Tema 1 – Fundamentos

- Java y Swing
- Entorno de desarrollo
- Crear y ejecutar una aplicación java – swing
- Generar un ejecutable de la aplicación
- Contenedores
- *Layouts*
- *Look & Feel*
- JComponent
- Manejo de eventos
- Práctica: Puzzle

Frontend desktop con Java y Swing

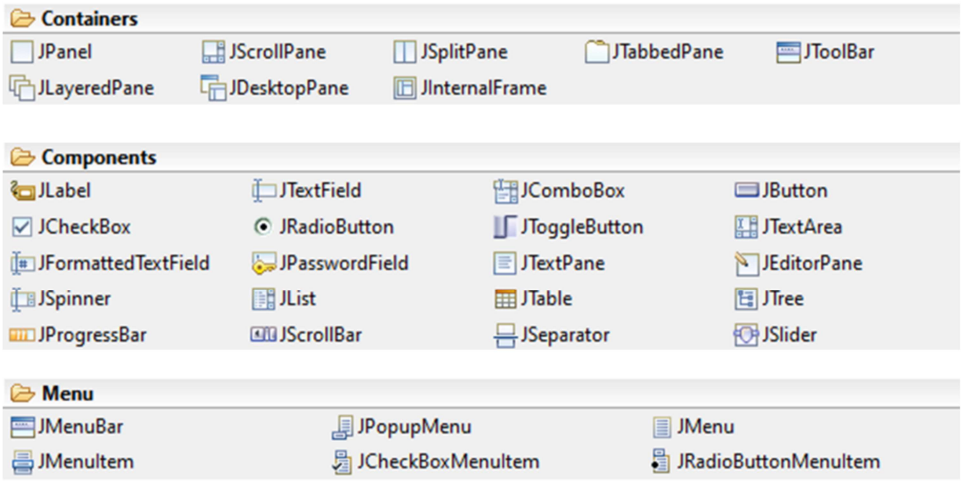
- Aplicación java (.jar) que se ejecuta sobre la máquina virtual de Java (JRE).
- **Swing** → Componentes visuales escritos en Java
  - Independientes de la plataforma
  - Soportan diferentes estilos ("Look & Feel")
  - No usan MVC
  - Paquetes `javax.swing.*`
  - Incluidos en el JDK desde la 1.2

- Un frontend para desktop con java es una aplicación que se ejecuta sobre la máquina virtual de java, denominado JRE, o Java Runtime Environment. En la asignatura vamos a aprender a desarrollar aplicaciones java para desktop con componentes Swing.
- Que es Swing? Es un conjunto de componentes escritos en java, que implementan controles de interfaz de usuario, desde botones a tablas, formularios y ventanas.
- La mayoría de los componentes se encuentran en el paquete javax.swing. Este paquete está incluido en el JDK desde la versión 1.2, por lo que no es necesario instalar ningún paquete adicional.
- Cada componente es una clase Java, que dispone de métodos para configurar su aspecto y contenido. Los componentes emiten y responden a eventos de usuario o del sistema.
- En general, no usan una arquitectura MVC, es decir, no se obliga a que el modelo, la vista y el controlador estén separados.

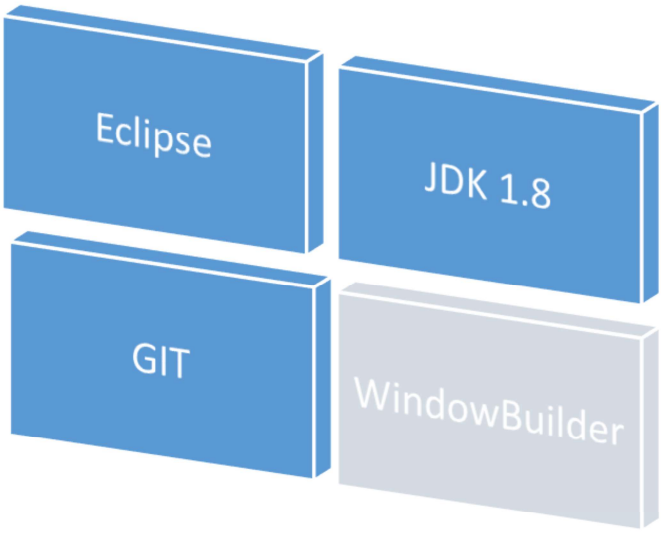
Documentación

https://docs.oracle.com/javase/tutorial/uiswing/TOC.html

Componentes Swing



Entorno de desarrollo



Primera aplicación en Java-Swing con Eclipse

- New → Project → Gradle Project
- Inicializar repositorio Git (opcional)
- Crear clase Main.java

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

    JFrame frame = new JFrame("Hola Mundo");
    frame.setSize(500, 500);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.setVisible(true);

}
```

- Run as... → Java application

Construir un ejecutable JAR

- Modificar fichero **build.gradle**:

```
...
jar {
       manifest {
        attributes 'Main-Class': 'xxxx.Main'
       }
}
...
```

Clase con main()

- Ejecutar tarea "jar" de gradle

```
> gradle jar
```

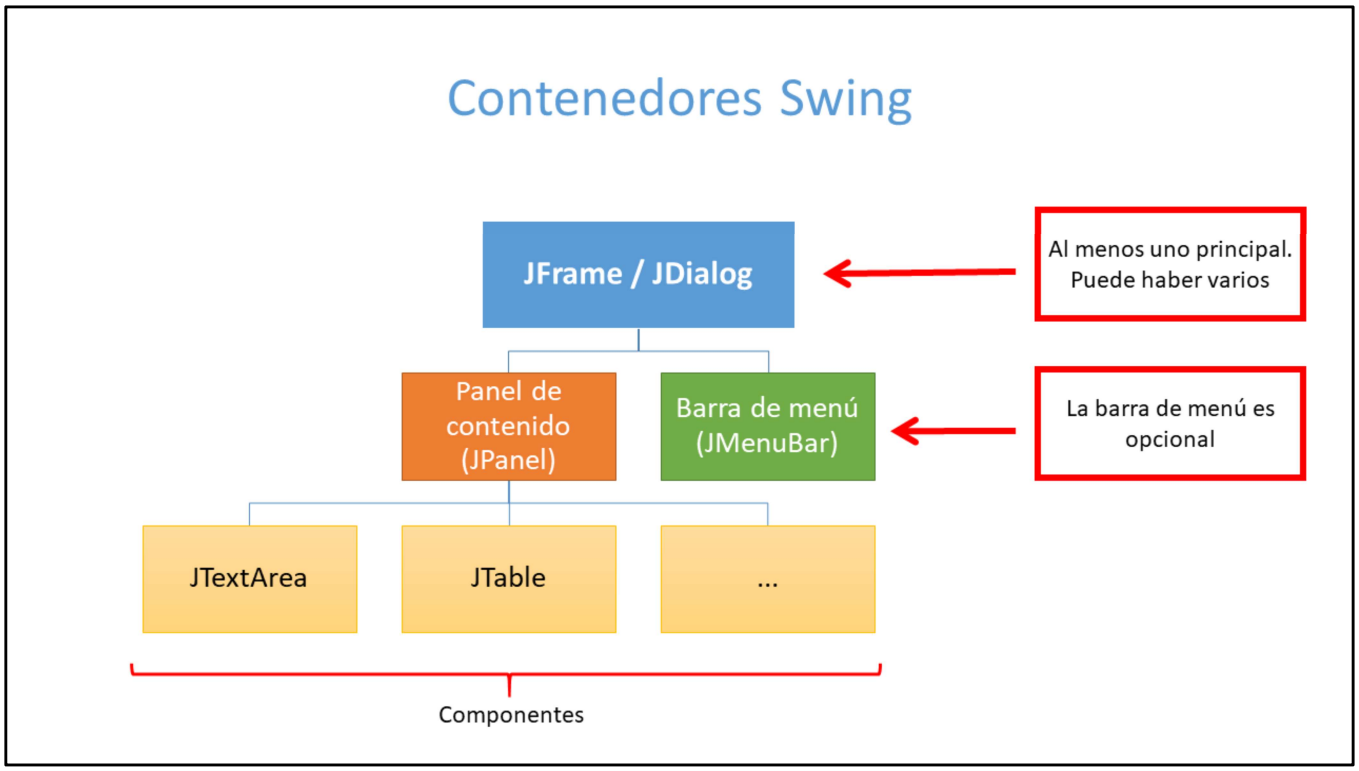
- El ejecutable jar se genera en **/build/libs**
- Para ejecutar la aplicación:

```
> java -jar aplicacion.jar
```

- Se puede generar un jar ejecutable con la tarea "jar" de gradle, pero antes es necesario especificar la clase que contiene la función "main" en el fichero MANIFEST.MF que se genera. Para ello, añadir al fichero "build.gradle":

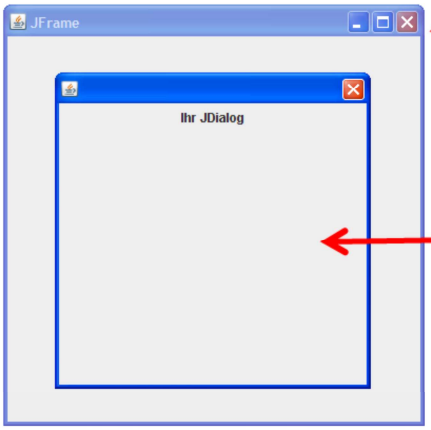
```
jar {
      manifest {
       attributes 'Main-Class': 'swing.Main'
      }
}
```

- El ejecutable se genera en "build/libs", y se puede ejecutar con "java –jar" en cualquier máquina que tenga instalado el JRE.



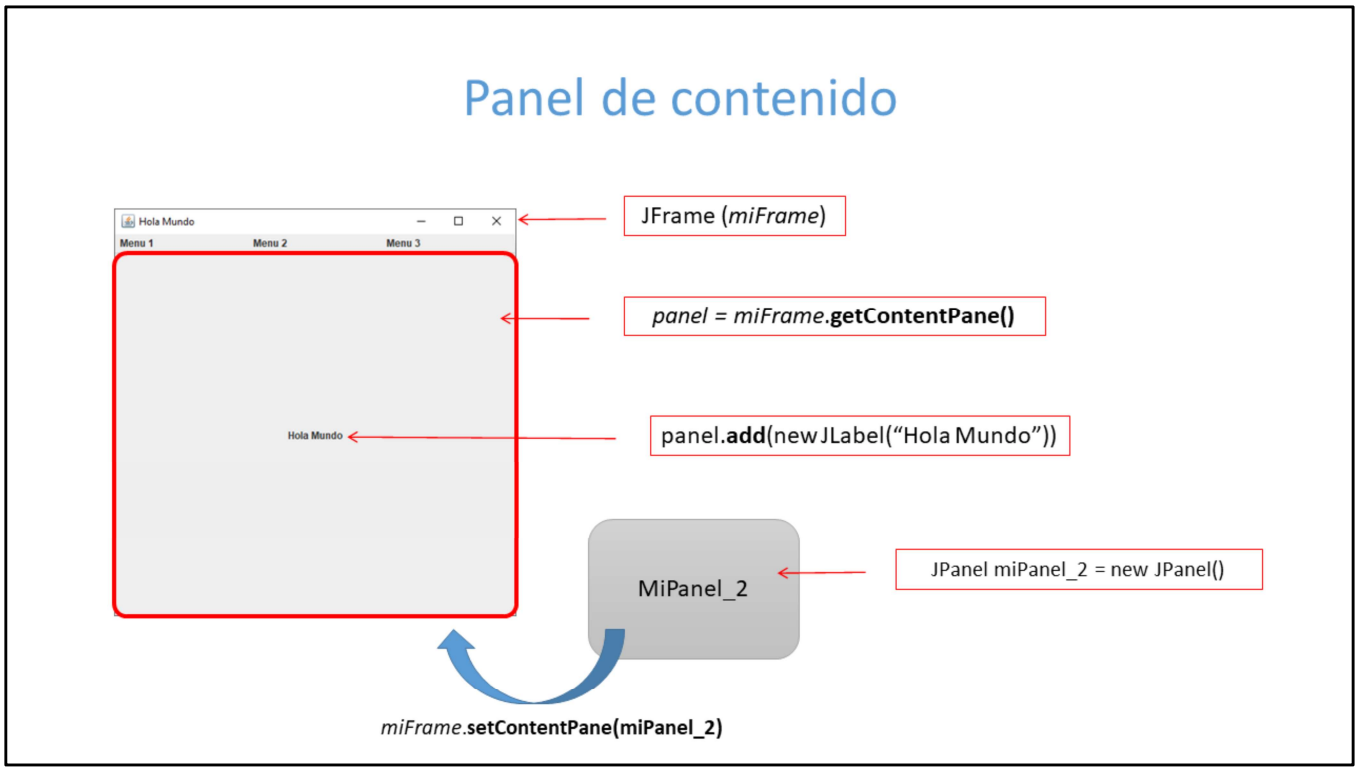
- Cada aplicación que usa Swing tiene al menos un contenedor principal, de tipo Frame, Dialog o Applet.
- Normalmente una aplicación de escritorio tendrá un Frame o Ventana principal, y uno o varios Dialogs.
- Cada uno de ellos es un contenedor de nivel superior, que contendrá a su vez una jerarquía de componentes.
- Un contenedor principal tiene un panel de contenido y opcionalmente un menú.

JFrame vs JDialog



JFrame
Ventana principal de la aplicación

JDialog
Dialogo emergente: modal, no dimensionable



Podemos acceder al panel de contenido de un contenedor con el método "getContentPane()".

También podemos sustituir el panel por defecto con otro contenedor con "setContentPane()". De esta forma podemos reutilizar paneles o vistas ya existentes.

Una vez que tenemos el panel de contenido, podemos añadirle más componentes con "add"

Distribución de componentes en contenedores con Layouts

- JFrame, JDialog y JPanel → Necesitan un **layout**
  - Especifica la distribución visual de los componentes
  - Formas de asignarlo:
    - En el constructor:

```
JPanel panel = new JPanel(new BoxLayout());
```

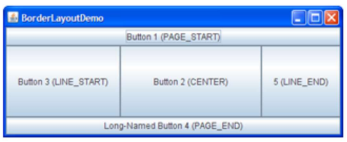
    - Con "`setLayout()`"

```
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout());
```

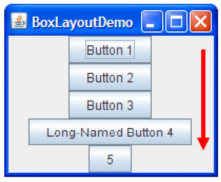
En los contenedores principales (JFrame, JDialog) y en el componente JPanel, es necesario especificar un layout para distribuir a los componentes que contiene.

- El layout se puede establecer en el constructor del componente
- O bien se puede establecer a posteriori con "setLayout"
- En ambos casos es necesario crear un objeto del tipo del layout que queremos.

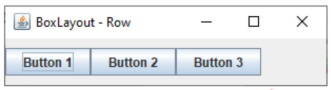
Principales tipos de Layout



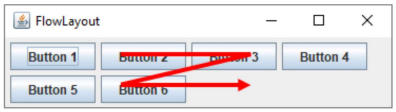
BorderLayout



BoxLayout (columna)

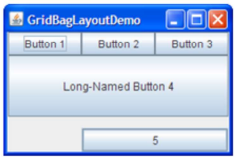


BoxLayout (fila)



FlowLayout

GridLayout



GridBagLayout

- BorderLayout: Distribuye los componentes en 5 regiones del contenedor.
- BoxLayout: alinea los componentes en una fila o columna.
- FlowLayout: Es el que tienen por defecto los paneles. Como en HTML, los componentes se colocan de izquierda a derecha y de arriba a abajo.
- GridLayout: Distribuye los componentes en un grid de n*m filas y columnas.
- GridBagLayout: Es un layout basado en un grid flexible. Se pueden unir celdas para formar layouts complejos.

Look and Feel

- El *Look and Feel* (L&F) define *de forma global* el aspecto visual y el comportamiento de los componentes.
- El JRE proporciona 2 L&F básicos:
  - Metal (Java L&F): Se ve igual en todas las plataformas. Se aplica por defecto. Tiene varios "temas" de colores y fuentes.
  - SystemLookAndFeel: Especifico de la plataforma. Se determina en tiempo de ejecución.
- Se puede cambiar por código (también por configuración), con la clase **UIManager**:

```
// L&F nativo del sistema
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

// L&F Java (Metal)
UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());

// Un L&F específico
UIManager.setLookAndFeel(new javax.swing.plaf.nimbus.NimbusLookAndFeel());
```

- El aspecto visual, el manejo de eventos y otros aspectos de los componentes, se controlan mediante el look and feel.
- Swing proporciona varios L&F, y se pueden usar otros de terceros. Para establecer el L&F se usa la clase UIManager, método estático setLookAndFeel().

JComponent

- Es la clase base para todos los componentes (salvo contenedores)
- Proporciona métodos y propiedades comunes para:
  - Aspectos visuales y Layout
  - Acceso a componentes relacionados

Todas las clases que empiezan por J, a excepción de los containers, derivan de **JComponent**. Por tanto es una clase importante, ya que proporciona muchos métodos y propiedades comunes a todos los componentes.

JComponent – Aspectos visuales

- setLayout
- setMinimumSize, setMaximumSize, setPreferredSize, setSize
- setBorder
- setEnabled
- setVisible
- setForeground, setBackground
- setFont
- setAlignmentX, Y
- setCursor

JComponent – Acceso a componentes relacionados

- add(Component)
- getTopLevelAncestor()
- getParent()
- getComponent(int), getComponents()

Manejo de eventos

- Eventos → Comunican cambios de estado.
  - Lanzados por los componentes
  - Son objetos derivados de *EventObject (java.util)*
  - Cada tipo de evento tiene diferente información
- Tipos de eventos comunes:
  - Teclado
  - Ratón
  - Ventana (abrir, cerrar, activar)
  - Componente (foco, cambio de valor...): específicos del componente.
  - **Acciones** (*actions*): asociadas a botones, menus, controles...

Los componentes de Swing comunican sus cambios de estado mediante eventos.

Los eventos en java son objetos de diferentes tipos, según sea el origen del evento. Todos derivan inicialmente de java.útil.EventObject, que contiene únicamente la propiedad “source” de tipo Object, que identifica el componente que origina el evento.

Los principales tipos de eventos que solemos manejar en una aplicación son:
- Eventos de teclado
- Eventos de raton
- Eventos de ventanas: abrir, cerrar, activar...
- Eventos específicos de componentes: recibir o perder el foco, cambios de valor o estado, etc.

Además de estos, hay un tipo de evento que se produce al realizar una Acción (Action). En Swing, una acción puede ser asociada a un botón o a un menú, de forma que cuando dicho botón o menú es pulsado, se lanza un evento para la acción asociada. Esto permite asociar una misma acción a un menú y a un botón, por ejemplo.

Manejo de eventos

- Manejadores de eventos (*event listeners*)

En la programación orientada a eventos, la aplicación debe responder a los eventos que lanzan los componentes, mediante manejadores de eventos, de igual forma que vimos en su momento en la asignatura de Angular. En java estos manejadores de eventos se denominan **event listeners**.

Este esquema usa el patrón Observador – Suscriptor.

Manejo de eventos

- Los ***listeners*** son del tipo adecuado al evento a manejar:
  - Eventos de teclado (KeyEvent) → *KeyListener*
  - Eventos de ratón (MouseEvent) → *MouseListener*
  - Eventos tipo Acción (ActionEvent) → *ActionListener*
  - ...
- Los listener hay que **registrarlos** en el componente que emite el evento:
  - componente.**add*XXXXX*Listener**(*<listener>*)
    - *addMouseListener, addKeyListener, addActionListener...*

Para capturar un evento de determinado tipo, debemos crear y registrar un objeto de tipo Listener adecuado al tipo de evento. Los listener implementan la interfaz EventListener, y además, en función del tipo de evento que manejen, deben implementar una interfaz especifica.

El listener debe ser registrado en el objeto o componente fuente del evento, para que sea llamado con la información del evento. Esto se realiza mediante el método "addListener" del componente

Manejo de eventos

- Manejo de un evento click de ratón en un botón:

```
JButton miBoton = new JButton("Boton 1");

MouseListener miListener = new MouseListener() {
       @Override
       public void mouseClicked(MouseEvent e) {
         JButton source = (JButton) e.getSource();
         System.out.println("Botón pulsado: " + source.getText());
       }
};

miBoton.addMouseListener(miListener);
...
otroBoton.addMouseListener(miListener);
```

A modo de ejemplo, vamos a ver qué tipos de objetos intervienen en el tratamiento de un clic sobre un botón.

Al hacer clic sobre un botón, dicho botón emite un evento de tipo MouseEvent, con información sobre el botón pulsado, las coordenadas x e y, etc. La propiedad “source” del evento apunta al propio botón pulsado. Si no hay ningún listener que procese el evento, este se pierde.

Por otra parte, podemos registrar en el botón un listener o manejador de eventos de ratón, añadiendo un objeto que implemente la interfaz MouseListener, con el método addMouseListener().

Podemos crear el listener y asignarlo de un solo paso, usando una clase anónima.

El código del ejemplo da error al compilar, ya que la interfaz MouseListener tiene más métodos que es necesario implementar. Para no tener que implementar todos los métodos de MouseListener, podemos usar en su lugar la clase MouseAdapter, y sobrescribir solo los métodos que nos hagan falta. Existen “adapters” para casi todos los tipos de listeners.