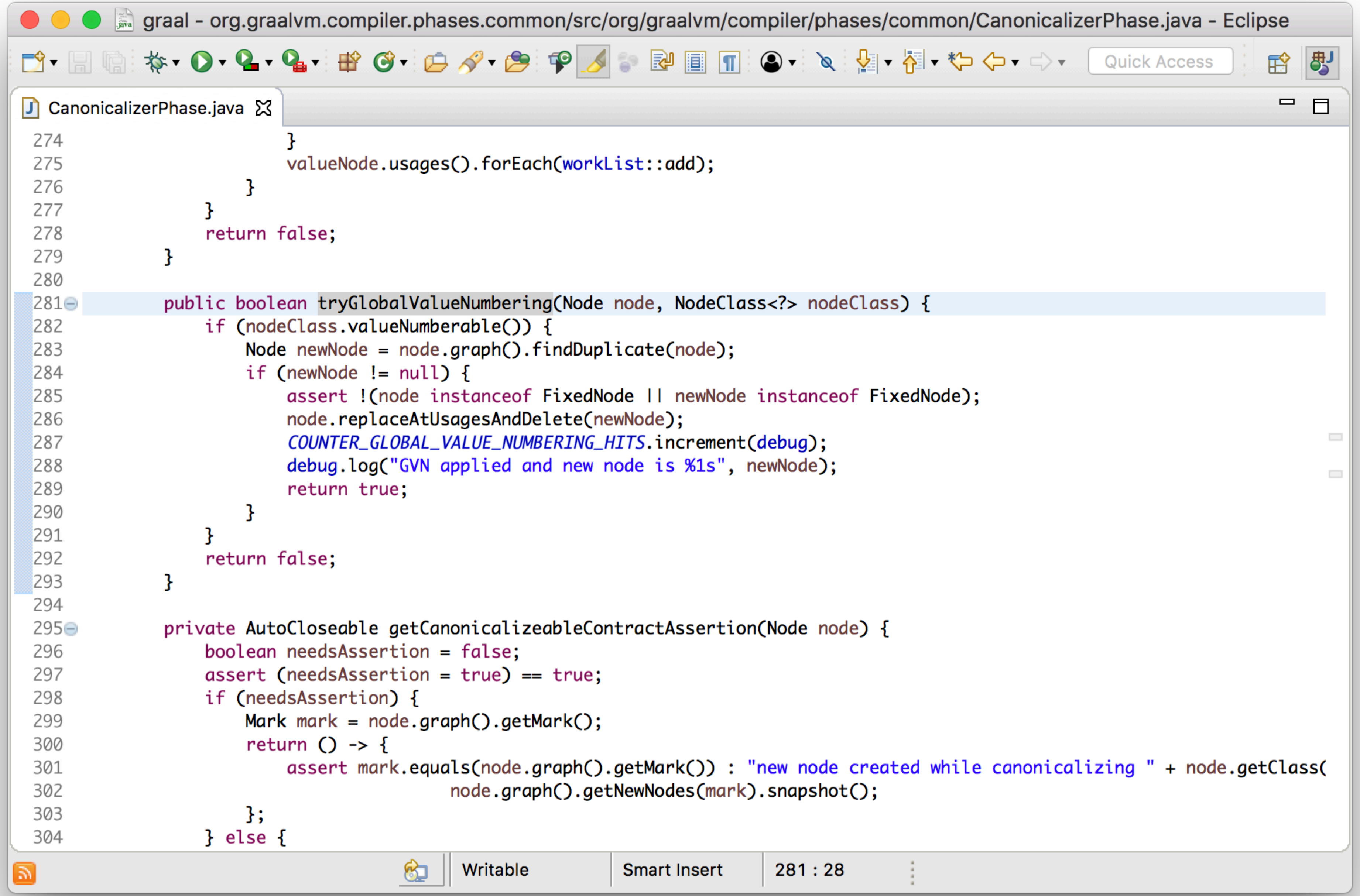

graal - org.graalvm.compiler.nodes/src/org/graalvm/compiler/nodes/calc/NegateNode.java - Eclipse
Quick Access
NegateNode.java
Writable
Smart Insert
69 : 34

```java
    }

    @Override
    public ValueNode canonical(CanonicalizerTool tool, ValueNode forValue) {
        ValueNode synonym = findSynonym(forValue, getOp(forValue));
        if (synonym != null) {
            return synonym;
        }
        return this;
    }

    protected static ValueNode findSynonym(ValueNode forValue) {
        ArithmeticOpTable.UnaryOp<Neg> negOp = ArithmeticOpTable.forStamp(forValue.stamp()).getNeg();
        ValueNode synonym = UnaryArithmeticNode.findSynonym(forValue, negOp);
        if (synonym != null) {
            return synonym;
        }
        if (forValue instanceof NegateNode) {
            return ((NegateNode) forValue).getValue();
        }
        if (forValue instanceof SubNode && !(forValue.stamp() instanceof FloatStamp)) {
            SubNode sub = (SubNode) forValue;
            return SubNode.create(sub.getY(), sub.getX());
        }
        return null;
    }

    @Override
    public void generate(NodeLIRBuilderTool nodeValueMap, ArithmeticLIRGeneratorTool gen) {
        nodeValueMap.setResult(this, gen.emitNegate(nodeValueMap.operand(getValue())));
    }
```

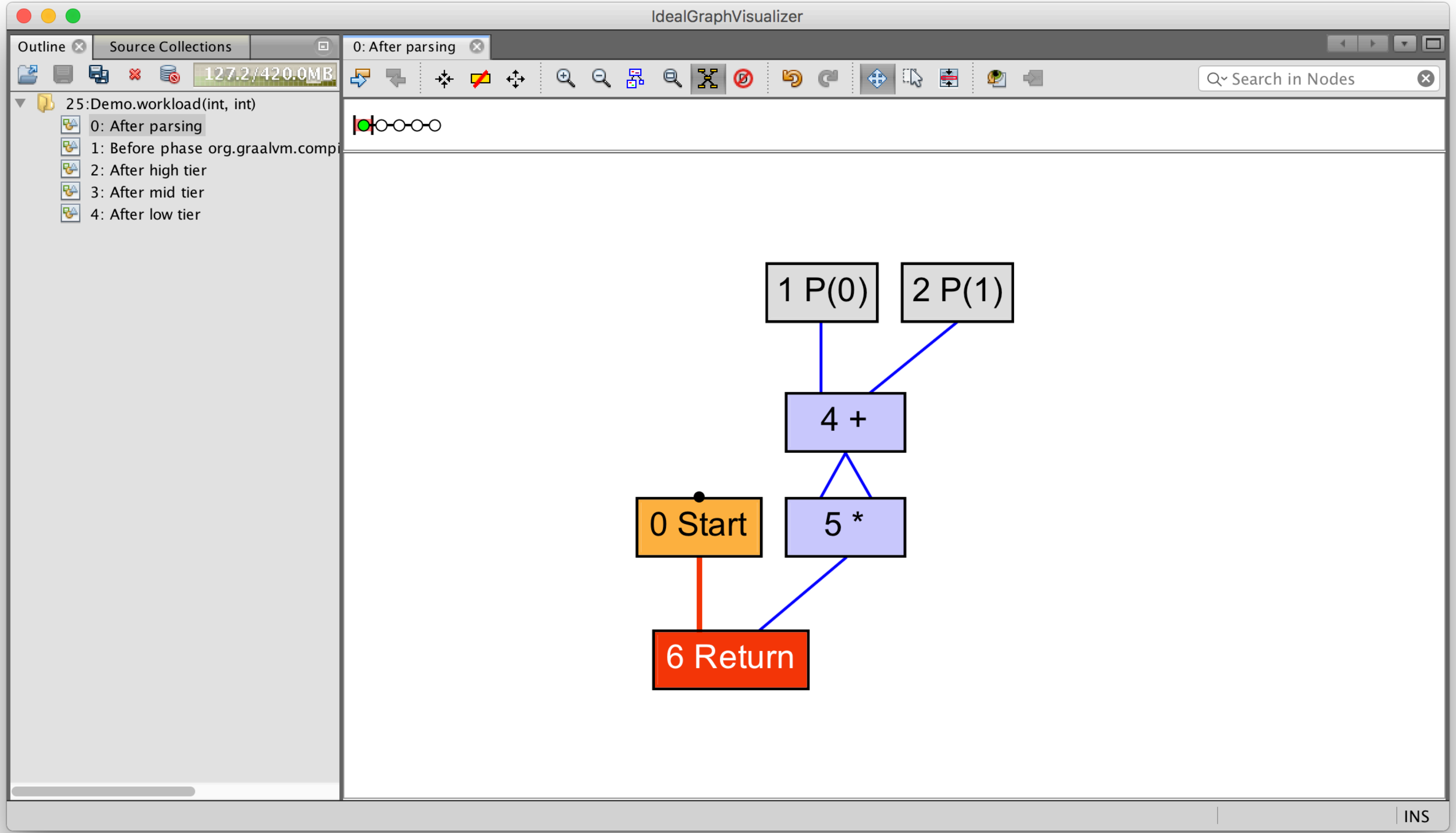
Global value numbering

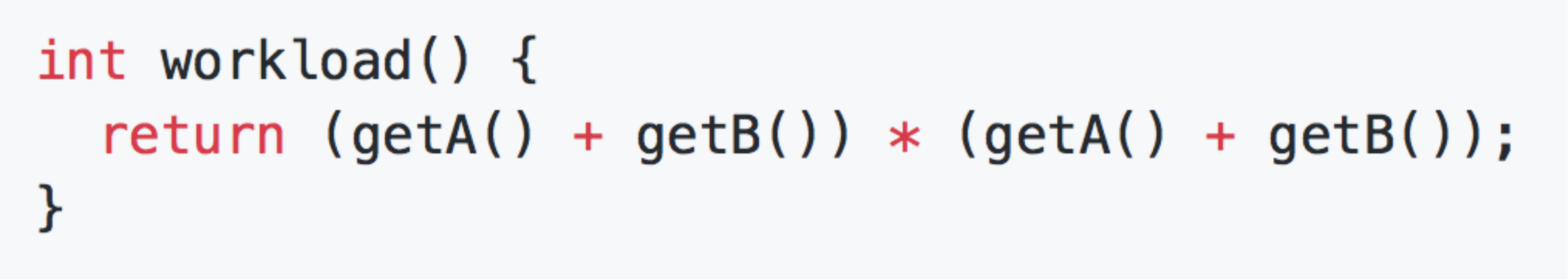


```
int workload(int a, int b) {
  return (a + b) * (a + b);
}
```

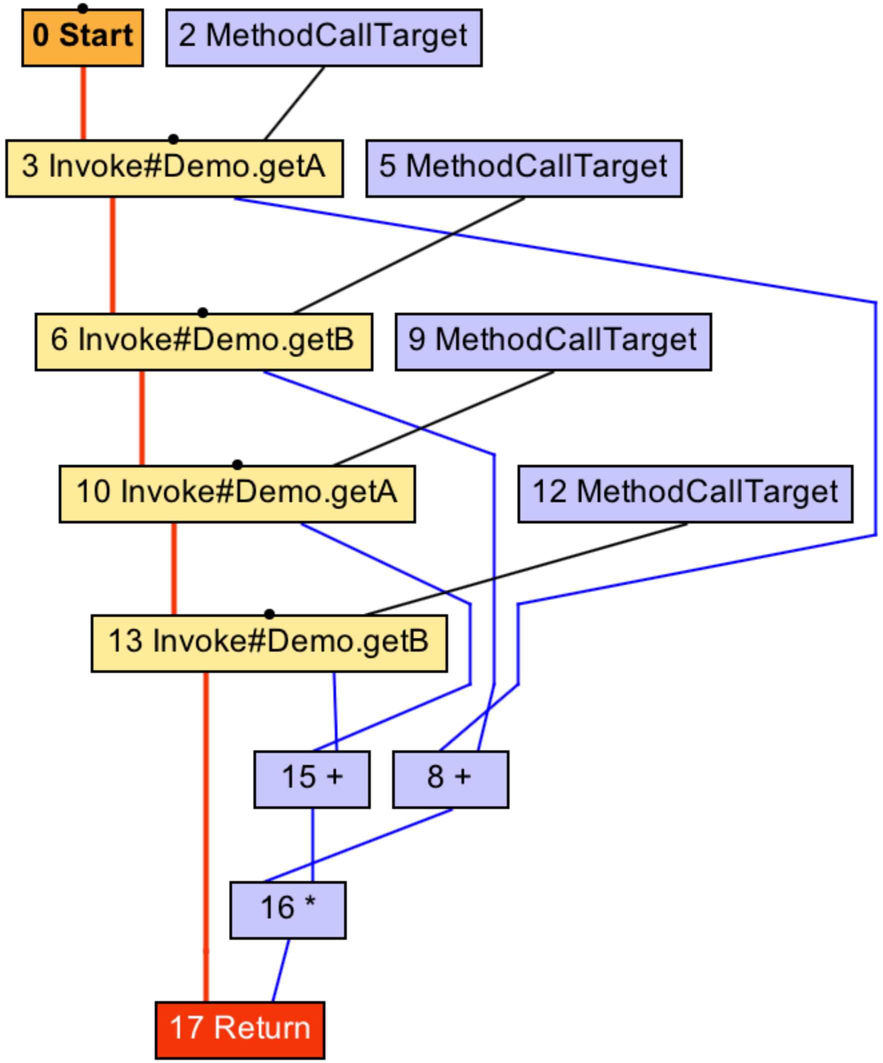

graal - org.graalvm.compiler.phases.common/src/org/graalvm/compiler/phases/common/CanonicalizerPhase.java - Eclipse
Quick Access
CanonicalizerPhase.java
Writable
Smart Insert
281 : 28

```java
                }
                valueNode.usages().forEach(workList::add);
            }
        }
        return false;
    }

    public boolean tryGlobalValueNumbering(Node node, NodeClass<?> nodeClass) {
        if (nodeClass.valueNumberable()) {
            Node newNode = node.graph().findDuplicate(node);
            if (newNode != null) {
                assert !(node instanceof FixedNode || newNode instanceof FixedNode);
                node.replaceAtUsagesAndDelete(newNode);
                COUNTER_GLOBAL_VALUE_NUMBERING_HITS.increment(debug);
                debug.log("GVN applied and new node is %1s", newNode);
                return true;
            }
        }
        return false;
    }

    private AutoCloseable getCanonicalizeableContractAssertion(Node node) {
        boolean needsAssertion = false;
        assert (needsAssertion = true) == true;
        if (needsAssertion) {
            Mark mark = node.graph().getMark();
            return () -> {
                assert mark.equals(node.graph().getMark()) : "new node created while canonicalizing " + node.getClass(
                                node.graph().getNewNodes(mark).snapshot();
            };
        } else {
```

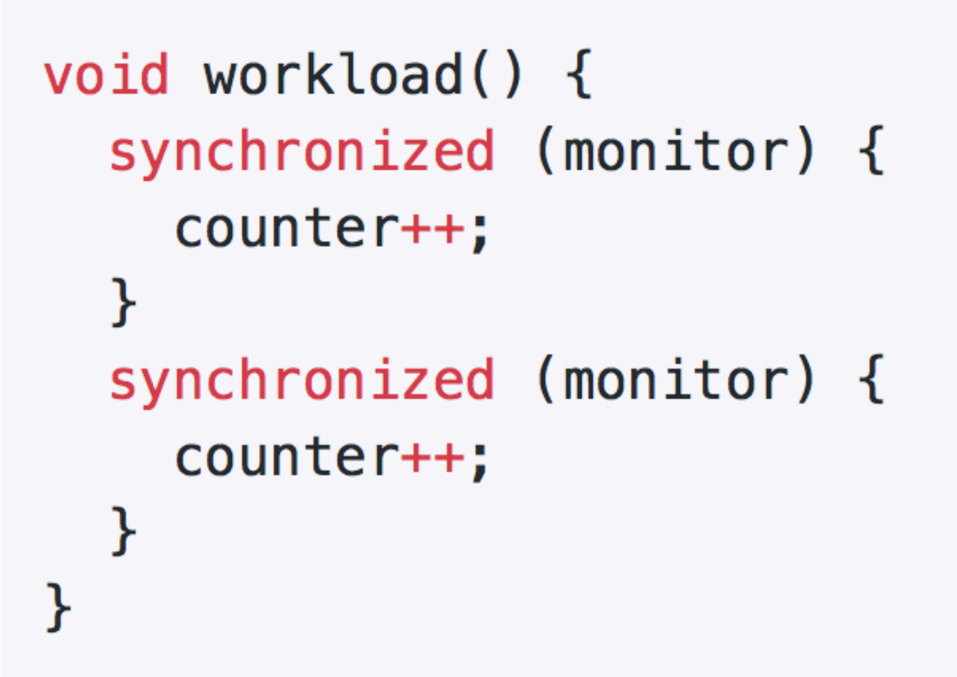

IdealGraphVisualizer
Outline
Source Collections
0: After parsing
127.2/420.0MB
Search in Nodes
25:Demo.workload(int, int)
0: After parsing
1: Before phase org.graalvm.compi
2: After high tier
3: After mid tier
4: After low tier
1 P(0)
2 P(1)
4 +
0 Start
5 *
6 Return
INS



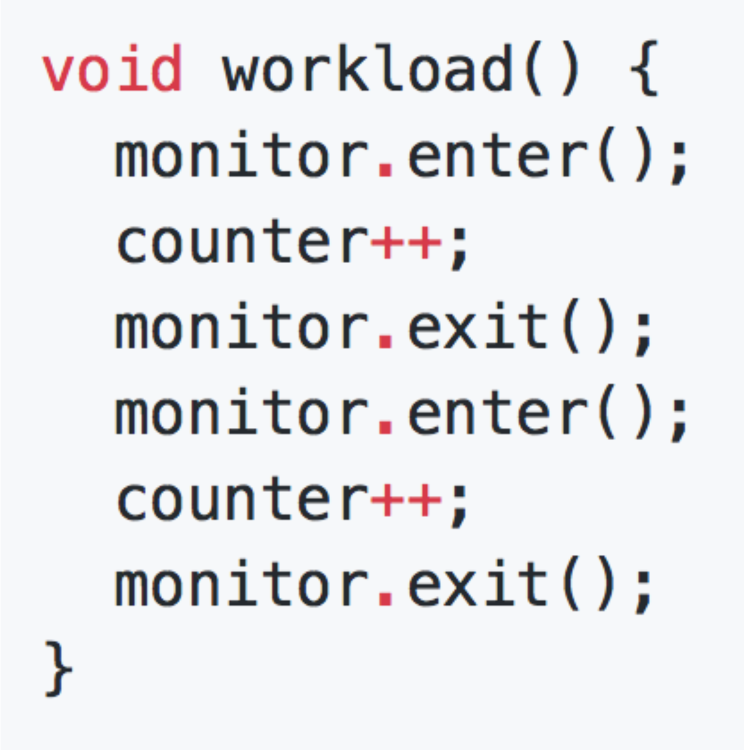
```
int workload() {
  return (getA() + getB()) * (getA() + getB());
}
```


0 Start
2 MethodCallTarget
3 Invoke#Demo.getA
5 MethodCallTarget
6 Invoke#Demo.getB
9 MethodCallTarget
10 Invoke#Demo.getA
12 MethodCallTarget
13 Invoke#Demo.getB
15 +
8 +
16 *
17 Return

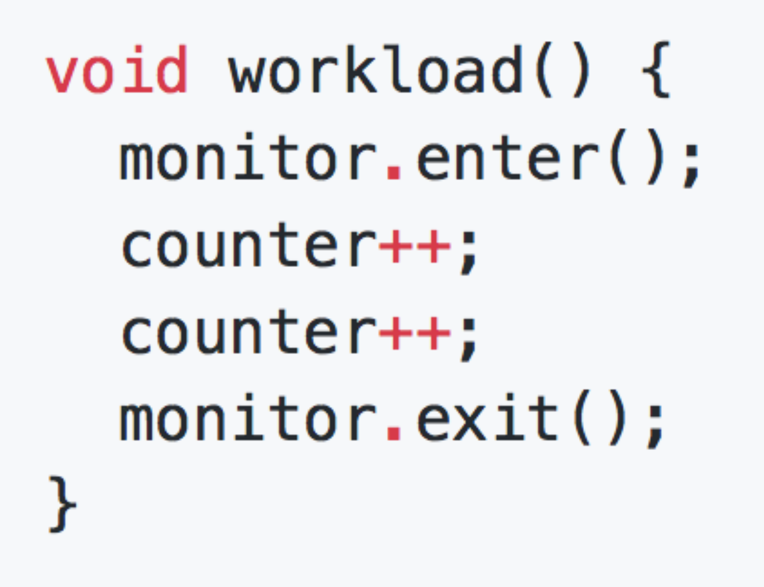
Lock coarsening



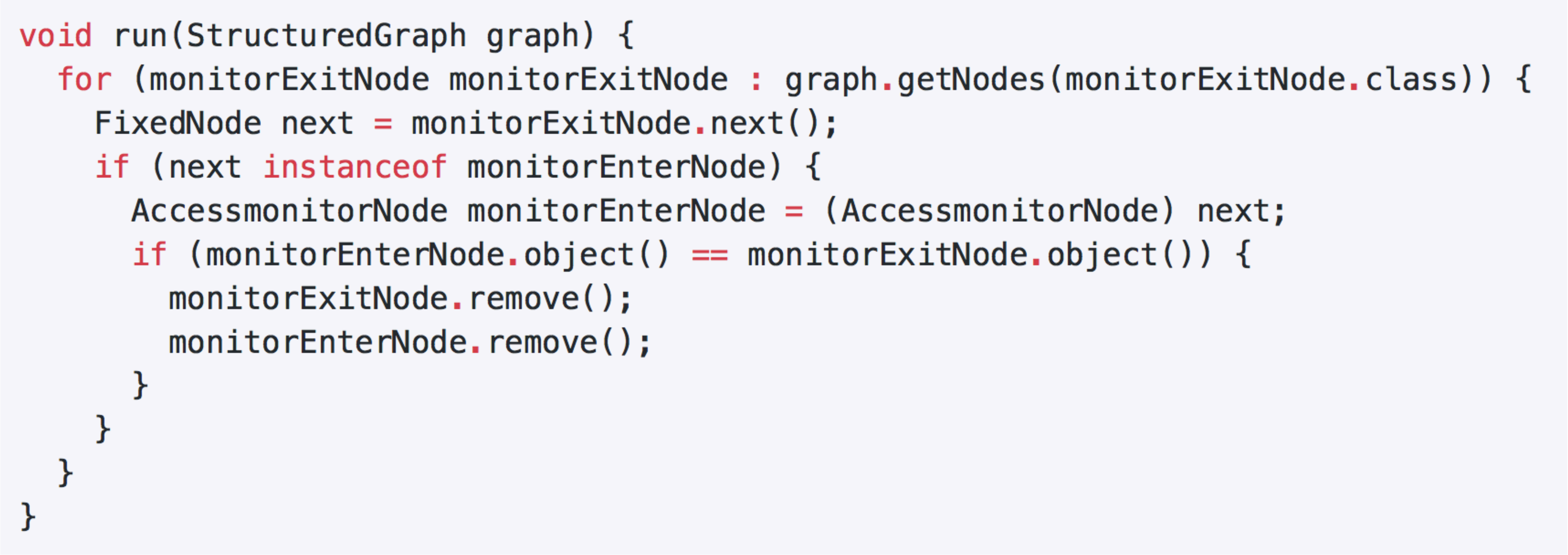
```
void workload() {
  synchronized (monitor) {
    counter++;
  }
  synchronized (monitor) {
    counter++;
  }
}
```



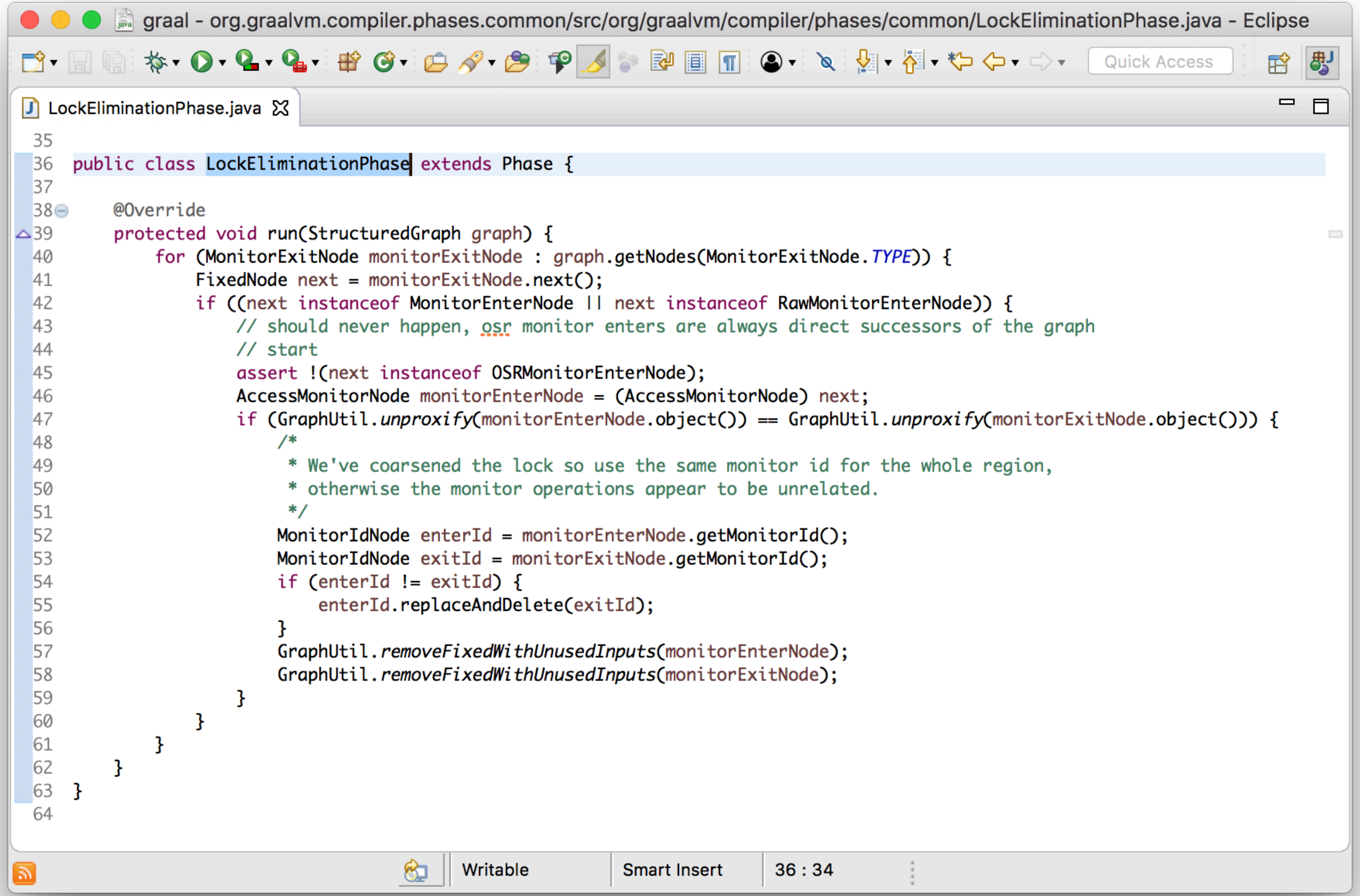
```
void workload() {
  monitor.enter();
  counter++;
  monitor.exit();
  monitor.enter();
  counter++;
  monitor.exit();
}
```



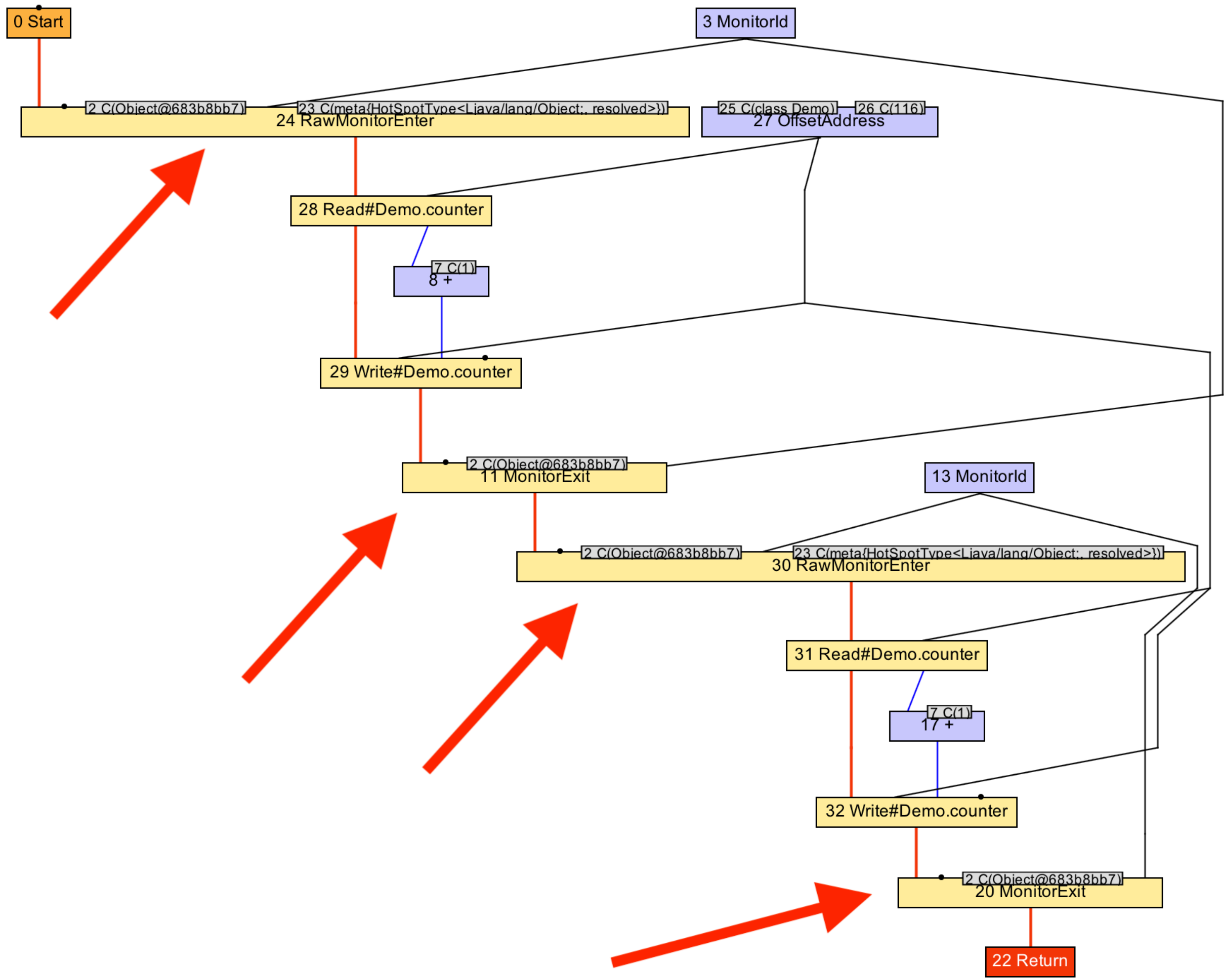
```
void workload() {
  monitor.enter();
  counter++;
  counter++;
  monitor.exit();
}
```

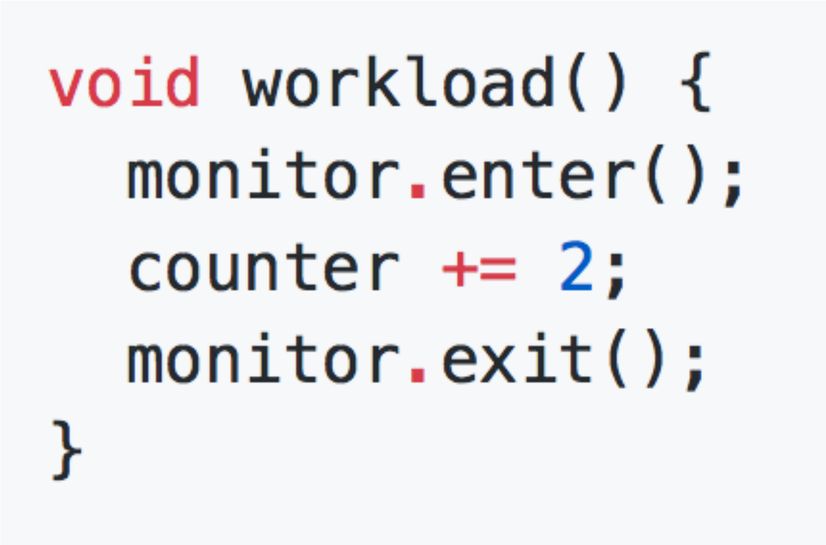


```java
void run(StructuredGraph graph) {
  for (monitorExitNode monitorExitNode : graph.getNodes(monitorExitNode.class)) {
    FixedNode next = monitorExitNode.next();
    if (next instanceof monitorEnterNode) {
      AccessmonitorNode monitorEnterNode = (AccessmonitorNode) next;
      if (monitorEnterNode.object()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

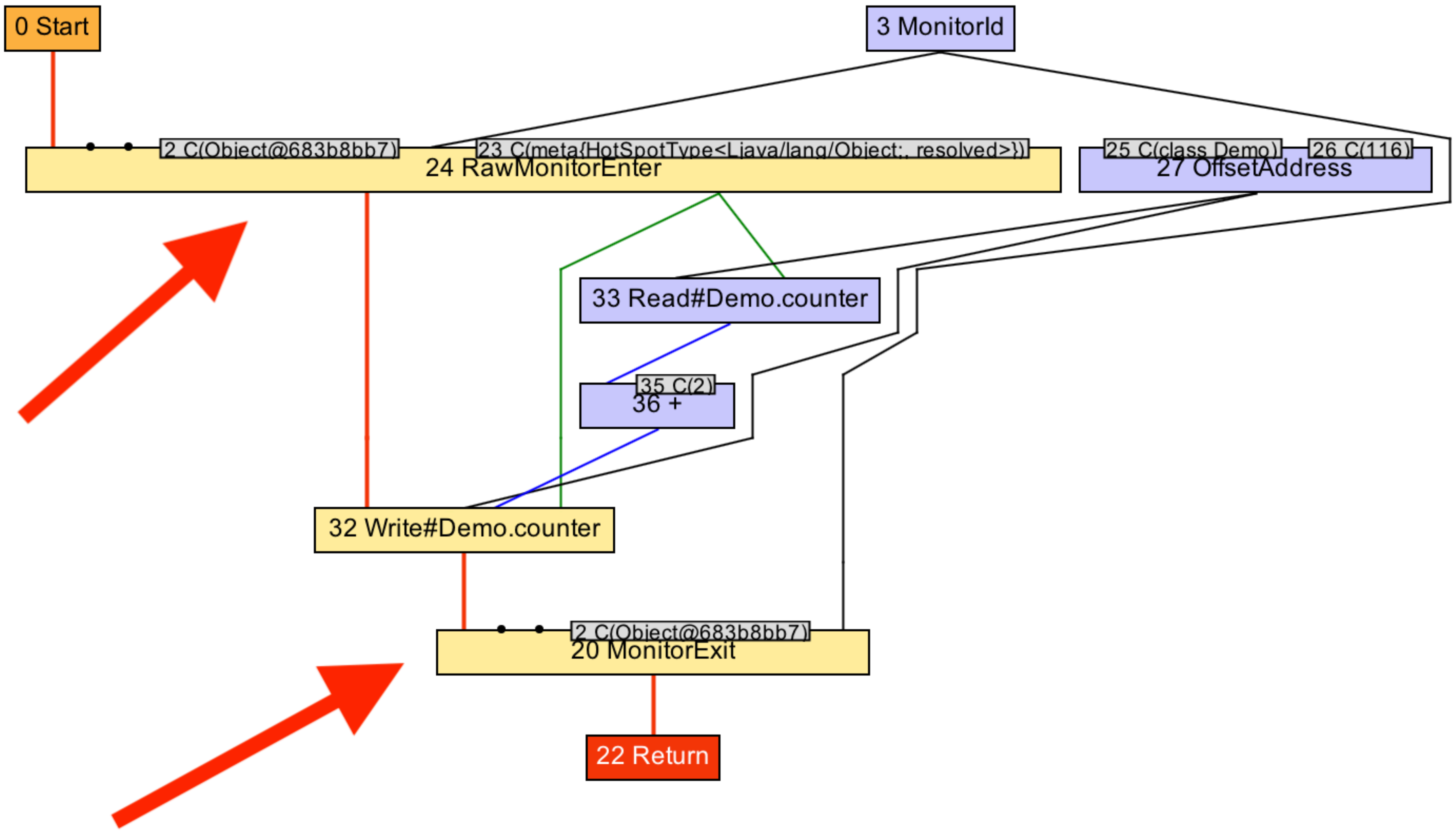

graal - org.graalvm.compiler.phases.common/src/org/graalvm/compiler/phases/common/LockEliminationPhase.java - Eclipse
Quick Access
LockEliminationPhase.java
Writable
Smart Insert
36 : 34

```java

public class LockEliminationPhase extends Phase {

    @Override
    protected void run(StructuredGraph graph) {
        for (MonitorExitNode monitorExitNode : graph.getNodes(MonitorExitNode.TYPE)) {
            FixedNode next = monitorExitNode.next();
            if ((next instanceof MonitorEnterNode || next instanceof RawMonitorEnterNode)) {
                // should never happen, osr monitor enters are always direct successors of the graph
                // start
                assert !(next instanceof OSRMonitorEnterNode);
                AccessMonitorNode monitorEnterNode = (AccessMonitorNode) next;
                if (GraphUtil.unproxify(monitorEnterNode.object()) == GraphUtil.unproxify(monitorExitNode.object())) {
                    /*
                     * We've coarsened the lock so use the same monitor id for the whole region,
                     * otherwise the monitor operations appear to be unrelated.
                     */
                    MonitorIdNode enterId = monitorEnterNode.getMonitorId();
                    MonitorIdNode exitId = monitorExitNode.getMonitorId();
                    if (enterId != exitId) {
                        enterId.replaceAndDelete(exitId);
                    }
                    GraphUtil.removeFixedWithUnusedInputs(monitorEnterNode);
                    GraphUtil.removeFixedWithUnusedInputs(monitorExitNode);
                }
            }
        }
    }
}

```


0 Start
3 MonitorId
2 C(Object@683b8bb7)
23 C(meta{HotSpotType<Ljava/lang/Object;, resolved>})
24 RawMonitorEnter
25 C(class Demo)
26 C(116)
27 OffsetAddress
28 Read#Demo.counter
7 C(1)
8 +
29 Write#Demo.counter
2 C(Object@683b8bb7)
11 MonitorExit
13 MonitorId
2 C(Object@683b8bb7)
23 C(meta{HotSpotType<Ljava/lang/Object;, resolved>})
30 RawMonitorEnter
31 Read#Demo.counter
7 C(1)
17 +
32 Write#Demo.counter
2 C(Object@683b8bb7)
20 MonitorExit
22 Return



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


0 Start
3 MonitorId
2 C(Object@683b8bb7)
23 C(meta{HotSpotType<Ljava/lang/Object;, resolved>})
24 RawMonitorEnter
25 C(class Demo)
26 C(116)
27 OffsetAddress
33 Read#Demo.counter
35 C(2)
36 +
32 Write#Demo.counter
2 C(Object@683b8bb7)
20 MonitorExit
22 Return

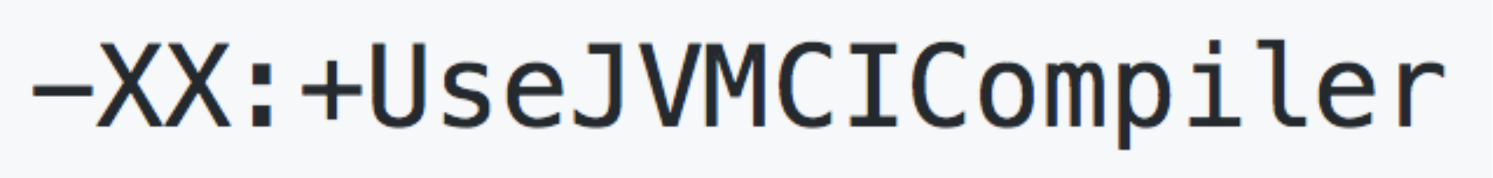
Some practicalities that I haven't talked about

Register allocation

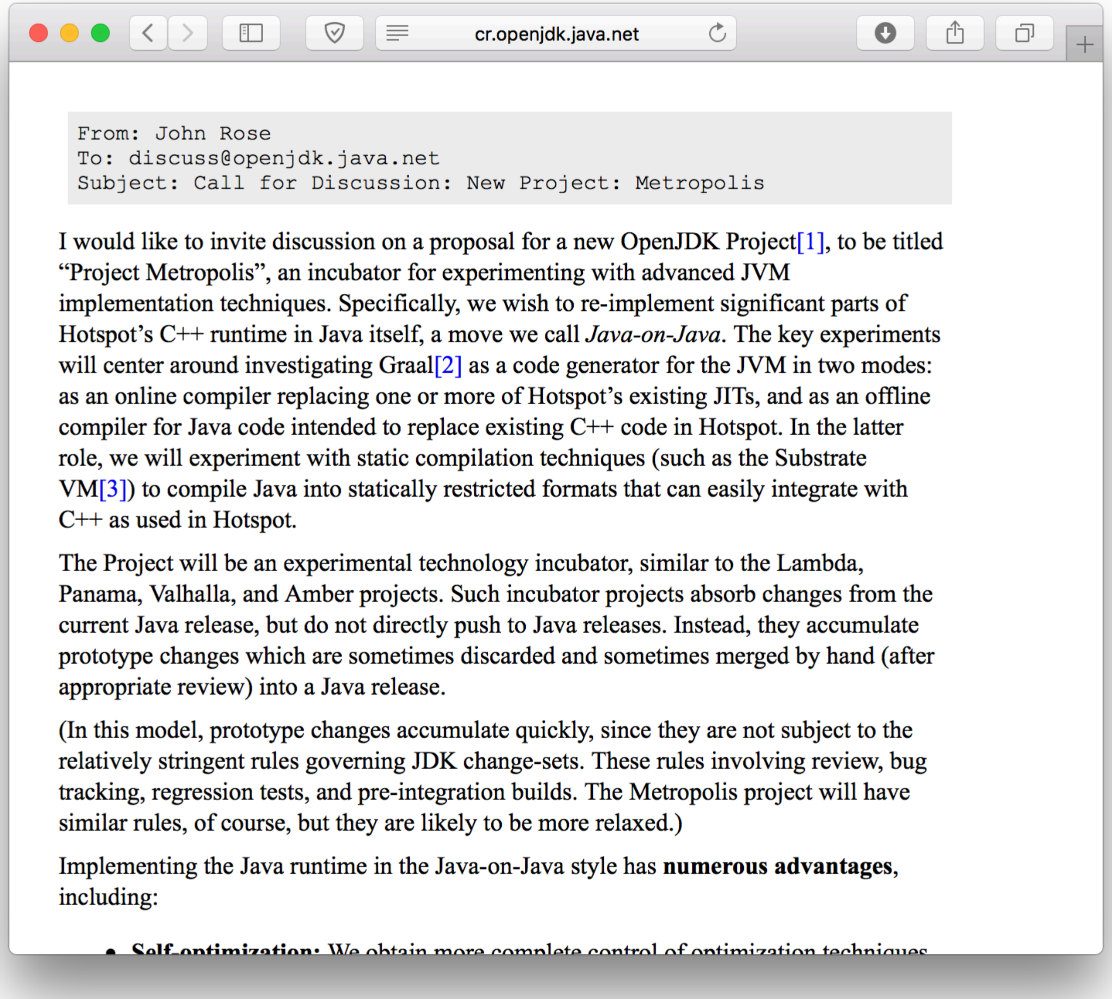
Scheduling

What can you use Graal for?

A final tier compiler



```
-XX:+UseJVMCICompiler
```



```
From: John Rose
To: discuss@openjdk.java.net
Subject: Call for Discussion: New Project: Metropolis
```

I would like to invite discussion on a proposal for a new OpenJDK Project[1], to be titled “Project Metropolis”, an incubator for experimenting with advanced JVM implementation techniques. Specifically, we wish to re-implement significant parts of Hotspot’s C++ runtime in Java itself, a move we call *Java-on-Java*. The key experiments will center around investigating Graal[2] as a code generator for the JVM in two modes: as an online compiler replacing one or more of Hotspot’s existing JITs, and as an offline compiler for Java code intended to replace existing C++ code in Hotspot. In the latter role, we will experiment with static compilation techniques (such as the Substrate VM[3]) to compile Java into statically restricted formats that can easily integrate with C++ as used in Hotspot.

The Project will be an experimental technology incubator, similar to the Lambda, Panama, Valhalla, and Amber projects. Such incubator projects absorb changes from the current Java release, but do not directly push to Java releases. Instead, they accumulate prototype changes which are sometimes discarded and sometimes merged by hand (after appropriate review) into a Java release.

(In this model, prototype changes accumulate quickly, since they are not subject to the relatively stringent rules governing JDK change-sets. These rules involving review, bug tracking, regression tests, and pre-integration builds. The Metropolis project will have similar rules, of course, but they are likely to be more relaxed.)

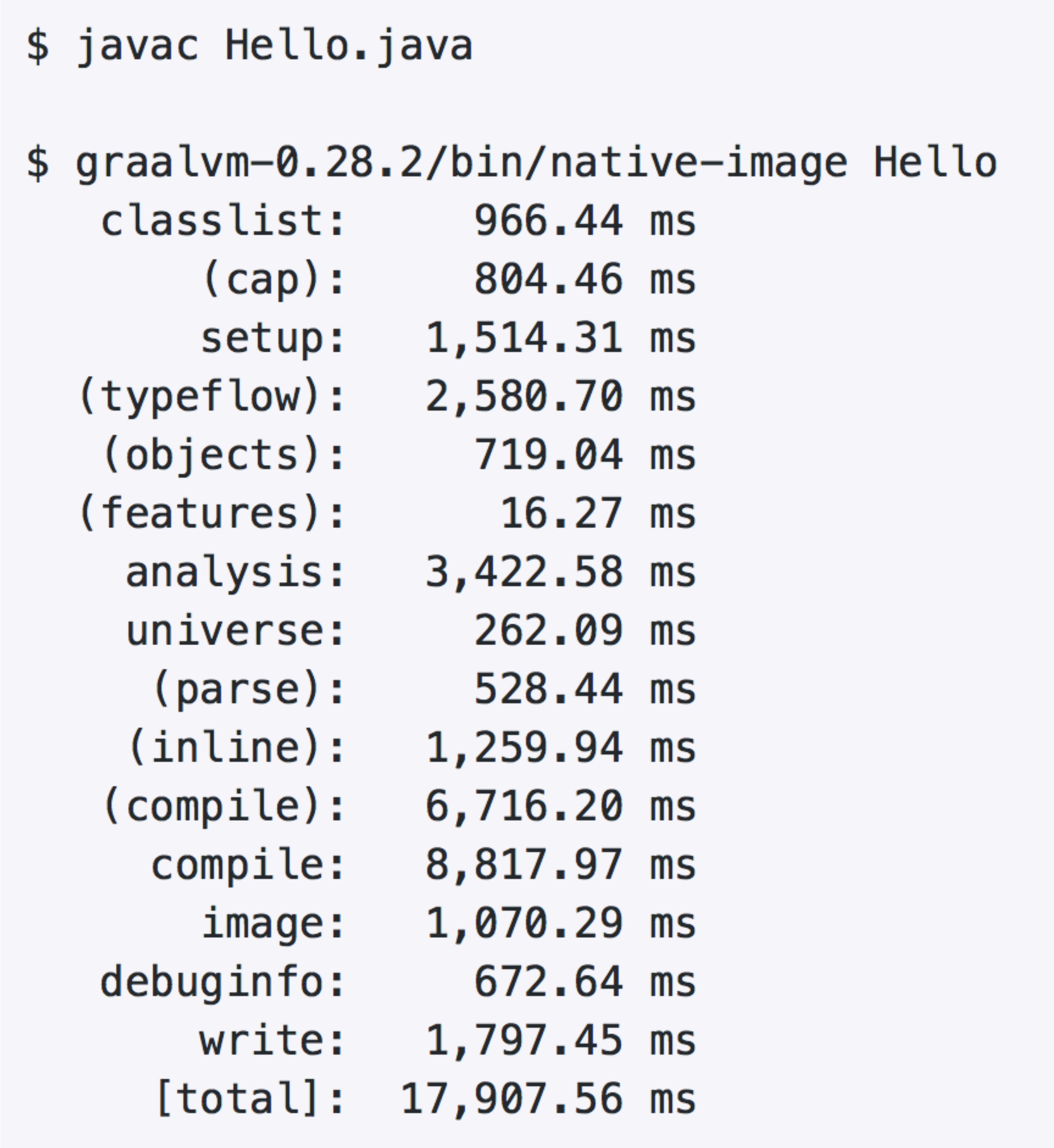
Implementing the Java runtime in the Java-on-Java style has **numerous advantages**, including:

- **Self-optimization:** We obtain more complete control of optimization techniques

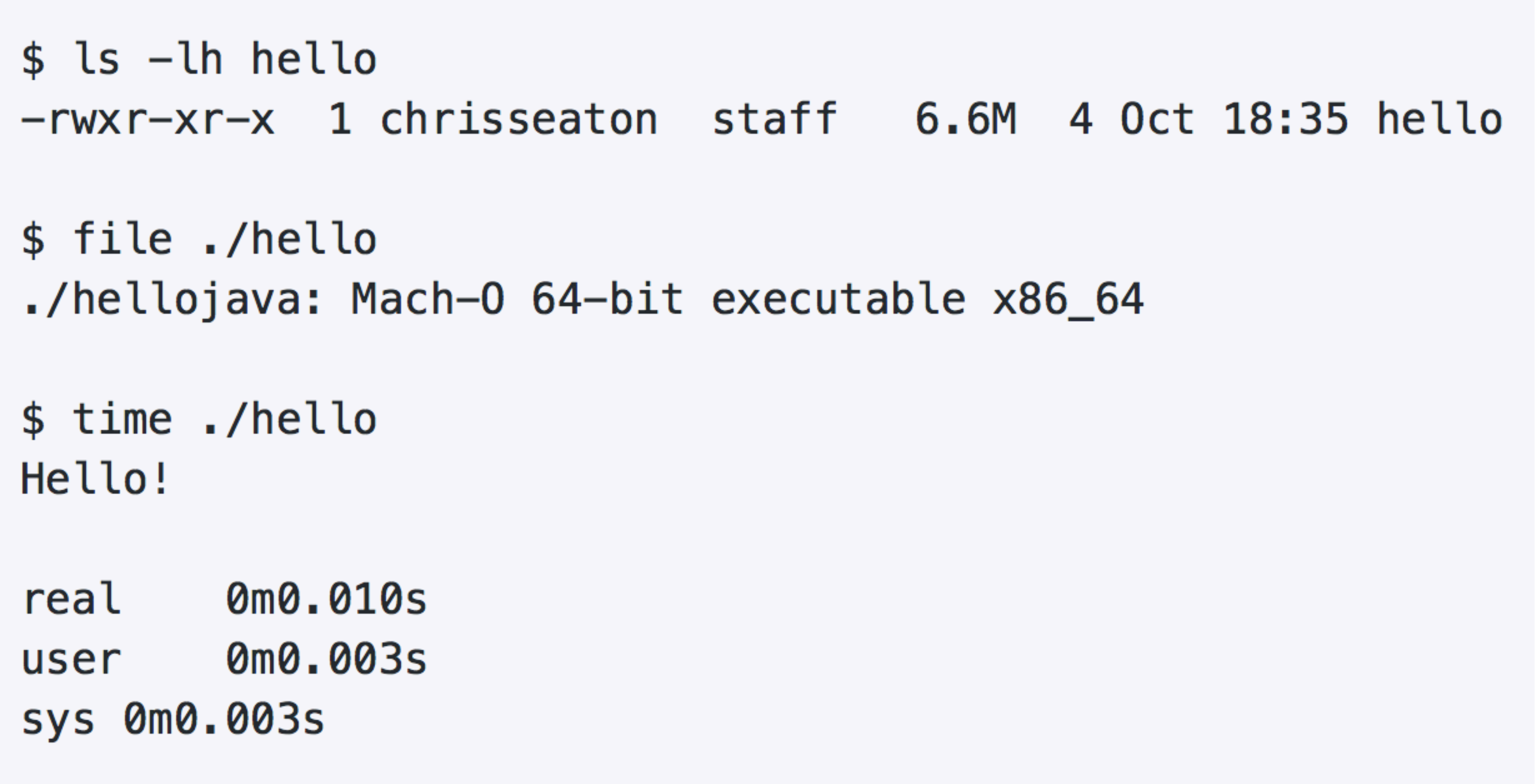
http://cr.openjdk.java.net/~jrose/metropolis/Metropolis-Proposal.html

Your own specific optimisations

Ahead of time compilation



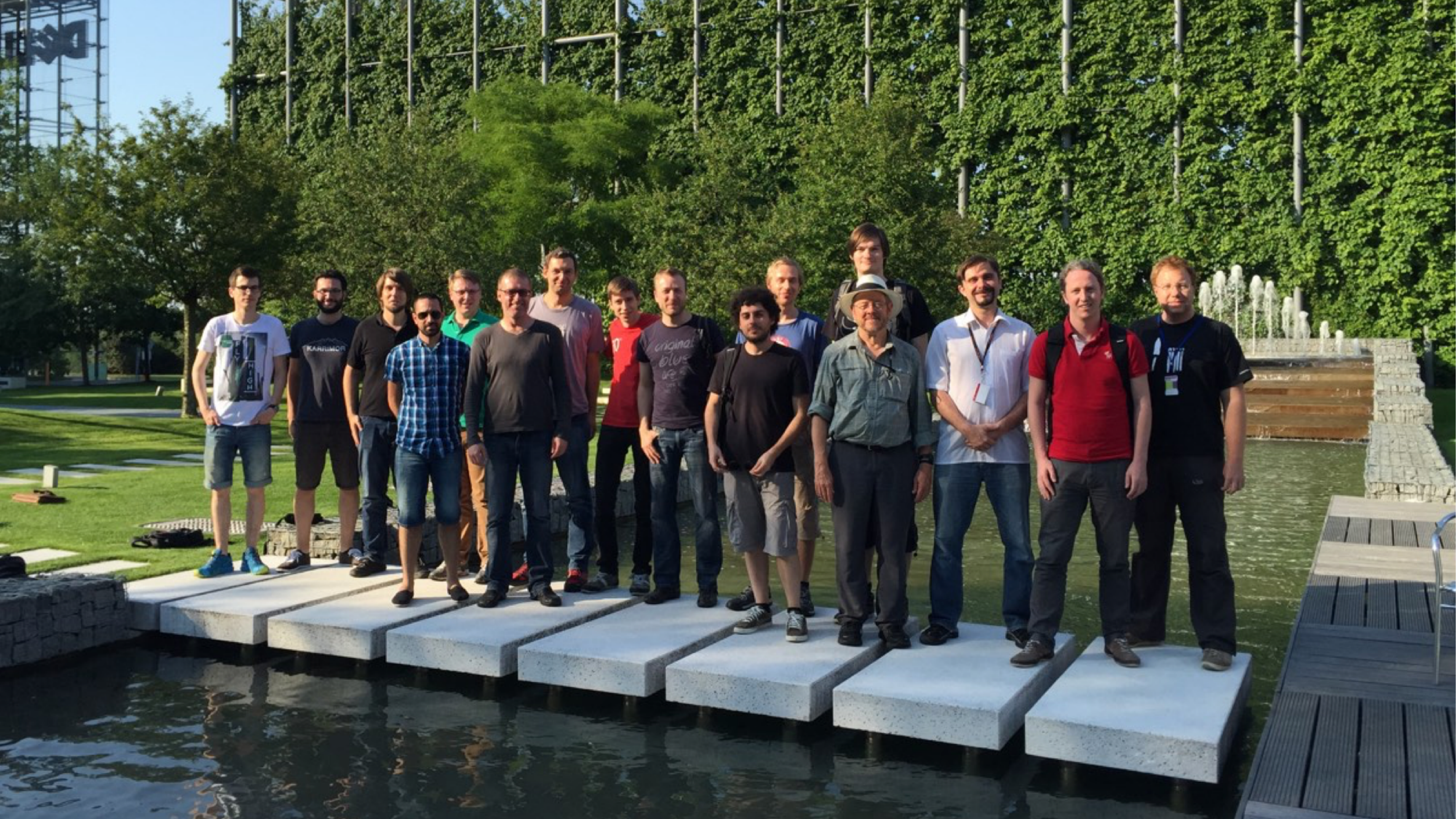
```
$ javac Hello.java

$ 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
```



```
$ 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


FLY HIGH
KARRIMOR
original blue

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.

ORACLE®

Q&A

Chris Seaton
Research Manager
Oracle Labs

chris.seaton@oracle.com
@ChrisGSeaton

ORACLE®

Integrated Cloud

Applications & Platform Services

ORACLE®