ASM framework

Eugene Kuleshov
eu@javatx.org

Class modification problems

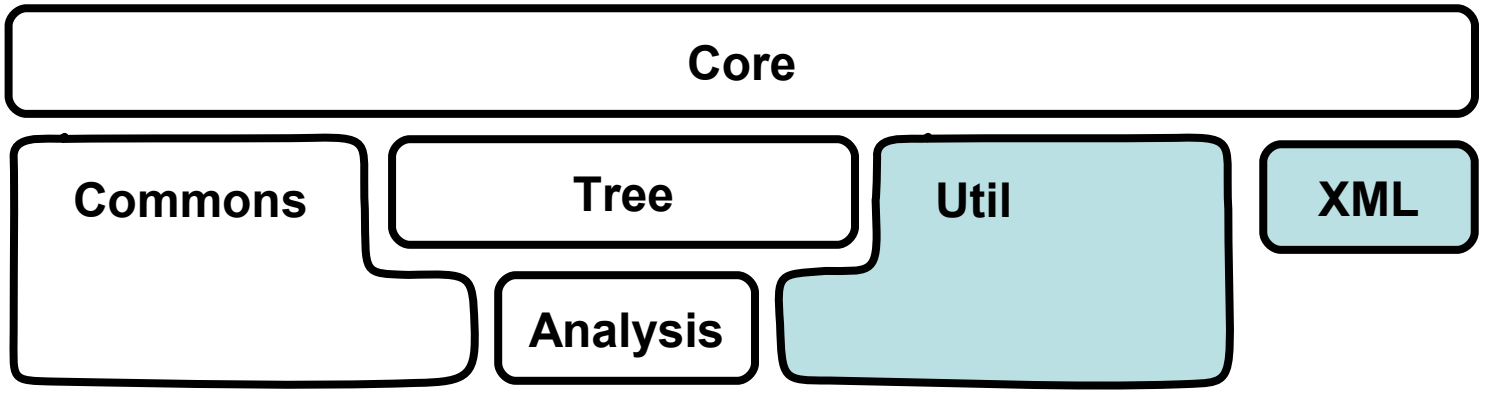
- Lots of serialization and deserialization details
- Constant pool management
  - missing or unused constants
  - constant pool indexes management
- Offsets (jump, exception table, local vars, etc)
  - become invalid if method code inserted or removed
- Computing maximum stack size and StackMap
  - require a control flow analysis

ASM bytecode framework

http://asm.objectweb.org/

- Goal: dynamic class generation and modification
  - need a very small and very fast tool
  - need a tool primarily adapted for simple transformations
  - do not need complete control over the produced classes
- Approach:
  - use the Visitor pattern without using an explicit object model
  - completely hide the (de)serialization and constant pool management details
  - represent jump offsets by Label objects
  - automatic computation of the max stack size and StackMap

ASM Packages



- org.objectweb.asm
  org.objectweb.asm.signature
- org.objectweb.asm.util
- org.objectweb.asm.commons
- org.objectweb.asm.tree
- org.objectweb.asm.tree.analysis
- org.objectweb.asm.xml

Core Package

Interfaces

- **ClassVisitor**
- **FieldVisitor**
- **MethodVisitor**
- **AnnotationVisitor**

Adapters

- **ClassAdapter**
- **MethodAdapter**

ClassReader
ClassWriter

Signature

- **SignatureReader**
- **SignatureVisitor**
- **SignatureWriter**

Opcodes, Type, Label, Attribute

Util Package

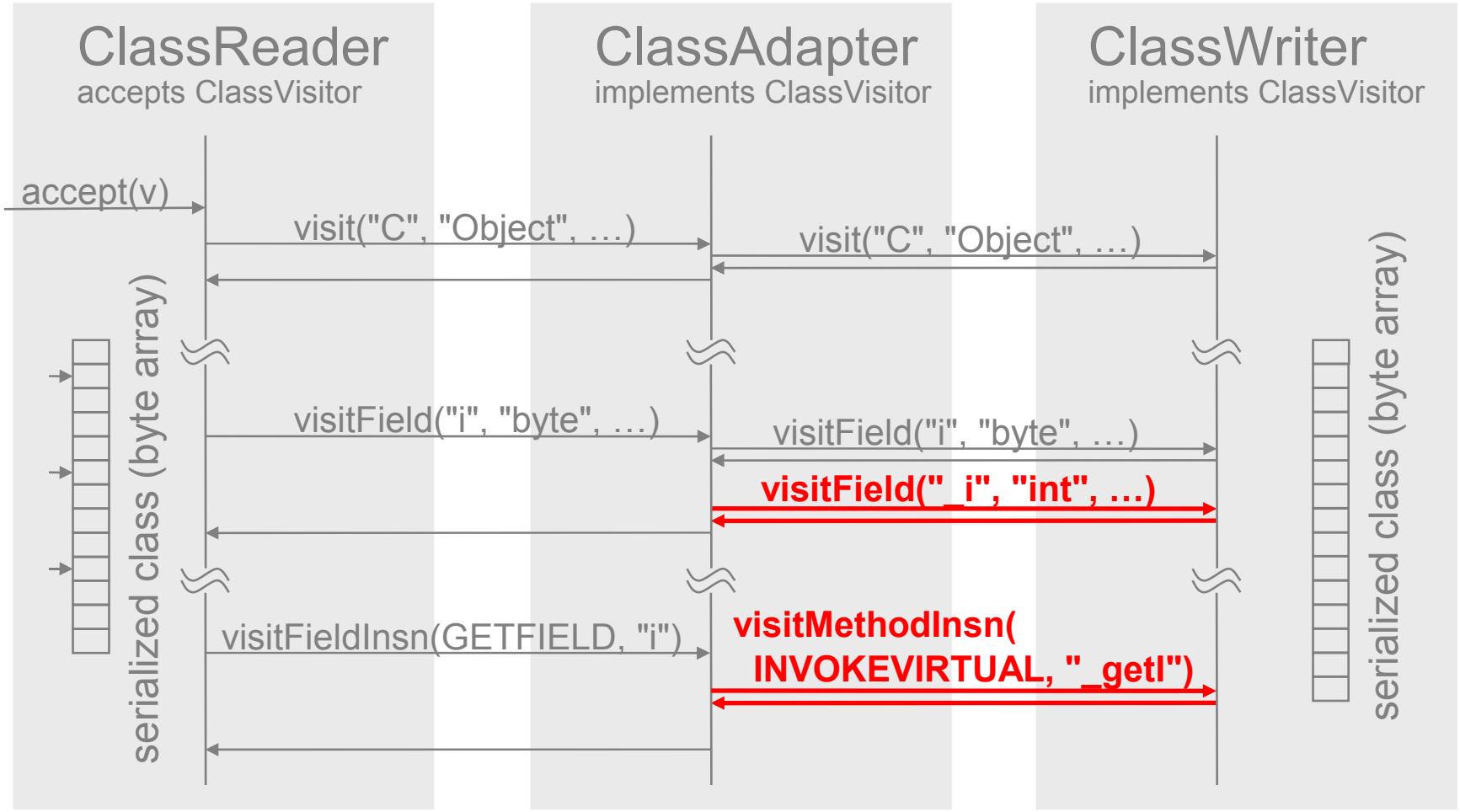
Class disassembler
TraceClassVisitor

ASM code generator
ASMifierClassVisitor

Constraints checker
CheckClassAdapter

All extends the **ClassAdapter**

ASM Main Idea



Example / Hello World

```
package asm;

public class HelloWorld {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }

}
```

Example / Hello World

```
ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAX);
cw.visit(V1_5, ACC_PUBLIC + ACC_SUPER,
    "asm/HelloWorld", null, "java/lang/Object", null);

MethodVisitor mv;
mv = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null);
mv.visitCode();
  mv.visitVarInsn(ALOAD, 0);
  mv.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V");
  mv.visitInsn(RETURN);
mv.visitMaxs(0, 0);
mv.visitEnd();

mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "main",
   "([Ljava/lang/String;)V", null, null);
mv.visitCode();
  mv.visitFieldInsn(GETSTATIC,
      "java/lang/System", "out", "Ljava/io/PrintStream;");
  mv.visitLdcInsn("Hello, World!");
  mv.visitMethodInsn(INVOKEVIRTUAL,
      "java/io/PrintStream", "println", "(Ljava/lang/String;)V");
  mv.visitInsn(RETURN);
mv.visitMaxs(0, 0);
mv.visitEnd();

cw.visitEnd();
return cw.toByteArray();
```

Example / Testing

```
ClassWriter cw = new ClassWriter(cr, ClassWriter.COMPUTE_MAXS);
ClassVisitor cv = new CheckClassAdapter(cw)
cv.visit(..);
// Generate class
cv.visitEnd();

final byte[] b = cw.toByteArray();
CheckClassAdapter.verify(new ClassReader(b),
   true, new PrintWriter(System.err)));

counter = (Counter) new ClassLoader() {
    Class cc = defineClass(c.getName(), b, 0, b.length);
  }.cc.newInstance();
```

The Class File Format

JVM specification - http://java.sun.com/docs/books/jvms/

- File header
- Constant Pool
  - field and method names, type descriptors,
    String literals and other constants
- Class header
- Attributes
  - Annotations
  - Fields
  - Methods
    - Code
    - Debug information
    - Exceptions
  - User defined attributes

```
ClassFile {
  u4 magic;
  u2 minor_version;
  u2 major_version;

  u2 cp_count;
  cp_info
    constant_pool[cp_count-1];

  u2 access_flags;
  u2 this_class;
  u2 super_class;
  u2 interfaces_count;
  u2 interfaces[interfaces_count];

  u2 fields_count;
  field_info
    fields[fields_count];

  u2 methods_count;
  method_info
    methods[methods_count];

  u2 attributes_count;
  attribute_info
    attributes[attributes_count];
}
```

Constant Pool

- Constant Pool
  - Tag
  - Info

```
CONSTANT_Utf8 1
CONSTANT_Integer 3
CONSTANT_Float 4
CONSTANT_Long 5
CONSTANT_Double 6
CONSTANT_Class 7
CONSTANT_String 8
CONSTANT_Fieldref 9
CONSTANT_Methodref 10
CONSTANT_InterfaceMethodref 11
CONSTANT_NameAndType 12
```

```
CONSTANT_Fieldref_info {
  u1 tag
  u2 class_index
  u2 name_and_type_index
}

CONSTANT_NameAndType_info {
  u1 tag
  u2 name_index
  u2 descriptor_index
}
```

Attributes

- General format

```
attribute_info {
   u2 attr_name_index
   u2 attr_length
   u1 info[attr_length]
 }
```

- Standard attributes

  SourceFile, ConstantValue, Code, StackMapTable, Exceptions, InnerClasses, EnclosingMethod, Synthetic, Signature, LineNumberTable, LocalVariableTable, Deprecated, SourceDebugExtension, RuntimeVisibleAnnotations, RuntimeInvisibleAnnotations, RuntimeVisibleParameterAnnotations, RuntimeInvisibleParameterAnnotations, AnnotationDefault

Superpackage attribute

JSR 294

- Superpackage attribute structure

```
Superpackage_attribute {
  u2 attribute_name_index;
  u4 attribute_length;
  u2 superpackage_name;
}
```

superpackage_name is an index into the constant pool. The constant pool entry at that index must be a CONSTANT_Utf8_info structure representing the name of the superpackage of which the class described by this ClassFile is a member. The superpackage name must be encoded in internal form.

Superpackage attribute (cont.)

JSR 294

```
ClassWriter cw;
...
SuperpackageAttribute att =
    new SuperpackageAttribute(“org/foo/MySuperPackage”);
cw.visitAttribute(att);
```

A custom Attribute class:

```
public class SuperpackageAttribute
    extends Attribute {
  public String name;

  protected SuperpackageAttribute(String name) {
    super("Superpackage");
    this.name = name;
  }

  protected ByteVector write(..) { ... }
  protected Attribute read(..) { ... }
}
```

Superpackage attribute (cont.)

JSR 294

Read and write Attribute data

```
protected Attribute read(ClassReader cr,
      int off, int len, char[] buf,
      int codeOff, Label[] labels) {
  String name = cr.readUTF8(off, buf);
  return new SuperpackageAttribute(name);
}

protected ByteVector write(ClassWriter cw,
      byte[] code, int len,
      int maxStack, int maxLocals) {
  int index = cw.newUTF8(name);
  return new ByteVector().putShort(index);
}
```

Reading custom attributes

```
Attribute[] attrs = new Attribute[] {
    new SuperpackageAttribute(),
    new SuperpackageExportTableAttribute(),
    new SuperpackageImportTableAttribute(),
    ...
  };
ClassReader.accept(cv, attrs, flags);
```

Traceable and ASMifiable

```
public class TraceableSuperpackageAttribute
    extends SuperpackageAttribute
    implements Traceable, ASMifiable {

  TraceableSuperpackageAttribute(String name) {
    super(name);
  }

  protected Attribute read(..) {
    SuperpackageAttribute attr = super.read(..);
    return new TraceableSuperpackageAttribute(attr.name);
  }

  public void trace(StringBuffer sb, Map labelNames) {
    sb.append(name);
  }

  public void asmify(StringBuffer sb, String var, Map labels) {
    sb.append("SuperPackageAttribute " + var
      + " = newSuperPackageAttribute(\"" + name + "\");\n");
  }
}
```

Offsets in the method code

JSR 308, StackMapTable

Code attribute can have reference to the code offset (JSR 308, StackMapTable)

```
{
  u2 offset;
} reference_info;
```

Converting method offsets to Label instances

```
  public int read(ClassReader cr, int off, char[] buf,
      int codeOff, Label[] labels) {
    ...
    Label label = getLabel(cr.readInt(off));
    off += 2;
    ...
  }

  private Label getLabel(int offset, Label[] labels) {
    Label l = labels[offset];
    if (l != null) {
      return l;
    }
    return labels[offset] = new Label();
  }
```

Offsets in the method code

JSR 308, StackMapTable

Writing method code offsets

```
protected ByteVector write(ClassWriter cw, byte[] code,
    int len, int maxStack, int maxLocals) {
  ByteVector bv = new ByteVector();
  ...
  bv.putShort(label.getOffset());
  ...
  return bv;
}
```

Accessing method info

Accessing method info from Code Attribute

```
public int read(ClassReader cr, int off, char[] buf,
      int codeOff, Label[] labels) {
   int methodOff = getMethodOffset(cr, codeOff, buf);
   int acc = cr.readUnsignedShort(methodOff); // method access
   int name = cr.readUTF8(methodOff + 2, buf); // method name
   int desc = cr.readUTF8(methodOff + 4, buf); // method desc
   ...
}
```

Structure of the method_info attribute:

```
method_info {
   u2 access_flags;
   u2 name_index;
   u2 descriptor_index;
   u2 attributes_count;
   attribute_info attributes[attributes_count];
}
```

Accessing method info

Search method_info based on offset of the Code_attribute

```
public static int getMethodOffset(ClassReader cr, int codeOff, char[] buf) {
  int off = cr.header + 6;
  int interfacesCount = cr.readUnsignedShort(off);
  off += 2 + interfacesCount * 2;
  int fieldsCount = cr.readUnsignedShort(off);
  off += 2;
  for (; fieldsCount > 0; --fieldsCount) {
    int attrCount = cr.readUnsignedShort(off + 6);  off += 8;  // fields
    for (; attrCount > 0; --attrCount) {
      off += 6 + cr.readInt(off + 2);
    }
  }
  int methodsCount = cr.readUnsignedShort(off);
  off += 2;
  for (; methodsCount > 0; --methodsCount) {
    int methodOff = off;
    int attrCount = cr.readUnsignedShort(off + 6);  off += 8;  // methods
    for (; attrCount > 0; --attrCount) {
      String attrName = cr.readUTF8(off, buf);  off += 6;
      if (attrName.equals("Code")) {
        if (codeOff == off) {
          return methodOff;
        }
      }
      off += cr.readInt(off - 4);
    }
  }
  return -1;
}
```

Bytecode Tricks

Debugger support

- Single source file per class
  - Source mapping **cv.visitSource(“source.foo”)**
  - Line numbers **mv.visitLineNumber(line, label)**
    *add an entry at the **beginning of each method***
  - Local variables **mv.visitLocalVariable(name, desc, signature, start, end, index)**
    *add an entry for **each local variable***
    *entries for **implicit variables** would make them visible to the debugger*
- Multiple source files per class
  - Use SMAP structure defined by JSR 45
    cv.visitSource(“source.foo”, smap)

Constant Pool

- **ClassConstantsCollector** and **ConstantPool** classes from the org.objectweb.asm.optimizer package
- **ClassWriter** methods for creating constants:
  newConst(..), newUTF8(..), newClass(..), newField(..), newMethod(..), newNameType(..)
- **ClassReader** methods to read raw bytecode:
  readByte(..), readUnsignedShort(..), readShort(..), readInt(..), readLong(..), readUTF8(int offset, char[] buf), readClass(int offset, char[] buf), readConst(int offset, char[] buf)

StackMapTable

- **ClassReader.SKIP_FRAMES**
- **ClassReader.EXPAND_FRAMES**
- **ClassWriter.COMPUTE_FRAMES**
- **ClassWriter.getCommonSuperClass(**
  String type1, String type2)
  by default uses Class.forName(), but can be implemented using ASM to read type info
- **MethodVisitor.visitFrame(int type,**
  int nLocal, Object[] local, int nStack, Object[] stack)
- **LocalVariablesSorter**
  provides basic support for incremental updates of StackMapTable information
- **AnalyzerAdapter**
  keeps track of stack map frame changes between visitFrame(..) calls

Example / Class Dependencies

```
final Set dependencies = new HashSet();
ClassReader cr = new ClassReader(is);
cr.accept(new RemappingClassAdapter(
    new EmptyVisitor(),
    new Remapper() {
      public String map(String typeName) {
        dependencies.add(typeName);
        return typeName;
      }
    }), ClassReader.SKIP_FRAMES);
```

Resources

- ASM Project web site
  http://asm.objectweb.org/
- ASM User Guide
  http://download.forge.objectweb.org/asm/asm-guide.pdf
- ASM Developer Guide
  http://asm.objectweb.org/doc/developer-guide.html
- Eugene Kuleshov blog
  http://www.jroller.com/eu/category/Java

Q & A

ClassVisitor

Modifiers, name, super, interfaces		visit(version, access, name, ..)
Source file name (optional)		visitSource(source, debug)
Inner class Name *		visitInnerClass(name, ...)
Enclosing class Name		visitOuterClass(owner, name, ..)
Annotation *		AnnotationVisitor visitAnnotation(desc, visible)
Attribute *		**visitAttribute(attribute)**
Field *	Modifiers, name, type	FieldVisitor visitField(access, name, desc, signature, value)
	Annotation *	
	Attribute *	
Method *	modifiers, name, params	MethodVisitor visitMethod(access, name, desc, signature, exceptions)
	Annotation *	
	Attribute *	
	Method code	
		visitEnd()

FieldVisitor

Field	Modifiers, name, type	FieldVisitor visitField(access, name, desc, signature, value)
	Annotation *	AnnotationVisitor visitAnnotation(desc, visible)
	Attribute *	**visitAttribute(attribute)**
		visitEnd()

MethodVisitor

Method	modifiers, name, params	MethodVisitor visitMethod(access, name, desc, signature, exceptions)
	Annotation *	AnnotationVisitor visitAnnotation(desc, visible)
		AnnotationVisitor visitAnnotationDefault()
		AnnotationVisitor visitParameterAnnotation(parameter, desc, visible)
	Attribute *	**visitAttribute(attribute)**
	Method code	visitCode()
	Try/catch block *	visitTryCatchBlock(start, end, handler, type)
	Local variable *	visitLocalVariable(name, desc, signature, ..)
	Line number *	visitLineNumber(line, startLabel)
	Instructions	...
	Line number *	visitMaxs(maxStack, maxLocals)
		visitEnd()

Signatures

Usually signatures are passed trough

To change signatures use **SignatureVisitor**, **SignatureReader** and **SignatureWriter** modeled on the visitor idea

```
SignatureReader r = new SignatureReader(signature);
SignatureWriter w = new SignatureWriter();
SignatureVisitor v =
   new RemappingSignatureAdapter(w, remapper);
if (typeSignature) r.acceptType(v);
else r.accept(v);
String remappedSignature = w.toString();
```

Tree Package

Class Structures

- **ClassNode / *ClassVisitor***
- **InnerClassNode**
- **AnnotationNode / *AnnotationVisitor***
- **FieldNode / *FieldVisitor***
- **MethodNode / *MethodVisitor***

Method Structures

- **InsnList**
- **TryCatchBlockNode**
- **LocalVariableNode**

Instruction List

- **FrameNode**
- **LabelNode**
- **LineNumberNode**
- **InsnNode**
- **IntInsnNode**
- **VarInsnNode**
- **TypeInsnNode**
- **FieldInsnNode**
- **MethodInsnNode**
- **JumpInsnNode**
- **LdcInsnNode**
- **IincInsnNode**
- **TableSwitchInsnNode**
- **LookupSwitchInsnNode**
- **MultiANewArrayInsnNode**

Example / ClassNode

```
ClassNode cn = new ClassNode();

ClassReader cr = new ClassReader(bytecode);
cr.accept(cn, 0);

... analyze or transform cn structures

ClassWriter cw =
  new ClassWriter(ClassWriter.COMPUTE_MAXS);
cn.accept(new MyClassAdapter(cw, cn));
```

Example / MethodNode

```
ClassAdapter ca = new ClassAdapter(cv) {
  public MethodVisitor visitMethod(int access,
        String name, String desc,
        String signature, String[] exceptions) {
    final MethodVisitor mv =
        super.visitMethod(access, name, desc,
                          signature, exceptions);
    return new MethodNode(access, name, desc,
                          signature, exceptions) {
       public void visitEnd() {
         ... analyze or transform this method code
         this.accept(mv);
       }
     };
   }
 };
```

Analysis Package

Analysis framework

Analyzer

Interpreter

Value

BasicValue

BasicInterpreter

BasicVerifier

SimpleVerifier

SourceValue

SourceInterpreter

Example / BasicInterpreter

Get type info for each variable and stack slot for each method instruction

```
Analyzer a = new Analyzer(new BasicInterpreter());
a.analyze(className, methodNode);

Frame[] frames = a.getFrames();
for (int i = 0; i < frames.length; i++) {
  Frame f = frames[i];
  for (int j = 0; j < f.getLocals(); ++j) {
    BasicValue local = (BasicValue) f.getLocal(j);
    // ... local.getType()
  }
  for (int j = 0; j < f.getStackSize(); ++j) {
    BasicValue stack = (BasicValue) f.getStack(j);
    // ...
  }
}
```

Example / SourceInterpreter

Get variable name used as parameter of the given method call

```
Analyzer a = new Analyzer(new SourceInterpreter());
Frame[] frames = a.analyze(cn.name, mn);

LabelNode label = findLineLabel(mn.instructions, line);
int index = findMethodCall(mn, label, methodName);

SourceValue stack = (SourceValue)
   frames[index].getStack(param);
Object insn = stack.insns.iterator().next();
if (insn instanceof VarInsnNode) {
  VarInsnNode vinsn = (VarInsnNode) insn;
  return ((LocalVariableNode)
      mn.localVariables.get(vinsn.var)).name;
}

return null;
```

Example (cont.)

Get variable name used as parameter of the given method call

```
LabelNode findLineLabel(InsnList insns, int line) {
  for (Iterator it = insns.iterator(); it.hasNext();) {
    Object n = it.next();
    if(n instanceof LineNumberNode && ((LineNumberNode) n).line==line) {
      return ((LineNumberNode) n).start;
    }
  }
  return null;
}

int findMethodCall(InsnList insns, LabelNode label, String name) {
  boolean foundLabel = false;
  for (int i = 0; i < insns.size(); i++) {
    AbstractInsnNode n = insns.get(i);
    if (!foundLabel && n==label) foundLabel = true;
    else if (foundLabel
        && n instanceof MethodInsnNode
        && ((MethodInsnNode) n).name.equals(name)) return i;
  }
  return -1;
}
```

Commons Package

Helpers

- **EmptyVisitor (all)**
- **AnalyzerAdapter**
- **LocalVariablesSorter**

Transformations

- **AdviceAdapter**
- **CodeSizeEvaluator**
- **JSRInlinerAdapter**
- **StaticInitMerger (class)**

Code generation

- **SerialVersionUIDAdder (class)**
- **GeneratorAdapter (method)**

Remapping

- **RemappingAnnotationAdapter**
- **RemappingClassAdapter**
- **RemappingFieldAdapter**
- **RemappingMethodAdapter**
- **RemappingSignatureAdapter**
  - **Remapper**
  - **SimpleRemapper**

Example / Merge Classes

```
public class MergeAdapter extends ClassAdapter {
  private ClassNode cn;
  private String className;

  public MergeAdapter(ClassVisitor cv, ClassNode cn) {
    super(cv);
    this.cn = cn;
  }

  public void visit(int version, int access, String name,
   String signature,
      String superName, String[] interfaces) {
    super.visit(version, access,
       name, signature, superName, interfaces);
    this.className = name;
  }
```

Example / Merge (Cont.)

```
public void visitEnd() {
  Iterator fieldIterator = cn.fields.iterator();
  while(fieldIterator.hasNext();) {
    ((FieldNode) fieldIterator.next()).accept(this);
  }
  Iterator methodIterator = cn.methods.iterator()
  while(methodIterator.hasNext()) {
    MethodNode mn = (MethodNode) methodIterator.next();
    MethodVisitor mv = cv.visitMethod(mn.access, mn.name,
        mn.desc, mn.signature,
        (String[])  mn.exceptions.toArray());
    mn.instructions.resetLabels();
    mn.accept(new RemappingMethodAdapter(mn.access,
        mn.desc, mv,
        new SimpleRemapper(cn.name, className)));
  }
  super.visitEnd();
}
```

Example / AdviceAdapter

Insert code on method enter

```
class EnteringAdapter extends
    AdviceAdapter {
  String name;
  int timeVar;
  Label timeVarStart = new Label();
  Label timeVarEnd = new Label();

  public
    PrintEnteringAdapter(MethodVisitor mv,
       int acc, String name,
       String desc) {
    super(mv, acc, name, desc);
    this.name = name;
  }
```

```
protected void onMethodEnter() {
    int timeVar =
       newLocal(Type.getType("J"));
    visitLocalVariable("timeVar", "J",
       null, timeVarStart, timeVarEnd,
       timeVar);
    visitLabel(timeVarStart);

    super.visitFieldInsn(GETSTATIC,
       "java/lang/System", "err",
       "Ljava/io/PrintStream;");
    super.visitLdcInsn("Entering " +  name);
    super.visitMethodInsn(INVOKEVIRTUAL,
       "java/io/PrintStream", "println",
       "(Ljava/lang/String;)V");
  }

  public void visitMaxs(int stack, int vars){
    visitLabel(timeVarEnd);
    super.visitMaxs(stack, vars);
  }
}
```

Example / AdviceAdapter

Insert code on method exit

```
class ExitingAdapter extends AdviceAdapter {
  private String methodName;

  public void onMethodExit(int opcode) {
    mv.visitFieldInsn(GETSTATIC, "java/lang/System",
       "err", "Ljava/io/PrintStream;");
    if(opcode==ATHROW) {
      mv.visitLdcInsn("Exiting on exception " + methodName);
    } else {
      mv.visitLdcInsn("Exiting " + methodName);
    }
    mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream",
      "println", "(Ljava/lang/String;)V");
  }
}
```

Example / AdviceAdapter

Insert try / finally

```
class FinallyAdapter extends
    AdviceAdapter {
  String methodName;
  Label startFinally = new Label();

  ...

  public void visitCode() {
    super.visitCode();
    mv.visitLabel(startFinally);
  }

  public void visitMaxs(int maxStack,
    int maxLocals) {
    Label endFinally = new Label();
    mv.visitTryCatchBlock(startFinally,
        endFinally, endFinally, null);
    mv.visitLabel(endFinally);

    onFinally(ATHROW);

    mv.visitInsn(ATHROW);
    mv.visitMaxs(maxStack, maxLocals);
  }

  protected void onMethodExit(int opcode) {
    if(opcode!=ATHROW) {
      onFinally(opcode);
    }
  }

  private void onFinally(int opcode) {
    mv.visitFieldInsn(GETSTATIC,
       "java/lang/System", "err",
       "Ljava/io/PrintStream;");
    mv.visitLdcInsn("Exiting " +
        methodName);
    mv.visitMethodInsn(INVOKEVIRTUAL,
        "java/io/PrintStream",
        "println",
        "(Ljava/lang/String;)V");
  }
}
```

Replace Field Access

```
public class FieldAccessAdapter extends MethodAdapter implements Opcodes {
  private final String cname;
  private final Map adapters;

  public FieldAccessAdapter(MethodVisitor mv, String cname, Map adapters) {
    super(mv);
    this.cname = cname;
    this.adapters = adapters;
  }

  public void visitFieldInsn(int opcode, String owner, String name, String
   desc) {
    Info info = matchingInfo(opcode, owner, name, desc);
    if(info!=null) {
      super.visitMethodInsn(INVOKESTATIC, cname,
         info.adapterName, info.adapterDesc);
    } else {
      super.visitFieldInsn(opcode, owner, name, desc);
    }
  }
  ...
```

Replace Method Call

```
public class MethodCallAdapter extends MethodAdapter implements Opcodes {
  private final String cname;
  private final Set infos;

  public MethodCallAdapter(MethodVisitor mv, String cname, Set infos) {
    super(mv);
    this.cname = cname;
    this.infos = infos;
  }

  public void visitMethodInsn(int opcode, String owner, String name, String
   desc) {
    Info info = matchingInfo(opcode, owner, name, desc);
    if(info!=null) {
      super.visitMethodInsn(INVOKESTATIC, cname, info.adapterName,
   info.adapterDesc);
    } else {
      super.visitMethodInsn(opcode, owner, name, desc);
    }
  }
  ...
```

Inline Method

```
class MethodCallInliner extends
    LocalVariablesSorter {
  private final String oldClass;
  private final String newClass;
  private final MethodNode mn;
  private List blocks = new ArrayList();
  private boolean inlining;

 public void visitTryCatchBlock(
    Label start, Label end,
    Label handler, String type) {
    if(inlining)
      super.visitTryCatchBlock(start,
        end, handler, type);
    else
      blocks.add(new CatchBlock(start,
        end, handler, type));
  }

  public void visitMethodInsn(int opcode,
    String owner, String name,
    String desc) {
    if(!canInline(owner, name, desc)) {
      mv.visitMethodInsn(opcode, owner,
        name, desc);
      return;
    }
    ...
    inlining = true;
    Label end = new Label();
    mn.instructions.resetLabels();
    mn.accept(new InliningAdapter(this,
      opcode==Opcodes.INVOKESTATIC
        ? Opcodes.ACC_STATIC : 0,
      desc,
      new Remapper(
        Collections.singletonMap(
          oldClass, newClass)),
      end));
    super.visitLabel(end);
    inlining = false;
  }

  public void visitMaxs(int stack, int
    locals) {
    for(Iterator it = blocks.iterator();
        it.hasNext(); ) {
      CatchBlock b =
        (CatchBlock) it.next();
      super.visitTryCatchBlock(b.start,
        b.end, b.handler, b.type);
    }
    super.visitMaxs(stack, locals);
  }
}
```

Inline Method (Cont.)

```
class InliningAdapter
    extends RemappingMethodAdapter
    implements Opcodes {
  private final LocalVariablesSorter lvs;
  private final Label end;

  public InliningAdapter(
        LocalVariablesSorter mv,
        Label end, int acc, String desc,
        Remapper remapper) {
    super(acc, desc, mv, remapper);
    this.lvs = mv;
    this.end = end;
    int off = 0;
    if((acc & ACC_STATIC)==0) off = 1;

    Type[] args =
        Type.getArgumentTypes(desc);
    for(int i = args.length-1; i>=0; i--) {
      super.visitVarInsn(
          args[i].getOpcode(ISTORE),
          i + off);
    }
    if(offset>0) {
      super.visitVarInsn(ASTORE, 0);
    }
}
```

```
  public void visitInsn(int opcode) {
    if(opcode!=RETURN) {
      super.visitInsn(opcode);
      return;
    }
    super.visitJumpInsn(GOTO, end);
  }

  public void visitMaxs(int stack,
      int locals) {
  }

  protected int newLocalMapping(Type t) {
    return lvs.newLocal(t);
  }
}
```