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

package jminusminus;

import java.io.DataOutputStream;
import java.io.IOException;
import static jminusminus.CLConstants.*;

/**
 * Representation of cp_info structure (JVM Spec Section 4.5). Classes
 * representing individual constant pool items inherit this class. Instances of
 * these classes are created and populated into the constant pool table when a
 * class is read using CLAbsorber or constructed using CLEmitter.
 */

abstract class CLCPInfo {

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

    /** Index of this object into the constant pool. */
    public int cpIndex;

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

    /**
     * Write the contents of this constant pool item 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);
    }

    /**
     * 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("%-10s", cpIndex);
    }

}

/**
 * Representation of CONSTANT_Class_info structure (JVM Spec Section 4.5.1).
 */

class CLConstantClassInfo extends CLCPInfo {

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

```

```
67      /**
68       * Construct a CLConstantClassInfo object.
69       *
70       * @param nameIndex
71       *            CONSTANT_Class_info.name_index item.
72       */
73
74      public CLConstantClassInfo(int nameIndex) {
75          super.tag = CONSTANT_Class;
76          this.nameIndex = nameIndex;
77      }
78
79      /**
80       * @inheritDoc
81       */
82
83      public void write(CLOutputStream out) throws IOException {
84          super.write(out);
85          out.writeShort(nameIndex);
86      }
87
88      /**
89       * @inheritDoc
90       */
91
92      public boolean equals(Object obj) {
93          if (obj instanceof CLConstantClassInfo) {
94              CLConstantClassInfo c = (CLConstantClassInfo) obj;
95              if (c.nameIndex == nameIndex) {
96                  return true;
97              }
98          }
99          return false;
100     }
101
102     /**
103      * @inheritDoc
104      */
105
106     public void writeToStdOut(PrettyPrinter p) {
107         super.writeToStdOut(p);
108         p.printf("%-20s%s\n", "Class", nameIndex);
109     }
110
111 }
112
113 /**
114  * Abstract super class of CONSTANT_Fieldref_info, CONSTANT_Methodref_info,
115  * CONSTANT_InterfaceMethodref_info structures (JVM Spec Section 4.5.2).
116  */
117
118 abstract class CLConstantMemberRefInfo extends CLCPInfo {
119
120     /** CONSTANT_Memberref_info.class_index item. */
121     public int classIndex;
122
123     /** CONSTANT_Memberref_info.name_and_type_index item. */
124     public int nameAndTypeIndex;
125
126     /**
127      * Construct a CLConstantMemberRefInfo object.
128      *
129      * @param classIndex
130      *            CONSTANT_Memberref_info.class_index item.
131      * @param nameAndTypeIndex
132      *            CONSTANT_Memberref_info.name_and_type_index item.
133      * @param tag
134      *            CONSTANT_Memberref_info.tag item.
135      */
```

```
136
137     protected CLConstantMemberRefInfo(int classIndex, int nameAndTypeIndex,
138             short tag) {
139         super.tag = tag;
140         this.classIndex = classIndex;
141         this.nameAndTypeIndex = nameAndTypeIndex;
142     }
143
144     /**
145      * @inheritDoc
146      */
147
148     public void write(CLOutputStream out) throws IOException {
149         super.write(out);
150         out.writeShort(classIndex);
151         out.writeShort(nameAndTypeIndex);
152     }
153
154     /**
155      * @inheritDoc
156      */
157
158     public boolean equals(Object obj) {
159         if (obj instanceof CLConstantMemberRefInfo) {
160             CLConstantMemberRefInfo c = (CLConstantMemberRefInfo) obj;
161             if ((c.tag == tag) && (c.classIndex == classIndex)
162                     && (c.nameAndTypeIndex == nameAndTypeIndex)) {
163                 return true;
164             }
165         }
166         return false;
167     }
168
169 }
170
171 /**
172  * Representation of CONSTANT_Fieldref_info structure (JVM Spec Section 4.5.2).
173  */
174
175 class CLConstantFieldRefInfo extends CLConstantMemberRefInfo {
176
177     /**
178      * Construct a CLConstantFieldRefInfo object.
179      *
180      * @param classIndex
181      *            CONSTANT_Fieldref_info.class_index item.
182      * @param nameAndTypeIndex
183      *            CONSTANT_Fieldref_info.name_and_type_index item.
184      */
185
186     public CLConstantFieldRefInfo(int classIndex, int nameAndTypeIndex) {
187         super(classIndex, nameAndTypeIndex, CONSTANT_Fieldref);
188     }
189
190     /**
191      * @inheritDoc
192      */
193
194     public void writeToStdOut(PrettyPrinter p) {
195         super.writeToStdOut(p);
196         p.printf("%-20s%-8s%-8s\n", "FieldRef", classIndex, nameAndTypeIndex);
197     }
198
199 }
200
201 /**
202  * Representation of CONSTANT_Methodref_info structure (JVM Spec Section 4.5.2).
203  */
204
```

```
205 class CLConstantMethodRefInfo extends CLConstantMemberRefInfo {
206
207     /**
208      * Construct a CLConstantMethodRefInfo object.
209      *
210      * @param classIndex
211      *            CONSTANT_Methodref_info.class_index item.
212      * @param nameAndTypeIndex
213      *            CONSTANT_Methodref_info.name_and_type_index item.
214      */
215
216     public CLConstantMethodRefInfo(int classIndex, int nameAndTypeIndex) {
217         super(classIndex, nameAndTypeIndex, CONSTANT_Methodref);
218     }
219
220     /**
221      * @inheritDoc
222      */
223
224     public void writeToStdOut(PrettyPrinter p) {
225         super.writeToStdOut(p);
226         p.printf("%-20s%-8s%-8s\n", "MethodRef", classIndex, nameAndTypeIndex);
227     }
228
229 }
230
231 /**
232  * Representation of CONSTANT_InterfaceMethodref_info structure (JVM Spec
233  * Section 4.5.2).
234  */
235
236 class CLConstantInterfaceMethodRefInfo extends CLConstantMemberRefInfo {
237
238     /**
239      * Construct a CLConstantInterfaceMethodRefInfo object.
240      *
241      * @param classIndex
242      *            CONSTANT_InterfaceMethodref_info.class_index item.
243      * @param nameAndTypeIndex
244      *            CONSTANT_InterfaceMethodref_info.name_and_type_index item.
245      */
246
247     public CLConstantInterfaceMethodRefInfo(int classIndex, int nameAndTypeIndex)
248 {
249         super(classIndex, nameAndTypeIndex, CONSTANT_InterfaceMethodref);
250     }
251
252     /**
253      * @inheritDoc
254      */
255
256     public void writeToStdOut(PrettyPrinter p) {
257         super.writeToStdOut(p);
258         p.printf("%-20s%-8s%-8s\n", "InterfaceMethodRef", classIndex,
259                 nameAndTypeIndex);
260     }
261
262 }
263
264 /**
265  * Representation of CONSTANT_String_info structure (JVM Spec Section 4.5.3).
266  */
267
268 class CLConstantStringInfo extends CLCPInfo {
269
270     /** CONSTANT_String_info.string_index item. */
271     public int stringIndex;
272
273     /**
```

```
273      * Construct a CLConstantStringInfo object.
274      *
275      * @param stringIndex
276      *            CONSTANT_String_info.string_index item.
277      */
278
279     public CLConstantStringInfo(int stringIndex) {
280         super.tag = CONSTANT_String;
281         this.stringIndex = stringIndex;
282     }
283
284     /**
285      * @inheritDoc
286      */
287
288     public void write(CLOutputStream out) throws IOException {
289         super.write(out);
290         out.writeShort(stringIndex);
291     }
292
293     /**
294      * @inheritDoc
295      */
296
297     public boolean equals(Object obj) {
298         if (obj instanceof CLConstantStringInfo) {
299             CLConstantStringInfo c = (CLConstantStringInfo) obj;
300             if (c.stringIndex == stringIndex) {
301                 return true;
302             }
303         }
304         return false;
305     }
306
307     /**
308      * @inheritDoc
309      */
310
311     public void writeToStdOut(PrettyPrinter p) {
312         super.writeToStdOut(p);
313         p.printf("%-20s%s\n", "String", stringIndex);
314     }
315
316 }
317
318 /**
319  * Representation of CONSTANT_Integer_info structure (JVM Spec Section 4.5.4).
320  */
321
322 class CLConstantIntegerInfo extends CLCPInfo {
323
324     /** The int number. */
325     public int i;
326
327     /**
328      * Construct a CLConstantIntegerInfo object.
329      *
330      * @param i
331      *            the int number.
332      */
333
334     public CLConstantIntegerInfo(int i) {
335         super.tag = CONSTANT_Integer;
336         this.i = i;
337     }
338
339     /**
340      * Return CONSTANT_Integer_info.bytes item.
341      *
```

```
342      * @return CONSTANT_Integer_info.bytes item.
343      */
344
345     public short[] bytes() {
346         short[] s = new short[4];
347         short mask = 0xFF;
348         int k = i;
349         for (int j = 0; j < 4; j++) {
350             s[3 - j] = (short) (k & mask);
351             k >>>= 8;
352         }
353         return s;
354     }
355
356     /**
357      * @inheritDoc
358      */
359
360     public void write(CLOutputStream out) throws IOException {
361         super.write(out);
362
363         // out is cast to DataOutputStream to resolve the
364         // writeInt()
365         // ambiguity
366         ((DataOutputStream) out).writeInt(i);
367     }
368
369     /**
370      * @inheritDoc
371      */
372
373     public boolean equals(Object obj) {
374         if (obj instanceof CLConstantIntegerInfo) {
375             CLConstantIntegerInfo c = (CLConstantIntegerInfo) obj;
376             if (c.i == i) {
377                 return true;
378             }
379         }
380         return false;
381     }
382
383     /**
384      * @inheritDoc
385      */
386
387     public void writeToStdOut(PrettyPrinter p) {
388         super.writeToStdOut(p);
389         p.printf("%-20s%s\n", "Integer", i);
390     }
391
392 }
393
394 /**
395  * Representation of CONSTANT_Float_info structure (JVM Spec Section 4.5.4).
396  */
397
398 class CLConstantFloatInfo extends CLCPInfo {
399
400     /** The floating-point number. */
401     public float f;
402
403     /**
404      * Construct a CLConstantFloatInfo object.
405      *
406      * @param f
407      *            the floating-point number.
408      */
409
410     public CLConstantFloatInfo(float f) {
```

```
411         super.tag = CONSTANT_Float;
412         this.f = f;
413     }
414 
415     /**
416      * Return CONSTANT_Float_info.bytes item.
417      *
418      * @return CONSTANT_Float_info.bytes item.
419      */
420 
421     public short[] bytes() {
422         short[] s = new short[4];
423         short mask = 0xFF;
424         int i = Float.floatToIntBits(f);
425         for (int j = 0; j < 4; j++) {
426             s[3 - j] = (short) (i & mask);
427             i >>>= 8;
428         }
429         return s;
430     }
431 
432     /**
433      * @inheritDoc
434      */
435 
436     public void write(CLOutputStream out) throws IOException {
437         super.write(out);
438         out.writeFloat(f);
439     }
440 
441     /**
442      * @inheritDoc
443      */
444 
445     public boolean equals(Object obj) {
446         if (obj instanceof CLConstantFloatInfo) {
447             CLConstantFloatInfo c = (CLConstantFloatInfo) obj;
448             if (c.f == f) {
449                 return true;
450             }
451         }
452         return false;
453     }
454 
455     /**
456      * @inheritDoc
457      */
458 
459     public void writeToStdOut(PrettyPrinter p) {
460         super.writeToStdOut(p);
461         p.printf("%-20s%s\n", "Float", f);
462     }
463 
464 }
465 
466 /**
467  * Representation of CONSTANT_Long_info structure (JVM Spec Section 4.5.5).
468  */
469 
470 class CLConstantLongInfo extends CLCPInfo {
471 
472     /** The long number. */
473     public long l;
474 
475     /**
476      * Return the 8 bytes of the long value.
477      *
478      * @return the 8 bytes of the long value.
479      */
```

```
480
481     private short[] bytes() {
482         short[] s = new short[8];
483         short mask = 0xFF;
484         long k = l;
485         for (int j = 0; j < 8; j++) {
486             s[7 - j] = (short) (k & mask);
487             k >>>= 8;
488         }
489         return s;
490     }
491
492     /**
493      * Construct a CLConstantLongInfo object.
494      *
495      * @param l
496      *            the long number.
497      */
498
499     public CLConstantLongInfo(long l) {
500         super.tag = CONSTANT_Long;
501         this.l = l;
502     }
503
504     /**
505      * Return CONSTANT_Long_info.low_bytes item.
506      *
507      * @return CONSTANT_Long_info.low_bytes item.
508      */
509
510     public short[] lowBytes() {
511         short[] s = bytes();
512         short[] l = new short[4];
513         l[0] = s[4];
514         l[1] = s[5];
515         l[2] = s[6];
516         l[3] = s[7];
517         return l;
518     }
519
520     /**
521      * Return CONSTANT_Long_info.high_bytes item.
522      *
523      * @return CONSTANT_Long_info.high_bytes item.
524      */
525
526     public short[] highBytes() {
527         short[] s = bytes();
528         short[] h = new short[4];
529         h[0] = s[0];
530         h[1] = s[1];
531         h[2] = s[2];
532         h[3] = s[3];
533         return h;
534     }
535
536     /**
537      * @inheritDoc
538      */
539
540     public void write(CLOutputStream out) throws IOException {
541         super.write(out);
542         out.writeLong(l);
543     }
544
545     /**
546      * @inheritDoc
547      */
548
```

```
549     public boolean equals(Object obj) {
550         if (obj instanceof CLConstantLongInfo) {
551             CLConstantLongInfo c = (CLConstantLongInfo) obj;
552             if (c.l == l) {
553                 return true;
554             }
555         }
556         return false;
557     }
558 
559     /**
560      * @inheritDoc
561      */
562 
563     public void writeToStdOut(PrettyPrinter p) {
564         super.writeToStdOut(p);
565         p.printf("%-20s%s\n", "Long", l);
566     }
567 
568 }
569 
570 /**
571  * Representation of CONSTANT_Double_info structure (JVM Spec Section 4.5.5).
572  */
573 
574 class CLConstantDoubleInfo extends CLCPInfo {
575 
576     /** The double precision floating-point number. */
577     public double d;
578 
579     /**
580      * Return the 8 bytes of the double precision floating-point value.
581      *
582      * @return the 8 bytes of the double precision floating-point value.
583      */
584 
585     private short[] bytes() {
586         short[] s = new short[8];
587         short mask = 0xFF;
588         long l = Double.doubleToLongBits(d);
589         for (int j = 0; j < 8; j++) {
590             s[7 - j] = (short) (l & mask);
591             l >>>= 8;
592         }
593         return s;
594     }
595 
596     /**
597      * Construct a CLConstantDoubleInfo object.
598      *
599      * @param d
600      *            the double precision floating-point number.
601      */
602 
603     public CLConstantDoubleInfo(double d) {
604         super.tag = CONSTANT_Double;
605         this.d = d;
606     }
607 
608     /**
609      * Return CONSTANT_Double_info.low_bytes item.
610      *
611      * @return CONSTANT_Double_info.low_bytes item.
612      */
613 
614     public short[] lowBytes() {
615         short[] s = bytes();
616         short[] l = new short[4];
617         l[0] = s[4];
```

```
618         l[1] = s[5];
619         l[2] = s[6];
620         l[3] = s[7];
621         return l;
622     }
623
624     /**
625      * Return CONSTANT_Double_info.high_bytes item.
626      *
627      * @return CONSTANT_Double_info.high_bytes item.
628      */
629
630     public short[] highBytes() {
631         short[] s = bytes();
632         short[] h = new short[4];
633         h[0] = s[0];
634         h[1] = s[1];
635         h[2] = s[2];
636         h[3] = s[3];
637         return h;
638     }
639
640     /**
641      * @inheritDoc
642      */
643
644     public void write(CLOutputStream out) throws IOException {
645         super.write(out);
646         out.writeDouble(d);
647     }
648
649     /**
650      * @inheritDoc
651      */
652
653     public boolean equals(Object obj) {
654         if (obj instanceof CLConstantDoubleInfo) {
655             CLConstantDoubleInfo c = (CLConstantDoubleInfo) obj;
656             if (c.d == d) {
657                 return true;
658             }
659         }
660         return false;
661     }
662
663     /**
664      * @inheritDoc
665      */
666
667     public void writeToStdOut(PrettyPrinter p) {
668         super.writeToStdOut(p);
669         p.printf("%-20s%s\n", "Double", d);
670     }
671
672 }
673
674 /**
675  * Representation of CONSTANT_NameAndType_info structure (JVM Spec Section
676  * 4.5.6).
677  */
678
679 class CLConstantNameAndTypeInfo extends CLCPInfo {
680
681     /** CONSTANT_NameAndType_info.name_index item. */
682     public int nameIndex;
683
684     /** CONSTANT_NameAndType_info.descriptor_index item. */
685     public int descriptorIndex;
686
```

```
687     /**
688      * Construct a CLConstantNameAndTypeInfo object.
689      *
690      * @param nameIndex
691      *            CONSTANT_NameAndType_info.name_index item.
692      * @param descriptorIndex
693      *            CONSTANT_NameAndType_info.descriptor_index item.
694      */
695
696     public CLConstantNameAndTypeInfo(int nameIndex, int descriptorIndex) {
697         super.tag = CONSTANT_NameAndType;
698         this.nameIndex = nameIndex;
699         this.descriptorIndex = descriptorIndex;
700     }
701
702     /**
703      * @inheritDoc
704      */
705
706     public void write(CLOutputStream out) throws IOException {
707         super.write(out);
708         out.writeShort(nameIndex);
709         out.writeShort(descriptorIndex);
710     }
711
712     /**
713      * @inheritDoc
714      */
715
716     public boolean equals(Object obj) {
717         if (obj instanceof CLConstantNameAndTypeInfo) {
718             CLConstantNameAndTypeInfo c = (CLConstantNameAndTypeInfo) obj;
719             if ((c.nameIndex == nameIndex)
720                     && (c.descriptorIndex == descriptorIndex)) {
721                 return true;
722             }
723         }
724         return false;
725     }
726
727     /**
728      * @inheritDoc
729      */
730
731     public void writeToStdOut(PrettyPrinter p) {
732         super.writeToStdOut(p);
733         p.printf("%-20s%-8s%-8s\n", "NameAndType", nameIndex, descriptorIndex);
734     }
735
736 }
737
738 /**
739  * Representation of CONSTANT_Utf8_info structure (JVM Spec Section 4.5.7).
740  */
741
742 class CLConstantUtf8Info extends CLCPInfo {
743
744     /** CONSTANT_Utf8_info.bytes item. */
745     public byte[] b;
746
747     /**
748      * Construct a CLConstantUtf8Info object.
749      *
750      * @param b
751      *            a constant string value.
752      */
753
754     public CLConstantUtf8Info(byte[] b) {
755         super.tag = CONSTANT_Utf8;
```

```
        this.b = b;
    }

    /**
     * Return CONSTANT_Utf8_info.length item.
     *
     * @return CONSTANT_Utf8_info.length item.
     */

    public int length() {
        return b.length;
    }

    /**
     * @inheritDoc
     */

    public void write(CLOutputStream out) throws IOException {
        super.write(out);
        out.writeUTF(new String(b));
    }

    /**
     * @inheritDoc
     */

    public boolean equals(Object obj) {
        if (obj instanceof CLConstantUtf8Info) {
            CLConstantUtf8Info c = (CLConstantUtf8Info) obj;
            if (new String(b).equals(new String(c.b))) {
                return true;
            }
        }
        return false;
    }

    /**
     * @inheritDoc
     */

    public void writeToStdOut(PrettyPrinter p) {
        super.writeToStdOut(p);
        p.printf("%-20s%s\n", "Utf8", new String(b));
    }

}
```