```java
// Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas

package jminusminus;

import java.io.IOException;
import java.util.ArrayList;
import static jminusminus.CLConstants.*;

/**
 * Representation of attribute_info structure (JVM Spec Section 4.8). Classes
 * representing individual attributes inherit this class. This file has
 * representations for all attributes specified in JVM Spec Second Edition,
 * including the ones that were added for JDK 1.5.
 *
 * Attributes are used in the ClassFile (CLFile), field_info (CLFieldInfo),
 * method_info (CLMethodInfo), and Code_attribute (CLCodeAttribute) structures
 * of the class file format. While there are many kinds of attributes, only some
 * are mandatory; these include:
 *
 * InnerClasses_attribute (class attribute) Synthetic_attribute (class, field,
 * and method attribute) Code_attribute (method attribute) Exceptions_attribute
 * (method attribute)
 *
 * CLAbsorber is capable of reading all attributes listed in this file. The ones
 * which it does not recognize, it simply skips them and reports a warning to
 * that extent.
 *
 * CLEmitter implicitly adds the required attributes to the appropriate
 * structure. The optional attributes have to be added explicitly using the
 * addClassAttribute(), addFieldAttribute(), addMethodAttribute(), and
 * addCodeAttribute() methods in CLEmitter.
 */

abstract class CLAttributeInfo {

    // The fields below represent the members of the
    // attribute_info
    // structure and are thus inherited by the child classes of
    // CLAttributeInfo. These classes define their own fields
    // (if any) representing the members of the individual
    // attribute_info structures they represent.

    /** attribute_info.attribute_name_index item. */
    public int attributeNameIndex;

    /** attribute_info.attribute_length item. */
    public long attributeLength;

    /**
     * Construct an CLAttributeInfo object.
     *
     * @param attributeNameIndex
     *            attribute_info.attribute_name_index item.
     * @param attributeLength
     *            attribute_info.attribute_length item.
     */

    protected CLAttributeInfo(int attributeNameIndex, long attributeLength) {
        this.attributeNameIndex = attributeNameIndex;
        this.attributeLength = attributeLength;
    }

    /**
     * Write the contents of this attribute to the specified output stream.
     *
     * @param out
     *             output stream.
     * @throws IOException
     *              if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(attributeNameIndex);
        out.writeInt((long) attributeLength);
    }

    /**
     * Write the contents of this attribute to STDOUT in a format similar to
     * that of javap.
     *
     * @param p
     *             for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Attribute Name Index: %s\n", attributeNameIndex);
        p.printf("Attribute Length: %s\n", attributeLength);
    }

}

/**
 * Representation of ConstantValue_attribute structure (JVM Spec Section 4.8.2).
 * This is a required field attribute.
 */

class CLConstantValueAttribute extends CLAttributeInfo {

    /** ConstantValue_attribute.constantvalue_index item. */
    public int constantValueIndex;

    /**
     * Construct a CLConstantValueAttribute object.
     *
     * @param attributeNameIndex
     *             ConstantValue_attribute.attribute_name_index item.
     * @param attributeLength
     *             ConstantValue_attribute.attribute_length item.
     * @param constantValueIndex
     *             ConstantValue_attribute.constantvalue_index item.
     */

    public CLConstantValueAttribute(int attributeNameIndex,
            long attributeLength, int constantValueIndex) {
        super(attributeNameIndex, attributeLength);
        this.constantValueIndex = constantValueIndex;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(constantValueIndex);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("ConstantValue {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Constant Value Index: %s\n", constantValueIndex);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of exception_table entry structure (JVM Spec Section 4.8.3).
 */

class CLExceptionInfo {

    /** exception_table_entry.start_pc item. */
    public int startPC;

    /** exception_table_entry.end_pc item. */
    public int endPC;

    /** exception_table_entry.handler_pc item. */
    public int handlerPC;

    /** exception_table_entry.catch_type item. */
    public int catchType;

    /**
     * Construct a CLExceptionInfo object.
     * 
     * @param startPC
     *            exception_table_entry.start_pc item.
     * @param endPC
     *            exception_table_entry.end_pc item.
     * @param handlerPC
     *            exception_table_entry.handler_pc item.
     * @param catchType
     *            exception_table_entry.catch_type item.
     */

    public CLExceptionInfo(int startPC, int endPC, int handlerPC, int catchType)
{
        this.startPC = startPC;
        this.endPC = endPC;
        this.handlerPC = handlerPC;
        this.catchType = catchType;
    }

    /**
     * Write the contents of this object to the specified output stream.
     * 
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(startPC);
        out.writeShort(endPC);
        out.writeShort(handlerPC);
        out.writeShort(catchType);
    }

    /**
     * Write the contents of this object to STDOUT in a format similar to that
     * of javap.
     * 
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("%-8s    %-6s    %-10s    %-10s\n", startPC, endPC, handlerPC,
                catchType);
    }

}

/**
 * Representation of Code_attribute structure (JVM Spec Section 4.8.2). This is
 * a required method attribute.
 */

class CLCodeAttribute extends CLAttributeInfo {

    /** Code_attribute.max_stack item. */
    public int maxStack;

    /** Code_attribute.max_locals item. */
    public int maxLocals;

    /** Code_attribute.code_length item. */
    public long codeLength;

    /**
     * Code_attribute.code item.
     */
    public ArrayList<Integer> code;

    /** Code_attribute.exception_table_length item. */
    public int exceptionTableLength;

    /** Code_attribute.exception_table item. */
    public ArrayList<CLExceptionInfo> exceptionTable;

    /** Code_attribute.attributes_count item. */
    public int attributesCount;

    /** Code_attribute.attributes item. */
    public ArrayList<CLAttributeInfo> attributes;

    /**
     * Construct and return an integer from the four unsigned bytes specified.
     *
     * @param a
     *            unsigned byte.
     * @param b
     *            unsigned byte.
     * @param c
     *            unsigned byte.
     * @param d
     *            unsigned byte.
     * @return an integer constructed from the four unsigned bytes specified.
     */

    private int intValue(int a, int b, int c, int d) {
        return (a << 24) | (b << 16) | (c << 8) | d;
    }

    /**
     * Construct a CLCodeAttribute object.
     *
     * @param attributeNameIndex
     *            Code_attribute.attribute_name_index item.
     * @param attributeLength
     *            Code_attribute.attribute_length item.
     * @param maxStack
     *            Code_attribute.max_stack item.
     * @param maxLocals
     *             Code_attribute.max_locals item.
     * @param codeLength
     *             Code_attribute.code_length item.
     * @param code
     *             Code_attribute.code item.
     * @param exceptionTableLength
     *             Code_attribute.exception_table_length item.
     * @param exceptionTable
     *             Code_attribute.exception_table item.
     * @param attributesCount
     *             Code_attribute.attributes_count item.
     * @param attributes
     *             Code_attribute.attributes item.
     */

    public CLCodeAttribute(int attributeNameIndex, long attributeLength,
            int maxStack, int maxLocals, long codeLength,
            ArrayList<Integer> code, int exceptionTableLength,
            ArrayList<CLExceptionInfo> exceptionTable, int attributesCount,
            ArrayList<CLAttributeInfo> attributes) {
        super(attributeNameIndex, attributeLength);
        this.maxStack = maxStack;
        this.maxLocals = maxLocals;
        this.codeLength = codeLength;
        this.code = code;
        this.exceptionTableLength = exceptionTableLength;
        this.exceptionTable = exceptionTable;
        this.attributesCount = attributesCount;
        this.attributes = attributes;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(maxStack);
        out.writeShort(maxLocals);
        out.writeInt(codeLength);
        for (int i = 0; i < code.size(); i++) {
            out.writeByte(code.get(i));
        }
        out.writeShort(exceptionTableLength);
        for (int i = 0; i < exceptionTable.size(); i++) {
            exceptionTable.get(i).write(out);
        }
        out.writeShort(attributesCount);
        for (int i = 0; i < attributes.size(); i++) {
            attributes.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Code {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Max Stack: %s\n", maxStack);
        p.printf("Max Locals: %s\n", maxLocals);
        p.printf("Code Length: %s\n", codeLength);
        p.printf("%-10s%-17s%s\n", "PC", "Opcode", "Operands");
        p.printf("%-10s%-17s%s\n", "--", "------", "--------");
        for (int i = 0; i < code.size(); i++) {
            int pc = i;
            int opcode = code.get(i);
```

```java
                String mnemonic = CLInstruction.instructionInfo[opcode].mnemonic;
                int operandBytes =
CLInstruction.instructionInfo[opcode].operandCount;
                short operandByte1, operandByte2, operandByte3, operandByte4;
                int pad, deflt;
                switch (operandBytes) {
                case 0:
                    p.printf("%-10s%-17s\n", pc, mnemonic);
                    break;
                case 1:
                    operandByte1 = code.get(++i).shortValue();
                    p.printf("%-10s%-17s%-5s\n", pc, mnemonic, operandByte1);
                    break;
                case 2:
                    operandByte1 = code.get(++i).shortValue();
                    operandByte2 = code.get(++i).shortValue();
                    p.printf("%-10s%-17s%-5s%-5s\n", pc, mnemonic, operandByte1,
                            operandByte2);
                    break;
                case 3:
                    operandByte1 = code.get(++i).shortValue();
                    operandByte2 = code.get(++i).shortValue();
                    operandByte3 = code.get(++i).shortValue();
                    p.printf("%-10s%-17s%-5s%-5s%-5s\n", pc, mnemonic,
                            operandByte1, operandByte2, operandByte3);
                    break;
                case 4:
                    operandByte1 = code.get(++i).shortValue();
                    operandByte2 = code.get(++i).shortValue();
                    operandByte3 = code.get(++i).shortValue();
                    operandByte4 = code.get(++i).shortValue();
                    p.printf("%-10s%-17s%-5s%-5s%-5s%-5s\n", pc, mnemonic,
                            operandByte1, operandByte2, operandByte3, operandByte4);
                    break;
                case DYNAMIC: // variable length instructions
                    if (opcode == TABLESWITCH) {
                        int low, high;
                        pad = 4 - ((i + 1) % 4);
                        i = i + pad + 1;
                        deflt = intValue(code.get(i++), code.get(i++), code
                                .get(i++), code.get(i++));
                        low = intValue(code.get(i++), code.get(i++), code.get(i++),
                                code.get(i++));
                        high = intValue(code.get(i++), code.get(i++),
                                code.get(i++), code.get(i));
                        p.printf("%-10s%s { // %s to %s \n", pc, mnemonic, low,
                                high);
                        for (int idx = low; idx <= high; idx++) {
                            int offset = intValue(code.get(++i), code.get(++i),
                                    code.get(++i), code.get(++i));
                            p.printf("%-10s    %s:%s\n", "", idx, offset);
                        }
                        p.printf("%-10s    default: %s\n", "", deflt);
                        p.printf("%-10s}\n", "");
                    } else { // LOOKUPSWITCH
                        int nPairs;
                        pad = 4 - ((i + 1) % 4);
                        i = i + pad + 1;
                        deflt = intValue(code.get(i++), code.get(i++), code
                                .get(i++), code.get(i++));
                        nPairs = intValue(code.get(i++), code.get(i++), code
                                .get(i++), code.get(i));
                        p.printf("%-10s%s { \n", pc, mnemonic);
                        for (int idx = 0; idx < nPairs; idx++) {
                            int match = intValue(code.get(++i), code.get(++i), code
                                    .get(++i), code.get(++i));
                            int offset = intValue(code.get(++i), code.get(++i),
                                    code.get(++i), code.get(++i));
                            p.printf("%-10s    %s:%s\n", "", match, offset);
                        }
                        p.printf("%-10s    default: %s\n", "", deflt);
                        p.printf("%-10s}\n", "");
                    }
                }
            }
            p.println();
            p.printf("// Exception Table (%s Items)\n", exceptionTableLength);
            p.printf("%s    %s    %s    %s\n", "Start PC", "End PC", "Handler PC",
                    "Catch Type");
            p.printf("%s    %s    %s    %s\n", "--------", "------", "----------",
                    "----------");
            for (int i = 0; i < exceptionTable.size(); i++) {
                exceptionTable.get(i).writeToStdOut(p);
            }
            p.println();
            p.printf("// Attributes (%s Items)\n", attributesCount);
            for (int i = 0; i < attributes.size(); i++) {
                CLAttributeInfo attributeInfo = attributes.get(i);
                attributeInfo.writeToStdOut(p);
            }
            p.indentLeft();
            p.printf("}\n");
        }

    }

    /**
     * Representation of Exceptions attribute structure (JVM Spec Section 4.8.4).
     * This is a required method attribute.
     */

    class CLExceptionsAttribute extends CLAttributeInfo {

        /** Exceptions_attribute.number_of_exceptions item. */
        public int numberOfExceptions;

        /** Exceptions_attribute.exception_index_table item. */
        public ArrayList<Integer> exceptionIndexTable;

        /**
         * Construct a CLExceptionsAttribute object.
         * 
         * @param attributeNameIndex
         *            Exceptions_attribute.attribute_name_index item.
         * @param attributeLength
         *            Exceptions_attribute.attribute_length item.
         * @param numberOfExceptions
         *            Exceptions_attribute.number_of_exceptions item.
         * @param exceptionIndexTable
         *            Exceptions_attribute.exception_index_table item.
         */

        public CLExceptionsAttribute(int attributeNameIndex, long attributeLength,
                int numberOfExceptions, ArrayList<Integer> exceptionIndexTable) {
            super(attributeNameIndex, attributeLength);
            this.numberOfExceptions = numberOfExceptions;
            this.exceptionIndexTable = exceptionIndexTable;
        }

        /**
         * @inheritDoc
         */

        public void write(CLOutputStream out) throws IOException {
            super.write(out);
            out.writeShort(numberOfExceptions);
            for (int i = 0; i < exceptionIndexTable.size(); i++) {
                out.writeShort(exceptionIndexTable.get(i));
```

```java
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Exceptions {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Number of Exceptions: %s\n", numberOfExceptions);

        // Get rid of the [] in the toString() value of the
        // exceptionIndexTable ArrayList.
        String exceptions = exceptionIndexTable.toString();
        exceptions = exceptions.substring(1, exceptions.length() - 2);

        p.printf("Exception Index Table: %s\n", exceptions);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of classes table entry structure (JVM Spec Section 4.8.5).
 */

class CLInnerClassInfo {

    /** classes_table_entry.inner_class_info_index item. */
    public int innerClassInfoIndex;

    /** classes_table_entry.outer_class_info_index item. */
    public int outerClassInfoIndex;

    /** classes_table_entry.inner_name_index item. */
    public int innerNameIndex;

    /** classes_table_entry.inner_class_access_flags item. */
    public int innerClassAccessFlags;

    /**
     * Construct a CLInnerClassInfo object.
     * 
     * @param innerClassInfoIndex
     *            classes_table_entry.inner_class_info_index item.
     * @param outerClassInfoIndex
     *            classes_table_entry.outer_class_info_index item.
     * @param innerNameIndex
     *            classes_table_entry.inner_name_index item.
     * @param innerClassAccessFlags
     *            classes_table_entry.inner_class_access_flags item.
     */

    public CLInnerClassInfo(int innerClassInfoIndex, int outerClassInfoIndex,
            int innerNameIndex, int innerClassAccessFlags) {
        this.innerClassInfoIndex = innerClassInfoIndex;
        this.outerClassInfoIndex = outerClassInfoIndex;
        this.innerNameIndex = innerNameIndex;
        this.innerClassAccessFlags = innerClassAccessFlags;
    }

    /**
     * Write the contents of this object to the specified output stream.
     * 
     * @param out
     *            output stream.
     * @throws IOException
     *              if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(innerClassInfoIndex);
        out.writeShort(outerClassInfoIndex);
        out.writeShort(innerNameIndex);
        out.writeShort(innerClassAccessFlags);
    }

    /**
     * Write the contents of this object to STDOUT in a format similar to that
     * of javap.
     *
     * @param p
     *             for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("%-11s    %-17s    %-10s    %s\n", innerClassInfoIndex,
                outerClassInfoIndex, innerNameIndex, CLFile
                        .innerClassAccessFlagsToString(innerClassAccessFlags));
    }

}

/**
 * Representation of InnerClasses_attribute structure (JVM Spec Section 4.8.5).
 * This is a required class attribute.
 *
 * Note that this is just to register the inner classes with its parent class,
 * and does not create the classes, which can be done using CLEmitter.
 */

class CLInnerClassesAttribute extends CLAttributeInfo {

    /** InnerClasses_attribute.number_of_classes item. */
    public int numberOfClasses;

    /** InnerClasses_attribute.classes item. */
    public ArrayList<CLInnerClassInfo> classes;

    /**
     * Construct a CLInnerClassesAttribute object.
     *
     * @param attributeNameIndex
     *             InnerClasses_attribute.attribute_name_index item.
     * @param attributeLength
     *             InnerClasses_attribute.attribute_length item.
     * @param numberOfClasses
     *             InnerClasses_attribute.number_of_classes item.
     * @param classes
     *             InnerClasses_attribute.classes item.
     */

    public CLInnerClassesAttribute(int attributeNameIndex,
            long attributeLength, int numberOfClasses,
            ArrayList<CLInnerClassInfo> classes) {
        super(attributeNameIndex, attributeLength);
        this.numberOfClasses = numberOfClasses;
        this.classes = classes;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(numberOfClasses);
        for (int i = 0; i < classes.size(); i++) {
            classes.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("InnerClassesAttribute {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Number of Classes: %s\n", numberOfClasses);
        p.printf("%s    %s    %s    %s\n", "Class Index", "Outer Class Index",
                "Name Index", "Access Flags");
        p.printf("%s    %s    %s    %s\n", "-----------", "-----------------",
                "----------", "------------");
        for (int i = 0; i < classes.size(); i++) {
            classes.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of EnclosingMethod_attribute structure (JVM Spec Section
 * 4.8.6).
 */

class CLEnclosingMethodAttribute extends CLAttributeInfo {

    /** EnclosingMethod_attribute.class_index item. */
    public int classIndex;

    /** EnclosingMethod_attribute.method_index item. */
    public int methodIndex;

    /**
     * Construct a CLEnclosingMethodAttribute object.
     *
     * @param attributeNameIndex
     *            EnclosingMethod_attribute.attribute_name_index item.
     * @param attributeLength
     *            EnclosingMethod_attribute.attribute_length item.
     * @param classIndex
     *            EnclosingMethod_attribute.class_index item.
     * @param methodIndex
     *            EnclosingMethod_attribute.method_index item.
     */

    public CLEnclosingMethodAttribute(int attributeNameIndex,
            long attributeLength, int classIndex, int methodIndex) {
        super(attributeNameIndex, attributeLength);
        this.classIndex = classIndex;
        this.methodIndex = methodIndex;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(classIndex);
        out.writeShort(methodIndex);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("EnclosingMethod {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Class Index: %s\n", classIndex);
        p.printf("Method Index: %s\n", methodIndex);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of Synthetic_attribute structure (JVM Spec Section 4.8.7).
 * This is a required class, field, and method attribute.
 */

class CLSyntheticAttribute extends CLAttributeInfo {

    /**
     * Construct a CLSyntheticAttribute object.
     *
     * @param attributeNameIndex
     *            Synthetic_attribute.attribute_name_index item.
     * @param attributeLength
     *            Synthetic_attribute.attribute_length item.
     */

    public CLSyntheticAttribute(int attributeNameIndex, long attributeLength) {
        super(attributeNameIndex, attributeLength);
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Synthetic {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of Signature_attribute structure (JVM Spec Section 4.8.8).
 */

class CLSignatureAttribute extends CLAttributeInfo {

    /** Signature_attribute.signature_index item. */
    public int signatureIndex;
```

```java

    /**
     * Construct a CLSignatureAttribute object.
     *
     * @param attributeNameIndex
     *            Signature_attribute.attribute_name_index item.
     * @param attributeLength
     *            Signature_attribute.attribute_length item.
     * @param signatureIndex
     *            Signature_attribute.signature_index item.
     */

    public CLSignatureAttribute(int attributeNameIndex, long attributeLength,
            int signatureIndex) {
        super(attributeNameIndex, attributeLength);
        this.signatureIndex = signatureIndex;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(signatureIndex);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Signature {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Signature Index: %s\n", signatureIndex);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of SourceFile_attribute structure (JVM Spec Section 4.8.9).
 */

class CLSourceFileAttribute extends CLAttributeInfo {

    /** SourceFile_attribute.sourcefile_index item. */
    public int sourceFileIndex;

    /**
     * Construct a CLSourceFileAttribute object.
     *
     * @param attributeNameIndex
     *            SourceFile_attribute.attribute_name_index item.
     * @param attributeLength
     *            SourceFile_attribute.attribute_length item.
     * @param sourceFileIndex
     *            SourceFile_attribute.sourcefile_index item.
     */

    public CLSourceFileAttribute(int attributeNameIndex, long attributeLength,
            int sourceFileIndex) {
        super(attributeNameIndex, attributeLength);
        this.sourceFileIndex = sourceFileIndex;
    }

    /**
```

```
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(sourceFileIndex);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("SourceFile {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Source File Index: %s\n", sourceFileIndex);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of SourceDebugExtension_attribute structure (JVM Spec Section
 * 4.8.10).
 */

class CLSourceDebugExtensionAttribute extends CLAttributeInfo {

    /** SourceDebugExtension.debug_extension item. */
    public byte[] debugExtension;

    /**
     * Construct a CLSourceDebugExtensionAttribute object.
     *
     * @param attributeNameIndex
     *            SourceDebugExtension_attribute.attribute_name_index item.
     * @param attributeLength
     *            SourceDebugExtension_attribute.attribute_length item.
     * @param debugExtension
     *            SourceDebugExtension_attribute.debug_extension item.
     */

    public CLSourceDebugExtensionAttribute(int attributeNameIndex,
            long attributeLength, byte[] debugExtension) {
        super(attributeNameIndex, attributeLength);
        this.debugExtension = debugExtension;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        for (int i = 0; i < debugExtension.length; i++) {
            out.writeByte(debugExtension[i]);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("SourceDebugExtension {\n");
        p.indentRight();
        super.writeToStdOut(p);
```

```java
        p.printf("Debug Extension: %s\n", new String(debugExtension));
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of line_number_table entry structure (JVM Spec Section
 * 4.8.11).
 */

class CLLineNumberInfo {

    /** line_number_table_entry.start_pc item. */
    public int startPC;

    /** line_number_table_entry.line_number item. */
    public int lineNumber;

    /**
     * Construct a CLLineNumberInfo object.
     *
     * @param startPC
     *            line_number_table_entry.start_pc item.
     * @param lineNumber
     *            line_number_table_entry.line_number item.
     */

    public CLLineNumberInfo(int startPC, int lineNumber) {
        this.startPC = startPC;
        this.lineNumber = lineNumber;
    }

    /**
     * Write the contents of this object to the specified output stream.
     *
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(startPC);
        out.writeShort(lineNumber);
    }

    /**
     * Return true if this LineNumber_info object is "equal to" the specified
     * LineNumber_info object, false otherwise.
     *
     * @param obj
     *            the reference LineNumber_info object with which to compare.
     * @return true if this LineNumber_info object is "equal to" the specified
     *         LineNumber_info object, false otherwise.
     */

    public boolean equals(Object obj) {
        if (obj instanceof CLLineNumberInfo) {
            CLLineNumberInfo c = (CLLineNumberInfo) obj;
            if (c.lineNumber == lineNumber) {
                return true;
            }
        }
        return false;
    }

    /**
```

```
     * Write the contents of this object to STDOUT in a format similar to that
     * of javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("%-8s    %-11s\n", startPC, lineNumber);
    }

}

/**
 * Representation of LineNumberTable_attribute structure (JVM Spec Section
 * 4.8.11).
 */

class CLLineNumberTableAttribute extends CLAttributeInfo {

    /** LineNumberTable_attribute.line_number_table_length item. */
    public int lineNumberTableLength;

    /** LineNumberTable_attribute.line_number_table item. */
    public ArrayList<CLLineNumberInfo> lineNumberTable;

    /**
     * Construct a CLLineNumberTableAttribute object.
     *
     * @param attributeNameIndex
     *            LineNumberTable_attribute.attribute_name_index item.
     * @param attributeLength
     *            LineNumberTable_attribute.attribute_length item.
     * @param lineNumberTableLength
     *            LineNumberTable_attribute.line_number_table_length item.
     * @param lineNumberTable
     *            LineNumberTable_attribute.line_number_table item.
     */

    public CLLineNumberTableAttribute(int attributeNameIndex,
            long attributeLength, int lineNumberTableLength,
            ArrayList<CLLineNumberInfo> lineNumberTable) {
        super(attributeNameIndex, attributeLength);
        this.lineNumberTableLength = lineNumberTableLength;
        this.lineNumberTable = lineNumberTable;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(lineNumberTableLength);
        for (int i = 0; i < lineNumberTable.size(); i++) {
            lineNumberTable.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("LineNumberTable {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Line Number Table Length: %s\n", lineNumberTableLength);
        p.printf("%s    %s\n", "Start PC", "Line Number");
        p.printf("%s    %s\n", "--------", "-----------");
        for (int i = 0; i < lineNumberTable.size(); i++) {
            lineNumberTable.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of local_variable_table entry structure (JVM Spec Section
 * 4.8.12).
 */

class CLLocalVariableInfo {

    /** local_variable_table_entry.start_pc item. */
    public int startPC;

    /** local_variable_table_entry.length item. */
    public int length;

    /** local_variable_table_entry.name_index item. */
    public int nameIndex;

    /** local_variable_table_entry.descriptor_index item. */
    public int descriptorIndex;

    /** local_variable_table_entry.index item. */
    public int index;

    /**
     * Construct a CLLocalVariableInfo object.
     * 
     * @param startPC
     *            local_variable_table_entry.start_pc item.
     * @param length
     *            local_variable_table_entry.length item.
     * @param nameIndex
     *            local_variable_table_entry.name_index item.
     * @param descriptorIndex
     *            local_variable_table_entry.descriptor_index item.
     * @param index
     *            local_variable_table_entry.index item.
     */

    public CLLocalVariableInfo(int startPC, int length, int nameIndex,
            int descriptorIndex, int index) {
        this.startPC = startPC;
        this.length = length;
        this.nameIndex = nameIndex;
        this.descriptorIndex = descriptorIndex;
        this.index = index;
    }

    /**
     * Write the contents of this object to the specified output stream.
     * 
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(startPC);
        out.writeShort(length);
        out.writeShort(nameIndex);
        out.writeShort(descriptorIndex);
        out.writeShort(index);
    }

    /**
     * Write the contents of this object to STDOUT in a format similar to that
     * of javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("%-8s    %-6s    %-10s    %-16s    %-5s\n", startPC, length,
                nameIndex, descriptorIndex, index);
    }

}

/**
 * Representation of LocalVariableTable_attribute structure (JVM Spec Section
 * 4.8.12).
 */

class CLLocalVariableTableAttribute extends CLAttributeInfo {

    /**
     * LocalVariableTable_attribute.local_variable_table_length item.
     */
    public int localVariableTableLength;

    /** LocalVariableTable_attribute.local_variable_table item. */
    public ArrayList<CLLocalVariableInfo> localVariableTable;

    /**
     * Construct a CLLocalVariableTableAttribute object.
     *
     * @param attributeNameIndex
     *            LocalVariableTable_attribute.attribute_name_index item.
     * @param attributeLength
     *            LocalVariableTable_attribute.attribute_length item.
     * @param localVariableTableLength
     *            LocalVariableTable_attribute.local_variable_table_length item.
     * @param localVariableTable
     *            LocalVariableTable_attribute.local_variable_table item.
     */

    public CLLocalVariableTableAttribute(int attributeNameIndex,
            long attributeLength, int localVariableTableLength,
            ArrayList<CLLocalVariableInfo> localVariableTable) {
        super(attributeNameIndex, attributeLength);
        this.localVariableTableLength = localVariableTableLength;
        this.localVariableTable = localVariableTable;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(localVariableTableLength);
        for (int i = 0; i < localVariableTable.size(); i++) {
            localVariableTable.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
```

```java
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("LocalVariableTable {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Local Variable Table Length: %s\n", localVariableTableLength);
        p.printf("%s    %s    %s    %s    %s\n", "Start PC", "Length",
                "Name Index", "Descriptor Index", "Index");
        p.printf("%s    %s    %s    %s    %s\n", "--------", "------",
                "----------", "----------------", "-----");
        for (int i = 0; i < localVariableTable.size(); i++) {
            localVariableTable.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of local_variable_type_table entry structure (JVM Spec Section
 * 4.8.13).
 */

class CLLocalVariableTypeInfo {

    /** local_variable_type_table_entry.start_pc item. */
    public int startPC;

    /** local_variable_type_table_entry.length item. */
    public int length;

    /** local_variable_type_table_entry.name_index item. */
    public int nameIndex;

    /** local_variable_type_table_entry.descriptor_index item. */
    public int signatureIndex;

    /** local_variable_type_table_entry.index item. */
    public int index;

    /**
     * Construct a CLLocalVariableTypeInfo object.
     *
     * @param startPC
     *            local_variable_type_table_entry.start_pc item.
     * @param length
     *            local_variable_type_table_entry.length item.
     * @param nameIndex
     *            local_variable_type_table_entry.name_index item.
     * @param signatureIndex
     *            local_variable_type_table_entry.signature_index item.
     * @param index
     *            local_variable_type_table_entry.index item.
     */

    public CLLocalVariableTypeInfo(int startPC, int length, int nameIndex,
            int signatureIndex, int index) {
        this.startPC = startPC;
        this.length = length;
        this.nameIndex = nameIndex;
        this.signatureIndex = signatureIndex;
        this.index = index;
    }

    /**
     * Write the content of this object to the specified output stream.
     *
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(startPC);
        out.writeShort(length);
        out.writeShort(nameIndex);
        out.writeShort(signatureIndex);
        out.writeShort(index);
    }

    /**
     * Write the contents of this object to STDOUT in a format similar to that
     * of javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("%-8s    %-6s    %-10s    %-16s    %-5s\n", startPC, length,
                nameIndex, signatureIndex, index);
    }

}

/**
 * Representation of LocalVariableTypeTable_attribute structure (JVM Spec
 * Section 4.8.12).
 */

class CLLocalVariableTypeTableAttribute extends CLAttributeInfo {

    /**
     * LocalVariableTypeTable_attribute. local_variable_type_table_length item.
     */
    public int localVariableTypeTableLength;

    /**
     * LocalVariableTypeTable_attribute.local_variable_type_table item.
     */
    public ArrayList<CLLocalVariableTypeInfo> localVariableTypeTable;

    /**
     * Construct a CLLocalVariableTypeTableAttribute object.
     *
     * @param attributeNameIndex
     *            LocalVariableTypeTable_attribute.attribute_name_index item.
     * @param attributeLength
     *            LocalVariableTypeTable_attribute.attribute_length item.
     * @param localVariableTypeTableLength
     *            LocalVariableTypeTable_attribute.
     *            local_variable_type_table_length item.
     * @param localVariableTypeTable
     *            LocalVariableTypeTable_attribute.local_variable_type_table
     *            item.
     */

    public CLLocalVariableTypeTableAttribute(int attributeNameIndex,
            long attributeLength, int localVariableTypeTableLength,
            ArrayList<CLLocalVariableTypeInfo> localVariableTypeTable) {
        super(attributeNameIndex, attributeLength);
        this.localVariableTypeTableLength = localVariableTypeTableLength;
        this.localVariableTypeTable = localVariableTypeTable;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(localVariableTypeTableLength);
        for (int i = 0; i < localVariableTypeTable.size(); i++) {
            localVariableTypeTable.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("LocalVariableTypeTable {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Local Variable Type Table Length: %s\n",
                localVariableTypeTableLength);
        p.printf("%s    %s    %s    %s    %s\n", "Start PC", "Length",
                "Name Index", "Signature Index", "Index");
        p.printf("%s    %s    %s    %s    %s\n", "--------", "------",
                "----------", "---------------", "-----");
        for (int i = 0; i < localVariableTypeTable.size(); i++) {
            localVariableTypeTable.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of Deprecated_attribute structure (JVM Spec Section 4.8.14).
 */

class CLDeprecatedAttribute extends CLAttributeInfo {

    /**
     * Construct a CLDeprecatedAttribute object.
     * 
     * @param attributeNameIndex
     *            Deprecated_attribute.attribute_name_index item.
     * @param attributeLength
     *            Deprecated_attribute.attribute_length item.
     */

    public CLDeprecatedAttribute(int attributeNameIndex, long attributeLength) {
        super(attributeNameIndex, attributeLength);
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Deprecated {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of annotation structure (JVM Spec Section 4.8.15).
 */

class CLAnnotation {

    /** annotation.type_index item. */
    public int typeIndex;

    /** annotation.num_element_value_pairs item. */
    public int numElementValuePairs;

    /** annotation.element_value_pairs item. */
    public ArrayList<CLElementValuePair> elementValuePairs;

    /**
     * Construct a CLAnnotation object.
     *
     * @param typeIndex
     *            annotation.type_index item.
     * @param numElementValuePairs
     *            annotation.num_element_value_pairs item.
     * @param elementValuePairs
     *            annotation.element_value_pairs item.
     */

    public CLAnnotation(int typeIndex, int numElementValuePairs,
            ArrayList<CLElementValuePair> elementValuePairs) {
        this.typeIndex = typeIndex;
        this.numElementValuePairs = numElementValuePairs;
        this.elementValuePairs = elementValuePairs;
    }

    /**
     * Write the contents of this object to the specified output stream.
     *
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(typeIndex);
        out.writeShort(numElementValuePairs);
        for (int i = 0; i < elementValuePairs.size(); i++) {
            elementValuePairs.get(i).write(out);
        }
    }

    /**
     * Write the content of this object to STDOUT in a format similar to that of
     * javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("Annotation {\n");
        p.indentRight();
        p.printf("Type Index: %s\n", typeIndex);
        p.printf("Number of Element-Value Pairs: %s\n", numElementValuePairs);
        for (int i = 0; i < elementValuePairs.size(); i++) {
            elementValuePairs.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of element_value union (JVM Spec Section 4.8.15.1).
 */

class CLElementValue {

    /** element_value.tag item. */
    public short tag;

    /** element_value.const_value_index item. */
    public int constValueIndex;

    /** element_value.enum_const_value.type_name_index item. */
    public int typeNameIndex;

    /** element_value.enum_const_value.const_name_index item. */
    public int constNameIndex;

    /** element_value.class_info_index item. */
    public int classInfoIndex;

    /** element_value.annotation_value item. */
    public CLAnnotation annotationValue;

    /** element_value.array_value.numValues item. */
    public int numValues;

    /** element_value.array_value.values item. */
    public ArrayList<CLElementValue> values;

    /**
     * Construct a CLElementValue object.
     *
     * @param tag
     *             element_value.tag item.
     * @param constValueIndex
     *             element_value.const_value_index item.
     */

    public CLElementValue(short tag, int constValueIndex) {
        this.tag = tag;
        this.constValueIndex = constValueIndex;
    }

    /**
     * Construct a CLElementValue object.
     *
     * @param typeNameIndex
     *             element_value.type_name_index item.
     * @param constNameIndex
     *             element_value.const_name_index item.
     */

    public CLElementValue(int typeNameIndex, int constNameIndex) {
        this.tag = ELT_e;
        this.typeNameIndex = typeNameIndex;
        this.constNameIndex = constNameIndex;
    }

    /**
     * Construct a CLElementValue object.
     *
     * @param classInfoIndex
     *            element_value.class_info_index item.
     */

    public CLElementValue(int classInfoIndex) {
        this.tag = ELT_c;
        this.classInfoIndex = classInfoIndex;
    }

    /**
     * Construct a CLElementValue object.
     *
     * @param annotationValue
     *            element_value.annotation_value item.
     */

    public CLElementValue(CLAnnotation annotationValue) {
        this.tag = ELT_ANNOTATION;
        this.annotationValue = annotationValue;
    }

    /**
     * Construct a CLElementValue object.
     *
     * @param numValues
     *            element_value.num_values.
     * @param values
     *            element_value.values.
     */

    public CLElementValue(int numValues, ArrayList<CLElementValue> values) {
        this.tag = ELT_ARRAY;
        this.numValues = numValues;
        this.values = values;
    }

    /**
     * Write the contents of this object to the specified output stream.
     *
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeByte(tag);
        switch (tag) {
        case ELT_B:
        case ELT_C:
        case ELT_D:
        case ELT_F:
        case ELT_I:
        case ELT_J:
        case ELT_S:
        case ELT_Z:
        case ELT_s:
            out.writeInt(constValueIndex);
            break;
        case ELT_e:
            out.writeInt(typeNameIndex);
            out.writeInt(constNameIndex);
            break;
        case ELT_c:
            out.writeInt(classInfoIndex);
            break;
```

```java
        case ELT_ANNOTATION:
            annotationValue.write(out);
            break;
        case ELT_ARRAY:
            out.writeInt(numValues);
            for (int i = 0; i < numValues; i++) {
                values.get(i).write(out);
            }
        }
    }

    /**
     * Write the content of this object to STDOUT in a format similar to that of
     * javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("ElementValue {\n");
        p.indentRight();
        p.printf("Tag: %c\n", tag);
        switch (tag) {
        case ELT_B:
        case ELT_C:
        case ELT_D:
        case ELT_F:
        case ELT_I:
        case ELT_J:
        case ELT_S:
        case ELT_Z:
        case ELT_s:
            p.printf("Constant Value Index: %s\n", constValueIndex);
            break;
        case ELT_e:
            p.printf("Type Name Index: %s\n", typeNameIndex);
            p.printf("Constant Name Index: %s\n", constNameIndex);
            break;
        case ELT_c:
            p.printf("Class Info Index: %s\n", classInfoIndex);
            break;
        case ELT_ANNOTATION:
            annotationValue.writeToStdOut(p);
            break;
        case ELT_ARRAY:
            p.printf("Number of Values: %s\n", numValues);
            for (int i = 0; i < numValues; i++) {
                values.get(i).writeToStdOut(p);
            }
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of the element_value_pairs table entry (JVM Spec Section
 * 4.8.15).
 */

class CLElementValuePair {

    /** element_value_pairs_table_entry.element_name_index item. */
    public int elementNameIndex;

    /** element_value_pairs_table_entry.value item. */
    public CLElementValue value;

    /**
     * Construct a CLElementValuePair object.
     *
     * @param elementNameIndex
     *            element_value_pairs_table_entry.element_name_index item.
     * @param value
     *            element_value_pairs_table_entry.value item.
     */

    public CLElementValuePair(int elementNameIndex, CLElementValue value) {
        this.elementNameIndex = elementNameIndex;
        this.value = value;
    }

    /**
     * Write the contents of this object to the specified output stream.
     *
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(elementNameIndex);
        value.write(out);
    }

    /**
     * Write the content of this object to STDOUT in a format similar to that of
     * javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("ElementValuePair {\n");
        p.indentRight();
        p.printf("Element Name Index: %s\n", elementNameIndex);
        value.writeToStdOut(p);
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of RuntimeVisibleAnnotations_attribute structure (JVM Spec
 * Section 4.8.15).
 */

class CLRuntimeVisibleAnnotationsAttribute extends CLAttributeInfo {

    /** RuntimeVisibleAnnotations_attribute.num_annotations item. */
    public int numAnnotations;

    /** RuntimeVisibleAnnotations_attribute.annotations item. */
    public ArrayList<CLAnnotation> annotations;

    /**
     * Construct a CLRuntimeVisibleAnnotationsAttribute object.
     *
     * @param attributeNameIndex
     *            RuntimeVisibleAnnotations_attribute.attribute_name_index item.
     * @param attributeLength
     *            RuntimeVisibleAnnotations_attribute.attribute_length item.
     * @param numAnnotations
     *             RuntimeVisibleAnnotations_attribute.num_annotations item.
     * @param annotations
     *             RuntimeVisibleAnnotations_attribute.annotations item.
     */

    public CLRuntimeVisibleAnnotationsAttribute(int attributeNameIndex,
            long attributeLength, int numAnnotations,
            ArrayList<CLAnnotation> annotations) {
        super(attributeNameIndex, attributeLength);
        this.numAnnotations = numAnnotations;
        this.annotations = annotations;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(numAnnotations);
        for (int i = 0; i < annotations.size(); i++) {
            annotations.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("RuntimeVisibleAnnotations {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Number of Annotations: %s\n", numAnnotations);
        for (int i = 0; i < annotations.size(); i++) {
            annotations.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of RuntimeInvisibleAnnotations_attribute structure (JVM Spec
 * Section 4.8.16).
 */

class CLRuntimeInvisibleAnnotationsAttribute extends CLAttributeInfo {

    /**
     * RuntimeInvisibleAnnotations_attribute.num_annotations item.
     */
    public int numAnnotations;

    /** RuntimeInvisibleAnnotations_attribute.annotations item. */
    public ArrayList<CLAnnotation> annotations;

    /**
     * Construct a CLRuntimeInvisibleAnnotationsAttribute object.
     *
     * @param attributeNameIndex
     *             RuntimeInvisibleAnnotations_attribute.attribute_name_index
     *             item.
     * @param attributeLength
     *             RuntimeInvisibleAnnotations_attribute.attribute_length item.
     * @param numAnnotations
     *             RuntimeVisibleAnnotations_attribute.num_annotations item.
     * @param annotations
     *             RuntimeInvisibleAnnotations_attribute.annotations item.
     */

    public CLRuntimeInvisibleAnnotationsAttribute(int attributeNameIndex,
            long attributeLength, int numAnnotations,
            ArrayList<CLAnnotation> annotations) {
        super(attributeNameIndex, attributeLength);
        this.numAnnotations = numAnnotations;
        this.annotations = annotations;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeShort(numAnnotations);
        for (int i = 0; i < annotations.size(); i++) {
            annotations.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("RuntimeInvisibleAnnotations {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Number of Annotations: %s\n", numAnnotations);
        for (int i = 0; i < annotations.size(); i++) {
            annotations.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of parameter_annotations_table entry structure (JVM Spec
 * Section 4.8.17).
 */

class CLParameterAnnotationInfo {

    /** parameter_annotations_table_entry.num_annotations item. */
    public int numAnnotations;

    /** parameter_annotations_table_entry.annotations item. */
    public ArrayList<CLAnnotation> annotations;

    /**
     * Construct a ParameterAnnotationInfo object.
     *
     * @param numAnnotations
     *             parameter_annotations_table_entry.num_annotations item.
     * @param annotations
     *             parameter_annotations_table_entry.annotations item.
     */

    public CLParameterAnnotationInfo(int numAnnotations,
            ArrayList<CLAnnotation> annotations) {
        this.numAnnotations = numAnnotations;
        this.annotations = annotations;
    }

    /**
     * Write the contents of this object to the specified output stream.
     *
     * @param out
     *            output stream.
     * @throws IOException
     *             if an error occurs while writing.
     */

    public void write(CLOutputStream out) throws IOException {
        out.writeShort(numAnnotations);
        for (int i = 0; i < annotations.size(); i++) {
            annotations.get(i).write(out);
        }
    }

    /**
     * Write the content of this object to STDOUT in a format similar to that of
     * javap.
     *
     * @param p
     *            for pretty printing with indentation.
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("ParameterAnnotationInfo {\n");
        p.indentRight();
        p.printf("Number of Annotations: %s\n", numAnnotations);
        for (int i = 0; i < annotations.size(); i++) {
            annotations.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of RuntimeVisibleParameterAnnotations_attribute structure (JVM
 * Spec Section 4.8.17).
 */

class CLRuntimeVisibleParameterAnnotationsAttribute extends CLAttributeInfo {

    /**
     * RuntimeVisibleParameterAnnotations_attribute.num_parameters item.
     */
    public short numParameters;

    /**
     * RuntimeVisibleParameterAnnotations_attribute. parameter_annotations item.
     */
    public ArrayList<CLParameterAnnotationInfo> parameterAnnotations;

    /**
     * Construct a CLRuntimeVisibleParameterAnnotationsAttribute object.
     *
     * @param attributeNameIndex
     *            RuntimeVisibleParameterAnnotations_attribute.
     *            attribute_name_index item.
     * @param attributeLength
     *            RuntimeVisibleParameterAnnotations_attribute.attribute_length
     *            item.
     * @param numParameters
     *            RuntimeVisibleParameterAnnotations_attribute.num_parameters
     *            item.
     * @param parameterAnnotations
     *            RuntimeVisibleParameterAnnotations_attribute.
     *            parameter_annotations item.
     */

    public CLRuntimeVisibleParameterAnnotationsAttribute(
            int attributeNameIndex, long attributeLength, short numParameters,
            ArrayList<CLParameterAnnotationInfo> parameterAnnotations) {
        super(attributeNameIndex, attributeLength);
        this.numParameters = numParameters;
        this.parameterAnnotations = parameterAnnotations;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeByte(numParameters);
        for (int i = 0; i < parameterAnnotations.size(); i++) {
            parameterAnnotations.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("RuntimeVisibleParameterAnnotations {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Number of Parameters: %s\n", numParameters);
        for (int i = 0; i < parameterAnnotations.size(); i++) {
            parameterAnnotations.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}

/**
 * Representation of RuntimeInvisibleParameterAnnotations_attribute structure
 * (JVM Spec Section 4.8.18).
 */

class CLRuntimeInvisibleParameterAnnotationsAttribute extends CLAttributeInfo {

    /**
     * RuntimeInvisibleParameterAnnotations_attribute.num_parameters item.
     */
    public short numParameters;

    /**
     * RuntimeInvisibleParameterAnnotations_attribute. parameter_annotations
     * item.
     */
    public ArrayList<CLParameterAnnotationInfo> parameterAnnotations;

    /**
     * Construct a CLRuntimeInvisibleParameterAnnotationsAttribute object.
     *
     * @param attributeNameIndex
     *            RuntimeInvisibleParameterAnnotations_attribute.
     *            attribute_name_index item.
     * @param attributeLength
     *            RuntimeInvisibleParameterAnnotations_attribute.
     *            attribute_length item.
     * @param numParameters
     *            RuntimeInvisibleParameterAnnotations_attribute.num_parameters
```

```java
     *              item.
     * @param parameterAnnotations
     *              RuntimeInvisibleParameterAnnotations_attribute.
     *              parameter_annotations item.
     */

    public CLRuntimeInvisibleParameterAnnotationsAttribute(
            int attributeNameIndex, long attributeLength, short numParameters,
            ArrayList<CLParameterAnnotationInfo> parameterAnnotations) {
        super(attributeNameIndex, attributeLength);
        this.numParameters = numParameters;
        this.parameterAnnotations = parameterAnnotations;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeByte(numParameters);
        for (int i = 0; i < parameterAnnotations.size(); i++) {
            parameterAnnotations.get(i).write(out);
        }
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("RuntimeInvisibleParameterAnnotations {\n");
        p.indentRight();
        super.writeToStdOut(p);
        p.printf("Number of Parameters: %s\n", numParameters);
        for (int i = 0; i < parameterAnnotations.size(); i++) {
            parameterAnnotations.get(i).writeToStdOut(p);
        }
        p.indentLeft();
        p.printf("}\n");
    }

}
/**
 * Representation of AnnotationDefault_attribute structure (JVM Spec Section
 * 4.8.2).
 */

class CLAnnotationDefaultAttribute extends CLAttributeInfo {

    /** AnnotationDefault_attribute.defaultValue item. */
    public CLElementValue defaultValue;

    /**
     * Construct a CLAnnotationDefaultAttribute object.
     *
     * @param attributeNameIndex
     *              AnnotationDefault_attribute.attribute_name_index item.
     * @param attributeLength
     *              AnnotationDefault_attribute.attribute_length item.
     * @param defaultValue
     *              AnnotationDefault_attribute.defaultValue item.
     */

    public CLAnnotationDefaultAttribute(int attributeNameIndex,
            long attributeLength, CLElementValue defaultValue) {
        super(attributeNameIndex, attributeLength);
        this.defaultValue = defaultValue;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        defaultValue.write(out);
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("AnnotationDefault {\n");
        p.indentRight();
        super.writeToStdOut(p);
        defaultValue.writeToStdOut(p);
        p.indentLeft();
        p.printf("}\n");
    }

}
```