```java
/**
 * TasksJob.java
 * <p>
 * Clase que implementa los métodos para la ejecución de las
 * funciones de la aplicación legada.
 * <p>
 * Radu Constantin Robu y Alberto Pérez
 */
package P3.Control;

import P3.Modelo.Tarea;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class TasksJob implements TasksAPI {
    private static final String NUEVO_FICHERO = "N";
    private static final String GUARDAR = "s";
    private static final String ANYADIR = "a";
    private static final String LISTAR = "l";
    private static final String ELIMINAR = "r";
    private static final String BUSCAR = "t";
    private static final String SI = "y";

    private static final String PATRON_IDTAREA = "^data: TASK NUMBER: .*$";
    private static final String PATRON_NOMBRE = "^data: NAME.*$";
    private static final String PATRON_DESCRIPCION = "^data: DESCRIPTION.*$";
    private static final String PATRON_FECHA = "^data: DATE .*$";
    private static final String TEXTO_FECHA = "data: DATE        : ";
    private static final String TEXTO_DESCRIPCION = "data: DESCRIPTION: ";
    private static final String TEXTO_NOMBRE = "data: NAME        : ";
    private static final String TEXTO_ID_TAREA = "data: TASK NUMBER: ";

    private static final String MENU_TASKS2 = "**MENU**";
    private static final String PANTALLA_NUEVO_FICHERO_TAREAS =
            "**NEW TASK FILE**";
    private static final String FICHERO_TAREAS_CREADO =
            "NEW TASK FILE HAS BEEN CREATED";
    private static final String MENSAJE_SALIDA = "BYE";
    private static final String PANTALLA_GUARDAR_TAREAS = "SAVE TASKS";
    private static final String PANTALLA_LISTAR_TAREAS = "**LIST TASK**";
    private static final String TAREAS_GUARDADAS = "TASKS HAVE BEEN SAVED";
    private static final String FIN_LISTA = "**END**";
    private static final String PANTALLA_BUSCAR_TAREAS = "**SEARCH TASK**";
    private static final String MENSAJE_CONFIRMAR_ELIMINAR = "CONFIRM (Y/N)";
    private static final String PANTALLA_TAREA_NO_ENCONTRADA = "TASK NOT FOUND";
    private static final String PANTALLA_ELIMINAR_TAREA = "**REMOVE TASK**";
    private static final String PANTALLA_ANYADIR_TAREA = "**ADD TASK**";

    private Mainframe mainframe;

    /**
     * Constructor de la clase.
     *
```

```
56       * @param mainframe
57       * @throws IOException
58       */
59      public TasksJob(Mainframe mainframe) throws IOException {
60          this.mainframe = mainframe;
61      }
62
63      /**
64       * Opción de tasks2 para crear un nuevo fichero de tareas.
65       *
66       * @return
67       * @throws IOException
68       * @throws InterruptedException
69       */
70      @Override
71      public boolean nuevoFicheroTareas()
72              throws IOException, InterruptedException {
73          if (mainframe.enviarString(NUEVO_FICHERO)) {
74              if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
75                  if (enviarRespuestaConfirmacion() && ficheroTareasCreado()) {
76                      return true;
77                  }
78              }
79          }
80          return false;
81      }
82
83      /**
84       * Método auxiliar para enviar al emulador la confirmación para crear un
85       * nuevo fichero de tareas.
86       *
87       * @return
88       * @throws IOException
89       * @throws InterruptedException
90       */
91      private boolean enviarRespuestaConfirmacion()
92              throws IOException, InterruptedException {
93          if (mainframe.esperarPantalla(PANTALLA_NUEVO_FICHERO_TAREAS)) {
94              if (mainframe.enviarString(SI)) {
95                  if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
96                      return true;
97                  }
98              }
99          }
100         return false;
101     }
102
103     /**
104      * Método auxiliar para esperar la pantalla de confirmación de la
105      * creación del fichero de tareas.
106      *
107      * @return
108      * @throws IOException
109      * @throws InterruptedException
110      */
```

```
111     private boolean ficheroTareasCreado()
112             throws IOException, InterruptedException {
113         if (mainframe.esperarPantalla(FICHERO_TAREAS_CREADO)) {
114             if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
115                 if (mainframe.esperarPantalla(MENU_TASKS2)) {
116                     return true;
117                 }
118             }
119         }
120         return false;
121     }
122 
123     /**
124      * Opción de tasks2 para añadir una nueva tarea.
125      *
126      * @param idTarea
127      * @param nombreTarea
128      * @param descripcionTarea
129      * @param fecha
130      * @return
131      * @throws IOException
132      * @throws InterruptedException
133      */
134     @Override
135     public CODIGO_ERROR anyadirTarea(String idTarea, String nombreTarea,
136                                      String descripcionTarea, String fecha)
137             throws IOException, InterruptedException {
138         CODIGO_ERROR codigo = CODIGO_ERROR.NOK;
139 
140         if (existeIdTarea(idTarea)) {
141             return CODIGO_ERROR.IDTAREA_REPETIDO;
142         }
143 
144         if (mainframe.enviarString(ANYADIR)) {
145             if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
146                 if (mainframe.esperarPantalla(PANTALLA_ANYADIR_TAREA)) {
147                     codigo = auxiliarEnviarDatosTarea(idTarea, nombreTarea,
148                             descripcionTarea, fecha);
149 
150                 }
151             }
152         }
153         return codigo;
154     }
155 
156     /**
157      * Método auxiliar para enviar los datos de la tarea.
158      *
159      * @param idTarea
160      * @param nombreTarea
161      * @param descripcionTarea
162      * @param fecha
163      * @return
164      * @throws IOException
165      * @throws InterruptedException
```

```
166      */
167     public CODIGO_ERROR auxiliarEnviarDatosTarea(
168             String idTarea, String nombreTarea, String
169             descripcionTarea,
170             String fecha) throws IOException, InterruptedException {
171         if (enviarIdTarea(idTarea) &&
172                 enviarNombreTarea(nombreTarea) &&
173                 enviarDescripcionTarea(descripcionTarea) &&
174                 enviarFechaTarea(fecha)) {
175             if (esperarPantallaMenu()) {
176                 return CODIGO_ERROR.OK;
177             }
178         }
179         return CODIGO_ERROR.NOK;
180     }
181 
182     /**
183      * Método auxiliar para enviar al emulador el ID de la tarea.
184      *
185      * @param idTarea
186      * @return
187      * @throws IOException
188      */
189     private boolean enviarIdTarea(String idTarea) throws IOException {
190         if (mainframe.enviarString(idTarea)) {
191             if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
192                 return true;
193             }
194         }
195         return false;
196     }
197 
198     /**
199      * Método auxiliar para enviar al emulador el nombre de la tarea.
200      *
201      * @param nombreTarea
202      * @return
203      * @throws IOException
204      */
205     private boolean enviarNombreTarea(String nombreTarea) throws IOException {
206         if (mainframe.enviarString(nombreTarea)) {
207             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
208                 return true;
209             }
210         }
211         return false;
212     }
213 
214     /**
215      * Método auxiliar para enviar al emulador la descripción de la tarea.
216      *
217      * @param descripcionTarea
218      * @return
219      * @throws IOException
220      */
```

```
221     private boolean enviarDescripcionTarea(String descripcionTarea)
222             throws IOException {
223         if (mainframe.enviarString(descripcionTarea)) {
224             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
225                 return true;
226             }
227         }
228         return false;
229     }
230 
231     /**
232      * Método auxiliar para enviar al emulador la fecha de la tarea.
233      *
234      * @param fecha
235      * @return
236      * @throws IOException
237      */
238     private boolean enviarFechaTarea(String fecha) throws IOException {
239         if (mainframe.enviarString(fecha)) {
240             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
241                 return true;
242             }
243         }
244         return false;
245     }
246 
247     /**
248      * Método auxiliar para esperar la pantalla de menú de la aplicación
249      * legada.
250      *
251      * @return
252      * @throws IOException
253      * @throws InterruptedException
254      */
255     private boolean esperarPantallaMenu()
256             throws IOException, InterruptedException {
257         if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
258             if (mainframe.esperarPantalla(MENU_TASKS2)) {
259                 return true;
260             }
261         }
262         return false;
263     }
264 
265     /**
266      * Verifica si existe ya una tarea con el idTarea introducido.
267      *
268      * @param idTarea
269      * @return
270      * @throws IOException
271      * @throws InterruptedException
272      */
273     private boolean existeIdTarea(String idTarea)
274             throws IOException, InterruptedException {
275         List<Tarea> tareas = listarTareas();
```

```
276         for (Tarea t : tareas) {
277             if (t.getId().equals(idTarea)) {
278                 return true;
279             }
280         }
281         return false;
282     }
283 
284     /**
285      * Opción de tasks2 para eliminar una tarea.
286      *
287      * @param idTarea
288      * @return
289      * @throws IOException
290      * @throws InterruptedException
291      */
292     @Override
293     public CODIGO_ERROR eliminarTarea(String idTarea)
294             throws IOException, InterruptedException {
295         CODIGO_ERROR codigo = CODIGO_ERROR.NOK;
296 
297         if (esperarPantallaEliminar()) {
298             if (enviarIdTarea(idTarea)) {
299                 codigo = auxiliarEliminarTarea();
300             }
301         }
302         return codigo;
303     }
304 
305     /**
306      * Método auxiliar que espera la pantalla de eliminación de una tarea.
307      *
308      * @return
309      * @throws IOException
310      * @throws InterruptedException
311      */
312     private boolean esperarPantallaEliminar()
313             throws IOException, InterruptedException {
314         if (mainframe.enviarString(ELIMINAR)) {
315             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
316                 if (mainframe.esperarPantalla(PANTALLA_ELIMINAR_TAREA)) {
317                     return true;
318                 }
319             }
320         }
321         return false;
322     }
323 
324     /**
325      * Método auxiliar para eliminar la tarea.
326      *
327      * @param idTarea
328      * @return
329      * @throws IOException
330      * @throws InterruptedException
```

```java
     */
    public CODIGO_ERROR auxiliarEliminarTarea()
            throws IOException, InterruptedException {
        if (mainframe.esperarPantalla(PANTALLA_TAREA_NO_ENCONTRADA)) {
            if (esperarPantallaMenu()) {
                return CODIGO_ERROR.IDTAREA_INCORRECTO;
            }
        } else if (esperarConfirmacionEliminar()) {
            return CODIGO_ERROR.OK;
        }
        return CODIGO_ERROR.NOK;
    }

    /**
     * Método auxiliar para esperar la confirmación de la eliminación de la
     * tarea.
     *
     * @return
     * @throws IOException
     * @throws InterruptedException
     */
    private boolean esperarConfirmacionEliminar()
            throws IOException, InterruptedException {
        if (mainframe.esperarPantalla(MENSAJE_CONFIRMAR_ELIMINAR)) {
            if (mainframe.enviarString(SI)) {
                if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
                    if (esperarPantallaMenu()) {
                        return true;
                    }
                }
            }
        }
        return false;
    }

    /**
     * Opción de tasks2 para buscar las tareas de una fecha concreta.
     *
     * @param fecha
     * @return
     * @throws IOException
     * @throws InterruptedException
     */
    @Override
    public List<Tarea> buscarTareas(String fecha)
            throws IOException, InterruptedException {
        List<Tarea> tareas = new ArrayList();
        if (esperarPantallaBuscar()) {
            if (enviarFechaTarea(fecha)) {
                if (empezarLecturaTareas()) {
                    tareas = obtenerListaTareas();
                    if (esperarPantallaMenu()) {
                        return tareas;
                    }
                }
```

```
386             }
387         }
388         return tareas;
389     }
390 
391     /**
392      * Método auxiliar que espera la pantalla de búsqueda de tareas.
393      *
394      * @return
395      * @throws IOException
396      * @throws InterruptedException
397      */
398     private boolean esperarPantallaBuscar()
399             throws IOException, InterruptedException {
400         if (mainframe.enviarString(BUSCAR)) {
401             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
402                 if (mainframe.esperarPantalla(PANTALLA_BUSCAR_TAREAS)) {
403                     return true;
404                 }
405             }
406         }
407         return false;
408     }
409 
410     /**
411      * Método auxiliar que empieza la lectura de las tareas de la aplicación
412      * legada.
413      *
414      * @return
415      * @throws IOException
416      * @throws InterruptedException
417      */
418     private boolean empezarLecturaTareas()
419             throws IOException, InterruptedException {
420         if (mainframe.esperarPantalla(FIN_LISTA)) {
421             if (mainframe.enviarComando(MainframeAPI.COMANDO_ASCII)) {
422                 return true;
423             }
424         }
425         return false;
426     }
427 
428 
429     /**
430      * Opción de tasks2 para listar las tareas.
431      *
432      * @return
433      * @throws IOException
434      * @throws InterruptedException
435      */
436     @Override
437     public List<Tarea> listarTareas()
438             throws IOException, InterruptedException {
439         List<Tarea> tareas = new ArrayList();
440         if (esperarPantallaListar()) {
```

```
441             if (mainframe.enviarComando(Mainframe.COMANDO_ASCII)) {
442                 tareas = obtenerListaTareas();
443                 if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
444                     if (mainframe.esperarPantalla(MENU_TASKS2)) {
445                         return tareas;
446                     }
447                 }
448             }
449         }
450         return tareas;
451     }
452 
453     /**
454      * Método auxiliar que espera la pantalla de listar tareas.
455      *
456      * @return
457      * @throws IOException
458      * @throws InterruptedException
459      */
460     private boolean esperarPantallaListar()
461             throws IOException, InterruptedException {
462         if (mainframe.enviarString(LISTAR)) {
463             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
464                 if (mainframe.esperarPantalla(PANTALLA_LISTAR_TAREAS)) {
465                     return true;
466                 }
467             }
468         }
469         return false;
470     }
471 
472     /**
473      * Obtiene la lista de tareas.
474      *
475      * @return
476      * @throws IOException
477      */
478     private List<Tarea> obtenerListaTareas() throws IOException {
479         String resultado = mainframe.obtenerRespuestaMaquina();
480         String[] lineas = resultado.split("\n");
481         return parsearTareas(lineas);
482     }
483 
484     /**
485      * Parsea la lista de tareas recibida del mainframe.
486      *
487      * @param resultado
488      * @return
489      */
490     private List<Tarea> parsearTareas(String[] resultado) {
491         String idTarea = "";
492         String nombre = "";
493         String descripcion = "";
494         String fecha = "";
495         List<Tarea> tareas = new ArrayList();
```

```
496
497         for (String line : resultado) {
498             if (line.matches(PATRON_IDTAREA)) {
499                 idTarea = line.replace(TEXTO_ID_TAREA, "").strip();
500             } else if (line.matches(PATRON_NOMBRE)) {
501                 nombre = line.replace(TEXTO_NOMBRE, "").strip();
502             } else if (line.matches(PATRON_DESCRIPCION)) {
503                 descripcion = line.replace(TEXTO_DESCRIPCION, "").strip();
504             } else if (line.matches(PATRON_FECHA)) {
505                 fecha = line.replace(TEXTO_FECHA, "").strip();
506                 tareas.add(new Tarea(idTarea, nombre, descripcion, fecha));
507             }
508         }
509         return tareas;
510     }
511
512     /**
513      * Opción de tasks2 para guardar las tareas.
514      *
515      * @return
516      * @throws IOException
517      * @throws InterruptedException
518      */
519     @Override
520     public boolean guardarTareas()
521             throws IOException, InterruptedException {
522         if (mainframe.enviarString(GUARDAR)) {
523             if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
524                 if (mainframe.esperarPantalla(TAREAS_GUARDADAS)) {
525                     if (mainframe.enviarComando(Mainframe.COMANDO_ENTER)) {
526                         if (mainframe.esperarPantalla(MENU_TASKS2)) {
527                             return true;
528                         }
529                     }
530                 }
531             }
532         }
533         return false;
534     }
535
536     /**
537      * Termina la comunicación con la terminal s3270.
538      *
539      * @param guardarTareas
540      * @return
541      * @throws IOException
542      * @throws InterruptedException
543      */
544     @Override
545     public boolean salir(String guardarTareas)
546             throws IOException, InterruptedException {
547         if (esperarPantallaSalir()) {
548             if (realizarSalida()) {
549                 return true;
550             } else if (esperarPantallaGuardar(guardarTareas)) {
```

```
551                 if (realizarSalida()) {
552                     return true;
553                 }
554             }
555         }
556         return false;
557     }
558 
559     /**
560      * Método auxiliar que espera la pantalla de salida de la aplicación
561      * legada.
562      *
563      * @return
564      * @throws IOException
565      * @throws InterruptedException
566      */
567     private boolean esperarPantallaSalir()
568             throws IOException, InterruptedException {
569         if (mainframe.enviarString(MainframeAPI.COMANDO_EXIT)) {
570             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
571                 return true;
572             }
573         }
574         return false;
575     }
576 
577     /**
578      * Método auxiliar que sale de la aplicación legada sin guardar cambios.
579      *
580      * @return
581      */
582     private boolean realizarSalida()
583             throws IOException, InterruptedException {
584         if (mainframe.esperarPantalla(MENSAJE_SALIDA)) {
585             if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
586                 return true;
587             }
588         }
589         return false;
590     }
591 
592     /**
593      * Método que espera la pantalla de guardado de tareas.
594      *
595      * @param guardarTareas
596      * @return
597      * @throws IOException
598      * @throws InterruptedException
599      */
600     private boolean esperarPantallaGuardar(String guardarTareas)
601             throws IOException, InterruptedException {
602         if (mainframe.esperarPantalla(PANTALLA_GUARDAR_TAREAS)) {
603             if (mainframe.enviarString(guardarTareas)) {
604                 if (mainframe.enviarComando(MainframeAPI.COMANDO_ENTER)) {
605                     return true;
```

```
606                 }
607             }
608         }
609         return false;
610     }
611 }
612
```