

1.00 Clase 15

Construcción de interfaces con Swing

Temas

- En esta clase, examinaremos cómo construir interfaces más complejas con Swing.
- No haremos una demostración de muchos componentes; consulte <http://java.sun.com/docs/books/tutorial/uiswing/components/components.html>
- Examinaremos:
 - cómo construir una interfaz componente a componente y contenedor a contenedor;
 - cómo utilizar la gestión de diseño para configurar su GUI;
 - cómo Swing utiliza eventos para permitir que la GUI interactúe con los usuarios;
- La clase de aprendizaje activo del viernes tratará este último tema.

Cómo construir las GUI

- Más adelante, a lo largo del curso, diseñaremos una calculadora sencilla.
- Construiremos la GUI anidando los componentes más simples en contenedores, y combinando, luego, los contenedores con otros más grandes.
- `JPanel` es el instrumento de trabajo de las interfaces más complejas. Resulta ser:
 - un buen contenedor de uso general;
 - una superficie estándar de diseño (Clase 17);
 - una clase básica para muchos componentes compuestos.

GUI de una calculadora

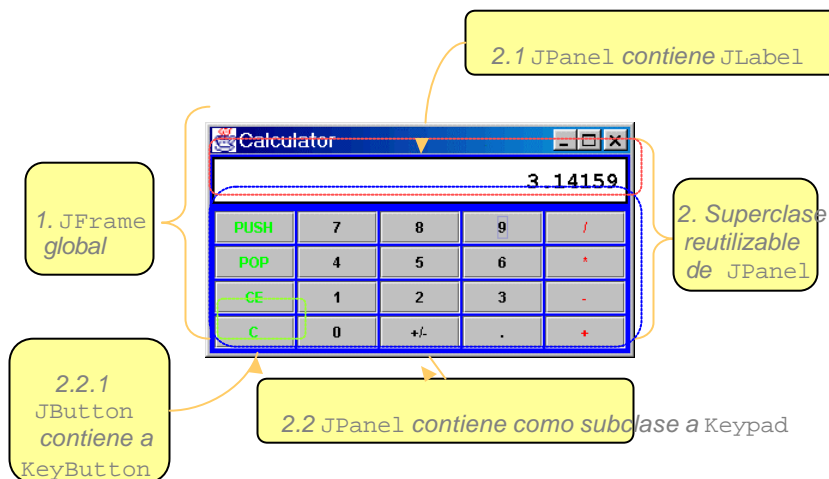
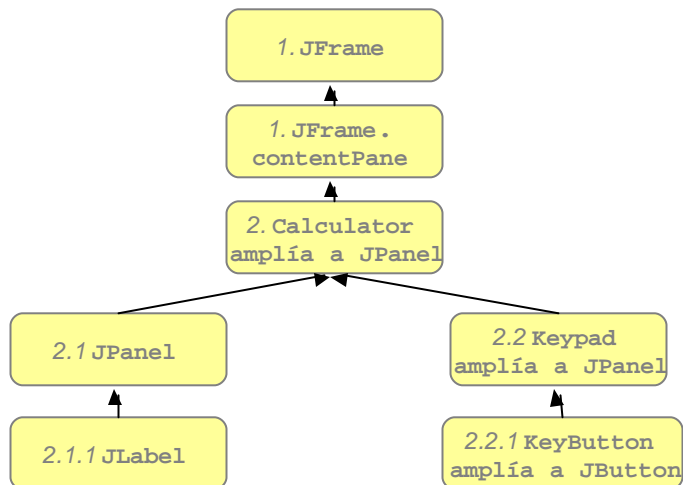


Diagrama de la GUI de la calculadora



Construcción de la calculadora, 1

```
public class CalculatorApp {
    public static void main( String[] args ) {
        JFrame top = new JFrame("Calculadora" );
        top.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE );
        Calculator calc = new Calculator();
        top.getContentPane().add( calc,
            BorderLayout.CENTER );

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

Construcción de la calculadora, 2

```
public class Calculator
    extends javax.swing.JPanel {
    private JLabel display = null;
    static final String EMPTYSTR = " ";
    public Calculator() {
        setLayout( new BorderLayout(4, 4) );
        JPanel displayPanel = new JPanel();
        displayPanel.setLayout( new FlowLayout (
            FlowLayout.RIGHT ) );
        display = new JLabel( EMPTYSTR );
        displayPanel.add( display );
        add( displayPanel, BorderLayout.NORTH );
        add( new Keypad(), BorderLayout.CENTER );
    }
}
```

2.1

2.1.1

2.2

¿Por qué declarar un componente como miembro de una instancia?

- JLabel display es un miembro de una instancia, pero no lo es el panel (JPanel displayPanel) que lo contiene.
- Declaramos los componentes como miembros cuando nos hace falta referirnos a ellos, normalmente para cambiarlos, después de que sean creados por el constructor.
- JLabel display mostrará los resultados de los cálculos. Vamos a necesitar una referencia para poder actualizar la pantalla de la calculadora.

Construcción de la calculadora, 3

2.2

```
class Keypad extends JPanel {
    Keypad() {
        setLayout( new GridLayout( 0, 5, 2, 2 ));
        setBackground( Color.blue );
        . . .
        add( new KeyButton( . . . ));
        . . .
    }
}
class KeyButton extends JButton { . . . }
```

2.2.1

Gestión de diseño, 1

- La gestión de diseño es el proceso por el cual se determina el tamaño y la ubicación de los componentes de un contenedor.
- Los contenedores de Java no controlan su distribución interna. Delegan esta tarea a su gestor de diseño, una instancia de otra clase.
- Si no le agrada el gestor de diseño del contenedor que viene por defecto, puede cambiarlo.

```
Container content = getContentPane();
content.setLayout( new FlowLayout() );
```

Gestión de diseño, 2

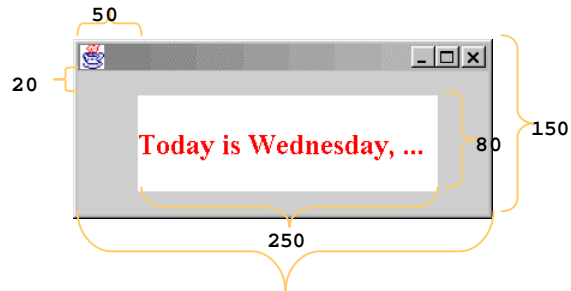
- **La gestión de diseño es intrínsecamente recursiva y avanza de arriba hacia abajo en la jerarquía de contenedores.**
- **Si un contenedor contiene a otro, el externo no puede ubicar al contenedor interno, ni redimensionarse a sí mismo hasta conocer el tamaño que necesitará el interno.**

Cómo desactivar la gestión de diseño

- **Aunque los gestores de diseño pueden añadir una complejidad adicional, tienen una buena razón de ser.**
- **El efecto de la gestión de diseño se puede desactivar eliminando el gestor de diseño del contenedor mediante una llamada a `setLayout(null)`.**
- **El usuario deberá entonces distribuir explícitamente cada componente en coordenadas absolutas de píxel, mediante llamadas a `setSize()` y `setLocation()` o a través de una sola llamada a `setBounds()`.**
- **El problema que presenta esta estrategia es su falta de flexibilidad.**

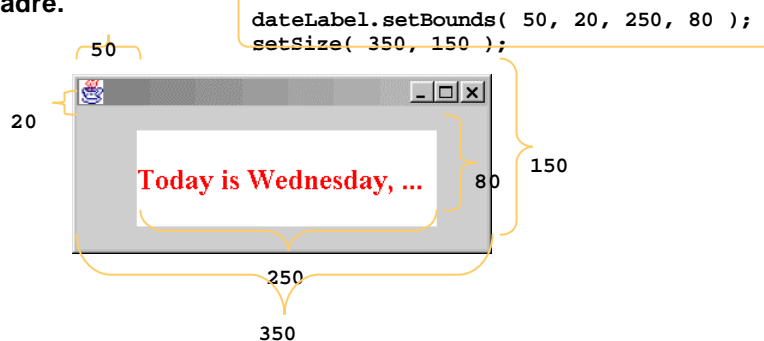
Today sin gestión de diseño

```
public Today1a( String dStr ) {  
    . . .  
    getContentPane().setLayout( null );  
    getContentPane().add( dateLabel );  
    dateLabel.setBounds( 50, 20, 250, 80 );  
    setSize( 350, 150 );  
}
```



Coordenadas

- Se miden en píxeles (ej. 640x480, 1024x768, etc.).
- El origen (0,0) está en la esquina superior izquierda.
- El eje X va de izquierda a derecha, y el eje Y de arriba hacia abajo.
- Cada componente está ubicado en el sistema de coordenadas padre.



3 buenas razones para utilizar la gestión de diseño

1. Es frecuente que no sepa el tamaño que tendrá su aplicación. Incluso si llama a `setSize()`, el usuario aún podrá redimensionar físicamente el tamaño de la ventana de la aplicación.
2. Java conoce mejor que usted el tamaño que tendrán los componentes. Por ejemplo, es difícil medir el tamaño de un `JLabel`, salvo haciendo pruebas. Si obtiene el tamaño correcto en un sistema y, luego, lo ejecuta en otro con un conjunto de fuentes distinto, el tamaño de `JLabel` no se determinará correctamente.
3. Una vez que haya diseñado una GUI, es posible que desee hacer cambios que pongan en peligro un diseño hecho a mano. Si está utilizando un gestor de diseño, el nuevo diseño aparecerá automáticamente, pero si está diseñando los botones a mano, tendrá que hacer frente a una tarea pesada.

Métodos Size

- Los componentes comunican sus necesidades de diseño al gestor de diseño del contenedor que los engloba mediante los métodos:
 - `public Dimension getMinimumSize()`
 - `public Dimension getPreferredSize()`
 - `public Dimension getMaximumSize()`
- Existen tres métodos "set" correspondientes que permiten cambiar las *pistas de tamaño* de un componente.
 - `public Dimension setMinimumSize(Dimension d)`
 - `Dimension d = new Dimension(int width, int height)`
 - `public Dimension setPreferredSize(Dimension d)`
 - `public Dimension setMaximumSize(Dimension d)`

Métodos *Size*, 2

- Utilizar estos métodos para cambiar el tamaño de un componente es generalmente bastante más seguro que utilizar el método explícito `setSize()`.
- El efecto del método `setSize()` solamente durará hasta que el entorno de Java diseñe nuevamente el contenedor.
- La mayoría de los componentes proporcionan buenas implementaciones por defecto para los métodos `get`. Por ejemplo, un botón determina su tamaño preferido para adaptar su imagen y / o su tamaño de fuente y etiqueta.

¿Cuándo tiene lugar el diseño?

Swing rediseñará automáticamente una GUI en las siguientes circunstancias:

1. cuando se hace visible por primera vez;
2. cuando se añade o se borra un componente;
3. cuando cambia el tamaño de un componente porque el usuario lo modifica físicamente (redimensionando el tamaño de la ventana) o por un cambio de contenidos (por ejemplo, el cambio de una etiqueta).

revalidate() y pack()

- Sin embargo, si usted cambia el tamaño de un componente una vez que se haya hecho visible, modificando una de sus propiedades de tamaño, no se activará un nuevo diseño hasta que llame a `revalidate()` en el componente.
- En el único momento en el que se debe pedir explícitamente un nuevo diseño es cuando se utiliza un `JFrame`. Un *frame* (marco) no se ajustará a su contenido a menos que se llame a `pack()`. También puede establecer su tamaño utilizando `setSize()`, pero los *frames* no poseen propiedades de tamaño. Si no llama a `pack()` o a `setSize()`, el *frame* se encogerá tanto que le costará bastante encontrarlo en la pantalla.

Gestores de diseño

- Swing facilita seis clases de gestión de diseño. Abarcan desde el simple `FlowLayout`, hasta el flexible, aunque a veces frustrante, `GridBagLayout`. Cada clase gestora implementa una *política* de diseño concreta.
- Los contenedores de Swing poseen gestores de diseño por defecto. Un panel de contenidos de `JFrame` utiliza `BorderLayout` y un `JPanel` utiliza `FlowLayout`.

FlowLayout

- `FlowLayout`, el más simple de los gestores, únicamente añade componentes de izquierda a derecha hasta que no caben más en la anchura del contenedor.
- Comienza, entonces, una segunda línea de componentes, la completa; comienza una tercera, y así sucesivamente.
- Cada línea está centrada dentro del contenedor que la encierra.
- `FlowLayout` respeta el tamaño preferido para cada componente y lo utilizará para anular un tamaño establecido mediante `setSize()` .

Ejemplo de `FlowLayout`, 1

```
public class Flow extends JFrame {
    private Font labelFont;
    private Border labelBorder;

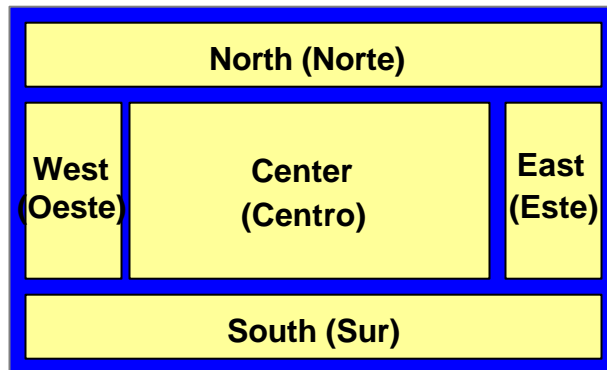
    public Flow( ) {
        setDefaultCloseOperation( EXIT_ON_CLOSE );
        labelFont = new Font( "SansSerif",Font.BOLD,24 );
        labelBorder =
            BorderFactory.createLineBorder(Color.red,1);
        getContentPane().setLayout( new FlowLayout() );
        setSize(200, 200 );
    }
}
```

Ejemplo de FlowLayout ,2

```
public void addLabel( String labelStr ) {
    JLabel label = new JLabel( labelStr );
    label.setFont( labelFont );
    label.setBorder( labelBorder );
    getContentPane().add( label );
}

public static void main (String args[])
{
    Flow flow = new Flow();
    flow.addLabel( "uno" );
    . . .
    flow.setVisible( true );
}
```

Zonas de BorderLayout



Política de BorderLayout

- Se especifica la zona mediante un segundo argumento de tipo `String` en el método `add()`. Por ejemplo, la siguiente línea de código añade el botón etiquetado "DoIt" en el centro de un contenedor.

```
add( new JButton( "DoIt" ), "Center" );  
// "Center" == BorderLayout.CENTER
```

- Un `BorderLayout` puede ampliar horizontalmente sus componentes Norte y Sur (si es que existen), verticalmente sus componentes Este y Oeste, y de ambos modos, el componente Centro, para ajustar el tamaño de su contenedor y las restricciones de sus otros cuatro sectores.
- Esto puede ser útil. Si coloca un `JPanel` en la zona Centro de un contenedor gestionado por un `BorderLayout`, el gestor siempre redimensionará el `JPanel` para aceptar todo el espacio extra, que es lo que normalmente se pretende si se está utilizando como superficie de diseño.

Grid Layout (Diseño en cuadrículas)

- La clase `GridLayout` es un gestor de diseño que dispone los componentes de un contenedor en una cuadrícula rectangular.
- El contenedor se divide en rectángulos iguales, y se ubica un componente en cada rectángulo.
- En el constructor normal se especifican el número de filas o columnas, pero no ambas. La que no es cero posee un número fijo de elementos; la otra aumenta a medida que se añaden componentes.

```
getContentPane().setLayout( new GridLayout(0,2));
```

La línea de código anterior diseñaría un `JFrame` en forma de cuadrícula de dos columnas. El número de filas dependería del número de componentes añadidos.

El modelo de eventos en Java

- Hasta ahora, para mostrar información nos hemos centrado en las GUI (con una excepción).
- ¿Cómo *interactúan* las GUI con los usuarios? ¿Cómo reconocen las aplicaciones que el usuario ha hecho algo?
- En Java, esto depende de 3 conceptos relacionados:
 - eventos: objetos que representan la acción de un usuario con el sistema;
 - fuentes de eventos: en Swing, éstos son componentes que pueden reconocer la acción del usuario, como un botón o un campo de texto editable;
 - oyentes de eventos: objetos que pueden responder cuando se produce un evento.

Fuentes de eventos

- Las fuentes de eventos pueden generar eventos.
- Las que les resultarán más interesantes son las subclases de `JComponents` como `JButtons` y `JPanels`.
- Averiguará el tipo de eventos que pueden generar cuando lea Javadoc.

Oyentes de eventos

- Un objeto se convierte en un oyente de eventos cuando su clase implementa una interfaz de oyente de eventos. Por ejemplo:

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

el tipo de oyente

el método que debe implementar

- Si relacionamos al oyente de eventos con la fuente de eventos, el primero es invocado cuando se produce el evento.

Eventos

- Los eventos son instancias de clases simples que facilitan información acerca de lo sucedido.
- Por ejemplo, las instancias de `MouseEvent` poseen los métodos `getX()` y `getY()` que le indicarán dónde tuvo lugar el evento de ratón (ej., el pulsado del ratón).
- Todos los métodos oyentes de eventos aceptan un evento como argumento.

¿Cómo planifico la recepción de un evento?

1. Piense en qué tipo de evento está interesado y de qué componente proviene.
2. Decida qué objeto va a *manipular* (sobre el que va a actuar) el evento.
3. Determine la interfaz de oyente adecuada para el tipo de evento en el que usted está interesado.
4. Escriba el/los método/s oyente/s adecuado/s para la clase del objeto manipulador.
5. Utilice un método `addEventListener()` para registrar al oyente con la fuente de evento.

La aplicación Hello, 1

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

public class Hello extends JFrame
    implements ActionListener
{
    private JButton button;
    private int state = 0;

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

The diagram illustrates the annotations in the code. A blue oval highlights the class name 'Hello' in the line 'public class Hello extends JFrame'. Another blue oval highlights the interface 'ActionListener' in the line 'implements ActionListener'. A third blue oval highlights the class 'JButton' in the line 'private JButton button;'. Three orange circles with arrows point to these elements: circle 1 points to 'Hello', circle 2 points to 'ActionListener', and circle 3 points to 'JButton'.

La aplicación Hello, 2

```
public Hello() {  
    setDefaultCloseOperation( EXIT_ON_CLOSE );  
    button = new JButton( "Hola" );  
    button.setFont( new Font( "SansSerif",  
                             Font.BOLD, 24 ) );  
    button.addActionListener( this );  
    getContentPane().add( button, "Center" );  
    setSize( 200, 200 );  
}
```

La aplicación Hello, 3

```
public void actionPerformed( ActionEvent e ) {  
    4 if ( state == 0 ) {  
        button.setText( "Goodbye" );  
        state++;  
    } else {  
        System.exit( 0 );  
    }  
}
```