```
// joi/10/juno/Juno.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.io.*;
import java.util.*;
import java.lang.*;

/**
 * Juno (Juno's Unix NOt) mimics a command line operating system
 * such as Unix.
 * <p>
 * A Juno system has a name, a set of Users, a JFile system,
 * a login process and a set of shell commands.
 *
 * @see User
 * @see JFile
 * @see ShellCommand
 *
 * @version 10
 **/

public class Juno
    implements Serializable
{
    private final static String os      = "Juno";
    private final static String version = "10";

    private String    hostName;   // host machine name
    private Map       users;      // lookup table for Users
    private transient OutputInterface  console;

    private Directory slash;      // root of JFile system
    private Directory userHomes;  // for home directories

    private ShellCommandTable commandTable; // shell commands

    // file containing Juno state
    private transient String fileName = null;

    // port used by Juno server for remote login
    private int junoPort = 2001;

    /**
     * Construct a Juno (operating system) object.
     *
     * @param hostName  the name of the host on which it's running.
     * @param echoInput should all input be echoed as output?
     * @param isGUI graphical user interface?
     * @param isRemote running as a server?
     */

    public Juno( String hostName, boolean echoInput,
                 boolean isGUI, boolean isRemote )
    {
```



```
        // Initialize the Juno environment ...

        this.hostName     = hostName;
        users             = new TreeMap();
        commandTable      = new ShellCommandTable();

        // the file system

        slash = new Directory( "", null, null );
        User root = new User( "root", "swordfish", slash,
                              "Rick Martin" );
        users.put( "root", root );
        slash.setOwner(root);
        userHomes = new Directory( "users", root, slash );
    }

    // Set up the correct console:
    // command line (default), graphical or remote.

    private void setupConsole( boolean echoInput, boolean isGUI,
                               boolean isRemote )
    {
        LoginInterpreter interpreter
            = new LoginInterpreter( this, null );

        if (isGUI) {
            console = new GUILoginConsole( hostName,
                                           this, interpreter, echoInput);
        }
        else if (isRemote) {
            console = new RemoteConsole( this, echoInput, junoPort );
        }
        else {
            console = new JunoTerminal( echoInput );
        }

        // Tell the interpreter about the console
        interpreter.setConsole( console );

        // If we're using a simple command line interface,
        // start that.  (Constructing a GUI starts the GUI.)
        // Shut down Juno when done

        if (!isGUI && !isRemote) {
            interpreter.CLILogin();
            shutDown();
        }
    }

    /**
     * Shut down this Juno system.
     *
     * Save state if required.
     */

    public void shutDown( )
```

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



```
169
170      public Directory getUserHomes()
171      {
172          return userHomes;
173      }
174
175      /**
176       * The shell command table for this system.
177       *
178       * @return the shell command table.
179       */
180
181      public ShellCommandTable getCommandTable()
182      {
183          return commandTable;
184      }
185
186      /**
187       * Look up a user by user name.
188       *
189       * @param username the user's name.
190       * @return the appropriate User object.
191       */
192
193      public User lookupUser( String username )
194      {
195          return (User) users.get( username );
196      }
197
198      /**
199       * Create a new User.
200       *
201       * @param userName the User's login name.
202       * @param home her home Directory.
203       * @param password her password.
204       * @param realName her real name.
205       * @return newly created User.
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       *
221       * -e:         Echo all input (useful for testing).
222       *
223       * -version:   Report the version number and exit.
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];
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 outStream = null;
326          try {
327              outStream = new ObjectOutputStream(
328                                  new FileOutputStream( fileName ) );
329              outStream.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                  outStream.close();
   340              }
   341              catch (Exception e ) {
   342              }
   343          }
   344      }
   345  }
```