

## CLFile.java

```
1  // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3  package jminusminus;
4
5  import java.io.IOException;
6  import java.util.ArrayList;
7  import static jminusminus.CLConstants.*;
8
9  /**
10   * Representation of the ClassFile structure (JVM Spec Section 4.2). An instance
11   * of CLFile is created when a class is read using CLAbsorber or constructed
12   * using CLEmitter.
13   *
14   * We have our own representation and don't use java.lang.Class because Java
15   * does not offer an interface to programmatically create a class file in memory
16   * other than creating it in one shot from a byte stream.
17   */
18
19  class CLFile {
20
21      // The fields below represent the members of the ClassFile
22      // structure. See JVM Spec Section 4.2 for details.
23
24      /** ClassFile.magic item. */
25      public long magic; // 0xCAFEBAE
26
27      /** ClassFile.minor_version item. */
28      public int minorVersion;
29
30      /** ClassFile.major_version item. */
31      public int majorVersion;
32
33      /** ClassFile.constant_pool_count item. */
34      public int constantPoolCount;
35
36      /** ClassFile.constant_pool item. */
37      public CLConstantPool constantPool;
38
39      /** ClassFile.access_flags item. */
40      public int accessFlags;
41
42      /** ClassFile.this_class item. */
43      public int thisClass;
44
45      /** ClassFile.super_class item. */
46      public int superClass;
47
48      /** ClassFile.interfaces_count item. */
49      public int interfacesCount;
50
51      /** ClassFile.interfaces item. */
52      public ArrayList<Integer> interfaces;
53
54      /** ClassFile.fields_count item. */
55      public int fieldsCount;
56
57      /** ClassFile.fields item. */
58      public ArrayList<CLFieldInfo> fields;
59
60      /** ClassFile.methods_count item. */
61      public int methodsCount;
62
63      /** ClassFile.methods item. */
64      public ArrayList<CLMethodInfo> methods;
65
66      /** ClassFile.attributes_count item. */
```

```

67     public int attributesCount;
68
69     /** ClassFile.attributes item. */
70     public ArrayList<CLAttributeInfo> attributes;
71
72     /**
73      * Write the contents of this class to the specified output stream.
74      *
75      * @param out
76      *         output stream.
77      * @throws IOException
78      *         if an error occurs while writing.
79      */
80
81     public void write(CLOutputStream out) throws IOException {
82         out.writeInt(magic);
83         out.writeShort(minorVersion);
84         out.writeShort(majorVersion);
85         out.writeShort(constantPoolCount);
86         constantPool.write(out);
87         out.writeShort(accessFlags);
88         out.writeShort(thisClass);
89         out.writeShort(superClass);
90         out.writeShort(interfacesCount);
91         for (int i = 0; i < interfaces.size(); i++) {
92             Integer index = interfaces.get(i);
93             out.writeShort(index.intValue());
94         }
95         out.writeShort(fieldsCount);
96         for (int i = 0; i < fields.size(); i++) {
97             CLMemberInfo fieldInfo = fields.get(i);
98             if (fieldInfo != null) {
99                 fieldInfo.write(out);
100             }
101         }
102         out.writeShort(methodsCount);
103         for (int i = 0; i < methods.size(); i++) {
104             CLMemberInfo methodInfo = methods.get(i);
105             if (methodInfo != null) {
106                 methodInfo.write(out);
107             }
108         }
109         out.writeShort(attributesCount);
110         for (int i = 0; i < attributes.size(); i++) {
111             CLAttributeInfo attributeInfo = attributes.get(i);
112             if (attributeInfo != null) {
113                 attributeInfo.write(out);
114             }
115         }
116     }
117
118     /**
119      * Write the contents of the class file to STDOUT in a format similar to
120      * that of javap.
121      */
122
123     public void writeToStdOut() {
124         PrettyPrinter p = new PrettyPrinter();
125         p.printf("Magic Number: %x\n", magic);
126         p.printf("Minor Version: %d\n", minorVersion);
127         p.printf("Major Version: %d\n", majorVersion);
128         p.printf("Access Flags: %s\n", classAccessFlagsToString(accessFlags));
129         p.println();
130         constantPool.writeToStdOut(p);
131         p.println();
132         p.printf("This Class Index: %d\n", thisClass);
133         p.printf("Super Class Index: %d\n", superClass);
134         p.println();
135         p.printf("// Fields (%d Items)\n", fieldsCount);

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

136     for (int i = 0; i < fields.size(); i++) {
137         CLMemberInfo fieldInfo = fields.get(i);
138         if (fieldInfo != null) {
139             fieldInfo.writeToStdOut(p);
140         }
141     }
142     p.println();
143     p.printf("// Methods (%d Items)\n", methodsCount);
144     for (int i = 0; i < methods.size(); i++) {
145         CLMemberInfo methodInfo = methods.get(i);
146         if (methodInfo != null) {
147             methodInfo.writeToStdOut(p);
148         }
149     }
150     p.println();
151     p.printf("// Attributes (%d Items)\n", attributesCount);
152     for (int i = 0; i < attributes.size(); i++) {
153         CLAttributeInfo attributeInfo = attributes.get(i);
154         attributeInfo.writeToStdOut(p);
155     }
156 }
157
158 /**
159  * Return (as a string) the class access permissions and properties
160  * contained in the specified mask of flags.
161  *
162  * @param accessFlags
163  *         mask of access flags.
164  * @return a string identifying class access permissions and properties.
165  */
166
167 public static String classAccessFlagsToString(int accessFlags) {
168     StringBuffer b = new StringBuffer();
169     if ((accessFlags & ACC_PUBLIC) != 0) {
170         b.append("public ");
171     }
172     if ((accessFlags & ACC_FINAL) != 0) {
173         b.append("final ");
174     }
175     if ((accessFlags & ACC_SUPER) != 0) {
176         b.append("super ");
177     }
178     if ((accessFlags & ACC_INTERFACE) != 0) {
179         b.append("interface ");
180     }
181     if ((accessFlags & ACC_ABSTRACT) != 0) {
182         b.append("abstract ");
183     }
184     if ((accessFlags & ACC_SYNTHETIC) != 0) {
185         b.append("synthetic ");
186     }
187     if ((accessFlags & ACC_ANNOTATION) != 0) {
188         b.append("annotation ");
189     }
190     if ((accessFlags & ACC_ENUM) != 0) {
191         b.append("enum ");
192     }
193     return b.toString().trim();
194 }
195
196 /**
197  * Return (as a string) the inner class access permissions and properties
198  * contained in the specified mask of flags.
199  *
200  * @param accessFlags
201  *         mask of access flags.
202  * @return a string identifying the inner class access permissions and
203  *         properties.
204  */

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

205
206 public static String innerClassAccessFlagsToString(int accessFlags) {
207     StringBuffer b = new StringBuffer();
208     if ((accessFlags & ACC_PUBLIC) != 0) {
209         b.append("public ");
210     }
211     if ((accessFlags & ACC_PRIVATE) != 0) {
212         b.append("private ");
213     }
214     if ((accessFlags & ACC_PROTECTED) != 0) {
215         b.append("protected ");
216     }
217     if ((accessFlags & ACC_STATIC) != 0) {
218         b.append("static ");
219     }
220     if ((accessFlags & ACC_FINAL) != 0) {
221         b.append("final ");
222     }
223     if ((accessFlags & ACC_INTERFACE) != 0) {
224         b.append("interface ");
225     }
226     if ((accessFlags & ACC_ABSTRACT) != 0) {
227         b.append("abstract ");
228     }
229     if ((accessFlags & ACC_SYNTHETIC) != 0) {
230         b.append("synthetic ");
231     }
232     if ((accessFlags & ACC_ANNOTATION) != 0) {
233         b.append("annotation ");
234     }
235     if ((accessFlags & ACC_ENUM) != 0) {
236         b.append("enum ");
237     }
238     return b.toString().trim();
239 }
240
241 /**
242  * Return (as a string) the field access permissions and properties
243  * contained in the specified mask of flags.
244  *
245  * @param accessFlags
246  *         mask of access flags.
247  * @return a string identifying the field access permissions and properties.
248  */
249
250 public static String fieldAccessFlagsToString(int accessFlags) {
251     StringBuffer b = new StringBuffer();
252     if ((accessFlags & ACC_PUBLIC) != 0) {
253         b.append("public ");
254     }
255     if ((accessFlags & ACC_PRIVATE) != 0) {
256         b.append("private ");
257     }
258     if ((accessFlags & ACC_PROTECTED) != 0) {
259         b.append("protected ");
260     }
261     if ((accessFlags & ACC_STATIC) != 0) {
262         b.append("static ");
263     }
264     if ((accessFlags & ACC_FINAL) != 0) {
265         b.append("final ");
266     }
267     if ((accessFlags & ACC_VOLATILE) != 0) {
268         b.append("volatile ");
269     }
270     if ((accessFlags & ACC_TRANSIENT) != 0) {
271         b.append("transient ");
272     }
273     if ((accessFlags & ACC_NATIVE) != 0) {

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

274         b.append("native ");
275     }
276     if ((accessFlags & ACC_SYNTHETIC) != 0) {
277         b.append("synthetic ");
278     }
279     if ((accessFlags & ACC_ENUM) != 0) {
280         b.append("enum ");
281     }
282     return b.toString().trim();
283 }
284
285 /**
286  * Return (as a string) the method access permissions and properties
287  * contained in the specified mask of flags.
288  *
289  * @param accessFlags
290  *         mask of access flags.
291  * @return a string identifying the method access permissions and
292  *         properties.
293  */
294
295 public static String methodAccessFlagsToString(int accessFlags) {
296     StringBuffer b = new StringBuffer();
297     if ((accessFlags & ACC_PUBLIC) != 0) {
298         b.append("public ");
299     }
300     if ((accessFlags & ACC_PRIVATE) != 0) {
301         b.append("private ");
302     }
303     if ((accessFlags & ACC_PROTECTED) != 0) {
304         b.append("protected ");
305     }
306     if ((accessFlags & ACC_STATIC) != 0) {
307         b.append("static ");
308     }
309     if ((accessFlags & ACC_FINAL) != 0) {
310         b.append("final ");
311     }
312     if ((accessFlags & ACC_SYNCHRONIZED) != 0) {
313         b.append("synchronized ");
314     }
315     if ((accessFlags & ACC_BRIDGE) != 0) {
316         b.append("bridge ");
317     }
318     if ((accessFlags & ACC_VARARGS) != 0) {
319         b.append("varargs ");
320     }
321     if ((accessFlags & ACC_NATIVE) != 0) {
322         b.append("native ");
323     }
324     if ((accessFlags & ACC_ABSTRACT) != 0) {
325         b.append("abstract ");
326     }
327     if ((accessFlags & ACC_STRICT) != 0) {
328         b.append("strict ");
329     }
330     if ((accessFlags & ACC_SYNTHETIC) != 0) {
331         b.append("synthetic ");
332     }
333     return b.toString().trim();
334 }
335
336 /**
337  * Return the integer value (mask) corresponding to the specified access
338  * flag.
339  *
340  * @param accessFlag
341  *         access flag.
342  * @return the integer mask.

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

```

343     */
344
345     public static int accessFlagToInt(String accessFlag) {
346         int flag = 0;
347         if (accessFlag.equals("public")) {
348             flag = ACC_PUBLIC;
349         }
350         if (accessFlag.equals("private")) {
351             flag = ACC_PRIVATE;
352         }
353         if (accessFlag.equals("protected")) {
354             flag = ACC_PROTECTED;
355         }
356         if (accessFlag.equals("static")) {
357             flag = ACC_STATIC;
358         }
359         if (accessFlag.equals("final")) {
360             flag = ACC_FINAL;
361         }
362         if (accessFlag.equals("super")) {
363             flag = ACC_SUPER;
364         }
365         if (accessFlag.equals("synchronized")) {
366             flag = ACC_SYNCHRONIZED;
367         }
368         if (accessFlag.equals("volatile")) {
369             flag = ACC_VOLATILE;
370         }
371         if (accessFlag.equals("bridge")) {
372             flag = ACC_BRIDGE;
373         }
374         if (accessFlag.equals("transient")) {
375             flag = ACC_TRANSIENT;
376         }
377         if (accessFlag.equals("varargs")) {
378             flag = ACC_VARARGS;
379         }
380         if (accessFlag.equals("native")) {
381             flag = ACC_NATIVE;
382         }
383         if (accessFlag.equals("interface")) {
384             flag = ACC_INTERFACE;
385         }
386         if (accessFlag.equals("abstract")) {
387             flag = ACC_ABSTRACT;
388         }
389         if (accessFlag.equals("strict")) {
390             flag = ACC_STRICT;
391         }
392         if (accessFlag.equals("synthetic")) {
393             flag = ACC_SYNTHETIC;
394         }
395         if (accessFlag.equals("annotation")) {
396             flag = ACC_ANNOTATION;
397         }
398         if (accessFlag.equals("enum")) {
399             flag = ACC_ENUM;
400         }
401         return flag;
402     }
403
404 }
405

```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder