Command Graphical Interface

Descripción

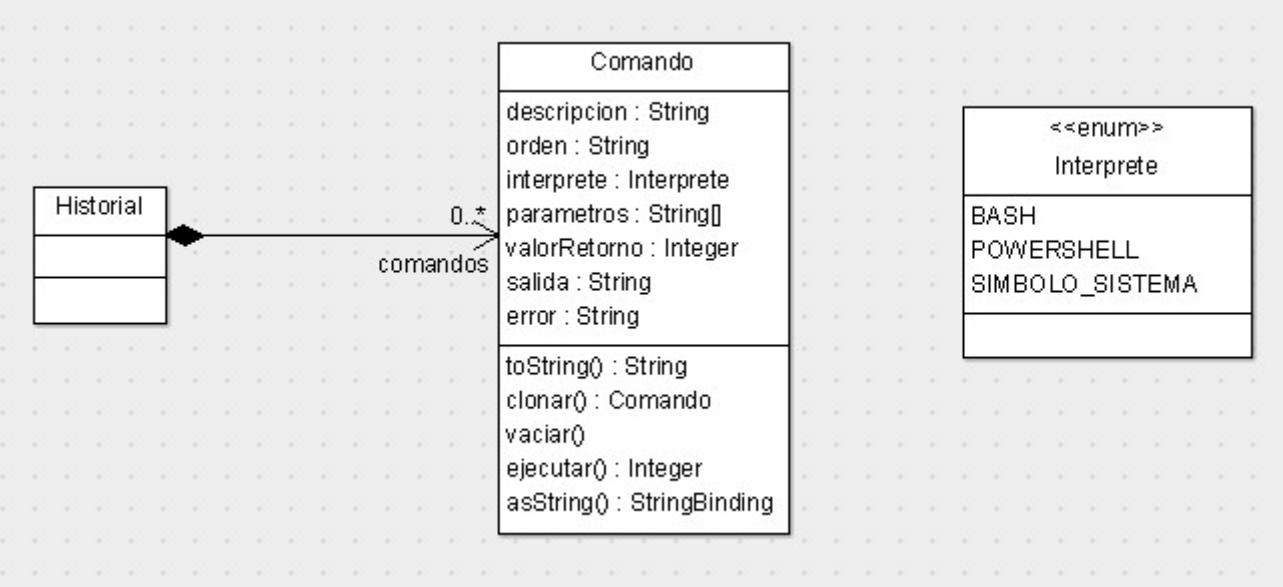
Implementar una aplicación con GUI empleando el framework JavaFX que permita la ejecución sencilla de comandos para diferentes intérpretes (BASH, PowerShell y Símbolo del sistema), así como guardar los comandos en un historial, para poder volver a ejecutarlos.

Se deberá aplicar el patrón de diseño MVC (Modelo-Vista-Controlador), y las vistas deberán implementarse en FXML.

Así mismo, será posible guardar/abrir el modelo de beans FX en/desde un fichero XML con extensión ".history".

Modelo de datos

El modelo de datos que deberá gestionar la aplicación es el siguiente:



Las clases del modelo de datos serán provistas por el profesor en el paquete "dad.cgi.model".

El significado de cada clase es el siguiente:

- **Historial**: Elemento raíz del modelo. Se utiliza como contenedor de comandos, y para guardar y cargar el modelo desde ficheros XML.
  - comandos [0..*]: Listado de comandos agregados al historial.
- **Comando**: Representación de un comando con sus parámetros que puede ser ejecutado y guarda el resultado de la ejecución.
  - descripcion: Texto descriptivo del comando.
  - orden: Comando/ejecutable/binario que se ejecutará.
  - interprete: Intérprete de comandos que se utilizará para ejecutar el comando.

    - parametros [0..*]: Listado de parámetros que recibirá el comando al ejecutarse.
    - valorRetorno: valor entero devuelto por el comando al terminar su ejecución.
    - salida: texto de salida que generó el comando durante su ejecución.
    - error: texto de salida generado por el comando durante la ejecución en caso de error.
    - toString(): devuelve la cadena completa que se enviará al intérprete para la ejecución del comando.
    - clonar(): devuelve una copia exacta del comando (es una instancia nueva)
    - vaciar(): resetea todas las propiedades del comando y vacía la lista de parámetros.
    - ejecutar(): ejecuta el comando en el intérprete indicado y devuelve el valor de retorno.
    - asString(): binding que devuelve lo mismo que el método "toString()" que se actualiza cuando cambia la "orden" o la lista de "parámetros". Con el resto de propiedades no se ve afectado.
- **Interprete**: Enumerado con los distintos intérpretes de comandos soportados.
    - BASH: Intérprete de comandos de los sistemas GNU/Linux.
    - POWERSHELL: Intérprete de comandos PowerShell de Windows.
    - SIMBOLO_SISTEMA: Intérprete de comandos clásico de Windows.

Ejemplo de uso del modelo de datos

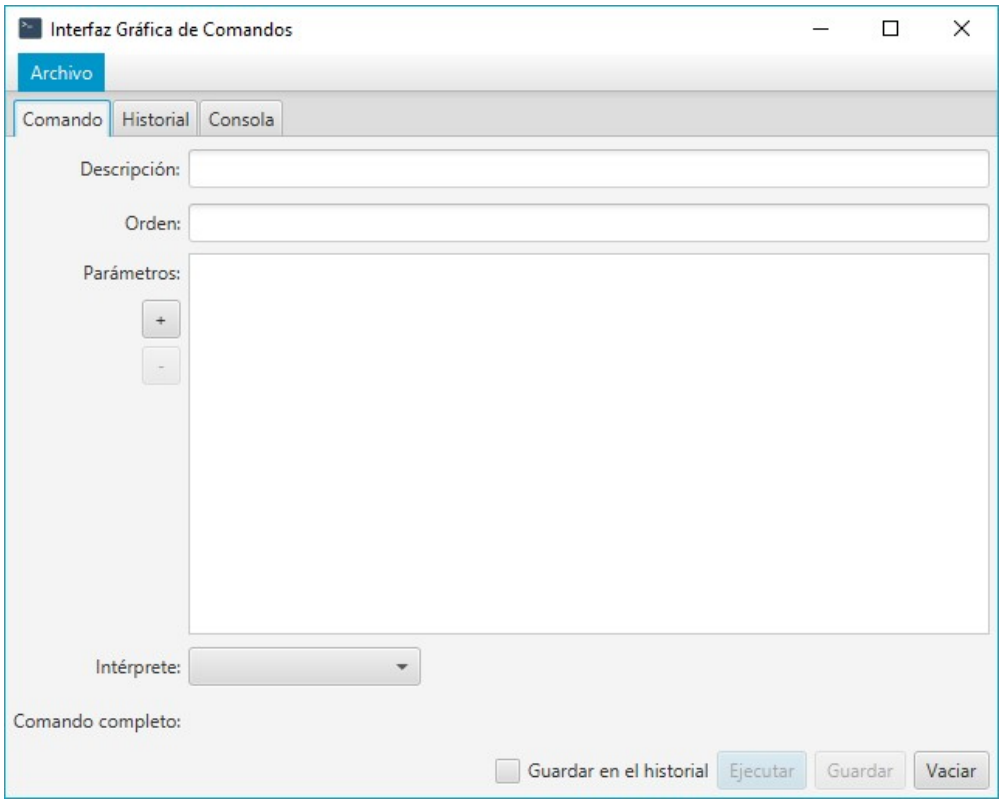
El siguiente código crea 3 comandos, los añade a un historial, lo guarda en un fichero y lo vuelve a cargar:

```
/*
 * Crea el comando 'netstat' con varios parámetros y lo ejecuta en el Símbolo del Sistema (cmd)
 */
Comando dir = new Comando();
dir.setDescripcion("Listar el contenido de la raíz de la unidad C:");
dir.asString().addListener((o, ov, nv) -> {
                System.out.println("Cambio detectado en el comando: " + nv);
        });

dir.setInterprete(Interprete.SIMBOLO_SISTEMA);
dir.setOrden("dir");
dir.getParametros().add("c:\\");
dir.ejecutar();

System.out.println("Comando: " + dir);
System.out.println("Salida:  " + dir.getSalida());
System.out.println("Error:   " + dir.getError());
System.out.println("Retorno: " + dir.getValorRetorno());

System.out.println("============================================================");

/*
 * Crea el comando 'netstat' con varios parámetros y lo ejecuta en el Símbolo del Sistema (cmd)
 */

Comando netstat = new Comando();
netstat.setDescripcion("Muestra los puertos abiertos");
netstat.setInterprete(Interprete.SIMBOLO_SISTEMA);
netstat.setOrden("netstat");
netstat.getParametros().add("-a");
netstat.getParametros().add("-n");
netstat.getParametros().add("-o");
netstat.getParametros().add("-p tcp");
netstat.ejecutar();
System.out.println("Comando: " + netstat);
System.out.println("Salida:  " + netstat.getSalida());
System.out.println("Error:   " + netstat.getError());
System.out.println("Retorno: " + netstat.getValorRetorno());

System.out.println("============================================================");

/*
 * Crea el comando 'Get-Process -Id 0' y lo ejecuta en PowerShell
 */

Comando gp = new Comando();
```

```
gp.setDescripcion("Muestra información sobre el proceso con PID 0");
gp.setInterprete(Interprete.POWERSHELL);
gp.setOrden("Get-Process");
gp.getParametros().add("-Id 0");
gp.ejecutar();
System.out.println("Comando: " + gp);
System.out.println("Salida:  " + gp.getSalida());
System.out.println("Error:   " + gp.getError());
System.out.println("Retorno: " + gp.getValorRetorno());

System.out.println("============================================================");

/*
 * Crea un historial nuevo, le añade los 3 comandos anteriores y lo guarda
 */

Historial historial = new Historial();
historial.getComandos().add(dir);
historial.getComandos().add(netstat);
historial.getComandos().add(gp);
historial.save(new File("test.history"));
System.out.println("Historial guardado");

/*
 * Lee el historial desde fichero
 */
historial = Historial.load(new File("test.history"));
System.out.println("Historial cargado");
```

La interfaz de la aplicación

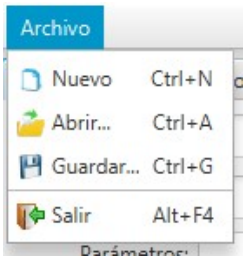
La aplicación contará con una única ventana (Stage), que tendrá el siguiente aspecto:



Puede implementarse con un único controlador y un único fichero FXML para la vista.

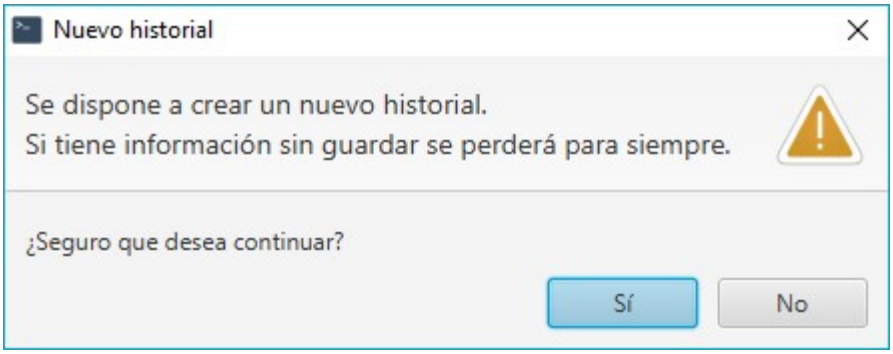
Barra de menú

En la parte superior tendrá una barra de menú con el menú "Archivo":

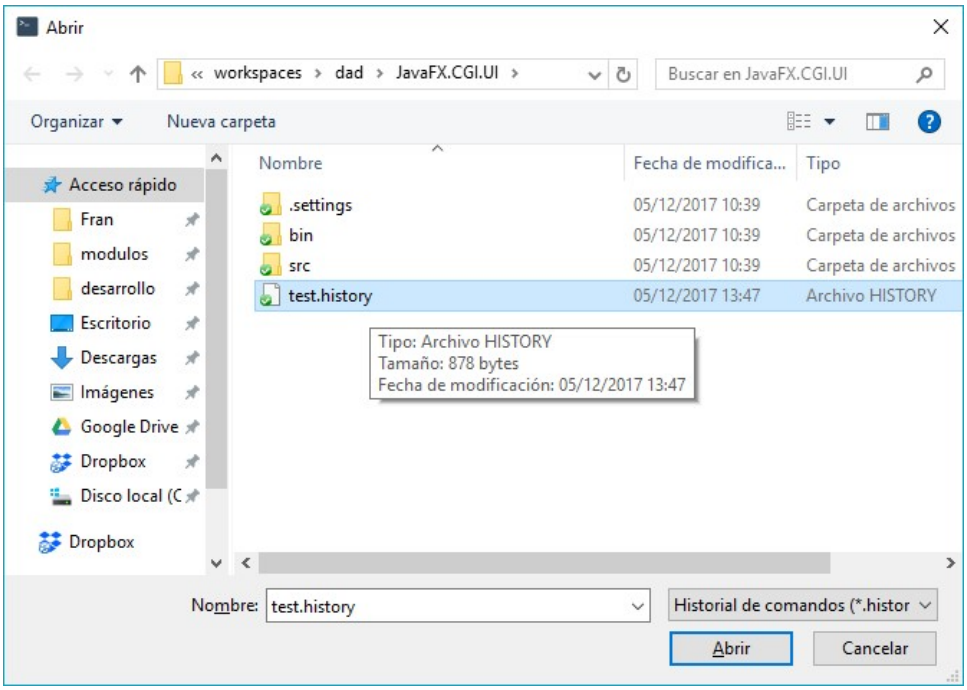


A continuación se explica lo que hace cada opción del menú:

- **Nuevo**:
  - Se creará un nuevo historial y se sobreescribirá el anterior.
  - Se deberá pedir confirmación antes de crear el historial nuevo con un diálogo como el siguiente:

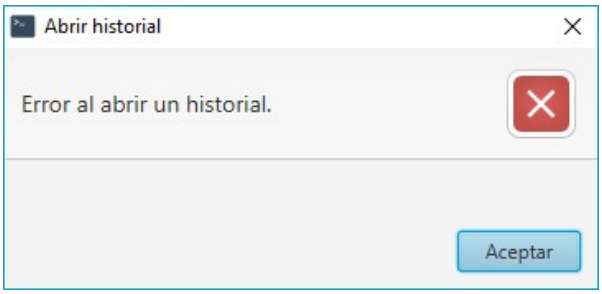


- **Abrir…:**
  - Se abrirá un historial desde un fichero con extensión ".history".
  - Se abrirá un "`FileChooser`" para elegir el fichero a abrir:

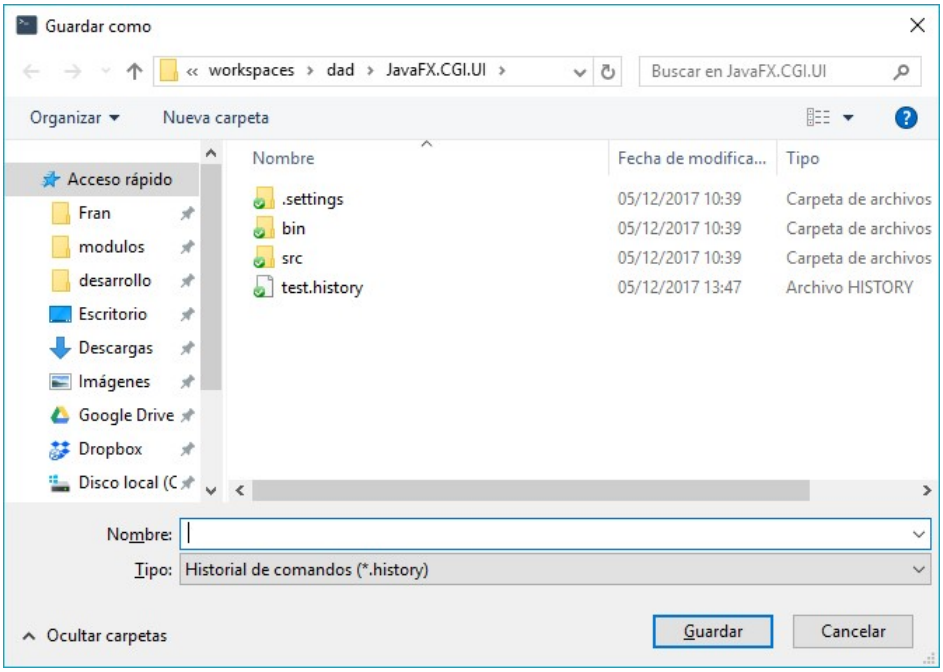


  - El "FileChooser" se deberá configurar de la siguiente forma:
    - Deberá tener dos filtros (`ExtensionFilter`):
      - *Historial de comandos (*.history)*
      - *Todos los archivos (*.*)*
    - Deberá tener el icono de la aplicación.

- Para abrir el modelo disponemos del método estático "`Historial.load(File fichero)`".
- Se deberá mostrar la pestaña "Historial" en la ventana principal si el fichero se carga bien.
- En caso de error, se mostrará un mensaje como el siguiente, que deberá incluir el mensaje de la excepción en el contenido:



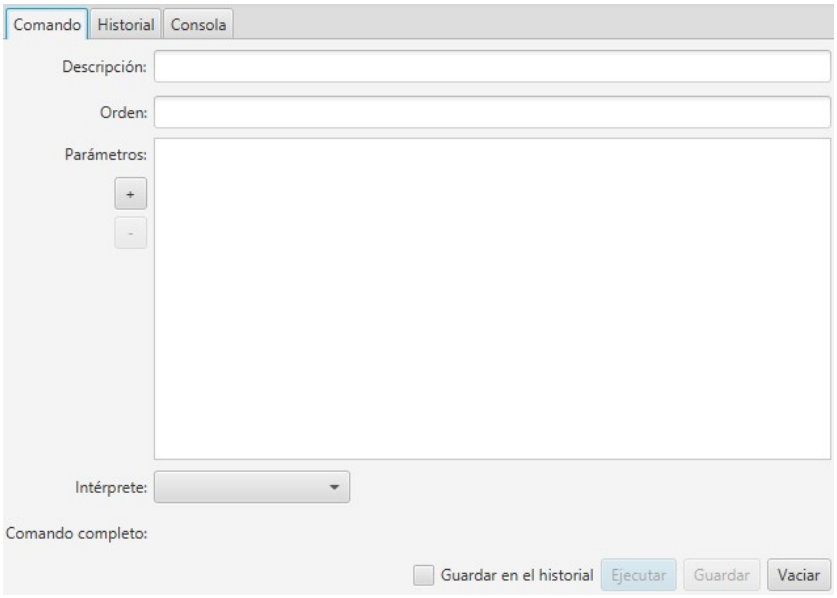
- **Guardar modelo**:
  - Se guardará el historial de comandos actual en un fichero ".history".
  - Se abrirá un "`FileChooser`" para elegir el fichero a guardar:



  - El "FileChooser" se deberá configurar de la siguiente forma:
    - Deberá tener dos filtros (`ExtensionFilter`):
      - *Historial de comandos (*.histgory)*
      - *Todos los archivos (*.*)*
    - Deberá tener el icono de la aplicación.
  - Para guardar el modelo disponemos del método "`save(File fichero)`" del historial.
  - En caso de error, se mostrará un mensaje (Alert) con la causa del error (mensaje de la excepción).

Pestaña "Comando"

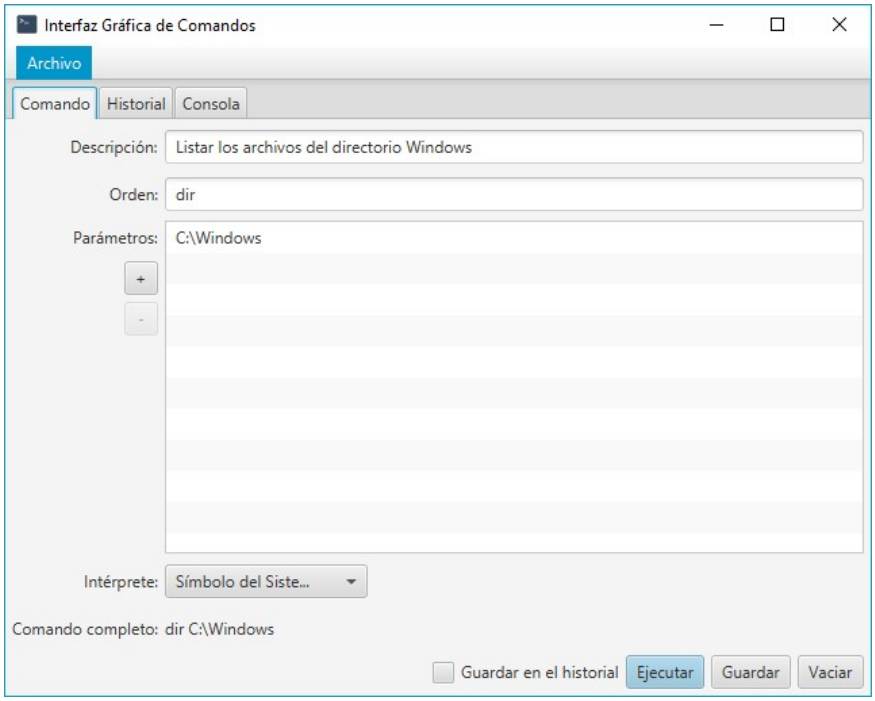
Desde aquí se crean los comandos nuevos, se pueden ejecutar y guardar en el historial.



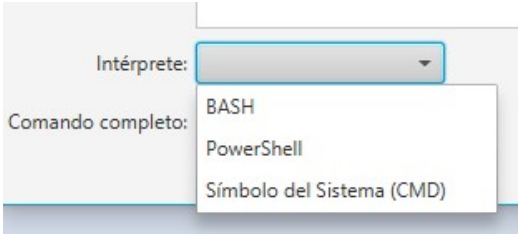
Es recomendable crear una instancia de "Comando" y bindearla a los componentes de la interfaz gráfica de esta pestaña para recoger los datos de los comandos.

"Descripción" y "Orden" son cuadros de texto, "Parámetros" es una lista (ListView), "Intérprete" es una lista desplegable (ComboBox), y "Comando completo" es una etiqueta.

A continuación un ejemplo del formulario relleno con datos de un comando:



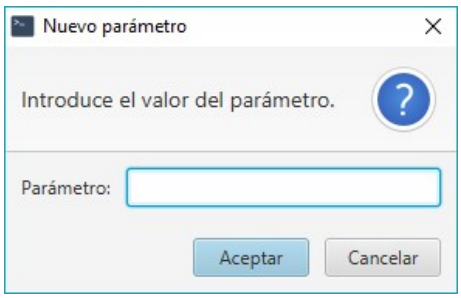
La lista desplegable "Intérprete" se deberá rellenar con los valores del enumerado del mismo nombre (`Interprete.values()`)



La etiqueta “Comando completo” irá cambiando según modifiquemos la orden y los parámetros del comando (usar propiedad “asString” del comando para esto).

Los botones de esta pestaña hacen lo siguiente:

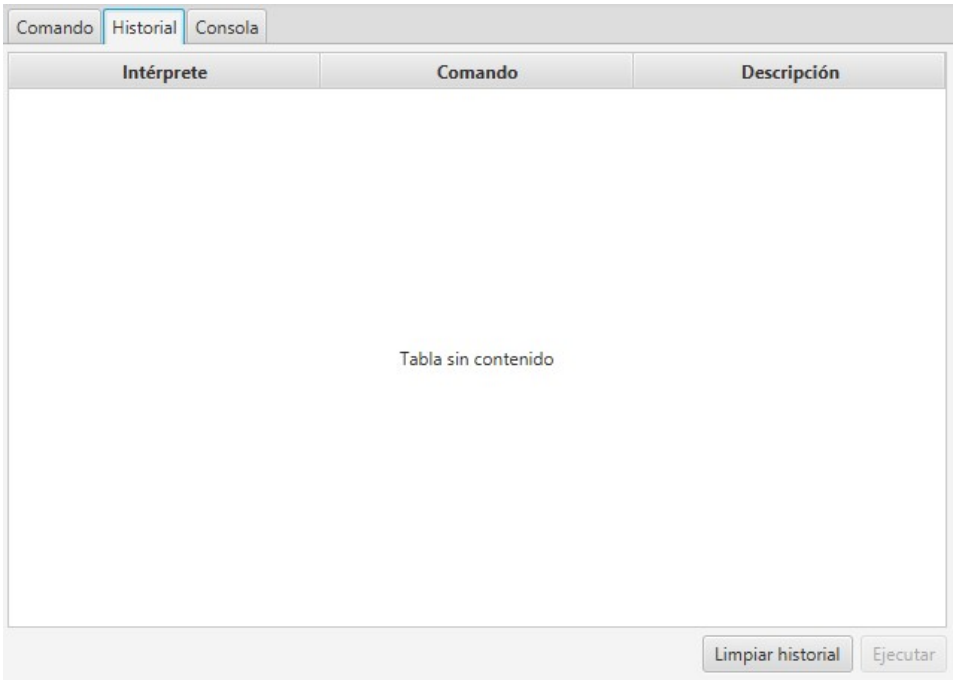
- **Más (+)**:
  - Abre un “InputTextDialog” como el siguiente para añadir un parámetro al comando:



  - Si se “Acepta” se añade, y si se “Cancela” o se cierra el diálogo no se hace nada.
- **Menos (-)**:
  - Elimina el parámetro seleccionado de la lista.
  - Si no hay ninguno seleccionado deberá deshabilitarse.
  - No es necesario pedir confirmación.
- **Ejecutar**:
  - Ejecuta el comando actual y cambia a la pestaña “Consola”, donde se deberá mostrar el resultado del comando (salida).
  - Si se activó “Guardar en el historial”, tras ejecutar el comando se clonará y se añadirá al historial.
  - Si la “Orden” está vacía y no se ha seleccionado “Intérprete”, deberá deshabilitarse el botón.
- **Guardar**:
  - Clona el comando actual, lo añade al historial y cambia a la pestaña “Historial”, sin ejecutar el comando.
  - Si la “Orden” está vacía y no se ha seleccionado “Intérprete”, deberá deshabilitarse el botón.
- **Vaciar**: Resetea el comando. La clase “Comando” dispone del método “vaciar()” para hacer esto.

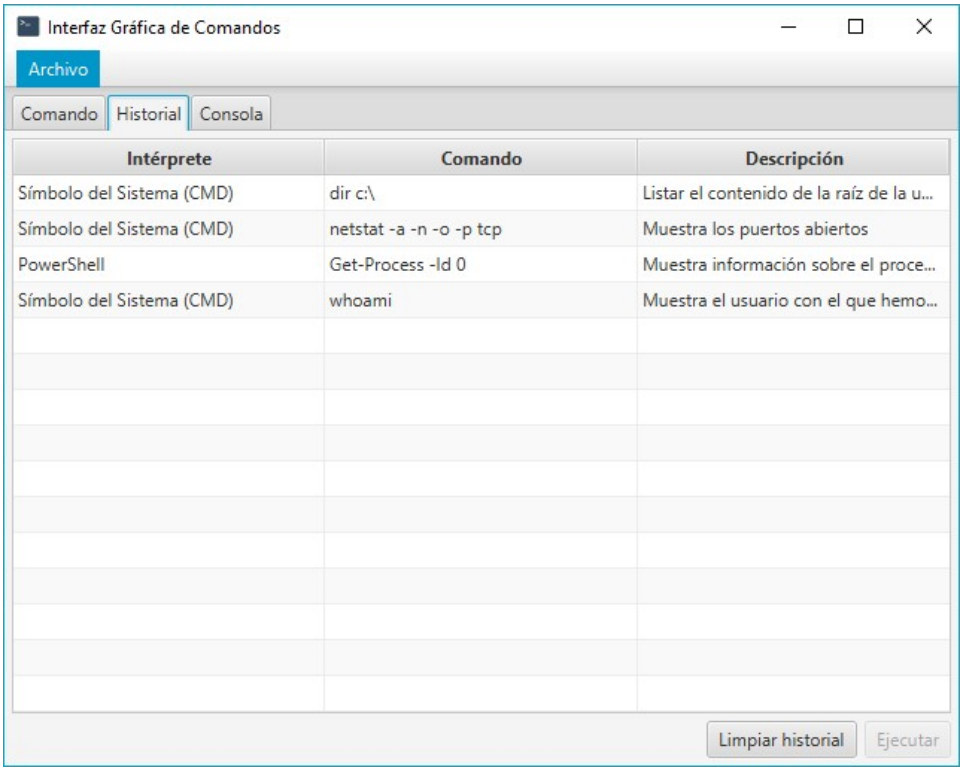
Pestaña “Historial”

Desde aquí podemos consultar el historial de comandos así como ejecutarlos.



Aquí debemos bindear la lista de comandos del historial a la tabla.

A continuación un ejemplo con datos en la tabla:



De cada comando se mostrará su propiedad “interprete”, “asString” y “descripcion”.

Los botones de esta pestaña hacen lo siguiente:

- **Limpiar historial**:
  - Vacía la lista de comandos del historial.
  - No es necesario pedir confirmación.

- **Ejecutar**:
    - Ejecuta el comando seleccionado y cambia a la pestaña “Consola”, donde se deberá mostrar el resultado del comando (salida).
    - Si no se ha seleccionado ningún comando, deberá deshabilitarse el botón.

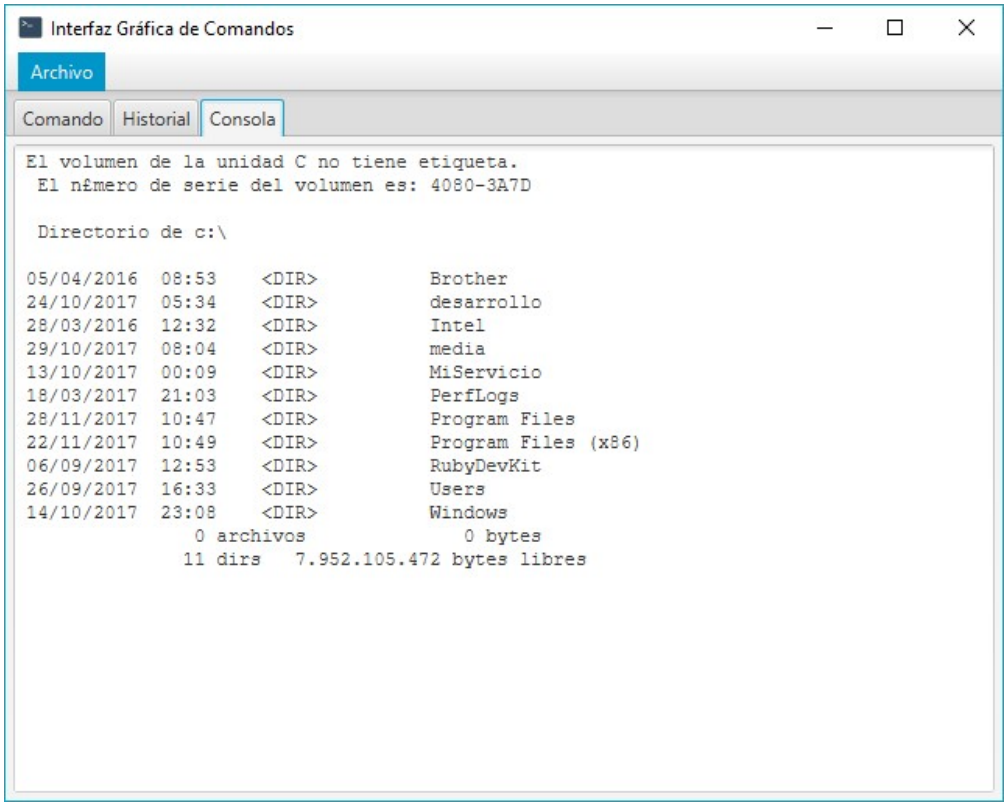
Pestaña "Consola"

Desde aquí vemos la salida del último comando ejecutado.

Simplemente incluirá un cuadro de texto multilínea (TextArea). Deberá aplicarse el siguiente estilo en línea al cuadro de texto de la siguiente forma o desde el mismo FXML:

```
consolaText.setStyle("-fx-font-family: monospace");
```

Esta pestaña no tiene ninguna funcionalidad; sólo mostrará el resultado del último comando ejecutado, como puede verse a continuación, tras ejecutar el comando "`dir c:\`":



Criterios de calificación

Criterios	Puntuación
Ventana principal	
Diseño	5
Opciones del menú	
Nuevo historial	5
Abrir historial + Cambiar a “Historial”	8
Guardar historial	4
Pestaña “Comando”	
Diseño	5
Añadir parámetro	3
Eliminar parámetro	2
Cargar lista desplegable “Intérprete”	3
Ejecutar comando + Guardar + Cambiar a “Consola”	14
Guardar comando en historial	10
Vaciar comando	4
Pestaña “Historial”	
Diseño + Listar comandos	5
Limpiar historial	3
Ejecutar comando + Cambiar a “Consola”	14
Pestaña “Consola”	
Diseño	5
Mostrar salida último comando	10
Total	100

El profesor determinará el grado de cumplimiento de cada apartado a partir de la ejecución de la aplicación, por lo que es sumamente importante que los apartados que queremos que puntúen se puedan probar.