

## Práctica 2.2 Profundizando en el Diseño Visual

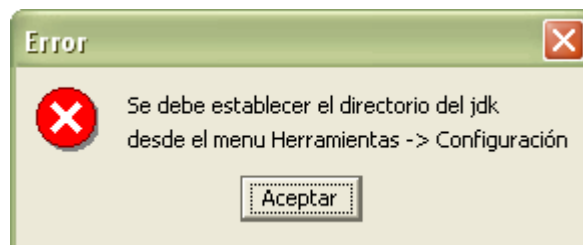
En la primera sesión de esta práctica hemos visto cómo podemos desarrollar una aplicación que contenga elementos visuales que la hacen más atractiva y más fácil de manejar por parte del usuario. Y en este segundo apartado profundizaremos un poco más en el diseño visual. Como objetivo de esta sesión se pretende que el alumno tome soltura en el diseño de aplicaciones visuales incorporando a las mismas componentes que las enriquezcan y dinamicen.

### Trabajar con Diálogos

En el desarrollo de aplicaciones se hace muy necesaria la comunicación entre el usuario y la aplicación, y que ésta comunicación se realice mediante una interfaz que sea amigable y a la fácil de comprender y de utilizar. Una forma que tenemos en Java para lograr este objetivo es mediante el uso de diálogos. Estos componentes visuales nos permiten ser entrada y salida de datos de nuestra aplicación. A través de los diálogos el usuario podrá obtener mucha información y a su vez comunicar información y elecciones a la aplicación.

Algunos de los diálogos más utilizados en las aplicaciones, como abrir o guardar ficheros, mensajes de error e informativos, etc, los podemos producir a partir de las clases `JFileChooser`, `JOptionPane` o la superclase genérica `JDialog` que nos permite personalizar el diálogo que deseemos mostrar.

En la figura podemos ver un diálogo que nos está mostrando un error:



Consultar la siguiente página para más información sobre cómo trabajar con diálogos: <http://java.sun.com/docs/books/tutorial/uiswing/components/dialog.html>

### La clase `JOptionPane`

Esta clase nos permite mostrar una serie de diálogos preestablecidos que pueden tener distinto carácter:

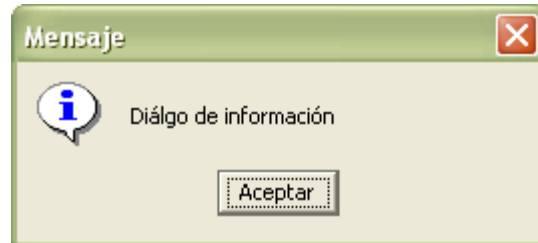
- Informativos
- Mensajes de error
- Mensajes de advertencia
- Elecciones
- Entrada de datos
- Etc.

El principal método que nos interesa a nosotros serán aquellos de la forma showXXXDialog y las XXX varían según el carácter del diálogo que deseemos. A continuación veremos alguno de estos métodos.

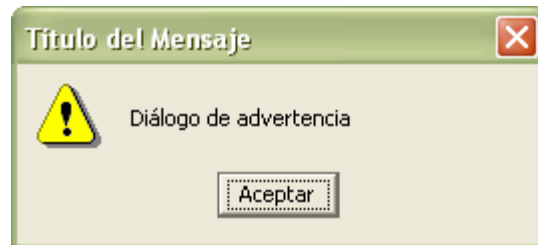
**El método showMessageDialog**

Este método nos permite mostrar diálogos que muestran un mensaje y contienen un botón de aceptación. Los parámetro mínimos necesarios dependen del carácter del mensaje aunque general mente son la ventana padre, el mensaje a mostrar, el título del diálogo y el tipo de mensaje que se mostrará. En las siguientes figuras podremos ver algunos ejemplos junto al código que lo genera.

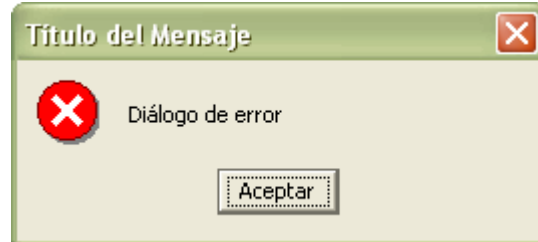
JOptionPane.showMessageDialog(ventana, "Diálogo de información");



JOptionPane.showMessageDialog(ventana, "Diálogo de advertencia", "Título del Mensaje", JOptionPane.WARNING\_MESSAGE);



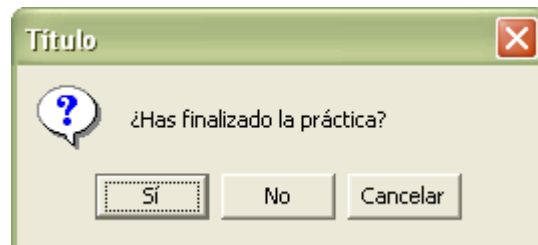
JOptionPane.showMessageDialog(ventana, "Diálogo de error", "Título del Mensaje", JOptionPane.ERROR\_MESSAGE);



**showOptionDialog**

Nos muestra un diálogo con los botones, iconos, mensajes, título, etc. que nosotros deseemos. A través de este método podremos cambiar el texto que aparece en los botones de los diálogos preestablecidos, así como otra serie de cambios. En las siguientes figuras veremos algunos ejemplos.

JOptionPane.showOptionDialog(this, "¿Has finalizado la práctica?", "Título", JOptionPane.YES\_NO\_CANCEL\_OPTION, JOptionPane.QUESTION\_MESSAGE, null, null, null);

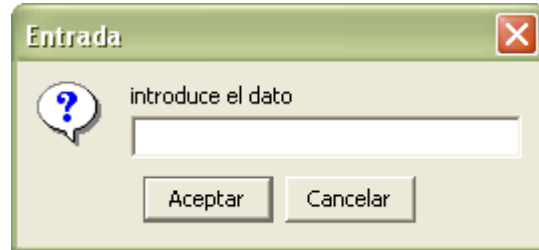


Este tipo de diálogos nos devolverá un valor entero con el que podremos conocer cual de los botones ha sido pulsado por el usuario. Se puede usar otro método **showConfirmDialog** con el que obtendríamos resultados parecidos.

### El método showInputDialog

Este método nos permite mostrar diálogos estandarizados que nos piden que introduzcamos algún dato. Al igual que en los métodos anteriores los argumentos que utilizan principalmente son el mensaje a mostrar, el título, etc. A continuación vemos un ejemplo de uso de este método.

```
String n =
JOptionPane.showInputDialog(this,
"Introduce el dato");
```



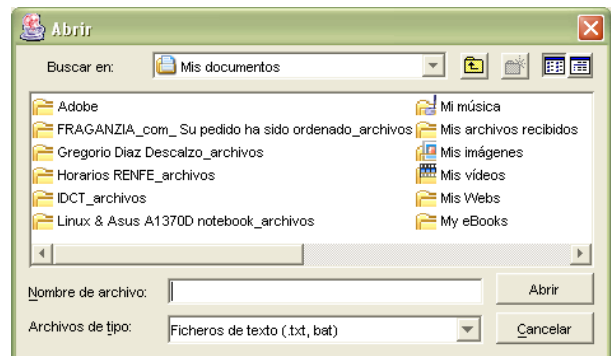
Como podemos observar en el código anterior este método nos devuelve una cadena que contiene los datos que hemos introducido en el diálogo.

### La clase JFileChooser & FileFilter

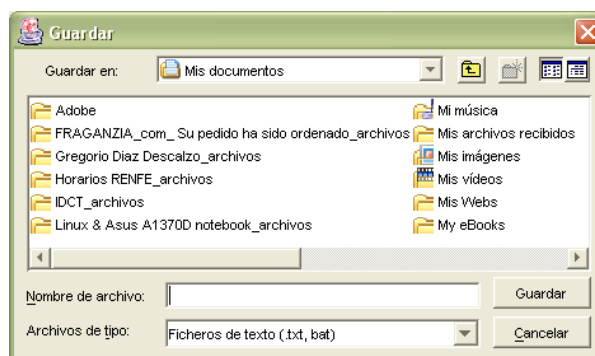
Estas clases permiten al desarrollador trabajar con diálogos que nos permiten elegir ficheros y directorios. La clase JFileChooser muestra diálogos estandarizados de distinto tipo para abrir ficheros, guardar ficheros, etc. Mientras que la clase FileFilter nos permitirá personalizar estos diálogos determinando que tipo de archivos, extensiones, directorios, etc utilizaremos.

Para mostrar los diálogos de apertura y guarda utilizaremos dos métodos de la clase JFileChooser estos son showOpenDialog y showSaveDialog respectivamente que nos devuelven el archivo que se ha seleccionado. Pero antes de llamar a estos métodos deberemos establecer el filtro que utilizaremos para seleccionar el o los archivos. Ahora veremos tres ejemplos uno de apertura de archivos, otro de guarda y uno que nos permitirá seleccionar directorios.

```
JFileChooser chooser = new JFileChooser();
ExampleFileFilter filter = new ExampleFileFilter();
File fichero = null;
filter.addExtension(".txt");
filter.addExtension(".bat");
filter.setDescription("Ficheros de texto");
chooser.setFileFilter(filter);
int returnVal = chooser.showOpenDialog(this);
if(returnVal == JFileChooser.APPROVE_OPTION)
    fichero = chooser.getSelectedFile();
```



Obtendremos el diálogo de guarda si cambiamos el método showOpenDialog por showSaveDialog.



Si queremos trabajar con directorios tendremos que cambiar el modo de selección usando la constante `JFileChooser.DIRECTORIES_ONLY`. El modo de selección también puede ser cambiado para permitir elegir mas de un fichero o directorio a la vez.

```
JFileChooser chooser = new JFileChooser();
ExampleFileFilter filter = new ExampleFileFilter();
File directorio = null;
chooser.setToolTipText("Selección del directorio");
chooser.setDialogTitle("Selecciona el directorio deseado");
chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
int returnVal = chooser.showOpenDialog(ventana);
if(returnVal == JFileChooser.APPROVE_OPTION){
    directorio = chooser.getSelectedFile();
}
```



Como último apunte en este tipo de diálogos decir que la clase `ExampleFileFilter` para ser usada debe ser añadida al proyecto o bien importada. El código fuente de esta clase lo podemos encontrar en `/demo/jfc/FileChooserDemo/ExampleFileFilter.java` a partir del directorio donde hallamos instalado el jdk de java.

### Trabajar con programas externos

Para trabajar con programas externos nos apoyaremos en el método `exec` de la clase **Runtime** el cual crea un proceso nativo devolviéndonos una instancia de la clase **Process**. Esta instancia puede ser utilizada para obtener información acerca de su ejecución. Así, La clase **Process** implementa una serie de métodos que nos permiten acceder a su información de entrada, salida y error. También nos permite obtener información acerca de su estado de ejecución, del estado en que ha finalizado, esperarnos a que finalice o incluso obligar su finalización.

Aunque hay que tener en cuenta que al usar el método `exec` de **Runtime** podemos tener problemas con algunos tipos de programas.

Para acceder a la información de salida, entrada y error deberemos usar los métodos `getOutputStream`, `getInputStream` y `getErrorStream` respectivamente. Es decir si queremos que nuestra aplicación de datos a un programa externo, los extraiga u obtenga los datos de error tendremos que utilizar estos métodos. Aunque debido a la limitación del tamaño del buffer de algunas plataformas es común que se produzcan bloqueos. Por lo tanto es aconsejable que se creen hilos para realizar esta labor, con lo que conseguiríamos que el programa principal no se bloquee y se siga ejecutando con normalidad. También deberemos tener en cuenta que los subprocessos que creamos **no** finalizan cuando ya no existan referencias al mismo, sino que se siguen ejecutando de

forma asíncrona. Por lo que seguirán ocupando recursos del sistema, por lo tanto deberemos tener especial cuidado al utilizar estos subprocesos.

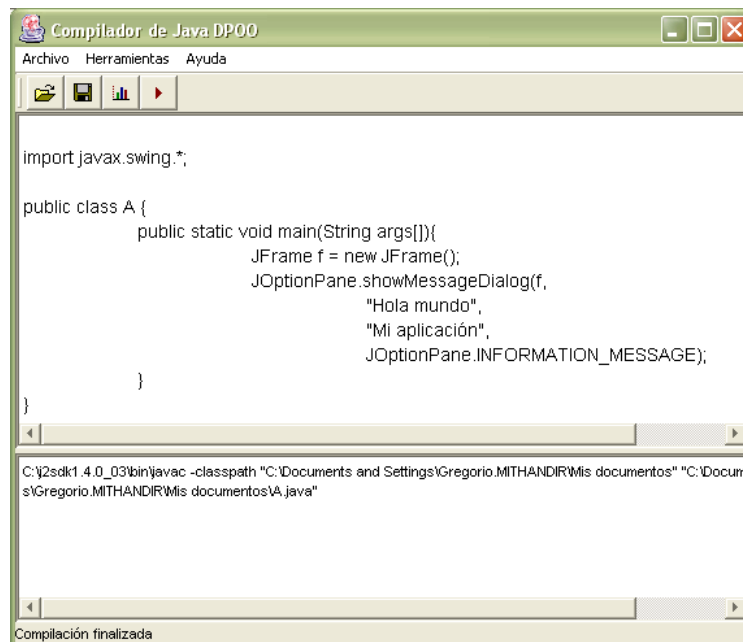
A continuación veremos un ejemplo de llamada a un programa externo.

```
try {
    String line = "";
    String cmdline = "C:\\miapp\\aplicacion.exe";
    Process p = Runtime.getRuntime().exec(cmdline);
    BufferedReader in = new BufferedReader(new InputStreamReader(p.getInputStream()));
    BufferedReader error = new BufferedReader(new InputStreamReader(p.getErrorStream()));
    for(line = error.readLine(); line!=null; line = error.readLine()){
        System.out.println(line);
    }
    error.close();
    while ((line = in.readLine()) != null) {
        System.out.println(line);
    }
    in.close();
}
catch (Exception err) {
    err.printStackTrace();
}
```

Este ejemplo llama a una aplicación situado en el directorio miapp, lee los resultados y los imprime en pantalla.

## Desarrollo

El desarrollo de la práctica consistirá en la implementación de una aplicación que nos permita editar archivos java para su posterior compilado y ejecución. La siguiente figura muestra el aspecto que debe ofrecer.



La aplicación estará formada por un menú, una barra de herramientas, una barra de estado y por un JSplitPane que a su vez contendrá dos JScrollPane y dentro de cada uno de ellos un JEditorPane.

El menú debe contener además de los ítem de archivo y ayuda, un nuevo ítem llamado Herramientas que contenga un subítem llamado Configuración que mostrará un diálogo de apertura de diálogo indicando que se seleccione dónde se encuentra el directorio dónde se ha instalado el jdk de java.

La barra de herramientas se compondrá de 4 botones botón de abrir fichero, guardar fichero, compilar fichero y ejecutar fichero.

Las clases de apoyo e imágenes las encontrareis en la página de la asignatura en la sección que pertenece a la práctica 2.

### Pasos a seguir:

Creación de componentes:

1. Crear una aplicación con todas las opciones del paso 2 del asistente.
2. Añadir un JSplitPane tal y como veíamos en la figura anterior. En la propiedad de orientación cambiarla por Vertical\_Split. Y la propiedad de OneTouchExpanded modificarla a true.
3. Añadir en la parte superior del JSplitPane un JScrollPane y otro en la parte inferior.
4. Añadir dentro de cada uno de los JScrollPane un JEditorPane.
5. Añadir un nuevo ítem al menú llamado Herramientas que contenga un Submenú llamado Configuración.
6. Crear dos nuevos botones dentro de la barra de herramientas. Cambiar la propiedad icon de cada uno de los cuatro botones usando las imágenes disponibles desde la página de la asignatura. Cambiar también la propiedad ToolTipText (muestra información del botón al pasar el ratón por encima de este) para que sea acorde a su función.

Incorporación del código

7. Incorporar al proyecto la clase Util disponible en la página de la asignatura.
8. Crear como un atributo de la ventana principal una instancia de la clase Util llamado util.
9. En el suceso actionPerformed del subítem Configuración pinchar dos veces y añadir en el código `util.fijarJdkPath(this);`
10. El botón de apertura en el suceso actionPerformed modificar el código añadiendo la sentencia `util.abrirFichero(this,this.statusBar,this.jEditorPane1);`
11. El botón de guardar en el suceso actionPerformed modificar el código añadiendo la sentencia `util.guardarFichero(this,this.statusBar,this.jEditorPane1);`
12. El botón de compilación en el suceso actionPerformed modificar el código añadiendo la sentencia `util.compilarFichero(this,this.statusBar,this.jEditorPane1);`
13. El botón de ejecución en el suceso actionPerformed modificar el código añadiendo la sentencia `util.ejecutarFichero(this,this.statusBar,this.jEditorPane1);`

Una vez incorporado el código se deben rellenar los espacios de código que se han dejado en blanco dentro de la clase Util, utilizando para ello la teoría que hemos visto a lo largo de la práctica.