

```

1 // foj/10/juno/juno.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.io.*;
7 import java.util.*;
8 import java.lang.*;
9
10 /**
11  * Juno (Juno's Unix NOX) mimics a command line operating system
12  * such as Unix.
13  * <p>
14  * A Juno system has a name, a set of Users, a JFile system,
15  * a login process and a set of shell commands.
16  *
17  * @see User
18  * @see JFile
19  * @see ShellCommand
20  *
21  * @version 10
22  */
23
24 public class Juno
25     implements Serializable
26 {
27     private final static String os      = "Juno";
28     private final static String version = "10";
29
30     private String  hostname; // host machine name
31     private Map    users; // lookup table for Users
32     private transient OutputInterface console;
33
34     private Directory slash; // root of JFile system
35     private Directory userHomes; // for home directories
36
37     private ShellCommandTable commandTable; // shell commands
38
39     // file containing Juno state
40     private transient String fileName = null;
41
42     // port used by Juno server for remote login
43     private int junoPort = 2001;
44
45     /**
46      * Construct a Juno (operating system) object.
47      *
48      * @param hostname the name of the host on which it's running.
49      * @param echoInput should all input be echoed as output?
50      * @param isGUI graphical user interface?
51      * @param isRemote running as a server?
52      */
53
54     public Juno( String hostname, boolean echoInput,
55                 boolean isGUI, boolean isRemote )
56 {

```

```

57 // Initialize the Juno environment ...
58 this.hostname      = hostname;
59 users              = new TreeMap();
60 commandTable      = new ShellCommandTable();
61
62 // the file system
63
64 slash = new Directory( "", null, null );
65 User root = new User( "root", "swordfish", slash,
66                      "Rick Martin" );
67 users.put( "root", root );
68 slash.setOwner( root );
69 userHomes = new Directory( "users", root, slash );
70
71 }
72
73 // Set up the correct console:
74 // command line (default), graphical or remote.
75
76 private void setupConsole( boolean echoInput, boolean isGUI,
77                            boolean isRemote )
78 {
79     LoginInterpreter interpreter
80         = new LoginInterpreter( this, null );
81
82     if (isGUI) {
83         console = new GUILoginConsole( hostname,
84                                       this, interpreter, echoInput );
85     }
86     else if (isRemote) {
87         console = new RemoteConsole( this, echoInput, junoPort );
88     }
89     else {
90         console = new JunoTerminal( echoInput );
91     }
92
93     // Tell the interpreter about the console
94     interpreter.setConsole( console );
95
96     // If we're using a simple command line interface,
97     // start that. (Constructing a GUI starts the GUI.)
98     // Shut down Juno when done
99
100    if (isGUI && !isRemote) {
101        interpreter.CLIlogin();
102        shutdown();
103    }
104
105    /**
106     * Shut down this Juno system.
107     *
108     * Save state if required.
109     */
110
111    public void shutdown( )
112 {

```

```

113     {
114         if (fileName != null) {
115             writeJuno( );
116         }
117     }
118 }
119 /**
120  * Set the name of file in which system state is kept.
121  * @param fileName the file name.
122  */
123
124
125 public void setFileName(String fileName)
126 {
127     this.fileName = fileName;
128 }
129
130 /**
131  * The name of the host computer on which this system
132  * is running.
133  * @return the host computer name.
134  */
135
136
137 public String getHostName()
138 {
139     return hostName;
140 }
141
142 /**
143  * The name of this operating system.
144  * @return the operating system name.
145  */
146
147
148 public String getOS()
149 {
150     return os;
151 }
152
153 /**
154  * The version number for this system.
155  * @return the version number.
156  */
157
158
159 public String getVersion()
160 {
161     return version;
162 }
163
164 /**
165  * The directory containing all user homes for this system.
166  * @return the directory containing user homes.
167  */
168

```

```

169
170     public Directory getUserHomes()
171     {
172         return userHomes;
173     }
174 }
175 /**
176  * The shell command table for this system.
177  * @return the shell command table.
178  */
179
180
181 public ShellCommandTable getCommandTable()
182 {
183     return commandTable;
184 }
185
186 /**
187  * Look up a user by user name.
188  * @param username the user's name.
189  * @return the appropriate User object.
190  */
191
192
193 public User lookupUser( String username )
194 {
195     return (User) users.get( username );
196 }
197
198 /**
199  * Create a new User.
200  * @param userName the User's login name.
201  * @param home her home Directory.
202  * @param password her password.
203  * @param realName her real name.
204  * @return newly created User.
205  */
206
207
208 public User createUser( String userName, Directory home,
209                        String password, String realName )
210 {
211     User newUser = new User( userName, password,
212                             home, realName );
213     users.put( userName, newUser );
214     return newUser;
215 }
216
217 /**
218  * The Juno system may be given the following command line
219  * arguments:
220  * -e: Echo all input (useful for testing).
221  * -version: Report the version number and exit.
222  */
223
224

```

```

225 * -g:          Support a GUI console.
226 *
227 * -remote     Start Juno server.
228 *
229 * -f filename File to read/write system state from/to
230 *
231 * [hostname]: The name of the host on which
232 *             Juno is running (optional).
233 */
234
235 public static void main( String[] args )
236 {
237     // Parse command line options
238
239     boolean echoInput   = false;
240     boolean versionQuery = false;
241     boolean isGUI       = false;
242     boolean isRemote    = false;
243     String  hostname    = "mars";
244     String  junoFileName = null;
245
246     for (int i=0; i < args.length; i++) {
247         if (args[i].equals("-e")) {
248             echoInput = true;
249         }
250         else if (args[i].equals("-version")) {
251             versionQuery = true;
252         }
253         else if (args[i].equals("-g")) {
254             isGUI = true;
255         }
256         else if (args[i].equals("-remote" )) {
257             isRemote = true;
258         }
259         else if (args[i].equals("-f")) {
260             junoFileName = args[i+1];
261         }
262         else {
263             hostname = args[i];
264         }
265     }
266
267     // If it's a version query give the version and exit
268     if ( versionQuery ) {
269         System.out.println( os + " version " + version );
270         System.exit(0);
271     }
272
273     // Create a new Juno or read one from a file.
274     Juno junoSystem = null;
275     if (junoFileName != null) {
276         junoSystem = readJuno( junoFileName );
277     }
278     if (junoSystem == null) {
279         junoSystem = new Juno(  hostname, echoInput,
280                               isGUI, isRemote );

```

```

281     }
282     junoSystem.setFileName( junoFileName );
283     junoSystem.setupConsole( echoInput, isGUI, isRemote );
284 }
285
286 // Read Juno state from a file.
287 //
288 // @param junoFileName the name of the file containing the system.
289 // @return the system, null if file does not exist.
290
291 private static Juno readJuno( String junoFileName )
292 {
293     File file = new File( junoFileName );
294     if (!file.exists()) {
295         return null;
296     }
297     ObjectInputStream instream = null;
298     try {
299         instream = new ObjectInputStream(
300             new FileInputStream( file ) );
301         Juno juno = (Juno) instream.readObject();
302         System.out.println(
303             "Juno state read from file " + junoFileName);
304         return juno;
305     }
306     catch (Exception e ) {
307         System.err.println("Problem reading " + junoFileName );
308         System.err.println(e);
309         System.exit(1);
310     }
311     finally {
312         try {
313             instream.close();
314         }
315         catch (Exception e) {
316         }
317     }
318     return null; // you can never get here
319 }
320
321 // Write Juno state to a file.
322
323 private void writeJuno()
324 {
325     ObjectOutputStream outputStream = null;
326     try {
327         outputStream = new ObjectOutputStream(
328             new FileOutputStream( fileName ) );
329         outputStream.writeObject( this );
330         System.out.println(
331             "Juno state written to file " + fileName);
332     }
333     catch (Exception e ) {
334         System.err.println("Problem writing " + fileName);
335         System.err.println(e);
336     }

```

```
337         finally {
338             try {
339                 outputStream.close();
340             }
341             catch (Exception e ) {
342             }
343         }
344     }
345 }
```