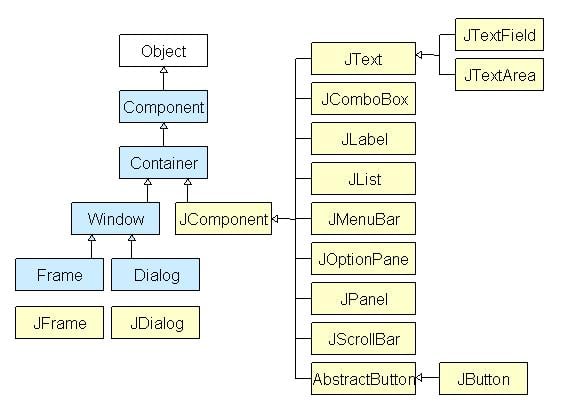
UD10 - Apuntes – Interfaces gráficas en Java

## Swing

Swing en Java es un kit de herramientas de interfaz gráfica de usuario (GUI) que incluye los componentes de la GUI. Swing proporciona un amplio conjunto de widgets y paquetes para crear componentes GUI sofisticados para aplicaciones Java. Swing es parte de Java Foundation Classes (JFC), que es una API para la programación de GUI de Java que proporciona GUI.

El Java Swing La biblioteca está construida sobre el kit de herramientas Java Abstract Widget (AWT), un conjunto de herramientas GUI más antiguo y dependiente de la plataforma. Puede utilizar los componentes de programación GUI simples de Java, como botones, textbox, etc. de la biblioteca y no es necesario crear los componentes desde cero.

## java swing Diagrama de jerarquía de clases



Todos los componentes en Java Swing son JComponent que se pueden agregar a clases de contenedor.

## ¿Qué es una clase de contenedor?

Las clases contenedoras son clases que pueden contener otros componentes. Entonces, para crear un Java Swing GUI, necesitamos al menos un objeto contenedor. Hay 3 tipos de Java Swing contenedores

* Panel: Es un contenedor puro y no es una ventana en sí misma. El único propósito de un Panel es organizar los componentes en una ventana.
* Cuadro: Es una ventana completamente funcional con su título e íconos.
* Diálogo: Se puede considerar como una ventana emergente que aparece cuando se debe mostrar un mensaje. No es una ventana completamente funcional como Frame.

## ¿Qué es la GUI en Java?

GUI (interfaz gráfica de usuario) en Java es un creador de experiencias visuales fácil de usar para aplicaciones Java. Está compuesto principalmente por componentes gráficos como botones, etiquetas, ventanas, etc. a través de los cuales el usuario puede interactuar con una aplicación. La GUI juega un papel importante en la creación de interfaces sencillas para aplicaciones Java.

## Cómo hacer una GUI en Java con ejemplo

Ahora crearemos una GUI en Java con Swing:

**import** javax**.**swing**.\*;**

class Gui**{**

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

JFrame frame **=** **new** JFrame**(**"My First GUI"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,**300**);**

JButton button **=** **new** JButton**(**"Press"**);**

frame**.**getContentPane**().**add**(**button**);**

frame**.**setVisible**(true);**

**}**

**}**

Ahora vamos a añadir un segundo botón:

**import** javax**.**swing**.\*;**

class gui**{**

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

JFrame frame **=** **new** JFrame**(**"My First GUI"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,**300**);**

JButton button1 **=** **new** JButton**(**"Button 1"**);**

JButton button2 **=** **new** JButton**(**"Button 2"**);**

frame**.**getContentPane**().**add**(**button1**);**

frame**.**getContentPane**().**add**(**button2**);**

frame**.**setVisible**(true);**

**}**

**}**

Como podrás ver, los botones se superponen:

Interfaz de usuario gráfica, Aplicación

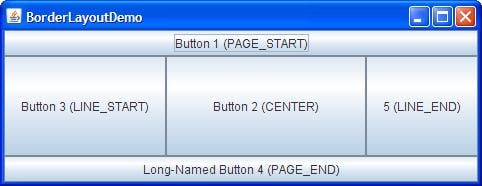
Descripción generada automáticamente

## Administrador de diseño de Java (layout managers)

El administrador de diseño se utiliza para diseñar (u organizar) los componentes Java de la GUI dentro de un contenedor. Hay muchos administradores de diseño, pero los más utilizados son:

### BorderLayout

BorderLayout divide el espacio del contenedor en cinco regiones: norte (top), sur (bottom), este (right), oeste (left) y centro (center). Cuando añades un componente a un contenedor que utiliza BorderLayout, puedes especificar en qué región quieres que se coloque el componente.



Veamos un ejemplo:

**import** javax**.**swing**.\*;**

**import** java**.**awt**.\*;**

public class BorderLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"BorderLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**400**,** 400**);**

// Set the frame layout to BorderLayout

frame**.**setLayout**(new** BorderLayout**());**

// Add components to different regions

frame**.**add**(new** JButton**(**"North"**),** BorderLayout**.**NORTH**);**

frame**.**add**(new** JButton**(**"South"**),** BorderLayout**.**SOUTH**);**

frame**.**add**(new** JButton**(**"East"**),** BorderLayout**.**EAST**);**

frame**.**add**(new** JButton**(**"West"**),** BorderLayout**.**WEST**);**

frame**.**add**(new** JButton**(**"Center"**),** BorderLayout**.**CENTER**);**

frame**.**setVisible**(true);**

**}**

**}**

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

### FlowLayout

FlowLayout es otro administrador de diseño proporcionado por Java Swing. A diferencia de BorderLayout, que divide el espacio del contenedor en cinco regiones, FlowLayout coloca los componentes en una dirección, de izquierda a derecha y de arriba a abajo, similar a cómo se fluyen las palabras en un párrafo de texto.

Cuando se llena una fila con componentes, FlowLayout comienza una nueva fila. Si la ventana se redimensiona, FlowLayout recalcula la posición de los componentes.

Veamos un ejemplo:

**import** javax**.**swing**.\*;**

**import** java**.**awt**.\*;**

public class FlowLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"FlowLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**400**,** 400**);**

// Set the frame layout to FlowLayout

frame**.**setLayout**(new** FlowLayout**());**

// Add components

**for** **(**int i **=** 1**;** i **<=** 5**;** i**++)** **{**

frame**.**add**(new** JButton**(**"Button " **+** i**));**

**}**

frame**.**setVisible**(true);**

**}**

**}**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente (imagen recortada)

### GridLayout

GridLayout es un administrador de diseño en Java que coloca los componentes en una cuadrícula rectangular de celdas, todas de igual tamaño. Cada componente se coloca en una celda de la cuadrícula de izquierda a derecha y de arriba a abajo.

**import** javax**.**swing**.\*;**

**import** java**.**awt**.\*;**

public class GridLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"GridLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,** 300**);**

// Set the frame layout to GridLayout

frame**.**setLayout**(new** GridLayout**(**3**,** 2**));** // 3 rows, 2 columns

// Add components

**for** **(**int i **=** 1**;** i **<=** 6**;** i**++)** **{**

frame**.**add**(new** JButton**(**"Button " **+** i**));**

**}**

frame**.**setVisible**(true);**

**}**

**}**

Tabla

Descripción generada automáticamente

### BoxLayout:

BoxLayout es un administrador de diseño en Java Swing que coloca los componentes en una sola fila o columna. Los componentes pueden tener diferentes tamaños y se pueden alinear vertical u horizontalmente.

**import** javax**.**swing**.\*;**

public class BoxLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"BoxLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,** 300**);**

// Create a panel to hold the buttons

JPanel panel **=** **new** JPanel**();**

// Set the panel layout to BoxLayout

panel**.**setLayout**(new** BoxLayout**(**panel**,** BoxLayout**.**Y\_AXIS**));**

// Add components

**for** **(**int i **=** 1**;** i **<=** 6**;** i**++)** **{**

panel**.**add**(new** JButton**(**"Button " **+** i**));**

**}**

// Add the panel to the frame

frame**.**add**(**panel**);**

frame**.**setVisible**(true);**

**}**

**}**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

### CardLayout:

CardLayout es un administrador de diseño en Java Swing que permite que múltiples componentes ocupen el mismo espacio en la interfaz de usuario. Solo un componente (o "tarjeta") es visible a la vez, lo que hace que CardLayout sea útil para implementar interfaces de usuario en las que se debe mostrar y ocultar diferentes paneles en función de las acciones del usuario.

**import** javax**.**swing**.\*;**

**import** java**.**awt**.\*;**

**import** java**.**awt**.**event**.**ActionEvent**;**

**import** java**.**awt**.**event**.**ActionListener**;**

public class CardLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"CardLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,** 300**);**

// Create a CardLayout

CardLayout cardLayout **=** **new** CardLayout**();**

JPanel panel **=** **new** JPanel**(**cardLayout**);**

// Add components

**for** **(**int i **=** 1**;** i **<=** 3**;** i**++)** **{**

panel**.**add**(new** JLabel**(**"Card " **+** i**),** "Card" **+** i**);**

**}**

// Add a button to switch between cards

JButton button **=** **new** JButton**(**"Next card"**);**

button**.**addActionListener**(new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

cardLayout**.**next**(**panel**);**

**}**

**});**

frame**.**add**(**panel**,** BorderLayout**.**CENTER**);**

frame**.**add**(**button**,** BorderLayout**.**SOUTH**);**

frame**.**setVisible**(true);**

**}**

**}**

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

### GridBagLayout

GridBagLayout es el administrador de diseño más flexible y complejo en Java Swing. Permite especificar la ubicación y el tamaño de los componentes en términos de una cuadrícula, pero las celdas de la cuadrícula pueden tener diferentes tamaños, y un componente puede ocupar más de una celda.

**import** javax**.**swing**.\*;**

**import** java**.**awt**.\*;**

public class GridBagLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"GridBagLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**400**,** 400**);**

// Set the frame layout to GridBagLayout

frame**.**setLayout**(new** GridBagLayout**());**

GridBagConstraints constraints **=** **new** GridBagConstraints**();**

// Add components

**for** **(**int i **=** 0**;** i **<** 3**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** 3**;** j**++)** **{**

constraints**.**gridx **=** i**;**

constraints**.**gridy **=** j**;**

frame**.**add**(new** JButton**(**"Button " **+** **(**i **\*** 3 **+** j **+** 1**)),** constraints**);**

**}**

**}**

frame**.**setVisible**(true);**

**}**

**}**

Tabla

Descripción generada automáticamente

### SpringLayout:

SpringLayout es un administrador de diseño flexible en Java Swing que permite especificar las relaciones (o "restricciones") entre los bordes de los componentes, lo que permite un control muy preciso sobre la ubicación y el tamaño de los componentes.

**import** javax**.**swing**.\*;**

**import** java**.**awt**.\*;**

public class SpringLayoutExample **{**

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

JFrame frame **=** **new** JFrame**(**"SpringLayout Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,** 300**);**

// Create a SpringLayout

SpringLayout layout **=** **new** SpringLayout**();**

JPanel panel **=** **new** JPanel**(**layout**);**

// Add components

JButton button1 **=** **new** JButton**(**"Button 1"**);**

JButton button2 **=** **new** JButton**(**"Button 2"**);**

panel**.**add**(**button1**);**

panel**.**add**(**button2**);**

// Set constraints

layout**.**putConstraint**(**SpringLayout**.**WEST**,** button1**,** 5**,** SpringLayout**.**WEST**,** panel**);**

layout**.**putConstraint**(**SpringLayout**.**NORTH**,** button1**,** 5**,** SpringLayout**.**NORTH**,** panel**);**

layout**.**putConstraint**(**SpringLayout**.**WEST**,** button2**,** 5**,** SpringLayout**.**EAST**,** button1**);**

layout**.**putConstraint**(**SpringLayout**.**NORTH**,** button2**,** 0**,** SpringLayout**.**NORTH**,** button1**);**

frame**.**add**(**panel**);**

frame**.**setVisible**(true);**

**}**

**}**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

# Gestión de eventos:

La gestión de eventos es un concepto fundamental en la programación de interfaces de usuario. Un evento es una acción que ocurre como resultado de la interacción del usuario con la interfaz de usuario, como hacer clic en un botón, mover el ratón, presionar una tecla, etc.

En Java Swing, la gestión de eventos se realiza mediante el uso de interfaces de escucha de eventos y métodos de manejo de eventos. Una interfaz de escucha de eventos es una interfaz que define uno o más métodos de manejo de eventos. Un método de manejo de eventos es un método que se invoca cuando ocurre un evento específico.

Veamos un ejemplo:

**import** javax**.**swing**.\*;**

**import** java**.**awt**.**event**.**ActionEvent**;**

**import** java**.**awt**.**event**.**ActionListener**;**

public class ButtonClickExample **{**

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

JFrame frame **=** **new** JFrame**(**"Button Click Example"**);**

frame**.**setDefaultCloseOperation**(**JFrame**.**EXIT\_ON\_CLOSE**);**

frame**.**setSize**(**300**,** 300**);**

// Create a button

JButton button **=** **new** JButton**(**"Click me"**);**

// Add an action listener to the button

button**.**addActionListener**(new** ActionListener**()** **{**

@Override

public void actionPerformed**(**ActionEvent e**)** **{**

System**.**out**.**println**(**"Button clicked!"**);**

**}**

**});**

frame**.**add**(**button**);**

frame**.**setVisible**(true);**

**}**

**}**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

<https://docs.oracle.com/javase/tutorial/uiswing///lookandfeel/plaf.html>

<https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html>