

```

1 // jol/5/files/JFile.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.util.Date;
7 import java.io.File;
8
9 /**
10  * A JFile object models a file in a hierarchical file system.
11  * <p>
12  * Extend this abstract class to create particular kinds of JFiles,
13  * e.g.:<br>
14  *   Directory _
15  *   * a JFile that maintains a list of the files it contains.<br>
16  *   * TextFile _
17  *   * a JFile containing text you might want to read.<br>
18  *
19  * @see Directory
20  * @see TextFile
21
22  * @version 5
23  */
24
25 public abstract class JFile
26 {
27     /**
28      * The separator used in pathnames.
29      */
30
31     public static final String separator = File.separator;
32
33     private String name; // a JFile knows its name
34     private String owner; // the owner of this file
35     private Date createDate; // when this file was created
36     private Date moddate; // when this file was last modified
37     private Directory parent; // the Directory containing this file
38
39     /**
40      * Construct a new JFile, set owner, parent, creation and
41      * modification dates. Add this to parent (unless this is the
42      * root Directory).
43      *
44      * @param name the name for this file (in its parent directory).
45      * @param creator the owner of this new file.
46      * @param parent the Directory in which this file lives.
47      */
48     protected JFile( String name, String creator, Directory parent )
49     {
50         this.name = name;
51         this.owner = creator;
52         this.parent = parent;
53         if (parent != null) {
54             parent.addJFile( name, this );
55         }
56     }

```

```

57         createDate = moddate = new Date(); // set dates to now
58     }
59
60     /**
61      * The name of the file.
62      *
63      * @return the file's name.
64      */
65
66     public String getName()
67     {
68         return name;
69     }
70
71     /**
72      * The full path to this file.
73      *
74      * @return the path name.
75      */
76
77     public String getPathName()
78     {
79         if (this.isRoot()) {
80             return separator;
81         }
82         if (parent.isRoot()) {
83             return separator + getName();
84         }
85         return parent.getPathName() + separator + getName();
86     }
87
88     /**
89      * The size of the JFile
90      * (as defined by the child class)..
91      *
92      * @return the size.
93      */
94
95     public abstract int getSize();
96
97     /**
98      * Suffix used for printing file names
99      * (as defined by the child class).
100
101      * @return the file's suffix.
102      */
103
104     public abstract String getSuffix();
105
106     /**
107      * Set the owner for this file.
108      *
109      * @param owner the new owner.
110      */
111
112     public void setOwner( String owner )

```

```

113     {
114         this.owner = owner;
115     }
116     /**
117     * The file's owner.
118     */
119     * @return the owner of the file.
120     */
121     public String getOwner()
122     {
123         return owner;
124     }
125     /**
126     * The date and time of the file's creation.
127     */
128     * @return the file's creation date and time.
129     */
130     * @return the file's creation date and time.
131     */
132     public String getCreateDate()
133     {
134         return createDate.toString();
135     }
136     /**
137     * Set the modification date to "now".
138     */
139     protected void setModDate()
140     {
141         modDate = new Date();
142     }
143     /**
144     * The date and time of the file's last modification.
145     */
146     * @return the date and time of the file's last modification.
147     */
148     public String getModDate()
149     {
150         return modDate.toString();
151     }
152     /**
153     * The Directory containing this file.
154     */
155     * @return the parent directory.
156     */
157     public Directory getParent()
158     {
159         return parent;
160     }
161 }

```

```

169     /**
170     * A JFile whose parent is null is defined to be the root
171     * (of a tree).
172     */
173     * @return true when this JFile is the root.
174     */
175     public boolean isRoot()
176     {
177         return (parent == null);
178     }
179     /**
180     * How a JFile represents itself as a String.
181     * That is,
182     * <pre>
183     * owner      size      modDate      name+suffix
184     * </pre>
185     * @return the String representation.
186     */
187     public String toString()
188     {
189         return getOwner() + "\t" +
190                getSize() + "\t" +
191                getModDate() + "\t" +
192                getName() + getSuffix();
193     }
194     // Unit test: main() and static support
195     private static Terminal terminal = new Terminal();
196     /**
197     * A unit test of JFile and its subclasses.
198     */
199     public static void main( String[] args )
200     {
201         out("Some hardwired, self documenting JFile system tests");
202         out("create and then explore JFile hierarchy");
203         out("    root      (owner sysadmin)");
204         out("    billhome (owner bill)");
205         out("    ebhome   (owner eb)");
206         out("    cs110    (owner eb)");
207         out("    diary    (owner eb)");
208         out("    insult   (owner bill)");
209         Directory root = new Directory( " ", "sysadmin", null );
210         Directory home1 = new Directory( "ebhome", "eb", root );
211         Directory home2 = new Directory( "billhome", "bill", root );
212         TextFile insult = new TextFile( "insult", "bill", home1,
213                                         "Your mother wore sneakers.");
214         insult.append( "\nIn the shower." );
215 }

```

```

225
226 Directory cs110 = new Directory( "cs110", "eb", home1);
227 cs110.addJFile( "diary",
228               new TextFile( "diary", "eb", cs110,
229                             "started work on Chapter 3"));
230
231 out("\nlist contents of the root directory:");
232 list( root );
233
234 out("\nlist contents of ebhome:");
235 list( home1 );
236
237 out("\nretrieve billhome, list its contents (empty):");
238 list( (Directory) root.retrieveJFile("billhome") );
239
240 out("\nretrieve insult, contents two line insult:");
241 type( (TextFile)home1.retrieveJFile("insult"));
242
243 out("\nretrieve file \"foo\" from ebhome, try to display it:");
244 type( (TextFile)home1.retrieveJFile("foo") );
245
246 out("\nlist contents of cs110 (one file):");
247 list( (Directory) home1.retrieveJFile("cs110") );
248
249 out("path to root:\t " + root.getPathName() );
250 out("path to ebhome:\t " + home1.getPathName() );
251 out("path to cs110:\t " + cs110.getPathName() );
252
253
254 // display a listing of the contents of a Directory
255
256 private static void list( Directory dir )
257 {
258     terminal.println( dir.getName() );
259     terminal.println( dir.getSize() +
260                     (dir.getSize() == 1
261                      ? " file:" : " files:") );
262
263     String[] fileNames = dir.getFileNames();
264     for ( int i = 0; i < fileNames.length; i++ ) {
265         String fileName = fileNames[i];
266         JFile jfile = dir.retrieveJFile( fileName );
267         terminal.println( jfile.toString() );
268     }
269
270 }
271
272 // display the contents of a TextFile
273
274 private static void type( TextFile file )
275 {
276     String whatToPrint;
277     if (file == null) {
278         whatToPrint = "no such file";
279     }
280     else {
281         whatToPrint = file.getContents();

```

```

281     }
282     terminal.println( whatToPrint );
283 }
284 // abbreviation for "terminal.println"
285
286 private static void out( String s )
287 {
288     terminal.println( s );
289 }
290
291 }

```