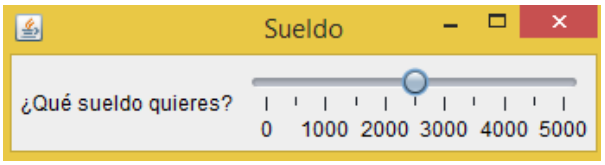
# ACTIVIDAD GUIADA 13
# Otros componentes Swing

## Reguladores

La forma más sencilla de obtener entradas numéricas de los usuarios consiste en usar un **regulador**, un componente que se puede arrastrar horizontal o verticalmente. En Swing, los reguladores se representan con la clase **`JSlider`**. Permiten elegir un número entre un valor mínimo y otro máximo. Estos valores se pueden mostrar en una etiqueta que incluye el valor mínimo, el máximo y los intermedios. En la figura siguiente se puede ver un ejemplo que crearemos más adelante.




Se puede crear un regulador horizontal con uno de los siguientes constructores:

- **`JSlider()`**: Crea un regulador con los valores por defecto (valor mínimo 0, el valor máximo 100 y el valor inicial 50).
- **`JSlider(int max, int min)`**: Crea un regulador con los valores mínimo y máximo especifica.
- **`JSlider(int max, int min, int valor)`**: Crea un regulador con los valores mínimo, máximo e inicial especificados.
- **`JSlider(int orientacion, int max, int min, int valor)`**: Crea un regulador con la orientación y los valores mínimo, máximo e inicial especificados. La orientación debe ser uno de los valores **`JSlider.VERTICAL`** o **`JSlider.HORIZONTAL`**.

La siguiente instrucción crea un regulador vertical para un número entre 1 y 1000, comenzando el selector en la posición 500:

```
JSlider sliAdivina = new JSlider(JSlider.VERTICAL, 1, 1000, 500)
```

Para mostrar una etiqueta para el regulador, debemos indicar la información que mostrará la etiqueta. Para ello se usan los siguientes métodos:

- **`setMajorTickSpacing(int frecuencia)`**. Indica la frecuencia de las marcas mayores. Estas se muestran con mayor grosor.
- **`setMinorTickSpacing(int frecuencia)`**. Indica la frecuencia de las marcas menores. Estas se muestran con menor grosor.
- **`setPaintTicks(boolean b)`**. Muestra las marcas si le pasamos el parámetro **`true`**.
- **`setPaintLabels(boolean b)`**. Muestra las etiquetas asociadas a las marcas mayores si le pasamos el parámetro **`true`**.

Por ejemplo, si queremos mostrar el deslizador mostrado anteriormente lo haríamos mediante el siguiente código (sólo se detalla el código correspondiente a la creación del mismo):

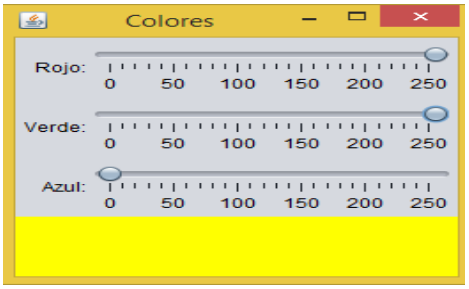
```
JSlider sliSueldo=new JSlider(0,5000);
sliSueldo.setMajorTickSpacing(1000);
sliSueldo.setMinorTickSpacing(500);
sliSueldo.setPaintTicks(true);
sliSueldo.setPaintLabels(true);
add(sliSueldo);
```

Para controlar la entrada de un regulador, debemos contar con una clase que implemente la interfaz **ChangeListener** del paquete **javax.swing.event**. Esta interfaz incluye un solo método **stateChanged**:

```
public void stateChanged(ChangeEvent event) {
 // instrucciones para procesar el evento
}
```

Para registrar un objeto como escuchador de cambios, invocaremos el método **addChangeListener(Object)** del mismo. Al mover el regulador, se invoca el método **stateChanged()** del objeto que escucha.

A continuación se muestra un ejemplo completo para obtener la siguiente pantalla:



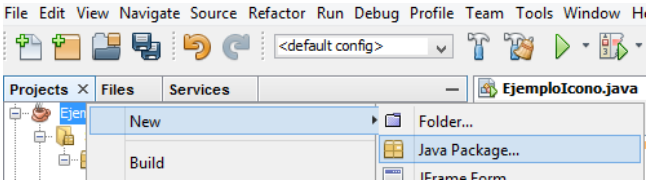

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;

public class Colores extends JFrame implements ChangeListener{
     JSlider sliRojo;
     JSlider sliVerde;
     JSlider sliAzul;
     JPanel pColor;

    public Colores(){
        super("Colores");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        GridLayout admin=new GridLayout(4,1);
        setLayout(admin);

        FlowLayout adminPanel=new FlowLayout(FlowLayout.RIGHT);
```

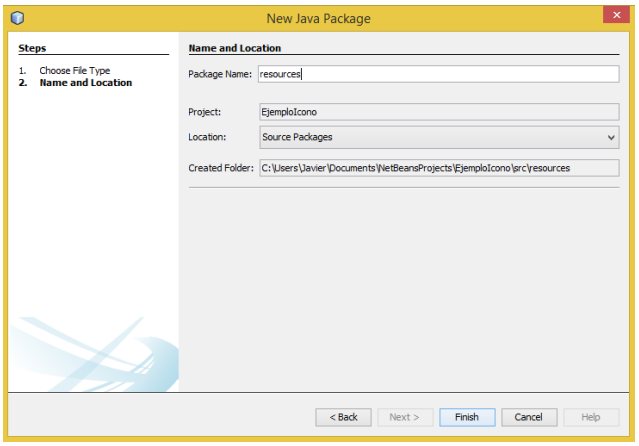
```
        JPanel pRojo=new JPanel();
        pRojo.setLayout(adminPanel);

        JLabel lblRojo=new JLabel("Rojo:");
        sliRojo=new JSlider(0,255,255);
        sliRojo.setMajorTickSpacing(50);
        sliRojo.setMinorTickSpacing(10);
        sliRojo.setPaintTicks(true);
        sliRojo.setPaintLabels(true);
        sliRojo.addChangeListener(this);
        pRojo.add(lblRojo);
        pRojo.add(sliRojo);
        add(pRojo);

        JPanel pVerde=new JPanel();
        pVerde.setLayout(adminPanel);
        JLabel lblVerde=new JLabel("Verde:");
        sliVerde=new JSlider(0,255,0);
        // … Misma configuracion que para el rojo
        sliVerde.addChangeListener(this);
        pVerde.add(lblVerde);
        pVerde.add(sliVerde);
        add(pVerde);

        JPanel pAzul=new JPanel();
        pAzul.setLayout(adminPanel);

        JLabel lblAzul=new JLabel("Azul:");
        sliAzul=new JSlider(0,255,0);
         // … Misma configuracion que para el rojo
        sliAzul.addChangeListener(this);
        pAzul.add(lblAzul);
        pAzul.add(sliAzul);
        add(pAzul);

        pColor=new JPanel();
        pColor.setBackground(Color.RED);
        add(pColor);

        pack();
        setVisible(true);
    }
    }

@Override
    public void stateChanged(ChangeEvent ce) {
      JSlider sliColor=(JSlider)ce.getSource();
       if(!sliColor.getValueIsAdjusting()){
              Color colorActual=new Color(sliRojo.getValue(),
                                          sliVerde.getValue(),
                                          sliAzul.getValue());
          pColor.setBackground(colorActual);
      }
 }
    public static void main(String[] args) {
       Colores frColores=new Colores();
    }
}
```

## Iconos de imágenes

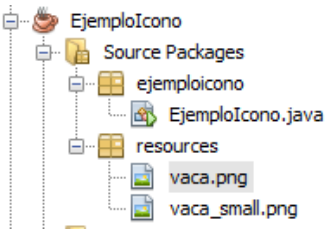
Una de las formas más sencillas de mejorar el aspecto visual de una GUI consiste en emplear iconos, pequeñas imágenes para identificar botones y otras partes de una interfaz. Con muchos de los componentes de la biblioteca de clases Swing puede etiquetar un componente con una imagen en lugar de texto por medio de la clase **`ImageIcon`** del paquete **`javax.swing`**.

Es conveniente tener los archivos de imagen organizados en un paquete específico para ello, dentro de nuestro proyecto. Para ello, pulsaremos con el botón derecho sobre el proyecto *New > Java Package* y creamos un paquete llamado *resources* destinado a almacenar las imágenes




A continuación, añadiremos los archivos desde su ubicación original a nuestro proyecto. Lo más sencillo es copiarlos (desde el explorador de archivos de Windows), y posteriormente colocarnos sobre el paquete *resources* y pulsando el botón derecho del ratón escoger *Pegar*.




Se puede crear un objeto **`ImageIcon`** a partir de un archivo de imagen. Existen varios métodos para hacerlo, pero nosotros vamos a crear un objeto de tipo **`Image`** que le pasaremos al constructor de **`ImageIcon`**. Dicho objeto se crea usando el método **`getClass().getResource(ruta_imagen)`** tal y como se muestra a continuación. Obsérvese que la ruta comienza por / y debe incluir el directorio *resources*.

```
ImageIcon iconVaca = new
ImageIcon(getClass().getResource("/resources/vaca_small.png"));
```

*NOTA: Aunque algunos sistemas operativos emplean el carácter* **`\`** *para separar carpetas y nombres de archivo, el constructor* ***`ImageIcon`*** *requiere el carácter* **`/`** *como separador.*

El formato gráfico empleado para crear el icono debe tener formato GIF, JPEG o PNG. La mayoría son GIF, muy indicado para mostrar pequeños gráficos con un número limitado de colores.

El constructor **ImageIcon** carga inmediatamente la imagen completa desde el archivo. Pueden usarse iconos de imagen como etiquetas y botones por medio de los métodos de constructor **JLabel(ImageIcon)** y **JButton(ImageIcon)**:

```
JButton btnVaca = new JButton(iconVaca);
```

Si agregamos la llamada anterior a un frame correctamente inicializado veremos un resultado como el siguiente:


El tamaño del botón en este caso depende del tamaño del icono (en este caso 256x256 píxeles). Si cogemos el icono **vaca_small.png** (de 64x64 píxeles), veremos el siguiente resultado, más acorde al tamaño habitual de un botón:


Varios componentes pueden tener un icono y una etiqueta de texto. La siguiente instrucción crea un botón con ambos elementos:

```
ImageIcon iconVaca = new ImageIcon(getClass().getResource("/resources/vaca_small.png"));
JButton btnVaca = new JButton("Vaca",iconVaca);
```


Los iconos de imagen suelen usarse en barras de herramientas, contenedores que agrupan varios componentes en una fila o columna.