PROGRAMACIÓN EN JAVA

TEMA: Cajas de diálogo

Bibliografía

- "JAVA 2 Interfaces gráficas y aplicaciones para Internet – 2ª Edición"
  Fco. Javier Ceballos. Ed. Ra-Ma, 2006
  Capítulo: 4

Contenido

- Cajas de diálogo
- Cajas de diálogo modales y no modales
- Cajas de diálogo predefinidas
- Visualizar datos (showMessageDialog)
- Confirmar datos (showConfirmDialog)
- Requerir datos (showInputDialog)
- Diálogo modal personalizado (showOptionDialog)
- Cajas de diálogo personalizadas
- Cajas de diálogo estándar
- Cajas de diálogo Abrir y Guardar
- Propiedades en las cajas de diálogo estándar
- Estableciendo nuestros filtros
- Caja de diálogo Color

Cajas de diálogo

- Caja de diálogo: Es una nueva ventana con controles que se utiliza, en un instante determinado, para aceptar nuevos datos o bien para visualizarlos
- La funcionalidad para manipular cajas de diálogo es proporcionada por las clases **JOptionPane** y **JDialog**
- Hay 3 formas de añadir cajas de diálogo a una aplicación:
  - *Cajas de diálogo predefinidas*: Cajas de diálogo creadas por medio de los métodos proporcionados por la clase **JOptionPane**; por ejemplo **showMessageDialog**
  - *Cajas de diálogo personalizadas*: Cajas de diálogo hechas a medida utilizando la clase **JDialog**
  - *Cajas de diálogo estándar*: Cajas de diálogo muy comunes. Por ejemplo la caja de diálogo *Abrir* o *Guardar* proporcionadas por la clase **JFileChooser**, o el diálogo *Color* proporcionado por la clase **JColorChooser**
- A diferencia de un objeto **JFrame** una caja de diálogo es una ventana secundaria que depende de otra

Cajas de diálogo modales y no modales

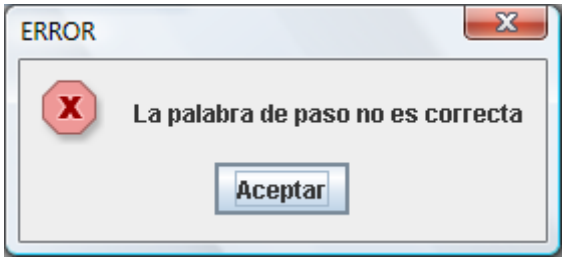
- Cuando una aplicación visualiza una caja de diálogo que tiene que ser cerrada para poder continuar con la aplicación estaremos ante la presencia de una caja de *diálogo modal*
- Si no es así, estaremos ante la presencia de una caja de *diálogo no modal*
- La clase **JOptionPane** crea únicamente diálogos modales
- La clase **JDialog** permite crear diálogos modales y no modales

Cajas de diálogo predefinidas

- La forma más fácil de solicitar un dato o de visualizar un resultado es utilizando las cajas de diálogo predefinidas
- Para ello, la biblioteca JFC (específicamente la clase **JOptionPane**) nos proporciona los siguientes métodos:
  - showMessageDialog
  - showConfirmDialog
  - showInputDialog
  - showOptionDialog

Visualizar datos (showMessageDialog)

- El método **showMessageDialog** visualiza un mensaje en una caja de diálogo semejante al de la siguiente figura



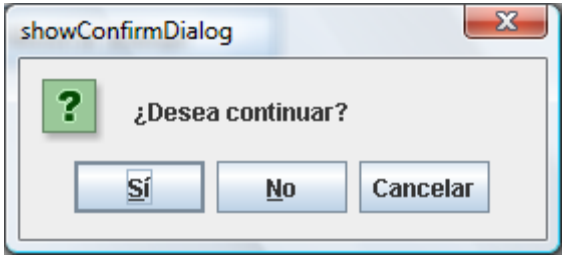
- La sintaxis del método **showMessageDialog** es:

public static void showMessageDialog(Component *componentePadre*, Object mensaje,
String *título*, int *tipoMensaje*, Icon *icono*)
throws HeadlessException

- *componentePadre*: Determina la ventana marco sobre la que se visualiza el diálogo; si es *null*, o si el *componentePadre* no es una ventana marco, se utiliza la ventana marco por omisión (normalmente el escritorio)
- *mensaje*: Mensaje a ser visualizado
- *título*: Título del diálogo
- *tipoMensaje*: Tipo de mensaje que se visualiza: ERROR_MENSSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MENSSAGE o PLAIN_MESSAGE (sin icono)
- *icono*: Icono que se muestra en el diálogo. Cuando no se especifica, se elige uno en función del tipo de mensaje. Si no se especifica el tipo de mensaje, el icono que se muestra es el de información
- Los dos primeros parámetros son obligatorios; el resto son opcionales

Confirmar datos (showConfirmDialog)

- El método **showConfirmDialog** visualiza un mensaje en una caja de diálogo semejante al de la siguiente figura



- La sintaxis del método **showConfirmDialog** es:

public static int showConfirmDialog(Component *componentePadre*, Object *mensaje*,
String *título*, int *botones*, int *tipoMensaje*, Icon *icono*)
throws HeadLessException

- *componentePadre*: Determina la ventana marco sobre la que se visualiza el diálogo; si es *null*, o si el *componentePadre* no es una ventana marco, se utiliza la ventana marco por omisión (normalmente el escritorio)
- *mensaje*: Mensaje a ser visualizado
- *título*: Título del diálogo
- *botones*: Botones que visualizará el diálogo: YES_NO_OPTION (botones Sí y No), o YES_NO_CANCEL_OPTION (botones Sí, No y Cancelar, esta es la opción por omisión)
- *tipoMensaje*: Tipo de mensaje que se visualiza: ERROR_MENSSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MENSSAGE o PLAIN_MESSAGE (sin icono)
- *icono*: Icono que se muestra en el diálogo. Cuando no se especifica, se elige uno en función del tipo de mensaje. Si no se especifica el tipo de mensaje, el icono que se muestra es el de interrogación
- Los dos primeros parámetros son obligatorios; el resto son opcionales

Confirmar datos (showConfirmDialog) (cont.)

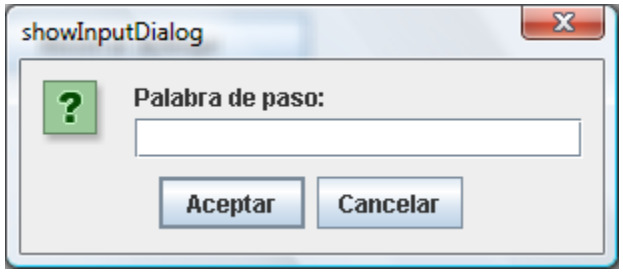
- ¿Cómo hacemos para capturar el botón pulsado en el diálogo **showConfirmDialog**?
- Ejemplo:

```
//...
public void btnDialogoActionPerformed(ActionEvent e)
{
    int boton = JOptionPane.showConfirmDialog(this,"¿Desea continuar?“,"showConfirmDialog“,
                        JOptionPane.YES_NO_CANCEL_OPTION,JOptionPane.QUESTION_MESSAGE);
    if(boton == JOptionPane.YES_OPTION)
        JOptionPane.showMessageDialog(this,"Se pulso Sí","showMessageDialog",
                                        JOptionPane.INFORMATION_MESSAGE);
    if(boton == JOptionPane.NO_OPTION)
        JOptionPane.showMessageDialog(this,"Se pulso No","showMessageDialog",
                                        JOptionPane.INFORMATION_MESSAGE);
    if(boton == JOptionPane.CANCEL_OPTION)
        JOptionPane.showMessageDialog(this,"Se pulso Cancelar","showMessageDialog",
                                        JOptionPane.INFORMATION_MESSAGE);
    if(boton == JOptionPane.CLOSED_OPTION)
      JOptionPane.showMessageDialog(this,"Se cerro el diálogo","showMessageDialog",
                                        JOptionPane.INFORMATION_MESSAGE);
}
//...
```

- El valor retornado por el método **showConfirmDialog** es un entero que indica qué botón se ha pulsado (YES_OPTION = Sí, NO_OPTION = No, CANCEL_OPTION = Cancelar o CLOSED_OPTION si se pulsó el botón cerrar)

Requerir datos (showInputDialog)

- El método **showInputDialog** visualiza una caja de diálogo semejante al de la siguiente figura



- La sintaxis del método **showInputDialog** es:

public static String showInputDialog(Object mensaje)
throws HeadlessException

- El método **showInputDialog** devuelve el objeto *String* tecleado si se pulsó Aceptar o *null* si se pulsó *Cancelar*

Requerir datos (showInputDialog) (cont.)

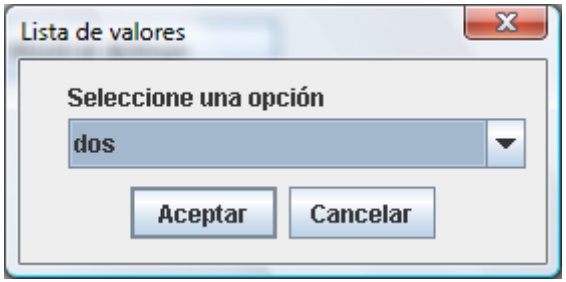
- Otro posible formato para el método showInputDialog es el siguiente:

public static Object showInputDialog(Component *componentePadre*, Object mensaje,
String *título*, int *tipoMensaje*, Icon *icono,*
Object[] *valores*, Object *valorInicial*)
throws HeadlessException

- Los dos primeros parámetros son obligatorios; el resto son opcionales
- El significado de los cinco primeros parámetros es igual que el visto en el método **showMessageDialog**, y el de los dos últimos es el siguiente:
- *valores*: Lista de valores seleccionables
- *valorInicial*: Valor seleccionado por omisión
- Ejemplo:

```
//…
Object[] valoresSeleccionables = {“uno”, “dos”, “tres”};
String s = (String)JOptionPane.showInputDialog(this,”Seleccione una opción”,
                                               ”Lista de valores”, JOptionPane.PLAIN_MESSAGE,
                                               new ImageIcon(“imágenes\\logo.gif”),
                                               valoresSeleccionables, valoresSeleccionables[1]);
//…
```

- Este código visualizará la siguiente caja de diálogo:



Diálogo modal personalizado (showOptionDialog)

- El método **showOptionDialog** visualiza una caja de diálogo modal con el mensaje, el título, los botones, el tipo de mensaje, el icono, los títulos de los botones y el botón seleccionado por omisión especificados
- Su sintaxis es:

```
public static int showOptionDialog(Component componentePadre, Object mensaje,
                                   String título, int botones, int tipoMensaje, Icon icono,
                                   Object [] títulosBotones, Object botónPorOmisión)
                            throws HeadlessException
```

- Este método devuelve un entero correspondiente al botón pulsado o el valor CLOSED_OPTION si se cerró el diálogo

Cajas de diálogo personalizadas

- Una caja de diálogo personalizada es un objeto de la clase **JDialog**
- Un objeto **JDialog** es un contenedor de nivel superior, por lo tanto, define un panel raíz que será el componente padre de cualquier otro que añadamos al diálogo
- Para añadir un control al diálogo utilizaremos la siguiente sintaxis:

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

- Para crear una caja de diálogo personalizada debemos crearnos una clase derivada de **JDialog** y los controles serán objetos miembros de esta clase, luego nos debemos crear un objeto de nuestra clase derivada de **JDialog** recientemente creada
- La clase **JDialog** tiene varios constructores:
  - **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 (**JFrame**) u otro diálogo (**JDialog**)
  - **JDialog(***Ventana padre***)**: Crea un diálogo no modal, sin título y con una ventana padre específica
  - **JDialog(***Ventana padre,* **boolean** *modal***)**: Crea un diálogo modal o no modal, sin título y con una ventana padre específica
  - **JDialog(***Ventana padre,* **String** *título***)**: Crea un diálogo no modal, con título y con una ventana padre específica
  - **JDialog(***Ventana padre,* **String** *título,* **boolean** *modal***)**: Crea un diálogo modal o no modal, con título y con una ventana padre específica
  - **JDialog(***Ventana padre,* **String** *título,* **boolean** *modal,* **GraphicsConfiguration** *gc***)**: Crea un diálogo modal o no modal, con título, con una ventana padre específica y con la configuración gráfica especificada

Cajas de diálogo personalizadas (cont.)

- Ejemplo:

```
//…
public class Ejemplo extends JDialog  //Clase derivada de JDialog
{
    public Ejemplo(Frame parent, boolean modal)
    {
        /*Invocación al constructor de la clase base pasando como argumentos una referencia a la
          ventana padre y un valor true si se desea un diálogo modal, o false en caso contrario*/
        super(parent, modal);
        initComponents();
        setSize(300,200);
    }
    private void initComponents()  //Aquí se deberán añadir los controles al diálogo
    {
        getContentPane().setLayout(null);
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                setVisible(false);  //Ocultamos el diálogo
                dispose();  //Eliminamos el diálogo
            }
        });
    }
}
```

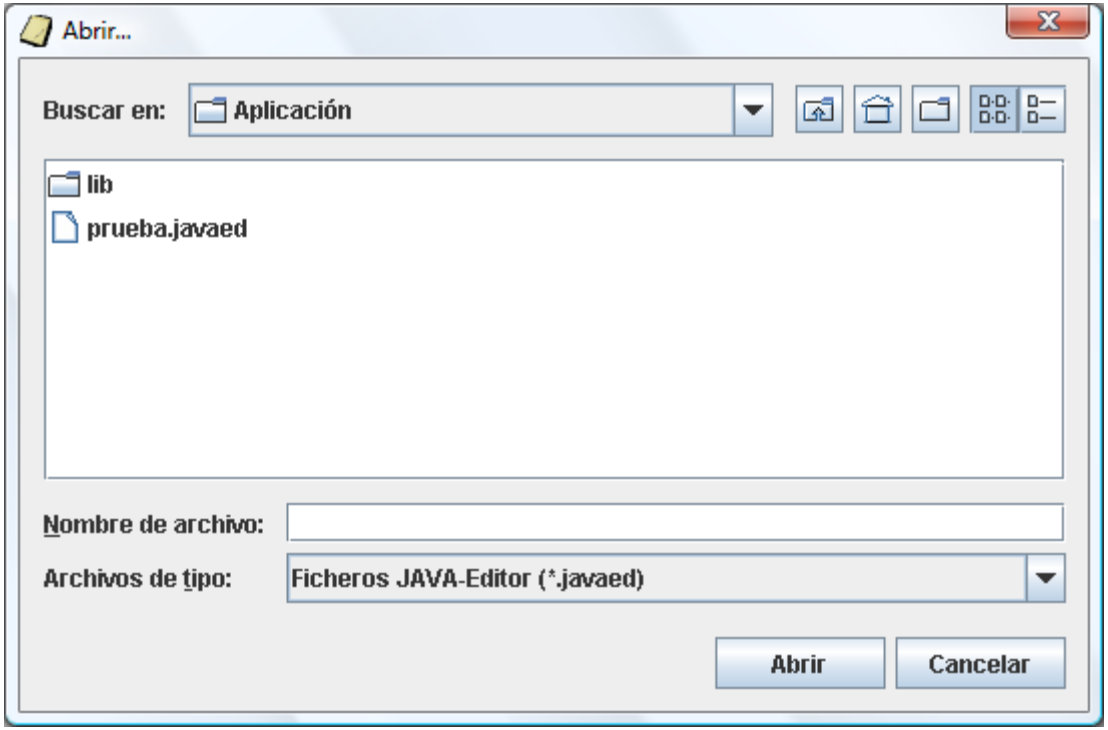
- Ahora, para mostrar el diálogo sólo nos faltaría crear un objeto de la clase *Ejemplo* y luego invocar a su método **setVisible**. Todo ello desde nuestro formulario principal

Cajas de diálogo estándar

- La biblioteca JFC incluye 2 clases, **JFileChooser** y **JColorChooser**, derivadas de **JComponent**
- Estas permiten visualizar las cajas de diálogo más comúnmente empleadas en el diseño de aplicaciones: Caja de diálogo *Abrir* (Open) o *Guardar* (Save), y la caja de diálogo *Color* (Color)

Cajas de diálogo Abrir y Guardar

- La caja de diálogo *Abrir* permite al usuario seleccionar una unidad de disco, un directorio, una extensión de archivo y un nombre de archivo
- Una vez realizada la selección, la propiedad *selectedFile* de la caja de diálogo contiene el nombre del archivo elegido
- El método **getSelectedFile** devuelve un objeto **File** con el que podremos acceder a dicho archivo



Cajas de diálogo Abrir y Guardar (cont.)

- Visualizar la caja de diálogo *Abrir*

```
//…
private void visualizarDialogoAbrir()
{
    //Creamos un objeto de la clase JFileChooser
    JFileChooser dlgAbrir = new JFileChooser();
    //Mostramos el diálogo invocando a showOpenDialog
    int opcion = dlgAbrir.showOpenDialog(this);
    if(opcion == JFileChooser.APPROVE_OPTION)
        JOptionPane(this,"El nombre del archivo seleccionado es: "+dlgAbrir.getSelectedFile().getName());
    else
        JOptionPane(this,"Se pulso el botón de Cancelar");
}
//…
```

- El valor de tipo **int** devuelto por **showOpenDialog** indica si el usuario aceptó (APPROVE_OPTION) la operación o la canceló (CANCEL_OPTION)
- El método **getName** de la clase **File** devuelve el nombre del fichero seleccionado

Cajas de diálogo Abrir y Guardar (cont.)

- Visualizar la caja de diálogo *Guardar*

```
//...
private void visualizarDialogoGuardar()
{
    //Creamos un objeto de la clase JFileChooser
    JFileChooser dlgGuardar = new JFileChooser();
    //Mostramos el diálogo invocando a showSaveDialog
    int opcion = dlgGuardar.showSaveDialog(this);
    if(opcion == JFileChooser.APPROVE_OPTION)
        JOptionPane(this,"Has guardado el archivo: "+dlgGuardar.getSelectedFile().getName());
    else
        JOptionPane(this,"Se pulso el botón de Cancelar");
}
//...
```

- El valor de tipo **int** devuelto por **showOpenDialog** indica si el usuario aceptó (APPROVE_OPTION) la operación o la canceló (CANCEL_OPTION)
- El método **getName** de la clase **File** devuelve el nombre del fichero seleccionado

Propiedades en las cajas de diálogo estándar

- La propiedad *acceptAllFileFilter* determina si el filtro “Todos los archivos” se incluye en la lista “Archivos de tipo:” (Por omisión, está incluido)
- La propiedad *choosableFileFilter* nos permite poner filtros más restrictivos
- Otra cosa que se puede determinar es si el diálogo durante la selección busca ficheros, directorios, o ambos, accediendo a las propiedades *directorySelectionEnabled* (*false* por omisión) y *fileSelectionEnabled* (true por omisión). Estas propiedades están basadas en la propiedad *fileSelectionMode* que por que por omisión vale FILES_ONLY
- El número de ficheros que pueden ser seleccionados está determinado por la propiedad *multiSelectionEnabled* que por omisión vale false
- La propiedad *selectedFile* contiene la selección simple, mientras que *selectedFiles* contendría la múltiple

Estableciendo nuestros filtros

- Por omisión las cajas de diálogo *Abrir* o *Guardar* muestran todos los ficheros y directorios, excepto los ocultos
- Para mostrar solamente algunos tipos de ficheros, tenemos que añadir al diálogo los filtros correspondientes
- Un filtro es un objeto de una clase derivada de la clase **FileFilter** del paquete **javax.swing.filechooser**
- Esta es una clase abstracta con 2 métodos abstractos: *accpet* y *getDescription*
- *accept*: Devuelve *true* si el archivo pasado como argumento satisface el filtro
- *getDescription*: Devuelve una cadena de caracteres que describe el filtro

Estableciendo nuestros filtros (cont.)

- A continuación se muestra un ejemplo de cómo hacer un filtro para filtrar, además de todos los directorios, los archivos con extensión *.doc*

```
//…
public class FiltroDoc extends FileFilter
{
    //Aceptar todos los directorios y todos los ficheros .doc
    public boolean accept(java.io.File f)
    {
        if(f.isDirectory())
            return true;
        String nombre = f.getName().toLowerCase();
        if(nombre != null)
            if(nombre.endsWith(“.doc”))
                return true;
        return false;
    }
    //Descripción del filtro
    public String getDescription()
    {
        return “Ficheros de Word (*.doc)”;
    }
}
```

Estableciendo nuestros filtros (cont.)

- Ahora sólo nos falta establecer el filtro anterior a nuestra caja de diálogo
- Para ello haremos uso del método **addChoosableFileFilter**
- Ejemplo:

```
//…
private void visualizarDialogoAbrir()
{
      JFileChooser dlgAbrir = new JFileChooser();
      dlgAbrir.addChoosableFileFilter(new FiltroDoc());
      dlgAbrir.setFileFilter(dlgAbrir.getChoosableFileFilters()[1]);
      int opcion = dlgGuardar.showOpenDialog(this);
      //…
}
//…
```

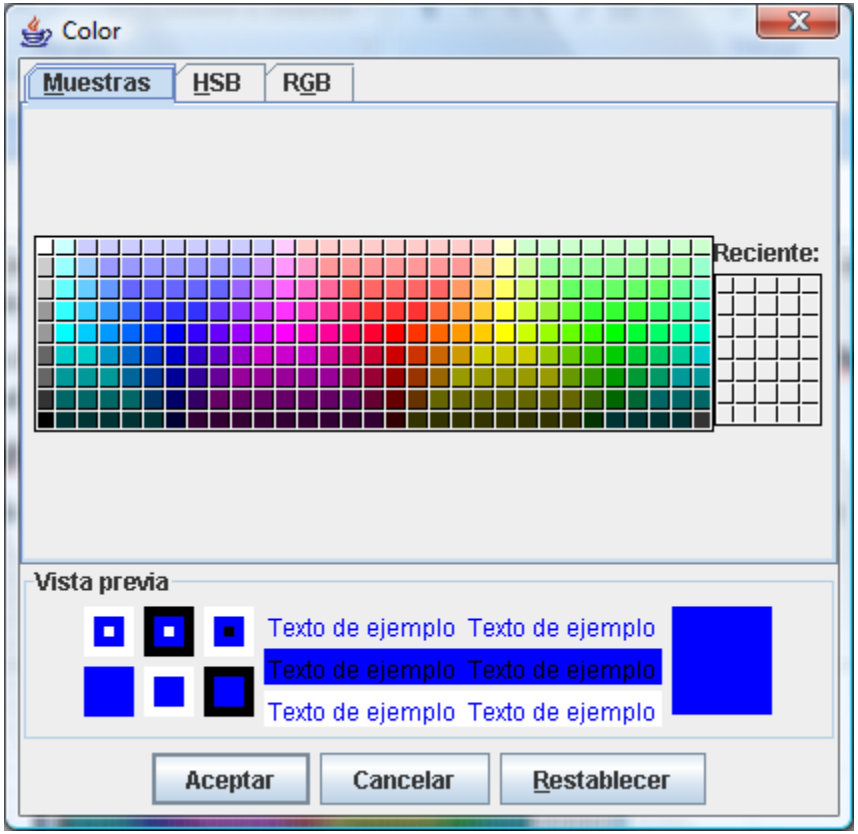
- El método **getChoosableFileFilters** devuelve una matriz de objetos **FileFilter** que contiene los filtros asignados, además del filtro por omisión
- El método **setFileFilter** fija el filtro que será visualizado cuando se muestre el diálogo

Caja de diálogo Color

- La caja de diálogo *Color* permite al usuario seleccionar un color de una paleta o crear y seleccionar un color personalizado
- Para visualizar la caja de diálogo *Color* basta con invocar al método estático **showDialog** que tiene la siguiente sintaxis:

Color showDialog(Component ventanaPadre, StringTítulo, Color colorInicial)

- Una vez realizada la selección, el método **showDialog** será el que devuelva el color seleccionado



Caja de diálogo Color (cont.)

- Ejemplo:

```
//…
private void mostrarDialogoColor()
{
      Color color;
      color = JColorChooser.showDialog(this,”Color”,Color.BLUE);
      if(color != null)
            JOptionPane.showMessageDialog(this,”El color seleccionado fue: ”+color.toString());
      else
            JOptionPane.showMessageDialog(this,”Se pulso el botón de Cancelar”);
}
//…
```