Unidad de trabajo 9: Interfaces gráficas

RESULTADO DE APRENDIZAJE
RA5. Realiza operaciones de entrada y salida de información, utilizando procedimientos específicos del lenguaje y librerías de clases.

Tabla de contenido

1. Instalar plugin

En eclipse el plugin ***WindowBuilder***, desarrollado por Google, que permite desarrollar de forma rápida y cómoda la GUI (interfaz gráfica de usuario) de las aplicaciones Java.

Desde Eclipse pulsamos el menú ***Help/Eclipse Marketplace.***

Buscamos ***WindowBuilder 1.9.3,*** aunque aparece la 1.9.4, pero es más novedosa, con pocas descargas, y seguramente menos probada. Para instalarlo pulsa el botón ***Install***. Véase la figura 2.33.

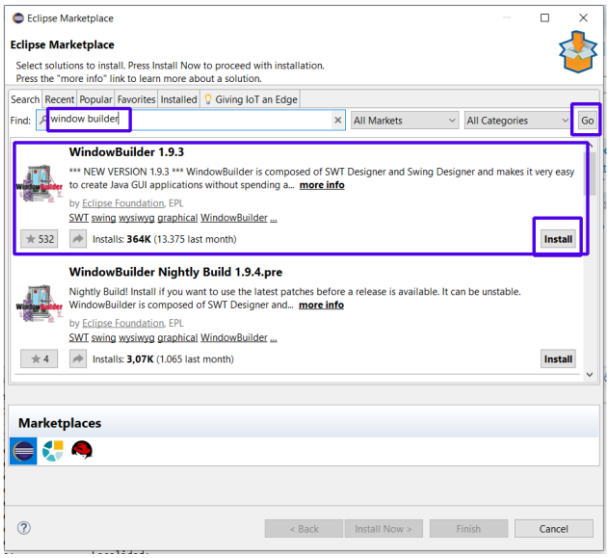


Figura 2.33. Eclipse MarketPlace.

Para comprobar que el plugin se ha instalado podemos abrir la ayuda de eclipse desde el menú ***Help/About Eclipse IDE***, y comprobar que aparece en la lista de plugins, véase la figura 2.34.

Figura 2.34. Plugin instalados en Eclipse.

2. Creación de un proyecto con una interfaz gráfica.

Clases que vamos a utilizar para trabajar con ventanas:

- ***JFrame*** es una clase utilizada en ***Swing*** (biblioteca gráfica de java) para generar ventanas sobre las cuales añadir distintos objetos con los que podrá interactuar o no el usuario. Normalmente la clase ***JFrame*** se emplea para crear la ventana principal de una aplicación. ***JFrame*** posee algunas nociones típicas de una ventana como minimizar, cerrar, maximizar y poder moverla.

- ***JDialog*** es una clase utilizada para generar cuadros de diálogo, se puede considerar como una ventana emergente que aparece cuando se debe mostrar un mensaje. No es una ventana completamente funcional como el ***JFrame***.
- ***JPanel*** es un contenedor puro y no es una ventana en sí misma. El único propósito es organizar los componentes en una ventana**.** Un ***JFrame*** contiene a un ***JPanel*** para colocar los componentes de la ventana.

Para crear un proyecto abrimos el menú ***File/New/Java Project***, y en la ventana que aparece se escribe el nombre del proyecto, se dejan las opciones por defecto y se pulsa el botón ***Finish***.

A continuación, se añade una ventana al proyecto, para ello nos posicionamos sobre el nombre del proyecto, pulsamos el botón derecho del ratón y en el menú contextual se elige ***New/Other***. Se busca la carpeta ***WindowBuilder/Swing Designer*** y se elige ***JFrame***, véase la figura 2.36.

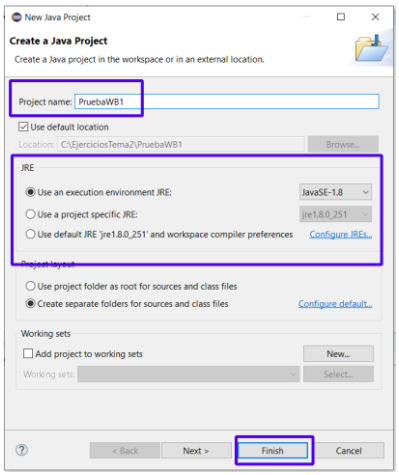


Figura 2.35. Creación de un proyecto Java

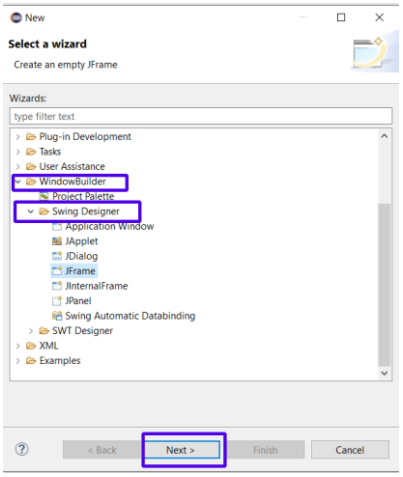


Figura 2.36. Creación de un JFrame.

Observa en el explorador del proyecto que se ha creado una clase con el nombre puesto a la ventana, esa clase se puede editar en modo ***Source (fuente)*** o en modo ***Desing (diseño)***. En la figura 2.37 se muestra la ventana en modo diseño, se distinguen varios bloques:

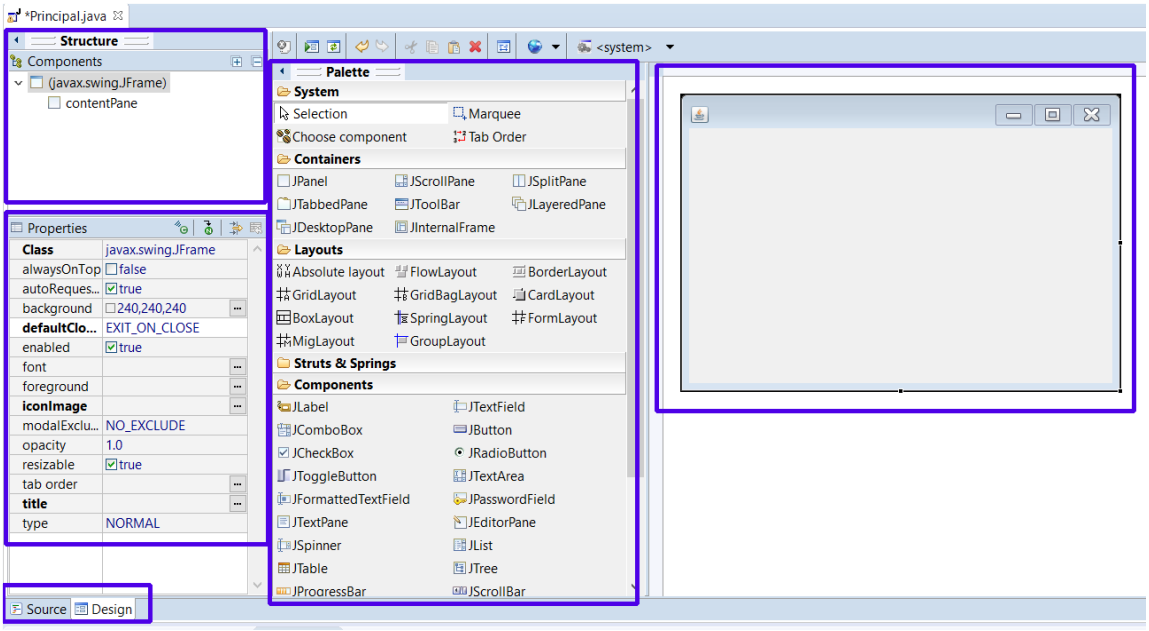


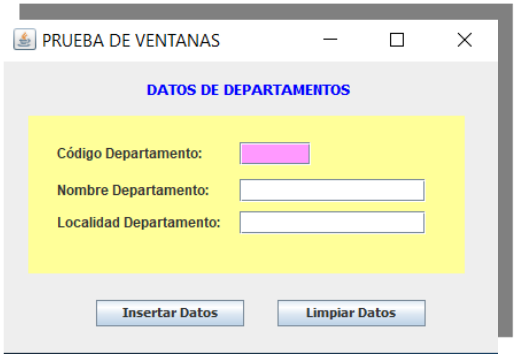
Figura 2.37. Vista diseño de la interfaz gráfica.

- ***Structure***, donde se ven de manera jerárquica los componentes que se han agregado a la ventana. Inicialmente aparece el primer componente el JFrame, y dentro un JPanel.
- ***Properties***, se muestran las propiedades del elemento o componente seleccionado, como el título, tipo de letra, tamaño, color, etc., estas propiedades se pueden cambiar.
- ***Palette***, se muestran los elementos de tipo contenedor que se pueden añadir a la ventana; los *layouts*, es decir la distribución de los elementos en la ventana; o los componentes, tipo etiquetas, cajas de texto, listas desplegables, etc., que se pueden añadir a la ventana
- ***La ventana o formulario***, que es donde se van añadiendo los elementos. Al seleccionar un componente de la estructura jerárquica, se seleccionará en esta ventana

IMPORTANTE:

Para añadir componentes a la ventana primero se pulsa ***Absolute layout***, en ***Palette/Layouts***, y se arrastra al marco interno de la ventana (este marco interno es un ***JPanel*** con el nombre de ***contentPane***) esto va a permitir colocar los componentes en cualquier parte de la ventana.

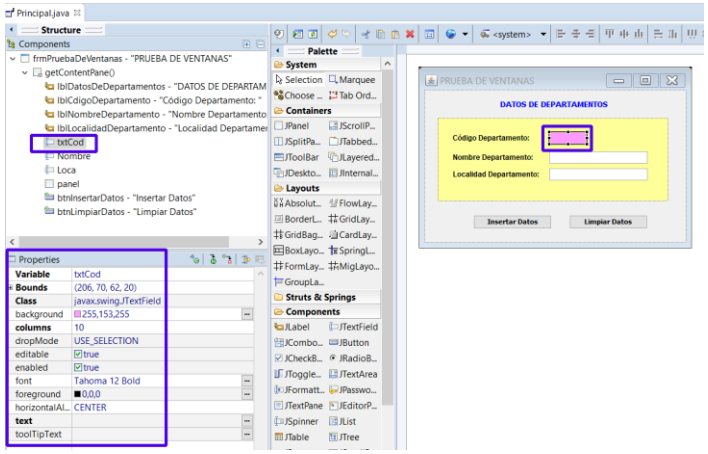
Ejemplo de ventana



Controles más comunes. Elementos de esta ventana:

- ***JLabel***. Representa a una etiqueta en la ventana, la utilizamos para poner títulos, en el ejercicio las etiquetas son el rótulo DATOS DE DEPARTAMENTOS, Código Departamento, Nombre Departamento, Localidad Departamento.
- ***JTextField***. Representa a las cajas de texto, se utilizan para entrada de datos. En el ejercicio son las casillas vacías.
- ***JButton***. Este componente se utiliza para añadir los botones de la ventana
- ***JPanel***. Es un panel que agrupa a componentes, en el ejercicio es la caja que encierra a los componentes ***JLabel*** y ***JTextField***. Primero se crea el *JPanel*, y luego se añaden los componentes.

Observa el panel de propiedades. En la propiedad ***Variable*** es donde ponemos el nombre del control, es importante el nombre, para luego poderlo referenciar dentro del programa. Para cambiar el aspecto utiliza: *background*, *font*, *foreground*, *text*, *horizontalAlignment*, *verticalAlignment* etc.



Observa el código fuente que se va generando en la clase al ir añadiendo los controles.

3. Añadir acción a los botones

Se hace doble clic sobre el botón y se visualiza el código ***actionPerformed*** asociado al botón, las acciones se añadirían en ese método. En el ejemplo se visualiza un mensaje:

```
JButton btnInsertarDatos = new JButton("Insertar Datos");
btnLimpiarDatos.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
        {
            System.out.println(" SE HA PULSADO EL BOTÓN INSERTAR");
         }
   });
```

Si se desea trabajar con los controles en toda la aplicación, para cargar valores o asignar valores, como es el caso de las cajas de texto, se debe de convertir el control a ***field***, es decir definir el control como un atributo de la clase. Se selecciona el control y en los botones de la paleta de propiedades se marca el botón ***Convert local to field***, véase la figura 2.40.

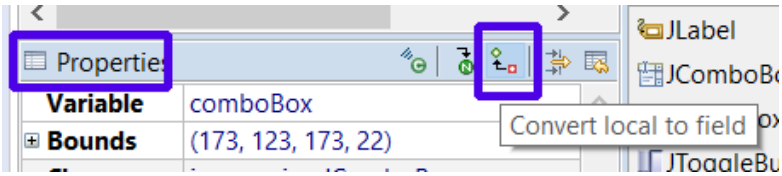


Figura 2.40. convertir Local a *field*.

- Para asignar valores a una caja de texto utilizamos el método ***setText***("*Texto a asignar*").
- Y para cargar el valor que tenga almacenado, el método ***getText***().

4. JCOMBOBOX

Añadimos un ***JCombobox*** para seleccionar un nombre de país. Una vez añadido a la ventana, abrimos la paleta de propiedades, y en la propiedad ***model***, añadimos los elementos del combo. Véase la figura 2.41.

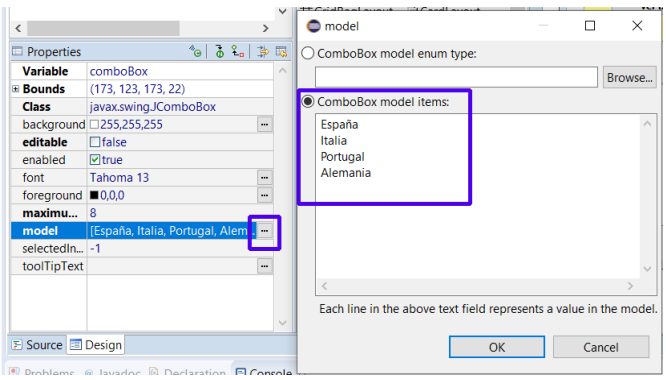


Figura 2.41. Propiedad ***model*** del JComboBox

Los métodos utilizados para cargar el elemento actual del ***ComboBox*** y saber lo que se ha seleccionado son ***getSelectedIndex(),*** que devuelve un entero con el índice del elemento seleccionado, el primer elemento tiene la posición 0. Y ***getSelectedItem()*** devuelve una cadena con el contenido.

```
System.out.println("Posición: " + comboBox.getSelectedIndex());
System.out.println("Contenido: " + comboBox.getSelectedItem());
```

Llenar un JComboBox con datos de una tabla

En el ejemplo el JComboBox se llama **listadepar** y es un atributo de clase. Este método crea un combo con datos de los códigos de departamento. Se utilizan los métodos **removeAllItems**(); para borrar los elementos que hubiera **addItem**(Objeto), para añadir un elemento

```
private void cargarcombodepart() {

	listadepar.removeAllItems(); //Limpio la lista
	listadepar.addItem("   "); //Añado el elemento en blanco
	try {
		Class.forName("com.mysql.jdbc.Driver");
		Connection conexion = DriverManager.getConnection
		 ("jdbc:mysql://localhost/ejemplo", "ejemplo", "ejemplo");
		Statement sentencia = conexion.createStatement();
		ResultSet res = sentencia.executeQuery(
				"select dept_no, dnombre from departamentos");
		while (res.next()){
			listadepar.addItem(res.getInt(1));
		}
		res.close();
		sentencia.close();
		conexion.close();

		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
```

El elemento seleccionado del combo se devuelve con el método **getSelectedItem().** Para saber la posición del elemento seleccionado utilizamos el método **getSelectedIndex().** Se empieza en la posición 0.

Para saber el número de elementos del combo utilizamos el método **getItemCount()**

Para seleccionar un elemento de la lista utilizamos el método **setSelectedIndex(posición),** en posición indicamos el índice del elemento a acceder, empezando en 0

```
int posicion = comboBox.getSelectedIndex(); //devuelve el índice del elemento seleccionado
String contenido = comboBox.getSelectedItem().toString(); //devuelve el contenido del elemento seleccionado
```

5. JTextArea

Añadimos a continuación un ***JTextArea***, para visualizar mensajes.

Si se va a utilizar este componente, se incluye dentro de un ***JScrollPane***, para añadir las barras de desplazamiento al textárea. Así pues, primero se crea el ***JScrollPane*** y luego el ***JTextArea***, que se arrastrará al *Viewport* del ***JScrollPane***. Véase la figura 2.42.

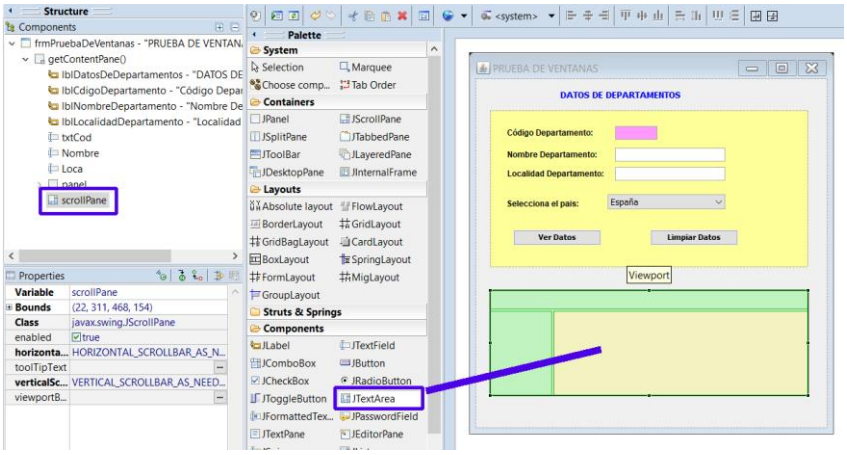


Figura 2.42. Insertar ***JTextArea*** en un ***JScrollPane***

Los métodos para asignar texto en el téxárea son ***setText("texto")***, si quiero limpiarlo asigno vacío, es decir ***setText("").***

Y cuando quiera añadir líneas utilizo el método ***append("Texto a añadir")***.

Para que no se pueda escribir y borrar en el textárea se desmarca la propiedad **Editable**.

Al utilizar ***append*** hay que añadir saltos de línea, porque si no, el texto se visualiza todo seguido. El salto de línea se pone con **\n**. Por ejemplo, si el *textárea* lo he llamado *textArea* pondré:

```
textArea.setText(""); //Limpio el textárea
textArea.setText("Primera línea."); //asigno contenido
textArea.append("\nEsta es la segunda línea"); //añado una línea
textArea.append("\nEsta es la siguiente"); //añado otra
```

6. JRadioButton

Normalmente los botones de radio suelen ir en un grupo para marcar sólo uno de ellos, e indicar una sola opción. Para crearlos, primero se añadirán los botones de radio, luego los seleccionamos a todos, y se indicará que forman parte del mismo grupo. Con el botón derecho del ratón con todos seleccionados elegiremos del menú contextual las opciones ***SetButtonGroup->New standard***.

Añadimos un grupo de botones de radio para indicar el tipo de departamento, por ejemplo, Gestión, Personal, Apoyo y Tecnológico. Y los llamamos igual. Véase la figura 2.43.

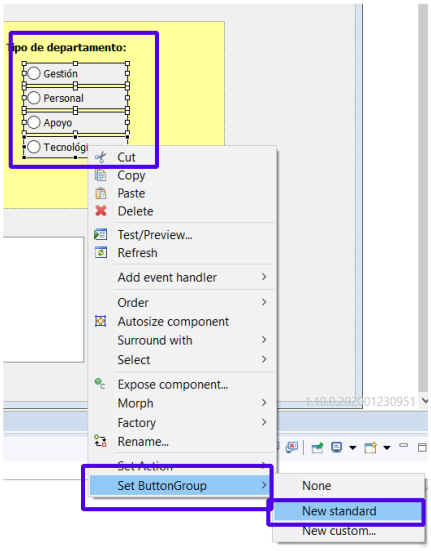


Figura 2.43. Añadir Grupo de botones de radio

Se creará un atributo de clase como este:

```
private final ButtonGroup buttonGroup = new ButtonGroup();
```

Y los botones se añadirán automáticamente al grupo con las instrucciones:

```
buttonGroup.add(gestion);
buttonGroup.add(personal);
buttonGroup.add(apoyo);
buttonGroup.add(tecnologico);
```

Cuando queramos saber qué botón se ha seleccionado, por ejemplo, al pulsar el botón *Insertar datos* del ejercicio anterior, cargaremos los botones en un ***ButtonModel***, cada uno en uno:

```
ButtonModel gest= gestion.getModel();
ButtonModel pers= personal.getModel();
ButtonModel apo= apoyo.getModel();
ButtonModel tecno= tecnologico.getModel();
```

Recuerda que los componentes para utilizarlos en toda la aplicación se tienen que convertir a ***Field***. Igualmente, en el programa habrá que importar la clase ***ButtonModel***. Se podrá hacer automáticamente yendo al error e importando la clase desde el menú contextual.

Y para preguntar por el botón seleccionado, se pregunta si el *ButtonModel* seleccionado del grupo coincide con alguno de los *ButtonModel* creados anteriormente. Este sería el código:

```
if (buttonGroup.getSelection()!=null)
{
    if (buttonGroup.getSelection().equals(gest))
                System.out.println("Pulsaste gestion ");
    if (buttonGroup.getSelection().equals(pers))
                System.out.println("Pulsaste personal ");
    if (buttonGroup.getSelection().equals(apo))
                System.out.println("Pulsaste apoyo ");
    if (buttonGroup.getSelection().equals(tecno))
                System.out.println("Pulsaste tecnológico ");
}
```

7. JCheckBox

Para preguntar si el componente se ha seleccionado o no, utilizaremos el método ***isSelected(),*** que devolverá *true* si se ha marcado, y *false* si no. Por ejemplo, en estas líneas visualizo el contenido de un ***JCheckBox*** con nombre *check* si este aparece marcado, es decir si está seleccionado:

```
if (check.isSelected()) {
      System.out.println("Check Pulsado: " + check.getText());
 }
```

8. Crear JDialog

Un ***JDialog*** es una ventana de diálogo que se utiliza para dar un mensaje o alguna especificación del programa o sencillamente dar a conocer un resultado. Se pueden añadir los controles que se deseen. A diferencia de un ***JFrame***, un ***JDialog*** no muestra el botón de aplicación en la barra de tareas cuando se está ejecutando, el ***JFrame*** sí. El ***JDialog*** es invocado desde una ventana padre, que puede ser un ***JFrame*** u otro ***JDialog***.

Puede ser ***modal*** o ***no modal***. Es modal cuando este al estar abierto, impide que se abran otras ventanas de la aplicación hasta que este se cierre. Y no modal cuando se pueden abrir más ventanas dentro de la aplicación, y no se necesita que se cierren.

Un ***JDialog*** es una clase, y tenemos que crearla añadiendo una nueva clase al proyecto. Así pues, dentro del proyecto de pruebas, pulsamos botón derecho del ratón y en el menú contextual se elige ***New/Other***. Se busca la carpeta ***WindowBuilder/Swing Designer*** y se elige ***JDalog.*** Escribimos el nombre, por ejemplo, *Dialogo1*, y nos aseguramos que la clase *Principal* y esta nueva clase, estén en el mismo paquete (*Package*). Se pulsa *Finish* y se crea la ventana.

Observa que se crea con el método ***main(),*** al tener este método podrá ejecutarse independiente de la ventana padre que lo abrirá. Prueba la ejecución y observa que no se muestra el icono de aplicación en la barra de tareas de Windows. Observa también que viene con dos botones ya definidos *OK* y *Cancel*. Estos se podrán cambiar si nos interesa escribir otros nombres. En el ejemplo creamos el siguiente cuadro de diálogo para insertar datos de oficinas:

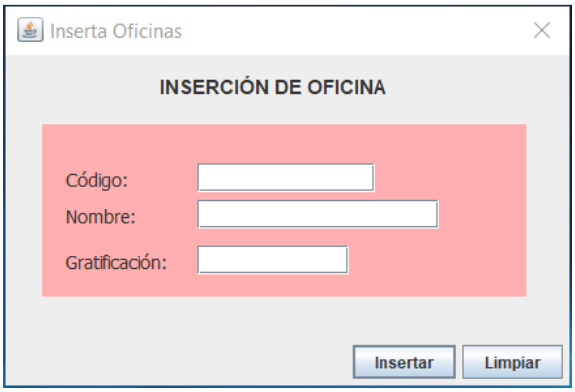


Figura 2.45. Ventana de Diálogo

Añadimos un botón en otra ventana y que abra el JDIALOG. El código a poner para abrir el JDIALOG al pulsar el botón el siguiente, se importará la clase *javax.swing.JDialog*

```
JButton btnNewButton = new JButton("Insertar Oficina");
btnNewButton.addActionListener(new ActionListener() {
```

```
        public void actionPerformed(ActionEvent arg0) {

                try {
                        Dialogo1 dialog = new Dialogo1();
                        dialog.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
                        dialog.setVisible(true);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
    });
```

Si pruebas la ejecución, verás que al pulsar el botón creado para abrir el jdialog, se abrirá múltiples veces. Por ser un **diálogo no modal.** Véase la figura 2.46.

Para cambiar el *JDialog* y **convertirlo a modal**, en el diseño del cuadro de diálogo, buscamos la propiedad ***modal*** en la paleta de propiedades y marcamos la casilla para que aparezca **true**. Prueba de nuevo la ejecución, y ahora hasta que no se cierre el cuadro de diálogo, no podremos movernos a otra ventana de la aplicación.

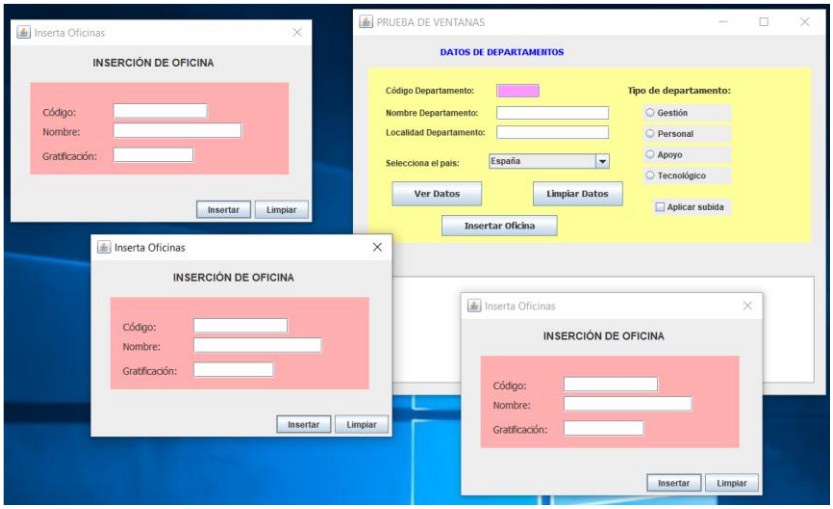


Figura 2.46. Diálogo no modal.