EBook Gratis

APRENDIZAJE swing

Free unaffiliated eBook created from
Stack Overflow contributors.

#swing

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swing

It is an unofficial and free swing ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official swing.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el swing

Observaciones

Swing ha sido reemplazado por JavaFX . Oracle generalmente recomienda desarrollar nuevas aplicaciones con JavaFX. Aún así: Swing será compatible con Java en el futuro inmediato. JavaFX también se integra bien con Swing, para permitir la transición de las aplicaciones sin problemas.

Se recomienda encarecidamente tener la mayoría de sus componentes Swing en el subproceso de despacho de eventos. Es fácil olvidarse de agrupar la configuración de su GUI en una llamada `invokeLater` . De la documentación de Java:

> El código de manejo de eventos Swing se ejecuta en un subproceso especial conocido como el subproceso de distribución de eventos. La mayoría del código que invoca los métodos Swing también se ejecuta en este hilo. Esto es necesario porque la mayoría de los métodos de objetos Swing no son "seguros para subprocesos": invocarlos desde múltiples subprocesos puede provocar interferencias en los subprocesos o errores de consistencia de la memoria. Algunos métodos de componentes Swing están etiquetados como "seguro para subprocesos" en la especificación de la API; Estos pueden ser invocados de forma segura desde cualquier hilo. Todos los demás métodos del componente Swing deben invocarse desde el hilo de despacho de eventos. Los programas que ignoran esta regla pueden funcionar correctamente la mayor parte del tiempo, pero están sujetos a errores impredecibles que son difíciles de reproducir.

Además, a menos que sea por una buena razón, *siempre* asegúrese de llamar a `setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)` o de lo contrario podría tener que lidiar con una pérdida de memoria si olvida destruir la JVM.

Examples

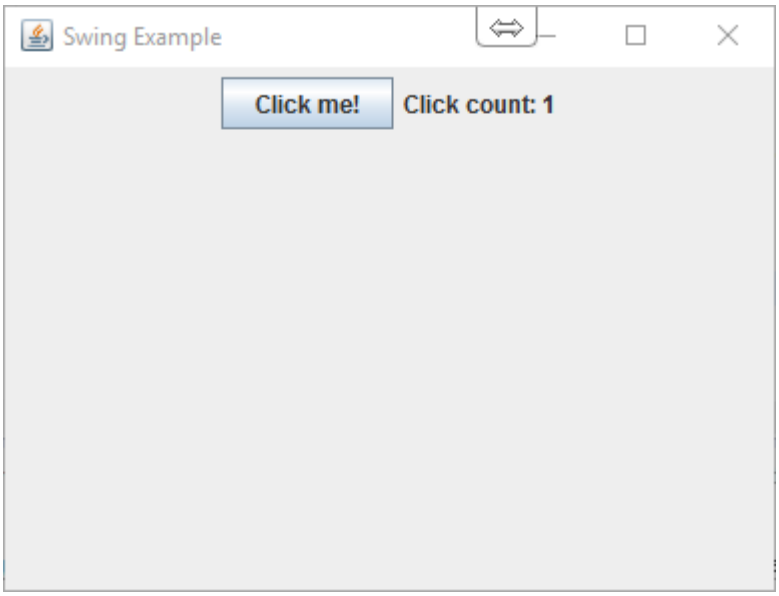
Incrementando con un botón.

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

/**
 * A very simple Swing example.
 */
public class SwingExample {
    /**
     * The number of times the user has clicked the button.
     */
```

```
    private long clickCount;

    /**
     * The main method: starting point of this application.
     *
     * @param arguments the unused command-line arguments.
     */
    public static void main(final String[] arguments) {
        new SwingExample().run();
    }

    /**
     * Schedule a job for the event-dispatching thread: create and show this
     * application's GUI.
     */
    private void run() {
        SwingUtilities.invokeLater(this::createAndShowGui);
    }

    /**
     * Create the simple GUI for this application and make it visible.
     */
    private void createAndShowGui() {
        // Create the frame and make sure the application exits when the user closes
        // the frame.
        JFrame mainFrame = new JFrame("Counter");
        mainFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // Add a simple button and label.
        JPanel panel = new JPanel();
        JButton button = new JButton("Click me!");
        JLabel label = new JLabel("Click count: " + clickCount);
        panel.add(button);
        panel.add(label);
        mainFrame.getContentPane().add(panel);

        // Add an action listener to the button to increment the count displayed by
        // the label.
        button.addActionListener(actionEvent -> {
            clickCount++;
            label.setText("Click count: " + clickCount);
        });

        // Size the frame.
        mainFrame.setBounds(80, 60, 400, 300);
        //Center on screen
        mainFrame.setLocationRelativeTo(null);
        //Display frame
        mainFrame.setVisible(true);
    }
}
```

Resultado

Como el botón etiquetado "Haga clic en mí!" se presiona el número de clics aumentará en uno:



"¡Hola Mundo!" en titulo de ventana con lambda

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Hello World!");
            frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            frame.setSize(200, 100);
            frame.setVisible(true);
        });
    }
}
```

Dentro del método `main` :
En la primera línea se llama a `SwingUtilities.invokeLater` pasa una expresión lambda con un bloque de código `() -> {...}` . Esto ejecuta la expresión lambda pasada en la EDT, que es la abreviatura de Subproceso de distribución de eventos, en lugar del subproceso principal. Esto es necesario, porque dentro del bloque de código de la expresión lambda, hay componentes de Swing que se crearán y actualizarán.

Dentro del bloque de código de la expresión lambda:
En la primera línea, una nueva instancia de `JFrame` llamada `frame` se crea utilizando el `new JFrame("Hello World!")` . Esto crea una instancia de ventana con "Hello World!" en su título. Luego, en la segunda línea, el `frame` se configura como `EXIT_ON_CLOSE` . De lo contrario, la ventana se cerrará, pero la ejecución del programa permanecerá activa. La tercera línea configura la instancia de `frame` para que tenga 200 píxeles de ancho y 100 píxeles de altura utilizando el método `setSize` . Hasta ahora la ejecución no mostrará nada en absoluto. Solo después de llamar a `setVisible(true)` en la cuarta línea, la instancia de `frame` está configurada para aparecer en la pantalla.

"¡Hola Mundo!" en titulo de ventana con compatibilidad

Usando `java.lang.Runnable` hacemos nuestro "¡Hola mundo!" Ejemplo disponible para usuarios de Java con versiones que se remontan a la versión 1.2:

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JFrame frame = new JFrame("Hello World!");
                frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                frame.setSize(200, 100);
                frame.setVisible(true);
            }
        });
    }
}
```

Lea Empezando con el swing en línea: https://riptutorial.com/es/swing/topic/2191/empezando-con-el-swing

Capítulo 2: Diseño de cuadrícula

Examples

Cómo funciona GridLayout

Un `GridLayout` es un administrador de diseño que coloca componentes dentro de una cuadrícula con el mismo tamaño de celda. Puede establecer el número de filas, columnas, la brecha horizontal y la brecha vertical utilizando los siguientes métodos:

- `setRows(int rows)`
- `setColumns(int columns)`
- `setHgap(int hgap)`
- `setVgap(int vgap)`

O puedes configurarlos con los siguientes constructores:

- `GridLayout(int rows, int columns)`
- `GridLayout(int rows, int columns, int hgap, int vgap)`

Si el número de filas o columnas es desconocido, puede establecer la variable respectiva en `0` . Por ejemplo:

```
new GridLayout(0, 3)
```

Esto hará que `GridLayout` tenga 3 columnas y tantas filas como sea necesario.

El siguiente ejemplo muestra cómo un `GridLayout` presenta componentes con diferentes valores para filas, columnas, espacio horizontal, espacio vertical vertical y tamaño de pantalla.

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import javax.swing.WindowConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class GridLayoutExample {

    private GridLayout gridLayout;
    private JPanel gridPanel, contentPane;
    private JSpinner rowsSpinner, columnsSpinner, hgapSpinner, vgapSpinner;
```

```
    public void createAndShowGUI() {
        gridLayout = new GridLayout(5, 5, 3, 3);

        gridPanel = new JPanel(gridLayout);

        final ChangeListener rowsColumnsListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setRows((int) rowsSpinner.getValue());
                gridLayout.setColumns((int) columnsSpinner.getValue());
                fillGrid();
            }
        };

        final ChangeListener gapListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setHgap((int) hgapSpinner.getValue());
                gridLayout.setVgap((int) vgapSpinner.getValue());
                gridLayout.layoutContainer(gridPanel);
                contentPane.revalidate();
                contentPane.repaint();
            }
        };

        rowsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getRows(), 1, 10, 1));
        rowsSpinner.addChangeListener(rowsColumnsListener);

        columnsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getColumns(), 1, 10,
1));
        columnsSpinner.addChangeListener(rowsColumnsListener);

        hgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getHgap(), 0, 50, 1));
        hgapSpinner.addChangeListener(gapListener);

        vgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getVgap(), 0, 50, 1));
        vgapSpinner.addChangeListener(gapListener);

        JPanel actionPanel = new JPanel();
        actionPanel.add(new JLabel("Rows:"));
        actionPanel.add(rowsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Columns:"));
        actionPanel.add(columnsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Horizontal gap:"));
        actionPanel.add(hgapSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Vertical gap:"));
        actionPanel.add(vgapSpinner);

        contentPane = new JPanel(new BorderLayout(0, 10));
        contentPane.add(gridPanel);
        contentPane.add(actionPanel, BorderLayout.SOUTH);

        fillGrid();

        JFrame frame = new JFrame("GridLayout Example");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setContentPane(contentPane);
        frame.setSize(640, 480);
```

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

    private void fillGrid() {
        gridPanel.removeAll();
        for (int row = 0; row < gridLayout.getRows(); row++) {
            for (int col = 0; col < gridLayout.getColumns(); col++) {
                JLabel label = new JLabel("Row: " + row + " Column: " + col);
                label.setHorizontalAlignment(JLabel.CENTER);
                label.setBorder(BorderFactory.createLineBorder(Color.GRAY));
                gridPanel.add(label);
            }
        }
        contentPane.revalidate();
        contentPane.repaint();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                new GridLayoutExample().createAndShowGUI();
            }
        });
    }

}
```

Lea Diseño de cuadrícula en línea: https://riptutorial.com/es/swing/topic/2780/diseno-de-cuadricula

Capítulo 3: Gestión de diseño

Examples

Diseño de la frontera

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

JPanel root = new JPanel(new BorderLayout());

root.add(new JButton("East"), EAST);
root.add(new JButton("West"), WEST);
root.add(new JButton("North"), NORTH);
root.add(new JButton("South"), SOUTH);
root.add(new JButton("Center"), CENTER);

JFrame frame = new JFrame();
frame.setContentPane(root);
frame.pack();
frame.setVisible(true);
```

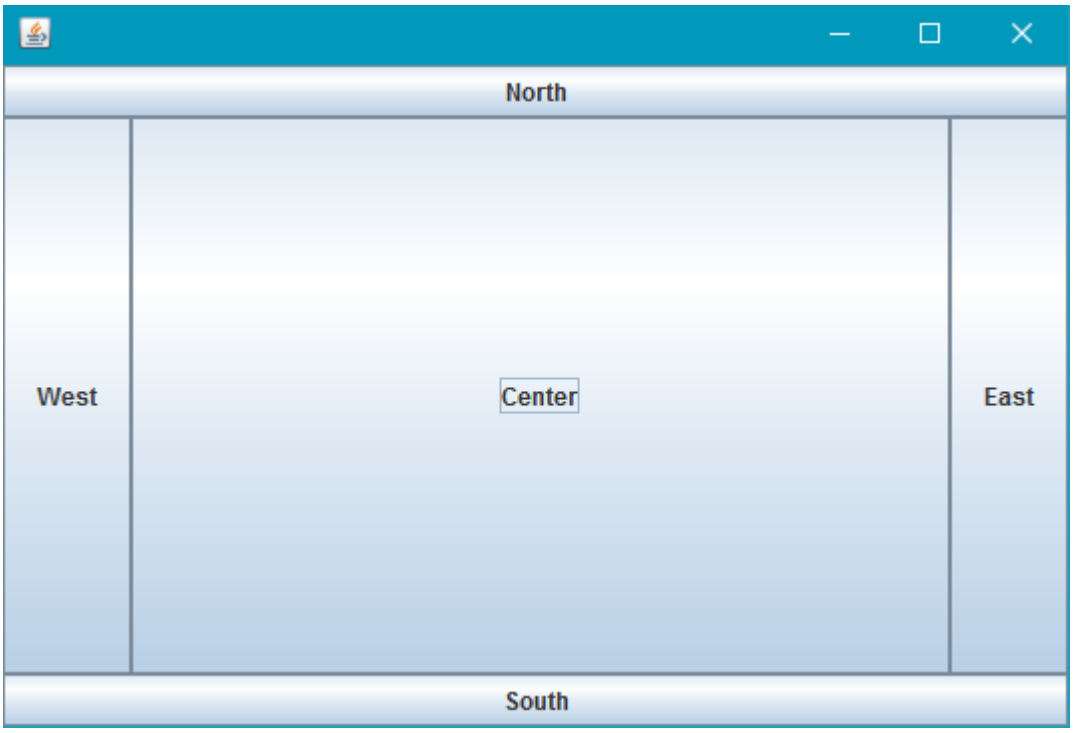
El diseño de bordes es uno de los administradores de diseño más simples. La forma de usar un administrador de diseño es establecer el administrador de un `JPanel` .

Las ranuras de diseño de bordes siguen las siguientes reglas:

- Norte y sur: altura preferida
- Este y Oeste: ancho preferido
- Centro: máximo espacio restante

En `BorderLayout` ranuras también pueden estar vacías. El administrador de diseño compensará automáticamente los espacios vacíos, cambiando el tamaño cuando sea necesario.

Aquí es cómo se ve este ejemplo:



Diseño de flujo

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

public class FlowExample {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JPanel panel = new JPanel();
                panel.setLayout(new FlowLayout());

                panel.add(new JButton("One"));
                panel.add(new JButton("Two"));
                panel.add(new JButton("Three"));
                panel.add(new JButton("Four"));
                panel.add(new JButton("Five"));

                JFrame frame = new JFrame();
                frame.setContentPane(Panel);
                frame.pack();
                frame.setVisible(true);
            }
        });
    }
}
```

El diseño de flujo es el administrador de diseño más simple que Swing tiene para ofrecer. El diseño de flujo intenta colocar todo en una línea, y si el diseño desborda el ancho, se ajustará a la línea. El orden se especifica por el orden en que agrega componentes a su panel.

Capturas de pantalla:

Diseño de cuadrícula

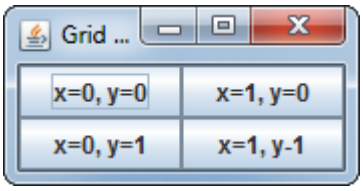
`GridLayout` permite organizar componentes en forma de una cuadrícula.

Pase el número de filas y columnas que desea que tenga la cuadrícula al constructor de `GridLayout` ; por ejemplo, el `new GridLayout(3, 2)` creará un `GridLayout` con 3 filas y 2 columnas.

Al agregar componentes a un contenedor con `GridLayout` , los componentes se agregarán fila por fila, de izquierda a derecha:

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

public class Example {
    public static void main(String[] args){
        SwingUtilities.invokeLater(Example::createAndShowJFrame);
    }

    private static void createAndShowJFrame(){
        JFrame jFrame = new JFrame("Grid Layout Example");

        // Create layout and add buttons to show restraints
        JPanel jPanel = new JPanel(new GridLayout(2, 2));
        jPanel.add(new JButton("x=0, y=0"));
        jPanel.add(new JButton("x=1, y=0"));
        jPanel.add(new JButton("x=0, y=1"));
        jPanel.add(new JButton("x=1, y-1"));

        jFrame.setContentPane(jPanel);
        jFrame.pack();
        jFrame.setLocationRelativeTo(null);
        jFrame.setVisible(true);
    }
}
```

Esto crea y muestra un `JFrame` que se parece a:



Una descripción más detallada está disponible: GridLayout

Lea Gestión de diseño en línea: https://riptutorial.com/es/swing/topic/5417/gestion-de-diseno

Capítulo 4: Gráficos

Examples

Usando la clase de gráficos

Introducción

La clase `Graphics` permite dibujar en componentes java como un `Jpanel` , se puede usar para dibujar cadenas, líneas, formas e imágenes. Esto se hace *anulando* el `paintComponent(Graphics g)` del `JComponent` que está dibujando usando el objeto `Graphics` recibido como argumento para hacer el dibujo:

`Board` clase

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

public class Board extends JPanel{

    public Board() {
        setBackground(Color.WHITE);
    }

    @override
    public Dimension getPreferredSize() {
        return new Dimension(400, 400);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // draws a line diagonally across the screen
        g.drawLine(0, 0, 400, 400);
        // draws a rectangle around "hello there!"
        g.drawRect(140, 180, 115, 25);
    }
}
```

clase envoltura `DrawingCanvas`

```
import javax.swing.*;

public class DrawingCanvas extends JFrame {

    public DrawingCanvas() {

        Board board = new Board();

        add(board); // adds the Board to our JFrame
```

```
        pack(); // sets JFrame dimension to contain subcomponents

        setResizable(false);
        setTitle("Graphics Test");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        setLocationRelativeTo(null); // centers window on screen
    }

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

Colores

Para dibujar formas con diferentes colores, debe configurar el color del objeto `Graphics` antes de cada llamada de dibujo usando `setColor` :

```
g.setColor(Color.BLUE);  // draws a blue square
g.fillRect(10, 110, 100, 100);

g.setColor(Color.RED);  // draws a red circle
g.fillOval(10, 10, 100, 100);

g.setColor(Color.GREEN);  // draws a green triangle
int[] xPoints = {0, 200, 100};
int[] yPoints = {100, 100, 280};
g.fillPolygon(xPoints, yPoints, 3);
```

Dibujando imagenes

Las imágenes se pueden dibujar en un `JComponent` usando el método `drawImage` de la clase `Graphics` :

cargando una imagen

```
BufferedImage img;
try {
    img = ImageIO.read(new File("stackoverflow.jpg"));
} catch (IOException e) {
    throw new RuntimeException("Could not load image", e);
}
```

dibujando la imagen

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    int x = 0;
    int y = 0;

    g.drawImage(img, x, y, this);
}
```

Las `x` y las `y` especifican la ubicación de la **esquina superior izquierda** de la imagen.

Usando el método de repintado para crear una animación básica

La clase MyFrame amplía JFrame y también contiene el método principal

```
import javax.swing.JFrame;


public class MyFrame extends JFrame{

    //main method called on startup
    public static void main(String[] args) throws InterruptedException {

        //creates a frame window
        MyFrame frame = new MyFrame();

        //very basic game loop where the graphics are re-rendered
        while(true){
            frame.getPanel().repaint();

            //The program waits a while before rerendering
            Thread.sleep(12);
        }
    }

    //the MyPanel is the other class and it extends JPanel
    private MyPanel panel;

    //constructor that sets some basic staring values
    public MyFrame(){
        this.setSize(500, 500);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creates the MyPanel with paramaters of x=0 and y=0
        panel = new MyPanel(0,0);
        //adds the panel (which is a JComponent because it extends JPanel)
        //into the frame
        this.add(panel);
        //shows the frame window
        this.setVisible(true);
    }

    //gets the panel
    public MyPanel getPanel(){
        return panel;
    }

}
```

La clase MyPanel que extiende JPanel y tiene el método paintComponent

```
import java.awt.Graphics;
import javax.swing.JPanel;


public class MyPanel extends JPanel{

    //two int variables to store the x and y coordinate
    private int x;
    private int y;

    //construcor of the MyPanel class
    public MyPanel(int x, int y){
        this.x = x;
        this.y = y;
    }

    /*the method that deals with the graphics
        this method is called when the component is first loaded,
         when the component is resized and when the repaint() method is
        called for this component
    */
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);

        //changes the x and y varible values
        x++;
        y++;

        //draws a rectangle at the x and y values
        g.fillRect(x, y, 50, 50);
    }

}
```

Lea Gráficos en línea: https://riptutorial.com/es/swing/topic/5153/graficos

Capítulo 5: GridBag Layout

Sintaxis

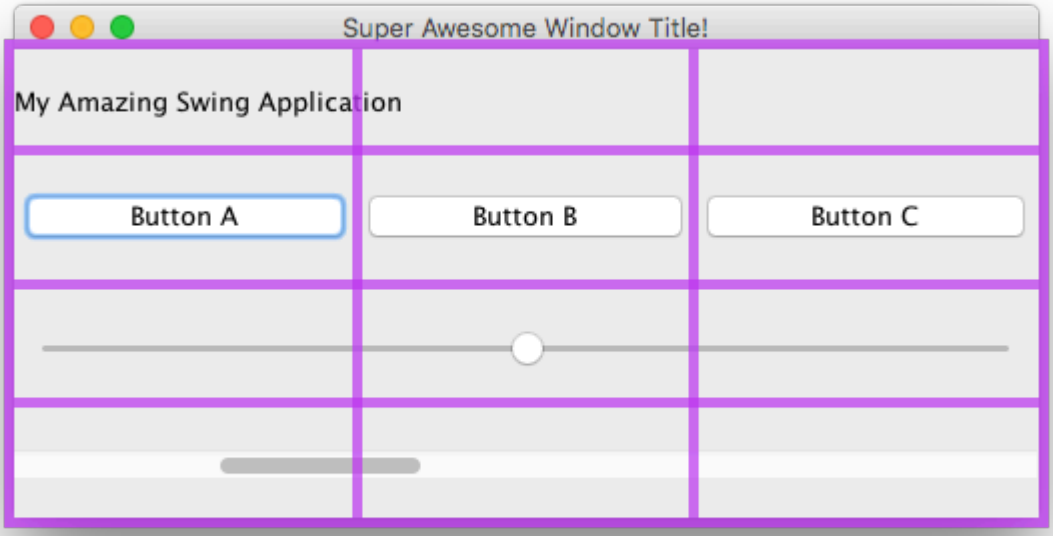
- frame.setLayout (new GridBagLayout ()); // Establecer GridBagLayout para el marco
- pane.setLayout (new GridBagLayout ()); // Establecer GridBagLayout para Panel
- Panel de JPanel = nuevo JPanel (nuevo GridBagLayout ()); // Establecer GridBagLayout para Panel
- GridBagConstraints c = new GridBagConstraints () // Inicializar una GridBagConstraint

Examples

¿Cómo funciona GridBagLayout?

Los diseños se utilizan cuando usted desea que sus componentes no solo se muestren uno junto al otro. El `GridBagLayout` es útil, ya que divide su ventana en filas y columnas, y usted decide en qué fila y columna colocar los componentes, así como en cuántas filas y columnas es grande el componente.

Tomemos esta ventana como ejemplo. Se han marcado las líneas de cuadrícula para mostrar el diseño.



Aquí, he creado 6 componentes, distribuidos utilizando un GridBagLayout.

Componente	Posición	tamaño
`JLabel` : "My Amazing Swing Application"	`0, 0`	`3, 1`

Componente	Posición	tamaño
`JButton` : "Botón A"	`0, 1`	`1, 1`
`JButton` : "Botón B"	`1, 1`	`1, 1`
`JButton` : "Botón C"	`2, 1`	`1, 1`
`JSlider`	`0, 2`	`3, 1`
`JScrollBar`	`0, 3`	`3, 1`

Tenga en cuenta que la posición `0, 0` está en la parte superior izquierda: los valores de `x` (columna) aumentan de izquierda a derecha, los valores de `y` (fila) aumentan de arriba a abajo.

Para comenzar a diseñar los componentes en un `GridBagLayout` , primero establezca el diseño de su `JFrame` o panel de contenido.

```
frame.setLayout(new GridBagLayout());
//OR
pane.setLayout(new GridBagLayout());
//OR
JPanel pane = new JPanel(new GridBagLayout()); //Add the layout when creating your content
pane
```

Tenga en cuenta que nunca define el tamaño de la cuadrícula. Esto se hace automáticamente a medida que agrega sus componentes.

Luego, deberá crear un objeto `GridBagConstraints` .

```
GridBagConstraints c = new GridBagConstraints();
```

Para asegurarse de que sus componentes llenen el tamaño de la ventana, es posible que desee establecer el peso de todos los componentes en 1. El peso se utiliza para determinar cómo distribuir el espacio entre columnas y filas.

```
c.weightx = 1;
c.weighty = 1;
```

Otra cosa que quizás desee hacer es asegurarse de que los componentes ocupen todo el espacio horizontal que puedan.

```
c.fill = GridBagConstraints.HORIZONTAL;
```

También puede establecer otras opciones de relleno si lo desea.

```
GridBagConstraints.NONE //Don't fill components at all
GridBagConstraints.HORIZONTAL //Fill components horizontally
GridBagConstraints.VERTICAL //Fill components vertically
GridBagConstraints.BOTH //Fill components horizontally and vertically
```

Al crear componentes, querrá establecer dónde debe ir en la cuadrícula y cuántos mosaicos de cuadrícula debe usar. Por ejemplo, para colocar un botón en la tercera fila en la segunda columna y ocupar un espacio de cuadrícula de 5 x 5, haga lo siguiente. Tenga en cuenta que la cuadrícula comienza en `0, 0` , no `1, 1` .

```
JButton button = new JButton("Fancy Button!");
c.gridx = 2;
c.gridy = 1;
c.gridwidth = 5;
c.gridheight = 5;
pane.add(buttonA, c);
```

Al agregar componentes a su ventana, recuerde pasar las restricciones como un parámetro. Esto se puede ver en la última línea en el ejemplo de código anterior.

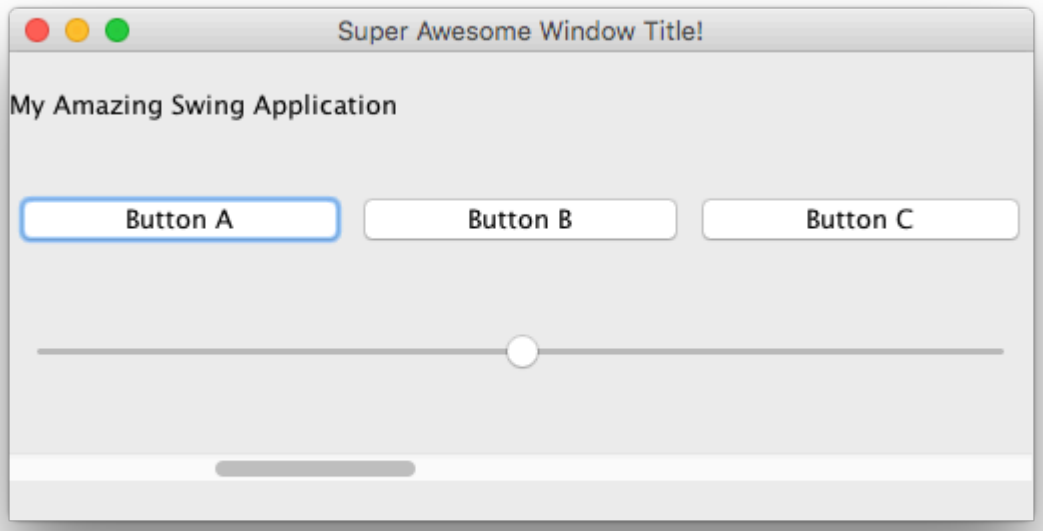
Puede reutilizar las mismas `GridBagConstraints` para cada componente; cambiarlo después de agregar un componente no cambia el componente agregado previamente.

Ejemplo

Aquí está el código para el ejemplo al comienzo de esta sección.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(new GridBagLayout()); //Create a pane to house all content, and give
it a GridBagLayout
frame.setContentPane(pane);

GridBagConstraints c = new GridBagConstraints();
c.weightx = 1;
c.weighty = 1;
c.fill = GridBagConstraints.HORIZONTAL;

JLabel headerLabel = new JLabel("My Amazing Swing Application");
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 0;
pane.add(headerLabel, c);

JButton buttonA = new JButton("Button A");
c.gridx = 0;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonA, c);

JButton buttonB = new JButton("Button B");
c.gridx = 1;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonB, c);
```

```
JButton buttonC = new JButton("Button C");
c.gridx = 2;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonC, c);

JSlider slider = new JSlider(0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 2;
pane.add(slider, c);

JScrollBar scrollBar = new JScrollBar(JScrollBar.HORIZONTAL, 20, 20, 0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 3;
pane.add(scrollBar, c);

frame.setVisible(true); //Show the window
```



Lea GridBag Layout en línea: https://riptutorial.com/es/swing/topic/3698/gridbag-layout

Capítulo 6: JList

Examples

Modificar los elementos seleccionados en un JList.

Dado un `JList` como

```
JList myList = new JList(items);
```

los elementos seleccionados en la lista se pueden modificar a través de `ListSelectionModel` de `JList` :

```
ListSelectionModel sm = myList.getSelectionModel();
sm.clearSelection();                        // clears the selection
sm.setSelectionInterval(index, index);    // Sets a selection interval
                                          // (single element, in this case)
```

Alternativamente, `JList` también proporciona algunos métodos convenientes para manipular directamente los índices seleccionados:

```
myList.setSelectionIndex(index);            // sets one selected index
                                            // could be used to define the Default Selection

myList.setSelectedIndices(arrayOfIndexes);  // sets all indexes contained in
                                            // the array as selected
```

Lea JList en línea: https://riptutorial.com/es/swing/topic/5413/jlist

Capítulo 7: Lo esencial

Examples

Retrasar una tarea de UI por un período específico

Todas las operaciones relacionadas con Swing suceden en un subproceso dedicado (el EDT - **E** vent **D** ispatch **T** hread). Si este hilo se bloquea, la interfaz de usuario no responde.

Por lo tanto, si desea retrasar una operación, no puede usar `Thread.sleep` . Utilice un `javax.swing.Timer` en `javax.swing.Timer` lugar. Por ejemplo, el siguiente `Timer` invertirá el texto de un `JLabel`

```
  int delay = 2000;//specify the delay for the timer
  Timer timer = new Timer( delay, e -> {
    //The following code will be executed once the delay is reached
    String revertedText = new StringBuilder( label.getText() ).reverse().toString();
    label.setText( revertedText );
  } );
  timer.setRepeats( false );//make sure the timer only runs once
```

A continuación se ofrece un ejemplo completo de ejecución que utiliza este `Timer` : la interfaz de usuario contiene un botón y una etiqueta. Al presionar el botón se invertirá el texto de la etiqueta después de un retraso de 2 segundos

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

public final class DelayedExecutionExample {

  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }

  private static void showUI(){
    JFrame frame = new JFrame( "Delayed execution example" );

    JLabel label = new JLabel( "Hello world" );
    JButton button = new JButton( "Reverse text with delay" );
    button.addActionListener( event -> {
      button.setEnabled( false );
      //Instead of directly updating the label, we use a timer
      //This allows to introduce a delay, while keeping the EDT free
      int delay = 2000;
      Timer timer = new Timer( delay, e -> {
        String revertedText = new StringBuilder( label.getText() ).reverse().toString();
        label.setText( revertedText );
        button.setEnabled( true );
      } );
      timer.setRepeats( false );//make sure the timer only runs once
      timer.start();
    } );
```

```
    frame.add( label, BorderLayout.CENTER );
    frame.add( button, BorderLayout.SOUTH );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Repita una tarea UI en un intervalo fijo

La actualización del estado de un componente Swing debe ocurrir en el subproceso de distribución de eventos (EDT). El `javax.swing.Timer` activa su `ActionListener` en el EDT, por lo que es una buena opción para realizar operaciones de Swing.

El siguiente ejemplo actualiza el texto de un `JLabel` cada dos segundos:

```
//Use a timer to update the label at a fixed interval
int delay = 2000;
Timer timer = new Timer( delay, e -> {
  String revertedText = new StringBuilder( label.getText() ).reverse().toString();
  label.setText( revertedText );
} );
timer.start();
```

A continuación se ofrece un ejemplo completo de ejecución que utiliza este `Timer` : la interfaz de usuario contiene una etiqueta y el texto de la etiqueta se revertirá cada dos segundos.

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

public final class RepeatTaskFixedIntervalExample {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated task example" );
    JLabel label = new JLabel( "Hello world" );

    //Use a timer to update the label at a fixed interval
    int delay = 2000;
    Timer timer = new Timer( delay, e -> {
      String revertedText = new StringBuilder( label.getText() ).reverse().toString();
      label.setText( revertedText );
    } );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Ejecutar una tarea de UI un número fijo de veces

En el `ActionListener` adjunto a un `javax.swing.Timer` , puede hacer un seguimiento de la cantidad de veces que el `Timer` ejecutó el `ActionListener` . Una vez que se alcanza el número requerido de veces, puede usar el método `Timer#stop()` para detener el `Timer` .

```
Timer timer = new Timer( delay, new ActionListener() {
  private int counter = 0;
  @Override
  public void actionPerformed( ActionEvent e ) {
    counter++;//keep track of the number of times the Timer executed
    label.setText( counter + "" );
    if ( counter == 5 ){
      ( ( Timer ) e.getSource() ).stop();
    }
  }
});
```

A continuación se muestra un ejemplo completo de ejecución que utiliza este `Timer` : muestra una interfaz de usuario en la que el texto de la etiqueta contará de cero a cinco. Una vez que se llega a cinco, el `Timer` se detiene.

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

public final class RepeatFixedNumberOfTimes {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated fixed number of times example" );
    JLabel label = new JLabel( "0" );

    int delay = 2000;
    Timer timer = new Timer( delay, new ActionListener() {
      private int counter = 0;
      @Override
      public void actionPerformed( ActionEvent e ) {
        counter++;//keep track of the number of times the Timer executed
        label.setText( counter + "" );
        if ( counter == 5 ){
          //stop the Timer when we reach 5
          ( ( Timer ) e.getSource() ).stop();
        }
      }
    });
    timer.setInitialDelay( delay );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Creando tu primer JFrame

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class FrameCreator {

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame();
            //JFrames will not display without size being set
            frame.setSize(500, 500);

            JLabel label = new JLabel("Hello World");
            frame.add(label);

            frame.setVisible(true);
        });
    }

}
```

Como puede observar, si ejecuta este código, la etiqueta se encuentra en una posición muy mala. Esto es difícil de cambiar de buena manera con el método `add` . Para permitir una colocación más dinámica y flexible, eche un vistazo a Swing Layout Managers .

Creando Subclase JFrame

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    private static CustomFrame statFrame;

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        JLabel label = new JLabel(labelText);
        add(label);

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }

    public void addLabel(String labelText) {
        JLabel label = new JLabel(labelText);
        add(label);
        this.validate();
    }
```

```
    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            statFrame = frame;
            frame.setVisible(true);
        });

        try {
            Thread.sleep(5000);
        } catch (InterruptedException ex) {
            //Handle error
        }

        SwingUtilities.invokeLater(() -> statFrame.addLabel("Oh, hello world too."));
    }

}
```

Para más información sobre FlowLayout aquí .

Escuchando un evento

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        //Add a button
        JButton btn = new JButton("Hello button");
        //And a textbox
        JTextField field = new JTextField("Name");
        field.setSize(150, 50);
        //This next block of code executes whenever the button is clicked.
        btn.addActionListener((evt) -> {
            JLabel helloLbl = new JLabel("Hello " + field.getText());
            add(helloLbl);
            validate();
        });
        add(btn);
        add(field);
```

```
    }

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            frame.setVisible(true);
        });
    }

}
```

Crear una ventana emergente "Por favor espere ..."

Este código se puede agregar a cualquier evento, como un oyente, un botón, etc. Aparecerá un `JDialog` bloqueo y permanecerá hasta que se complete el proceso.

```
final JDialog loading = new JDialog(parentComponent);
JPanel p1 = new JPanel(new BorderLayout());
p1.add(new JLabel("Please wait..."), BorderLayout.CENTER);
loading.setUndecorated(true);
loading.getContentPane().add(p1);
loading.pack();
loading.setLocationRelativeTo(parentComponent);
loading.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
loading.setModal(true);

SwingWorker<String, Void> worker = new SwingWorker<String, Void>() {
    @Override
    protected String doInBackground() throws InterruptedException
        /** Execute some operation */
    }
    @Override
    protected void done() {
        loading.dispose();
    }
};
worker.execute(); //here the process thread initiates
loading.setVisible(true);
try {
    worker.get(); //here the parent thread waits for completion
} catch (Exception e1) {
    e1.printStackTrace();
}
```

Añadiendo JButtons (Hello World Pt.2)

Suponiendo que haya creado correctamente un JFrame y que Swing haya sido importado ...

Puedes importar Swing completamente

```
import javax.Swing.*;
```

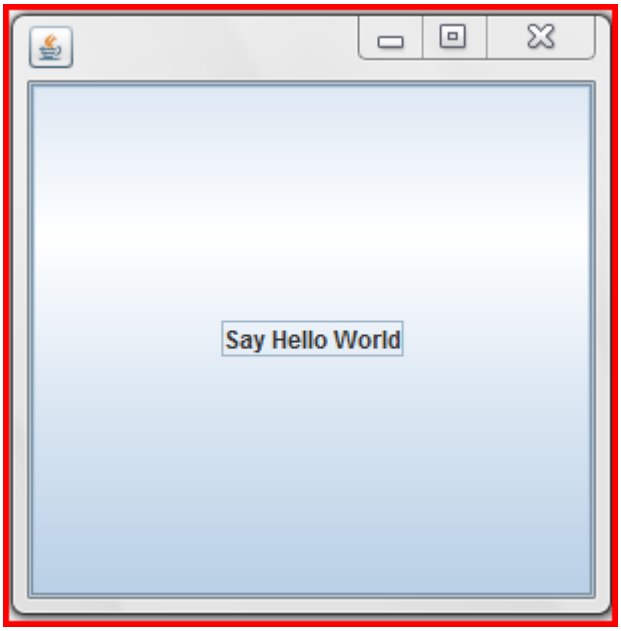
o puede importar los componentes de Swing / marco que desea utilizar

```
import javax.Swing.Jframe;
import javax.Swing.JButton;
```

Ahora vamos a agregar el Jbutton ...

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

    JFrame frame = new JFrame(); //creates the frame
    frame.setSize(300, 300);
    frame.setVisible(true);

    ////////////////////////ADDING BUTTON BELOW//////////////////////////////
    JButton B = new JButton("Say Hello World");
    B.addMouseListener(new MouseAdapter() {

        public void mouseReleased(MouseEvent arg0) {
            System.out.println("Hello World");
        }

    });
    B.setBounds(0, 0,frame.getHeight(), frame.getWidth());
    B.setVisible(true);
    frame.add(B);
    ////////////////////////////////////////////////////////////////////////
}
```

Al ejecutar / compilar este código, debería obtener algo como esto ...



Cuando se hace clic en el botón ... "Hello World" también debería aparecer en su consola.

Lea Lo esencial en línea: https://riptutorial.com/es/swing/topic/5415/lo-esencial

Capítulo 8: MigLayout

Examples

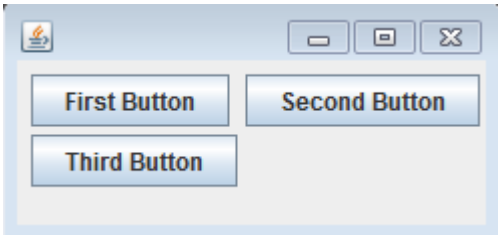
Elementos de envoltura

Este ejemplo muestra cómo colocar 3 botones en total con 2 botones en la primera fila. Luego se produce una envoltura, por lo que el último botón está en una nueva fila.

Las restricciones son cadenas simples, en este caso "envolver" al colocar el componente.

```
public class ShowMigLayout {

    // Create the elements
    private final JFrame demo = new JFrame();
    private final JPanel panel = new JPanel();
    private final JButton button1 = new JButton("First Button");
    private final JButton button2 = new JButton("Second Button");
    private final JButton button3 = new JButton("Third Button");

    public static void main(String[] args) {
        ShowMigLayout showMigLayout = new ShowMigLayout();
        SwingUtilities.invokeLater(showMigLayout::createAndShowGui);
    }

    public void createAndShowGui() {
        // Set the position and the size of the frame
        demo.setBounds(400, 400, 250, 120);

        // Tell the panel to use the MigLayout as layout manager
        panel.setLayout(new MigLayout());

        panel.add(button1);
        // Notice the wrapping
        panel.add(button2, "wrap");
        panel.add(button3);

        demo.add(panel);
        demo.setVisible(true);
    }
}
```

Salida:



Lea MigLayout en línea: https://riptutorial.com/es/swing/topic/2966/miglayout

Capítulo 9: Patrón MVP

Examples

Ejemplo de MVP simple

Para ilustrar un ejemplo simple del uso del patrón MVP, considere el siguiente código que crea una IU simple con solo un botón y una etiqueta. Cuando se hace clic en el botón, la etiqueta se actualiza con el número de veces que se ha hecho clic en el botón.

Tenemos 5 clases:

- Modelo - El POJO para mantener el estado (M en MVP)
- Vista: la clase con el código UI (V en MVP)
- ViewListener - Interfaz que proporciona métodos para responder a las acciones en la vista
- Presentador: responde a la entrada y actualiza la vista (P en MVP)
- Aplicación: la clase "principal" para reunir todo y lanzar la aplicación.

Una clase de "modelo" mínima que solo mantiene una única variable de `count` .

```
/**
 * A minimal class to maintain some state
 */
public class Model {
    private int count = 0;

    public void addOneToCount() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

Una interfaz mínima para notificar a los oyentes:

```
/**
 * Provides methods to notify on user interaction
 */
public interface ViewListener {
    public void onButtonClicked();
}
```

La clase de vista construye todos los elementos de la interfaz de usuario. La vista, y *solo* la vista, debe hacer referencia a los elementos de la interfaz de usuario (es decir, no hay botones, campos de texto, etc. en el presentador u otras clases).

```
/**
 * Provides the UI elements
```

```
 */

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

public class View {
    // A list of listeners subscribed to this view
    private final ArrayList<ViewListener> listeners;
    private final JLabel label;

    public View() {
        final JFrame frame = new JFrame();
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setLayout(new GridLayout());

        final JButton button = new JButton("Hello, world!");

        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                notifyListenersOnButtonClicked();
            }
        });
        frame.add(button);

        label = new JLabel();
        frame.add(label);

        this.listeners = new ArrayList<ViewListener>();

        frame.setVisible(true);
    }

    // Iterate through the list, notifying each listner individualy
    private void notifyListenersOnButtonClicked() {
        for (final ViewListener listener : listeners) {
            listener.onButtonClicked();
        }
    }

    // Subscribe a listener
    public void addListener(final ViewListener listener) {
        listeners.add(listener);
    }

    public void setLabelText(final String text) {
        label.setText(text);
    }
}
```

La lógica de notificación también puede codificarse así en Java8:

```
        ...
        final Button button = new Button("Hello, world!");
        // In order to do so, our interface must be changed to accept the event parametre
        button.addActionListener((event) -> {
            notifyListeners(ViewListener::onButtonClicked, event);
            // Example of calling methodThatTakesALong, would be the same as callying:
            // notifyListeners((listener, long)->listener.methodThatTakesALong(long), 10L)
            notifyListeners(ViewListener::methodThatTakesALong, 10L);
        });
        frame.add(button);
        ...

/**
 * Iterates through the subscribed listeneres notifying each listener individually.
 * Note: the {@literal '<T>' in private <T> void} is a Bounded Type Parametre.
 *
 * @param <T>      Any Reference Type (basically a class).
 *
 * @param consumer A method with two parameters and no return,
 *                 the 1st parametre is a ViewListner,
 *                 the 2nd parametre is value of type T.
 *
 * @param data     The value used as parametre for the second argument of the
 *                 method described by the parametre consumer.
 */
private <T> void notifyListeners(final BiConsumer<ViewListener, T> consumer, final T data) {
    // Iterate through the list, notifying each listener, java8 style
    listeners.forEach((listener) -> {

        // Calls the funcion described by the object consumer.
        consumer.accept(listener, data);

        // When this method is called using ViewListener::onButtonClicked
        // the line: consumer.accept(listener,data); can be read as:
        // void accept(ViewListener listener, ActionEvent data) {
        //     listener.onButtonClicked(data);
        // }

    });
}
```

La interfaz debe ser rediseñada para tomar el ActionEvent como un parámetro:

```
public interface ViewListener {
    public void onButtonClicked(ActionEvent evt);
    // Example of methodThatTakesALong signature
    public void methodThatTakesALong(long );
}
```

Aquí solo se necesita un método de notificación, el método de escucha real y su parámetro se pasan como parámetros. En caso necesario, esto también puede usarse para algo menos ingenioso que el manejo de eventos reales, todo funciona siempre que haya un método en la interfaz, por ejemplo:

```
        notifyListeners(ViewListener::methodThatTakesALong, -1L);
```

El presentador puede ver la vista y agregarse a sí mismo como un oyente. Cuando se hace clic en el botón en la vista, la vista notifica a todos los oyentes (incluido el presentador). Ahora que se notifica al presentador, puede tomar las medidas adecuadas para actualizar el modelo (es decir, el estado de la aplicación) y luego actualizar la vista en consecuencia.

```
/**
 * Responsible to responding to user interaction and updating the view
 */
public class Presenter implements ViewListener {
    private final View view;
    private final Model model;

    public Presenter(final View view, final Model model) {
        this.view = view;
        view.addListener(this);
        this.model = model;
    }

    @Override
    public void onButtonClicked() {
        // Update the model (ie. the state of the application)
        model.addOneToCount();
        // Update the view
        view.setLabelText(String.valueOf(model.getCount()));
    }
}
```

Para poner todo junto, la vista se puede crear e inyectar en el presentador. Del mismo modo, un modelo inicial puede ser creado e inyectado. Si bien ambos *pueden* crearse en el presentador, inyectarlos en el constructor permite realizar pruebas mucho más simples.

```
public class Application {
    public Application() {
        final View view = new View();
        final Model model = new Model();
        new Presenter(view, model);
    }

    public static void main(String... args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new Application();
            }
        });
    }
}
```

Lea Patrón MVP en línea: https://riptutorial.com/es/swing/topic/5154/patron-mvp

Capítulo 10: StyledDocument

Sintaxis

- doc.insertString (índice, texto, atributos); // los atributos deben ser un conjunto de atributos

Examples

Creando un DefaultStyledDocument

```
try {
    StyledDocument doc = new DefaultStyledDocument();
    doc.insertString(0, "This is the beginning text", null);
    doc.insertString(doc.getLength(), "\nInserting new line at end of doc", null);
    MutableAttributeSet attrs = new SimpleAttributeSet();
    StyleConstants.setBold(attrs, true);
    doc.insertString(5, "This is bold text after 'this'", attrs);
} catch (BadLocationException ex) {
    //handle error
}
```

DefaultStyledDocuments probablemente serán tus recursos más utilizados. Se pueden crear directamente y subclasificar la clase abstracta `StyledDocument` .

Añadiendo StyledDocument a JTextPane

```
try {
        JTextPane pane = new JTextPane();
        StyledDocument doc = new DefaultStyledDocument();
        doc.insertString(0, "Some text", null);
        pane.setDocument(doc); //Technically takes any subclass of Document
    } catch (BadLocationException ex) {
        //handle error
    }
```

El JTextPane se puede agregar a cualquier formulario de Swing GUI.

Copiando DefaultStyledDocument

`StyledDocuments` generalmente no implementa clones, por lo que debe copiarlos de una forma diferente si es necesario.

```
try {
        //Initialization
        DefaultStyledDocument sourceDoc = new DefaultStyledDocument();
        DefaultStyledDocument destDoc = new DefaultStyledDocument();
        MutableAttributeSet bold = new SimpleAttributeSet();
        StyleConstants.setBold(bold, true);
        MutableAttributeSet italic = new SimpleAttributeSet();
```

```
        StyleConstants.setItalic(italic, true);
        sourceDoc.insertString(0, "Some bold text. ", bold);
        sourceDoc.insertString(sourceDoc.getLength(), "Some italic text", italic);

        //This does the actual copying
        String text = sourceDoc.getText(0, sourceDoc.getLength()); //This copies text, but
loses formatting.
        for (int i = 0; i < text.length(); i++) {
            Element e = destDoc.getCharacterElement(i); //A Elment describes a particular part
of a document, in this case a character
            AttributeSet attr = e.getAttributes(); //Gets the attributes for the character
            destDoc.insertString(destDoc.getLength(), text.substring(i, i+1), attr); //Gets
the single character and sets its attributes from the element
        }
    } catch (BadLocationException ex) {
        //handle error
    }
```

Serialización de un DefaultStyledDocument a RTF

Usando la biblioteca AdvancedRTFEditorKit puede serializar un `DefaultStyledDocument` a una cadena RTF.

```
try {
   DefaultStyledDocument writeDoc = new DefaultStyledDocument();
   writeDoc.insertString(0, "Test string", null);

   AdvancedRTFEditorKit kit = new AdvancedRTFEditorKit();
   //Other writers, such as a FileWriter, may be used
   //OutputStreams are also an option
   Writer writer = new StringWriter();
   //You can write just a portion of the document by modifying the start
   //and end indexes
   kit.write(writer, writeDoc, 0, writeDoc.getLength());
   //This is the RTF String
   String rtfDoc = writer.toString();

   //As above this may be a different kind of reader or an InputStream
   StringReader reader = new StringReader(rtfDoc);
   //AdvancedRTFDocument extends DefaultStyledDocument and can generally
   //be used wherever DefaultStyledDocument can be.
   //However for reading, AdvancedRTFDocument must be used
   DefaultStyledDocument readDoc = new AdvancedRTFDocument();
   //You can insert at different values by changing the "0"
   kit.read(reader, readDoc, 0);
   //readDoc is now the same as writeDoc
} catch (BadLocationException | IOException ex) {
    //Handle exception
    ex.printStackTrace();
}
```

Lea StyledDocument en línea: https://riptutorial.com/es/swing/topic/5416/styleddocument

Capítulo 11: Swing Workers y el EDT

Sintaxis

- Clase abstracta pública SwingWorker <T, V>
- T: el tipo de resultado que devuelve este método SwingWorker's doInBackground y get.
- V: el tipo utilizado para llevar a cabo resultados intermedios mediante los métodos de publicación y procesamiento de este SwingWorker.
- T doInBackground (): la función abstracta que debe anularse. El tipo de devolución es T.

Examples

Tema principal y evento de despacho

Como cualquier otro programa de Java, cada programa de swing comienza con un método principal. El método principal es iniciado por el hilo principal. Sin embargo, los componentes de Swing deben crearse y actualizarse en el hilo de despacho de eventos (o en breve: EDT). Para ilustrar la dinámica entre el hilo principal y el EDT, eche un vistazo a este Hello World! ejemplo.

El hilo principal solo se utiliza para delegar la creación de la ventana a la EDT. Si la EDT aún no se ha iniciado, la primera llamada a `SwingUtilities.invokeLater` configurará la infraestructura necesaria para procesar los componentes Swing. Además, la EDT permanece activa en el fondo. El hilo principal morirá directamente después de iniciar la configuración de la EDT, pero la EDT permanecerá activa hasta que el usuario salga del programa. Esto se puede lograr presionando el cuadro de cierre en la instancia de `JFrame` visible. Esto cerrará el EDT y el proceso del programa se realizará por completo.

Encuentre los primeros N números pares y muestre los resultados en un JTextArea donde los cálculos se realizan en segundo plano.

```
import java.awt.EventQueue;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.SwingWorker;

class PrimeNumbersTask extends SwingWorker<List<Integer>, Integer> {
    private final int numbersToFind;
```

```
    private final JTextArea textArea;

    PrimeNumbersTask(JTextArea textArea, int numbersToFind) {
        this.numbersToFind = numbersToFind;
        this.textArea = textArea;
    }

    @Override
    public List<Integer> doInBackground() {
        final List<Integer> result = new ArrayList<>();
        boolean interrupted = false;
        for (int i = 0; !interrupted && (i < numbersToFind); i += 2) {
            interrupted = doIntenseComputing();
            result.add(i);
            publish(i); // sends data to process function
        }
        return result;
    }

    private boolean doIntenseComputing() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            return true;
        }
        return false;
    }

    @Override
    protected void process(List<Integer> chunks) {
        for (int number : chunks) {
            // the process method will be called on the EDT
            // thus UI elementes may be updated in here
            textArea.append(number + "\n");
        }
    }
}

public class SwingWorkerExample extends JFrame {
    private JTextArea textArea;

    public SwingWorkerExample() {
        super("Java SwingWorker Example");
        init();
    }

    private void init() {
        setSize(400, 400);
        setLayout(new GridLayout(1, 1));
        textArea = new JTextArea();
        add(textArea);

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

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

```
        SwingWorkerExample ui = new SwingWorkerExample();
        EventQueue.invokeLater(() -> {
            ui.setVisible(true);
        });

        int n = 100;
        PrimeNumbersTask task = new PrimeNumbersTask(ui.textArea, n);
        task.execute(); // run async worker which will do long running task on a
        // different thread
        System.out.println(task.get());
    }
}
```

Lea Swing Workers y el EDT en línea: https://riptutorial.com/es/swing/topic/3431/swing-workers-y-el-edt

Capítulo 12: temporizador en JFrame

Examples

Temporizador en jframe

Supongamos que tiene un botón en su programa Java que hace una cuenta regresiva. Aquí está el código de temporizador de 10 minutos.

```
private final static long REFRESH_LIST_PERIOD=10 * 60 * 1000; //10 minutes

Timer timer = new Timer(1000, new ActionListener() {

   @Override
   public void actionPerformed(ActionEvent e) {
    if (cnt > 0) {
     cnt = cnt - 1000;
     btnCounter.setText("Remained (" + format.format(new Date(cnt)) + ")");
    } else {
     cnt = REFRESH_LIST_PERIOD;
     //TODO
    }

   }
  });

  timer.start();
```

Lea temporizador en JFrame en línea: https://riptutorial.com/es/swing/topic/6745/temporizador-en-jframe

Capítulo 13: Usando Look and Feel

Examples

Usando el sistema L&F

Swing soporta bastantes L & Fs nativos.
Siempre puede instalar uno fácilmente sin tener que llamar a una clase específica de L&F:

```
public class SystemLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
                {
                    // Installing native L&F as a current application L&F
                    // We do not know what exactly L&F class is, it is provided by the
UIManager
                    UIManager.setLookAndFeel ( UIManager.getSystemLookAndFeelClassName () );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some natively-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
```

```
                content.add ( new JButton ( "Native-looking button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Estos son los soportes nativos de L & Fs JDK (OS -> L&F):

OS	Nombre de L&F	Clase de L&F
Solaris, Linux con GTK +	GTK +	com.sun.java.swing.plaf.gtk.GTKLookAndFeel
Otros Solaris, Linux	Motivo	com.sun.java.swing.plaf.motif.MotifLookAndFeel
Windows clásico	Windows	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows XP	Windows XP	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows Vista	Windows Vista	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Macintosh	Macintosh	com.apple.laf.AquaLookAndFeel *
IBM UNIX	IBM	javax.swing.plaf.synth.SynthLookAndFeel *
HP UX	HP	javax.swing.plaf.synth.SynthLookAndFeel *

* estos L & F son suministrados por el proveedor del sistema y el nombre real de la clase L&F puede variar

Usando L&F personalizado

```
public class CustomLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
```

```
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
                {
                    // Installing custom L&F as a current application L&F
                    UIManager.setLookAndFeel ( "javax.swing.plaf.metal.MetalLookAndFeel" );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some pretty-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
                content.add ( new JButton ( "Metal button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Puede encontrar una gran lista de Swing L & Fs disponibles en el tema aquí: Java Look and Feel (L&F)
Tenga en cuenta que algunos de esos L & F pueden estar bastante desactualizados en este momento.

Lea Usando Look and Feel en línea: https://riptutorial.com/es/swing/topic/3627/usando-look-and-feel

Capítulo 14: Usando Swing para interfaces gráficas de usuario

Observaciones

Saliendo de la aplicación en ventana cerrar

Es fácil olvidarse de salir de la aplicación cuando se cierra la ventana. Recuerda añadir la siguiente línea.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the JFrame is closed
```

Examples

Creando una ventana vacía (JFrame)

Creando el JFrame

Crear una ventana es fácil. Solo tienes que crear un `JFrame` .

```
JFrame frame = new JFrame();
```

Titulando la ventana

Es posible que desee dar un título a su ventana. Puede hacerlo pasando una cadena al crear el `JFrame` o llamando a `frame.setTitle(String title)` .

```
JFrame frame = new JFrame("Super Awesome Window Title!");
//OR
frame.setTitle("Super Awesome Window Title!");
```

Configuración del tamaño de la ventana

La ventana será lo más pequeña posible cuando se haya creado. Para hacerlo más grande, puedes establecer su tamaño explícitamente:

```
frame.setSize(512, 256);
```

O puede tener el tamaño del marco en función del tamaño de su contenido con el método `pack()` .

```
frame.pack();
```

Los `setSize()` y `pack()` se excluyen mutuamente, así que usa uno u otro.

Qué hacer en la ventana Cerrar

Tenga en cuenta que la aplicación **no se** cerrará cuando la ventana se haya cerrado. Puede salir de la aplicación después de que la ventana se haya cerrado diciéndole al `JFrame` que haga eso.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

Alternativamente, puedes decirle a la ventana que haga otra cosa cuando está cerrada.

```
WindowConstants.DISPOSE_ON_CLOSE //Get rid of the window
WindowConstants.EXIT_ON_CLOSE //Quit the application
WindowConstants.DO_NOTHING_ON_CLOSE //Don't even close the window
WindowConstants.HIDE_ON_CLOSE //Hides the window - This is the default action
```

Creación de un panel de contenido

Un paso opcional es crear un panel de contenido para su ventana. Esto no es necesario, pero si desea hacerlo, cree un `JPanel` y llame a `frame.setContentPane(Component component)` .

```
JPanel pane = new JPanel();
frame.setContentPane(pane);
```

Mostrando la ventana

Después de crearlo, querrá crear sus componentes y luego mostrar la ventana. Mostrando la ventana se hace como tal.

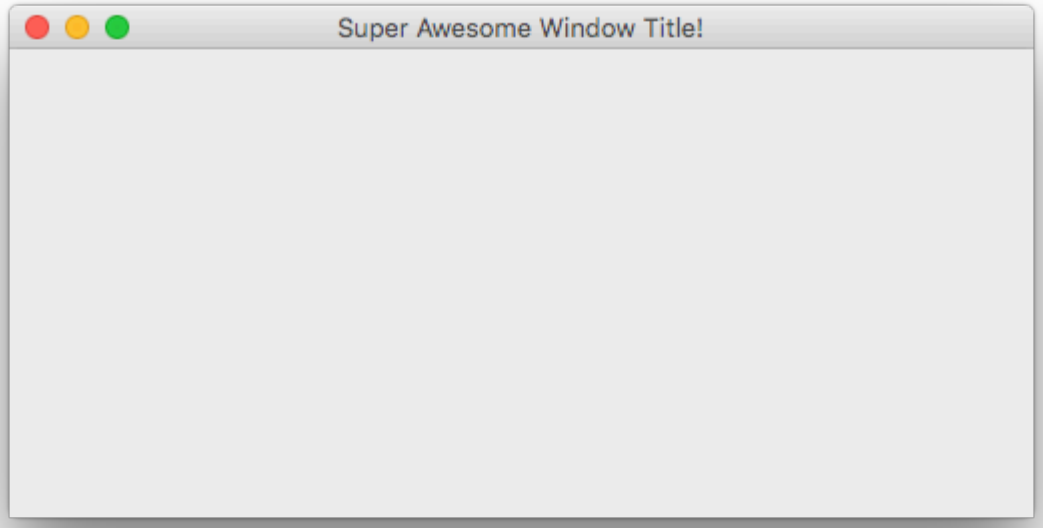
```
frame.setVisible(true);
```

Ejemplo

Para aquellos de ustedes que les gusta copiar y pegar, aquí hay un código de ejemplo.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
```

```
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

frame.setVisible(true); //Show the window
```



Añadiendo Componentes

Un componente es una especie de elemento de interfaz de usuario, como un botón o un campo de texto.

Creando un componente

Crear componentes es casi idéntico a crear una ventana. Sin embargo, en lugar de crear un `JFrame` , crea ese componente. Por ejemplo, para crear un `JButton` , haga lo siguiente.

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

Muchos componentes pueden tener parámetros pasados cuando se crean. Por ejemplo, a un botón se le puede dar un texto para mostrar.

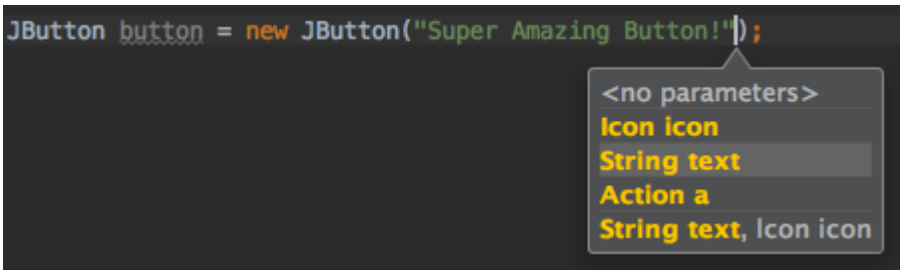
```
JButton button = new JButton("Super Amazing Button!");
```

Si no desea crear un botón, puede encontrar una lista de componentes comunes en otro ejemplo en esta página.

Los parámetros que se pueden pasar a ellos varían de un componente a otro. Una buena manera

de verificar lo que pueden aceptar es mirar los parámetros dentro de su IDE (si usa uno). Los accesos directos predeterminados se enumeran a continuación.

- IntelliJ IDEA - Windows / Linux: `CTRL + P`
- IDEA IntelliJ - OS X / macOS: `CMD + P`
- Eclipse: `CTRL + SHIFT + Space`
- NetBeans: `CTRL + P`



Mostrando el componente

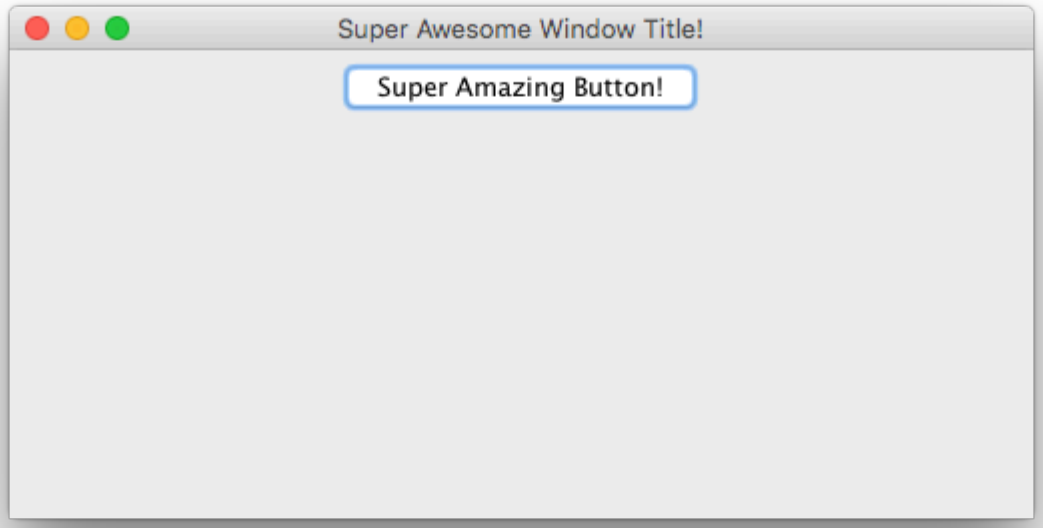
Después de crear un componente, normalmente establecerá sus parámetros. Después de eso, debe ponerlo en algún lugar, como en su `JFrame` , o en su panel de contenido si creó uno.

```
frame.add(button); //Add to your JFrame
//OR
pane.add(button); //Add to your content pane
//OR
myComponent.add(button); //Add to whatever
```

Ejemplo

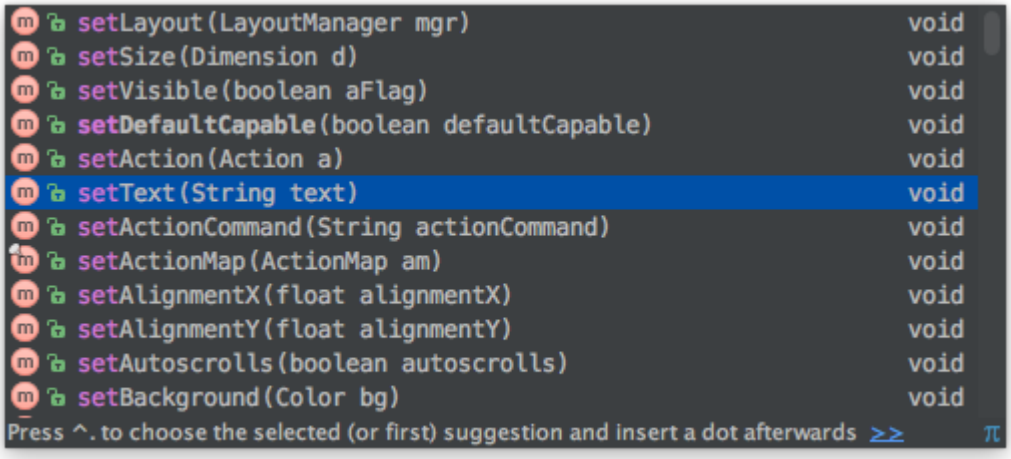
Este es un ejemplo de cómo crear una ventana, configurar un panel de contenido y agregarle un botón.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

JButton button = new JButton("Super Amazing Button!"); //Create the button
pane.add(button); //Add the button to the content pane

frame.setVisible(true); //Show the window
```



Configuración de parámetros para componentes

Los componentes tienen varios parámetros que se pueden configurar para ellos. Varían de un componente a otro, por lo que una buena manera de ver qué parámetros se pueden configurar para los componentes es comenzar a escribir `componentName.set` y dejar que su IDE complete el autocompletado (si usa un IDE) sugiera métodos. El acceso directo predeterminado en muchos IDE, si no se muestra automáticamente, es `CTRL + Space` .



Parámetros comunes que son compartidos entre todos los componentes.

Descripción	Método
Establece el tamaño más pequeño que puede tener el componente (solo si el administrador de diseño respeta la	`setMinimumSize(Dimension minimumSize)`

Descripción	Método
propiedad de tamaño mínimo)	
Establece el tamaño más grande que puede tener el componente (solo si el administrador de diseño respeta la propiedad maximumSize)	`setMaximumSize(Dimension maximumSize)`
Establece el tamaño preferido del componente (solo si el administrador de diseño respeta la propiedad preferredSize)	`setPreferredSize(Dimension preferredSize)`
Muestra u oculta el componente.	`setVisible(boolean aFlag)`
Establece si el componente debe responder a la entrada del usuario.	`setEnabled(boolean enabled)`
Establece la fuente del texto.	`setFont(Font font)`
Establece el texto de la información sobre herramientas.	`setToolTipText(String text)`
Establece el color de fondo del componente.	`setBackground(Color bg)`
Establece el color de fondo (color de fuente) del componente	`setForeground(Color bg)`

Parámetros comunes en otros componentes.

Componentes comunes	Descripción	Método
`JLabel` , `JButton` , `JCheckBox` , `JRadioButton` , `JToggleButton` , `JMenu` , `JMenuItem` , `JTextArea` , `JTextField`	Establece el texto mostrado.	`setText(String text)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Set es un valor numérico entre los valores mínimo y máximo del componente	`setValue(int n)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Establecer es el valor más pequeño posible que la propiedad de `value` puede ser	`setMinimum(int n)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Conjunto es el mayor valor posible que la propiedad de `value` puede ser	`setMaxmimum(int n)`
`JCheckBox` , `JToggleBox`	Establece si el valor es verdadero o falso (por ejemplo: ¿se debe marcar una casilla de	`setSelected(boolean b)`

Componentes comunes	Descripción	Método
	verificación?)	

Componentes comunes

Descripción	Clase
Botón	`JButton`
Caja	`JCheckBox`
Menú desplegable / cuadro combinado	`JComboBox`
Etiqueta	`JLabel`
Lista	`JList`
Barra de menús	`JMenuBar`
Menú en una barra de menú	`JMenu`
Artículo en un menú	`JMenuItem`
Panel	`JPanel`
Barra de progreso	`JProgressBar`
Boton de radio	`JRadioButton`
Barra de desplazamiento	`JScrollBar`
Deslizador	`JSlider`
Spinner / selector de número	`JSpinner`
Mesa	`JTable`
Árbol	`JTree`
Área de texto / campo de texto multilínea	`JTextArea`
Campo de texto	`JTextField`
Barra de herramientas	`JToolBar`

Haciendo Interfaces de Usuario Interactivas

Tener un botón allí está muy bien, pero ¿cuál es el punto si hacer clic en él no hace nada? `ActionListener` se usan para decirle a su botón, u otro componente que haga algo cuando está

activado.

Agregar `ActionListener` s se hace como tal.

```
buttonA.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //Code goes here...
        System.out.println("You clicked the button!");
    }
});
```

O, si estás utilizando Java 8 o superior ...

```
buttonA.addActionListener(e -> {
    //Code
    System.out.println("You clicked the button!");
});
```

Ejemplo (Java 8 y superior)

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create a pane to house all content
frame.setContentPane(pane);

JButton button = new JButton("Click me - I know you want to.");
button.addActionListener(e -> {
    //Code goes here
    System.out.println("You clicked me! Ouch.");
});
pane.add(buttonA);

frame.setVisible(true); //Show the window
```

Organizar el diseño de componentes

Agregar componentes uno tras otro da como resultado una interfaz de usuario que es difícil de usar, porque todos los componentes están en **alguna parte** . Los componentes se ordenan de arriba a abajo, cada componente en una "fila" separada.

Para remediar esto y brindarle a los desarrolladores la posibilidad de diseñar componentes fácilmente, Swing tiene `LayoutManager` s.

Estos LayoutManagers se tratan más ampliamente en Introducción a los administradores de diseño, así como los temas separados del Administrador de diseño:

- Diseño de cuadrícula

- GridBag Layout

Lea Usando Swing para interfaces gráficas de usuario en línea:
https://riptutorial.com/es/swing/topic/2982/usando-swing-para-interfaces-graficas-de-usuario

Creditos

S. No	Capítulos	Contributors
1	Empezando con el swing	Community, Freek de Bruijn, Petter Friberg, Vogel612, XavCo7
2	Diseño de cuadrícula	Lukas Rotter, user6653173
3	Gestión de diseño	explv, J Atkin, mayha, pietrocalzini, recke96, Squidward, XavCo7
4	Gráficos	Adel Khial, Ashlyn Campbell, Squidward
5	GridBag Layout	CraftedCart, Enwired, mayha, Vogel612
6	JList	Andreas Fester, Squidward, user6653173
7	Lo esencial	DarkV1, DonyorM, elias, Robin, Squidward
8	MigLayout	hamena314, keuleJ
9	Patrón MVP	avojak, ehzawad, Leonardo Pina, sjngm, Squidward
10	StyledDocument	DonyorM, Squidward
11	Swing Workers y el EDT	dpr, isaias-b, rahul tyagi
12	temporizador en JFrame	SSD
13	Usando Look and Feel	Mikle Garin
14	Usando Swing para interfaces gráficas de usuario	CraftedCart, mayha, Michael, Vogel612, Winter