CLConstantPool.java

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

package jminusminus;

import java.io.IOException;
import java.util.ArrayList;

/**
 * Representation of a class' constant_pool table (JVM Spec Section 4.5). An
 * instance of CLConstantPool is created when a class is read using CLAbsorber
 * or constructed using CLEmitter.
 */

class CLConstantPool {

    // The fields below represent the members of the
    // constant_pool structure. See JVM Spec Section 4.5 for details.

    /** Index of the next item into the constant pool. */
    private int cpIndex;

    /** List of constant pool items. */
    private ArrayList<CLCPInfo> cpItems;

    /**
     * Look for the specified item in the constant pool. If it exists, return
     * its index. Otherwise, add the item to the constant pool and return its
     * (new) index.
     * 
     * @param cpInfo
     *            the item to find or add.
     * @return index constant pool index of the item.
     */

    private int findOrAdd(CLCPInfo cpInfo) {
        int index = find(cpInfo);
        if (index == -1) {
            index = addCPItem(cpInfo);
        }
        return index;
    }

    /**
     * Construct a CLConstantPool object.
     */

    public CLConstantPool() {
        cpIndex = 1;
        cpItems = new ArrayList<CLCPInfo>();
    }

    /**
     * Return the size of the constant pool.
     * 
     * @return the size of the constant pool.
     */

    public int size() {
        return cpItems.size();
    }

    /**
     * Return the constant pool index of the specified item if it exists in the
     * pool, -1 otherwise.
     * 
     * @param cpInfo
     *             item to find.
     * @return the constant pool index or -1.
     */

    public int find(CLCPInfo cpInfo) {
        int index = cpItems.indexOf(cpInfo);
        if (index != -1) {
            CLCPInfo c = cpItems.get(index);
            index = c.cpIndex;
        }
        return index;
    }

    /**
     * Return the constant pool item at the specified index, or null if the
     * index is invalid.
     *
     * @param i
     *             constant pool index.
     * @return the constant pool item or null.
     */

    public CLCPInfo cpItem(int i) {
        if (((i - 1) < 0) || ((i - 1) >= cpItems.size())) {
            return null;
        }
        return cpItems.get(i - 1);
    }

    /**
     * Add the specified (non null) item to the constant pool table and return
     * its index.
     *
     * @param cpInfo
     *             the item to add to the constant pool table.
     * @return constant pool index of the item.
     */

    public int addCPItem(CLCPInfo cpInfo) {
        int i = cpIndex++;
        cpInfo.cpIndex = i;
        cpItems.add(cpInfo);

        // long and double, with their lower and higher words,
        // are treated by JVM as two items in the constant pool. We
        // have a single representation for each, so we add a null as
        // a placeholder in the second slot.
        if ((cpInfo instanceof CLConstantLongInfo)
                || (cpInfo instanceof CLConstantDoubleInfo)) {
            cpIndex++;
            cpItems.add(null);
        }
        return i;
    }

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

    public void write(CLOutputStream out) throws IOException {
        for (int i = 0; i < cpItems.size(); i++) {
            CLCPInfo cpInfo = cpItems.get(i);
            if (cpInfo != null) {
                cpInfo.write(out);
            }
        }
    }

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

    public void writeToStdOut(PrettyPrinter p) {
        p.printf("// Constant Pool (%s Items)\n", size());
        p.printf("%-10s%-20s%s\n", "Index", "Item Type", "Content");
        p.printf("%-10s%-20s%s\n", "-----", "---------", "-------");
        for (int i = 1; i <= size(); i++) {
            CLCPInfo cpInfo = cpItem(i);
            if (cpInfo != null) {
                cpInfo.writeToStdOut(p);
            }
        }
    }

    // Following methods are helpers for creating singleton
    // instances of the different constant pool items -- classes extending
    // CLCPInfo objects in our representation. Each method
    // accepts as arguments the information needed for creating a
    // particular contant pool item, creates an instance
    // of the item, checks if the item already exists in the
    // constant pool, adds it if not, and returns the (possibly new) index
    // of the item.

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantClassInfo.
     * 
     * @param s
     *            class or interface name in internal form.
     * @return constant pool index.
     */

    public int constantClassInfo(String s) {
        CLCPInfo c = new CLConstantClassInfo(constantUtf8Info(s));
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantFieldRefInfo.
     * 
     * @param className
     *            class or interface name in internal form.
     * @param name
     *            name of the field.
     * @param type
     *            descriptor of the field.
     * @return constant pool index.
     */

    public int constantFieldRefInfo(String className, String name, String type) {
        CLCPInfo c = new CLConstantFieldRefInfo(constantClassInfo(className),
                constantNameAndTypeInfo(name, type));
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantMethodRefInfo.
     *
     * @param className
     *            class or interface name in internal form.
     * @param name
     *            name of the method.
     * @param type
     *            descriptor of the method.
     * @return constant pool index.
     */

    public int constantMethodRefInfo(String className, String name, String type)
{
        CLCPInfo c = new CLConstantMethodRefInfo(constantClassInfo(className),
                constantNameAndTypeInfo(name, type));
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantInterfaceMethodRefInfo.
     *
     * @param className
     *            class or interface name in internal form.
     * @param name
     *            name of the method.
     * @param type
     *            descriptor of the method.
     * @return constant pool index.
     */

    public int constantInterfaceMethodRefInfo(String className, String name,
            String type) {
        CLCPInfo c = new CLConstantInterfaceMethodRefInfo(
                constantClassInfo(className), constantNameAndTypeInfo(name,
                        type));
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantStringInfo.
     *
     * @param s
     *            the constant string value.
     * @return constant pool index.
     */

    public int constantStringInfo(String s) {
        CLCPInfo c = new CLConstantStringInfo(constantUtf8Info(s));
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantIntegerInfo.
     *
     * @param i
     *            the constant int value.
     * @return constant pool index.
     */

    public int constantIntegerInfo(int i) {
        CLCPInfo c = new CLConstantIntegerInfo(i);
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantFloatInfo.
     *
     * @param f
     *            the constant floating-point value.
     * @return constant pool index.
     */

    public int constantFloatInfo(float f) {
        CLCPInfo c = new CLConstantFloatInfo(f);
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantLongInfo.
     *
     * @param l
     *            the constant long value.
     * @return constant pool index.
     */

    public int constantLongInfo(long l) {
        CLCPInfo c = new CLConstantLongInfo(l);
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantDoubleInfo.
     *
     * @param d
     *            the constant double value.
     * @return constant pool index.
     */

    public int constantDoubleInfo(double d) {
        CLCPInfo c = new CLConstantDoubleInfo(d);
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantNameAndTypeInfo.
     *
     * @param name
     *            field or method name.
     * @param type
     *            field or method type descriptor.
     * @return constant pool index.
     */

    public int constantNameAndTypeInfo(String name, String type) {
        CLCPInfo c = new CLConstantNameAndTypeInfo(constantUtf8Info(name),
                constantUtf8Info(type));
        return findOrAdd(c);
    }

    /**
     * Return the constant pool index of a singleton instance of
     * CLConstantUtf8Info.
     *
     * @param s
     *            the constant string value.
     * @return constant pool index.
     */

    public int constantUtf8Info(String s) {
        CLCPInfo c = new CLConstantUtf8Info(s.getBytes());
        return findOrAdd(c);
    }
```

```
}
```