

```

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     /**
118      * The file's owner.
119      */
120     * @return the owner of the file.
121     */
122
123     public String getOwner()
124     {
125         return owner;
126     }
127
128     /**
129      * The date and time of the file's creation.
130      */
131     * @return the file's creation date and time.
132     */
133
134     public String getCreateDate()
135     {
136         return createDate.toString();
137     }
138
139     /**
140      * Set the modification date to "now".
141      */
142
143     protected void setModDate()
144     {
145         modDate = new Date();
146     }
147
148     /**
149      * The date and time of the file's last modification.
150      */
151     * @return the date and time of the file's last modification.
152     */
153
154     public String getModDate()
155     {
156         return modDate.toString();
157     }
158
159     /**
160      * The Directory containing this file.
161      */
162     * @return the parent directory.
163     */
164
165     public Directory getParent()
166     {
167         return parent;
168     }

```

```

169
170     /**
171      * A JFile whose parent is null is defined to be the root
172      * (of a tree).
173      */
174     * @return true when this JFile is the root.
175     */
176
177     public boolean isRoot()
178     {
179         return (parent == null);
180     }
181
182     /**
183      * How a JFile represents itself as a String.
184      * That is,
185      * <pre>
186      *   owner      size      modDate      name+suffix
187      * </pre>
188      */
189     * @return the String representation.
190     */
191
192     public String toString()
193     {
194         return getOwner() + "\t" +
195             getSize() + "\t" +
196             getModDate() + "\t" +
197             getName() + getSuffix();
198     }
199
200     // Unit test: main() and static support
201
202     private static Terminal terminal = new Terminal();
203
204     /**
205      * A unit test of JFile and its subclasses.
206      */
207
208     public static void main( String[] args )
209     {
210         out("Some hardwired, self documenting JFile system tests");
211         out("create and then explore JFile hierarchy");
212         out("    root      (owner sysadmin)");
213         out("    billhome (owner bill)");
214         out("    ebhome   (owner eb)");
215         out("    cs110    (owner eb)");
216         out("    diary   (owner eb)");
217         out("    insult  (owner bill)");
218         Directory root = new Directory( " ", "sysadmin", null );
219         Directory home1 = new Directory( "ebhome", "eb", root );
220         Directory home2 = new Directory( "billhome", "bill", root );
221
222         TextFile insult = new TextFile( "insult", "bill", home1,
223             "Your mother wore sneakers.");
224         insult.append( "\nIn the shower." );

```

```

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 }

```

```

1 // fo1/5/files/Directory.java
2 //
3 //
4 // Copyright 2003 Ethan Bolker and Bill Campbell
5
6 import java.util.*;
7
8 /**
9  * Directory of JFiles.
10
11  * A Directory is a JFile that maintains a
12  * table of the JFiles it contains
13  *
14  * @version 5
15  */
16
17 public class Directory extends JFile
18 {
19     private TreeMap jfiles; // table for JFiles in this Directory
20
21     /**
22      * Construct a Directory.
23
24      * @param name    the name for this Directory (in its parent Directo
25      * @param creator the owner of this new Directory
26      * @param parent  the Directory in which this Directory lives.
27      */
28
29     public Directory( String name, String creator, Directory parent)
30     {
31         super( name, creator, parent );
32         jfiles = new TreeMap();
33     }
34
35     /**
36      * The size of a directory is the number of TextFiles it contains.
37
38      * @return the number of TextFiles.
39      */
40
41     public int getSize()
42     {
43         return jfiles.size();
44     }
45
46     /**
47      * Suffix used for printing Directory names;
48      * we define it as the (system dependent)
49      * name separator used in path names.
50
51      * @return the suffix for Directory names.
52      */
53
54     public String getSuffix()
55     {
56         return JFile.separator;

```

```

57     }
58
59     /**
60      * Add a JFile to this Directory. Overwrite if a JFile
61      * of that name already exists.
62
63      * @param name the name under which this JFile is added.
64      * @param afile the JFile to add.
65      */
66
67     public void addJFile( String name, JFile afile)
68     {
69         jfiles.put( name, afile );
70         setModdate();
71     }
72
73     /**
74      * Get a JFile in this Directory, by name .
75
76      * @param filename the name of the JFile to find.
77      * @return the JFile found.
78      */
79
80     public JFile retrieveJFile( String filename )
81     {
82         JFile afile = (JFile)jfiles.get( filename );
83         return afile;
84     }
85
86     /**
87      * Get the contents of this Directory as an array of
88      * the file names, each of which is a String.
89
90      * @return the array of names.
91      */
92
93     public String[] getFileNamees()
94     {
95         return (String[])jfiles.keySet().toArray( new String[0] );
96     }
97 }

```

```

1 // jol/5/files/TextFile.java
2 //
3 //
4 // Copyright 2003 Ethan Bolker and Bill Campbell
5
6 /**
7  * A TextFile is a JFile that holds text.
8  *
9  * @version 5
10 */
11
12 public class TextFile extends JFile
13 {
14     private String contents; // The text itself
15
16     /**
17      * Construct a TextFile with initial contents.
18      *
19      * @param name    the name for this TextFile (in its parent Directory
20      * @param creator the owner of this new TextFile
21      * @param parent  the Directory in which this TextFile lives.
22      * @param initialContents the initial text
23      */
24
25     public TextFile( String name, String creator, Directory parent,
26                     String initialContents )
27     {
28         super( name, creator, parent );
29         setContents( initialContents );
30     }
31
32     /**
33      * Construct an empty TextFile.
34      *
35      * @param name    the name for this TextFile (in its parent Directory
36      * @param creator the owner of this new TextFile
37      * @param parent  the Directory in which this TextFile lives
38      */
39
40     TextFile( String name, String creator, Directory parent )
41     {
42         this( name, creator, "" );
43     }
44
45     /**
46      * The size of a text file is the number of characters stored.
47      *
48      * @return the file's size.
49      */
50
51     public int getSize()
52     {
53         return contents.length();
54     }
55
56     /**

```

```

57      * Suffix used for printing text file names is "".
58      *
59      * @return an empty suffix (for TextFiles).
60      */
61
62     public String getSuffix()
63     {
64         return "";
65     }
66
67     /**
68      * Replace the contents of the file.
69      *
70      * @param contents the new contents.
71      */
72
73     public void setContents( String contents )
74     {
75         this.contents = contents;
76         setModDate();
77     }
78
79     /**
80      * The contents of a text file.
81      *
82      * @return String contents of the file.
83      */
84
85     public String getContents()
86     {
87         return contents;
88     }
89
90     /**
91      * Append text to the end of the file.
92      *
93      * @param text the text to be appended.
94      */
95
96     public void append( String text )
97     {
98         setContents( contents + text );
99     }
100
101     /**
102      * Append a new line of text to the end of the file.
103      *
104      * @param text the text to be appended.
105      */
106
107     public void appendLine( String text )
108     {
109         this.setContents( contents + '\n' + text );
110     }
111
112     }

```