CLPath.java

```
1   // Copyright 2013 Bill Campbell, Swami Iyer and Bahar Akbal-Delibas
2
3   package jminusminus;
4
5   import java.io.BufferedInputStream;
6   import java.io.File;
7   import java.io.FileInputStream;
8   import java.io.FileNotFoundException;
9   import java.io.IOException;
10  import java.util.ArrayList;
11  import java.util.StringTokenizer;
12  import java.util.zip.ZipEntry;
13  import java.util.zip.ZipFile;
14
15  /**
16   * This class can be used to locate and load system, extension, and user-defined
17   * class files from directories and zip (jar) files. The code for this class has
18   * been adapted from the Kopi (http://www.dms.at/kopi/) project.
19   */
20
21  class CLPath {
22
23      /**
24       * Stores the individual directories, zip, and jar files from the class
25       * path.
26       */
27      private ArrayList<String> dirs;
28
29      /**
30       * Return a list of conceptual directories defining the class path.
31       *
32       * @param classPath
33       *            the directory names defining the class path.
34       * @return a list of conceptual directories defining the class path.
35       */
36
37      private ArrayList<String> loadClassPath(String classPath) {
38          ArrayList<String> container = new ArrayList<String>();
39
40          // Add directories/jars/zips from the classpath
41          StringTokenizer entries = new StringTokenizer(classPath,
42                  File.pathSeparator);
43          while (entries.hasMoreTokens()) {
44              container.add(entries.nextToken());
45          }
46
47          // Add system directories
48          if (System.getProperty("sun.boot.class.path") != null) {
49              entries = new StringTokenizer(System
50                      .getProperty("sun.boot.class.path"), File.pathSeparator);
51              while (entries.hasMoreTokens()) {
52                  container.add(entries.nextToken());
53              }
54          } else {
55              float version = Float
56                      .parseFloat(System.getProperty("java.version"));
57              if (version > 1.1) {
58                  String dir = System.getProperty("java.home")
59                          + File.separatorChar + "lib" + File.separatorChar
60                          + "rt.jar";
61                  container.add(dir);
62              }
63          }
64          return container;
65      }
66
```

```
67      /**
68       * Construct a CLPath object.
69       */
70
71      public CLPath() {
72          this(null, null);
73      }
74
75      /**
76       * Construct a CLPath object.
77       *
78       * @param path
79       *            the directory names defining the class path, separated by path
80       *            separator.
81       * @param extdir
82       *            the directory for the Java extension classes.
83       */
84
85      public CLPath(String path, String extdir) {
86          if (path == null) {
87              // No path specified, use CLASSPATH
88              path = System.getProperty("java.class.path");
89          }
90          if (path == null) {
91              // Last resort, use current directory
92              path = ".";
93          }
94          dirs = loadClassPath(path);
95          if (extdir == null) {
96              // Java extension classes
97              extdir = System.getProperty("java.ext.dirs");
98          }
99          if (extdir != null) {
100             File extDirectory = new File(extdir);
101             if (extDirectory.isDirectory()) {
102                 File[] extFiles = extDirectory.listFiles();
103                 for (int i = 0; i < extFiles.length; i++) {
104                     File file = extFiles[i];
105                     if (file.isFile()
106                             && (file.getName().endsWith(".zip") || file
107                                     .getName().endsWith(".jar"))) {
108                         dirs.add(file.getName());
109                     } else {
110                         // Wrong suffix; ignore
111                     }
112                 }
113             }
114         }
115     }
116
117     /**
118      * Return a CLInputStream instance for the class with specified name
119      * (fully-qualified; tokens separated by '/') or null if the class was not
120      * found.
121      *
122      * @param name
123      *            the fully-qualified name of the class -- java/util/ArrayList
124      *            for example.
125      * @return a CLInputStream instance for the class with specified name or
126      *         null if the class was not found.
127      */
128
129     public CLInputStream loadClass(String name) {
130         CLInputStream reader = null;
131         for (int i = 0; i < dirs.size(); i++) {
132             String dir = dirs.get(i);
133             File file = new File(dir);
134             if (file.isDirectory()) {
135                 File theClass = new File(dir, name.replace('/',
```

```
                        File.separatorChar)
                        + ".class");
                if (theClass.canRead()) {
                    try {
                        reader = new CLInputStream(new BufferedInputStream(
                                new FileInputStream(theClass)));
                    } catch (FileNotFoundException e) {
                        // Ignore
                    }
                }
            } else if (file.isFile()) {
                try {
                    ZipFile zip = new ZipFile(dir);
                    ZipEntry entry = zip.getEntry(name + ".class");
                    if (entry != null) {
                        reader = new CLInputStream(zip.getInputStream(entry));
                    }
                } catch (IOException e) {
                    // Ignore
                }
            } else {
                // Bogus entry; ignore
            }
        }
        return reader;
    }

}

```