# **Primeros pasos en Swing**

## 

## 

Hecho por Marco Batista Calado

## 

## 

## 

## 

[**Primeros pasos en Swing 1**](#_n1gkib1ajl2w)

[**Introducción 3**](#_nywb0dz0cwhw)

[**Posicionar una ventana en el centro de cualquier pantalla 4**](#_qbr86xdirien)

[Clases y métodos utilizados 4](#_bzm8c8wodywp)

[Código básico para centrar una ventana 4](#_oz4rcwe0m0a1)

[**Láminas con JPanel 5**](#_f4kbecop8w6r)

[Clase JPanel 5](#_ozx7q117s98l)

[Pasos generales para dibujar en un JPanel 5](#_ma5g6mw6l9je)

[Estructura básica para dibujar en un JPanel 5](#_yu8e0ksj8v0r)

[**Dibujos con Graphics2D 6**](#_u5edjlnhlrx0)

[Clases clave en Graphics2D 6](#_n1v7j8oshozm)

[Método draw(Shape s) 6](#_req6oxh3nvp7)

[**Colores en Graphics2D y JPanel 7**](#_tygfim12spn6)

[Manejo de Colores en Graphics2D 7](#_x6hlotgl6hwt)

[Métodos de color en JPanel 7](#_tmkz1hf4ul)

[**Fuentes en Graphics2D y Font 8**](#_asleaj4p5f9)

[Método setFont() en Graphics2D 8](#_izyewnnzdjn5)

[Clase Font 8](#_o29l8zshyb8x)

[Estilos de fuente 8](#_ign3utkz96a7)

[**Imágenes en Java 9**](#_84cmlnqcgwqk)

[Paquete java.awt.Image 9](#_kwh8o3qpq0mk)

[Paquete javax.imageio y java.io 9](#_989sfqb9yrpe)

[Paquete java.awt.Graphics 9](#_oypg4zg5hpn)

[**Eventos, Fuentes y Oyentes en Java Swing 10**](#_xvbj78hsecx9)

[Definición de Eventos 10](#_2fxowng1upmd)

[1. ¿Qué desencadena la acción? → Objeto Evento 10](#_6ujc9fdh97cs)

[2. ¿Quién desencadena la acción? → Objeto Fuente 10](#_yffpsbf7nimd)

[3. ¿Quién recibe la acción? → Objeto Listener (Oyente) 10](#_sw4ghj99k0fr)

[**Eventos de Ventana en Java 11**](#_ge432haw33q)

[Métodos de WindowListener 11](#_pb1ywrq3b1nf)

[Adapter Classes 11](#_oww3luhb8xfl)

[**Controlando el Estado de la Ventana 12**](#_vu7wn797n4f8)

[Interfaz WindowStateListener y la clase WindowEvent 12](#_ww1kqsxrebip)

[**Eventos de Teclado 12**](#_ypisoj9wjii)

[**Eventos de Ratón 13**](#_q9fjjvkuqwdw)

[**Eventos de Foco 14**](#_4vtfmy86cuke)

[Eventos de Foco en Ventanas: 14](#_i36yoz8djuia)

[**Eventos: Múltiples Fuentes 14**](#_esnrgq7cqawn)

[Asignando Acciones al Teclado 15](#_asrxny36re8h)

[Múltiples Oyentes 15](#_mbcfgych9lqn)

[**Layouts/Disposiciones 16**](#_u7r8bi93x2om)

[**Componentes Swing 17**](#_5an9d8nxs8xb)

[Cuadros de Texto 17](#_951j2zz6x9s)

[Cambios en un Campo de Texto 18](#_n1b8d8jcpru)

[Áreas de Texto 18](#_aw8f96j5r8qn)

[Checkbox 18](#_n3vyco3jxmmq)

[Radio Button 18](#_w3fw6941mvm9)

[ComboBox 18](#_25s698hptcdl)

[Slider 18](#_cni98hfh78qm)

[Spinner 19](#_49rrcngrhy9r)

[Construcción de Menús 19](#_xjnl4elfpgxr)

## **Introducción**

Tras completar el primer curso del ciclo formativo de grado superior en Desarrollo de Aplicaciones Web y considerando mi formación previa en el ciclo de Administración de Sistemas Informáticos en Red, decidí aprovechar el verano entre 1º y 2º año para adentrarme en el mundo de las interfaces gráficas mediante Swing. Hasta ese momento, no había tenido la oportunidad de trabajar con herramientas para la creación de interfaces gráficas, por lo que mi objetivo era familiarizarme con esta tecnología.

Aunque Swing es una herramienta robusta para desarrollar interfaces gráficas en Java, es conocida por ser bastante más enrevesada y anticuada en comparación con soluciones más modernas y accesibles. Por ejemplo, el desarrollo de interfaces gráficas en C# usando Visual Studio 2022, Python con Tkinter, o JavaScript con Electron tiende a ser mucho más directo y amigable. A pesar de esta complejidad, decidí enfrentar el desafío que representa Swing, ya que creo que adquirir experiencia con esta tecnología puede ser muy valioso, especialmente para el año académico que estoy a punto de comenzar.

Para prepararme de la mejor manera, realicé un curso exhaustivo disponible en [este enlace](https://youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk&si=IDBb7TU5AEAlevhJ), impartido por Juan Díaz en su canal "Píldoras Informáticas". El curso abarca desde el video 55 hasta el 114, habiendo visualizado alrededor de 22 horas de contenido e invertido aproximadamente 15 horas adicionales en la realización de los ejercicios prácticos para consolidar mi aprendizaje.

Los ejercicios realizados durante el curso están disponibles en mi repositorio de GitHub:<https://github.com/marcobatistacalado/PrimerosPasosEnSwing>. Este repositorio incluye todos los ejercicios que fui realizando a lo largo del curso, lo cual refleja mi proceso de aprendizaje y mi evolución en el manejo de Swing.

Este documento de apuntes ha sido elaborado con el propósito de ayudar a quienes deseen iniciarse en el uso de Swing. Incluye información detallada, así como imágenes del curso y la colaboración de ChatGPT para mejorar la redacción y presentación. Espero que este material sea útil y que facilite tanto mi propio crecimiento como el de otros en el campo del desarrollo de interfaces gráficas.

## 

## **Posicionar una ventana en el centro de cualquier pantalla**

Para centrar una ventana de **Swing** en el centro de la pantalla, puedes utilizar el Toolkit, que es una clase que contiene diversos métodos para interactuar con el sistema de ventanas del SO.

### **Clases y métodos utilizados**

* **Toolkit**: Esta clase proporciona la interfaz con los recursos del sistema de ventanas del sistema operativo huésped. Entre sus métodos más útiles están:
  + getDefaultToolkit(): Método estático que devuelve una instancia del Toolkit asociado con el sistema de ventanas actual.
  + getScreenSize(): Devuelve un objeto Dimension que contiene el ancho y alto de la pantalla.

### **Código básico para centrar una ventana**

1. **Obtener el tamaño de la pantalla**: Usamos Toolkit.getDefaultToolkit().getScreenSize() para obtener el tamaño de la pantalla.
2. **Obtener el tamaño de la ventana**: Utilizamos los métodos de la ventana de Swing (como JFrame) para obtener su ancho y alto.
3. **Calcular la posición de la ventana**: Para centrar la ventana, restamos la mitad del tamaño de la ventana al tamaño de la pantalla.
4. **Establecer la posición**: Usamos setLocation(int x, int y) del JFrame para colocar la ventana en las coordenadas calculadas.

## 

## 

## 

## 

## 

## 

## **Láminas con JPanel**

### **Clase JPanel**

**JPanel** es una clase en **Swing** que actúa como un contenedor para organizar otros componentes, pero también se puede usar como un lienzo para dibujar formas, texto, imágenes, etc. Para realizar estos dibujos, es necesario sobrescribir el método paintComponent(Graphics g).

### **Pasos generales para dibujar en un JPanel**

1. **Crear una clase que herede de JPanel**: Para personalizar el comportamiento del panel y poder dibujar sobre él, necesitamos crear una clase que extienda de JPanel.
2. **Sobrescribir el método paintComponent(Graphics g)**: El método paintComponent(Graphics g) es de la clase JComponent, que es la clase superior de JPanel. Este método es donde colocamos el código para realizar cualquier dibujo o escritura sobre el panel.
3. **Utilizar el objeto Graphics**: Dentro del método paintComponent, se recibe un objeto de tipo Graphics, que proporciona métodos para dibujar figuras geométricas, texto e imágenes en el JPanel.

### **Estructura básica para dibujar en un JPanel**

1. **Crear la clase que herede de JPanel**.
2. **Sobrescribir el método paintComponent() para personalizar el panel**.

## 

## **Dibujos con Graphics2D**

La clase **Graphics** es la forma básica de dibujar en Java, pero con el tiempo se ha quedado obsoleta para algunas tareas más avanzadas o precisas. La clase **Graphics2D**, que extiende Graphics, ofrece más herramientas para el desarrollo de gráficos más sofisticados, incluyendo formas geométricas avanzadas, transformaciones, rotaciones, suavizado de bordes, entre otros.

### **Clases clave en Graphics2D**

1. **Graphics2D**: Extiende Graphics y permite utilizar formas geométricas avanzadas, trazos más detallados y efectos gráficos.
2. **Shape** (interfaz): Es implementada por clases que representan diferentes formas geométricas. Entre las clases más comunes que implementan Shape están:
   * **Rectangle2D**: Para dibujar rectángulos.
   * **Ellipse2D**: Para dibujar elipses y círculos.
   * **Line2D**: Para dibujar líneas.

### **Método draw(Shape s)**

El método draw(Shape s) se utiliza para dibujar cualquier objeto que implemente la interfaz Shape. Esto hace que sea fácil y flexible trabajar con formas geométricas.

## 

## 

## 

## 

## 

## **Colores en Graphics2D y JPanel**

### **Manejo de Colores en Graphics2D**

La clase Graphics2D permite definir colores de dibujo a través del método **setPaint()**. Este método define el color o pintura que se utilizará para dibujar formas o rellenar áreas.

* **setPaint(Color c)**: Establece el color que se usará para dibujar figuras geométricas, texto o cualquier otro objeto gráfico.

### **Métodos de color en JPanel**

En la clase JPanel también tenemos métodos para cambiar los colores del fondo y del texto de los componentes que se añaden al panel.

* **setBackground(Color c)**: Cambia el color de fondo del JPanel. Este método no afecta a las figuras que dibujas dentro del panel, sino al color de fondo del propio panel.
* **setForeground(Color c)**: Cambia el color de primer plano, es decir, el color que se usará para el texto o los componentes añadidos al panel, como JLabel o JButton.

## 

## **Fuentes en Graphics2D y Font**

Cuando trabajamos con texto en Java, podemos personalizar las fuentes usando la clase **Font** y establecerla a través del método **setFont()** de **Graphics2D**. Sin embargo, hay que tener en cuenta que no todas las fuentes estarán disponibles en todos los sistemas, ya que cada usuario tiene su propio conjunto de fuentes instaladas.

### **Método setFont() en Graphics2D**

* **setFont(Font f)**: Cambia la fuente actual que se usará para dibujar texto en el componente. Esta fuente afectará cualquier texto que se dibuje después de que el método sea llamado.

### **Clase Font**

La clase Font permite crear un objeto de fuente con un tipo, un estilo y un tamaño.

* **Font(String name, int style, int size)**:
  + **name**: Nombre de la fuente (por ejemplo, "Arial", "Times New Roman").
  + **style**: Estilo de la fuente, que puede ser Font.PLAIN, Font.BOLD, Font.ITALIC o combinaciones de estos.
  + **size**: Tamaño de la fuente.

### **Estilos de fuente**

* **Font.PLAIN**: Fuente normal, sin negrita ni cursiva.
* **Font.BOLD**: Fuente en negrita.
* **Font.ITALIC**: Fuente en cursiva.
* **Font.BOLD + Font.ITALIC**: Combinación de negrita y cursiva.

## 

## **Imágenes en Java**

### **Paquete java.awt.Image**

La clase **Image** es parte del paquete java.awt y representa las imágenes en Java. Existen varios métodos importantes que puedes usar con esta clase, como obtener las dimensiones de la imagen.

* **getWidth()**: Devuelve el ancho de la imagen.
* **getHeight()**: Devuelve la altura de la imagen.

### **Paquete javax.imageio y java.io**

Para cargar imágenes desde archivos o URLs, se utiliza la clase **ImageIO**, que pertenece al paquete javax.imageio, y la clase File del paquete java.io. El método **read()** de ImageIO permite leer imágenes desde distintas fuentes.

* **ImageIO.read(File f)**: Carga una imagen desde un archivo local.
* **ImageIO.read(URL url)**: Carga una imagen desde una URL.

### **Paquete java.awt.Graphics**

Una vez que tienes la imagen, puedes dibujarla en un JPanel o cualquier otro componente gráfico utilizando la clase **Graphics** y su método **drawImage()**. También puedes copiar partes de la pantalla o de la imagen usando **copyArea()**.

* **drawImage(Image img, int x, int y, ImageObserver observer)**: Dibuja la imagen en el componente en la posición (x, y).
* **copyArea(int x, int y, int width, int height, int dx, int dy)**: Copia un área del componente gráfico desde una región específica y la desplaza a otra posición.

## 

## **Eventos, Fuentes y Oyentes en Java Swing**

### **Definición de Eventos**

Un **evento** en Java es un desencadenante de una acción que puede ocurrir cuando interactuamos con la interfaz gráfica, como hacer clic con el ratón, mover una ventana o presionar una tecla. Cada vez que ocurre un evento, se genera un **objeto evento** que encapsula la información sobre ese suceso.

Existen **tres factores clave** para entender cómo funcionan los eventos:

1. **¿Qué desencadena la acción?** → **Objeto Evento**
2. **¿Quién desencadena la acción?** → **Objeto Fuente**
3. **¿Quién recibe la acción?** → **Objeto Listener** (Oyente)

### **1. ¿Qué desencadena la acción? → Objeto Evento**

El **objeto evento** es el que contiene la información sobre lo que ha ocurrido, como qué botón del ratón se ha presionado o qué tecla se ha pulsado. En Java, todos los eventos heredan de la clase **EventObject**.

Algunos tipos comunes de eventos son:

* **ActionEvent**: Generado por acciones como hacer clic en un botón. Está en el paquete java.awt.event.
* **MouseEvent**: Generado por la interacción con el ratón.
* **WindowEvent**: Generado por eventos relacionados con ventanas (abrir, cerrar, minimizar, etc.). También está en java.awt.event.

### **2. ¿Quién desencadena la acción? → Objeto Fuente**

La **fuente** es el componente que origina el evento. Por ejemplo, si haces clic en un botón, ese botón es la fuente del evento. En Java, cualquier componente visual, como un botón (JButton), puede ser una fuente de eventos.

### **3. ¿Quién recibe la acción? → Objeto Listener (Oyente)**

El **oyente** o **listener** es el objeto que escucha los eventos y responde a ellos. Los oyentes implementan interfaces específicas que contienen los métodos necesarios para manejar los diferentes tipos de eventos. Algunos ejemplos de oyentes son:

* **ActionListener**: Para escuchar eventos de acciones como los clics en botones.
* **MouseListener**: Para escuchar eventos relacionados con el ratón.
* **WindowListener**: Para escuchar eventos relacionados con las ventanas.

## 

## **Eventos de Ventana en Java**

Java proporciona la interfaz **WindowListener** para capturar eventos que suceden en las ventanas (como cuando una ventana se cierra, se minimiza, etc.). Al implementar esta interfaz, se deben declarar los **7 métodos** correspondientes a los diferentes estados que puede atravesar una ventana:

### **Métodos de WindowListener**

Al implementar **WindowListener**, se deben manejar estos **7 métodos obligatorios**:

1. **windowActivated(WindowEvent e)**Se llama cuando la ventana gana el foco (es activada).
2. **windowClosed(WindowEvent e)**Se llama cuando la ventana ya ha sido cerrada.
3. **windowClosing(WindowEvent e)**Se llama justo antes de que la ventana sea cerrada, es decir, cuando se hace clic en el botón de cerrar.
4. **windowDeactivated(WindowEvent e)**Se llama cuando la ventana pierde el foco (es desactivada).
5. **windowDeiconified(WindowEvent e)**Se llama cuando la ventana es restaurada después de haber sido minimizada.
6. **windowIconified(WindowEvent e)**Se llama cuando la ventana es minimizada.
7. **windowOpened(WindowEvent e)**Se llama cuando la ventana es abierta.

### **Adapter Classes**

Las adapter classes simplifican la implementación de interfaces de eventos al proporcionar implementaciones vacías de todos sus métodos. Esto permite sobrescribir solo los métodos que se necesitan, mejorando la limpieza y optimización del código.

* **KeyAdapter**: Simplifica KeyListener.
* **WindowAdapter**: Simplifica WindowListener.
* **MouseAdapter**: Simplifica MouseListener.

Otras interfaces relacionadas:

* **WindowFocusListener**
* **WindowStateListener**
* **WindowListener**
* **EventListener**

## **Controlando el Estado de la Ventana**

Para controlar los cambios en el estado de una ventana (minimizada, maximizada, restaurada), se utiliza la interfaz **WindowStateListener**. Este oyente es especialmente útil cuando se necesita detectar cambios en el tamaño o estado de la ventana.

### **Interfaz WindowStateListener y la clase WindowEvent**

El método **windowStateChanged(WindowEvent e)** de WindowStateListener captura los cambios de estado de la ventana. Dentro de este método, puedes utilizar los siguientes métodos de WindowEvent para obtener más detalles:

* **getNewState()**: Devuelve el **nuevo estado** de la ventana.
* **getOldState()**: Devuelve el **estado anterior** de la ventana.

Estos métodos devuelven enteros, que se pueden comparar con constantes predefinidas en la clase Frame:

* **Frame.NORMAL**: La ventana está en su tamaño normal.
* **Frame.ICONIFIED**: La ventana está minimizada.
* **Frame.MAXIMIZED\_BOTH**: La ventana está maximizada en ambas direcciones (horizontal y vertical).

## **Eventos de Teclado**

**KeyListener** (i): Interfaz para manejar eventos de teclado.

* **keyPressed(KeyEvent e)**: Se llama cuando se presiona una tecla.
* **keyReleased(KeyEvent e)**: Se llama cuando se suelta una tecla.
* **keyTyped(KeyEvent e)**: Se llama cuando una tecla ha sido presionada y luego liberada (combinación de eventos).

**Constantes de Teclas**:

* **VK\_0, VK\_S, etc.**: Valores constantes que representan códigos específicos de teclas.

**KeyEvent**:

* **getKeyCode()**: Obtiene el código de la tecla presionada.
* **getKeyChar()**: Obtiene el carácter asociado con la tecla presionada.

**KeyAdapter** (©): Clase adaptadora que permite sobrescribir solo los métodos necesarios de KeyListener.

## **Eventos de Ratón**

**MouseListener**: Interfaz para manejar eventos de ratón.

* **mousePressed(MouseEvent e)**: Se llama cuando se presiona un botón del ratón.
* **mouseReleased(MouseEvent e)**: Se llama cuando se suelta un botón del ratón.
* **mouseClicked(MouseEvent e)**: Se llama cuando se hace clic en un botón del ratón.
* **mouseEntered(MouseEvent e)**: Se llama cuando el cursor del ratón entra en el área de un componente.
* **mouseExited(MouseEvent e)**: Se llama cuando el cursor del ratón sale del área de un componente.

**MouseAdapter**: Clase adaptadora que permite sobrescribir solo los métodos necesarios de MouseListener.

**MouseEvent**: Clase que representa eventos del ratón.

* **Constantes para botones**:
  + **BUTTON1\_DOWN\_MASK**: Botón izquierdo.
  + **BUTTON2\_DOWN\_MASK**: Rueda del ratón.
  + **BUTTON3\_DOWN\_MASK**: Botón derecho.
* **Métodos**:
  + **getX()**: Obtiene la posición horizontal del ratón en el componente.
  + **getY()**: Obtiene la posición vertical del ratón en el componente.
  + **getModifiersEx()**: Obtiene las modificaciones del teclado usadas durante el evento.
  + **getClickCount()**: Obtiene el número de clics realizados por el usuario.

**MouseMotionListener**: Interfaz para manejar eventos de movimiento del ratón.

* **mouseDragged(MouseEvent e)**: Se llama cuando el ratón se arrastra.
* **mouseMoved(MouseEvent e)**: Se llama cuando el ratón se mueve sin arrastrar.

## 

## **Eventos de Foco**

**FocusListener**: Interfaz para manejar eventos de enfoque.

* **focusGained(FocusEvent e)**: Se llama cuando un componente gana el foco.
* **focusLost(FocusEvent e)**: Se llama cuando un componente pierde el foco.

**FocusAdapter**: Clase adaptadora que permite sobrescribir solo los métodos necesarios de FocusListener.

### **Eventos de Foco en Ventanas:**

**WindowFocusListener**: Interfaz para manejar eventos de enfoque de ventanas.

* **windowGainedFocus(FocusEvent e)**: Se llama cuando una ventana gana el foco.
* **windowLostFocus(FocusEvent e)**: Se llama cuando una ventana pierde el foco.

**WindowAdapter**: Clase adaptadora que permite sobrescribir solo los métodos necesarios de WindowFocusListener.

## **Eventos: Múltiples Fuentes**

**Action**: Interfaz para manejar acciones.

* **actionPerformed(ActionEvent evento)**: Método que se llama cuando se realiza una acción.
* **setEnabled(boolean b)**: Activa o desactiva la acción.
* **isEnabled()**: Consulta el estado de la acción.
* **putValue(String clave, Object valor)**: Almacena pares clave-valor.
* **getValue(String clave)**: Obtiene el valor asociado a una clave.
* **addPropertyChangeListener(PropertyChangeListener oyente)**: Agrega un oyente para cambios de propiedades.
* **removePropertyChangeListener(PropertyChangeListener oyente)**: Elimina un oyente de cambios de propiedades.

**AbstractAction**: Clase base para Action que implementa todos los métodos mencionados anteriormente, evitando la necesidad de escribirlos manualmente.

### 

### 

### **Asignando Acciones al Teclado**

**KeyStroke**: Representa una combinación de teclas.

* **getKeyStroke(String s)**: Obtiene un KeyStroke a partir de una cadena de texto.

**InputMap**: Mapa que asocia combinaciones de teclas a acciones.

* **put(KeyStroke, Object)**: Asocia un KeyStroke a un objeto.

**JPanel**: Componente en el que se pueden asignar acciones de teclado.

* **getInputMap(int condición)**: Obtiene el InputMap asociado a una condición específica.

**ActionMap**: Mapa que asocia objetos a acciones.

* **put(Object, acción)**: Asocia un objeto a una acción.

**Pasos para asignar acciones a combinaciones de teclas**:

1. Crear el InputMap para el componente al que se aplicará la entrada.
2. Crear la combinación de teclas usando KeyStroke.
3. Asignar la combinación de teclas al InputMap.
4. Asociar el objeto del InputMap con una acción en el ActionMap.

### **Múltiples Oyentes**

Hasta ahora, hemos visto cómo manejar eventos utilizando varias interfaces y clases adaptadoras para simplificar la implementación de oyentes y acciones en Java.



## **Layouts/Disposiciones**

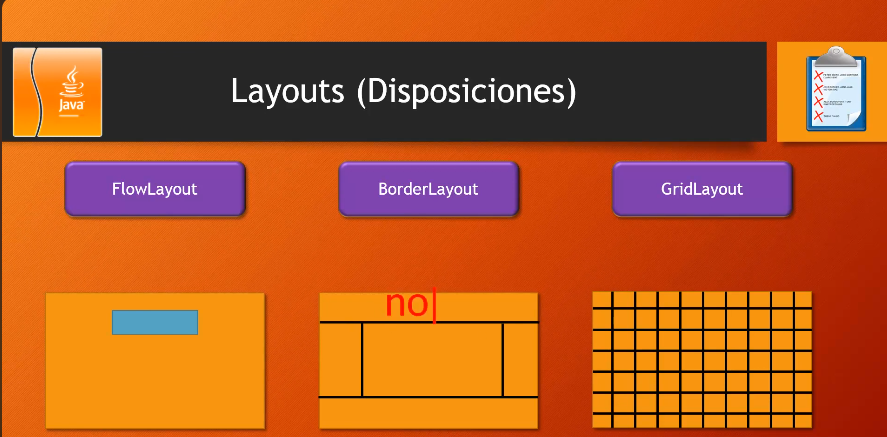
**FlowLayout**: Disposición que coloca los componentes en una línea, de izquierda a derecha, ajustando automáticamente las filas según el espacio disponible. Por defecto, los componentes se centran en la línea.

**BorderLayout**: Disposición que divide el contenedor en cinco áreas: Norte, Sur, Este, Oeste y Centro. Cada componente se coloca en una de estas áreas.

**GridLayout**: Disposición que organiza los componentes en una rejilla de celdas de tamaño fijo. Puedes especificar el número de filas y columnas.

**Container**: Clase base para todos los contenedores en Swing.

* **setLayout(LayoutManager mgr)**: Método para establecer el layout del contenedor.

******

## 

## **Componentes Swing**

### **Cuadros de Texto**

* **JTextComponent**: Clase base para componentes de texto.
  + **setText(String t)**: Establece el texto del componente.
  + **getText()**: Obtiene el texto del componente.
* **JTextField**: Cuadro de texto de una sola línea.
* **JTextArea**: Área de texto para múltiples líneas.
  + **setLineWrap(boolean b)**: Activa o desactiva el ajuste de línea.
  + **getLineWrap()**: Obtiene el estado del ajuste de línea.
* **JLabel**: Muestra texto o imágenes, útil para etiquetas.

### **Cambios en un Campo de Texto**

* **getDocument()**: Obtiene el Document asociado con el campo de texto.
  + **addDocumentListener(DocumentListener listener)**: Agrega un oyente para cambios en el documento.
  + **removeUpdate(DocumentEvent e)**: Llamado cuando se elimina texto.
  + **insertUpdate(DocumentEvent e)**: Llamado cuando se inserta texto.
  + **changedUpdate(DocumentEvent e)**: Llamado cuando el texto cambia (para formato de documento).

### **Áreas de Texto**

* **JTextArea**:
  + **getText()**: Obtiene el texto del área de texto.
  + **setLineWrap(boolean b)**: Activa o desactiva el ajuste de línea.
  + **getLineWrap()**: Obtiene el estado del ajuste de línea.

### **Checkbox**

* **JCheckBox**:
  + **isSelected()**: Determina si el checkbox está seleccionado.
  + **setSelected(boolean b)**: Establece si el checkbox está seleccionado.

### **Radio Button**

* **JRadioButton**: Crea un botón de radio.
* **ButtonGroup**: Agrupa botones de radio para que solo uno esté seleccionado a la vez.

### **ComboBox**

* **JComboBox**:
  + **addItem(Object item)**: Agrega un elemento al combobox.
  + **getSelectedItem()**: Obtiene el elemento seleccionado.

### **Slider**

* **JSlider**:
  + **setPaintTicks(boolean b)**: Activa o desactiva la visualización de marcas.
  + **setMajorTickSpacing(int spacing)**: Establece el espaciado de las marcas mayores.
  + **setMinorTickSpacing(int spacing)**: Establece el espaciado de las marcas menores.
  + **setPaintLabels(boolean b)**: Activa o desactiva la visualización de etiquetas.

### **Spinner**

* **JSpinner**:
  + **JSpinner()**: Constructor predeterminado.
  + **JSpinner(SpinnerModel model)**: Constructor con un modelo de spinner.

### **Construcción de Menús**

* **JMenuBar**: Menú principal de la ventana.
* **JMenu**: Elemento del menú principal.
* **JMenuItem**: Ítem dentro de un menú.
  + **JMenuItem(String texto)**: Crea un ítem con texto.
  + **JMenuItem(String texto, Icon icono)**: Crea un ítem con texto e ícono.
* **JCheckBoxMenuItem**: Ítem de menú con checkbox.
* **JRadioButtonMenuItem**: Ítem de menú con botón de radio.
* **setAccelerator(KeyStroke keyStroke)**: Establece un atajo de teclado para el ítem de menú. Ejemplo: KeyStroke.getKeyStroke(KeyEvent.VK\_D, InputEvent.CTRL\_DOWN\_MASK).
* **addSeparator()** (m): Agrega un separador en el menú.
* **JPopupMenu**: Menú emergente.
  + **setComponentPopupMenu(Component c)**: Asocia un menú emergente con un componente.

**Índice de Ejercicios Prácticos**

A continuación se presenta un listado de los ejercicios prácticos que se realizan a lo largo del curso de Swing.

**PAQUETE: graficos**

1. CreandoMarcos

2. CreandoMarcoCentrado

3. EscribiendoEnMarco

4. PruebaDibujo

5. TrabajandoColores

6. TrabajandoConFuentes

6.1 ComprobarFuentesDelSistema

7. PruebaImagenes

***EVENTOS***

8. PruebaEventos

9. PruebaEventos2

10. PruebaEventosVentanas: WindowListener (7 métodos)

11. PruebaEventosVentanas2: AdapterClasses(WindowAdapter)

12. CambioEstado (Control de ventanas: windowEvent)

13. EventosTeclado

14. EventosRaton

15. EventosFoco

16. EventosFocoVentana

17. EventosFuentes\_Acciones

18. EventosVariosOyentes

***LAYAOUTS***

19. Layaouts

20. Layaout\_Grid\_Calculadora

***COMPONENTES SWING***

21. CuadroTexto

22. EventosCuadroTexto

23. EventosCuadroTexto\_password

24. AreaTexto

25. CheckBoxEjemplo

26. RadioButtonEjemplo

27. RadioButtonEjemplo2

28. ComboBoxEjemplo

29. SlideEjemplo

30. SlideEjemplo2

31. MenuCreacion

33. MarcoMenuConImagen

34. MarcoMenuConCheckBox\_RadioButton

35. MarcoMEmergente

**PAQUETE: practicaProcesadorTexto**

32. Procesador