Selected files

3 printable files

prog-maven\tema12-ficheros\src\main\java\content\Terminal\FileManagerException.java
prog-maven\tema12-ficheros\src\main\java\content\Terminal\MiniFileManager.java
prog-maven\tema12-ficheros\src\main\java\content\Terminal\MiniTerminal.java

prog-maven\tema12-ficheros\src\main\java\content\Terminal\FileManagerException.java

```java
package content.Terminal;

/**
 * La clase {@code FileManagerException} representa una excepción personalizada
 * para manejar errores relacionados con las operaciones del sistema de archivos
 * dentro de la terminal.
 *
 * Extiende la clase base {@link Exception}, lo que permite lanzar y capturar
 * esta excepción de manera específica en los métodos del gestor de archivos.
 */
public class FileManagerException extends Exception {

    /**
     * Constructor que permite crear una nueva instancia de
     * {@code FileManagerException}
     * con un mensaje de error específico.
     *
     * @param mensaje El mensaje que describe el error ocurrido.
     */
    public FileManagerException(String mensaje) {
        super(mensaje); // Llama al constructor de la clase Exception con el mensaje proporcionado
    }
}

```

prog-maven\tema12-ficheros\src\main\java\content\Terminal\MiniFileManager.java

```java
package content.Terminal;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.sql.Date;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

```

```java
/**
 * La clase {@code MiniFileManager} proporciona funcionalidades para manipular
 * el
 * sistema de archivos de forma similar a una terminal.
 * Incluye operaciones como navegar entre directorios, listar contenido,
 * crear o eliminar archivos/directorios, mover archivos, y más.
 */
class MiniFileManager {

    /** Nombre del usuario que está ejecutando el programa. */
    private static String username;

    /** Nombre del host o equipo donde se ejecuta el programa. */
    private static String computerName;

    /** Constantes de color para resaltar en el prompt (verde, azul, reset). */
    public static final String GREEN = "\033[1;32m";
    public static final String BLUE = "\033[1;34m";
    public static final String RESET = "\033[0m";

    /** Nombre del sistema operativo actual en minúsculas. */
    private static final String os = System.getProperty("os.name").toLowerCase();

    /**
     * Obtiene el nombre del host (equipo) actual.
     *
     * @return nombre del equipo o "unknown-host" si no puede determinarse.
     */
    public String getComputername() {
        try {
            computerName = InetAddress.getLocalHost().getHostName();
        } catch (UnknownHostException e) {
            computerName = "unknown-host";
        }
        return computerName;
    }

    /**
     * Obtiene el nombre del usuario actual del sistema operativo.
     *
     * @return nombre del usuario actual.
     */
    public String getUsername() {
        if (os.contains("win")) {
            username = System.getenv("USERNAME");
        } else {
            username = System.getenv("USER");
        }
        return username;
    }

    /**
     * Genera el prompt de la terminal personalizado con usuario, host y ruta.
     *
```

```
72      * @return Cadena del prompt formateado.
73      */
74     public String getPrompt() {
75         String prompt = MiniTerminal.getDirectorioActual().getAbsolutePath().replace('\\',
'/');
76         if (os.contains("win"))
77             prompt = prompt.substring(2); // Omitir letra de unidad (ej. C:) en Windows
78         return String.format("%s%s@%s%s:~%s%s%s$  ", GREEN, getUsername(),
getComputername(), RESET, BLUE, prompt,
79                 RESET);
80     }
81
82     /**
83      * Retorna la ruta actual como una cadena.
84      *
85      * @return Ruta actual del directorio.
86      */
87     public String getPwd() {
88         return MiniTerminal.getDirectorioActual().getPath();
89     }
90
91     /**
92      * Cambia el directorio de trabajo actual al especificado.
93      *
94      * @param dir Nombre del directorio destino.
95      * @return true si el cambio se realizó correctamente.
96      * @throws FileManagerException si el directorio no existe o es inválido.
97      */
98     public boolean cd(String dir) throws FileManagerException {
99         switch (dir) {
100             case "." -> throw new FileManagerException("/. No es un directorio");
101             case ".." -> {
102                 File actual = MiniTerminal.getDirectorioActual();
103                 File padre = actual.getParentFile();
104                 if (actual == null || padre == null ||
actual.getAbsolutePath().equals("/"))
105                     return false;
106                 MiniTerminal.directorioActual = padre;
107                 return true;
108             }
109             default -> {
110                 if (dir.matches("\\.{3,}"))
111                     throw new FileManagerException("/" + dir + " No es un directorio");
112             }
113         }
114
115         File nuevoDir = new File(MiniTerminal.getDirectorioActual(),
dir).getAbsoluteFile();
116         if (!nuevoDir.isDirectory())
117             throw new FileManagerException("/" + dir + " No es un directorio");
118         MiniTerminal.directorioActual = nuevoDir;
119         return true;
120     }
121
```

```
122     /**
123      * Devuelve información sobre el archivo: tamaño y fecha de modificación.
124      *
125      * @param ruta Archivo o directorio.
126      * @return Cadena con tamaño y fecha.
127      */
128     public static String info(File ruta) {
129         Date fecha = new Date(ruta.lastModified());
130         return String.format("%-10d %-20s ", ruta.length(), fecha);
131     }
132 
133     /**
134      * Lista los archivos y carpetas de un directorio.
135      *
136      * @param ruta Directorio a listar.
137      * @param info true para mostrar información detallada.
138      * @throws FileNotFoundException si el directorio no existe.
139      * @throws FileManagerException  si no es un directorio válido.
140      */
141     public void ls(File ruta, boolean info) throws FileNotFoundException,
    FileManagerException {
142         if (!ruta.exists() || !ruta.isDirectory())
143             throw new FileManagerException(ruta.getName() + " No es un directorio");
144 
145         File[] archivos = ruta.listFiles();
146         Arrays.sort(archivos, Comparator.comparing(File::isFile)); // Directorios antes
    que archivos
147 
148         if (info)
149             System.out.printf("%-10s %-20s %-12s\n", "Tamaño", "UltimaModificacion",
    "Nombre");
150 
151         for (File f : archivos) {
152             if (f.isDirectory())
153                 System.out.printf("%s %s%s%s  \n", info ? info(f) : "", BLUE, f.getName(),
    RESET);
154             else
155                 System.out.printf("%s %s  \n", info ? info(f) : "", f.getName());
156         }
157     }
158 
159     /**
160      * Crea un nuevo directorio en la ubicación actual.
161      *
162      * @param ruta Nombre del nuevo directorio.
163      * @throws FileManagerException si ya existe.
164      */
165     public void crearDirectorio(String ruta) throws FileManagerException {
166         File nuevaCarpeta = new File(MiniTerminal.getDirectorioActual(), ruta);
167         if (nuevaCarpeta.exists())
168             throw new FileManagerException(" Ya existe este directorio");
169         nuevaCarpeta.mkdir();
170     }
171 
```

```
172     /**
173      * Elimina archivos o contenido de un directorio.
174      *
175      * @param rutaString Ruta del archivo o directorio.
176      * @throws FileManagerException si el archivo no existe o tiene subcarpetas.
177      */
178     public void remove(String rutaString) throws FileManagerException {
179         File ruta = new File(MiniTerminal.getDirectorioActual(), rutaString);
180         if (!ruta.exists())
181             throw new FileManagerException("No existe el archivo o direcorio");
182 
183         if (ruta.isFile()) {
184             ruta.delete();
185             return;
186         }
187 
188         boolean tieneSubcarpetas = false;
189         for (File f : ruta.listFiles()) {
190             if (f.isDirectory())
191                 tieneSubcarpetas = true;
192             else
193                 f.delete();
194         }
195 
196         if (tieneSubcarpetas)
197             throw new FileManagerException("Aviso: El directorio tiene subcarpetas,
    dejando intactas...");
198 
199         ruta.delete(); // Intenta borrar si quedó vacío
200     }
201 
202     /**
203      * Mueve un archivo/directorio a otro destino.
204      *
205      * @param file1String Ruta de origen.
206      * @param file2String Ruta de destino.
207      * @throws FileManagerException si falla la operación.
208      */
209     public void move(String file1String, String file2String) throws FileManagerException {
210         File file1 = new File(MiniTerminal.getDirectorioActual(), file1String);
211         File file2 = new File(MiniTerminal.getDirectorioActual(), file2String);
212 
213         if (!file1.exists())
214             throw new FileManagerException("No existe el directorio o archivo");
215 
216         if (file2.isDirectory())
217             file2 = new File(file2, file1.getName());
218 
219         if (!file1.renameTo(file2))
220             throw new FileManagerException("Error al mover ");
221     }
222 
223     /**
224      * Muestra todos los comandos disponibles de la terminal.
```

```
225      */
226     public void help() {
227         System.out.println(
228                 """
229                         Comandos ->
230                         pwd                             (muestra la ruta actual)
231                         cd <DIR>                        (cambia de directorio)
232                         ls                              (lista carpetas y archivos)
233                         ll                              (lista con información adicional)
234                         mkdir <DIR>                     (crea un nuevo directorio)
235                         rm <FILE>                       (elimina archivos dentro de un
directorio)
236                         mv <FILE1> <FILE2>              (mueve o renombra un
archivo/directorio)
237                         mostrar <FICHERO>               (muestra el contenido de un
archivo)
238                         sustituir <FICHERO> <ORI> <FIN> (reemplaza cadenas en archivo)
239                         help                            (muestra este mensaje)
240                         exit                            (sale de la terminal)
241                         """);
242     }
243 
244     /**
245      * Muestra el contenido de un archivo en consola.
246      *
247      * @param fichero Nombre del archivo a mostrar.
248      * @throws FileManagerException si no es un archivo válido.
249      */
250     public void mostrar(String fichero) throws FileManagerException {
251         File file = new File(MiniTerminal.getDirectorioActual(), fichero);
252         Path ruta = Paths.get(file.toURI());
253 
254         if (!Files.exists(ruta))
255             throw new FileManagerException("No es un fichero");
256 
257         try {
258 
259             Files.lines(ruta).forEach(System.out::println);
260 
261         } catch (IOException e) {
262             System.out.println(e.getMessage());
263         }
264     }
265 
266     /**
267      * Sustituye una cadena por otra dentro de un archivo de texto.
268      *
269      * @param fichero        Nombre del archivo.
270      * @param cadenaOriginal Cadena a buscar.
271      * @param cadenaFinal    Cadena que la reemplaza.
272      * @throws FileManagerException si el archivo no es válido.
273      */
274     public void sustituir(String fichero, String cadenaOriginal, String cadenaFinal)
throws FileManagerException {
```

```
275
276         File file = new File(MiniTerminal.getDirectorioActual(), fichero);
277         Path ruta = Paths.get(file.toURI());
278
279         if (!Files.exists(ruta))
280             throw new FileManagerException("No es un fichero");
281
282         try {
283
284             List<String> lineas = Files.lines(ruta)
285                     .map(l -> l.contains(cadenaOriginal) ? l.replace(cadenaOriginal,
    cadenaFinal) : l)
286                     .collect(Collectors.toList());
287
288             Files.write(ruta, lineas, StandardOpenOption.TRUNCATE_EXISTING);
289
290         } catch (IOException e) {
291             System.out.println(e.getMessage());
292         }
293     }
294
295 }
296
```

prog-maven\tema12-ficheros\src\main\java\content\Terminal\MiniTerminal.java

```
1  package content.Terminal;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.util.Scanner;
6
7  /**
8   * La clase {@code MiniTerminal} simula una terminal interactiva básica,
9   * permitiendo al usuario ejecutar comandos relacionados con el sistema de
10  * archivos.
11  * Soporta comandos como {@code pwd}, {@code cd}, {@code ls}, {@code mkdir},
12  * {@code rm},
13  * {@code mv}, {@code mostrar}, {@code sustituir}, entre otros.
14  */
15 public class MiniTerminal {
16
17     /**
18      * Directorio actual de trabajo.
19      * Inicialmente se establece al directorio de trabajo del sistema operativo.
20      */
21     protected static File directorioActual = new File(System.getProperty("user.dir"));
22
23     /**
24      * Instancia de {@link MiniFileManager} utilizada para manejar
25      * operaciones sobre el sistema de archivos.
26      */
27     private static MiniFileManager mfm = new MiniFileManager();
28
29     /**
```

```
30       * Devuelve el directorio actual en el que se encuentra la terminal.
31       *
32       * @return El directorio actual como un objeto {@link File}.
33       */
34      public static File getDirectorioActual() {
35          return directorioActual;
36      }
37
38      /**
39       * Valida si el número de argumentos proporcionado para un comando
40       * se encuentra dentro de los límites esperados.
41       *
42       * @param comando Nombre del comando.
43       * @param args    Argumentos pasados al comando.
44       * @param min     Número mínimo de argumentos esperados.
45       * @param max     Número máximo de argumentos permitidos.
46       * @return {@code true} si el número de argumentos es válido; {@code false} en
47       *         caso contrario.
48       */
49      private static boolean validarArgumentos(String comando, String[] args, int min, int
    max) {
50          if (args.length < min) {
51              System.out.println("Faltan argumentos");
52          } else if (args.length > max) {
53              System.out.println("Demasiados argumentos");
54          }
55          return args.length >= min && args.length <= max;
56      }
57
58      /**
59       * Método principal que inicia la ejecución de la terminal.
60       * Lee comandos desde la entrada estándar y ejecuta la lógica correspondiente
61       * llamando a métodos de {@link MiniFileManager}.
62       *
63       * @param args Argumentos de línea de comandos (no utilizados en esta
64       *             implementación).
65       */
66      public static void main(String[] args) {
67          String comando;
68          Scanner scanner = new Scanner(System.in);
69          boolean running = true;
70
71          // Bucle principal de ejecución de la terminal
72          while (running) {
73              // Muestra el prompt actual
74              System.out.print(mfm.getPrompt());
75
76              // Lee la línea introducida por el usuario
77              comando = scanner.nextLine().trim();
78
79              // Separa los argumentos considerando comillas dobles
80              String[] argumentos = comando.split(" (?=([^\"]*\"[^\"]*\")*[^\"]*$)");
81
82              // Elimina comillas al inicio y final de cada argumento
```

```
83              for (int i = 0; i < argumentos.length; i++) {
84                  argumentos[i] = argumentos[i].replaceAll("^\"|\"$", "");
85              }
86
87              // Procesa el comando
88              switch (argumentos[0]) {
89
90                  case "pwd" ->
91                      System.out.println(validarArgumentos("pwd", argumentos, 1, 1) ?
    mfm.getPwd() : "");
92
93                  case "cd" -> {
94                      if (validarArgumentos("cd", argumentos, 2, 2)) {
95                          try {
96                              mfm.cd(argumentos[1]);
97                          } catch (FileManagerException e) {
98                              System.out.println(e.getMessage());
99                          }
100                     }
101                 }
102
103                 case "ls" -> {
104                     if (validarArgumentos("ls", argumentos, 1, 1)) {
105                         try {
106                             mfm.ls(directorioActual, false);
107                         } catch (FileNotFoundException e) {
108                             System.out.println("No se encontró el archivo o directorio");
109                         } catch (FileManagerException e) {
110                             System.out.println(e.getMessage());
111                         }
112                     }
113                 }
114
115                 case "ll" -> {
116                     if (validarArgumentos("ll", argumentos, 1, 1)) {
117                         try {
118                             mfm.ls(directorioActual, true);
119                         } catch (FileNotFoundException e) {
120                             System.out.println("No se encontró el archivo o directorio");
121                         } catch (FileManagerException e) {
122                             System.out.println(e.getMessage());
123                         }
124                     }
125                 }
126
127                 case "mkdir" -> {
128                     if (validarArgumentos("mkdir", argumentos, 2, 2)) {
129                         try {
130                             mfm.crearDirectorio(argumentos[1]);
131                         } catch (FileManagerException e) {
132                             System.out.println(e.getMessage());
133                         }
134                     }
135                 }
```

```
136
137                 case "rm" -> {
138                     if (validarArgumentos("rm", argumentos, 2, 2)) {
139                         try {
140                             mfm.remove(argumentos[1]);
141                         } catch (FileManagerException e) {
142                             System.out.println(e.getMessage());
143                         }
144                     }
145                 }
146
147                 case "mv" -> {
148                     if (validarArgumentos("mv", argumentos, 3, 3)) {
149                         try {
150                             mfm.move(argumentos[1], argumentos[2]);
151                         } catch (FileManagerException e) {
152                             System.out.println(e.getMessage());
153                         }
154                     }
155                 }
156
157                 case "help" -> mfm.help();
158
159                 case "mostrar" -> {
160                     if (validarArgumentos("mostrar", argumentos, 2, 2)) {
161                         try {
162                             mfm.mostrar(argumentos[1]);
163                         } catch (FileManagerException e) {
164                             System.out.println(e.getMessage());
165                         }
166                     }
167                 }
168
169                 case "sustituir" -> {
170                     if (validarArgumentos("sustituir", argumentos, 4, 4)) {
171                         try {
172                             mfm.sustituir(argumentos[1], argumentos[2], argumentos[3]);
173                         } catch (FileManagerException e) {
174                             System.out.println(e.getMessage());
175                         }
176                     }
177                 }
178
179                 case "exit" -> running = false;
180
181                 default -> System.out.println("Comando no reconocido");
182             }
183         }
184
185         scanner.close();
186     }
187 }
188
```