```
// joi/5/jfiles/JFile.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.util.Date;
import java.io.File;

/**
 * A JFile object models a file in a hierarchical file system.
 * <p>
 * Extend this abstract class to create particular kinds of JFiles,
 * e.g.:<br>
 * Directory  -
 *    a JFile that maintains a list of the files it contains.<br>
 * TextFile  -
 *    a JFile containing text you might want to read.<br>
 *
 * @see Directory
 * @see TextFile
 *
 * @version 5
 */

public abstract class JFile
{
    /**
     * The separator used in pathnames.
     */

    public static final String separator = File.separator;

    private String    name;       // a JFile knows its name
    private String    owner;      // the owner of this file
    private Date      createDate; // when this file was created
    private Date      modDate;    // when this file was last modified
    private Directory parent;     // the Directory containing this file

    /**
     * Construct a new JFile, set owner, parent, creation and
     * modification dates. Add this to parent (unless this is the
     * root Directory).
     *
     * @param name     the name for this file (in its parent directory).
     * @param creator  the owner of this new file.
     * @param parent   the Directory in which this file lives.
     */

    protected JFile( String name, String creator, Directory parent )
    {
        this.name   = name;
        this.owner  = creator;
        this.parent = parent;
        if (parent != null) {
            parent.addJFile( name, this );
        }
```



```
        createDate  = modDate = new Date(); // set dates to now
    }

    /**
     * The name of the file.
     *
     * @return the file's name.
     */

    public String getName()
    {
        return name;
    }

    /**
     * The full path to this file.
     *
     * @return the path name.
     */

    public String getPathName()
    {
        if (this.isRoot()) {
            return separator;
        }
        if (parent.isRoot()) {
            return separator + getName();
        }
        return parent.getPathName() + separator + getName();
    }

    /**
     * The size of the JFile
     * (as defined by the child class)..
     *
     * @return the size.
     */

    public abstract int getSize();

    /**
     * Suffix used for printing file names
     * (as defined by the child class).
     *
     * @return the file's suffix.
     */

    public abstract String getSuffix();

    /**
     * Set the owner for this file.
     *
     * @param owner the new owner.
     */

    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      // display the contents of a TextFile
272
273      private static void type( TextFile file )
274      {
275         String whatToPrint;
276         if (file == null) {
277             whatToPrint = "no such file";
278         }
279         else {
280             whatToPrint = file.getContents();
```



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