

```

1  /**
2   * Mainframe.java
3   * <p>
4   * Clase que contiene la implementación de los métodos para conectarse con el
5   * mainframe y ejecutar la aplicación legada.
6   * <p>
7   * Radu Constantin Robu y Alberto Pérez
8   */
9  package P3.Control;
10
11  import java.io.*;
12
13  import static java.lang.Thread.sleep;
14
15  public class Mainframe implements MainframeAPI {
16      private static final String NOK = "NOK";
17      private static final String TASKS2JOB = "tasks2.job";
18      private static final String MENU_TASKS2 =
19          "TASK MANAGEMENT 2.0 BY TURO-SL SOFT";
20      private static final int DELAY = 50;
21
22      private Process proceso;
23      private PrintWriter outStream = null;
24      private BufferedReader inStream = null;
25
26      private static Mainframe singleton = null;
27
28      /**
29       * Constructor de la clase.
30       *
31       * @throws IOException
32       */
33      private Mainframe() throws IOException {
34          proceso = Runtime.getRuntime().exec(TERMINAL_SIN_PANTALLA);
35          inStream = new BufferedReader(new InputStreamReader(
36              proceso.getInputStream()));
37          outStream = new PrintWriter(new OutputStreamWriter(
38              proceso.getOutputStream()));
39      }
40
41      /**
42       * Devuelve la instancia del objeto de la clase.
43       *
44       * @return
45       * @throws IOException
46       */
47      public static Mainframe getInstance() throws IOException {
48          if (singleton == null) {
49              singleton = new Mainframe();
50          }
51          return singleton;
52      }
53
54      /**
55       * Realiza la conexión con la máquina.

```

```

56  *
57  * @param host
58  * @return
59  * @throws IOException
60  */
61  private boolean conectarHost(String host) throws IOException {
62      outputStream.println(CONNECT + host);
63      outputStream.flush();
64      return ejecutarSiguienteComando();
65  }
66
67  /**
68   * Obtiene la respuesta textual del mainframe en la terminal s3270.
69   *
70   * @return
71   * @throws IOException
72   */
73  public String obtenerRespuestaMaquina() throws IOException {
74      String resultado = "";
75      String line = "";
76      do {
77          line = inputStream.readLine();
78          if (line == null ||
79              line.matches(PATRON_RESPUESTA_MAINFRAME_ERROR)) {
80              return NOK;
81          }
82          resultado += line + "\n";
83      } while (inputStream.ready());
84      return resultado;
85  }
86
87  /**
88   * Verifica si se inicia sesión correctamente.
89   *
90   * @return
91   * @throws InterruptedException
92   * @throws IOException
93   */
94  private RESPUESTAS_INICIO_SESION falloInicioSesion()
95      throws InterruptedException, IOException {
96      if (esperarPantalla(MENSAJE_USUARIO_EN_USO)) {
97          if (enviarComando(COMANDO_ENTER)) {
98              return RESPUESTAS_INICIO_SESION.USUARIO_EN_USO;
99          }
100     } else if (esperarPantalla(MENSAJE_IDIOMA_NO_SOPORTADO)) {
101         if (enviarComando(COMANDO_ENTER)) {
102             return RESPUESTAS_INICIO_SESION.OK;
103         }
104     } else if (esperarPantalla(MENSAJE_USUARIO_INCORRECTO)) {
105         if (enviarComando(COMANDO_ENTER)) {
106             return RESPUESTAS_INICIO_SESION.USUARIO_INCORRECTO;
107         }
108     } else if (esperarPantalla(MENSAJE_CONTRASENYA_INCORRECTA)) {
109         if (enviarComando(COMANDO_ENTER)) {
110             return RESPUESTAS_INICIO_SESION.CONTRASENYA_INCORRECTA;

```

```

111     }
112 }
113     return RESPUESTAS_INICIO_SESION.NOK;
114 }
115
116 /**
117  * Conexión con el host.
118  *
119  * @param host
120  * @param username
121  * @param password
122  * @return
123  * @throws InterruptedException
124  * @throws IOException
125  */
126 public RESPUESTAS_INICIO_SESION conexion(String host, String username,
127                                           String password)
128     throws InterruptedException, IOException {
129     RESPUESTAS_INICIO_SESION resultadoInicioSesion;
130     if (conectarHost(host)) {
131         if (login(username, password)) {
132             resultadoInicioSesion = falloInicioSesion();
133             if (resultadoInicioSesion == RESPUESTAS_INICIO_SESION.OK) {
134                 if (ejecutarTasksJob()) {
135                     return RESPUESTAS_INICIO_SESION.OK;
136                 }
137             } else {
138                 return resultadoInicioSesion;
139             }
140         }
141     }
142     return RESPUESTAS_INICIO_SESION.NOK;
143 }
144
145 /**
146  * Inicio de sesión en el mainframe.
147  *
148  * @param username
149  * @param password
150  * @return
151  * @throws InterruptedException
152  * @throws IOException
153  */
154 private boolean login(String username, String password)
155     throws InterruptedException, IOException {
156     if (esperarPantalla(PANTALLA_CONEXION)) {
157         if (enviarComando(COMANDO_ENTER)) {
158             if (esperarPantalla(PANTALLA_LOGIN)) {
159                 if (enviarString(username)) {
160                     if (enviarComando(COMANDO_ENTER)) {
161                         if (enviarString(password)) {
162                             if (enviarComando(COMANDO_ENTER)) {
163                                 return true;
164                             }
165                         }
166                     }
167                 }
168             }
169         }
170     }
171     return false;
172 }

```

```

166     }
167     }
168     }
169     }
170     }
171     return false;
172 }
173
174 /**
175  * Ejecuta la aplicación legada.
176  *
177  * @return
178  * @throws InterruptedException
179  * @throws IOException
180  */
181 public boolean ejecutarTasksJob() throws InterruptedException,
182     IOException {
183     if (esperarPantalla(PANTALLA_MENU_PRINCIPAL)) {
184         if (enviarString(TASKS2JOB)) {
185             if (enviarComando(COMANDO_ENTER)) {
186                 if (esperarPantalla(MENU_TASKS2)) {
187                     return true;
188                 }
189             }
190         }
191     }
192     return false;
193 }
194
195 /**
196  * Realiza la salida controlada del mainframe a través de la terminal
197  * s3270.
198  *
199  * @return
200  * @throws IOException
201  */
202 @Override
203 public boolean logout() throws IOException {
204     enviarString(CADENA_OFF);
205     enviarComando(COMANDO_ENTER);
206     enviarComando(COMANDO_EXIT);
207
208     return true;
209 }
210
211 /**
212  * Método sobreescrito para enviar un comando al mainframe.
213  *
214  * @param comando
215  * @return
216  * @throws IOException
217  */
218 @Override
219 public boolean enviarComando(String comando) throws IOException {
220     outputStream.println(comando);

```

```

221     outputStream.flush();
222     if (comando.equals(COMANDO_ASCII)) {
223         return true;
224     } else {
225         return ejecutarSiguienteComando();
226     }
227 }
228
229 /**
230  * Envía una cadena de texto a la terminal s3270.
231  *
232  * @param mensaje
233  * @return
234  * @throws IOException
235  */
236 @Override
237 public boolean enviarString(String mensaje) throws IOException {
238     outputStream.println(String.format(FORMATO_CADENA_TEXTO, mensaje));
239     outputStream.flush();
240     return ejecutarSiguienteComando();
241 }
242
243 /**
244  * Método sobreescrito para esperar una pantalla del emulador.
245  *
246  * @param lineaABuscar
247  * @return
248  * @throws IOException
249  * @throws InterruptedException
250  */
251 @Override
252 public boolean esperarPantalla(String lineaABuscar)
253     throws IOException, InterruptedException {
254     Long maxTime = TIEMPO_EJEC_MAXIMO + System.currentTimeMillis();
255     String resultado = "";
256     do {
257         enviarComando(COMANDO_ASCII);
258         sleep(DELAY);
259         resultado = obtenerRespuestaMaquina();
260         if (resultado.contains(lineaABuscar)) {
261             return true;
262         }
263     } while (maxTime > System.currentTimeMillis());
264
265     return false;
266 }
267
268 /**
269  * Método para esperar para poder ejecutar el siguiente comando.
270  *
271  *
272  * @return
273  * @throws IOException
274  */
275 @Override

```

```
276     public boolean ejecutarSiguienteComando() throws IOException {
277         String line = "";
278         Long maxTime = TIEMPO_EJEC_MAXIMO + System.currentTimeMillis();
279         while (maxTime > System.currentTimeMillis()) {
280             line = inStream.readLine();
281             if (line == null ||
282                 line.matches(PATRON_RESPUESTA_MAINFRAME_ERROR)) {
283                 System.out.println(line);
284                 return false;
285             } else if (line.matches(PATRON_RESPUESTA_MAINFRAME_OK)) {
286                 return true;
287             }
288         }
289         return false;
290     }
291
292 }
293
```