

GUIs en Java

Dr. Antonio LaTorre
e-mail: atorre@fi.upm.es

Índice

- **Introducción**
- Programando con Swing
 - Introducción y Jerarquía de Swing
 - Gestión de Eventos: Events, Listeners y Adapters
 - Posicionamiento de los componentes: Layouts
 - Interfaz gráfica para el desarrollo de GUIs
 - Controles Swing

Introducción

- Las interfaces gráficas permiten desarrollar aplicaciones más complejas
- Aumentan la interactividad y la productividad
- Sin embargo, su desarrollo conlleva una serie de complicaciones añadidas
 - Los componentes de la interfaz deben ser programados
 - Necesitamos mecanismos para disponerlos en la pantalla
 - Necesitamos mecanismos de control de eventos
- Afortunadamente, hay bibliotecas que nos facilitan mucho las cosas...

Tipos de Aplicaciones

- Aplicaciones autónomas
 - Aplicaciones de consola
 - **Aplicaciones con interfaz gráfico (GUI)**
- Applets
- Aplicaciones Java Web Start

Aplicaciones autónomas

- Sólo necesitan la máquina virtual de Java (JVM)
- Almacenadas en el disco duro del usuario
- Se ejecutan con: **java ClaseMain**
- Interfaz formada por una o varias ventanas
- Aplicación finaliza cuando se cierra la ventana
- Sin restricciones de seguridad específicas

Applets

- Necesitan de un navegador aparte de la JVM
- Normalmente, se cargan desde un servidor Web
- Comienzan su ejecución cuando se carga la página Web que la contiene
- Finaliza su ejecución cuando se cierra la página
- Su interfaz se incluye en una región rectangular del HTML
- Tiene restricciones de seguridad

Aplicaciones Web Start

- Necesitan un Gestor de Aplicaciones además de la JVM
- Normalmente, se cargan desde la Web y se almacenan en el disco duro local
- Se ejecutan al pulsar un enlace en una Web
- Interfaz formado por ventanas
- Finaliza cuando se cierra la ventana
- Tienen restricciones de seguridad

Bibliotecas GUI

- **A**bstract **W**indowing **T**oolkit (AWT)
- **S**tandard **W**idget **T**oolkit (SWT)
- Swing / JFC (desde JDK 1.1.5)

AWT

- Look & Feel dependiente de la plataforma
- Funcionalidad independiente de la plataforma
- Controles más básicos
- Estándar hasta la versión 1.1.5
- Proporcionan la gestión de eventos

SWT

- Look & Feel nativo en cada sistema
- Más ligero que Swing
- En desarrollo (puede que no todos los controles estén disponibles)
- No incluida en JDK (hay que descargar e importar las clases)

Swing / JFC

- Núcleo de las Java Foundation Classes
- Estándar desde la versión de JDK 1.1.5
- Java Look & Feel (independiente de la plataforma)
 - Pluggable Look & Feel: Windows, Mac OS X, Linux
- Otras APIs adicionales:
 - Accesibilidad
 - Internacionalización

Paquetes JFC

- javax.accessibility
- javax.swing.plaf
- javax.swing.text.html
- **javax.swing**
- javax.swing.plaf.basic
- javax.swing.text.parser
- javax.swing.border
- javax.swing.plaf.metal
- javax.swing.text.rtf
- javax.swing.colorchooser
- javax.swing.plaf.multi
- javax.swing.tree
- **javax.swing.event**
- javax.swing.table
- javax.swing.undo
- javax.swing.filechooser
- javax.swing.text

Índice

- Introducción
- **Programando con Swing**
 - **Introducción y Jerarquía de Swing**
 - Gestión de Eventos: Events, Listeners y Adapters
 - Posicionamiento de los componentes: Layouts
 - Interfaz gráfica para el desarrollo de GUIs
 - Controles Swing

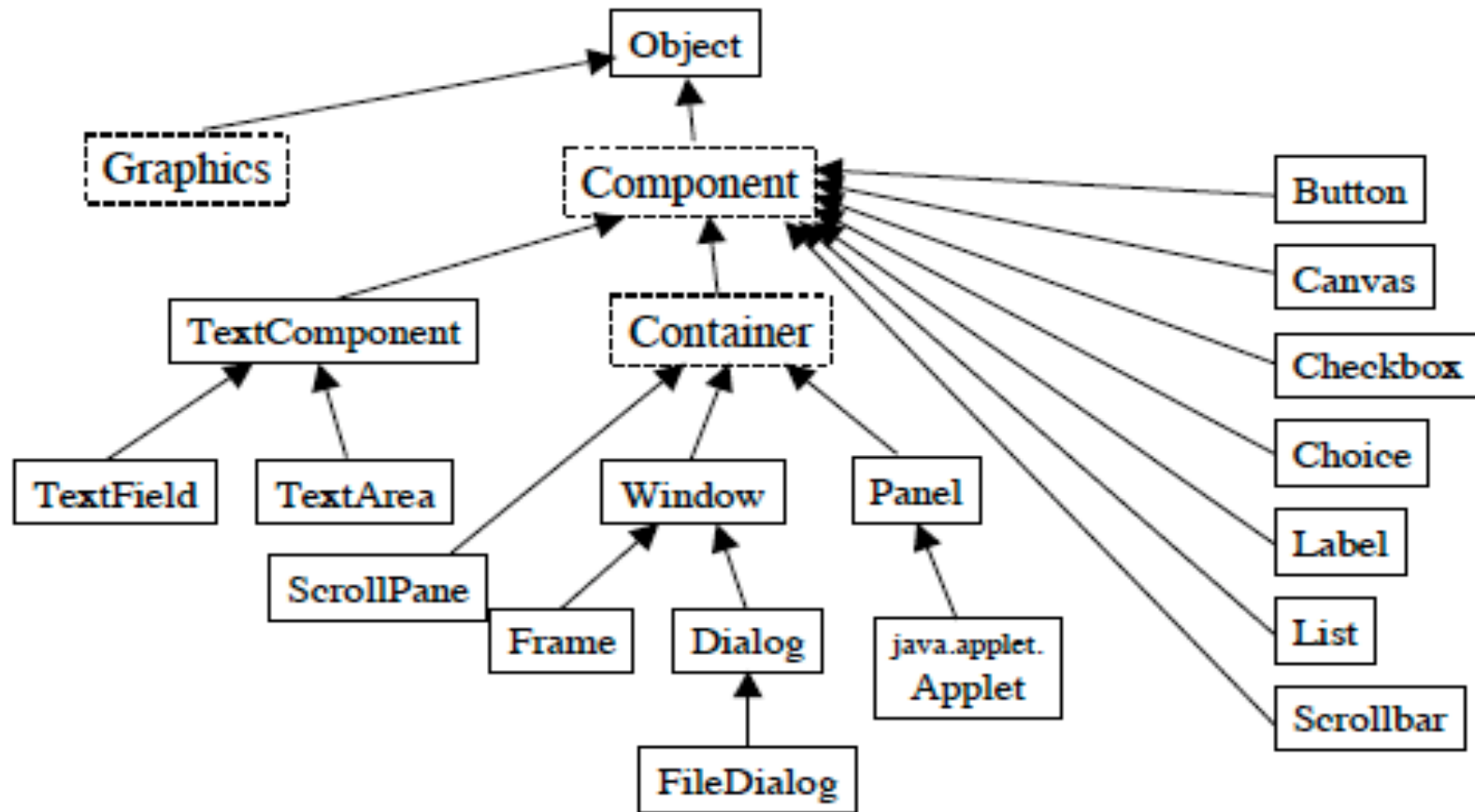
Elementos básicos

- Componentes GUI (widgets)
 - Elementos visuales del interfaz
 - Un programa con GUI es un conjunto de componentes anidados
- Administradores de diseño (Layouts managers)
 - Gestionan la organización de los componentes
- Creación de gráficos y texto (clase Graphics)
- Interactividad: manejo de Eventos

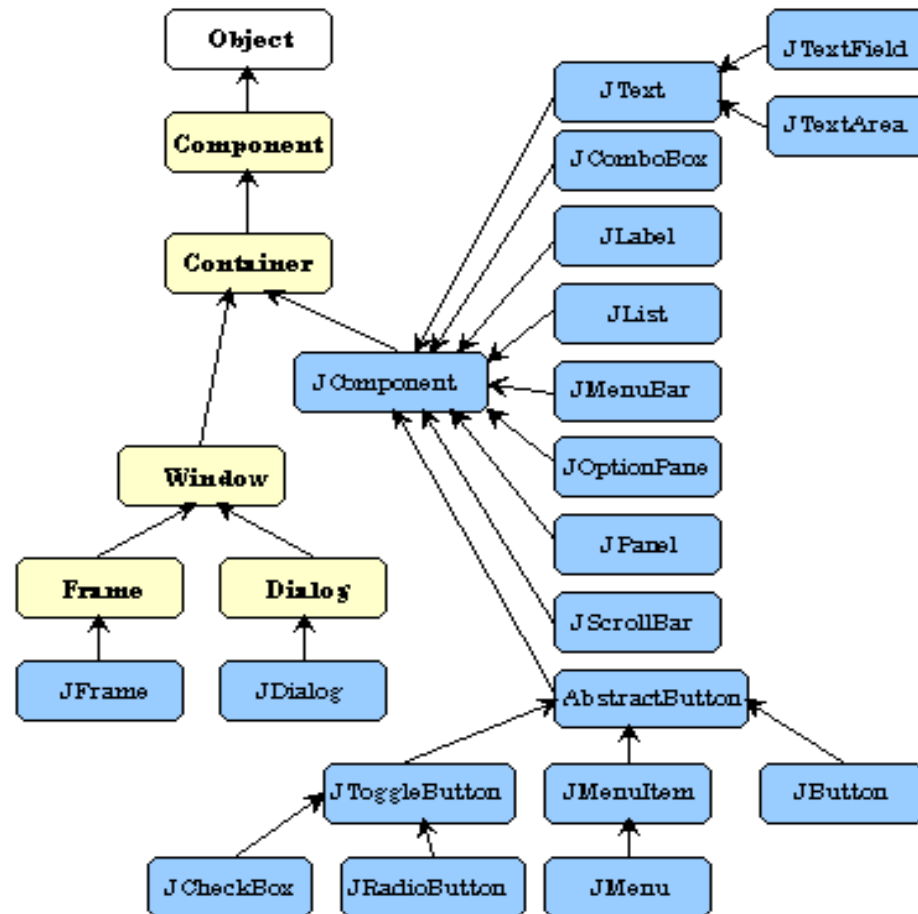
Componentes de la GUI

- Un componente es una instancia de una clase
- Se crean como cualquier otro objeto Java
- Tipos de componentes:
 - Contenedores: contienen a otros componentes
 - Lienzo (clase Canvas): superficie de dibujo
 - Componentes de Interfaz: botones, listas, menús, casillas de verificación (checkboxes), texto, etc.
 - Elementos de construcción de ventanas: ventanas, marcos, barras de menús, cuadros de diálogo, etc.

Jerarquía de clases AWT



Jerarquía de clases Swing



Primer programa con GUI

```
import javax.swing.*;

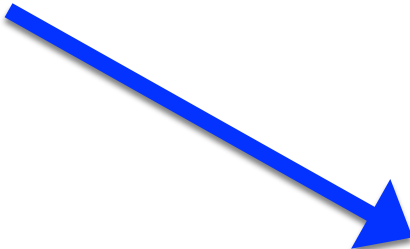
public class Main {
    public static void main(String[] args) {
        MySimpleGUI gui = new MySimpleGUI();
    }
}

class MySimpleGUI extends JFrame {
    public MySimpleGUI() {
        setSize(400, 200);
        setTitle("Ventana de tipo JFrame");
        setVisible(true);
    }
}
```

Primer programa con GUI

```
import javax.swing.*;
```

```
public class Main {  
    public static void main(String[] args) {  
        MySimpleGUI gui = new MySimpleGUI();  
    }  
}  
  
class MySimpleGUI extends JFrame {  
    public MySimpleGUI() {  
        setSize(400, 200);  
        setTitle("Ventana de tipo JFrame");  
        setVisible(true);  
    }  
}
```

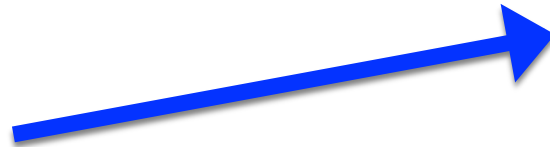


Importamos el paquete que
contiene los controles Swing

Primer programa con GUI

```
import javax.swing.*;  
public class Main {  
    public static void main(String[] args) {  
        MySimpleGUI gui = new MySimpleGUI();  
    }  
}
```

Creamos la clase que representa nuestra ventana extendiendo JFrame



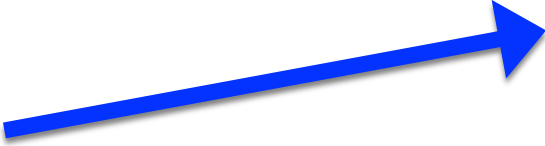
```
class MySimpleGUI extends JFrame {  
    public MySimpleGUI() {  
        setSize(400, 200);  
        setTitle("Ventana de tipo JFrame");  
        setVisible(true);  
    }  
}
```

Primer programa con GUI

```
import javax.swing.*;  
public class Main {  
    public static void main(String[] args) {  
        MySimpleGUI gui = new MySimpleGUI();  
    }  
}
```

En el constructor definimos el tamaño y el título de la ventana (y la hacemos visible)

class MySimpleGUI extends JFrame {



```
    public MySimpleGUI() {  
        setSize(400, 200);  
        setTitle("Ventana de tipo JFrame");  
        setVisible(true);  
    }
```

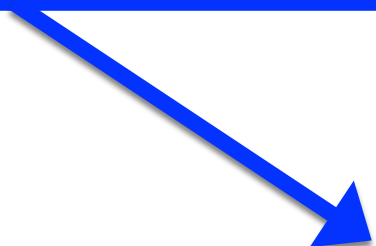
```
}
```

Primer programa con GUI

```
import javax.swing.*;
```

```
public class Main {  
    public static void main(String[] args) {  
        MySimpleGUI gui = new MySimpleGUI();  
    }  
}
```

```
class MySimpleGUI extends JFrame {  
    public MySimpleGUI() {  
        setSize(400, 200);  
        setTitle("Ventana de tipo JFrame");  
        setVisible(true);  
    }  
}
```



En el método main instanciamos la clase que acabamos de crear, para que nos muestre la ventana

Jerarquía de Swing

1. JComponent

1. AbstractButton

1. **JButton**: El botón típico de cualquier aplicación gráfica
2. **JMenuItem**
 1. **JMenu**: Elemento de un menú
 2. **JCheckBoxMenuItem**: Elemento de un menú que puede ser seleccionado
 3. **JRadioButtonMenuItem**: Elemento de un menú que forma parte de un conjunto del que sólo puede haber seleccionado uno
3. **JToggleButton**: Botón de dos estados
 1. **JCheckBox**: Elemento que puede estar seleccionado o no
 2. **JRadioButton**: Se usa junto con **ButtonGroup**, y sólo puede haber uno seleccionado

2. JColorChooser: Panel de selección de color

3. JComboBox: Lista desplegable de la que se puede elegir un elemento

4. JDesktopPane: Contenedor de frames internos

5. JFileChooser: Panel de selección de fichero

Jerarquía de Swing

6. **JInternalFrame**: Frame que puede colocarse dentro de otro contenedor
7. **JLabel**: Etiqueta donde se pueden poner texto e imágenes
8. **JLayeredPane**: Panel donde los objetos pueden estar a distinta profundidad
9. **JList**: Lista de elementos de la que podemos elegir uno o más elementos
10. **JMenuBar**: Barra superior del programa que contiene **JMenu's**
11. **JOptionPane**: Permite mostrar un diálogo (junto con **JDialog**)
12. **JPanel**: Contenedor genérico sobre el que se añaden otros componentes
13. **JPopupMenu**: Menú emergente que aparece al hacer click con el botón derecho del ratón
14. **JProgressBar**: Barra de progreso típica que se usa cuando una operación lleva cierto tiempo
15. **JScrollBar**: Barra de desplazamiento
16. **JScrollPane**: Panel contenedor con dos barras de desplazamiento
17. **JSeparator**: Línea separadora (por ejemplo, dentro de un menú)

Jerarquía de Swing

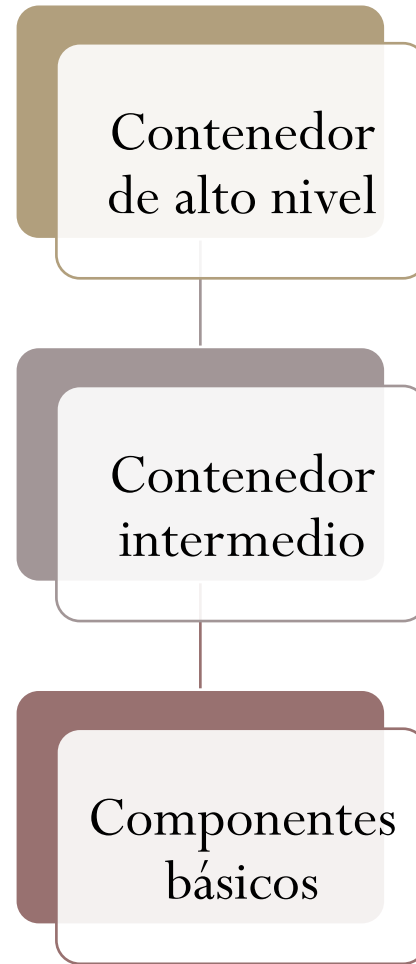
- 18. **JSlider**: Barra para seleccionar valores gráficamente
- 19. **JSpinner**: Permite seleccionar valores de una lista pulsando arriba y abajo
- 20. **JSplitPane**: Panel dividido en dos partes
- 21. **JTabbedPane**: Contenedor múltiple en el que seleccionamos un conjunto de componentes a través de pestañas
- 22. **JTable**: Componente para mostrar información de forma tabular
- 23. **JTextComponent**
 - 1. **JEditorPane**: Facilita la creación de un editor
 - 2. **JTextArea**: Permite la inserción de texto en múltiples líneas
 - 3. **TextField**: Igual que el anterior, pero sólo en una línea
 - 1. **JFormattedField**: Permite introducir texto con formato
 - 2. **JPasswordField**: El texto se oculta con el símbolo que prefiramos
- 24. **JToolBar**: Contenedor de iconos que suele aparecer en la parte superior
- 25. **JToolTip**: Texto emergente que aparece al situar el ratón sobre un control
- 26. **JTree**: Permite mostrar información jerarquizada en forma de árbol

Jerarquía de Swing

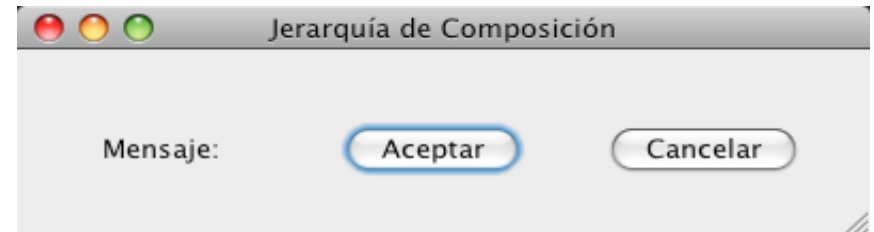
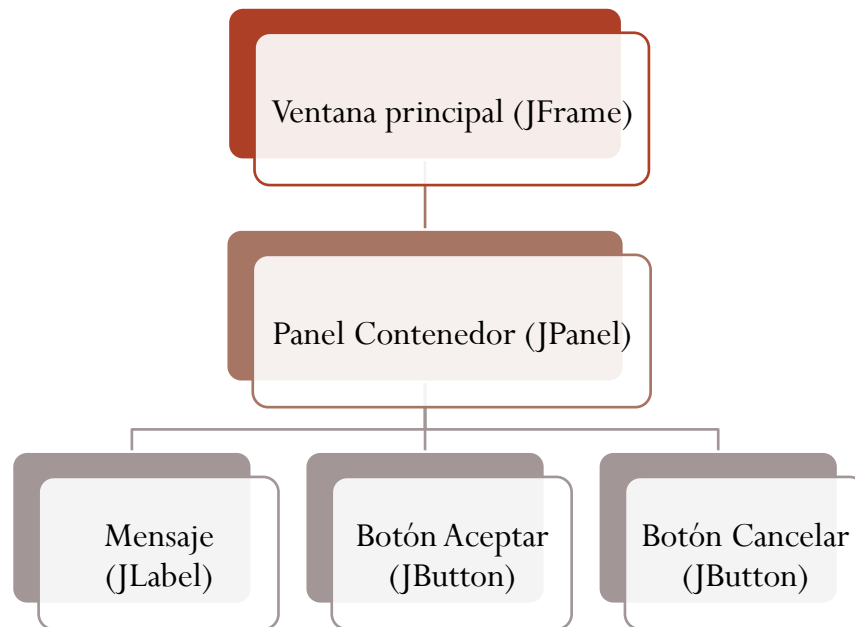
2. **Window** (AWT)

1. **JFrame**: Ventana básica de Swing
2. **JDialog**: Ventana modal (no se permite acceder a la ventana madre)
normalmente usada para mostrar diálogos (aceptar/cancelar, etc.)

Jerarquía de Composición



Ejemplo de Jerarquía de Composición



Pasos básicos en la construcción de una interfaz

1. Crear una nueva clase para nuestra ventana (o directamente instanciar JFrame)
2. Crear los componentes de nuestra interfaz
3. Crear uno o varios contenedores intermedios
4. Asociar los componentes al contenedor
5. Asociar el contenedor a la ventana
6. Hacer visible la ventana

Otro Ejemplo de Interfaz

// 1) Creamos la clase ventana

```
public class EjemploInterfaz extends JFrame {
```

```
    public EjemploInterfaz () {  
        initComponents();  
    }
```

```
    private void initComponents() {
```

// 2) Configuramos los parámetros de la ventana

```
        setSize(300, 200);
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
```

// 2) Crear los componentes

```
        JLabel etiqueta1 = new JLabel("Mensaje");
```

```
        JTextField campoDeTexto = new JTextField(20);
```

```
        JButton boton = new JButton("Aceptar");
```

Otro Ejemplo de Interfaz

// 3) Crear un contenedor

```
JPanel panelDeContenido = new JPanel();
```

// 4) Asociar los componentes al contenedor

```
panelDeContenido.add(etiqueta1);
```

```
panelDeContenido.add(campoDeTexto);
```

```
panelDeContenido.add(boton);
```

// 5) Asociar el contenedor a la ventana

```
setContentPane(panelDeContenido);
```

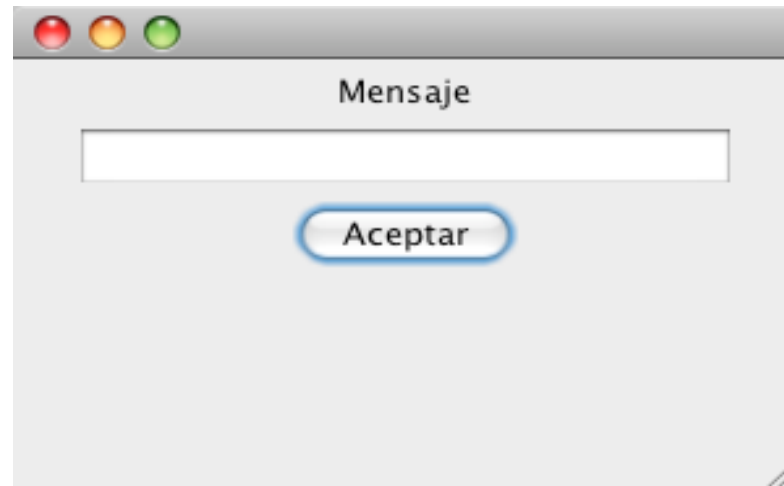
// 6) Hacer visible la ventana

```
setVisible(true);
```

```
}
```

```
}
```

Otro Ejemplo de Interfaz



- Si cerramos la ventana (botón rojo) el programa no finaliza, sólo se destruye la ventana (propiedad `JFrame.DISPOSE_ON_CLOSE`)
 - Para salir del programa: `JFrame.EXIT_ON_CLOSE`
- El botón de “Aceptar”, sin embargo, no hace nada
 - ¡¡¡**NO** hay un **Evento** asociado!!!

Índice

- Introducción
- **Programando con Swing**
 - Introducción y Jerarquía de Swing
 - **Gestión de Eventos: Events, Listeners y Adapters**
 - Posicionamiento de los componentes: Layouts
 - Interfaz gráfica para el desarrollo de GUIs
 - Controles Swing

Eventos en Java

- Todo componente de la GUI puede generar eventos
 - Acciones del usuario sobre el componente
 - Temporizaciones
 - Cambios de estado, etc.
- Modelo de Delegación
 - La responsabilidad de gestionar un evento que sucede en un componente (**Fuente**) la tiene otro objeto (**Oyente**)
- Propagación de Eventos
 - El componente *Fuente* del evento invoca a un método del objeto *Oyente*, pasándole como un argumento un objeto que almacena toda la información relativa al evento

Fuentes y Oyentes (Sources y Listeners)

- Fuente (Source)
 - Objeto que origina o lanza eventos
 - En su API se definen los eventos que puede lanzar
 - Tiene interfaz para registrar Oyentes para cada tipo de evento
 - Un único oyente: `set<TipoEvento>Listener`
 - Varios oyentes: `add<TipoEvento>Listener`
 - Eliminar un oyente: `remove<TipoEvento>Listener`
- Oyente (Listener)
 - Objeto que gestiona o responde a eventos
 - Define uno o más métodos a ser invocados por la Fuente de eventos en respuesta a cada evento específico
 - Objeto que implementa la interfaz `<TipoEvento>Listener`

Jerarquía de Eventos

- Los eventos se representan dentro de una jerarquía de clases
 - Una subclase para cada evento o tipo de eventos relacionados
 - Encapsulan toda la información del evento
 - Tipo del evento: *getID()*
 - Objeto Fuente sobre el que se produjo el evento: *getSource()*
 - Posición (x,y) donde se produjo el evento (eventos de ratón): *getPoint()*
 - Tecla que se pulsó (eventos de teclado): *getKeyChar()* y *getKeyCode()*
 - Estado de las teclas modificadoras (CTRL, SHIFT, etc.): *getModifiers()*
 - etc.
 - Raíz de la jerarquía de clases: **java.util.EventObject**
 - Se pueden crear nuevos eventos a partir de `java.util.EventObject` o de **java.awt.AWTEvent**

Eventos: bajo nivel y semánticos

- Eventos de bajo nivel
 - Representan entradas o interacciones de bajo nivel con la GUI
 - Cambio de tamaño de un componente, cambio de foco, etc.
- Eventos semánticos
 - Eventos de alto nivel que encapsulan la semántica del modelo del componente
 - Hacer una acción, un cambio en el estado de un componente, etc.
 - No están asociados a una clase particular de componente, sino a cualquiera que implemente un modelo semántico similar
 - Evento “Action” lanzado por:
 - Un botón pulsado una vez
 - Un elemento de una lista pulsado dos veces seguidas
 - Cuando se pulsa enter en una caja de texto, etc.

Eventos: bajo nivel y semánticos

Eventos de bajo nivel	
Evento	Significado
ComponentEvent	Cambio en el tamaño, posición o visibilidad de un componente
FocusEvent	Cambio de foco (capacidad de un componente de recibir entrada desde el teclado)
KeyEvent	Operación con el teclado
MouseEvent	Operación con los botones del ratón o movimientos del ratón
WindowEvent	Cambio de estado en una ventana
AncestorEvent	Cambio en la composición, visibilidad o posición de un elemento superior (ancestro) de la jerarquía de composición
Eventos de alto nivel	
ActionEvent	Realización de la acción específica asociada al componente
ChangeEvent	Cambio en el estado del componente
ItemEvent	Elemento seleccionado o deseleccionado
CaretEvent	Cambio en la posición del cursor de inserción en un componente que gestiona texto
ListSelectionEvent	Cambio en la selección actual en una lista

Oyentes (Listeners) de Eventos

- Son objetos que implementan la interfaz correspondiente al tipo de evento a escuchar:
 - ActionListener
 - WindowListener
 - MouseListener
 - etc.
- Las clases que extienden estos interfaces tienen que implementar **TODOS** los métodos de la interfaz
 - ¿Qué pasa si sólo nos interesa uno de los eventos?
 - Muchos métodos vacíos
 - Poca legibilidad
 - Solución: **Adaptadores**

Adaptadores (Adapters) de Eventos

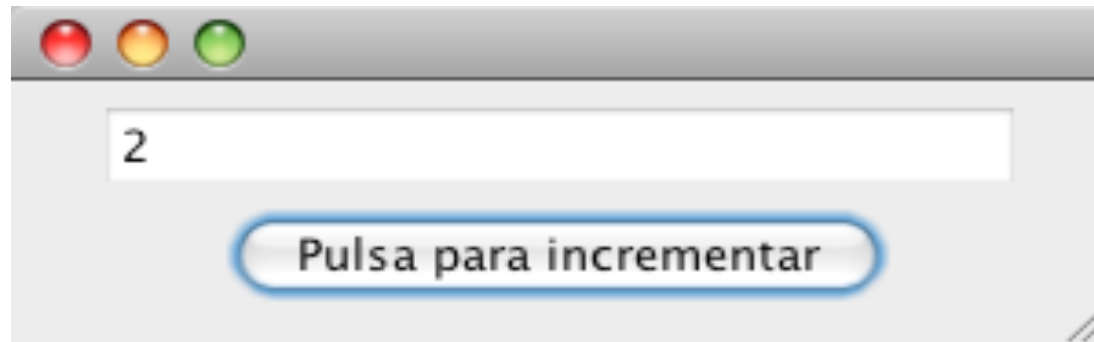
- La API nos proporciona implementaciones por defecto de los Listeners
 - En realidad, nos proporciona implementaciones vacías de todos los métodos relacionados con un tipo de eventos
 - Sólo tenemos que extender de una clase adaptadora y reescribir el método deseado

Resumen del funcionamiento de los eventos

- Los objetos que desean gestionar los eventos y recibir dichas notificaciones tienen que registrarse como *Oyentes* e implementar los métodos de la interfaz correspondiente
- Cuando ocurre un evento la *Fuente* informa a los *Oyentes* registrados invocando a dichos métodos (**callback**)
- En un programa Swing, normalmente la *Fuente* de eventos es un componente GUI y el *Oyente* es un objeto “adaptador” que implementa el o los *Oyentes* adecuados para que la aplicación gestione los eventos
- El *Oyente* también puede ser otro componente Swing que implementa uno o más interfaces *Oyentes* para agrupar objetos del GUI

Ejemplo Práctico

- Programa que permite incrementar un número cada vez que se pulse un botón
- Inicialmente el cuadro de texto contiene un “0”



Ejemplo Práctico

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

```
public class EjemploListeners extends JFrame {
    public EjemploListeners () {
        initComponents();
    }
    private void initComponents() {
        setSize(300, 100);
        text = new JTextField("0", 20);
        text.setEditable(false);
        button = new JButton("Pulsa para incrementar");
        button.addMouseListener(new MyClickListener());
        panel = new JPanel();
        panel.add(text);
        panel.add(button);
        setContentPane(panel);
    }
    private class MyClickListener extends MouseAdapter {
        public void mouseClicked(MouseEvent event) {
            numClicks++;
            text.setText(String.valueOf(numClicks));
        }
        private int numClicks;
    }
    private JPanel panel;
    private JTextField text;
    private JButton button;
}
```

Creamos la caja de texto, con un "0" al principio y hacemos que no sea editable

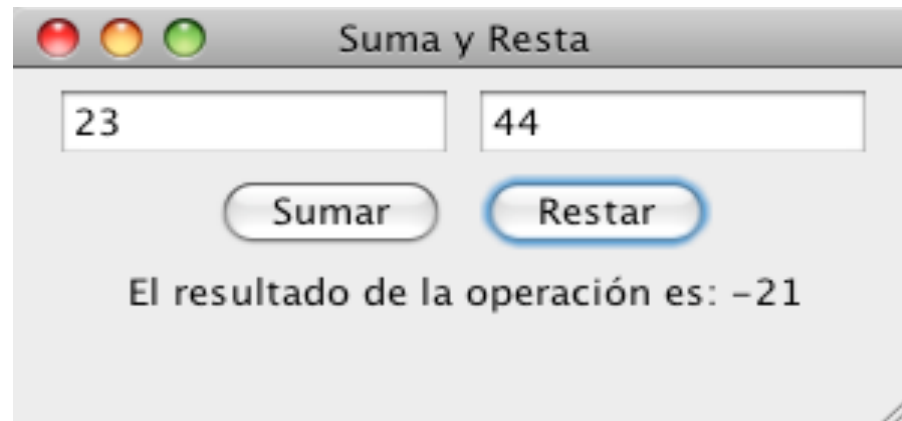
Creamos el botón y le añadimos es Listener (en este caso sólo nos fijamos en los clicks)

Creamos un panel y añadimos tanto el botón como la caja de texto

Extendemos la clase MouseAdapter y redefinimos el método mouseClicked para que se incremente el número de clicks y se modifique el contenido de la caja de texto

Ejercicio: Suma y Resta

- Crear un programa que permita sumar o restar dos números
- Añadir dos cajas de texto, una para cada número
- Añadir dos botones, uno para sumar y otro para restar
- Añadir una etiqueta para mostrar el resultado
- Cada botón tendrá su propio Listener para “escuchar” el click



Ejercicio 2: : Suma y Resta (2)

- Extender el ejercicio anterior para que tanto el botón de sumar como el de restar compartan el mismo Listener
- Para ello, en el método *mouseClicked* deberemos comprobar qué botón generó el evento
- En función de qué botón fue pulsado, realizar una operación u otra

Ejercicio 3: Conversor Euros a Pesetas

- Construir una ventana de 300x400 píxeles con el título “Conversor de Euros a Pesetas”
- Al cerrar la ventana, la ejecución del programa finalizará
- La ventana debe mostrar una etiqueta con el texto: “Importe en Euros” y un cuadro de texto donde introducir el importe
- Aparecerá una etiqueta con el texto “Pulse el botón para convertir el importe en pesetas” y un botón con el texto “Convertir”
- Al pulsar el botón, se deberá calcular el valor del importe en pesetas y mostrar el nuevo valor en una tercera etiqueta

Eventos de la clase JComponent

Tipo de Eventos	Evento
Eventos del Foco (Focus)	Foco obtenido (<i>focusGained</i>) Foco perdido (<i>focusLost</i>)
Eventos de entrada de teclado (Key)	Tecla presionada (<i>keyPressed</i>) Tecla soltada (<i>keyReleased</i>) Tecla presionada y soltada (<i>keyTyped</i>)
Eventos de ratón (Mouse)	Tecla del ratón presionada y soltada (<i>mouseClicked</i>) El ratón entra en un componente (<i>mouseEntered</i>) El ratón sale de un componente (<i>mouseExited</i>) Tecla del ratón presionada (<i>mousePressed</i>) Tecla del ratón soltada (<i>mouseReleased</i>)
Eventos de movimiento del ratón (MouseMotion)	El ratón se ha movido mientras una tecla se encuentra pulsada. El componente está siendo arrastrado (<i>mouseDragged</i>) El ratón se ha movido y no se está pulsando ninguna tecla (<i>mouseMoved</i>)
Eventos de la rueda del ratón (MouseWheel)	La rueda ha rotado. (<i>mouseWheelMoved</i>)

Eventos de la clase AbstractButton (JButton)

Tipo de eventos	Evento
Eventos de acción (Action)	Se ha hecho clic en el botón (<i>actionPerformed</i>)
Eventos de cambio (Change)	El estado del botón ha cambiado (<i>stateChanged</i>)
Eventos de selección de un item (Item)	Ha cambiado el estado de selección del botón (<i>itemStateChanged</i>)

Eventos de la clase JComboBox

Tipo de eventos	Evento
Eventos de acción (Action)	Se ha seleccionado un elemento. Si se puede escribir en el cuadro de texto se genera este evento cuando se pulsa la tecla ENTER (<i>actionPerformed</i>)
Eventos de selección de un item (Item)	Ha cambiado la selección de la lista desplegable (<i>itemStateChanged</i>) (Se disparan dos eventos)

Eventos de la clase Window (JFrame)

Tipo de eventos	Evento
Eventos de foco en la ventana (WindowFocus)	Foco obtenido por la ventana (<i>windowGainedFocus</i>) Foco perdido por la ventana (<i>windowLostFocus</i>)
Eventos del ciclo de vida de la ventana (Window)	Ventana activada (<i>windowActivated</i>) La ventana ha sido cerrada como resultado de llamar al método dispose (<i>windowClosed</i>) El usuario ha pulsado el botón de cierre de la ventana (<i>windowClosing</i>) La ventana ya no es la ventana activa (<i>windowDeactivated</i>) La ventana ha pasado de minimizada a normal (<i>windowDeiconified</i>) La ventana se ha minimizado (<i>windowIconified</i>) La ventana se ha hecho visible por primera vez (<i>windowOpened</i>)
Eventos de estado de la ventana (WindowState)	La ventana ha cambiado de estado en cuanto a minimizada, maximizada, restaurada, etc. (<i>WindowStateChanged</i>)

Evento de la clase JTextComponent (JTextField)

Tipo de eventos	Evento
Eventos de entrada de texto (InputMethod)	El cursor ha cambiado de posición (<i>caretPositionChanged</i>) El texto introducido por teclado ha cambiado (<i>inputMethodTextChanged</i>)
Eventos de cursor (Caret)	El cursor ha cambiado de posición (<i>caretUpdate</i>)

- Nota adicional:
 - Los eventos generados en la clase padre de un componente serán generados también en ese componente
 - Por ejemplo, los eventos generados en la clase **java.awt.Component** son generados en cualquier componente

Ejercicio

- Crear una ventana con una etiqueta, un botón y un área de texto
- La etiqueta debe medir 200x300 y tener un color de fondo sólido
- El botón debe llevar el texto “Pincha aquí”
- La ventana tiene que ser suficientemente grande para albergar los tres controles
- Hay que crear Listeners para los eventos de Entrada y Salida de la etiqueta y el Action sobre el botón
- Cada uno de esos Listeners añadirá una línea al área de texto diciendo qué evento ha sucedido

Ejercicio (Pistas)

- Métodos para fijar el tamaño
 - `setMinimumSize()` → `Dimension(width, height)`
 - `setMaximumSize()`
 - `setPreferredSize()`
- Métodos para darle color a la etiqueta y hacerla opaca
 - `setOpaque(true)`
 - `setBackground()`-----→ `Color.NombreColor`
- Control para el área de texto: `JTextArea`
 - `JTextArea (fils, cols)`
 - `textArea.append("Texto")`

Ejercicio

- Escriba un programa que muestre una ventana con un único botón de Salir
- Se deben capturar los eventos de Cerrar Ventana y Click en el botón Salir
 - Pista: **setDefaultCloseOperation**
(JFrame.DO_NOTHING_ON_CLOSE)
- Ambos eventos deben compartir el mismo código de tratamiento
- Lo que se debe hacer es mostrar una segunda ventana con dos botones, cuyo título sea “¿Seguro que desea salir?”
 - El botón de Aceptar, sale del programa: `System.exit(0)`
 - El botón de Cancelar cierra esta ventana y vuelve a la ventana principal
 - Método para cerrar una ventana: `dispose()`

Otras maneras de implementar los Oyentes

- Implementando directamente los métodos en nuestra clase

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

public class EjemploListeners extends JFrame implements ActionListener {
    public EjemploListeners () {
        initComponents();
    }
    private void initComponents() {
        setSize(300, 100);
        text = new JTextField("0", 20);
        text.setEditable(false);
        button = new JButton("Pulsa para incrementar");
        button.addActionListener(this);
        panel = new JPanel();
        panel.add(text);
        panel.add(button);
        add(panel);
    }
    public void actionPerformed(ActionEvent event) {
        numClicks++;
        text.setText(String.valueOf(numClicks));
    }
    private int numClicks;
    private JPanel panel;
    private JTextField text;
    private JButton button;
}
```

Otras maneras de implementar los Oyentes

- Implementando clases adaptadoras anónimas

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

public class EjemploListeners extends JFrame {
    public EjemploListeners () {
        initComponents();
    }
    private void initComponents() {
        setSize(300, 100);
        text = new JTextField("0", 20);
        text.setEditable(false);
        button = new JButton("Pulsa para incrementar");
        button.addMouseListener(
            new MouseAdapter () {
                public void mouseClicked(MouseEvent event) {
                    numClicks++;
                    text.setText(String.valueOf(numClicks));
                }
                private int numClicks;
            });
        panel = new JPanel();
        panel.add(text);
        panel.add(button);
        add(panel);
    }
    private JPanel panel;
    private JTextField text;
    private JButton button;
}
```

Ejercicios

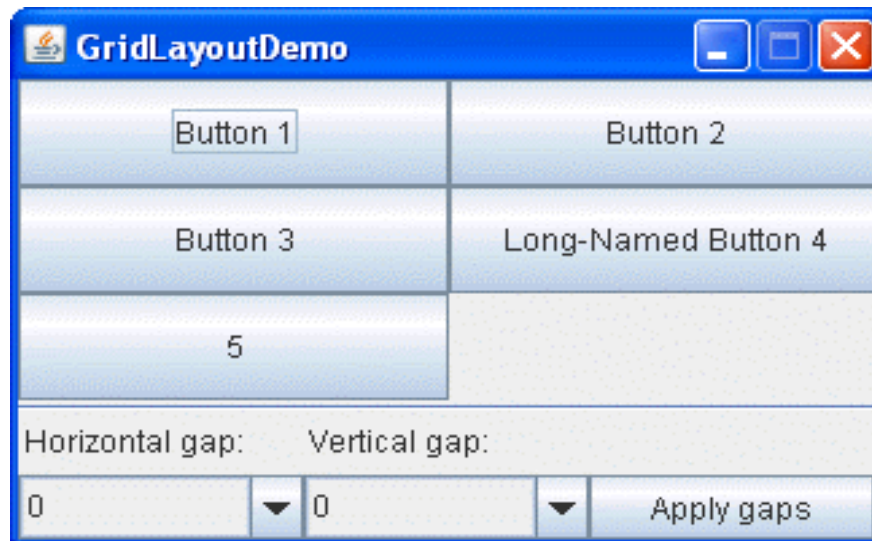
- Rehacer el ejercicio del conversor de euros con
 - El método del Listener implementado directamente en nuestra clase ventana
 - Creando una clase adaptadora anónima para gestionar los eventos
 - Añadir un botón de borrado que limpie la caja de texto
 - Pista: `setText("")`
 - Añadir un botón que nos permita salir de la aplicación

Índice

- Introducción
- **Programando con Swing**
 - Introducción y Jerarquía de Swing
 - Gestión de Eventos: Events, Listeners y Adapters
 - **Posicionamiento de los componentes: Layouts**
 - Interfaz gráfica para el desarrollo de GUIs
 - Controles Swing

Posicionamiento de los componentes: Layouts

- Los *Layouts* nos permiten distribuir los controles de distintas maneras en un contenedor
- Cada contenedor puede tener su Layout:
 - Por ejemplo, una ventana puede tener un Layout, contener dos JPanels dispuestos según ese Layout y, luego, cada JPanel podría tener su propio Layout



Tipos de Layout

- BorderLayout

- BoxLayout

- CardLayout

- FlowLayout

- GridLayout

- GridBagLayout

- GroupLayout

- SpringLayout

Se pueden usar “a mano”

A medio camino

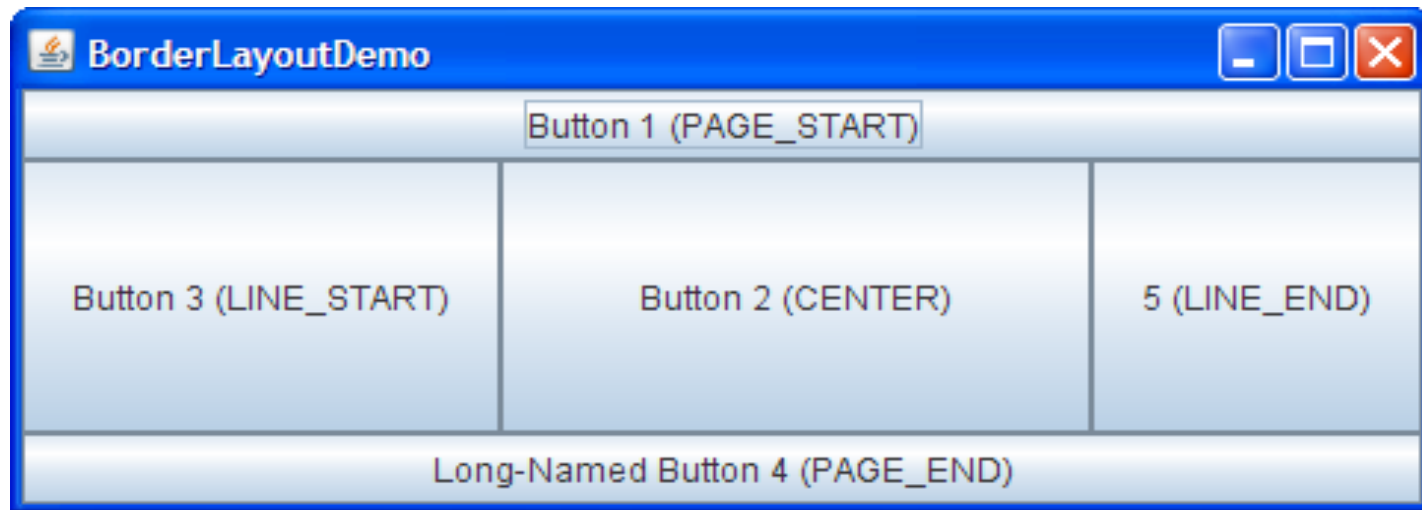
Mejor usar con un IDE

BorderLayout

- Los paneles de contenido de los JFrame, por defecto, están inicializados con este tipo de Layout
- Permite colocar componentes (simples o contenedores) en cinco posiciones: Arriba, Abajo, Izquierda, Derecha y Centro
- Las posiciones se indican al añadir el componente al contenedor (mediante el método add que ya hemos visto)
 - Arriba: PAGE_START o NORTH
 - Abajo: PAGE_END o SOUTH
 - Izquierda: LINE_START o WEST
 - Derecha: LINE_END o EAST
 - Centro: CENTER

BorderLayout

- Funciones importantes del API:
 - BorderLayout(int horizontalGap, int verticalGap)
 - setHgap (int)
 - setVgap (int)



BorderLayout

// Creamos el layout con gap de 5 píxeles en ambos lados

```
BorderLayout layout = new BorderLayout( 5, 5 );
```

// Asociamos el layout al Frame actual

```
setLayout( layout );
```

// Añadimos el botón al Frame, en la posición Norte

```
JButton button = new JButton("NORTH");
```

```
add( button, BorderLayout.NORTH );
```

Ejercicio

- Crear la interfaz que se observa dos transparencias más atrás usando BorderLayout para colocar los botones
- Al pulsar un botón, ese botón debe ocultarse (necesitaremos un Listener que controle la pulsación de los botones)
 - El resto de botones ocuparán el espacio disponible automáticamente
- Si había otro botón oculto, debe volver a mostrarse (sólo debe haber un botón oculto cada vez)
 - Consejo: usar un array de JButtons y el mismo Listener para todos

BoxLayout

- Este layout nos permite dos tipos de disposición de los componentes:
 - Unos encima de otros (sólo un componente por fila)
 - Unos al lado de los otros (todos en la misma fila)
- En el constructor del Layout se le asocia con el contenedor
 - También se indica sobre qué eje se va a hacer la ordenación
 - Horizontal: `BoxLayout.X_AXIS`
 - Vertical: `BoxLayout.Y_AXIS`
- Los componentes se crean y añaden normalmente
 - Podemos decidir cómo se alinearán entre ellos mediante un valor real que medirá el desplazamiento con respecto a los bordes. Hay varias constantes predefinidas: `TOP_ALIGNMENT`, `BOTTOM_ALIGNMENT`, `LEFT_ALIGNMENT`, `RIGHT_ALIGNMENT` y `CENTER_ALIGNMENT`

BoxLayout

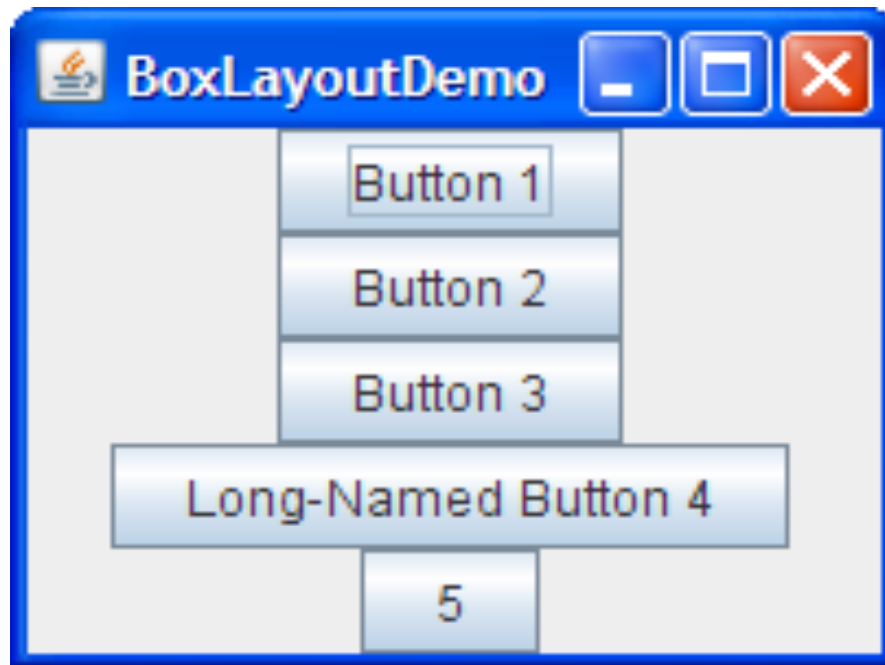
```
// Creamos una ventana y recuperamos su contenedor
JFrame frame = new JFrame("BoxLayoutDemo");
// Alternativa a crearnos nuestro JPanel, usar el contenedor por defecto del JFrame
//JPanel container = new JPanel();
Container container = frame.getContentPane();

// Asociamos al contenedor un BoxLayout vertical
container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));

// Creamos un botón, decimos que lo alinee por el medio y lo añadimos al
// contenedor
JButton button = new JButton(text);
button.setAlignmentX(Component.CENTER_ALIGNMENT);
container.add(button);
```

BoxLayout

- Éste sería un ejemplo de BoxLayout

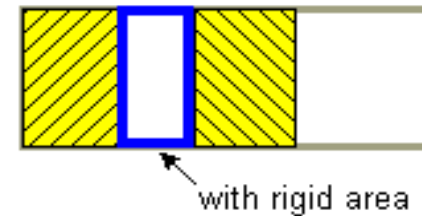
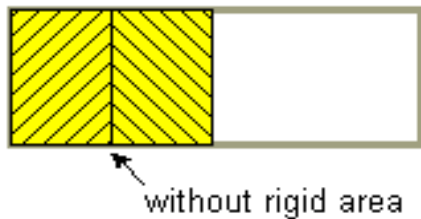


Peligros de los Layouts

- Tienden a ocupar el máximo espacio disponible
 - Intentan “rellenar” todo el espacio posible...
 - ... respetando las restricciones de los componentes (a veces)
 - `setMinimumSize(Dimension d)` – Tamaño mínimo (horizontal y vertical)
 - `setPreferredSize(Dimension d)` – Tamaño preferido (horizontal y vertical)
 - `setMaximumSize(Dimension d)` – Tamaño máximo (horizontal y vertical)
- Muchos Layouts ignoran el tamaño máximo (p.ej.: `BorderLayout`)
- Sólo se usa el tamaño preferido cuando el resto del espacio está “lleno”
 - Por otros controles o por “rellenadores” transparentes

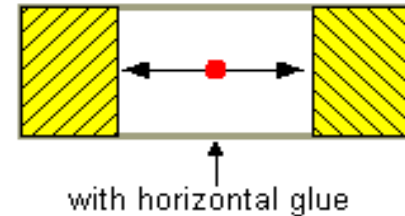
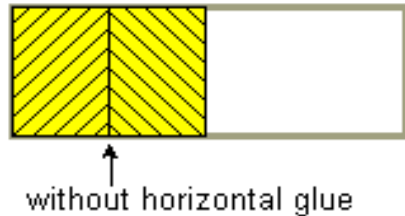
Rellenadores invisibles (Invisible Fillers)

- Creados a partir de la clase Box (**siempre con ¡BoxLayout!!**)
 - Espacio fijo entre componentes: Rigid Area
 - `Box.createRigidArea(Dimension size)`



- Espacio ajustable: Glue

- `Box.createHorizontalGlue()` y `Box.createVerticalGlue()`

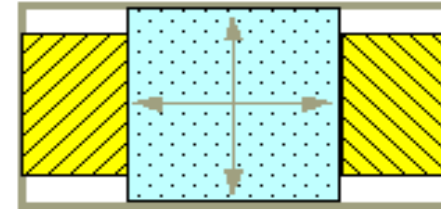


Rellenadores invisibles (Invisible Fillers)

- Espacio Fijo con valores mínimo, preferido y máximo: Custom Box.Filler
 - `Box.Filler(Dimension minSize, Dimension prefSize, Dimension maxSize)`



without a custom filler

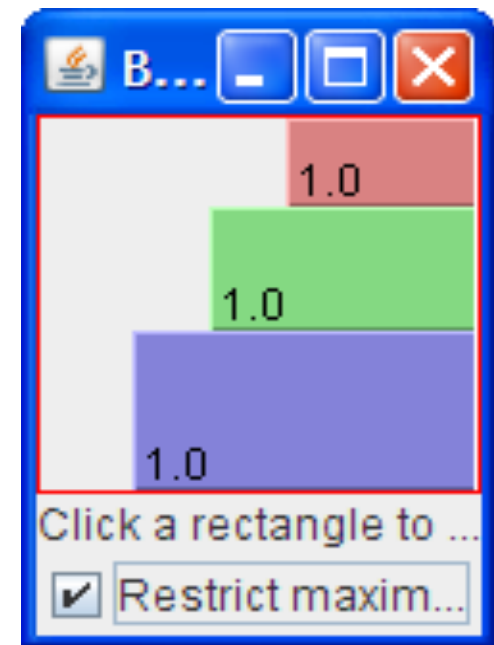
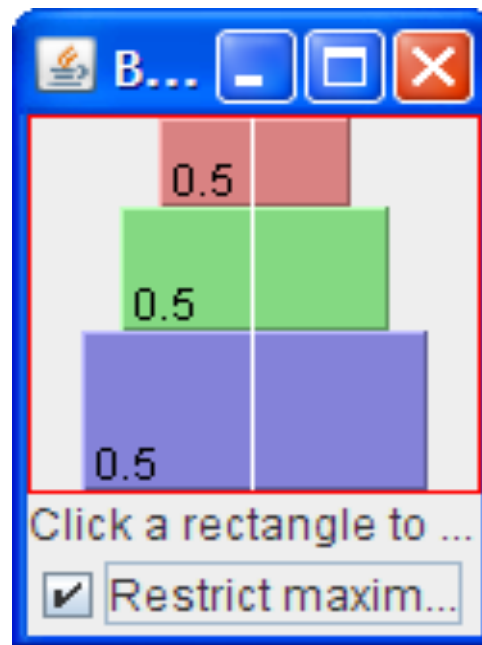
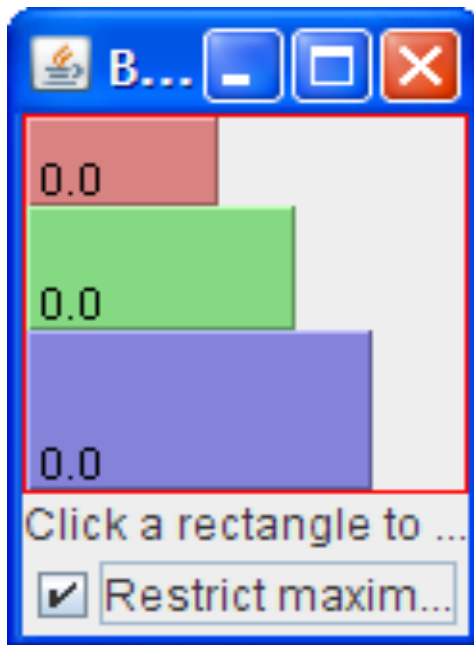


with a custom filler

- Si no queremos darle al JFrame un tamaño al crearlo
 - invocar al método de empaquetado: **`frame.pack()`**

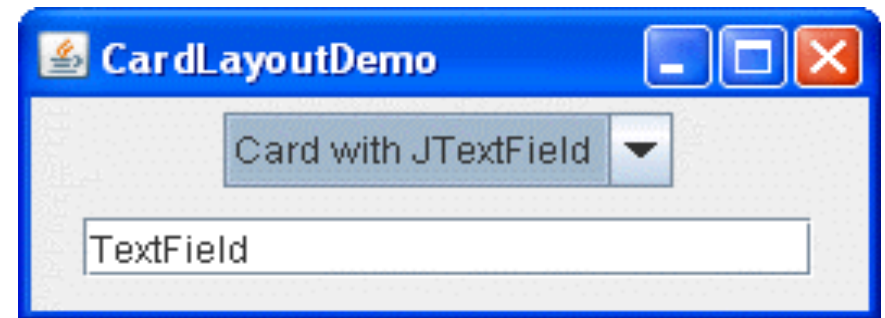
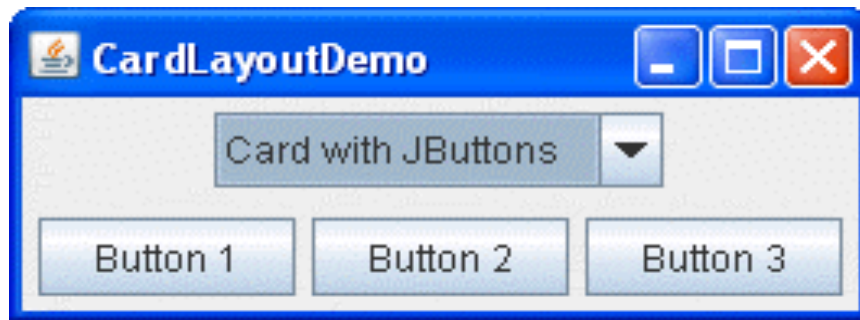
Ejercicio

- Crear una ventana y asociarle un BorderLayout
- Añadir tres etiquetas opacas de distintos colores y tamaños (*setMaximumSize*)
- Conseguir los efectos de las tres imágenes de abajo



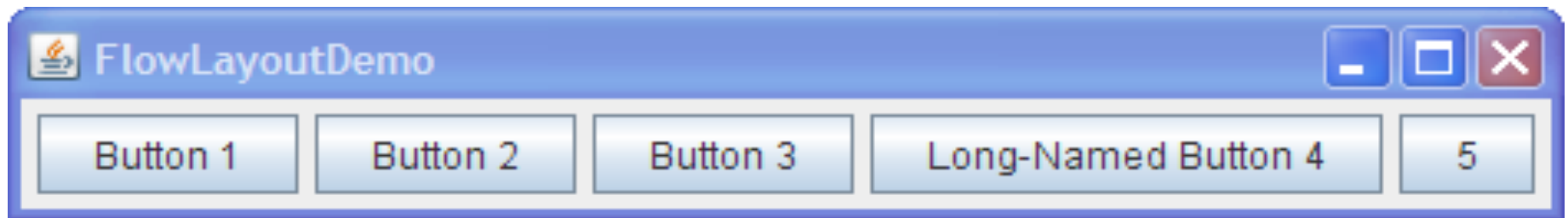
CardLayout

- Permite definir un área donde mostrar distintos componentes en distintos momentos
 - Un efecto parecido se consigue con un JTabbedPane



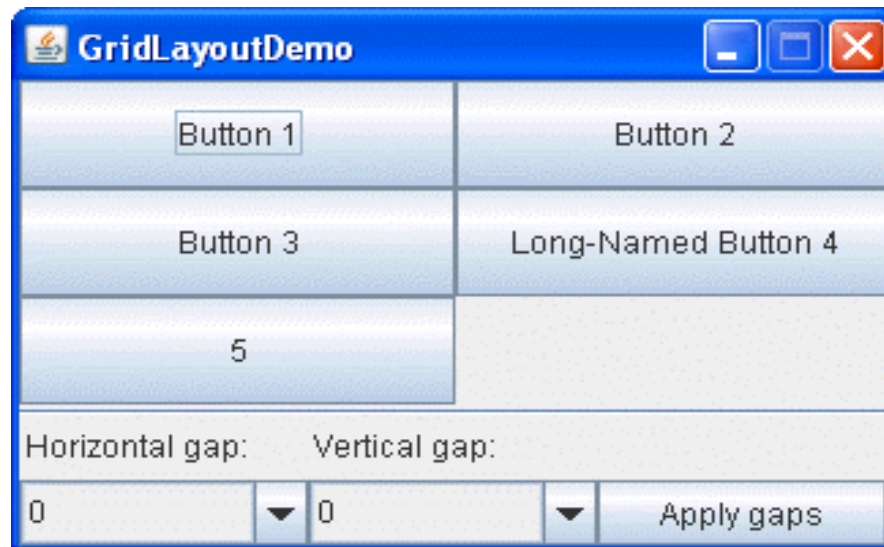
FlowLayout

- Es el layout por defecto de JPanel
 - Lo hemos estado usando hasta ahora sin saberlo...
- Simplemente, muestra los componentes en una única fila
 - Comienza una nueva fila si el espacio en la anterior no es suficiente



GridLayout

- Distribuye un número determinado de componentes en una matriz de m filas y n columnas (el número de filas es opcional)
- Al construir el objeto podemos seleccionar el número de filas y de columnas, así como el espacio horizontal y vertical entre los componentes
 - `GridLayout(int rows, int cols, int hgap, int vgap)`



GridLayout

```
// Creamos un panel
JPanel compsToExperiment = new JPanel();

// Creamos un GridLayout de 2 columnas y tantas filas
// como sean necesarias
GridLayout experimentLayout = new GridLayout(0,2);

// Asociamos el layout al panel
compsToExperiment.setLayout(experimentLayout);

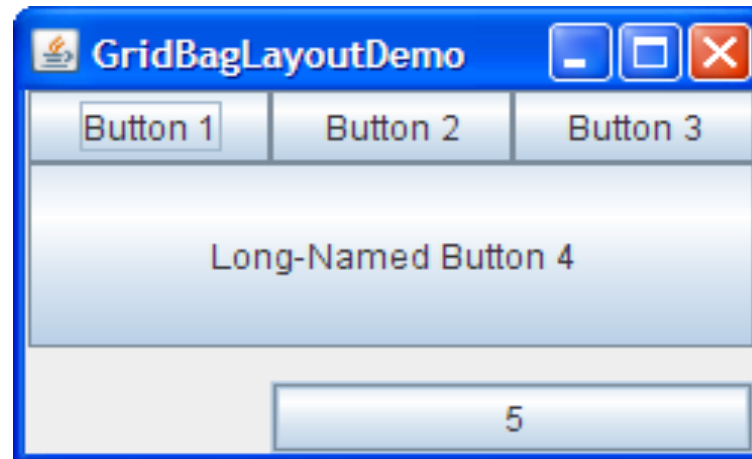
// Añadimos el botón al panel normalmente
compsToExperiment.add(new JButton("Button 1"));
```

Ejercicio

- Crear una ventana con la interfaz de dos transparencias atrás
- El contenedor principal tiene que tener un layout de tipo Border
 - En la parte superior crearemos un panel con layout de tipo Grid
 - En la parte central añadiremos un separador (JSeparator)
 - En la parte inferior crearemos un panel con layout de tipo Grid (Nota: el espacio de la derecha de las etiquetas lo conseguimos con una etiqueta vacía)
- En lugar de JComboBox utilizaremos JTextField para introducir los valores de Gap
- El botón de la derecha, al ser pulsado, debe modificar el Gap del GridLayout superior con los valores de las cajas de texto

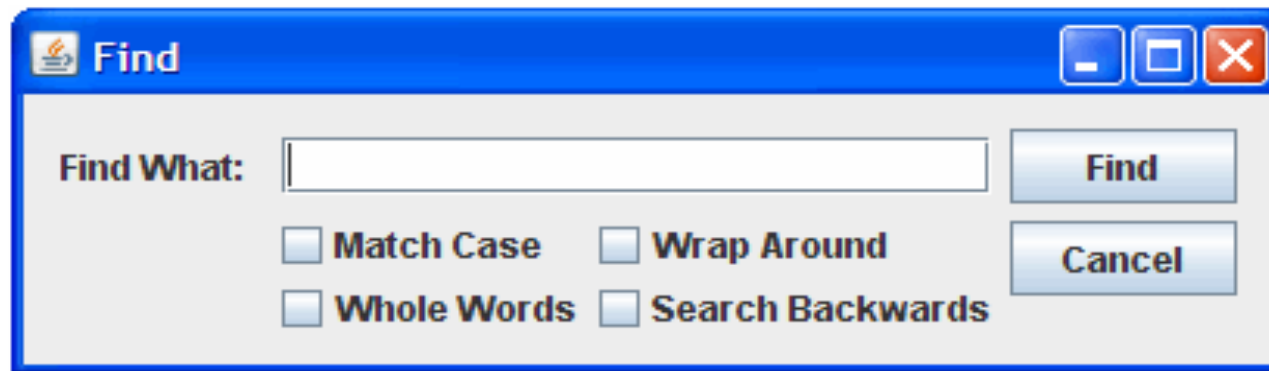
GridBagLayout

- Permite situar los controles en una rejilla (igual que el anterior). Sin embargo...
 - No todas las filas/columnas tienen la misma altura/anchura
 - Un mismo componente puede expandirse varias filas/columnas
 - Podemos añadir restricciones sobre lo que puede un componente crecer



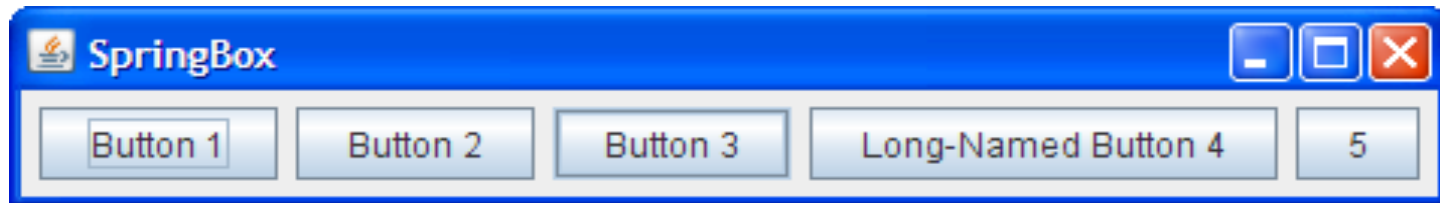
GroupLayout

- Layout muy complejo pensado para ser usado junto con diseñadores gráficos de interfaces
- Maneja los layouts horizontal y vertical por separado
 - La posición de cada componente tiene que ser especificada por duplicado



SpringLayout

- Layout muy complejo pensado para ser usado junto con diseñadores gráficos de interfaces
- Permite definir relaciones precisas entre los bordes de cada par de componentes (distancia entre bordes, por ejemplo)

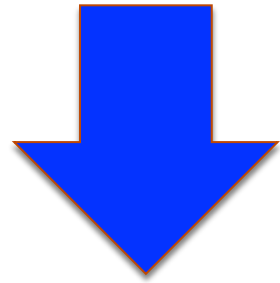


Índice

- Introducción
- **Programando con Swing**
 - Introducción y Jerarquía de Swing
 - Gestión de Eventos: Events, Listeners y Adapters
 - Posicionamiento de los componentes: Layouts
 - **Interfaz gráfica para el desarrollo de GUIs**
 - Controles Swing

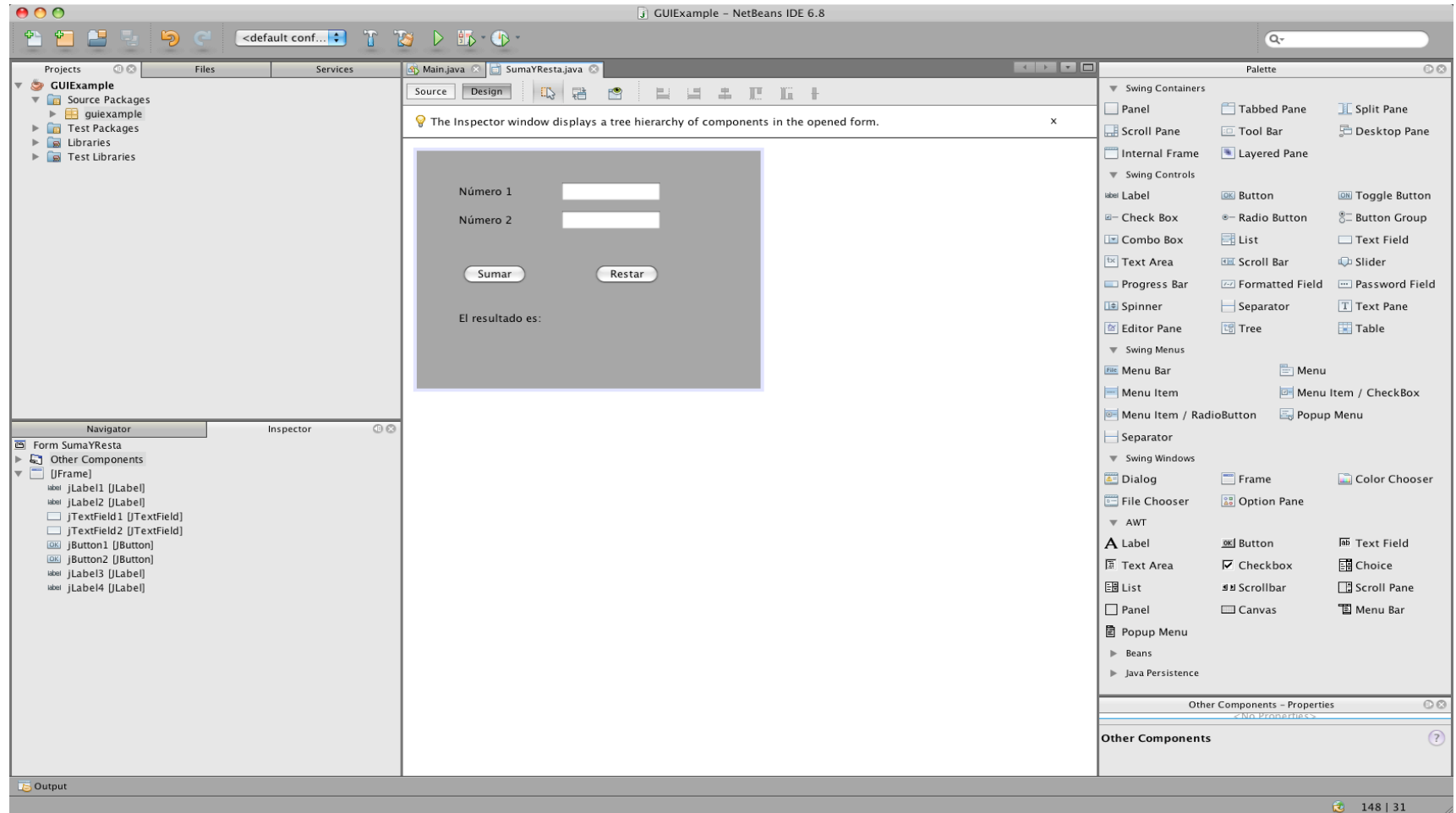
Netbeans GUI Designer

¿No sería maravilloso ahorrarse todo este esfuerzo de programación?



!!!Netbeans GUI Designer!!!

Netbeans GUI Designer



Netbeans GUI Designer

- Podemos crear un nuevo formulario desde el Menú Archivo
- Permite arrastrar los componentes desde la “Paleta” a la ventana que hayamos seleccionado
- Podemos modificar las propiedades de los componentes desde el menú lateral
- **NO** podemos modificar el código que genera automáticamente para pintar los componentes

Índice

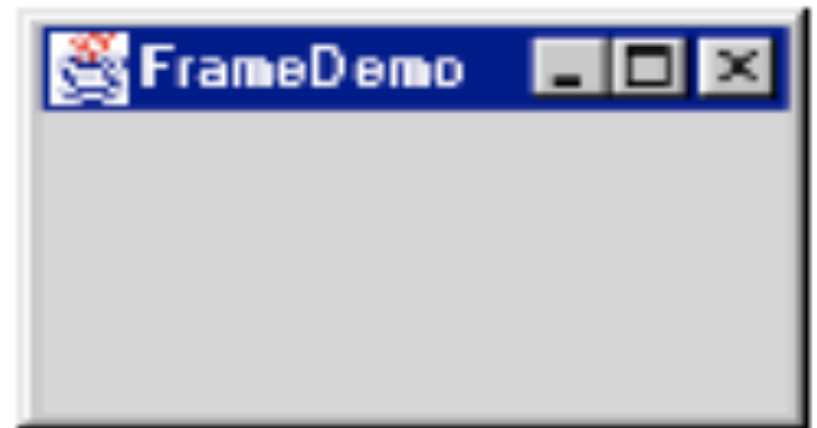
- Introducción
- **Programando con Swing**
 - Introducción y Jerarquía de Swing
 - Gestión de Eventos: Events, Listeners y Adapters
 - Posicionamiento de los componentes: Layouts
 - Interfaz gráfica para el desarrollo de GUIs
 - **Controles Swing**

Clase JFrame

- La clase JFrame proporciona una ventana principal de aplicación con su funcionalidad normal (p.ej.: borde, título, menús) y un **panel de contenido**
- Los contenidos se añaden en el panel de contenidos (content pane) accesible a través del método **getContentPane** (por defecto, un objeto de tipo

JPane, aunque puede cambiarse con **setContentPane**)

- La barra de menú puede fijarse con **setMenuBar**



Clase JFrame

- Algunos métodos y propiedades útiles:
 - `public JFrame()`
 - `public JFrame(String titulo)`
 - `public void setTitle(String titulo)`
 - `public void setSize(int ancho, int alto)`
 - `public void setDefaultCloseOperation(int operacion)`
 - `JFrame.EXIT_ON_CLOSE`
 - `WindowConstants.DO_NOTHING_ON_CLOSE`
 - `WindowConstants.HIDE_ON_CLOSE`
 - `WindowConstants.DISPOSE_ON_CLOSE`
 - `public void pack()`

Clase JFrame

- `public void setResizable(boolean resizable)`
- `public void setExtendedState(int state)`
 `Frame.NORMAL`
 `Frame.ICONIFIED`
 `Frame.MAXIMIZED_BOTH`
 `Frame.MAXIMIZED_HORIZ`
 `Frame.MAXIMIZED_VER`
- `public void setLocation(int x, int y)`
- `public void setLocationRelativeTo(Component c)`
- `public void setVisible(boolean visible)`
- `public void setContentPane(Container c)`
- `public void setJMenuBar(JMenuBar menubar)`

Ejercicio

- Crear una ventana que contenga dos botones
 - El primer botón nos permite mostrar u ocultar una segunda ventana (que se crea al construir la primera pero no se muestra)
 - El segundo botón nos permite mover la segunda ventana (sólo si está visible) valores aleatorios con respecto a la posición actual
- La ventana auxiliar debe:
 - Tener un botón para salir de programa
 - Aparecer siempre centrada después de ocultarla
 - Si se pincha en el icono de cerrar, debe ocultarse únicamente
 - No puede ser cambiada de tamaño

Clase JDialog

- La clase JDialog es la clase raíz de las ventanas secundarias que implementan cuadros de diálogo en Swing
 - Dependen de una ventana principal (normalmente JFrame) y si la ventana principal se cierra, se iconiza o se desiconiza, las ventanas secundarias realizan la misma operación de forma automática.
- Las ventanas modales bloquean la interacción del usuario con otras ventanas
 - Se utilizan sólo cuando hay que garantizar que el usuario recibe un mensaje o proporciona una información que es necesaria.

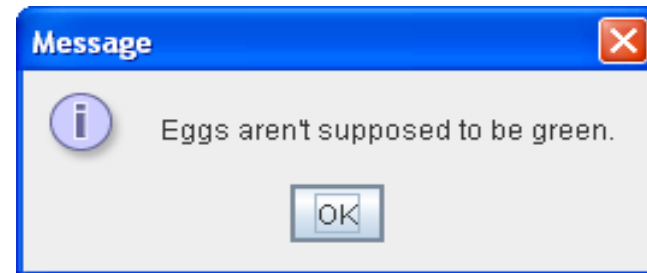
Clase JOptionPane

- Permite crear nuevos cuadros de diálogo o usar algunos de los más comunes:
 - *Message*, para informar al usuario sobre algún hecho relevante
 - *Confirm*, para realizar una pregunta al usuario con las posibilidades básicas de respuesta de Sí, No o Cancelar.
 - *Input*, para solicitar una entrada del usuario
 - *Option*, permite crear una ventana personalizada de cualquiera de los tipos anteriores
- Si no usamos ninguno de éstos, instanciamos un objeto nuevo de la clase JOptionPane y se lo asociamos a un objeto de la clase JDialog

showMessageDialog

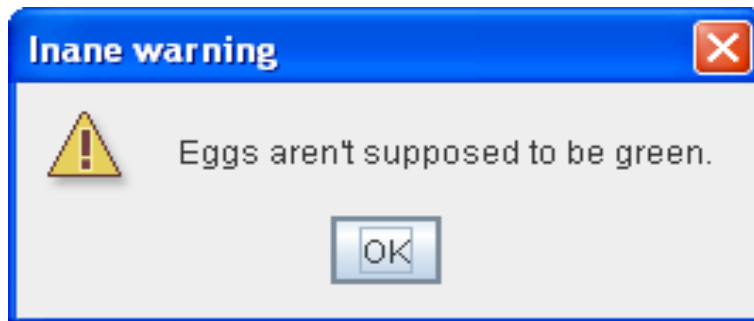
//default title and icon

```
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be  
green.", "Message", JOptionPane.INFORMATION_MESSAGE);
```



//custom title, warning icon

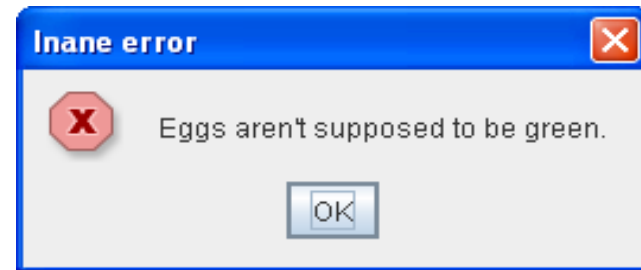
```
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be  
green.", "Inane warning", JOptionPane.WARNING_MESSAGE);
```



showMessageDialog

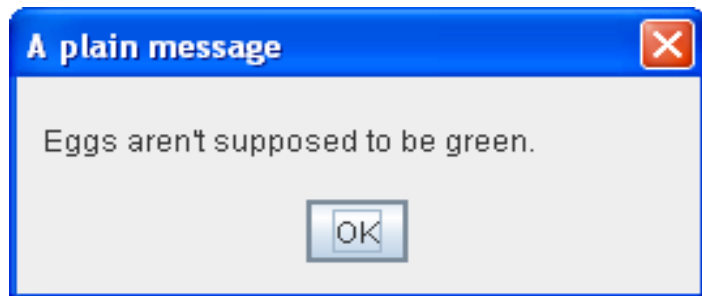
// custom title, error icon

```
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be  
green.", "Inane error", JOptionPane.ERROR_MESSAGE);
```



// custom title, no icon

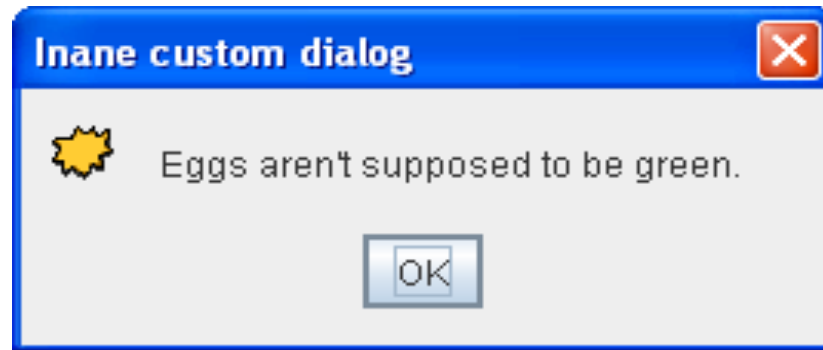
```
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be  
green.", "A plain message", JOptionPane.PLAIN_MESSAGE);
```



showMessageDialog

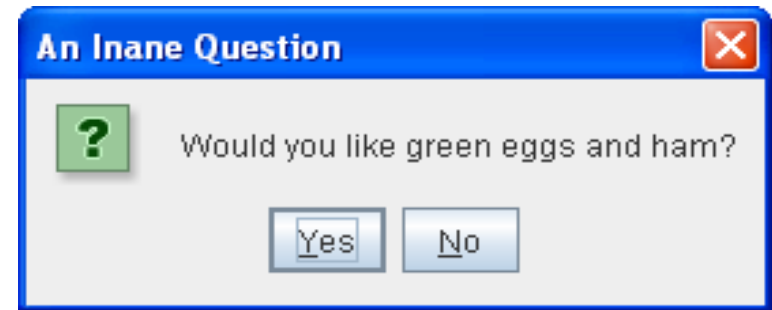
// custom title, custom icon

```
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be  
green.", "Inane custom dialog", JOptionPane.INFORMATION_MESSAGE,  
icon);
```

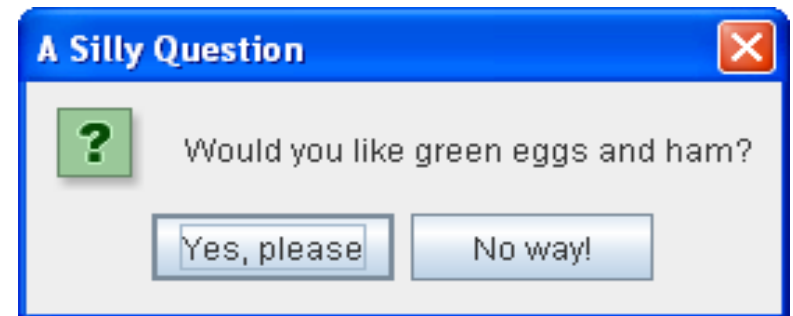


showConfirmDialog

```
// default icon, custom title  
int n = JOptionPane.showConfirmDialog(frame, "Would you like green eggs and  
ham?", "An Inane Question", JOptionPane.YES_NO_OPTION);
```



```
Object[] options = {"Yes, please",  
                    "No way!"};  
int n = JOptionPane.showOptionDialog(frame, "Would you like green eggs  
and ham?", "A Silly Question", JOptionPane.YES_NO_OPTION,  
JOptionPane.QUESTION_MESSAGE,  
null, // do not use a custom Icon  
options, // the titles of buttons  
options[0]); // default button title
```



showInputDialog

```
Object[] possibilities = {"ham", "spam", "yam"};  
String s = (String)JOptionPane.showInputDialog(frame, "Complete  
the sentence:\n" + "\"Green eggs and...\"", "Customized Dialog",  
JOptionPane.PLAIN_MESSAGE, icon, possibilities, "ham");
```

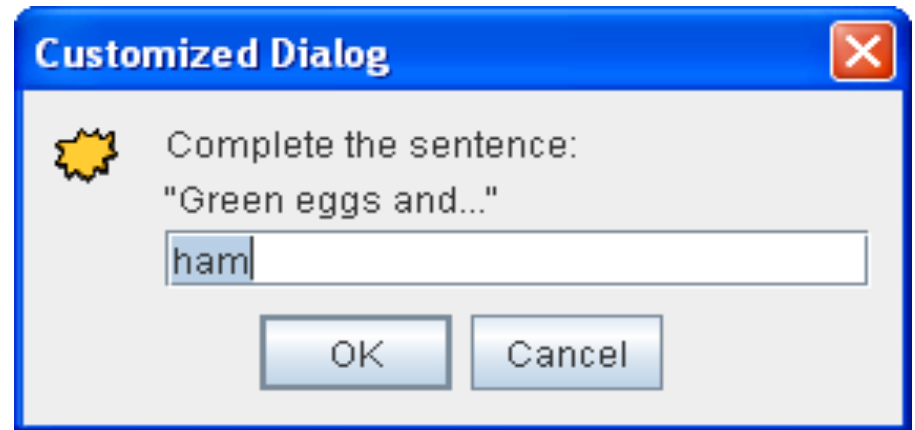
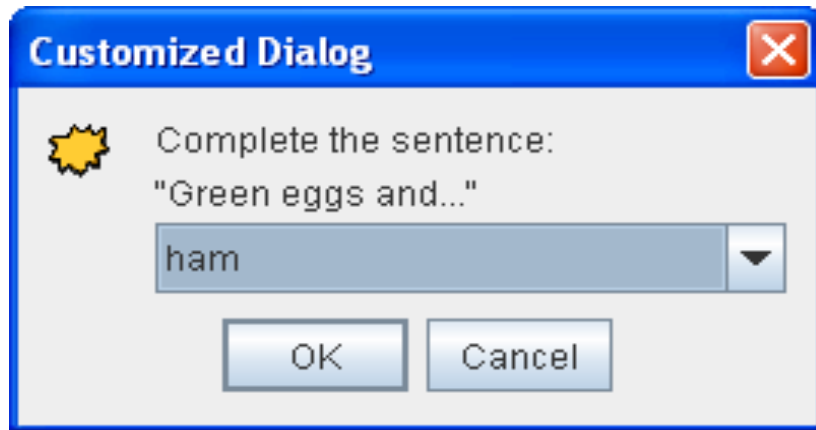
```
//If a string was returned, say so.
```

```
if ((s != null) && (s.length() > 0)) {  
    etiqueta.setText("Green eggs and... " + s + "!");  
    return;  
}
```

```
//If you're here, the return value was null/empty.
```

```
etiqueta.setText ("Come on, finish the sentence!");
```

showInputDialog



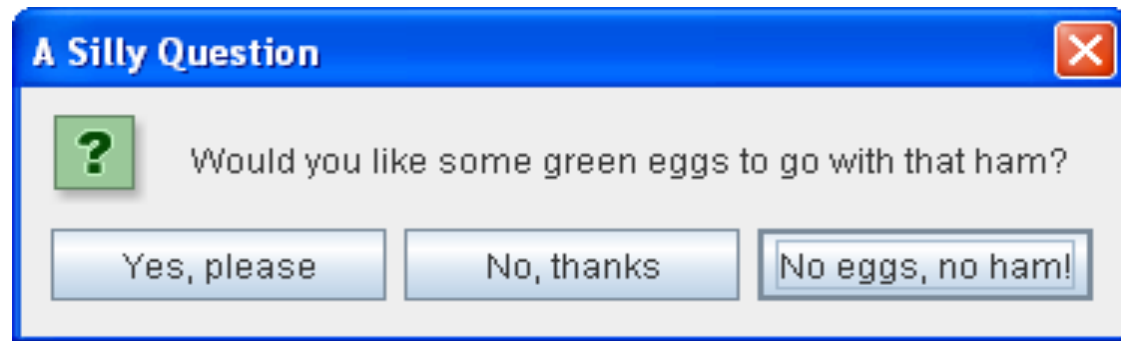
- Si no nos interesa limitar las opciones del usuario y preferimos mostrar una caja de texto, tenemos que pasar “*null*” en lugar de *possibilities*

showOptionDialog

```
// Custom button text
```

```
Object[] options = {"Yes, please", "No, thanks", "No eggs, no ham!"};
```

```
int n = JOptionPane.showOptionDialog(frame, "Would you like some green  
eggs to go " + "with that ham?", "A Silly Question",  
JOptionPane.YES_NO_CANCEL_OPTION,  
JOptionPane.QUESTION_MESSAGE, null, options, options[2]);
```



Valores de retorno

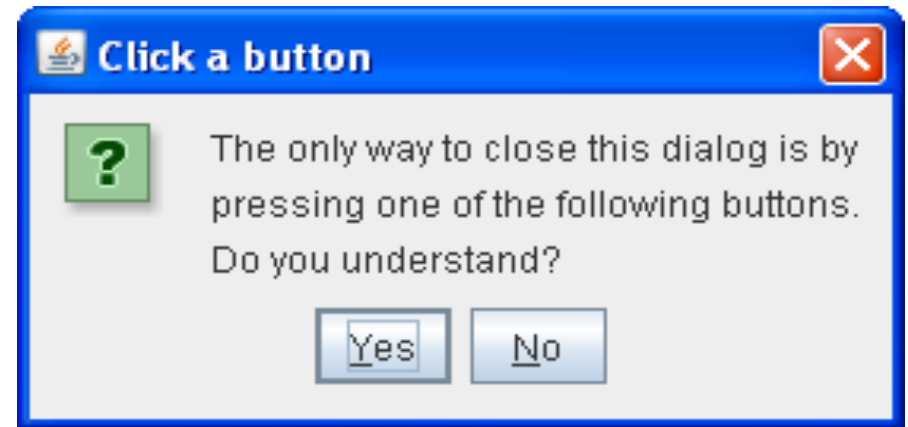
- Si el método `showXXXDialog` devuelve un entero, los posibles valores de retorno son:
 - `JOptionPane.YES_OPTION`
 - `JOptionPane.NO_OPTION`
 - `JOptionPane.CANCEL_OPTION`
 - `JOptionPane.OK_OPTION`
 - `JOptionPane.CLOSED_OPTION`
- En otro caso, devolverá un `String` con la opción seleccionada (como en los ejemplos que hemos visto)

JOptionPane + JDialog

```
JDialog dialog = new JDialog(frame, "Click a button", true);
```

```
JOptionPane optionPane = new JOptionPane("The only way to close this  
dialog is by\n" + "pressing one of the following buttons.\n" + "Do you  
understand?", JOptionPane.QUESTION_MESSAGE,  
JOptionPane.YES_NO_OPTION);
```

```
dialog.setContentPane(optionPane);
```



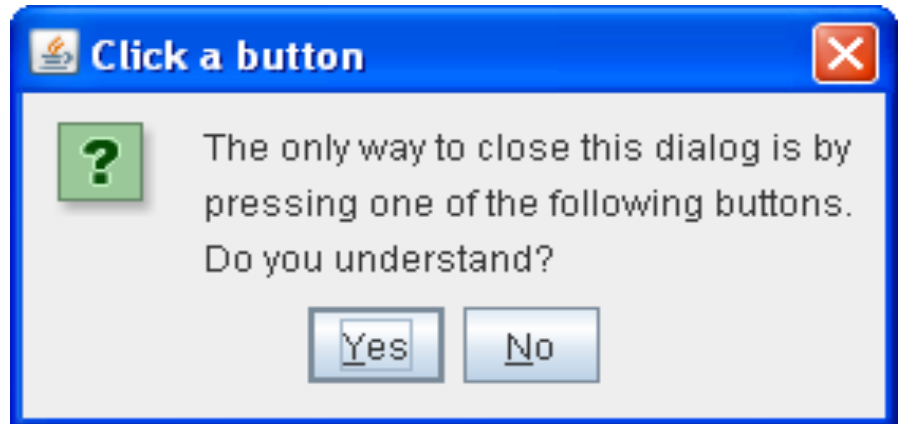
JOptionPane + JDialog

```
public class CreateDialogFromOptionPane {  
    public static void main(final String[] args) {  
        JFrame parent = new JFrame();
```

```
        JOptionPane optionPane = new JOptionPane("The only way to close this  
        dialog is by\n" + "pressing one of the following buttons.\n" + "Do you understand?",  
        JOptionPane.QUESTION_MESSAGE, JOptionPane.YES_NO_OPTION);
```

```
        JDialog dialog = optionPane.createDialog(parent, "Manual Creation");
```

```
        dialog.setVisible(true);  
    }  
}
```

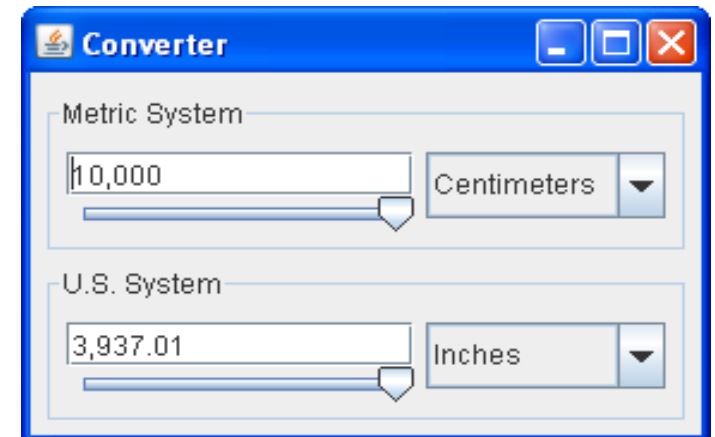


Ejercicio

- Crear una ventana principal (JFrame)
- Al pulsar en el botón de cerrar la ventana, debe aparecer un diálogo de confirmación que nos pregunte si realmente queremos cerrar la ventana. Los textos de los botones deben personalizarse
- Ha de haber un botón que nos muestre un mensaje (sólo con un botón de aceptar)
- Otro botón nos mostrará un diálogo para introducir un texto que deberá mostrarse en una etiqueta de la ventana principal

Clase JPanel

- Contenedor de propósito general donde organizar los elementos de la ventana
- Se puede insertar cualquier tipo de componente en un panel
 - Distintos Layout Managers pueden ser usados en distintos paneles
- El procedimiento de uso habitual consiste en:
 - Construir el panel
 - Añadirle componentes
 - Insertarlo en el contenedor adecuado



Clase JPanel

- Algunos métodos útiles
 - `public JPanel()`
 - `public JPanel(LayoutManager layout)`
 - `public Component add(Component comp)`
 - `public setLayout(LayoutManager l)`
 - `public void remove(Component c)`

Clase JLabel

- Área que permite mostrar un **texto**, una **imagen** o **ambos**
- No es editable, y no puede obtener el foco del teclado
- Se suele utilizar con otros componentes para indicar su finalidad (por ejemplo, con un JTextField para explicar qué texto debe ser introducido)
- Por defecto, su fondo es transparente (por lo que no basta con darle un color de fondo para que éste se vea, hay que hacerlas opacas)



Clase JLabel

- Algunos métodos y propiedades útiles:
 - `public JLabel(String texto)`
 - `public JLabel(Icon icono)`
 - `public JLabel(String text, int alineacionHorizontal)`
 - Tipos de alineado: `JLabel.LEFT`, `JLabel.RIGHT`, `JLabel.CENTER`, `JLabel.LEADING`, `JLabel.TRAILING`
 - `public void setText(String texto)`
 - `public String getText()`
 - `public Icon getIcon()`
 - `public setIcon(Icon icono)`

Clase JTextField

- Campo de texto de **una sola línea**
- Existen especializaciones para tipos concretos de información (que veremos a continuación)



- Suelen usarse junto con un JLabel que indican el texto que debemos introducir en ese campo

Clase JTextField

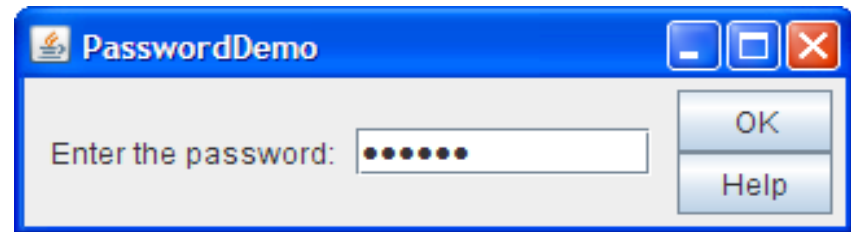
- Algunos métodos útiles
 - `public JTextField(int numCols)`
 - `public JTextField(String texto, int numCols)`
 - `public void setFont(Font fuente)`
 - `public String getText()`
 - `public String getSelectedText()`
 - `public void copy()`
 - `public void cut()`
 - `public void paste()`
 - `public void setEditable(boolean b)`

Ejercicio

- Crear una ventana con dos cajas de texto, una editable, y la otra no
- Habrá tres botones: uno para copiar, otro para cortar y otro para pegar
- Cuando se pulse en uno de los dos primeros botones, se deberá copiar/cortar **únicamente** el texto seleccionado
- Si se pulsa el botón de pegar, el texto copiado/cortado habrá que pegarlo en la caja de texto no editable

Clase JPasswordField

- Podemos usar una especialización de JTextField que oculta los caracteres que vamos tecleando (para meter contraseñas, p.ej.)
- Se puede personalizar el carácter que se muestra para ocultar la información
- La gestión de eventos es la misma que en el caso de JTextField
- Algunos métodos útiles:
 - `public JPasswordField()` y `public JPasswordField(String ini, int ancho)`
 - `public char[] getPassword()`
 - `public void setEchoChar(char oculta)`
 - `public void selectAll()`



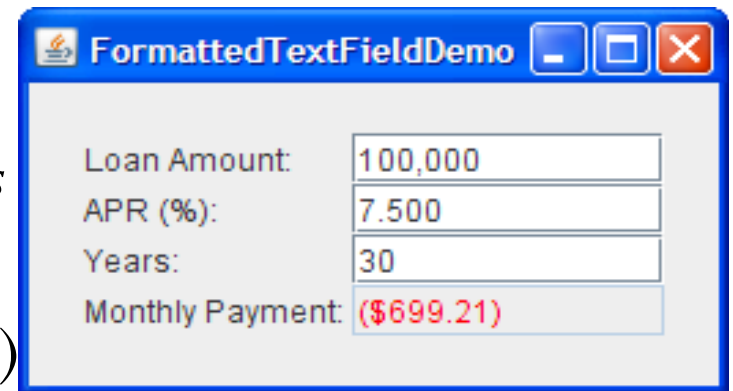
Ejercicio

- Crear una ventana como la de la anterior transparencia
- La clase que representa nuestra ventana tendrá un atributo con una contraseña que nosotros elijamos
- Si se pulsa el botón de OK, se deberá comparar la contraseña con la que tenemos almacenada
- Mostrará un diálogo informando si la contraseña es correcta o no

Clase JFormattedTextField

- Permite introducir y mostrar texto con un formato predefinido
- Hace uso de un objeto *formatter* que permite transformar lo que se ve en el valor que hay por debajo y viceversa
- Podemos usar alguno de los *formatters* por defecto o crear el nuestro propio
- Diferencia entre texto (lo que vemos) y valor (lo que se esconde detrás)
- Más información en:

<http://java.sun.com/docs/books/tutorial/uiswing/components/formattedtextfield.html>

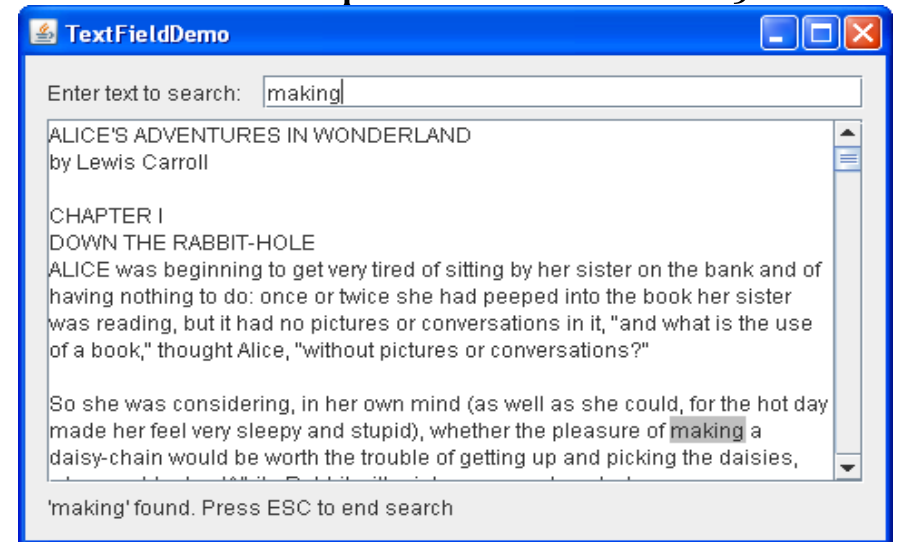


Clase JFormattedTextField

- Algunos métodos útiles
 - `public JFormattedTextField()`
 - `public JFormattedTextField(AbstractFormatter formateador)`
 - `public Object getValue()`
 - `public void setValue(Object valor)`
 - `public void setFocusLostBehavior(int comportamiento)`
 - COMMIT_OR_REVERT, COMMIT, REVERT y PERSIST
 - `public void setCommitsOnValidEdit(boolean comportamiento)`
 - `public boolean isEditValid()`

Clase JTextArea

- Es un campo de texto que admite un número indeterminado de filas y columnas
- Si necesitamos poder hacer *scroll* tenemos que añadir el objeto a un JScrollPane
 - Cuidado, porque si usamos el editor de, p.ej., Netbeans, lo añadirá automáticamente, y podría parecer que no es necesario...
- Se le pueden aplicar los métodos vistos en la clase JTextField

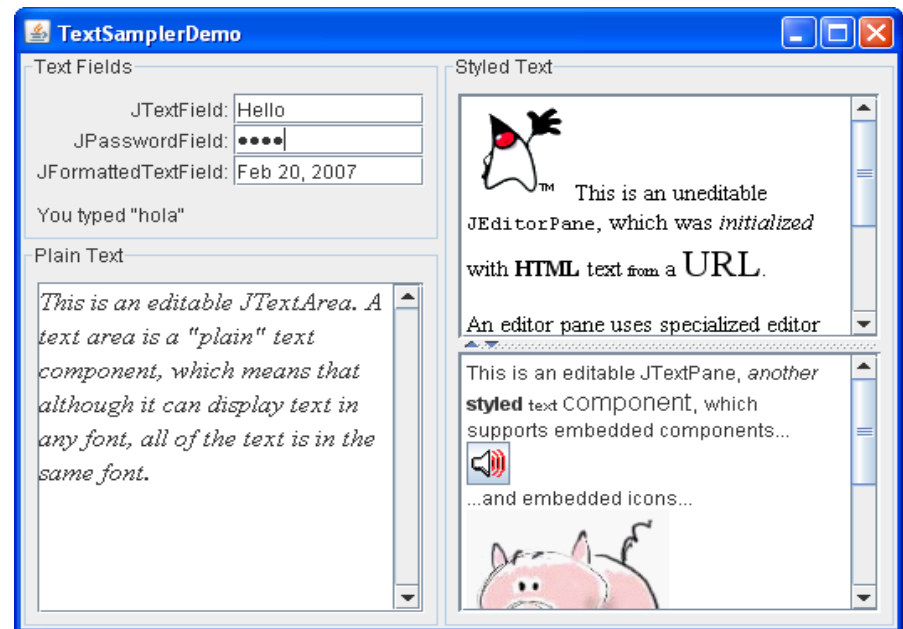


Clase JTextArea

- Algunos métodos útiles
 - `public JTextArea()`
 - `public JTextArea(int filas, int columnas)`
 - `public JTextArea(String texto, int filas, int columnas)`
 - `public int getRows()`
 - `public int getColumns()`
 - `public void append(String texto)`
 - `public void insert(String texto, int posicion)`
 - `public void setCaretPosition(int position)`
 - Podemos recuperar la longitud de un JTextArea mediante:
`myTextArea.getDocument().getLength()`

Clases JEditorPane y JTextPane

- Clases para insertar texto con formato
- JTextPane extiende a JEditorPane
 - Ambos permiten mostrar texto enriquecido
 - JEditorPane permite cargar texto desde una URL
 - JTextPane implementa el modelo vista-controlador
- Hacen uso de los StyleEditorKits
- Pueden mostrar imágenes
 - JEditorPane sólo desde HTML



Clases JEditorPane y JTextPane

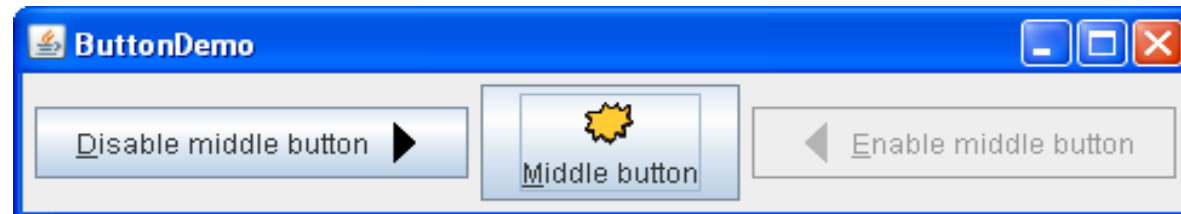
- Algunos métodos útiles de JEditorPane
 - `public JEditorPane(URL pagina)`
 - `public JEditorPane(String pagina)`
 - `public void setPage(URL pagina)`
 - `public void setPage(String pagina)`
 - `public URL getPage()`
- Algunos métodos útiles de JTextPane
 - `public JTextPane()`
 - `public JTextPane(StyledDocument doc)`
 - `public StyledDocument getStyledDocument()`
 - `public void setStyledDocument(StyledDocument doc)`

Ejercicio

- Crear un JEditorPane a partir del contenido de una URL

Clase JButton

- Los botones típicos de las aplicaciones con GUI
- Su principal función es la de generar un evento cuando se hace “*click*” sobre ellos
- Su contenido pueden ser tanto texto, como imágenes o combinaciones de ambos



Clase JButton

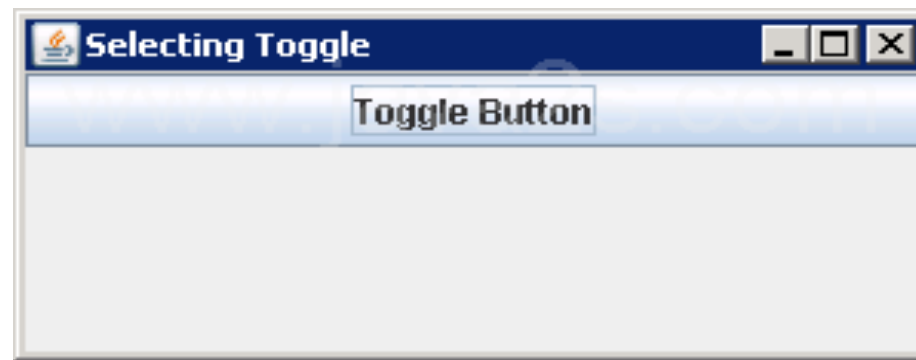
- Algunos métodos útiles
 - `public JButton(String texto)`
 - `public JButton(Icon icono)`
 - `public JButton(String texto, Icon icono)`
 - `public String getText()`
 - `public void setText(String label)`
 - `public void setEnabled(boolean b)`
 - `public void setMnemonic(int i)`
 - `KeyEvent.VK_0 – VK_9`
 - `KeyEvent.VK_A – VK_Z`

Ejercicio

- Crear la ventana de dos transparencias atrás (las imágenes no son necesarias)
- Cuando se pulsa en el botón de la izquierda, el botón central debe deshabilitarse
- Cuando se pulsa el botón de la derecha, el botón central debe volver a habilitarse
- Tenemos que añadir mnemónicos para ejecutar la acción de cada uno de los botones
 - Al invocar un botón mediante su mnemónico, se lanza el evento `ActionEvent...`

Clase JToggleButton

- Botón de dos estados (pulsado o no pulsado)
- Ejemplo de estos botones
 - Barras de tareas de editores de texto (negrita, subrayado, cursiva, etc.)
- Puede usarse tanto con texto como con imágenes
- API muy parecida a la de la clase JButton

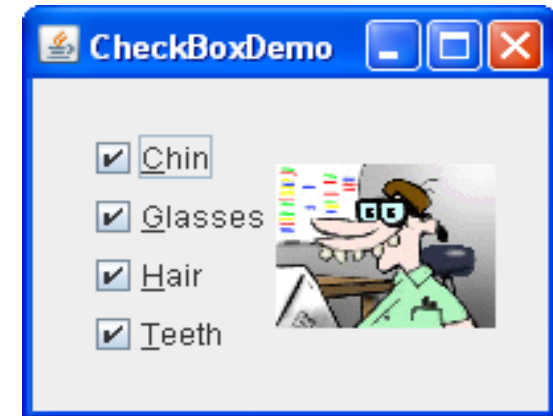


Clase JToggleButton

- Algunos métodos útiles
 - `public JToggleButton(String texto, boolean estado)`
 - `public JToggleButton(Icon icono, boolean estado)`
 - `public JToggleButton(String texto, Icon icono, boolean estado)`
 - `public boolean isSelected()`
 - `public void setSelected()`

Clase JCheckBox

- Componente que puede estar seleccionado o no y que muestra su estado
- No hay limitación en cuanto al número de Checkboxes que pueden estar seleccionados
- Apariencia como en la imagen
- Pueden mostrar un texto o imagen que indiquen su finalidad
- También se les puede asociar atajos de teclado
- Pueden ser usados en menús mediante la clase JCheckBoxMenuItem



Clase JCheckBox

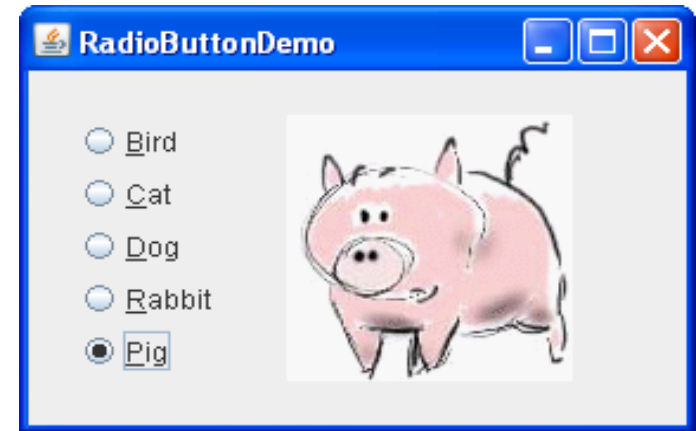
- Algunos métodos útiles
 - `public JCheckBox()`
 - `public JCheckBox(String texto, boolean seleccionado)`
 - `public JCheckBox(Icon icono, boolean seleccionado)`
 - `public JCheckBox(String texto, Icon icono, boolean seleccionado)`
 - `public boolean isSelected()`
 - `public void setSelected(boolean b)`
 - `public String getText()`
 - `public void setEnabled(boolean b)`

Ejercicio

- Vamos a crear la ventana de dos transparencias atrás
- La etiqueta de la derecha debe mostrar la imagen correspondiente a la combinación de valores seleccionada con los Check Boxes
- Hay una imagen por cada combinación: con la barbilla cambiada, el pelo cambiado, gafas cambiadas, dientes cambiados, pelo y gafas cambiados a la vez, etc.
- Después de cada cambio en la selección debemos actualizar el contenido de la etiqueta con el contenido adecuado

Clases JRadioButton y ButtonGroup

- **ButtonGroup** permite agrupar una serie de casillas de verificación (**JRadioButton**) de entre las que sólo puede seleccionarse una
- Marcar una de las casillas implica que el resto sean desmarcadas automáticamente
- Hay que crear un ButtonGroup y luego añadirle los botones
- Si se añade más de un botón seleccionado inicialmente, prevalecerá el **primero** de ellos



Clases JRadioButton y ButtonGroup

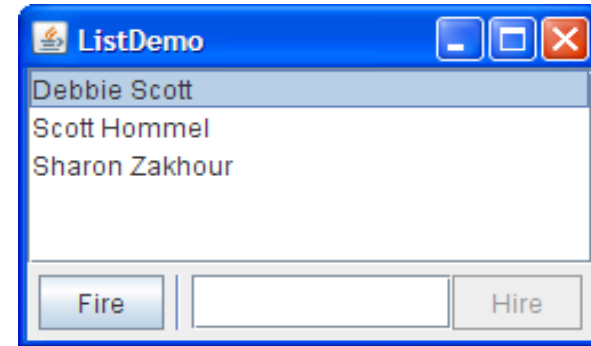
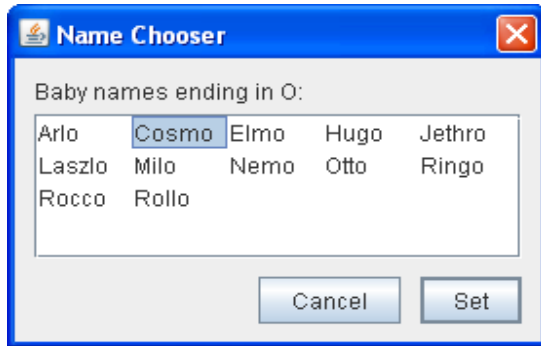
- Algunos métodos útiles de ButtonGroup
 - `public ButtonGroup()`
 - `public void add(AbstractButton b)`
- Algunos métodos útiles de JRadioButton
 - `public JRadioButton()`
 - `public JRadioButton(String texto, boolean seleccionado)`
 - `public boolean isSelected()`
 - `public void setSelected(boolean b)`
 - `public String getText()`
 - `public void setEnabled(boolean b)`

Ejercicio

- Vamos a crear la ventana de dos transparencias atrás
- Hay que crear un ButtonGroup y añadirle varios JRadioButtons
- Cada uno de estos botones nos permitirá seleccionar el animal a mostrar en la etiqueta de la derecha
- El contenido de la etiqueta se actualizará en función de la selección llevada a cabo

Clase JList

- Lista sobre la que se puede ver y seleccionar **más de un elemento** simultáneamente



- Si hay más elementos de los que se puede visualizar, podemos usar JScrollPane para que aparezcan barras de desplazamiento
- Puede configurarse la orientación y cómo se seleccionan los elementos

Clase JList

- Algunos métodos útiles
 - `public JList()`
 - `public JList(Object[] listItems)`
 - `public JList(Vector listItems)`
 - `public int getSelectedIndex()`
 - `public int[] getSelectedIndices()`
 - `public Object getSelectedValue()`
 - `public Object[] getSelectedValues()`
 - `public void setLayoutOrientation(int orientacion)`
 - `JList.VERTICAL`, `JList.HORIZONTAL_WRAP` y `JList.VERTICAL_WRAP`
 - `public void setSelectionMode(int modo)`
 - `ListSelectionModel.SINGLE_SELECTION`,
`ListSelectionModel.SINGLE_INTERVAL_SELECTION`,
`ListSelectionModel.MULTIPLE_INTERVAL_SELECTION`
 - `public ListModel getModel()`

Clase JList

- ListModel
 - Es el contenedor real de los datos
 - Puede personalizarse (aunque DefaultListModel suele ser suficiente)
 - Por defecto, los otros constructores crean JLists de sólo lectura

- Uso de ListModel

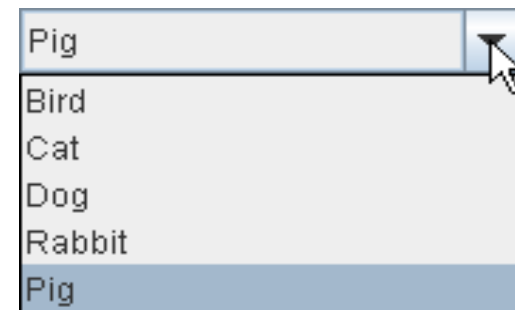
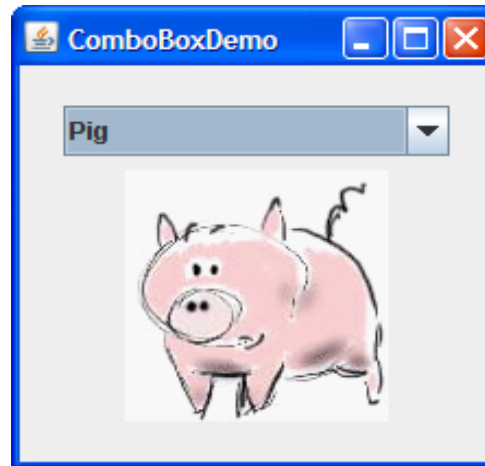
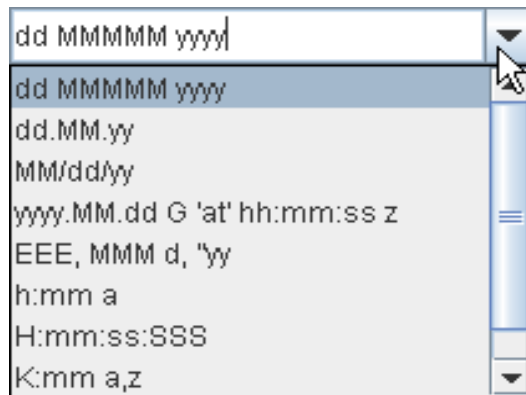
```
DefaultListModel listModel = new DefaultListModel;  
listModel.addElement(elemento);  
JList = new JList(listModel);  
listModel.addElement(otro_elemento);  
listModel.removeElement(indice);  
listModel.clear();
```

Ejercicio

- Hay que crear una ventana que contendrá un JList en la parte superior y un cuadro de texto y dos botones en la parte inferior
- Uno de los botones servirá para añadir elementos a la lista y el otro para borrarlos
- El botón de borrar elementos sólo estará activo cuando haya algún elemento (o varios) de la lista seleccionado. Al hacer click, el elemento (o elementos) seleccionado será borrado
- El botón de añadir elementos sólo estará activo si en el cuadro de texto hay algún texto introducido. Al hacer click, el texto que haya en el cuadro de texto se añadirá como un nuevo elemento a la lista

Clase JComboBox

- Lista desplegable de la que sólo se puede elegir una opción
- Se pueden crear JComboBoxes tanto editables como no editables
- Si se hace editable, podemos utilizar el evento *Action* para añadir el nuevo elemento



Clase JComboBox

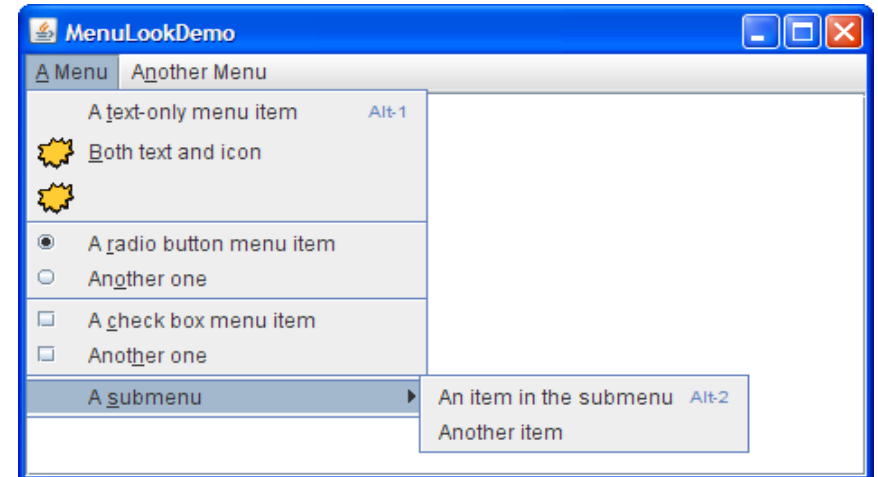
- Algunos métodos útiles
 - `public JComboBox()`
 - `public JComboBox(Object[] items)`
 - `public JComboBox(Vector items)`
 - `public void addItem(Object item)`
 - `public Object getItem(int indice)`
 - `public int getSelectedIndex()`
 - `public Object getSelectedItem()`
 - `public void setEditable(boolean editable)`

Ejercicios

- Repetiremos el ejercicio de los JRadioButtons
- En lugar de seleccionar el animal a mostrar mediante dichos botones, lo haremos usando un JComboBox
- El JComboBox será no editable
- Por otro lado, crearemos una ventana con un JComboBox que sea editable
- Si escribimos nuevo texto y pulsamos *Enter* (lanzará un evento acción en el JComboBox) el texto que hayamos introducido se añadirá como un elemento nuevo del JComboBox

Clases JMenuBar, JMenu, JMenuItem, etc.

- Barra superior de un programa con GUI que permite seleccionar una serie de acciones
 - Pueden contener sublistas (submenús)
- La estructura básica se compone de:
 - JMenuBar: la barra superior
 - JMenu: cada uno de los menús
 - JMenuItem: cada uno de los elementos de un menú (p.ej. JCheckBoxMenuItem, JRadioButtonMenuItem o incluso JMenu)



Clases JMenuBar, JMenu, JMenuItem, etc.

- Algunos métodos útiles de JMenuBar
 - `public JMenuBar()`
 - `public JMenu add(JMenu m)`
 - `public JMenu getMenu(int indice)`
- Algunos métodos útiles de JMenu
 - `public JMenu(String s)`
 - `public JMenu add(JMenuItem menuItem)`
 - `public JMenu add(String s)`
 - `public void addSeparator()`
 - `public JMenuItem getItem(int pos)`

Clases JMenuBar, JMenu, JMenuItem, etc.

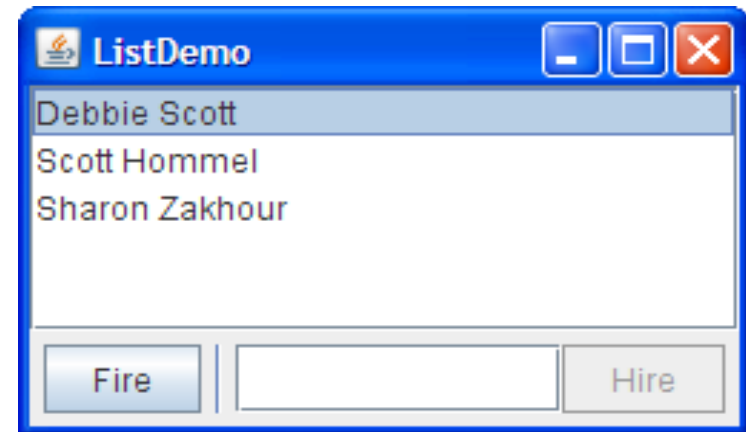
- Algunos métodos útiles de JMenuItem
 - `public JMenuItem()`
 - `public JMenuItem(String texto)`
 - `public JMenuItem(Icon icono)`
 - `public void setEnabled(boolean b)`
- Mnemónicos y Aceleradores
 - `public void setMnemonic(int mnemonico)`
 - `public void setAccelerator(KeyStroke combinacion)`
- Añadiendo la barra de menú a la ventana (JFrame)
 - `public void setMenuBar(MenuBar mb)`

Ejercicio

- Reharemos el ejercicio de copiar, cortar y pegar
- En lugar de usar botones para llevar a cabo las acciones, crearemos una barra de menú, añadiremos un menú de “Editar” y en este menú añadiremos los items de “Copiar”, “Cortar” y “Pegar”
- En este caso, en lugar de JTextFields usaremos JTextAreas y ambas áreas serán editables
- Controlaremos en qué área está el foco actualmente y realizaremos la acción sobre el texto que haya seleccionado en el área que tenga el foco
- Asignaremos los mnemónicos habituales a las tres acciones

Clase JSeparator

- Control muy sencillo de utilizar
- Se utiliza sobre todo para separar elementos dentro de un menú
- Aunque se puede usar para separar cualquier tipo de componente
- Para añadirlo a un menú
 - `menu.addSeparator()`
- Y a una barra de herramientas
 - `barra.addSeparator()`

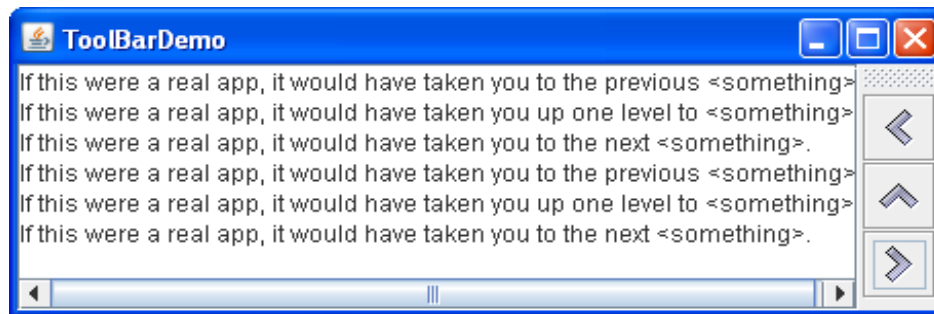
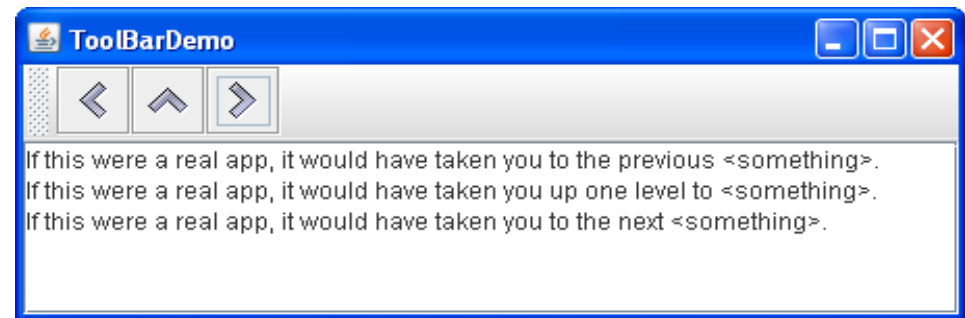


Clase JSeparator

- Algunos métodos útiles
 - `public JSeparator()`
 - `public JSeparator(int orientacion)`
 - `SwingConstants.HORIZONTAL` y `SwingConstants.VERTICAL`
 - `public int getOrientation()`
 - `public void setOrientation(int orientacion)`

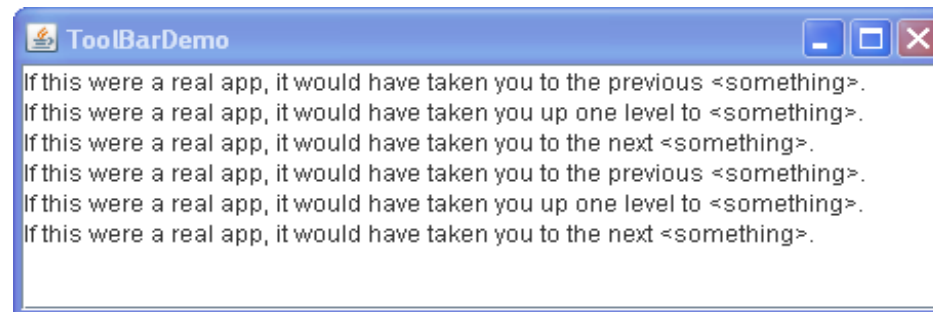
Clase JToolBar

- Contenedor que agrupa varios componentes (normalmente botones con imágenes)
- Puede organizarse en filas o columnas
- Por defecto, la barra puede moverse (siempre que usemos BorderLayout)



Clase JToolBar

- Algunos métodos útiles
 - `public JToolBar()`
 - `public JToolBar(int orientacion)`
 - `public JToolBar(String titulo)`
 - `public Component addComponent(Component comp)`
 - `public addSeparator()`
 - `public void setFloatable(boolean b)`
 - `public boolean isFloatable()`



Ejercicio

- Reharemos el ejercicio de copiar, cortar y pegar (una vez más...)
- En lugar de mediante botones o menús, usaremos una Barra de Herramientas para añadir la funcionalidad
- Permitiremos que la barra de herramientas pueda ser arrastrada

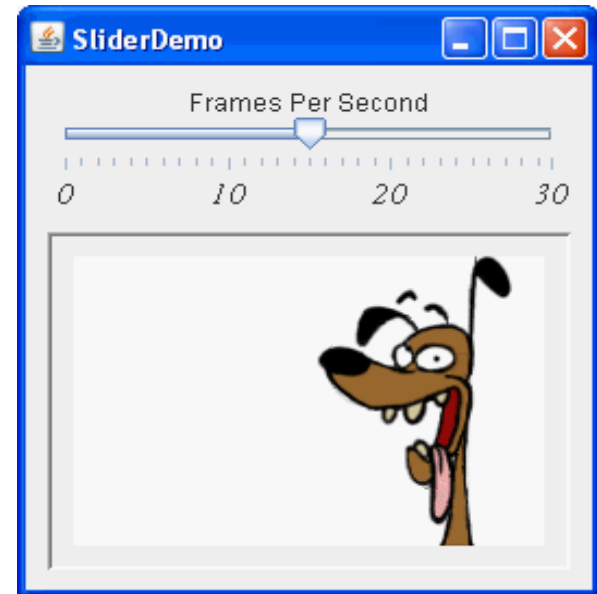
Clase JScrollBar

- Permiten desplazarse por contenidos de mayor tamaño que el área visible
- Normalmente, no se usan por separado, sino junto con un JScrollPane
- Algunos métodos:
 - `public JScrollBar()`
 - `public JScrollBar(int orientacion, int valor, int visible, int min, int max)`
 - `public void setOrientation(int orientacion)`



Clase JSlider

- Componente que permite seleccionar un valor numérico comprendido entre un máximo y un mínimo deslizando una barra
- Las etiquetas se nombran con sus valores numéricos (aunque se pueden personalizar)
- Los nuevos valores pueden recibirse con un `ChangeListener`
 - Se invoca al método `stateChanged()`



Clase JSlider

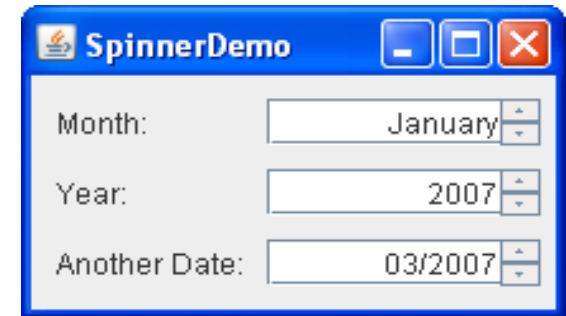
- Algunos métodos útiles
 - `public JSlider()`
 - `public JSlider(int orientacion, int min, int max, int val_inicial)`
 - `public void setInverted(boolean invertido)`
 - `public void setMajorTickSpacing(int espacio)`
 - `public void setMinorTickSpacing(int espacio)`
 - `public void setPaintTicks(boolean pintar_ticks)`
 - `public void setPaintLabels(boolean pintar_etiquetas)`
 - `public void setLableTable(Dictionary etiquetas)`

Ejercicio

- Crear una ventana que contenga una etiqueta coloreada con unos tamaños horizontal y vertical iniciales
- Añadir dos JSliders, y dos etiquetas, una a la izquierda de cada slider que digan “Tamaño horizontal” y “Tamaño vertical”
- Cada uno de los sliders debe servir para modificar el tamaño horizontal y vertical, respectivamente

Clase JSpinner

- Sirven para elegir un valor entre un rango de valores
- Permiten introducir valores directamente...
- ... y usar los controles para moverse entre los valores permitidos
- Los valores permitidos para el spinner son controlados por su modelo asociado
- Podemos usar alguno de los modelos predefinidos o definir un modelo personalizado
 - SpinnerListModel
 - SpinnerNumberModel
 - SpinnerDateModel
- Los cambios de valor se controlan con ChangeListener



Clase JSpinner

- Algunos métodos útiles
 - `public JSpinner()`
 - `public JSpinner(SpinnerModel modelo)`
 - `public void setValue(Object valor)`
 - `public Object getValue()`
 - `public Object getNextValue()`
 - `public Object getPreviousValue()`
 - `public SpinnerModel getModel()`

Ejercicio

- Vamos a crear la pantalla de dos transparencias hacia atrás
- Cada uno de los tres JSpinners tiene que usar un modelo de datos distinto
 - El primero, SpinnerListModel
 - El segundo, SpinnerNumberModel
 - El tercero, SpinnerDateModel
- Tenemos que asociar a cada modelo un valor mínimo y otro máximo

Clase JToolTip

- Consejos que aparecen al poner el ratón sobre un componente
- Se puede definir con el método *setToolTipText* de la clase *JComponent*



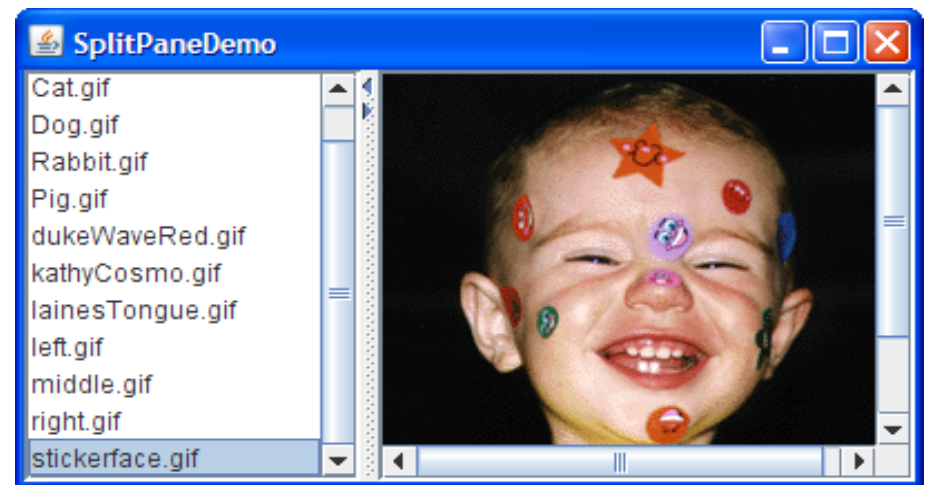
- En componentes con distintas partes (como JTabbedPane) podemos definir un ToolTip para cada parte

Clase JToolTip

- Algunos métodos útiles
 - `public void setToolTipText(String texto)`
 - `public String getToolTipText()`
 - `public String getToolTipText(MouseEvent evento)`

Clase JSplitPane

- Panel que permite situar contenido en dos áreas, una encima de la otra, o a izquierda y derecha
- Mediante el deslizador central, el usuario puede seleccionar el ancho que va a ocupar cada una de las partes
- Muchas veces se usa junto con un JScrollPane
- Se pueden anidar varios JSplitPanes
- El comportamiento del desplazador se controla con los tamaños mínimos y preferidos de los componentes



Clase JSplitPane

- Algunos métodos útiles
 - `public JSplitPane()`
 - `public JSplitPane(int orientacion, boolean repintar)`
 - `public JSplitPane(int orientacion, boolean repintar, Component comp1, Component comp2)`
 - `public void setOrientation(int orientacion)`
 - `JSplitPane.HORIZONTAL_SPLIT`, `JSplitPane.VERTICAL_SPLIT`
 - `public void setDividerSize(int tamaño)`
 - `public void setOneTouchExpandable(boolean expandible)`
 - `public void set[Top | Bottom | Right | Left] Component(Component comp)`
 - `public void addComponent(Component comp)`

Ejercicio

- Vamos a rehacer el ejercicio de la selección de las imágenes de los animales
- Usaremos un JSplitPane para dividir la ventana en dos partes
- La parte de la izquierda albergará un JList con la lista de animales
- En la parte de la derecha pondremos la etiqueta que usaremos para mostrar la imagen
- Cada vez que seleccionemos un animal distinto de la lista actualizaremos la imagen de la etiqueta

Clase JTabbedPane

- Contenedor que permite gestionar distintos conjuntos de componentes en el mismo panel
- Podemos seleccionar a qué contenedor añadir los componentes
- Se puede seleccionar dónde situar las pestañas
- Se pueden usar teclas, atajos o el ratón para navegar por las pestañas

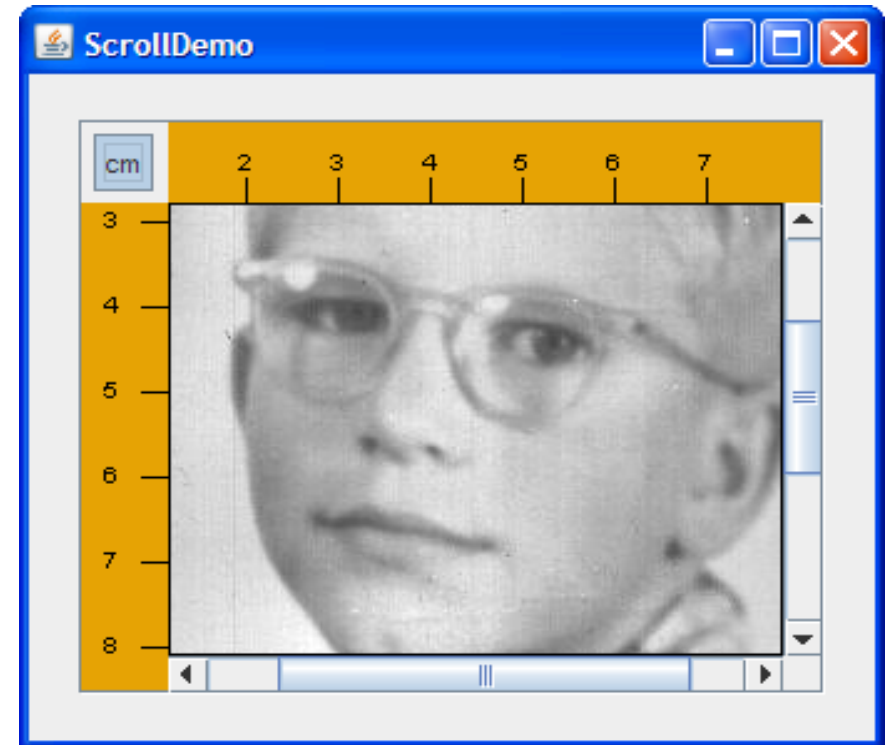


Clase JTabbedPane

- Algunos métodos útiles
 - `public JTabbedPane()`
 - `public JTabbedPane(int posicion, int layout)`
 - `public void addTab(String titulo, Icon icono, Component comp, String tooltip)`
 - `public void insertTab(String titulo, Icon icono, Component comp, String tooltip, int posicion)`
 - `public void remove(Component comp)`
 - `public void removeTabAt(int posicion)`
 - `public void setComponentAt(int posicion, Component comp)`

Clase JScrollPane

- Contenedor para componentes con barras de desplazamiento
- Se puede establecer el contenido en el momento de la construcción o a través de su *view port*
- Podemos seleccionar el comportamiento de las barras de desplazamiento
- Se puede personalizar la decoración del panel

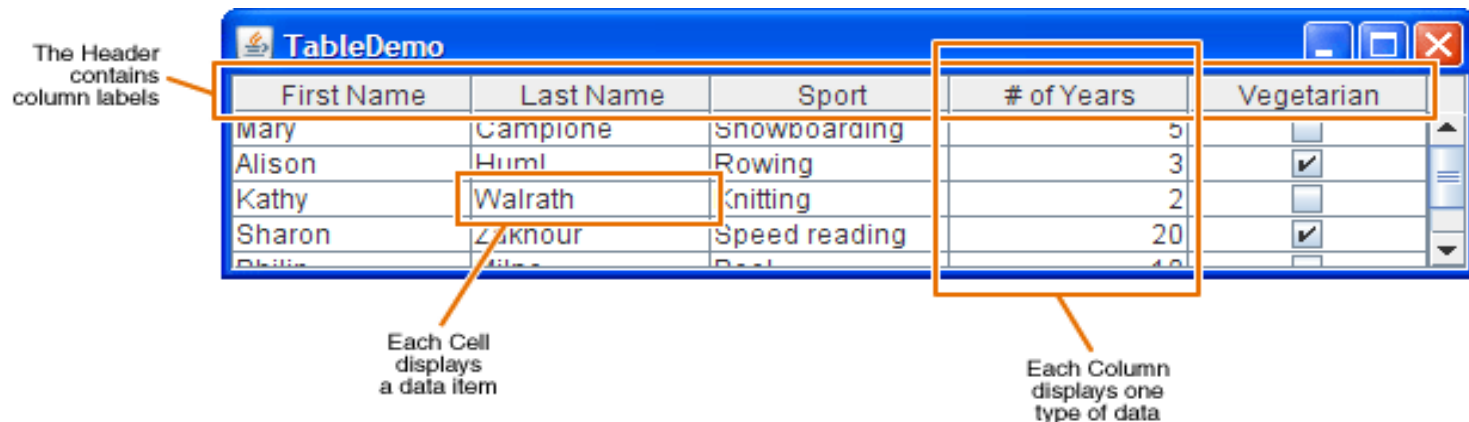


Clase JScrollPane

- Algunos métodos útiles
 - `public JScrollPane()`
 - `public JScrollPane(Component comp, int politica_horiz, int politica_vert)`
 - `public JViewport getViewport()`
 - `public void setColumnHeaderView(Component comp)`
 - `public void setRowHeaderView(Component comp)`
 - `public void setWheelScrollingEnabled(boolean scroll)`

Clase JTable

- Nos permite mostrar información de forma tabular
- Podemos construir una tabla pasando un Vector/Array con la cabecera y otro con el contenido en el constructor
 - Todas las celdas serán editables
 - Todos los tipos de datos se mostrarán por igual (como Strings)
 - Para solventarlo se crea (o usa uno de los de por defecto) un TableModel



Clase JTable

- Normalmente, añadiremos la tabla a un contenedor/panel
 - Si es un JScrollPane, la cabecera es añadida automáticamente
 - Si usamos otro panel, hay que añadir la cabecera a mano en la posición que deseemos
 - Con el método `getTable()` recuperamos una referencia a la cabecera
- El ancho de una columna se asigna automáticamente
 - Si la ventana crece, las columnas crecen por igual
 - Si cambiamos el ancho de una columna, las de la derecha se ajustan
- El modelo de la tabla es el que nos permite acceder al contenido
 - Y también registrar el tratamiento de eventos

Clase JTable

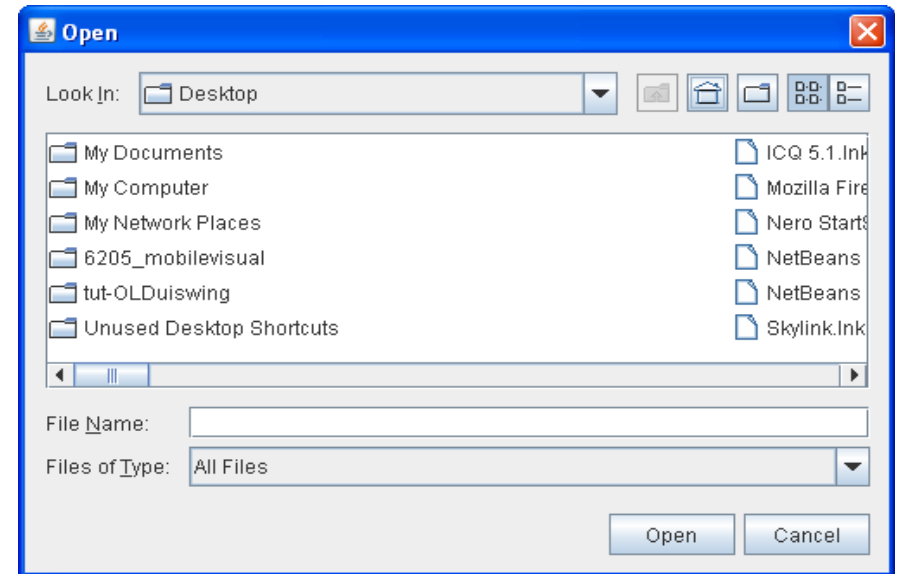
- Conceptos: Editores y Renderizadores
 - Los editores nos permiten añadir contenido a una tabla
 - Pueden ser JTextFields o JComboBoxes, por ejemplo
 - Los renderizadores determinan cómo se pinta el contenido de un tipo de celdas
 - Pueden personalizarse
- El contenido de la tabla puede ordenarse por columnas
- Para más información:
<http://java.sun.com/docs/books/tutorial/uiswing/components/table.html>

Ejercicio

- Crear una ventana que contenga una tabla con datos
- Debe ser ordenable por cualquier columna
- Debe admitir la selección por filas, con sólo una fila activa en cada momento
- Deben mostrarse los encabezados de cada columna
- Debe permitir hacer scroll en caso de que el contenido no quepa en la ventana
- La ventana tiene que tener, además, un botón que, al pulsarlo, muestre un diálogo con el contenido de esa celda (para comprobar que efectivamente estamos leyendo los datos de la tabla)

Clase JFileChooser

- Control predefinido para navegar por la estructura de directorios
- La ventana que muestra es una venta modal
- Los directorios se recuerdan entre usos
- Se puede cambiar el comportamiento para que permita seleccionar sólo ficheros o directorios
- Se puede personalizar la vista y los filtros utilizados



Clase JFileChooser

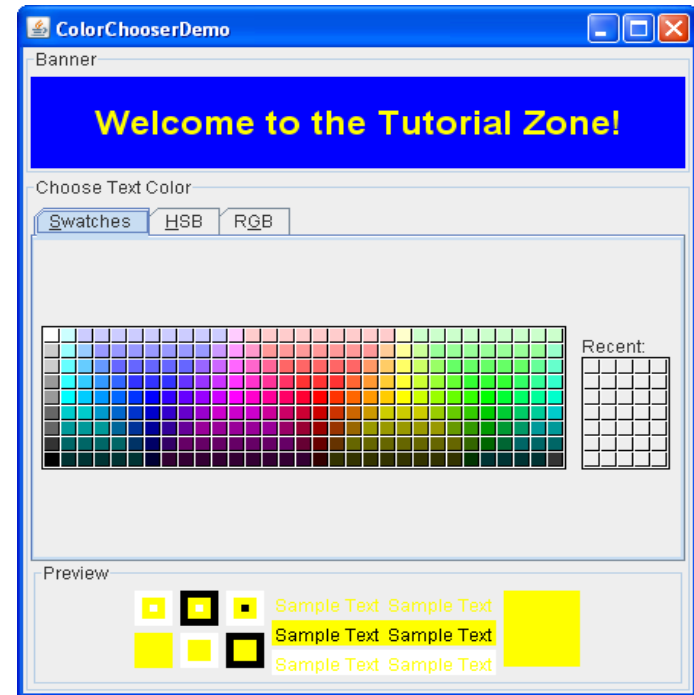
- Algunos métodos útiles
 - `public JFileChooser()`
 - `public JFileChooser(String directorio)`
 - `public int showDialog(Component padre, String confirmacion)`
 - `APPROVED_OPTION`, `CANCEL_OPTION` y `ERROR_OPTION`
 - `public File getSelectedFile()`
 - `public File[] getSelectedFiles()`
 - `public setFileSelectionMode(int modo)`
 - `FILES_ONLY`, `DIRECTORIES_ONLY` y `FILES_AND_DIRECTORIES`
 - `public void setMultiSelectionEnabled(boolean multiseleccion)`
 - `public void setCurrentDirectory(File directorio)`

Ejercicio

- Vamos a crear una ventana que contenga un campo de texto para mostrar información (JTextField no editable), un botón y un grupo de JRadioButtons con tres botones
- El botón, al pulsarlo, debe mostrar un diálogo JFileChooser
- Cuando seleccionemos un fichero o directorio, el nombre de dicho fichero o directorio tendremos que mostrarlo en el campo de texto
- Los tres JRadioButtons deben permitir seleccionar si lo que queremos poder escoger son ficheros, directorios o ambas cosas

Clase JColorChooser

- Componente predefinido que permite seleccionar un color de entre una paleta de colores
- Presenta dos áreas
 - El área de selección
 - El área de previsualización
- Se puede ocultar el área de previsualización
- Se puede personalizar el área de selección



Clase JColorChooser

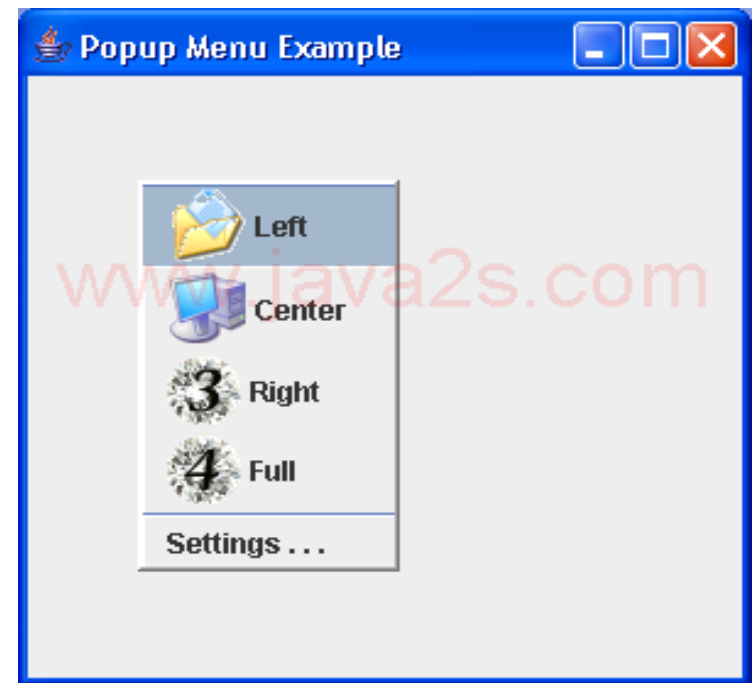
- Algunos métodos útiles
 - `public JColorChooser()`
 - `public JColorChooser(Color color_inicial)`
 - `public Color showDialog(Component padre, String titulo, Color color_inicial)`
 - `public void setPreviewPanel(JComponent panel)`
 - `public Color getColor()`
 - `public ColorSelectionModel getSelectionModel()` : Permite acceder al modelo, que es el que genera los eventos de cambio de color seleccionado

Ejercicio

- Tenemos que crear la ventana de dos transparencias atrás
- Está dividida en dos partes
 - La superior tiene una etiqueta donde al texto le hemos dado un color y también hemos modificado el color de fondo
 - La parte inferior muestra un diálogo de selección de color
- Al seleccionar un color en la parte inferior, deberemos cambiar el color del texto de la etiqueta adecuadamente

Clase JPopupMenu

- Nos permite mostrar menús contextuales al pulsar el botón derecho sobre un componente
- Hay que controlar los eventos de ratón sobre los componentes en los que deseemos habilitarlo
- Usamos un método especial de la clase *SwingUtilities* que nos dirá si el click recibido es de botón derecho
- Se compone de los mismos *MenuItem*s que los menús superiores



Clase JPopupMenu

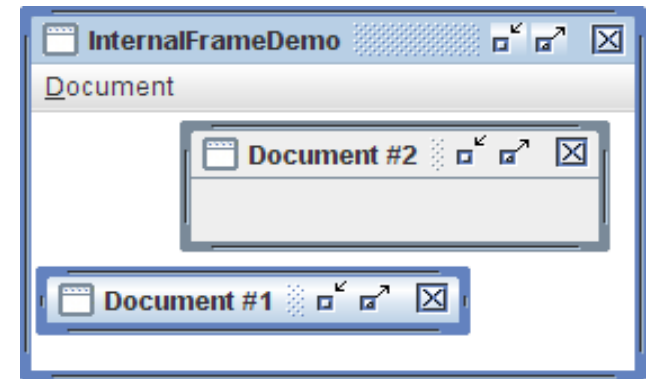
- Algunos métodos útiles
 - `public JPopupMenu()`
 - `public JMenuItem add(JMenuItem item)`
 - `public JMenuItem add(String item)`
 - `public void addSeparator()`
 - `public void insert(Component item, int pos)`
 - `public void remove(int pos)`
 - `public void removeAll()`
 - `public void show(Component comp, int x, int y)`
- Método necesario definido en *MouseEvent*
 - `public boolean isPopupTrigger()`

Ejercicio

- Vamos a extender la última versión del editor con opciones de copiar, cortar y pegar
- Añadiremos la opción de llevarlo a cabo mediante un JPopupMenu que aparecerá al hacer click con el botón derecho en cualquiera de las cajas de texto

Clases JDesktopPane y JInternalFrame

- Permiten crear aplicaciones multiventana dentro de un mismo JFrame
- Se instancia JDesktopPane y se añaden tantos JInternalFrames como deseemos
- A los JInternalFrames hay que darles un tamaño (si no, tendrán tamaño cero y no se verán)
- Si no se especifica una posición, aparecerán en la posición (0,0)
- Se añaden componentes como en JFrame
- Hay que hacer visibles los JInternalFrames
- Generan Internal Frames events (que son capturados con JInternalFrameListeners)



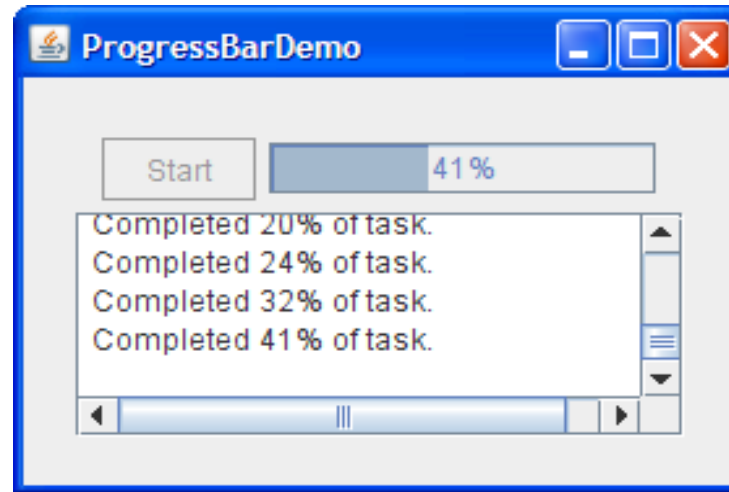
Clases JDesktopPane y JInternalFrame

- Algunos métodos útiles de JDesktopPane
 - `public JDesktopPane()`
 - `public Component add(Component internal_frame)`
- Algunos métodos útiles de JInternalFrame
 - `public JInternalFrame(String titulo, boolean redim, boolean maxim, boolean minim, boolean close)`
 - `public Container getContentPane()`
 - `public void setVisible(boolean visible)`
 - `public void setLocation(int x, int y)`
 - `public void setSize(Dimension tam)`

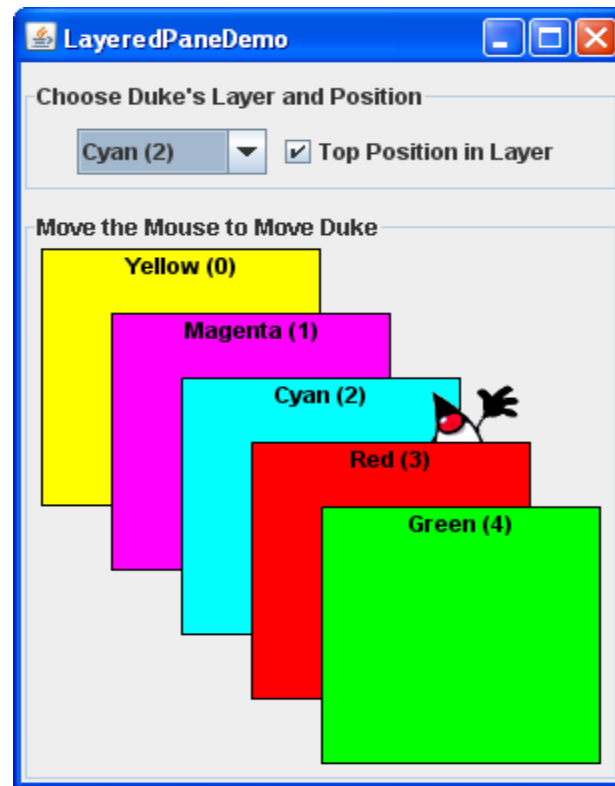
Ejercicio

- Hay que crear una ventana que contenga un JDesktopPane
- Añadiremos una barra de menú a la ventana principal, con un menú, que nos permita crear nuevos JFrame dentro del JDesktopPane
- Al crearlos, les daremos una posición aleatoria dentro del JDesktopPane
- También les daremos un tamaño aleatorio
- No olvidemos hacerlos visibles después de añadirlos

Clase JProgressBar



Clase JLayeredPane



Clase JTree

