Introducción a Swing con IntelliJ

Introducción a Swing con IntelliJ

Swing es un framework de Java que permite crear ventanas con componentes habituales de las interfaces gráficas, como botones, campos de texto, listas desplegables, etc.

Los componentes de Swing se basan en el patrón MVC:

- **Modelo**: son los datos del componente
- **Vista**: es la representación gráfica de los datos
- **Controlador**: toma una entrada del usuario a través de la vista, y refleja los cambios sobre los datos del componente.

Cada interfaz de usuario queda definida por:

- **Componentes**: son los elementos gráficos que contiene, como botones, campos de texto, etc.
- **Layout**: es la forma en que se organizan los elementos entre sí.
- **Comportamiento**: es el conjunto de eventos que ocurren cuando un usuario interactúa con los componentes.

En general podemos encontrar dos tipos de componentes:

- **Contenedores**: son elementos que permiten alojar otros componentes. Los tres tipos de contenedores principales son:
  - **Marco**: ventana principal con título, icono de cierre, donde se pueden insertar otros componentes que no sean de tipo marco.
  - **Panel**: es un elemento que permite agrupar y organizar otros componentes. Un panel puede incluir otros paneles.
  - **Diálogo**: es un marco, aunque se comporta como una ventana emergente.
- **Controles**: existen muchos tipos de controles predefinidos, aunque se pueden añadir otros. Los elementos mínimos para poder crear una ventana con funcionalidad son:
  - **JLabel**: es una etiqueta de texto.
  - **JButton**: es un botón.
  - **JTextField**: es un campo de texto.

Crear una aplicacion con Swing

Para crear una nueva aplicación con Swing en IntelliJ, no hay que hacer nada diferente al crear el proyecto. Las aplicaciones Swing se pueden crear manualmente o bien utilizando el plugin *Swing UI Designer* de IntelliJ que permite diseñar gráficamente un formulario.

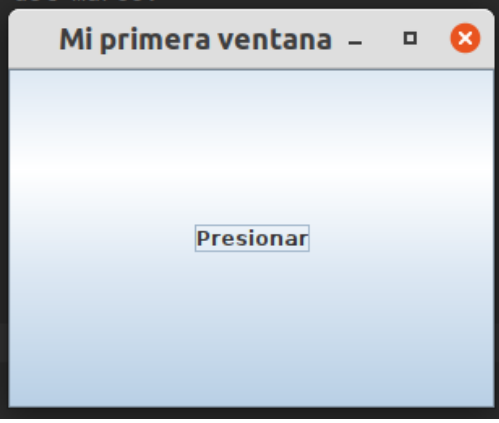
Creación manual de una aplicación con Swing

La creación manual de una aplicación con Swing puede ser tediosa, ya que requiere añadir manualmente los componentes, y entender los diferentes layouts. El código necesario para crear una nueva aplicación con Swing manualmente es el siguiente:

```
        // 1. Creación de un marco y asignación de texto a la barra de
título.
        JFrame frame = new JFrame("Mi primera ventana");
        // 2. Establecer el comportamiento del botón de cierre del marco.
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 3. Definir el tamaño de la ventana.
        frame.setSize(300, 300);
        // 4. Crear un nuevo componente.
        JButton boton = new JButton("Presionar");
        // 5. Añadir el botón al marco.
        frame.getContentPane().add(boton);
        // 6. Mostrar el marco
        frame.setVisible(true);
```

Si colocamos este código en un método *main* podemos ejecutar la ventana.

```
public static void main(String[] args) {

    // 1. Creación de un marco y asignación de texto a la barra de título.
    JFrame frame = new JFrame( title: "Mi primera ventana");
    // 2. Establecer el comportamiento del botón de cierre del marco.
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    // 3. Definir el tamaño de la ventana.
    frame.setSize( width: 300,  height: 300);
    // 4. Crear un nuevo componente.
    JButton boton = new JButton( text: "Presionar");
    // 5. Añadir el botón al marco.
    frame.getContentPane().add(boton);
    // 6. Mostrar el marco
    frame.setVisible(true);
}
```



En este punto el botón no hace nada.

Estrategias de cierre

El método *setDefaultCloseOperation* permite asignar uno de los siguientes valores al atributo *DefaultCloseOperation*, que indica qué hace la ventana cuando se pulsa el icono de cierre:

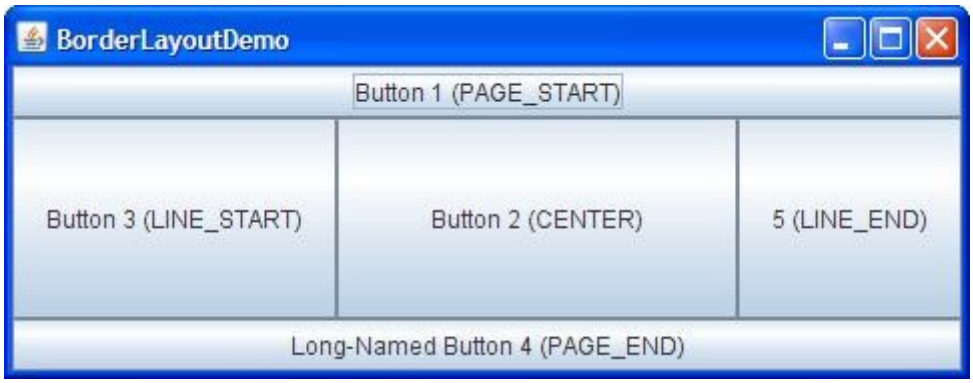
- **JFrame.EXIT_ON_CLOSE**: cerrar la aplicación. Se utiliza en la ventana principal. Al cerrar esta ventana, la aplicación termina.
- **JFrame.HIDE_ON_CLOSE**: oculta la ventana, pero mantiene la aplicación corriendo.

- **JFrame.DISPOSE_ON_CLOSE**: cierra la ventana, libera los recursos pero mantiene la aplicación corriendo. Se utiliza en ventanas secundarias, si no queremos que al cerrar una de ellas se cierre la aplicación completa.
- **JFrame.DO_NOTHING_ON_CLOSE**: ignora el clic sobre sobre el botón de cierre.

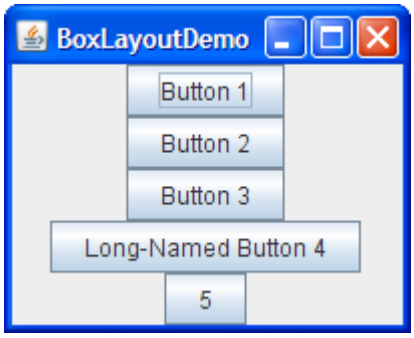
LAYOUTS

Los layouts definen el modo en que se colocan los elementos en una ventana. Algunos de estos layouts son:

- **BorderLayout**: es el layout por defecto. Coloca los elementos en cinco zonas: *top, bottom, left, right* y *center*



- **BoxLayout**: coloca los elementos en una única columna

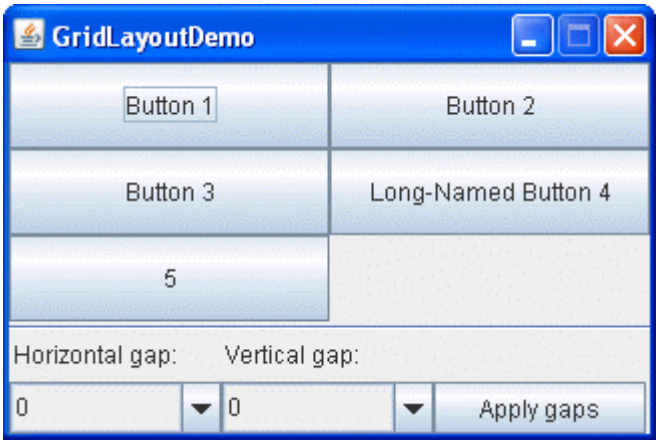


- **CardLayout**: permite que un área contenga diferentes componentes según el momento.

- **FlowLayout**: coloca los componentes en una única fila. Si los elementos no caben en una única fila, comienza una nueva fila.

- **GridBagLayout**: alinea los componentes en una cuadrícula de celdas, permitiendo a los componentes expandirse por más de una celda (igual que una tabla HTML).

- **GridLayout**: dispone los componentes en un número de filas y columnas.



Algunos ejemplos con layout

La forma de utilizar cada layout es diferente, y es necesario consultar la documentación. A continuación se mostrarán algunos ejemplos:

- **FlowLayout**

```
        JFrame frame = new JFrame("Mi primera ventana");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        JPanel panelSuperior = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        panelSuperior.setBackground(Color.cyan);
        panelSuperior.add(new JButton("Sup.1"));
        panelSuperior.add(new JButton("Sup.2"));
        panelSuperior.add(new JButton("Sup.3"));
        panelSuperior.add(new JButton("Sup.4"));
        frame.setContentPane(panelSuperior);
        frame.setVisible(true);
```

> Además de *FlowLayout.LEFT* (alineación izquierda) se puede usar *FlowLayout.RIGHT* (alineación derecha) o no poner nada (alineación centrada).

- **BoxLayout**

```
        JFrame frame = new JFrame("Mi primera ventana");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        panel.setBackground(Color.red);
        panel.add(new JButton("1"));
        panel.add(new JButton("2"));
        panel.add(new JButton("3"));
        frame.setContentPane(panel);
        frame.setVisible(true);
```

> La orientación de los componentes puede ser *BoxLayout.X_AXIS* (horizontal) o *BoxLayout.Y_AXIS* (vertical)

- **BorderLayout**

```
        JFrame frame = new JFrame("Mi primera ventana");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());
        panel.setBackground(Color.red);
        panel.add(new JButton("centro"),BorderLayout.CENTER);
        panel.add(new JButton("izquierda"),BorderLayout.WEST);
```

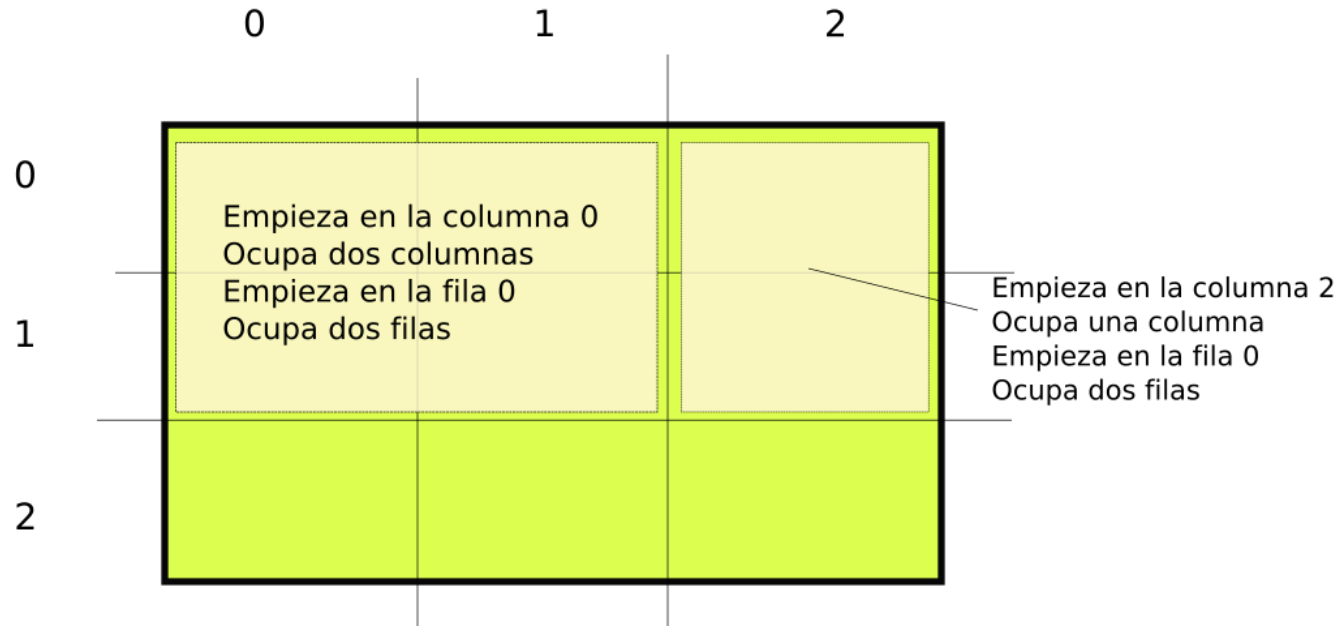
```
        panel.add(new JButton("derecha"),BorderLayout.EAST);
        panel.add(new JButton("arriba"),BorderLayout.NORTH);
        panel.add(new JButton("abajo"),BorderLayout.SOUTH);
        frame.setContentPane(panel);
        frame.setVisible(true);
```

- **GridLayout**

```
        JFrame frame = new JFrame("Mi primera ventana");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);

        JPanel panelPrincipal = new JPanel();
        panelPrincipal.setLayout(new GridLayout(3,2,10,10));
        // 3 - filas, 2 - columnas, 10 - espaciado horizontal, 10 -
espaciado vertical
        panelPrincipal.setBackground(Color.red);
        panelPrincipal.add(new JButton("1.1"));
        panelPrincipal.add(new JButton("1.2"));
        panelPrincipal.add(new JButton("2.1"));
        panelPrincipal.add(new JButton("2.2"));
        panelPrincipal.add(new JButton("3.1"));
        frame.setContentPane(panelPrincipal);
        frame.setVisible(true);
```

- **GridBagLayout**

Este layout requiere el uso de restricciones para ir modificando las dimensiones de cada componente conforme se va agregando. Antes de empezar, hay que pensar el diseño del grid. Por ejemplo:




```
JFrame frame = new JFrame("Mi primera ventana");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300, 300);

JPanel panelPrincipal = new JPanel();
panelPrincipal.setLayout(new GridBagLayout());

GridBagConstraints restricciones = new GridBagConstraints ();

panelPrincipal.setBackground(Color.red);
```

```
        restricciones.gridx = 0; // El área de texto empieza en la columna
cero.
        restricciones.gridy = 0; // El área de texto empieza en la fila
cero
        restricciones.gridwidth = 2; // El área de texto ocupa dos
columnas.
        restricciones.gridheight = 2; // El área de texto ocupa 2 filas.
        restricciones.weightx = 1; // Expandir el espacio ocupado
horizontalmente
        restricciones.weighty = 1; // Expandir el espacio ocupado
verticalmente
        restricciones.fill = GridBagConstraints.BOTH; // Expandir el
componente
        panelPrincipal.add(new JButton("1.1"),restricciones);

        restricciones.gridx = 2;
        restricciones.gridy = 1;
        restricciones.gridwidth = 1;
        restricciones.gridheight = 1;

        panelPrincipal.add(new JButton("1.2"), restricciones);

        restricciones.gridx = 0;
        restricciones.gridy = 2;
        restricciones.gridwidth = 1;
        restricciones.gridheight = 1;

        panelPrincipal.add(new JButton("2.1"), restricciones);

        restricciones.gridx = 1;
        restricciones.gridy = 2;
        restricciones.gridwidth = 1;
        restricciones.gridheight = 1;

        panelPrincipal.add(new JButton("2.2"),restricciones);

        restricciones.gridx = 2;
        restricciones.gridy = 2;
        restricciones.gridwidth = 1;
        restricciones.gridheight = 1;

        panelPrincipal.add(new JButton("2.3"), restricciones);
```

```
        frame.setContentPane(panelPrincipal);
        frame.setVisible(true);
```

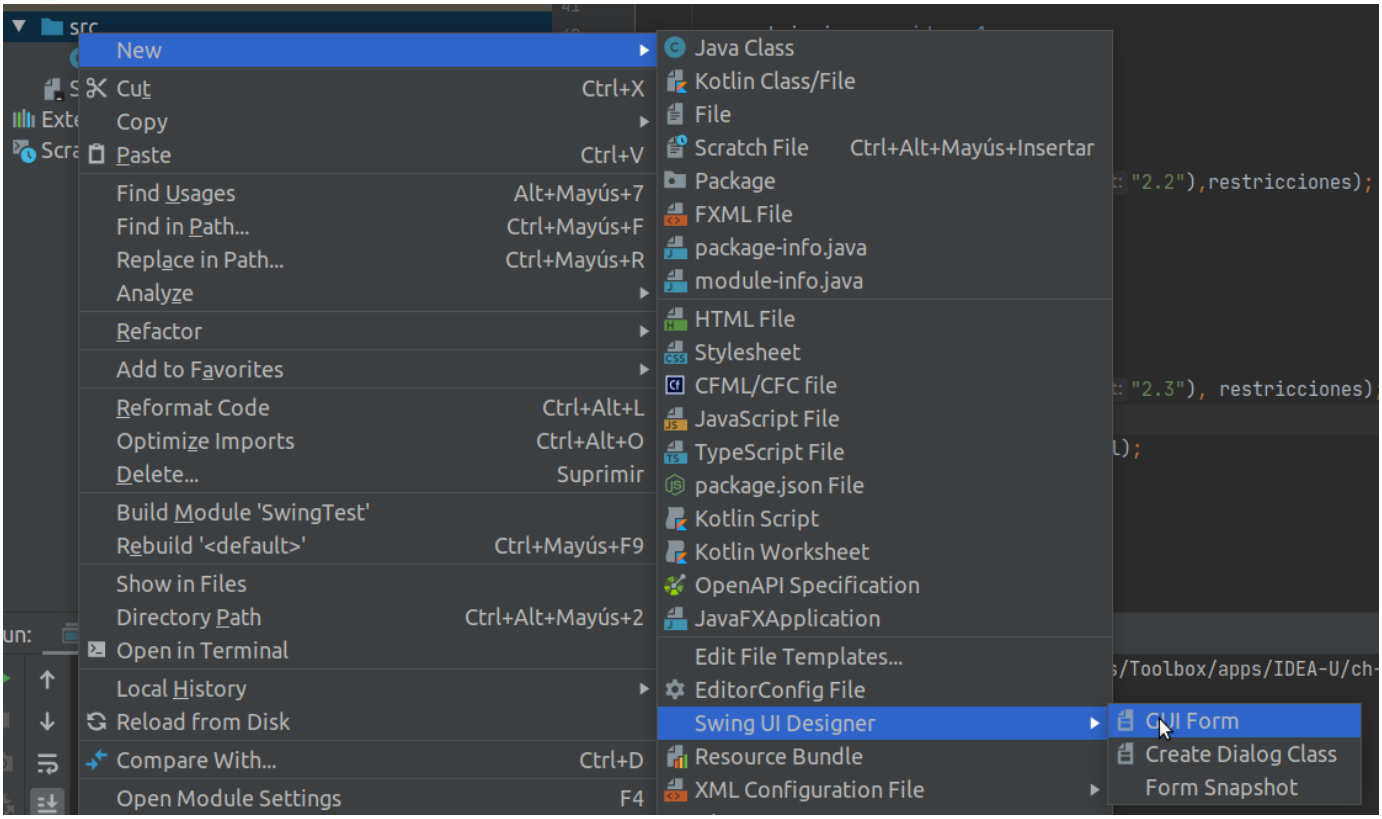
- **Combinar layouts**

Dado que un panel puede incluir otros paneles, se pueden combinar los diferentes tipos de layouts, unos dentro de otros. Por ejemplo:

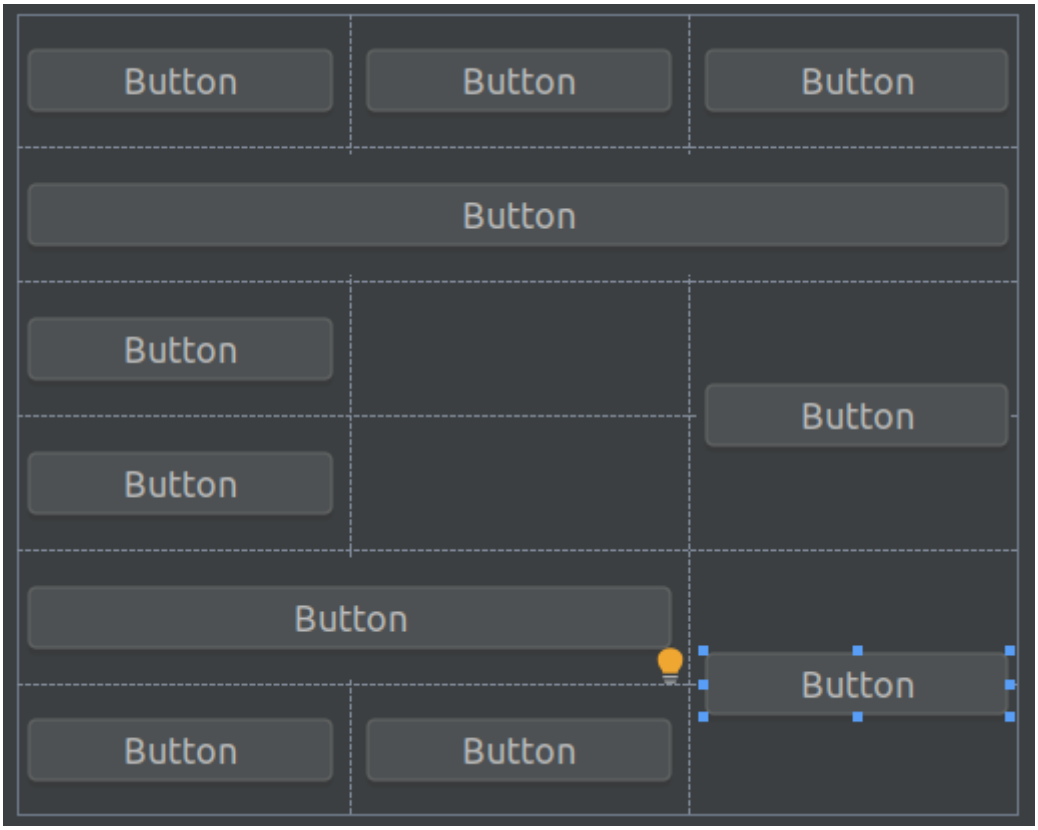
```
        JFrame frame = new JFrame("Mi primera ventana");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);

        JPanel panelPrincipal = new JPanel();
        panelPrincipal.setLayout(new BorderLayout());

        JPanel panelIzquierdo = new JPanel();
        panelIzquierdo.setLayout(new
BoxLayout(panelIzquierdo,BoxLayout.Y_AXIS));
        panelIzquierdo.add(new JButton("izquierda_1"));
        panelIzquierdo.add(new JButton("izquierda_2"));
        panelIzquierdo.add(new JButton("izquierda_3"));


        panelPrincipal.setBackground(Color.red);
        panelPrincipal.add(new JButton("centro"),BorderLayout.CENTER);
        panelPrincipal.add(panelIzquierdo,BorderLayout.WEST);
        panelPrincipal.add(new JButton("derecha"),BorderLayout.EAST);
        panelPrincipal.add(new JButton("arriba"),BorderLayout.NORTH);
        panelPrincipal.add(new JButton("abajo"),BorderLayout.SOUTH);
        frame.setContentPane(panelPrincipal);
        frame.setVisible(true);
```

Creación de un formulario con el plugin de IntelliJ

IntelliJ, así como la mayoría de IDEs, permite diseñar visualmente un formulario. Una vez creado el proyecto, elegimos la opción *New\Swing UI Designer\GUI Form*.



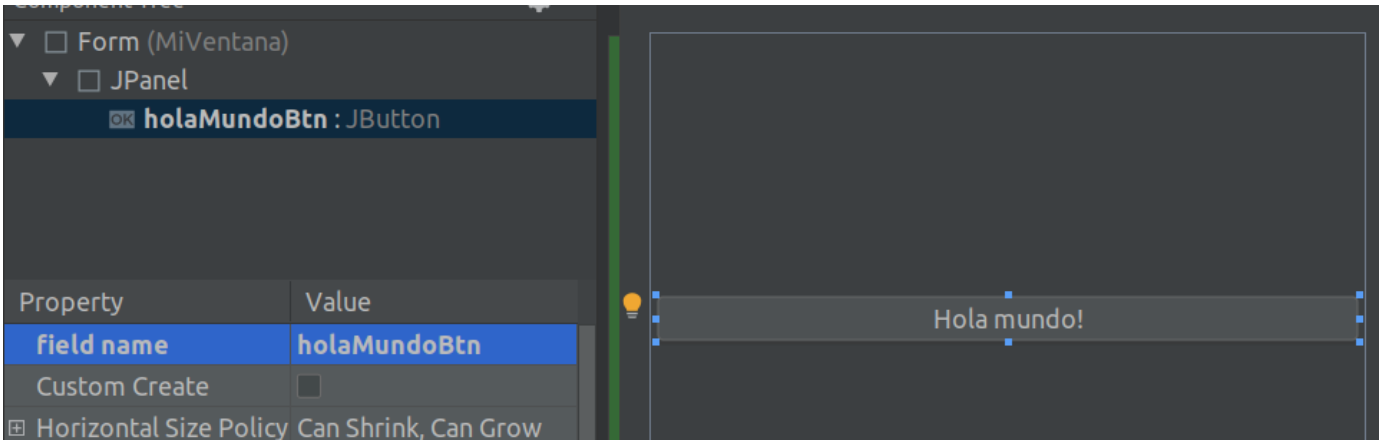
Después asignamos un nombre y podemos empezar a añadir componentes. Por defecto, el layout empleado por IntelliJ es *GridLayoutManager*. Mediante este plugin podemos conseguir diseños mucho más complejos con menos esfuerzo.



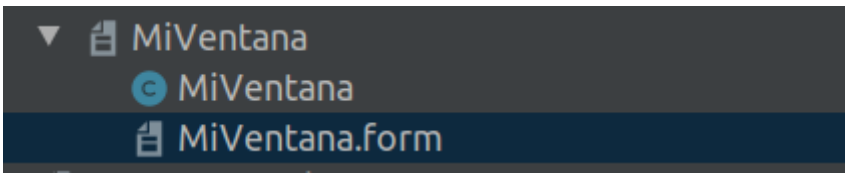
Agregar comportamiento a un componente.

Cuando creamos un formulario, se crea una clase asociada al formulario, donde agregaremos el comportamiento. En dicha clase, se declaran todos los componentes insertados en el formulario.

Supongamos que contamos con un formulario sencillo como el siguiente:



Dentro de la clase asociada al formulario, se declara el componente *holaMundoBtn*




```
public class MiVentana {
    private JButton holaMundoBtn;
}
```

Queremos que al pulsar el botón "Hola mundo!" se muestre por consola el texto "Hola mundo!". Para ello, debemos definir el comportamiento del botón en el constructor de la clase, del siguiente modo:

```
public class MiVentana {
    private JButton holaMundoBtn;

    public MiVentana(){
        holaMundoBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                // Acciones aquí
            }
        });
    }
}
```

En nuestro caso, simplemente escribiremos por consola "Hola mundo!" con un simple *System.out.println*

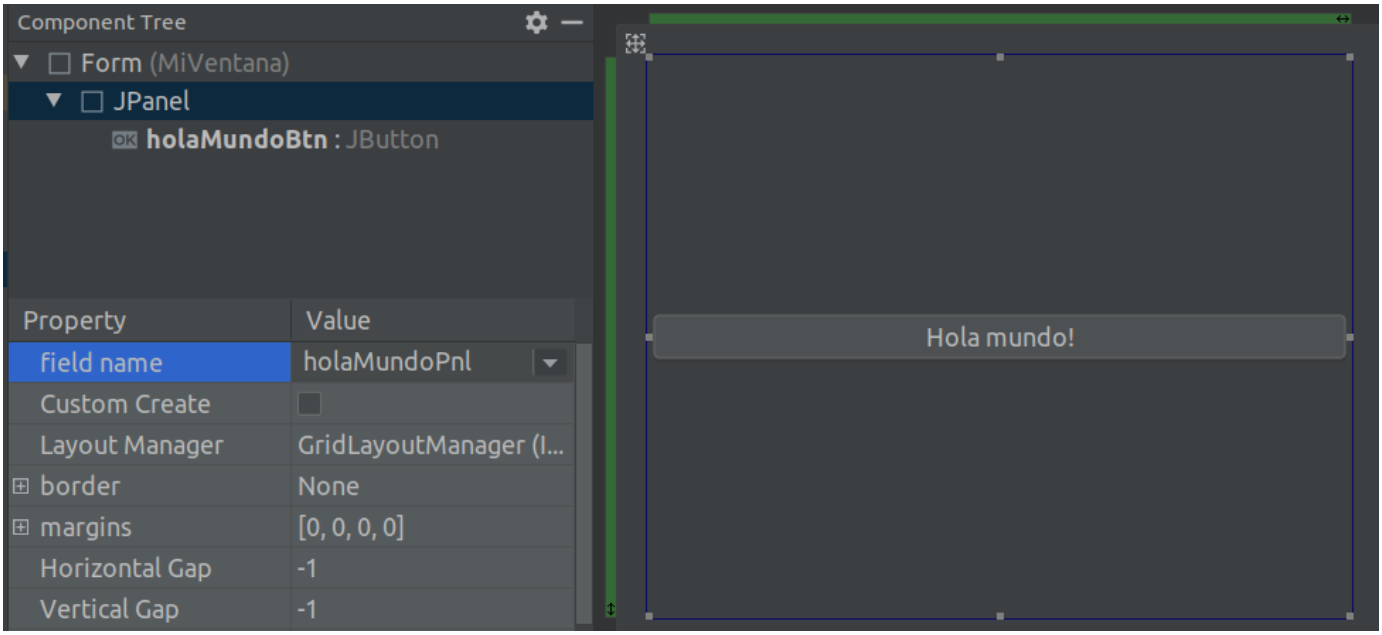
Mostrar una ventana diseñada con el UI Designer

Para poder ejectuar nuestra aplicación, necesitamos hacer lo mismo que hicimos anteriormente (cuando creamos la ventana manualmente) en el método *main*:

```
        JFrame frame = new JFrame("Hola mundo!");
        frame.setContentPane(new MiVentana().getPanel()); // ATENCIÓN
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
```

La linea con el comentario *ATENCIÓN* requiere explicación. El marco necesita un panel donde estén los componentes. Por ello, es preciso hacer dos cosas:

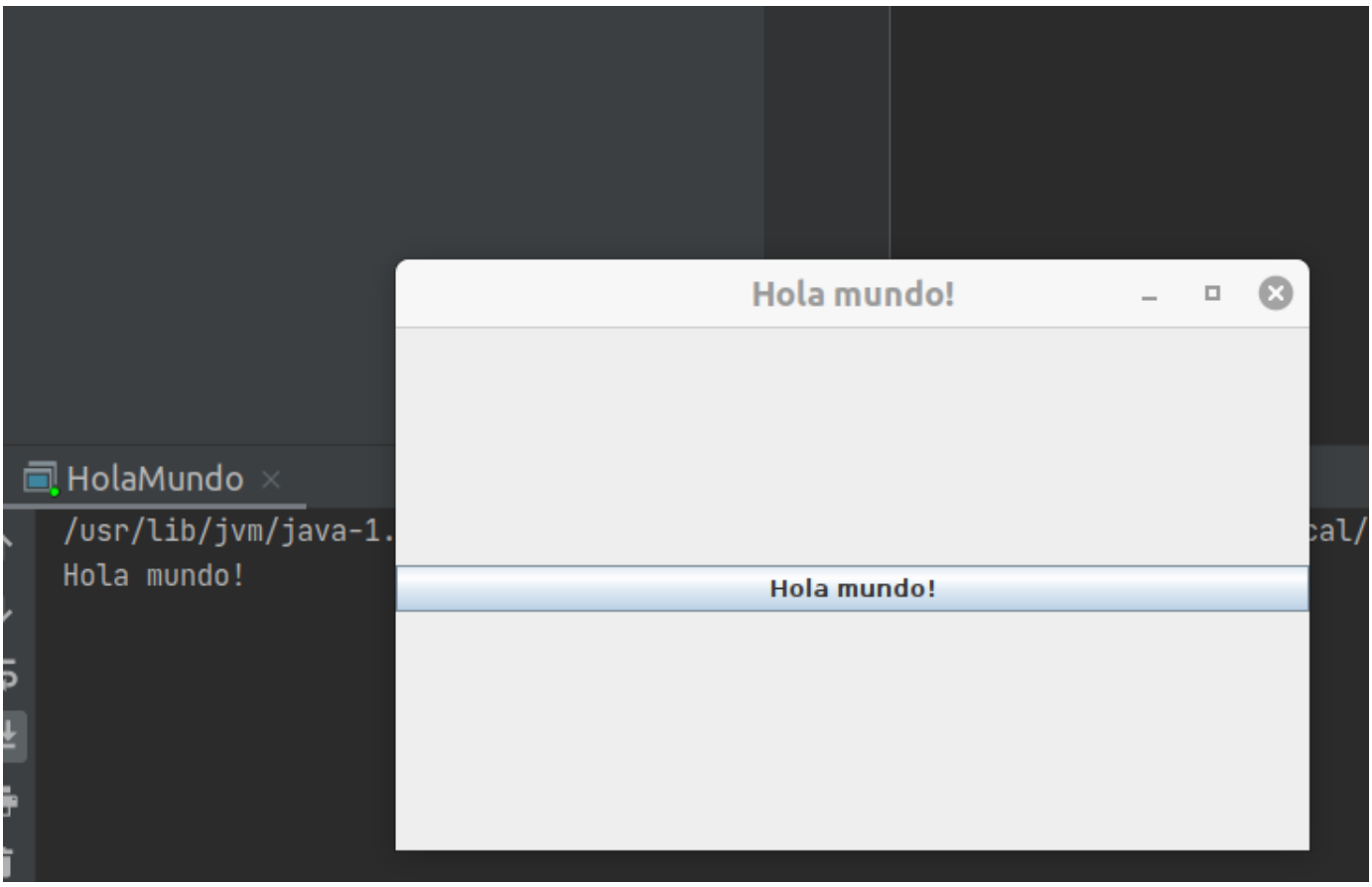
1. Dar un nombre al componente *JPanel* en el UI Designer.



2. Crear un getter en la clase del formulario, que lo devuelva.

```java
public class MiVentana {
    private JButton holaMundoBtn;
    private JPanel holaMundoPnl;

    public MiVentana(){
        holaMundoBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                System.out.println("Hola mundo!");
            }
        });
    }

    public JPanel getPanel(){
        return holaMundoPnl;
    }
}
```

El resultado de todos estos elementos juntos es el siguiente:



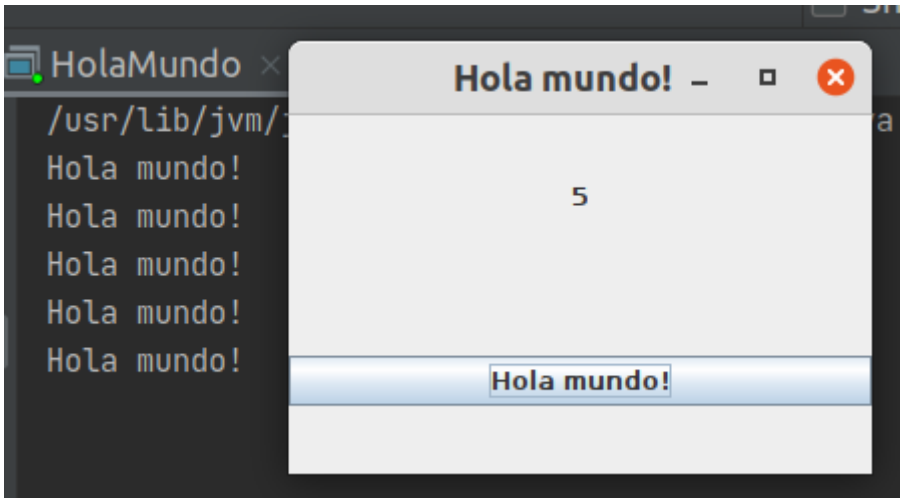
> NOTA: Se pueden añadir muchas otras configuraciones a un marco para que tenga un tamaño determinado y para se ubice en un cierto lugar. Por ejemplo, las siguientes líneas definen un tamaño específico para el marco, y lo ubican en el centro de la pantalla:
>
> ```
> ...
> Toolkit toolkit = Toolkit.getDefaultToolkit();
> int height = toolkit.getScreenSize().height - 200;
> int width = toolkit.getScreenSize().width - 400;
> frame.setPreferredSize(new Dimension(width, height));
> frame.pack();
> frame.setLocationRelativeTo(null);
> ...
> ```
>
> Si tenemos necesidades de ese tipo, basta con rebuscar un poco en la documentación.

Componentes actuando sobre otros componentes

Cada componente tiene sus propios métodos para modificar su estado. Desde un componente podemos llamar a los métodos de otro componente. Por ejemplo, supongamos que cada vez que pulse el botón "holaMundoBtn", el valor de una etiqueta (*JLabel*) se incrementa. Entonces necesitaremos utilizar el método *setText* de *JLabel*. La clase del formulario queda como sigue:

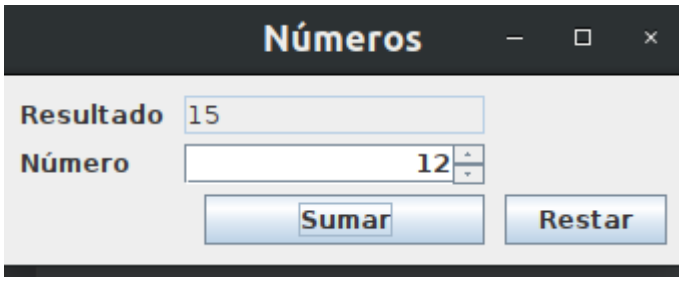
```
public class MiVentana {
    private JButton holaMundoBtn;
    private JPanel holaMundoPnl;
```

```
    private JLabel numClicsLbl;
    private int numClics;

    public MiVentana(){
        numClics = 0;
        numClicsLbl.setText(String.valueOf(numClics));

        holaMundoBtn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                System.out.println("Hola mundo!");
                numClics++;
                numClicsLbl.setText(String.valueOf(numClics));
            }
        });
    }

    public JPanel getPanel(){
        return holaMundoPnl;
    }
}
```



Actividad para practicar

Crea una interfaz Swing con la siguiente interfaz:



Al introducir un número, debe mostrarse el resultado de la operación solicitada.

> Nota: el campo asociado a la etiqueta *Número* es de tipo *JSpinner*.