ACTIVIDAD GUIADA 11
Diálogos

En muchos casos las aplicaciones necesitan, para poder continuar, aceptar nuevos datos o bien visualizarlos. Una opción para ello es utilizar una nueva ventana con los controles que permitan esas operaciones. En estos casos, las ventanas reciben el nombre de **cajas de diálogo.**

Las cajas de diálogo que podemos añadir a una aplicación se pueden clasificar en:

- ***Cajas de diálogo predefinidas***. Son cajas creadas por medio de los métodos proporcionados por la clase **`JOptionPane`**, por ejemplo **`showMessageDialog()`**.
- ***Cajas de diálogo personalizadas***. Son cajas de diálogo hechas a medida, para lo cual se proporciona la clase **`JDialog`**. En una caja de diálogo podemos utilizar cualquier de los componentes Swing vistos hasta ahora.
- ***Cajas de diálogo estándar***. Son cajas de diálogo muy comunes, por ejemplo la caja de diálogo *Abrir* o *Guardar* proporcionada por la clase **`JFileChooser`**, o el diálogo Color proporcionado por la clase **`JColorChooser`**.

A diferencia de un objeto JFrame (ventana primaria), una caja de diálogo es una ventana secundaria, es decir, una ventana que depende de otra.

Las cajas de diálogo pueden crearse de manera **modal** o **no modal:**

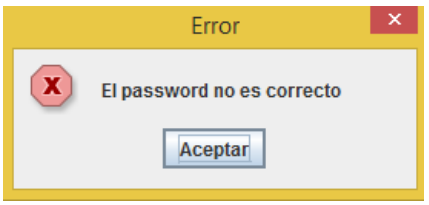
- Cuando una aplicación visualiza una caja de diálogo que tiene que ser cerrada para poder continuar (normalmente pulsando el botón *Aceptar* o *Cancelar*) estamos en el caso de una caja de diálogo **modal**.
- Si no es así, estamos en el caso de una caja de diálogo **no modal**.

La clase JOptionPane permite crear diálogos modales, mientras que la clase JDialog permite crear diálogos modales y no modales.

Cajas de diálogo predefinidas

La forma más fácil de solicitar un dato de usuario o de visualizar un resultado o un mensaje es usando las cajas de diálogo que proporciona la clase **`JOptionPane.`** Por ejemplo, la siguiente sentencia, invocada desde un frame muestra el siguiente diálogo:

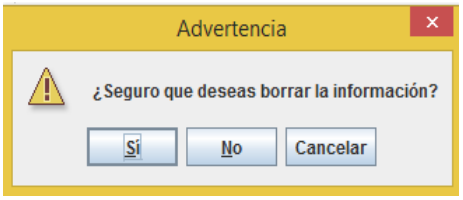
```
JOptionPane.showMessageDialog(
      this,
      "El password no es correcto",
      "Error",
      JOptionPane.ERROR_MESSAGE);
```



showMessageDialog()		
Visualiza un mensaje en una caja de diálogo		
Parámetros	`Component componentePadre`	Determina la ventana marco (un `JFrame`) sobre la que se visualiza el diálogo. Si es null o no es una ventana marco, se usa el marco por omisión (normalmente el escritorio).
	`Object mensaje`	Mensaje a visualizar
	`String titulo [opcional]`	Título del diálogo
	`int tipoMensaje [opcional]`	Tipo de mensaje que se visualiza: `ERROR_MESSAGE`, `INFORMATION_MESSAGE`, `WARNING_MESSAGE`, `QUESTION_MESSAGE` o `PLAIN_MESSAGE` (sin icono)
	`Icon icono [opcional]`	Icono que se muestra. Si no se especifica se elige en función del tipo de mensaje. Si tampoco se especifica el tipo de mensaje se muestra el icono de información.

Por ejemplo, la siguiente sentencia, invocada desde un frame, muestra el siguiente diálogo:

```
int opcion=JOptionPane.showConfirmDialog(
 this,
 "¿Seguro que deseas borrar la información?",
 "Advertencia",
 JOptionPane.YES_NO_CANCEL_OPTION,
 JOptionPane.WARNING_MESSAGE);
```

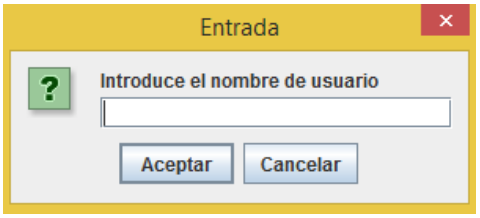


showConfirmDialog()		
Visualiza un mensaje en una caja de diálogo y proporciona varios botones destinados a obtener una confirmación del usuario. La función retorna un entero que indica qué botón se ha pulsado (`YES_OPTION`, `NO_OPTION`, `OK_OPTION`, `CANCEL_OPTION` o `CLOSED_OPTION` si se pulsó el botón de cierre de ventana).		
Parámetros	`Component componentePadre`	No varía respecto al anterior
	`Object mensaje`	No varía respecto al anterior
	`String titulo [opcional]`	No varía respecto al anterior
	`int botones [opcional]`	Botones que visualiza el diálogo: `YES_NO_OPTION`, `YES_NO_CANCEL_OPTION` [x defecto], `OK_CANCEL_OPTION`
	`int tipoMensaje [opcional]`	No varía respecto al anterior
	`Icon icono [opcional]`	No varía respecto al anterior

Por ejemplo, la siguiente sentencia, invocada desde un frame:

```
String nombre=JOptionPane.showInputDialog("Introduce el nombre de usuario");
```

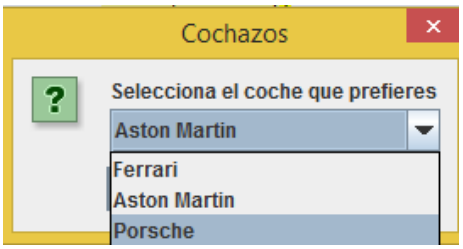
Muestra el siguiente diálogo:



Mientras que la siguiente sentencia:

```
String[] cochazos={"Ferrari","Aston Martin","Porsche"};
        JOptionPane.showInputDialog(
                this,
                "Selecciona el coche que prefieres",
                "Cochazos",
                 JOptionPane.QUESTION_MESSAGE,
                 null,
                 cochazos,
                 cochazos[1]);
```

Muestra el siguiente diálogo:

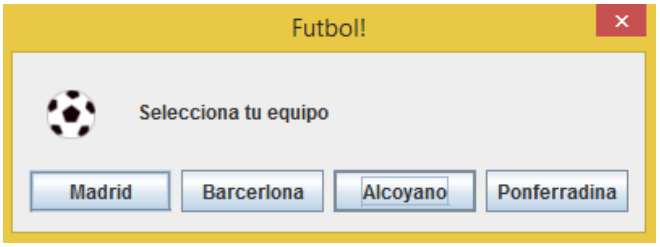


showInputDialog()		
Visualiza un mensaje en una caja de diálogo y permite al usuario introducir información de tipo textual. Devuelve el String que se ha escogido o null si se pulsó el botón de *Cancelar*		
Parámetros	`Component componentePadre [opcional]`	No varía respecto al anterior
	`Object mensaje`	No varía respecto al anterior
	`String titulo [opcional]`	No varía respecto al anterior
	`int tipoMensaje [opcional]`	No varía respecto al anterior
	`Icon icono [opcional]`	No varía respecto al anterior
	`Object[] valores [opcional]`	Lista de valores seleccionables
	`Object[] valorInicial`	Valor seleccionado por omisión.

Por último, el siguiente código:

```
String[] opciones={"Madrid","Barcerlona","Alcoyano","Ponferradina"};
        int seleccion = JOptionPane.showOptionDialog(
    this,
    "Selecciona tu equipo",
    "Futbol!",
    JOptionPane.DEFAULT_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    new ImageIcon(getClass().getResource("/resources/icono_balon.png")),
    opciones,
    "Alcoyano");
```

Muestra el siguiente diálogo:



showOptionDialog()		
Permite crear un diálogo modal personalizado con el mensaje, el título, los botones, el tipo de mensaje, los títulos de los botones y el botón seleccionado por omisión especificados. Devuelve un entero correspondiente al botón pulsado o CLOSED_OPTION si se cerró el diálogo		
Parámetros	`Component componentePadre [opcional]`	No varía respecto al anterior
	`Object mensaje`	No varía respecto al anterior
	`String titulo [opcional]`	No varía respecto al anterior
	`int botones [opcional]`	No varía respecto al anterior
	`int tipoMensaje`	No varía respecto al anterior
	`Icon icono [opcional]`	No varía respecto al anterior
	`Object[] titulosBotones [opcional]`	Lista de títulos de los botones
	`Object[] botonPorOmision`	Valor seleccionado por omisión.

Cajas de diálogo personalizadas

Una caja de diálogo personalizada es u objeto de la clase **`javax.swing.JDialog`**

Al igual que sucedía con **`JFrame`**, un objeto **`JDialog`** es un contenedor de nivel superior, luego define un panel raíz que será el componente padre de cualquier otro que añadamos al dialogo.

```
dialogo.getContentPanel().add(control);
```

Por tanto, para crear un diálogo seguiremos el mismo criterio que hemos utilizado hasta ahora para crear frames, es decir, un diálogo será un objeto de una clase derivada de **`JDialog`** y los controles serán objetos miembro de la misma. La clase **`JDialog`** tiene varios constructores, como por ejemplo:

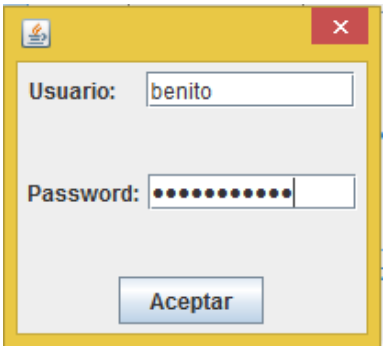
- **`JDialog()`**: Crea un diálogo no modal, sin título y sin una ventana padre específica. La ventana padre, si se especifica, tiene que ser una ventana marco (objeto derivado de **`JFrame`**) u otro diálogo (objeto derivado de **`JDialog`**).
- **`JDialog(ventanaPadre)`**: Crea un diálogo no modal, sin título y con una ventana padre específica.
- **`JDialog(ventanaPadre, boolean modal)`**: Crea un diálogo no modal, sin título y con una ventana padre específica.
- **`JDialog(ventanaPadre, String titulo)`**: Crea un diálogo no modal, con título y con una ventana padre específica.
- **`JDialog(ventanaPadre, String titulo, boolean modal)`**: Crea un diálogo modal o no modal, con título y con una ventana padre específica.

Vamos a crear un diálogo personalizado teniendo en cuenta los siguientes aspectos:

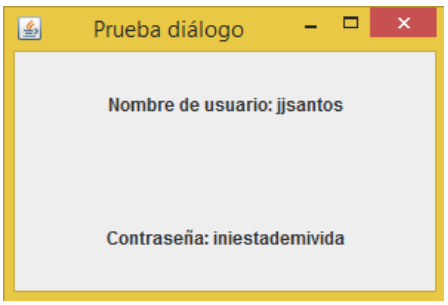
- El diálogo hereda de la clase **`JDialog`** tal y como se especificó que debe hacerse.
- El diálogo se compone de tres paneles que se ubican utilizando un **`GridLayout`**
- Para el campo de contraseña se ha utilizado un tipo especial de **`JTextField`** llamado **`JPasswordField`**. A efectos prácticos se utiliza de manera muy parecida, aunque el contenido aparece enmascarado.
    - Por razones de seguridad, el método **`getText()`** de este componente está deprecado (obsoleto, luego no se recomienda su uso) y se utiliza el método **`getPassword()`**, que devuelve un array de **`char`**.
- El diálogo también puede implementar una interfaz **`Listener`** al igual que los frames y cualquier clase que implementemos. En este caso implementamos un **`ActionListener`** que asociaremos a la pulsación del botón.
    - La acción asociada a la pulsación del botón consiste simplemente en hacer que el diálogo no sea visible. Si lo cerráramos, posteriormente no podríamos acceder a los datos ha introducido.
- Haremos los datos del diálogo sean accesibles mediante los correspondientes métodos públicos **`getUsuario()`** y **`getPassword().`**

```
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class DialogoUsuario  extends JDialog implements ActionListener{
    private JTextField txtUsuario;
    private JPasswordField txtPassword;
    private JButton btnAceptar;

   public DialogoUsuario(JFrame padre){
       super(padre,true);
       inicializaComponentes();
       pack();
       setVisible(true);
   }

      public void inicializaComponentes(){
       setLayout(new GridLayout(3,1,20,20));

       JLabel lblUsuario=new JLabel("Usuario:    ");
       JLabel lblPassword=new JLabel("Password:");

       txtUsuario=new JTextField(10);
       txtPassword=new JPasswordField(10);

       btnAceptar=new JButton("Aceptar");
       btnAceptar.addActionListener(this);

       JPanel pUsuario=new JPanel();
       pUsuario.add(lblUsuario);
       pUsuario.add(txtUsuario);
       getContentPane().add(pUsuario);

       JPanel pPassword=new JPanel();
       pPassword.add(lblPassword);
       pPassword.add(txtPassword);
       getContentPane().add(pPassword);

       JPanel pAceptar=new JPanel();
       pAceptar.add(btnAceptar);
       getContentPane().add(pAceptar);
   }
    @Override
    public void actionPerformed(ActionEvent ae) {
        setVisible(false);
    }

    public String getUsuario(){
        return txtUsuario.getText();
    }

    public String getPassword(){
        return String.valueOf(txtPassword.getPassword());
    }
```

El diálogo tiene el siguiente aspecto:



Para utilizar el diálogo y a modo de ejemplo, se ha desarrollado un frame que lanza el diálogo y posteriormente hace uso del mismo para pedir nombre y contraseña a un usuario, valores que posteriormente se muestran en dos etiquetas del propio marco. Se accede a los datos del diálogo a través de los métodos públicos habilitados para este fin. Al arrancar el programa, se muestra en primer lugar el diálogo. Al ser el diálogo modal, el frame no se muestra hasta que no se ha cerrado el diálogo. Este es el aspecto del marco una vez que se han leído los datos:



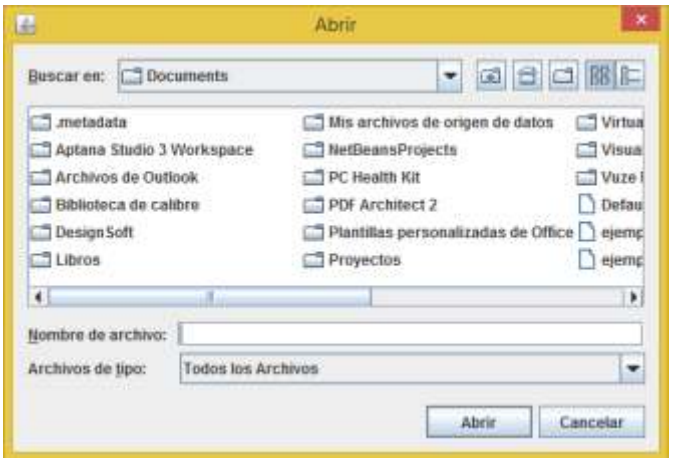

```
public class PruebaDialogoFrame extends JFrame {

    public PruebaDialogoFrame(){
       super("Prueba diálogo");

       DialogoUsuario dialogoUsuario=new DialogoUsuario(this);

       setLayout(new GridLayout(2,1,20,20));
       JLabel lblUsuario=new JLabel("Nombre de usuario: ", JLabel.CENTER);
       add(lblUsuario);
       JLabel lblPassword=new JLabel("Contraseña: ", JLabel.CENTER);
       add(lblPassword);

       lblUsuario.setText(lblUsuario.getText()+dialogoUsuario.getUsuario());
       lblPassword.setText(lblPassword.getText()+dialogoUsuario.getPassword());

       setSize(300,200);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setVisible(true);
    }

    public static void main(String[] args){
        PruebaDialogoFrame prueba=new PruebaDialogoFrame();
    }
}
```

Cajas de diálogo estándar

Como ejemplo de este tipo veremos las dos cajas de diálogo más comúnmente empleadas en el diseño de aplicaciones: La caja de diálogo *Abrir o Guardar* (que se muestran mediante un componente denominado **`JFileChooser`**) y la caja de diálogo *Color* (que se muestra con un componente llamado **`JColorChooser`**)

Cajas de diálogo Abrir y Guardar

La caja de diálogo *Abrir* permite al usuario seleccionar una unidad de disco, un directorio, una extensión de fichero y un nombre de fichero. Una vez realizada la selección, la propiedad **`selectedFile`** (accesible mediante el método **`getSelectedFile()`** que devuelve un objeto **`File`**) de la caja de diálogo contiene el nombre del fichero elegido.



Para visualizar la caja de diálogo *Abrir*:

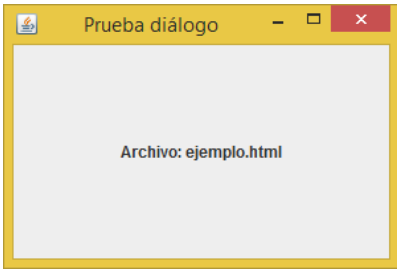
1. Creamos un objeto de la clase **`JFileChooser`**
2. Modificamos los valores por omisión de sus propiedades si es necesario.
3. Mostramos el diálogo invocando a **`showOpenDialog()`** o **`showSaveDialog()`**

El valor de tipo **`int`** devuelto por **`showXXXDialog()`** indica:

- Si el usuario aceptó la operación (**`JFileChooser.APPROVE_OPTION`**)
- Si el usuario canceló la operación (**`JFileChooser.CANCEL_OPTION`**)
- Si se ha producido algún error (**`JFileChooser.ERROR_OPTION`**)

De acuerdo con lo expuesto, el siguiente código permite seleccionar un archivo. Se ha ubicado el diálogo dentro de un frame donde, una vez seleccionado el archivo, se mostrará el nombre del mismo en una etiqueta:

Este es el aspecto del frame una vez que hemos escogido el archivo:



```
public class PruebaDialogoAbrir extends JFrame {

    public PruebaDialogoAbrir(){
        super("Prueba diálogo");

      JLabel lblArchivo=new JLabel("Archivo: ", JLabel.CENTER);

       JFileChooser dlgAbrir=new JFileChooser();
       int opcion=dlgAbrir.showOpenDialog(this);
       if(opcion==JFileChooser.APPROVE_OPTION){
           String nombreArchivo=dlgAbrir.getSelectedFile().getName();
           lblArchivo.setText(lblArchivo.getText()+nombreArchivo);

       }
       add(lblArchivo);

      setSize(300,200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
    }

    public static void main(String[] args){
        PruebaDialogoAbrir prueba=new PruebaDialogoAbrir();
    }
}
```

La caja de diálogo *Guardar* es idéntica a la caja de diálogo *Abrir*, tan sólo cambia el título de la caja de diálogo.

Filtros

Hemos visto que, por omisión, la caja de diálogo *Abrir* o *Guardar* muestra todos los ficheros y directorios, salvo los ocultos. Para mostrar únicamente algunos tipos de ficheros, tenemos que añadir al diálogo los filtros correspondientes.

Un filtro es un objeto derivado de la clase **`FileFilter`** que implementa dos métodos abstractos:

- El método **`accept()`** devuelve true si el fichero que le pasamos como argumento (tipo **`File`**) satisface el filtro.
- El método **`getDescription()`** devuelve la cadena de caracteres que describe el filtro.

```
public class FiltroJPG extends FileFilter{

    @Override
    public boolean accept(File file) {
        String nombreFichero=file.getName().toLowerCase();
        if(nombreFichero!=null)
            if(nombreFichero.endsWith(".jpeg") ||
              (nombreFichero.endsWith(".jpg")))
                return true;
        return false;
    }

    @Override
    public String getDescription() {
        return "Ficheros en formato JPG";
    }
}
```

Por ejemplo, si sólo queremos ver los ficheros **.jpg** podemos hacer este filtro:

```
dlgAbrir.setFileFilter(new FiltroJPG());
```

A continuación pasaríamos el filtro al **JFileChooser** usando su método **setFileFilter().** Veremos que el diálogo asociado sólo permite escoger archivos según el filtro especificado.

A partir del JDK 1.6 se incluye una clase filtro **FileNameExtensionFilter** que permite especificar únicamente la extensión y a la cual podemos invocar de manera muy sencilla. El constructor de la clase recibe la descripción y los tipos de archivo aceptados:

```
FileNameExtensionFilter filtro= new FileNameExtensionFilter("JPG &
GIF", "jpg", "gif");
```

Otros aspectos

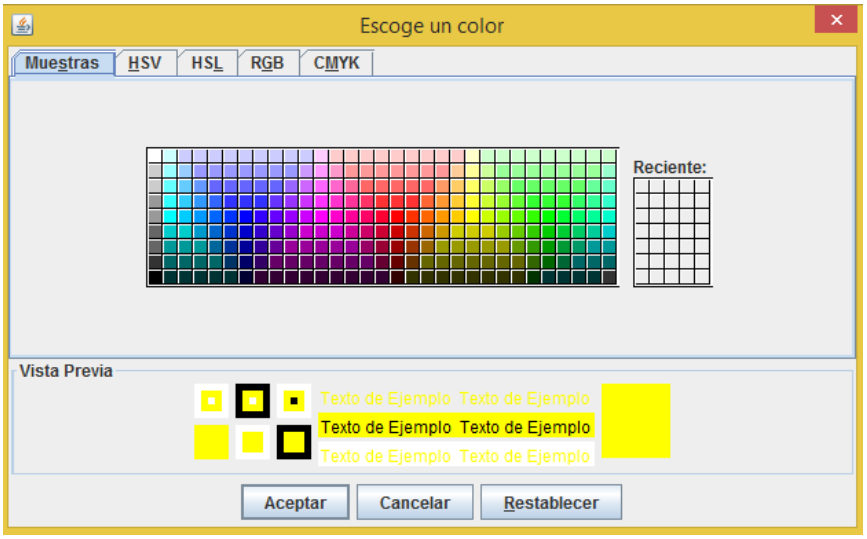
El **JFileChooser** se abre por defecto en el directorio de trabajo del usuario (**C:\Users\usuario\Documents** en Windows 7, **/home/usuario en Linux**). Podemos elegir el directorio en el que queremos que se abra llamando al método **setCurrentDirectory()** y pasándole la ruta en cuestión.

Por defecto, un **JFileChooser** sólo deja escoger ficheros. Si queremos que también deje elegir directorios, debemos llamar a **setFileSelectionMode()**, pasándole como parámetro una de las siguientes constantes:

- **JFileChooser.FILES_ONLY**
- **JFileChooser.DIRECTORIES_ONLY**
- **JFileChooser.FILES_AND_DIRECTORIES**

Cajas de diálogo Color

La caja de diálogo *Color* permite al usuario seleccionar un color de una paleta (panel *Muestras*) o crear y seleccionar un panel personalizado (paneles HSV: *Hue-Saturation-Value*, HSL: *Hue-Saturation-Light*, RGB: *Red-Green-Blue* y CMYK: *Cyan-Magenta-Yellow-Key*).



Para visualizar la caja de diálogo invocar al método estático **`showDialog()`** de la clase **`JColorChooser`**:

- **`Color showDialog(Component ventanaPadre, String titulo, Color colorInicial)`**

Una vez realizada la selección, el método **`showDialog()`** será el que devuelva el color seleccionado.

De acuerdo con lo expuesto, el marco que se muestra a continuación imprime el color que hemos seleccionado en un diálogo. Nótese que al convertir un color a cadena obtenemos los niveles RGB del mismo:

```
public class PruebaDialogoColor extends JFrame {

    public PruebaDialogoColor(){
        super("Prueba diálogo");

      JLabel lblColor=new JLabel("Color: ", JLabel.CENTER);

       Color color=JColorChooser.showDialog(this,"Escoge un color",Color.yellow);
       if(color!=null)
           lblColor.setText(lblColor.getText()+color.toString());

       add(lblColor);

      setSize(300,200);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setVisible(true);
    }

    public static void main(String[] args){
        PruebaDialogoColor prueba=new PruebaDialogoColor();
    }
}
```