Guía Completa sobre Java Swing

Java Swing es parte de las Java Foundation Classes (JFC) y proporciona un conjunto de componentes gráficos "ligeros" (escritos completamente en Java) para construir interfaces de usuario de escritorio. A diferencia de su predecesor, AWT (Abstract Window Toolkit), los componentes Swing no dependen directamente de los componentes nativos del sistema operativo, lo que les da una mayor portabilidad y flexibilidad en su apariencia.

1. Conceptos Fundamentales

Componentes (Components)

Son los bloques de construcción visuales de la interfaz. Cada componente es una subclase de javax.swing.JComponent (que a su vez hereda de java.awt.Component). Algunos de los más comunes son:

- JLabel: Muestra texto o imágenes no editables.
- JButton: Un botón estándar que el usuario puede pulsar.
- JTextField: Un campo para introducir una sola línea de texto.
- JTextArea: Un área para introducir múltiples líneas de texto.
- JCheckBox: Una casilla de verificación (seleccionada/no seleccionada).
- JRadioButton: Un botón de opción (normalmente usado en grupos donde solo uno puede estar seleccionado).
- JComboBox: Una lista desplegable.
- JList: Una lista que muestra varios ítems para seleccionar uno o más.
- JTable: Muestra datos en formato de tabla (filas y columnas).
- JScrollPane: Proporciona barras de desplazamiento para componentes grandes (como JTextArea o JTable).
- JPanel: Un contenedor genérico usado para agrupar otros componentes. Es fundamental para organizar la interfaz.

Contenedores (Containers)

Son componentes especiales diseñados para contener y organizar otros componentes. Los principales son:

- JFrame: Representa la ventana principal de la aplicación. Tiene borde, título y botones de control (minimizar, maximizar, cerrar).
- JDialog: Una ventana secundaria, típicamente para mensajes, entradas específicas o configuraciones.
- JPanel: El contenedor más versátil. Se usa para agrupar componentes dentro de un JFrame, JDialog u otro JPanel. Es la base para crear secciones complejas en la interfaz.
- JScrollPane: Aunque también es un componente, actúa como contenedor para el componente que necesita desplazamiento.

Gestores de Diseño (Layout Managers)

Determinan cómo se posicionan y dimensionan los componentes dentro de un contenedor. Swing ofrece varios:

- FlowLayout: Coloca los componentes uno tras otro, en fila, de izquierda a derecha. Si no caben en una fila, pasa a la siguiente. Es el layout por defecto de JPanel.
- BorderLayout: Divide el contenedor en cinco regiones: Norte, Sur, Este, Oeste y Centro. Cada región puede contener un componente. Es el layout por defecto de JFrame y JDialog.
- GridLayout: Organiza los componentes en una rejilla rectangular de igual tamaño.
- BoxLayout: Coloca los componentes en una sola fila (horizontal) o columna (vertical). Respeta los tamaños preferidos de los componentes.
- GridBagLayout: El más potente y complejo. Permite colocar componentes en una rejilla flexible, especificando restricciones detalladas para cada uno (posición, tamaño, expansión, alineación).
- null Layout (Posicionamiento Absoluto): Permite especificar las coordenadas (x, y) y el tamaño (ancho, alto) exactos de cada componente. **Generalmente desaconsejado** porque no se adapta a diferentes tamaños de ventana, resoluciones o fuentes, haciendo la aplicación frágil y poco portable.

Eventos y Escuchadores (Events and Listeners)

Swing utiliza un **modelo de delegación de eventos** para manejar la interacción del usuario:

1. **Fuente del Evento:** Un componente con el que el usuario interactúa (ej. un JButton).
2. **Objeto Evento:** Cuando ocurre una interacción (ej. clic en un botón), la fuente crea un objeto evento (ej. ActionEvent) que encapsula la información sobre lo ocurrido.
3. **Escuchador (Listener):** Un objeto que implementa una interfaz específica (ej. ActionListener). Está *registrado* en la fuente.
4. **Notificación:** La fuente notifica a todos sus escuchadores registrados enviándoles el objeto evento (llamando a un método específico de la interfaz, ej. actionPerformed(ActionEvent e)).
5. **Respuesta:** El código dentro del método del escuchador se ejecuta en respuesta al evento.

Tipos comunes de eventos y sus listeners:

- ActionEvent -> ActionListener (clics en botones, selección en menús, Enter en JTextField)
- MouseEvent -> MouseListener (clic, presionar, soltar, entrar, salir del componente), MouseMotionListener (mover, arrastrar)
- KeyEvent -> KeyListener (presionar, soltar, teclear carácter)
- WindowEvent -> WindowListener (abrir, cerrar, activar, desactivar ventana)
- ItemEvent -> ItemListener (cambio de estado en JCheckBox, JRadioButton, JComboBox)
- ListSelectionEvent -> ListSelectionListener (cambio de selección en JList, JTable)

2. Construcción Básica de una GUI

Pasos típicos para crear una ventana simple:

1. Crear una instancia de JFrame.
2. Crear los componentes (JLabel, JButton, etc.).
3. Crear JPanels si es necesario para agrupar componentes.
4. Establecer el LayoutManager para el JFrame (su contentPane) y para cada JPanel.
5. Añadir los componentes a los JPanels y los JPanels (u otros componentes directamente) al contentPane del JFrame.
6. Registrar listeners a los componentes que necesiten responder a eventos.
7. Establecer el comportamiento de cierre (setDefaultCloseOperation).
8. Dimensionar la ventana (setSize() o pack()). pack() ajusta el tamaño de la ventana al tamaño preferido de sus componentes según el layout.
9. Hacer visible la ventana (setVisible(true)). **Importante:** Esto debe hacerse generalmente *después* de añadir todos los componentes y, crucialmente, en el *Event Dispatch Thread* (EDT).

```
import javax.swing.*;
import java.awt.*; // Necesario para Layout Managers y a veces para colores/fuentes
import java.awt.event.*; // Necesario para eventos y listeners

public class VentanaSimple {

    public static void main(String[] args) {
        // Es buena práctica asegurar que el código Swing se ejecute en el EDT
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                crearYMostrarGUI();
            }
        });
    }

    private static void crearYMostrarGUI() {
        // 1. Crear el JFrame (la ventana principal)
        JFrame frame = new JFrame("Mi Primera Ventana Swing");
        // 7. Establecer qué hacer cuando se cierra la ventana
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // 8. Establecer un tamaño inicial (alternativa a pack())
        frame.setSize(400, 200);
        // Centrar la ventana en la pantalla
        frame.setLocationRelativeTo(null);

        // 2. Crear componentes
```

```
        JLabel etiqueta = new JLabel("¡Hola, Swing!");
        etiqueta.setHorizontalAlignment(SwingConstants.CENTER); // Centrar texto en la etiqueta

        JButton boton = new JButton("Haz clic aquí");

        // 6. Añadir un listener al botón (usando una clase anónima)
        boton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Acción a realizar cuando se hace clic en el botón
                JOptionPane.showMessageDialog(frame, "¡Botón pulsado!");
            }
        });

        // 4. Establecer el Layout Manager para el contentPane del JFrame
        // BorderLayout es el default para JFrame, pero lo ponemos explícitamente
        frame.getContentPane().setLayout(new BorderLayout(10, 10)); // Espacio horizontal y
vertical entre componentes

        // 5. Añadir componentes al contentPane del JFrame
        frame.getContentPane().add(etiqueta, BorderLayout.CENTER); // Añadir etiqueta en el
centro
        frame.getContentPane().add(boton, BorderLayout.SOUTH);    // Añadir botón en la parte
inferior

        // Alternativa usando un JPanel con FlowLayout:
        /*
        JPanel panel = new JPanel(); // FlowLayout por defecto
        panel.add(etiqueta);
        panel.add(boton);
        frame.getContentPane().add(panel, BorderLayout.CENTER);
        */

        // 8. (Opcional) Ajustar tamaño de la ventana a los componentes
        // frame.pack(); // Descomentar si prefieres que la ventana se ajuste

        // 9. Hacer visible la ventana
        frame.setVisible(true);
    }
}
```

- **Nota sobre el EDT:** El código SwingUtilities.invokeLater(...) asegura que la creación y manipulación de la GUI ocurra en el hilo correcto (Event Dispatch Thread), lo cual es

crucial en Swing.

3. Conceptos Intermedios

Manejo de Eventos Avanzado

- **Clases Anónimas:** Convenientes para listeners simples (como en el ejemplo anterior).
- **Expresiones Lambda (Java 8+):** Aún más concisas para interfaces funcionales (como ActionListener).
  boton.addActionListener(e -> JOptionPane.showMessageDialog(frame, "¡Botón pulsado con Lambda!"));

- **Clases Internas:** Clases definidas dentro de otra clase. Pueden acceder a miembros de la clase contenedora. Útil si el listener es complejo o reutilizado por pocos componentes.
- **Clases Separadas:** Implementar el listener en una clase aparte. Bueno para lógica compleja o reutilizable en múltiples ventanas.
- **Obtener la Fuente:** El objeto ActionEvent (y otros) tiene el método getSource() para saber qué componente originó el evento.

Más Componentes Útiles

- **Menús:** JMenuBar, JMenu, JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem. Se añaden al JFrame con setJMenuBar().
- **Diálogos Estándar:** JOptionPane para mostrar mensajes (showMessageDialog), pedir confirmación (showConfirmDialog), solicitar entrada (showInputDialog).
- **Selección de Archivos:** JFileChooser para permitir al usuario seleccionar archivos o directorios.
- **Barras de Herramientas:** JToolBar para botones de acceso rápido.
- **Sliders y Progress Bars:** JSlider, JProgressBar.

Pintura Personalizada (paintComponent)

Para dibujar formas, imágenes o crear componentes con apariencia única, se sobrescribe el método paintComponent(Graphics g) de un JPanel (o cualquier JComponent).

- **Importante:** Siempre llamar a super.paintComponent(g) al principio para que el componente se dibuje correctamente (fondo, bordes, etc.).
- El objeto Graphics (o mejor, Graphics2D obtenido haciendo un cast) proporciona métodos para dibujar (drawLine, drawRect, fillRect, drawOval, fillOval, drawString, drawImage, etc.).
- La pintura debe ser rápida y solo debe ocurrir en respuesta a una llamada del sistema (no llames a paintComponent directamente). Si necesitas repintar, llama a repaint().

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

```
class PanelDibujo extends JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g); // MUY IMPORTANTE

        Graphics2D g2d = (Graphics2D) g; // Mejor usar Graphics2D para más opciones

        // Configurar renderizado suave (antialiasing)
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);

        // Dibujar un rectángulo rojo
        g2d.setColor(Color.RED);
        g2d.fillRect(20, 20, 100, 50);

        // Dibujar un círculo azul
        g2d.setColor(Color.BLUE);
        g2d.fillOval(150, 20, 60, 60);

        // Dibujar texto
        g2d.setColor(Color.BLACK);
        g2d.setFont(new Font("Arial", Font.BOLD, 16));
        g2d.drawString("Dibujo Personalizado", 40, 120);
    }

    // Es buena práctica definir el tamaño preferido
    @Override
    public Dimension getPreferredSize() {
        return new Dimension(300, 200);
    }
}

// Para usarlo:
// JFrame frame = ...
// PanelDibujo panelDibujo = new PanelDibujo();
// frame.getContentPane().add(panelDibujo, BorderLayout.CENTER);
```

Look and Feel (L&F)

Controla la apariencia general de los componentes Swing. Puedes cambiarlo para que la aplicación se vea diferente:

- **Metal:** El L&F clásico de Java (por defecto en algunas versiones antiguas).
- **Nimbus:** Un L&F más moderno introducido en Java 6 update 10.

- **System:** Intenta imitar la apariencia nativa del sistema operativo donde se ejecuta la aplicación (Windows, macOS, Linux GTK+).
- Otros L&F de terceros.

Se establece *antes* de crear cualquier componente, usualmente al inicio del main:

```
public static void main(String[] args) {
    try {
        // Intentar establecer el Look and Feel del Sistema Operativo
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.err.println("No se pudo establecer el Look and Feel del sistema: " + e);
        // O intentar Nimbus como alternativa
        // try { UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel"); }
catch (Exception nimbusE) {}
    }

    // El resto del código para lanzar la GUI...
    SwingUtilities.invokeLater(() -> crearYMostrarGUI());
}
```

4. Conceptos Avanzados

El Hilo de Despacho de Eventos (Event Dispatch Thread - EDT)

- **Regla de Oro: Todo el código que crea, modifica o interactúa con componentes Swing *debe* ejecutarse en el EDT.** Esto incluye crear ventanas, añadir componentes, cambiar texto de etiquetas, habilitar/deshabilitar botones, y manejar eventos.
- **¿Por qué?** Swing no es *thread-safe*. Acceder a los componentes desde múltiples hilos sin coordinación puede causar errores visuales, corrupción de datos e incluso bloqueos. El EDT es el único hilo que maneja todos los eventos de la GUI y las actualizaciones de pintura.
- **SwingUtilities.invokeLater(Runnable r):** Pone el Runnable en la cola de eventos del EDT para que se ejecute *después* de que se procesen los eventos pendientes. Es la forma estándar de iniciar la GUI y de realizar actualizaciones desde otros hilos. No bloquea el hilo actual.
- **SwingUtilities.invokeAndWait(Runnable r):** Similar a invokeLater, pero bloquea el hilo actual hasta que el Runnable haya terminado de ejecutarse en el EDT. ¡Cuidado! **Nunca llames a invokeAndWait desde el EDT**, causaría un bloqueo total (deadlock). Útil para obtener un resultado del EDT hacia otro hilo.
- **Tareas Largas: Nunca** realices tareas que consuman mucho tiempo (ej. acceso a red, operaciones de archivo pesadas, cálculos complejos) directamente en el EDT (ej. dentro de un ActionListener). Esto "congelará" la interfaz de usuario, haciéndola insensible.

SwingWorker

La solución estándar para ejecutar tareas largas en segundo plano y actualizar la GUI de forma segura:

- Es una clase genérica (SwingWorker<T, V>). T es el tipo del resultado final de la tarea, V es el tipo de los resultados intermedios publicados.
- Sobrescribes el método doInBackground(): Aquí va el código de la tarea larga. **Este método NO se ejecuta en el EDT.** Puedes devolver el resultado final (T).
- Sobrescribes el método process(List<V> chunks): Se ejecuta en el EDT. Recibe los resultados intermedios publicados desde doInBackground() usando el método publish(V... chunks). Útil para actualizar barras de progreso, logs, etc.
- Sobrescribes el método done(): Se ejecuta en el EDT *después* de que doInBackground() termina (normalmente o por excepción). Aquí puedes obtener el resultado final llamando a get() (¡cuidado, get() puede lanzar excepciones si la tarea falló!) y actualizar la GUI final.

```
import javax.swing.*;
import java.awt.*;
import java.util.List;
```

```
import java.util.concurrent.ExecutionException;

public class EjemploSwingWorker extends JFrame {

    private JLabel estadoLabel;
    private JProgressBar progressBar;
    private JButton startButton;

    public EjemploSwingWorker() {
        super("Ejemplo SwingWorker");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());

        estadoLabel = new JLabel("Estado: Listo");
        progressBar = new JProgressBar(0, 100);
        startButton = new JButton("Iniciar Tarea Larga");

        startButton.addActionListener(e -> iniciarTarea());

        add(startButton);
        add(progressBar);
        add(estadoLabel);

        pack();
        setLocationRelativeTo(null);
    }

    private void iniciarTarea() {
        startButton.setEnabled(false); // Deshabilitar botón mientras corre
        progressBar.setValue(0);     // Reiniciar barra
        estadoLabel.setText("Estado: Trabajando...");

        // Crear y ejecutar el SwingWorker
        MiTareaWorker worker = new MiTareaWorker();
        worker.execute();
    }

    // Clase interna que extiende SwingWorker
    class MiTareaWorker extends SwingWorker<String, Integer> { // Resultado final String,
progreso Integer

        @Override
        protected String doInBackground() throws Exception {
```

```
        // --- Tarea Larga (NO en EDT) ---
        int progreso = 0;
        while (progreso < 100) {
            // Simular trabajo
            Thread.sleep(50); // ¡NUNCA sleep en el EDT! Aquí está bien.

            progreso += 2;

            // Publicar progreso intermedio
            publish(progreso);
        }
        // --- Fin Tarea Larga ---
        return "¡Tarea completada con éxito!"; // Resultado final
    }

    @Override
    protected void process(List<Integer> chunks) {
        // --- Actualizar GUI con progreso (SÍ en EDT) ---
        int ultimoProgreso = chunks.get(chunks.size() - 1);
        progressBar.setValue(ultimoProgreso);
        estadoLabel.setText("Estado: Progreso " + ultimoProgreso + "%");
    }

    @Override
    protected void done() {
        // --- Actualizar GUI al finalizar (SÍ en EDT) ---
        try {
            String resultado = get(); // Obtener resultado de doInBackground()
            estadoLabel.setText("Estado: " + resultado);
            progressBar.setValue(100); // Asegurar que llega al 100%
        } catch (InterruptedException | ExecutionException e) {
            estadoLabel.setText("Estado: Error - " + e.getMessage());
            e.printStackTrace(); // Manejar el error apropiadamente
             progressBar.setValue(0);
        } finally {
            startButton.setEnabled(true); // Rehabilitar el botón
        }
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        new EjemploSwingWorker().setVisible(true);
```

```
        });
    }
}
```

Modelo-Vista-Controlador (MVC) en Swing

Componentes como JList, JTable, y JTree están diseñados siguiendo (más o menos) el patrón MVC:

- **Modelo:** Contiene los datos (ListModel, TableModel, TreeModel). Ej: DefaultListModel, DefaultTableModel. Tú interactúas con el modelo para añadir, eliminar o modificar datos.
- **Vista:** El componente Swing en sí (JList, JTable), responsable de *mostrar* los datos del modelo.
- **Controlador:** Los listeners que reaccionan a la interacción del usuario con la Vista y actualizan el Modelo o realizan otras acciones. A menudo, la lógica del controlador se mezcla en la clase que contiene la vista.

Usar los modelos desacopla los datos de su presentación, haciendo el código más organizado y mantenible, especialmente para datos dinámicos o grandes.

Otros Temas Avanzados

- **Componentes Personalizados:** Crear tus propios componentes reutilizables extendiendo JPanel o JComponent y usando paintComponent.
- **Arrastrar y Soltar (Drag and Drop):** Swing tiene soporte incorporado para operaciones D&D.
- **Accesibilidad (a11y):** Soporte para tecnologías asistivas (lectores de pantalla). Componentes Swing implementan la interfaz Accessible.
- **Acciones (Action):** Una interfaz que encapsula no solo el ActionListener, sino también propiedades como texto, icono, tooltip, estado habilitado/deshabilitado. Útil para representar la misma acción en un menú y una barra de herramientas.

5. Buenas Prácticas y Consideraciones

- **Usa Layout Managers:** Evita el null layout siempre que sea posible. Elige el layout adecuado para cada situación. Anida JPanels con diferentes layouts para lograr diseños complejos.
- **Respeta el EDT:** La regla más importante. Usa invokeLater y SwingWorker.
- **Organiza tu Código:** Separa la lógica de la GUI de la lógica de negocio. Usa clases diferentes para ventanas principales, diálogos, paneles complejos y listeners si se vuelven grandes.
- **Considera MVC:** Especialmente para JList, JTable, JTree.
- **Nomenclatura:** Usa nombres descriptivos para tus componentes (ej. nombreTextField, aceptarButton).
- **Alternativas Modernas:** Swing es una tecnología madura y robusta, pero para nuevas aplicaciones de escritorio, también existen JavaFX (el sucesor recomendado por Oracle) o frameworks multiplataforma basados en tecnologías web (como Electron) que pueden ofrecer una apariencia más moderna o facilidades diferentes. Sin embargo, Swing sigue siendo ampliamente utilizado y es fundamental conocerlo para mantener o trabajar con muchas aplicaciones Java existentes.

6. Conclusión

Java Swing proporciona un ecosistema completo para construir interfaces gráficas de usuario en Java. Dominar sus conceptos clave (componentes, contenedores, layouts, eventos) y, sobre todo, entender y respetar el modelo de hilos (EDT y SwingWorker) es esencial para crear aplicaciones robustas, responsivas y mantenibles. Aunque tiene una curva de aprendizaje y algunas peculiaridades, sigue siendo una herramienta potente y relevante en el desarrollo Java.