INTRODUCCIÓN A SWING

El **diseño de toda interfaz** conlleva, a grandes rasgos, los siguientes pasos:

- **Decidir la estructura de la interfaz**:
  - Qué componentes gráficos se van a utilizar, y cómo se van a relacionar estos componentes)

- **Decidir la disposición (layout) de los componentes:**
  - Existen dos tipos de componentes: **contenedores** y **componentes** atómicos. Los contenedores sirven para organizar los componentes contenidos en los mismos. Esta organización se denomina disposición (o layout)

- **Decidir el comportamiento de la interfaz**: gestión de eventos:
  - Algunos componentes son controles: permiten reaccionar ante eventos del usuario. El comportamiento se especifica programando las respuestas a dichos eventos. Normalmente, dichas respuestas supondrán invocar funcionalidades de la lógica de la aplicación.
  - Conviene mantener la **interfaz y la lógica lo más independientes** posible (se usan patrones para lograr esto).

Los controles señalizan eventos. Hay diferentes tipos de eventos, dependiendo de los controles.

La forma de tratar eventos en Swing (y en AWT, a partir de JDK 1.1) es mediante un mecanismo denominado **delegación.**

Por cada tipo de evento notificado por un control, el control acepta un oyente de dicho evento (métodos ***addXXXListener***). Dicho oyente ha de implementar una interfaz (interface de Java, no gráfica) adecuada (***XXXListener***).

Cuando se produce un evento, el control invoca un método apropiado del oyente. Es en este método donde se trata el evento, estas clases están declaradas en el paquete **java.awt.event.**

Elementos básicos de una GUI:

Componentes GUI (*widgets*):

Son los objetos visuales del interfaz.

- Un programa gráfico es un conjunto de componentes anidados.
- **Por ejemplo:** ventanas, contenedores, menús, barras, botones, campos de texto, etc.

Disposición (layout):

Determinan cómo se colocan los componentes para lograr un GUI cómodo de utilizar.

- ***Layout managers****:* Gestionan la organización de los componentes gráficos de la interfaz.

Eventos:

Describen la interactividad, y se tratan para dar respuesta a la entrada del usuario:

- Desplazamiento del ratón, selección en un menú, botón pulsado, etc.

Creación de gráficos y texto - Clase Graphics

- Define fuentes, pinta textos
- Para dibujo de líneas, figuras, coloreado, ...

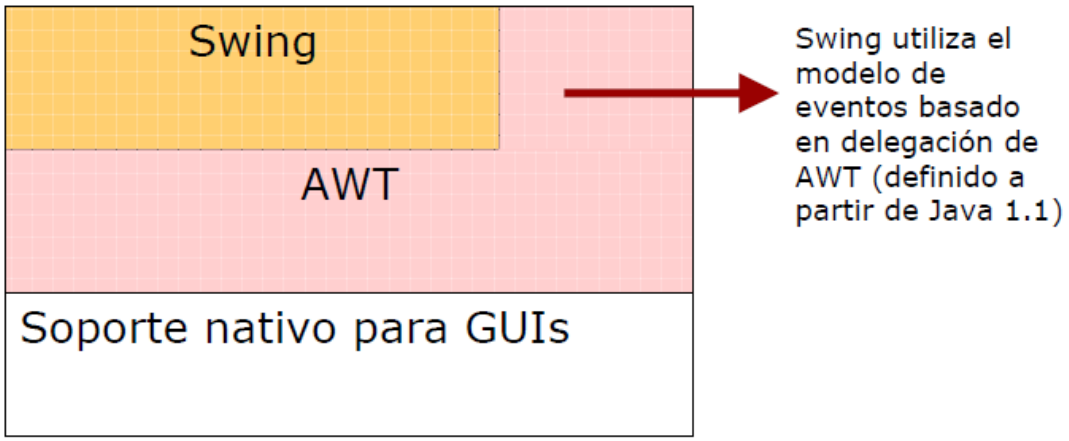
Bibliotecas de componentes para GUI

Abstract Windowing Toolkit (AWT)

- "Look & Feel" dependiente de la plataforma.
- La apariencia de ventanas, menús, etc. es distinta en Windows, Mac, Motif, y otros sistemas.
- Funcionalidad independiente de la plataforma.
- Básico y experimental.
- Estándar hasta la versión JDK 1.1.5.

Swing / Java Foundation Classes (desde JDK 1.1.5)

- Look & Feel y funcionalidad independiente de la plataforma.
- Desarrollado 100% en Java.
- Portable: si se elige un look&feel soportado por Swing (o se programa uno) puede asegurarse que la GUI se verá igual en cualquier plataforma.
- Mucho más completo que AWT.



Componentes del AWT

Contenedores (Container)
Contienen otros componentes (u otros contenedores)

- Estos componentes se tienen que añadir al contenedor y para ciertas operaciones se pueden tratar como un todo.
- Mediante un gestor de diseño controlan la disposición (*layout*) de estos componentes en la pantalla.

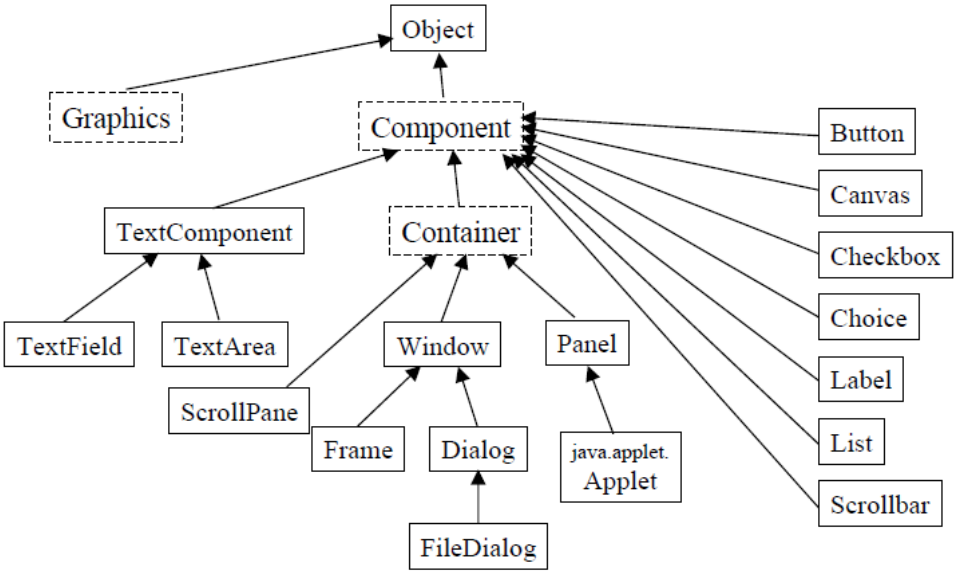
Ejemplo: Panel, Frame, Applet

Lienzo (clase Canvas)
Superficie simple de dibujo.

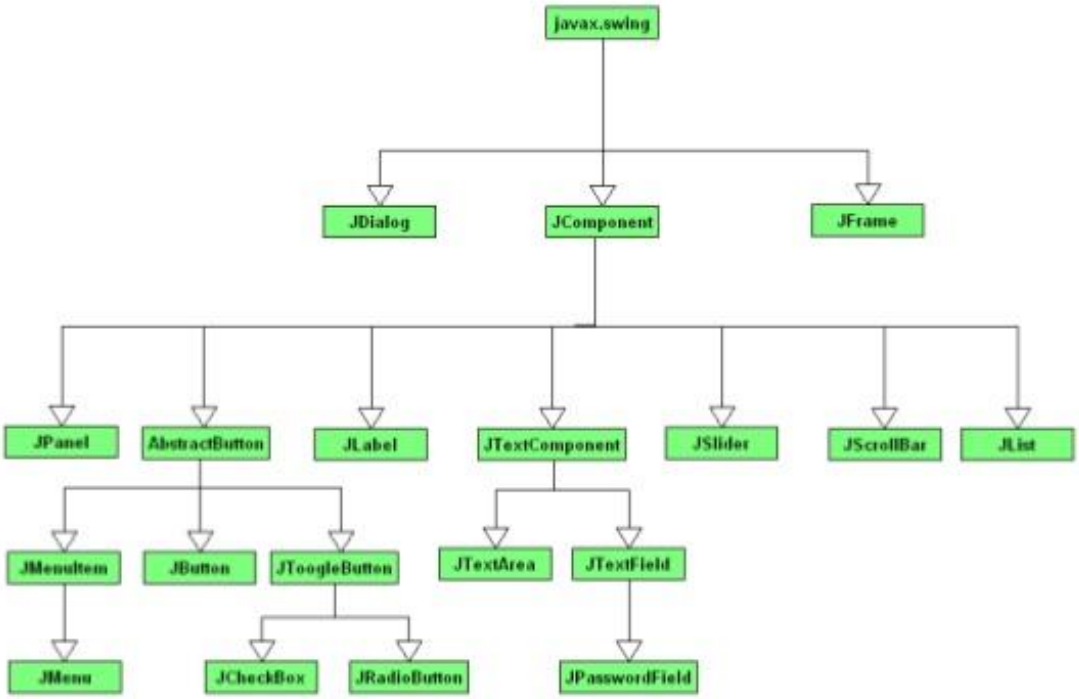
Componentes de interfaz de usuario
Botones, listas, menús, casillas de verificación, campos de texto, etc.

Componentes de construcción de ventanas
Ventanas, marcos, barras de menús, cuadros de diálogo.

Jerarquía de componentes del AWT



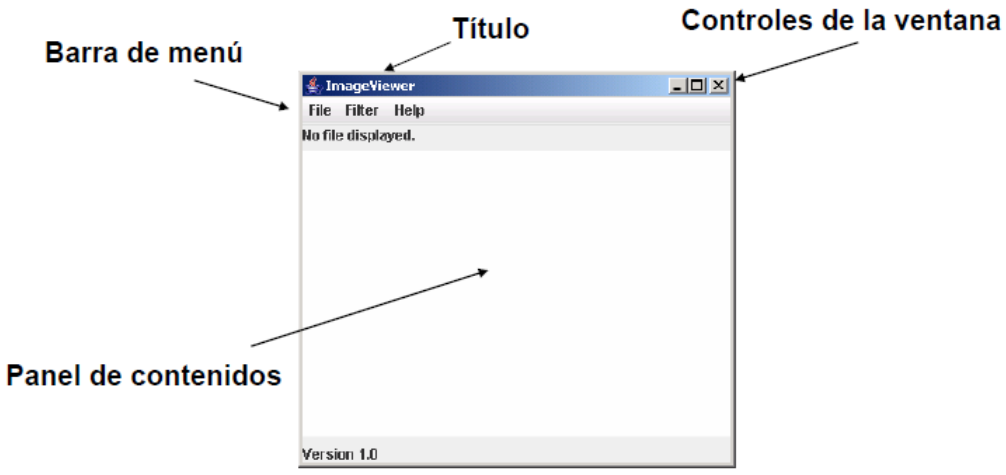
Clases de Paquete de Swing



Contenedores

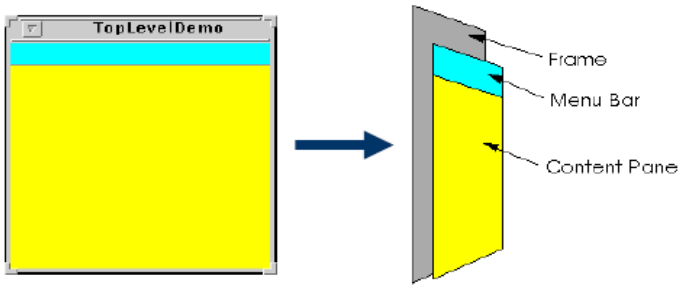
- Panel
  - Sirve para colocar botones, etiquetas, etc.
  - No existe sin una ventana que lo albergue
  - Un *applet* es un panel
- Window
  - Sirve para crear nuevas ventanas independientes
  - Ventanas gestionadas por el administrador de ventanas de la plataforma (Windows, Motif, Mac, etc.)
  - Normalmente se usan dos tipos de ventanas:
    - **Frame**: ventana donde se pueden colocar menús
    - **Dialog**: ventana para comunicarse con el usuario
      - Se usan para colocar botones, etiquetas, etc.
      - Cumple la misma función que un panel, pero en una ventana independiente

Elementos de una ventana



La clase javax.swing.**JFrame**

- Toda aplicación Swing tiene, al menos, un contenedor raíz (una ventana)
- La clase JFrame proporciona ventanas al uso (aunque puede haber otro tipo de ventanas)
- A su vez, JFrame incluye una serie de elementos:



- **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** setJMenuBar

Crea una ventana con un texto

```
/**
 * Create the Swing frame and its content.
 */
private void makeFrame()
{
    frame = new JFrame("ImageViewer");
    Container contentPane = frame.getContentPane();

    JLabel label = new JLabel("I am a label.");
    contentPane.add(label);

    frame.pack();
    frame.setVisible(true);
}
```

Para mostrar una ventana en una posición concreta disponemos del método setLocation(x, y) de la clase JFrame.

Cuando se crea un objeto **JFrame**, se crea un objeto **Container (AWT)**. El objeto JFrame usa el panel de contenido (Container) para albergar los componentes del *frame*. Dos formas de hacerlo:

Modo 1:

1. Obtenemos el panel de contenido del frame (this, es el JFrame):

```
Container panel = this.getContentPane();
```

2. Añadimos componentes a dicho panel:

```
panel.add(unComponente);
```

Modo 2:

A partir de 1.5 también se puede hacer directamente sobre el JFrame (lo hace automáticamente sobre su panel):

```
this.add(unComponente);
```

Componentes: Etiquetas, Botones, Campos, …

- Botón - Clase *JButton*
  - Botón de interacción que puede tener una etiqueta
- Etiqueta - Clase *JLabel*
  - Muestra una cadena de sólo lectura
    - Normalmente para asociar el texto con otro componente
- Campo de texto - Clase *JTextField*
  - Campo de una línea que permite introducir y editar texto
- Area de texto - Clase *JTextArea*
  - Campo de texto de varias líneas
  - Mayor funcionalidad
    - Añadir, reemplazar e insertar texto
    - Barras de desplazamiento
- Menús - Clase *JMenuBar, JMenu, y JMenuItem*
  - Para definir una barra de menús, cada menú y los elementos de cada menú

Todos los elementos anteriores pertenecen a Swing (la J delante los distingue de los elementos AWT antiguos).

Uso de componentes

1) Crear el componente
   - usando new:

     JButton b = new JButton("Correcto");

2) Añadirlo al contenedor
   - usando add:

     contentPane.add(b); // añadir en el contenedor contentPane

     // crear y añadir el componente en una sóla operación:

     contentPane.add(new Label("Etiqueta 1"));

     contentPane.add(new Label("Etiqueta 2"));
   - Si luego se quiere quitar, usar *remove*(componente)

3) Invocar métodos sobre el componente y manejar eventos

   System.out.println(b.getLabel());

   b.setLabel("etiqueta modificada");

Creación de una barra de menús

```
private void makeMenuBar(JFrame frame) {
    JMenuBar menubar = new JMenuBar();
    frame.setJMenuBar(menubar);

    // create the File menu
    JMenu fileMenu = new JMenu("File");
    menubar.add(fileMenu); // se añade a la barra de menús

    JMenuItem openItem = new JMenuItem("Open");
    fileMenu.add(openItem);  // se añade al menú File

    JMenuItem quitItem = new JMenuItem("Quit");
    fileMenu.add(quitItem);
}
```

Tratamiento de eventos

- Los eventos se corresponden a las interacciones del usuario con los componentes
- Los componentes están asociados a distintos tipos de eventos
  - Las ventanas (p.ej. JFrame) están asociadas con *WindowEvent*
  - Los menús están asociados con *ActionEvent*
- Se pueden definir objetos que saben cómo tratar los eventos
  - Estos objetos serán notificados de la ocurrencia de un evento
  - Por eso se llaman ***listeners***
  - Tienen definidos métodos que se llaman cuando ocurre el evento asociado

Tratamiento de eventos

- Dos categorías de eventos:
  - Eventos de bajo nivel
    - Están relacionados con la interacción física con la interfaz (por ejemplo, ¿qué botón del ratón se ha pulsado?)
    - Ejemplos: MouseInputEvent, WindowEvent y KeyEvent
  - Eventos de alto nivel, o semánticos
    - Representan operaciones lógicas realizadas sobre los elementos (por ejemplo, se ha pulsado el botón "Salir" en la interfaz)
    - ActionEvent

Eventos de acción: ActionEvent

- Indica que se ha producido un evento sobre un componente de la interfaz
  - JButton, JList, JTextField, JMenuItem, etc.
- El método que se invoca en los listeners está definido en la interfaz ActionListener

```
public interface ActionListener extends EventListener {
   void actionPerformed(ActionEvent e) ;
}
```

  - Luego la clase del objeto oyente debe implementar el método *actionPerformed*
- Los componentes tienen métodos para poder añadir o quitar objetos oyentes
  - **addActionListener**(ActionListener l)
  - **removeActionListener**(ActionListener l)

Ejemplo: ActionListener para ImageViewer

```
public class ImageViewer implements ActionListener
{
    …
    public void actionPerformed(ActionEvent e)
    {
        String command = e.getActionCommand();
        if(command.equals("Open")) {
            …
        }
        else if (command.equals("Quit")) {
            …
        }
        …
    }
    …
    private void makeMenuBar(Jframe frame)
    {
        …
        openItem.addActionListener(this);
        …
    }
}
```

Problemas con esta solución

- El ejemplo muestra un tratamiento centralizado de los eventos en la propia clase principal

- Aunque funciona…
- Es mejor definir clases específicas por cada evento
  - Mayor escalabilidad
  - Evitar identificar los componentes por el texto

- Veamos la alternativa, usando clases anidadas

Clases anidadas

- Las definiciones de clase se pueden anidar

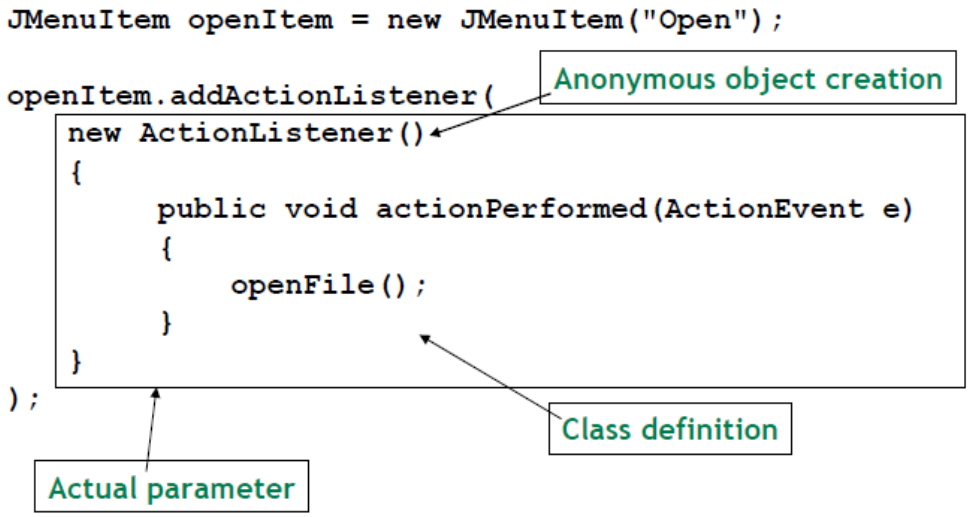
```
public class Contenedora
{
    …
    private class Anidada
    {
        …
    }
}
```

Clases anidadas anónimas

- Obedecen las reglas de las clases internas
- Se usan para crear objetos específicos en un momento, para los que no es necesario dar un nombre de clase
- Tienen una sintaxis especial
- La instancia es siempre referenciada por su supertipo, ya que no tiene nombre el subtipo

- Las instancias de las clases internas estarán dentro de la clase contenedora
- Las instancias de la clase interna tienen acceso a la parte privada de la clase contenedora
  - Esto es práctico porque el tratamiento de un evento requiere normalmente acceso al estado de la aplicación

ActionListener anónimo



Otro ***ejemplo*** sería el programar la reacción a pulsar la ventana:

Tratamiento del cierre de ventana

```
frame.addWindowListener(new WindowAdapter() {
       public void windowClosing(WindowEvent e)
       {
           System.exit(0);
       }
   });
```

- WindowAdapter es una implementación de la interfaz WindowListener donde todas las operaciones están declaradas como vacías

NOTA:

Ver ejemplo ComponenteJList1 y ComponenteJList2 (primero ver qué es JList con ejemplos simples)

NOTA

Observamos que el código creado por WindowBuilder, es el siguiente:

```
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                JFrameEjemplo window = new JFrameEjemplo();
                window.setVisible(true);
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null,
e.getMessage());
            }
        }
    });
}
```

¿Es necesario incorporar todo ese código o solo con este código es necesario?:

```
public static void main(String[] args) {
            JFrameEjemplo window = new JFrameEjemplo();
            window.setVisible(true);
}
```

El procesamiento completo de Swing se realiza en un hilo llamado **EDT (Event Dispatching Thread).** Por lo que de la forma tradicional podría bloquearse este hilo, ya que tu programa implícitamente lo está usando (con Swing).

Por ese motivo la manera de asegurar que la GUI no sea bloqueada es usarla así:

```
public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
          public void run() {
              new NewJFrame().setVisible(true);
          }
      });
}
```

Con esto aseguramos que la aplicación sea ejecutada en otro hilo (thread).

Nota: **Runnable** es una **interfaz de Java que dispone de un método run y nos permite ejecutar una tarea en paralelo desde un programa main usando la clase Thread**. Una clase que implemente Runnable, tiene que implementar un método sin argumentos, denominado **run**.

Disposición de los componentes (*layout manager*)

- Cómo se colocan los componentes (usando el método *add*) depende del gestor de disposición del contenedor (*layout manager*)
- Tipos de disposiciones:
  - FlowLayout
    - Los componentes se ponen de izquierda a derecha hasta llenar la línea, y se pasa a la siguiente. Cada línea se centra
      - Por defecto, en paneles y applets
  - BorderLayout
    - Se ponen los componentes en un lateral o en el centro
    - se indica con una dirección:"East", "West", "North", "South", "Center"
      - Por defecto, en ventanas JFrame
  - GridLayout
    - Se colocan los componentes en una rejilla rectangular (filas x cols)
    - Se añaden en orden izquierda-derecha y arriba-abajo
- Para poner una disposición se utiliza el método *setLayout()*:

```
GridLayout  nuevolayout = new GridLayout(3,2);
setLayout(nuevolayout);
```

Disposición de los componentes (Layout Manager)

http://chuwiki.chuidiang.org/index.php?title=Uso_de_Layouts
https://inforux.wordpress.com/2009/01/20/java-practicando-con-boxlayout/

Qué es un Layout

En java, cuando hacemos ventanas, la clase que decide cómo se reparten los botones (y demás controles) dentro de la ventana se llama **Layout**. Esta clase es la que decide en qué posición van los botones y demás componentes, si van alineados, en forma de matriz, cuáles se hacen grandes al agrandar la ventana, etc. Otra cosa importante que decide el **Layout** es qué tamaño es el ideal para la ventana en función de los componentes que lleva dentro.

Con un **layout** adecuado, el método **pack()** de la ventana hará que coja el tamaño necesario para que se vea todo lo que tiene dentro.

FlowLayout

FlowLayout. Coloca todos los componentes que le añadimos alineados de izquierda a derecha, haciendo que cada uno ocupe lo que necesita. Este es el layout por defecto para los JPanel y JApplet.

Si hay hueco de sobra en horizontal, los componentes aparecerán centrados. Si falta hueco, los componentes se partirán automáticamente en varias filas.

BoxLayout

BoxLayout. Es un Layout simple que permite poner los componentes de forma horizontal (igual que un FlowLayout) o de forma vertical.

Para crear una clase BoxLayout, necesitamos 2 argumentos: el objeto contenedor, y la clase que indica la forma en que ordenará los componentes.

BoxLayout.X_AXIS - Forma Horizontal
BoxLayout.Y_AXIS - Forma Vertical

El siguiente código coloca los componentes en la ventana (el botón y la etiqueta) en vertical y centrados.

```
import java.awt.Component;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

public class PruebaBoxLayout {
        public static void main(String [] args)
        {
                // Se crea la ventana con el BoxLayout
                JFrame v = new JFrame();
                v.getContentPane().setLayout(new
BoxLayout(v.getContentPane(),BoxLayout.Y_AXIS));

                // Se crea un botón centrado y se añade
                JButton boton = new JButton("B");
                boton.setAlignmentX(Component.CENTER_ALIGNMENT);
                v.getContentPane().add(boton);

                // Se crea una etiqueta centrada y se añade
                JLabel etiqueta = new JLabel("una etiqueta larga");
                etiqueta.setAlignmentX(Component.CENTER_ALIGNMENT);
                v.getContentPane().add(etiqueta);

                // Visualizar la ventana
                v.pack();
                v.setVisible(true);
                v.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        }
}
```

Lo de **BoxLayout.Y_AXIS** es para que coloque los componentes en vertical, de arriba a abajo. Con **BoxLayout.X_AXIS** los coloca igual que un FlowLayout, de izquierda a derecha.

También hay **BoxLayout.LINE_AXIS** e **BoxLayout.PAGE_AXIS**, pero el comportamiento por defecto es igual. El comportamiento es distinto si al contenedor le cambiamos el **setComponentOrientation()**, en cuyo caso podemos conseguir que los componentes vayan de derecha a izquierda según los vamos añadiendo o de abajo a arriba.

Para que los componentes salgan centrados (la etiqueta y el botón) hay que ir llamando uno por uno a su método **setAlignmentX(Component.CENTER_ALIGNMENT)**. Puedes probar a llamar a uno sí y a otro no, o a poner "alignment" distintos a cada uno. Conseguirás colocaciones verdaderamente extrañas.

BorderLayout

BorderLayout. El **BorderLayout** divide la ventana en 5 partes: centro, arriba, abajo, derecha e izquierda.

Hará que los componentes que pongamos arriba y abajo ocupen el alto que necesiten, pero los estirará horizontalmente hasta ocupar toda la ventana.

Los componentes de derecha e izquierda ocuparán el ancho que necesiten, pero se les estirará en vertical hasta ocupar toda la ventana.

El componente central se estirará en ambos sentidos hasta ocupar toda la ventana.

El **BorderLayout** es adecuado para ventanas en las que hay un componente central importante (una tabla, una lista, etc) y tiene menús o barras de herramientas situados arriba, abajo, a la derecha o a la izquierda.

Este es el layout por defecto para los JFrame y JDialog.

GridLayout

El **GridLayout** pone los componentes en forma de matriz (cuadrícula), estirándolos para que tengan todo el mismo tamaño. El **GridLayout** es adecuado para hacer tableros, calculadoras en que todos los botones son iguales, etc.

```
// Creación de los botones
JButton boton[] = new JButton[9];
for (int i=0;i<9;i++)
   boton[i] = new JButton(Integer.toString(i));

// Colocación en el contenedor
contenedor.setLayout (new GridLayout (3,3));  // 3 filas y 3 columnas
for (int i=0;i<9;i++)
    contenedor.add (boton[i]);  // Añade los botones de 1 en 1.
```

Para la práctica seguiremos el tutorial de WindowBuilder:

http://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=55&codigo=128&inicio=40