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;

```

```
18 /**
19  * La clase {@code MiniFileManager} proporciona funcionalidades para manipular
20  * el
21  * sistema de archivos de forma similar a una terminal.
22  * Incluye operaciones como navegar entre directorios, listar contenido,
23  * crear o eliminar archivos/directorios, mover archivos, y más.
24  */
25 class MiniFileManager {
26
27     /** Nombre del usuario que está ejecutando el programa. */
28     private static String username;
29
30     /** Nombre del host o equipo donde se ejecuta el programa. */
31     private static String computerName;
32
33     /** Constantes de color para resaltar en el prompt (verde, azul, reset). */
34     public static final String GREEN = "\033[1;32m";
35     public static final String BLUE = "\033[1;34m";
36     public static final String RESET = "\033[0m";
37
38     /** Nombre del sistema operativo actual en minúsculas. */
39     private static final String os = System.getProperty("os.name").toLowerCase();
40
41     /**
42      * Obtiene el nombre del host (equipo) actual.
43      *
44      * @return nombre del equipo o "unknown-host" si no puede determinarse.
45      */
46     public String getComputername() {
47         try {
48             computerName = InetAddress.getLocalHost().getHostName();
49         } catch (UnknownHostException e) {
50             computerName = "unknown-host";
51         }
52         return computerName;
53     }
54
55     /**
56      * Obtiene el nombre del usuario actual del sistema operativo.
57      *
58      * @return nombre del usuario actual.
59      */
60     public String getUsername() {
61         if (os.contains("win")) {
62             username = System.getenv("USERNAME");
63         } else {
64             username = System.getenv("USER");
65         }
66         return username;
67     }
68
69     /**
70      * Genera el prompt de la terminal personalizado con usuario, host y ruta.
71      *
```

```
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      * @return true si el directorio se creó correctamente.
164      * @throws FileManagerException si ya existe.
165      */
166     public boolean crearDirectorio(String ruta) throws FileManagerException {
167         File nuevaCarpeta = new File(MiniTerminal.getDirectorioActual(), ruta);
168         if (nuevaCarpeta.exists())
169             throw new FileManagerException(" Ya existe este directorio");
170         return nuevaCarpeta.mkdir();
171     }
```

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

```java
    /**
     * Muestra todos los comandos disponibles de la terminal.
     */
    public void help() {
        System.out.println(
                """
                        Comandos ->
                        pwd                             (muestra la ruta actual)
                        cd <DIR>                        (cambia de directorio)
                        ls                              (lista carpetas y archivos)
                        ll                              (lista con información adicional)
                        mkdir <DIR>                     (crea un nuevo directorio)
                        rm <FILE>                       (elimina archivos dentro de un
directorio)
                        mv <FILE1> <FILE2>              (mueve o renombra un
archivo/directorio)
                        mostrar <FICHERO>               (muestra el contenido de un
archivo)
                        sustituir <FICHERO> <ORI> <FIN> (reemplaza cadenas en archivo)
                        help                            (muestra este mensaje)
                        exit                            (sale de la terminal)
                        """);
    }

    /**
     * Muestra el contenido de un archivo en consola.
     *
     * @param fichero Nombre del archivo a mostrar.
     * @throws FileManagerException si no es un archivo válido.
     */
    public void mostrar(String fichero) throws FileManagerException {
        File file = new File(MiniTerminal.getDirectorioActual(), fichero);
        Path ruta = Paths.get(file.toURI());

        if (!Files.exists(ruta))
            throw new FileManagerException("No es un fichero");

        try {

            Files.lines(ruta).forEach(System.out::println);

        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }

    /**
     * Sustituye una cadena por otra dentro de un archivo de texto.
     *
     * @param fichero        Nombre del archivo.
     * @param cadenaOriginal Cadena a buscar.
     * @param cadenaFinal    Cadena que la reemplaza.
     * @throws FileManagerException si el archivo no es válido.
     */
```

```
276     public void sustituir(String fichero, String cadenaOriginal, String cadenaFinal)
    throws FileManagerException {
277
278         File file = new File(MiniTerminal.getDirectorioActual(), fichero);
279         Path ruta = Paths.get(file.toURI());
280
281         if (!Files.exists(ruta))
282             throw new FileManagerException("No es un fichero");
283
284         try {
285
286             List<String> lineas = Files.lines(ruta)
287                     .map(l -> l.contains(cadenaOriginal) ? l.replace(cadenaOriginal,
    cadenaFinal) : l)
288                     .collect(Collectors.toList());
289
290             Files.write(ruta, lineas, StandardOpenOption.TRUNCATE_EXISTING);
291
292         } catch (IOException e) {
293             System.out.println(e.getMessage());
294         }
295     }
296
297 }
298
```

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

```
                        try {
                            mfm.remove(argumentos[1]);
                        } catch (FileManagerException e) {
                            System.out.println(e.getMessage());
                        }
                    }
                }

                case "mv" -> {
                    if (validarArgumentos("mv", argumentos, 3, 3)) {
                        try {
                            mfm.move(argumentos[1], argumentos[2]);
                        } catch (FileManagerException e) {
                            System.out.println(e.getMessage());
                        }
                    }
                }

                case "help" -> mfm.help();

                case "mostrar" -> {
                    if (validarArgumentos("mostrar", argumentos, 2, 2)) {
                        try {
                            mfm.mostrar(argumentos[1]);
                        } catch (FileManagerException e) {
                            System.out.println(e.getMessage());
                        }
                    }
                }

                case "sustituir" -> {

                    try {
                        String argumento2;
                        String argumento3;

                        if (argumentos.length < 4) {
                            argumento2 = argumentos[2];
                            argumento3 = argumentos[3];
                        } else {
                            argumento2 = comando.substring(comando.indexOf("<") + 1,
comando.indexOf(">"));
                            argumento3 = comando.substring(comando.lastIndexOf("<") + 1,
comando.lastIndexOf(">"));
                        }

                        mfm.sustituir(argumentos[1], argumento2, argumento3);

                    } catch (FileManagerException e) {
                        System.out.println(e.getMessage());
                    }

                }

```

```
                case "exit" -> running = false;

                default -> System.out.println("Comando no reconocido");
            }
        }

        scanner.close();
    }
}

```