PRACTICA INTERNAL FRAME – CUADROS DE DIÁLOGOS -

Duración estimada: 200 min.

Objetivos:

- Trabajar con ventanas internas en aplicaciones
- Trabajar con cuadros de diálogo

Recursos necesarios:

- Apuntes dados en clase.
- Guión de la práctica
- Internet.

Introducción:

1. **Trabajar con el Color:**

   Para trabajar con la clase **Color** debemos consultar https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/awt/Color.html

2. **Trabajar con Font:**

   Para trabajar con la clase **Font** debemos consultar https://docs.oracle.com/javase/8/docs/api/java/awt/Font.html

3. **Ventana de diálogo:**

   Una ventana de diálogo es una ventana que aparece de forma independiente a la ventana gráfica. Los principales objetivos de las ventanas de diálogo son mostrar avisos y solicitar datos a los usuarios.

   El formato general de una ventana de diálogo es

   JOptionPane.show<tipo>Dialog();

   donde:

   El número de parámetros es diferente dependiendo del tipo de ventana a mostrar, y <tipo> puede ser:

   - **Message para mostrar un mensaje de aviso,**
   - **Input para capturar un dato de entrada, o**
   - **Confirm para confirmar una pregunta con un sí o un no.**

4. **Ventanas de aviso:**

   Una ventana de aviso se construye con la siguiente instrucción:

   JOptionPane.showMessageDialog(null, "Mensaje de aviso");

   y con la que se obtiene el siguiente resultado: **mensajeAviso**

Hay una palabra nueva en el primer argumento: null. En general, cuando creamos una ventana es porque nuestra aplicación trabaja con más ventanas. Al mostrar la ventana podemos forzar a que dependa de otras ventanas o por el contrario, con null, indicamos que no depende de ninguna y además se mostrará en el centro.

5. Ventanas para petición de datos:

Existen varios tipos de ventanas para la petición de datos. La más sencilla que puede construirse viene dada por el siguiente código

```
String s = JOptionPane.showInputDialog("Mensaje de introducción de datos : ");
```

almacena en el String s el valor introducido en el campo de texto. Si cierras la ventana sin introducir ningún texto se guardará el valor null en la variable s.

En el caso de que quieras que el usuario seleccione una de entre dos posibles opciones entonces es recomendable usar esta otra línea de código:

```
int n = JOptionPane.showConfirmDialog(null, "Mensaje", "Titulo del Diálogo", JOptionPane.YES_NO_OPTION);
```

que genera una ventana con dos botones:

En este caso retornará el valor entero 1 si seleccionas "No" y el valor 0 si seleccionas "Yes". Si cierras la ventana sin seleccionar ninguna opción se guardará el valor -1 en la variable n.

6. Frames internos:

El JinternalFrame es una ventana especial que ni es ventana ni es nada. De hecho, no hereda de Window. En realidad es un componente java que se dibuja dentro de otro componente, pero adornado con la barra de título de una ventana y sus botones de maximizar, minimizar y cerrar. Puede incluso arrastrarse y modificarse de tamaño siempre que se meta dentro del componente adecuado.

Un ***JInternalFrame*** es una ventana que va metida dentro de un panel y no puede salirse de él. En *java* el panel adecuado y especializado en el manejo de ***JInternalFrame*** es el *JDesktopPane*..

Con este código el JinternalFrame se comportará como una ventana, pero que no puede salirse del JDesktop que la contiene.

Con la clase JinternalFrame se puede mostrar un Jframe como una ventana dentro de otra ventana. Para crear un frame interno que parezca un diálogo sencillo, se pueden utilizar los cuadros de diálogo..

Los frames internos utilizan la decoración de ventana del aspecto y comportamiento Metal. Sin embargo, la ventana que los contiene tiene decoración de aspecto y comportamiento nativo (en este caso, Motif).

El código para utilizar frames internos es similar en muchas formas al código para utilizar frames normales Swing. Los frames internos no son ventanas, por lo que de alguna forma son diferentes de los frames. Por ejemplo, debemos añadir un frame interno a un contenedor (normalmente un JDesktopPane). Un frame interno no genera eventos window; en su lugar, las acciones del usuario que podrían causar que un frame dispara eventos windows hacen que en un frame interno se disparen eventos "internal frame".

Como los frames internos se han implementado con código independiente de la plataforma, ofrecen algunas características que los frames no pueden ofrecer:

- Una de esas características es que los frames internos ofrecen más control sobre su estado y capacidades. Se puede minimizar o maximizar un frame interno, a través del código.
- También se puede especificar el icono que va en la barra de título del frame interno.
- Incluso podemos especificar si el frame tiene soporte de decoración de ventanas, redimensionado, minimización, cerrado y maximización.
- Otra característica es que los frames internos se han diseñado para trabajar con paneles por capas. El API JinternalFrame contiene métodos como ***moveToFront*** que funcionan sólo si el contenedor de un frame interno es un layeredpane.

- **Reglas de utilización de Frames Internos**
  - Se debe seleccionar el tamaño del frame interno. Si no se selecciona el tamaño del frame interno, tendrá tamaño cero y nunca será visible. Se puede seleccionar el tamaño utilizando uno de estos métodos:***setSize,pack o setBounds***.
  - Como regla, se debe seleccionar la posición del frame interno. Si no se selecciona la localización, empezará en 0,0 (la esquina superior izquierda de su contenedor). Se pueden utilizar los métodos ***setLocation o setBounds*** para especificar la esquina superior izquierda del frame interno en relación a su contenedor.
  - Para añadir componentes a un frame interno, se añaden al panel de contenidos del propio frame interno.
  - Un frame interno se debe añadir a un contenedor. Si no lo añadimos a un contenedor (normalmente un JDesktopPane), el frame interno no aparecerá.
  - Normalmente no se tiene que llamar a show o setVisible para los frames internos.

- Los frames internos disparan eventos "internal frame", no eventos "window". El manejo de eventos "internal frame" es casi idéntico al manejo de eventos "window".

- **Crear un Frame Interno**

Constructor	Propósito
JInternalFrame() **JInternalFrame(String)** **JInternalFrame(String, boolean)** **JInternalFrame(String, boolean, boolean)** **JInternalFrame(String, boolean, boolean, boolean)** **JInternalFrame(String, boolean, boolean, boolean, boolean)**	Crea un ejemplar de **JInternalFrame**. El primer argumento especificar el título (si existe) a mostrar por el frame interno. El resto de los argumentos especifican si el frame interno debería contener decoración permitiendo al usuario que redimensione, cierre, maximice y minimice el frame interno (por este orden). El valor por defecto para cada argumento booleano es **false**, lo que significa que la operación no está permitida.
Métodos de la clase**JOptionPane**: • **showInternalConfirmDialog** • **showInternalInputDialog** • **showInternalMessageDialog** • **showInternalOptionDialog**	Crea un **JinternalFrame** que simila un diálogo.

- **Añadir Componentes a un Frame Interno**

Método	Propósito
void setContentPane(Container) **Container getContentPane()**	Selecciona u obtiene el panel de contenido del frame interno, que generalmente contiene todo e GUI del frame interno, con la excepción de la barra de menú y las decoraciones de la ventana.
void setMenuBar(JMenuBar) **JMenuBar getMenuBar()**	Selecciona u obtiene la barra de menú del frame interno. Observa que estos nombres de método no contienen "J", al contrario que sus métodos equivalentes de **JFrame**. En las siguientes versiones de Swing y del JDK 1.2, **JinternalFrame** añadira **setJMenuBar** y **getJMenuBar**, que se deberían utilizar en vez de los métodos existentes.

- **Especificar el Tamaño y la Posición del Frame Interno**

Método	Propósito
void pack()	Dimensiona el frame interno para que sus componentes tenga sus tamaños preferidos.
void setLocation(Point) **void setLocation(int, int)**	Seleciona la posición del frame interno. (Heredada de **Component**).
void setBounds(Rectangle) **void setBounds(int, int, int, int)**	Explicitámente selecciona el tamaño y la localización del frame interno (Heredada de **Component**).
void setSize(Dimension)	Explicitámente selecciona el tamaño del frame interno. (Heredada

void setSize(int, int)	de **Component**).

- **Realizar Operaciones de Ventana sobre el Frame Interno**

Método	**Propósito**
void setDefaultCloseOperation(int) **int getDefaultCloseOperation()**	Selecciona u obtiene lo que hace el frame interno cuando el usuario intenta "cerrar" el frame. El valor por defecto es **HIDE_ON_CLOSE**. Otros posibles valores son **DO_NOTHING_ON_CLOSE** y **DISPOSE_ON_CLOSE**.
void addInternalFrameListener(InternalFrameListener) **void removeInternalFrameListener(InternalFrameListener)**	Añade o elimina un oyente de "internal frame" (**JinternalFrame** es equivalente a un oyente de "window").
void moveToFront() **void moveToBack()**	Si el padre del frame interno es un layeredpane, mueve el frame interno adelante o detrás (respectivamente) por sus capas).
void setClosed(boolean) **boolean isClosed()**	Selecciona u obtiene si el frame interno está cerrado actualmente.
void setIcon(boolean) **boolean isIcon()**	Minimiza o maximiza el frame interno, o determina si está minimizado actualmente.
void setMaximum(boolean) **boolean isMaximum()**	Maximiza o restaura el frame interno o determina si está maximizado.
void setSelected(boolean) **boolean isSelected()**	Selecciona u obtiene si el frame interno esta actualmente "seleccionado" (activado).

- **Controlar la decoración y las capacidades de la ventana**

Método	**Propósito**
void setFrameIcon(Icon) **Icon getFrameIcon()**	Seleciona u obtiene el icono mostrado en la barra de título del frame interno (normalmente en la esquina superior izquierda).
void setClosable(boolean) **boolean isClosable()**	Selecciona u obtiene si el usuario puede cerrar el frame interno.
void setIconifiable(boolean) **boolean isIconifiable()**	Selecciona u obtiene si el frame interno puede ser minimizado.
void setMaximizable(boolean) **boolean isMaximizable()**	Selecciona u obtiene si el usuario puede maximizar el frame interno.
void setResizable(boolean) **boolean isResizable()**	Selecciona u obtiene si el frame interno puede ser redimensionado.
void setText(String) **String getText()**	Selecciona u obtiene el título de la ventana.
void setWarningString(String) **String getWarningString()**	Selecciona u obtiene cualquier string de aviso mostrado en la ventana.

Estudia los siguientes ejemplos

1. ***Ejemplo1***: *Estudia el siguiente código que se corresponde a una aplicación que crea ventanas internas.*

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class InternalVentana extends JFrame implements
InternalFrameListener,ActionListener {

    JTextArea texto;
    JDesktopPane escritorio;
    JInternalFrame ventana;
    JInternalFrame escuchaVentana;
    static final String MOSTRAR = "show";
    static final String LIMPIAR = "clear";
    String nuevaLinea = "\n";
    static final int ancho = 500;
    static final int alto = 300;

    public InternalVentana() {
        super("InternalVentana");


        addWindowListener(new WindowAdapter() {
            public void windowClosing(InternalFrameEvent e) {
                System.exit(0);
            }
        });


        escritorio= new JDesktopPane();

        escritorio.setPreferredSize(new Dimension(desktopWidth,
desktopHeight));
        setContentPane(escritorio);

        createDisplayWindow();
        escritorio.add(displayWindow);

        Dimension displayTamaño = displayWindow.getSize();
        displayWindow.setSize(ancho, displaySize.height);

        //The following probably would save significant time if we reused
        //the internal frame.  We can't reuse it, due to bug #4128975.
        //createListenedToWindow();
    }

    //Create the window that displays event information.
    protected void createDisplayWindow() {
        JButton b1 = new JButton("Show internal frame");
        b1.setActionCommand(SHOW);
        b1.addActionListener(this);
```

```
        JButton b2 = new JButton("Clear event info");
        b2.setActionCommand(CLEAR);
        b2.addActionListener(this);

        display = new JTextArea(5, 40);
        display.setEditable(false);
        JScrollPane textScroller = new JScrollPane(display);

        displayWindow = new JInternalFrame("Event Watcher",
                                           true,  //resizable
                                           false, //not closable
                                           false, //not maximizable
                                           true); //iconifiable
        JPanel contentPane = new JPanel();

contentPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        contentPane.setLayout(new BoxLayout(contentPane,
                                            BoxLayout.Y_AXIS));
        b1.setAlignmentX(CENTER_ALIGNMENT);
        contentPane.add(b1);
        contentPane.add(Box.createRigidArea(new Dimension(0, 5)));
        contentPane.add(textScroller);
        contentPane.add(Box.createRigidArea(new Dimension(0, 5)));
        b2.setAlignmentX(CENTER_ALIGNMENT);
        contentPane.add(b2);

        displayWindow.setContentPane(contentPane);
        displayWindow.pack();
    }

    //Create the listened-to window.
    protected void createListenedToWindow() {
        listenedToWindow = new JInternalFrame("Event Generator",
                                              true,  //resizable
                                              true,  //closable
                                              true,  //maximizable
                                              true); //iconifiable
        //The next statement is necessary to work around bug 4128975.
        listenedToWindow.setDefaultCloseOperation(
                                 WindowConstants.DISPOSE_ON_CLOSE);
        listenedToWindow.setSize(300, 100);
    }

    public void internalFrameClosing(InternalFrameEvent e) {
        displayMessage("Internal frame closing", e);
    }

    public void internalFrameClosed(InternalFrameEvent e) {
        displayMessage("Internal frame closed", e);
        listenedToWindow = null;
    }

    public void internalFrameOpened(InternalFrameEvent e) {
        displayMessage("Internal frame opened", e);
        //XXX: Why do we get one of these when we dispose of a window?
        //XXX: And not when you first show the window?
    }

    public void internalFrameIconified(InternalFrameEvent e) {
        displayMessage("Internal frame iconified", e);
```

```
    }

    public void internalFrameDeiconified(InternalFrameEvent e) {
        displayMessage("Internal frame deiconified", e);
    }

    public void internalFrameActivated(InternalFrameEvent e) {
        displayMessage("Internal frame activated", e);
    }

    public void internalFrameDeactivated(InternalFrameEvent e) {
        displayMessage("Internal frame deactivated", e);
    }

    void displayMessage(String prefix, InternalFrameEvent e) {
        String s = prefix + ": " + e.getSource();
        display.append(s + newline);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals(SHOW)) {
            //Can't show/setVisible, due to bug 4128975.
            //listenedToWindow.setVisible(true);

            //Crea una nueva ventana interna.
            if (listenedToWindow == null) {
                createListenedToWindow();
                listenedToWindow.addInternalFrameListener(this);
                desktop.add(listenedToWindow);
                Dimension size = listenedToWindow.getSize();
                listenedToWindow.setLocation(desktopWidth/2 -
size.width/2,
                                             desktopHeight - size.height);
            }
        } else {
            display.setText("");
        }
    }

    public static void main(String[] args) {
        JFrame frame = new InternalFrameEventDemo();
        frame.pack();
        frame.setVisible(true);
    }
}
```

2. ***Ejemplo2: Ejecuta la siguiente aplicación que muestra una ventana principal con su cuadro de diálogo***

```
public class VentanaPrincipal extends JFrame implements ActionListener {

private Container contenedor;/**declaramos el contenedor*/
JButton botonCambiar;/**declaramos el objeto Boton*/
JLabel labelTitulo;/**declaramos el objeto Label*/
private VentanaPrincipal miVentanaPrincipal;

public VentanaPrincipal(){
/**permite iniciar las propiedades de los componentes*/
```

```
iniciarComponentes();
/**Asigna un titulo a la barra de titulo*/
setTitle("CoDejaVu : JFrame VentanaPrincipal");
/**tamaño de la ventana*/
setSize(300,180);
/**pone la ventana en el Centro de la pantalla*/
setLocationRelativeTo(null);
}

/**
* @param miVentana
* Enviamos una instancia de la ventana principal
*/

public void setVentanaPrincipal(VentanaPrincipal miVentana) {
miVentanaPrincipal=miVentana;
}

private void iniciarComponentes() {
contenedor=getContentPane();/**instanciamos el contenedor*/
/**con esto definmos nosotros mismos los tamaños y posicion
* de los componentes*/

contenedor.setLayout(null);

/**Propiedades del boton, lo instanciamos, posicionamos y
* activamos los eventos*/
botonCambiar= new JButton();
botonCambiar.setText("Iniciar");
botonCambiar.setBounds(100, 80, 80, 23);
botonCambiar.addActionListener(this);

 /**Propiedades del Label, lo instanciamos, posicionamos y
 * activamos los eventos*/
labelTitulo= new JLabel();
labelTitulo.setText("VENTANA PRINCIPAL");
labelTitulo.setBounds(80, 20, 180, 23);

 /**Agregamos los componentes al Contenedor*/
      contenedor.add(labelTitulo);
      contenedor.add(botonCambiar);
 }
 /**Agregamos el evento al momento de llamar la otra ventana*/
 @Override
 public void actionPerformed(ActionEvent evento) {
if (evento.getSource()==botonCambiar)
      {
/*enviamos la instancia de la ventana principal para que
* esta sea Padre de la ventana de dialogo*/
      VentanaConfirmacion miVentanaConfirmacion=new
VentanaConfirmacion(miVentanaPrincipal,true);
miVentanaConfirmacion.setVisible(true);
}
}
}
```

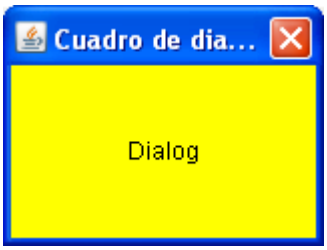
Jdialog:

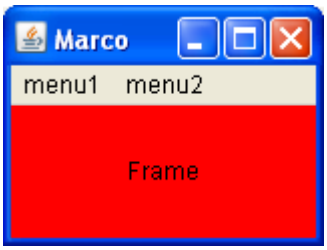
```
public class VentanaConfirmacion extends JDialog{
```

```
private Container contenedor;
JLabel labelTitulo;

public VentanaConfirmacion(VentanaPrincipal miVentanaPrincipal, boolean
modal){
/**Al llamar al constructor super(), le enviamos el
* JFrame Padre y la propiedad booleana que determina
* que es hija*/
super(miVentanaPrincipal, modal);
contenedor=getContentPane();
contenedor.setLayout(null);
//Asigna un titulo a la barra de titulo
setTitle("CoDejaVu : JDialog VentanaConfirmacion");

labelTitulo= new JLabel();
labelTitulo.setText("VENTANA DE CONFIRMACION");
labelTitulo.setBounds(20, 20, 180, 23);

contenedor.add(labelTitulo);
//tamaño de la ventana
setSize(350,150);
//pone la ventana en el Centro de la pantalla
setLocationRelativeTo(null);
}
}
```

Secuencia y desarrollo:

3. ***Ejercicio1***: ***Ejemplo1*** : *Implementa las siguientes ventanas en modo texto. Estudia las diferencias de cada una de ellas.*





Creación de una ventana

```
import java.applet.*;
import java.awt.*;

public class Ventana extends Window {
Ventana(Frame fr,int x,int y,int dx, int dy) {
        super(fr);
        Label lbl=new Label("Window",Label.CENTER);
        lbl.setBackground(Color.cyan);
        add("Center",lbl);
        setSize(dx,dy);
        setLocation(x,y);
        setVisible(true);
}
}
```

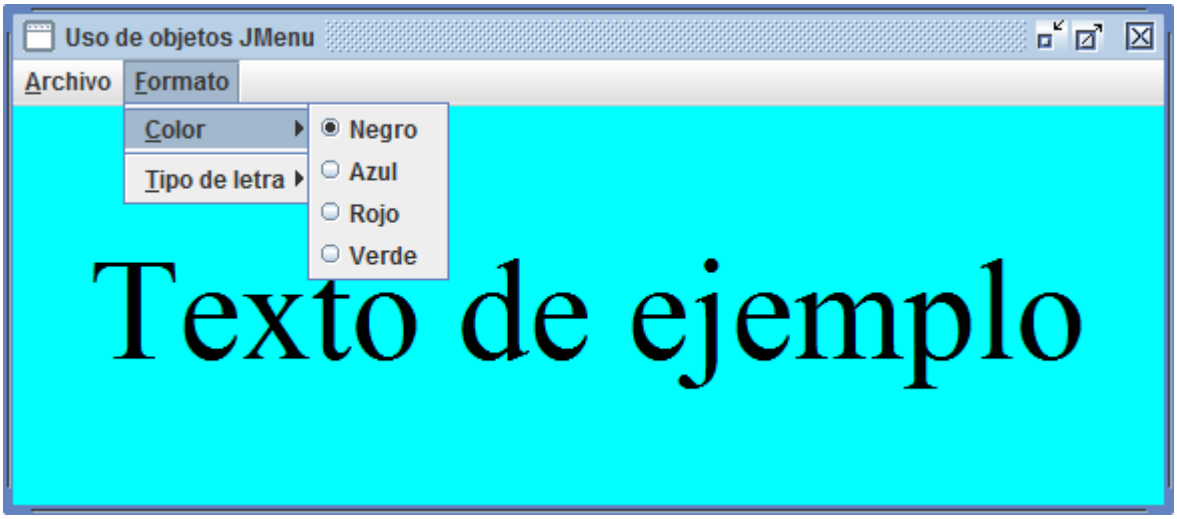
Creación de un marco

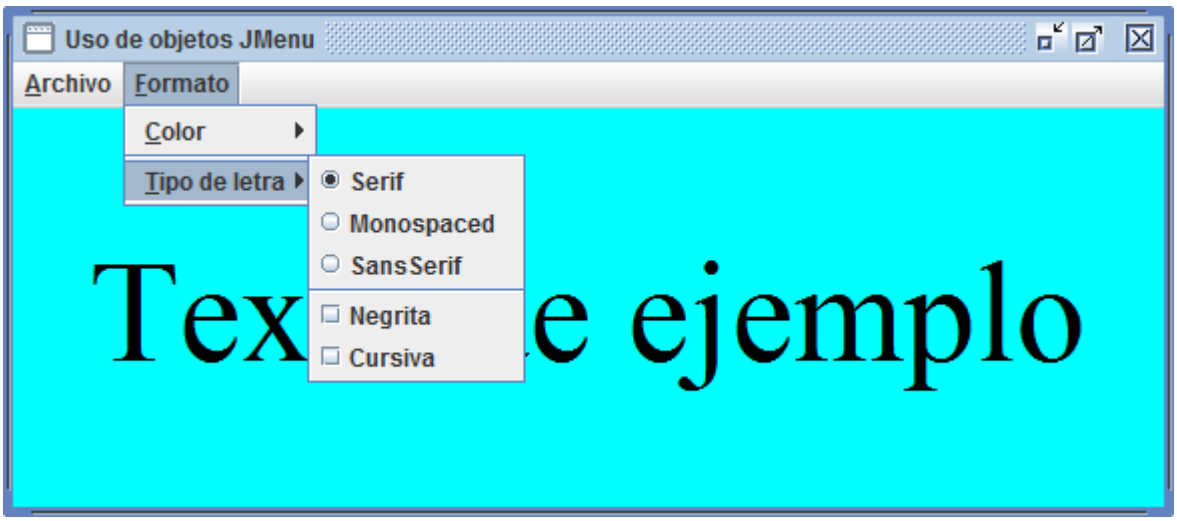
```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class Marco extends JFrame implements WindowListener {

Marco(String título,int x,int y,int dx,int dy) {
        super(título);
        setBackground(Color.red);
        Jlabel lbl=new Jlabel("Ventana principal);
        setBackground(Color.red);
        add("Center",lbl);

        JmenuBar menubar=new MenuBar();
        setMenuBar(menubar);
        Menu menu1=new Menu("menu1");
        menu1.add(new MenuItem("item 1.1"));
        menu1.add(new MenuItem("item 1.2"));
        menu1.add(new MenuItem("item 1.3"));
        Menu menu2=new Menu("menu2");
        menu2.add(new MenuItem("item 2.1"));
        menu2.add(new MenuItem("item 2.2"));
        menubar.add(menu1);
        menubar.add(menu2);


        setSize(dx,dy);
        setLocation(x,y);
        setVisible(true);
        addWindowListener(this);
}

/*------------ WindowListener ----------*/
   public void windowOpened(WindowEvent e){}
   public void windowActivated(WindowEvent e){}
   public void windowDeactivated(WindowEvent e){}
   public void windowIconified(WindowEvent e){}
   public void windowDeiconified(WindowEvent e){}
   public void windowClosed(WindowEvent e){}
public void windowClosing(WindowEvent e){
        hide();
}

}
```

Creación de un cuadro de diálogo

```
public class Dialogo extends JDialog {

Dialogo(Frame madre,String título,boolean modal,int x,int y,int dx,int dy) {
        super(madre,título,modal);
        Label lbl=new Label("Dialog",Label.CENTER);
        lbl.setBackground(Color.yellow);
        add("Center",lbl);
        setSize(dx,dy);
        setLocation(x,y);
        setVisible();
}
        }
```

1. Ejercicio1:





- Añadir al menú **Archivo** la opción **Nuevo** para abrir una nueva ventana interna.