Wrapper

SL. Práctica 3

Diego Sanz Fuertes | 825015

11 de Enero de 2024

Índice

1 Resumen ejecutivo **2**

2 Introducción **3**

3 Herramientas utilizadas **3**

4 Paso 1: Aprender a utilizar el mainframe **3**

5 Paso 2: Crear un wrapper de x3270 **3**

6 Paso 3: Crear una interfaz **5**

7 Problemas encontrados **9**

8 Conclusiones **11**

9 Referencias **12**

1 Resumen ejecutivo

En esta práctica se ha desarrollado un wrapper sobre el emulador de terminales x3270 y se ha creado una interfaz para controlar una aplicación en un mainframe a través de éste.

Para ello se ha hecho uso de Java y su librería de interfaces Swing. El wrapper se ha implementado lanzando x3270 como un proceso desde Java y interactuando con él a traves de la entrada/salida estándar. Se ha creado un método por cada comando necesario para realizar la interfaz.

La interfaz es capaz de conectarse al mainframe, iniciar sesión, abrir el programa y realizar todas sus posibles acciones. Esto se ha conseguido mediante el uso del wrapper previamente implementado y lógica que se basa en el estado actual de la pantalla, como si existen más paginas en un listado.

Los problemas encontrados se deben a la latencia entre el cliente y el servidor y han podido ser mitigados insertando retardos donde era necesario.

2 Introducción

El objetivo de la práctica consiste en la creación de un wrapper sobre el emulador de terminales x3270 para interactuar con un mainframe que utiliza el protocolo 3270. Una vez creado el wrapper se pide crear una interfaz sobre un programa ejecutado en el mainframe mediante el wrapper.

El wrapper creado es sobre el emulador x3270. Se podría crear otra capa de abstracción y crear otro wrapper sobre el wrapper que incluya la posibilidad de añadir otros emuladores de otros protocolos. Se ha considerado que esto es innecesario para esta práctica y, si fuese necesario, sería sencillo de implementar en un futuro.

3 Herramientas utilizadas

Para la realización de esta práctica se han utilizado las siguientes herramientas:

- x3270: Un emulador de la terminal IBM 3270.
- Java 19 y swing.
- IntelliJ IDEA: IDE y editor de interfaces.

Herramientas no utilizadas: filipesimoes/j3270: un wrapper sobre x3270 para Java.

Las herramientas utilizadas para la realización de la memoria han sido Obsidian y pandoc.

4 Paso 1: Aprender a utilizar el mainframe

El primer paso para realizar la práctica ha sido aprender como funciona el mainframe. A través del emulador x3270, se ha realizado una conexión al mainframe siguiendo los pasos indicados en la práctica.

Una vez comprendido como funciona, se ha utilizado s3270, la version “headless” del emulador en el que el mainframe se controla mediante comandos como “connect”, “string” y “ascii”.

5 Paso 2: Crear un wrapper de x3270

Para este paso se ha creado un paquete en Java con el fin de encapsular el comportamiento del emulador, permitiendo llamar a los comandos desde Java para poder comunicarse con el Mainframe.

Para interactuar con s3270, se ha utilizado la clase `Process` la cual permite ejecutar un programa y leer/escribir su entrada/salida estándar. Para leer se ha utilizado `BufferedReader` y para escribir `Printwriter`.

```
Process process = new ProcessBuilder()
                .command(ws3270_path)
                .start();
Printwriter writer = new PrintWriter(process.outputWriter(), true);
BufferedReader reader = process.inputReader();
```

Se han implementado métodos para interactuar con la entrada/salida de manera mas cómoda y proveer información de depuración. Estos métodos son: `printfln`, `ignoreLine`, `readLine` y `emptyReader`. Para gestionar los errores se ha implementado un método `throwIfNotOk` que, si s3270 no devuelve "ok", lanza una excepción y vacía el reader.

Una vez implementados estos métodos, se ha procedido a implementar los diferentes comandos que se han utilizado de s3270. Como se puede observar a continuación, los comandos que no devuelven datos se han implementado de manera similar.

```
public void string(String string) throws IOException {
    printfln("string(\"%s\")", string);
    throwIfNotOk(format("Couldn't use string(\"%s\")", string));
}

public void enter() throws IOException {
    printfln("enter");
    throwIfNotOk("Couldn't enter");
}

public String ascii() throws IOException {
    int lines = 43;
    printfln("ascii");
    var sb = new StringBuilder();

    for (int i = 0; i < lines - 1; i++) {
        sb.append(reader.readLine().replace("data: ", ""));
        sb.append("\n");
    }
    sb.append(reader.readLine().replace("data: ", ""));

    throwIfNotOk("Couldn't ascii");
    return sb.toString();
}
```

Por último se han creado métodos que combinan varios comandos para aumentar la legibilidad del código. Por ejemplo:

```
public String waitAscii() throws IOException {
```

```
    waitOutput();
    waitSeconds(0.5);
    return ascii();
}

public void stringEnterWait(String string) throws IOException {
    string(string);
    enter();
    waitSeconds(0.3);
}
```

6 Paso 3: Crear una interfaz

Una vez desarrollado el wrapper, se ha procedido a crear una interfaz de depuración y otra interfaz del programa de tareas mediante Java Swing. La interfaz de depuración ha resultado realmente útil para navegar por el mainframe, validar el wrapper, calcular tiempos de espera y salir de errores al utilizar la interfaz del programa de tareas.

Para la interfaz del programa de tareas se ha utilizado el IDE IntelliJ IDEA, que permite posicionar los elementos de forma gráfica sin tener que compilar y utiliza xml para describir las interfaces, lo que hace que el código Java resultante sea mas limpio.

Una vez diseñadas las interfaces, se le ha añadido la funcionalidad al abrirse y a los botones. Al abrirse el programa se intenta conectar a la dirección IP a través del emulador s3270, ambas rutas definidas como constantes en el archivo fuente principal.

```
// MainForm::MainForm()
...
try {
    w = new Wrapper(WS3270_PATH);
} catch (Exception e) {
    JOptionPane.showMessageDialog(this.mainPanel,
            "Error loading ws3270 in path: " + WS3270_PATH);

    System.exit(0);
}

try {
    // Connect
    w.connect(IP);
} catch (Exception ex) {
    JOptionPane.showMessageDialog(this.mainPanel,
            "Cannot connect to " + IP);

    System.exit(0);
}
```

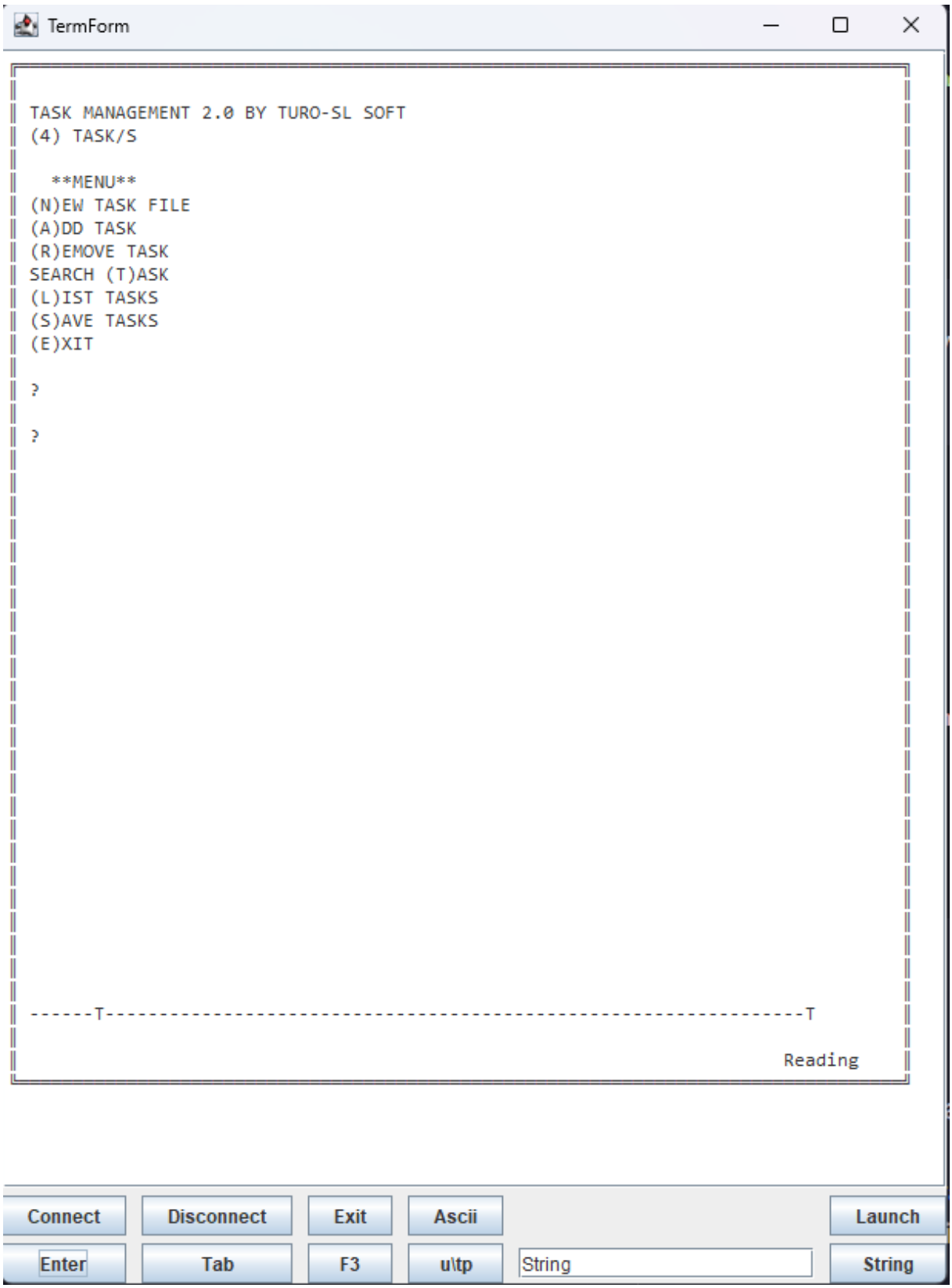

Figura 1: Interfaz de depuración

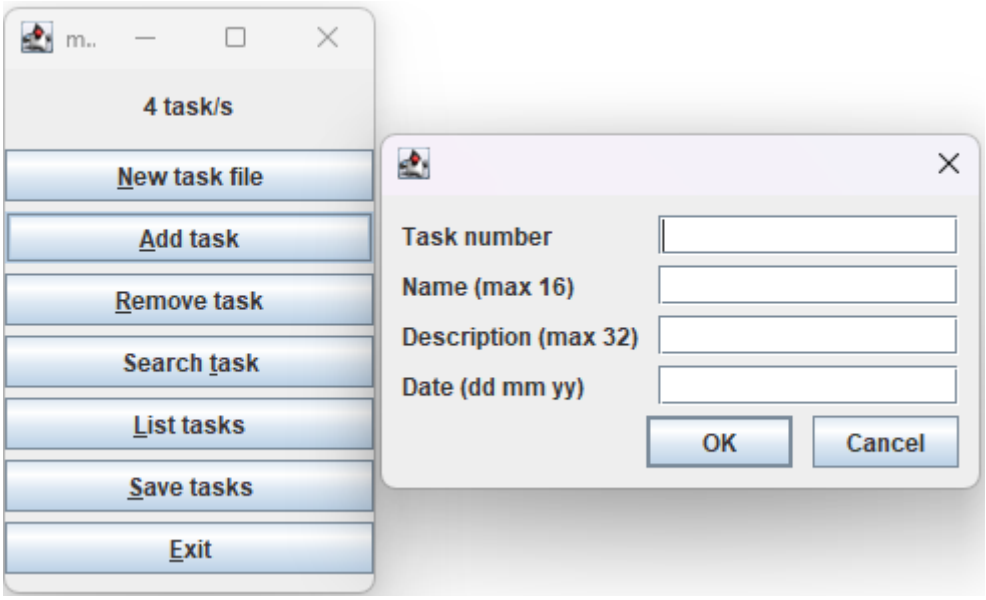

Figura 2: Interfaz de programa de tareas

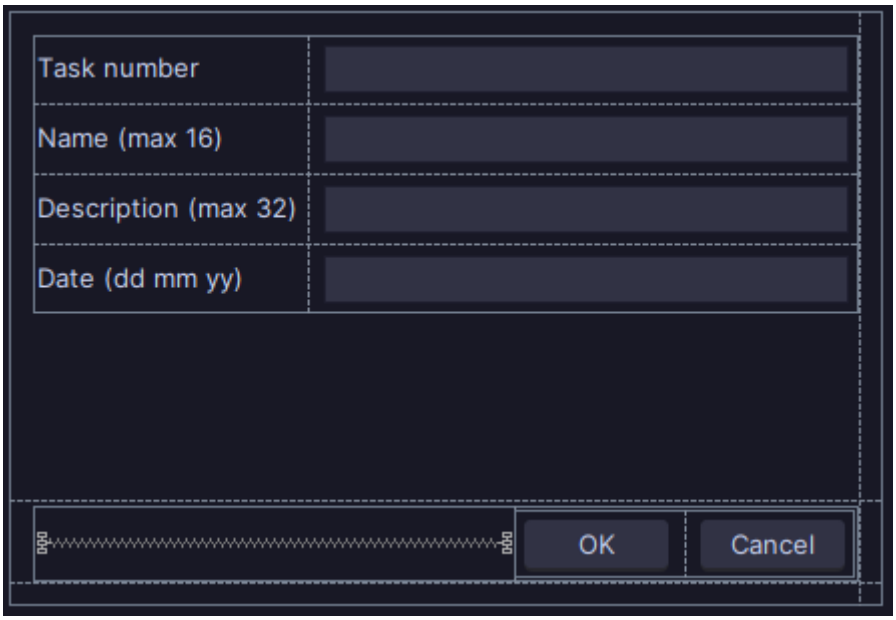

Figura 3: Crear tareas IDE

```
21
22 try {
23     // Login
24     w.enter();
25     w.waitSeconds(0.3);
26     w.string("prog");
27     w.tab();
28     w.waitSeconds(0.3);
29     w.stringEnterWait("prog123");
30     w.enter();
31     w.waitSeconds(0.3);
32     // Open the program
33     w.stringEnterWait("tasks2.job");
34
35     updateTaskNumber();
36
37 } catch (Exception ex) {
38     JOptionPane.showMessageDialog(this.mainPanel,
39             "Error while logging-in");
40 }
```

El botón de añadir tarea abre una ventana en la que se pueden introducir los valores de ésta y son validados antes de devolverlos a la anterior ventana. En el caso de que la clave esté duplicada, se volverá a mostrar la ventana para su modificación.

```
1  // addTaskButton.addActionListener
2  var addTaskDialog = new AddTaskDialog(null);
3  addTaskDialog.setLocationRelativeTo(this.mainPanel);
4
5  addTaskDialog.setVisible(true);
6  var task = addTaskDialog.getTask();
7  if (task != null) {
8      try {
9          w.stringEnterWait("a");
10         w.stringEnterWait(String.valueOf(task.number()));
11
12         while (!w.ascii().contains("TASK NAME (MAX 16 CAR):")) {
13
14             w.enter();
15             JOptionPane.showMessageDialog(this.mainPanel,
16                     "TASK NUMBER REPEATED");
17             addTaskDialog.setVisible(true);
18             task = addTaskDialog.getTask();
19
20             w.stringEnterWait(String.valueOf(task.number()));
21         }
22         w.stringEnterWait(task.name());
23         w.stringEnterWait(task.description());
24         w.stringEnterWait(task.date());
25
```

```
26          w.enter();
27          w.waitSeconds(0.3);
28          updateTaskNumber();
29
30      } catch (Exception ex) {
31          JOptionPane.showMessageDialog(this.mainPanel,
32                      "Error adding task");
33      }
34  }
```

El botón de eliminar invoca una ventana modal que pide al usuario el id de la tarea a ser eliminada. Los botones de búsqueda y de listado son muy similares, ya que los datos se visualizan de la misma manera, una ventana modal y tienen en cuenta la paginación cuando existen muchas tareas, su única diferencia es que la de búsqueda requiere de un modal para introducir el criterio de búsqueda, necesario también para obtener el punto de inicio de la lista.

```
1  // searchTaskButton.addActionListener
2  try {
3      w.stringEnterWait("t");
4      String input = JOptionPane.showInputDialog(this.mainPanel,
5              "Enter date in format dd mm yy");
6      while (input.isEmpty()) {
7          input = JOptionPane.showInputDialog(this.mainPanel,
8                  "Enter date in format dd mm yy");
9      }
10     w.stringEnterWait(input);
11
12     var listOfTasks = parseListOfTasks(input);
13
14     if (listOfTasks.isBlank())
15         listOfTasks = "Tasks not found";
16     JOptionPane.showMessageDialog(this.mainPanel, listOfTasks);
17
18 } catch (Exception ex) {
19     JOptionPane.showMessageDialog(this.mainPanel,
20                 "Error searching task");
21 }
```

Por último el botón de nuevo archivo, guardado y salida son una serie de comandos preestablecidos los cuales no varían. También se ha incluido un contador del numero de tareas, obtenido cada vez que se realiza una acción que lo pueda modificar.

7 Problemas encontrados

Algunos de los problemas encontrados han sido los siguientes:

- Al intentar iniciar sesión y otras acciones seguidas en el mainframe sin esperas para hacer la aplicación mas rápida, se obtenían resultados incorrectos y no deterministas, aun con el uso del comando `wait(output)`. La solución ha sido añadir 0.3 segundos de retardo cada vez que se realiza un comando `enter` o `tab` y se quiere realizar otro input.
- El mainframe se quedaba bloqueado en el caso de cerrar la conexión sin salir del programa o, a veces, de la sesión del sistema operativo.

8 Conclusiones

- Tras la realización de esta práctica se ha apreciado aun más los wrappers que hacen los programadores para utilizar librerías escritas en otros lenguajes como OpenSSL o SDL.
- Resulta muy satisfactorio escribir un wrapper sobre una librería o comando de esta manera.
- Se debería utilizar una librería existente en vez de una propia si es posible, en este caso existía otro wrapper para Java de x3270. No se ha utilizado porque va en contra del objetivo de la práctica
- Es un paradigma curioso de programación, en vez de un modelo MVC, la vista es el modelo, esto trae posibles desincronizaciones del estado entre la vista y el controlador.

9 Referencias

- x3270 http://x3270.bgp.nu/